Laboratorio 4

Gonzalo Alfaro Caso

May 2019

1 Implementación de una ALU 'simplificada' de 32 bits

Para empezar con esta tarea se debe describir los modulos que se tomarán en cuenta para la implementación. Para esto se tomará la imagen referenciada en el ejercicio ***(img1)***. Es necesario comentar que esta ALU no esta completamente capacitada para recibir registros de tipo signed integer, el flag de overflow solo se activará cuando se sume un numero mayor de lo posible.

1.1 Register File 32x32

La implementación de este modulo incluye un archivo de texto con los registros que se ingresaran en el registro de 32x32 bits. El orden de los registros tiene el mismo que tendría un register file de la arquitectura MIPS ***(img2)***. En este caso solo los registros temporales tienen un valor diferente de 0 y son los que se utilizarán para la demostración del funcionamiento.

1.2 Multiplaxer 2x1

Este es un multiplaxer simple que tiene como entradas la salida 'data2' del register file y la inmediate data que debería venir de otro modulo en una implementación más completa, pero para este caso sera una señal enviada como entrada. La salida de este multiplaxer depende de un selector de 1 bit que tambien es enviado como señal en este caso.

1.3 ALU

Este modulo al igual igual que el multiplaxer esta hecho de manera estructural, ya que el resultado no depende del resultado anterior. En este caso solo se necesitó hacer que sea capáz de Realizar las operaciones AND, OR, ADD, SUB, SLT y NOR. Sus entradas dependen de las salidas 'data1' proveniente del register file y del output que tenga el multiplaxer.

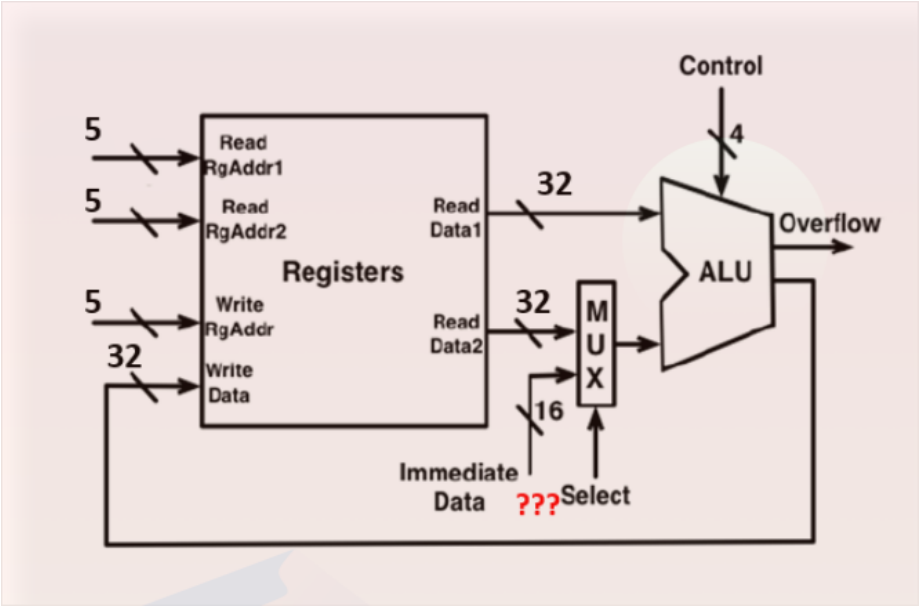

Figure 1: Datapath simplificado

2 Codigo fuente HDL de Verilog

2.1 Register File 32x32

```
1 module regfile32x32(clk,rgAddR1,rgAddR2,dataR1,dataR2,rgAddW,dataW);
2 input clk;
3 input [4:0] rgAddR1, rgAddR2, rgAddW;
4 input [31:0] dataW;
5 reg signed [31:0] regf [0:31];
6 output reg [31:0] dataR1,dataR2;
7
8 integer i;
9
10 initial
11   $readmemb("registers.txt",regf);
12
13 always @(posedge clk)
14 begin
15   dataR1 = regf[rgAddR1];
16   dataR2 = regf[rgAddR2];
17 end
18
19 always @(negedge clk)
20 begin
```

Name	Register Number	Usage	Should preserve on call?
$zero	0	the constant 0	no
$v0 - $v1	2-3	returned values	no
$a0 - $a3	4-7	arguments	yes
$t0 - $t7	8-15	temporaries	no
$s0 - $s7	16-23	saved values	yes
$t8 - $t9	24-25	temporaries	no
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return address	yes

Figure 2: Registros en arquitectura MIPS

```
21   regf[rgAddW] = dataW;
22 end
23
24 endmodule
```

2.2 Multiplaxer 2x1

```
1 module multiplaxer2x1(Rdata2,Idata,slc,out);
2 input [31:0] Rdata2;
3 input [15:0] Idata;
4 input slc;
5 output [31:0] out;
6
7 assign out = slc ? Idata:Rdata2;
8
9 endmodule
```

2.3 ALU

```
1 module alu32(Rdata1,Mdata,ctrl,ovf,out);
2 input [31:0] Rdata1,Mdata;
```

```
 3 input [3:0] ctrl;
 4 output ovf;
 5 output [31:0] out;
 6
 7 assign out = ctrl==0 ? Rdata1 & Mdata:
 8             ctrl==1 ? Rdata1 | Mdata:
 9             ctrl==2 ? Rdata1 + Mdata:
10             ctrl==6 ? Rdata1 - Mdata:
11             ctrl==7 ? Rdata1 < Mdata:
12             ctrl==12 ? Rdata1 ^ Mdata:0;
13
14 assign ovf = ctrl==2 ? Rdata1>out: 0;
15 endmodule
```

2.4 Testbench

Para probar si la implementación funciona a la perfección se realizó cada una de las funcionalidades que se pedian. Además se añadió un output extra que muestra un caso de overflow al sumar el número máximo con 1.

```
 1 module tb_ALU35();
 2 reg clk;
 3 reg [4:0] rgAddR1, rgAddR2, rgAddW;
 4 reg [31:0] dataW;
 5 wire [31:0] dataR1, dataR2;
 6
 7 regfile32x32 U1 (clk,rgAddR1,rgAddR2,dataR1,dataR2,rgAddW,dataW);
 8
 9 reg [15:0] Idata;
10 reg slc;
11 wire [31:0] outM;
12
13 multiplaxer2x1 U2 (dataR2,Idata,slc,outM);
14
15 reg [3:0] ctrl;
16 wire ovf;
17 wire [31:0] outA;
18
19 alu32 U3 (dataR1,outM,ctrl,ovf,outA);
20
21 initial
22 begin
23  clk=1;
24  rgAddR1 = 10; // $t2
25  rgAddR2 = 11; // $t3
26  rgAddW = 10; // $t2
27
28  slc = 0;
```

```
29  ctrl = 0;
30
31  #1
32  clk=0;
33  dataW = outA;
34
35  //....................
36
37  rgAddR1 = 10; // $t2
38  rgAddR2 = 15; // $t7
39
40  ctrl = 1;
41
42  #1
43  clk=0;
44  dataW = outA;
45
46  //....................
47
48  rgAddR1 = 10; // $t2
49  Idata = 5;
50
51  slc = 1;
52  ctrl = 2;
53
54  #1
55  clk=0;
56  dataW = outA;
57
58  //....................
59
60  rgAddR1 = 10; // $t2
61  rgAddR2 = 15; // $t7
62
63  slc = 0;
64  ctrl = 6;
65
66  #1
67  clk=0;
68  dataW = outA;
69
70  //....................
71
72  rgAddR1 = 10; // $t2
73  rgAddR2 = 8; // $t0
74
75  ctrl = 7;
76
77  #1
78  clk=0;
```

```
 79   dataW = outA;
 80
 81   //.....................
 82
 83   rgAddR1 = 10; // $t2
 84   rgAddR2 = 23; // $t8
 85
 86   ctrl = 12;
 87
 88   #1
 89   clk=0;
 90   dataW = outA;
 91
 92   //.....................
 93
 94   rgAddR1 = 24; // $t8
 95   rgAddR2 = 25; // $t9
 96
 97   ctrl = 2;
 98
 99   #1
100   clk=0;
101   rgAddW = 24; // $t8
102   dataW = outA;
103
104 end
105
106 initial
107   #10 $finish;
108
109 always
110   #1 clk=!clk;
111
112 initial
113 begin
114   $monitor("RegFile: input1 = %d input2 = %d \nMux: slc = %d imData
    = %d \nALU: ctrl=%d output=%d overflow = %d (clk=%d) \n ------",
    dataR1,dataR2,slc,Idata,ctrl,outA,ovf,clk);
115 end
116
117 endmodule
```

3 Outputs y archivos generales

En esta sección se añaden los outputs de compilar todos los modulos y el test-bench y el texto que incluye los datos ultilizados para el register file.

```
RegFile: input1 = 45 input2 = 21
```

```
Mux: slc = 0 imData = x
ALU: ctrl= 0 output= 5 overflow = 0 (clk=1)
 ------
RegFile: input1 = 5 input2 = 121
Mux: slc = 0 imData = x
ALU: ctrl= 1 output= 125 overflow = 0 (clk=1)
 ------
RegFile: input1 = 125 input2 = 121
Mux: slc = 1 imData = 5
ALU: ctrl= 2 output= 130 overflow = 0 (clk=1)
 ------
RegFile: input1 = 130 input2 = 121
Mux: slc = 0 imData = 5
ALU: ctrl= 6 output= 9 overflow = 0 (clk=1)
 ------
RegFile: input1 = 9 input2 = 10
Mux: slc = 0 imData = 5
ALU: ctrl= 7 output= 1 overflow = 0 (clk=1)
 ------
RegFile: input1 = 1 input2 = 0
Mux: slc = 0 imData = 5
ALU: ctrl=12 output= 1 overflow = 0 (clk=1)
 ------
RegFile: input1 = 4294967295 input2 = 1
Mux: slc = 0 imData = 5
ALU: ctrl= 2 output= 0 overflow = 1 (clk=1)
```

```
 1 00000000000000000000000000000000
 2 00000000000000000000000000000000
 3 00000000000000000000000000000000
 4 00000000000000000000000000000000
 5 00000000000000000000000000000000
 6 00000000000000000000000000000000
 7 00000000000000000000000000000000
 8 00000000000000000000000000000000
 9 00000000000000000000000000001010
10 00000000000000000000001001000100
11 00000000000000000000000000101101
12 00000000000000000000000000010101
13 00000000000000000000000000011101
14 00000000000000000000000001100001
15 00000000000000000000000000010101
16 00000000000000000000000001111001
17 00000000000000000000000000000000
18 00000000000000000000000000000000
19 00000000000000000000000000000000
20 00000000000000000000000000000000
21 00000000000000000000000000000000
22 00000000000000000000000000000000
```

```
23 00000000000000000000000000000000
24 00000000000000000000000000000000
25 11111111111111111111111111111111
26 00000000000000000000000000000001
27 00000000000000000000000000000000
28 00000000000000000000000000000000
29 00000000000000000000000000000000
30 00000000000000000000000000000000
31 00000000000000000000000000000000
32 00000000000000000000000000000000
```