1. a) Binary search
   b) Logic controller : FF → even parity, 00 → odd parity

2. a) Read & display macro to read & display a string
   b) Logic controller : BCD up-down counter

3. a) Bubble sort (CV mode)
   b) Logic controller : Multiply 2, 8-bit no's

4. a) Store 2 strings from i/p & check if they are equal
   b) 7-Segment display : Fire & Help

5. a) Palindrome check
   b) 7-Segment display : Rolling 12-char msg

6. a) What is your name ?
   b) 8×3 keypad : Row & col of key pressed

7. a) nCr using recursion (CV mode)
   b) Stepper motor interface : CW or CCW by N steps

8. a) System time display
   b) CRO : Sine wave using DAC interface

9. a) Create or delete file
   b) CRO : Half rectified sine wave using DAC interface

10. a) Move cursor to specified location
    b) CRO : Fully rectified sine wave using DAC interface

Functional values (for Interrupts)

For the function call of type

```
mov ah, <val>
int 21h
```

ah value	Meaning
1	Read a char from keyboard store in AL
2	Write a char stored in DL to display
4ch	Exit / Terminate program
9	Write a string pointed by DX to display

10/0ah	Read a string from keyboard into memory location pointed by DX
	The memory location (say STR) must be defined as
	`STR db 50`
	`db 0`
	`db 50 dup(0)`

2ch	Get system time & store it in CH, CL, DH (Hrs, Mins, Secs)
8	Getch function to allow user to input without displaying on the screen (echo)
3ch	Creates a file whose filename is pointed by DX (CX value must be set to 0 before using this function)
41h	Deletes a file whose filename is pointed by DX

For other interrupts

`mov ah, 2` `int 10h`	Moves the cursor to row, col stored in DH, DL

`mov ah, 6`
`mov al, 0`
`mov bh, 7`
`mov ch, 0`
`mov cl, 0`
`mov dh, 24`
`mov dl, 79`
`int 10h`
	AX = 0600H, BH = 7
	CX = 0000H, DX = 2479H
	Clears the screen

`mov ah, 1` `int 16h`	Checks if key is pressed If yes, carry flag C = 1 if no, zero flag Z = 1

1.a) Binary Search

```
.model small
printf macro msg
        mov ah, 9
        lea dx, msg
        int 21h
        endm
exit  macro
        mov ah, 4ch
        int 21h
        endm
.data
    a   dw   1111h, 2222h, 3333h
    n   dw   ($-a)/2
    key dw   3333h
    low_  dw  ?
    mid_  dw  ?
    high_ dw  ?
    msg1  db  'Successful search $'
    msg2  db  'Unsuccessful search $'
.code
    mov ax, @data
    mov ds, ax
    mov low_, 0
    mov ax, n
    dec ax
    mov high_, ax
l1:
    mov si, low_
    cmp si, high_
    jg l4
    add si, high_
    shr si, 1
```

(low = 0, high = n-1)

(if low > high, goto l4)

(mid = (low + high) / 2)

```
mov mid-, si
mov ax, key
shl si, 1          (a is a dw array)
cmp ax, a[si]      (if key != a[mid], goto L2)
jne L2
printf msg1        (if equal, display successful)
exit
L2: cmp ax, a[si]  (if key > a[mid], goto L3)
    jg L3
    mov ax, mid-   (if key < a[mid],
    dec ax          high = mid - 1
    mov high-, ax   goto L1)
    jmp L1
L3: mov ax, mid-   (else
    inc ax          low = mid + 1
    mov low-, ax    goto L1)
    jmp L1
L4: printf msg2    (display unsuccessful)
    exit
    end
```

1.6) Logic controller Parity check (FF-even 00-odd)

```
.model small
.code
    mov dx, 0e403h        (control reg.
    mov al, 82h            B -> i/p , A -> o/p)
    out dx, al
    mov dx, 0e401h        (Port B
    in al, dx              Take i/p)
    mov cx, 8             (8 bits to process)
    mov ah, 0             (count of 1's)
n2: ror al, 1             (if rightmost bit is 1,
    jc n1                  goto n1)
    dec cl                (one more bit processed)
    jnz n2                (repeat n2 till all bits processed)
    jmp nxt               (after processing)
n1: inc ah                (count++)
    dec cl                (processed one more bit)
    jnz n2
nxt: mov al, ah           (move count to al)
    and al, 1             (A.1 = 0 if even parity
    cmp al, 0              display FF)
    jz dff
    mov al, 00h           (else display 00)
    jmp next
dff: mov al, 0ffh         (disp FF)
next: mov dx, 0e400h      (output to port A)
    out dx, al
    mov si, 4fffh         (delay loop)
ret2: mov di, 0ffffh
ret1: dec di
    jnz ret1
    dec si
    jnz ret2
```

```
mov al, ah          ( display the count
mov dx, 0e400h        of 1's in i/p, To port A o/p)
out dx, al
mov ah, 4ch         (exit)
int 21h
end
```

2.a) Read & Display macros for string

file: rcd.mac

```
read macro
    mov ah, 1
    int 21h
    endm
```

file: disp.mac

```
display macro
    mov ah, 2
    int 21h
    endm
```

file: macros.asm

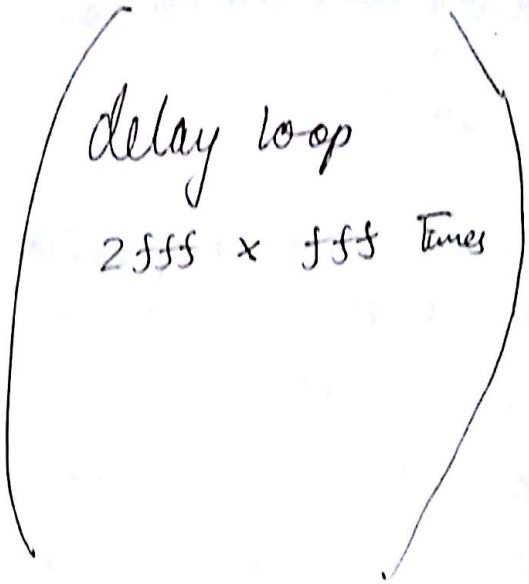
```
include c:\masm\rcd.mac
include c:\masm\disp.mac
.model small
.data
    str db 10 dup(?)        (string of 10 chars)
.code
    mov ax, @data
    mov ds, ax
    mov si, 0               (count of chars = 0)
rd1: read                   (read a char
    cmp al, 0dh              compare with [enter] key)
    je finish               (if enter, goto finish)
    mov str[si], al         (copy read char to str)
    inc si                  (increment str pointer)
    jmp rd1                 (repeat)
finish: mov cx, si          (move count to cx)
    mov si, 0               (reset si)
    mov dl, 10              (display linefeed char \n)
    display
disp1: mov dl, str[si]      (copy string char to dl)
    display                 (display dl)
    inc si                  (increment str pointer)
    loop disp1              (repeat until cx = 0)
    mov ah, 4ch
    int 21h                 (exit)
    end
```

2.6) Logic Controller : BCD up-down counter

```
.model small
.code
    mov dx, 0e403h   (control reg)
    mov al, 80h      (all o/p ports)
    out dx, al
    mov al, 0        (counter = 0)
    mov dx, 0e400h   (port A as o/p)
up: out dx, al       (display the no.)
    call delay
    add al, 1        (counter++)
    daa              (convert hexadecimal result to decimal)
    call stop        (check if key pressed)
    cmp al, 99h      (check if counter reached 99)
    jne up
down: out dx, al     (display counter)
    call delay
    add al, 99h
    daa
    call stop
    cmp al, 99h      (if counter reached 99 again)
    jne down
stop: push ax        (since al is used in pgm)
    mov ah, 1
    int 16h          (check if key pressed)
    jnz exit
    pop ax           (restore al)
    ret
exit: mov ah, 4ch
    int 21h          (exit)
```

```
 99      98              00
+99     +99     . . .   +99
---     ---             ---
[1]98   [1]97           [0]99
 c al    c al            al
```

```
delay: mov si, ffffh
ret 2: mov di, offfh
ret 1: dec di
       jnz ret 1
       dec si
       jnz ret 2
       ret
       end
```



3.a) Bubble Sort (CV mode)

- model small
- data

```
array db 85h, 95h, 25h, 45h, 55h, 15h, 65h, 45h
len dw ($ - array)
```

- code

```
mov ax, @data
mov ds, ax
mov bx, len
dec bx                    ( bx = n-1 )
np: mov cx, bx            ( cx = bx, si = 0 )   (next pass)
    mov si, 0
ni: mov al, array[si]     ( al = a[si]          (next iteration)
    inc si                  if al <= a[si+1]
    cmp al, array[si]       goto next )
    jbe next
    xchg al, array[si]    ( else exchange al, a[si]
    mov array[si-1], al     a[si-1] = al )
next: loop ni             (loop ni till cx = 0)
    dec bx                ( bx--
    jnz np                  if (bx != 0) goto np )
    mov ah, 4ch           (exit)
    int 21h
end
```

3.b) Logic Controller : multiply 2. 8-bit no's

- model small
- data

```
portA  dw  0e400h
portB  dw  0e401h
portC  dw  0e402h
ctlw   dw  0e403h
```

- code

```
mov ax, @data
mov ds, ax
mov al, 82h        (portB → i/p, portA → o/p)
mov dx, ctlw
out dx, al

mov dx, portB
in al, dx          (i/p 1st no. & store in bl)
mov bl, al
call delay         (after delay, i/p 2nd no. into al)
in al, dx

mul bl             (ax ← al × bl)

mov dx, portA
out dx, al         (output lower byte of product)
call delay
mov al, ah         (after delay, o/p higher byte of product)
out dx, al
mov ah, 4ch        (exit)
int 21h
```

```
delay proc          (delay procedure
    push ax          Save value of ax & cx
    push cx
    mov ax, 6fffh
agn1: mov cx, 0ffffh   (6fff x ffff times
agn:  loop agn          loop
    dec ax
    jnz agn1
    pop cx          (restore values
    pop ax           of cx & ax
    ret
delay endp          (end procedure)
    end
```

4a) Check if strings are equal & disp their length

```
.model small
.data
    str1 db 100
         db 0
         db 100 dup(0)
    str2 db 100
         db 0
         db 100 dup(0)
    m1   db 'Length of str1: '
    l1   db ?, 10, 13, '$'
    m2   db 'Length of str2: '
    l2   db ?, 10, 10, '$'
    m4   db 'Not equal $'
    m3   db 'Equal $'
    m5   db 'Enter first string: $'
    m6   db [illegible] 10, 13, 'Enter second string: $
.code
    mov ax, @data
    mov ds, ax
    mov es, ax

    lea dx, m5
    mov ah, 9
    int 21h          } print m5

    lea dx, str1
    mov ah, 0ah
    int 21h          } read str1
```

(format: size, length, string)

(format used for ah = 0ah int 21h to input string)

```
lea dx, m6          (print m6)
mov ah, 9
int 21h

lea dx, str2        (read str2)
mov ah, 0ah
int 21h

mov cl, str1+1      (length of strings into cl & bl)
mov bl, str2+1

add cl, 30h         (convert to ascii)
add bl, 30h

mov l1, cl          (move to l1 & l2)
mov l2, bl

sub cl, 30h         (convert back to hexadecimal)
sub bl, 30h

lea dx, m1          (display length of str1)
mov ah, 9
int 21h
lea dx, m2          (display length of str2)
int 21h
cmp cl, bl          (if lengths are equal, goto next)
je next

dm4: lea dx, m4     (display not equal)
     mov ah, 9
     int 21h
     jmp exit
```

```
next:  lea si, str1+2          (load strings into si & di
       lea di, str2+2
       cld                     (clear direction flag)
       repe cmpsb              (compare str1 & str2
       jnz dn4                  bytewise till cx=0
                                if unequal, goto dn4)
equal: lea dx, m3
       mov ah, 9               (display equal)
       int 21h
exit : mov ah, 4ch             (exit)
       int 21h
       end
```

46) 7seg. disp. : Fire & Help

```
• model small
• data
    fire db 86h, 0afh, 0cfh, 8eh      (7 seg code for e, r, i, f)
    help db 8ch, 0c7h, 86h, 89h       (& p, l, e, h)
• code
    mov ax, @data
    mov ds, ax
    mov dx, 0e403h
    mov al, 80h                       (all o/p ports)
    out dx, al
bak: lea si, fire                     (display 'fire' & delay for some time)
    call display
    call delay
    lea si, help                      (display 'help' >> )
    call display
    call delay
    mov ah, 1                         (check if any key is pressed,
    int 16h                            if not, repeat from bak)
    jz bak
    mov ah, 4ch                       (exit)
    int 21h
display: mov ch, 04                   (4 characters to display)
bak2: mov bl, 08                      (8 segments to light up)
    mov al, [si]                      (load the character into al)
next: rol al, 01                      (rotate left)
    mov dx, 0e401h                    (output the leftmost bit)
    out dx, al
```

```
        push ax          (save value of al)
        mov dx, 0e402h   (port C for
        mov al, 0ffh      o/p one
        out dx, al        clock pulse
        mov al, 00h       & displaying the segment
        out dx, al
        pop ax           (restore value of al)
        dec bl           (go to next segment)
        jnz next
        inc si           (next character)
        loop back2
        ret
delay : mov si, 2ffffh
rep2 : mov di, 6ffffh
rep1 : dec di
       jnz rep1
       dec si
       jnz rep2
       ret
       end
```

(delay
2ffff * 6ffff
times)

5a) Palindrome check

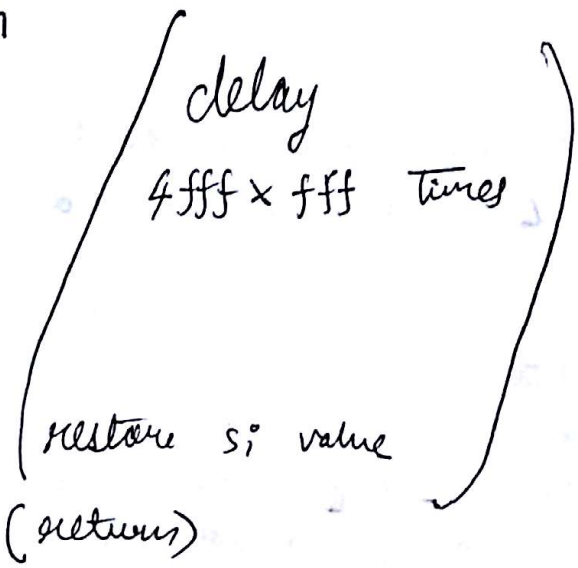
```
.model small
.stack 100
.data
    str  db  'malayalam'
     n   db  $-str
    rstr db  10 dup(0)
    msg1 db  'Palindrome $'
    msg2 db  'Not a palindrome $'
.code
    mov ax, @data
    mov ds, ax
    mov es, ax
    mov cl, n          (cl = n-1)
    dec cl
    mov di, cx         (di = n-1)
    inc cx             (cx = n)
bak: mov ah, str[di]   (ah = str[di])
    mov rstr[si], ah   (rstr[si] = ah)
    inc si             (si++
    dec di              di--)
    loop bak           (repeat till cx = 0)
    lea si, str        (si → str
    lea di, rstr        di → rstr)
    cld
    mov cl, n          (while cx != 0,
    repe cmpsb          cmp string bytewise)
    je dmsg1           (if equal disp palindrome)
    lea dx, msg2       (else disp not palindrome)
    jmp xit
```

```
dmsg1: lea dx, msg1      (disp palindrome)
xit : mov ah, 9
      int 21h            (disp the string at dx)
      mov ah, 4ch
      int 21h            (exit)
      end
```

56) 7-seg. display : Dayanandacool

```
.model small
.data
  list db  0c7h, 0a3h, 0a7h, 88h, 0a1h, 0abh, 89h, 0abh, 83h, 99h, 83h, 0a1h
  ;        d     o     c     A     d     n     A     n     A     y     A     d
.code
    mov ax,@data
    mov ds,ax
    mov al, 80h          (CW = 80h
    mov dx, 0e403h        all ports o/p)
    out dx,al
rpt: mov cx,12           (12 chars to repeatedly display)
     lea si, list        (si → list)
nc: mov al,[si]          (al = list[si])
     call disp           (displays 1 char.
     call delay           delays
     call stop            check for key press)
     inc si              (si++)
     loop nc             (loops till cx=0, all chars displayed)
     jmp rpt             (repeats cx=12)

stop: mov ah, 1
      int 16h            (check for key press)
      jnz exit
      ret
exit: mov ah, 4ch        (exit)
      int 21h

disp: mov bl, 8          (8 segments)
nbit: rol al, 1          (for each segment, extract leftmost bit)
      mov dx, 0e401h
      out dx, al         (output sent to port B)
      push ax            (save al value)
      mov dx, 0e402h
      mov al, 0ffh
      out dx, al          (send a clock pulse to
      mov al, 00h          port C to display
      out dx, al           current segment)
      pop ax             (restore al value)
      dec bl             (1 bit/segment processed)
      jnz nbit           (repeat till bl=0)
      ret
```

```
delay : push si                    (save si value)
        mov si, 4fffh
ret 2 : mov di, 0fffh
ret 1 : dec di
        jnz ret 1
        dec si
        jnz ret 2
        pop si                     (restore si value)
        ret                        (returns)
        end
```



6a) What is your name?

```
.model small
.data
    msg db 'What is your name?'
    nam db 50 dup(0)
.code
    mov ax,@data
    mov ds, ax
    [redacted]
    mov si, 0
    mov ah, 1
bak: int 21h
    mov nam[si], al
    inc si
    cmp al, 13
    jne bak
    mov byte ptr nam[si], '$'
    call clr
    call setc
    lea dx,msg
    mov ah, 9
    int 21h
    mov ah, 4ch
    int 21h
clr: mov ax, 0600h
    mov bh, 7
    mov cx, 0
    mov dx, 2479h
    int 10h
    ret
```

si = 0
ip char to al
nam[si] = al
si++
if al != 13
repeat
nam[si] = '$'

(clears screen
sets cursor)

(prints 'what is your name?'
followed by name)

ah = 6 , al = 0
bh = 7
ch = 0 , cl = 0
dh = 24 , dl = 79
int 10h

(clears the screen)

```
sete:  mov  ah, 2
       mov  bh, 0
       mov  dh, 12
       mov  dl, 20
       int  10h
       ret
       end
```

ah = 2
bh = 0
dh, dl = row, col
int 10h moves the cursor to row, col

6 b) keypad

```
.model small
.data
    keys db '0123456789abcdefghijklmn'
    keys db ?
    msg1 db 10,13,'The row no. is: '
    row db ?,13,10,'$'
    msg2 db 'The col. no. is: '
    col db ?,'$'
.code
    mov ax,@data
    mov ds,ax
    mov dx,0e403h
    mov al,90h
    out dx,al          (port A is i/p)
rep1: mov dx,0e402h
    mov al,01h
    out dx,al          (enable first row (001) from port C, 01h)
    mov dx,0e400h
    in al,dx
    cmp al,00
    jnz fr             (take 8 bit i/p from row 1, if non zero, go to fr)
    mov dx,0e402h
    mov al,02h
    out dx,al          (enable 2nd row (010) from port C, 02h)
    mov dx,0e400h
    in al,dx
    cmp al,00
    jnz sr             (8-bit i/p from row 2, if non zero go to sr)
    mov dx,0e402h
    mov al,04h
    out dx,al          (enable 3rd row (100) from port C, 04h)
```

```
mov dx, 0e400h
in al, dx
cmp al, 00
jnz fr
jmp rep1
```
(take 8-bit i/p from row 3, if non zero goto fr)

```
fr: call delay
    lea si, keys
    mov row, 31h
    mov cl, 30h
next1: inc cl
    shr al, 1
    jc disp
    inc si
    jmp next1
```
(delay)
(si → keys) (first row)
(row = ascii 1)
(cl tracks col)
(extract rightmost bit of al (i/p) & check if it is 1 (key pressed) (if yes, goto disp) (else si++, repeat)

```
sr: call delay
    lea si, keys
    add si, 8
    mov row, 32h
    mov cl, 30h
next2: inc cl
    shr al, 1
    jc disp
    inc si
    jmp next2
```
(si → keys + 8) (second row)
(row = ascii 2)
(Same as next1)

```
tr: call delay
    lea si, keys
    add si, 10h
    mov row, 33h
    mov cl, 30h
next3: inc cl
    shr al, 1
    jc disp
    inc si
    jmp next3
```
(si → keys + 16) (third row)
(row = ascii 3)
(Same as next1 & next2)

```
disp: mov dl, [si]
      mov ah, 2
      int 21h
```
(display char stored in dl
i.e. list [si])

```
      lea dx, msg1
      mov ah, 9
      int 21h
```
(display row)

```
      mov col, dl
```
(col = dl)

```
      lea dx, msg2
      int 21h
```
(display col)

```
      jmp exit

delay: mov si, 2fffh
bak2: mov di, 0ffffh
bak1: dec di
      jnz bak1
      dec si
      jnz bak2
      ret

exit: mov ah, 4ch
      int 21h
      end
```
(delay loop
2fff x ffff times)

(exit)

7a) nCr using recursion $(nCr = n-1C_r + n-1C_{r-1})$

(cv mode.)

```
.model small
.data
    n   dw 5
    r   dw 3
    ncr dw 0
.code
    mov ax, @data
    mov ds, ax
    mov ax, n          (ax = n)
    mov bx, r          (bx = r)
    call ncrpro
    mov ah, 4ch
    int 21h
ncrpro: cmp bx, ax     (nCn = 1)
        je res1
        cmp bx, 0      (nC0 = 1)
        je res1
        cmp bx, 1      (nC1 = n)
        je resn
        dec ax         (n--)
        cmp bx, ax     (n == r ? goto incr)
        je incr
        push ax        (save n, r
        push bx
        call ncrpro    n-1Cr
        pop bx         restore n, r)
        pop ax
        dec bx         (r--)
        push ax        (save n, r
        push bx
        call ncrpro    n-1Cr-1
        pop bx         restore n, r)
        pop ax
        ret
```

```
res1: inc ncr
      ret
incr: inc ncr
resn: add ncr, ax
      ret
      end
```

(ncr += 1)

(ncr += 1)

(ncr += n)

7b) Stepper motor (n steps CW or CCW)

```
.model small
.code
    mov dx, 0e403h      (all o/p ports)
    mov al, 80h
    out dx, al
    mov cx, 20          (n = 20 steps)
    mov dx, 0e400h      (port A → o/p)
    mov al, 88h         (10001000) (excitation bits)
                         a   a'
repl: out dx, al        (move the motor)
    call delay          (delay)
    rol al, 1           (rol for CW, ror for CCW)
    loop repl           (Till cx = 0)
    mov ah, 4ch
    int 21h             (exit)
delay: mov si, 2ffffh   (delay loop
bak2: mov di, 0ffffh     2fff x ffff times
bak1: dec di
    jnz bak1
    dec si
    jnz bak2
    ret
end
```

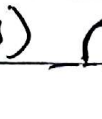
8.a) System Time

```
.model small
.code
    mov ah, 2ch        (gets system time & stores in ch, cl, dh = hr, min, sec)
    int 21h
    mov al, ch         (display ch = hrs)
    call disp
    mov dl, ':'
    mov ah, 2          (display ':')
    int 21h
    mov al, cl         (display cl = mins)
    call disp
    mov dl, ':'
    mov ah, 2          (display ':')
    int 21h
    mov al, dh         (display dh = secs)
    call disp
    [redacted]
    mov ah, 4ch        ([redacted] exit)
    int 21h
display proc near
    aam                (converts hex in al to unpacked bcd in ax)
    add ax, 3030h      (converts unpacked bcd in ax to ascii)
    mov bx, ax         (store ax in bx)
    mov dl, bh
    mov ah, 2          (display upper byte)
    int 21h
    mov dl, bl
    int 21h            (display lower byte)
    ret                (returns)
disp endp              (end procedure)
    end
```

8.b) Sine wave using dac interface

(127 sin θ, θ = 0 to 90, step 10)

```
.model small
.data
    a db 00,22,43,63,81,97,109,119,125,127
.code
    mov ax, @data
    mov ds, ax
    mov al, 80h          (CW = 80h, all are o/p ports)
    mov dx, 0e403h
    out dx, al
start: mov si, 0ffffh    (no. of sine waves)
    mov bx, 0            (index for a[])
    mov dx, 0e401h       (Port B for o/p)

b1: mov al, a[bx]
    add al, 127          (These values are above x-axis)
    out dx, al
    inc bx               (bx++)
    cmp bx, 9            (if bx < 9, goto b1)
    jb b1                (jump if below)

b2: mov al, a[bx]
    add al, 127          (Values above x-axis)
    out dx, al
    dec bx               (bx--)
    cmp bx, 0            (if bx != 0, goto b2)
    jnz b2
```



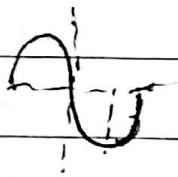
```
b3: mov al, a[bx]
    mov cl, 127
    sub cl, al
    mov al, cl
    out dx, al
    inc bx
    cmp bx, 9
    jb b3
```

(127 - a[bx] — These values are below x-axis)

(bx++) if bx < 9, goto b3

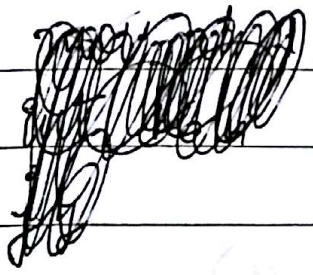
```
b4: mov al, a[bx]
    mov cl, 127
    sub cl, al
    mov al, cl
    out dx, al
    dec bx
    jnz b4

    dec si
    jnz b1
```

(127 - a[bx] — These values are below x-axis)



(bx--) if bx != 0, goto b4

(still 0ffffh no. of sine waves are plotted)



```
mov ah, 4ch
int 21h
end
```

(end)

9.a) Create or delete file

```
.model small
.data
    m1 db 10,13,"Enter file name to create : $"
    m2 db 10,13,"Enter file name to delete : $"
    m3 db "1. Create",10,13,"0. delete $"
    m4 db 10,13,"Enter choice : $"
    m5 db "Error $"
    f_name db 80 dup(0)

.code
    mov ax,@data
    mov ds,ax
    lea dx,m3
    mov ah,9          (print menu)
    int 21h
    lea dx,m4
    int 21h           (print prompt)
    mov ah,1
    int 21h           (take character i/p)
    cmp al,'1'
    je c_file         (if i/p is 1, goto c_file)
    cmp al,'0'
    je d_file         (if i/p is 0, goto d_file)
error: lea dx,m5
    mov ah,9          (else, display error)
    int 21h
    mov ah,4ch
    int 21h           (exit)
```

```
c_file: lea dx, m1          (display prompt)
        mov ah, 9
        int 21h
        call read           (read f_name)
        mov cx, 0           (cx=0, ah=3ch
        mov ah, 3ch          dx → filename
        lea dx, f_name       creates a file)
        int 21h
        jc error            (if error, c=1)
        mov ah, 4ch         (exit)
        int 21h
d_file: lea dx, m2          (display prompt)
        mov ah, 9
        int 21h
        call read           (read f_name)
        lea dx, f_name      (ah=41h, dx → filename
        mov ah, 41h          deletes a file)
        int 21h
        jc error            (if error, c=1)
        mov ah, 4ch         (exit)
        int 21h

read:   mov ah, 1           (ah=1, read a char)
        lea si, f_name      (si → f_name)
back:   int 21h             (read 1 char into al)
        cmp al, 0dh         (if char is 13 [enter])
        jz done             (goto done)
        mov [si], al        (else, move the char to f_name[si])
        inc si              (si++)
        jmp back            (repeat)
done:   ret                 (return)
        end
```

9.b) Half rectified sine wave using DAC

```
.model small
.data
    a dw 00,22,43,63,81,97,109,119,125,127
.code
    mov ax, @data
    mov ds, ax
    mov dx, 0e403h
    mov al, 80h          (CW = 80h, all o/p ports)
    out dx, al
    mov dx, 0e400h       (port A → o/p)
    mov cx, 0ffffh       (loops ffff times)
    mov bx, 0            (index for a[])

b1: mov al, a[bx]
    add al, 127
    out dx, al
    inc bx
    cmp bx, 9
    jb b1

b2: mov al, a[bx]
    add al, 127
    out dx, al
    dec bx
    jnz b2
```

```
        mov si, 14
rpt:    mov al, 127
        out dx, al
        dec si
        jnz rpt

        loop b1
        mov ah, 4ch
        int 21h
        end
```

(for drawing straight line b/c waves)

(loops ffff times)

(exit)

10.a) Read i/p coordinates in bcd & move cursor to that location

```
.model small
.data
    m1 db 10,13,"Enter row no: $"
    m2 db 10,13,"Enter col. no: $"
    m3 db 10,13,"Press any key to stop $"
    row db ?
    col db ?
.code
    mov ax,@data
    mov ds,ax
    lea dx,m1          (Prompt for row)
    mov ah,9
    int 21h
    call read          (read row into al
    mov row,al          store in row)
    lea dx,m2          (Prompt for col)
    mov ah,9
    int 21h
    call read          (read col into al
    mov col,al          store in col)
    lea dx,m3          (display m3)
    mov ah,9
    int 21h
    mov ah,2           (ah = 2,
    mov dh,row          dh,dl = row,col
    mov dl,col         int 10h moves cursor to row,col)
    int 10h
    mov ah,8           (i/p char w/o echo (like getch()))
    int 21h
    mov ah,4ch         (exit)
    int 21h
```

```
read: mov ah, 1          (i/p a char,
      int 21h                 mask the upper nibble)
      and al, 0fh
      mov bl, al         (save first byte in al)
      mov ah, 1          (i/p a char,
      int 21h                 mask the upper nibble
      and al, 0fh             al has 2nd byte
      mov ah, bl              ah has 1st byte
      add                (convert ascii to packed bcd
                          stores in al)
      ret                (return)
      end
```

10.b) Fully rectified sine wave using DAC interface

```
.model small
a db 00, 22, 43, 63, 81, 97, 109, 119, 125, 127
.code
    mov ax, @data
    mov ds, ax
    mov dx, 0e403h        (CW = 80h
    mov al, 80h            all o/p ports)
    out dx, al
    mov dx, 0e400h        (Port A → o/p)
    mov cx, 0ffffh        (loop ffff times)
    mov bx, 0             (index for a[])
b1: mov al, a[bx]
    add al, 127           (for 254 / 127 / 0)
    out dx, al
    inc bx
    cmp bx, 9             (if bx<9, goto b1)
    jb b1

b2: mov al, a[bx]
    add al, 127           (for 254 / 127 / 0)
    out dx, al
    dec bx
    jnz b2                (if bx != 0, goto b2)
    loop b1               (if cx != 0, goto b1)
    mov ah, 4ch
    int 21h               (exit)
    end
```

repe → repeat if equal

jbe → jump if Less than or equal to

cld → clear direction flag

offset xyz → gives offset address of xyz

($-a)/2 → used when array is dw

mul bl → multiplies al × bl & stores in ax (16 bits)

mul bx → " ax × bx " in ax & dx (32 bits)

CV mode : (codeview mode)

```
masm /zi   fname.asm ;
link /co   fname.obj ;
cv fname
```

f5 → press to run pgm

d variable_name (to display value of the variable)

cmp al, 0dh → check if enter character (↵) was pressed
10 → linefeed char (\n)
13 → carriage return (\r)

je → jump if equal
inc → increment
db → define byte , dw → define word
@ → address of
loop → executes till cx becomes 0

small → only one code segment
large → more than one ,,

lea → load effective address
aam → ascii adjust after multiplication
aad → ascii adjust before division (unpacked bcd to binary)
converts ax to binary & stores in al
$ → end of string ('\0')

and al, 0fh → masking the upper nibble & 0010 xxxx
0000 1111
0000 xxxx
daa → decimal adjust after addition