Codes

Bubble Sort:

```
[org 0x0100]
jmp Start

data: dw 60, 55, 45, 50, 40, 35, 25, 30, 10, 0

bubblesort:
  push bp
  mov bp, sp
  sub sp, 2
  push ax
  push bx
  push cx
  push si
  mov bx, [bp+6]     ; Load start of array in bx
  mov cx, [bp+4]     ; Load count of elements in cx
  dec cx
  shl cx, 1

mainloop:
  mov si, 0
  mov word [bp-2], 0 ; Reset swap flag to no swaps

innerloop:
  mov ax, [bx+si]
  cmp ax, [bx+si+2]
  jbe noswap

  xchg ax, [bx+si+2]
  mov [bx+si], ax
  mov word [bp-2], 1

noswap:
  add si, 2
  cmp si, cx
  jne innerloop

  cmp word [bp-2], 1
  je mainloop
```

```
  pop si
  pop cx
  pop bx
  pop ax
  mov sp, bp
  pop bp
  ret 4

Start:
  mov ax, data
  push ax
  mov ax, 10          ; Size of Array
  push ax
  call bubblesort

  mov ax, 0x4c00
  int 0x21
```

Find a Binary pattern in a Binary Number:

```
[org 0x0100]
jmp Start

findPattern:
   push bp
   mov bp, sp
   push bx
   push cx
   push dx
   push si

   mov bx, [bp + 8]
   mov dx, [bp + 6]
   mov cx, [bp + 4]
   mov ax, 0 ; Clear AX

   cmp cx, 0
   jbe NotFound ; If n is 0 or negative, return -1
   cmp cx, 16
   jae NotFound ; If n is greater than or equal to 16, return -1

Search:
   mov si, bx
   and si, dx
```

```
   cmp si, dx
   je Found
   shl dx, 1
   inc cx
   jmp Search

NotFound:
   mov ax, -1 ; Set ax to -1 (pattern not found)
   jmp Done

Found:
   mov ax, 16
   sub ax, cx ; Set ax to Index (pattern found)

Done:
   pop si
   pop dx
   pop cx
   pop bx
   pop bp
   ret 6

Start:
   mov bx, 1110111100001010b ; 16-bit number
   push bx
   mov dx, 111100b ; 6-bit pattern
   push dx
   mov cx, 6 ; size of pattern
   push cx
   call findPattern

   ; AX now contains the starting index of the pattern or -1 if not found

End:
mov ax, 0x4c00 ; Terminate the program
int 0x21
```

Find Difference Between two Sets:

```
[org 0x0100]
jmp Start

S1: db -3, -1, 2, 5, 6, 8, 9
S2: db -2, 2, 6, 7, 9
```

```
Size1: dw 7
Size2: dw 5

find_Difference:
   push bp
   mov bp, sp
   push ax
   push bx
   push cx
   push dx
   push si
   push di

   mov si, 0
   loop1:
      mov bx, [bp + 10] ; Set1
      mov al, [bx + si]

      mov di, 0
      loop2:
         mov bx, [bp + 8]  ; Set2
         mov dl, [bx + di]

         cmp al, dl
         jne next

         push si

         loop3:
            mov bx, [bp + 10]
            mov ah, [bx + si + 1]
            mov [bx + si], ah
            inc si
            cmp si, [bp + 6]
            jne loop3
            dec si
            mov byte [bx + si], 0
            dec word [bp + 6]
            pop si
            cmp si, [bp + 6]
            je done
            jmp loop1

      next:
         inc di
```

```
          cmp di, [bp + 4]
          jne loop2
          inc si
          cmp si, [bp + 6]
          je done
          jmp loop1

done:
    pop di
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    pop bp
    ret 8

Start:
   mov ax, S1
   push ax
   mov ax, S2
   push ax
   mov ax, [Size1]
   push ax
   mov ax, [Size2]
   push ax
   call find_Difference

   mov ax, 0x4c00
   int 0x21
```

Remove Duplicate Values from Array:

```
[org 0x0100]
jmp Start

array: db 1, 2, 1, 3, 4, 7, 9, 3, 2, 5
size: dw 10

removeDuplicates:
   push bp
   mov bp, sp
   push ax
   push bx
```

```
    push cx
    push dx
    push si
    push di

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

    mov dx, 0
    mov ax, 0

    loop1:
       mov bx, dx
       mov al, [si + bx]
       mov di, dx
       add di, 1
       cmp di, cx
       je done

       loop2:
          mov bx, di
          cmp al, [si + bx]
          je duplicate
          inc di
          cmp di, cx
          je next
          jmp loop2

       next:
          inc dx
          cmp dx, cx
          je done
          jmp loop1

       duplicate:
          mov al, [si + bx + 1]
          mov [si + bx], al
          inc bx
          cmp bx, cx
          jne duplicate
          dec bx
          mov byte [si + bx], 0
          dec cx
          inc dx
          cmp dx, cx
```

```
          je done
          jmp loop1

done:
   pop di
   pop si
   pop dx
   pop cx
   pop bx
   pop ax
   pop bp
   ret 4

Start:
   mov ax, array
   push ax
   mov ax, [size]
   push ax
   call removeDuplicates

End:
   mov ax, 0x4c00
   int 0x21
```

Clear Screen:

```
[org 0x0100]
jmp Start

clrscr:
    push es
    push ax
    push cx
    push di

    mov ax, 0xb800
    mov es, ax
    xor di, di
    mov ax, 0x0720
    mov cx, 2000
    cld
    rep stosw

    pop di
```

```
    pop cx
    pop ax
    pop es
    ret

Start:
    call clrscr

    mov ax, 0x4c00
    int 0x21
```

Print String:

```
[org 0x0100]
jmp Start

message: db 'hello world', 0

strlen:
    push bp
    mov bp, sp
    push es
    push cx
    push di

    les di, [bp+4]
    mov cx, 0xffff
    xor al, al
    repne scasb
    mov ax, 0xffff
    sub ax, cx
    dec ax

    pop di
    pop cx
    pop es
    pop bp
    ret 4

printstr:
    push bp
    mov bp, sp
    push es
    push ax
```

```
    push cx
    push si
    push di

    push ds
    mov ax, [bp+4]
    push ax
    call strlen

    cmp ax, 0
    jz exit

    mov cx, ax
    mov ax, 0xb800
    mov es, ax
    mov al, 80
    mul byte [bp+8]
    add ax, [bp+10]
    shl ax, 1
    mov di, ax
    mov si, [bp+4]
    mov ah, [bp+6]
    cld

nextchar:
    lodsb
    stosw
    loop nextchar

exit:
    pop di
    pop si
    pop cx
    pop ax
    pop es
    pop bp
    ret 8

Start:
    call clrscr

    mov ax, 30
    push ax                 ; Push x position
    mov ax, 20
    push ax                 ; Push y position
```

```
    mov ax, 0x71            ; Blue on white attribute
    push ax
    mov ax, message
    push ax                 ; Push address of message
    call printstr

    mov ax, 0x4c00
    int 0x21
```

Screen Scrolling:

```
[org 0x0100]
jmp Start

scrollup:
    push bp
    mov bp, sp
    push ax
    push cx
    push si
    push di
    push es
    push ds

    mov ax, 80
    mul byte [bp+4]
    mov si, ax
    push si
    shl si, 1
    mov cx, 2000
    sub cx, ax
    mov ax, 0xb800
    mov es, ax
    mov ds, ax
    xor di, di
    cld
    rep movsw

    mov ax, 0x0720
    pop cx
    rep stosw

    pop ds
    pop es
```

```
    pop di
    pop si
    pop cx
    pop ax
    pop bp
    ret 2

scrolldown:
    push bp
    mov bp, sp
    push ax
    push cx
    push si
    push di
    push es
    push ds

    mov ax, 80
    mul byte [bp+4]
    push ax
    shl ax, 1
    mov si, 3998
    sub si, ax
    pop ax
    push ax
    mov cx, 2000
    sub cx, ax
    mov ax, 0xb800
    mov es, ax
    mov ds, ax
    mov di, 3998
    std
    rep movsw

    mov ax, 0x0720
    pop cx
    rep stosw

    pop ds
    pop es
    pop di
    pop si
    pop cx
    pop ax
    pop bp
```

```
    ret 2

Start:
    mov ax, 5
    push ax                 ; Push the number of lines to scroll
    call scrollup

    mov ax, 0x4c00
    int 0x21
```

String Comparison:

```
[org 0x0100]
jmp Start

msg1: db 'hello world', 0
msg2: db 'hello WORLD', 0

strcmp:
    push bp
    mov bp, sp
    push cx
    push si
    push di
    push es
    push ds

    lds si, [bp+4]   ; Point ds:si to the first string
    les di, [bp+8]   ; Point es:di to the second string

    push ds
    push si
    call strlen      ; Calculate the length of the first string
    mov cx, ax

    push es
    push di
    call strlen      ; Calculate the length of the second string
    cmp cx, ax       ; Compare the lengths of both strings
    jne exitfalse

    add cx, 1
    mov ax, 1        ; Store 1 in ax to be returned
    repe cmpsb       ; Compare both strings
```

```
    jcxz exitsimple

exitfalse:
    mov ax, 0        ; Store 0 to mark them as unequal

exitsimple:
    pop ds
    pop es
    pop di
    pop si
    pop cx
    pop bp
    ret 8

Start:
    push ds          ; Push the segment of the first string
    mov ax, msg1
    push ax          ; Push the offset of the first string
    push ds          ; Push the segment of the second string
    mov ax, msg2
    push ax          ; Push the offset of the second string
    call strcmp      ; Call the strcmp subroutine

    mov ax, 0x4c00
    int 0x21
```

Print Number:

```
[org 0x0100]
jmp Start

printnum:
    push bp
    mov bp, sp
    push es
    push ax
    push bx
    push cx
    push dx
    push di

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

```
    mov ax, [bp+4] ; Load the number to be printed into AX
    mov bx, 10
    mov cx, 0

nextdigit:
    mov dx, 0
    div bx         ; Divide AX by 10
    add dl, 0x30
    push dx

    inc cx
    cmp ax, 0
    jnz nextdigit

    mov di, 0

nextpos:
    pop dx
    mov dh, 0x07
    mov [es:di], dx
    add di, 2
    loop nextpos

    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    pop es
    pop bp
    ret 2

Start:
    call clrscr

    mov ax, 4529   ; Number
    push ax
    call printnum

    mov ax, 0x4c00
    int 0x21
```

Swap Screen Left/Right:

```
[org 0x0100]
jmp Start

Swap_Left_Right:
   push bp
   mov bp, sp
   push ax
   push bx
   push cx
   push dx
   push si
   push di
   push es
   push ds

   mov cx, 0
   mov dx, 0
   mov bx, 0
   mov ax, 0xb800
   mov es, ax
   mov ds, ax

   loop1:
      mov ax, 80
      mul cl
      mov di, ax
      add di, 0
      shl di, 1

      mov ax, 80
      mul dl
      mov si, ax
      add si, 40
      shl si, 1

      mov bx, si

      loop2:
         mov ax, [es:di]
         xchg ax, [ds:si]
         mov [es:di], ax

         add di, 2
         add si, 2
```

```
         cmp di, bx
         je nextRow
         jmp loop2

      nextRow:
         add cl, 1
         add dl, 1
         cmp cl, 25
         je done
         jmp loop1
done:
   pop ds
   pop es
   pop di
   pop si
   pop dx
   pop cx
   pop bx
   pop ax
   pop bp
   ret

Start:

   call Swap_Left_Right

   mov ax, 0x4c00
   int 0x21
```

Swap Screen Diagonally:

```
[org 0x0100]
jmp Start

Swap_Diagonally:
   push bp
   mov bp, sp
   push ax
   push bx
   push cx
   push dx
   push si
   push di
   push es
```

```
    push ds

    mov cx, 0
    mov dx, 13
    mov bx, 0

    mov ax, 0xb800
    mov es, ax
    mov ds, ax

    loop1:
       mov ax, 80
       mul cl
       mov di, ax
       add di, 0
       shl di, 1

       mov ax, 80
       mul dl
       mov si, ax
       add si, 40
       shl si, 1

       mov ax, 80
       mul cl
       mov bx, ax
       add bx, 40
       shl bx, 1

       loop2:
          mov ax, [es:di]
          xchg ax, [ds:si]
          mov [es:di], ax

          add di, 2
          add si, 2
          cmp di, bx
          je nextRow
          jmp loop2

       nextRow:
          add cl, 1
          add dl, 1
          cmp cl, 12
          je done
```

```
         jmp loop1

done:
   pop ds
   pop es
   pop di
   pop si
   pop dx
   pop cx
   pop bx
   pop ax
   pop bp
   ret

Start:

   call Swap_Diagonally

   mov ax, 0x4c00
   int 0x21
```

Image Mirroring:

```
[org 0x0100]
jmp Start

Screen_Mirror:
   push bp
   mov bp, sp
   push ax
   push bx
   push cx
   push dx
   push si
   push di
   push es
   push ds

   mov cx, 0
   mov dx, 79
   mov bx, 0

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

```
    mov ds, ax

    loop1:
       push cx
       mov cx, 0
       mov ax, 80
       mul cl
       mov di, ax
       pop cx
       add di, cx
       shl di, 1

       push cx
       mov cx, 24
       mov ax, 80
       mul cl
       mov si, ax
       pop cx
       add si, cx
       shl si, 1
       mov bx, si

       push cx
       mov cx, 0
       mov ax, 80
       mul cl
       pop cx
       mov si, ax
       add si, dx
       shl si, 1

    loop2:
       mov ax, [es:di]
       xchg ax, [ds:si]
       mov [es:di], ax

       add di, 160
       add si, 160
       cmp di, bx
       je nextCol
       jmp loop2

    nextCol:
       add cx, 1
       sub dx, 1
```

```
      cmp cx, 40
      je done
      jmp loop1

done:
   pop ds
   pop es
   pop di
   pop si
   pop dx
   pop cx
   pop bx
   pop ax
   pop bp
   ret

Start:
   call Screen_Mirror

   mov ax, 0x4c00
   int 0x21
```

Copy & Pasting:

```
[org 0x0100]
jmp Start

Copy:
   push bp
   mov bp, sp
   push ax
   push bx
   push cx
   push si
   push di
   push es
   push ds

   push ds
   pop es
   mov ax, 0xb800
   mov ds, ax

   mov di, memory
```

```
   mov si, [bp + 8]   ; Starting Point
   mov bx, 0

   cld

   Copyloop1:
      mov cx, [bp + 4]  ; Number of Columns
      push si
      rep movsw
      inc bx
      cmp bx, [bp + 6]  ; Number of Rows
      je doneCopy
      pop si
      add si, 160
      jmp Copyloop1

doneCopy:
   pop si
   pop ds
   pop es
   pop di
   pop si
   pop cx
   pop bx
   pop ax
   pop bp
   ret 6

Paste:
   push bp
   mov bp, sp
   push ax
   push bx
   push cx
   push si
   push di
   push es
   push ds

   mov ax, 0xb800
   mov es, ax

   mov si, memory
   mov di, [bp + 8]   ; Starting Point
   mov bx, 0
```

```
   cld

   Pasteloop1:
      mov cx, [bp + 4]  ; Number of Columns
      push di
      rep movsw
      inc bx
      cmp bx, [bp + 6]  ; Number of Rows
      je donePaste
      pop di
      add di, 160
      jmp Pasteloop1

donePaste:
   pop si
   pop ds
   pop es
   pop di
   pop si
   pop cx
   pop bx
   pop ax
   pop bp
   ret 6

Start:
   mov ax, 644  ; Starting Point
   push ax
   mov ax, 6    ; Number of Rows to Copy
   push ax
   mov ax, 6    ; Number of Columns to Copy
   push ax
   call Copy

   mov ax, 658
   push ax
   mov ax, 6
   push ax
   mov ax, 6
   push ax
   call Paste

End:
   mov ax, 0x4c00
```

```
   int 0x21

memory: dw 0
```

Draw Triangle:

```
[org 0x0100]
jmp Start

drawTri:
   push bp
   mov bp, sp
   push ax
   push cx
   push di
   push es

   mov ax, 0xb800
   mov es, ax

   mov di, [bp + 6]    ; Starting Point
   mov cx, [bp + 4]    ; Size
   mov ax, 0x072B

   loop1:
      mov [es:di], ax
      dec cx
      cmp cx, 0
      je next1
      add di, 160
      jmp loop1

   next1:
      mov cx, [bp + 4]
   loop2:
      rep stosw

   next2:
      mov cx, [bp + 4]
   loop3:
      mov [es:di], ax
      dec cx
      cmp cx, 0
      je done
```

```
      sub di, 160
      sub di, 2
      jmp loop3

done:
   pop es
   pop di
   pop cx
   pop ax
   pop bp
   ret

Start:
   mov ax, 644  ; Starting Point
   push ax
   mov ax, 10   ; Size
   push ax
   call drawTri

End:
   mov ax, 0x4c00
   int 0x21
```

```
[org 0x0100]
jmp Start

type: dw 1, 2, 3
x: dw 30, 40, 50
y: dw 2, 8, 16
height: dw 6, 8, 6

Delay:
    ; Outer loop
    mov cx, 40
outerLoop:
    ; Inner loop
    mov dx, 65535
innerLoop:
    dec dx
    jnz innerLoop

    dec cx
    jnz outerLoop
```

```
    ret

drawTri1:
    push bp
    mov bp, sp
    push ax
    push bx
    push cx
    push dx
    push di

    ; Calculate Starting Point
    mov ax, 0
    mov al, 80
    mul byte [bp + 6]
    add ax, [bp + 8]
    shl ax, 1
    mov di, ax

    mov cx, 1
    mov dx, [bp + 4]

    mov ax, 0xb800
    mov es, ax

    mov al, 0x2B
    mov ah, 0x01

    cld

loop2:
    push di
    push cx
    rep stosw
    pop cx
    add cx, 1
    pop di
    add di, 160
    sub dx, 1
    cmp dx, 0
    je done1
    jmp loop2

done1:
    pop di
```

```
    pop dx
    pop cx
    pop bx
    pop ax
    pop bp
    ret 6

drawTri2:
    push bp
    mov bp, sp
    push ax
    push bx
    push cx
    push dx
    push di

    ; Calculate Starting Point
    mov ax, 0
    mov al, 80
    mul byte [bp + 6]
    add ax, [bp + 8]
    shl ax, 1
    mov di, ax

    mov cx, 1
    mov dx, [bp + 4]

    mov ax, 0xb800
    mov es, ax

    mov al, 0x2B
    mov ah, 0x02

    std

loop3:
    push di
    push cx
    rep stosw
    pop cx
    add cx, 1
    pop di
    add di, 160
    sub dx, 1
    cmp dx, 0
```

```
    je done2
    jmp loop3

done2:
    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    pop bp
    ret 6

drawTri3:
    push bp
    mov bp, sp
    push ax
    push bx
    push cx
    push dx
    push di

    ; Calculate Starting Point
    mov ax, 0
    mov al, 80
    mul byte [bp + 6]
    add ax, [bp + 8]
    shl ax, 1
    mov di, ax

    mov cx, 1
    mov dx, [bp + 4]

    mov ax, 0xb800
    mov es, ax

    mov al, 0x2B
    mov ah, 0x04

    cld

loop4:
    push di
    push cx
    rep stosw
    pop cx
```

```
    add cx, 1
    pop di
    add di, 160
    sub dx, 1
    cmp dx, 0
    je next
    jmp loop4

next:
    ; Calculate Starting Point
    mov ax, 0
    mov al, 80
    mul byte [bp + 6]
    add ax, [bp + 8]
    shl ax, 1
    mov di, ax

    mov cx, 1
    mov dx, [bp + 4]

    mov ax, 0xb800
    mov es, ax

    mov al, 0x2B
    mov ah, 0x04

    std

loop5:
    push di
    push cx
    rep stosw
    pop cx
    add cx, 1
    pop di
    add di, 160
    sub dx, 1
    cmp dx, 0
    je done3
    jmp loop5

done3:
    pop di
    pop dx
    pop cx
```

```
    pop bx
    pop ax
    pop bp
    ret 6

printTri:
    push bp
    mov bp, sp
    push ax

    mov ax, [bp + 10]
    cmp ax, 1
    je draw1
    cmp ax, 2
    je draw2
    cmp ax, 3
    je draw3

draw1:
    mov ax, [bp + 8]
    push ax
    mov ax, [bp + 6]
    push ax
    mov ax, [bp + 4]
    push ax

    call drawTri1
    jmp done

draw2:
    mov ax, [bp + 8]
    push ax
    mov ax, [bp + 6]
    push ax
    mov ax, [bp + 4]
    push ax

    call drawTri2
    jmp done

draw3:
    mov ax, [bp + 8]
    push ax
    mov ax, [bp + 6]
    push ax
```

```
    mov ax, [bp + 4]
    push ax

    call drawTri3

done:
    pop ax
    pop bp
    ret 8

Start:
    call clrscr

    mov bx, 0

loop1:
    push word [type + bx]
    push word [x + bx]
    push word [y + bx]
    push word [height + bx]

    call printTri
    call Delay
    call clrscr

    add bx, 2
    cmp bx, 4
    ja End
    jmp loop1

End:
    mov ax, 0x4c00
    int 0x21
```

Draw Rectangle:

```
[org 0x0100]
jmp Start

top: dw 5
bottom: dw 15
left: dw 30
right: dw 70
```

```
drawRect:
   push bp
   mov bp, sp
   push ax
   push bx
   push cx
   push di
   push si

   mov ax, 0xb800
   mov es, ax

   ; Calculating the Top Left position
   mov al, 80
   mul byte [bp + 10]
   add ax, [bp + 6]
   shl ax, 1
   mov di, ax
   push di

   ; Calculating the Width
   mov cx, [bp + 4]
   sub cx, [bp + 6]
   push cx

   ; Print the Top of Rectangle
   mov ah, 0x07
   mov al, 0x2B  ; +
   rep stosw

   pop bx
   pop di
   push bx
   dec bx
   shl bx, 1
   add di, 160

   ; Calculating the Height
   mov cx, [bp + 8]
   sub cx, [bp + 10]
   sub cx, 2

   mov al, 0x7C ; |

loop1:
```

```
   mov si, di
   mov word [es:si], ax
   add si, bx
   mov word [es:si], ax
   sub si, bx
   add di, 160
   loop loop1

   pop cx

   ; Print the Top of Rectangle
   mov ah, 0x07
   mov al, 0x2D  ; -
   rep stosw

   pop si
   pop di
   pop cx
   pop bx
   pop ax
   pop bp
   ret 8

Start:
   call clrscr

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

   call drawRect

End:
   mov ax, 0x4c00
   int 0x21
```

Remove Duplicates from Screen:

```
[org 0x0100]
jmp Start

fillscr:
    push es
```

```
    push ax
    push di

    mov ax, 0xb800
    mov es, ax
    mov di, 0
    mov ah, 0x07
    mov al, 97

nextlocation:
    mov word [es:di], ax
    add al, 1
    cmp al, 104
    je repeat
    add di, 2
    cmp di, 4000
    jne nextlocation
    jmp donefill

repeat:
    mov al, 97
    add di, 2
    cmp di, 4000
    jne nextlocation

donefill:
    pop di
    pop ax
    pop es
    ret

removeDuplicates:
    push bp
    mov bp, sp
    push ax
    push bx
    push cx
    push dx
    push si
    push di
    push es

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

```
    mov ax, 0

    mov ax, 80
    mov bx, [bp + 6]
    sub bx, 1
    mul bl
    mov bx, [bp + 4]
    sub bx, 1
    add ax, bx
    mov bx, ax
    shl bx, 1
    mov di, bx
    mov bx, [es:di]

    mov cx, [bp + 6]
    sub cx, 1

    mov dx, 0

    mov ax, 80
    mul cl
    mov di, ax
    add di, 0
    shl di, 1

    mov dx, ax
    add dx, 79
    shl dx, 1

loop1:
    mov ax, 0
    mov ax, [es:di]

    cmp al, bl
    je duplicate
    add di, 2
    cmp di, dx
    je done
    jmp loop1

duplicate:
    mov si, di
loop2:
    add si, 2
    mov ax, [es:si]
```

```
    sub si, 2
    mov [es:si], ax
    add si, 2
    cmp si, dx
    jne loop2
    mov ax, ' '
    mov [es:si], ax
    add di, 2
    cmp di, dx
    je done
    jmp loop1

done:
    pop es
    pop di
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    pop bp
    ret 4

Start:
    call clrscr
    call fillscr

    mov ax, 0
    int 0x16

    mov ax, 1 ; 1st row
    push ax
    mov bx, 1 ; 1st column
    push bx
    call removeDuplicates

    mov ax, 0
    int 0x16

End:
    mov ax, 0x4c00
    int 0x21
```

Remove Punctutation from String:

```
[org 0x0100]
jmp Start

string: db "Mr. Ali, Usman, & Anwar! Doing what???? want to travel????", 0

strlen:
    push bp
    mov bp, sp
    push es
    push cx
    push di
    les di, [bp+4]
    mov cx, 0xffff
    xor al, al
    repne scasb
    mov ax, 0xffff
    sub ax, cx
    dec ax
    pop di
    pop cx
    pop es
    pop bp
    ret 4

removehelper:
    push bp
    mov bp, sp
    push ax
    push bx
    push cx
    push dx
    push si
    push di

    mov si, [bp + 6]
    mov cx, [bp + 4]
    sub cx, 1

    mov dx, 0
    mov ax, 0

loop1:
    mov bx, dx
    mov al, [si + bx]
```

```
    cmp al, [bp + 8]
    je punctuation
    jmp loop1

punctuation:
    mov al, [si + bx + 1]
    mov [si + bx], al
    add bx, 1
    cmp bx, cx
    jne punctuation
    mov byte [si + bx], 0
    sub cx, 1
    add dx, 1
    cmp dx, cx
    je done
    jmp loop1

done:
    pop di
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    pop bp
    ret 6

removePunctuation:
    push bp
    mov bp, sp
    push ax
    push bx
    push cx
    push dx
    push si
    push di

    push ds
    mov ax, [bp+4]
    push ax
    call strlen

    cmp ax, 0
    jz doneremoving
```

```
    push ax
    mov bx, [bp+4]
    push bx
    mov cx, ','
    push cx
    call removehelper

    push ax
    mov bx, [bp+4]
    push bx
    mov cx, '?'
    push cx
    call removehelper

    push ax
    mov bx, [bp+4]
    push bx
    mov cx, ' '
    push cx
    call removehelper

    push ax
    mov bx, [bp+4]
    push bx
    mov cx, '!'
    push cx
    call removehelper

    push ax
    mov bx, [bp+4]
    push bx
    mov cx, '&'
    push cx
    call removehelper

doneremoving:
    pop di
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    pop bp
    ret 2
```

```
Start:
    mov ax, string
    push ax
    call removePunctuation

End:
    mov ax, 0x4c00
    int 0x21
```

Reverse String:

```
[org 0x0100]
jmp Start

string: db "I am Mr X", 0

strlen:
    push bp
    mov bp, sp
    push es
    push cx
    push di
    les di, [bp+4]
    mov cx, 0xffff
    xor al, al
    repne scasb
    mov ax, 0xffff
    sub ax, cx
    dec ax
    pop di
    pop cx
    pop es
    pop bp
    ret 4

reverseString:
    push bp
    mov bp, sp
    push ax
    push bx
    push cx
    push dx
    push si
    push di
```

```
    push ds
    mov ax, [bp+4]
    push ax
    call strlen

    cmp ax, 0
    jz donereversing

    mov cx, ax
    mov dx, ax

    mov si, [bp+4]

    mov bx, 0

loop1:
    mov ax, [si + bx]
    push ax
    dec cx
    cmp cx, 0
    je next
    inc bx
    jmp loop1

next:
    mov si, string2

    mov bx, 0

loop2:
    pop ax
    mov [si + bx], ax
    dec dx
    cmp dx, 0
    je donereversing
    inc bx
    jmp loop2

donereversing:
    inc bx
    mov byte [si + bx], 0

    pop di
    pop si
```

```
    pop dx
    pop cx
    pop bx
    pop ax
    pop bp
    ret 2

Start:

    mov ax, string
    push ax
    call reverseString

End:
    mov ax, 0x4c00
    int 0x21

string2: db 0
```

Find Vowels in a String and Print Data on Screen:

```
[org 0x0100]
jmp Start

string: db "Mr. Ali, Usman, & Anwar! Doing what???? want to travel????", 0
message1: db "Vowel", 0
message2: db "Count", 0
Vowel1: db "a or A", 0
Vowel2: db "e or E", 0
Vowel3: db "i or I", 0
Vowel4: db "o or O", 0
Vowel5: db "u or U", 0

clrscr:
    push es
    push ax
    push cx
    push di

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

```
    cld
    rep stosw

    pop di
    pop cx
    pop ax
    pop es
    ret

printstr:
    push bp
    mov bp, sp
    push es
    push ax
    push cx
    push si
    push di

    push ds
    mov ax, [bp+4]
    push ax
    call strlen

    cmp ax, 0
    jz exit

    mov cx, ax
    mov ax, 0xb800
    mov es, ax
    mov al, 80
    mul byte [bp+8]
    add ax, [bp+10]
    shl ax, 1
    mov di, ax
    mov si, [bp+4]
    mov ah, [bp+6]
    cld

nextchar:
    lodsb
    stosw
    loop nextchar

exit:
    pop di
```

```
    pop si
    pop cx
    pop ax
    pop es
    pop bp
    ret 8

printnum:
    push bp
    mov bp, sp
    push es
    push ax
    push bx
    push cx
    push dx
    push di

    mov ax, 0xb800
    mov es, ax

    mov ax, [bp+4] ; Load the number to be printed into AX
    mov bx, 10
    mov cx, 0

nextdigit:
    mov dx, 0
    div bx         ; Divide AX by 10
    add dl, 0x30
    push dx

    inc cx
    cmp ax, 0
    jnz nextdigit

    mov al, 80
    mul byte [bp+6]
    add ax, [bp+8]
    shl ax, 1
    mov di, ax

nextpos:
    pop dx
    mov dh, 0x07
    mov [es:di], dx
    add di, 2
```

```
    loop nextpos

    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    pop es
    pop bp
    ret 6

strlen:
    push bp
    mov bp, sp
    push es
    push cx
    push di

    les di, [bp+4]
    mov cx, 0xffff
    xor al, al
    repne scasb
    mov ax, 0xffff
    sub ax, cx
    dec ax

    pop di
    pop cx
    pop es
    pop bp
    ret 4

counthelper:
    push bp
    mov bp, sp
    push ax
    push bx
    push cx
    push si
    push di

    mov si, [bp + 6]
    mov cx, [bp + 8]
    dec cx
    mov di, [bp + 4]
```

```
    add di, 0x20

    mov ax, 0
    mov bx, 0
    mov dx, 0

loop1:
    mov al, [si + bx]
    cmp ax, [bp + 4]
    je count
    cmp ax, di
    je count
    inc bx
    cmp bx, cx
    je done
    jmp loop1

count:
    inc dx
    inc bx
    cmp bx, cx
    je done
    jmp loop1

done:
    pop di
    pop si
    pop cx
    pop bx
    pop ax
    pop bp
    ret 6

countVowels:
    push bp
    mov bp, sp
    push ax
    push bx
    push cx
    push dx
    push si
    push di

    push ds
    mov ax, [bp+4]
```

```
    push ax
    call strlen

    push ax

    push ax
    mov bx, [bp+4]
    push bx
    mov cx, 0x41
    push cx
    call counthelper

    mov ax, 30
    push ax                 ; Push x position
    mov ax, 11
    push ax                 ; Push y position
    mov ax, 0x07
    push ax
    mov ax, Vowel1
    push ax                 ; Push address of message
    call printstr

    mov ax, 40
    push ax                 ; Push x position
    mov ax, 11
    push ax                 ; Push y position
    mov ax, dx
    push ax
    call printnum

    pop ax
    push ax

    push ax
    mov bx, [bp+4]
    push bx
    mov cx, 0x45
    push cx
    call counthelper

    mov ax, 30
    push ax                 ; Push x position
    mov ax, 12
    push ax                 ; Push y position
    mov ax, 0x07
```

```
    push ax
    mov ax, Vowel2
    push ax                 ; Push address of message
    call printstr

    mov ax, 40
    push ax                 ; Push x position
    mov ax, 12
    push ax                 ; Push y position
    mov ax, dx
    push ax
    call printnum

    pop ax
    push ax

    push ax
    mov bx, [bp+4]
    push bx
    mov cx, 0x49
    push cx
    call counthelper

    mov ax, 30
    push ax                 ; Push x position
    mov ax, 13
    push ax                 ; Push y position
    mov ax, 0x07
    push ax
    mov ax, Vowel3
    push ax                 ; Push address of message
    call printstr

    mov ax, 40
    push ax                 ; Push x position
    mov ax, 13
    push ax                 ; Push y position
    mov ax, dx
    push ax
    call printnum

    pop ax
    push ax

    push ax
```

```
    mov bx, [bp+4]
    push bx
    mov cx, 0x4F
    push cx
    call counthelper

    mov ax, 30
    push ax                 ; Push x position
    mov ax, 14
    push ax                 ; Push y position
    mov ax, 0x07
    push ax
    mov ax, Vowel4
    push ax                 ; Push address of message
    call printstr

    mov ax, 40
    push ax                 ; Push x position
    mov ax, 14
    push ax                 ; Push y position
    mov ax, dx
    push ax
    call printnum

    pop ax

    push ax
    mov bx, [bp+4]
    push bx
    mov cx, 0x55
    push cx
    call counthelper

    mov ax, 30
    push ax                 ; Push x position
    mov ax, 15
    push ax                 ; Push y position
    mov ax, 0x07
    push ax
    mov ax, Vowel5
    push ax                 ; Push address of message
    call printstr

    mov ax, 40
    push ax                 ; Push x position
```

```
    mov ax, 15
    push ax                 ; Push y position
    mov ax, dx
    push ax
    call printnum

donecountVowels:
    pop di
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    pop bp
    ret 2

Start:
    call clrscr

    mov ax, 30
    push ax                 ; Push x position
    mov ax, 10
    push ax                 ; Push y position
    mov ax, 0x07
    push ax
    mov ax, message1
    push ax                 ; Push address of message
    call printstr

    mov ax, 40
    push ax                 ; Push x position
    mov ax, 10
    push ax                 ; Push y position
    mov ax, 0x07
    push ax
    mov ax, message2
    push ax                 ; Push address of message
    call printstr

    mov ax, string
    push ax
    call countVowels

End:
```

```
    mov ax, 0x4c00
    int 0x21
```

Set Cursor Position:

```
mov ah, 02h
mov bh, 0
mov dh, 10 ; Row 10
mov dl, 20 ; Column 20
int 10h
```

Write String at Cursor Position (In Default Colour):

```
mov dx, msg
mov ah, 09h
int 21h


msg: db 'Hello!','$'
```

Write String (In Any Colour):

```
message: db 'Hello World'

start:
 mov ah, 13h    ; service 13 - print string
 mov al, 1      ; subservice 01 – update cursor
 mov bh, 0      ; output on page 0
 mov bl, 07h    ; normal attrib
 mov dx, 0A03h  ; row 10 column 3
 mov cx, 11     ; length of string
 push cs
 pop es         ; segment of string
 mov bp, message ; offset of string
 int 10h       ; call BIOS video service

   mov ah, 0x13
   mov al, 1
   mov bh, 0
   mov bl, 0x07          ; attribute
   mov dh, [asterikRow]
   mov dl, [asterikCol]
   push ax
```

```
   push ds
   mov ax, asterik
   push ax
   call strlen
   mov cx, ax            ; length of string
   pop ax
   push cs
   pop es
   mov bp, asterik       ; offset of string
   int 0x10
```

Take Input:

```
mov si, userInput

input:
    mov ah, 1
    int 21h

    cmp al, 13
    je end

    mov [si], al
    inc si

    jmp input

end:
  mov byte [si], 0


userInput: db 0
```

Wait for any Key press:

```
mov ah, 0
int 16h

; Keystroke is saved in 'al'
```

Hardware Interrupt Hooking:

```
[org 0x0100]
jmp Start

oldkb: dd 0

kbisr:
   push ax
   in al, 0x60
   cmp al, 0x2a
   jne nextcmp

   cmp word [cs:timerflag], 1
   je exit

   mov word [cs:timerflag], 1
   jmp exit

nextcmp:
   cmp al, 0xaa
   jne nomatch

   mov word [cs:timerflag], 0
   jmp exit

nomatch:
   pop ax
   jmp far [cs:oldkb]

exit:
   mov al, 0x20
   out 0x20, al

   pop ax
   iret

timer:
   push ax

   cmp word [cs:timerflag], 1
   jne skipall

   inc word [cs:seconds]
   push word [cs:seconds]
   call printnum
```

```
skipall:
   mov al, 0x20
   out 0x20, al

   pop ax
   iret

Start:
   xor ax, ax
   mov es, ax
   mov ax, [es:9*4]
   mov [oldkb], ax
   mov ax, [es:9*4+2]
   mov [oldkb+2], ax
   cli
   mov word [es:9*4], kbisr
   mov [es:9*4+2], cs
   mov word [es:8*4], timer
   mov [es:8*4+2], cs
   sti

l1:
   mov ah, 0
   int 0x16

   cmp al, 27
   jne l1

   mov ax, [oldisr]
   mov bx, [oldisr+2]
   cli
   mov [es:9*4], ax
   mov [es:9*4+2], bx
   sti

   mov ax, 0x4c00
   int 0x21
```

Terminate and Resist:

```
[org 0x0100]
jmp Start
```

```
oldkb: dd 0

kbisr:
   push ax
   in al, 0x60
   cmp al, 0x2a
   jne nextcmp

   cmp word [cs:timerflag], 1
   je exit

   mov word [cs:timerflag], 1
   jmp exit

nextcmp:
   cmp al, 0xaa
   jne nomatch

   mov word [cs:timerflag], 0
   jmp exit

nomatch:
   pop ax
   jmp far [cs:oldkb]

exit:
   mov al, 0x20
   out 0x20, al

   pop ax
   iret

timer:
   push ax

   cmp word [cs:timerflag], 1
   jne skipall

   inc word [cs:seconds]
   push word [cs:seconds]
   call printnum

skipall:
   mov al, 0x20
   out 0x20, al
```

```
   pop ax
   iret

Start:
   xor ax, ax
   mov es, ax
   mov ax, [es:9*4]
   mov [oldkb], ax
   mov ax, [es:9*4+2]
   mov [oldkb+2], ax
   cli
   mov word [es:9*4], kbisr
   mov [es:9*4+2], cs
   mov word [es:8*4], timer
   mov [es:8*4+2], cs
   sti

   mov dx, start
   add dx, 15
   mov cl, 4
   shr dx, cl
   mov ax, 0x3100
   int 0x21
```

Asterisk Game:

```
[org 0x0100]
jmp Start

asterik: db '*', 0
asterikRow: db 12
asterikCol: db 40

upFlag: dw 1
downFlag: dw 0
leftFlag: dw 0
rightFlag: dw 0

seconds: dw 0

oldkb: dd 0
oldtime: dd 0
```

```
clrscr:
    push es
    push ax
    push cx
    push di

    mov ax, 0xb800
    mov es, ax
    xor di, di
    mov ax, 0x0720
    mov cx, 2000
    cld
    rep stosw

    pop di
    pop cx
    pop ax
    pop es
    ret

strlen:
    push bp
    mov bp, sp
    push es
    push cx
    push di

    les di, [bp+4]
    mov cx, 0xffff
    xor al, al
    repne scasb
    mov ax, 0xffff
    sub ax, cx
    dec ax

    pop di
    pop cx
    pop es
    pop bp
    ret 4

printnum:
    push bp
    mov bp, sp
    push es
```

```
    push ax
    push bx
    push cx
    push dx
    push di

    mov ax, 0xb800
    mov es, ax

    mov ax, [bp+4] ; Load the number to be printed into AX
    mov bx, 10
    mov cx, 0

nextdigit:
    mov dx, 0
    div bx         ; Divide AX by 10
    add dl, 0x30
    push dx

    inc cx
    cmp ax, 0
    jnz nextdigit

    mov al, 80
    mul byte [bp+6]
    add ax, [bp+8]
    shl ax, 1
    mov di, ax

nextpos:
    pop dx
    mov dh, 0x07
    mov [es:di], dx
    add di, 2
    loop nextpos

    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    pop es
    pop bp
    ret 6
```

```
PrintAsterik:
   push bp
   push ax
   push bx
   push cx
   push dx

   mov ah, 0x13
   mov al, 0
   mov bh, 0
   mov bl, 0x07          ; attribute
   mov dh, [asterikRow]
   mov dl, [asterikCol]
   push ax
   push ds
   mov ax, asterik
   push ax
   call strlen
   mov cx, ax            ; length of string
   pop ax
   push cs
   pop es
   mov bp, asterik       ; offset of string
   int 0x10

   pop dx
   pop cx
   pop bx
   pop ax
   pop bp
   ret

movUp:
   dec byte[asterikRow]
   call PrintAsterik
   ret

movDown:
   inc byte[asterikRow]
   call PrintAsterik
   ret

movLeft:
   dec byte[asterikCol]
   call PrintAsterik
```

```
   ret

movRight:
   inc byte[asterikCol]
   call PrintAsterik
   ret

Check:
   cmp byte[asterikRow], 25
   jge EndGame

   cmp byte[asterikRow], -1
   jle EndGame

   cmp byte[asterikCol], 80
   jge EndGame

   cmp byte[asterikRow], -1
   jle EndGame

   jmp Continue

EndGame:
   call clrscr

   mov ax, 40
   push ax                 ; Push x position
   mov ax, 12
   push ax                 ; Push y position
   push word [cs:seconds]
   call printnum

   mov ax, [oldkb]
   mov bx, [oldkb+2]
   mov cx, [oldtime]
   mov dx, [oldtime+2]
   cli
   mov [es:9*4], ax
   mov [es:9*4+2], bx
   mov [es:8*4], cx
   mov [es:8*4+2], dx
   sti

   mov ax, 0x4c00
   int 0x21
```

```
Continue:
   ret

kbisr:
   push ax
   in al, 0x60

Upkey:
   cmp al, 0x48
   jne Downkey

   mov word [cs:upFlag], 1
   mov word [cs:downFlag], 0
   mov word [cs:leftFlag], 0
   mov word [cs:rightFlag], 0
   jmp exit

Downkey:
   cmp al, 0x50
   jne Leftkey

   mov word [cs:upFlag], 0
   mov word [cs:downFlag], 1
   mov word [cs:leftFlag], 0
   mov word [cs:rightFlag], 0
   jmp exit

Leftkey:
   cmp al, 0x4B
   jne Rightkey

   mov word [cs:upFlag], 0
   mov word [cs:downFlag], 0
   mov word [cs:leftFlag], 1
   mov word [cs:rightFlag], 0
   jmp exit

Rightkey:
   cmp al, 0x4D
   jne nomatch

   mov word [cs:upFlag], 0
   mov word [cs:downFlag], 0
   mov word [cs:leftFlag], 0
```

```
   mov word [cs:rightFlag], 1
   jmp exit

nomatch:
   pop ax
   jmp far [cs:oldkb]

exit:
   mov al, 0x20
   out 0x20, al

   pop ax
   iret

timer:
   push ax

   call clrscr

   inc word [cs:seconds]
   mov ax, 2
   push ax                 ; Push x position
   mov ax, 2
   push ax                 ; Push y position
   push word [cs:seconds]
   call printnum

   cmp word [cs:upFlag], 1
   jne next1

   call movUp
   jmp doneinterrupt

next1:
   cmp word [cs:downFlag], 1
   jne next2

   call movDown
   jmp doneinterrupt

next2:
   cmp word [cs:leftFlag], 1
   jne next3

   call movLeft
```

```
   jmp doneinterrupt

next3:
   cmp word [cs:rightFlag], 1
   jne doneinterrupt

   call movRight

doneinterrupt:
   call Check

   mov al, 0x20
   out 0x20, al

   pop ax
   iret

Start:
   call clrscr
   call PrintAsterik

   mov ah, 0
   int 16h

   xor ax, ax
   mov es, ax
   mov ax, [es:9*4]
   mov [oldkb], ax
   mov ax, [es:9*4+2]
   mov [oldkb+2], ax
   mov ax, [es:8*4]
   mov [oldtime], ax
   mov ax, [es:8*4+2]
   mov [oldtime+2], ax
   cli
   mov word [es:9*4], kbisr
   mov [es:9*4+2], cs
   mov word [es:8*4], timer
   mov [es:8*4+2], cs
   sti

l1:
   jmp l1
```

Weird Number:

```
[org 0x0100]
jmp Start

message1: db 'Enter a Number: ', 0
message2: db 'NOT A WEIRD NUMBER', 0
message3: db 'WEIRD NUMBER', 0

userInput: dw 0

clrscr:
    push es
    push ax
    push cx
    push di

    mov ax, 0xb800
    mov es, ax
    xor di, di
    mov ax, 0x0720
    mov cx, 2000
    cld
    rep stosw

    pop di
    pop cx
    pop ax
    pop es
    ret

strlen:
    push bp
    mov bp, sp
    push es
    push cx
    push di

    les di, [bp+4]
    mov cx, 0xffff
    xor al, al
    repne scasb
    mov ax, 0xffff
    sub ax, cx
    dec ax
```

```
    pop di
    pop cx
    pop es
    pop bp
    ret 4

printnum:
    push bp
    mov bp, sp
    push es
    push ax
    push bx
    push cx
    push dx
    push di

    mov ax, 0xb800
    mov es, ax

    mov ax, [bp+4] ; Load the number to be printed into AX
    mov bx, 10
    mov cx, 0

nextdigit:
    mov dx, 0
    div bx         ; Divide AX by 10
    add dl, 0x30
    push dx

    inc cx
    cmp ax, 0
    jnz nextdigit

    mov al, 80
    mul byte [bp+6]
    add ax, [bp+8]
    shl ax, 1
    mov di, ax

nextpos:
    pop dx
    mov dh, 0x07
    mov [es:di], dx
    add di, 2
```

```
    loop nextpos

    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    pop es
    pop bp
    ret 6

UserInput:
   push bp
   push ax
   push bx
   push cx
   push dx

   mov ah, 0x13
   mov al, 1
   mov bh, 0
   mov bl, 0x07          ; attribute
   mov dh, 12
   mov dl, 36
   push ax
   push ds
   mov ax, message1
   push ax
   call strlen
   mov cx, ax            ; length of string
   pop ax
   push cs
   pop es
   mov bp, message1       ; offset of string
   int 0x10

   mov bx, userInput

   inputloop:
      mov ah, 0
      int 0x16

      cmp al, 13
      je endInput
```

```
      sub al, 0x30
      mov cx, 10
      mov dx, 0
      mov dl, al
      mov ax, 0
      mov ax, [bx]
      mul cl
      add ax, dx
      mov [bx], ax

      jmp inputloop

endInput:
   pop dx
   pop cx
   pop bx
   pop ax
   pop bp
   ret

WeirdNumber:
   push ax
   push bx
   push cx
   push dx

   mov bx, [userInput]
   dec bx
   mov cx, 0

loop1:
   mov ax, [userInput]
   mov dx, 0
   div bx

   cmp dx, 0
   jne nextCheck

   push bx
   inc cx

nextCheck:
   dec bx
   cmp bx, 0
   je nextStep1
```

```
   jmp loop1

nextStep1:
   mov dx, 0
loop2:
   pop ax
   add dx, ax
   dec cx
   cmp cx, 0
   je nextStep2
   jmp loop2

nextStep2:
   cmp dx, [userInput]
   jg greater

   mov si, 1
   jmp doneChecking

greater:
   mov si, 0

doneChecking:
   pop dx
   pop cx
   pop bx
   pop ax
   ret

Start:
   call clrscr
   call UserInput

   mov ax, 36
   push ax                 ; Push x position
   mov ax, 13
   push ax                 ; Push y position
   push word [userInput]
   call printnum

   call WeirdNumber

   cmp si, 1
   jne Weird
```

```
   mov ah, 0x13
   mov al, 0
   mov bh, 0
   mov bl, 0x07          ; attribute
   mov dh, 15
   mov dl, 36
   push ax
   push ds
   mov ax, message2
   push ax
   call strlen
   mov cx, ax            ; length of string
   pop ax
   push cs
   pop es
   mov bp, message2       ; offset of string
   int 0x10

   jmp End

Weird:
   mov ah, 0x13
   mov al, 0
   mov bh, 0
   mov bl, 0x07          ; attribute
   mov dh, 15
   mov dl, 36
   push ax
   push ds
   mov ax, message3
   push ax
   call strlen
   mov cx, ax            ; length of string
   pop ax
   push cs
   pop es
   mov bp, message3       ; offset of string
   int 0x10

End:
   mov ax, 0x4c00
   int 0x21
```

Screen Saver:

```
[org 0x0100]
jmp Start

microseconds: dw 0
seconds: dw 0

message1: db 'Screen Saver!', 0

oldkb: dd 0

memory: times 2000 dw 0

flag: dw 0

clrscr:
    push es
    push ax
    push cx
    push di

    mov ax, 0xb800
    mov es, ax
    xor di, di
    mov ax, 0x0720
    mov cx, 2000
    cld
    rep stosw

    pop di
    pop cx
    pop ax
    pop es
    ret

strlen:
    push bp
    mov bp, sp
    push es
    push cx
    push di

    les di, [bp+4]
    mov cx, 0xffff
    xor al, al
```

```
    repne scasb
    mov ax, 0xffff
    sub ax, cx
    dec ax

    pop di
    pop cx
    pop es
    pop bp
    ret 4

Copy:
   push bp
   mov bp, sp
   push ax
   push bx
   push cx
   push si
   push di
   push es
   push ds

   push ds
   pop es
   mov ax, 0xb800
   mov ds, ax

   mov di, memory
   mov si, [bp + 8]   ; Starting Point
   mov bx, 0

   cld

   Copyloop1:
      mov cx, [bp + 4]  ; Number of Columns
      push si
      rep movsw
      inc bx
      cmp bx, [bp + 6]  ; Number of Rows
      je doneCopy
      pop si
      add si, 160
      jmp Copyloop1

doneCopy:
```

```
   pop si
   pop ds
   pop es
   pop di
   pop si
   pop cx
   pop bx
   pop ax
   pop bp
   ret 6

Paste:
   push bp
   mov bp, sp
   push ax
   push bx
   push cx
   push si
   push di
   push es
   push ds

   mov ax, 0xb800
   mov es, ax

   mov si, memory
   mov di, [bp + 8]   ; Starting Point
   mov bx, 0

   cld

   Pasteloop1:
      mov cx, [bp + 4]  ; Number of Columns
      push di
      rep movsw
      inc bx
      cmp bx, [bp + 6]  ; Number of Rows
      je donePaste
      pop di
      add di, 160
      jmp Pasteloop1

donePaste:
   pop si
   pop ds
```

```
   pop es
   pop di
   pop si
   pop cx
   pop bx
   pop ax
   pop bp
   ret 6

kbisr:
   push ax
   in al, 0x60

   cmp word [cs:flag], 0
   je exit

   mov word [cs:flag], 0

   mov ax, 0  ; Starting Point
   push ax
   mov ax, 25    ; Number of Rows to Copy
   push ax
   mov ax, 80    ; Number of Columns to Copy
   push ax
   call Paste

exit:
   mov al, 0x20
   out 0x20, al

   pop ax
   iret

timer:
   push ax

   cmp word [cs:flag], 1
   je skipall

   inc word [cs:microseconds]
   cmp word [cs:microseconds], 15
   jbe skipall

   mov word [cs:microseconds], 0
```

```
   inc word [cs:seconds]
   cmp word [cs:seconds], 10
   jbe skipall

   mov word [cs:flag], 1

   mov word [cs:seconds], 0

   mov ax, 0  ; Starting Point
   push ax
   mov ax, 25    ; Number of Rows to Copy
   push ax
   mov ax, 80    ; Number of Columns to Copy
   push ax
   call Copy

   call clrscr

   mov ah, 0x13
   mov al, 0
   mov bh, 0
   mov bl, 0x07          ; attribute
   mov dh, 12
   mov dl, 30
   push ax
   push ds
   mov ax, message1
   push ax
   call strlen
   mov cx, ax            ; length of string
   pop ax
   push cs
   pop es
   mov bp, message1       ; offset of string
   int 0x10

skipall:
   mov al, 0x20
   out 0x20, al

   pop ax
   iret

Start:
   xor ax, ax
```

```
   mov es, ax
   mov ax, [es:9*4]
   mov [oldkb], ax
   mov ax, [es:9*4+2]
   mov [oldkb+2], ax
   cli
   mov word [es:9*4], kbisr
   mov [es:9*4+2], cs
   mov word [es:8*4], timer
   mov [es:8*4+2], cs
   sti

l1:
   jmp l1
```