



Instituto Tecnológico de Costa Rica

**Curso: IC-3101 Arquitectura de Computadores
I Semestre 2015**

Tarea Programada #3: Máquina Enigma

**Profesor
Erick Hernández Bonilla**

Estudiantes:

- ❖ Melissa Molina Corrales 2013006074
- ❖ Liza Chaves Carranza 2013016573
- ❖ Gabriel Pizarro Picado 201216833
- ❖ Fabián Monge García 2014088148

Índice

Propósito y Descripción del proyecto.....	3
Funcionamiento del programa.....	5
Leer y abrir archivos.....	6
Carga de rotores seleccionados.....	8
Desplazamientos de los rotores.....	11
Bibliografía.....	16

Propósito y Descripción del proyecto

Enigma era el nombre de una máquina que disponía de un mecanismo de cifrado rotatorio, que permitía usarla tanto para cifrar como para descifrar mensajes.

Su fama se debe a haber sido adoptada por las fuerzas militares de Alemania desde 1930. Su facilidad de manejo y supuesta inviolabilidad fueron las principales razones para su amplio uso. Su sistema de cifrado fue finalmente descubierto y la lectura de la información que contenían los mensajes supuestamente protegidos es considerada, a veces, como la causa de haber podido concluir la Segunda Guerra Mundial.

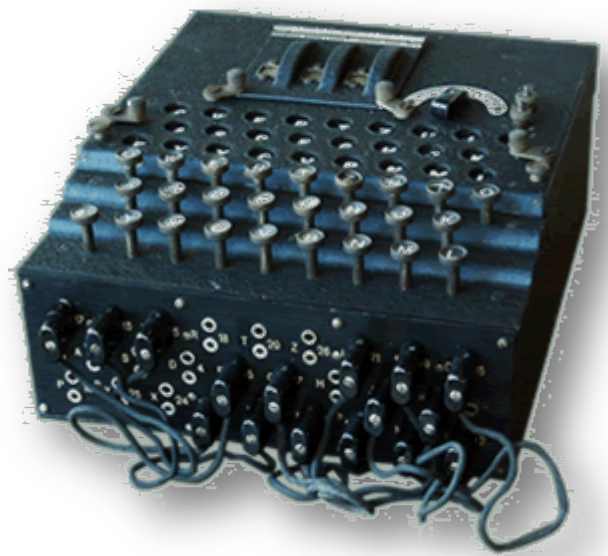


Figura 1. Máquina Enigma.

El objetivo de este proyecto es implementar una máquina enigma, esta tendrá el mismo funcionamiento que en la Segunda Guerra Mundial.

La tarea será programada en YASM (Ensamblador) en la sintaxis de Intel de x64.

El funcionamiento será el siguiente:

La máquina estará compuesta por un plugboard, un reflector y 5 rotores con configuraciones distintas de los cuales se escogerán 3 para llevar a cabo el funcionamiento.

El plugboard intercambiará las letras ingresadas, así por ejemplo cuando se ingresa una X esta será cambiada por una F según la configuración del plugboard. Su configuración será la siguiente:

XF, PZ, SQ, GR, AJ, UO, CN, BV, TM, KI

Después la letra pasa por tres rotores que intercambian las letras y luego el reflector que intercambia la letra una vez más y regresa a través de los tres rotores, por el plugboard y finalmente se muestra la letra encriptada por la máquina. Su configuración será de la siguiente manera:

JPGVOUMFYQBENHZRDKASXLICTW

Funcionamiento de los rotores

Cada vez que se desea encriptar una letra el rotor de la derecha avanza una posición, cuando este de una vuelta (lo cual corresponde a recorrer las 26 letras del alfabeto), avanza el rotor del medio y cuando este complete una vuelta, avanza el rotor de la izquierda de esta manera se van intercambiando las letras.

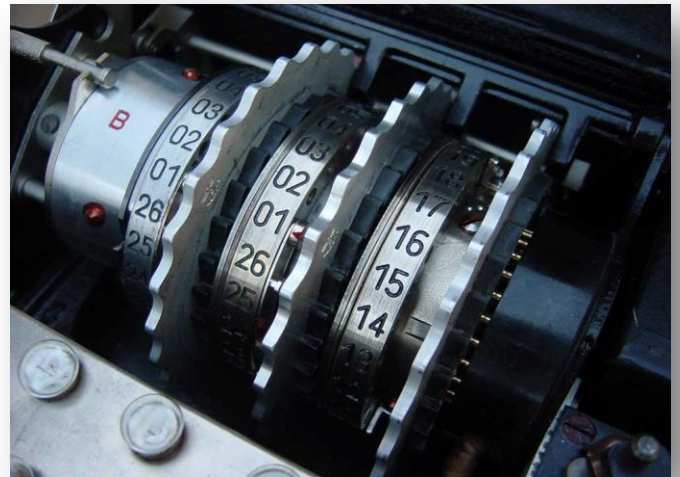


Figura 2. Rotores de una máquina enigma.

Estos rotores contarán con una posición inicial y su configuración será la siguiente:

Rotor 1	EKMFLGDQVZNTOWYHXUSPAIBRCJ
Rotor 2	AJDKSIRUXBLHWTMCQGZNPYFVOE
Rotor 3	BDFHJLCPRTXVZNYEIWGAKMUSQO
Rotor 4	ESOVpzJAYQUIRHXLNFTGKDCMWB
Rotor 5	VZBRGITYUPSDNHLXAWMJQOFECK

Funcionamiento del programa

Para el funcionamiento de la máquina enigma se cargarán las configuraciones en 2 archivos, el primer archivo tendrá los rotores a utilizar, las posiciones iniciales de cada uno de ellos y la configuración del plugboard y en el segundo archivo se pasará el mensaje que se desea encriptar o desencriptar.

Ejemplo del formato del archivo con las configuraciones:

IV, V, I = Rotores
23, 09, 20 = Posiciones iniciales de cada rotor
XF, PZ, SQ, GR, AJ, UO, CN, BV, TM, KI = Configuración del Plugboard

Luego de cargar los archivos, el programa procederá a encriptar el mensaje que se haya ingresado en el segundo archivo y mostrará en pantalla una animación de todo el proceso de encriptación, y finalmente mostrará la palabra encriptada.

Leer y abrir archivos

Una parte importante del programa es abrir y leer los archivos con las configuraciones y el mensaje a encriptar para llevar a cabo esta funcionalidad se implementó un programa que se encarga de leer los archivos que se le pasen y cuenta con varios procedimientos encargados de abrir y leer los archivos, quitar los espacios en blanco y las comas del archivo de configuraciones para efectos de facilitar la implementación de la máquina.



Figura 3. Procedimiento para abrir y leer un archivo

```
.limpiar_orden_rotor:  
  
    inc r10  
    cmp byte[textoFinal + r10], 10  
    je .limpiar_comienzo_rotor_aux  
    cmp byte[textoFinal + r10], ','  
    je .limpiar_orden_rotor  
  
    mov al, byte[textoFinal + r10]  
    mov byte[buffRotoresSeleccionados + r9], al  
    inc r9  
    jmp .limpiar_orden_rotor  
  
.limpiar_comienzo_rotor_aux:  
    xor r9, r9  
    mov r9, 0  
  
.limpiar_comienzo_rotor:  
  
    inc r10  
    cmp byte[textoFinal + r10], 10  
    je .limpiar_plugboard_aux  
    cmp byte[textoFinal + r10], ','  
    je .limpiar_comienzo_rotor  
  
    mov al, byte[textoFinal + r10]  
    mov byte[buffDesplazamientoRotores + r9], al  
    inc r9  
    jmp .limpiar_comienzo_rotor
```

```
.limpiar_plugboard_aux:  
    xor r9, r9  
    mov r9, 0  
  
.limpiar_plugboard:  
  
    inc r10  
    cmp byte[textoFinal + r10], 0h  
    je .imprimir  
    cmp byte[textoFinal + r10], ','  
    je .limpiar_plugboard  
    cmp byte[textoFinal + r10], ','  
    je .limpiar_plugboard  
  
    mov al, byte[textoFinal + r10]  
    mov byte[plugboard + r9], al  
    inc r9  
    jmp .limpiar_plugboard
```

Figura 4. Procedimientos para quitar los espacios en blanco y las comas de los datos del archivo de configuraciones.

Carga de rotores seleccionados

Para el funcionamiento de la máquina se necesita escoger 3 de los 5 rotores, estos 3 rotores se le pasan en el archivo de configuraciones, una vez se conozcan los rotores a usar, se necesita saber sus configuraciones y sus posiciones iniciales para esto se crean procedimientos que cargan los rotores seleccionados en diferentes buffers.

Se utilizarán buffers para guardar los 3 rotores seleccionados y variables para guardar las configuraciones de los rotores y la configuración del reflector.

```
buffRotorUno resb buffRotorLen
buffRotorDos resb buffRotorLen
buffRotorTres resb buffRotorLen
```

Figura 5. Buffers en donde se guardaran los rotores seleccionados.

```
rotor1: db "EKMFLGDQVZNTOWYHXUSPAIBRCJ"
rotor2: db "AJDKSIRUXBLHWTMCQGZNPYFVOE"
rotor3: db "BDFHJLCPRTXVZNYEIWGAKMUSQO"
rotor4: db "ESOVJPZJAYQUIRHXLNFTGKDCMWB"
rotor5: db "VZBRGITYUPSDNHLXAWMJQOFECK"
reflector: db "JPGVOUMFYQBENHZRDKASXLICTW"
```

Figura 6. Variables que guardan las configuraciones de cada uno de los rotores y la configuración del reflector.

Como se mencionó anteriormente se crea un procedimiento que carga los rotores en diferentes buffers para ser usados más adelante para poder encriptar el mensaje.

```
CargarRotoresSeleccionados:
    push rcx
    push rax
    push rsi
    push rdi
    push r8
    push r9
    xor rsi, rsi
    mov rcx, 1

    .begin:
        cmp byte[buffRotoresSeleccionados + rsi], 00h
        je .exit

        mov r8, buffRotoresSeleccionados
        mov r9, buffNumRomano
        call CargarNumeroABuffer

        mov rax, buffNumRomano
        call CambiarRomanoADecimal
        mov rax, buffNumRomano
        call LimpiarBuff
        push rsi
        call AsignarRotorSeleccionado
        call AsignarBufferDeRotorSeleccionado
        call CargarRotorABuffer
        pop rsi

        inc rcx
        jmp .begin
```

Figura 7. Procedimiento principal para cargar los rotores.

Como complemento a este procedimiento se utiliza un procedimiento que cambia los números romanos con los que vienen identificados los rotores en el archivo de configuraciones a números decimales esto con el fin de facilitar la carga de estos a los diferentes buffers.

```

CargarNumeroABuffer:
    push rax
    push rcx
    xor rcx, rcx

    .begin:
        cmp byte[r8 + rsi], 00h
        je .exit

        cmp byte[r8 + rsi], ','
        je .nextNum

        mov al, byte[r8 + rsi]
        mov byte[r9 + rcx], al

        inc rcx
        inc rsi
        jmp .begin

    .nextNum:
        inc rsi

    .exit:
        pop rcx
        pop rax
        ret

```



```

CambiarRomanoADecimal:
    xor rdx, rdx

    cmp byte[rax], 'V'
    je .cinco

    cmp byte[rax + 1], 'V'
    je .cuatro

    cmp byte[rax + 2], 'I'
    je .tres

    cmp byte[rax + 1], 'I'
    je .dos

    mov rdx, 1
    jmp .exit

    .dos:
        mov rdx, 2
        jmp .exit

    .tres:
        mov rdx, 3
        jmp .exit

    .cuatro:
        mov rdx, 4
        jmp .exit

```

Figura 8. Procedimiento que carga los números romanos de los rotores al buffer de rotores seleccionados.

Figura 9. Procedimiento que cambia los números romanos de los rotores a números decimales.

Una vez se haya pasado los números romanos de los rotores a números en decimal se utilizan procedimientos para asignarle las configuraciones de los rotores seleccionados a cada uno de los buffers mostrados anteriormente.

```

AsignarRotorSeleccionado:
    cmp rdx, 1
    je .SeleccionarRotorUno

    cmp rdx, 2
    je .SeleccionarRotorDos

    cmp rdx, 3
    je .SeleccionarRotorTres

    cmp rdx, 4
    je .SeleccionarRotorCuatro

    cmp rdx, 5
    je .SeleccionarRotorCinco

    .SeleccionarRotorUno:
        mov rsi, rotor1
        jmp .exit

    .SeleccionarRotorDos:
        mov rsi, rotor2
        jmp .exit

    .SeleccionarRotorTres:
        mov rsi, rotor3
        jmp .exit

    .SeleccionarRotorCuatro:
        mov rsi, rotor4
        jmp .exit

    .SeleccionarRotorCinco:
        mov rsi, rotor5
        jmp .exit
  
```



```

AsignarBufferDeRotorSeleccionado:
    cmp rcx, 1
    je .bufferRotorUno

    cmp rcx, 2
    je .bufferRotorDos

    cmp rcx, 3
    je .bufferRotorTres

    .bufferRotorUno:
        mov rdi, buffRotorUno
        jmp .exit

    .bufferRotorDos:
        mov rdi, buffRotorDos
        jmp .exit

    .bufferRotorTres:
        mov rdi, buffRotorTres
        jmp .exit

    .exit:
        ret
  
```

Figura 11. Procedimiento que asigna al rdi el buffer para guardar el rotor asignado.

Figura 10. Procedimiento que asigna al rsi el rotor que se debe pasar.

Finalmente se utiliza un procedimiento que se encarga de cargar los rotores a los buffers, copia lo que tenga el rsi (Las configuraciones de los rotores seleccionados) en el rdi (buffers para los rotores) 26 veces que es el tamaño de cada uno de los rotores.



```

CargarRotorABuffer:
    push rcx

    xor rcx, rcx
    mov rcx, 26
    rep movsb

    pop rcx
    ret
  
```

Figura 12. Procedimiento que carga un rotor a un buffer.

Desplazamientos de los rotores

Para poder encriptar el mensaje que se le pase a la máquina se necesita que los rotores hagan rotaciones para intercambiar las letras cada vez que se quiera encriptar una letra, para esto se implementan algunos procedimientos que permitan mover de alguna manera la posición de los rotores para que estos vayan cambiando las letras.

Este procedimiento va a recorrer el buffer en donde están las posiciones iniciales de los rotores y dependiendo del número que haya en el registro rbx ese será el rotor que hay que rotar y seleccionará el rotor correspondiente, guardado en cada uno de los diferentes buffers y lo rotará la cantidad de veces que haya en el registro rcx.

```
RotarRotores:
    push rcx
    push rsi
    push r8
    push r9
    push rbx
    xor rsi, rsi
    mov rbx, 1

    .begin:
        cmp byte[buffDesplazamientoRotores + rsi], 00h
        je .exit

        mov r8, buffDesplazamientoRotores
        mov r9, despTemp
        call CargarNumeroABuffer

        mov r8, despTemp
        call Atoi

        mov rax, despTemp
        call LimpiarBuff

        push rsi
        mov rcx, rdx

        cmp rbx, 1
        je .selectRotorUno

        cmp rbx, 2
        je .selectRotorDos

        cmp rbx, 3
        je .selectRotorTres

        .selectRotorUno:
            mov rsi, buffRotorUno
            jmp .rotar

        .selectRotorDos:
            mov rsi, buffRotorDos
            jmp .rotar

        .selectRotorTres:
            mov rsi, buffRotorTres
            jmp .rotar

        .rotar:
            call RotateLeftRotor

        pop rsi
        inc rbx
        jmp .begin

    .exit:
        pop rbx
        pop r9
        pop r8
        pop rsi
        pop rcx
        ret
```

Figura 13. Procedimiento que desplaza los rotores n veces.

Como parte de este procedimiento para poder realizar las rotaciones a los rotores, se implementa un procedimiento para hacer las rotaciones correspondientes, importante mencionar que a los rotores se les aplicará una rotación a la izquierda.

```
RotateLeftRotor:
    push rax
    push rbx
    push rdx
    xor rax, rax
    xor rbx, rbx

    .begin:
        cmp rcx, 0
        je .exit
        mov al, byte[rsi]

    .ciclo:
        cmp byte[rsi + rbx + 1], 00h
        je .agregarDigito

        mov dl, byte[rsi + rbx + 1]
        mov byte[rsi + rbx], dl
        inc rbx
        jmp .ciclo

    .agregarDigito:
        mov byte[rsi + rbx], al
        dec rcx
        mov rbx, 0
        jmp .begin

    .exit:
        pop rdx
        pop rbx
        pop rax
        ret
```

Figura 14. Procedimiento que aplica una rotación a la izquierda a un buffer.

Para convertir el desplazamiento de string a integer se usará el procedimiento Atoi.

```
Atoi:
    push rax
    push rbx
    push rsi
    push rdi

    xor rsi, rsi
    xor rax, rax
    xor rbx, rbx
    xor rdx, rdx
    xor rdi, rdi

    .begin:
        mov al, byte[r8 + rsi]
        cmp al, 00h
        je .exit

        cmp al, 30h
        jb .exit

        cmp al, 39h
        ja .exit

        sub al, 30h

        mov rdi, rax
        inc rsi
        jmp .seguir

    .seguir:
        mov al, byte[r8 + rsi]
        cmp al, 00h
        je .exit

        mov rbx, 10
        mul rbx
        mov rdx, rax
        add rdx, rdi
        jmp .begin

    .exit:
        mov rax, rdx
        mov rbx, 10
        mul rbx
        mov rdx, rax
        add rdx, rdi

        pop rdi
        pop rsi
        pop rbx
        pop rax
        ret
```

Figura 15. Procedimiento ATOI convierte de string a integer.

Bibliografía

Dade, L. (2006-2015). *Enigma Emulator*. Obtenido de <http://enigma.louisedade.co.uk/index.html>

Enigma rotor details. (10 de Mayo de 2015). Obtenido de Wikipedia:
http://en.wikipedia.org/wiki/Enigma_rotor_details

Rijmenants, D. (14 de Febrero de 2015). *Cipher Machines and Cryptology*. Obtenido de
<http://users.telenet.be/d.rijmenants/index.htm>

Simon, P. R. (31 de mayo de 2014). *Crypto Museum*. Obtenido de
<http://www.cryptomuseum.com/crypto/enigma/working.htm>

UCDavis. (1 de Febrero de 2010). *The Enigma Code*. Obtenido de <https://www.youtube.com/watch?v=nCL2Fl6prH8>