Universidad de Guadalajara

Centro universitario de ciencias exactas e ingeniería división de electrónica y computación

Seminario de solución de problemas de Arquitectura de Computadoras

Reporte

Proyecto cifrado Vigenère

Alumno: Oswaldo Luna Grados

Código: 211718256

Sección: D14

Profesor: J. Ernesto López Arce Delgado

Introducción

La necesidad de esconder información es algo que se ha llevado a cabo en mucho tiempo atrás. Sobre todo en la guerra cuando se envía un mensaje de un ejército a los altos mandos o gobernantes, si este mensaje era interceptado por el enemigo se podía perder la victoria.

En la computación se adoptó la criptografía pues estaba la necesidad de encubrir información en las redes o el Internet. Como es una compra de una laptop en Amazon se necesita que solo los dos sepan la información delicada.

El cifrado Vigenère es un criptosistema por sustitución polialfabética periódica línea. Inventado por el criptógrafo francés Blaise de Vigenère en el siglo XVI.

El cifrado está basado en el cifrado cesar donde al igual que este hay un texto simple donde el carácter o letra los cuales forman una tabla.

La tabla es una matriz de 27 x 27 (si es solo letras) y no tiene una posición de inicio fácil de encontrar. Si seguimos con solo letras dentro de la tabla el primer renglón inicia de la A el segundo desde la B y así con todas las columnas.

Si se le asigna un número a cada letra por lo que se le asigna una posición diferente a cada letra en las columnas y renglón.

El cifrado Vigenère recibe un mensaje y una llave a usar para concatenar y dar un nuevo mensaje simétrico. Cada letra del abecedario tiene un valor numérico y consiste en hacer una suma, letra por letra del mensaje con la llave, cuando llegamos al límite se empieza la llave otras vez hasta terminar con el ultimo carácter del mensaje.

Los símbolos en el mensaje son las letras del alfabeto donde k= $\{k_0, k_1, \ldots, k_{D-1}\}$ donde D es la longitud. La siguiente forma expresa la transformación lineal de cifrado:

$$E(k_i) = (k_i + X_i) mod\ L$$

Donde k_i es la letra en el texto a cifrar en la posición de i, X_i es el carácter de la llave en la misma posición de carácter del texto a cifrar. Y L es el tamaño del alfabeto o el tamaño del código ASCII. Y para descifrar se resta el valor de la llave del mensaje encriptado. La función seria la siguiente:

$$E(k_i) = (k_i - X_i) mod\ L$$

El código diseñado en java es un ejemplo de cómo realizar el cifrado y descifrado de una cadena de byte. Este código es el que se muestra en el código 1.

```
9  public class Vigenere {
10     private byte Encriptar(byte a,byte codigo){
11         int z =  (a+codigo);
12         byte c = (byte)(z%256);
13         return c;
14
15     }
16     private byte DesEncriptar(byte a, byte codigo){
17         int z = (a-codigo);
18         byte c = (byte) (z%256);
19         return c;
20     }
21     public byte[] Encriptar(byte[] mensaje,byte[] key){
22         int i;//1
23         int sizeKey = key.length;
24         int size = mensaje.length;
25         byte[] z = new byte[size];
26         for(i=0;i<size;i++){
27             z[i] = Encriptar(mensaje[i],key[i%sizeKey]);
28         }
29         return z;
30
31     }
32     public byte[] DesEncriptar(byte[] mensaje,byte[] key){
33         int i;
34         int sizeKey = key.length;
35         int size = mensaje.length;
36         byte[] z = new byte[size];
37         for(i=0;i<size;i++){
38             z[i] = DesEncriptar(mensaje[i],key[i%sizeKey]);
39         }
40         return z;
41     }
42
43 }
```

Código 1. Cifrado y descifrado Vigenère en java de una palabra y por carácter.

Objetivo general

Se busca implementar un cifrado de Vigenère tanto encriptación de una palabra como desencriptación de está. El cifrado se ejecutara como lo que sería el código java en el código 1, se traducirá a ensamblador y de este a bits. Se implementara las instrucciones tipo J, I y R para realizar el cifrado junto con una subrutina. Además que en lugar de solo combinar números y letras se optó por usar todos los caracteres ASCII. Por lo que será de 256.

Objetivos específicos

Objetivo a implementar

Inicializar banco con las celdas del banco en 0.

Ingresar la palabra “Encriptar” en el banco de registros, la cual se dividirá cada letra en diferentes celdas dentro banco y se guardara en formato tipo ASCII. Desde el registro 0 hasta el 8. Lo cual en el formato decimal seria 69, 110, 99, 114, 105, 112, 116, 97, 114.

Guardar en memoria secundaria la palabra a encriptar. En las celdas 0 hasta el 8.

Se Ingresara en el banco de registro la llave llamada “Llave” dividida en cada celda. La cual se guardara en los registros 10 al 14. Y en decimal es 76, 108, 97, 118, 101.

Después se realiza una suma de cada letra la palabra a encriptar con la llave y se guardara en los registros 20 al 28, en el caso de cuando se llegue a la letra ‘i’ del mensaje a encriptar y la letra ‘e’ se empezara a sumar la palabra donde sigue con el inicio de la llave y así con cada letra. Es decir ‘p’ se sumara con ‘L’ y así hasta completar la palabra a encriptar.

Se entrara a una subrutina que hará modulo cada letra de la suma con 256 pues es número de caracteres dentro del código ASCII. Para indicar que es una subrutina comprueba al final si el registro 30 es igual al 31. El cual el 30 será 256 después de sumar llave con mensaje y 31 será igual a 256 después de restar la llave del mensaje encriptado. Si los registros son iguales saltara para guardar el mensaje desencriptado, si no guardara el mensaje encriptado. Este resultado será guardado en los registros. 0 al 8. Lo que remplazara en el banco de registro la palabra a encriptar con la palabra encriptada. Que quedaría como 145, 218, 196, 232, 206, 188, 224, 194, 232 en forma decimal. Después se guardara la palabra encriptada en la memoria secundaria dividida por letras en cada celda. Desde la celda 20 al 28.

Se restara cada letra de la llave a cada letra de la palabra encriptada. Si se llega acabar la llave se empezara a restar desde el inicio de la llave hasta terminar de restar en toda la palabra encriptada. Se guardara el resultado en los registro 20 al 28.

Se saltara a la subrutina del módulo donde se realizara la operación de modulo a la resta de la llave a la palabra encriptada.

Se guardara la palabra desencriptada en la memoria secundaria. En las celdas 30 a 38.

Código ensamblador

el siguiente código en ensamblador son las operaciones para realizar la encriptación y desencriptación de la palabra. Además de guardarlo dentro de la memoria en el procesador desarrollado. El cual realiza addi, sw, add, mod, beq, sub y j al menos una vez.

```
//Guardar en banco de registros la palabra encriptar
//Donde la celda 15 tiene 0
addi $t0,$t15, 69
addi $t1,$t15, 110
addi $t2,$t15, 99
addi $t3,$t15, 114
addi $t4,$t15, 105
addi $t5,$t15, 112
addi $t6,$t15, 116
addi $t7,$t15, 97
addi $t8,$t15, 114
//Guardar en memoria Encriptar y el tamaño de la palabra
sw $t0, 0
sw $t1, 1
sw $t2, 2
sw $t3, 3
sw $t4, 4
sw $t5, 5
sw $t6, 6
sw $t7, 7
sw $t8, 8
//Llave
addi $t10, 76
addi $t11, 108
addi $t12, 97
addi $t13, 118
addi $t14, 101
//instruccion para corregir error de inmediata
add $t18,$t16,$t17
//suma de llave y palabra
add $t20,$t0,$t10
add $t21,$t1,$t11
add $t22,$t2,$t12
add $t23,$t3,$t13
add $t24,$t4,$t14
add $t25,$t5,$t10
add $t26,$t6,$t11
```

```
add $t27,$t7,$T12
add $t28,$t8,$t13
// crear subrutina  para comprobar ubicacion de instruccion
addi $t30,$t15, 256                    //actual instruccion 33
// la celda 29 contiene 256
addi $t29,$t15, 256                //140
//instruccion  para corregir error de inmediata
add $t18,$t16,$t17
// opeacion de modulo de palabra
mod $t0,$t20,$t29
mod $t1,$t21,$t29
mod $t2,$t22,$t29
mod $t3,$t23,$t29
mod $t4,$t24,$t29
mod $t5,$t25,$t29
mod $t6,$t26,$t29
mod $t7,$t27,$t29
mod $t8,$t28,$t29
// comprobar si esta modulando  la parabra a encriptar o desencriptar
beq $t30,$t31,22                   //instruccion  46
//Guardar  palabra encriptada
sw $t0, 20
sw $t1, 21
sw $t2, 22
sw $t3, 23
sw $t4, 24
sw $t5, 25
sw $t6, 26
sw $t7, 27
sw $t8, 28
//instruccion  para corregir error de inmediata
addi $t30,$t15, 256
add $t18,$t16,$t17
//restar  llave  de la palabra encriptada
sub $t20,$t0,$t10            //instruccion  54
sub $t21,$t1,$t11
sub $t22,$t2,$t12
sub $t23,$t3,$t13
sub $t24,$t4,$t14
sub $t25,$t5,$t10
sub $t26,$t6,$t11
sub $t27,$t7,$T12
sub $t28,$t8,$t13
```

```
//ir a subrutina de modulo de palabra
addi $t31,256
j 34                                    //instruccion 64
//Guardar palabra desencriptada
sw $t0, 30
sw $t1, 31
sw $t2, 32
sw $t3, 33
sw $t4, 34
sw $t5, 35
sw $t6, 36
sw $t7, 37
sw $t8, 38
//ultima instruccion es 77, contando subrutina es la 88
```

Código 2. Encriptación y desencriptación Vigenère en ensamblador.

Como se observa se guardara en memoria la palabra a encriptar, ya encriptada y desencripata, además de realizar una subrutina que realiza la operación modulo para encriptar y desencriptar el mensaje sin sobre escribir otra vez la misma operación, esta utiliza unos registros, la instrucción j y beq. Dentro del registro 29 está el número 256 para realizar el modulo. Y los registros 30 y 31 reciben 256 para comprobar si debe realizar la instrucción beq, en el caso de que 31 tenga 256 realizara el salto si no solo seguirá la siguiente instrucción. Se crea instrucciones extras puesto que las instrucciones inmediatas y de guardado realizan más en su tiempo en el ciclo y causan errores en las demás instrucciones.

Desarrollo

Implementaciones nuevas al procesador monociclo.

Se implementó los siguientes módulos para poder ejecutar el J y beq dentro del procesador.
Dentro de los nuevos componentes están 2 multiplexores, un sumador y recorrer dos bits a la izquierda, además de agregar en la unidad de control las funciones para reconocer estos nuevos componentes.
En la imagen 1 se muestra el código del módulo del multiplexor que recibe del sumador de 4 bits, sumador con recorrido de 2 bits a la izquierda, de la ALU la bandera de cero y un indicador de la unidad de control para indicar si se realiza una instrucción beq o una instrucción diferente. El cual realiza una and de la entrada cero y del de la unidad de control para seleccionar entre el valor de sumador de 4 o el recorrido de 2 bits a la izquierda.

```
`timescale 1ns /1ps
module MuxPC(
    input [31:0] inAdd4,inAdd2bits,
    input zero,branch,
    output reg[31:0] outMuxPC2
);
wire sel;
assign sel=zero&branch;
always @(*)
begin
    if(sel == 1) outMuxPC2=inAdd2bits;//Elige el sumador de sign + sumador4bits
    else outMuxPC2=inAdd4;//Elige solo sumador4bits
end
endmodule
```

Imagen 1. Multiplexor para instrucciones beq.

Como se ve en la imagen 2 es el módulo del multiplexor 2 que funciona para reconocer las instrucciones tipo J. donde además de ser un multiplexor, realiza un recorrido de 2 a la izquierda y en los bits 31 a 28 son recibidos del sumador de 4 bits. El seleccionar lo recibe de la unidad de control el cual decide si es una instrucción j o de otro tipo.

```
`timescale 1ns /1ps
module MuxPC_2(//incluir shift left //para type J
    input [3:0] inAdd4,
    input [31:0] inMuxPC,
    input [25:0] inInstru,
    input selCtrl,
    output reg[31:0] outPC
);
wire [31:0] shiftLeft;
wire [31:0] resul;
assign resul[31:28]=inAdd4;
assign resul[27:0]=0;
assign shiftLeft=((32'd0+inInstru)<<2)+(resul+32'd0);

always @(*)
begin
    if(selCtrl == 1) outPC=shiftLeft;//Elige instr 25:0 con recorrido a la izq con bits de sumador4bits
    else outPC=inMuxPC;//Elige multiplexor PC1
end
endmodule
```

Imagen 2. Multiplexor para instrucciones J.

En la imagen 3 está un módulo donde se implementa un sumador de la extensión de signo y la instrucción leída. Realizando un recorrido de 2 a la izquierda de los bits de la extensión de signo.

```
`timescale 1ns /1ps
module add2bits(
    input [31:0]inAdd,in2bits,
    output reg [31:0]out
);
always@(*)
begin
    out=inAdd+(in2bits <<2);
end
initial out=0;
endmodule
```

Imagen 3. Módulo de sumador y recorrido de 2 a la izquierda.

Dentro de los cambios en los anteriores en los módulos, esta los casos para beq y J. como se ve en la imagen 4 dentro de la unidad de control.

```
27          //Type J
28          6'd2://J jump
29          begin
30              selControl=0;//no importa
31              enW_Bank=1; //lectura
32              selMuxMem_ALU=1;//no importa
33              selMuxSign_Bank=1;//no importa
34              selMuxAddr=0;//Elige segunda direccion
35              enW_Mem=0;
36              enR_Mem=0;
37              branch=0;
38              selMuxPC2=1;//Elige 25:0 de instruccion
39          end
40          //Type I
41          6'd4://beq
42          begin
43              selControl=1;//restar si son iguales resul=0
44              enW_Bank=1; //leer
45              selMuxMem_ALU=1;//no importa
46              selMuxSign_Bank=0;//Elige DR2
47              selMuxAddr=0;//no importa
48              enW_Mem=0;
49              enR_Mem=0;
50              branch=1;//funcion de beq
51              selMuxPC2=0;
52          end
```

Imagen 4. Modulo unidad de control funciones agregadas.

Dentro del TopLevel se hizo la siguiente conexión de módulos nuevos como se observa en la imagen 5.

```
96    //nuevas funciones
97    MuxPC inst_MuxPC(
98        .inAdd4(add4_PC),
99        .inAdd2bits(add2_MuxPC),
100       .zero(ZeroALU_And),.branch(Ctrl_And),
101       .outMuxPC2(MuxPC1_muxPC2)
102   );
103   MuxPC_2 inst_MuxPC2(//para type j
104       .inAdd4(add4_PC[31:28]),
105       .inMuxPC(MuxPC1_muxPC2),
106       .inInstru(instr[25:0]),
107       .selCtrl(Ctrl_MuxPC2),
108       .outPC(MuxPC2_PC)              //hacer salida para ver funcion en TB
109   );
110   add2bits inst_add2(
111       .inAdd(add4_PC),
112       .in2bits(outSign_MuxALU),
113       .out(add2_MuxPC)
114   );
```

Imagen 5. TopLevel conexiones dentro de los modulos nuevos.

Desarrollo de cifrado y desifrado Vigenère

Puesto que no se puede iniciar el banco de registro en 0 desde el modulo se inicializa en el testbench con las entradas extras en el TopLevel. En la imagen 6 muestra el Testbench donde se inicializa el banco de registro con un tiempo de 20 ns y empieza la señal de reloj hasta 178 veces. En un tiempo de 40 ns.

```
27  );
28  initial begin
29      //Iniciamos el banco de registros en cero
30      sel=0;
31      EWIniciar=0;//poner en modo escritura
32      for(i=0;i<32;i=i+1)
33      begin
34          dirIniciar=i;
35          escribir=0;
36          #20;
37      end
38      //iniciamos las instrucciones
39      sel=1;
40      #(40)clk=0;
41      i=0;
42      while(i<=178)
43      begin
44          #(80)clk=!clk;
45          i=i+1;
46      end
47      $stop;
48  end
```

Imagen 6. Testbench inicializar módulo de pruebas.

En la imagen 7 se observa el ware de cómo se inicia cada celda en 0 en el banco de registro.

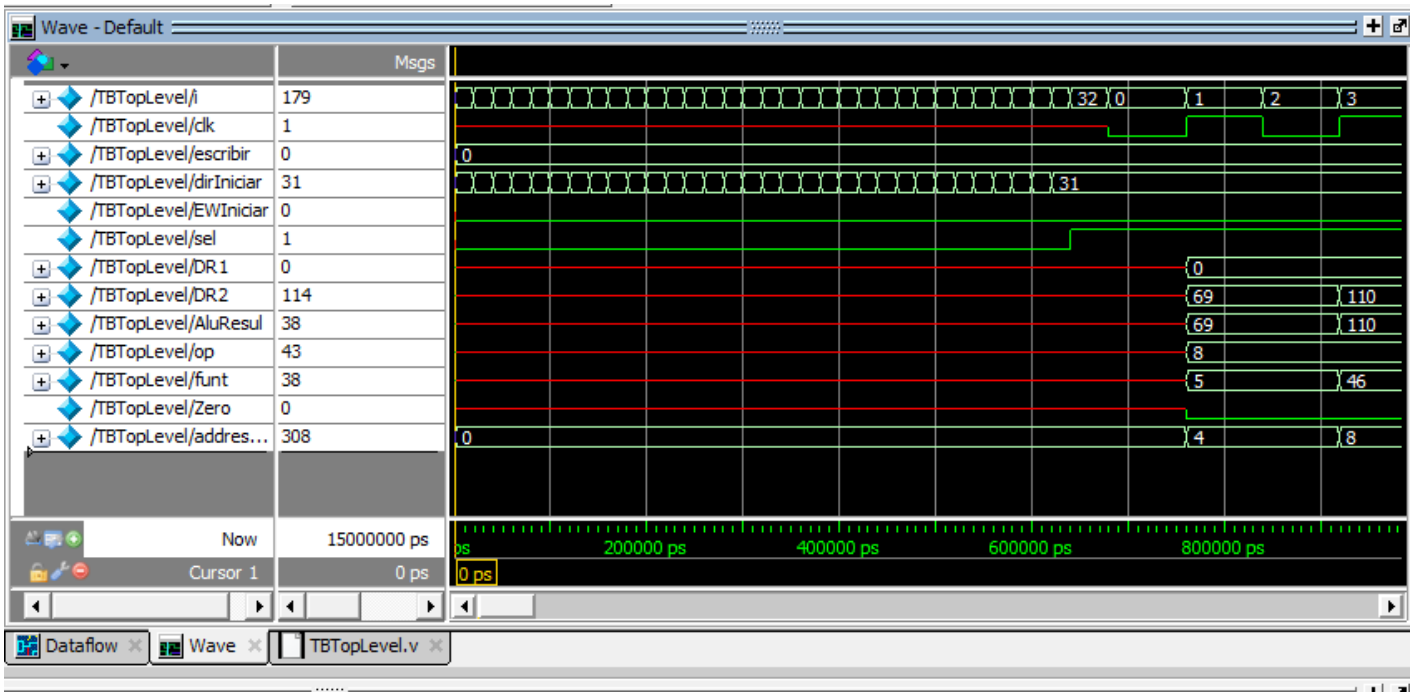

Imagen 7. Wareform inicializando banco de registro.

Lo siguiente es iniciar las instrucciones mediante la señal de reloj. Las primeras instrucciones son ingresar cada letra en cada registro de la palabra “Encriptar” que en ASCII son los números mostrados en la salida llamada DR2 Según la imagen 8. Además se observa en formato ASCII.

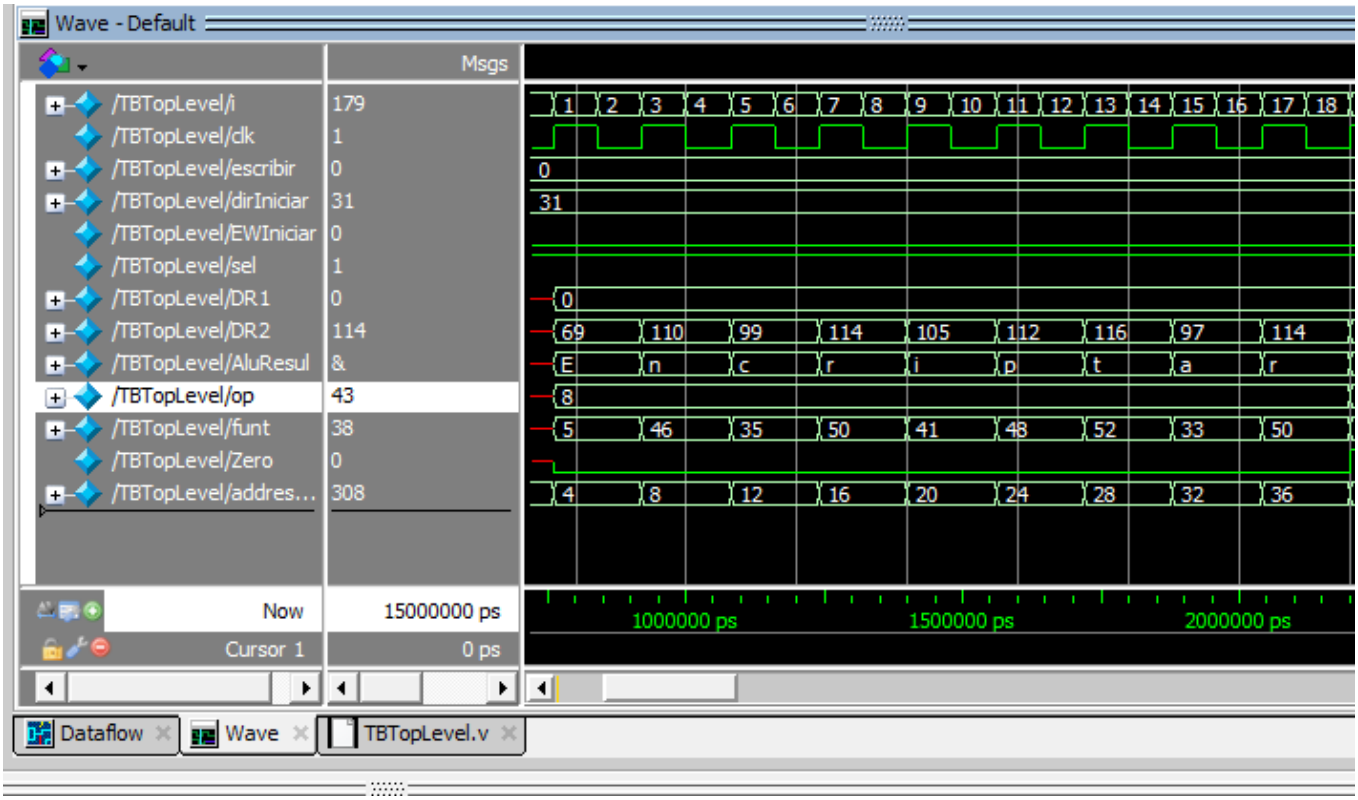

Imagen 8. Ingresando “Encriptar” a banco de registros mostrando en formato ASCII.

Como se observa se puede ver en la última salida el numero en el contador del pc de la siguiente instrucción. También esta una para ver el tipo de instrucción y si es de tipo R mostrara su función. También hay un indicador de que en la ALU está en alto la bandera de cero. Se muestra la salida de DR1 y DR2 del banco de registros.

Las siguientes instrucciones son guardar la palabra en la memoria secundaria. En las celdas 0 al 8 dentro de esta. En la imagen 9 se muestra el ware de estas instrucciones sw y en la imagen 10 se observa las celdas dentro de la memoria secundaria.

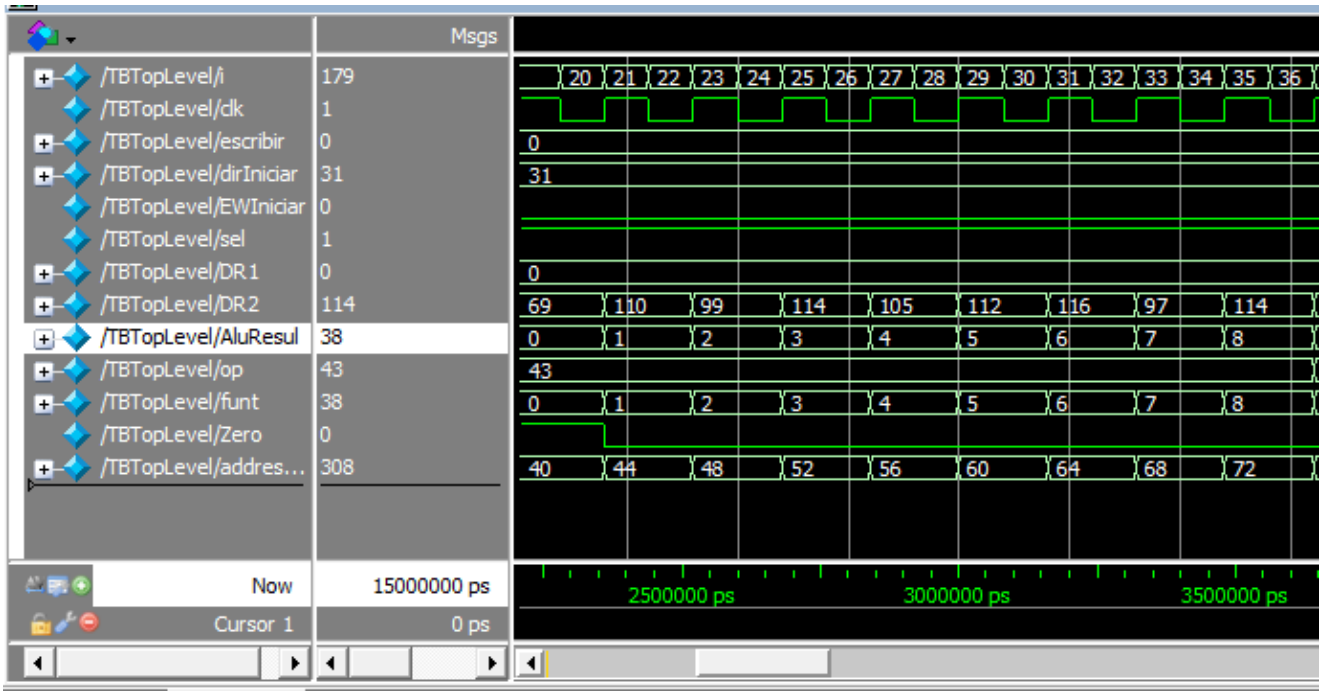

Imagen 9. Ware de las instrucciones SW para guardar el mensaje a encriptar.

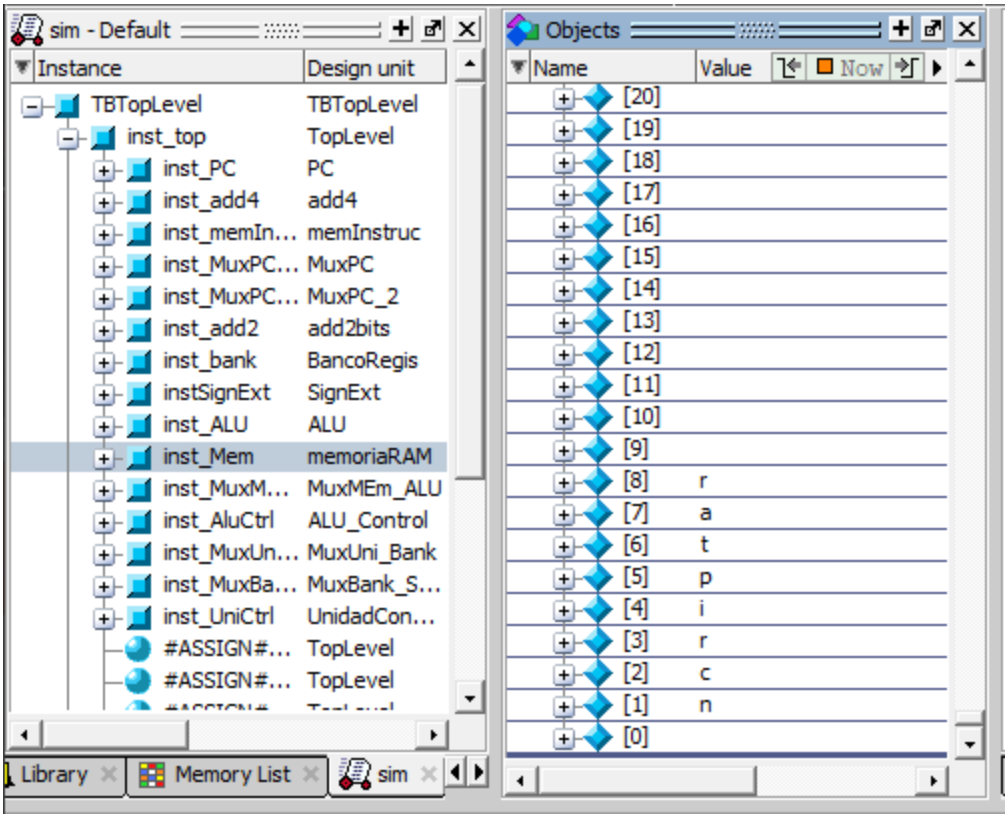

Imagen 10. Memoria secundaria mostrando guardado de mensaje a encriptar.

El siguiente paso es ingresar en el banco de registros la llave con el mensaje “Llave”, donde es guardado en las celdas 10 al 14. En la imagen 11 se observa en la salida AluResul las letras de la llave.

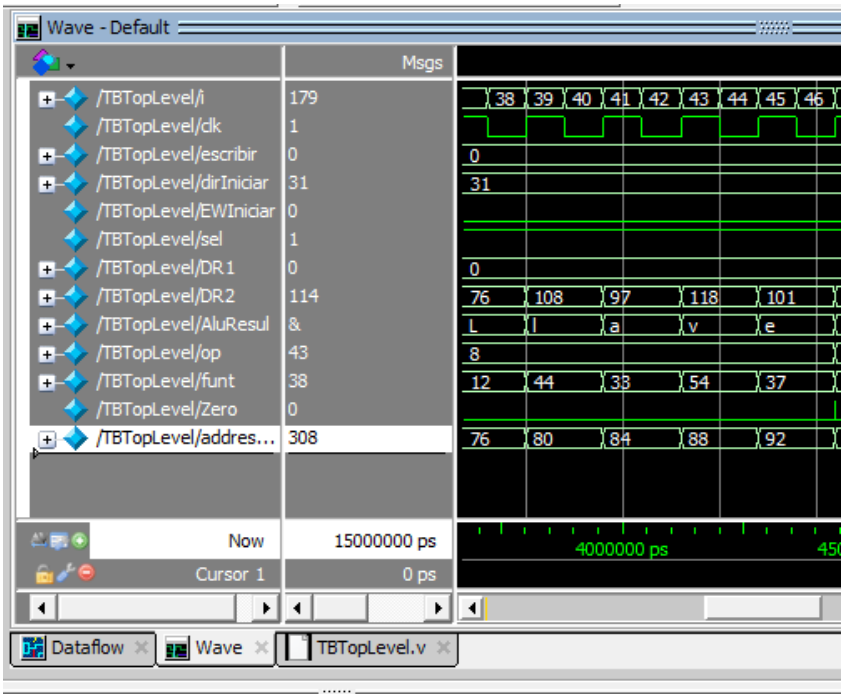

Imagen 11. Ingresando llave a banco de registros.

Después que tenemos la llave y mensaje lo siguiente será sumar letra con letra de estos dos. Lo cual termina como la imagen 12 donde el resultado será guardado en la dirección 20 al 28. Y como se ve cuando se termina las letras en la llave se suma donde se quedó el mensaje con la primera letra de la llave.

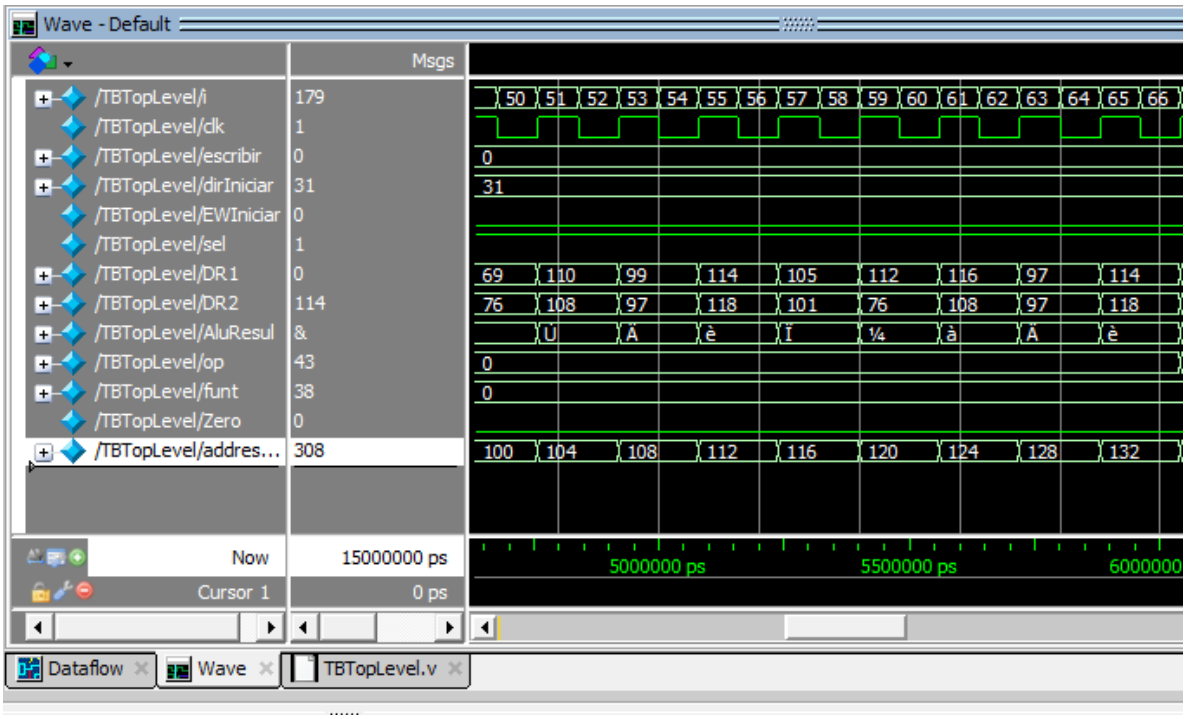

Imagen 12. Instrucciones de suma de mensaje y llave enviando los valores a las celdas 20 al 28.

Al finalizar las instrucciones de suma realizamos una suma inmediata que servirá para comparar para comprobar si se saltara o seguirá el flujo normal de instrucciones. Esta se guardara en la celda 30, con un valor de 256 al igual que la celda 29 la cual se usara para realizar las operaciones modulo.

En las instrucciones modulo se saca el módulo de las celdas 20 al 28 con 256 para no salirse del código ASCII. Se muestra estas acciones en la imagen 13.

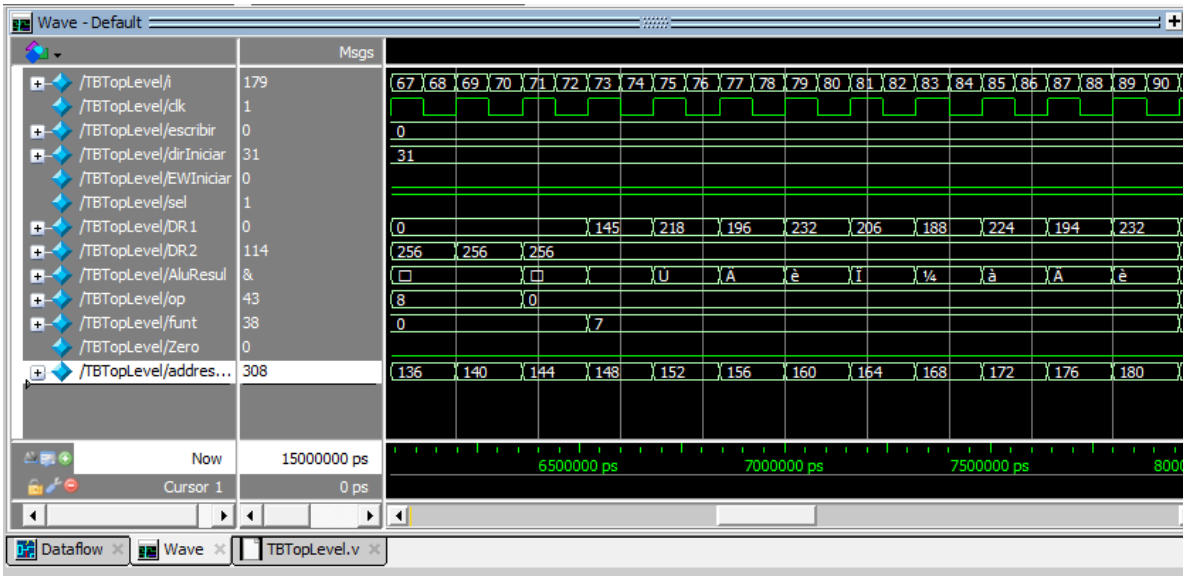

Imagen 13. Subrutina de operación de modulo. Usado para encriptar.

Al terminar las operaciones tipo modulo se realiza una instrucción tipo beq para comprobar si se salta o no para terminar la encriptación o desencriptación.

Las siguientes instrucciones es guardar el mensaje encriptado en la memoria secundaria en las celdas. En la imagen 14 está la forma en ASCII en la memoria.

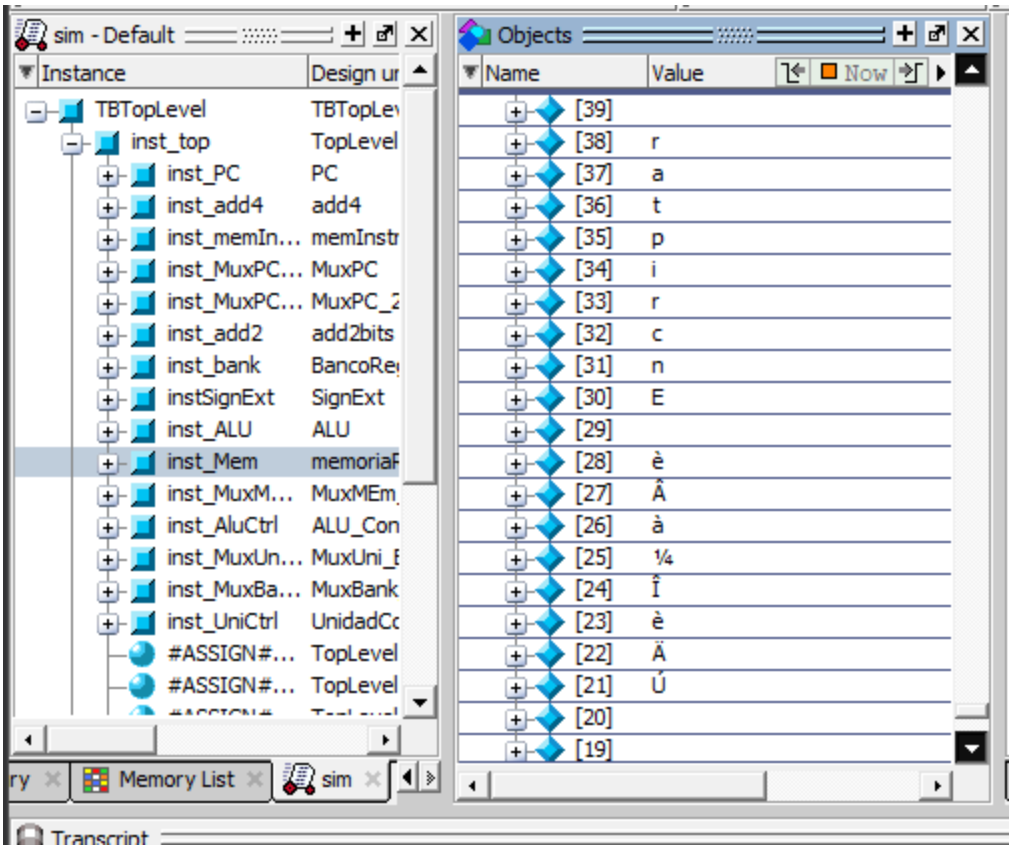

Imagen 14. Memoria secundaria mostrando celdas 20 a 28 el mensaje encriptado.

Seguido de estas instrucciones restamos del mensaje encriptado la llave donde si se termina las letras en la llave, en la letra donde se quedó se le resta la primera letra de la llave. La imagen 15 muestra el ware de estas instrucciones.

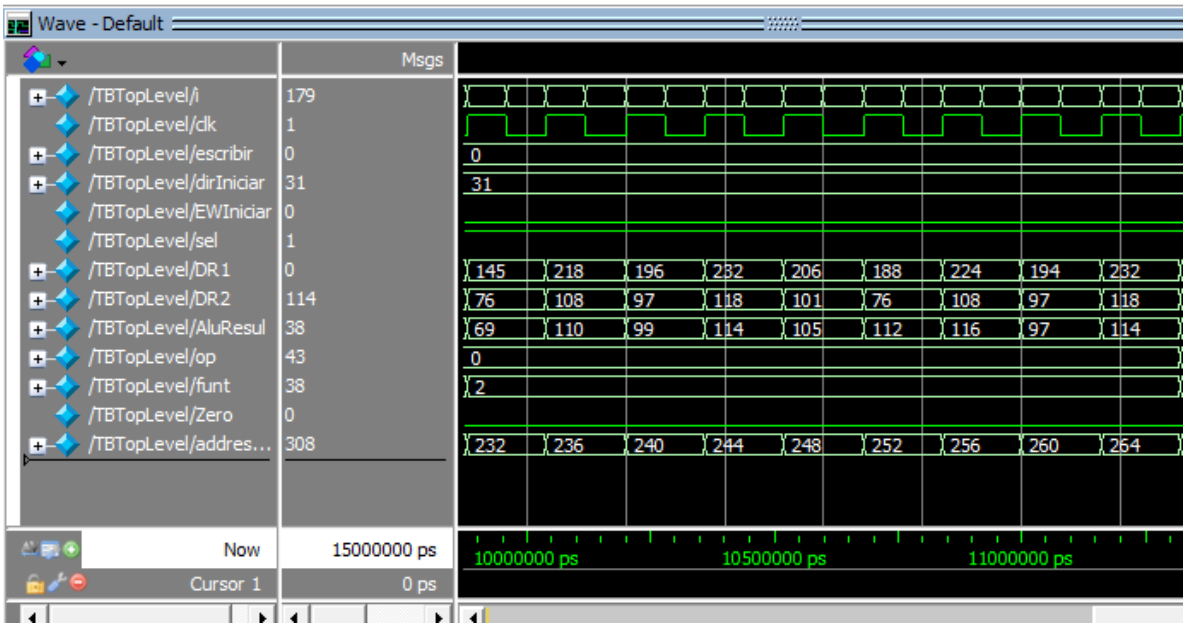

Imagen 15. Instrucciones para restar la llave del mensaje encriptado.

Después de estas acciones a la celda 31 del banco de registros ingresamos 256, para que cuando saltemos en la siguiente instrucción vuelva para guardar el mensaje desencriptado. Las imágenes 16 y 17 muestran como las instrucciones están salteadas donde la dirección 268 pasa a ser 136 y realizar las operaciones modulo pasadas para desencriptar el mensaje.

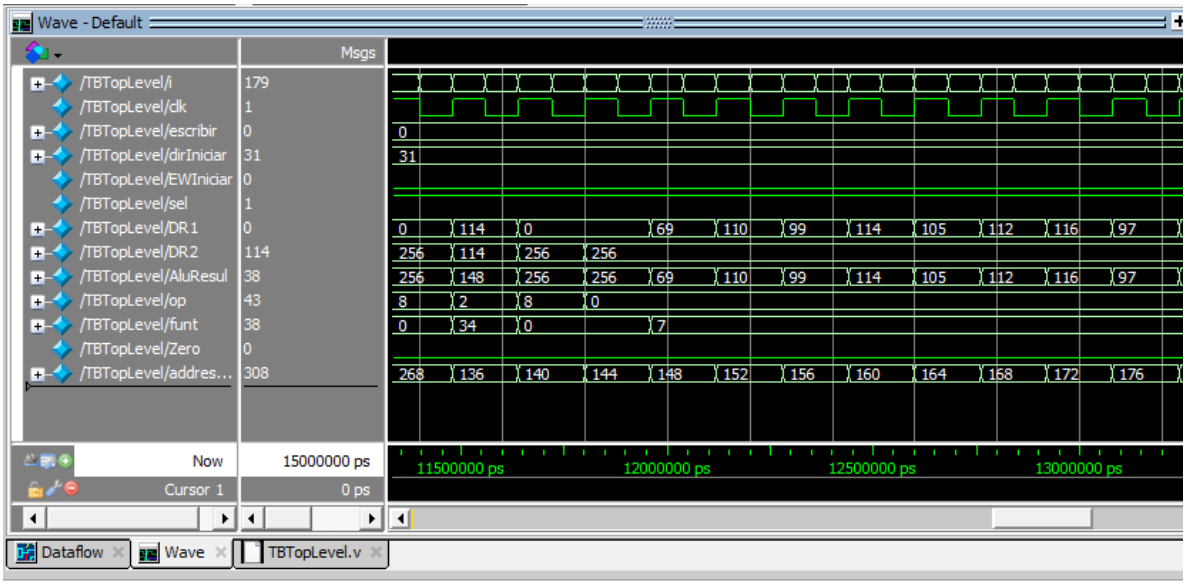

Imagen 16. Instrucción de salto y subrutina de modulo para desencriptar mensaje.

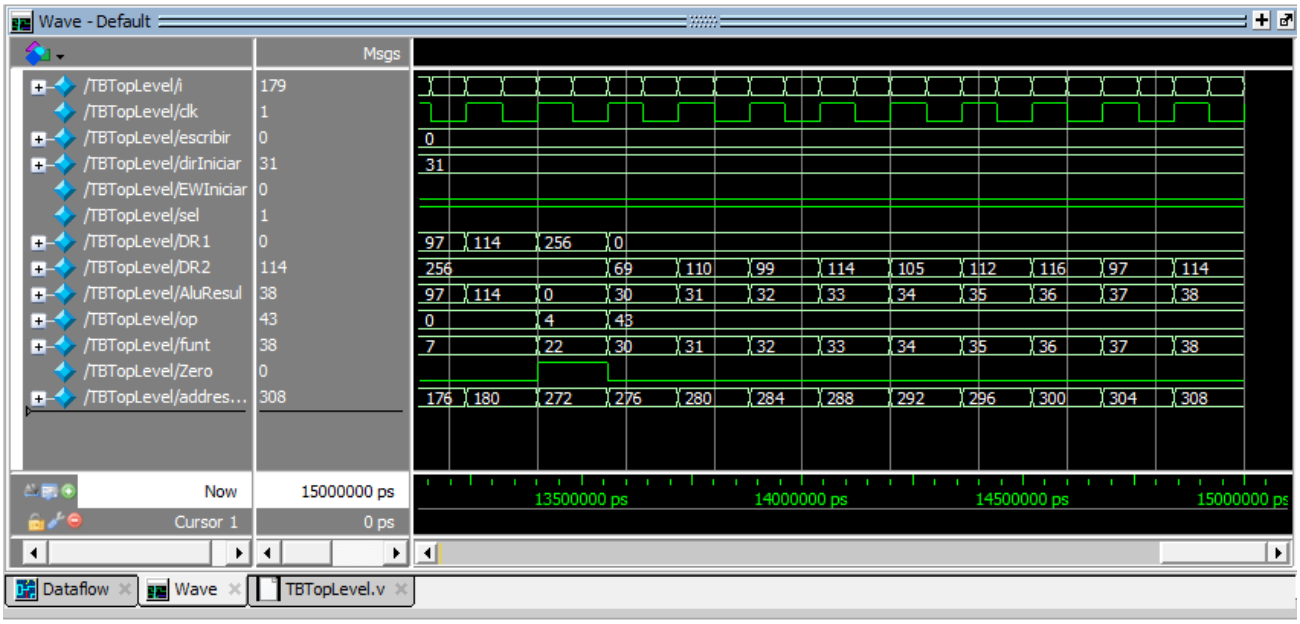

Imagen 17. Instrucciones de subrutina modulo y instrucción beq para ir a las instrucciones de guardado de mensaje desencriptado.

En la imagen 17 también se puede observar el salto de instrucción mediante la instrucción beq para ir a la siguiente instrucción siguiente de la instrucción J anterior. La cual continua guardando el mensaje en memoria en las celdas 30 a 38 como se puede ver en la imagen 18.

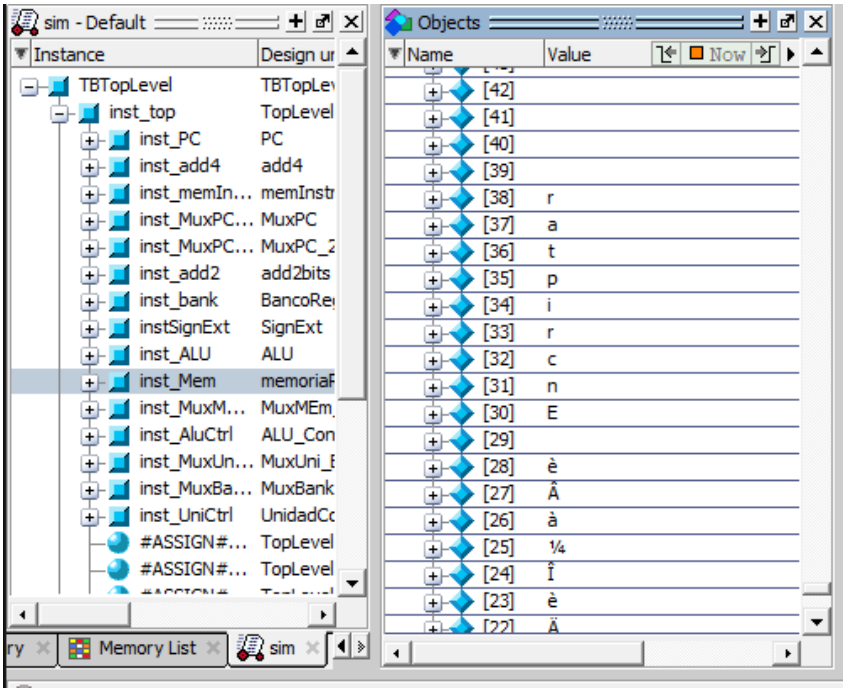

Imagen 18. Mensaje desencriptado en memoria secundaria en las celdas 30 a 38.

Código en lenguaje maquina

El código a usar para ejecutar dentro del testbench es el siguiente código 3.

```
//Guardar en banco de registros la palabra "Encriptar"
001000 01111 00000 0000000001000101
001000 01111 00001 0000000001101110
001000 01111 00010 0000000001100011
001000 01111 00011 0000000001110010
001000 01111 00100 0000000001101001
001000 01111 00101 0000000001110000
001000 01111 00110 0000000001110100
001000 01111 00111 0000000001100001
001000 01111 01000 0000000001110010
//Guardar en memoria el mensaje la palabra
101011 01111 00000 0000000000000000
101011 01111 00001 0000000000000001
101011 01111 00010 0000000000000010
101011 01111 00011 0000000000000011
101011 01111 00100 0000000000000100
101011 01111 00101 0000000000000101
101011 01111 00110 0000000000000110
101011 01111 00111 0000000000000111
101011 01111 01000 0000000000001000
//Guardar en banco de registros la llave "Llave"
001000 01111 01010 0000000001001100
001000 01111 01011 0000000001101100
001000 01111 01100 0000000001100001
001000 01111 01101 0000000001110110
001000 01111 01110 0000000001100101//instruccion N.23
//instruccion para corregir error de inmediata
000000 10000 10001 10010 00000 000000
//Suma de llave y palabra
000000 00000 01010 10100 00000 000000
000000 00001 01011 10101 00000 000000
000000 00010 01100 10110 00000 000000
000000 00011 01101 10111 00000 000000
000000 00100 01110 11000 00000 000000
000000 00101 01010 11001 00000 000000
000000 00110 01011 11010 00000 000000
000000 00111 01100 11011 00000 000000
000000 01000 01101 11100 00000 000000
//Iniciar Subrutina
001000 01111 11110 0000000100000000 //instruccion 34
//Operacion de modulo
```

001000 01111 11101 0000000100000000
//instruccion para corregir error de inmediata
000000 10000 10001 10010 00000 000000

000000 10100 11110 00000 00000 000111
000000 10101 11110 00001 00000 000111
000000 10110 11110 00010 00000 000111
000000 10111 11110 00011 00000 000111
000000 11000 11110 00100 00000 000111
000000 11001 11110 00101 00000 000111
000000 11010 11110 00110 00000 000111
000000 11011 11110 00111 00000 000111
000000 11100 11110 01000 00000 000111
//Comprobar si esta modulando la parabra a encriptar o desencriptar
000100 11110 11111 0000000000010110 //instruccion 46
//Guardar palabra encriptada
101011 01111 00000 0000000000010100
101011 01111 00001 0000000000010101
101011 01111 00010 0000000000010110
101011 01111 00011 0000000000010111
101011 01111 00100 0000000000011000
101011 01111 00101 0000000000011001
101011 01111 00110 0000000000011010
101011 01111 00111 0000000000011011
101011 01111 01000 0000000000011100
//Restar llave de la palabra encriptada
//instrucciones para corregir error de inmediata
001000 01111 01110 0000000001100101
000000 10000 10001 10010 00000 000000

000000 00000 01010 10100 00000 000001
000000 00001 01011 10101 00000 000001
000000 00010 01100 10110 00000 000001
000000 00011 01101 10111 00000 000001
000000 00100 01110 11000 00000 000001
000000 00101 01010 11001 00000 000001
000000 00110 01011 11010 00000 000001
000000 00111 01100 11011 00000 000001
000000 01000 01101 11100 00000 000001
//Ir a subrutina de modulo de palabra
001000 01111 11111 0000000100000000
000010 00000000000000000000100010
//Guardar palabra desencriptada

```
101011 01111 00000 0000000000011110
101011 01111 00001 0000000000011111
101011 01111 00010 0000000000100000
101011 01111 00011 0000000000100001
101011 01111 00100 0000000000100010
101011 01111 00101 0000000000100011
101011 01111 00110 0000000000100100
101011 01111 00111 0000000000100101
101011 01111 01000 0000000000100110//instruccion 77,88
```

Código 3. Código de instrucciones en lenguaje máquina para ejecutar el encriptado y desencriptado Vigenére.

Conclusión

Como se observó en el desarrollo se llegó al objetivo esperado logrando hacer que el mensaje en formato decimal 69, 110, 99, 114, 105, 112, 116, 97, 114 es convertido a 145, 218, 196, 232, 206, 188, 224, 194, 232 y demostrando la desencriptación de esta, en este caso es un ejemplo simple donde solo se usó palabras cortas pero se puede usar para archivos en diferentes formatos para mejor uso de la llave y modulación en formato ASCII.

Como se observó se utilizó los tres tipos de instrucciones donde al menos se ejecutó una vez cada instrucción. En el caso de uso de beq se pudo observar que se necesita contar la instrucción siguiente para sumarle un numero para saltar a diferencia de J donde solo se necesita el valor en la instrucción para brincar a esa dirección.

Referencias

https://www.britannica.com/topic/Vigenere-cipher

http://www.criptored.upm.es/crypt4you/temas/criptografiaclasica/leccion9.html

http://arquitecturadecomputadorasheiner.blogspot.com/2017/10/ciclo-fetch-decode-execute.html

https://es.scribd.com/doc/30420235/Ciclo-Fetch#fullscreen&from_embed

Singh, Simon (1999). "Chapter 2: Le Chiffre Indéchiffrable". The Code Book. Anchor Book, Random House. ISBN 0-385-49532-3.

R. Morelli, R. Morelli, Historical Cryptography: The Vigenere Cipher, Trinity College Hartford, Connecticut

https://es.wikipedia.org/wiki/Cifrado_de_Vigen%C3%A8re

http://www.dma.fi.upm.es/recursos/aplicaciones/matematica_discreta/web/aritmetica_modular/polialfabeto.html