

“Seminario de Solución de Problemas de Arquitectura de Computadoras”

Universidad de Guadalajara
Centro Universitario de Ciencias Exactas e Ingenierías
División de electrónica y computación
Departamento de ciencias computacionales
Licenciatura en Ingeniería Informática (INNI) 611387
Ingeniería Informática
Sección D11
A. 8 - ISA - BR-ALU-RAM

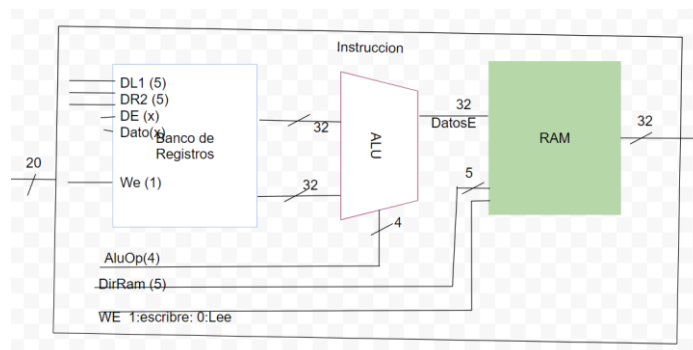
Acuña Concha Claudio Joaquín
Código de alumno: 217809873
Profesor: Jorge Ernesto López Arce Delgado
Viernes 25 de octubre de 2024

Introducción

La investigación de qué procesador usa la Nintendo Switch y la asignación por bloqueo y no bloqueo ya se realizó en la actividad #6.

Objetivo

El objetivo principal de esta actividad es que poder crear la ISA, la cual contendrá a la ALU, el banco de registros y la RAM.



Esto es para que podamos terminar con lo visto en la última clase, donde ya habíamos realizado la ALU y el banco de registros.

La idea también es saber usar el módulo de instrucción, ya que hay que hacerlo de acuerdo con la siguiente tabla:

INSTRUCCION					
OP1	Op2	WE_BR	ALU_OP	Dir_RAM	WE_RAM
5 bit [19:15]	5 bit [14:10]	1 bit [9]	3 bit [8:6]	5 bit [5:1]	1bit [1]

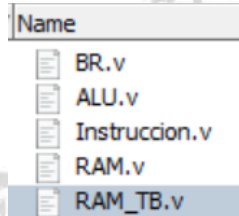
Las operaciones que se realizarán son AND, OR, Suma, Resta y MAYORQ.

Para las operaciones en la RAM usaremos la siguiente tabla:

dir	Datos en RAM	
0		resta
1		resta
2		resta
3		suma
4		suma
5		suma
6		otros
7		

Desarrollo

Los archivos necesarios para la actividad son los siguientes:



Empezaré mostrando el código del BR.v:

```
1 `timescale 1ns/1ns
2
3 //Definición del módulo
4 module Banco(
5     input [4:0]DL1,
6     input [4:0]DL2,
7     input [4:0]DE,
8     input [31:0]Dato,
9     input WE,
10    output reg [31:0]op1,
11    output reg [31:0]op2
12 );
13
14 //Definir (mem) registro bidimensional
15 reg [31:0]BR [0:31];
16
17 always @*
18 begin
19     if(WE)
20     begin
21         BR[DE]=Dato;
22     end
23
24     op1 = BR[DL1];
25     op2 = BR[DL2];
26
27 end
28
29 endmodule
```

Como se puede ver, lo primero que se define en el banco de registros son las entradas DL1, DL2 y DE con 5 bits cada uno, luego WE con un solo bit y las salidas op1 y op2 tipo reg con 32 bits. Luego se hace el arreglo bidimensional de registros de 32 bits haciendo que la primera dirección sea 0. Luego sigue un bloque always donde siempre que ocurre algún cambio en alguna dirección si WE es 1, se va a guardar el dato que llegue al banco de registros en la dirección que menciona el usuario usando DE. Además, aunque se esté escribiendo o no, se va a leer, esto con las líneas 24-25.

Lo que sigue es el código de ALU.v:

```
1 `timescale 1ns/1ns
2
3 module ALU(
4
5     input [31:0] Ope1,
6     input [31:0] Ope2,
7     input [2:0] AluOp,
8     output reg [31:0] Resultado
9 );
10
11 always @*
12 begin
13     case (AluOp)
14         3'b000: //AND
15         begin
16             Resultado = Ope1 & Ope2;
17         end
18         3'b001: //OR
19         begin
20             Resultado = Ope1 | Ope2;
21         end
22         3'b010: //SUMA
23         begin
24             Resultado = Ope1 + Ope2;
25         end
26         3'b100: //RESTA
27         begin
28             Resultado = Ope1 - Ope2;
29         end
30         3'b111: //MAYORQ
31         begin
32             Resultado = Ope1 > Ope2 ? 1:0;
33         end
34     endcase
35 end
36
37
38 endmodule
```

Primero defino las entradas de la ALU, como Ope1 y Ope2 de 32 bits, AluOp de 3 bits y la salida tipo reg de 32 bits Resultado. Hay un bloque always que detecta cuando hay un cambio en cualquier registro, haciendo que se realicen las operaciones comportamentales (AND, OR, SUMA, RESTA, MAYORQ)

El archivo que sigue es RAM.v:

```
1 `timescale 1ns/1ns
2
3 module RAM(
4     input [4:0] DirRam,
5     input [31:0] DatosE,
6     input WE,
7     output reg [31:0] DatosS
8 );
9
10
11 reg [31:0] mem [0:31];
12
13 always @* begin
14     DatosS = mem[DirRam];
15
16     if (WE) begin
17         if (DirRam >= 0 && DirRam <= 2) begin
18             mem[DirRam] = DatosE; // Datos de resta
19         end else if (DirRam >= 3 && DirRam <= 5) begin
20             mem[DirRam] = DatosE; // Datos de suma
21         end else if (DirRam >= 6 && DirRam <= 7) begin
22             mem[DirRam] = DatosE; // Otros datos
23         end
24     end
25 end
26
27 endmodule
```

Se definen las entradas DirRam de 5 bits, DatosE de 32 bits y WE de un solo bit, junto con la salida DatosS tipo reg de 32 bits.

Se crea un arreglo bidimensional de 32 bits. Hay un bloque always que detecta cualquier cambio en los registros, separando los datos de las operaciones obtenidos en la dirección que les corresponde respecto a la siguiente imagen:

dir	Datos en RAM
0	resta
1	resta
2	resta
3	suma
4	suma
5	suma
6	otros
7	

El siguiente archivo es Instrucción.v:

```

1 timescale 1ns/1ns
2
3 module ISA(
4     input [19:0] Instruccion,
5     output [31:0] Salida
6 );
7
8 wire [31:0] d1BR_op1Alu;
9 wire [31:0] d2BR_op2Alu;
10 wire [31:0] DatosE_RAM;
11 wire [31:0] DatosS_RAM;
12
13 Banco instBanco (
14     .DL1(Instruccion[19:15]),
15     .DL2(Instruccion[14:10]),
16     .DE(0),
17     .Dato(0),
18     .WE(Instruccion[9]),
19     .op1(d1BR_op1Alu),
20     .op2(d2BR_op2Alu)
21 );
22
23 ALU instAlu(
24     .Ope1(d1BR_op1Alu),
25     .Ope2(d2BR_op2Alu),
26     .AluOp(Instruccion[8:6]),
27     .Resultado(Salida)
28 );
29
30 RAM instRam (
31     .DirRam(Instruccion[5:1]),
32     .DatosE(DatosE_RAM),
33     .WE(Instruccion[0]),
34     .DatosS(Salida)
35 );
36
37
38 endmodule

```

En el archivo instrucción se tiene una entrada de 20 bits que serán usados para dividir y manejar las operaciones de los otros módulos, además de que tiene una salida de 32 bits.

Se instancian los módulos Banco, ALU y RAM. Lo importante a detallar es que los 20 bits de entrada del módulo Instrucción están repartidos entre las diferentes entradas de los módulos que se instanciaron:

Banco instBanco:

```
.DL1(instruccion[19:15]),  
.DL2(instruccion[14:10]),
```

ALU instAlu:

```
.AluOp(instruccion[8:6]),
```

RAM instRam:

```
.DirRam(instruccion[5:1]),  
.WE(instruccion[0]),
```

Lo que sigue es mostrar el Test Bench, el cual es necesario para mostrar el correcto funcionamiento de todos los módulos:

```
1  `timescale 1ns/1ns  
2  
3  module TB_RAM;  
4  
5      reg [4:0] DirRam;  
6      reg [31:0] DatosE;  
7      reg WE;  
8      wire [31:0] DatosS;  
9  
10     reg [4:0] DL1;  
11     reg [4:0] DL2;  
12     reg [4:0] DE;  
13     reg [31:0] Dato;  
14     reg WE_BR;  
15     wire [31:0] op1;  
16     wire [31:0] op2;  
17  
18     reg [31:0] Ope1;  
19     reg [31:0] Ope2;  
20     reg [2:0] AluOp;  
21     wire [31:0] Resultado;  
22  
23     // Instancia RAM  
24     RAM ram_inst (  
25         .DirRam(DirRam),  
26         .DatosE(DatosE),  
27         .WE(WE),  
28         .DatosS(DatosS)  
29     );  
30  
31     // Instancia del Banco de Registros  
32     Banco br_inst (  
33         .DL1(DL1),  
34         .DL2(DL2),  
35         .DE(DE),  
36         .Dato(Dato),  
37         .WE(WE_BR),  
38         .op1(op1),  
39         .op2(op2)  
40     );  
41  
42     // Instancia de la ALU  
43     ALU alu_inst (  
44         .Ope1(Ope1),  
45         .Ope2(Ope2),  
46         .AluOp(AluOp),  
47         .Resultado(Resultado)  
48     );
```

En esta parte del Test Bench, se declaran los cables junto con las instancias del Banco de registros, la ALU y la RAM. Hago esto porque no tuve claro en cómo hacer el TB, así que básicamente tomé todos los módulos y les puse valores a sus entradas para corroborar el funcionamiento, como muestro a continuación:

```

50 initial begin
51     // Caso 1: Resta
52     // Almacenar valores de resta en RAM
53     DirRam = 5'b00000; // Dirección 0 en RAM
54     DatosE = 32'd30; // Primer operando (30)
55     WE = 1; // Habilitar escritura
56     #10;
57     WE = 0; // Deshabilitar escritura
58     #10;
59
60     DirRam = 5'b00001; // Dirección 1 en RAM
61     DatosE = 32'd10; // Segundo operando (10)
62     WE = 1; // Habilitar escritura
63     #10;
64     WE = 0; // Deshabilitar escritura
65     #10;
66
67     // Almacenar los valores en el Banco de Registros
68     DE = 5'b00000; // Dirección de escritura en BR
69     Dato = 32'd30; // Dato (30)
70     WE_BR = 1; // Habilitar escritura en BR
71     #10;
72     WE_BR = 0; // Deshabilitar escritura
73     #10;
74
75     DE = 5'b00001; // Dirección de escritura en BR
76     Dato = 32'd10; // Dato (10)
77     WE_BR = 1; // Habilitar escritura en BR
78     #10;
79     WE_BR = 0; // Deshabilitar escritura
80     #10;
81
82     // Leer los datos del Banco de Registros para realizar la resta
83     DL1 = 5'b00000; // Leer el primer operando
84     DL2 = 5'b00001; // Leer el segundo operando
85     #10;
86
87     Ope1 = op1; // Cargar el primer operando
88     Ope2 = op2; // Cargar el segundo operando
89     AluOp = 3'b110; // Operación de resta
90     #10;
91
92     $display("Resta: %d - %d = %d", Ope1, Ope2, Resultado);

```

La imagen muestra el primer caso, que es una resta, en la que los datos van pasando a través de todos los módulos hasta llegar a la lectura del banco de registros para que se puedan realizar las operaciones. Lo mismo sucede con las otras operaciones, la suma y AND.

```

50 // Caso 2: Suma
51 // Almacenar valores de suma en RAM
52 DirRam = 5'b00011; // Dirección 3 en RAM
53 DatosE = 32'd20; // Primer operando (20)
54 WE = 1; // Habilitar escritura
55 #10;
56 WE = 0; // Deshabilitar escritura
57 #10;
58
59 DirRam = 5'b00100; // Dirección 4 en RAM
60 DatosE = 32'd15; // Segundo operando (15)
61 WE = 1; // Habilitar escritura
62 #10;
63 WE = 0; // Deshabilitar escritura
64 #10;
65
66 // Almacenar los valores en el Banco de Registros
67 DE = 5'b00011; // Dirección de escritura en BR
68 Dato = 32'd20; // Dato (20)
69 WE_BR = 1; // Habilitar escritura en BR
70 #10;
71 WE_BR = 0; // Deshabilitar escritura
72 #10;
73
74 DE = 5'b00100; // Dirección de escritura en BR
75 Dato = 32'd15; // Dato (15)
76 WE_BR = 1; // Habilitar escritura en BR
77 #10;
78 WE_BR = 0; // Deshabilitar escritura
79 #10;
80
81 // Leer los datos del Banco de Registros para realizar la suma
82 DL1 = 5'b00011; // Leer el primer operando
83 DL2 = 5'b00100; // Leer el segundo operando
84 #10;
85
86 Ope1 = op1; // Cargar el primer operando
87 Ope2 = op2; // Cargar el segundo operando
88 AluOp = 3'b000; // Operación de suma
89 #10;
90
91 $display("Suma: %d + %d = %d", Ope1, Ope2, Resultado);

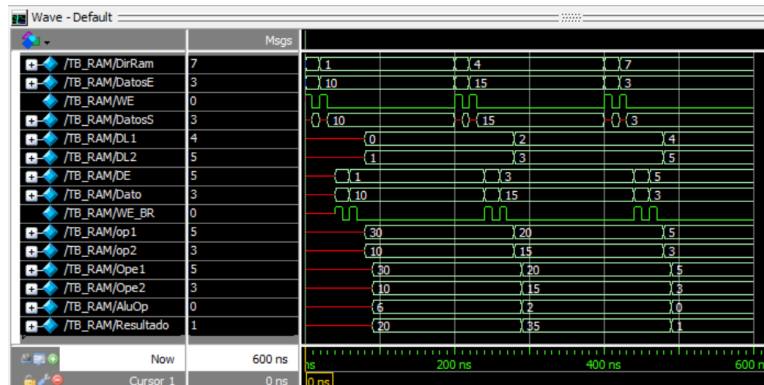
```

```

50 // Caso 3: Operación AND
51 // Almacenar valores para la operación AND en RAM
52 DirRam = 5'b00101; // Dirección 5 en RAM
53 DatosE = 32'd1; // Primer operando (5)
54 WE = 1; // Habilitar escritura
55 #10;
56 WE = 0; // Deshabilitar escritura
57 #10;
58
59 DirRam = 5'b00110; // Dirección 6 en RAM
60 DatosE = 32'd1; // Segundo operando (1)
61 WE = 1; // Habilitar escritura
62 #10;
63 WE = 0; // Deshabilitar escritura
64 #10;
65
66 // Almacenar los valores en el Banco de Registros
67 DE = 5'b00101; // Dirección de escritura en BR
68 Dato = 32'd1; // Dato (1)
69 WE_BR = 1; // Habilitar escritura en BR
70 #10;
71 WE_BR = 0; // Deshabilitar escritura
72 #10;
73
74 DE = 5'b00110; // Dirección de escritura en BR
75 Dato = 32'd1; // Dato (1)
76 WE_BR = 1; // Habilitar escritura en BR
77 #10;
78 WE_BR = 0; // Deshabilitar escritura
79 #10;
80
81 // Leer los datos del Banco de Registros para realizar la operación AND
82 DL1 = 5'b00101; // Leer el primer operando
83 DL2 = 5'b00110; // Leer el segundo operando
84 #10;
85
86 Ope1 = op1; // Cargar el primer operando
87 Ope2 = op2; // Cargar el segundo operando
88 AluOp = 3'b100; // Operación AND
89 #10;
90
91 $display("AND: %d & %d = %d", Ope1, Ope2, Resultado);

```

Una vez en la simulación, se puede ver que los resultados que se obtienen son los correctos, se puede comprobar aún más en la consola, ya que hice que se escribieran los resultados de las operaciones.



```
# Resta:      30 -      10 =      20
# Suma:       20 +      15 =      35
# AND:        5 AND      3 =      1
# ** Note: $stop : C:/Users/LENOVO/Desktop/Seminario Arquitectura Compu/Actividades/#8/RAM_TB.v(186)
# Time: 600 ns Iteration: 0 Instance: /TB_RAM
# Break in Module TB_RAM at C:/Users/LENOVO/Desktop/Seminario Arquitectura Compu/Actividades/#8/RAM_TB.v line 186
```

Una vez visto que todo está correcto, lo que sigue es hacer el decodificador de instrucciones, para ello tomé un archivo .txt y escribí las operaciones de la misma forma que nos había dicho el profesor:

```
RESTA $0,$4,$6
RESTA $1,$7,$1
RESTA $2,$17,$24

LEER $0
LEER $1
LEER $2

SUMA $3,$1,$10
SUMA $4,$5,$19
SUMA $5,$22,$20

LEER $3
LEER $4
LEER $5

MAYORQ $6,$6,$16
MAYORQ $7,$11,$17

LEER $6
LEER $7

AND $6,$21,$13
AND $7,$2,$4

LEER $6
LEER $7

OR $6,$9,$26
OR $7,$8,$5

LEER $6
LEER $7
```

Esto es un set de instrucciones en tipo ensamblador, la idea es que el decodificador cambie cada línea de instrucciones a un formato de 20 bits.

Las direcciones están marcadas con el símbolo '\$' y separados por comas. La primera dirección en cada línea indica en qué dirección de la RAM se va a guardar , y los datos que siguen son las direcciones del banco de registros que van a ser tomados sus valores para la operación correspondiente.

Si tomamos como ejemplo la primera línea:

RESTA \$0,\$4,\$6

El resultado debería ser el siguiente:

00100_00110_0_110_00000_1

Los primeros 5 bits representan la primera dirección del banco de registros, los siguientes 5 bits representan la segunda dirección del banco de registros, El siguiente bit representa WE para que solo sea de lectura en el banco de registros, los siguientes 3 bits representan la selección de la operación que se quiere hacer, en este caso es la resta, y la dirección de la resta es 110, los siguientes 5 bits representan la dirección de la RAM en la que se va a almacenar el resultado, el último bit representa el WE de la RAM, en este caso es 1 porque se quiere que se escriban los datos.

A continuación, muestro la primera parte del código del decodificador:

```
1 import tkinter as tk
2 from tkinter import filedialog, ttk, messagebox
3 import re
4
5 last_opl_per_register = {}
6
7 def decode_instruction(instruction):
8     global last_opl_per_register
9
10    op_code_map = {
11        'RESTA': '110',
12        'SUMA': '010',
13        'MAYORQ': '111',
14        'AND': '000',
15        'OR': '001',
16    }
17
18    # Verificar si es una instrucción LEER
19    if instruction.startswith('LEER'):
20        match = re.match(r'LEER\s+(\d+)', instruction)
21        if match:
22            reg_num = int(match.group(1))
23            if reg_num in last_opl_per_register:
24                opl = last_opl_per_register[reg_num]
25            else:
26                opl = '00000'
27            return f"XXXXX_XXXXX_X_XXX_{opl}_0"
28
29    match = re.match(r'(\w+)\s+(\d+)\s+(\d+)(?:\s+(\d+))?', instruction)
30    if match:
31        operation = match.group(1)
32        opl = f"{int(match.group(2)):05b}"
33        op2 = f"{int(match.group(3)):05b}"
34        we_br = '0'
35        alu_op = op_code_map[operation]
36
37        if len(match.groups()) == 4 and match.group(4):
38            dir_ram = f"{int(match.group(4)):05b}"
39            last_opl_per_register[int(match.group(2))] = opl
40        else:
41            dir_ram = '00000'
42
43        return f"{op2}_{dir_ram}_{we_br}_{alu_op}_{opl}_{1}"
44    return None
```

El codificador empieza leyendo el archivo de instrucciones línea por línea. Mientras va leyendo, identifica el tipo de instrucción (si es una operación: cuál es o si es una lectura). Dependiendo de esto, aplica reglas específicas para construirlo de forma binario con el formato que ya expliqué. Primero se consulta un diccionario interno que traduce las

operaciones a códigos binarios, asignando valores a ALU_OP según corresponda; luego convierte los registros o valores de operandos en binarios de 5 bits, como \$4 en 00100. Al detectar instrucciones de tipo 'LEER' se muestra la última dirección usada en la RAM. Luego, el se traduce toda esta información a un formato final como 00100_00110_0_110_00000_1.

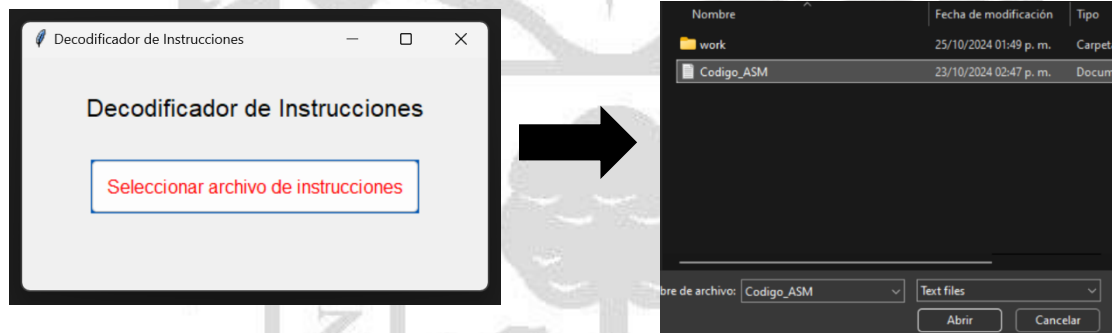
Lo que sigue es la parte del código que hace que toda la información se guarde en un archivo .txt, junto con el código para que cuando se ejecute el programa, se abra una ventana que permita seleccionar un archivo .txt de la computadora para que pueda ser procesado.

```
46 def process_file():
47     # Abrir un cuadro de diálogo para seleccionar el archivo
48     file_path = filedialog.askopenfilename(title="Seleccionar archivo de instrucciones", filetypes=[("Text files", "*.txt")])
49
50     if not file_path:
51         return
52
53     with open(file_path, 'r') as file:
54         instructions = file.readlines()
55
56     results = []
57
58     for instruction in instructions:
59         decoded = decode_instruction(instruction.strip())
60         if decoded:
61             results.append(decoded)
62
63     # Guardar los resultados en un nuevo archivo
64     output_path = file_path.replace('.txt', '_decoded.txt')
65     with open(output_path, 'w') as file:
66         for result in results:
67             file.write(result + '\n')
68
69     messagebox.showinfo("Éxito", f"Archivo procesado y guardado como: {output_path}")
70
71 # Configurar la interfaz gráfica
72 root = tk.Tk()
73 root.title("Decodificador de Instrucciones")
74 root.geometry("400x200")
75 root.configure(bg="#f0f0f0")
76
77
78 style = ttk.Style()
79 style.configure("TButton", padding=10, relief="flat", background="#0056b3", foreground="red", font=("Arial", 12))
80 style.map("TButton", background=[("active", "#004080")])
81
82 frame = ttk.Frame(root, padding=20)
83 frame.pack(expand=True, fill="both")
84
85
86 title_label = ttk.Label(frame, text="Decodificador de Instrucciones", font=("Arial", 16), background="#f0f0f0")
87 title_label.pack(pady=10)
88
89
90 process_button = ttk.Button(frame, text="Seleccionar archivo de instrucciones", command=process_file)
91 process_button.pack(pady=20)
92
93
94 root.mainloop()
```

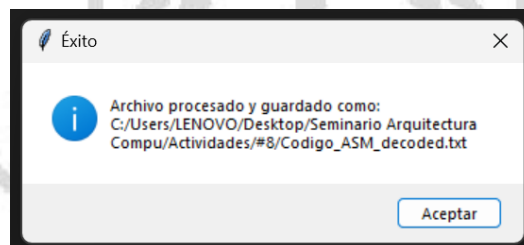
Por último, con el archivo el set de instrucciones en ensamblador del .txt que había mostrado antes, voy a hacer una prueba de la ejecución del programa:

```
RESTA $0,$4,$6  
RESTA $1,$7,$1  
RESTA $2,$17,$24  
  
LEER $0  
LEER $1  
LEER $2  
  
SUMA $3,$1,$10  
SUMA $4,$5,$19  
SUMA $5,$22,$20  
  
LEER $3  
LEER $4  
LEER $5  
  
MAVORQ $6,$6,$16  
MAVORQ $7,$11,$17  
  
LEER $6  
LEER $7  
  
AND $6,$21,$13  
AND $7,$2,$4  
  
LEER $6  
LEER $7  
  
OR $6,$9,$26  
OR $7,$8,$5  
  
LEER $6  
LEER $7
```


Al ejecutarlo, se abre la siguiente ventana, que me permite seleccionar el .txt anterior:



Tras haber seleccionado el archivo, aparecerá una ventana indicando que se guardó correctamente junto con la dirección de la ubicación del archivo:



Una vez obtenido el archivo decodificado, nos podemos dar cuenta de que efectivamente realizó bien los cambios a binario, respetando los formatos tanto para las operaciones como para las lecturas:



```
RESTA $0,$4,$6
RESTA $1,$7,$1
RESTA $2,$17,$24

LEER $0
LEER $1
LEER $2

SUMA $3,$1,$10
SUMA $4,$5,$19
SUMA $5,$22,$20

LEER $3
LEER $4
LEER $5

MAYORQ $6,$6,$16
MAYORQ $7,$11,$17

LEER $6
LEER $7

AND $6,$21,$13
AND $7,$2,$4

LEER $6
LEER $7

OR $6,$9,$26
OR $7,$8,$5

LEER $6
LEER $7
```

```
00100_00110_0_110_00000_1
00111_00001_0_110_00001_1
10001_11000_0_110_00010_1
XXXXX_XXXXX_X_XXX_00000_0
XXXXX_XXXXX_X_XXX_00001_0
XXXXX_XXXXX_X_XXX_00010_0
00001_01010_0_010_00011_1
00101_10011_0_010_00100_1
10110_10100_0_010_00101_1
XXXXX_XXXXX_X_XXX_00011_0
XXXXX_XXXXX_X_XXX_00100_0
XXXXX_XXXXX_X_XXX_00101_0
00110_10000_0_111_00110_1
01011_10001_0_111_00111_1
XXXXX_XXXXX_X_XXX_00110_0
XXXXX_XXXXX_X_XXX_00111_0
10101_01101_0_000_00110_1
00010_00100_0_000_00111_1
XXXXX_XXXXX_X_XXX_00110_0
XXXXX_XXXXX_X_XXX_00111_0
01001_11010_0_001_00110_1
01000_00101_0_001_00111_1
XXXXX_XXXXX_X_XXX_00110_0
XXXXX_XXXXX_X_XXX_00111_0
```

Conclusión

Esta actividad ha sido por lejos la más difícil y complicada de la unidad de aprendizaje, ya que más de saber qué hace cada cosa, hay que estar siempre atento a los módulos que vamos creando para no perder el hilo de lo que se quiere crear. Por otro lado el decodificador estuvo algo difícil, ya que hace tiempo que no hacía algo así, sin embargo se pudo lograr y definitivamente ya voy entendiendo más el cómo usar todo lo que hemos estado viendo a través de las clases.

Referencias

-<http://digital.unex.es/wiki/doku.php?id=pub:vlog>