CH#04

1. Write a program to swap every pair of bits in the AX register.

Solution:;[org 0x0100]

```
; start:          mov ax, ABCD
;                               mov bx, 1010101010101010b
;                               mov dx, 0101010101010101b

;                               and bx,ax
;                               and dx,ax

;                               shr bx,1
;                               shl dx,1

;                               or bx,dx

;                               mov ax,bx

; end:                  mov ax, 0x4c00
;                               int 21h
```

2.Give the value of the AX register and the carry flag after each of the following instructions.

```
stc
mov ax, <your rollnumber>
adc ah, <first character of your name>
cmc
xor ah, al
mov cl, 4
shr al, cl
rcr ah, cl
```

```
[org 0x0100]

stc  ; AX:0, CF: 1

 mov ax, 0x2365  ; AX:2365, CF: 1

 adc ah, 0xA  ; AX:2E65,  CF: 0

 cmc  ; AX:2E65,  CF: 1

 xor ah, al  ; AX:4B65, CF: 0
```

```
 mov cl, 4  ; AX:4B65, CF: 0

 shr al, cl  ; AX:4B06,  CF: 0

 rcr ah, cl  ; AX:6406,  CF: 1
mov ax, 0x4c00

int 21h
```

3.Write a program to swap the nibbles in each byte of the AX register

```
;[org 0x0100]

; start:            mov ax, ABCD
;                                 mov bx, 1111000011110000b
;                                 mov dx, 0000111100001111b

;                                 and bx,ax
;                                 and dx,ax

;                                 shr bx,4
;                                 shl dx,4

;                                 or bx,dx

;                                 mov ax,bx

; end:                    mov ax, 0x4c00
;                                 int 21h
```

4. Calculate the number of one bits in BX and complement an equal number of least significant bits in AX. HINT: Use the XOR instruction

```
[org 0x0100]

                        mov ax, 0x1234
                        mov bx, 0xABCD
                        mov dx, 1000000000000000b
                        mov si, 0
                        mov cx, 0

;Calculating the no. of bits in bx

loop1:          cmp dx,0
                        jz  part2

                        test bx,dx
                        jz  skip_inc

                        inc  si
```

```
skip_inc:       shr  dx,1
                jmp  loop1


;complementing from left to right the least significant bits of ax (one at a time)

part2:    cmp si,0
          jz  end

          mov dx, 0000000000000001b


loop2:    xor ax,dx
          shl dx,1
          inc cx
          cmp cx,si
          jnz loop2


end:            mov ax, 0x4c00
                int 21h
```

5.Write a program to multiply two 32bit numbers and store the answer in a 64bit location.

```
[org 0x0100]

                jmp start

buffer:                 db
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32
startingBit:    db 105

start:                  mov dx,1111111100000000b
                        mov bx,8
                        mov cx,0
                        mov ax,0
                        mov si,0

                        mov al, [startingBit]

                        div bl

                        mov bx, 0

                        mov bl,al
                        mov si,bx
```

```
                    cmp ah,0
                    jz  scenario0
                    jnz scenario1


;Desired Byte doesn't split into two bytes

scenario0:      mov bl, [buffer + si]

                    mov ax,bx
                    jmp end


;Desired Byte splits into two bytes

scenario1:          mov cl, ah
                        mov bh, [buffer + si]       ;Done to
                        mov bl, [buffer + si + 1]   ;maintain the order of bytes in bx
                        shl bx, cl
                        and dx, bx

                        mov ax, 0
                        mov al, dh
                        jmp end


end:                mov ax,0x4c00
                        int 21h
```

6. Declare a 32byte buffer containing random data. Consider for this problem that the bits in these 32 bytes are numbered from 0 to 255. Declare another byte that contains the starting bit number. Write a program to copy the byte starting at this starting bit number in the AX register. Be careful that the starting bit number may not be a multiple of 8 and therefore the bits of the desired byte will be split into two bytes

```
[org 0x0100]

                jmp start

buffer:             db
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32
startingBit:    db 105

start:              mov dx,1111111100000000b
                        mov bx,8
                        mov cx,0
                        mov ax,0
                        mov si,0

                        mov al, [startingBit]
```

```
                    div bl

                    mov bx, 0

                    mov bl,al
                    mov si,bx

                    cmp ah,0
                    jz  scenario0
                    jnz scenario1


;Desired Byte doesn't split into two bytes

scenario0:      mov bl, [buffer + si]

                    mov ax,bx
                    jmp end


;Desired Byte splits into two bytes

scenario1:          mov cl, ah
                        mov bh, [buffer + si]       ;Done to
                        mov bl, [buffer + si + 1]   ;maintain the order of bytes in bx
                        shl bx, cl
                        and dx, bx

                        mov ax, 0
                        mov al, dh
                        jmp end


end:                mov ax,0x4c00
                        int 21h
```

7. AX contains a number between 0-15. Write code to complement the corresponding bit in BX. For example if AX contains 6; complement the 6th bit of BX.

```
[org 0x0100]

start:          mov ax, 7
                    mov bx, 0xABCD

                    mov cx,ax
                    mov dx,1000000000000000b
                    shr dx,cl
                    xor bx,dx
```

```
end:            mov ax, 0x4c00
                int 21h
```

8. AX contains a non-zero number. Count the number of ones in it and store the result back in AX. Repeat the process on the result (AX) until AX contains one. Calculate in BX the number of iterations it took to make AX one. For example BX should contain 2 in the following case:
AX = 1100 0101 1010 0011 (input – 8 ones)
AX = 0000 0000 0000 1000 (after first iteration – 1 one)
AX = 0000 0000 0000 0001 (after second iteration – 1 one) STOP

```
[org 0x0100]

        jmp start

arr: dw
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,
38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64


start:          mov ax, 13

                        sub sp,2

                        push ax

                        call myalloc

                        pop ax


                        push ax                 ;Index
                        push 13         ;Bits

                        call myfree


end:            mov ax, 0x4c00
                        int 21h



;---------------------------------------------------------------


myalloc:        push bp
                        mov bp,sp

                        sub sp,4
;creating space for two local variables
```

```
;bp - 2 will be used to store the index temporarily

;bp - 4 will be used to hold the status of zeroes whether they

;                       are currently being checked or not.


                        pusha

                        mov ax, 0
                        mov bx, 0
                        mov cx, 0

                        mov word [bp - 2], -1           ;index is -1 by default

                        mov si, [bp + 4]                                        ;No. of zeroes to be
checked

                        cmp si, 0                       ;If no. zero bits to be checked is 0, then do nothing
                        jz  dreturn

                        mov dx, 1000000000000000b                               ;mask for testing bits


                        mov word [bp - 4], 0            ;Currently we don't have a zero at hand


loop1:          test byte [arr + bx] , dh
                        jnz  reset

                        cmp word [bp - 4],1             ;If some zeroes are at hand then don't store a
new index
                        jz  loop2

zeroFound:      mov word [bp - 2], cx                                   ;Storing the index of the first
zero found
                        mov word [bp - 4], 1            ;Currently a zero is found .

loop2:          inc ax                                                                  ;No. of zeroes
currently checked
                        cmp ax,si
                        jz  changeto1

l1:                     inc cx

                        cmp cx, 0x400                                                   ;0x400bits is
equivalent to 1024 bits
                        jz  return                      ;Means you are at the end of the arr
```

```
                shr dh,1
                cmp dh,0
                jz  update

                jmp loop1


update:         mov dx, 1000000000000000b
                add bx, 1
                jmp loop1


reset:          mov ax, 0
                mov word [bp - 2], -1
                mov word [bp - 4],  0
                jmp l1

dreturn:        jmp return              ;Used because of short range jump issue at line 53

; After finding that many consecutive zero bits in the array , making them one

changeto1:      mov ax,0
                mov bx,0
                mov cx,0
                mov dx,8

                mov ax, [bp - 2] ;starting bit (index)

                div dl

                mov dx, 0

                mov dl,al
                mov bx,dx       ;Now bx contains the byte number which contains the starting
index

                mov dx,1000000000000000b

                cmp ah,0
                jnz scenario1


scenario0:      ;Desired Byte doesn't split into two bytes

loop3:          or byte [arr + bx], dh

                inc cl
                cmp cl, [bp + 4]
```

```
                        jz  return

                        shr dh,1
                        cmp dh,0
                        jz  update1
                        jmp loop3


;Desired Byte splits into two bytes

scenario1:      mov cl, ah
                        shr dx, cl

                        jmp loop3


update1:        mov dx,1000000000000000b
                        add bx,1
                        jmp loop3



return:         mov ax, [bp - 2]
                        mov [bp + 6], ax

                        popa

                        add sp, 4
                        pop bp

                        ret  2

;---------------------------------------------------------------



myfree:         push bp
                        mov bp,sp
                        pusha

                        mov ax,0
                        mov bx,0
                        mov cx,0
                        mov dx,8

                        mov ax, [bp + 6]

                        div dl
```

```
                        mov dx, 0

                        mov dl,al
                        mov bx,dx          ;Now bx contains the byte number which contains the starting
index

                        mov dx,0111111111111111b

                        cmp ah,0
                        jnz _scenario1


_scenario0:     ;Desired Byte doesn't split into two bytes

_loop3:         and byte [arr + bx], dh

                        inc cl
                        cmp cl, [bp + 4]
                        jz  _return

                        shr dh,1
                        cmp dh,0
                        jz  _update1
                        jmp _loop3


;Desired Byte splits into two bytes

_scenario1:     mov cl, ah
                        shr dx, cl

                        jmp _loop3


_update1:       mov dx,0111111111111111b
                        add bx,1
                        jmp _loop3



_return:        popa
                                pop bp
                                ret 4

;-------------------------------------------------------------
```