CHAPTER 5

CHAP_5_Ex_Q_3_:

```
[org
0x0100
]

          start:               mov ax, 4

                                      sub sp,2
                                      push ax


                                      call fibonacci


                                      pop ax


          end:                 mov ax, 0x4c00
                                      int 21h




          fibonacci:           push bp
                                      mov bp,sp


                                      sub sp,2


                                      pusha


                                      mov ax, [bp + 4]


          basecase1:           cmp ax,1
                                      jnz basecase2
                                      mov word [bp + 6],1
                                      jmp return



          basecase2:           cmp ax,0
                                      jnz calls
                                      mov word [bp + 6],0
```

```
                        jmp return



calls:          sub sp,2
                        dec ax
                        push ax


call1:          call fibonacci


                        pop word [bp - 2]           ;A local variable used
to store the return value from the first

;recursive call


                        sub sp,2
                        dec ax
                        push ax


call2:          call fibonacci
                        pop dx


                        add dx, [bp - 2]
                        mov [bp + 6],dx


return:         popa
                        add sp,2
                        pop bp
                        ret 2




;-------------------------------------------------------
;Logic Explained


;Carefully read this function




;       int fibonacci (int n)
;       {
;                       int x;                      ;The local variable made to
store the result from first recursive call
;
```

```
;                       if( n == 1 )
;                          return 1;
;
;                       if( n == 0 )
;                         return 0;
;
;
;                       x = fibonacci (n - 1);
;                       x = x + fibonacci (n - 2);
;
;                       return x;
;       }


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

Chap_05_Ex_Q_04:

```
[org
0x0100
]


start:                  mov ax, 6

                                sub sp,2
                                push ax


                                call fibonacci


                                pop ax


end:                    mov ax, 0x4c00
                                int 21h







fibonacci:              push bp
                                mov bp,sp


                                pusha


                                mov ax, [bp + 4]
```

```
                        mov word [bp + 6], 0         ;Initializing the
return value to 0


case1:          cmp ax,1
                        jnz case0
                        mov word [bp + 6], 1
                        jmp return



case0:          cmp ax,0
                        jnz l1
                        mov word [bp + 6], 0
                        jmp return



l1:                     mov dx, 0
                        mov bx, 0                    ;bx= F(0) = 0
                        mov cx, 1                    ;cx= F(1) = 1

loop1:          cmp ax,1
                        jz return


                        mov dx,cx
                        add dx,bx
                        mov [bp + 6],dx


                        mov bx, cx
                        mov cx, [bp + 6]


                        dec ax
                        jmp loop1


return:         popa
                        pop bp
                        ret 2



;-------------------------------------------------------
;Logic Explained


;Carefully read this function
```

```
;       int fibonacci (int n)
;       {
;                       if( n == 1 )
;                          return 1;
;
;                       if( n == 0 )
;                         return 0;
;
;                       int f0 = 0;
;                       int f1 = 1;
;                       int fn = 0;
;
;                       for(int i=1 ; i<n ; i++)
;                       {
;                              fn = f1 + f0
;                              f0 = f1
;                              f1 = fn
;                       }
;
;                       return fn;
;       }


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

Chap_05_Ex_Q_05:

```
[org
0x0100
]

                        jmp start




new_stack_segment: dw 0x1234
new_stack_offset:  dw 0xFFFE




start:                          mov ax,0xABCD    ;Test values
                                       mov cx,0
```

```
                                        push ax
                                        push 123


                                        push word [new_stack_segment]
                                        push word [new_stack_offset]

                                        call switch_stack


                                        pop cx
                                        pop bx

end:                            mov ax, 0x4c00
                                        int 21h



switch_stack:           push bp
                                        mov bp,sp

                                        pusha

                                        mov bx,sp
                                        sub bx,2

                                        mov si,0xFFFC           ;si will be used to
make a copy of the old stack and it is currently

;pointing at the bottom element of the old stack

                                        mov sp, [bp + 4]             ;new offset
                                        mov ss, [bp + 6]             ;new stack
segment



loop1:                          push word [si]
                                        sub si,2
                                        cmp si,bx
                                        jnz loop1




return:                         popa
                                        pop bp
                                        ret 4
```

Chap_05_Ex_Q_06:

```
[org
0x0100
]
```

```
                jmp start



arr:    dw 0,0,0,0,0,0,0,0



start:          push word testFunct
                        call addtoset


                        push word testFunct
                        call addtoset


                        call callset


end:            mov ax, 0x4c00
                        int 21h




;An implied operand say any register which stores the count of the offsets in the
array
;will make the solution simpler


addtoset:               push bp
                                mov bp, sp
                                pusha


                                mov ax, 0
                                mov bx, 0


                                mov dx, [bp + 4]                        ;Offset to be
copied


loop1:                  cmp ax, [arr + bx]
                                jnz  skip


                                mov [arr + bx], dx
```

```
                        jmp return


skip:           add bx, 2
                        cmp bx, 16
                        jnz loop1




return:         popa
                        pop bp
                        ret 2




;An implied operand say any register which stores the count of the offsets in the
array
;will make the solution simpler


callset:        push bp
                        mov bp, sp
                        pusha

                        mov ax, 0
                        mov bx, 0


_loop1:         cmp ax, [arr + bx]
                        jz  skipcall


                        call [arr + bx]


skipcall:       add bx, 2
                        cmp bx, 16
                        jnz _loop1



_return:        popa
                        pop bp
                        ret
```

```
testFunct:            ;Does nothing.

                                ret
```

Chap_05_Ex_Q_07:

```
[org
0x0100
]

                        jmp start



arr:    dw 0,0,0,0,0,0,0,0



start:          push word testFunct
                        call addtoset

                        push word testFunct
                        call addtoset

                        call callset

end:            mov ax, 0x4c00
                        int 21h




;An implied operand say any register which stores the count of the offsets in the
array
;will make the solution simpler


addtoset:               push bp
                                mov bp, sp
                                pusha

                                mov ax, 0
                                mov bx, 0
```

```
                                mov dx, [bp + 4]                        ;Offset to be
copied


loop1:                  cmp ax, [arr + bx]
                                jnz  skip


                                mov [arr + bx], dx
                                jmp return


skip:                   add bx, 2
                                cmp bx, 16
                                jnz loop1




return:                 popa
                                pop bp
                                ret 2




;An implied operand say any register which stores the count of the offsets in the
array
;will make the solution simpler


callset:                push bp
                                mov bp, sp
                                pusha

                                mov ax, 0
                                mov bx, 0


_loop1:                 cmp ax, [arr + bx]
                                jz  skipcall


                                call [arr + bx]


skipcall:               add bx, 2
                                cmp bx, 16
                                jnz _loop1
```

```
_return:              popa
                              pop bp
                              ret



testFunct:            ;Does nothing.


                              ret
```

Chap_05_Ex_Q_08:

```
[or
g
0x0
100
]


            jmp start


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


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

CHAPTER 6

Chap_06_Ex_Q_02:

```
[org
0x0100
]

                        jmp start


    character: dw '*'



    start:                                  call clrscr

                                                    push word [character]

                                                    call clash


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

```
clrscr:                         mov ax, 0xb800
                                mov es, ax
                                xor di,di
                                mov ax,0x0720
                                mov cx,2000


                                cld
                                rep stosw

                                ret




clash:                          push bp
                                mov bp,sp
                                pusha




                                mov ax, 0xb800
                                mov es, ax


                                mov bx, 12




                                ;Calculating the starting
position
                                mov al, 80
                                mul bl
                                shl ax, 1


                                mov si, ax
                                mov di, si


                                add di, 158


                                mov cx, 38


                                ;Loading the characters
                                mov al, [bp + 4]
                                mov ah, 0x07
```

```
Printing1:                      mov word [es:si], ax
                                mov word [es:di], ax

                                call _delay
                                call _delay

                                mov word [es:si], 0x0720
                                mov word [es:di], 0x0720

                                add si, 2
                                sub di, 2

                                loop Printing1

                                mov cx, 38

Printing2:                      mov word [es:si], ax
                                mov word [es:di], ax

                                call _delay
                                call _delay

                                mov word [es:si], 0x0720
                                mov word [es:di], 0x0720

                                sub si, 2
                                add di, 2

                                loop Printing2

                                mov cx, 38

                                jmp Printing1



_delay:                         mov dx, 0xFFFF

l1:                             dec dx
```

```
                                        jnz l1


                                        ret



        return:                         popa
                                        pop bp
                                        ret 2
```

Chap_06_Ex_Q_03:

```
[org
0x0100
]


               jmp start


        _segment:    dw 0xF8AB
        _offset:       dw 0xFFFF


        start: call clrscr

                       push word [_segment]
                       push word [_offset]


                       call printaddr




        end:   mov ax, 0x4c00
                       int 21h









        clrscr:              mov ax, 0xb800
                                     mov es, ax
                                     xor di,di
                                     mov ax,0x0720
```

```
                        mov cx,2000


                        cld
                        rep stosw

                        ret









;A mini sub routine to used by printaddr


print:          cmp bl, 9
                        jle Decimal
                        jg  Hex



Decimal:        add bl, 0x30
                        jmp l1

Hex:            add bl, 55
                        jmp l1


l1:                     mov word [es:di], bx
                        add di, 2


return:         ret




;Main sub-routine


printaddr:      push bp
                        mov bp,sp
                        pusha
```

```
                        mov ax, 0xb800
                        mov es, ax

                        ;Calculating the Physical Address

                        mov ax, [bp + 6]            ;segment address
                        mov bx, 0x10

                        mul bx

                        add ax, [bp + 4]        ;adding the offset
                        adc dx, 0

                        mov di, 0

                        mov bh, 0x07




;Printing Most Signicant Nibble of PA present in dx

Nibble_1st:         mov bl, 00001111b
                        and bl, dl
                        call print




;Printing the ax part of PA

Nibble_2nd:         mov bl, 11110000b
                        and bl, ah
                        shr bl, 4
                        call print
```

```
Nibble_3rd:          mov bl, 00001111b
                            and bl, ah
                            call print



Nibble_4th:          mov bl, 11110000b
                            and bl, al
                            shr bl, 4
                            call print



Nibble_5th:          mov bl, 00001111b
                            and bl, al
                            call print



_return:             popa
                            pop bp
                            ret 4
```

Chap_06_Ex_Q_04:

```
[org
0x0100
]

                jmp start



arr: times 500 db 0



start:          call spacechecker



end:            mov ax, 0x4c00
```

```
                        int 21h



spacechecker:               pusha

                                            mov ax, 0xb800
                                            mov es, ax

                                            ;Attributes wali byte locations par
tw kabhi space nahi miley gi.
                                            ;Isi lye array ki odd numbered bits
ko pehley hi 1 kar do.



                                            ;Firstly setting all bits to 1

                                            mov cx, 2000
                                            mov ax, 1111111111111111b

_setAllBits:                or [arr + bx], ax

                                            add bx, 2
                                            loop _setAllBits



                                            mov bl, 0x20          ;loading the
space character



                                            ;Now checking for spaces

                                            mov cx, 2000
                                            mov di, 0
                                            mov si, 0
```

```
                                        mov dl, 10000000b
                                        mov al, 01111111b

checkSpace:                             cmp byte [es:di], bl
                                        jnz _bit1
                                        jz  _bit0



_bit0:                                  and [arr + si], al
                                        jmp l1



_bit1:                                  or [arr + si], dl
                                        jmp l1



l1:                                     shr dl, 2        ;Skipping the odd
numbered bits as they are already set to 1
                                        shr al, 2        ;previously

                                        cmp dl, 0
                                        jnz l2

                                        mov dl, 10000000b
                                        mov al, 01111111b

                                        inc si

l2:                                     add di, 2
                                        loop  checkSpace



return:                                 popa
                                        ret
```

Chap_06_Ex_Q_05:

```
org
0x0100
]

        jmp start


top:    dw 10             ;Starting Row
bottom: dw 20              ;Ending   Row
left:   dw 30                 ;Starting Column
right:  dw 60              ;Ending Column



start:         call clrscr


                       push word [top]
                       push word [bottom]
                       push word [left]
                       push word [right]


                       call drawrect


end:           mov ax, 0x4c00
                       int 21h







clrscr:                mov ax, 0xb800
                              mov es, ax                          ;Loading the
video memory


                              xor di,di


                              mov ax,0x0720
                              mov cx,2000


                              cld
                              rep stosw
```

```
                        ret




drawrect:           push bp
                        mov  bp, sp
                        pusha



                        ; bp + 4  =  right
                        ; bp + 6  =  left
                        ; bp + 8  =  bottom
                        ; bp + 10 =  top



                        ;Calculating the top left position of the rectangle


                        mov al, 80
                        mul byte [bp + 10]
                        add ax,  [bp + 6]
                        shl ax, 1
                        mov di, ax


                        push di                             ;Saving for
later use



                        mov ah, 0x07                ;Storing the attribute



                        ;Calculating the width of the rectangle


                        mov cx, [bp + 4]
                        sub cx, [bp + 6]


                        push cx                             ;Saving for
later use
```

```
                        mov al, '+'

loop1:          rep stosw









                        pop bx

                        pop di

                        push bx

                        dec bx
                        shl bx, 1

                        add di, 160



                        ;Calculating the height of the rectangle

                        mov cx, [bp + 8]
                        sub cx, [bp + 10]

                        sub cx, 2                       ;Excluding the
top and bottom row
                        mov al, '|'

loop2:          mov si, di

                        mov word [es:si], ax
```

```
                        add si, bx

                        mov word [es:si], ax

                        sub si, bx

                        add di, 160

                        loop loop2









                        pop cx

                        mov al, '-'

loop3:          rep stosw



return:         popa
                        pop bp
                        ret 8
```