

Centro universitario: CUCEI

Materia: Seminario de solución de problemas de arquitectura de computadoras

Alumno: Ramos Preciado Alan Josafat

Código: 218130165

Carrera: Ingeniería en informática (INNI)

título de investigación: TAREA 9 DATAPATH

Fecha 07/10/2024

Contents

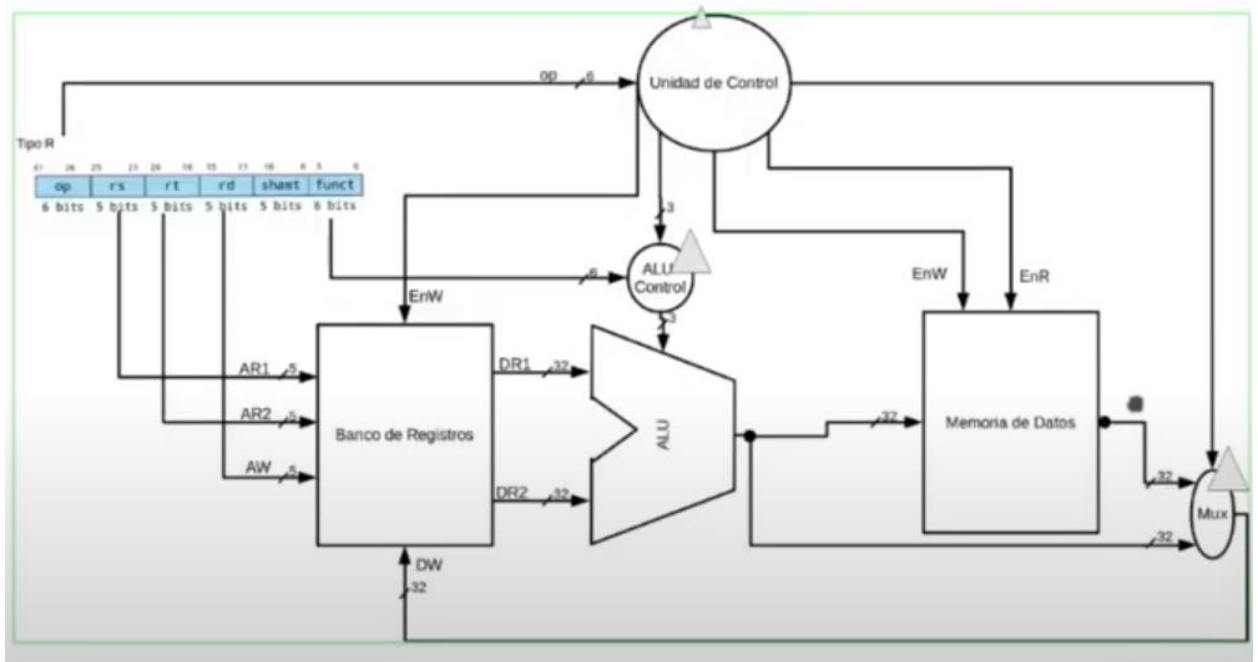
INTRODUCCIÓN.....	3
OBJETIVO	3
Los módulos creados son los siguientes.	3
Comprobación de compilación	10
Tb de datapath.....	10
Creación y formato de nuestro convertidor de intrucciones	11
Simulamos y corremos.....	14
CONCLUSIÓN.....	15
REFERENCIAS.....	15

INTRODUCCIÓN

Existen diferentes arquitecturas en los procesadores pero en esta ocasión vamos a usar la arquitectura de tipo de MIPS en la que nos centraremos en las instrucciones de tipo R

OBJETIVO

Nuestro objetivo es desarrollar lo siguiente en donde lo adaptaremos para recibir instrucciones de tipo R y que funcione para las operaciones aritméticas.



DESARROLLO

Empezaremos con una captura de pantalla de cada módulo.

Los módulos creados son los siguientes.

-Banco de registros.

-ALU

-Unidad de control

-Memoria de datos

-ALU control

-Mux

-Datapath

```
RecordBank.v
1  `timescale 1ns/1ns
2
3  module RecordBank(
4
5      //inputs
6
7      input EnW,
8      input [4:0] AR1,
9      input [4:0] AR2,
10     input [4:0] AW,
11     input [31:0] DW,
12
13     //outputs
14     output reg [31:0] DR1,
15     output reg [31:0] DR2
16 );
17
18     reg [31:0] RecordBank [0:31];
19
20     initial
21     begin
22         $readmemb("rbddata.txt", RecordBank); // Cargar datos desde el archivo
23     end
24
25     always @*
26     begin
27         if (EnW)
28         begin
29             RecordBank[AW] = DW;
30         end
31
32         // Leer valores de los registros
33         DR1 = RecordBank[AR1];
34         DR2 = RecordBank[AR2];
35     end
36
37 endmodule
```

```

1  `timescale 1ns/1ns
2
3  module ALU(
4
5      //inputs
6      input [31:0] DR1,
7      input [31:0] DR2,
8      input [2:0] ALUControl,
9
10     //outputs
11     output reg[31:0] ALUOutput
12 );
13 // add, sub, slt, and, or, xor, nor
14 always @*
15 begin
16     case (ALUControl)
17         //ADD
18         3'b001:
19             begin
20                 ALUOutput = DR1 + DR2;
21             end
22         //SUB
23         3'b010:
24             begin
25                 ALUOutput = DR1 - DR2;
26             end
27         //SLT
28         3'b011:
29             begin
30                 ALUOutput = DR1 > DR2 ? 1 : 0;
31             end
32         //AND
33         3'b100:
34             begin
35                 ALUOutput = DR1 & DR2;
36             end
37         //OR
38         3'b101:
39             begin
40                 ALUOutput = DR1 | DR2;
41             end
42         //XOR
43         3'b110:
44             begin
45                 ALUOutput = DR1 ^ DR2;
46             end
47         //NOR
48         3'b111:
49             begin
50                 ALUOutput = ~(DR1 | DR2);
51             end
52     endcase
53 end
54 endmodule
55
56 module ALUControl(
57
58     //inputs
59     input [2:0] UnitControlRequest,
60     input [5:0] funct,
61
62     //outputs
63     output reg[2:0] OpALU
64 );
65
66 always @*
67 begin
68     case (UnitControlRequest)
69         //type r
70         3'b000:
71             begin
72                 case(funct)
73                     6'b100000:
74                         begin
75                             //ADD
76                             OpALU = 3'b001;
77                         end
78                     6'b100010:
79                         begin
80                             //SUB
81                             OpALU = 3'b010;
82                         end
83                     6'b101010:
84                         begin
85                             //SLT
86                             OpALU = 3'b011;
87                         end
88                     6'b100100:
89                         begin
90                             //AND
91                             OpALU = 3'b100;
92                         end
93                     6'b100101:
94                         begin
95                             //OR
96                             OpALU = 3'b101;
97                         end
98                     6'b100110:
99                         begin
100                            //XOR
101                            OpALU = 3'b110;
102                        end
103                     6'b100111:
104                         begin
105                            //NOR
106                            OpALU = 3'b111;
107                        end
108                 endcase
109             end
110         endcase
111     end
112 endmodule

```

ControlUnit.v

```
1  `timescale 1ns/1ns
2
3  module ControlUnit(
4
5      //inputs
6      input [5:0] OpCode,
7      output reg BR_En,
8      output reg [2:0] AluC,
9      output reg EnW,
10     output reg EnR,
11     output reg Mux1
12
13 );
14
15 always @*
16 begin
17     case (OpCode)
18     6'b000000: //type R
19     begin
20
21         BR_En = 1'b1;
22         AluC = 3'b000;
23         EnW = 1'b0;
24         EnR = 1'b0;
25         Mux1 = 1'b1;
26
27     end
28     endcase
29 end
30
31 endmodule
```

C:\Users\alan_\Desktop\DataPath\DataMemory.v

```
1  `timescale 1ns/1ns
2
3  module DataMemory(
4
5      //inputs
6      input [31:0] DataIn,
7      input EnW,
8      input EnR,
9
10     //outputs
11     output reg [31:0] DW
12
13 );
14
15 reg [31:0] DataMemory [0:3];
16
17 always @*
18 begin
19     if(EnW)
20     begin
21         DataMemory[1] = DataIn;
22     end
23     else if(EnR)
24     begin
25         DW = DataMemory[1];
26     end
27 end
28
29 endmodule
```

Mux.v

```
1  `timescale 1ns/1ns
2
3  module Mux(
4
5      //inputs
6      input sel,
7      input [31:0] A,
8      input [31:0] B,
9
10     //outputs
11     output reg[31:0] C
12
13 );
14
15 always @*
16 begin
17     if(sel)
18     begin
19         C=B;
20     end
21     else
22     begin
23         C=A;
24     end
25 end
26
27 endmodule
```


Teniendo creados los modulos lo siguiente es crear el data path donde lo que utilizaremos es una instancia de cada modulo creado y hacer conexiones con cables.









```

Datapath.v
1  `timescale 1ns/1ns
2
3  module Datapath(
4
5      //inputs
6      input [31:0] instruction
7  );
8
9  //wires to Control Unit
10 wire [5:0] instruction_to_Control_Unit;
11
12 //wires to RecordBank
13 wire [4:0] instruction_to_rb_ar1;
14 wire [4:0] instruction_to_rb_ar2;
15 wire [4:0] instruction_to_rb_aw;
16 wire ControlUnit_to_rb_enw;
17 wire [31:0] mux_to_rb_dw;
18
19 //wires to ALU
20 wire [31:0] rb_to_alu_dr1;
21 wire [31:0] rb_to_alu_dr2;
22 wire [2:0] alucontrol_to_alu_opalu;
23
24 //wires to ALU Control
25 wire [2:0] controlunit_to_alucontrol_type;
26 wire [5:0] instruction_to_alucontrol_funcnt;
27
28 //wires to DataMemory
29 wire [31:0] alu_to_datamemory_data;
30 wire controlunit_to_datamemory_enw;
31 wire controlunit_to_datamemory_enr;
32
33 //wires to Mux
34 wire controlunit_to_mux;
35 wire [31:0] datamemory_to_mux_datain;
36 wire [31:0] alu_to_mux_dataint;
37
38 //instances
39
40 ControlUnit controlUnitInstance(
41     .OpCode(instruction[31:26]),
42     .BR_En(ControlUnit_to_rb_enw),
43     .AluC(controlunit_to_alucontrol_type),
44     .EnW(controlunit_to_datamemory_enw),
45     .EnR(controlunit_to_datamemory_enr),
46     .Mux1(controlunit_to_mux)
47 );
48
49
50 ALUControl ALUControlInstance(
51     .UnitControlRequest(controlunit_to_alucontrol_type),
52     .funcnt(instruction[5:0]),
53     .OpALU(alucontrol_to_alu_opalu)
54 );
55
56
57 RecordBank RecordBankInstance(
58     .EnW(ControlUnit_to_rb_enw),
59     .AR1(instruction[25:21]),
60     .AR2(instruction[20:16]),
61     .AW(instruction[15:11]),
62     .DW(mux_to_rb_dw),
63     .DR1(rb_to_alu_dr1),
64     .DR2(rb_to_alu_dr2)
65 );
66
67
68 ALU ALUInstance(
69     .DR1(rb_to_alu_dr1),
70     .DR2(rb_to_alu_dr2),
71     .ALUControl(alucontrol_to_alu_opalu),
72     .ALUOutput(alu_to_datamemory_data)
73 );
74
75
76
77 DataMemory DataMemoryInstance(
78     .DataIn(alu_to_datamemory_data),
79     .EnW(controlunit_to_datamemory_enw),
80     .EnR(controlunit_to_datamemory_enr),
81     .DW(datamemory_to_mux_datain)
82 );
83
84
85 Mux muxInstance(
86     .sel(controlunit_to_mux),
87     .A(datamemory_to_mux_datain),
88     .B(alu_to_datamemory_data),
89     .C(mux_to_rb_dw)
90 );
91
92
93
94 endmodule

```

Al tener eso listo nos pasaremos a ver si funciona

Comprobación de compilación

	RecordBank.v	✓	Verilog 0	11/07/2024 02:46:27 ...
	ControlUnit.v	✓	Verilog 1	11/07/2024 02:13:10 ...
	ALU.v	✓	Verilog 2	11/07/2024 02:13:13 ...
	ALUControl.v	✓	Verilog 3	11/07/2024 02:13:10 ...
	DataMemory.v	✓	Verilog 4	11/07/2024 02:44:30 ...
	Mux.v	✓	Verilog 5	11/07/2024 02:13:12 ...
	Datapath.v	✓	Verilog 6	11/07/2024 02:25:46 ...
	tb_dp.v	✓	Verilog 7	11/07/2024 02:40:55 ...

Tb de datapath

Al comprobar su funcionamiento creamos un tb para probar el funcionamiento

```
1  `timescale 1ns/1ns
2
3  module tb_dp;
4
5      reg [31:0] instrucciontb;
6
7      Datapath dataPathIntance(
8          .instruction(instrucciontb)
9      );
10
11     integer i;
12     reg [31:0] setIntruccion [0:20];
13
14     initial
15     begin
16         $readmemb("32intructions.txt", setIntruccion);
17
18         for(i = 0; i < 20; i = i + 1)
19         begin
20             instrucciontb = setIntruccion[i];
21             #10;
22         end
23
24         $finish;
25     end
26
27 endmodule
28
29
```

El tb recibe la información (instrucciones de un archivo llamada 32intructions.txt)

Y el initial de banco de registros tiene una memoria que se llena con los números del uno al 31 tomándolos de otro txt

Para facilitar la creación de instrucciones se creo una gui con Python que nos ayuda a convertir las instrucciones de decimal a binario.

Creación y formato de nuestro convertidor de intrucciones

32	ADD
34	SUB
42	SLT
36	AND
37	OR
38	XOR
39	NOR

31	26	25	21	20	16	15	11	10	6	5	0
SPECIAL 000000			rs		rt		rd		0 00000		ADD 100000
6			5		5		5		5		6

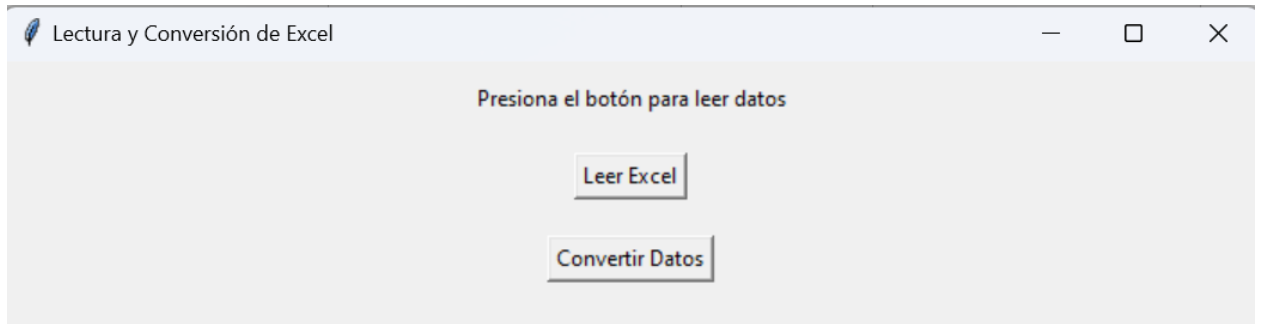
Teniendo en cuenta el libro ADD es igual a 100000 en binario pero en decimal seria el 32 por lo que al poner el 32 y luego pasarlo a binario estaríamos representando la instrucción de add $32 = 100000$ y se hace lo mismo con las otras operaciones.

En ese caso ese seria el numero que representa cada operación en decimal.

Para los siguientes caso utilizaremos nuestra GUI.py

```
PS C:\Users\alan_\Desktop\DataPath> py .\GUI.py
```

Que nos abre lo siguiente



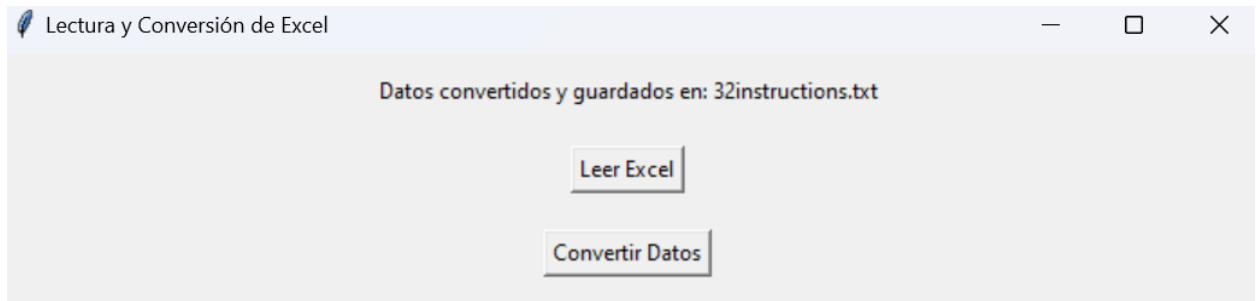
Al presionar leer el Excel nos muestra los datos leídos en consola

```
PS C:\Users\alan\Desktop\DataPath> py .\GUI.py
  op 6  rs 5  rt 5  rd 5  shawt 5  funct 6
0      0      1      2      3      0      32
1      0      1      2      4      0      34
2      0      1      2      5      0      42
3      0      1      2      6      0      36
4      0      1      2      7      0      37
5      0      1      2      8      0      38
6      0      1      2      9      0      39
```

Que efectivamente son los mimos que tenemos en el Excel

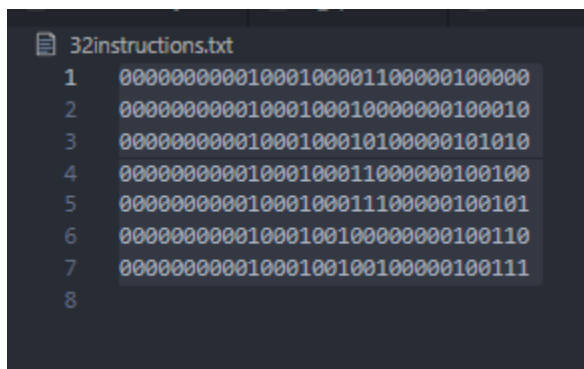
A	B	C	D	E	F	
op 6	rs 5	rt 5	rd 5	shawt 5	funct 6	
0	1	2	3	0	32	
0	1	2	4	0	34	
0	1	2	5	0	42	
0	1	2	6	0	36	
0	1	2	7	0	37	
0	1	2	8	0	38	
0	1	2	9	0	39	

Después presionamos el botol de crear instrucciones



Nos confirma que se guardaron en el .txt

Ahora chequearemos el txt a ver si contiene las instrucciones



Esas son las instrucciones que utilizaremos en nuestro tb

Ahora nos pasamos a model sim a simular el tb del datapath

Donde tendremos 3 memorias

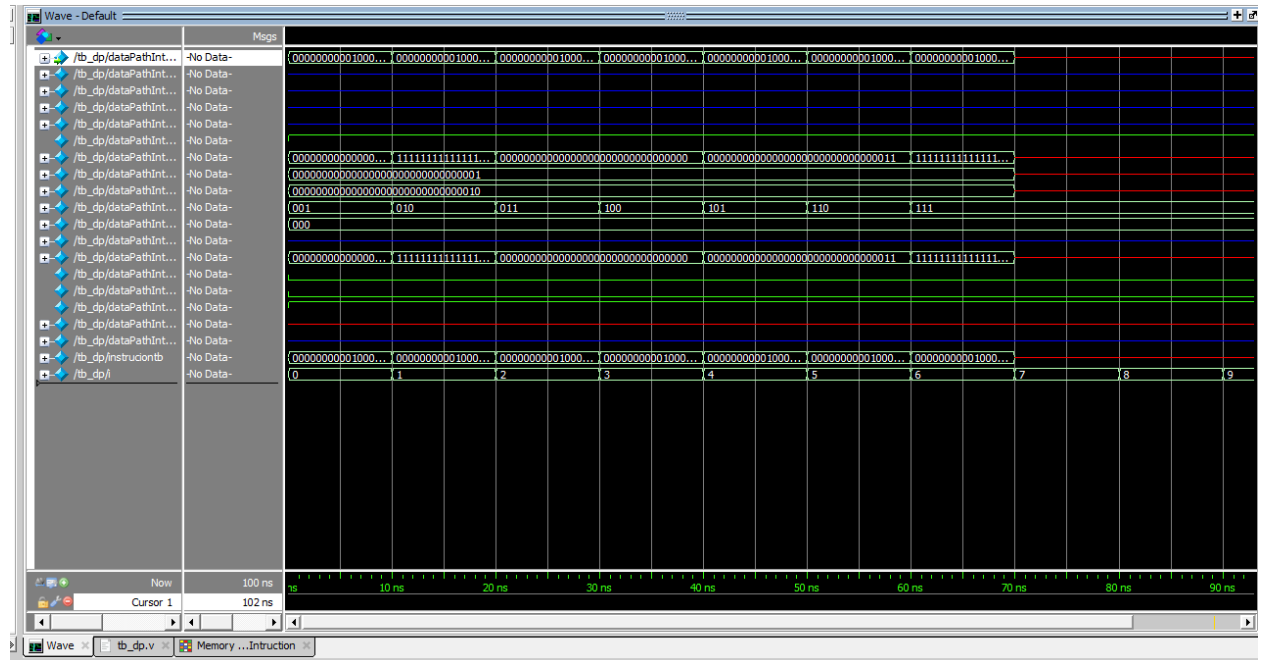
Instance	Range	Depth	Width
◆ /tb_dp/setInstructio...	[0:20]	21	32
◆ /tb_dp/dataPathInt...	[0:31]	32	32
◆ /tb_dp/dataPathInt...	[0:3]	4	32

Una del banco de registros

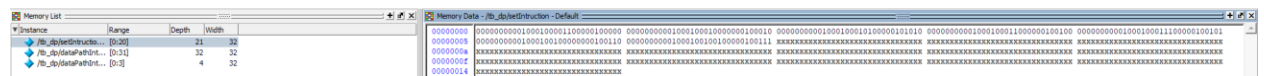
Una de las instrucciones

Y otra de la memoria de datos

Simulamos y corremos

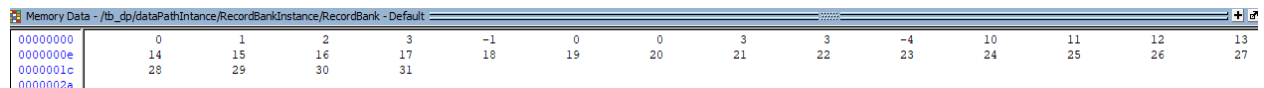


Al parecer todo bien pero tenemos que comprobar que es lo que tenemos al final después de correr todas las instrucciones



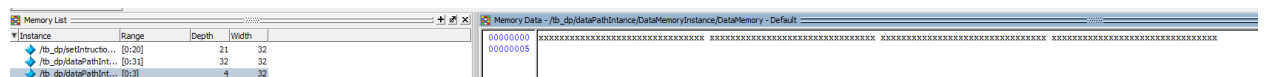
Esas son las instrucciones ingresadas

Lo siguiente es nuestro banco de registros



En donde se comprueba las operaciones que se hicieron

Y la memoria de datos queda vacía porque nunca habilitamos el escribir en ella. Por lo que es correcto que este vacía



CONCLUSIÓN

Estas tareas se ven complejas pero cuando se divide todo por bloques y se tiene todo organizado solo es y escribiendo poco a poco y la dificultad no aumenta mucho.

REFERENCIAS

MIPS® Architecture for Programmers Volume II-A: The MIPS32® Instruction Set Manual
Document Number: MD00086 Revision 6.06 December 15, 2016