Qno1)

```
[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
```

QNo2)

```
[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 only

                    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
```

Qn03)
;Write a subroutine 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
```

Qno4)

```
[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
```

Qno5)
;Virtual Window on the Screen

```
[org 0x0100]

	jmp start


character:	dw 'H'

				;top, left, bottom, right, current row, current column, normal attribute,
cursor attribute
address:	dw 0,	20,	10	,	70,		10,			25,
		07,				10000111b


start:		;call clrscr

			push word [character]
			push address

			call virtualWindow

exit:		mov ax, 0x4c00
			int 21h



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

;Clear Screen
clrscr:			pusha
				push es

				mov ax, 0xb800
```

```
                              mov es, ax
                              xor di,di
                              mov ax,0x0720
                              mov cx,2000

                              cld
                              rep stosw

                              pop es
                              popa
                              ret
;-----------------------------------------------------------------------------------------------------------

scrollUp:         pusha
                        push es
                        push ds



                        ;Calculating the starting point of the VW

                        mov al, 80
                        mul byte [si]
                        add ax, [si+2]
                        shl ax,  1

                        push ax


                        ;Loading the video memory
                        mov ax, 0xb800
                        mov es, ax
                        mov ds, ax

                        pop ax

                        ;Height times loop chaley ga
                        mov dx, [bp - 2]
                        inc dx

                        cld



a1:                     mov di, ax                              ;Destination Point

                        mov si, ax                              ;Source Point

                        add si, 160
```

```
                    ;Width jitney character move hon ge har iteration mai
                    mov cx, [bp - 4]

                    rep movsw

                    add ax, 160

                    dec dx
                    jnz a1


                    ;Width jitney character move hon ge har iteration mai
                    mov cx, [bp - 4]

                    sub ax, 160
                    mov di,ax
                    mov ax, 0x720

                    rep stosw


                    pop ds
                    pop es
                    popa
                    ret

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

virtualWindow:push bp
                        mov bp, sp

                        sub sp, 4                          ; Making three local variables for
storing length and width
                                                                   ; of the Virtual Window on
the screen


                        ;bp - 2                  ;Height
                        ;bp - 4                            ;Width

                        ;bp + 4                            ;Address
                        ;bp + 6                            ;Character

                        ;Address + 0             ;Top
                        ;Address + 2             ;Left
                        ;Address + 4             ;Bottom
                        ;Address + 6             ;Right
                        ;Address + 8             ;Current Row
                        ;Address + 10            ;Current Column
                        ;Address + 12            ;Normal Attribute
                        ;Address + 14            ;Cursor Attribute
```

```
push es
pusha



mov si, [bp + 4]

;Calculating Height
mov ax, [si]
mov bx, [si + 4]

sub bx, ax

mov [bp - 2], bx        ;Height



;Calculating Width
mov ax, [si + 2 ]
mov bx, [si + 6]

sub bx, ax

mov [bp - 4], bx        ;Width



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


;Calculating the required position

mov ax, 0xb800
mov es, ax

;Exact Row
mov ax, [si]
add ax, [si + 8]

mov bx, ax


;Exact Column
mov ax, [si + 2]
add ax, [si + 10]

mov dx, ax
```

```
                        ;Exact Position
                        mov al, 80
                        mul bl
                        add ax, dx
                        shl ax,  1
                        mov di, ax


                        ;Loading al with the character to be written
                        mov al, [bp + 6]

                        mov ah, [si + 12]

                        mov [es:di], ax


                        inc dx

                        cmp dx, [si + 6]
                        jle l2

                        mov dx, [si + 2]

                        inc bx

                        cmp bx, [si + 4]
                        jle l2

                        ;call scroll Scroll Up
                        call scrollUp


                        ;Exact Position
l2:                     mov al, 80
                        mul bl
                        add ax, dx
                        shl ax,  1
                        mov di, ax


                        ;Loading the character to be written
                        mov ah, [si + 14]
                        mov al, '_'

                        mov [es:di], ax



return:             popa
                        pop es
```

```
                                        add sp, 4

                                        pop bp
                                        ret 4
```

Qno6)

```
;  Write a subroutine "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
```