

ElaNanguanJoseLuisObiangGrupoCNASM



PRÁCTICA NASM - CRIPTOGRAFÍA

GITHUB NASM

Nombre: Jose Luis Obiang Ela Nanguan

Grupo: C

Profesor: Raúl Lérida

Fecha 29/05/2022

Índice

1. Enunciado
2. Memoria
3. Código
4. Datos de prueba
5. Bibliografía
6. Opinión

1. Enunciado

Enunciado de la Práctica de NASM

Se trata de realizar, en NASM, un proceso que realice un pequeño trabajo de "criptografía". Lo primero que hará nuestro programa es visualizar por pantalla el literal "Programa sobre CRIPTOGRAFÍA", en la posición 2,2 de pantalla (Fila = 2, Columna = 2), para indicarnos que va a comenzar el proceso de entrada de valores:

A continuación, nuestro programa, va a ir visualizando las posiciones de un vector donde vamos a ir introduciendo las "**traducciones alfabéticas**" correspondientes a la clave numérica introducida. Así, visualizaremos en posiciones consecutivas de pantalla, los literales correspondientes a la entrada de los valores para que el usuario los vaya introduciendo "a su gusto". En la posición 4, 2, de pantalla (Fila = 4, Columna = 2), visualizaremos el literal "**Introduce el contenido de la posición 0:**" para que introduzcamos la letra **A**. En la posición 5, 2, de pantalla (Fila = 5, Columna = 2), visualizaremos el literal "**Introduce el contenido de la posición 1:**" para que introduzcamos la letra **B**, ... En la posición

13, 2, de pantalla (Fila = 13, Columna = 2), visualizaremos el literal "Introduce el contenido de la posición 9:" para que introduzcamos la letra **J**.

La correspondencia se denota a continuación: POSICIONES DEL VECTOR CONTENIDOS

A su vez, el programa irá almacenando estos valores en sus posiciones del vector correspondientes.

Hay que comprobar que todos los valores introducidos (desde la **A** hasta la **J**), NO SE REPITAN. Si ocurre esto, se debe sacar un mensaje "**¡CUIDADO! Ese valor ya existe.**", en la posición 15, 02 y permitir que se introduzca de nuevo otro valor no repetido:

A continuación, se nos pide que introduzcamos una clave de, al menos, 6 dígitos (**pueden ser menos de 6. 6 máximo**). Para ello visualizaremos en la posición 15, 2, el literal "**Introduce la CLAVE (6 caracteres numéricos):**".

Por ejemplo, vamos a meter los dígitos **189427**:

El programa nos tiene que devolver los valores correspondientes a la clave introducida. Es decir, viendo los valores que hemos introducido en el vector, al **1**, le corresponde el valor **B**, al **8**, le corresponde el valor **I**, al **9**, le corresponde el valor **J**, al **4**, le corresponde el valor **E**, al **2**, le corresponde el valor **C**, y al **7**, le corresponde el valor **H**.

El resultado es: **BIJECH**

Estos valores los visualizaremos a través del literal "**Solución:**" que pondremos en la posición 17, 2:

Por último, el programa nos preguntará si queremos realizar otra operación mediante el literal "**¿Otra Operación (S/N)?:**", en la posición 19, 2. A esta pregunta podemos responder "S" o "s" para realizar otra operación o "otra respuesta" para finalizar el programa:

Si respondemos "s" o "S" volveremos a pedir una nueva clave a través del literal "Introduce la CLAVE (6 caracteres numéricos)": Los literales "Solución:" y "¿Otra Operación (S/N)?:" desaparecen de pantalla.

Veamos ahora lo que sucede cuando introducimos una clave de 4 dígitos, por ejemplo 1234: Si ahora respondemos "N" o "n", el programa acaba.

2. Memoria

Antes de empezar con el desarrollo del programa en lenguaje de bajo nivel, Ensamblador, primero he conseguido hacerlo en Java(lenguaje interpretado de alto nivel) para que de esta forma sepa cómo proceder a hacerlo en un lenguaje más parecido al lenguaje máquina. He aplicado la ley de divide y vencerás para así segmentar el programa y hacerlo más fácil de entender.

Este es el Código del programa en Java:

Dicho código se compone de 4 partes:

1. Fill array:

- Primero he creado un vector de tipo char con tamaño 10 que almacene cada una de las distintas claves(vClaves)
- letter: variable de tipo char que almacena temporalmente la clave introducida para luego compararlo con cada una de las claves anteriores del vector para así poder controlar que no estemos introduciendo claves duplicadas, lo que significa que solo se almacenara la clave siempre y cuando no este previamente almacenada en el vector
- verify: función que compara la nueva clave introducida con cada una de las claves anteriores del vector
- Luego recorro dicho vector y por cada posición muestro un mensaje al usuario concatenado con la posición actual del vector y así introduzca las claves por cada posición.

2. Enter the string Numbers

- clave: variable de tipo string que almacena la cadena de numeros a traducir, pedimos al usuario que introduzca una cadena la cual traduciremos mediante las claves
- cada caracter de la cadena es un numero que logicamente representa una posicion del vector, entonces para la traduccion lo que he hecho es recorrer la cadena y por cada iteracion he convertido cada caracter a numero para asi pasarlo al vector como indice y acceder a la clave correspondiente a dicha posicion y finalmente almacenarlos todos en una variable "traduccion"

3. Show translate keys

- Mostramos la traduccion
- Preguntamos al usuario si quiere volver a traducir, la respuesta se almacena en una variable "resp": while resp='S' || resp='s' && resp!='N'

```

1 import java.io.BufferedReader;
2 import java.io.IOException;
3 import java.io.InputStreamReader;
4 import java.util.Arrays;
5
6 public class CriptografiaNasm {
7     public static void main(String[] args) throws IOException {
8         //Declaracion de las variables a usar a lo largo del programa
9         BufferedReader input=new BufferedReader(new InputStreamReader(System.in));
10        char[] vClaves=new char[10];      //Se trata del vector el cual almacenará las distintas claves
11        char letter, resp='S';          //letter: variable que almacenará temporalmente la clave introducida
12                                         // para luego compararlo con las claves anteriores del vector para así poder evitar
duplicados
13                                         //resp: variable que almacena 'S'/'s'/'N'
14        String clave="";
15        System.out.println("Programa sobre CRIPTOGRAFIA.\n"); //Mostramos el mensaje de bienvenida
16        for(int i=0; i<10; i++) //Recorremos el vector para ir almacenando las distintas claves por cada posicion
17        {
18            System.out.print("Introduce el contenido de la posicion "+i+": ");
19            letter=input.readLine().charAt(0);
20            while(verify(vClaves, letter)){ //Nos aseguramos que la clave introducida no se repita
21                System.out.print("Introduce el contenido de la posicion "+i+": "); //Para ello comprobamos el valor
actual con las claves anteriores del vector
22                letter=input.readLine().charAt(0); //En caso de que ya exista volvemos a pedir la clave
23            }
24            vClaves[i]=letter; //Almacenamos la clave sabiendo que no se repite ya que no se saltaría a esta línea de
código si se encuentra una clave ya duplicada
25        }
26        while((resp=='S' || resp=='s') && resp!='N') {
27            StringBuilder traducida=new StringBuilder();
28            System.out.print("Introduce la CLAVE (6 caracteres maximo): ");
29            clave = input.readLine(); //Introducimos la cadena de valores a traducir
30
31            /**por cada iteración obtenemos un carácter de la cadena a traducir.
32             * Cada carácter es un dígito que representa una posición del vector de claves.
33             * Por ejemplo si introducimos 356789: 3-->D; 5-->F; 6-->G; 7-->H; 8-->I; 9-->J
34             * Cada uno de los caracteres de la cadena a traducir es una posición del vector de claves.
35             * Por lo tanto, solo necesitamos recorrer la cadena y acceder a cada uno de los caracteres,
36             * convertirlos a entero y pasarlo como indicador de posición del vector, luego accedemos a
37             * los elementos de cada posición y los almacenamos en una nueva variable llamada traducida.
38             */
39            for (int i = 0; i < clave.length(); i++) {
40                traducida.append(vClaves[Character.getNumericValue(clave.charAt(i))]); //añadimos las traducciones a
la variable traducción
41            }
42
43            System.out.println("Solucion: " + traducida.toString()+"\n"); //Mostramos la traducción
44            System.out.print("Otra operacion (S|s/N)?: "); //Preguntamos al usuario si quiere volver a traducir
45            resp = input.readLine().charAt(0); //Si respuesta es S|s continua y en caso contrario si es N se termina
el programa
46            if(resp=='N'){
47                System.out.println("\n"+FIN DEL PROGRAMA");
48            }
49        }
50    }
51
52
53 /**
54  * Verify boolean. función que comprueba si la clave introducida ya existe.
55  * Devuelve true si ya existe y en caso contrario false
56  *
57  * @param vClaves the v claves
58  * @param letter the letter
59  * @return the boolean
60  */
61 public static boolean verify(char[] vClaves, char letter){
62     int cont = 0;
63     boolean existe= false;
64     while(cont<vClaves.length && !existe){
65         if(vClaves[cont]==letter){
66             existe=true;
67         }
68         cont++;
69     }
70     return existe;
71 }
72 }
73

```

Ejecución del programa en Java:

```
Run: CriptografiaNasm x
▶ Programa sobre CRIPTOGRAFIA.

↑ Introduce el contenido de la posicion 0: A
↓ Introduce el contenido de la posicion 1: B
■ Introduce el contenido de la posicion 2: C
✖ Introduce el contenido de la posicion 3: A
🖨️ Introduce el contenido de la posicion 3: B
⠄ Introduce el contenido de la posicion 3: D
⠄ Introduce el contenido de la posicion 4: E
⠄ Introduce el contenido de la posicion 5: F
⠄ Introduce el contenido de la posicion 6: G
⠄ Introduce el contenido de la posicion 7: H
⠄ Introduce el contenido de la posicion 8: I
⠄ Introduce el contenido de la posicion 9: J
⠄ Introduce la CLAVE (ó caracteres maximo): 654327
Solucion: GFEDCH

Otra operacion (S|s/N)?: S
Introduce la CLAVE (ó caracteres maximo): 8765
Solucion: IHGF

Otra operacion (S|s/N)?: s
Introduce la CLAVE (ó caracteres maximo): 98
Solucion: JI

Otra operacion (S|s/N)?: N

FIN DEL PROGRAMA

Process finished with exit code 0
```

El programa en sí es fácil de entender pero no es fácil de desarrollar en lenguaje de bajo nivel porque se necesitan muchas más instrucciones ya que existe una correspondencia más directa entre una instrucción en ensamblador y una en lenguaje máquina, dichas instrucciones se pueden resumir en muy pocas líneas de código como podemos observar en Java. El manejo de los vectores y de las variables es muy sencillo en cuanto a lenguajes de alto nivel, en Ensamblador necesitas un buen conocimiento de las distintas interrupciones que se pueden dar a nivel máquina, los registros, el paso de dirección a memoria o de memoria a dirección, el tamaño de bits por registro a usar, cómo recorrer los vectores y saberse el uso de cada uno de los registros especiales, así como acceder a los distintos valores de un vector y poder compararlos entre sí o con otro registro o valor.

1. Declaracion de Constantes,variables, macros e implementacion de funciones

a. Constantes:

```
;Initialization interruption constant
sys_exit equ 0x1      ;interruption of output
sys_write equ 0x4      ;interruption of writing
sys_read equ 0x3       ;interruption of reading
stdin equ 0            ;system of input
stdout equ 0x1         ;system of output
service equ 0x80        ;service of successful exit
```

```
;screen messages
L_msg0 equ $- msg0
L_msg1 equ $ - msg1
L_puntos equ $- puntos
L_msg4 equ $- msg4
L_msg5 equ $- msg5
L_msg6 equ $ - msg6

;row and column positions
L_position1 equ $- position1
L_position2 equ $- position2
L_position3 equ $- position3
L_position4 equ $- position4
L_position5 equ $- position5
L_position6 equ $- position6
L_position7 equ $- position7
L_position8 equ $- position8
L_position9 equ $- position9
L_position10 equ $- position10
L_position11 equ $- position11
L_position12 equ $- position12
L_position13 equ $- position13
L_position14 equ $- position14
L_position15 equ $- position15
L_hideLine equ $- hideLine
;clear
L_clear equ $- clear
;Initialize content array
size equ $- vKeys
sizeKeyNumber equ $- keyNumber
```

b. Variables

```

;screen messages
msg0 db 27,"[1;3m",27,"[1;35m","Programa sobre CRIPTOGRAFIA."
msg1 db 27,"[1;34m","Introduce el contenido de la posicion ",0h
puntos db ': '
msg3 db 27,"[1;31m",";CUIDADO! Ese valor ya existe."
msg4 db 27,"[1;32m","Introduce la clave (6 caracteres como maximo):"
msg5 db "Solucion: ",27,"[1;94m"
msg6 db 27,"[1;33m","Otra operacion (S|s/N)?: "
|
        ;row and column positions
position1 db 27, "[02;02H" ;move cursor to row 2 and column 2
position2 db 27, "[04;02H" ;move cursor to row 4 and column 2
position3 db 27, "[05;02H" ;move cursor to row 5 and column 2
position4 db 27, "[06;02H" ;move cursor to row 6 and column 2
position5 db 27, "[07;02H" ;move cursor to row 7 and column 2
position6 db 27, "[08;02H" ;move cursor to row 8 and column 2
position7 db 27, "[09;02H" ;move cursor to row 9 and column 2
position8 db 27, "[10;02H" ;move cursor to row 10 and column 2
position9 db 27, "[11;02H" ;move cursor to row 11 and column 2
position10 db 27, "[12;02H" ;move cursor to row 12 and column 2
position11 db 27, "[13;02H" ;move cursor to row 13 and column 2
position12 db 27, "[15;02H" ;move cursor to row 15 and column 2
position13 db 27, "[17;02H" ;move cursor to row 17 and column 2
position14 db 27, "[17;12H" ;move cursor to row 17 and column 1
position15 db 27, "[19;02H" ;move cursor to row 19 and column 2
hideLine times 100 db ' '
                ;clear
clear db 27, "[2J"

        ;Initialize content array
vKeys times 10 db '0'    ;ten position array
keyNumber times 6 db '0'
index db 1      ;counter
i db 1       ;other ;counter
                ;Declaration variables without initialize
segment .bss
key resb 2      ;variable that stores the key that we introduce
v_i resb 1      ;index of array
resp resb 1     ;variable that stores the user's response

```

- c. Macros: Son basicamente procedimientos ya que son funciones que reciben parametros pero que no devuelven nada.

```
Macros to reuse code
%macro write 2 ;show screen message
    mov ecx, %1
    mov edx, %2
    mov eax, sys_write
    mov ebx, stdout
    int service
%endmacro

%macro read 2 ;keyboard reading
    mov eax, sys_read
    mov ebx, stdin
    mov ecx, %1
    mov edx, %2
    int service
%endmacro

%macro indexVector 2 ;%1-> first parameter: is the current index. %2-> second parameter:
    cmp BYTE[i],%1 ;is the jump function if the current index is equal to the value
    je %2 ;is the function of jump, we jump to the corresponding function.
%endmacro

%macro showInfPost 2 ;show message to enter the key to translate
    write %1, %2 ;%1-> first parameter: positions fil and column
    write msg1, L_msg1 ;show message corresponding to the position of the vector
    write v_i,1 ;show the current index
    write puntos, L_puntos ;show the tow points --> we get "Enter the content of the <index>: "
    jmp intro ;unconditional jump to function to introduce a value.
%endmacro

%macro showMessage 4 ;show message in a position
    write %1, %2
    write %3, %4
%endmacro
```

d. Funciones

```

introKey:      ;function that show a message and to introduce values corresponding to the position of
               ;the vector
intro:          ;we introduce the value/key
verify:         ;function to verify if a key exists
fill:           ;function to store the key entered and increment the registers edi, esi and index i
incrementIndex: ;function increment the index to continue comparing
re_enter:        ;function, we ask the user again to enter the key in the same position when the
               ;previously entered key already exists
showIntro:      ;function control index vector
post?:          ;show "Introduce el contenido de la posición ?: "
               ;show message for introduce the key
introKeyNumber: ;we ask the user to enter the key of numbers to translate
translateKey:   ;we translate the key numbers
ciclo_impresion:;we travel the key of numbers
ciclo_lecatura: ;read the new array of numbers
keyContinue:    ;we hide the message "Otra operacion (S|s/N)?:" && "Solucion: " && key
               ;Numbers
exit:          ;exit
clearScreen:   ;clear screen

```

1. Explicacion del codigo. Mi codigo se estructura en 4 partes:

a. Fill Array

- Primero he creado un vector de tamaño 10 que almacene cada una de las distintas claves(vKeys).
- He creado un bucle para recorrer el vector e ir introduciendo las distintas claves por cada posición del vector, cada vez que se introduce una clave se llama a la función **showIntro**, la cual nos muestra "*Introduce el contenido de la posición i:*" para que así introduzcamos la clave, dicha clave siempre que se introduce se almacena en dos registros al y bl, de manera que el registro bl es el registro comparador, gracias a la función **verify**, comparamos dicho registro bl con las claves anteriores del vector pero para ello primero inicializamos un registro contador ecx, el cual se incrementa siempre que iteramos por cada comparación del valor del registro bl con los elementos que ocupan posiciones anteriores del array llamando así a la función **incrementIndex** el cual incrementa el registro contador ecx siempre y cuando el total de iteraciones sea inferior al tamaño del vector, en caso contrario y no se encuentre un duplicado de la nueva clave introducida se llama a la función **fill**, la cual nos permite almacenar la nueva clave almacenada en el registro bl en el vector e inmediatamente incrementamos los registros edi(registro de control de puntero), esi(registro que almacena la dirección del vector) y el registro índice que almacena la posición actual para así poder visualizarlo en pantalla y de esa manera saber por qué posición vamos. Almacenamos tantas veces como indica el tamaño del array.

b. Enter the array Numbers

- Una vez rellenado el array con las distintas claves de traducción, procedemos a introducir valores numéricos en el nuevo array de números gracias al bucle **ciclo_lecatura** para luego traducir cada uno de los valores con sus correspondientes claves almacenadas anteriormente en el array
- Después de haber llenado el array de números a traducir a sus correspondientes claves llamamos a la función **translateKey**, la cual con el bucle **ciclo_impresion** recorre el array de números a traducir, entonces cada elemento iterado del array lo almacenamos en el registro al para transformarlo en carácter con la instrucción **\add**

`al, 17` de manera que el elemento 0 en ascii sería el número 65 que corresponde a la A y así sucesivamente.

Por cada elemento iterado y traducido lo mostramos por pantalla en su correspondiente ubicación en fila y columna

c. Continue?

- Preguntamos al usuario si quiere volver a traducir, la respuesta se almacena en una variable "resp": comparamos su respuesta con 'S' || 's' && 'N' y segun haya introducido terminaría el programa o seguiría traduciendo

3. Código



```
;Initialization interruption constant
sys_exit equ 0x1      ;interruption of output
sys_write equ 0x4      ;interruption of writing
sys_read equ 0x3       ;interruption of reading
stdin equ 0            ;system of input
stdout equ 0x1         ;system of output
service equ 0x80        ;service of successsful exit

;Macros to reuse code
%macro write 2 ;show screen message
    mov ecx, %1
    mov edx, %2
    mov eax, sys_write
    mov ebx, stdout
    int service
%endmacro

%macro read 2 ;keyboard reading
    mov eax, sys_read
    mov ebx, stdin
    mov ecx, %1
    mov edx, %2
    int service
%endmacro

%macro indexVector 2      ;%1-> first parameter: is the current index. %2->
second parameter:
    cmp BYTE[i],%1          ;is the jump function if the current index is
equal to the value
    je %2                  ;is the function of jump, we jump to the
corresponding function.
%endmacro

%macro showInfPost 2      ;show message to enter the key to translate
write %1, %2              ;%1-> first parameter: positions fil and column
write msg1, l_msg1          ;show message corresponding to the position of the
```

```

write msg1, L_msg1      ;show message corresponding to the position of the
vector
write v_i,1              ;show the current index
write puntos, L_puntos  ;show the tow points --> we get "Enter the content of
the <index>: "
jmp intro                ;unconditional jump to function to introduce a value.
%endmacro

%macro showMessage 4      ;show message in a position
write %1, %2
write %3, %4
%endmacro

segment .data
        ;screen messages
msg0 db 27,"[1;3m",27,"[1;35m","Programa sobre CRIPTOGRAFIA."
L_msg0 equ $- msg0
msg1 db 27,"[1;34m","Introduce el contenido de la posicion ",0h
L_msg1 equ $ - msg1
puntos db ':'
L_puntos equ $- puntos
msg3 db 27,"[1;31m","¡CUIDADO! Ese valor ya existe."
L_msg3 equ $- msg3
msg4 db 27,"[1;32m","Introduce la clave (6 caracteres como maximo): "
L_msg4 equ $- msg4
msg5 db "Solucion: ",27,"[1;94m"
L_msg5 equ $- msg5
msg6 db 27,"[1;33m","Otra operacion (S|s/N)?:"
L_msg6 equ $ - msg6

        ;row and column positions
position1 db 27, "[02;02H" ;move cursor to row 2 and column 2
L_position1 equ $- position1
position2 db 27, "[04;02H" ;move cursor to row 4 and column 2
L_position2 equ $- position2
position3 db 27, "[05;02H" ;move cursor to row 5 and column 2
L_position3 equ $- position3
position4 db 27, "[06;02H" ;move cursor to row 6 and column 2
L_position4 equ $- position4
position5 db 27, "[07;02H" ;move cursor to row 7 and column 2
L_position5 equ $- position5
position6 db 27, "[08;02H" ;move cursor to row 8 and column 2
L_position6 equ $- position6
position7 db 27, "[09;02H" ;move cursor to row 9 and column 2
L_position7 equ $- position7
position8 db 27, "[10;02H" ;move cursor to row 10 and column 2
L_position8 equ $- position8
position9 db 27, "[11;02H" ;move cursor to row 11 and column 2
L_position9 equ $- position9
position10 db 27, "[12;02H" ;move cursor to row 12 and column 2
L_position10 equ $- position10

```

```

position11 db 27, "[13;02H"      ;move cursor to row 13 and column 2
L_position11 equ $- position11
position12 db 27, "[15;02H"      ;move cursor to row 15 and column 2
L_position12 equ $- position12
position13 db 27, "[17;02H"      ;move cursor to row 17 and column 2
L_position13 equ $- position13
position14 db 27, "[17;12H"      ;move cursor to row 17 and column 12
L_position14 equ $- position14
position15 db 27, "[19;02H"      ;move cursor to row 19 and column 2
L_position15 equ $- position15
hideLine times 100 db ' '
L_hideLine equ $- hideLine
            ;clear
clear db 27, "[2J"
L_clear equ $- clear
            ;Initialize content array
vKeys times 10 db '0'    ;ten position array
size equ $- vKeys
keyNumber times 6 db '0'
sizeKeyNumber equ $- keyNumber
index db 1      ;counter
i db 1       ;other ;counter
            ;Declaration variables without initialize
segment .bss
key resb 2      ;variable that stores the key that we introduce
v_i resb 1      ;index of array
resp resb 1     ;variable that stores the user's response

segment .text
global _start

_start:
        ;set to zero
        xor eax, eax
        xor ebx, ebx
        xor ecx, ecx
        xor edx, edx

        Call clearScreen
        showMessage position1, L_position1, msg0, L_msg0      ;show "Programa
sobre CRIPTOGRAFIA."

        ;initialize counters
        mov BYTE[index],0
        mov BYTE[i],0

        mov esi, vKeys ;store the first value of the position of vKeys
        mov edi, 0 ;control value
        introKey:   ;function to introduce values corresponding to the
position of the vector
            mov cl, BYTE[i] ;save index i to register
            add cl, '0' ;get corresponding ascii value

```

```

        add cl, 0      ;get corresponding ascii value
        mov BYTE[v_i], cl    ;store the ascii value in the variable v_i
        Call showIntro      ;we show the message corresponding to the
position of the vector

        intro:           ;we introduce the value/key
        read key,2   ;key reading
        mov al, BYTE[key]  ;store the key content in register al
        mov bl,BYTE[key]   ;store the key content in another register
which we will verify
                                ;with the previous elements of the vector

;check if the key entered already exists
        xor ecx, ecx    ;set register ecx to zero
        verify: ;we verify
        mov al, BYTE[vKeys+ecx]    ;we access the element of the
position vector
                                ;indicated by the memory address
of the ecx register
        cmp al, bl  ;check if the key entered is equal to any of the
previous keys
        je re_enter ;if equals we ask the user again to enter the key
in the same position
        Call incrementIndex    ;if else we increment the index to
continue comparing

        fill:           ;function to store the key entered and increment
the registers edi, esi and index i
        ;~ ;store in the corresponding position of the array
        mov [esi], bl
;increment esi, edi and index i
        inc esi
        inc edi
        inc BYTE[i]
        showMessage position12, L_position12,hideLine, L_hideLine
;hide --"¡CUIDADO! Ese valor ya existe."--
        cmp edi, size    ;verify the size of array with his current
index
        jb introKey ;if the current index is minor repeat the function
                                ;when full array
        Call introKeyNumber   ;jump function that will ask the user
to enter the key of numbers to translate

incrementIndex:    ;function increment the index to continue comparing
        inc ecx
        cmp ecx, size    ;we check that the index does not exceed the size of the
vector
        jb verify     ;if the index is not exceed the size, we compare again
ret

re_enter:          ;function, we ask the user again to enter the key in the

```

```
same position
                ;when the previously entered key already exists
showMessage position12,L_position12, msg3,L_msg3      ;"¡CUIDADO! Ese valor
ya existe."
jmp introKey

showIntro: ;function control index vector
indexVector 0,post1
indexVector 1,post2
indexVector 2,post3
indexVector 3,post4
indexVector 4,post5
indexVector 5,post6
indexVector 6,post7
indexVector 7,post8
indexVector 8,post9
indexVector 9,post10
ret

        ;show "Introduce el contenido de la posicion "
post1:
;show message for introduce the key
showInfPost position2, L_position2

post2:
;show message for introduce the key
showInfPost position3, L_position3

post3:
;show message for introduce the key
showInfPost position4, L_position4

post4:
;show message for introduce the key
showInfPost position5, L_position5

post5:
;show message for introduce the key
showInfPost position6, L_position6

post6:
;show message for intEroduce the key
showInfPost position7, L_position7

post7:
;show message for introduce the key
showInfPost position8, L_position8

post8:
;show message for introduce the key
showInfPost position9, L_position9
```

```

post9:
;show message for introduce the key
showInfPost position10, L_position10

post10:
;show message for introduce the key
showInfPost position11, L_position11

introKeyNumber:           ;we ask the user to enter the key of numbers to
translate
    showMessage position12,L_position12, msg4,L_msg4      ;show "Introduce la
clave (6 caracteres como maximo): "
    mov ebp, keyNumber
    mov edi, 0
    ciclo_lecatura: ;read the new array of numbers
        read ebp, 1
        inc ebp
        inc edi
        cmp edi, sizeKeyNumber
        jb ciclo_lecatura ;read the key the numbers to translate with a
maximum size of six characters
    Call translateKey

translateKey:             ;we translate the key numbers
    showMessage position13,L_position13, msg5,L_msg5      ;show "Solucion: "
    mov esi, keyNumber
    mov edi, 0
    write position14, L_position14
    ciclo_impresion:          ;we travel the key of numbers
        mov al,BYTE[keyNumber+edi]
        add al, 17 ;convert key to character/translate
        mov BYTE[keyNumber+edi], al
        write esi, 1 ;show the translated key
        inc esi
        inc edi
        cmp edi, 6      ;we check that the index does not exceed the size of
the key of numbers
        jb ciclo_impresion ;repeat the function when the index is less than
the size
        showMessage position15, L_position15,msg6, L_msg6      ;show "Otra
operacion (S|s/N)?:"
        read resp,2      ;continue or not
        mov al, BYTE[resp]
        cmp al, 'S'
        je keyContinue
        cmp al, 's'
        je keyContinue
        cmp al, 'N'
    Call exit

```

```
keyContinue:           ;we hide the message "Otra operacion (S|s/N)?:"  &&
"Solucion: "  && key Numbers
    write position12, L_position12
    write hideLine, L_hideLine
    write position13, L_position13
    write hideLine, L_hideLine
    write position14, L_position14
    write hideLine, L_hideLine
    write position15, L_position15
    write hideLine, L_hideLine
Call introKeyNumber      ;we re-enter the key of numbers

exit:          ;exit
    mov eax, sys_exit
    mov ebx, stdin
    int service

clearScreen:    ;clear screen
    push rcx
    push rdx

    write clear, L_clear

    pop rcx
    pop rdx
    ret
```

4. Datos de Prueba

KaliLoen X

Aplicaciones Lugares Terminal 29 de may 03:52

Terminal

Programa sobre CRIPTOGRAFIA.

Introduce el contenido de la posicion 0: A
Introduce el contenido de la posicion 1: B
Introduce el contenido de la posicion 2: C
Introduce el contenido de la posicion 3: A

```
38 jmp intro ;unconditional jump to function to introduce a value.
39 %endmacro
40
41 %macro showMessage 4 ;show
42 write %1, %2
43 write %3, %4
44 %endmacro
```

;CUIDADO! Ese valor ya existe.

```
46
47 segment .data
48
```

KaliLoen X

Aplicaciones Lugares Terminal 29 de may 03:54

Terminal

Programa sobre CRIPTOGRAFIA.

Introduce el contenido de la posicion 0: A
Introduce el contenido de la posicion 1: B
Introduce el contenido de la posicion 2: C
Introduce el contenido de la posicion 3: D
Introduce el contenido de la posicion 4: E
Introduce el contenido de la posicion 5: F
Introduce el contenido de la posicion 6: G
Introduce el contenido de la posicion 7: H
Introduce el contenido de la posicion 8: I
Introduce el contenido de la posicion 9: J

```
44 %endmacro
45
46 Introduce la clave (6 caracteres como maximo): 189427
47
48 Solucion: BIJECH
```

otra operacion (S/s/N)?: S

```
49      ;screen messages
50      rcsq0 db 27,[1;35m,[1;34m,"Programa sobre CRIPTOGRAFIA."
51      L_msg0 equ $ - msg0
52      msg1 db 27,[1;34m,"Introduce el contenido de la posicion ",0h
```

```
Introduce el contenido de la posicion 0: A
Introduce el contenido de la posicion 1: B
Introduce el contenido de la posicion 2: C
Introduce el contenido de la posicion 3: D
Introduce el contenido de la posicion 4: E
Introduce el contenido de la posicion 5: F
Introduce el contenido de la posicion 6: G
Introduce el contenido de la posicion 7: H
Introduce el contenido de la posicion 8: I
Introduce el contenido de la posicion 9: J
245          mov [keyNumber+edi], al
Introduce la clave(6 caracteres como maximo): 8765 key
Solucion: IHGF
Otra operacion (S/s/N)?: N
749          cmp edi, /      ;we check that the index does not exceed the size of the key of i
750          jbe ciclo_impresion ;repeat the function when the index is less than the size
751          showMessage position15, L_position15,msg6, L_msg6      ;show "Otra operacion (S
(program exited with code: 0)
Press return to continue
```

5. Bibliografía

Para realizar dicha práctica he hecho un curso de lenguaje ensamblador en este enlace de Youtube

Haz click



Y he acudido a la documentación del Campus Virtual

6. Opinión

En cuanto a mi opinión personal, esta práctica es fácil de entender debido a los antecedentes con CODE2 pero ha sido tedioso el tener que lidiar con el recorrido del vector, el tener que asegurarnos que no haya duplicados en el vector. En varias ocasiones cometía varios errores en cuanto al uso de registros, la asignación inmediata y directa, los bits de cada registro, etc.. Uno de los grandes problemas fue el controlar que no haya duplicados porque en principio solo me funcionaba para las dos primeras claves pero luego me permitía el almacenamiento de duplicados cosa que corregí porque me di cuenta que no hacía bien el salto a la función que incrementaba el registro contador ecx que es el que nos permitía pasar de una clave a otra anterior controlando que dicho registro no supere el tope que el tamaño del array de claves y así poder compararla con la clave actual. Otro problema fue el tener que realizar una correcta traducción ya que no convertía correctamente cada uno de los elementos del nuevo vector de números a traducir a su correspondiente clave debido al mala conversión de dichos elementos.