CHAPTER 5

Write a recursive function to calculate the Fibonacci of a number. The number is passed as a parameter via the stack and the calculated Fibonacci number is returned in the AX register. A local variable should be used to store the return value from the first recursive call. Fibonacci function is defined as follows:
Fibonacci(0) = 0
Fibonacci(1) = 1
Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)

```
[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
```

Write the above Fibonacci function iteratively. HINT: Use two registers to hold the current and the previous Fibonacci numbers in a loop.

```
[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
```

Write a function switch_stack meant to change the current stack and will be called as below. The function should destroy no registers.

```
push word [new_stack_segment]
push word [new_stack_offset]
call switch_stack
[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
```

Write a function “addtoset” that takes offset of a function and remembers this offset in an array that can hold a maximum of 8 offsets. It does nothing if there are already eight offsets in the set. Write another function “callset” that makes a call to all functions in the set one by one.

```
[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
```

Do the above exercise such that “callset” does not use a CALL or a JMP to invoke the functions. HINT: Setup the stack appropriately such that the RET will execute the first function, its RET execute the next and so on till the last RET returns to the caller of “callset.”

```
[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
                pusha
```

```
                    call callset
                    popa
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:    mov ax, 0
                    mov bx, 14
_loop1:         cmp ax, [arr + bx]
                    jz _skip
                    push word [arr + bx]
_skip:          sub bx,  2
                    cmp bx, -2
                    jnz _loop1
_return:    ret
testFunct:  ;Does nothing.
                        ret
```

CHAPTER 6

Write an infinite loop that shows two asterisks moving from right and left centers of the screen to the middle and then back. Use two empty nested loops with large counters to introduce some delay so that the movement is noticeable.

```
[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
```

Write a function “printaddr” that takes two parameters, the segment and offset parts of an address, via the stack. The function should print the physical address corresponding to the segment offset pair passed at the top left of the screen. The address should be printed in hex and will therefore occupy exactly five columns. For example, passing 5600 and 7800 as parameters should result in 5D800 printed at the top left of the screen.

```
[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
```

Write code that treats an array of 500 bytes as one of 4000 bits and for each blank position on the screen (i.e. space) sets the corresponding bit to zero and the rest to one.

```
[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
```

Write a function “drawrect” that takes four parameters via the stack. The parameters are top, left, bottom, and right in this order. The function should display a rectangle on the screen using the characters + - and |.

```
[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
```

Write code to find the byte in AL in the whole megabyte of memory such that each memory location is compared to AL only once.

```
[org 0x0100]
        jmp start
flag:       db 0
start:          mov al, 0x0F                    ;Byte to find
                mov bx, 0x0000                  ;Starting from segment 0x0000
l1:             mov es, bx
                mov cx, 0xFFFF
                mov di, 0
                repne scasb
                je found
                add bx, 1000
                cmp bx, 0000
                jz notFound
                jnz l1
found:          mov byte [flag], 1
                jmp exit
notFound:       jmp exit
exit:           mov ax, 0x4c00
                int 21h
```

Write a far procedure to reverse an array of 64k words such that the first element becomes the last and the last becomes the first and so on. For example if the first word contained 0102h, this value is swapped with the last word. The next word is swapped with the second last word and so on. The routine will be passed two parameters through the stack; the segment and offset of the first element of the array.

```
[org 0x0100]
                jmp start
_segment:       dw 0x3000
_offset :       dw 0x1000
start:          push word [_segment] ;
                        push word [_offset]  ;
                        call reverseArray
end:            mov ax, 0x4c00
                        int  21h
reverseArray:   push bp
                        mov bp, sp
                        pusha
                        mov cx, 0xFFFF;To compare 64k words, the comparision should be done for 32k words onl
                        mov ds, [bp + 6]                ;Starting Segment
                        mov si, [bp + 4]                ;Starting Offset
                        mov di, si                              ;Ending Offset
                        mov ax, ds
                        add ax, 0x2000                  ;going to the third segment
                        mov es, ax
                        std                                             ;set direction flag
                        cmp di, 0
 scenario0:             jnz loop1                       ;We have three overlapping segments
 scneario1:             mov dx, es
                        sub dx, 0x1000          ;We have two non-overlapping segments
                        mov es, dx
                        mov di, 0xFFFE
loop1:          mov ax, [es : di]
                        movsw
                        add si, 2                       ;because 2 has been subtracted by movsw
                        mov [si], ax                    ;swapping values
                        add si, 2
                        cmp si, 0xFFFF                  ;check if the segment has ended
                        jne l1
                        mov dx, ds
                        add dx, 0x1000                  ;if ended move on to the next segment
                        mov ds, dx
                        mov si, 0                       ;resetting si
l1:                     cmp di, 0xFFFE                  ;if the last seg has ended
                        jne l2
                        mov dx, es
                        sub dx, 0x1000                  ;going to the 2nd segment backward
                        mov es, dx
                        mov di, 0xFFFE                  ;resetting to last word
l2:                     loop loop1
return:         popa
                        pop bp
                        ret 4
```

Write a near procedure to copy a given area on the screen at the center of the screen without using a temporary array. The routine will be passed top, left, bottom, and right in that order through the stack. The parameters passed will always be within range, the height will be odd and the width will be even so that it can be exactly centered.

```
[org 0x0100]
        jmp start
top:    dw 17
bottom: dw 20
left:   dw 15
right:  dw 30
```

```
start:          push word [top]
                push word [left]
                push word [bottom]
                push word [right]
                call copyAtCenter
end:            mov ax, 0x4c00
                int 21h
copyAtCenter:   push bp
                mov bp, sp
                pusha
                push es
                push ds
                ;bp+4 = right
                ;bp+6 = bottom
                ;bp+8 = left
                ;bp+10 = top
                mov ax, 0xB800
                mov es, ax
                ;Center of screen
                ;Row = 12
                ;Col = 39,40
                mov bx, 39                              ;Mid Col
                mov dx, 12                              ;Mid Row
                ;Calculating Width
                mov ax, [bp + 4]
                sub ax, [bp + 8]
                push ax                                 ;Saving width for later use
                sub ax, 2
                shr ax, 1
                ;Getting to the required starting column
                sub bx, ax
                ;Calculating height
                mov ax, [bp + 6]
                sub ax, [bp + 10]
                push ax                                 ;Saving height for later use
                sub ax, 1
                shr ax, 1
                ;Getting to the required starting row
                sub dx, ax
                ;Staring position of source
                mov al , 80
                dec byte [bp + 10]
                mul byte [bp + 10]    ;Top
                dec byte [bp + 8]
                add ax, [bp + 8]        ;Left
                shl ax, 1
                mov si, ax
                ;Starting position of destination
                mov al, 80                                              ;Load al with columns per row
                mul dl                                                  ;Multiply with y position
                add ax, bx                                              ;add x position
                shl ax, 1
                mov di, ax
                pop ax                          ;Height
                pop cx                          ;Width
                push es
                pop  ds
                mov bx, 0
                ;Now moving the area to the center
l1:             push si
                push di
```

```
                push cx
                rep movsw
                pop cx
                pop di
                pop si
                add si, 160
                add di, 160
                inc bx
                cmp bx, ax
                jnz l1
                pop ds
                pop es
return:         popa
                pop bp
                ret 8
```

Write code to find two segments in the whole memory that are exactly the same. In other words find two distinct values which if loaded in ES and DS then for every value of SI [DS:SI]=[ES:SI].

```
[org 0x0100]
start:          sub sp, 2                       ;return value
                call findEqualSegments
                pop bx
        ;ax = 1 indicates two equal segments are found otherwise
        ;bx = 0
end:            mov ax, 0x4c00
                int  21h
findEqualSegments:push bp
                mov bp, sp
                pusha
                mov word [bp + 4], 0
                mov ax, 0
                mov dx, 0
                mov si, 0
                mov di, 0
                mov cx, 0xFFFF
                ;Finding tw non-overlapping and equal segments
                ;There are a total of 16 distinct segments in a memory of 1MB
                mov ds, ax              ;Starting from the segment 0x0000 (1st Segment)
                mov ax, 0x1000
                mov es, ax                      ;2nd Segment
                cld
loop1:          repe cmpsb              ;repeat while equal cx times
                je areEqual             ;if the segments were equal
check_ES:       mov ax, es
                cmp ax, 0xF000          ;checking for the last segment (16th Segment)
                jz  check_DS
                mov ax, es                      ;Next non-overlapping segment
                add ax, 0x1000
                mov es, ax
                mov di, 0
                mov si, 0
                mov cx, 0xFFFF
                jmp loop1
check_DS:       mov ax, ds
                cmp ax, 0xF000 ;If DS = 0xF000, it means we are at the last segment, and this
                ;segment doesn't need to be compared with itself. So no further
                ;processing is to be done and we haven't found two ;equal segments
                jz return
                mov ax, ds                      ;Next non-overlapping segment
                add ax, 0x1000
                mov ds, ax
                add ax, 0x1000
```

```
                                        mov es, ax
                                        mov si, 0
                                        mov di, 0
                                        mov cx, 0xFFFF
                                        jmp loop1
areEqual:       mov word [bp + 4], 1            ;Two equal segments are found
return:                         popa
                                        pop bp
                                        ret
;Two overlapping and equal segments can be found, but the processing takes too much time.
;Anyways, the code for that is given below
;instead of adding 0x1000, now 0x0001 is being added and instead of comparing with 0xF000, now the comparision
;is being done with 0xFFFF
;                                       mov ds, ax              ;Starting from the segment 0x0000 (1st Segment)
;                                       mov ax, 0x0001
;                                       mov es, ax                      ;2nd Segment
;                                       cld
;loop1:                         repe cmpsb                      ;repeat while equal cx times
;                                       je areEqual             ;if the segments were equal
;check_ES:                      cmp es, 0xFFFF                  ;checking for the last segment
;                                       jz  check_DS
;                                       mov ax, es              ;Next overlapping segment
;                                       add ax, 0x0001
;                                       mov es, ax
;                                       mov di, 0
;                                       mov si, 0
;                                       mov cx, 0xFFFF
;                                       jmp loop1
;check_DS:                      cmp ds, 0xFFFF  ;If DS = 0xFFFF, it means we are at the last segment, and this

        ;segment doesn't need to be compared with itself. So no further
                                                        ;processing is to be done and we haven't found two equal
segments
;                                       jz return
;                                       mov ax, ds                              ;Next overlapping
segment
;                                       add ax, 0x0001
;                                       mov ds, ax
;                                       add ax, 0x0001
;                                       mov es, ax
;                                       mov si, 0
;                                       mov di, 0
;                                       mov cx, 0xFFFF
;                                       jmp loop1
```

Write a function “strcpy” that takes the address of two parameters via stack, the one pushed first is source and the second is the destination. The function should copy the source on the destination including the null character assuming that sufficient space is reserved starting at destination.

```
[org 0x0100]
start:          push src
                        push dest
                        call strcpy
end:            mov ax, 0x4c00
                        int 21h
strLen:                 push bp
                                mov bp, sp
                                pusha
                                push es
                                push ds
                                pop es
                                mov di, [bp+4]          ;Point di to string
                                mov cx, 0xFFFF          ;Load Maximum No. in cx
```

```
                                mov al, 0                       ;Load a zero in al
                                repne scasb                             ;find zero in the string
                                mov ax, 0xFFFF                  ;Load Maximum No. in ax
                                sub ax, cx      ;Find change in cx
                                dec ax                                  ;Exclude null from length
                                mov [bp+6], ax
                                pop es
                                popa
                                pop bp
                                ret 2
strcpy:                 push bp
                                mov bp, sp
                                pusha
                                push es
                                ;bp + 6  = src address
                                ;bp + 4  = dest address
                                mov si, [bp + 6]                                        ;Setting si to source str
                                push ds
                                pop  es                 ;Setting es
                                mov di, [bp + 4]                                        ;Setting di to destination str
                                sub sp, 2
                                push word [bp + 6]
                                call strLen                             ;Calculating the length of source string because
ultimately the source and the destination will be of the same size
                                pop cx
                                inc cx      ;Incrementing cx by one so that null character gets included in the string length
                                rep movsb
                                pop es
return:                 popa
                                pop bp
                                ret 4
src:    db 'My name is NULL',0
dest:   db 000000000000000
```