



ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

Organización de Computadores (CCPG1049)

Implementación de Sistema de Cifrado César en Assembly x86

Integrantes:

- Luis Emmanuel Vargas Silva
- Geovanny Jahir Diaz Loor

Mgtr. Freddy Ronald Veloz De La Torre

28 de noviembre de 2025

Guayaquil - Ecuador

Contents

1 Introducción	3
2 Marco Teórico	4
2.1 Concepto del Cifrado César	4
2.2 Desarrollo e Implementación	4
2.3 Estructura de Datos y Lectura	4
2.4 Lógica de Encriptación	5
2.5 Lógica de Desencriptación y Manejo de Negativos.....	6
2.6 Caracteres Especiales.....	6
3 Manual de Usuario y Pruebas	7
3.1 Menú Principal.....	7
3.2 Proceso de Cifrado	7
3.3 Proceso de Descifrado.....	8
4 Conclusiones	9
5 Referencias.....	10
6 Apéndice A: Código Fuente	11
6.1 Github:	11
6.2 Código	11

1 Introducción

El presente proyecto tiene como objetivo la implementación de un sistema de cifrado y descifrado de mensajes utilizando el algoritmo César, desarrollado en lenguaje ensamblador para la arquitectura 8086. Este desarrollo responde a los requisitos de la primera evaluación de la materia Organización de Computadores, buscando reforzar el aprendizaje en la manipulación de cadenas, bucles, operaciones aritméticas y la organización modular en procedimientos dentro del entorno de bajo nivel.

El programa permite al usuario interactuar mediante una consola, ofreciendo la capacidad de procesar mensajes de hasta 35 caracteres y aplicar un desplazamiento configurable de entre 1 y 5 posiciones. Para su ejecución y validación, se utiliza el emulador emu8086, garantizando la compatibilidad con las instrucciones del microprocesador 8086.

2 Marco Teórico

2.1 Concepto del Cifrado César

El cifrado César es una técnica de criptografía simple basada en la sustitución mono-alfabética.

El algoritmo consiste en desplazar cada letra del mensaje original un número fijo de posiciones en el alfabeto. Matemáticamente, el proceso se define mediante aritmética modular para asegurar un comportamiento circular (wrap-around), donde la letra 'Z' desplazada una posición se convierte en 'A'.

La fórmula general para el cifrado es:

$$C = (P + k) \bmod 26$$

Donde:

- C es la posición de la letra cifrada (0-25).
- P es la posición de la letra original.
- k es el desplazamiento o clave .

Para el descifrado, se aplica la operación inversa: $P = (C - k) \bmod 26$

En casos donde la resta resulta en un número negativo, se suma el módulo (26) para corregir la posición.

2.2 Desarrollo e Implementación

El sistema se desarrolló utilizando una estructura modular en Assembly 8086. A continuación, se detallan los componentes críticos del código y su lógica.

2.3 Estructura de Datos y Lectura

Se definió un buffer en el segmento de datos (.DATA) capaz de almacenar 35 caracteres más un terminador \$ requerido por las interrupciones de DOS. El control de la longitud del texto se

realiza mediante un contador en el registro CX, deteniendo la lectura si se presiona la tecla ENTER (ASCII 13) o si se alcanza el límite de caracteres.

Fragmento de código

; Fragmento de validación de longitud (*Ver Apéndice A*)

cmp cx, 35 ; limite de 35 caracteres

jl LEER_TEXTO ; Si es menor, continua leyendo

2.4 Lógica de Encriptación

El núcleo del procesamiento distingue entre letras mayúsculas, minúsculas y caracteres no alfabéticos para preservar la integridad del mensaje.

1. **Normalización:** Debido a que los códigos ASCII de las letras no comienzan en 0 (A=65, a=97), es necesario normalizar el valor restando la base ASCII antes de operar.
2. **Operación Modular:** Se suma el desplazamiento y se calcula el residuo de la división por 26 para manejar la circularidad del alfabeto.
3. **Desnormalización:** Se suma la base ASCII original al resultado para obtener el carácter cifrado final.

El siguiente segmento de código ilustra el tratamiento para mayúsculas:

Fragmento de código

ES_MAYUSCULA:

sub al, 'A' ; normalizar a 0-25

add al, bl ; sumar desplazamiento

mov ah, 0

mov cl, 26

```
div cl      ; dividir entre 26  
mov al, ah  ; tomar residuo (modulo 26)  
add al, 'A' ; volver a ASCII
```

(Véase el código completo en Apéndice A)

2.5 Lógica de Descriptación y Manejo de Negativos

En el proceso inverso, al restar el desplazamiento, el resultado puede ser negativo (ej. 'A' menos 3 posiciones). La implementación verifica el signo del resultado; si es negativo, se suma 26 para rotar al final del alfabeto.

Fragmento de código

ES_MAYUSCULA_DES:

```
sub al, 'A'  
sub al, bl  ; restar desplazamiento  
cmp al, 0   ; verificar si es negativo  
jge POSITIVO_MAY  
add al, 26  ; ajustar si es negativo (vuelta atrás)
```

2.6 Caracteres Especiales

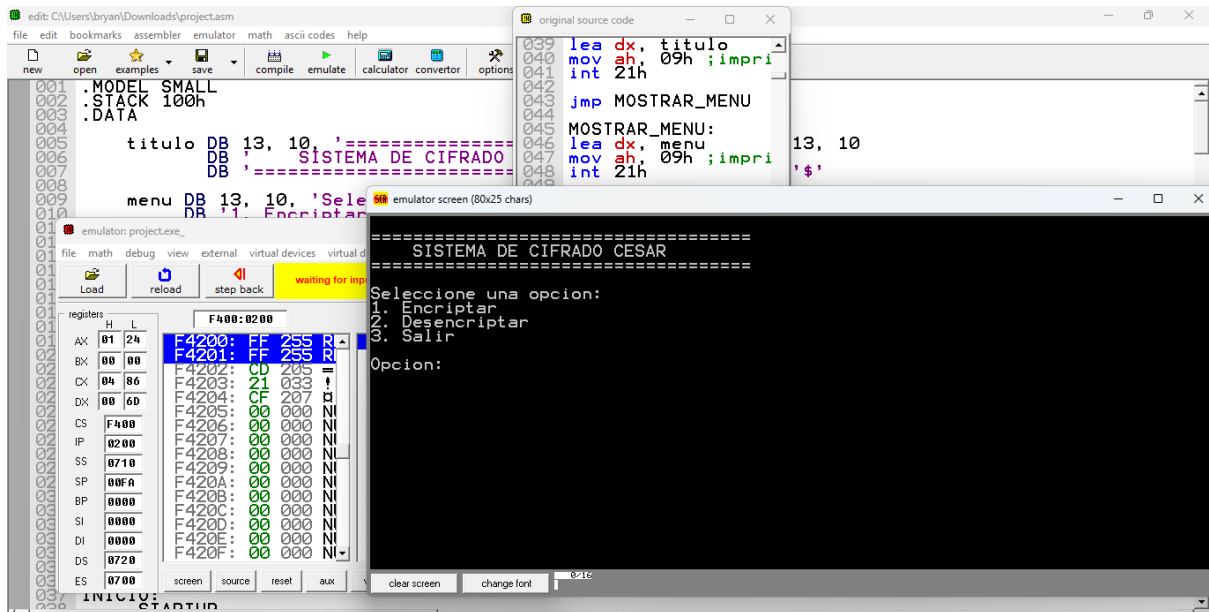
El sistema implementa una política de preservación para cualquier carácter que no sea una letra (espacios, números, signos). Estos se imprimen directamente sin modificaciones, saltando la lógica de cifrado.

3 Manual de Usuario y Pruebas

3.1 Menú Principal

Al iniciar el programa en emu8086, se presenta un menú interactivo con tres opciones. El sistema valida que el usuario ingrese únicamente los números '1', '2' o '3'.

Captura 1: Menú Principal



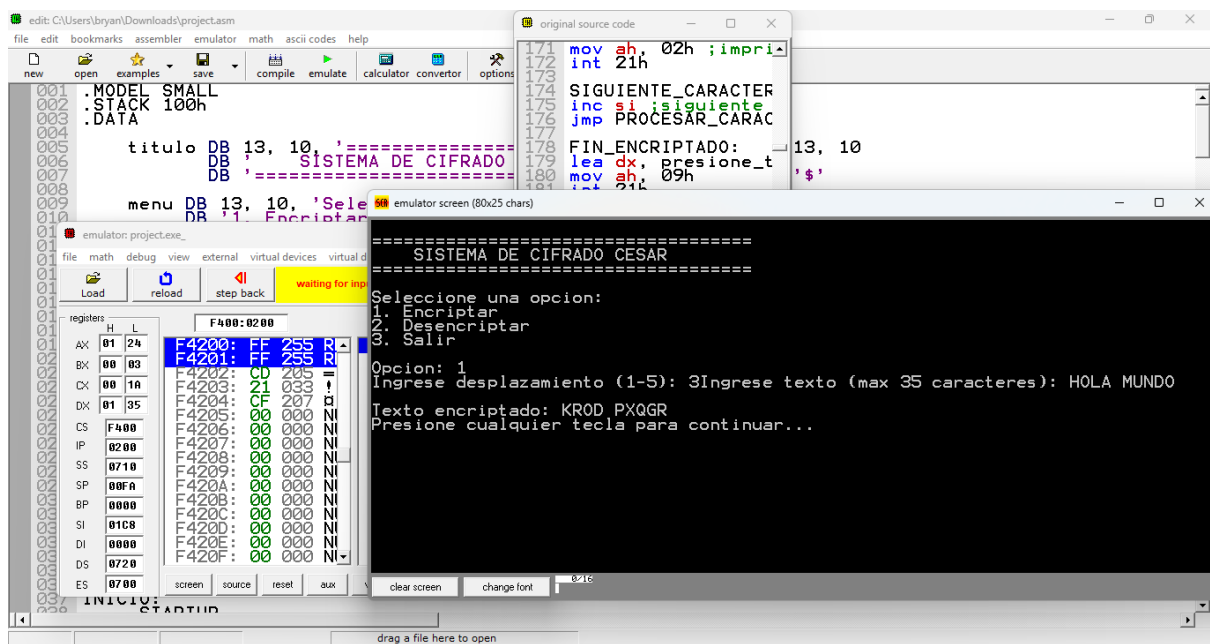
3.2 Proceso de Cifrado

El usuario selecciona la opción '1', ingresa un desplazamiento (validado entre 1 y 5) y posteriormente el texto.

Caso de Prueba:

- **Entrada:** HOLA MUNDO
- **Desplazamiento:** 3
- **Resultado Esperado:** KROD PXQGR.

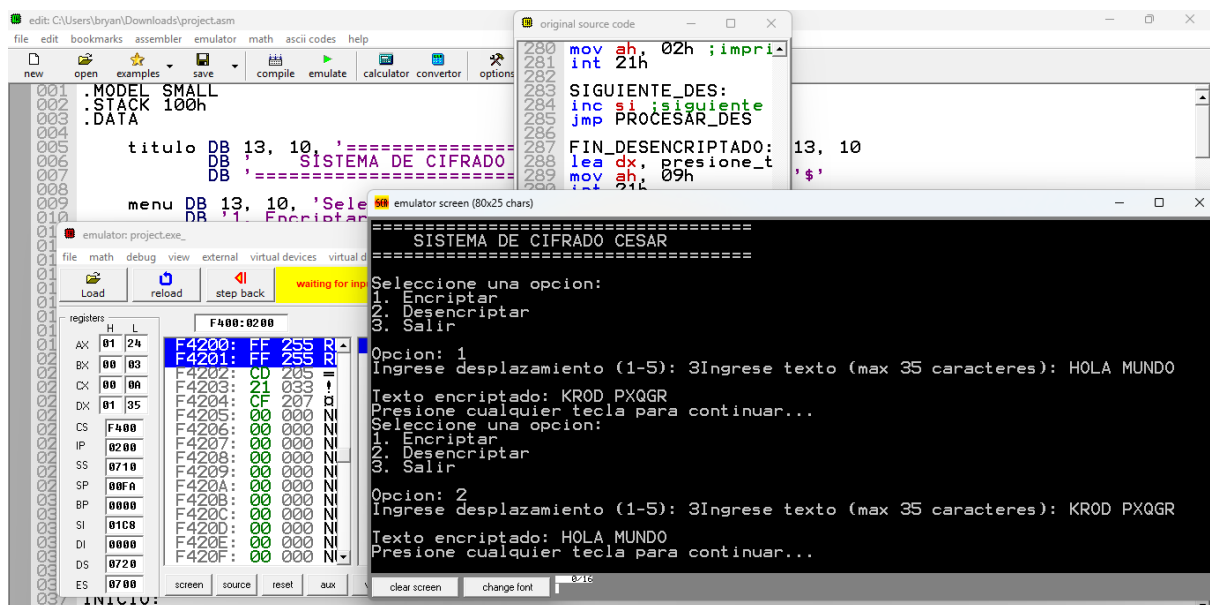
Captura 2: Ejecución de Cifrado



3.3 Proceso de Descifrado

Para verificar la reversibilidad, se selecciona la opción '2' utilizando el texto cifrado obtenido anteriormente.

Captura 3: Ejecución de Descifrado



4 Conclusiones

La implementación del Cifrado César en Assembly x86 permitió comprender a profundidad el manejo de registros y la aritmética a nivel de byte. Se logró cumplir con todos los requisitos funcionales, incluyendo la validación de límites de buffer (35 caracteres) y el manejo correcto del desbordamiento del alfabeto (wrap-around) mediante la división entera DIV para obtener el módulo.

Si bien el algoritmo presenta limitaciones inherentes de seguridad y un rango de desplazamiento restringido por diseño (1-5), el proyecto demuestra eficazmente la capacidad de manipular datos ASCII y controlar el flujo del programa mediante saltos condicionales y comparaciones.

5 Referencias

1. Escuela Superior Politécnica del Litoral. (2025). *Documentación del Proyecto: Organización de Computadores 2025-2*. Guayaquil: ESPOL.
2. Caesar Cipher Visualizer. (s.f.). Recuperado de <https://caesar-cipher.com/>
3. emu8086 Documentation. (s.f.). Recuperado de <http://www.emu8086.com/>

6 Apéndice A: Código Fuente

6.1 Github:

<https://github.com/g30dl/emu8086-caesar-encryption.git>

6.2 Código

A continuación, se presenta el código completo desarrollado para el proyecto:

Fragmento de código

.MODEL SMALL

.STACK 100h

.DATA

titulo DB 13, 10, '=====', 13, 10

DB ' SISTEMA DE CIFRADO CESAR', 13, 10

DB '=====', 13, 10, '\$'

menu DB 13, 10, 'Seleccione una opcion:', 13, 10

DB '1. Encriptar', 13, 10

DB '2. Desencriptar', 13, 10

DB '3. Salir', 13, 10

DB 13, 10, 'Opcion: \$'

error_opcion DB 13, 10, 'Error: Opcion invalida!', 13, 10

DB 'Por favor ingrese un numero entre 1 y 3.', 13, 10, '\$'

msg_salida DB 13, 10, 13, 10, 'Gracias por usar el sistema!', 13, 10

DB 'Hasta pronto...', 13, 10, '\$'

presione_tecla DB 13, 10, 'Presione cualquier tecla para continuar...\$'

msg_desplazamiento DB 13, 10, 'Ingrese desplazamiento (1-5): \$'

msg_texto DB 'Ingrese texto (max 35 caracteres): \$'

msg_resultado DB 13, 10, 13, 10, 'Texto encriptado: \$'

texto_entrada DB 36 DUP('\$') ;buffer para 35 caracteres + terminador

desplazamiento DB 0

desplazamiento_inv DB 0

.CODE

INICIO:

.STARTUP

lea dx, titulo

mov ah, 09h ;imprimir titulo

int 21h

jmp MOSTRAR_MENU

MOSTRAR_MENU:

lea dx, menu

mov ah, 09h ;imprimir el menu

int 21h

mov ah, 01h ;leer caracter

int 21h

cmp al, '1' ;validar rango 1-3

jl OPCION_INVALIDA

cmp al, '3'

jg OPCION_INVALIDA

```

cmp al, '1'
je ENCRIPITAR
cmp al, '2'
je DESENCRIPTAR
cmp al, '3'
je SALIR

jmp OPCION_INVALIDA

```

OPCION_INVALIDA:

```

lea dx, error_opcion
mov ah, 09h ;imprimir cadena
int 21h

```

```

lea dx, presione_tecla
mov ah, 09h
int 21h

```

```

mov ah, 01h ;esperar tecla
int 21h

```

```

jmp MOSTRAR_MENU

```

;===== ENCRIPACION

ENCRIPITAR:

```

lea dx, msg_desplazamiento
mov ah, 09h ;imprimir cadena
int 21h

```

```

mov ah, 01h ;leer caracter

```

```
int 21h
sub al, '0' ;convertir ASCII a numero
mov desplazamiento, al
```

```
cmp al, 1 ;validar rango 1-5
jl MOSTRAR_MENU
cmp al, 5
jg MOSTRAR_MENU
```

```
lea dx, msg_texto
mov ah, 09h ;imprimir cadena
int 21h
```

```
lea si, texto_entrada ;SI apunta al buffer
mov cx, 0 ;contador de caracteres
```

LEER_TEXTO:

```
mov ah, 01h ;leer caracter
int 21h
```

```
cmp al, 13 ;verificar ENTER
je FIN_Lectura
```

```
mov [si], al ;guardar en buffer
inc si
inc cx
cmp cx, 35 ;limite de 35 caracteres
jl LEER_TEXTO
```

FIN_Lectura:

```
mov byte ptr [si], '$' ;terminar cadena
```

```
lea dx, msg_resultado
mov ah, 09h ;imprimir cadena
int 21h
```

```
lea si, texto_entrada
mov bl, desplazamiento
```

PROCESAR_CHARACTER:

```
mov al, [si] ;cargar caracter actual
cmp al, '$' ;fin de cadena
je FIN_ENCRIPTADO
```

```
cmp al, 'A' ;verificar si es mayuscula A-Z
jl NO_ES_LETRA
cmp al, 'Z'
jle ES_MAYUSCULA
```

```
cmp al, 'a' ;verificar si es minuscula a-z
jl NO_ES_LETRA
cmp al, 'z'
jle ES_MINUSCULA
```

NO_ES_LETRA:

```
mov dl, al
mov ah, 02h ;imprimir caracter sin cambios
int 21h
jmp SIGUIENTE_CHARACTER
```

ES_MAYUSCULA:

```
sub al, 'A' ;normalizar a 0-25
add al, bl ;sumar desplazamiento
mov ah, 0
```

```

mov cl, 26
div cl ;dividir entre 26
mov al, ah ;tomar residuo (modulo 26)
add al, 'A' ;volver a ASCII
mov dl, al
mov ah, 02h ;imprimir caracter
int 21h
jmp SIGUIENTE_CARACTER

```

ES_MINUSCULA:

```

sub al, 'a' ;normalizar a 0-25
add al, bl ;sumar desplazamiento
mov ah, 0
mov cl, 26
div cl ;dividir entre 26
mov al, ah ;tomar residuo (modulo 26)
add al, 'a' ;volver a ASCII
mov dl, al
mov ah, 02h ;imprimir caracter
int 21h

```

SIGUIENTE_CARACTER:

```

inc si ;siguiente caracter
jmp PROCESAR_CARACTER

```

FIN_ENCRIPTADO:

```

lea dx, presione_tecla
mov ah, 09h
int 21h
mov ah, 01h ;esperar tecla
int 21h
jmp MOSTRAR_MENU

```

```
;=====
```

```
;===== DESENCRIPTAR
```

DESENCRIPTAR:

```
lea dx, msg_desplazamiento
mov ah, 09h ;imprimir cadena
int 21h
```

```
mov ah, 01h ;leer caracter
int 21h
sub al, '0' ;convertir ASCII a numero
mov desplazamiento, al
```

```
cmp al, 1 ;validar rango 1-5
jl MOSTRAR_MENU
cmp al, 5
jg MOSTRAR_MENU
```

```
lea dx, msg_texto
mov ah, 09h ;imprimir cadena
int 21h
```

```
lea si, texto_entrada ;SI apunta al buffer
mov cx, 0 ;contador de caracteres
```

LEER_TEXTO_DES:

```
mov ah, 01h ;leer caracter
int 21h
cmp al, 13 ;verificar ENTER
```

```
je FIN_LECTURA_DES
mov [si], al ;guardar en buffer
inc si
inc cx
cmp cx, 35 ;limite de 35 caracteres
jl LEER_TEXTO_DES
```

FIN_LECTURA_DES:

```
mov byte ptr [si], '$' ;terminar cadena
```

```
lea dx, msg_resultado
mov ah, 09h ;imprimir cadena
int 21h
```

```
lea si, texto_entrada
mov bl, desplazamiento
```

PROCESAR_DES:

```
mov al, [si] ;cargar caracter actual
cmp al, '$' ;fin de cadena
je FIN_DESENCRIPTADO
```

```
cmp al, 'A' ;verificar si es mayuscula A-Z
jl NO_ES_LETRA_DES
cmp al, 'Z'
jle ES_MAYUSCULA_DES
```

```
cmp al, 'a' ;verificar si es minuscula a-z
jl NO_ES_LETRA_DES
cmp al, 'z'
jle ES_MINUSCULA_DES
```

NO_ES_LETRA_DES:

```
mov dl, al
mov ah, 02h ;imprimir caracter sin cambios
int 21h
jmp SIGUIENTE_DES
```

ES_MAYUSCULA_DES:

```
sub al, 'A' ;normalizar a 0-25
sub al, bl ;restar desplazamiento

cmp al, 0 ;verificar si es negativo
jge POSITIVO_MAY
add al, 26 ;ajustar si es negativo
```

POSITIVO_MAY:

```
add al, 'A' ;volver a ASCII
mov dl, al
mov ah, 02h ;imprimir caracter
int 21h
jmp SIGUIENTE_DES
```

ES_MINUSCULA_DES:

```
sub al, 'a' ;normalizar a 0-25
sub al, bl ;restar desplazamiento

cmp al, 0 ;verificar si es negativo
jge POSITIVO_MIN
add al, 26 ;ajustar si es negativo
```

POSITIVO_MIN:

```
add al, 'a' ;volver a ASCII
mov dl, al
```

```
mov ah, 02h ;imprimir caracter
int 21h
```

SIGUIENTE_DES:

```
inc si ;siguiente caracter
jmp PROCESAR_DES
```

FIN_DESENCRIPTADO:

```
lea dx, presione_tecla
mov ah, 09h
int 21h
mov ah, 01h ;esperar tecla
int 21h
jmp MOSTRAR_MENU
```

```
;=====
```

SALIR:

```
lea dx, msg_salida
mov ah, 09h ;imprimir cadena
int 21h
```

```
mov ah, 4Ch ;terminar programa
int 21h
```

END INICIO