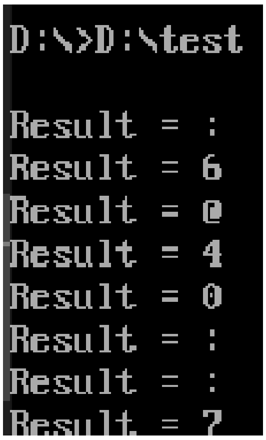
EXP2

IMPLEMENTATION OF VARIOUS ALU OPERATIONS (ADD, SUB, MUL, DIV, AND, OR, XOR, NOT) THROUGH ASSEMBLY LANGUAGE PROGRAMMING FOR 8086 USING MASM AND DEBUG.

```
.model small
.stack 100h
.data
num1 db 8
num2 db 2
result db ?
msg db 0Ah,0Dh,'Result = $'
.code
main proc
    mov ax, @data
    mov ds, ax

;======== ADDITION ========
    mov al, num1
    add al, num2       ; AL = 8 + 2 = 10
    mov result, al
    lea dx, msg
    mov ah, 9
    int 21h
    mov dl, result
    add dl, 30h
    mov ah, 2
    int 21h

;======== SUBTRACTION ========
 mov al, num1
    sub al, num2       ; AL = 8 - 2 = 6
    mov result, al
    lea dx, msg
    mov ah, 9
    int 21h
mov dl, result
    add dl, 30h
    mov ah, 2
    int 21h
;======== MULTIPLICATION ========
    mov al, num1
    mov bl, num2
    mul bl             ; AL = 8 * 2 = 16
    mov result, al
 lea dx, msg
    mov ah, 9
    int 21h
 mov dl, result
    add dl, 30h
    mov ah, 2
    int 21h
;======== DIVISION ========
    mov al, num1
    mov ah, 0
    mov bl, num2
    div bl             ; AL = quotient, AH = remainder
 lea dx, msg
    mov ah, 9
    int 21h
    mov dl, al         ; print quotient
    add dl, 30h
    mov ah, 2
    int 21h
;======== AND ========
    mov al, num1
    and al, num2
    mov result, al
  lea dx, msg
    mov ah, 9
    int 21h
mov dl, result
    add dl, 30h
    mov ah, 2
    int 21h
;======== OR ========
    mov al, num1
    or al, num2
    mov result, al
    lea dx, msg
    mov ah, 9
    int 21h
  mov dl, result
    add dl, 30h
    mov ah, 2
    int 21h
;======== XOR ========
    mov al, num1
    xor al, num2
    mov result, al
    lea dx, msg
    mov ah, 9
    int 21h
  mov dl, result
    add dl, 30h
    mov ah, 2
    int 21h
;======== NOT ========
    mov al, num1
    not al
    mov result, al
 lea dx, msg
    mov ah, 9
    int 21h
mov dl, result
    and dl, 0Fh        ; just display lower nibble
    add dl, 30h
    mov ah, 2
    int 21h
;======== EXIT ========
    mov ah, 4ch
    int 21h
main endp
end main
```

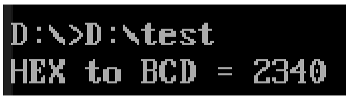


```
D:\>D:\test

Result = :
Result = 6
Result = @
Result = 4
Result = 0
Result = :
Result = :
Result = 7
```

EXP 3
IMPLEMENTATION OF NUMBER CONVERSION (HEX TO BCD, ASCII TO BCD, BCD TO ASCII) USING MASM

HEX TO BCD

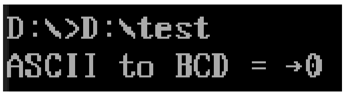
```
.model small
.stack 100h
.data
num dw 0256h          ; Example: 0256h = 598 decimal
msg db 'HEX to BCD = $'
.code
main proc
    mov ax, @data
    mov ds, ax

    mov ax, num        ; AX = 0256h (Hex)
    mov bx, 10
    mov cx, 0

    ; Show message
    mov ah, 9
    lea dx, msg
    int 21h

next_digit:
    mov dx, 0
    div bx             ; AX / 10
    push dx            ; Save remainder
    inc cx
    cmp ax, 0
    jne next_digit

display:
    pop dx
    add dl, 30h        ; Convert to ASCII
    mov ah, 2
    int 21h
    loop display

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



```
D:\>D:\test
HEX to BCD = 2340
```

ASCII TO BCD

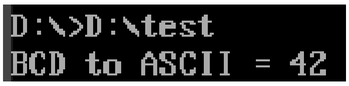
```
.model small
.stack 100h
.data
a1 db '4'     ; ASCII '4'
a2 db '2'     ; ASCII '2'
msg db 'ASCII to BCD = $'
.code
main proc
    mov ax, @data
    mov ds, ax
    mov al, a1
    sub al, 30h      ; '4' → 4
    mov ah, a2
    sub ah, 30h      ; '2' → 2
    mov bl, 10
    mul bl           ; 4 * 10 = 40
    add al, ah       ; 40 + 2 = 42
    mov ah, 9
    lea dx, msg
    int 21h
    mov bl, 10
    div bl           ; AL/10 → tens, AH → ones
    mov bh, ah
    mov dl, al
    add dl, 30h
    mov ah, 2
    int 21h
    mov dl, bh
    add dl, 30h
    mov ah, 2
    int 21h
  mov ah, 4ch
    int 21h
main endp
end main
```



```
D:\>D:\test
ASCII to BCD = →0
```

BCD TO ASCII

```
.model small
.stack 100h
.data
bcd db 42
msg db 'BCD to ASCII = $'
.code
main proc
    mov ax, @data
    mov ds, ax
 mov al, bcd
    mov ah, 0
    mov bl, 10
    div bl            ; AL = tens, AH = ones

    mov bh, ah
    mov bl, al
mov ah, 9
    lea dx, msg
    int 21h
mov dl, bl
    add dl, 30h
    mov ah, 2
    int 21h
mov dl, bh
    add dl, 30h
    mov ah, 2
    int 21h
mov ah, 4ch
    int 21h
main endp
end main
```

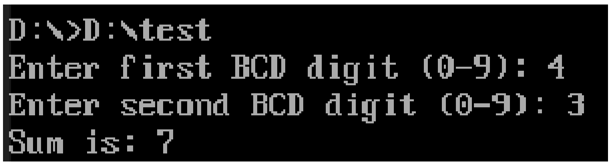


```
D:\>D:\test
BCD to ASCII = 42
```

EXP 4

IMPLEMENTATION OF TWO 8 BIT BCD ADDITION WITH ACCEPTING INPUT FROM KEYBOARD AND DISPLAYING OUTPUT ON MONITOR USING INT21H INTERRUPTS

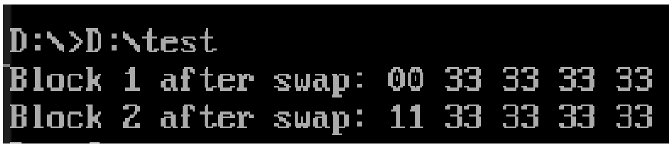
```
.model small
.stack 100h
.data
    msg1 db 'Enter first BCD digit (0-9): $'
    msg2 db 13,10,'Enter second BCD digit (0-9): $'
    msg3 db 13,10,'Sum is: $'
.code
main:
    mov ax, @data
    mov ds, ax
  ; --- Ask for first digit ---
    lea dx, msg1
    mov ah, 09h
    int 21h
 ; --- Read first digit ---
    mov ah, 01h
    int 21h
    sub al, '0'     ; Convert ASCII to number
    mov bl, al      ; Save in BL
 ; --- Ask for second digit ---
    lea dx, msg2
    mov ah, 09h
    int 21h
; --- Read second digit ---
    mov ah, 01h
    int 21h
    sub al, '0'
    add bl, al      ; Add to previous number
; --- Show message ---
    lea dx, msg3
    mov ah, 09h
    int 21h
; --- Convert result to ASCII ---
    mov al, bl
    add al, '0'
    mov dl, al
    mov ah, 02h
    int 21h         ; Show sum
 ; --- Exit ---
    mov ah, 4ch
    int 21h
end main
```



```
D:\>D:\test
Enter first BCD digit (0-9): 4
Enter second BCD digit (0-9): 3
Sum is: 7
```

EXP6
BLOCK TRANSFER AND BLOCK EXCHANGE USING INDEX REGISTERS

```
.model small
.stack 100h
.data
block1 db 10h, 20h, 30h, 40h, 50h    ; First block of 5 bytes
block2 db 01h, 02h, 03h, 04h, 05h    ; Second block of 5 bytes
.code
main PROC
    mov ax, @data
    mov ds, ax              ; DS -> data segment
    mov es, ax              ; ES -> same segment for simplicity
mov cx, 5               ; Number of bytes to swap
    lea si, block1          ; Load offset of block1 into SI
    lea di, block2          ; Load offset of block2 into DI
    cld                    ; Clear direction flag for forward increment
exchange_loop:
    mov al, [si]            ; Load byte from block1
    mov bl, [di]            ; Load byte from block2
   mov [si], bl            ; Store block2 byte into block1
    mov [di], al            ; Store block1 byte into block2
 inc si                 ; Next byte in block1
    inc di                 ; Next byte in block2
   loop exchange_loop      ; Loop CX times
 ; Program end (in DOS)
    mov ax, 4c00h
    int 21h
main ENDP
END main
```

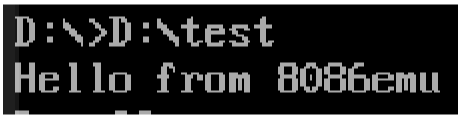


```
D:\>D:\test
Block 1 after swap: 00 33 33 33 33
Block 2 after swap: 11 33 33 33 33
```

EXP 7

IMPLEMENT FILE OPERATIONS [DOS INTERRUPTS IN C/MASM]

```
.model small
.stack 100h
.data
filename db 'MYFILE.TXT',0
writeMsg db 'Hello from 8086emu!$'
buffer   db 80 dup(0)
handle   dw ?
.code
start:
    mov ax, @data
    mov ds, ax
; Create file
    mov ah, 3Ch         ; DOS create file
    mov cx, 0           ; Normal attribute
    lea dx, filename
    int 21h
    jc error
    mov handle, ax
; Write message
    mov ah, 40h         ; DOS write file
    mov bx, handle
    lea dx, writeMsg
    mov cx, 18          ; Length of message excluding
'$'
    int 21h
    jc error
; Close file
    mov ah, 3Eh         ; DOS close file
    mov bx, handle
   int 21h
    jc error
; Open file for reading
    mov ah, 3Dh         ; DOS open file
    mov al, 0           ; Read mode
    lea dx, filename
    int 21h
    jc error
    mov handle, ax
; Read file content
    mov ah, 3Fh         ; DOS read file
    mov bx, handle
    lea dx, buffer
    mov cx, 80
    int 21h
    jc error
; Terminate string for printing
   mov si, ax              ; AX = number of characters
read
   mov byte ptr [buffer+si], '$'; Display content
    mov ah, 09h         ; DOS print string
    lea dx, buffer
    int 21h
; Close file
    mov ah, 3Eh
    mov bx, handle
    int 21h
    jc error
; Exit program
    mov ah, 4Ch
    int 21h
error:
    ; Simple error exit
    mov ah, 4Ch
    int 21h
end start
```

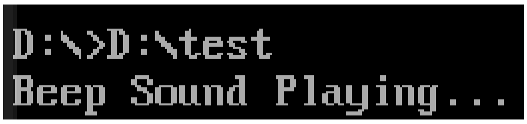


```
D:\>D:\test
Hello from 8086emu
```

EXP 8

IMPLEMENT I/O INTERFACING USING INBUILT SPEAKERS OF IBM PC

```
.model small
.stack 100h
.data
msg db 'Beep Sound Playing...$'
.code
main proc
   mov ax, @data
   mov ds, ax

   ; Print message to console
   mov ah, 09h       ; DOS print function
   lea dx, msg       ; Load message address
   int 21h           ; Display message

   ; Turn on speaker
   in al, 61h
   or al, 03h
   out 61h, al

   ; Simple delay loop to keep the sound on
   mov cx, 50000
delay:
   loop delay

   ; Turn off speaker
   in al, 61h
   and al, 0FCh
   out 61h, al

   ; Exit program
   mov ah, 4Ch
   int 21h
main endp
end main
```

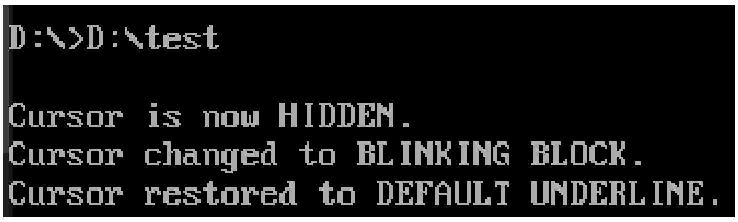


```
D:\>D:\test
Beep Sound Playing...
```

EXP 9

IMPLEMENTATION OF CURSOR ACTIVITY LIKES HIDING CURSOR AND CHANGING IT TO BOX SIZE USING INT 10H INTERRUPTS

```
.model small
.stack 100h
.data
msgHide db 13,10,'Cursor is now HIDDEN.$'
msgBlock db 13,10,'Cursor changed to BLINKING BLOCK.$'
msgRestore db 13,10,'Cursor restored to DEFAULT UNDERLINE.$'
.code

start:
    mov ax, @data
    mov ds, ax

; ------------------
; Hide cursor
; ------------------
    mov ah, 01h       ; BIOS: set cursor shape
    mov cx, 0         ; CX=0 hides the cursor
    int 10h

    ; Print status
    mov ah, 09h
    lea dx, msgHide
    int 21h

; ------------------
; Change to blinking block cursor
; ------------------
    mov ah, 01h
    mov cx, 0F00h     ; CH = end line (15), CL = start (0)
    int 10h

    ; Print status
    mov ah, 09h
    lea dx, msgBlock
    int 21h

; Wait for key press to observe the change
    mov ah, 00h
    int 16h

; ------------------
; Restore default underline cursor
; ------------------
    mov ah, 01h
    mov cx, 0706h     ; CH=7, CL=6
    int 10h

    ; Print status
    mov ah, 09h
    lea dx, msgRestore
    int 21h

; Exit program
    mov ah, 4Ch
    int 21h

end start
```

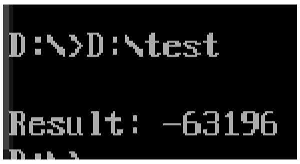


```
D:\>D:\test

Cursor is now HIDDEN.
Cursor changed to BLINKING BLOCK.
Cursor restored to DEFAULT UNDERLINE.
```

EXP10

IMPLEMENT BOOTH’S MULTIPLICATION ALGORITHM

```
.model small
.stack 100h
.data
multiplicand db 6       ; Change as needed
multiplier   db -3      ; Change as needed
count db 8
result dw 0
temp db 0
msg db 13,10,'Result: $'
negSign db '-' , '$'
buffer db 6 dup('$')    ; Buffer to display number (max 5 digits + '$')

.code
main proc
    mov ax, @data
    mov ds, ax

    ; ---------- Booth's Multiplication ----------
    mov al, multiplier   ; Q
    mov bl, multiplicand ; M
    xor ah, ah           ; A = 0
    xor dl, dl           ; Q-1 = 0
    mov cl, count        ; Loop counter

booth_loop:
    mov temp, al
    and temp, 1
    cmp temp, dl
    je skip_op
    cmp temp, 0
    je do_sub
do_add:
    add ah, bl
    jmp do_shift
do_sub:
    sub ah, bl
skip_op:
do_shift:
    rcr dl,1
    rcr al,1
    rcr ah,1
    dec cl
    jnz booth_loop

    mov result, ax   ; Store result in memory

; ------------- Display Result -------------
    mov ah, 09h
    lea dx, msg
    int 21h

    mov ax, result   ; AX contains final result
    cmp ax, 0
    jge convert_pos  ; If positive, go to convert
    ; If negative, print '-' and make ax positive
    mov ah, 09h
    lea dx, negSign
    int 21h
    neg ax           ; Convert to positive for display

convert_pos:
    ; Convert number in AX to ASCII in buffer (decimal conversion)
    mov bx, 10
    lea di, buffer+5   ; Point to end of buffer
    mov byte ptr [di], '$'
    dec di
convert_loop:
    xor dx, dx
    div bx           ; Divide AX by 10 -> quotient in AX, remainder in DX
    add dl, '0'      ; Convert remainder to ASCII
    mov [di], dl     ; Store digit
    dec di
    cmp ax, 0
    jne convert_loop

    inc di           ; Move to first valid digit

    ; Display number
    mov ah, 09h
    mov dx, di
    int 21h

    ; Exit program
    mov ah, 4Ch
    int 21h
main endp
end main
```

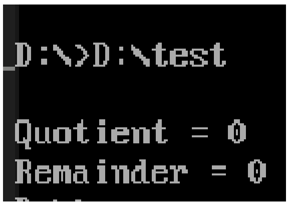


```
D:\>D:\test

Result: -63196
```

EXP 11

IMPLEMENT DIVISION ALGORITHM (NON-RESTORING AND/OR RESTORING)

```
.model small
.stack 100h
.data
    dividend db 25          ; Change this to your dividend
    divisor  db 4           ; Change this to your divisor
    quotient db 0
    remainder db 0
    msgQ db 13,10,'Quotient = $'
    msgR db 13,10,'Remainder = $'
.code
start:
    mov ax, @data
    mov ds, ax

    mov al, dividend        ; AL = dividend
    mov bl, divisor         ; BL = divisor
    xor ah, ah              ; AH = 0 (remainder)
    mov cl, 8               ; bit counter (8 bits)
    mov dl, 0               ; quotient in DL

div_loop:
    shl ah, 1               ; remainder <<= 1
    rcl al, 1               ; dividend <<= 1 through carry
    cmp ah, bl
    jb skip_subtract
    sub ah, bl
    shl dl, 1
    or dl, 1
    jmp continue_loop
skip_subtract:
    shl dl, 1
continue_loop:
    dec cl
    jnz div_loop

    mov quotient, dl
    mov remainder, ah

    ; ----- Display Quotient -----
    mov ah, 09h
    lea dx, msgQ
    int 21h

    mov al, quotient
    add al,'0'
    mov dl, al
    mov ah,02h
    int 21h

    ; ----- Display Remainder -----
    mov ah,09h
    lea dx, msgR
    int 21h

    mov al, remainder
    add al,'0'
    mov dl, al
    mov ah,02h
    int 21h

    mov ah,4Ch
    int 21h
end start
```

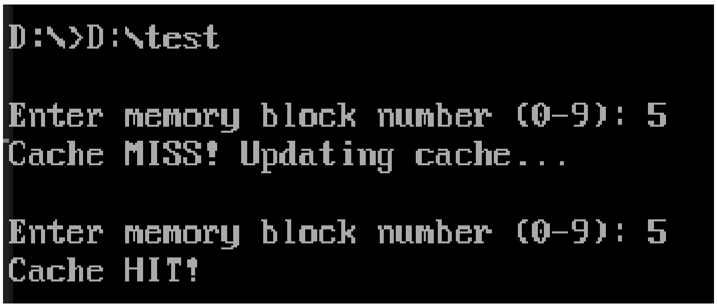


```
D:\>D:\test

Quotient = 0
Remainder = 0
```

EXP 12

IMPLEMENTATION OF MAPPING TECHNIQUES OF CACHE MEMORY

```
.model small
.stack 100h
.data
; Main memory blocks
main_mem db 10,20,30,40
; Cache memory (2 lines)
cache db 0,0
valid db 0,0          ; Valid bits
; Memory access sequence
access_seq db 0,1,2,3,0
msg_hit db 'HIT$'
msg_miss db 'MISS$'
.code
main proc
   mov ax,@data
   mov ds,ax
mov si,0              ; index in access_seq
next_access:
   mov al,access_seq[si]  ; block to access
   and al,01h            ; cache line = block % 2

   mov bl,al             ; cache line index
; Check valid and content
   mov dl,valid[bl]
   cmp dl,0
   je miss
   mov dl,cache[bl]
   cmp dl,access_seq[si]
   je hit
miss:
   mov cache[bl],access_seq[si]
   mov valid[bl],1
   mov ah,9
   lea dx,msg_miss
   int 21h
   jmp next_step
hit:
   mov ah,9
   lea dx,msg_hit
   int 21h
next_step:
   ; New line
   mov ah,2
   mov dl,0Dh
   int 21h
   mov dl,0Ah
   int 21h
  inc si
   cmp si,5
   jl next_access
 ; Exit
   mov ah,4Ch
   int 21h
main endp
end main
```

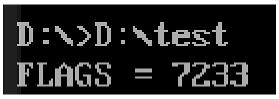


```
D:\>D:\test

Enter memory block number (0-9): 5
Cache MISS! Updating cache...

Enter memory block number (0-9): 5
Cache HIT!
```

EXP 13

DISPLAYING 8086 PROCESSOR’S FLAG REGISTER CONTENT ON MONITOR..MODEL SMALL

```
.model small
.stack 100h
.data
msg db 'FLAGS = $'
hex_digits db '0123456789ABCDEF'

.code
main proc
   mov ax, @data
   mov ds, ax

   ; Print "FLAGS = "
   mov ah, 09h
   lea dx, msg
   int 21h

   pushf              ; Push FLAGS to stack
   pop ax             ; AX = FLAGS

; =============================
; DISPLAY HIGH NIBBLE OF AH
; =============================
   mov bl, ah         ; Move AH into BL
   shr bl, 4          ; Get high nibble
   mov bh, 0          ; Clear upper byte to use BX as index
   mov si, offset hex_digits
   add si, bx         ; SI now points to correct hex digit
   mov dl, [si]       ; Load digit into DL
   mov ah, 02h
   int 21h

; =============================
; DISPLAY LOW NIBBLE OF AH
; =============================
   mov bl, ah
   and bl, 0Fh        ; Get low nibble
   mov bh, 0
   mov si, offset hex_digits
   add si, bx
   mov dl, [si]
   mov ah, 02h
   int 21h

; =============================
; DISPLAY HIGH NIBBLE OF AL
; =============================
   mov bl, al
   shr bl, 4
   mov bh, 0
   mov si, offset hex_digits
   add si, bx
   mov dl, [si]
   mov ah, 02h
   int 21h

; =============================
; DISPLAY LOW NIBBLE OF AL
; =============================
   mov bl, al
   and bl, 0Fh
   mov bh, 0
   mov si, offset hex_digits
   add si, bx
   mov dl, [si]
   mov ah, 02h
   int 21h

; Exit program
   mov ah, 4Ch
   int 21h
main endp
end main
```



```
D:\>D:\test
FLAGS = 7233
```

EXP 5
DRAWING BASIC SHAPES LIKE RECTANGLE TRANGLE ETC USING BIOS SERVICE

```
.model small
.stack 100h
.code

main:
    mov ax, 13h        ; Set video mode 13h (320x200, 256 colors)
    int 10h

    ; -------- Draw Rectangle --------
    ; Coordinates: (50,50) to (150,100)
    ; Color: 4 (red)

    mov cx, 50         ; left x
draw_rect_top:
    mov dx, 50         ; y = 50 (top line)
    call draw_pixel

    mov dx, 100        ; y = 100 (bottom line)
    call draw_pixel

    inc cx
    cmp cx, 150
    jbe draw_rect_top

    mov dx, 50         ; top y
draw_rect_sides:
    mov cx, 50         ; x = 50 (left line)
    call draw_pixel

    mov cx, 150        ; x = 150 (right line)
    call draw_pixel

    inc dx
    cmp dx, 100
    jbe draw_rect_sides

    ; -------- Draw Triangle --------
    ; Right-angle triangle at (200,50)

    mov cx, 0
next_line:
    mov dx, 0
```

```
draw_tri_line:
    mov al, 2           ; color = 2 (green)
    mov bh, 0
    mov ah, 0Ch         ; BIOS: write pixel
    mov si, cx
    add si, 200         ; x = 200 + cx
    add dx, 50          ; y = 50 + dx
    int 10h
    inc dx
    cmp dx, cx
    jbe draw_tri_line

    inc cx
    cmp cx, 50
    jbe next_line
 ; Wait for key press
    mov ah, 00h
    int 16h
 ; Return to text mode (mode 03h)
    mov ax, 03h
    int 10h
  ; Exit
    mov ah, 4Ch
    int 21h
; ------- Subroutine: draw_pixel -------
; Uses CX = x, DX = y
draw_pixel:
    mov al, 4           ; color = 4 (red)
    mov bh, 0
    mov ah, 0Ch         ; BIOS function: write pixel
    int 10h
    ret
end main
```