UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

ENTRADA SALIDA ENSAMBLADOR LINUX DE 32 BITS

LAURA JULIANA CÁRDENAS SÁNCHEZ
CÓDIGO ESTUDIANTIL: 20231578062

JUAN DAVID ORTIZ GORDILLO
CÓDIGO ESTUDIANTIL: 20231578110

DOCENTE JAVIER ORLANDO DAZA TORRES

ARQUITECTURA DE COMPUTADORES

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
SEDE TECNOLÓGICA

BOGOTÁ D.C

24 DE NOVIEMBRE DE 2025

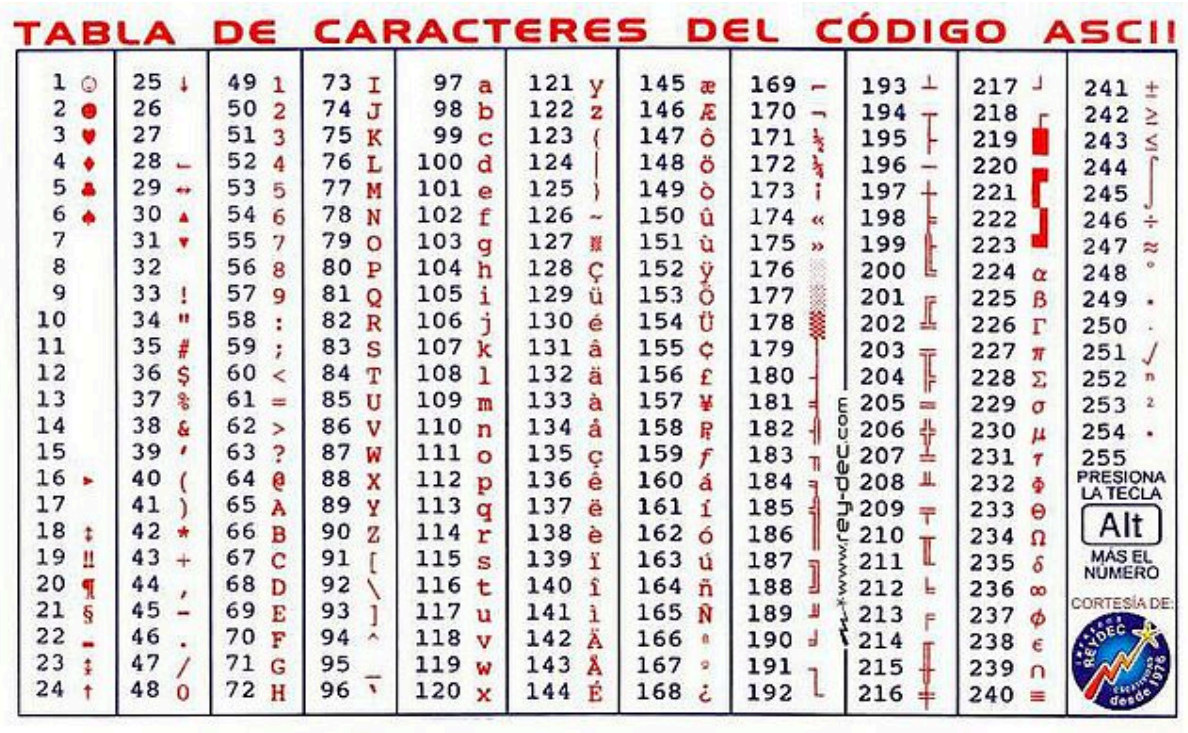
Actividades

1. Modificar el mensaje para que diga “Ingrese una letra:”.

2. Agregar validación: si el carácter es una vocal, mostrar “Es una vocal”.

3. Leer una cadena completa (usa un buffer de mayor tamaño y cambia edx).

4. Mostrar el código ASCII del carácter leído.




```
section .data
    mensaje db "Ingrese una letra: ",0
    eco db 10,"Tecla presionada: ",0

    vocal_msg db 10,"Es una vocal",10,0
    no_vocal_msg db 10,"No es vocal",10,0

    ascii_msg db "Codigo ASCII: ",0

    nl db 10     ; salto de línea

section .bss
    buffer resb 100       ; buffer para leer cadena
```

```
    numbuf resb 10          ; buffer para número ASCII
```

- buffer = donde guardamos lo que escribe el usuario
- numbuf = para almacenar los dígitos del ASCII convertido a texto

```
section .text
    global _start

_start:

    ; Mostrar mensaje inicial
    mov eax, 4
    mov ebx, 1
    mov ecx, mensaje
    mov edx, 20
    int 0x80
```

- eax = 4 → syscall write()
- ebx = 1 → stdout stdout (Salida estándar (lo que se imprime en pantalla))
- ecx = dirección del mensaje
- edx = longitud
- int 0x80 = llamar al kernel Linux para hacer la syscall.

```
    ; Leer cadena (pero solo usamos el primer carácter)
    mov eax, 3
    mov ebx, 0
    mov ecx, buffer
    mov edx, 100
    int 0x80
```

- eax = 3 → syscall read()
- ebx = 0 → stdout (Entrada estándar (lo que se escribe en teclado))
- ecx = dirección al buffer
- edx = longitud
- int 0x80 = llamar al kernel Linux para hacer la syscall.

```
    ; Eco
    mov eax, 4
```

```
mov ebx, 1

mov ecx, eco

mov edx, 20

int 0x80
```

- eax = 4 → syscall write()
- ebx = 1 → stdout stdout (Salida estándar (lo que se imprime en pantalla))
- ecx = dirección del mensaje
- edx = longitud
- int 0x80 = llamar al kernel Linux para hacer la syscall.

; Mostrar primer carácter

```
mov eax, 4

mov ebx, 1

mov ecx, buffer

mov edx, 1

int 0x80
```

- eax = 4 → syscall write()
- ebx = 1 → stdout stdout (Salida estándar (lo que se imprime en pantalla))
- ecx = dirección del buffer
- edx = longitud
- int 0x80 = llamar al kernel Linux para hacer la syscall.

; Validar vocal (may/min)

```
mov al, [buffer]
```

- Cargamos la letra a al

; Vocales minúsculas

```
cmp al, 'a'

je es_vocal

cmp al, 'e'

je es_vocal

cmp al, 'i'

je es_vocal

cmp al, 'o'

je es_vocal
```

```
    cmp al, 'u'
    je es_vocal

    ; Vocales mayúsculas
    cmp al, 'A'
    je es_vocal
    cmp al, 'E'
    je es_vocal
    cmp al, 'I'
    je es_vocal
    cmp al, 'O'
    je es_vocal
    cmp al, 'U'
    je es_vocal

    jmp no_es_vocal
```

- cmp X, Y -> compara
- je (jump if equal) salta si son iguales
- Si coincide con alguna vocal → va a es_vocal
- Si no coincide → va a no_es_vocal

```
es_vocal:
    mov eax, 4
    mov ebx, 1
    mov ecx, vocal_msg
    mov edx, 14
    int 0x80
    jmp mostrar_ascii
```

- eax = 4 → syscall write()
- ebx = 1 → stdout stdout (Salida estándar (lo que se imprime en pantalla))
- ecx = dirección del mensaje
- edx = longitud
- int 0x80 = llamar al kernel Linux para hacer la syscall.

```
no_es_vocal:
    mov eax, 4
```

```
mov ebx, 1
mov ecx, no_vocal_msg
mov edx, 15
int 0x80
```

- eax = 4 → syscall write()
- ebx = 1 → stdout stdout (Salida estándar (lo que se imprime en pantalla))
- ecx = dirección del mensaje
- edx = longitud
- int 0x80 = llamar al kernel Linux para hacer la syscall.

```
; Mostrar código ASCII

mostrar_ascii:

    ; Mostrar texto "Codigo ASCII: "
    mov eax, 4
    mov ebx, 1
    mov ecx, ascii_msg
    mov edx, 14
    int 0x80
```

- eax = 4 → syscall write()
- ebx = 1 → stdout stdout (Salida estándar (lo que se imprime en pantalla))
- ecx = dirección del mensaje
- edx = longitud
- int 0x80 = llamar al kernel Linux para hacer la syscall.

```
    ; Convertir ASCII a texto
    movzx eax, byte [buffer]   ; cargar ASCII
    call int_to_ascii
```

- movzx = mueve y llena los bits superiores con ceros.

```
    ; Imprimir número guardado
    mov eax, 4
    mov ebx, 1
    mov ecx, numbuf
    mov edx, 3
```

int 0x80

- eax = 4 → syscall write()
- ebx = 1 → stdout stdout (Salida estándar (lo que se imprime en pantalla))
- ecx = dirección del section .bss
- edx = longitud
- int 0x80 = llamar al kernel Linux para hacer la syscall.

; Salto de línea final

mov eax, 4

mov ebx, 1

mov ecx, nl

mov edx, 1

int 0x80

- eax = 4 → syscall write()
- ebx = 1 → stdout (Salida estándar (lo que se imprime en pantalla))
- ecx = Salto de línea
- edx = longitud
- int 0x80 = llamar al kernel Linux para hacer la syscall.

; Salir

mov eax, 1

xor ebx, ebx

int 0x80

- eax = 1 → syscall exit()
- ebx = ebx → stdout (Código de salida)
- int 0x80 = llamar al kernel Linux para hacer la syscall.

; Convertir número entero → ASCII en numbuf

int_to_ascii:

mov ebx, numbuf

mov ecx, 0 **; contador**

- ebx = numbuf → Apunta al inicio del buffer donde guardaremos los dígitos.
- ecx = 0 → Contador de cuántos dígitos llevamos guardados.

convert_loop:

```
mov edx, 0
mov edi, 10
```

- Dividir entre 10, porque queremos obtener dígitos decimales.

```
div edi          ; eax / 10
```

Divide EAX entre 10:

- EAX = cociente → el número sin el último dígito
- EDX = residuo → el dígito que acabamos de extraer

```
add edx, '0'       ; residuo → dígito ASCII
```

- Convierte el dígito numérico en carácter ASCII.

```
mov [ebx + ecx], dl
```

- Guarda ese dígito ASCII en el buffer numbuf

```
inc ecx
test eax, eax
```

- Revisa si ya terminamos.
- Si EAX = 0, no quedan más dígitos.

```
jnz convert_loop
```

- Si aún hay números (EAX ≠ 0), vuelve a repetir el ciclo.

```
; Invertir los caracteres
mov esi, 0
dec ecx

rev_loop:
mov al, [ebx + esi]
```

- Lee el carácter del inicio.

```
mov ah, [ebx + ecx]
```

- Lee el carácter del final.

```
mov [ebx + esi], ah
```

- Pone el carácter final al inicio.

```
mov [ebx + ecx], al
```

- Pone el carácter inicial al final.

inc esi

dec ecx

cmp esi, ecx

- Compara los índices del inicio y del fin.

jl rev_loop

- Mientras el carácter inicial < carácter final, sigue invirtiendo.
- Cuando se cruzan → significa que ya está invertido.

ret

- ret devuelve el control a la instrucción siguiente al call

Explicación del ASCII

La manera de hacerlo es:

1. Divide el número entre 10
2. El residuo (0–9) se convierte a un carácter sumándole '0'
3. Guarda cada dígito
4. Repite hasta que se acaben los dígitos
5. Finalmente invierte los dígitos porque salieron al revés

Ejemplo para 267:

- 267 / 10 → 26, residuo 7 → '7'
- 26 / 10 → 2, residuo 6 → '6'
- 2 / 10 → 0, residuo 2 → '2'

Se guardan como "762" → se invierte → **"267"**