B

Instruction Set Summary

Mnemonic	Description	Clock cycles	Number of bytes
AAA	ASCII adjust for addition	4	1
AAD	ASCII adjust for division	60	2
AAM	ASCII adjust for multiplication	83	2
AAS	ASCII adjust for subtraction	4	1
ADC	Add with carry		
	Register to register	3	2
	Memory to register	9 + EA	2-4
	Register to memory	16 + EA	2-4
	Immediate to register	4	3-4
	Immediate to memory	17 + EA	3-6
	Immediate to accumulator	4	2-3
ADD	Addition		
	Register to register	3	2
	Memory to register	9 + EA	2-4
	Register to memory	16 + EA	2-4
	Immediate to register	4	3-4
	Immediate to memory	17 + EA	3-6
	Immediate to accumulator	4	2-3
AND	Logical AND		
	Register to register	3	2
	Memory to register	9 + EA	2-4
	Register to memory	16 + EA	2-4
	Immediate to register	4	3-4
	Immediate to memory	17 + EA	3-6
	Immediate to accumulator	4	2-3
CALL	Call a procedure		
	Intrasegment direct	19	3
	Intrasegment indirect through register	16	2
	Intrasegment indirect through memory	21 + EA	2-4
	Intersegment direct	28	5
	Intersegment indirect	37 + EA	2-4
CBW	Convert byte to word	2	1

Mnemonic	Description	Clock cycles	Number of bytes
CLC	Clear carry flag	2	1
CLD	Clear direction flag	2	1
CLI	Clear interrupt flag	2	1
CMC	Complement carry flag	2	1
CMP	Compare Register to register Memory to register Register to memory Immediate to register Immediate to memory Immediate to accumulator	 3 9 + EA 9 + EA 4 10 + EA 4	 2 2-4 2-4 3-4 3-6 2-3
CMPS/	Compare string/		1
CMPSB/	Compare byte string/		
CMPSW	Compare word string Not repeated Repeated	 22 9 + 22/rep	
CWD	Convert word to double word	5	1
DAA	Decimal adjust for addition	4	1
DAS	Decimal adjust for subtraction	4	1
DEC	Decrement by 1 16-bit register 8-bit register Memory	 2 3 15 + EA	 1 2 2-4
DIV	Unsigned division 8-bit register 16-bit register 8-bit memory 16-bit memory	 80–90 144–162 (86–96) + EA (150–168) + EA	 2 2 2-4 2-4
ESC	Escape Register Memory	 2 8 + EA	 2 2-4
HLT	Halt	2	1
IDIV	Integer division 8-bit register 16-bit register 8-bit memory 16-bit memory	 101–112 165–184 (107–118) + EA (171–190) + EA	 2 2 2-4 2-4
IMUL	Integer multiplication 8-bit register 16-bit register 8-bit memory 16-bit memory	 80–98 128–154 (86–104) + EA (134–160) + EA	 2 2 2-4 2-4

Mnemonic	Description	Clock cycles	Number of bytes
IN	Input from I/O port Fixed port Variable port	 10 8	 2 1
INC	Increment by 1 16-bit register 8-bit register Memory	 2 3 15 + EA	 1 2 2-4
INT	Interrupt Type=3 Type≠3	 52 51	 1 2
INTO	Interrupt if overflow Interrupt is taken Interrupt is not taken	 53 4	1
IRET	Return from interrupt	24	1
JA/	Jump if above/	16/4	2
JNBE	Jump if not below or equal		
JAE/	Jump if above or equal/	16/4	2
JNB/	Jump if not below/		
JNC	Jump if not carry		
JB/	Jump if below/	16/4	2
JNAE/	Jump if not above or equal/		
JC	Jump if carry		
JBE/	Jump if below or equal/	16/4	2
JNA	Jump if not above		
JCXZ	Jump if CX is zero	18/6	2
JE/	Jump if equal/	16/4	2
JZ	Jump if zero		
JG/	Jump if greater/	16/4	2
JNLE	Jump if not less or equal		
JGE/	Jump if greater or equal/	16/4	2
JNL	Jump if not less		
JL/	Jump if less/	16/4	2
JNGE	Jump if not greater or equal		
JLE/	Jump if less or equal/	16/4	2
JNG	Jump if not greater		

Mnemonic	Description	Clock cycles	Number of bytes
JMP	Jump Intrasegment direct short Intrasegment direct Intersegment direct Intrasegment indirect through memory Intrasegment indirect through register Intersegment indirect	 15 15 15 18 + EA 11 24 + EA	 2 3 5 2-4 2 2-4
JNE/	Jump if not equal/	16/4	2
JNZ	Jump if not zero		
JNO	Jump if not overflow	16/4	2
JNP/	Jump if not parity/	16/4	2
JPO	Jump if parity odd		
JNS	Jump if not sign	16/4	2
JO	Jump if overflow	16/4	2
JP/	Jump if parity/	16/4	2
JPE	Jump if parity even		
JS	Jump if sign	16/4	2
LAHF	Load AH from flags	4	1
LDS	Load pointer using DS	16 + EA	2-4
LEA	Load effective address	2 + EA	2-4
LES	Load pointer using ES	16 + EA	2-4
LOCK	Lock bus	2	1
LODS/	Load string/		1
LODSB/	Load byte string/		
LODSW	Load word string Not repeated Repeated	 12 9 + 13/rep	
LOOP	Loop	17/5	2
LOOPE/	Loop if equal/	18/6	2
LOOPZ	Loop if zero		
LOOPNE/	Loop if not equal/	19/5	2
LOOPNZ	Loop if not zero		

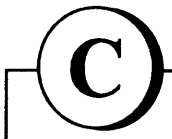
Mnemonic	Description	Clock cycles	Number of bytes
MOV	Move		
	Accumulator to memory	10	3
	Memory to accumulator	10	3
	Register to register	2	2
	Memory to register	8 + EA	2-4
	Register to memory	9 + EA	2-4
	Immediate to register	4	2-3
	Immediate to memory	10 + EA	3-6
	Register to SS, DS, or ES	2	2
	Memory to SS, DS, or ES	8 + EA	2-4
	Segment register to register	2	2
	Segment register to memory	9 + EA	2-4
MOVS/	Move string/		1
MOVSB/	Move byte string/		
MOVSW	Move word string		
	Not repeated	18	
	Repeated	9 + 17/rep	
MUL	Unsigned multiplication		
	8-bit register	70-77	2
	16-bit register	118-133	2
	8-bit memory	(76-83) + EA	2-4
	16-bit memory	(124-139) + EA	2-4
NEG	Negate		
	Register	3	2
	Memory	16 + EA	2-4
NOP	No operation	3	1
NOT	Logical NOT		
	Register	3	2
	Memory	16 + EA	2-4
OR	Logical OR		
	Register to register	3	2
	Memory to register	9 + EA	2-4
	Register to memory	16 + EA	2-4
	Immediate to accumulator	4	2-3
	Immediate to register	4	3-4
	Immediate to memory	17 + EA	3-6
OUT	Output to I/O port		
	Fixed port	10	2
	Variable port	8	1
POP	Pop word off stack		
	Register	8	1
	Segment register SS, DS, or ES	8	1
	Memory	17 + EA	2-4
POPF	Pop flags off stack	8	1

Mnemonic	Description	Clock cycles	Number of bytes
PUSH	Push word onto stack Register Segment register Memory	 11 10 16 + EA	 1 1 2-4
PUSHF	Push flags onto stack	10	1
RCL	Rotate left through carry Register with single-shift Register with variable-shift Memory with single-shift Memory with variable-shift	 2 8 + 4-bit 15 + EA 20 + EA + 4-bit	 2 2 2-4 2-4
RCR	Rotate right through carry Register with single-shift Register with variable-shift Memory with single-shift Memory with variable-shift	 2 8 + 4-bit 15 + EA 20 + EA + 4-bit	 2 2 2-4 2-4
REP	Repeat string operation	2	1
REPE/	Repeat operation while equal/	2	1
REPZ	Repeat operation while zero		
REPNE/	Repeat operation while not equal/	2	1
REPNZ	Repeat operation while not zero		
RET	Return from procedure Intrasegment Intrasegment with constant Intersegment Intersegment with constant	 8 12 18 17	 1 3 1 3
ROL	Rotate left Register with single-shift Register with variable-shift Memory with single-shift Memory with variable-shift	 2 8 + 4-bit 15 + EA 20 + EA + 4-bit	 2 2 2-4 2-4
ROR	Rotate right Register with single-shift Register with variable-shift Memory with single-shift Memory with variable-shift	 2 8 + 4-bit 15 + EA 20 + EA + 4-bit	 2 2 2-4 2-4
SAHF	Store AH into flags	4	1
SAL/	Shift arithmetic left/		
SHL	Shift logical left Register with single-shift Register with variable-shift Memory with single-shift Memory with variable-shift	 2 8 + 4-bit 15 + EA 20 + EA + 4-bit	 2 2 2-4 2-4

Mnemonic	Description	Clock cycles	Number of bytes
SAR	Shift arithmetic right		
	Register with single-shift	2	2
	Register with variable-shift	8 + 4-bit	2
	Memory with single-shift	15 + EA	2-4
	Memory with variable-shift	20 + EA + 4-bit	2-4
SBB	Subtract with borrow		
	Register from register	3	2
	Memory from register	9 + EA	2-4
	Register from memory	16 + EA	2-4
	Immediate from accumulator	4	2-3
	Immediate from register	4	3-4
	Immediate from memory	17 + EA	3-6
SCAS/	Scan string/		1
SCASB/	Scan byte string/		
SCASW	Scan word string		
	Not repeated	15	
	Repeated	9 + 15/rep	
SHR	Shift logical right		
	Register with single-shift	2	2
	Register with variable-shift	8 + 4-bit	2
	Memory with single-shift	15 + EA	2-4
	Memory with variable-shift	20 + EA + 4-bit	2-4
STC	Set carry flag	2	1
STD	Set direction flag	2	1
STI	Set interrupt flag	2	1
STOS/	Store string/		1
STOSB/	Store byte string/		
STOSW	Store word string		
	Not repeated	11	
	Repeated	9 + 10/rep	
SUB	Subtraction		
	Register from register	3	2
	Memory from register	9 + EA	2-4
	Register from memory	16 + EA	2-4
	Immediate from accumulator	4	2-3
	Immediate from register	4	3-4
	Immediate from memory	17 + EA	3-6
TEST	Test		
	Register with register	3	2
	Memory with register	9 + EA	2-4
	Immediate with accumulator	4	2-3
	Immediate with register	5	3-4
	Immediate with memory	11 + EA	3-6

Mnemonic	Description	Clock cycles	Number of bytes
WAIT	Wait while $\overline{\text{TEST}}$ pin not asserted	3 + 5n	1
XCHG	Exchange		
	Register with accumulator	3	1
	Register with memory	17 + EA	2-4
	Register with register	4	2
XLAT/	Translate	11	1
XLATB			
XOR	Logical exclusive OR		
	Register with register	3	2
	Memory with register	9 + EA	2-4
	Register with memory	16 + EA	2-4
	Immediate with accumulator	4	2-3
	Immediate with register	4	3-4
	Immediate with memory	17 + EA	3-6

❑❑❑



Debugger

A debugger is a program which allows us to load our object code program into system memory, execute the program, and debug it.

How does a debugger help in debugging a program ?

1. The debugger allows us to look at the contents of registers and memory locations after our program runs.
2. It allows us to change the contents of register and memory locations and rerun the program.
3. Some debugger allows us to stop execution after each instruction so we can check or alter memory and register contents.
4. A debugger also allows us to set a breakout at any point in our program. When we run a program, the system will execute instructions up to this breakpoint and stop. We can then examine register and memory contents to see if the results are correct at that point. If the results are correct, we can move the break point to a later point in our program. If results are not correct, we can check the program up to that point to find out why they are not correct.

In short, debugger tools can help us to isolate problems in our program.

The debugger program provides commands by which user can interact with the program which is to be debugged. A good way to learn debugger commands is to sit at the computer while reading this material and experiment with each command.

A (Assemble)

Assemble a program into machine language. The syntax is

```
A [address]
```

Address is assumed to be the offset from the address in CS, unless another segment value is given.

Examples

1. A 200 Assemble at CS:200h.

2. A Assemble from the current location.

3. A DS:1000 Assemble at DS:1000h.

Each time we press ENTER at the end of a line, DEBUG prompts us for the next line of input. Each input line starts with a segment-offset address. To terminate our input, press ENTER on a blank line. For example :

```
- A 200
6014 : 0200 mov ah, 2
6014 : 0202 mov bh, 30
6014 : 0204 int 21
6014 : 0206                    (boldface text typed by the programmer)
```

D (DUMP)

The D command displays memory on the screen as single bytes in both hexadecimal and ASCII. There are two syntax formats :

Syntax 1 : D [address]

Syntax 2 : D [range]

If address is not specified, the location begins where the last D command left off, or at location DS:0 if the command us being typed for the first time. An address may consist of a segment-offset address or just an offset :

```
D   E000:0      (segment-offset)
D   ES:200      (segment register-offset)
D   200         (offset)
```

The default segment is DS, so it is not necessary to specify segment unless we want to dump an offset from another segment location.

A range may be given, telling DEBUG to dump all bytes within the range :

```
D   100  200;  Dump DS:0100 through 0200
```

Other segment registers or absolute addresses may be used, as shown in the following examples.

Examples

1.	D	Dump 128 bytes from the last referenced location.
2.	D SS:0A	Dump the bytes at offsets 0-A from SS.
3.	D 300:0	Dump 128 bytes at offset zero from segment 0300h.
4.	D0 200	Dump offset 0-200 from DS.
5.	D 100 L 20	Dump 20th bytes, starting at offset 100h from DS.

E (Enter)

The E command places individual bytes in memory. We must supply a starting memory location where the values will be stored. If only an offset value is entered, the offset is assumed to be from DS. Otherwise, a 32-bit address may be entered, or another segment register may be used. The syntax is :

E address	Enter new byte value at address
E address [list]	Replace the contents of memory starting at the specified address with the values contained in the list.

To begin entering hexadecimal or character data at DS:200, type E 200. Press the space bar to advance to the next byte, and press ENTER to stop. To enter a string into memory starting at location CS:200, type E CS:200 "This is a string".

F (Fill)

The F command fills a range of memory with a single value or a list of values. The range must be specified as two offset addresses or segment-offset addresses. The syntax is :

F range list

Examples

1. F 100 200, ` '	Fill with spaces.
2. F CS:0100 CS:0200, FF	Fill with hex 0FF.
3. F 200 L 30 'A'	Fill 30 hex bytes with the letter 'A', starting at location 200.

G (Execute)

The G command executes the program in the memory. We can specify a starting address and a breakpoint, causing the program to stop at a given address. The syntax is :

G [= startaddress] brkptaddress [brkptaddress...]

If no breakpoint is specified, the program runs until it stops by itself and returns to DEBUG. Upto 10 breakpoints may be specified.

Examples

1. G	Execute from IP to the end of the program.
2. G 100	Execute from the IP to CS:100h and stop.
3. G = 100 500	Begin execution at offset 100h and stop before the instruction at offset 500h

H (Hexarithmetic)

The H command performs addition and subtraction on two hexadecimal numbers, entered in the following syntax :

H value1 value2

Example

H 10 20	Hexadecimal values 10 and 20 are added and subtracted
0030 0010	← displayed by Debugger

L (Load)

The L command loads a file (or disk sectors) into memory. To read a file, it is necessary to first initialize its name with the N (Name) command. If no address is specified, the file is loaded at CS:100, Debugger sets BX and CX to the number of bytes read. Syntax :

L [address]

Example :

N B:MYPROG1.COM (Initialize the file name)

L (Load at CS:100)

M(Move)

The M command copies a block of data from one memory location to another. The syntax is :

M range address

Range consists of the starting and ending locations of the bytes to be copied. Address is the target location, to which the data will be copied. All offsets are assumed to be from DS, unless specified otherwise.

Examples

1. M 200 205 300	Move bytes in the range DS:200-205 to location DS:0300.
2. M CS:500 510 CS:100	Same as above, except that all offsets are relative to the segment value in CS.

N (Name)

The N command initializes a filename (and file control block) in memory before using the load or write commands. The syntax is :

N [d :] [filename] [.ext]

Example

N b:myfile.asm

Space

Execute the next instruction and stop; if the instruction calls a procedure, execute the procedure and then stop. The LOOP instruction and string primitive instructions (SCAS, LODS, etc.) are executed completely upto the next instruction.

R (Register)

The R command may be used in one of three ways :

- Display the contents of one register, allowing it to be changed.
- Display registers, flags, and the next instruction about to be executed.
- Display the eight flag settings, allowing any or all of them to be changed.

Syntax :

R [registername]

Examples

1. R Display the contents of all registers.
2. R IP Display the contents of IP and allow it to be modified.
3. R CX Same (for the CX register).
4. R F Display all flags and change them if desired.

Once the R F command has displayed the flags, we can change an individual flag by typing its new state. For example, we could set the zero flag by typing the following two commands :

R F

ZR

S (Search)

The S command searches a range of addresses for a list of bytes or a string. The syntax is :

S range list

Examples

1. S 100 200 0F Search DS : 100 to DS : 200 for the value 0Fh.
2. S 100 200 AABB Search for the sequence AABB.
3. S 100 200 TRY Search for the word TRY.

T (Trace)

The T command executes one or more instructions from the current CS:IP location or an optional address, if specified. The contents of the registers are shown after each instruction is executed. The syntax is :

T [= address] [, value]

Examples

1. T Trace one instruction from the current location
2. T 5 Trace five instructions.
3. T = 5, 10 Start tracing at CS:5, and trace the next 16 steps.

This command traces individual loop iterations, so we can use it to debug statements within a loop. Also, the T command traces into procedure calls, whereas the P command executes a procedure call in its entirely without tracing.

U (Unassemble)

The U command translates memory into assembly language mnemonics. This is called unassembling or disassembling memory. If we don't specify an address, debugger disassembles from the location where the last U command is left off. If the command is used for the first time after loading debugger, memory is disassembled from location CS : 100. The syntax is :

Syntax 1 : U [address]

Syntax 2 : U [range]

Examples

1. U	Unassemble the next 32 bytes from the current location.
2. U 0	Unassemble 32 bytes from location 0.
3. U 100, 200	Unassemble all bytes from offset 100h-200h.

W (Write)

The W command writes a block of memory to a file or to individual disk sectors. If we want to write to a file, it must first be initialized with the N command. (If the file was just loaded either on the DOS command line or with the Load command, we do not need to repeat the Name command.)

Place the number of bytes to be written in BX and CX (BX contains the high 16 bits, and CX contains the low 16 bits). If a file is 256 bytes long, for example, the BX and CX registers will contain the following :

BX = 0000 CX = 0100

Examples

1. N MYFILE.COM	Initialize the filename MYFILE.COM on the default drive.
2. R CX 20	Set the CX register to 20h, the length of the file.
3. W	Write 20h bytes to the file, starting at CS : 100.
4. W 0	Write from location CS : 0 to the file.Q

❑❑❑

Microprocessors Lab
8086 Software Experiments

■ Programs Involving Data Transfer Instructions

1. Program Statement : Write an 8086 assembly language program to transfer a block of 256 bytes of data from offset 1000H in DS to offset 2000H in DS.

Program a) : Using loop instruction

```
          .model small
          .stack 100
          .data
          .code
          MOV AX, @data          ; [Initialize
          MOV DS, AX             ; data segment]
          MOV CX, 00FFH          ; Initialize counter
          MOV SI, 1000H          ; Initialize source pointer
          MOV DI, 2000H          ; Initialize destination pointer
BACK:     MOV AL, [SI]           ; Read byte
          MOV [DI], AL           ; Store byte
          INC SI                 ; Increment source pointer
          INC DI                 ; Increment destination pointer
          LOOP BACK              ; Repeat until CX = 0
          END
```

Program b) : Using string instruction

```
          .model small
          .stack 100
          .data
          .code
          MOV AX, @data          ; [Initialize
          MOV DS, AX             ; data segment]
          MOV ES, AX             ; Initialize extra segment with same
                                   address as for data segment
          MOV CX, 00FFH          ; Initialize counter
          CLD                    ; Clear direction flag to increment
                                   SI and DI after each iteration
          MOV SI, 1000H          ; Initialize source pointer
          MOV DI, 2000H          ; Initialize destination pointer
REP       MOVSB                  ; Decrement CX and perform MOVSB
                                 ; until CX = 0
          END
```

2. Program Statement : Write an 8086 assembly language program to transfer a block of 256 word of data from offset 000H in DS to offset 2000H in DS in the reverse order. So that byte at offset 1000H should be copied at offset 20FFH in DS.

Program : Using loop instruction

```
        .model small
        .stack 100
        .data
        .code
        MOV AX, @data          ; [Initialize
        MOV DS, AX             ; data segment]
        MOV CX, 00FFH          ; Initialize counter
        MOV SI, 1000H          ; Initialize source pointer
        MOV DI, 20FFH          ; Initialize destination pointer
BACK:   MOV AX, [SI]           ; Read word
        MOV [DI], AX           ; Store word
        INC SI                 ; Increment source pointer
        DEC DI                 ; Decrement destination pointer
        LOOP BACK              ; Repeat until CX = 0
        END
```

3. Program Statement : Write an 8086 assembly language program to interchange a block of 128 bytes of data from offset 1000H in DS and data from offset 2000H in DS.

Program :

```
        .model small
        .stack 100
        .data
        .code
        MOV AX, @data          ; [initialize
        MOV DS, AX             ; data segment]
        MOV ES, AX             ; initialize extra segment
        MOV CX, 0080H          ; initialize counter
        MOV SI, 00             ; initialize offset in DS
        MOV DI, 00             ; initialize offset in ES
BACK:   MOV AL, DS : [SI]      ; Get the byte from first block
        MOV AH, ES : [DI]      ; Get the byte from second block
        MOV ES : [DI], AL      ; Store the byte in second block
        MOV DS : [SI],AH       ; Store the byte in first block
            INC SI             ; increment source pointer
            INC DI             ; increment destination pointer
            LOOP BACK          ; Repeat until CX = 0
            END
```

4. Program Statement : Write an 8086 assembly language program to transfer a block of 256 bytes of data from offset 1000H in DS to offset 1080H in DS.

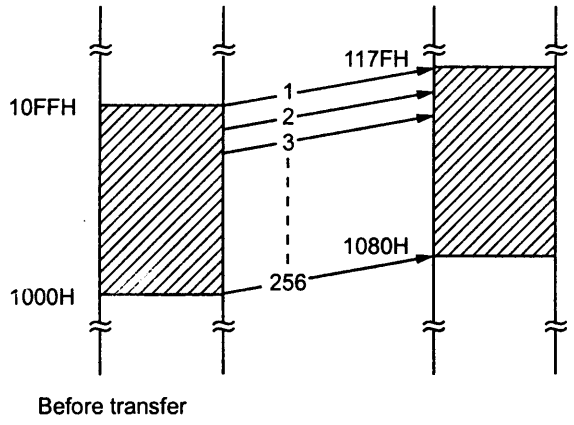


Fig. 1

The data transfer specified in the program is a overlapping transfer as shown in the Fig. 1. In this case, if we transfer first byte from source location to first byte in the destination location, data byte from the source block at location 1080H will be overwritten. To avoid such overwriting, we have to transfer data from last location in both block as shown in the Fig. 1. In numbers given in the Fig. 1 indicate sequence of byte transfer.

Program :

```
        .model  small
        .stack  100
        .data
        .code
        MOV  AX,  @data     ;  [Initialize
        mOV  DS,  AX        ;  data segment]
        MOV  ES,  AX        ;  Initialize extra segment with same
                            ;  address as for data segment
        MOV  CX,  00FFH     ;  Initialize counter
        STD                 ;  Set direction flag to decrement
                            ;  SI and DI after each iteration
        MOV  SI,  10FFH     ;  Initialize source pointer
        MOV  DI,  117FH     ;  Initialize destination pointer
   REP  MOVSB               ;  Decrement CX and perform MOVSB
                            ;  until CX = 0
        END
```

5. Program Statement : Write 8086 ALP to perform non-overlapped and overlapped block transfer :

Program ; Refer program 17 page No. 5-18 .

■ Programs Involving Arithmetic and Logical Operations

1. Program Statement : Addition of two bytes : Refer page No. 3-70.

2. Program Statement : Average of two numbers : Refer page No. 3-70.

3. Program Statement : Performs addition or subtraction : Refer page No. 3-71.

4. Program Statement : Addition of two 32-bit numbers :

```
; This program adds two numbers
```

```
TITLE Addition of two 32-bit numbers
.model small
.data
no1     dd   8111FFFFh
no2     dd   92224444h
result  dd   ?
carry   db   0
.code
start:  mov  ax, @data           ; [loads the address of data
        mov  ds, ax              ;  segment in DS]
        mov  ax,word ptr no1     ; Get the LS word of first
                                 ; number in AX
        add  ax,word ptr no2     ; Add the LS word of second
                                 ; number to it
        mov  word ptr result,ax  ; Save LS word of result
        mov  bx, offset[no1]
        mov  ax,word ptr [bx+2]  ; Get the MS word of first
                                 ; number in AX
        mov  bx, offset[no2]
        adc  ax,word ptr [bx+2]  ; Add the MS word of second
                                 ; number to it with carry
        mov  bx, offset result
        mov  [bx+2],ax           ; Save MS word of result
        adc  carry,0             ; save any carry after MS word
                                 ; addition
        mov  ah,4ch              ; [ Exit
        int  21h                 ;    to DOS ]
end  start
end
```

5. Program Statement : Program to evaluate the expression W = X * (X + Y – Z)

(Softcopy of this program, P34.asm is available at www.vtubooks.com)

```
 .MODEL  SMALL
 .DATA
    W  DW  ?
    X  DB  10H
    T  DB  30H
    Z  DB  20H
 .  CODE
START:  MOV  AX,@DATA  ; [Initialize
        MOV  DS,AX     ;  data segment]
        MOV  AL,Y
        SUB  AL,Z      ; Y - Z
        ADD  AL,X      ; X + (Y - Z)
        MOV  CL,X
        MUL  CL        ; X * (X + Y - Z)
        MOV  W,AX      ; Store result
        END  START
        END
```

6. Program Statement : Code conversion binary to BCD : Refer page No. 5-26.

7. Program Statement : Code conversion BCD to binary : Refer page No. 5-26.

8. Program Statement : Converting ASCII to binary : Refer page No. 3-83.

9. Program Statement : Converting binary to ASCII : Refer page No. 3-77.

10. Program Statement : Program to find LCM of two 16-bit unsigned numbers : Refer page No. 5-67.

11. Program Statement : Program to find HCF of two numbers. : Refer page No. 5-68.

12. Program Statement : Program to find LCM of two given numbers. : Refer page No. 5-70

13. Program Statement : Program to calculate factorial of a number. : Refer page No. 5-5

(Softcopy of this porgram, P26.asm is available at www. vtubook.com.

14. Program Statement : Multiplication of two 8-bit numbers : Refer page No. 5-32

■ Programs Involving Bit Manipulation Instructions like Checking _

1. Program Statement : Separate even and odd numbers in the array : Refer page No. 3-72.

2. Program Statement : Separate positive or negative numbers in the array.

[**Note** : This program is same as program 1 except MSB is tested instead of LSB].

3. Program Statement : Find logical 1's and 0's in a given data

```
 .model  small
.stack  100
.data
dat  db  0aH
one_c  db  0
zero_c  db  0
.code
start:  mov  ax,@data       ;  [ initialise
        mov  ds,ax          ;     data segment ]
        mov  al,dat         ;  get byte
        mov  cx,08          ;  set bit counter
back:   rol  al,1           ;  move MSB in carry
        jnc  incz           ;  check bit for 0 or 1
        inc  one_c          ;  if 1, increment one counter
```

```
next:   loop back          ; repeat until CX = 0
        end start
        end
```

4. Program Statement : Program to find whether given code is 2 out of 5 code or not.

```
; it is assume that number coded in 2 out of 5 code is
; stored in the 5 least significant bits of a byte
.model small
.stack 100
.data
dat     db 03H
one_c   db 0
mes     db 10,13,'It is a valid 2 out of 5 code $'
mes1    db 10,13,'It is not a valid 2 out of 5 code $'
.code
start:  mov ax,@data  ; [ Initialise
        mov ds,ax     ;    data segment ]
        mov al,dat    ; get code
        mov cx,05     ; initialise counter
 back:  ror al,1      ; rotate the contents of AL right by 1
        jnc next      ; if not carry skip the next instruction
        inc one_c     ; if carry is 1 increment one counter
 next:  loop back     ; repeat until CX = 0

        cmp one_c,02  ; check if one count is 2
        lea dx,mes1   ; if not set pointer to corresponding
                      ; message
        jnz skip
        lea dx,mes    ; if yes set pointer to corresponding
                      ; message
 skip:  mov ah,09h    ; load function number
        int 21h       ; call the function

        mov ah,4ch    ; [exit
        int 21h       ;  to DOS ]
        end start
        end
```

5. Program Statement : Program to find whether given word is nibblewise palindrome or not.

```
.model small
.stack 100
.data
dat     dw 8228H
temp    dw 0
mes     db 10,13,'The word is nibblewise palindrome $'
mes1    db 10,13,'The word is not nibblewise palindrome $'
```

```
.code
start:  mov ax,@data  ; [ Initialise
        mov ds,ax     ;    data segment ]
        mov dx,dat    ; get word
        mov bx,dx     ; make a copy of word

        mov ch,02     ; initialise iteration counter

        mov cl,04     ; initialise rotation counter
back:   rol dx,cl     ; rotate the contents of DX left by 4
        mov temp,dx
        and dx,000fh
        mov ax,bx
        and bx,000fh
        cmp bx,dx
        jnz ter       ; if not carry skip the next instruction
        mov bx,ax     ; restore contents of BX
        mov dx,temp   ; restore contents of DX

        ror bx,cl     ; rotate the contents of BX right by 4
        dec ch        ; decrement iteration counter
        jnz back      ; if counter is not zero repeat

        mov ah,09h    ; load function number
        lea dx,mes    ; set pointer to message
        int 21h
        jmp last

ter:    mov ah,09h    ; load function number
        lea dx,mes1   ; set pointer to message1
        int 21h

last:   mov ah,4ch    ; [ exit
        int 21h       ;    to DOS ]
        end start
        end
```

6. Program Statement : Program to find whether given word is bitwise palindrome or not.

```
.model  small
.stack  100
.data
dat     dw  8b31H
one_c   db  0
mes     db  10,13,'The word is palindrome $'
mes1    db  10,13,'The word is not palindrome $'
```

```
.code
start:  mov ax,@data  ; [ Initialise
        mov ds,ax     ;    data segment ]
        mov ax,dat    ; get word
        mov bx,ax     ; make a copy of word
        mov cx,08     ; initialise counter
back:   ror ax,1      ; rotate the contents of AX right by 1
        jnc next      ; if not carry skip the next instruction
        rol bx,1      ; rotate the contents of BX left by 1
        jc do         ; bits are palindrome so continue
        jmp ter       ; else terminate
next:   rol bx,1      ; rotate the contents of BX left by 1
        jc ter        ; else terminate
do:     loop back     ; repeat until CX = 0

        mov ah,09h    ; load function number
        lea dx,mes    ; set pointer to message
        int 21h
        jmp last

ter:    mov ah,09h    ; load function number
        lea dx,mes1   ; set pointer to message1
        int 21h

last:   mov ah,4ch    ; [ exit
        int 21h       ;    to DOS ]
        end start
        end
```

7. Program Statement : Write an ALP that will pack four 12-bit quantities in four consecutive words into three consecutive words.

```
.model small
.stack 100
.data
NO1 DW 0 123H, 0456H, 0789H, 0ABCH
NO2 DW 3 DUP (0)
.code
START: MOV AX, @DATA
       MOV DS, AX
       XOR DI, DI
       MOV AX, NO1 [DI]
       MOV CL, 04H
       ROR AX, CL
       MOV BX, NO1 [DI+2]
       MOV CL, 08H
```

```
AND BX, 0F00H
ROR BX, CL
OR AX, BX
MOV NO2 [DI], AX
INC DI
INC DI
MOV AX, NO1 [DI]
ROR AX, CL
AND AX, FF00H
MOV BX, NO1 [DI+2]
MOV CL, 04
ROL BX, CL
OR AL, BH
MOV NO2 [DI], AX
MOV CL, 08H
ROL BX, CL
MOV AX, BX
INC DI
INC DI
MOV BX, [DI+2]
OR AH, BH
OR AL, BL
MOV NO2 [DI], AX
END START
```

■ Programs Involving Branch/Loop Instructions

1. **Program Statement :** Sum of array : Refer page No. 3-71.
2. **Program Statement :** Find maximum number in the array : Refer page No. 3-70.
3. **Program Statement :** Search a number in the array : Refer page No. 3-71.
4. **Program Statement :** Program to sort array by bubble sort method

(Softcopy of this program, P23.asm is available at www.vtubooks.com)

```
.model small
.data
    M1      DB 10,13,'Enter number of elememts in the array :$'
    M2      DB 10,13,'The sorted array is :',0dh,0ah,'$'
    M3      DB 10,13,'$'
    ARRAY   DB 100 DUP(0)
.code
START:      MOV AX,@data          ; [ Initialise
            MOV DS,AX             ;    data segment ]
            MOV AH,09H            ; Display message M1
            MOV DX,OFFSET M1
            INT 21H
```

```
        MOV  AH,01H          ; Input the number of elements
                             ; in the array
        INT  21H
        SUB  AL,30H          ; Get the number from its
                             ; ASCII code
        MOV  BH,AL           ; Save the number of elements
        MOV  BL,AL           ; Save another copy of the
                             ; number of elements
        XOR  DI,DI           ; Clear pointer
        MOV  CL,4            ; Initialize rotation count
BACK:   MOV  AH,09H          ; Carrige return and next line
        MOV  DX,OFFSET M3
        INT  21H
        MOV  AH,01           ; [Read 2 digits of a
        INT  21H             ;  number
        SUB  AL,30H          ;  from
        SAL  AL,CL           ;  keyboard
        MOV  CH,AL           ;  and
        MOV  AH,01           ;  from the
        INT  21H             ;  packed number]
        SUB  AL,30H
        OR   CH,AL
        MOV  [ARRAY+DI],CH   ; Save the packed number
        INC  DI              ; Increment pointer
        DEC  BL              ; Decrement count
        JNZ  BACK            ; If count not = 0 goto BACK
        MOV  CH,BH           ; Get the number of elements
                             ; as a counter
AGAIN1: MOV  DI,0            ; Initialize pointer to 0
AGAIN:  MOV  DL,[ARRAY+DI]   ; Get the number
        CMP  DL,[ARRAY+DI+1] ; Compare if with the next
                             ; number
        JBE  SKIP            ; If it is less or equal
                             ; goto skip
        MOV  AH,[ARRAY+DI+1] ; [ Otherwise
        MOV  [ARRAY+DI],AH   ;    exchange
        MOV  [ARRAY+DI+1],DL ;    two numbers ]
SKIP:   INC  DI              ; Increment pointer
        MOV  AH,0            ; [ Check for
        MOV  AL,BH           ;    last number
        DEC  AL              ;    in the array ]
        CMP  AX,DI
        JNZ  AGAIN           ; if no goto AGAIN
        DEC  CH              ; Decrement counter
        JNZ  AGAIN1          ; If counter not = 0 goto
                             ; AGAIN1
```

```
        MOV  AH,09H          ; Display message M1
        MOV  DX,OFFSET  M2
        INT  21H
        MOV  BL,BH           ; Get the number of
                             ; elements as a counter
        XOR  DI,DI           ; Clear pointer
BACK1:  MOV  AH,09H          ; Carrige return and next
                             ; line
        MOV  DX,OFFSET  M3
        INT  21H
        MOV  DL,[ARRAY+DI]   ; [ Display
        AND  DL,0F0H         ;   the
        SAR  DL,CL           ;   sorted
        ADD  DL,30H          ;   numbers
        MOV  AH,02           ;   in
        INT  21H             ;   the
        MOV  DL,[ARRAY+DI]   ;   Array ]
        AND  DL,0FH
        ADD  DL,30H
        MOV  AH,02
        INT  21H
        INC  DI
        DEC  BL
        JNZ  BACK1
TER:    MOV  AH,4CH              ; [ Terminate
        INT  21H                 ;     and Exit to DOS ]
        END  START
```

5. Program Statement : Write a program to sort given 16-bit unsigned numbers in the ascending order by insertion sort method.

Let us consider a array of 4 numbers : array[0] = 52, array[1] = 14, array[2] = 68, array[3] =38. In the insertion sort method, we treat first number (52) as sorted. Then we insert the second number (14) in the correct position, so that first two elements in the array are sorted. Then we insert the third number (68) in the correct position, so that first three elements in the array are sorted. This process is continued till the last number.

(Softcopy of this program, P35.asm is available at www.vtubooks.com)

```
NAME Insertion Sort
TITLE Sorting of numbers in ascending order
.MODEL SMALL
.STACK 100
.DATA
    NUM DW 0052H,0014H,0068H,0038H
.CODE
```

```
START: MOV  AX,@DATA ;  [Initialise
       MOV  DS,AX    ;   data segment ]
       MOV  DX,2     ; Value in DX indicates the location of
                     ; number to be inserted
LOOP2: MOV  CX,DX
       DEC  CX       ; Value in CX indicated the maximum
                     ; number of comparisons required to insert
                     ; insert the element in the correct position
       MOV  SI,CX        ; [As numbers are 16-bit they are at
                         ;  an offset of 2, i.e. 0,2,4,6 from
       ADD  SI,SI        ;  location A. Thus SI is loaded with
                         ;  CX * 2 ]
       MOV  AX,NUM[SI]   ; Get the number
LOOP1: CMP  NUM[SI-2],AX ; Compare it with previous number
       JBE  NEXT         ; if previous number is below or
                         ; equal go to NEXT
       MOV  DI,NUM[SI-2] ; [ Insert the number
       MOV  NUM[SI],DI   ;    in proper position ]
       DEC  SI           ; [ Adjust pointer by
       DEC  SI           ;    decrementing SI twice ]
       DEC  CX           ; Decrement comparison counter
       JNZ  LOOP1        ; If not zero, repeat
NEXT:  MOV  NUM[SI],AX   ; Insert the number in proper position
       INC  DX           ; Points to next number to be
                         ; inserted
       CMP  DX,4         ; Check if all numbers are inserted
       JBE  LOOP2        ; If not continue
       MOV  AH,4CH       ; [ Terminate and
       INT  21H              Exit to DOS ]
       END  START
```

6. Program Statement : Sorting of array in the descending order by selection sort method.

Write a program to sort given 8-bit unsigned numbers in the desending order by selection sort method.

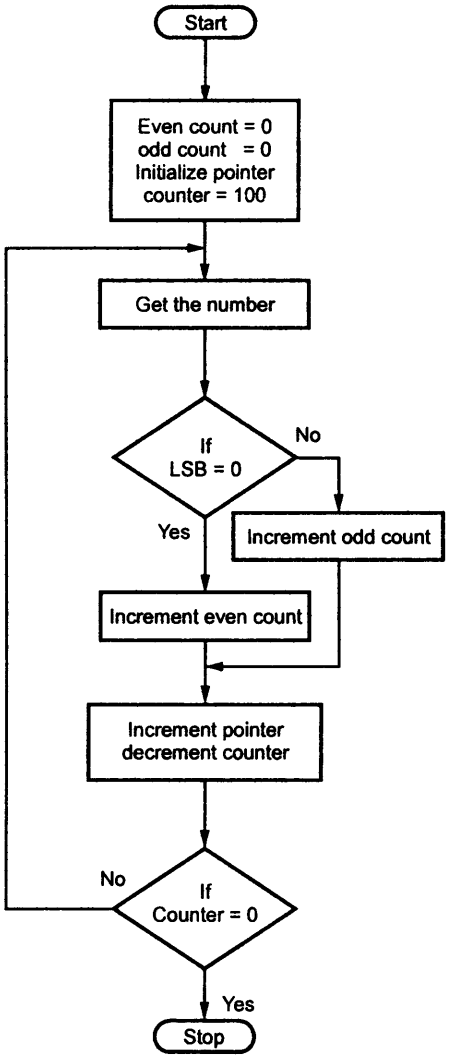
In the selection sort the smallest number is searched and it is replaced with the last number in the array for desending order sorting.

(Softcopy of this program, P36.asm is available at www.vtubooks.com)

```
NAME Selection Sort
TITLE Sorting of numbers in desending order
.MODEL SMALL
 .STACK 100
 .DATA
 NUM DB 12H,45H,79H,66H
 .CODE
START:  MOV AX,@DATA  ; [ Initialise
        MOV DS,AX     ;   data segment ]
        MOV BX,3      ; The value in BX gives the number of
                      ; passes required to complete the sorting
LOOP2:  MOV CX,BX     ; The value in CX gives the number of
                      ; passes required in a one pass
        MOV DI,0      ; Points the first element in array

        MOV AL,NUM[DI] ; Get the number
        MOV DX,DI      ; Save the pointer
LOOP1:  INC DI         ; Points the next element in the array
        CMP AL,NUM[DI] ; Compare number with the next number
        JB NEXT        ; If below go to NEXT
        MOV AL,NUM[DI] ; [ Save the smallest number in AL and
        MOV DX,DI      ;    save its pointer in DX ]
NEXT:   LOOP LOOP1     ; Repeat until CX = 0
        XCHG AL,NUM[DI]; replace smallest number with the last
        MOV NUM[DX],AL ; number in the array
        DEC BX         ; Decrement pass count
        JNZ LOOP2      ; If pass count is not = 0, repeat
        MOV AH,4CH     ; [ Terminate and
        INT 21H            Exit to DOS ]
        END START
```

7. Program Statement : Program to count the even and odd numbers from given list of numbers.

```
.model small
.STACK 100
.data
    ARRAY DW 12H, 13H, ...
.code
START :    MOV AX, @data        ; [Initialise
           MOV DS, AX           ;  data segment ]
           MOV CL, 64H          ; Count = 100
           XOR DI, DI           ; Initialise evencount = 0
           XOR SI, SI           ; Initialise oddcount = 0
           LEA BP, ARRAY        ; Initialise array base pointer
BACK :     MOV AX, DS:[BP]      ; Get the number
```




```
            AND  AX  0001H          ; Mask all bits except LSB
            JZ  NEXT                ; If LSB = 0 go to next
            INC  SI                 ; Increment odd count
            JMP  SKIP
NEXT  :     INC  DI                 ; increment even count
SKIP  :     INC  BP                 ; increment pointer
            DEC  CL                 ; decrement counter
            JNZ  BACK               ; if not 0 go to back
            END  START
```

8. Program Statement : Write an 8086 assembly language program (ALP) to add array of N number stored in the memory : Refer page No. 5-14.

9. **Program Statement :** Write 8086 ALP to find and count negative numbers from the array of signed numbers stored in memory : Refer page No. 5-23.

10. **Program Statement :** Sort given array in the assending or descending order as per the instruction given by the user : Refer page No. 5-62.

■ Programs on String Manipulations

1. **Program Statement :** Read and validate password : Refer page No. 5-4.

2. **Program Statement :** Reverse the words in string : Refer page No. 5-7.

3. **Program Statement :** Search numbers, alphabets, special characters in a given string : Refer page No. 5-9.

   (Softcopy of this program P20.asm is available at www.vtubooks.com)

4. **Program Statement :** Program to find whether a string is palindrome or not. : Refer page No. 5-12.

5. **Program Statement :** Program to display given string in lower case : Refer page No. 5-13.

6. **Program Statement :** Program to search a given byte in the string. : Refer page No. 5-66.

7. **Program Statement :** Program to perform binary search.

```
.model small
.stack 100
.data
Result dB?
ARRAY DB 01H, 02H, 03H, 04H, 05H
KEY EQU 02H
LEN EQU 05H
.code
START: MOV AX, @data    ; [Initialize
       MOV DS, AX       ;  data segment]
       MOV BL, 1
       MOV DL, LEN
       MOV CL, KEY
AGAIN: CMP BL, DL
       JA FAIL
       MOV AL, BL
       ADD AL, DL
       SHR, AL, 1
       MOV AH, 00
```

```
        MOV  SI,  AX
        DEC  SI
        ADD  SI,  SI
        CMP  CL,  ARRAY  [SI]
        JAE  BIGER
        DEC  AL
        MOV  DL,  AL
        JMP  AGAIN
BIGER:  JE  SUCCESS
        INC  AL
        MOV  BL,  AL
        JMP  AGAIN
SUCCESS:   MOV  RESULT,  AL
        END  START
```

8. Program Statement : Write 8086 ALP for the following operations on the string entered by the user.

a. Calculate length of the string.

b. Reverse the string

c. Check whether the string is palindrome or not.

Make your program user friendly by providing MENU like :

a. Enter the string.

b. Calculate length of string.

c. Reverse string.

d. Check palindrome.

e. Exit.

Refer page No. 5-44.

9. Program Statement : Write 8086 ALP to perform string manipulation. The strings to be accepted from the user is to be stored in code segment Module_1 and write FAR PROCEDURES in code segment Module_2 for following operations on the string.

a. Concatenation of two strings.

b. Compare two strings.

c. Number of occurrences of a sub-string in the given string.

d. Find number of words, characters and capital letters from the given text in the data segment.

Note : Use PUBLIC and EXTERN directive. Create •OBJ files of both the modules and link them to create an EXE file.

In this experiment we have to write two •asm programs one for accepting strings and one for procedures.

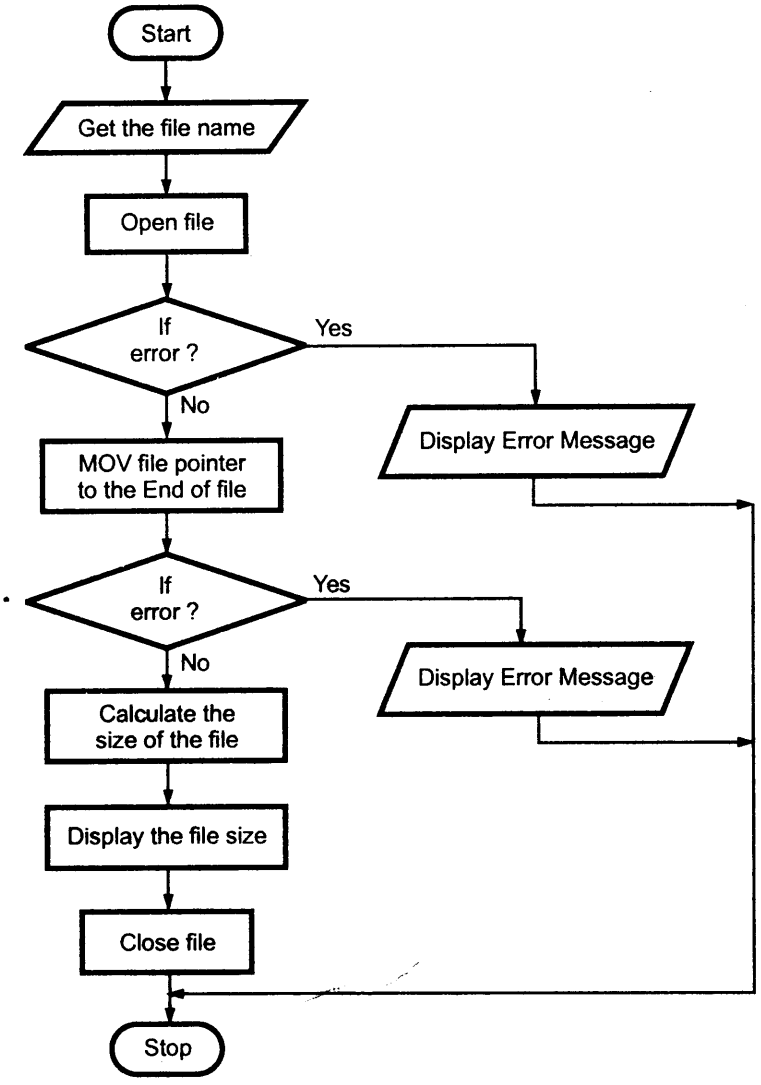
Refer page No. 5-52.

■ Programs to use DOS Interrupt INT21H Function Calls

1. **Program Statement :** Reading a character from keyboard : Refer page No. 4-16.
2. **Program Statement :** Buffered keyboard input : Refer page No. 4-17.
3. **Program Statement :** Display of character : Refer page No. 4-19.
4. **Program Statement :** Display of string : Refer page No. 4-20.
5. **Program Statement :** Program to find file size

   (Softcopy of this program, P29.asm is available at www.vtubooks.com)

Flowchart :



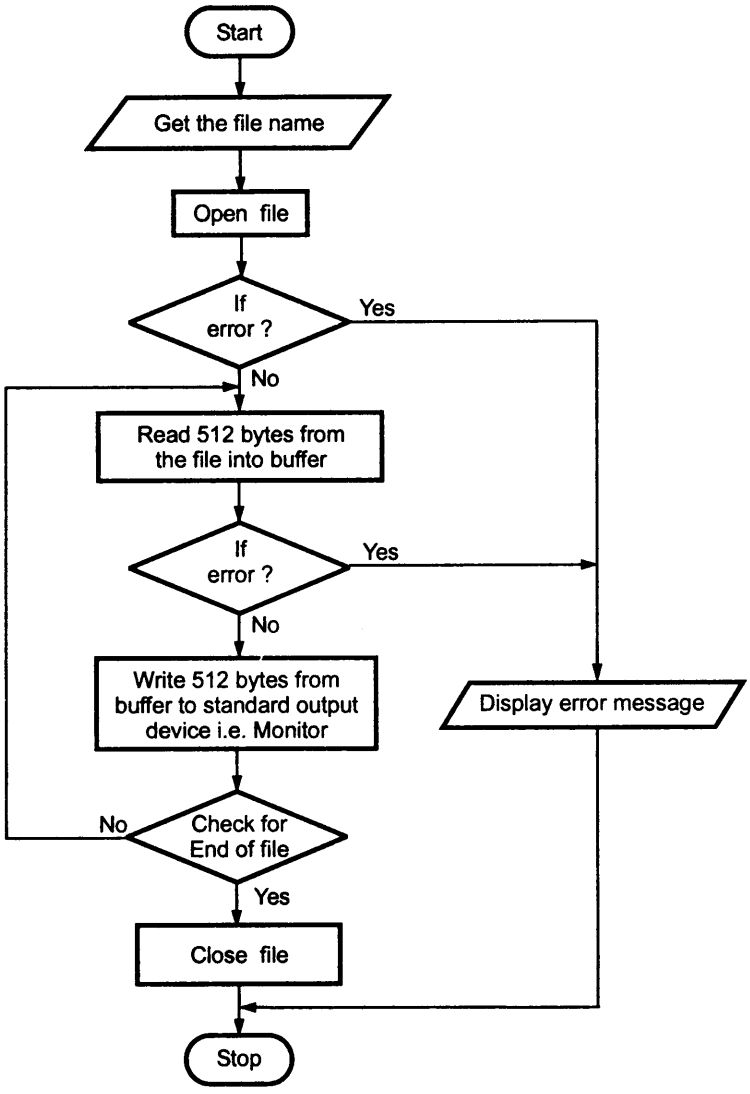
```
.model small
.stack 100
TITLE  FILE   SIZE
.data
   BUFFER     DB 30
   ACTUAL     DB 06
   FILE       DB 30 DUP (0)
   HANDLE     DW 0
   M1         DB 10,13,'ENTER THE FILE NAME : $'
   M2         DB 10,13,'FILE OPEN ERROR $'
   M3         DB 10,13,'FILE CLOSE ERROR $'
   M4         DB 10,13,'FILE POINTER ERROR$'
   M5         DB 10,13,'FILE SIZE IS $'
   SIZE1      DW 0
   SIZE2      DW 0
.code
START: MOV    AX,@data
       MOV    DS,AX            ; Display message M1
       MOV    AH,09H
       LEA    DX,M1
       INT    21H
       MOV    AH,0AH           ; I/P the file name
       LEA    DX,BUFFER
   INT    21H
   LEA    BX,ACTUAL          ; Load number of character
                             ; in file
   MOV    CL,[BX]            ; Name in CX
   MOV    CH,0
   INC    BX
   ADD    BX,CX
   MOV    BYTE PTR[BX],0     ; Convert file name to
                             ; ASCII
   MOV    AH,3DH             ; By putting 0 at it's end
                             ; string
   MOV    AL,0
   LEA    DX,FILE            ; Open file
   INT    21H
   JC     ERROR2             ; If error display error
                             ; Message else
   MOV    HANDLE,AX          ; Save file handle
   MOV    AH,42H
   MOV    BX,HANDLE          ; MOV file pointer to the
                             ; end Of file
   MOV    AL,2
```

```
        MOV     CX,0
        MOV     DX,0
        INT     21H
        JC      ERROR3
        MOV     SIZE1,DX        ; DX:AX gives new offset of
                                ; file Pointer
        MOV     SIZE2,AX
        MOV     AH,9
        LEA     DX,M5
        INT     21H
        MOV     DX,SIZE1
        MOV     AX,SIZE2
        MOV     BX,10
        MOV     CX,5
BACK1:  DIV     BX              ; Convert hex to BCD
        PUSH    DX
        XOR     DX,DX
        LOOP    BACK1
        MOV     CX,5
BACK2:  POP     DX              ; Display file size
        ADD     DL,30H
        MOV     AH,2
        INT     21H
        LOOP    BACK2
        JMP     EXIT
ERROR2: MOV     AH,9            ; Display file open error
        LEA     DX,M2
        INT     21H
        JMP     EXIT
ERROR3: MOV     AH,9
        LEA     DX,M3           ; Display file close error
        INT     21H
        JMP     EXIT
ERROR4: MOV     AH,9            ; Display file pointer error
        LEA     DX,M4
        INT     21H
        JMP     TERMIN
EXIT: MOV   AH,3EH              ; Close the file
        MOV     BX,HANDLE
        INT     21H
        JC      ERROR4
TERMIN: MOV     AH,4CH          ; Terminate & exit to DOS
        INT     21H
        END     START
```

6. Program Statement : Simulation of type command of DOS.

(Softcopy of this program, P30.asm is available at www.vtubooks.com)

Flowchart :




```
.model small
;SIMULATION OF TYPE COMMAND OF DOS
.data
ERROR   DB  13,10,'UNSUCCESSFUL : $'
MES_1   DB  13,10,'ENTER FNAME : ',0ah,0dh, '$'
MES_2   DB  13,10, 'FILE IS TYPED $'
FNAME   DB  22              ; Buffer for type command
```

```
        DB  0
        DB  22  DUP  (0)
BUFF    DB  2048  DUP  (0)
HANDLE  DW  ?               ;  Buffer  for  HANDLE
.code
BEGIN:  MOV  AX,@data       ;  Initialise  data  SEGMENT
        MOV     DS,AX
        MOV     AH,09H
        LEA     DX,MES_1    ;  Display  message
        INT     21H
        MOV     AH,0AH      ;  Enter  file  name
        LEA     DX,FNAME
        INT     21H
        MOV     SI,DX           ;  Make  ASCIIZ
        MOV     BL,[SI+1]
        ADD     SI,02H
        MOV     BYTE  PTR[SI+BX],00H
                                ;  Open  the  file
        MOV     AH,3DH          ;  Open  file  to  type
        MOV     AL,02H
        MOV     DX,OFFSET  FNAME[2]
        INT     21H
        JC      UNS
        MOV     HANDLE,AX       ;  Store  handle
        XOR     SI,SI
ME:     MOV     AH,3FH          ;  Read  file
        MOV     BX,HANDLE
        MOV     CX,512
        LEA     DX,BUFF
        INT     21H
        JC      UNS
        MOV     SI,AX
        MOV     AH,40H          ;  Write  file
        MOV     BX,0001
        MOV     CX,SI
        LEA     DX,BUFF
        INT     21H
        CMP     CX,512
        JE      ME
        MOV     AH,3EH          ;  Close  file
        MOV     BX,HANDLE
        INT     21H
        JMP     AWAIT
UNS:    MOV     AH,09H
```

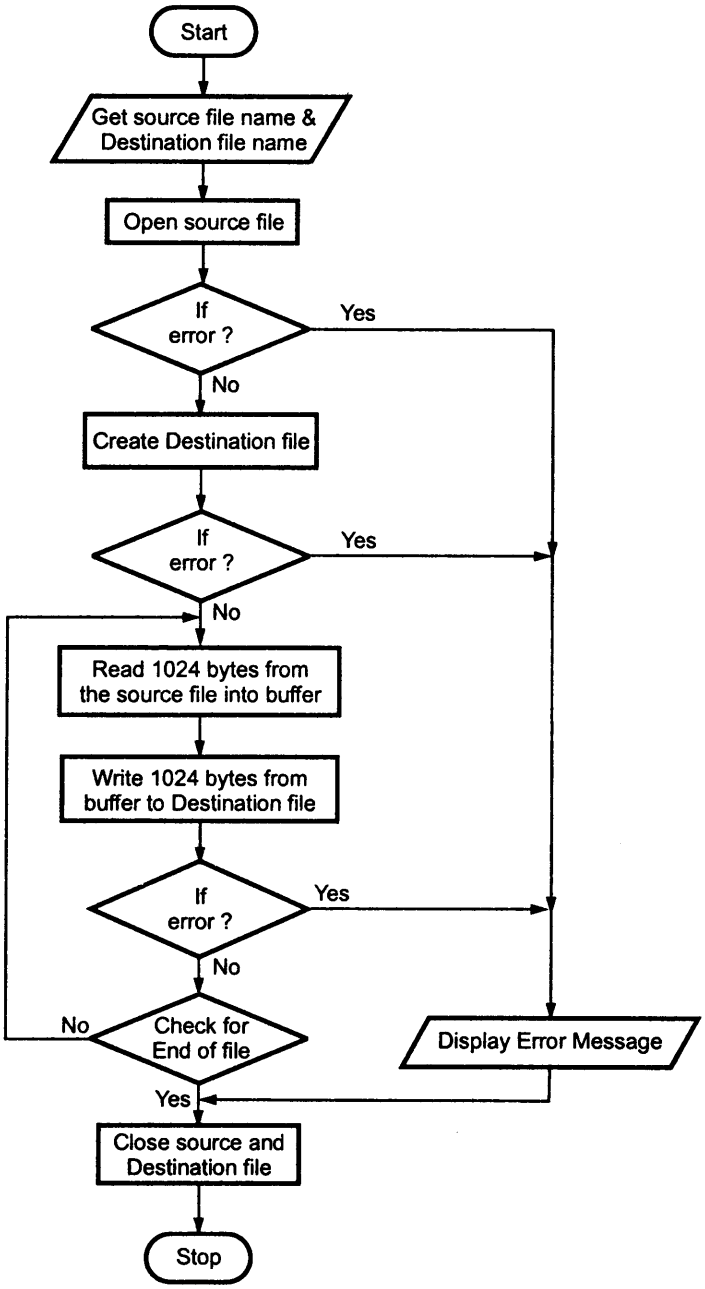
```
        MOV     DX,OFFSET  ERROR
        INT     21H
AWAIT:  MOV     AH,09H
        MOV     DX,OFFSET  MES_2   ; File  is  typed
        INT     21H
        MOV     AH,4CH             ; Terminate
        INT     21H
        END     BEGIN
        END
```

7. Program Statement : Simulation of copy command of DOS.

(Softcopy of this program, P31.asm is available at www.vtubooks.com)

Flowchart : (See Flowchart on next page)

```
.model  small
;   SIMULATION  OF  COPY  COMMAND  OF  DOS
.data
    MES_1   DB  13,10,'ENTER  THE  SOURCE  FILE  NAME  :  $'
    MES_3   DB  13,10,'ENTER  THE  DESTINATION,  FILE  :  $'
    MES_4   DB  13,10,'THE  FILE  IS  COPIED  :  $'
    MES_2   DB  13,10,'UNSUCCESSFUL  $'
    FNAME1  DB  80                    ;  Source  buffer
            DB  0
            DB  80  DUP  (0)
    FNAME2  DB  80
            DB  0
            DB  80  DUP  (0)
    HANDLE1 DW  ?
    HANDLE2 DW  ?
    BUFF    DB  2048  DUP  (0)
    SAVE_AX DW  0
.code
BEGIN  :    MOV     AX,@data          ;  Initialise  data  segment
            MOV     DS,AX
            MOV     AH,09H
            MOV     DX,OFFSET  MES_1  ;  Display  message  1
            INT     21H
            MOV     AH,0AH
            LEA     DX,FNAME1         ;  Read  source  file  name
                                      ;  from
            INT     21H               ;  Key  board
            MOV     SI,DX
            MOV     BL,[SI+1]
            ADD     SI,02H
            MOV     BYTE  PTR[SI+BX],00H
```




```
MOV     AH,09H
MOV     DX,OFFSET  MES_3    ;  Display  message  3
INT     21H
MOV     AH,0AH
LEA     DX,FNAME2           ;  Read  destination  file  name
INT     21H                 ;  From  key  board
SUB     BH,BH               ;  Clear  BX
```

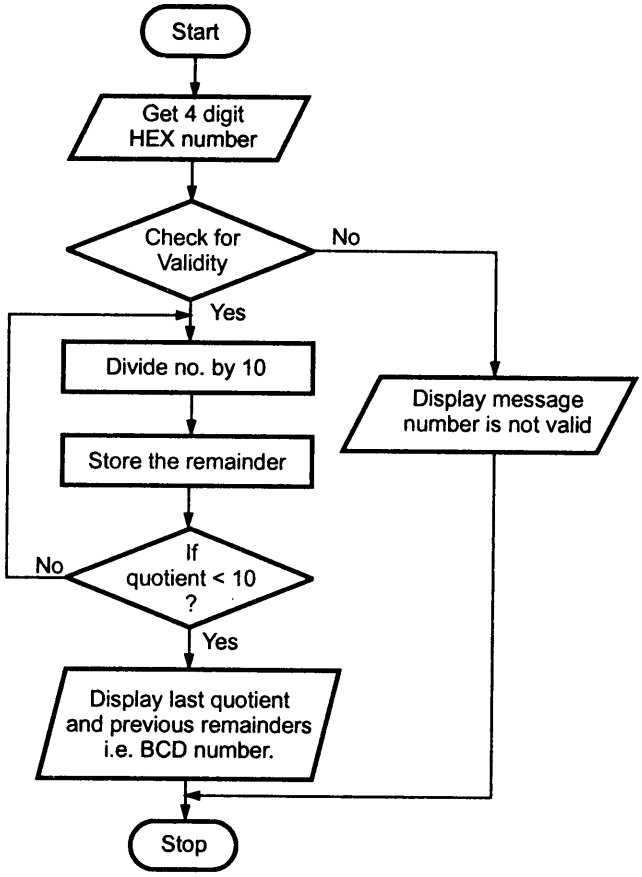
```
        MOV     SI,DX
        MOV     BL,[SI+1]
        ADD     SI,02H
        MOV     BYTE PTR[SI+BX],00H
                                ; Open the source file
        MOV     AH,3DH
        MOV     AL,02H
        MOV     DX,OFFSET FNAME1[2]
        INT     21H
        JC      UNS
        MOV     HANDLE1,AX      ; Create the destination
                                ; file
        MOV     AH,3CH
        MOV     CX,0
        MOV     DX,OFFSET FNAME2[2]
        INT     21H
        JC      UNS
        MOV     HANDLE2,AX      ; Read the source file
ME :    MOV     AH,3FH
        MOV     BX,HANDLE1
        MOV     DX,OFFSET BUFF
        MOV     CX,1024
        INT     21H
        JC      UNS
        MOV     SAVE_AX,AX
        CMP     AX,0
        JE      DONE            ; Write to destination file
        MOV     AH,40H
        MOV     BX,HANDLE2
        MOV     DX,OFFSET BUFF
        MOV     CX,SAVE_AX
        INT     21H
        JC      UNS
        JMP     ME              ; Close the files
        MOV     AH,3EH
        MOV     BX,HANDLE1
        INT     21H
        MOV     BX,HANDLE2
        INT     21H
        JMP     DONE            ; Error message
UNS :   MOV     AH,09H
        MOV     DX,OFFSET MES_2
        INT     21H
        JMP     TER
```

```
DONE  :     MOV     AH,09H
            MOV     DX,OFFSET  MES_4
            INT     21H
TER   :     MOV     AH,4CH              ;  [ Terminate and
            INT     21H                 ;     Exit to DOS ]
            END     BEGIN
```

8. Program Statement : Program to convert HEX to BCD.

(Softcopy of this program, P27.asm is available at www.vtubooks.com)

Flowchart :




```
.model  small
.stack  100
.data
    CR      EQU     0AH
    LF      EQU     0DH
    INP     DB2     DUP(0)
```

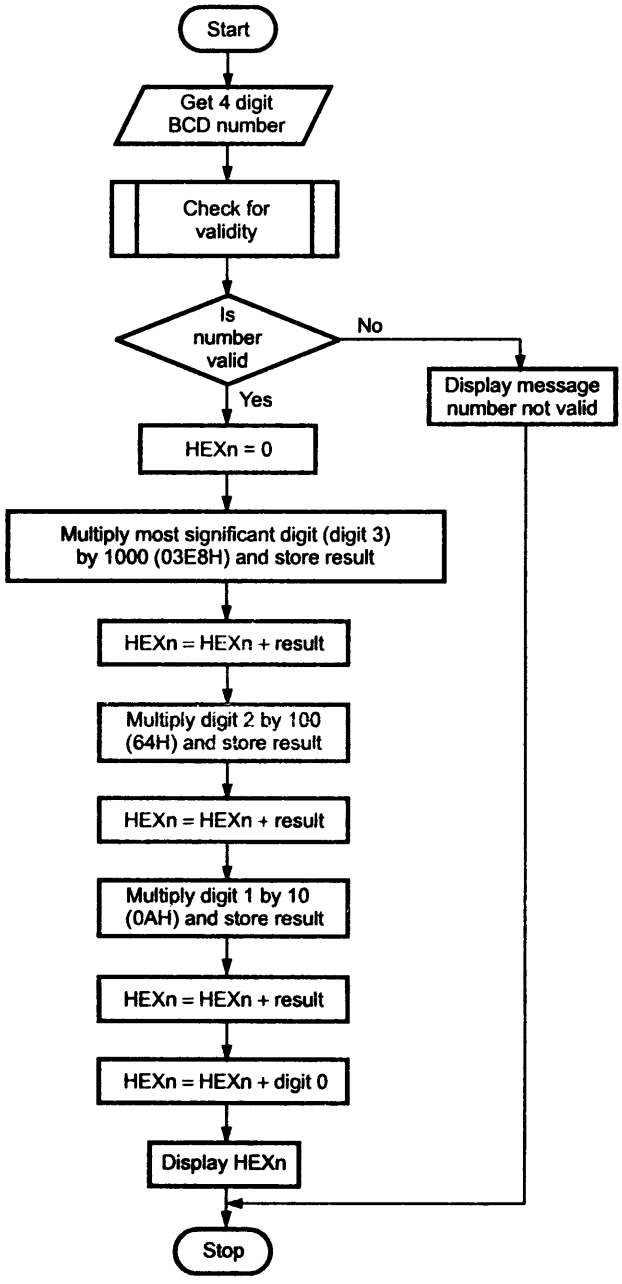
```
MES1    DB      CR,LF,'ENTER HEX NO $'
MES2    DB      CR,LF,'BCD NO $',CR,LF
MES3    DB      CR,LF,'INPUT IS INVALID HEX NO $'
BUFF    DB      05
        DB      00
        DB      05 DUP(0)
CNT1    DW      0

.code
MAIN :  MOV     AX,@data        ; [Initialise
        MOV     DS,AX           ; data segment]
                                ; HEX to BCD
        MOV     AH,09H
        LEA     DX,MES1         ; Display string
        INT     21H
        MOV     AH,0AH          ; Buffered keyboard I/P
        LEA     DX,BUFF
        INT     21H
        MOV     CH,02H
        XOR     DI,DI
        MOV     SI,01H
        INC     DI
        INC     DI
BACK:   MOV     AH,BUFF[DI]
                                ; ASCII to HEX
        CMP     AH,60H
        JG      ALPHA
        CMP     AH,3AH
        JL      NEXT
        SUB     AH,37H
        JMP     DISP
NEXT:   SUB     AH,30H
        JMP     DISP
ALPHA:  SUB     AH,57H
DISP:
        MOV     CL,04H
        ROR     AH,CL
        INC     DI
        MOV     AL,BUFF[DI]
        CMP     AL,60H
        JG      ALPHA1
        CMP     AL,3AH
        JL      NEXT1
        SUB     AL,37H
        JMP     DISP1
```

```
NEXT1:      SUB     AL,30H
            JMP     DISP1
ALPHA1:     SUB     AL,57H
DISP1:
            OR      AH,AL
            MOV     INP[SI],AH
            DEC     SI
            INC     DI
            DEC     CH
            JNZ     BACK
            MOV     CNT1,00H
            MOV     AX,WORD PTR INP
LOOP2:      MOV     CH,00H
            MOV     BX,0AH
            MOV     DX,00H
            DIV     BX              ; Divide the number by 10
            PUSH    DX              ; Store the remainder
            INC     CNT1
            CMP     AX,0
            JNZ     LOOP2
            MOV     AH,09H
            LEA     DX,MES2         ; Display string
            INT     21H
DIS :       POP     DX
            MOV     AH,02H
            ADD     DL,30H
            INT     21H
            DEC     CNT1
            CMP     CNT1,0
            JNZ     DIS
            JMP     EXIT
TER:        MOV     DX,00H
            MOV     AX,09H
            LEA     DX,MES2         ; I/P is invalid HEX number
            INT     21H
EXIT:       MOV     AH,4CH          ; [ Exit
            INT     21H             ;    to DOS ]
            END     MAIN
            END
```

9. Program Statement : Program to convert BCD to HEX.

(Softcopy of this program, P28.asm is available at www.vtubooks.com)

Flowchart :




```
.model  small
.stack  100
.data
    CR      EQU     0AH
    LF      EQU     0DH
    MES_1   DB      CR,LF,'ENTER  4-DIGIT  BCD  NO  $'
```

```
      MES_2   DB      CR,LF,'HEX NO : $'
      MES_3   DB      CR,LF,'INPUT IS INVALID BCD $'
      MULTI   DW      1,10,100,1000
      RESULT  DW(00)
      INP     DB  05
              DB  00
              DB  05  DUP(0)
      BUFF    DB  05
              DB  00
              DB  05  DUP (0)
      CNT1    DW  0
.code
MAIN  :       MOV     AX,@data          ; [ Initialise
              MOV     DS,AX             ;    data segment ]
              MOV     AH,09H
              MOV     DX,OFFSET MES_1   ; Display
              INT     21H
              XOR     DI,DI
              MOV     AH,0AH
              MOV     DX,OFFSET INP     ; I/P number
              INT     21H
              XOR     AX,AX
              XOR     DI,DI
              MOV     CX,0004H
              INC     DI
BACK:         INC     DI
              MOV     AL,INP[DI]        ; Check for validity
              CMP     AL,30H
              JB      TER
              CMP     AL,39H
              JG      TER
              SUB     AL,30H
              MOV     INP[DI],AL
              DEC     CX
              JNZ     BACK
              JMP     NEXT
TER:          MOV     DX,00H
              MOV     AH,09H
              MOV     DX,OFFSET MES_3   ; I/P is invalid BCD number
              INT     21H
              MOV     AH,4CH            ; Terminate
              INT     21H
NEXT:         MOV     DX,0000H
              XOR     DI,DI
```

```
INC     DI
INC     DI
MOV     CH,INP[DI]
XOR     SI,SI
ADD     SI,06H              ; Index for multiplier
MOV     AL,INP[DI]
MOV     AH,00H
MOV     BX,MULTI[SI]
MUL     BX                  ; Multiply by 1000
ADD     RESULT,AX
XOR     SI,SI
ADD     SI,04H
INC     DI
MOV     AL,INP[DI]
MOV     AH,00H
MOV     BX,MULTI[SI]
MUL     BX                  ; Multiply by 100
ADD     RESULT,AX
INC     DI
XOR     SI,SI
ADD     SI,02H
MOV     AL,INP[DI]
MOV     AH,00H
MOV     BX,MULTI[SI]
MUL     BX                  ; Multiply by 10
ADD     RESULT,AX
INC     DI
XOR     SI,SI
ADD     SI,00H
MOV     AL,INP[DI]
MOV     AH,00H
MOV     BX,MULTI[SI]
MUL     BX                  ; Multiply by 1
ADD     RESULT,AX
MOV     AH,09H
MOV     DX,OFFSET MES_2     ; Display string
INT     21H
XOR     SI,SI
MOV     AX,RESULT           ; HEX to ASCII
MOV     BH,AH
MOV     CL,04H
AND     AH,0F0H
SHR     AH,CL
CMP     AH,09H
```

```
            JBE     SKIP1
            ADD     AH,37H
            JMP     SKIP2
SKIP1:      ADD     AH,30H
SKIP2:      MOV     DL,AH
            MOV     AH,02H
            INT     21H
            MOV     AH,BH
            AND     AH,0FH
            CMP     AH,09H
            JBE     SKIP3
            ADD     AH,37H
            JMP     SKIP4
SKIP3:      ADD     AH,30H
SKIP4:      MOV     DL,AH
            MOV     AH,02H          ; Character O/P
            INT     21H
            MOV     AX,RESULT
            MOV     BL,AL
            MOV     CL,04H
            AND     AL,0F0H
            SHR     AL,CL
            CMP     AL,09H
            JBE     SKIP5
            ADD     AL,37H
            JMP     SKIP6
SKIP5:      ADD     AL,30H
SKIP6:      MOV     DL,AL
            MOV     AH,02H
            INT     21H
            MOV     AL,BL
            AND     AL,0FH
            CMP     AL,09H
            JBE     SKIP7
            ADD     AL,37H
            JMP     SKIP8
SKIP7:      ADD     AL,30H
SKIP8:      MOV     DL,AL
            MOV     AH,02H
            INT     21H
            MOV     AH,4CH              ; [ Terminate
            INT     21H                 ;    Exit to DOS ]
            END     MAIN
            END
```

10. Program Statement : Program to read system date

```
.model small
.stack 100
.data
yy dw ?
mm db ?
d  db ?

T_Day DW sun,mon,tue,wed,thu,fri,sat
sun db 'Sunday, $'
mon db 'Monday, $'
tue db 'Tuesday, $'
wed db 'Wednesday, $'
thu db 'Thursday, $'
fri db 'Friday, $'
sat db 'Saturday, $'

T_mon dw jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,de_
jan db 'January, $'
feb db 'February, $'
mar db 'March, $'
apr db 'April, $'
may db 'May, $'
jun db 'June, $'
jul db 'July, $'
aug db 'August, $'
sep db 'September, $'
oct db 'October, $'
nov db 'November, $'
de_ db 'December, $'

.code
DisChar macro char
        push ax       ; save AX
        push dx       ; save DX
        mov dl,char   ' display character
        mov ah,02
        int 21h
        pop dx        ; restore DX
        pop ax        ' restore AX
        endm

start:  mov ax,@data  ; [ Initialise
        mov ds,ax     ;    data segment ]
        call P_date   ; Display code
```

```
        mov ah,4ch      ; [ exit
        int 21h         ;    to DOS ]

P_date  proc near
        mov ah,2ah      ; load function number
        int 21h         ; call function
        mov yy,cx       ; save year
        mov mm,dh       ; save mon
        mov d,dl        ; save day
        mov ah,0        ; get a day of week
        rol ax,1
        mov si, offset T_day; address day table
        add si,ax
        mov dx,[si]     ; address day of week
        mov ah,09h      ; display day of week
        int 21h
        mov al,d ; get day of month
        mov ah,00h
        aam
        or ah,ah        ; convert to BCD
        jz digit0       ; if tens is 0
        add ah,30h      ; convert tens
        DisChar ah      ; display tens
digit0: add al,30h      ; convert units
        DisChar al      ; display unit
        DisChar ' '     ; leave space

        mov al,mm ; get month
        sub al,1
        mov ah,0
        rol ax,1
        mov si, offset T_mon; address month table
        add si,ax
        mov dx,[si]     ; address month
        mov ah,09h      ; display month
        int 21h

        mov ax,yy       ; read year
        cmp ax,2000     ; check for year 2000
        jb dis19        ; if below year 2000
        sub ax, 2000 ; scale for 00-99
        DisChar '2'     ; display 2
        DisChar '0'     ; display 0
        jmp skip
dis19:  sub ax, 1900 ; scale 1900-1999
```

```
        DisChar '1'    ; display 1
        DisChar '9'    ; display 9
skip:   aam            ; convert to BCD
        add ax, 3030h  ; convert to ASCII
        DisChar ah     ; display tens
        DisChar al     ; display unit
        ret
        P_date endp
        end
```

11. Program Statement : Program to set system date

```
.model small
.stack 100
.data
mes     db 10,13,'Enter the date with format : dd:mm:yy   $'
mes1    db 10,13,'Date : $'
buff    db 10
        db 0
        db 10 dup(0)
yy      db ?
mm      db ?
d       db ?
.code
start:  mov ax,@data      ; [ Initialise
        mov ds,ax         ;    data segment ]
        call P_date
        mov ah,4ch        ; [ exit
        int 21h           ;    to DOS ]

P_date  proc near
        mov ah,09h        ; display message
        lea dx,mes
        int 21h
        mov ah,09h        ; display message1
        lea dx,mes1
        int 21h

        mov ah,0ah        ; read time
        lea dx,buff
        int 21h

        mov cl,4          ; set rotation counter
        mov dl,00h        ; clear register
        lea si,buff          ; [ set pointer
        add si,2          ;    to address the time ]
```

```
back:   mov  al,[si]        ; read day
        cmp  al,':'         ; and convert its
        jz   ter            ; value to BCD from ASCII
        rol  dl,cl
        sub  al,30h
        add  dl,al
        inc  si
        jmp  back

ter:    mov  dh,dl          ; Convert BCD to HEX
        and  dl,0f0h
        ror  dl,cl
        mov  al,10
        mul  dl
        and  dh,0fh
        add  al,dh
        mov  d,al

        mov  dl,0           ; clear register
        inc  si             ; increament pointer
back1:  mov  al,[si]        ; read month
        cmp  al,':'         ; and convert its
        jz   ter1           ; value to BCD from ASCII
        rol  dl,cl
        sub  al,30h
        add  dl,al
        inc  si
        jmp  back1

ter1:   mov  dh,dl          ; Convert BCD to HEX
        and  dl,0f0h
        ror  dl,cl
        mov  al,10
        mul  dl
        and  dh,0fh
        add  al,dh
        mov  mm,al

        mov  dl,0           ; clear register
        inc  si             ; increament pointer
back2:  mov  al,[si]        ; read year
        cmp  al,13          ; and convert its
        jz   ter2           ; value to BCD from ASCII
        rol  dl,cl
        sub  al,30h
```

```
        add  dl,al
        inc  si
        jmp  back2

ter2:   mov  dh,dl          ; Convert BCD to HEX
        and  dl,0f0h
        ror  dl,cl
        mov  al,10
        mul  dl
        and  dh,0fh
        add  al,dh
        mov  yy,al

skip2:  mov  ah,2bh         ; function number
        mov  cl,yy
        mov  ch,00
        add  cx,2000        ; load years
        mov  dh,mm          ; load month
        mov  dl,d           ; load day
        int  21h            ; call DOS function
        ret
    P_date  endp
    end
```

12. Program Statement : Program to read system time

```
.model  small
.stack  100
.data
hh  db  ?
mm  db  ?

.code

DisChar  macro  char
         push  ax
         push  dx
         mov  dl,char
         mov  ah,02
         int  21h
         pop  dx
         pop  ax
         endm

start:   mov  ax,@data     ; [ Initialise
         mov  ds,ax        ;     data segment ]
         call  P_time
```

```
          mov  ah,4ch        ;  [ exit
          int  21h           ;     to DOS  ]

P_time    proc near
          mov  ah,2ch        ;  load function number
          int  21h           ;  call function
          mov  hh,ch         ;  save hour
          mov  mm,cl         ;  save minutes
          cmp  ch,12         ;  check for PM or AM
          jb   digit0        ;  check for below 12
          sub  ch,12         ;  adjust to 12 hours
digit0:   mov  al,ch
          mov  ah,00h
          aam                ;  Convert hours
          add  ax,  3030h    ;  Convert hours to ASCII
          DisChar  ah        ;  Display tens
          DisChar  al        ;  Display units
          DisChar  ':'       ;  Display colon
          mov  al,cl
          mov  ah,00h
          aam                ;  Convert minutes
          add  ax,  3030h    ;  Convert minutes to ASCII
          DisChar  ah        ;  Display tens
          DisChar  al        ;  Display units
          DisChar  ' '       ;  leave space
          cmp  hh,12         ;  Check for AM or PM
          jb   am            ;  if AM go to display AM
          DisChar  'P'       ;  display p
          jmp  skip
  am:     DisChar  'A'       ;  display A
  skip:   DisChar  'M'       ;  display M

          ret
       P_time  endp
       end
```

13. Program Statement : Program to set system time

```
.model  small
.stack  100
.data
mes db 10,13,'Enter the time with format : hh:mm followed by AM
or PM $'
mes1    db  10,13,'Time  :  $'
buff    db  10
        db   0
```

```
        db  10 dup(0)
hh  db  ?
mm  db  ?

.code

start:      mov ax,@data  ; [ Initialise
            mov ds,ax     ;    data segment ]
            call P_time
            mov ah,4ch    ; [ exit
            int 21h       ;    to DOS ]

P_time      proc near
            mov ah,09h    ; display message
            lea dx,mes
            int 21h
            mov ah,09h    ; display message1
            lea dx,mes1
            int 21h

            mov ah,0ah    ; read time
            lea dx,buff
            int 21h
            mov cl,4      ; set rotation counter
            mov dl,00h    ; clear register
            lea si,buff   ; [ set pointer
            add si,2      ;    to address the time ]
back:       mov al,[si]   ; read hours
            cmp al,':'    ; and convert its
            jz ter        ; value to BCD from ASCII
            rol dl,cl
            sub al,30h
            add dl,al
            inc si
            jmp back

ter:        mov dh,dl     ; Convert BCD to HEX
            and dl,0f0h
            ror dl,cl
            mov al,10
            mul dl
            and dh,0fh
            add al,dh
            mov hh,al
```

```
            mov dl,0        ; clear register
            inc si          ; increament pointer
back1:      mov al,[si]     ; read minutes
            cmp al,' '      ; and convert its
            jz ter1         ; value to BCD from ASCII
            rol dl,cl
            sub al,30h
            add dl,al
            inc si
            jmp back1

ter1:       mov dh,dl       ; Convert BCD to HEX
            and dl,0f0h
            ror dl,cl
            mov al,10
            mul dl
            and dh,0fh
            add al,dh
            mov mm,al

            inc si          ; increament pointer
            mov ch,[si]     ; read next character
            cmp ch,'P'      ; check whether it is a PM
            jnz skip2
            add hh,0ch      ; if PM add 12 in hours
skip2:      mov ah,2dh      ; function number
            mov ch,hh       ; load hours
            mov cl,mm       ; load minutes
            mov dx,0000h    ; load seconds and hundredths of second
            int 21h
            ret
      P_time endp
      end
```

■ Interfacing Experiments

1. **Program Statement :** Interface 4 × 4 matrix keyboard to 8086 microprocessor using 8255 : Refer page no. 10-6.

2. **Program Statement :** Interface an 8-digit 7 segment LED display using 8255 to the 8086 microprocessor system and write an 8086 assembly language routine to display message on the display. : Refer page no. 10-14.

3. **Program Statement :** Interface stepper motor to the 8086 microprocessor system and write an 8086 assembly language program to control the stepper motor. : Refer Page No. 9-33.

4. **Program Statement - Real Time Clock :** Generate a real time clock by generating a periodic interrupt request signal on the $\overline{NMI}$ input of 8086. : Refer Page No. 6-26.

5. **Program Statement :** Design a pre-settable alarm system using 8253/54 timer. Use thumbwheel switches to accept 4 digit value in seconds. Alarm should last for 5 seconds. Do not use interrupt.

Fig. 2 shows the 8086 microprocessor based pre-settable alarm system. Thumbwheel switches are interfaced through 8255 ports. Timing parameters are derived from the 8253/54. 74LS138 decoder is used to generate chip select signals for 8253/54 and 8255. One more 74LS138 decoder is used to generate $\overline{IOR}$, $\overline{IOW}$, $\overline{MEMR}$, and $\overline{MEMW}$ signals.

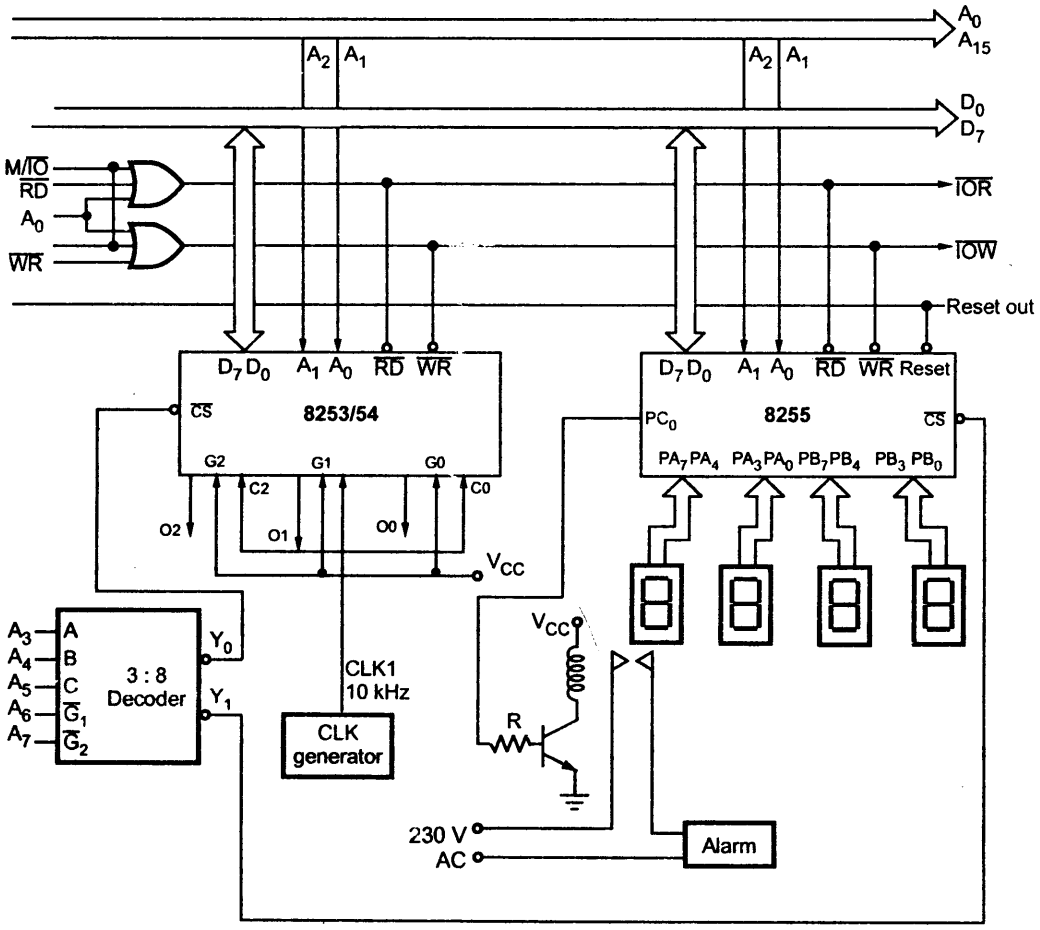


Fig. 2 Pre-settable alarm system

Counter 0 of 8253/54 is programmed in mode 0 to give the pre-settable time period and counter 2 is programmed in mode 0 to give delay of 5 seconds. As we know, clock input for 8253/54 is 10 kHz count required to get 1 second time interval is

$$\text{Count} = \frac{\text{Required period}}{\text{Clock period}} = \frac{1 \text{ sec}}{100\ \mu\text{s}} = 10000 = 1 \times 10^4 = 271\text{H}$$

This count value is loaded in the count register of the counter 1 and counter 1 is programmed in mode 2 to generate square wave with frequency 1 Hz. The output of counter 1 is fed to the clock input of counter 0 and counter 2.

To read four digit of count, we need four thumbwheels. One thumbwheel switch can be interfaced using four input lines. So to interface four thumbwheels we need 16 lines. The IC 8255 is used to interface these thumbwheel switches. Two thumbewheel switches are connected to port A and other two are connected to port B.

Address map :

Ports / Control register	Address lines								Address
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Counter 0	0	0	0	0	0	0	0	0	00H
Counter 1	0	0	0	0	0	0	1	0	02H
Counter 2	0	0	0	0	0	1	0	0	04H
Control register	0	0	0	0	0	1	1	0	06H
Port A	0	0	0	0	1	0	0	0	08H
Port B	0	0	0	0	1	0	1	0	0AH
Port C	0	0	0	0	1	1	0	0	0CH
Control register	0	0	0	0	1	1	1	0	0EH

Control Word to Read/Write value in count register of counter 0 of 8253/54

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
SC_1	SC_2	RW_1	RW_0	M_2	M_1	M_0	BCD	
0	0	1	1	0	0	0	1	= 31 H

D_0 = 1 BCD Count

D_1, D_2, D_3 = 000 Mode : Square wave rate generator

D_4, D_5 = 11 Read/ Write lower byte first and then higher byte.

D_6, D_7 = 00 Counter 0

Control Word to Read/Write value in count register of counter 1 of 8253/54

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
SC_1	SC_2	RW_1	RW_0	M_2	M_1	M_0	BCD	
0	1	1	1	0	0	0	1	= 71H

$D_0 = 1$ BCD Count

$D_1, D_2, D_3 = 000$ Mode : Square wave rate generator

$D_4, D_5 = 11$ Read/Write lower byte first and then higher byte.

$D_6, D_7 = 01$ Counter 1

Control Word to latch value in count register of counter 0 of 8253/54

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
SC_1	SC_2	RW_1	RW_0	M_2	M_1	M_0	BCD	
0	0	0	0	0	0	0	1	= 01H

$D_0 = 1$ BCD Count

$D_1, D_2, D_3 = 000$ Mode : Square wave rate generator

$D_4, D_5 = 00$ Read/ Write lower byte first and then higher byte.

$D_6, D_7 = 00$ Counter 0

Control Word to latch value in count register of counter 1 of 8253/54

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
SC_1	SC_2	RW_1	RW_0	M_2	M_1	M_0	BCD	
0	1	0	0	0	0	0	1	= 41H

$D_0 = 1$ BCD Count

$D_1, D_2, D_3 = 000$ Mode : Square wave rate generator

$D_4, D_5 = 00$ Read/ Write lower byte first and then higher byte.

$D_6, D_7 = 01$ Counter 1

Control Word to initialise 8255 : Port A = I/P, Port B =I/P, Port C = O/P

BSR/IO	MODE A		PA	PC_H	MODE B	PB	PC_L	
1	0	0	1	0	0	1	0	= 92H

Program :

```
        MOV  AL,92H       ; Load control word in accumulator
        OUT  0EH,AL       ; Initialise 8255 by sending control
                          ; word at the
                          ; address of control register
START:  IN   AL,08H       ; Get the lower two digit of the count
        MOV  BL,AL        ; Store the lower  two digit of
                          ; the count
        IN   AL,0AH       ; Get the higher two digit of the count
        MOV  BH,AL        ; Store the higher two digit of
                          ; the count
        MOV  AL,31H
        OUT  06,AL        ; Loads control word (31H) in the control
                          ; register to
                          ; load 16-bit count in the count register
                          ; of counter 0
        MOV  AL,BL
        OUT  00,AL        ; Loads lower byte of the count.
        MOV  AL,BH
        OUT  00,AL        ; Loads higher byte of the count.
BACK:   MOV  AL,01H
        OUT  06,AL        ; Loads control word (01H) in the control
                          ; register to
                          ; latch 16-bit count in the count
                          ; register of counter 0
        IN   AL,00H       ; Get the lower two digits of the count
        CMP  AL,00H       ; Compare with zero
        JNZ  BACK         ; Repeat
        IN   AL,00H       ; Get the higher two digits of the count
        CMP  AL,00H       ; Compare with zero
        JNZ  BACK         ; Repeat
        MOV  AL,01H       ; Load bit pattern to run Alarm
        OUT  0CH,AL       ; Send it to Port C
        CALL DELAY        ; Wait for 5 Seconds
        MOV  AL,00H       ; Load bit pattern to stop Alarm
        OUT  0CH,AL       ; Send it to Port C
        JMP  START        ; Repeat
```

Delay routine :

This delay routine gives a delay of 5 seconds. Counter1 of 8253/54 is used to give delay of 1 second. As output of counter1 is used as a clock for counter2, the count in the counter2 acts as a multiplying factor. Therefore, by loading 05H in the count register of counter2 we get a delay of 5 (5 × 1) seconds.

```
        MOV  AL,71H
```

```
        OUT   06,AL         ; Loads control word (71H)in the
                            ; control register.
        MOV   AL,10H
        OUT   02,AL         ; Loads lower byte of the count.
        MOV   AL,27H        ; Loads higher byte of the count.
        OUT   02H
        MOV   AL,B1H
        OUT   06H,AL
        MOV   AL,05H
        OUT   04,AL         ; Loads lower byte of the count.
        MOV   AL,00H        ; Loads higher byte of the count.
        OUT   04,AL
BACK:   MOV   AL,81H
        OUT   06,AL         ; Loads control word (81H) in the
                            ; control register
                            ; to latch 16-bit count in the count
                            ; register of
                            ; counter 2
        IN    AL,04H        ; Get the lower two digits of the count
        CMP   AL,00         ; Compare with 00H
        JNZ   BACK          ; If not zero, Repeat
        IN    AL,04H        ; Get the higher two digits of the count
        CMP   AL,00         ; Compare with 00H
        JNZ   BACK          ; If not zero, Repeat
        RET                 ; Return to main program
```

6. Program Statement : DC Motor Speed and Direction Control

Let us see the application of 8254 timer as a DC motor speed controller. Fig. 3 shows the schematic diagram of the DC motor and its associated driver circuitry. As shown in the Fig . 3 driver circuitry consists of IC 8254, a flip-flop, buffers and transistors. (See Fig. 3 on next page)

When flip-flop outputs are : Q = 0 and $\overline{Q}$ = 1, inverters 1 and 3 give output logic 0 and inverters 2 and 4 give output logic 1. As a result, transistors Q_1 and Q_4 turn ON and transistors Q_2 and Q_3 turn OFF. Due to this V_{CC}, supply is applied to the positive lead of the DC motor and ground is applied to the negative lead of the DC motor. This connection causes DC motor to rotate in the forward direction.

When flip-flop outputs are : Q = 1 and $\overline{Q}$ = 0, the conditions of transistors are reversed and DC motor rotate in the reverse direction. Thus the direction of motor can be controlled by changing state of the flip-flop.

If the duty cycle of the flip-flop output is kept 50%, then motor current is positive and negative for equal amount of time. If flip-flop output is varied with enough frequency then due to inertia of motor, motor will not rotate at all. But it will exhibit some holding