

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 Actividad 11

Datapath para instrucción tipo R y tipo I

Alumno: Oswaldo Luna Grados

Código: 211718256

Sección: D14

Profesor: J. Ernesto López Arce Delgado

Introducción

Las instrucciones como se dijo anteriormente hay diferentes tipos como el tipo I.

Las instrucciones tipo I tienen un campo inmediato de 16 bits que codifica:

Un operador inmediato es decir no hay un valor guardado dentro de un registro en el banco.

Operaciones donde implican guardar y leer de una memoria. En el caso de la instrucción lw. Las instrucciones de carga mueven datos de la memoria a un registro. La dirección para la carga es la suma de un registro especificado en la instrucción y un valor constante que se codifica en la instrucción.

En el caso de sw la instrucción implica que mueve un valor a la memoria donde la dirección se almacenada es la suma de un registro más el valor inmediato en la instrucción.

Un espacio de paginación es un tipo de volumen lógico con espacio de disco asignado que almacena la información, que se ubica en la memoria virtual a la que no se accede actualmente. El espacio de paginación no puede utilizar menos de 16 MB.

Objetivos

Se busca mejorar la actividad anterior donde se ejecutó las instrucciones tipo R donde veíamos las salidas del banco, la salida de la ALU //corregir

Se ejecutara las instrucciones sw, lw, subi, add, ori, andi y addi donde se usara cada una 3 veces.

Como se ve en la tabla 1 en la parte de unidad de control se refiere los valores que recibe de esta a la ALUControl.

Unidad De control	0000	0001	0010	0011	0100	0101	1000
Operación	addi	subi	andi	ori	xori	slti	Tipo R

Tabla 1. Posibles operaciones según la entrada de la unidad de control a la ALUControl.

Instrucción	opUniCtrl	Function	ALU op
add	1000	000000	000
sub	1000	000010	001
mult	1000	001000	010
div	1000	011010	011
And	1000	100100	100
Or	1000	100101	101
xor	1000	000110	110
slt	1000	101010	111

Tabla 2. Operaciones del tipo R.

En la tabla 2 se muestra las opciones de entrada y la operación de la ALU realizara en la ALUControl.

Op	selCtrl	enW_Bank	MuxMemALU	MuxSign_ALU	MuxAddr	memW	memR
001000(addi)	000	0	1	1	0	0	0
001001(subi)	101	0	1	1	0	0	0
001010(slti)	100	0	1	1	0	0	0
001100(andi)	001	0	1	1	0	0	0
001101(ori)	010	0	1	1	0	0	0
001110(xori)	011	0	1	1	0	0	0
100011(lw)	000	0	0	1	0	0	1
101011(sw)	000	0	0	1	0	1	0

Tabla 3. Operaciones dentro de unidad de control.

Dentro la tabla 3 están las instrucciones tipo I que puede realizarse y muestra las salidas de la unidad de control según la instrucción.

El tamaño de paginación a usar en la memoria será de 100.

Desarrollo

Se implementó los mismos módulos de la actividad anterior pero se modificó y se agregó los siguientes módulos:

UnidadControl se agregó los puertos para los seleccionadores de los multiplexores agregados y se creó los casos en que recibe diferentes operaciones para las instrucciones tipo I.

ALU_Control se modificó la entrada de la unidad de control a 4 bytes, se creó casos para las instrucciones tipo I además de dejar para la tipo R.

MuxUni_Bank se creó modulo que conecta de las instrucciones la dirección secundaria o la de destino en la dirección de escritura dentro del banco de registros.

MuxBank_signExt selecciona entre ADR2 del banco y la instrucción del 15 al 0 extendiendo el bit de signo

SignExt recibe de la instrucción los bits del 15 al 0, donde el ultimo bit usa para para rellenar los bits más significativos para completar los 32 bits.

```

1  timescale 1ns /1ps
2  module MuxUni_Bank(
3      input [4:0] ins_Inst, ind_Inst,
4      input selUni,
5      output reg[4:0] outBank
6  );
7      always @(*)
8      begin
9          if(selUni == 1) outBank=ind_Inst; //Elige la direccion de registro
10         else outBank=ins_Inst; //Elige la direccion del segundo registro
11     end
12 endmodule

```

Imagen 1. Módulo multiplexor de direcciones en instrucción a banco.

La imagen 1 muestra el modulo del multiplexor para elegir entre las direcciones dentro de la instrucción las cuales son la secundaria que son del bit 20 al 16 y la de destino del 15 al 11, estas son enviadas a la dirección de escritura dentro del banco de registro.

```

1  timescale 1ps /1ps
2  module MuxBank_SignExt(
3      input [31:0] inBank, inSignExt,
4      input selUni,
5      output reg [31:0] outS_ALU
6  );
7      always @(*)
8      begin
9          if(selUni==1) outS_ALU=inSignExt; //Elegir guardar valor en memoria
10         else outS_ALU=inBank; //Elegir guardar valor en ALU
11     end
12 endmodule

```

Imagen 2. Modulo Multiplexor entre ADR2 del banco y extensión de signo a la ALU.

En la imagen 2 recibe la extensión de signo de la instrucción o la ADR2 del banco de registro para la segunda entrada de la ALU.

```

1  timescale 1ns /1ps
2  module SignExt(
3      input [15:0] in,
4      output reg[31:0] out
5  );
6      always @(*)
7      begin
8          if(in[15]==1)
9              begin
10                 out[31:16]=8'd0;
11                 out[15:0]=in;
12             end
13         else
14             begin
15                 out[31:16]=8'd65535;
16                 out[15:0]=in;
17             end
18     end
19 endmodule

```

Imagen 3. Módulo Extensión de signo

La extensión de signo como se observa en la imagen 3 recibe los bits de la instrucción del 15 al 0 y le agrega bits hacia el más significativo para completar 32 bits ya sea solo poniendo 1 o 0. Esto se determina por el bit más significativo, si es 1 rellena los nuevos de solo 1 y si es 0 entonces rellena con 0 todos los nuevos bits agregados.

```

1  `timescale lps /lps
2  module ALU_Control(
3      input [3:0]inControl,
4      input [5:0] inFunct,
5      output reg[2:0] outAlu
6  );
7  always @(*)
8  begin
9      case(inControl)
10         4'd0:outAlu=0;//addi
11         4'd1:outAlu=1;//subi
12         4'd2:outAlu=4;//andi
13         4'd3:outAlu=5;//ori
14         4'd4:outAlu=6;//xori
15         4'd5:outAlu=7;//slti
16
17         4'd8://si inControl=8 hacer funciones de inFunct
18         begin
19             case(inFunct)
20                 6'd0:outAlu=0;//add
21                 6'd2:outAlu=1;//sub
22                 6'd8:outAlu=2;//mult
23                 6'b011010:outAlu=3;//div
24                 6'd4:outAlu=4;//and
25                 6'd5:outAlu=5;//or
26                 6'd6:outAlu=6;//xor
27                 6'b101010:outAlu=7;//slt
28                 default:outAlu=0;
29             endcase
30         end
31         default:outAlu=0;
32     endcase
33 end
34 endmodule

```

Imagen 4. Módulo ALU control

El módulo de ALU control se aumentó los bits que recibe de la unidad de control (inControl) a 4 bits donde revisa si es uno de los casos posibles para hacer las operaciones inmediatas o de la función.

```

1  `timescale 1ps /1ps
2  module UnidadControl(
3      input [5:0] op,
4      output reg enW_Bank,enW_Mem,
5      output reg enR_Mem,selMuxMem_ALU,
6      output reg selMuxAddr,
7      output reg selMuxSign_Bank,
8      output reg [3:0]selControl
9  );
10 always@(*)
11 begin
12     case(op)
13         0://tipo R
14         begin
15             selControl=4'b1000;//ALUControl reconoce funct
16             enW_Bank=0;
17             selMuxMem_ALU=1;//Elige ALU
18             selMuxSign_Bank=0;//Elige banco
19             selMuxAddr=1;//Elige direccion destino
20             enW_Mem=0;
21             enR_Mem=0;
22         end
23         //Type I
24         6'd8://addi
25         begin
26             selControl=0;//sumar
27             enW_Bank=0;//escribir
28             selMuxMem_ALU=1;//Elige ALU
29             selMuxSign_Bank=1;//Elige extension de signo
30             selMuxAddr=0;//Elige segunda direccion
31             enW_Mem=0;
32             enR_Mem=0;
33         end

```

Imagen 5. Módulo unidad de control parte 1.

La imagen 5 muestra el código del módulo de unidad de control donde se creó casos para reconocer entre instrucciones tipo R o de tipo I según los bits de operación en la instrucción.

```

34      6'd10: //slti
35      begin
36          selControl=4'd4;//slt
37          enW_Bank=0;//escribir
38          selMuxMem_ALU=1;//Elige ALU
39          selMuxSign_Bank=1;//Elige extension de signo
40          selMuxAddr=0;//Elige segunda direccion
41          enW_Mem=0;
42          enR_Mem=0;
43      end
44      6'd12: //andi
45      begin
46          selControl=1;//and
47          enW_Bank=0;//escribir
48          selMuxMem_ALU=1;//Elige ALU
49          selMuxSign_Bank=1;//Elige extension de signo
50          selMuxAddr=0;//Elige segunda direccion
51          enW_Mem=0;
52          enR_Mem=0;
53      end
54      6'd13: //ori
55      begin
56          selControl=4'd2;//or
57          enW_Bank=0;//escribir
58          selMuxMem_ALU=1;//Elige ALU
59          selMuxSign_Bank=1;//Elige extension de signo
60          selMuxAddr=0;//Elige segunda direccion
61          enW_Mem=0;
62          enR_Mem=0;
63      end
64      6'd14: //xori

```

Imagen 6. Módulo unidad de control parte 2.

```

64      6'd14://xori
65      begin
66          selControl=4'd3;//xor
67          enW_Bank=0;//escribir
68          selMuxMem_ALU=1;//Elige ALU
69          selMuxSign_Bank=1;//Elige extension de signo
70          selMuxAddr=0;//Elige segunda direccion
71          enW_Mem=0;
72          enR_Mem=0;
73      end
74      6'b100011: //lw
75      begin
76          selControl=4'd0;//sumar
77          enW_Bank=0;//escribir
78          selMuxMem_ALU=0;//Elige memoria
79          selMuxSign_Bank=1;//Elige extension de signo
80          selMuxAddr=0;//Elige segunda direccion
81          enW_Mem=0;
82          enR_Mem=1;//cargar a banco datos
83      end
84      6'b101011://sw
85      begin
86          selControl=4'd0;//sumar
87          enW_Bank=0;//escribir
88          selMuxMem_ALU=0;
89          selMuxSign_Bank=1;//Elige extension de signo
90          selMuxAddr=0;//Elige segunda direccion
91          enW_Mem=1;//guardar en memoria
92          enR_Mem=0;
93      end
94      default:selControl=4'b1111;
95  endcase
96 end
97 endmodule

```

Imagen 7. Módulo unidad de control parte 3.

Dentro del módulo TopLevel se modificó agregando y cambiando cables, agregando las instancias de los multiplexores y de la extensión de signo.


```

1  `timescale 1ns /1ps
2  module TopLevel(
3      input [31:0]inst,
4      input [31:0]escribir, //recibiria los valores de entrada
5      input [4:0]dirIniciar,dirLeer,
6      input EWIniciar,
7      input [1:0]sel,
8      output [31:0]DR1,DR2,
9      output [31:0]AluResul
10 );
11 //unidad de control a otros modulos
12 wire Ctrl_Bank;
13 wire [3:0] Ctrl_AlucCtrl;
14 wire Ctrl_MemEW;
15 wire Ctrl_MemER;
16 wire Ctrl_MuxMem_ALU;
17 wire Ctrl_MuxAddr;
18 wire Ctrl_MuxSign_Bank;
19 //conexiones de Banco a otros modulos
20 wire[31:0] DR1_A,DR2_B,DR2_Index;
21 wire[31:0] DR2_MuxALU;
22 //ALU_Ctrl a ALU
23 wire [2:0] AlucCtrl_ALU;
24 //ALU a otros dispositivos
25 wire [31:0]ALU_Mux,ALU_Mem;
26 //Memoria a MuxDest
27 wire [31:0]Mem_Mux;
28 //MuxMem_ALU a Bank
29 wire [31:0]MuxMem_ALU_Bank;
30 //MuxAddr a bank
31 wire [4:0]MuxAddr_Bank;
32 //MuxDR2Bank o sign a alu
33 wire [31:0]MuxALU_DR2Bank_Sign;
34 //Extension de signo a Mux ALU
35 wire [31:0]outSign_MuxALU;
36 //registros para elegir entre muestra o funcion controlada por UC
37 reg EWR;

```

Imagen 8. Módulo TopLevel parte 1.

```

37 reg EWR;
38 reg [31:0]RWBank;
39 reg [4:0]AW;
40 reg [4:0]AR;
41 assign DR1=DR1_A;
42 assign DR2=DR2_MuxALU;
43 assign AluResul=ALU_Mux;
44 assign ALU_Mem=ALU_Mux;
45 always @(*)
46 begin
47     if(sel==1)
48     begin
49         RWBank=MuxMem_ALU_Bank;
50         AW=MuxAddr_Bank;
51         EWR=Ctrl_Bank;
52         AR=inst[25:21];
53     end
54     else
55     begin
56         RWBank=escribir;
57         AW=dirIniciar;
58         EWR=EWIniciar;
59         AR=dirLeer;
60     end
61 end
62
63 BancoRegis inst_bank(
64     .AR1(AR),
65     .AR2(inst[20:16]),
66     .AW(AW),
67     .dateW(RWBank),
68     .EWR(EWR),
69     .DR1(DR1_A), .DR2(DR2_MuxALU)
70 );
71 SignExt instSignExt(
72     .in(inst[15:0]),
73     .out(outSign_MuxALU)
74 );

```

Imagen 9. Módulo TopLevel parte 2.

```

75 ALU inst_ALU(
76     .in_A(DR1_A),
77     .in_B(MuxALU_DR2Bank_Sign),
78     .in_sel(AluCtrl_ALU),
79     .o_S(ALU_Mux)
80 );
81 memoriaRAM inst_Mem(
82     .dato(ALU_Mem),
83     .index(DR2_Index),
84     .WE(Ctrl_MemEW),
85     .RE(Ctrl_MemER),
86     .outs(Mem_Mux)
87 );
88 MuxMEM_ALU inst_MuxMem_ALU(
89     .inMem(Mem_Mux),
90     .inALU(ALU_Mux),
91     .outBanReg(MuxMem_ALU_Bank)
92 );
93 ALU_Control inst_AlucCtrl(
94     .inControl(Ctrl_AlucCtrl),
95     .inFunct(inst[5:0]),
96     .outAlu(AlucCtrl_ALU)
97 );
98 MuxUni_Bank inst_MuxUni_Bank(
99     .inS_Inst(inst[20:16]),
100     .inD_Inst(inst[15:11]),
101     .selUni(Ctrl_MuxAddr),
102     .outBank(MuxAddr_Bank)
103 );
104 MuxBank_SignExt inst_MuxBank_Sign(
105     .inBank(DR2_MuxALU),
106     .inSignExt(outSign_MuxALU),
107     .selUni(Ctrl_MuxSign_Bank),
108     .outS_ALU(MuxALU_DR2Bank_Sign)
109 );

```

Imagen 10. Módulo Toplevel parte 3

```

110 UnidadControl inst_UniCtrl(
111     .op(inst[31:26]),
112     .enW_Bank(Ctrl_Bank),
113     .enW_Mem(Ctrl_MemEW),
114     .enR_Mem(Ctrl_MemER),
115     .selMuxMem_ALU(Ctrl_MuxMem_ALU),
116     .selMuxAddr(Ctrl_MuxAddr),
117     .selMuxSign_Bank(Ctrl_MuxSign_Bank),
118     .selControl(Ctrl_AlucCtrl)
119 );
120 endmodule

```

Imagen 11. Módulo TopLevel parte 4.

```

1  `timescale 1ns /1ns
2  module TBTopLevel;
3  reg [31:0] inst;
4  reg [31:0]escribir;//recibiria los valores de entrada
5  reg [4:0]dirIniciar,dirLeer;
6  reg EWIniciar;
7  reg [1:0]sel;
8  wire [31:0]DR1,DR2;
9  wire [31:0]AluResul;
10  TopLevel inst_top(
11      .inst(inst),
12      .escribir(escribir),
13      .dirIniciar(dirIniciar),
14      .dirLeer(dirLeer),
15      .EWIniciar(EWIniciar),
16      .sel(sel),
17      .DR1(DR1),.DR2(DR2),
18      .AluResul(AluResul)
19  );
20  initial begin
21      //Iniciar registros de prueba inicializando solo 5
22      //registros en 0
23      sel=0;
24      EWIniciar=0;//poner en modo escritura
25      dirIniciar=0;
26      escribir=0;//r0=0
27      #20;
28      dirIniciar=1;
29      escribir=0;//r1=0
30      #20;
31      dirIniciar=2;
32      escribir=0;//r2=0
33      #20;
34      dirIniciar=3;
35      escribir=0;//r3=0
36      #20;
37      dirIniciar=4;
38      escribir=0;//r4=0
39      #20;

```

Imagen 12. Testbench topLevel parte 1.

```

42     sel=1;
43     inst=32'b0010000000100000000000000010100;//addi r1,r0,20
44     #20;
45     inst=32'b00110100001000100000000000000001;//ori r2,r1,1
46     #20;
47     inst=32'b00100100001000000000000000001010 ;//subi r0,r1,10
48     #20;
49     inst=32'b00110000010000110000000000000000;//andi r3,r2,0
50     #20;
51     inst=32'b00100000100000010000000001010000;//addi r4,r1,80
52     #20;
53     inst=32'b00000000001001001010000000000000;//add r20,r1,r4= r20 20
54     #20;
55     inst=32'b1010111010000001000000000110010;//sw r1,r20,50
56     #20;
57     inst=32'b1000111010000101000000000110010;//lw r5,r20,50
58     #20;
59     inst=32'b00000000101000100011000000001000;//mul r6,r5,r2
60     #20;
61     inst=32'b00000000000000010111000000000111;//slt r0,r1,r14
62     #20;
63     sel=0;
64     EWINiciar=1;//mostrar si se guardo
65     dirLeer=5'b00001;//ver addi
66     #20;
67     dirLeer=5'b00010;//ver ori
68     #20;
69     dirLeer=5'b00000;//ver subi
70     #20;
71     dirLeer=5'b00011;//ver andi
72     #20;
73     dirLeer=5'b00101;//ver lw
74     #20;
75     dirLeer=5'b00110;//ver mul
76     #20;
77     dirLeer=5'b00000;//ver slt
78     #20;
79     $stop;
80 end
81 endmodule

```

Imagen 13. Testbench toplevel parte 2.

En la imagen 14 se observa el ware final del testbench.



Imagen 14. Ware del testbench.

Conclusión

Las instrucciones tipo I se usan para cargar y guardar de la memoria, para operar en un solo registro y guardar en otro. Las operaciones de cargar y guardar sirven para solicitar más información de fuera del banco de registros para operar más opciones a la vez que estas operaciones se pueden guardar para su uso posterior.

Referencias

Patterson, D. A., & Hennessy, J. L. (2013). Computer Organization and Design MIPS Edition: The Hardware/Software Interface. Newnes.

<http://www.fdi.ucm.es/profesor/mendias/512/docs/tema16.pdf>

<https://s3-eu-west-1.amazonaws.com/downloads-mips/documents/MD00086-2B-MIPS32BIS-AFP-6.06.pdf>

<https://www.d.umn.edu/~gshute/mips/rtype.xhtml>

<http://www.pitt.edu/~kmram/CoE0147/lectures/datapath3.pdf>

<http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html>

<https://www.computerhope.com/jargon/d/datapath.htm>

<https://courses.cs.washington.edu/courses/cse378/09wi/lectures/lec08.pdf>

<https://homepage.cs.uiowa.edu/~ghosh/6016.90.pdf>

https://www.ibm.com/support/knowledgecenter/es/ssw_aix_72/com.ibm.aix.osdevice/pagspacdefsiz.htm