Explicación del funcionamiento del programa

La estructura del programa se va a basar en cuatro partes:

-La primera se encargará de iniciar y cargar los datos que nos dan en .data:

```
        .data
cad:    .asciz "Texto de prueba: cuantas letras distintas hay?"
EoD:    .ascii "E" @E para Encriptar, D para Desencriptar
sem:    .word 6 @sustituir X por un número cualquiera entre 0-31 para establecer la semilla
vec:    .word 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 @intento de vector

        .text
@-------------------------------------------------------------------------------------------------------------
@ Programa invocador
@-------------------------------------------------------------------------------------------------------------
@
@  r0= dirección de la cadena        r1= caracter actual de la cadena    r2= semilla         r3= dirección del vector      r4= "E" o "D"     r5= posición
@  r6=0xDF           r7=letraencriptada o desencriptada
@
@-------------------------------------------------------------------------------------------------------------
main:   ldr r0, =cad
        ldr r4, =EoD
        ldrb r4, [r4]
        ldr r2, =sem
        ldr r2, [r2]
        ldr r3, =vec
        mov r5, #0

```

En esta parte también nos encargamos de asegurar que el dato que viene en la etiqueta “EoD” de .data es una “E” de encriptar o una “D” desencriptar y tiene que estar en mayúsculas. Para que esté en mayúsculas, lo que hacemos es una máscara con un “and” cogiendo el registro r6 (donde hemos guardado el valor “0xDF”) y el registro r4 (donde está la letra correspondiente de la etiqueta “EoD”). Si la letra en r4 es una “E” nos iremos a la etiqueta “Enc”, de encriptar, y si la letra es una “D” nos iremos a la etiqueta “Desnc”, de desencriptar. Si por un casual, nos introducen algo que no es ni una E ni una D (excepción) , se realiza un salto incondicional a la etiqueta “error” donde se mostrará por pantalla el error y se volverá al programa principal mediante “mov pc, lr”.

```
        mov r6, #0xDF
        and r4, r6 @Para poner la letra que haya en r1 en mayúscula
        cmp r4, #'E' @Comparamos si es una E para encriptar
        beq Enc
        cmp r4, #'D' @Comparamos si es una D para desencriptar
        beq Desnc
        b error
        b fin
```

```
error:   @cuando no nos dan la letra E ni D
        @ escribir un mensaje de error en pantalla
         mov pc, lr
```

-La segunda parte del problema se compone de los whiles que se van a usar para encriptar o desencriptar, dependiendo de la letra que haya en la etiqueta “EoD” corresponden a las acciones de desencriptar y encriptar.

Si la letra es una “E”, nos vamos a ir a la etiqueta “Enc”. Esta etiqueta nos lleva a un while que se va a encargar de recorrer toda la cadena que queremos encriptar, de tal forma que repetimos el while hasta que el carácter actual de la cadena sea igual

a 0 (ya que la cadena está guardada en .asciz y cuando esta cadena acaba se guarda un 0).

Dentro de este while, nos encontramos primero con un if/else donde nos aseguramos de que la letra actual en la que estamos es una letra mayúscula que se encuentra entre la “A” y la “Z” (volvemos a hacer una máscara con r6 y esta vez con r1 ya que es el registro donde guardamos el carácter actual de la cadena).

Por tanto, “if” letra actual es menor que A o mayor que Z, vamos a realizar un salto a la etiqueta “noletra” donde mostraremos por pantalla la letra actual en la que estamos y la posición donde nos encontramos en la cadena.

Si el if anterior no se cumple, eso quiere decir que nuestra letra actual es una letra y por tanto vamos a llamar a la subrutina “encriptar” de la cual hablaremos más tarde. Es importante que antes de entrar a la subrutina, guardemos los valores de entrada que vamos a modificar del programa principal (que son r0 y r2) mediante un push{} . Después de encriptar la letra actual (que está en r2), hacemos un pop{r0} para devolver los valores que había en r0 antes de entrar a la subrutina, guardamos el contenido de r2 en r7 (está libre) y luego ya podemos hacer otro pop{r2} para devolver a r2 el contenido de las semillas. Es importante hacer en este orden el pop{} ya que primero se desapila el registro con el número menor (r0) y luego el mayor (r2).
En este mismo else, tenemos que modificar el valor de la semilla sumándole 1. El problema es que la semilla sólo puede llegar al valor 31 ya que el vector tiene 32 posiciones, y para esto hacemos otro if/else: si la semilla vale 31, modificamos su valor por 0 (se ha “sumado 1”); si no, la semilla no ha llegado todavía a su límite y podemos sumarle 1 tranquilamente.

Por último, modificamos el valor de la posición de la cadena y cambiamos el carácter actual por el siguiente carácter de la cadena.

Este sería el planteamiento en python:

```
Enc:
        @hacemos un while para recorrer toda la cadena que queremos encriptar
        @El problema en python sería:
        @
        @ semilla = 6
        @ vector = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]
        @ cadena = "Texto de prueba: cuantas letras distintas hay?"
        @ posición = 0
        @ caracteractual= direcciónцadena[posición]
        @ cadSolución= []
        @
        @ while caracteractual != 0:
        @
        @       if caracteractual < "A" and  caracteractual > "Z":          ( el caracter actual no es una letra)
        @               print ( caracteractual + posición )
        @
        @
        @       else:     (si es una letra empezamos a encriptar)
        @
        @               letraencriptada= encriptar(semilla, vector, caracteractual)
        @               cadsolución += letraencriptada
        @
        @               if semilla = 31:
        @                       semilla = 0
        @
        @               else:
        @                       semilla += 1           (para que pase al siguiente contenido del vector)
        @
        @       posición += 1
        @       caracteractual= cadena[posición]        (cogemos el siguiente caracter de la cadena)
```

Y este en ensamblador:

```
Enc:
        ldrb r1, [r0, r5]     @el caracter actual de la cadena se guarda en r1
while1: cmp r1, #0       @lo comparamos con cero ya que la cadena está guardada en ascii y esp quiere decir que cuando lleguemos a el caracter 0 se habrá acabado la cadena
        beq finWhile1
        and r1, r6     @r6 = 0xDF
if1:    cmp r1, #'A'
        blt noletra
        cmp r1, #'Z'
        bgt noletra
        b else1
noletra:
        @imprimimos en el simulador de pantalla la posición (r5) y imprimimos el caracter actual (r1)
        b finIE
else1:
        push{r0,r2}
        bl encriptar
        pop{r0}
        ldrb r7, [r2]    @guardamos la letra encriptada en r7
        @tenemos que guardar la letra encriptada en un word o en una cadena que será la cadena solución
        pop{r2}

if2:    cmp r2, #31   @la semilla solo puede llegar hasta 31
        beq limite
        b else2
limite: mov r2, #0
        b FinIE2

else2:  add r2, #1

finIE2:
finIE:
        add r5, #1
        ldr r1, [r0, r5]
        b while1
```

Si la letra de la etiqueta “EoD” era una “D” , vamos a ir a la etiqueta “Desnc”. En esta etiqueta volvemos a tener un while que va a recorrer la cadena que queremos desencriptar (vamos a suponer que la cadena está bien encriptada y que no se encriptaron, ni espacios, ni números decimales, ni nada distinto a una letra).
Antes de empezar este while ,que va a tener la misma condición que el while que usamos para encriptar, guardamos el r1 el primer carácter de la cadena.

Y ahora, como no tenemos que comprobar nada más, nos ponemos a desencriptar llamando a la subrutina "desencriptar” (antes de llamarla hacemos un push de r0 y

r2 como hicimos en la subrutina “encriptar” ya que vamos a usar esos registros en la subrutina). Más tarde especificamos que hacemos en esta subrutina.

Después de desencriptar el carácter de la cadena, modificamos el valor de la semilla tal y como hicimos cuando encriptamos, y luego sumamos a la posición de la cadena un 1 y ponemos como carácter actual al siguiente carácter que hay en la cadena.

Este sería el planteamiento en python:

```
@hacemos un while para recorrer toda la cadena que queremos desencriptar
@El problema en python sería:
@
@ semilla = 6
@ vector = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]
@ cadena = "Texto de prueba: cuantas letras distintas hay?"
@ posición = 0
@ caracteractual= direcciónc adena[posición]
@ cadSolución= []
@
@ while caracteractual != 0:
@
@       letradesencriptada= desencriptar(semilla, vector, caracteractual)
@       cadsolución += letradesencriptada
@
@       if semilla = 31:
@               semilla = 0
@
@       else:
@               semilla += 1            (para que pase al siguiente contenido del vector)
@
@       posición += 1
@       caracteractual= cadena[posición]        (cogemos el siguiente caracter de la cadena)
```

Y este en ensamblador:

```
Desnc:
        ldrb r1, [r0, r5]     @el caracter actual de la cadena se guarda en r1
while2: cmp r1, #0      @lo comparamos con cero ya que la cadena está guardada en ascii y esp quiere decir que cuando lleguemos a el caracter 0 se habrá acabado la cadena
        beq finWhile2
        and r1, r6      @r6 = 0xDF

        push{r0,r2}
        bl desencriptar
        pop{r0}
        ldrb r7, [r2]    @guardamos la letra encriptada en r7
        @tenemos que guardar la letra encriptada en un word o en una cadena que será la cadena solución
        pop{r2}

if3:    cmp r2, #31   @la semilla solo puede llegar hasta 31
        beq limite2
        b else3
limite2:        mov r2, #0
        b FinIE3

else3:  add r2, #1

finIE3:
        add r5, #1
        ldr r1, [r0, r5]
        b while2


finWhile2:      b seguir
```

- La tercera parte del programa se corresponde a las subrutinas, que son las responsables de encriptar y desencriptar los caracteres de la cadena.

  En la subrutina “encriptar”, cargamos el valor que hay en la posición (el valor de la semilla, r2) del vector (r3) en r0 (r0 en el programa principal correspondía a la dirección de la cadena por eso hemos usado un push{r0}). Luego le sumamos al carácter actual de la cadena (r1) el valor que hay en r0

y lo guardamos en r2 (r2 en el programa principal correspondía a la semilla, por eso también introducimos en el push{} r2, quedando push{r0, r2}).
Por último solo tenemos que poner la instrucción “mov pc, lr” para volver donde nos quedamos cuando llamamos a la subrutina.

Este sería el planteamiento en python:

```
@ def encriptar(semilla, vector, caracteractual):
@       contenidovector= vector[semilla]
@       letradesencriptada= caracteractual + contidovector
@       return letradesencriptada
```

Y este en ensamblador:

```
186 encriptar:
187         ldr r0, [r3, r2]
188         add r2, r0, r1
189         mov pc, lr
```

En la subrutina “desencriptar” hacemos lo mismo que en la subrutina ”desencriptar” con la diferencia que ahora restamos el valor de r0 con r1 (carácter actual) y lo guardamos en r2.

Este sería el planteamiento en python:

```
@ def encriptar(semilla, vector, caracteractual):
@       contenidovector= vector[semilla]
@       letradesencriptada= caracteractual - contenidovector
@       return letradesencriptada
```

Y este en ensamblador:

```
209 desencriptar:
210         ldr r0, [r3, r2]
211         sub r2, r0, r1
212         mov pc, lr
```

-La última parte de la estructura sería ya mostrar en pantalla la cadena encriptada o desencriptada en el simulador que dispone QtARSim y ya tendríamos el programa hecho.

```
8 seguir:
9 @ ya tenemos la cadena solución que estará encriptada o desencriptada dependiendo lo que nos hayan pedido.
0 @ Ahora hay que monstrar en pantalla la cadena y ya estaría el problema
1 @
2 @ El problema en python sería:
3 @ print( cadsolución )
4
5 fin:    wfi
```