5.1 MICROPROCESSOR APPLICATIONS

Microprocessor based systems may be used in almost every field of life and these applications increasing day by day at a very fast rate. For example, it can be used to control the temperature of a furnace, pressure of a boiler, speed of an electric motor, automatic control of furnace in a power plant, military equipments like radars, tanks etc., lift control, traffic control, robot control etc.

5.1.1 Temperature Control System

For the measurement of temperature, a transducer is used to convert temperature into electrical quantity. The electrical output of a transducer is proportional to temperature. If the electrical signal is small, it is amplified using amplifiers. The electrical signal is applied to an A/D converter which converts the analog voltage into equivalent digital signal. The output of ADC is connected to I/O ports which in turn are connected to the microprocessor subsystem.

The block diagram of a temperature monitoring system is shown in Fig. 5.1.1

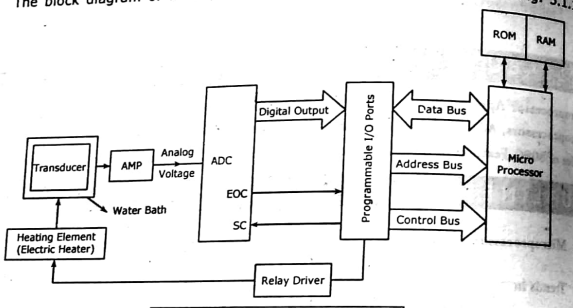


Fig. 5.1.1 Temperature Controller

The major parts of a temperature monitoring system are,

Transducer : For the measurement of temperature and to convert it into electrical signal, transducer is used. Thermistors, thermocouples, resistance thermometers etc., are some of the transducers. A thermistor is a semiconductor device having a large negative temperature coefficient. The resistance of the thermistor changes proportional to the temperature. Thermocouples are most widely used transducers to measure temperature. A thermocouple consists of two dissimilar metals joined end to end. When one junction is kept cold and the other junction is heated gradually, current flows through the thermocouple. The current produced is known as thermo-electric current and the e.m.f produced is known as thermo e.m.f. Thermocouple materials for some range of temperature are shown.

Material	Temperature Range °C
Iron-constantan	– 200 to + 1300
Copper-Constantan	– 200 to + 400
Tungsten-Rhenium	0 to 2000

The output of a thermocouple proportional to the temperature of the furnace or oven is in millivolt. It is to be amplified using amplifier. The amplified voltage is given as the input to A/D converter.

A/D Converter : An analog to digital converter converts the analog voltage into equivalent digital value.

Let us take that the analog input given to the ADC is converted to digital equivalent and the digital data is available in the accumulator. Now the data available in the accumulator is proportional to the temperature. Let, us denote the data corresponding to that temperature as T. Let, us also assume that there are 2 values corresponding to the temperatures (Lower and Upper limits) in two consecutive memory locations X and Y. They represent lower limit temperature T_L and upper limit temperature T_U. The measured temperature T is compared with T_U and T_L. If the measured temperature is greater than the upper limit, the microprocessor sends a control signal to the relay to switch OFF the heater so that the temperature is reduced. If the measured temperature is less than the lower limit, the microprocessor sends a control signal to the relay to switch ON the heater to increase the temperature.

Relay driver is switched ON (or) OFF using A_7 line of port A. The circuit connection is shown in Fig. 5.1.2.

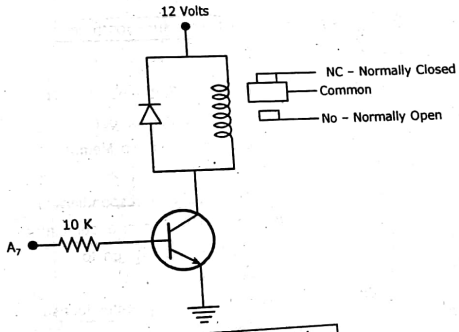


Fig. 5.1.2 Relay Driver

The flowchart for checking and controlling the temperature as shown Fig. 5.1.2 and the program are shown below.

Let us assume that the values T_L and T_U are stored in locations 2060_H and 2061_H respectively. The program calls a subroutine named as ADC. The program written for ADC 0809 can be taken as the subroutine with two changes. The initialization of 8255 is included in the main program itself and left out in the subroutine. Also, the HLT instruction in the ADC program is replaced by RET instruction.

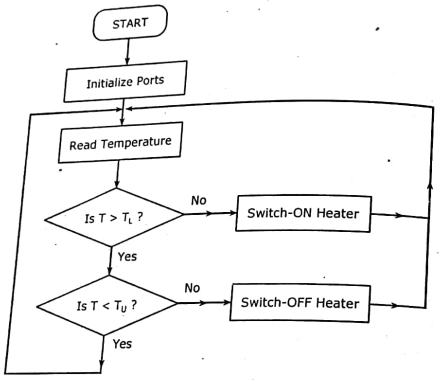


Fig. 5.1.3 Flow Chart for Temperature Controller

Program for Temperature Controller

```
        MVI     A, 8AH      ; Initialize 8255 Port A out, Port B
        OUT     F3H         ; in, port C_U in, Port C_L out
        LXI     H, 2060H    ; Store T_L and T_U in two Memory
                            ; locations
LOOP1 : CALL    ADC         ; The digital value corresponding to
                            ; the temperature T is in accumulator
        CMP     M           ; If T is greater than T_L, go to
        JNC     LOOP2       ; compare T and T_U
        MVI     A, 80H      ; Else, switch ON the heater to rise
        OUT     PA          ; the temperature
        JMP     LOOP1
```



```
LOOP2 : INX     H
        CMP     M           ; Compare T and T_U. If T is greater
        JC      LOOP1       ; than T_U, switch OFF the heater
        MVI     A, 00H
        OUT     P_A
        JMP     LOOP1       ; Continue in the loop
```

5.1.2 Hex Keyboard Interface

Sixteen key switches are connected in a matrix of 4 rows and 4 columns. The keys are numbered 0, 1, 2 ....9, A, B, C, D, E and F. To get the data from the keyboard the following three tasks must be performed,

(1) Detect a key press. (2) Debounce the key press.

(3) Encode the key press (Produce a standard code for the pressed key).

The three tasks can be done with hardware, software or the combination of the two.

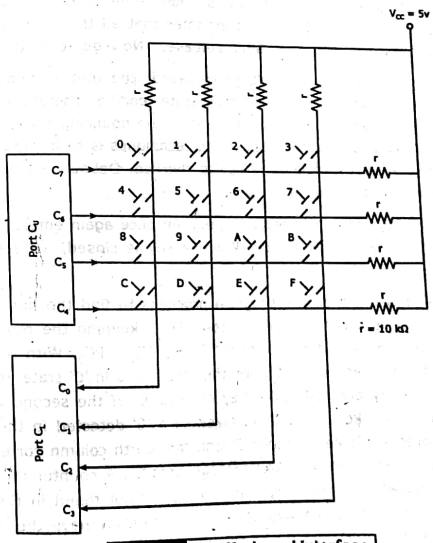


Fig. 5.1.4 Hex Keyboard Interface

Fig. 5.1.4 shows how a hexadecimal keypad can be connected to a couple of microcomputer ports so that the three interfacing tasks can be done as part of a program. Let us use the programmable peripheral interface chip 8255 for this purpose. The 8255 is initialized so that port C_L acts as input port and port C_U acts as output port. The rows of the matrix are connected to four output lines of port C_U, namely C_4, C_5, C_6 and C_7. The column lines of the matrix are connected to four lines of port C_L which are C_0, C_1, C_2 and C_3.

When no keys are pressed, the column lines are held high by the pull-up resistors connected to +5V. Pressing a key connects a row to a column. If a logic '0' is output on a row and a key in that row is pressed, then the column which contains that key also goes to '0' state. This can be detected on the input port. If we know the row and the column of the pressed key, we then know which key was pressed, and we can convert this information into any code we want, to represent that key.

The required algorithm is given step by step,

(1) Give '0's to all rows through the output port C_U and check the columns through port C_L. If any of the columns reads '0', it means that a key already pressed is not yet released. In that case the program enters a loop repeatedly checking the columns. Once all the columns read '1' state, it indicates that all the keys are in open state and any key pressed earlier has been released. Now go to next step.

(2) When a mechanical key is pressed or released, the metal contacts of the key momentarily bounce before giving a steady state reading. Therefore, it is necessary to eliminate the bounce before reading the key. The bouncing time is approximately 20 ms (milliseconds). One of the debouncing techniques is to introduce a time delay of about 20 ms. After the delay, the key is checked. Debounce delay is introduced every time a key is pressed or released.

(3) Now with all the keys in open state, the program once again enters a loop, but this time waiting for a key to be closed. When a key is closed, a debounce delay is introduced.

(4) When a key is pressed, then the program proceeds to find the number of the key pressed. First a '0' is given to the first row (PC_7), keeping the remaining rows in '1' state. The columns are checked for a '0' with PC_0 – PC_3. With reference to the diagram Fig. 5.1.4 we can see that when the first row is in '0' state and first column reads '0' then the key pressed is the key marked '0'. If the second reads '0', then the key pressed is the key marked '1'. Likewise a '0' detected in the third column means the key pressed is key number '2' and the fourth column corresponds to key number '3'. The columns are checked through port C_L. A counter is initialized with zero and incremented by one every time the key is not found in the column that is being checked. This way, the counter will automatically have the number of the key that is to be detected.

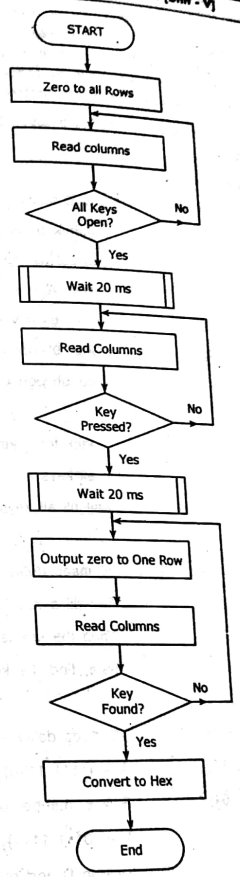


Fig. 5.1.5 Hex Keyboard Interface

(5) When the key is not found in the first row, a '0' is given to the second row (PC_6) and the columns are checked one by one to see whether the key pressed is 4, 5, 6 or 7. In the second row, the counter also correspondingly advances. The procedure is repeated for third and fourth rows. Thus by checking the junctions of four rows and four columns all the sixteen keys in the matrix keyboard can be detected.

The below program detects the key that is pressed and stores the number of the key in a memory location.

Program for Hex Keyboard Interface

```
        MVI   A, 81H      ; A, B and CU to act as
        OUT   F3H         ; output ports and Port CL
                          ; to act as input Port
        XRA   A           ; Make A = 0 and E = 0
        MOV   E, A        ; E stores key number
        OUT   F2H         ; Give '0's to all rows
        NOP               ; through port CU
        NOP
L1 :    IN    F2H         ; Check for a key closure
        ANI   0FH         ; If all keys are not open
        CPI   0FH         ; wait till all keys are open
        JNZ   L1
        CALL  DEBOUNCE    ; 20 msec delay
L2 :    IN    F2H         ; Wait till a key is pressed
        ANI   0FH         ; When the key is in pressed
        CPI   OFH         ; state, find the key
        JZ    L2
        CALL  DEBOUNCE    ; 20 msec delay
        MVI   A, FEH      ; A = (1111 1110)2
        MVI   B, 04H      ; B = 4, number of rows
L3 :    RRC               ; A = (0111 1111)2
        MOV   D, A        ; Save in D and output this code to
                          ; Port CU to give a '0' to the first
```



```
        OUT   F2H         ; row while the remaining rows
        NOP               ; are in '1' state
        NOP
        IN    F2H         ; Check the columns
        ANI   0FH
        MVI   C, 04H      ; C = 4, number of columns
        RAR               ; Port CL bits are rotated right
L4 :                      ; to find which column
        JNC   L5          ; is in '0' state
        INR   E           ; If a '0' is not found in
        DCR   C           ; first column increment E
        JNZ   L4          ; and check other columns
        MOV   A, D        ; key not in first row
        DCR   B           ; Give '0' to other rows one
        JNZ   L3          ; by one till all rows are
                          ; scanned
L5 :    MOV   A, E        ; The number of the key found
        STA   2250H       ; is stored in memory
        HLT               ; End of program
```

Subroutine Debounce

```
        PUSH  D           ; Save DE contents in stack
        LXI   D, XXXXH    ; Take a suitable value in DE
L6 :    DCX   D           ; Decrement DE
        MOV   A, D        ; Move the contents of D to A
        ORA   E           ; to check if (D) = (E) = 0
        JNZ   L6          ; else continue in loop
        POP   D           ; Retrieve D contents from stack
        RET               ; Return to main program
```

5.1.3 Stepper Motor Control System

A stepper motor is a specially constructed DC motor. It has 4 windings arranged such that instead of running in the usual continuous fashion the motor rotates in precise steps, from one fixed position to another. The common step sizes range from 0.9 to 30 degrees. The stepping action is achieved by proper combination of fields. Fig. 5.1.6 shows the schematic of the stepper motor windings.

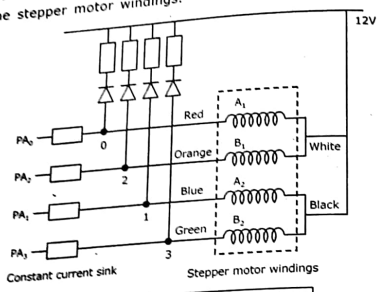


Fig. 5.1.6 Stepper Motor Windings

Table 5.1.1 shows the pulsing sequence of windings, when bit D0 is in 0 level the coil A1 is energised, and so on. Changing the excitation from A1 and A2 to A2 and B1 causes the shaft to rotate by one step clockwise. Each successive line of the table c one step. Reversing the order in the sequence table causes the shaft to rotate clockwise. Thus, the shaft can be given precise angular rotations.

Table 5.1.1 Switching Sequence for Full Step Rotation

Clock Wise	B2	B1	A2	A1	Anti Clock Wise
	D3	D2	D1	D0	
	1	1	0	0	
	1	0	0	1	
	0	0	1	1	
	0	1	1	0	
	1	1	0	0	

Such motors can be used to provide precise movements as in robot arms, floppy ves, print head movement and paper rotation in computer printers, machine tool control ic. In all these and many other applications stepper motors are used.

The coils require 1 amp current for rotation, at 12V. The 12V power supply should be capable of supplying more current. Transistor circuits are used as driver stages as given in Fig. 5.1.6. The PNP transistor T1 is ON when D3 bit is 0 it is OFF when D3 bit is 1. T2 is ON when T1 is ON, it is OFF when T1 is OFF. When T2 conducts, there is a current through the coil B2. From this it is clear than when D3 bit is zero, coil B2 is energised.

Similarly there are pairs of transistor circuits to energise the coils A1, A2, and B1 as shown in Table 5.1.2.

Table 5.1.2 Bit Assignment

Zero Level Bit	Energised Coil
D0	A1
D1	A2
D2	B1
D3	B2

The pulsing of these stages can be done with the program given. From Table 5.1.1 it can be found that the starting word 1100 is essentially shifted left one bit in each step of the sequence. The PPI chip 8255 on the kit can be used for the purpose of interfacing.

The program starts with the word 1100 and output it through one of the ports. It shifts the word left and outputs it through the same port for each step of rotation of the shaft. For anti-clock wise rotation, the word is shifted right. The number of steps required and the direction of rotation (D0 = 0 for clock wise, 1 for anti-clockwise) are loaded in registers.

```
MVI A, 80          ; Configure 8255 for all ports as output
OUT 43             ; 43 is the CW register
MVI C, XX          ; Counter for number of steps
MVI B ,YY          ; YY = 0 for clockwise YY = 1 for anti-clockwise
MOV A, B
CPI 01
JNZ clock          ; Goto clockwise routine
MVI D, CC          ; Load suitable code for data
```

```
Loop .1 :   MOV A, D
            OUT 40                  ; Send data to port A
            RLC
            MOV
            MOV D, A
            CALL Delay
            DRC C                   ; Decrement the counter
            JNZ Loop 1              ; Check counter = 0
            HLT                     ; Halt
Clock :     MVI D, CC               ; Load suitable code for data
Loop 2 :    MOV A, D
            OUT 40                  ; Send data to port A
            RRC
            MOV D, A
            CALL DELAY
            DCR C                   ; Decrement the counter
            JNZ Loop 2              ; Check counter = 0
            HLT
Delay :     PUSH D                  ; Save registers
            LXI D YYYY              ; Load suitable number for delay neaded
Loop 3 :    DCX D
            MOV A, D                ; Check for 0
            ORA E
            JNZ Loop 3
            POP D                   ; Restore registers
            RET                     ; Return
```

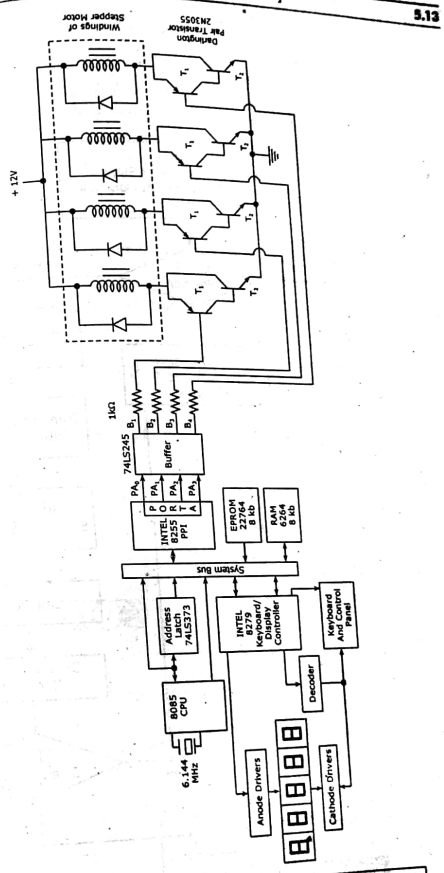



Fig. 5.1.7 Microprocessor based Stepper Motor Control System

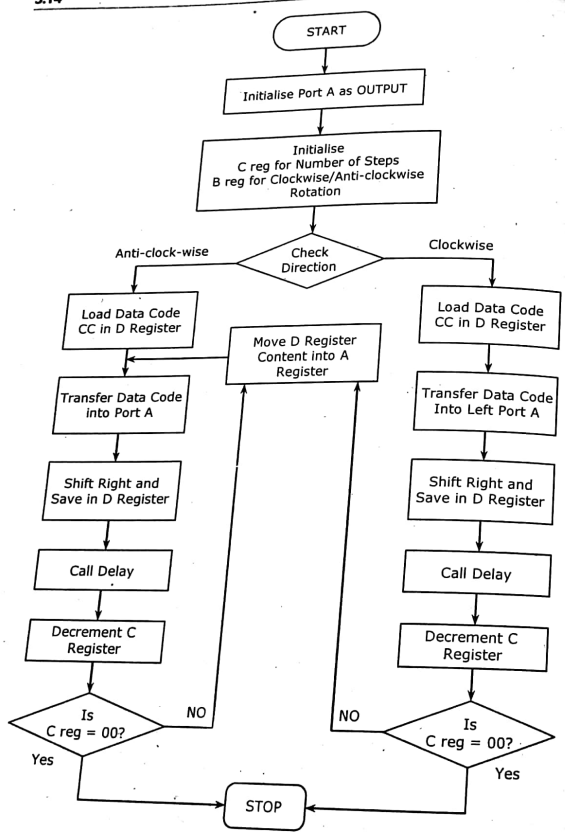


Fig. 5.18 Flowchart for Stepper Motor Control

5.4 Traffic Light Control System

Connecting 6 LEDs to port A and 6 LEDs to port B a simple traffic light control can be simulated and displayed on the LEDs.

Let us imagine a four-road junction. Two roads run along East-West direction and two roads run along North-South direction. A simple traffic light control can be arranged assuming the following condition,

1) Whenever traffic is allowed in the East-West direction, traffic is stopped in the North-South direction.

2) Whenever traffic is allowed in the North-South direction, traffic is stopped in the East-West direction.

Red, Yellow and Green lamps are provided on one side of each road.

Red indicates 'Stop'.

Yellow indicates 'Alert' / 'Change' / 'Ready'.

Green indicates 'Go'.

This arrangement is shown in Fig. 5.1.9.

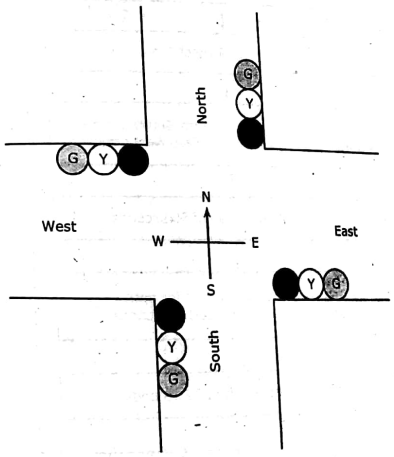


Fig. 5.1.9 Traffic Light Arrangement

Whenever red lamps glow on the East-West road, traffic flow is stopped along these roads. Therefore traffic is allowed in the North-South roads and green lamps for these roads glow. After a few seconds, the arrangement is reversed so that traffic is allowed in the East-West road and stopped along North-South road. The lights change color for these roads.

For the four sets of lamps, whenever light changes from red to green or green to red, yellow lamps are allowed to glow for a few seconds to avoid abrupt changes.

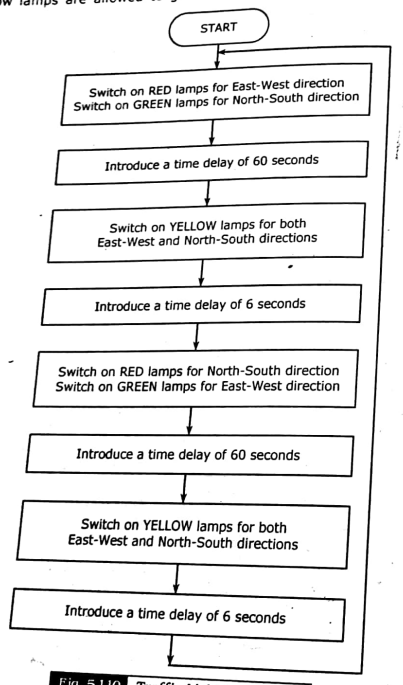


Fig. 5.1.10 Traffic Light Simulation

Lamps can be connected to lamp drivers, which switch on the lamps using thyristors / relays. But the input to the lamp drivers can be logic '0' or logic '1'. The microprocessor can be interfaced to standard output ports or programmable peripheral devices like 8255 / 8155.

The sequence of operation to be performed can be given in the form of flow chart given in Fig. 5.1.10.

Let, us assume that an 8255 is interfaced with 8085. Port A, Port B and Port C are configured as output ports. The outputs are connected to lamp drivers as shown in Fig. 5.1.11.

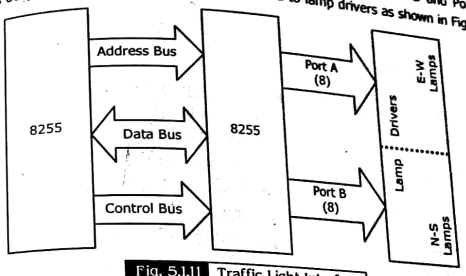


Fig. 5.1.11 Traffic Light Interface

Let us arrange the lamp connection as shown below,

PA_7	PA_6	PA_5	PA_4	PA_3	PA_2	PA_1	PA_0
X	R_W	Y_W	G_W	X	R_E	Y_E	G_E

ie., Port A controls the lamps for East-West direction.

Assuming '1' for ON and '0' for OFF, the bit combination,

0 1 0 0 0 1 0 0 = 44_H

will make red lamps 'ON' on the East-West road.

Similarly, the bit combination,

0 0 1 0 0 0 1 0 = 22_H

makes the yellow lamps 'ON" on the East-West road.

The bit combination,

0 0 0 1 0 0 0 1 = 11_H

makes the green lamps 'ON' on the East-West road.

The same arrangement can be repeated for North-South direction using Port B.

Therefore, the combinations used are,

44H for stop (Red)
22H for Ready/Alert (Yellow)
11H for Go (Green)

The same combinations can be used in Port A for East-West direction and in Port B for North-South direction.

Let us assume the port addresses of 8255 as follows,

Port A = $F0_H$
Port B = $F1_H$
Port C = $F2_H$
Control register = $F3_H$

Just following the flow chart given in Fig. 5.1.10, the assembly language program for traffic light control can be written as shown below,

```
        MVI   A, 80H     ; Initialize 8255 so that ports A, B and C
        OUT   F3H        ; act as output ports
START : MVI   A, 44H     ; Switch ON red lamps for East-West
        OUT   F0H        ; direction
        MVI   A, 11H     ; Switch ON green lamps for North-South
        OUT   F1H        ; direction
        CALL  DELAY1     ; Delay for 60 seconds
        MVI   A, 22H     ; Switch ON yellow for both East-West and
        OUT   F0H        ; North-South directions
        MVI   A, 22H
        OUT   F1H
        CALL  DELAY2     ; Delay for 6 seconds
        MVI   A, 44H     ; Switch ON red for North-South
        OUT   F1H        ; direction
        MVI   A, 11H     ; Switch ON green for East-West
        OUT   F0H        ; direction
        CALL  DELAY1     ; Delay for 60 seconds
        MVI   A, 22H     ; Switch ON yellow for both East-West and
        OUT   F0H        ; North-South directions
        MVI   A, 22H
        OUT   F1H
        CALL  DELAY2     ; Delay for 6 seconds
        JMP   START      ; Repeat
```


5.1.3 D.C Motor Speed Control System

The microprocessor based speed control system can be used to automatically control the speed of the motor. The speed of the D.C motor is varied by varying the armature voltage and the field voltage is kept constant. A controlled rectifier using SCRs develops the required armature voltage and the uncontrolled rectifier generates the required field voltage from the A.C supply. The microprocessor controls the speed of the motor by varying the firing angle of SCRs in the controlled rectifier. A keyboard and display in the system allow the operator to enter desired speed and display the actual speed.

The microprocessor based D.C. motor speed control system is shown in Fig. 5.1.12.

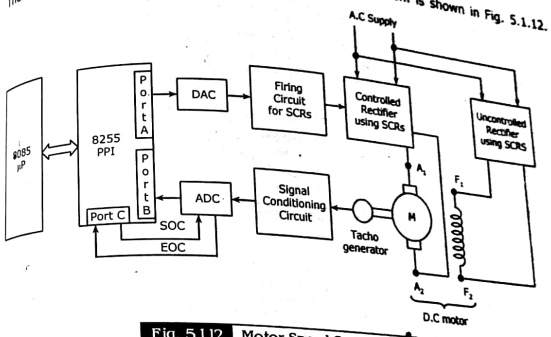


Fig. 5.1.12 Motor Speed Control System

The speed of the D.C. motor is measured using tacho generator. It produces an analog voltage proportional to the speed of the motor. Then the analog signal is scaled to desired level by the signal conditioning circuit and converted into digital signal using Analog-to-Digital Converter (ADC). The ADC in interfaced to 8085 microprocessor through the Port B and Port C of 8255, Programmable Peripheral Interface (PPI). The microprocessor can send a Start-Of-Conversion (SOC) to ADC through Port C. It can read the digital data from Port B of 8255. This digital data is proportional to actual speed of the D.C. motor.

Thus, the microprocessor calculates the actual speed (N_A) and display it. Also the microprocessor compares the actual speed with the desired speed (N_D) entered by the operator. If there is a difference between N_A and N_D then the error is calculated. This error is modified by a digital control algorithm (PI or PID or Fuzzy logic) to produce a digital control signal. This digital signal is converted to analog signal by the Digital-to-Analog Converter (DAC). The analog control signal is used to vary the firing angle of SCRs in the controlled rectifiers.

The operation of DC motor speed control system using microprocessor is shown in the flowchart of Fig. 5.1.13.

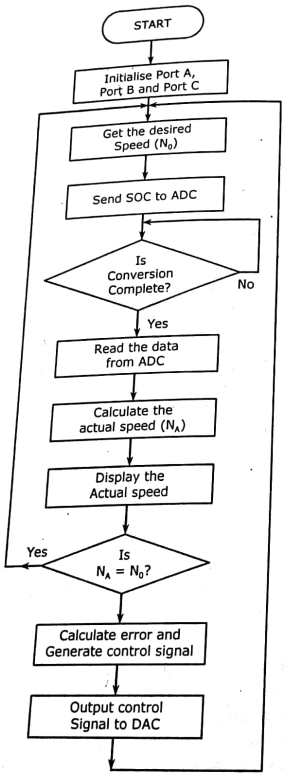


Fig. 5.1.13 Flow Chart for D.C Motor Speed Control

5.1.6 Other Applications

Science and Technology : While the development of computers is itself an outcome of remarkable developments in science and technology, it is now being increasingly used to further scientific and technological advancements. The importance of speed, precision, accuracy in sensitive areas are being accounted for effectively and being monitored through computers. Its wide application is found in space probes, chemical experiments, navigation, meteorology, astronomy etc.

Personal Computers : Personal computers are low cost microcomputers. These personal computers are very useful for general purpose computation. These will be widely used in offices in the near further. Personal computers are available with CRT and floppy disk/ hard disk system. A television screen can also be used in place of CRT resulting in further reduction in cost. The cost of personal computer lies some where in the range of ₹ 25,000 /- Personal computers are programmed in assembly language as well as in high level languages. Compilers for various high-level languages stored in floppy disks are available for personal computers. One can work in FORTAN, BASIC, PASCAL (or) COBOL. One has to use the compiler for the language which is desired. Personal computers are designed to operate in a number of high-level languages. The CPU of personal computers which are available today in Indian market are INTEL 8088 or 80286. Some personal computers are available with two floppy disk drives and some with one hard disk drive and one floppy disk drive system. The 640 KB memory is used in these types.

Word Processor : Word processors are now used in a office, press and communication system. It is microcomputer specially designed for bulk typing and repeated typing work. It also provides the facility for storage of the text and editing of the text. For mass storage, a magnetic device known as floppy disk, is commonly used with a word processor. The word processor is now also used to press to facilitate book printing. Where word processor is used, conventional composition for texts are not used. The text is entered into the word processor from the manuscript. This is displayed on the CRT screen. The operator checks and correct errors. The text is stored in a floppy disk and prints are taken out. The prints are sent to the author for correction. The operator again displays the text (Which was stored in the floppy disk) on the CRT screen. Now he makes correction again as pointed out by the author, if any. Now operator takes out final prints of the text. The final prints are used for the preparation of plates for offset printing.

Teletex System : A present telex system of communication is widely used in offices and business organisations. The process of preparing and transmitted information on telex is time consuming. A new method known as teletex has been introduced for communication. Word processors are used in the teletex system. One word processor information to another word processor. The new teletex system is ten times faster isting telex system.

Education and Research : The use of computers at various levels of academics and education have helped to impart knowledge to students in a highly scientific and practical manner. Individual student records can be maintained and the structure of lesson plans and course materials can be suitably adjusted to the student's individual requirements. Computer Aided Instruction (CAI) or Computer Aided Learning (CAL) are comprehensive terminologies to describe the imparting of eduction through computers in class room. Computers are also used to mark multiple choice questions in various examination. The calculation of results and percentages are also greatly aided by computers. Computers are also being used in research centers and laboratories of medical, scientific, technological, industrial and commercial kinds. They help to efficiently maintain relevant results which can be quickly and easily accessed or retrieved and often leads to breakthroughs, inventions and innovations, discoveries and new policies which would normally have taken a much longer time.

Commerce and Industry : The microprocessor based system have number of applications in industry. For example, we take power plant control. A microprocessor-based system indicates the voltage, power, power factor, current etc., of each generator on CRT. If there is any deviation from the normal value, it indicates the abnormal condition by flickering light. It also corrects them automatically and gives alarm if required. A microprocessor-based system is used for automatic tripping of the faulty part of system, if there is short circuit or some other type of abnormality. Similarly, boiler temperature and pressure measurement and display, their automatic control, if they deviate from normal values, are done by a microcomputer. A microprocessor based system or a microcomputer being expensive forms a part of the equipment to be controlled. They also help in administrative reports such as payrolls, stock control and replacements, employee and customer details, invoices etc. They are also used in long term planning to forecast buying/selling, demand/supply, profit/loss trends and help in developing suitable policies. The effective use of computers in financial accounting has facilitated the transaction of goods and commodities without monetary exchanges i.e., by a system of credit and debit operations through credit cards.

Law and Order : The advantage of speed, accuracy and wide networks made available through the use of computers, have been suitably exploited by the police and investigation departments to keep a tab on criminals and convicts on the run. Minute details of speech information, physical features, fingerprints and ballistic tests can be matched for accurate detection through computers.

Lawyers, who are known to be in possession of shelves and shelves of books that over the walls of their rooms, can now avail of the use of computers by storing large volumes of cases and judicial proceedings on very effective storage media, which can be easily and directly accessed or retrieved.

Transport and Communication : The flow of vehicular traffic, cargo, people in transit from one place to another, are being effectively monitored through computers to prevent crowding, traffic jams, overloading and avoidable delays. The use of microcomputer for flight, train and bus reservation make the booking of tickets easy and quick. For plane reservation microcomputers are connected to a super4computer via satellite. The officer incharge can get informations from other cities of the world regarding the availability of seats in plane for a particular data. They also get informations about hotels, climate etc. of other places. Hotels are also booked for passengers immediately. All these are done within a few minutes. This system of reservation is also being used in railways. One can reserve his seat for ownward journey from his home town. Microcomputers are also used for railway signaling and other controls, and for providing informations of trains timing etc. These types of microprocessor-based systems are being introduced in our country. Computers also help to cover large distances through telecommunication and satellite links. The increasing incorporation of computer technology in these areas has helped to handle the increase in the number of telephone calls, international subscriber dialing facilities, television, radio broadcasts and mobile phones etc.

Medicine : The last few decades have witnessed a large number of discoveries in the field of medicine. A lot of credit can be attributed to the use of computers in medical tests and experiments. Apart from, this computer have wide applications in hospitals, in keeping records of patients case histories, the symptoms of rare and common diseases and their corresponding medication, monitoring heart-beat, pulse, blood pressure and other such vital human features through computerized devices such as ECG monitors and CAT scanner. They are also used to monitor the availability of feds, Icoation of doctors and patients in large hospitals, calculation of hospital staff payrolls and patient's medical fees etc.

Home and Entertainment : At home, personal-computers are being used to keep personal banking details, addresses of friends and even kitchen-store items. The automatic washing machine too are apt example of the application of computer-based technology at home. In the field of entertainment, computers are used in video games, toys, music etc.

A wonderful application of the microprocessor-based system is the 'ROBOT'. Robots have mechanical hands and legs, and optical sensors to see obstacles. They are controlled by microcomputer. They perform tasks according to the programs fed to it. Programs can be fed using keyboards fitted in the robot. Robots are used as domestic servants. They can do many works in hospitals, which are presently done by nurses. In industry, they can do the work in a situation where human being can't work, for example, to work in a place where temperature is very high, to handle chemicals which can't be touched by operator, to work in a place where huclear radiation is enough to cause damage to the operator, etc.

Thus, it is seen that microcomputers/computers have numerous applications and they are going to have a great impact on every aspect of our lives in the near future.

5.2 TRENDS IN MICROPROCESSOR TECHNOLOGY

The first commercially available microprocessor was the Intel 4004, a 4-bit processor, produced in 1971. In 1972-73, Intel came out with 8008, an 8-bit processor. After further modification, 8080 and 8085 were introduced, which are advanced versions of 8-bit processors. By mid 70s, Intel again introduced a revolutionary 16-bit microprocessor Intel 8086/88.

8086 was the first 16-bit microprocessor. Intel 80186 is an upgraded version of 8086. Similarly 80286, 80386, 80486 and pentiums are also upgraded versions of their predecessors. 80386 is a 32-bit microprocessor developed by Intel. In 1989, 80486 (32-bit) was introduced by Intel. The PENTIUM microprocessor was developed by INTEL in 1992. Later Intel pentium 4 was introduced.

Since, the early eighties we have witnessed a parallel development of a new architectural trend : the Reduced Instruction Set Computer (RISC). The Intel x86 and Motorola M68000 families belong to what is considered to be the opposite class of RISC : the Complex Instruction Set Computer (CISC). The chronology of development of Intel, Motorola, and some RISC systems summarized in Table 5.2.1.

Table 5.2.1 Microprocessor Chrolonology

Year	Intel	Motorola	RISC
1971	4004		
1972	8008		
1974	8080	6800	
1975			IBM 801 start
1976	8085		
1977		6809	
1978	8086		
1979	8088	68000	
1980	80186	68008	UCB RISC announced
1981	80188		Stanford MIPS
1982	80286		IBM announced
1983		68010	Transputer
1984		68020	
1985	80386		
1986			MIPS R2000, IBM ROMP, HP-PA Am29000, SPARC
1987		68030	M88000, R3000, i860, R6000
1888	80376		Am29050, RS/6000, R4000, MC88110
1989	80486	68040	Alpha, R4400, Super SPARC, Power PC 601
1990			
1992			
1993	Pentium		
1994		68060	

Note

(1) HP-PA = Hewlett-Packard Precision Architecture.

(2) MIPS MIPS = Microprocessor without interlocked pipeline stages.

(3) RS = RICS system.

(4) UCB = University of California, Berkeley.

5.2.1 8-Bit Microprocessor

In 1973, Intel introduced Intel 8008 its first 8-bit microprocessor. This microprocessor uses PMOS technology. In 1973, a more powerful and fast 8-bit microprocessor, Intel 8080 by intel Corporation USA. 8080 uses NMOS technology so they are faster and compatible with TTL. The Intel 8080 was widely used in control application and small computers were also designed using the Intel 8080.

In 1975, Intel developed another 8-bit microprocessor, It was Intel 8085 which uses NMOS technology and uses only single +5V power supply. Other companies also developed 8-bit microprocessor. Motorola's MC6800 and MC6809, Zilog's Z80 and Z800, National Semi-conductor's NSC800, Rockwell Internationals PPS-8 also developed in the same year.

5.2.1.1 Zilog Microprocessor

The ZILOG Z80 is an 8-bit microprocessor, manufactured with NMOS technology. The Z80 is available in a 40-pin DIP (Dual In-line Package). It requires a single external clock and a single 5V power supply. The maximum internal clock of standard Z80 is 2.5 MHz and for Z80-A it is 4 MHz. The Z80 provides more registers, extra addressing modes and a much larger instruction set than 8085. It also has built-in-logic to refresh dynamic RAM memories.

The signals of Z80 are shown in Fig. 5.2.1. The Z80 communicates with other system modules via three functionally separate buses : data, address and control buses.

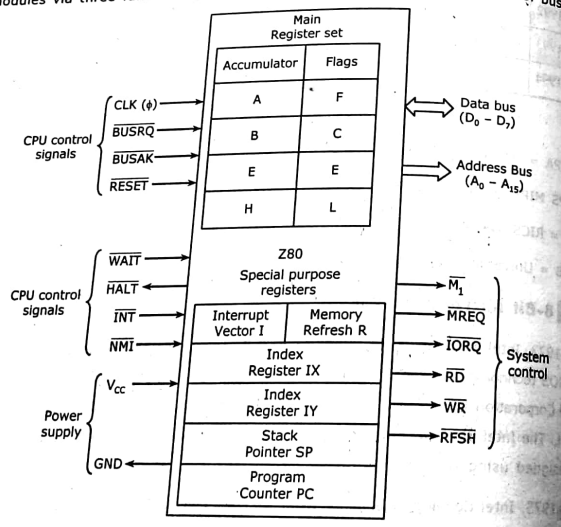


Fig. 5.2.1 Signals of Z80

It operates on 8-bit data and uses 16-bit memory address. The physical memory size of Z80 system is 64kb.

The control bus has three types of control signals. They are listed below,

(1) **System Control Signals**

- $\overline{M_1}$ - First machine cycle of an instruction.
- $\overline{MREQ}$ - Memory request.
- $\overline{IORQ}$ - I/O request.
- $\overline{RD}$ - Read control.
- $\overline{WR}$ - Write control.
- $\overline{RFSH}$ - Refresh cycle.

(2) **CPU Control Signals**

- $\overline{WAIT}$ - Wait request.
- $\overline{HALT}$ - Halt request.
- $\overline{INT}$ - Interrupt request.
- $\overline{NMI}$ - Non-maskable interrupt.

(3) **Bus Controls Signals**

- $\overline{BUSRQ}$ - BUS request.
- $\overline{BUSAK}$ - BUS acknowledge.
- $\overline{RESET}$ - System reset.
- $\overline{CLK}$ (ϕ) - Clock input.

The ALU is eight bits wide and performs similar functions to those of the 8085 ALU. The Z80 has two independent 8-bit accumulators A and A' and two independent flag registers F and F'. The ALU operation involving accumulator A affects the flag register F. The ALU operation involving accumulator A' affects the flag register F'.

The flag registers has six flags and they are sign (S and S'), zero (Z and Z'), carry (C and C'), parity/over flow (P/V and P'/V'), half carry (H and H') and subtract (N and N').

The Z80 has two sets of 8-bit general purpose register. Each set has 6 registers. They are B, C, D, E, H & L and B', C', E', H', & L'. They can be used individually as 8-bit registers or as 16-bit register pairs. The allowed pairs are BC, DE and HL and B' C', D' E' and H' L'.

The 16-bit Program Counter (PC) and Stack Pointer (SP) registers are same as that of the 8085 microprocessor and operate in exactly the same way. The registers IX and IY allows two independent Indexed addressing mode.

The Z80 includes an 8-bit interrupt vector (I). It is used in one of the interrupt response mode of the processor. It holds the upper eight bits of a memory pointer or vector address. The lower eight bits of this pointer are supplied (As a vector number) by the interrupting device that requests service. The CPU then uses this 16-bit vector address to make an indirect call to the memory location that holds the first instruction of the interrupt service routine. This feature allows the vector table to be located any where in memory.

The Z80 also contains an 8-bit memory refresh Register (R) that contains the current memory refresh address, thus, providing for automatic, totally transparent refresh of external dynamic RAM memories. Although the programmer can load this register for testing purposes, the R register is not normally used by the programmer.

The Z80 can execute 158 instruction types. The microprocessor includes all the instructions of 8080A microprocessor with total software compatibility at the machine code level.

The size of Z80 instruction is one to four-bytes. One byte instruction has 1-byte opcode alone. The 2-byte instruction has 1 or 2 byte opcode plus data byte/device number/ displacement.

Every Z80 instruction consists of one to six machine cycles. All types of machine cycle consist of either three or four states. Some Z80 instructions always insert wait states (T_w) between the states T_2 and T_3. The basic operation of the Z80 is analogous to that of the INTEL 8085. The main difference is that instead of IO/$\overline{M}$ of 8085, the Z80 has $\overline{MREQ}$ and $\overline{IORQ}$. They are activated along with $\overline{RD}$ and $\overline{WR}$ for the memory or I/O access.

5.2.1.2 Motorola 6800

The Motorola 6800 product family is originally introduced in 1974. The 6800 microprocessor CPU is manufactured in NMOS technology on a 40-pin chip, has TTL compatible pins and it is the first 8-bit single chip microprocessor to exploit a single 5V power supply. The 6800 CPU can drive from seven to ten 6800 family devices x without buffering. A two-phase external clock (1MHz, maximum) must be externally supplied.

The 6800 CPU has three buses to communicate with the other system modules, they are data, address and control buses. The data bus is bidirectional and has 8-lines, D_0 – D_7. The address bus has 16 lines, A_0 – A_{15}. The processor operates on 8-bit data and uses 16-bit address for memory and I/O devices.

The control bus carries two types of signals called control bus signals and CPU (microprocessor) supervisory signals,

(1) Control Bus Signals

VMA	-	Valid memory address.
R/$\overline{W}$	-	Read/Write control.
$\overline{IRQ}$	-	Interrupt request.
ϕ_2	-	Phase-2 of clock.
$\overline{RESET}$	-	System reset.

(2) CPU Supervisory Signals

BA	-	Bus acknowledge.
$\overline{HALT}$	-	Halt request.
TSC	-	Tristate control.
DBE	-	Data bus enable.
$\overline{NMI}$	-	Non-maskable interrupt.
ϕ_1	-	Phase-1 of clock.
$\overline{IRQ}$	-	Interrupt request.
$\overline{RESET}$	-	System reset.

The architecture of 6800 includes the ALU, 16-bit Program Counter (PC), 16-bit stack pointer, 16-bit index or general purpose register, two 8-bit accumulators and a condition code register.

The 6800 has a set of seventy two instructions. They are classified as data handling, arithmetic, logic, control transfer, data test, condition codes, address maintenance and interrupt handling.

A 6800 instruction may be one, two or three bytes long, its length being closely related to the addressing mode used.

Usually every 6800 instruction cycle consists of two to eight machine cycles, all of which are identical in length (Except interrupt instructions which require longer instruction execution cycles). In the 6800, a machine cycle is one and the same thing as a clock cycle or state.

5.2.1.3 Hitachi HD64180

The HD64180 is an 8-bit MPU that is based on microcoded execution unit and advanced CMOS manufacturing technology. It was developed by Hitachi. It is a partially dynamic Z80 with on-chip peripherals. It provides the benefits of high performance, reduced system cost and low power operation while maintaining complete compatibility with the large base of industry standard 8-bit software.

Performance is derived from a high clock speed, pipelining, enhanced instruction set and integrated Memory Management Unit (MMU) with 1M or 512 k bytes of memory physical address space. The HD64180 requires operation at specific frequencies inorder to generate standard data transmission rates. System cost is reduced due to key system functions incorporated on-chip including MMU, two channel Asynchronous Serial Communication Interface (ASCI), clocked serial I/O port two channel 16-bit Programmable Reload Timer (PRT), Interrupt controller and dual bus interface.

The MMU maps the CPU's 64 k byte logical memory address space into a 512 k byte physical memory address space. The MMU preserves software object-code compatibility while providing extended memory access and uses an efficient common area scheme. The two channel Direct Memory Access Controller (DMAC) provides high speed memory-to-memory memory-to-I/O and memory-to-memory mapped I/O transfer. The DMAC can directly access the full 512 bytes of physical memory address space.

The Asynchronous Serial Communication Interface (ASCI) provides two full-duplex UARTs (Universal Asynchronous Receiver/Transmitters) and includes a programmable data-transmission rate generator, modem control signals. The clocked serial I/O port provides a half-duplex clocked serial transmitter and receiver. The Programmable Reload Timer (PRT) contains two separate channels, each consisting of 16-bit timer data and 16-bit timer reload registers.

Features

(1) On-chip Memory Management Unit (MMU) which supports 1M bytes of memory.

(2) Two channel Direct Memory Access Controller (DMAC) which supports memory-to-memory, memory-to-I/O and memory-to-memory mapped I/O transfers.

(3) Two channel, full duplex Asynchronous Serial Communication Interface (ASCI) with programmable baud rate generator and modem control handshake signals.

(4) Two channel 16-bit Programmable Reload Timer (PRT) for output waveform generation.

(5) One channel clocked serial I/O port with serial/parallel shift register.

(6) Four external and eight internal interrupts.

(7) On-chip clock generator.

(8) Operating frequency upto 10 MHz.

5.2.2 16-Bit Microprocessors

The 8-bit microprocessor's were followed by 16-bit microprocessors. In 1978, Intel introduced Intel 8086, a 16-bit microprocessor. Example of other 16-bit microprocessor, are intel 80186, Intel 8088, Intel 80188, Intel 80286, Motorola 68000, Zilog Z8000, Fairchild 9440 etc.

16-bit microprocessors have larger address bus and larger main memory that can accommodate more software as well as larger software like compilers. So, use of high level languages like Pascal, BASIC, FORTRAN is possible. 16-bit microprocessors have powerful instructions like integer multiplication, division and floating point instructions.

5.2.2.1 Intel 8086/88

This family consists of two types of 16-bit multiprocessors, Intel's 8086 and 8088. The main difference between 8086 and 8088 is how the processors communicate with the outside world. The 8088 has an 8-bit external data path to memory and I/O, while the 8086 has a 16-bit external data path. This means that the 8088 will have to do two READ operations to read a 16-bit word into memory. In most other aspects, the processors are identical. It may be note that the 8088 accesses memory in bytes. No alterations are needed to run software written for one microprocessor on the other. Because of similarities between these two processors, we will restrict our discussion only on the 8086. The 8088 is used in designing the IBM personal computer.

5.2.2.2 Intel 80186

The 80186 is a 16-bit microprocessor, which is operated at a clock frequency of 8 MHz and +5V dc supply, and has 68 pins. The basic architecture of 80186 is similar to 8086. It can execute all the instructions of 8086 microprocessor, but includes few new instructions. Even the same instructions of 8086 run faster in 80186, because hardware enhancement in the execution unit and bus interface unit.

The 80186 has 6 extra features on its chip than 8086. The extra features integrated on its chip are,

(1) Clock generator.

(2) Two channel DMA unit.

(3) Interrupt controller.

(4) Programmable timer/counter.

(5) Chip select logic.

(6) Ready control logic.

As the 80186 is a 16-bit microprocessor it has 16-bit external data bus. Like 8086, the 80186 also has 20-bit address bus, which makes it capable of addressing 1MB memory. With the addition of few new instructions, the instruction set of 80186, is upward compatible with an 8086. The additional instructions are included to simplify assembly language program and operations with high level languages.

5.2.2.3 Intel 80286

The 80286 is a high performance microprocessor with memory management and protection. It has 68 pins and can operate at clock frequencies of 8 MHz, 10 MHz and 12.5 MHz, with power supply of +5V dc. It has 24 address lines and 16 data lines. Due to 24 address lines, it has memory addressing capability of 2^{24} = 16 Mbytes. The internal memory management feature increases the storage space to 1 Giga bytes of virtual space. The concept of ***virtual memory*** was introduced in 80286.

The 80286 is an advanced version of 8086. It is designed for multiprogramming/ multiuser/multitasking environments. It provides separation between code and data modules. In fact, an 80286 is an 8086 that is optimized to execute instructions in fewer clock cycles, that makes 80286 faster. The 80286 operates in two modes : real address mode and Protected Virtual Address Mode (PVAM). In both modes processor operates with full performance. The 80286 provides special operations to support the efficient implementation and execution of operating systems. It means one instruction can end execution of one task, save its state, switch to a new task, load its state, and start execution of the new task.

5.2.2.4 Motorola MC 68000

The MC68000 is the Motorola's first 16-bit microprocessor. It has 16-bit data bus and 24-bit address bus to address upto 16 Mb of physical memory space. The 16 Mb physical memory space is organised as two banks of 8 Mbs each. The MC68000 is designed using HMOS transistors and requires a single +5V supply.

In 68000 family Motorola has released a number of processors which share a common base architecture but differ in data bus size, address bus size, instruction set, operating system support and performance.

The 68000 does not have on-chip clock circuitry and hence it require an external clock generator to generate the required clock and supply to the processor. Its maximum internal clock is 25 MHz. The 68000 is available with maximum clock rating of 6, 8, 10, 12.5, 16.67 and 25 MHz.

The 68000 has general register based architecture and in this architecture any register can be used as an accumulator or scratch pad register. The internal address and data registers of 68000 are 32-bits wide and its ALU is 16-bit wide. It operates on five different data types and they are 4-bit BCD, 8-bit, 16-bit and 32-bit binary data. The 68000 has 56 basic instructions and has more than 1000 opcodes. It has 14 addressing mod and supports only memory mapped I/O.

The 68000 has two operating modes and they are,

(1) Supervisor mode.

(2) User mode.

The supervisor mode is also called the operating system mode, and in this mode the processor can execute all the instructions. Upon hardware reset the 68000 enters supervisor mode. The processor can switch from supervisor mode to user mode by clearing the S-bit in the status register. The processor can switch from user mode to supervisor mode by recognization of a trap/reset/interrupt.

5.2.3 High End-High Performance Processors

The 32 and 64-bit microprocessors are available at high end of microprocessor range. These processors include Intel 80386 and 80486, Pentium processors and RISC processors.

5.2.3.1 Intel 80386

The 80386 is an advanced 32-bit microprocessor designed for applications requiring high performance. This processor is a logical extension of 80286 and is optimized for multitasking operating system. The 32-bit registers and data paths support 32-bits and data types. The processor addresses upto 4GB of physical memory and 64 terabytes of virtual memory. The intergrated memory management and protection architecture includes address translation registers, advanced multitasking hardware and a protection mechanism to support operating system. Additionally, the 80386 allows the simultaneous running of multiple operating systems. The instruction pipelining, on chip address translation, and high bus band width ensure short average instruction execution time and high system throughput. The 80386 offers new testability and debugging features. Testability features include a self test and direct access to the page translation cache.

Register Organisation of 80386 : The register sizes are extended to 32-bits, in 80386 microprocessor. These new extended registers are referred to as EAX, EBX, ECX and EDX. Still lower 16-bit registers are referred as AX, BX, CX and DX to support previous versions of Intel microprocessors, which can be still divided into 8-bit registers, such as AH and AL. The Index (SI and DI) and Pointer (IP, BP and SP) registers are also extended to bits (as ESI, EDI, EIP, EBP and ESP). Two additional segment registers FS and GS are added. These extended registers are not available in real mode.

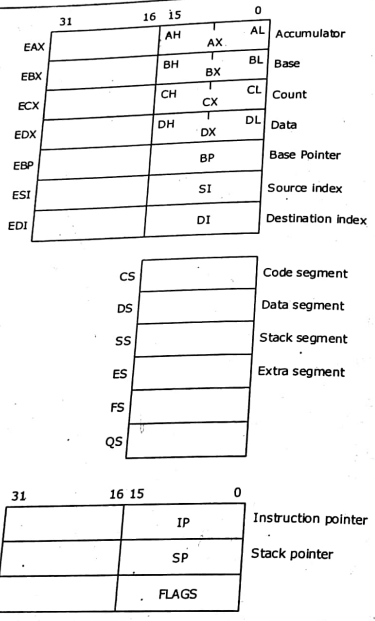


Fig. 5.2.2 Register Organisation of 80386 and Higher Microprocessor

Flag register is also extended to 32-bit register for protected mode. Two additional flags VM and RF.

D_{31}	D_{30}	D_{29}	D_{28}	D_{27}	D_{26}	D_{25}	D_{24}	D_{23}	D_{22}	D_{21}	D_{20}	D_{19}	D_{18}	D_{17}	D_{16}
X	X	X	X	X	X	X	X	X	X	X	X	X	X	VM	RF

VM - Virtual Mode.

RF - Resume Flag.

Fig. 5.2.3 High-Order 16-bits of Flag Register for Protected Mode

The 80386 also contains three 32-bit control registers,

(1) **Machine Control Register :** Contains machine status word and additional bits dealing with the co-processor, paging and protected mode.

(2) **Page Fault Linear Address Register :** It stores the 32-bit address that caused the last page fault, used in virtual environment. When the part of the program, which is required to be executed, is not loaded into the memory, a page fault is generated.

(3) **Page Directory Base Register :** It stores the physical memory address of the beginning of the page directory table.

5.2.3.2 Intel 80486

The 80486 microprocessor is a 32-bit microprocessor, which is highly integrated device containing more than 1,200,000 transistors. It has a Memory Management Unit (MMU), a complete numeric co-processor that is compatible with the 80387 co-processor, a high speed cash memory that contains 8KB of space. The 80486 is upward compatible with the 80386 microprocessor.

The 80486 processor is very much similar to 80386. It is also a 32-bit processor with 32-bit register set, 32-bit data bus and 32-bit address bus. It can also address 4 GB of memory space using the same addressing features. It has all instructions of 80386 with a few additional instructions. Although the features of 80486 are same as 80386, but it has following improvements :

One more flag is added at D_{17} bit of the register, called VM - Virtual Mode.

It has a redesigned internal architecture, which allows many 80486 instructions to execute with fewer clock cycles than those required by 80386. This reduction in clock cycles adds additional speed.

It has added an 8KB cache memory, which is a very high-speed memory, with an access time usually ten times faster than that of conventional RAM. The cache memory is discussed in detail in the next section.

One more improvement in 80386 is that it has internal co-processor. Due to inbuilt co-processor the speed and performance is further improved.

The 80486 has 4 versions : 80486SX, 80486DX, 80486DX2, 80486DX4. The 80486DX2 operates at 66 MHz and 80486DX4 operates at 100 MHz.

5.2.3.3 Intel Pentium Processor

The Pentium is a high performance processor due to its *superscalar architecture* with massive integer pipelining and powerful *on-chip floating-point unit*. The pentium is a 64-bit RISC based microprocessor. It has 32-bit address bus and 64-bit data bus. As other processors, Pentium also has capability to support previous microprocessor. There are two major computer architectures are in use : Complex Instruction Set Computers (CISC) and Reduced Instruction Set Computers (RISC). To maintain the backward compatibility, pentium is designed as the mixture of both CISC and RISC technologies.

It has improved on-chip math co-processor, as compared to 80486, hence increased performance. Its features are,

(1) **Superscalar and Superpipelined Architecture :** It has superscalar architecture, which supports the technique of Multiple Instruction Issue (MII). In the technique of MII, microprocessor is capable of issuing more than one instructions per single processor cycle, for which microprocessor has more than one execution channels. Intel CPU architectures upto 80486, only one instruction is issued to the execution unit per cycle.

It also has superpipelined architecture. It introduces two integer pipelines U and V, where each one is of 4-stage pipeline. These are capable of executing two integer instructions at the same time, under special condition.

(2) **Branch Prediction :** Pentium uses a technique called branch prediction to maintain a steady flow of instructions into the pipelines. In previous microprocessors, the queue is dumped and reloaded, while executing Jump and Call instructions. Hence, in Pentium microprocessor a branch target buffer in introduced to support branch prediction, which maintains a copy of instructions in a different part of program (e.g., subroutine) located at an address called the branch target.

5.2.3.4 Intel Pentium Pro-Processor

The Pentium pro is a 32-bit processor with 64-bit data bus and 36-bit address bus to address upto 64Gb of physical memory space. It is released in the year 1995 and consists of 5.5 million transistors. It is a 387 pin IC and available in PGA (Pin Grid Array) package. It is available with maximum internal clock ratings of 150/166/180/200 MHz.

The features of pentium pro-processor are,

(1) Three way superscalar architecture.

(2) Five parallel execution units and 12-stage super pipeline.

(3) Dual cavity PGA ceramic packages with a CPU die and a secondary cache die.

(4) Out of order execution and speculative execution.

(5) Dual Independent Bus (DIB) architecture.

(6) Register renaming.

(7) Error checking and correcting codes.

(8) Improved power management with two extra modes (Stop Grant and Auto H... modes).

(9) Internal micro-ops similar to RISC like instructions.

(10) Transactional I/O bus.

(11) Scalable upto four processors.

(12) Fault analysis/recovery.

(13) Integrated level two (secondary) cache of 256k/512k/1MB.

(14) Internal thermal protection.

(15) Automatic selection of power supply voltage.

5.2.3.5 RISC Processors

RISC processor uses new technology to design the architecture of the computers, called Reduced Instruction Set Computers (RISC). In this technology, the emphasis is given to make the instruction set smaller, using fewer instructions and simpler addressing modes. This reduced set of operations is easier to implement on silicon, resulting in faster performance.

The RISC microprocessors do not use the microprogramming technique. Some examples of RISC microprocessors are, PowerPC 601, 603, 604, 620, 750, Intel's PA8000, i860, i960, DEC's Alpha 21064, 21164, 21264, MIPS 4000 series, 500 series, 10000, 12000 series, SUN's SPARC, etc.

The aim of the development of RISC processor is to increase computing speed by reducing the execution time of instructions. This can be achieved using hardwired control of the processor. RISC processors use simple and frequently used instructions employing hardwired control technique. The complex instructions can be implemented by computer software. RISC processors are faster and cheaper than CISC processors. The main advantage of CISC processor over a RISC processor is that it can run a large number of already existing software, which is a powerful marketing advantage.

The important features of RISC microprocessor are,

(1) It has a few instruction types and addressing modes.

(2) Instruction set is simpler resulting in increased processing speed. RISC processors have fixed and simplified instruction format. All instructions are decoded by hardware.

(3) Each instruction is executed in one cycle.

(4) Most RISC instructions involve only register-to-register operations. Memory access is limited only to load and store instructions.

(5) A large number of general purpose registers are used to make processing register intensive. Most of the computations are performed using registers rather than memory.

...arge cache memory is employed.

...ardwired control is used and microprogramming is not used.

Table 5.2.2 shows the comparison between RISC and CISC.

Table 5.2.2 Comparison between RISC and CISC

S.No.	RISC	CISC
(1)	RISC stands for Reduced Instruction Set Computer.	CISC stands for Complex Instruction Set Computer
(2)	RISC has relative less instructions, addressing modes and instruction formats. As a result, a relatively small and simple decoding and executing hardware subsystem of the CPU is required.	CISC has relatively more instructions, addressing modes and instruction formats. As a result, a relatively large and complex decoding and executing hardware subsystem of the CPU is required.
(3)	The chip area dedicated to the control unit is less. There is more area available for other features.	The chip area dedicated to the control unit is more. There is less area available for other features.
(4)	Simpler and smaller control unit results in reduced number of design errors and higher reliability.	More complex and large control unit results in increased number of design errors and reduced reliability.
(5)	RISC design approach is suitable for efficient handling of pipelines.	CISC design approach is not suitable for efficient handling of pipelines.
(6)	Throughput is more.	Throughput is less.
(7)	It takes a shorter time to complete the design of a RISC control unit. Due to shorter design time, the chances of the end product becoming obsolete are less.	It takes a longer time to complete the design of a CISC control unit. Due to longer design time, the chances of the end product becoming obsolete are more.
(8)	Overall design cost of RISC control unit is less.	Overall design cost of CISC control unit is more.
(9)	Because of the simplicity, less number of instruction formats and standard instruction lengths, design of virtual memory management subsystem is easier.	Because of the complexity, more number of instruction formats and different instruction lengths, design of virtual memory management subsystem is difficult.
(10)	Since the total number of instructions in a RISC system is small, compiler design is simpler. RISC instruction set presents a reduced burden on the compiler.	Since the total number of instructions in a CISC system is large, compiler design is more complex. CISC instruction set presents an increased burden on the compiler.
(11)	The availability of a relatively large number of CPU registers in a RISC permits a more efficient code optimization stage in a compiler.	CISC system does not provide as many CPU registers as in RISC. Therefore, it does not permit efficient code optimization stage in a compiler.
(12)	RISC compiler is simpler.	CISC compiler is complex.

5.2.4 Single Chip Microcontrollers

The advancement in technology made possible to fabricate a single chip microcomputer called a ***microcontroller***. A microcontroller contains all the essential elements of a microcomputer on a single chip. The components available on a microcontroller chip are CPU, RAM, ROM/EPROM, I/O ports, timer/counters, interrupts, decoders, A/D converters etc.

5.2.4.1 Intel MCS-51 Single Chip Family

The Intel's MCS-51 family includes 8031, 8051 and 8751 single-chip microcomputers. These three microcontrollers use HMOS technology. They provide greater speed, larger program and data memory spaces, more flexible I/O and peripheral capabilities and lower system cost. The 8751 contains 4k bytes of EPROM program memory fabricated on-chip, while the 8051 replaces the EPROM with 4k bytes of lower-cost mask-programmed ROM. The 8031 has no program memory on-chip. Instead it accesses upto 64k bytes of program memory from external memory. Otherwise, the three microcontrollers are identical.

The Intel's MCS-51 is the most widely used 8-bit microcontroller in single-chip microcontroller family. These microcontrollers can operate with a 12 MHz clock frequency. The features of MCS-51 family microcontrollers are,

(1) Four programmable input/output ports.

(2) Two 16-bit timer/counters.

(3) 4k bytes of ROM (or) EPROM.

(4) 128 bytes of data memory.

(5) 21 Special Function Registers (SFRs).

(6) A serial I/O port with a UART.

(7) Five interrupt lines, three for internal operations and two for external signals.

The MCS-51 incorporates a number of special features which support the direct manipulation and testing of individual bits and allow the use of single-bit variables in performing logical operations. Due to these features MCS-51 is regarded as "Boolean processor". Their instruction set includes binary and BCD operations, bit set/reset functions and logical functions. However, their true power comes when they are used with the microcomputer's byte processing and numerical capabilities.

5.2.4.2 Motorola MC68HC11 Microcontroller Family

The popular 68HC11 is a powerful 8-bit data, 16-data address microcontroller from Motorola (the sole supplier) with an instruction set that is similar to the older 68xx parts (6801, 6805, 6809). The 68HC11 has a common memory architecture in which instructions, data, I/O and timers all share the same memory space.

Depending on the variety, the 68HC11 has built-in EEPROM/OTPROM, RAM, digital I/O, timers, A/D converter, PWM generator, pulse accumulator and synchronous and asynchronous communications channels. Typical current requirement is less than 20 mA.

It is an 8-bit microcontroller with on-chip peripherals like timers, serial input/output and 8-bit A/D converter. It has 256 bytes of internal R/W memory which can be further expanded to 64k system memory when it is used in expanded mode. It can operate in four different modes. It can operated with a clock frequency of 2 MHz. It has 40 input/output lines. It is available in 48-pin package.

The features of MC68HC11 microcontroller family are,

(1) On-chip peripherals such as timers, serial I/O and 8-bit A/D converter.

(2) 40-input/output pins with multiple functions.

(3) 16-bit timer.

(4) 256 bytes of internal RAM (R/W) memory.

(5) 8k bytes of ROM and 512 bytes of EPROM.

(6) 8-bit pulse accumulator circuit.

(7) Four operating modes.

(8) Serial communication and serial peripheral interface.

(9) Less current requirement (<20mA).

5.3 INTRODUCTION TO 8086 MICROPROCESSOR

The Intel 8086 CPU is a 16-bit microprocessor and is used as heart of microcomputer. It is N-channel HMOS microprocessor. HMOS means high speed MOS technology, the CMOS version microprocessors are also available for use. This microprocessor works on 5V and max current drawn is 36mA while CMOS version draws only 10 mA. The clock frequency for various versions are 5, 8 and 10 MHz. This design has 29000 transistors as an single chip which has 40 pins.

The term 16-bit means that its internal architecture such as Arithmetic Logic Unit (ALU), internal registers and most of the instructions have 16-bit binary word. The 8086 has 20 address lines and 16 data lines. It can directly address upto 2^{20} = 1 Mbytes of memory locations. The each location is one byte wide or 8-bit location. The word for this microprocessor will be stored in two consecutive memory locations. If first byte of a word is at even location then the CPU can read the entire word in one operation, if the first byte of word is stored in odd address then CPU will read first byte in first operation and second byte in second operation hence, consumes two machine cycles.

The peripheral chips designed for earlier microprocessor (8080 and 8085) can be used with this microprocessor with slight change or no change. The memory addressing techniques are different in this CPU from 8085 and 8080, however interfacing techniques are similar with few additional signals.

The 8086 microprocessor uses 20 address lines which are divided in two parts,

(1) Higher order lines (4-lines).

(2) Low order lines (16-lines).

All these 20 address lines are multiplexed with data lines and status signals.

5.3.1 Features of 8086 Microprocessor

Some of the features of 8086 microprocessor,

(1) 8086 is a 16 bit microprocessor. It means that ALU, its internal registers and most of its instructions are designed to work with 16-bit data.

(2) It is fabricated using HMOS technology, and packaged it in a 40 dual in line (DIP) package.

(3) The 8086 has 20 address lines and 16 data lines. As it uses 20 address lines it can directly address upto 2^{20} = 1 MB of memory.

(4) The 8086 requires an external asymmetric clock source with 33% duty cycle.

(5) The maximum internal clock of 8086 is 5 MHz. The other versions of 8086, 8086-1, 8086-2, 8086-4 have maximum internal clock frequency of 10 MHz, 8 MHz and 4 MHz respectively.

(6) 8086 can generate 2^{16} = 64 KB of accessing I/O mapped devices.

(7) 1 MB of addressable memory of 8086 are organized as two memory banks of 512 KB each, namely even or lower bank and odd or higher bank.

(8) The 8086 can operate in two modes, maximum and minimum mode. In minimum mode, it works as a single microprocessor, whereas in maximum mode it can work in a multiprocessor configuration.

(9) The 8086 has two family of processors. They are 8086 and 8088. The 8088 uses 8 bit data bus externally but 8086 uses 16 bit data bus.

(10) 8086 has fourteen 16-bit registers.

(11) 20 lines of address bus operate in multiplexed mode. The lower order 16 address lines are multiplexed with data and 4 higher order address bus lines are multiplexed with status signals.

(12) It operates on a single +5V power supply.

(13) The 8086 has a powerful instructions set with a range of addressing modes. It can perform bit, byte word and block operations.

5.3.2 Internal Architecture of 8086

Fig. 5.3.1 depicts the architecture (functional block diagram) of 8086. As shown in Fig. 5.3.1, the 8086 architecture is divided into two independent functional units as,

(1) Bus Interface Unit (BIU).

(2) Execution Unit (EU).

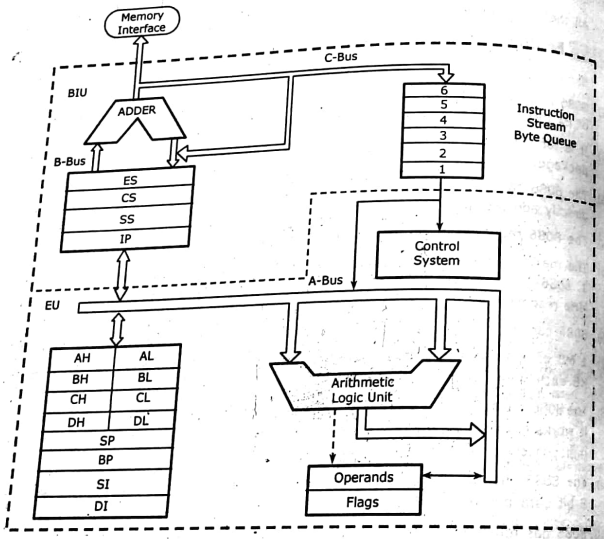


Fig. 5.3.1 Architecture (Functional Diagram) of 8086

5.3.2.1 Bus Interface Unit

The Bus Interface Unit (BIU) is responsible for establishing communications with external devices and peripherals including memory via the system bus. BIU serves the following purposes,

(1) It fetches instruction from memory.

(2) It reads data from I/O port and memory.

(3) It writes data into I/O port and memory.

(4) It supports instruction queuing.

(5) It provides the address relocation facility.

The three functional blocks of a bus interface unit are namely, instruction queue, segment registers and instruction pointer,

(1) **Instruction Queue :** Bus Interface Unit (BIU) prefetches six instruction bytes in advance from the memory. The prefetched instructions are stored in a group of high speed registers known as the instruction queue, this instruction queue is based on first-in first-out (FIFO). The BIU works in parallel with the EU, and BIU fetches the instruction bytes while the EU is executing an instruction. The simultaneous operations of BIU and EU are possible only when the EU does not require the system bus. The process of fetching the next instruction in advance while the EU is executing the current instruction, is known as 'pipelining'. This technique speeds up the execution on programs.

(2) **Segment Registers :** 8086 has the capability of addressing 1 Mega Byte (2^{20} bits ≈ 1 MB) of memory, which is divided into 16 logical segments. Each segment contain 64K bytes of memory. But at any given time 8086 works with only four 64 kB segments.

Each segment is associated with segment registers. They are,

(i) Code Segment (CS) register.

(ii) Data Segment (DS) register.

(iii) Stack Segment (SS) register.

(iv) Extra Segment (ES) register.

These registers are used to store the upper 16-bits of starting address of the four memory segments. BIU generates a 20-bit address using the segment and the offset components of an address.

(3) **Instruction Pointer :** The IP register holds the address of the next instruction which is to be executed next. It contains the offset value (distance in byte from base or starting address) of the next instruction. Instruction pointer in advance points to next instruction in the memory, when the instruction is executing. Instruction pointer is similar to the program counter in 8085 microprocessor.

5.3.2.2 Execution Unit

The Execution Unit (EU) informs the BIU from where the instruction or data to be fetched. Execution unit performs the following functions,

- It picks up the instructions from the instruction queue of BIU.
- It decodes the instructions and then executes the instruction.
- It updates the status of flag register.

(1) **Control Unit :** It is responsible for controlling and co-ordinating all the activities performed by various sub-units. This unit fetches the instructions from instruction queue, decodes the instructions and execute the instructions. Control unit issues various control signals such as read/write data etc., to perform various operations.

(2) **ALU :** It performs all arithmetic and logic operations and result may be stored temporarily in general purpose registers or index registers.

(3) **General Purpose Register :** The registers such as AX, BX, CX and DX are general purpose registers. They are used to store the data temporarily. Using this registers, the data can be accessed very fast.

(4) **Pointer and Index Registers :** These registers are normally used to get the data from the memory or write the data into memory. The pointer registers are BP, SP and IP. The index registers are SI and DI.

(5) **Flag Register :** The flag register is a combination of 16 flip-flops that can be individually set or reset accordingly after executing the instruction which can be used to control the various operations of EU.

COMMENT : *The internal bus used for ALU operations is called A-bus, the internal bus used by BIU to calculate the physical address is called B-bus and the internal bus used to fetch the instruction/data from the external interface or memory to instruction queue is called C-bus.*

5.3.3 Register Organization of 8086

The Intel 8086 microprocessor contains fourteen 16-bit registers as shown in Fig. 5.3.2. They are grouped as,

(1) General purpose registers.

(2) Pointer and index registers.

(3) Segment registers.

(4) Instruction pointer.

(5) Flag register.

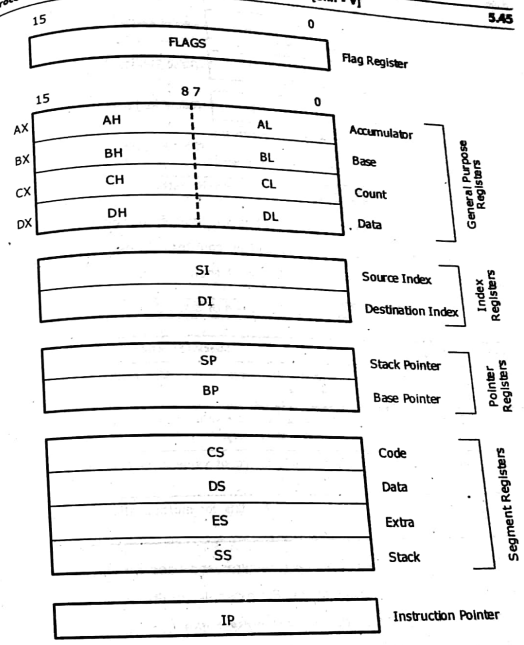


Fig. 5.3.2 8086 CPU Registers

5.3.3.1 General Purpose Registers

Intel 8086 has four 16-bit general purpose registers, which are AX, BX, CX and DX. ch of them can hold 16 bit information. Normally, in many instructions we require 8086 work on 8-bit data. To facilitate this, these registers can be thought of as divided into 8 bit registers (AX: AL, AH, BX: BL, BH, CX: CL, CH, DX: DL, DH) where 'L' indicates ower byte and 'H' indicates the higher byte information.

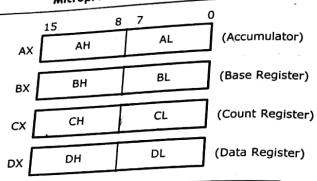


Fig. 5.3.3 General Purpose Registers

The GPRs are used as the source or destination register during the transfer of data and computations, as pointers to memory and as counters. These registers are named after special functions carried out by each one of them as shown in Table 5.3.1.

Table 5.3.1 Special Functions of 8086 GPRs

Register	Name of Register	Special Function
AX	Accumulator	Stores the 16-bit result of certain arithmetic and logic operations.
BX	Base register	Used to hold the base value in base addressing mode to access data in memory.
CX	Count register	Used to hold the count value in SHIFT, ROTATE and LOOP instructions.
DX	Data register	Used to hold the data for multiplication and division operations.

(1) **AX Register :** This register is known as accumulator and used for operations involving I/O and most of the arithmetic operations. For example, multiply, divide and translate instructions use AX register by default. AX register can be used as a single 16-bit register or two 8-bit registers AH and AL. The splitting of the AX register into AH and AL registers is convenient for performing the data-byte operations.

(2) **BX Register :** This register is known as base register. This is the only general purpose register which can also be used as index register. It is used as a pointer to a memory location, for accessing the elements of an array from the memory. It can be used in computations. BX register can be used as a single 16-bit register or two 8-bit registers BH and BL.

(3) **CX Register :** This register is known as counter register. The value stored in this register is used to control the number of times a loop has to be executed or the number which indicates the number of shifts required either left or right. In string instructions like REP, LOOP microprocessor will use register CX as implicit counter. Similarly in rotate/shift instructions, microprocessor will use register CL as 8-bit counter.

(4) **DX Register :** The register DX functions as the data register. The register DX is the only register used as an I/O address pointer in the IN and OUT instructions. The DX register is used as the default operand in 16-bit multiplication and division operations. For example, in 32/16 bits division, DX register holds the 16 MSBs of the dividend and AX register holds the 16 LSBS of the dividend. DX register can be used as a single 16-bit register or two 8-bit registers DH and DL.

5.3.3.2 Pointer and Index Registers

The effective address of any memory location will be 16-bits. Registers which are used for storing 16-bit effective address of memory location are called as memory pointers. These registers are classified as follows,

(1) **Index Registers :** 8086 contains two 16 bit index registers. They are Source Index (SI) and Destination Index (DI) registers. These two registers are used while dealing with string operations.

The contents of SI are added to the contents of DS register to get the actual 20-bit address of data (string data) in the data segment. Similarly, the contents of DI are added to the contents of ES register to get the actual 20-bit address of data (string data) in extra segment. The contents of these registers are incremented or decremented after the corresponding string related instructions are executed.

Both SI and DI can be used to hold 16 bit data temporarily, if it is necessary,

(2) **Pointer Registers :** The registers that contain the offset value of an instruction within a code segment or offset value of data within stack segment are called pointer registers.

(i) **Instruction Pointer (IP) Register :** IP register is a 16-bit register that contains the offset address of the next instruction to be executed in the code segment (It is similar to program counter in 8085 microprocessor). In other words it holds the 16-bit offset address of the next instruction to be executed. This address is added to the starting (base) address of code segment to generate 20-bit physical address of the memory location containing the instruction to be executed next.

Physical address is obtained using the following relation,

$$\text{Physical address} = (CS)^* 10H + (IP)$$

(ii) Base Pointer (BP) Register : BP register is a 16-bit register that contains the offset address of the data or the parameters within the stack segment. This register is always associated with SS register. So, logical address of the data within the stack segment is given by the pair SS:BP and physical address is obtained using the following relation,

$$\text{Physical address} = (SS)^* 10H + (BP)$$

This register can also be combined with SI or DI to get the data from the stack segment. In such situation, the addressing mode that is used is base indexed addressing. The contents of this register are incremented/decremented automatically after the corresponding stack related instructions are executed.

(iii) Stack Pointer (SP) Register : SP register is a 16-bit register that contains the offset address of the data which is present on top of the stack. This register is also associated with SS register. The physical address of the data which is on top of the stack can be obtained using the following relation,

$$\text{Physical address} = SS^* 10H + (SP)$$

5.3.3.3 Segment Registers

The segment registers are used to store the starting address of the respective segments. They store only upper 16-bits or upper 4 nibbles of the starting address of the respective segments since the size of each segment register CS, DS, ES and SS is 16-bit. The Fig. 5.3.4 shows the graphical view of CS, DS, ES and SS registers and their relationship to the code segment, data segment, extra segment and stack segment.

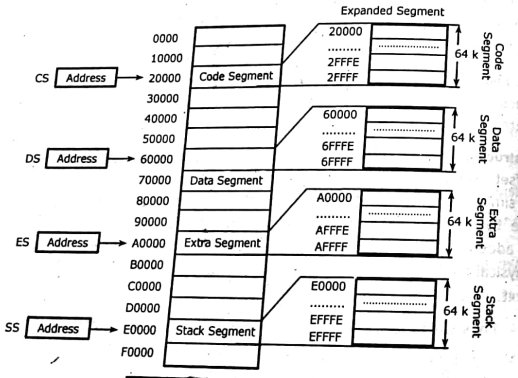


Fig. 5.3 Segment Registers

CS Register : It contains the starting address of the program's code segment in the memory. For normal programming purpose, this register is not directly referenced.

DS Register : It contains the starting address of the program's data segment in the memory. The instructions use the value stored in DS register to locate the data.

ES Register : It contains the starting address of the program's extra data segment in the memory. This segment register is used only during string manipulation. In this context, ES register is always associated with DI register.

SS Register : This register contains the starting address of the programs stack segment in the memory. This register permits the implementation of stack in memory. For normal programming purpose, this register is not directly referenced.

EXAMPLE PROBLEM 5.1

If the code segment for an 8086 program starts at address 70400H, what number will be in the CS register?

SOLUTION

It is given that code segment starts at 70400H which is 20-bit address. But, CS register is 16-bit address. Only upper 16-bits (i.e., upper 4 nibbles) of 20-bit address i.e., 7040H is stored in CS register.

So, CS = 7040 H

5.3.3.4 Flag Registers

Intel 8086's flag register is a 16-bit register which contains 16 flip-flops. Out of 16-bits only 9 bits are meaningful, the other 7-bits do not contain any useful information. These 9 flags are divided into two parts,

(1) Status/conditional flags.

(2) Control flags.

The complete 16-bit configuration of 8086 flag register is shown in Fig. 5.3.5.

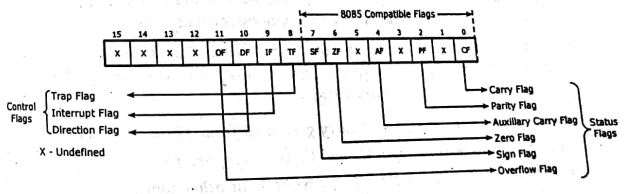


Fig. 5.3.5 Flag Register of 8086

(1) **Conditional/Status Flags :** When microprocessor performs any operation in ALU, then depending upon the status of result obtained in ALU, microprocessor will store corresponding status bit into the flip-flops of these status flags. There are 6 status flags out of 9 flags.

These are explained as follows,

(i) Auxiliary Carry Flag (AF)

- Auxiliary carry flag is set to '1' if, there is a carry from the fourth bit position into the fifth bit position during addition of two 8-bit numbers, or if there is a carry from the eighth bit position into the ninth bit position during addition of two 16-bit numbers.
- Otherwise AF is cleared
- This flag is used only by BCD arithmetic instruction.

(ii) Carry Flag (CF) : Carry flag is set to '1' if the addition of two binary numbers produces a carry out of MSB position. If there is no carry, then CF = 0. This flag is used only when manipulating two unsigned numbers.

(iii) Parity Flag : This flag is set to 1 if the result of an operation contains (only lower byte of the result is considered) even number of 1-bits. Otherwise, the flag is set to zero.

Example : Assume that the result in the 16-bit destination register after an arithmetic/logical operation is,

$$\underbrace{00101101}_{\text{Higher order 8 bits}} \qquad \underbrace{10110110}_{\text{Lower order 8 bits}}$$

Since the number of 1's in the lower order 8 bits is 5 i.e., odd, so parity flag will reset. Therefore PF = 0

(iv) Zero Flag (ZF) : This flag will set to '1' if the result of an arithmetic/logical operation is zero, otherwise it is cleared.

(v) Sign Flag (SF) : This flag is set to '1' if the MSB of the result is one i.e., if the result of any computation is negative, otherwise, it is zero.

(vi) Overflow Flag (OF) : This flag is significant only for signed number. This flag is set when the result of an arithmetic operation generates a signed number that cannot fit into the destination operand.

NOTE : *Suppose CF represent carry generated from MSB and C_1 is the carry generated into MSB. If both CF and C_1 are same, overflow flag will be 0 and if they are different overflow, flag will be 1. In other words,*

$$\text{Overflow} = CF \oplus C_1$$

EXAMPLE PROBLEM 5.2

Find the status of the conditional flags after adding two numbers 76H and 99H.

SOLUTION

$$\begin{array}{r} 76H \\ 99H \\ \hline 10FH \end{array} \Rightarrow \begin{array}{r} 1\ 1\ 1\ 1 \\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 0 \\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1 \\ \hline 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1 \end{array}$$

(i) As carry is generated from MSB, so CF = 1.

(ii) As there is no carry generated from 4th bit position to 5th bit position. So AF = 0.

(iii) As the result of operation is not zero, so ZF = 0.

(iv) As the number of 1's in the lower order 8 bits is 4 i.e., even. So PF = 1.

(v) As the MSB is 0. So SF = 0.

(vi) 0 F ⊕ carry generated = 1 ⊕ 1 = 0

$\therefore$ CF = 1, AF = 0, ZF = 0, PF = 1, SF = 0, OF = 0

(2) **Control Flags :** The 8086 has three control flags in the flag register which are used to control certain operations of the microprocessor. These three flags are different from the 6 conditional flags in the way they are set or reset. The six conditional flags are set or reset by the EU on the basis of the results of some arithmetic or logical operations. The control flags are intentionally set or reset to control specific instructions in the program.

The three control flags are given below,

(i) Direction Flag (DF) : DF is used with string instructions. If DF is zero, the string is processed starting from the lowest address to the highest address, i.e., auto-incrementing mode, otherwise DF = 1 causes string bytes to be accessed from lower address to the higher address.

(ii) Interrupt Flag (IF) : This flag is used to either enable/disable external maskable interrupts,

If IF = 1, the 8086 enables the interrupts.

If IF = 0, the 8086 disables the interrupts.

(iii) Trap Flag (TF) : TF is used for the single step control. If TF is set, the processor enters the single step execution of each instruction. In this mode, the 8086 generates an internal interrupt after execution of each instruction. The processor executes the current instruction and control is transferred to the trap interrupt service routine.

If TF = 0, then microprocessor will execute complete program in one operation and it is called "Free running operation".

5.3.4 Memory Organization of 8086

The Intel 8086 microprocessor has a 20-bit address lines and hence can address 1 MB of memory. But since it has 16-bit registers, so it is impossible to directly represent a 20-bit address. This problem of representing 20-bit address using only 16-bit registers is possible using a scheme known as segmented memory.

5.3.4.1 Memory Segmentation

The complete 1MB of memory is divided into 16 blocks where size of each block is 64 KB (64K = 64 × 1024 = 65536 bytes). Each block is called a segment. This concept of dividing the memory into segments or blocks using which we can access data in 1 MB of memory is called segmented memory. Each segment can contain code (program) or data. The concept of segmented memory is shown in Fig. 5.3.6.

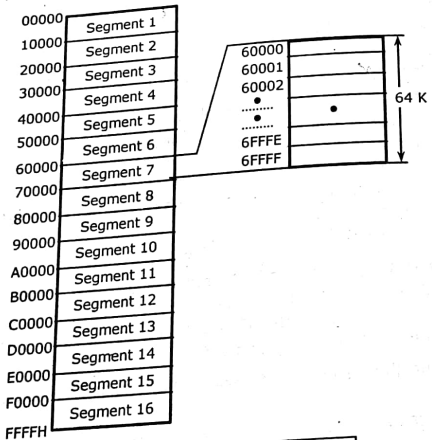


Fig. 5.3.6 Memory Segmentation

From Fig. 5.3.6, it can be noticed that, segment can start at any address, but should be divisible by 16 (i.e., 10 H). Out of these 16 segments, 8086 defined four 64-K segments. The four segments that are supported by 8086/88 are,

(i) **Code Memory Segment(CMS) :** It is used to store only instruction codes of the program. For defining base address of code memory segment, we have to transfer the corresponding base address into code segment register.

(ii) **Data Memory Segment (DMS) :** It is used to store data as well as result. For defining base address of data memory segment, we have to transfer 16 MSB's of the corresponding base address into data segment register.

(iii) **Extra Memory Segment (EMS) :** It is also used to store data as well as result. For defining base address of extra memory segment, we have to transfer 16 MSB's of the corresponding base address into extra segment register.

(iv) **Stack Memory Segment (SMS) :** It is used for storing stack of useful data using PUSH and POP instruction. For defining base address of stack memory segment, we have to transfer 16 MSB's of the corresponding base address into stack segment register.

How four segments might be positioned in memory at any given time?

The segments might be positioned in memory at any given time using following ways.

(1) Segment can be contiguous i.e., data segment may start at 40000H, extra segment at 50000H, stack segment at 60000H and code segment at 70000H. Thus, all the segments are positioned in the memory in sequence.

(2) Segments can be non-contiguous i.e., data segment may start at 6C500H, extra segment at E0000H, stack segment at B5000H and code segment at 30000H as shown in Fig. 5.3.7. Thus, all the segments mentioned are not contiguous in memory.

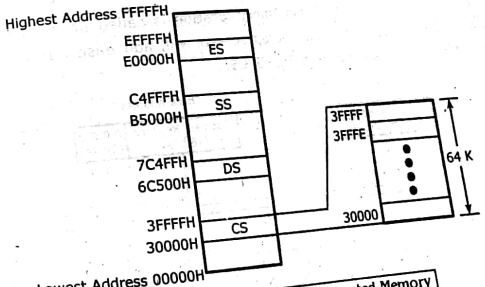


Fig. 5.3.7 Non-Contiguous Segmented Memory

(3) Segments can be overlapped (for small programs which do not require all 64K bytes in each segment). For instance, code and data segment may start at 30000H.

(4) Segments can be duplicated. One code segment may be at 30000H and another code segment may be at 80000H.

5.3.4.2 Generation of Physical Address

Before studying the way to generate the physical address of data (operands) in memory, let us see "What is an offset value or offset address?"

Offset Value (or) Offset Address : The distance of the data in data segment from the base address (i.e., starting address of the data segment) or the distance of an instruction in code segment from the starting address of the code segment represent the offset value or offset address. For example, consider the data segment whose starting address is 60000H.

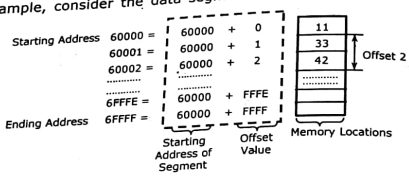


Fig. 5.3.8

The data 42 is at a distance of 2 bytes from the beginning of the segment which starts at 60000H. So, we say that offset value of 42 is 2, offset value of 33 is 1 and offset value of 11 is 0. The range of offset is 0000H to FFFFH.

The 20-bit actual address of a memory location is called physical address. For generating 20-bit physical address, microprocessor will add base address and offset (effective) address (EA), as shown in Fig. 5.3.9.

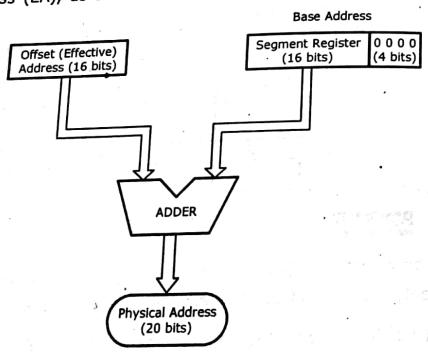


Fig. 5.3.9 Generation of Physical Address

The procedure to obtain physical address is as follows,

Step 1 : Selected segment register contents are shifted left four bits, which gives base address.

Step 2 : Add base address to effective address to get the physical address.

To illustrate this procedure, let us consider the IP content as 123CH. This is called the offset value or effective address and let the contents of segment register CS be 2345H. This means that the code segment starts at 23450H and we want to fetch an instruction from a memory location which is 123CH bytes away from the beginning of the code segment. Thus, the BIU adds up 23450H and 123CH using the adder in the BIU and sends out the sum 2468CH as the Physical Address (PA) on the 20 address pins.

Base Value in CS : 2345 H

```
20-bit Starting Address =   23450
EA (or) Offset Value    =    123C
                        (+)
Physical Address        =  2468 CH
```

COMMENT : *While calculating the physical address note the following,*

(1) *Instruction Pointer (IP) is always associated with CS register.*

(2) *SP and BP are always associated with SS register.*

(3) *SI/DI/BX are always associated with DS register.*

(4) *The segment registers SS, CS, DS and DS holds the upper 16 bits of the starting address of each of the segments respectively.*

EXAMPLE PROBLEM 5.9

What will be the physical address of the datum, given EA of a datum as 2359H and DS as 490B0H?

SOLUTION

```
20-bit BA (490B0H) =   0100 1001 0000 1011 0000
Offset Value (or) EA (2359H) =   0000 0010 0011 0101 1001
                              (+)
Physical Address =   0100 1011 0100 0000 1001
                       4    B    4    0    9
```

∴ PA = 4B409H

EXAMPLE PROBLEM 5.4

The physical branch address is 5A230H when CS = 5200H. What will be the physical address if CS is changed to 7800H?

SOLUTION

Since, physical address = Offset value + Base address

∴ Offset value = PA − BA

Physical Address (5A230H) =	0101	1010	0010	0011	0000
Base Address (52000H) =	0101	0010	0000	0000	0000
(−)					
Offset Value =	0000	1000	0010	0011	0000
	0	8	2	3	0

∴ Offset value = $(8230)_H$

But, given to find PA when CS changed to 7800H

20-bit CS Address (78000H) =	0111	1000	0000	0000	0000
Offset Value (8230H) =	0000	1000	0010	0011	0000
(+)					
Physical Address	1000	0000	0010	0011	0000
	8	0	2	3	0

∴ PA = 80230H

5.3.4.3 Advantages of Segmented Memory

The main advantages of memory segmentation are,

(1) It allows the memory capacity to be 1 MB eventhough the actual addresses to be handled by instructions are 16-bits.

(2) It allows use of separate memory areas for program, data and stack, thus the protection of these is possible.

(3) For large programs, it can use multiple segments for program code, data and stack.

(4) Program relocation can be done very easily. Program relocation means having the ability to run the same program in different memory areas without changing the addresses in the program itself.

5.4 ADDRESSING MODES OF 8086

The 8086 addressing modes can be classified into six main categories as shown in fig. 5.4.1.

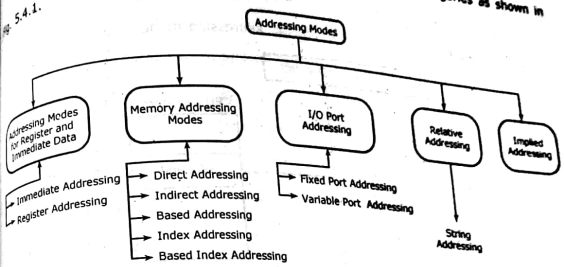


Fig. 5.4.1 Classification of Addressing Modes

The detailed explanation of these addressing modes are studied in the following sub-sections.

5.4.1 Register and Immediate Addressing Modes

These addressing modes have been provided for instructions, where operand can be accessed directly.

5.4.1.1 Immediate Addressing

In immediate addressing mode, the operand (data) required for executing the instruction is given directly along with instruction. The data may be 8 or 16 bits size. The immediate operands can only be the source operands.

Instruction Format

Opcode	Immediate

Examples

MOV AL, 34 H ; The instruction moves the 8-bit immediate data 34 H to AL register.

MOV CX, 3500 H ; The instruction moves the 16-bit immediate data 3500H to CX register.

5.4.1.2 Register Addressing

In this addressing mode, 8/16 bit data (operand) required for executing the instruction is present in CPU registers and the name of that register is given along with the instruction, such addressing mode is called register addressing mode.

Instruction Format : Instruction format of register addressing mode is shown in Fig. 5.4.2.

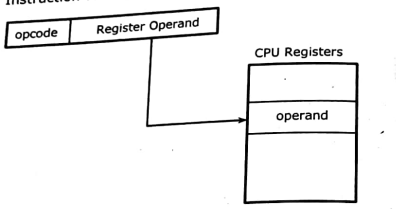


Fig. 5.4.2 Instruction Format of Register Addressing Mode

Examples

(1) MOV CX, SI : Here, both the source and destination operands are specified in registers. It moves the contents of SI to CX register.

	Before	After
(CX)	124 C	23A2
(SI)	23A2	23A2

(2) MOV LOC, BL : The source is a 8 bit data and the destination is a memory lo n. It moves the contents of BL register to the symbolic memory location LOC.

	Before	After
(DS: LOC)	D2	34
(BL)	34	34

(3) MOV DX, AX : It moves the contents of AX to DS register.

	Before	After
(DS)	1200	F2A2
(AX)	F2A2	F2A2

COMMENT

(1) *If both operands are present, the size of both operand registers should be same for example.*

MOV CL, AX

Invalid instruction, since operand size doesn't match.

(2) *Segment registers can also be used as operands with the exception that CS register cannot be used as destination operand. For example.*

MOV DS, AX → Valid

MOV CS, AX → Invalid, CS register cant be the destination operand.

MOV AX, ES → Valid

5.4.2 Memory Addressing Modes

In memory addressing modes, location of an operand is placed in a memory. To access the operand (data) in memory, 8086 must generate a 20 bit physical address or memory address. The memory address of an operand is obtained by,

Memory address = Starting address of memory segment + Offset

(1) The 16 MSBs of the starting address of memory segment resides in the corresponding memory segment. The BIU then generates 20-bit starting address of the memory segment by appending with four zeros as the least significant hex digit.

(2) How far the operand's memory location is within a memory segment from the starting address of the segment is called offset or effective address. An offset value is determined by adding any combination of three address elements,

Displacement : It is an 8-bit or 16-bit immediate value given in the instruction.

Base : It is the content of the base register, BX or BP.

Index : It is the content of the index register, SI or DI.

Offset or EA = {Base register} + {Index register} + {8/16 bit displacement}

The combination of these three address elements give six memory addressing modes plained in the following subsections.

5.4.2.1 Direct Addressing Mode

In direct addressing mode 16 bit offset value or effective address of operand in memory location is specified directly in the instruction itself.

Example : MOV AX, [4000 H]

This instruction moves the contents of memory location to AX register whose offset value is 4000H. The 20 bit memory address of operand is calculated using 4000H as offset and content of DS as starting address.

Generation of 20 Bit Memory Address of Operand

Let,

DS = 0000H, and given EA = 4000H

Base Address = 0 0 0 0 0 H

Offset/EA = (+) 4 0 0 0 H

Memory/Physical Address = 0 4 0 0 0 H

Data Flow Diagram

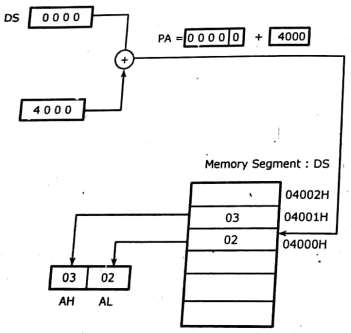


Fig. 5.4.3 Data Flow Diagram for Instruction MOV AX, (4000H)

COMMENT : *The number of bytes to be copied is determined by the size of the destination.*

Example : MOV Al, [4000H]

Since AL is a 8-bit register, the data stored in memory location determined using offset value 4000H and contents of DS register is copied to AL.

5.4.2.2 Indirect Addressing Mode

In this addressing mode, the offset address of the data to be accessed is present in the registers, specified in the instruction. The registers used to hold the offset address are BX, SI and DI. That is,

$$EA = \begin{Bmatrix} BX \\ SI \\ DI \end{Bmatrix}$$

The 20-bit memory address of operand data is obtained by adding offset to the starting address. i.e.,

$$\text{Memory Address} = \{ds\} : \begin{Bmatrix} BX \\ SI \\ DI \end{Bmatrix}$$

Instruction Format : The instruction format for indirect address is shown in Fig. 5.4.4.

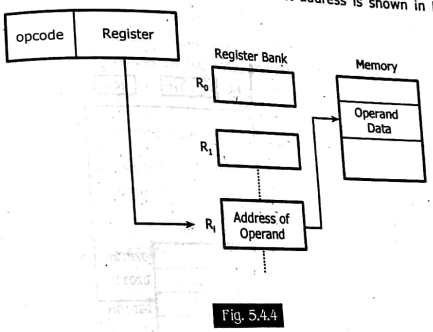


Fig. 5.4.4

If SI, DI, BX are registers used to hold offset value, then default segment register is DS.

As seen in Fig. 5.4.4 it needs one memory access and one register access to get the operand data.

EXAMPLE PROBLEM 5.5

What happens if the given instruction MOV AX, [BX] is executed?

SOLUTION

The content of BX gives the offset address. The content of memory obtained by adding starting address and offset is moved to AL register and the content of next memory location is moved to AH register. This is made clear by following explanation.

Generation of Memory Address of Operand : The default segment register used is DS if the register is BX.

Let, DS = 0010H and EA : [BX] = 2000H

MOV AX, [BX]

Let,

Base Address BA = 0001 0

(Effective address) EA = 2000

(+)

(Physical Address) PA = 02010 H

Data Flow Diagram

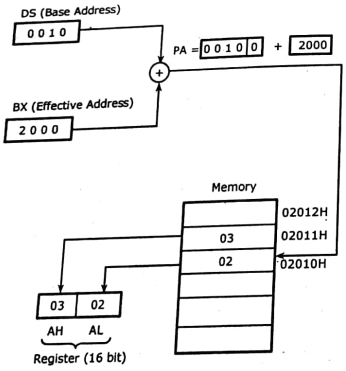


Fig. 5.4.5

COMMENT : *The register BP should not be used in place of BX in the above ins*
The register BP should be associated with 8-bit or 16-bit displacement.

Based Addressing Mode

In this addressing mode, the part of the offset address of the data is stored in the register BX or BP. The remaining part of the offset address is specified as displacement. So, actual offset address also called effective offset address is obtained by adding the contents of BX or BP register with the displacement. The displacement can be 8-bit or 16-bit signed number i.e., the displacement can be +ve or -ve.

(1) When BX is used to hold the EA, the default segment register used for calculating PA is DS.

(2) When BP is used to hold the EA, the default segment register used is SS.

(3) $$EA = \begin{Bmatrix} BX \\ (or) \\ BP \end{Bmatrix} + \{\text{Displacement (8 bit or 16 bit)}\}$$

$$PA \text{ (or) } MA = \{DS * 10H\} + \{BX\} + \{\text{Displacement}\}$$

$$\{SS * 10H\} + \{BP\} + \{\text{Displacement}\}$$

EXAMPLE PROBLEM 5.6

What happens if the instruction MOV AX, [BX + 23H] is executed?

SOLUTION

This instruction copies a word from data segment at the offset address : (BX) + 23 H.

Generation of Physical Address of Operand : Let EA : BX = 0050H. The effective address is calculated by sign-extending the 8-bit displacement 23 H to 16-bit and adding to the content of BX. Thus,

Effective address = 0050H + 0023H

= 0073H.

Let, DS : 1000H

EA = 0073H

BA : DS = 10000

(+)

PA = 10730 H

Thus, the content of memory location 10730H is moved into AL register and ent of next memory location 10731H is moved into AH register.

Representation of Signed Number : The displacement may be +ve or –ve number.

In 2's complement notation, a positive number starts with a 0 in the MS bit position, and the other bits provide its, magnitude. Therefore, +2 is represented using 8 bits as 0 000 0010.

In 2's complement notation, a negative number is represented as the 2's complement of the same magnitude positive number. As +2 is represented as 0 000 0010 using 8 bits, –2 is represented as 1 111 1110 or FEH.

Sign Extension : Depending on the magnitude of the displacement, the assembler codes the displacement using one or two bytes. If the displacement is a small value in the range –128 to 127, it is coded as a one byte value.

When the magnitude of the displacement is larger than the above mentioned range, the assembler codes it as a two byte value.

EXAMPLE PROBLEM 5.7

What happens if the instruction –2[BX], AC24H is executed?

SOLUTION

The assembler codes this displacement -2 as FEH (since 2's complement representation of -2 using 8-bits is 11111110_2 = FEH). This FEH sign extended as FFFEH forms part of the EA. Let, BX contents be 4200H.

Effective/Offset Address Calculation

```
BX :          4200 H
Displacement :  FFFE H
              (1) 41FE H   = Effective Address
               ↑
          Discard carry
```

Physical Address Calculation : Let, DS: 1000H.

```
Starting Address =  1000 0
EA =                  41FE
                (+)
MA/PA =             141F EH
```

Therefore lower significant hex digits 24H is moved into memory location 141F[illegible] and higher significant HEX digits ACH is moved into next memory location 141[illegible]

5.4.4 Indexed Addressing with Displacement

In indexed addressing with displacement, a part of the EA is specified as the contents of an Index register (SI or DI), and the other part is provided in the instruction itself. This part provided in the instruction is called as the Displacement. (For explanation of the displacement part, see Based addressing with displacement).

The sum of the contents of part of the EA specified in an index register, and the 16 bit effective displacement provides the complete 16 bit EA,

$$EA = \begin{Bmatrix} SI \\ (or) \\ DI \end{Bmatrix} + \begin{Bmatrix} \text{8-bit displacement or} \\ \text{16-bit displacement} \end{Bmatrix}$$

In case of 8-bit displacement it is sign extended to 16-bit before adding to index value. Always DS register is used to calculate the physical address while accessing the data using the registers SI and DI,

$$PA = (DS)*10H + \begin{Bmatrix} SI \\ DI \end{Bmatrix} + \begin{Bmatrix} \text{8-bit displacement or} \\ \text{16-bit displacement} \end{Bmatrix}$$

EXAMPLE PROBLEM 5.8

Consider the instruction MOV AX, 23H[SI]. What will happen if this instruction is executed?

SOLUTION

This instruction copies a byte from data segment memory location at offset value (SI) + 23H from starting address into AL register and next memory location into AH register.

Effective/Offset Address Calculation : Let SI : 0032H and given displacement as 23H and sign extending the displacement using 16 bits, we get 0023H. So effective offset value is,

```
SI : 0032H
Displacement : 0023H
EA = 0055H
```

Physical Address Calculation : Let, DS : 2550 H

```
SA/BA =  2005 0
EA =       0055
     (+)
PA =   200A 5H
```

Thus, contents of memory location 20045H are moved to AL register and contents of next memory location 200A6H are moved to AH register.

5.4.2.5 Based Index Addressing

In this addressing mode the effective address is given by sum of base value, index value and an 8-bit or 16-bit displacement specified in the instruction. The base value is stored in BX or BP register. The index value is stored in SI or DI register. In case of 8-bit displacement, it is sign extended to 16-bit before adding to base value. This type of addressing will be useful in addressing two dimensional arrays where we require two modifiers.

$$EA = \begin{Bmatrix} BX \\ (or) \\ BP \end{Bmatrix} + \begin{Bmatrix} SI \\ (or) \\ DI \end{Bmatrix} + \begin{Bmatrix} 8\text{-bit displacement (or)} \\ 16\text{-bit displacement} \end{Bmatrix}$$

If displacement is of 8 bit. Then it is converted into 16 bit by extending the sign bit. Example of this addressing mode are,

(1) MOV A H, 1970 H [BX] [DI].

(2) MOV DX, [BP + SI + 9F H] or MOV DX, 9F H [BP] [SI].

EXAMPLE PROBLEM 5.9

Explain the instruction.

MOV DX, [BP + SI +9F H] or MOV DX, 9F H [BP] [SI].

SOLUTION

Let, BP = 1823, SI = 2910 and SS contains 8100 and sign extension of 9F = FF9F H

	1823H	(Contents of BP)
	2910H	(Contents of SI)
	+ FF9FH	(16-bit displacement)
Carry generated →	(1) 40D2H	

Neglect carry if generated.

Now let us calculate the physical address,

B.A = 81000H
E.A = 40D2H
P.A = 850D2H

5.4.3 I/O Port Addressing

The 8086 microprocessor is connected to I/O devices and memory for communication. The microprocessor directly interact with memory but not with I/O devices because, the I/O devices are electromechanical devices and are very slow when compared to the speed of the microprocessor. So, microprocessor communicates with I/O devices through I/O ports. I/O ports are semiconductor chips working at the speed of the microprocessor.

The two different I/O port addressing schemes are,

(1) Fixed port addressing.

(2) Variable port addressing

5.4.3.1 Fixed I/O Port Addressing

In this mode of addressing, the port address of the I/O device is specified as the part of the instruction. So, during execution, the port address of the I/O device can not be changed.

Examples

(1) IN AL 70H : A byte of data from the specified port is copied into AL register.

(2) IN AX, 70H : 16-bit data is copied into AX register from port 70H.

(3) OUT 70H, AL : A byte of data from AL register is moved to the output port whose address is 70 H.

(4) OUT 70H, AX : 16-bit data from AX register is moved to the output port whose address is 70H.

5.4.3.2 Variable I/O Port Addressing

In contrast to the fixed I/O port addressing, in variable I/O port addressing the I/O port address is not specified explicitly as part of the instruction. The address of the port is stored in DX register before addressing the I/O port. During execution, as we change the contents of DX register, we can address different I/O ports. This method of addressing an I/O port is called variable port addressing.

Examples

(1) IN AL DX : An 8-bit data from the port whose address is available in DX is moved to AL register.

(2) IN AX, DX : An 16-bit data from the port whose address is available in DX register is moved to AX register.

(3) OUT DX, AL : 8-bit data from AL register is moved to the output port whose address is specified in DX.

(4) OUT DX, AX : 16-bit data from AX register is moved to output port whose address is specified in DX register.

5.4.4 Relative Addressing Mode

In this addressing mode, the effective address of a program instruction is specified relative to Instruction Pointer (IP) by an 8-bit signed displacement.

Example

JZ 0AH

Sign extend

$000\ A_H \leftarrow 0\ A_H$

If ZF = 1, then (IP) ← (IP) + $000A_H$,

$EA_C = (IP) + 000A_H$

$BA_C = (CS) \times 16_{10}$

$MA_C = BA_C + EA_C$

Note : *Suffix C refer to code memory*

If ZF = 1, then the program control jumps to new code address as calculated above.
If ZF = 0, then next instruction of the program is executed.

5.4.4.1 String Addressing Mode

This addressing mode is employed in string instructions to operate on string data. In string addressing mode the Effective Address (EA) of source data is stored in SI register and the EA of destination data is stored in DI register.

The segment register used for calculating base address for source data is DS and can be overridden. The segment register used for calculating base address for destination is A_S and cannot be overridden.

This addressing mode also supports auto increment/decrement of index registers SI and DI depending on Direction Flag (DF). If DF = 1, then the content of index registers are decremented to point to next byte/word of the string after execution of a string instruction. If DF = 0, then the content of index registers are incremented point to previous byte/word of the string after execution of a string instruction. (For word operand the content of index registers are incremented/decremented by two and for byte operand the content of index registers are incremented/decremented by one).

EXAMPLE PROBLEM 5.10

What happens if the instruction MOVS WORD is executed?

SOLUTION

This instruction move a word of string data from one memory location to another memory location. The address of source memory location is calculated by multiplying the content of DS by 16_{10} and adding to SI. The address of destination memo location is calculated by multiplying the content of ES by 16_{10} and adding to

After the move operation, if DF = 1, then the content of index registers DI and SI are decremented by two. If DF = 0, then the content of index registers DI and SI are incremented by two,

$EA_S = (SI)$

$BA_S = (DS) \times 16_{10}$

$MA_S = BA_S + EA_S$

$EA_D = (DI)$

$BA_D = (ES) \times 16_{10}$

$MA_D = BA_D + EA_D$

$(MA_D) \leftarrow (MA_S)$

If DF = 1, then (SI) − 2 ← (SI) and (DI) ← (DI) − 2

If DF = 0, then (SI) + 2 ← (SI) and (DI) ← (DI) + 2

5.4.5 Implied Addressing Mode

In implied addressing mode, the instruction does not contain the operands. Even though explicitly the operands are not specified, the microprocessor implicitly knows in which register data is available and in which register the computed result should be stored.

Example : Decimal adjust accumulator.

Note : That there are no operands in the instruction. Even though the operands are not specified, the microprocessor knows that AH or AL register should be used during execution and the result is stored in AL or AH register.

5.5 INSTRUCTION SET OF 8086

The 8086 instruction set consists of the instructions equivalent to those typically found in the 8085 plus many new instructions. Some new instructions like multiplication and division, string operations, bit manipulation instructions, and interrupt instructions are available in the Intel 8086. The instruction set in 8086 contains no-operand, single-operand, and two-operand instructions.

Except string instructions, 8086 instructions do not permit memory to memory operations. The 8086 instruction set is categorized as following eight groups,

(1) **Data Copy/Transfer Instructions :** These instructions are used to transfer data from source operand to destination operand. This group includes all the store, move, load, exchange, input and output instructions.

(2) **Arithmetic and Logical Instructions :** These instructions perform arithmetic, logical, increment, decrement, compare and scan operations.

(3) **Branch Instructions :** These instructions transfer the control of program execution to the specified address location. Branch includes all the call, jump, interrupt and return instructions.

(4) **Loop Instructions :** These instructions can be used to implement unconditional and conditional loops if these instructions have REP prefix with CX register used as count register. These instructions include LOOP, LOOPNZ and LOPZ instructions. With the use of these instructions, the different loop structures can be implemented.

(5) **Machine Control Instructions :** These instructions are used to control the machine status. This group includes NOP, HLT, WAIT and LOCK instructions.

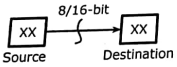
(6) **Flag Manipulation Instructions :** These instructions directly affect the flag register of 8086. This group includes instructions like CLD, STD, CLI, STI, etc.

(7) **Shift and Rotate Instructions :** These instructions perform the bit-wise shifting or rotation in either direction with or without a count in CX.

(8) **String Instructions :** These instructions perform various string manipulation operations like load, move, scan, compare, store, etc. and are only to be operated upon the strings.

5.5.1 Data Transfer Instructions

Data transfer instructions are used to transfer 8/16 bit-data from source to destination. Notice that the size or capacity of source and destination should be same,



(1) Data transfer instructions doesn't affect the flags.

(2) Data transfer instructions are again classified into four groups, as shown in Fig. 5.5.1.

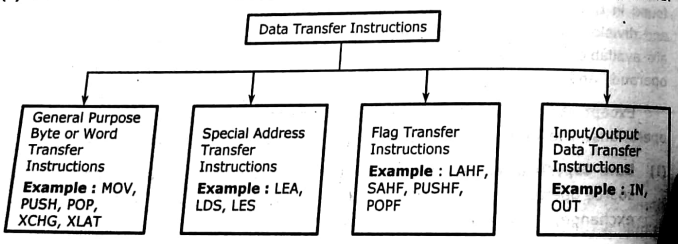


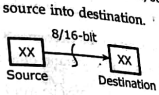
Fig. 5.5.1 Classification of Data Transfer Instructions

5.5.1 General Purpose Data Transfer Instructions

MOV : MOVE

SYNTAX : MOV Destination, Source

DESCRIPTION : The instruction "MOV destination, source" copies or transfer 8/16 bit data from source into destination.



The possible combination of source and destination is shown in the Table 5.5.1.

Table 5.5.1 Possible Combination of Source and Destination in MOV Instruction

Source	Destination
Reg	Reg
Reg	Mem
Mem	Reg
Immediate data	Reg/Mem
Seg.Reg	Reg/Mem

FLAGS AFFECTED : None

EXAMPLES

```
MOV BX, 592FH      ; Loads the immediate number 592FH in BX
MOV CL, [357AH]    ; Copy the contents of memory location, at a displacement of
                     357AH from data segment base, into the CL register.
MOV [734AH], BX    ; Copy the contents of BX register to two memory locations in the data
                   ; segment. Copy the contents of BL register to memory location at
                   ; a displacement of 734AH and BH register to memory location at a
                   ; displacement of 734BH.
MOV DS, CX         ; Copy word from CX register to data segment register.
```

NOTE : *Both source and destination operands cannot be memory locations for MOV instruction.*

PUSH : PUSH REGISTER OR MEMORY ONTO THE STACK

SYNTAX : PUSH Source

DESCRIPTION : The "Push Source" instruction moves the contents of the specified register/memory location onto the stack. After each PUSH operation, the stack pointer is decremented by 2. The current stack-top is always occupied by the previously pushed data.

So, the push operation decrements SP by two and then stores the two byte contents of the operand onto the stack,

(i) Here, the source must be word (16-bit). The source of the word can be a general purpose register, a segment register or memory location.

(ii) Always notice that after PUSH instruction, the higher byte data of source operand is pushed first onto the stack and then the lower byte data.

EXAMPLES

```
PUSH AX          ; Copies content of AX onto the top of stack and decrement SP by 2
PUSH DS          ; Copies content of DS onto the top of stack and decrements SP by 2
PUSH [6000H]     ; Copies a word from memory in DS onto the stack at a offset of 600H
```

POP : POP REGISTER OR MEMORY

SYNTAX : Pop Destination

DESCRIPTION : The POP instruction (POP destination) copies a word from the stack location pointed by the stack pointer to a destination. After the word is copied to the specified destination, the stack pointer is automatically incremented by 2. In the POP instruction, the destination can be a general purpose register, a segment register or a memory location.

EXAMPLES

```
POP AC           ; Copy a word from top of stack to C and increment SP by 2
POP DS           ; Copy a word from top of stack to DS and increment SP by 2.
POP NEXT[BX]     ; Copy a word from top of stack to memory in DS (i.e., PA = EA + DS)
                 ; with EA = NExT + [BX]
```

NOTE : POP CS is an illegal instruction.

XCHG : EXCHANGE

SYNTAX : XCHG destination, source.

DESCRIPTION : The instruction "XCHG destination, source" exchanges the contents of source and destination. The size of destination and source must be equal.

The possible combinations of source and destination is given in the Table 5.5.2.

Table 5.5.2

Source	Destination
Reg	Reg
Reg	Mem

This instruction cannot exchange the contents of two memory locations.

FLAGS AFFECTED : None

EXAMPLES

```
XCHG AL, BL      ; This instruction exchanges byte in BL with byte in AL.
```

Before Instruction		After Instruction	
AL	BL	AL	BL
19	15	19	15

XCHG CX, [7800H] : Exchanges word in CX with word in memory at EA = 7800H and PA = EA + DS.

Let DS : 2500H

Physical Address Calculation

```
Base Address =  25000 H
EA =          (+) 7800 H
PA =            2C800 H
```

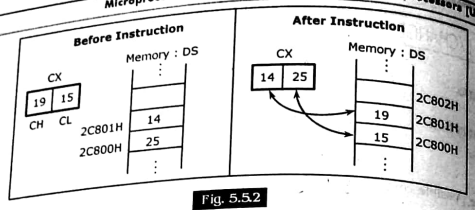


Fig. 5.5.2

Note that higher byte data occupies the higher address and the lower byte data occupies the lower address.

COMMENT

(1) XCHG CX, 7800H is not permitted, because immediate data is not allowed to exchange its contents with register or memory.

(2) Data cannot be exchange directly with segment registers. For instance, XCHG DS, AX is an illegal instruction.

XLAT / XLATB : TRANSLATE A BYTE

SYNTAX : XLA/XLATB

DESCRIPTION : The instruction XLAT/XLATB replaces a byte in the AL register with a byte from the lookup table. Prior to execute this instruction, BX is initialized to contain the start address of the look up table. The instruction gets a byte from a location pointed to by [BX + AL] and is transferred to AL register i.e.,

(AL) ← DS : [(BX) + (AL)]

It is an instruction with no operands. It is an instruction with implied mode of addressing. This instruction is useful to convert from one code to another (translation) using lookup table.

FLAGS AFFECTED : None

EXAMPLE : The contents of the registers and memory before and after executing the XLAT instruction is shown in Fig. 5.5.3.

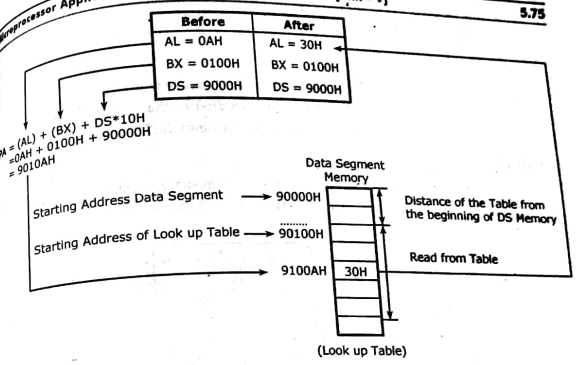


Fig. 5.5.3 Illustration of XLAT Instruction

5.5.1.2 Special Address Transfer Instructions

LEA : LOAD EFFECTIVE ADDRESS

SYNTAX : LEA Reg, Effective Address of Source

DESCRIPTION : The "Load effective address" instruction copies the effective address of memory location named as source into the given register. The register should be of 16 bits size, since the EA of any memory location is of 16 bits.

FLAGS AFFECTED : None

EXAMPLES

```
LEA CX, [SI]          ; Copies the number (address) in the SI register to CX register.
LEA BX, 56H [SI]      ; Loads the BX register with an EA=56H+SI.
LEA AX, 1800H [BX+SI] ; Loads the AX register with EA=1800H+[BX]+[SI]
```

LDS : LOAD DATA SEGMENT

SYNTAX : LDS Register, memory

DESCRIPTION : This instruction copies the contents of one word from first two memory locations into the register specified in the instruction and the next consecutive two memory locations into the DS register.

EXAMPLES

```
LDS CX, [1234] ; This instruction copies contents of memory at displacement of 1234H,
               ; 1235H to CX. Then copy contents at displacement of 1236H and 1237H
               ; in DS.
```

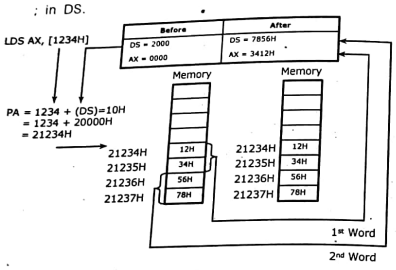
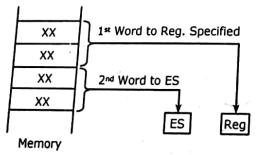


Fig. 5.5.4

LES : LOAD EXTRA SEGMENT

SYNTAX : *LES Register, Memory*

DESCRIPTION : The instruction "LES register, memory" copies content of first two memory locations into one of the 16-bit general, index or pointer registers. Then it loads the next consecutive two memory locations into extra memory segment. The register should be of 16 bits.



EXAMPLE

```
LES CX, [3483H]   ; This instruction copies contents of memory at displacement of
                  ; 3483H in DS to CL, contents of 3484H in DS to CH and copy
                  ; the contents of memory at displacement of 3485H and 3486H
                  ; in DS to ES register.

LES DI, [43H + BX] ; This instruction copies the content of memory location with
                   ; EA = BX + 43, BX + 43 + 1 to DI and with EA = BX + 43 +
                   ; 2, BX + 43 + 3 to ES.
```

Total Effective Address		Physical Address	
BX =	9000H	BA =	30000 H
Displacement =	(+) 0043H	EA =	(+) 9043 H
EA =	9043H	PA =	39043 H

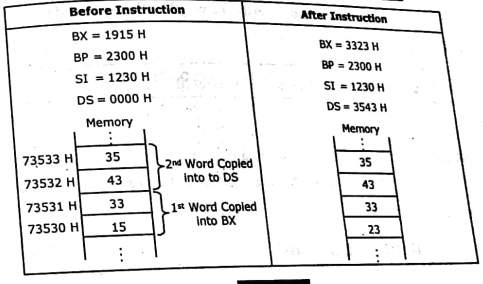


Fig. 5.5.5

5.5.1.3 Flag Transfer Instructions

LAHF : LOAD AH WITH FLAGS

SYNTAX : LAHF

DESCRIPTION : This instruction copies the contents of lower byte of 8086 flag register to AH register. This instruction is used (primarily) to translate 8085 software into 8086/8088 software.

FLAGS AFFECTED : None

SAHF : STORE AH TO LOWER BYTE OF FLAGS REGISTER

SYNTAX : SAHF

DESCRIPTION : This instruction copies the contents of the AH register into the lower byte of the 8086 flag register.

FLAGS AFFECTED : The value of 5 flags of 8 LSBs, of flag register is changed i.e., CF, AF, SF, ZF, PF will change. Rest of the flags OF, DF, IF and TF will remain unchanged.

PUSHF : PUSH FLAG REGISTER ONTO TOP OF STACK

SYNTAX : PUSHF

DESCRIPTION : This instruction decrements the stack pointer by 2 and copies the word in the flag register to the memory locations pointed by the stack pointer.

FLAGS AFFECTED : None

POPF : POP WORD FROM TOP OF STACK TO FLAG REGISTER

SYNTAX : POPF

DESCRIPTION : This instruction copies a word from the two memory locations at the top of stack to the flag register and increments stack pointer by 2.

FLAGS AFFECTED : None

5.5.1.4 Input-Output Instructions

8086 will not directly communicate with input/output (I/O) devices, as the I/O devices are electromechanical and are very slow compared to the speed of operation of the microprocessor. I/O port is like a register, which will be outside the microprocessor, utilized for communication with the fast microprocessor and a provision also for communication with the slow I/O device.

The IN and OUT instructions transfer data between accumulator and input/output ports. IN instructions inputs 8-bit data from 8-bit port into AL or 16-bit data from 16-bit port into AX. Similarly OUT instruction outputs 8-bit data from AL into 8-bit port or 16-bit data from AX into 16-bit port. The port may have 8-bit or 16-bit address. The 8-bit address is referred to as fixed address and is directly specified in the instruction. The 16-bit address is referred to as variable address and is loaded into DX and used in the instruction. The instructions used for this purpose are IN and OUT instructions.

IN : INPUT A BYTE/WORD FROM PORT

SYNTAX : Inputs a byte or word from port.

DESCRIPTION : The IN instruction will copy data from a port to the accumulator. If an 8-bit port is read, the data will go to AL and if an 16-bit port is read the data will go to AX.

The IN instruction can be executed in two different addressing modes, as direct and indirect.

(i) **Direct** : In direct addressing mode, 8-bit address of the port is a part of the instruction.

Examples

```
IN AL, 0F8H      ; Copy a byte from port 0F8H to AL
IN AX, 95H       ; Copy a word from port 95H to AX
```

(ii) **Indirect** : In indirect addressing, the address of the port is referred from DX register. Since DX is a 16-bit register, the port address can be any number between 0001H to FFFFH. Therefore, it is possible to address upto 65,536 ports in this mode.

Examples

```
MOV DX, 30F8H    ; Load 16-bit address of the port in DX
IN AL, DX        ; Copy a byte from 8-bit port 30F8H to AL
IN AX, DX        ; Copy a word from 16-bit port 30F8H to AX
```

FLAGS AFFECTED : None

OUT : OUTPUT A BYTE/WORD TO PORT

SYNTAX : Outputs a byte or word from port.

DESCRIPTION : The OUT instruction copies a byte from AL or a word from AX to the specified port, it can be executed in two different addressing modes, as direct and indirect.

(i) **Direct :** In direct addressing mode 8-bit address of the port is a part of the instruction.

Examples

```
OUT 0F8H, AL     ; Copy contents of AL to 8-bit port 0F8H
OUT 0F8H, AX     ; Copy contents of AX to 16-bit port 0F8H
```

(ii) Indirect : In indirect addressing, the address of the port is referred from DX register. It has advantage of accessing 2^{15} i.e., 65536 ports.

Examples

```
MOV DX, 30F8H    ; Load 16-bit address of the port in DX
MOV DX, AL       ; Copy the contents of AL to port 30F8H
MOV DX, AX       ; Copy the contents of AX to port 30F8H
```

FLAGS AFFECTED : None

5.5.2 Arithmetic Instructions

Arithmetic instructions perform addition, subtraction, multiplication and division operations. The increment and decrement operations also belongs to this type of instructions. The operands are either the registers or memory locations or immediate data depending upon the addressing mode. Arithmetic instructions change the conditional flags.

5.5.2.1 Addition Instructions

ADD : ADDITION

SYNTAX : ADD Destination, Source

DESCRIPTION : The instruction "ADD Destination, Source" adds a number from source to a number from destination and puts the result in the destination. The source may be an immediate number, a register, or a memory location. The source and the destination in an instruction cannot be both memory locations. The source and destination both must be a word or byte. If we want to add a byte to a word, we must copy the byte to a word location and fill the upper byte of the zeroes before adding.

EXAMPLES

```
ADD BL, AL    ; This instruction adds the content of register AL with BL and stores the
              ; result in BL.
```

Let us assume the contents of AL as 94H and BL as 82H. The addition of these two unsigned numbers is as follows,

Hexadecimal Addition	**Binary Addition**
Carry ⟶ 1 94 H	Carry ⟶ 1 10010100
(+) 82 H	(+) 10000010
Carry ⟶ 1 16 H	Carry ⟶ 100010110

Notice that the result of operation is 16H with a carry 1. The contents of register before and after addition operation is,

Before	After
AL = 94 H	AL = 94 H
BL = 82 H	BL = 16 H

FLAGS AFFECTED : C = 1 (Since there is a carry out of MSB)

P = 0 (Since the result is not even parity)

AC = 0 (Since the carry is not generated from bit 3 to bit 4 during addition).

Z = 0 (Since the result is not zero)

S = 0 (Set to MSB)

O = 1 (Set to 1 since the carry generated from MSB and carry generated into MSB are different)

```
ADD CX, [BX]  ; This instruction adds the word from memory location in DS at an
              ; effective address of BX and word in CX, stores the result in CX.
```

ADC : ADD WITH CARRY

SYNTAX : ADC Destination, Source

DESCRIPTION : This instruction adds a number from a source to a number from destination. This also adds status of carry flag to the result. The source can be immediate data, memory location or a register. The destination may be a register, memory location but both cannot be memory locations.

If CF is 1 then dest ← Source + destination + 1

If CF is 0 then dest ← Source + destination + 0

FLAGS AFFECTED : AF, CF, OP, PF, SF, ZF

EXAMPLES

```
ADC AX, 20H   ; Adds the content of register AX with 20H along with carry flag and stores
              the result in AX. i.e., AX ← AX + 20H + CF
```

DAA : DECIMAL ADJUST AFTER ADDITION

SYNTAX : DAA

DESCRIPTION : This instruction converts the result of the addition of two BCD numbers to a valid BCD number. The result will be stored only in AL. On execution of this instruction, if the lower nibble in greater than 9, after addition or if AF is set, then 06 will be added to the lower nibble in AL. If the result of the upper nibble in AL register is greater than 9, or if carry flag is set after addition, then 60H, is added to the AL register and the carry flag is set.

FLAGS AFFECTED : SF, ZF, AF, PF, CF, OF (Undefined)

EXAMPLES

Suppose, AL = 43H, CL = 17H.

```
                1 1 1  <-- Carry
  43 H    0 1 0 0 0 0 1 1
  17 H    0 0 0 1 0 1 1 0
  5A H    0 1 0 1 1 0 1 0 : Lower nibble
              0 1 0 1 : Add 6
          0 1 1 0 0 0 0 0
Correct result =  6     0
```

Since the lower nibble of result 5A, i.e., A > 9. DAA instruction adjust AL ← 5A + 06 and produces a result of 60H in AL.

FLAGS AFFECTED : C = 0, Z = 0, S = 0, P = 1, AC = 0.

INC : INCREMENT

SYNTAX : INC Destination

DESCRIPTION : The INC (Increment) instruction adds the value in the destination b... The operand may be a byte or a word and is treated as an unsi... binary number.

FLAGS AFFECTED : OF, SF, ZF, AF and PF. CF is not affected.

EXAMPLES

```
INC DL              ; Increment the content in register DL by 1 and stores the result in
                    ; DL i.e., DL ← DL + 1

INC BYTE PTR[SI]    ; Add 1 to the byte present in memory location whose offset address
                    ; is specified in SI specified in SI register.

INC WORD PTR[BX]    ; Add 1 to the word present in memory location whose offset address
                    ; is specified in BX register.

INC [Bx]            ; Add 1 to the byte/word present in a memory location whose index
                    ; is given by BX register.
```

AAA : ASCII ADJUST AFTER ADDITION

SYNTAX : AAA

DESCRIPTION : AAA (ASCII Adjust after Addition) changes the contents of register AL to a valid unpacked BCD number. The higher nibbles of register AL is filled with zeros. The AAA adjustment is performed as mentioned below.

STEP 1 : If the low-order 4 bits of the AL register are between 0 and 9 and the AF is 0 then go to step 3.

STEP 2 : If the low-order 4-bits of the AL register are between A and F or the AF = 1, then add 06 to the AL register, and add 1 to AH register, and set the AF to 1.

STEP 3 : Clear the higher-order four bits of the AL register.

D : AAA updates AF and CF, the contents of OF, PF, SF and ZF are undefined following execution of AAA.

EXAMPLE PROBLEM 5.11

Explain how the AAA adjusts the contents of AX after performing addition operation. Let the content of AX after addition be 3124H.

SOLUTION

Table 5.5.3

Before Instruction	After Instruction
AX = 3124H	AX = 320AH
CF = 0	CF = 0
AF = 1	AF = 1

	AH		AL	
Contents of AX =	0011	0001	0010	0100
AF = 1, ADD 06 to AL Register, 1 to AH = (+)	0000	0001	0000	0110
Clear 4 MSBs Register of AL =	0011	0010	0000	1010
	3	2	0	A

5.5.2.2 Subtraction Instructions

SUB : SUBTRACT

SYNTAX : SUB Destination, Source

DESCRIPTION : This instruction subtracts the contents of a source register, from the contents of a destination register or memory location and the result is placed in the destination location. The source and destination must both be of same size (byte or word) and both cannot be memory locations.

FLAGS AFFECTED : CF, PF, AF, ZF, SF, OF

EXAMPLE

```
SUB AX, 0200H ; Subtracts the Immediate data from AX and stores the result in AX.
SUB AX, BX    ; Subtracts the content of register BX from AX and stores the result
              ; in AX.
```


SBB : SUBTRACT WITH BORROW

SYNTAX : SBB Destination, Source

DESCRIPTION : This instruction is same as the SUB instruction except that the value in the carry flag is also subtracted. That is, the source and CF are both subtracted from the destination. The source and destination must both be of either 8-bit or 16-bit. All values are assumed to be binary.

FLAGS AFFECTED : OF, SF, ZF, AF, PF, CF.

EXAMPLE

```
SBB AX, 0200H ; The immediate data 0200H is subtracted from the contents in AX register
              ; including carry flag.
```

DEC : DECREMENT

SYNTAX : DEC Destination

DESCRIPTION : This instruction decrements the value in the destination by 1. The destination is assumed to be a binary number and can be a register (except a segment register) or memory location.

FLAGS AFFECTED : OF, SF, ZF, AF, PF

EXAMPLES

```
DEC DL ; DL ← DL – 1
DEC BX ; BX ← BX – 1
```

DAS : DECIMAL ADJUST AFTER SUBTRACTION

SYNTAX : DAS

DESCRIPTION : This instruction is used to ensure that the result of subtracting two packed BCD numbers is a valid BCD number. Hence it is used immediately after the subtraction of two packed BCD numbers. This instruction always works on AL. Hence the result of subtraction should be in AL. This instruction will modify the content of AL, if required, to give the correct answer in BCD form.

The modification is based on the following logic. If the lower nibble in AL after subtraction is greater than 9 or AF is set then subtract 06 from lower nibble of AL. If the result in the higher nibble is greater than 9 or CF is set subtract 60H from AL.

FLAGS AFFECTED : CF, AF, ZF, PF, SF. OF is undefined

EXAMPLES

```
SUB AL, CL    ; Subtract the packed number available in CL from packed number
              ; available in AL. Result goes to AL
DAS           ; Decimal adjust the result
```

The detail explanation is similar to the one given under DAA instruction. Just note that addition operations should be replaced with subtraction operations.

AAS : ASCII ADJUST AFTER SUBTRACTION

SYNTAX : AAS

DESCRIPTION : This instruction is used to ensure that the result of subtraction of two ASCII codes, of decimal numbers, gives the correct answer in unpacked BCD. Hence it is used immediately after the subtraction of two ASCII codes of decimal numbers.

This instruction always works on AL. Hence the result of subtraction should be in AL. After AAS is executed the result is in AL and AH is decremented by 1 or unchanged.

The ASCII code for numbers 0 to 9 are 30H to 39H. Hence, before subtraction, '3' of higher nibble should be masked. But 8086 will allow us to subtract these ASCII codes without masking and AAS will correct the answer as given below.

After the subtraction check Carry Flag (CF). If it is zero no modification to the answer i.e., AL or AH. If the CF is set i.e., a borrow was taken, then make higher nibble of AL zero, then subtract 6 from AL and decrement AH by 1'. This decrement allows multiple digit ASCII number to be subtracted from each other.

NOTE : *That AH should be initialized to zero before each subtraction.*

FLAGS AFFECTED : CF, AF : Other status flags (ZF, SF, PF, OF) undefined

EXAMPLE

```
MOV AH, 0     ; Initialize AH to 0 before ASCII subtraction
SUB AL, BL    ; Subtract ASCII number available in BL from Al
AAS           ; ASCII adjust the result
```


(1) Let us assume that AL has 39H and BL has 35H. After subtraction the result will be 04. Carry Flag is 0, so no modification by AAS and AL has correct answer is 04.

(2) Let us assume that AL has 37H and BL has 38H. After subtraction the AL will contain FFH and carry flag (borrow) is set. Now AAS will make higher nibble of AL zero and subtract 6 from AL and the answer will be 9 which is correct. AH will be decremented by 1 and is used by multiple digit ASCII number subtraction.

NOTE : *The result 9 is treated as 10's complement from a actual answer in a single digit subtraction. Here actual answer is –1 and 9 is 10's complement of –1.*

NEG : NEGATE

SYNTAX : NEG Destination

DESCRIPTION : This instruction performs a 2's complement of the specified destination in the instruction. To do 2's complement, it subtracts the contents of destination from zero and stores the result back in the destination. The destination can be register or memory location.

FLAGS AFFECTED : OF, SF, ZF, AF, PF, CF

EXAMPLES

```
NEG AX                ; Replace content of AX with its 2's complement.
NEG BYTE PTR [SI]     ; Replace the byte available in data segment at an offset of [SI]
                      ; with its 2's complement.
```

CMP : COMPARE

SYNTAX : CMP Destination, source

DESCRIPTION : This instruction "CMP destination, source" compares a byte or word from destination with a byte or word from source. The comparison is actually done by subtracting the source from the destination. The source and destination are not changed. The results of the comparison depends on the status of CF and ZF as given below,

```
CF   ZF
0    1     ; Destination operand = Source operand
0    0     ; Destination operand > Source operand
1    0     ; Destination operand < Source operand
```

The source can be register or a memory location or an immediate operand. The destination can be a register or memory location. The source and destination i.e., both cannot be memory locations. The source and destination must be of the same data type i.e., byte or a word

EXAMPLES

```
CMP BL, 60H      ; Compare the content of BL with 60H
CMP AX, 50H[SI]  ; Compare the content of AX with content of memory location in data
                 ; segment whose offset value 50H + SI
```

NOTE : *Compare instruction is usually followed immediately after jump instructions to change program execution flow based on result.*

5.5.2.3 Multiplication Instructions

MUL : UNSIGNED MULTIPLICATION BYTE OR WORD

SYNTAX : MUL Source.

DESCRIPTION : This instruction multiplies an unsigned binary number in a register or memory location times an unsigned number in AL if 8-bit or AX if 16-bit. If two 8-bit numbers are multiplied then a 16-bit answer will be found in AX. If two 16-bit numbers are multiplied then a 32-bit answer will be found in DX-AX (High byte in DX, low byte in AX).

FLAGS AFFECTED : AF, ZF, SF and PF

EXAMPLES

```
MUL BL   ; Contents of BL are multiplied with contents of AL and result is stored in AX.
MUL CX   ; Contents of CD register are multiplied with contents of AX register and
         ; result gets stored in DX:AX. Most significant word in DX and least
         ; significant word in AX.
```

IMUL : INTEGER MULTIPLICATION

SYNTAX : IMUL Source

DESCRIPTION : This instruction multiplies a signed number in source with a signed number in accumulator. The source can be a register or a memory location. The source cannot be an immediate operand

If the source specifies a byte then it is multiplied by AL and result will be put in AX. If the source specifies a word multiplied by AX and signed result will got to XD:AX.

In case a byte has to be multiplied by a word, convert word by extending the sign bit. Use CBW for this purpose multiply word by word and result goes to DX:AX.

FLAGS AFFECTED : CF, OF, Other status flags AF, ZF, SF, PF undefined. CF and OF will be zero if most significant part of result (MSB for byte multiplication and MSW for word multiplication) contains all 0s or all 1s. Else it is 1.

EXAMPLE

```
IMUL DL   ; Multiply signed byte of AL with signed byte of DL. Result goes to AX.
```

AAM : ASCII ADJUST AFTER MULTIPLICATION (BCD ADJUST AFTER MULTIPLY)

SYNTAX : AAM

DESCRIPTION : This instruction is used to ensure that after multiplication of two unpacked BCD digits, the correct result is available as two unpacked BCD digits. It is used immediately after multiplication instruction.

This instruction always works on AL and result will be available in AX. AH will have most significant digit and AL will have least significant digit.

If ASCII numbers are to be multiplied then mask the upper 4 bits of each before multiplication.

FLAGS AFFECTED : PF, SF and ZF. Other flags AF, CF, OF are undefined.

EXAMPLES

```
MOV AL, 05   ; Unpacked BCD 5 in AL
MOV BL, 07   ; Unpacked BCD 6 in BL
MUL BI       ; Multiply the two and result goes to AX
AAM          ; Adjust the result to two impaced BCDs.
```

Multiplication will give the answer as 0023H in AX.

After AAM, it will be 0305 in AX which is unpacked BCD for 35.

If this number has to be sent to CRT terminal then convert it to ASCII by ORing it with 3030H i.e., OR AX, 3030H.

5.5.2.4 Division Instructions

DIV : UNSIGNED DIVISION

SYNTAX : DIV Source

DESCRIPTION : This instruction can divide a 16-bit unsigned binary number in AX by an 8-bit unsigned binary number in a register or memory location. If we want to divide one 8-bit number by another, we must first change the dividend in AL into a 16-bit number by placing 00H in AH. After execution, the result (quotient) will be in AL and the remainder in AH.

DIV can also divide a 32-bit unsigned binary number in DX:AX (high order word in DX, low-order word in AX) by a 16-bit unsigned-binary number in a register or memory location. If we wish to divide one 16-bit number by another, we must first convert the dividend in AX into a 32-bit number in DX:AX by placing 0000H in DX. The result (quotient) will be in AX and the remainder in DX.

FLAGS AFFECTED : All Flags are undefined

EXAMPLES

```
DIV BL   ; The contents of AX register are divided by contents in BL and quotient
         ; is stored in AL and remainder is stored in AH.

DIV CD   ; Divides the double word in DX:AX by the contents of BX. Quotient is
         ; stored in AX and remainder is stored in DX.
```

CBW : CONVERT SIGNED BYTE TO WORD

SYNTAX : None

DESCRIPTION : This instruction takes bit 7 (the highest-order bit) of AL and duplicates it in every bit of AH. This converts an 8-bit signed-binary number in AL into a 16-bit signed-binary number in AX. This must be done before division (IDIV) involving two 8-bit signed-binary numbers to convert the dividend (in AL) into its 16-bit form (in AX). (For unsigned numbers place 00H in AH). It can also be used before multiplication (IMUL) involving an 8-bit operand and a 16-bit. The 8-bit operand can be converted to a 16-bit operand before the instruction is executed.

FLAGS AFFECTED : None

CWD : CONVERT SIGNED WORD TO DOUBLE WORD

SYNTAX : None

DESCRIPTION : This instruction is similar to the CBW instruction except that it converts 16-bit values into 32-bit values instead of 8-bit to 16-bit. It takes bit 15 (the highest order bit) of AX and duplicates it in every bit of DX. This converts a 16-bit signed-binary number in AX into a 32-bit signed-binary number in DX : AX (high 16 bits in DX, low 16 bits in AX). This must be done before division involving two 16-bit numbers to convert the dividend (in AX) into its 32-bit forms (in DX:AX).

FLAGS AFFECTED : None

IDIV : INTEGER DIVISION

SYNTAX : IDIV Source

DESCRIPTION : This instruction divides a signed number by another signed number in source. The source can be a register or a memory location. The source cannot be immediate operand.

If the dividend (numerator) is a word then it should be in AX and the divisor should be a byte. After the division, AL will contain the signed quotient and AH, the signed remainder.

If the dividend is a double word then most significant word of dividend should be in DX and least significant word of dividend should be in AX. After the division AX will contain the signed quotient and DX will contain the signed remainder.

If you want to divide signed byte by signed byte then put the dividend byte in AL and extend the sign bit to AH by using instruction CBW. This instruction fills up AH with sign bit i.e., If MSB of AL is 0 then all 0s are filled up in AH. If MSB of AL is 1 then all 1s are filled up in AH. Similarly, if you want to divide signed word by signed word, then put dividend sign word in AX and extend the sign bit to DX by using instruction CWD

The sign of the remainder will be the sign of dividend.

FLAGS AFFECTED : All flags are undefined.