Equipo 3

PROYECTO FINAL

Seminario de solución de problemas de Arquitectura de Computadoras.

Sección: D12

Clave: I7024

Maestro: Lopez Arce Delgado Jorge Ernesto

Participantes del equipo 3:

- Juan Pablo Calzada Avalos
- Daniel Sanchez Zepeda
- Branco Musich Flores

Link del GitHub:

https://github.com/Darkstar1403/Sem.-Arquitectura-2020B-D12-ArribaPython/

Introducción

Procesador MIPS de 32 bits

El procesador es la parte primordial de una computadora, ya que gracias a él una computadora es posible ejecutar distintos programas, la arquitectura MIPS32 o MIPS de 32 bits permite mayor rendimiento y eficiencia que se encuentran en miles de millones de productos electrónicos desde pequeños microcontroladores hasta equipos de red de alta gama.

El procesador MIPS de 32 bits se basa en un conjunto de instrucciones codificadas regularmente de longitud fija y utiliza un modelo de datos de carga / almacenamiento, esta optimizada de tal manera que permite la ejecución de lenguajes de alto nivel, las operaciones aritméticas y lógicas utilizan un formato de tres operando, lo que permite a los compiladores optimizar la formulación de expresiones complejas.

La disponibilidad de 32 registros de propósito general permite a los compiladores optimizar aún más la generación de código para el rendimiento al mantener los datos a los que se accede con frecuencia en los registros.

Tiene una flexibilidad de sus caches de altos rendimientos y esquemas de administración de memoria son los puntos fuertes estas ventajas con opciones de control de caché bien definidas, el tamaño de las cachés de instrucciones y datos puede variar entre 256 bytes y 4 MB.

MIPS32 se basa en un set de instrucciones RISC de longitud fija (32 bits) con una codificación estándar, así como un modelo de carga y almacenamiento de datos.

Set de instrucciones

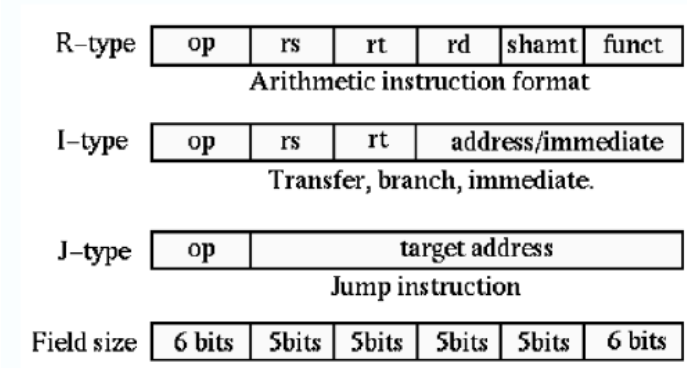
El set de instrucciones contiene una gran variedad de operaciones, entre las más importantes se encuentran:

- 21 instrucciones aritméticas.
- 8 instrucciones lógicas.
- 8 instrucciones para la manipulación de bits.
- 12 instrucciones de comparación.
- 25 instrucciones de salto y branch.
- 15 instrucciones de carga (load).
- 10 instrucciones de almacenamiento.
- 8 instrucciones de desplazamiento.
- 4 instrucciones misceláneas.

De manera general, las instrucciones se pueden separar en tres categorías distintas:

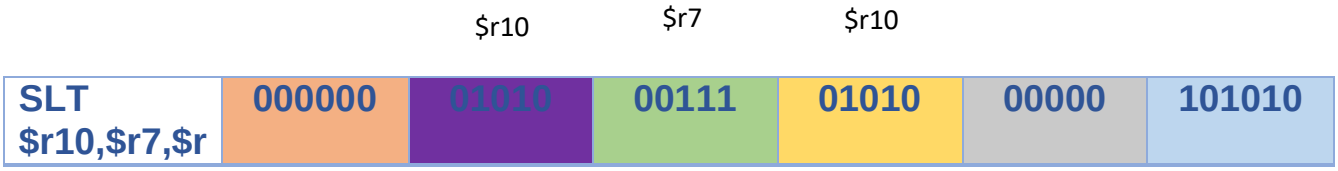
- Tipo R: Ejecutan operaciones aritméticas y lógicas.
- Tipo I: Transfieren datos entre registros, ejecutan operaciones aritmético/lógicas inmediatas, se encargan de hacer branches.
- Tipo J: Realizan saltos una instrucción a otra.

Cada tipo de instrucción distribuye los 32 bits de manera distinta:



Cuando una instrucción necesita campos más largos que los mostrados anteriormente aparece un problema.

Por ejemplo, la instrucción de carga debe especificar dos registros y una constante. Si la dirección usara uno de los campos de 5 bits del formato anterior, la constante dentro de la instrucción de carga estaría limitada a solo 32. Esta constante se usa para seleccionar elementos de tablas grandes o estructuras de datos y frecuentemente necesita ser mucho mayor que 32, por lo que este campo de 5 bits es demasiado pequeño para ser útil en ocasiones de trabajo real.



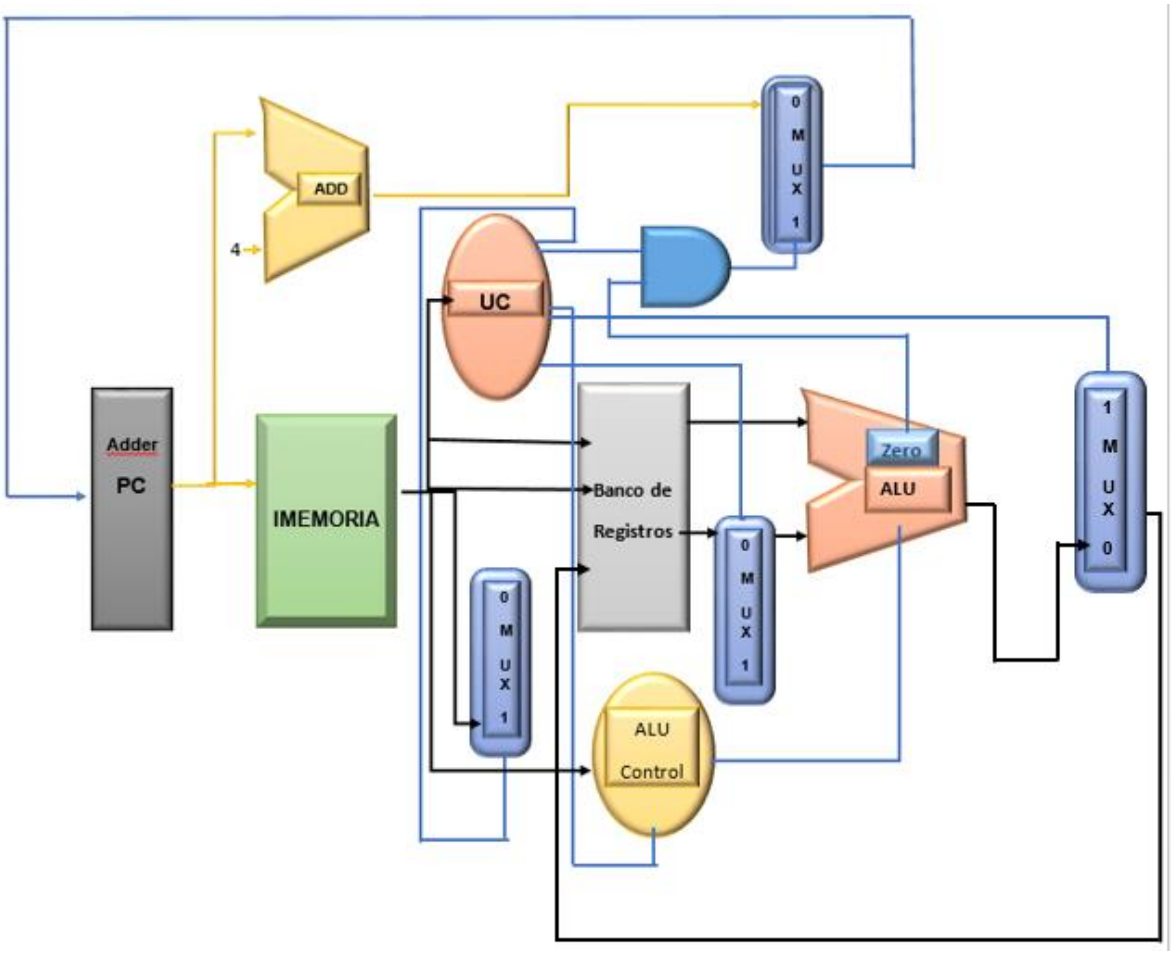
En la imagen anterior podemos presenciar de una instrucción MIPS de tipo R de 32 bits “SLT”, donde si el primer registro es mayor que el segundo entonces se formara un tercer registro del primer registró.

Los signos de dólares representan que la operación se va a realizar en los registros con esos identificadores.

La representación binaria está dividida en 6 colores que representan los 6 campos de una instrucción MIPS de tipo R. El procesador identifica que tipo de instrucción es gracias a los dígitos binarios en el primer y último campo (naranja y azul bajito respectivamente). Comparado con la imagen donde se muestra la distribución de bits para cada tipo de instrucción, el primer y último campo corresponde a los campos op y funct en las instrucciones de tipo R. En este caso, el procesador reconoce que esta instrucción es un tercer registro por el cero en el primer campo (naranja) y el 42 en el último campo (azul bajito).

Los operando se representan en los campos morada y verde, en la imagen de referencia tienen los nombres rs y rt. El resultado se guarda en el campo de color amarillo, rd en la imagen de referencia. El campo gris representa la cantidad de shift que no se usa en esta operación.

Fase 1


ADD
4
0
M
U
X
1
UC
Adder
PC
IMEMORIA
Banco de
Registros
Zero
ALU
1
M
U
X
0
0
M
U
X
1
0
M
U
X
1
ALU
Control

Instrucción	Tipo	Sintaxis
Add	R	Add $rd, $rs, $rt
Sub	R	Sub $rd, $rs, $rt
Mul	R	Mul $rd, $rs, $rt
Div	R	Div $rs, $rt
Or	R	Or $rd, $rs, $rt
And	R	And $rd, $rs, $rt
Slt	R	Slt $rd, $rs, $rt
Nop	R	NOP
addi	I	addi $rs,$rt,imm
subi	I	subi $rs,$rt,imm
ori	I	ori $rt,$rs,imm
andi	I	andi $rt,$rs,imm
lw	I	lw $rt,offset(base)
sw	I	sw $rt,offset(base)
slti	I	slti $rt,$rs,inmediato
beq	I	beq $rs,$rt,offset
bne	**I**	**bne $rs, $rt, imm**
bgtz	I	bgtz $rs,imm
adiu	I	adiu $rt,$rs.imm
J	J	j destino

Desarrollo

Fase 1

Ciclo fetch

Memoria de instrucciones

Su función es almacenar las instrucciones del programa que se va a ejecutar y enviar estas instrucciones al resto de los elementos del procesador.

Una peculiaridad de este módulo es que tiene un ancho de palabra de 8 bits; como las instrucciones tienen un ancho de 32 bits, esto implica que las instrucciones tienen que estar segmentadas en 4 partes de 8 bits cada una. Para que al final se entreguen las instrucciones completas, se tienen que concatenar 4 direcciones de memoria en la salida de este módulo. A su vez, esto implica que las instrucciones están numeradas en múltiplos de 4. Por ejemplo la instrucción 1 está en la posición 0, por lo que se concatenan las posiciones 0, 1, 2 y 3. Si se quiere acceder a la instrucción 2, hay que indicarle a la memoria de instrucciones que lea los contenidos de la posición 4, de esta manera se concatenan las posiciones 4, 5, 6 y 7. Si se quiere acceder a la instrucción 3, hay que indicar la posición 8 y así sucesivamente.

Como se dijo antes, tiene un ancho de 8 bits y una profundidad de 127 espacios de memoria. Cuenta con una entrada de 8 bits que indica cuál posición de memoria se va a leer y una salida de 32 bits que entrega las instrucciones completas.

```
`timescale 1ns/1ns

module Imemoria(
input [31:0] pcDir,
output reg [31:0] instOut
);
//      Registro encargado de almacenar las instrucciones
reg [7:0] mem_inst [511:0];
//      Registros que me dice en que numero de indice voy
//      Ciclo encargado de mandar como una sola instruccion 4 celdas de memoria
always @(*)
begin
    instOut = {mem_inst[pcDir], mem_inst[pcDir + 1], mem_inst[pcDir + 2], mem_inst[pcDir + 3]};
end
initial
begin
//000000_00000_00001_10100_00000_100000 SUMA $20 = $0+$1
mem_inst[0] = 8'b00000000;
mem_inst[1] = 8'b00000001;
mem_inst[2] = 8'b10100000;
mem_inst[3] = 8'b00100000;
//000000_00101_00110_10101_00000_100010 RESTA $21 = $5-$6
mem_inst[4] = 8'b00000000;
mem_inst[5] = 8'b10100110;
mem_inst[6] = 8'b10101000;
mem_inst[7] = 8'b00100010;
//000000_01010_01011_10110_00000_100100 AND $22 = $10 & $11
mem_inst[8] = 8'b00000001;
mem_inst[9] = 8'b01001011;
mem_inst[10] = 8'b10110000;
mem_inst[11] = 8'b00100100;
//000000_01110_01111_10111_00000_100101 OR $23 = $14 | $15
mem_inst[12] = 8'b00000001;
mem_inst[13] = 8'b11001111;
mem_inst[14] = 8'b10111000;
mem_inst[15] = 8'b00100101;
//000000_10010_10011_11000_00000_101010 SLT $24 = $18 < $17?1:0
mem_inst[16] = 8'b00000010;
mem_inst[17] = 8'b01010011;
mem_inst[18] = 8'b11000000;
mem_inst[19] = 8'b00101010;
//000000_10011_01000_11001_00000_011000 MULTIPLICACION $25 = $19 * $8
mem_inst[20] = 8'b00000010;
mem_inst[21] = 8'b01101000;
mem_inst[22] = 8'b11001000;
mem_inst[23] = 8'b00011000;
//000000_00011_00010_11010_00000_011010 DIVISION $26 = $3 / $2
mem_inst[24] = 8'b00000000;
mem_inst[25] = 8'b01100010;
mem_inst[26] = 8'b11010000;
mem_inst[27] = 8'b00011010;
//000000_10010_10011_11011_00000_000000 NOP $27 = 0
mem_inst[28] = 8'b00000010;
mem_inst[29] = 8'b01010011;
mem_inst[30] = 8'b11011000;
mem_inst[31] = 8'b00000000;
end
endmodule
```

PC

Se encarga de indicarle a la memoria de instrucciones qué número instrucción es la que se va a ejecutar.

Por las características de la memoria de instrucciones, este módulo sólo recibe valores múltiplos de 4.

Cuenta con una entrada de 8 bits que recibe el valor de la siguiente instrucción a ejecutar, este valor proviene de un sumador. También tiene una salida de 8 bits que envía el valor de la instrucción a ejecutar a la memoria de instrucciones.

```
1   `timescale 1ns/1ns
2   module Pc(
3       input [31:0] dirIn,
4       input clk,
5       output reg [31:0] dirOut
6   );
7
8   reg [31:0] pcl;
9
10  always@(posedge clk)
11  begin
12      if(clk == 1)
13          begin
14              dirOut = pcl;
15              pcl = dirIn;
16          end
17  end
18  initial begin
19      pcl = 0;
20  end
21  endmodule
22
```

.

AdderPc

Este módulo recibe el valor de la instrucción que se está ejecutando actualmente y le añade 4 a ese valor.

Específicamente, le suma 4 al valor de la instrucción actual por las características de la memoria de instrucciones: las instrucciones están segmentadas en 4 partes cada una, por lo que si queremos ir a la siguiente instrucción, necesitamos avanzar 4 posiciones.

Cuenta con una entrada de 8 bits que recibe el valor de la instrucción actual y una salida que envía el valor de la siguiente instrucción.

```
1   `timescale 1ns/1ns
2
3   module AdderPc(
4       input [31:0] pcIn,
5       output reg[31:0] dataOut
6   );
7   integer b4;
8   always@(*)begin
9       b4 = 3'b100;
10      case(b4)
11          3'b100:
12                  begin
13                          dataOut = pcIn + b4;
14                  end
15      endcase
16  end
17
18  initial
19  begin
20          dataOut = 32'd0;
21  end
22  endmodule
```

Banco de registros

El propósito de este módulo es contener los 32 registros característicos de MIPS, con los cuales se realizan una gran cantidad de operaciones en el procesador.

Cada registro tiene un ancho de palabra de 32 bits. Estos registros pueden proporcionar operando para realizar operaciones aritméticas, valores para calcular posiciones de memoria, etc. Es capaz de leer dos registros a la vez y de escribir datos a cualquiera de los 32 registros. Puede leer y escribir datos al mismo tiempo.

Cuenta con 3 entradas de 5 bits cada una, 2 indican cuáles registros se van a leer y 1 indica en cuál registro se van a escribir datos. Tiene una entrada de 32 bits que recibe los datos a escribir, estos datos pueden venir de la memoria de instrucciones o ser el resultado de alguna operación lógica o aritmética. También cuenta con 2 salidas de 32 bits cada una que envían los valores de los registros indicados por las entradas hacía el resto del procesador.

```
`timescale 1ns/1ns
module Banco_registros(
input[4:0] dirL1, dirL2, dirW,
input wr,
input[31:0] datoIn,
output reg [31:0] dataOut1, dataOut2
);
reg [31:0]BRegistros[31:0];

initial
begin
BRegistros[0] = 32'd12;
BRegistros[1] = 32'd45;
BRegistros[2] = 32'd2;
BRegistros[3] = 32'd50;
BRegistros[4] = 32'd47;
BRegistros[5] = 32'd64;
BRegistros[6] = 32'd24;
BRegistros[7] = 32'd7;
BRegistros[8] = 32'd132;
BRegistros[9] = 32'd127;
BRegistros[10] = 32'd38;
BRegistros[11] = 32'd96;
BRegistros[12] = 32'd35;
BRegistros[13] = 32'd94;
BRegistros[14] = 32'd82;
BRegistros[15] = 32'd22;
BRegistros[16] = 32'd130;
BRegistros[17] = 32'd101;
BRegistros[18] = 32'd177;
BRegistros[19] = 32'd115;
end

always @(*)
begin

        if(wr == 1)

        begin
                BRegistros[dirW] = datoIn;
        end
        dataOut1 = BRegistros[dirL1];
        dataOut2 = BRegistros[dirL2];

end
endmodule

```

Mux21

Solo indica que lo que se va a guardar en el banco de registros sale de la Ram o de la Alu

```
1   `timescale 1ns/1ns
2   module MUX21(
3   input [31:0] memOut,aluOut,
4   input sel,
5   output reg [31:0] dataOut
6   );
7
8   always @(*)
9   begin
10          case(sel)
11                  1'b1:
12                  begin
13                          dataOut = memOut;
14                  end
15                  1'b0:
16                  begin
17                          dataOut = aluOut;
18                  end
19          endcase
20  end
21  endmodule
22
```

Mux21PC

El Mux sirve para que dirección de las instrucciones vamos a pedir, para que salte hacia otra instrucción

```
`timescale 1ns/1ns
module MUX21PC(
input [31:0] shiftOut,addPcOut,
input sel,
output reg [31:0] dataOut
);

always @(*)
begin
        case(sel)
                1'b1:
                begin
                        dataOut = shiftOut;
                end
                1'b0:
                begin
                        dataOut = addPcOut;
                end
        endcase
end
endmodule

```

Mux21BR

Este Mux da uso y paso después del banco del registro

```
`timescale 1ns/1ns
module MUX21BR(
input [31:0] signOut,brOut,
input sel,
output reg [31:0] dataOut
);

always @(*)
begin
        case(sel)
                1'b1:
                begin
                        dataOut = signOut;
                end
                1'b0:
                begin
                        dataOut = brOut;
                end
        endcase
end
endmodule

```

Mux21InstMem

El Mux sirve para las instrucciones de tipo r y permite manejar otro tipo de instrucciones donde se debe de registrar en el banco de registros.

```
1   `timescale 1ns/1ns
2   module MUX21INST(
3   input [4:0] rtOut,rdOut,
4   input sel,
5   output reg [4:0] dataOut
6   );
7
8   always @(*)
9   begin
10          case(sel)
11                  1'b1:
12                  begin
13                          dataOut = rdOut;
14                  end
15                  1'b0:
16                  begin
17                          dataOut = rtOut;
18                  end
19          endcase
20  end
21  endmodule
22
```

DataPathR

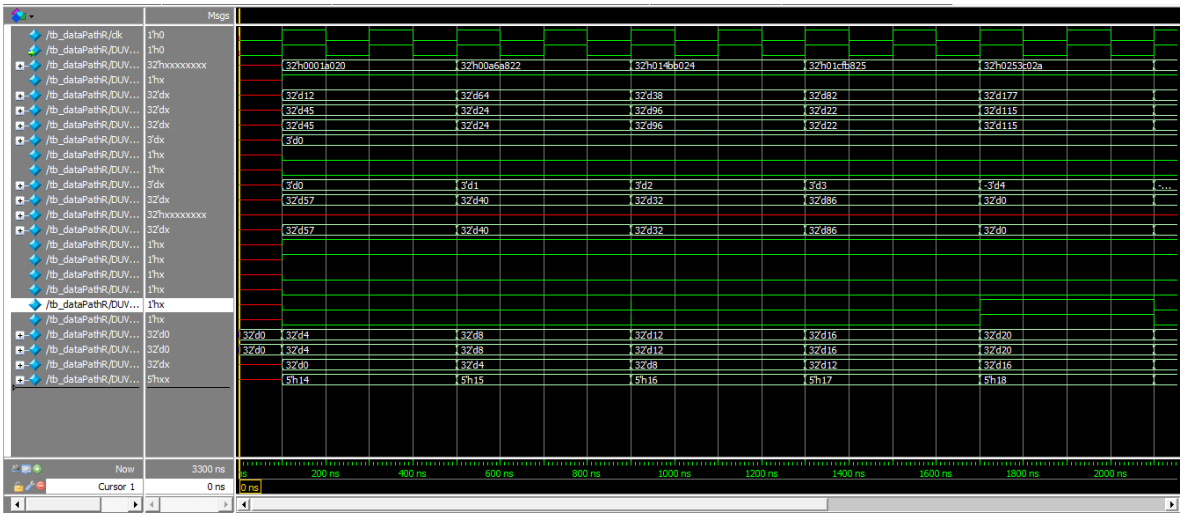
Al final realizamos conexiones con cables desde el DataPathR para que al final a la hora de ejecutar el programa realice la simulación buscada

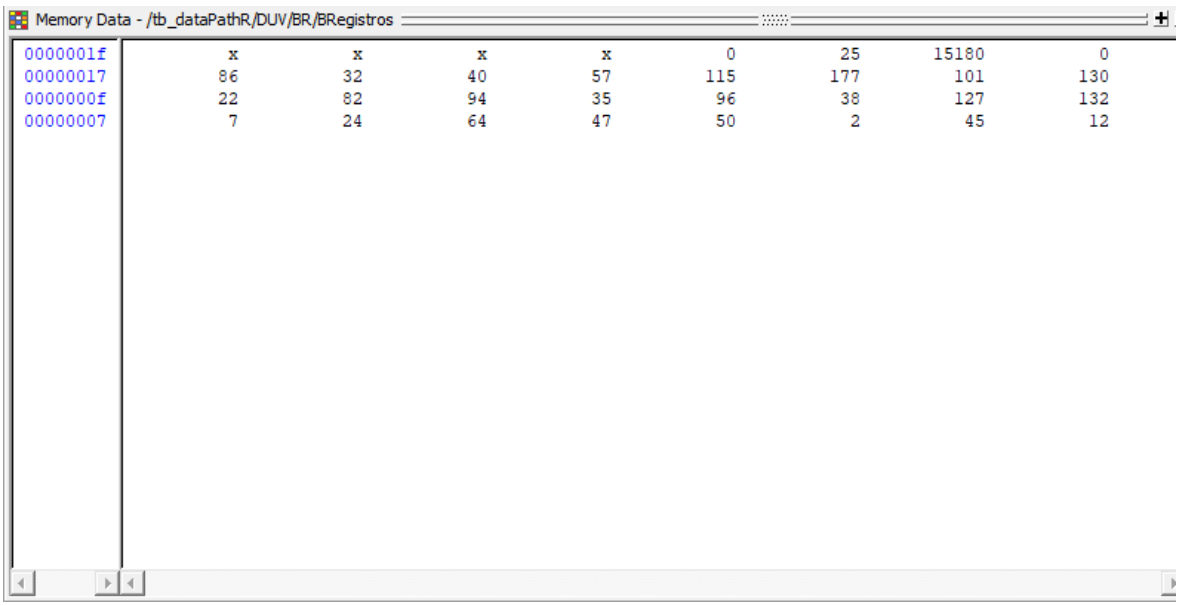
```
1   `timescale 1ns/1ns
2   module DataPathR(
3   input clk
4   );
5
6   wire [31:0] instOut;//Memoria de instrucciones a UC, BR, MUXBR,
7   wire regWrite;//Unidad de Control a Banco de Registro
8   wire [31:0]RData1, RData2;//Banco de Registro a ALU y Mem, el segundo va al mux
9   wire [31:0]MuxBrOut;//MUX21BR a ALU
10  wire [2:0]UALUOp;//Unidad de Control a ALU Control
11  wire memWrite;//Unidad de Control a Mem para activar escritura
12  wire memReg;//Unidad de control a Mux21
13  wire [2:0] ALUCout;//ALU Control a ALU
14  wire [31:0] ResALU;//ALU a Mem y MUX21
15  wire [31:0] ReadData;//Mem a MUX
16  wire [31:0] MuxOut;//MUX a Banco de Registro
17  wire regDist;// Unidad de Control a MUX21InstMem
18  wire branch;//Unidad de Control a MUXPc
19  wire memRead;//Unidad de Control a Mem para activar lectura
20  wire ALUSrc;//Unidad de Control a MUX21BR
21  wire cl;//AND de branch y zeroflag
22  wire zeroF;
23  wire [31:0]Mux21Pc;//Mux21Pc a Pc
24  wire [31:0]adderOut;//adderPc a MUX21Pc
25  wire [31:0]pcOut;//Pc a memoria de instrucciones y adderPc
26  wire [4:0]muxInstOut;//MUXInstMem a banco de resgistros
27
28  Pc pc(
29  .dirIn(Mux21Pc), .clk(clk), .dirOut(pcOut)
30  );
31
32  Imemoria instMem(
33  .pcDir(pcOut), .instOut(instOut)
34  );
35
36  AdderPc add(
37  .pcIn(pcOut), .dataOut(adderOut)
38  );
```

```
39
40  UC UCDR(
41  .opCode(instOut[31:26]), .amemToReg(memReg), .regWrite(regWrite),
42  .amemToWrite(memWrite),.branch(branch), .ALUSrc(ALUSrc),
43  .amemToRead(memRead), .regDist(regDist), .aluOp(UALUOp)
44  );
45
46  MUX21 mux1(
47  .memOut(ReadData), .aluOut(ResALU), .sel(memReg), .dataOut(MuxOut)
48  );
49
50  MUX21INST muxInst(
51  .rtOut(instOut[20:16]), .rdOut(instOut[15:11]), .sel(regDist), .dataOut(muxInstOut)
52  );
53
54  MUX21PC muxPc(
55  .shiftOut(32'd0), .addPcOut(adderOut), .sel(32'd0), .dataOut(Mux21Pc)
56  );
57
58  Banco_registros BR
59  (
60  .dirL1(instOut[25:21]), .dirL2(instOut[20:16]),
61  .dirW(muxInstOut),.wr(regWrite), .datoIn(MuxOut),
62  .dataOut1(RData1), .dataOut2(RData2)
63  );
64
65  MUX21BR muxBr(
66  .signOut(32'd0), .brOut(RData2), .sel(ALUSrc), .dataOut(MuxBrOut)
67  );
68
69  assign c1= branch & zeroF; //assign de la compuerta and que se conecta al muxPc
70
71  ALU_Control ALUC(.func(instOut[5:0]), .sel(UALUOp), .aluF(ALUCout));
72
73  ALU alu (.A(RData1), .B(MuxBrOut), .SEL(ALUCout), .RESULTADO(ResALU), .ZF(zeroF));
74
75  AMem mem(.dir(ResALU), .dataIn(RData2), .wr(memWrite), .rd(memRead), .dataOut(ReadData));
76
77  endmodule
78
```

Tb_DatapathR

```
1   `timescale 1ns/1ns
2   module tb_dataPathR();
3   reg clk = 0;
4
5   DataPathR DUV(.clk(clk));
6
7   always #100 clk = ~(clk);
8
9   initial
10  begin
11  #3300;
12  $stop;
13  end
14
15  endmodule
16
```





Conclusiones