**Lab # 6 - MIPS Datapath for R-type and I- Type Instructions**

CECS 341 – Computer Architecture Organization

Shawn Joseph,  ID# 025671644

Ty Tanaka, ID# 018208955

Yiang Shen,ID# 025794325

Professor: Jose Aceves

Logo

Description automatically generated

California State University, Long Beach

College of Engineering

1250 Bellflower Blvd, Long Beach, CA 90840

Lab Due Date 12/8/2020

**Goal/Objective**

The goal for this lab was to modify our R-type and I-type datapath to successfully execute J type instructions as well. We did so by adding another shift left 2 module, as well as by adding a Mux and by modifying our control file. In addition, we needed to create several .dat files for testing our code.

**Technical Description/Steps**

First, we needed to create a new shift left module that shifted a value left by 2 and had an input of 26/an output of 28 wires. We passed bits 26 through 0 of the instruction into this module and concatenated it with bits 31 through 28 of the updated program counter value. Afterwards, we routed this to bit 1 of a new Mux with an output of the program counter’s input and routed the previous update to the program counter to bit 0 of the program counter. This Mux is enabled by Jump, a new signal from the control unit that is enabled for J-type instructions.

To create the memory file, using the initial vales from the lab assignment, I used the last datapath file as a template and replaced the values with new ones. To create the program file, I did the same- first, I used the old file as a template, and then I assembled the instructions and put them in the project using the same format.

Waveform diagram:

Chart

Description automatically generated

Datapath schematic:

Diagram

Description automatically generated

Mux4x1 schematic:

Chart, diagram

Description automatically generated

SignExtend schematic:

A picture containing diagram

Description automatically generated

Mux32x1 schematic:

Diagram

Description automatically generated

ShiftLeft2 schematic:

Diagram

Description automatically generated

Control unit schematic:

Diagram, schematic

Description automatically generated

Program counter:

Diagram

Description automatically generated

ALU:

Diagram, schematic

Description automatically generated

ShiftLeft2\_26:

Diagram

Description automatically generated

Output:

Initial Memory values:

0: 00

1: 00

2: 00

3: 01

4: 00

5: 00

6: 00

7: 02

8: 00

9: 00

10: 00

11: 03

12: 00

13: 00

14: 00

15: 04

16: 00

17: 00

18: 00

19: 05

20: xx

21: xx

22: xx

23: xx

24: xx

25: xx

26: xx

27: xx

28: xx

29: xx

30: xx

31: xx

32: xx

33: xx

34: xx

35: xx

36: xx

37: xx

38: xx

39: xx

40: xx

41: xx

42: xx

43: xx

44: xx

Number: 0 Time: 30000, Output: 00000000

Number: 2 Time: 50000, Output: 00000005

Number: 4 Time: 70000, Output: 00000000

Number: 6 Time: 90000, Output: ffffffff

Number: 8 Time: 110000, Output: ffffffff

Number: 10 Time: 130000, Output: 00000001

Number: 12 Time: 150000, Output: 00000001

Number: 14 Time: 170000, Output: 00000001

Number: 16 Time: 190000, Output: 00000004

Number: 18 Time: 210000, Output: 00000000

Number: 20 Time: 230000, Output: ffffffff

Number: 22 Time: 250000, Output: ffffffff

Number: 24 Time: 270000, Output: 00000002

Number: 26 Time: 290000, Output: 00000003

Number: 28 Time: 310000, Output: 00000002

Number: 30 Time: 330000, Output: 00000008

Number: 32 Time: 350000, Output: 00000000

Number: 34 Time: 370000, Output: ffffffff

Number: 36 Time: 390000, Output: ffffffff

Number: 38 Time: 410000, Output: 00000003

Number: 40 Time: 430000, Output: 00000006

Number: 42 Time: 450000, Output: 00000003

Number: 44 Time: 470000, Output: 0000000c

Number: 46 Time: 490000, Output: 00000000

Number: 48 Time: 510000, Output: ffffffff

Number: 50 Time: 530000, Output: ffffffff

Number: 52 Time: 550000, Output: 00000004

Number: 54 Time: 570000, Output: 0000000a

Number: 56 Time: 590000, Output: 00000004

Number: 58 Time: 610000, Output: 00000010

Number: 60 Time: 630000, Output: 00000000

Number: 62 Time: 650000, Output: ffffffff

Number: 64 Time: 670000, Output: ffffffff

Number: 66 Time: 690000, Output: 00000005

Number: 68 Time: 710000, Output: 0000000f

Number: 70 Time: 730000, Output: 00000005

Number: 72 Time: 750000, Output: 00000014

Number: 74 Time: 770000, Output: 00000000

Number: 76 Time: 790000, Output: ffffffff

Number: 78 Time: 810000, Output: ffffffff

Number: 80 Time: 830000, Output: xxxxxxxx

Number: 82 Time: 850000, Output: xxxxxxxx

Number: 84 Time: 870000, Output: 00000006

Number: 86 Time: 890000, Output: 00000018

Number: 88 Time: 910000, Output: 00000000

Register 0: 00000000

Register 1: 00000000

Register 2: 00000000

Register 3: 00000000

Register 4: 00000000

Register 5: 00000000

Register 6: 00000000

Register 7: 00000000

Register 8: 00000006

Register 9: xxxxxxxx

Register 10: 00000005

Register 11: 00000018

Register 12: xxxxxxxx

Register 13: 0000000e

Register 14: 0000000f

Register 15: 00000010

Register 16: 00000011

Register 17: 00000012

Register 18: 00000013

Register 19: 00000014

Register 20: 00000015

Register 21: 00000016

Register 22: 00000017

Register 23: 00000018

Register 24: 00000019

Register 25: 0000001a

Register 26: 00000000

Register 27: 00000000

Register 28: 00000000

Register 29: 00000000

Register 30: 00000000

Register 31: 00000000

Final Memory values:

0: 00

1: 00

2: 00

3: 01

4: 00

5: 00

6: 00

7: 02

8: 00

9: 00

10: 00

11: 03

12: 00

13: 00

14: 00

15: 04

16: 00

17: 00

18: 00

19: 05

20: xx

21: xx

22: xx

23: xx

24: xx

25: xx

26: xx

27: xx

28: xx

29: xx

30: xx

31: xx

32: xx

33: xx

34: xx

35: xx

36: xx

37: xx

38: xx

39: xx

40: xx

41: xx

42: xx

43: xx

HDL code:

module ProgramCounter(

input clk,

input reset,

input [31:0] Din,

output reg[31:0] Dout

);

always @(posedge clk) begin

if(reset)

Dout = 0;

else

Dout = Din;

end

endmodule

module ShiftLeftTwo(

input [31:0] in,

output reg [31:0] out

);

always @(in) begin

out <= in << 2;

end

endmodule

module Mux32x1(

input [31:0] zero,

input [31:0] one,

input ctrl,

output reg [31:0] out

);

always @(zero or one or ctrl) begin

case(ctrl)

1'b0: out <= zero;

1'b1: out <= one;

//default: $display("Error in Mux32x1");

endcase

end

endmodule

module Mux4x1(

input [4:0] zero,

input [4:0] one,

input ctrl,

output reg [4:0] out

);

always @(zero or one or ctrl) begin

case(ctrl)

1'b0: out <= zero;

1'b1: out <= one;

//default: $display("Error in Mux4x1");

endcase

end

endmodule

module SignExtend(

input [15:0] in,

output reg [31:0] out

);

always @(in) begin

out <= {{16{in[15]}}, in[15:0]};

end

endmodule

module Control(

input [5:0] Op,

input [5:0] Func,

output reg RegDst, MemRead, MemToReg, MemWrite, ALUSrc, RegWrite, Jump,

output reg [1:0] Branch,

output reg [3:0] ALUCntl

);

always @(\*) begin

case(Op)

6'b0:begin

RegWrite <= 1'b1;

RegDst <= 1'b1;

Branch <= 2'b0;

MemRead <= 1'b0;

MemToReg <= 1'b0;

MemWrite <= 1'b0;

ALUSrc <= 1'b0;

Jump <= 1'b0;

case(Func)

6'h20:begin //add

ALUCntl <= 4'b1010;

end

6'h21:begin //addu

ALUCntl <= 4'b0010;

end

6'h22:begin //sub

ALUCntl <= 4'b1110;

end

6'h23:begin //subu

ALUCntl <= 4'b0110;

end

6'h24:begin //and

ALUCntl <= 4'b0000;

end

6'h25:begin //or

ALUCntl <= 4'b0001;

end

6'h26:begin //xor

ALUCntl <= 4'b0011;

end

6'h27:begin //nor

ALUCntl <= 4'b1100;

end

6'h2A:begin //SLT

ALUCntl <= 4'b1111;

end

6'h2B:begin //SLTU

ALUCntl <= 4'b0100;

end

default:begin //invalid code

//$display("Error! Check control file");

end

endcase

end

6'h08:begin //addi

RegWrite <= 1'b1;

Branch <= 2'b0;

MemWrite <= 1'b0;

MemRead <= 1'b0;

MemToReg <= 1'b0;

RegDst <= 1'b0;

ALUCntl <= 4'b1010;

ALUSrc <= 1'b1;

Jump <= 1'b0;

end

6'h09:begin //addiu

Branch <= 2'b0;

RegWrite <= 1'b1;

MemWrite <= 1'b0;

MemRead <= 1'b0;

MemToReg <= 1'b0;

RegDst <= 1'b0;

ALUCntl <= 4'b0010;

ALUSrc <= 1'b1;

end

6'h0C:begin //And immediate

RegWrite <= 1'b1;

Branch <= 2'b0;

MemWrite <= 1'b0;

MemRead <= 1'b0;

MemToReg <= 1'b0;

RegDst <= 1'b0;

ALUCntl <= 4'b0000;

ALUSrc <= 1'b1;

Jump <= 1'b0;

end

6'h0D:begin //Ori

RegWrite <= 1'b1;

Branch <= 2'b0;

MemWrite <= 1'b0;

MemRead <= 1'b0;

MemToReg <= 1'b0;

RegDst <= 1'b0;

ALUSrc <= 1'b1;

MemWrite <= 1'b0;

ALUCntl <= 4'b0001;

Jump <= 1'b0;

end

6'h23:begin //lw

Branch <= 2'b0;

RegDst <= 1'b0;

RegWrite <= 1'b1;

ALUSrc <= 1'b1;

ALUCntl <= 4'b1010;

MemWrite <= 1'b0;

MemRead <= 1'b1;

MemToReg <= 1'b1;

Jump <= 1'b0;

end

6'h2B:begin //sw

Branch <= 2'b0;

RegDst <= 1'b0;

RegWrite <= 1'b0;

ALUSrc <= 1'b1;

ALUCntl <= 4'b1010;

MemWrite <= 1'b1;

MemRead <= 1'b0;

MemToReg <= 1'b0;

Jump <= 1'b0;

end

6'h04:begin //beq

RegDst <= 1'b0;

Branch <= 2'b01;

RegWrite <= 1'b0;

ALUCntl <= 4'b1110;

ALUSrc <= 1'b0;

MemWrite <= 1'b0;

MemRead <= 1'b0;

MemToReg <= 1'b0;

Jump <= 1'b0;

end

6'h05:begin //bne

RegDst <= 1'b0;

Branch <= 2'b10;

RegWrite <= 1'b0;

ALUCntl <= 4'b1110;

ALUSrc <= 1'b0;

MemWrite <= 1'b0;

MemRead <= 1'b0;

MemToReg <= 1'b0;

Jump <= 1'b0;

end

6'h0A:begin //slti

Branch <= 2'b0;

RegWrite <= 1'b1;

MemWrite <= 1'b0;

MemRead <= 1'b0;

MemToReg <= 1'b0;

RegDst <= 1'b0;

ALUCntl <= 4'b1111;

ALUSrc <= 1'b1;

Jump <= 1'b0;

end

6'h0B:begin //sltiu

Branch <= 2'b0;

RegWrite <= 1'b1;

MemWrite <= 1'b0;

MemRead <= 1'b0;

MemToReg <= 1'b0;

RegDst <= 1'b0;

ALUCntl <= 4'b0100;

ALUSrc <= 1'b1;

Jump <= 1'b0;

end

6'h2:begin //jump

Branch <= 2'b11;

RegWrite <= 1'b0;

MemWrite <= 1'b0;

MemRead <= 1'b0;

MemToReg <= 1'b0;

RegDst <= 1'b0;

ALUCntl <= 4'b1110;

ALUSrc <= 1'b0;

Jump <= 1'b1;

end

default:begin

Branch <= 2'b0;

RegWrite <= 1'b0;

MemWrite <= 1'b0;

MemRead <= 1'b0;

MemToReg <= 1'b0;

RegDst <= 1'b0;

ALUCntl <= 4'b0100;

ALUSrc <= 1'b1;

Jump <= 1'b0;

//$display("Error! check control file. Opcode: %h, Function: %h", Op, Func);

end

endcase

end

endmodule

module shiftLeft2\_26(

input [25:0] in,

output [27:0] out

);

assign out = {in, 2'b0};

endmodule

module ALU(

input [31:0] A,

input [31:0] B,

input [3:0] ALUCntl,

output reg [31:0] ALUout,

output reg C,

output reg N,

output Z,

output reg V

);

reg signed [31:0]A\_s,B\_s;

assign Z = (ALUout == 32'b0)?1'b1:1'b0;

always@(\*)begin

A\_s = A;

B\_s = B;

case(ALUCntl)

4'b1010:begin //add

{C,ALUout} = A\_s + B\_s;

if((A\_s[31] & B\_s[31] & ~ALUout[31])||(~A\_s[31] & ~B\_s[31] & ALUout[31]))//If a is pos,b is pos and result is neg, Or vice visa

V = 1'b1;

else

V = 1'b0;

N = ALUout[31];

end

4'b1110:begin //Sub

{C,ALUout} = A\_s - B\_s;

if((A\_s[31] & ~B\_s[31] & ~ALUout[31])||(~A\_s[31] & B\_s[31] & ALUout[31]))

V = 1'b1;

else

V = 1'b0;

N = ALUout[31];

end

4'b0010:begin //add Unsigned

{C,ALUout} = A + B;

V = C;

N = 1'b0;

end

4'b0110:begin //Sub Unsigned

{C,ALUout} = A - B;

V = C;

N = 1'b0;

end

4'b0000:begin //And

{V,C,ALUout} = {2'bx,(A&B)};

N = ALUout[31];

end

4'b0001:begin //Or

{V,C,ALUout} = {2'bx,(A|B)};

N = ALUout[31];

end

4'b0011:begin //XOr

{V,C,ALUout} = {2'bx,(A^B)};

N = ALUout[31];

end

4'b1100:begin //Nor

{V,C,ALUout} = {2'bx,~(A|B)};

N = ALUout[31];

end

4'b0111:begin //Not

{V,C,ALUout} = {2'bx,~(A)};

N = ALUout[31];

end

4'b1101:begin //Not

{C,ALUout} = {(A)<<1};

V = 1'bx;

N = ALUout[31];

end

4'b1111:begin //SLT

C = 0;

V = 0;

N = 0;

if(A[31] & ~B[31])

ALUout = 0;

else if(B[31] & ~A[31])

ALUout = 32'hFFFFFFFF;

else if(~B[31] & ~A[31])

begin

if(A > B)

ALUout = 32'hFFFFFFFF;

else

ALUout = 0;

end

else

begin

if(A < B)

ALUout = 32'hFFFFFFFF;

else

ALUout = 0;

end

end

4'b0100: begin //SLTU

C = 0;

V = 0;

N = 0;

if(A > B)

ALUout = 32'hFFFFFFFF;

else

ALUout = 0;

end

default: begin

C = 0;

N = 0;

V = 0;

ALUout = 0;

//$display("Error in ALU!");

end

endcase

end

endmodule

module DataMem(

input clk,

input mem\_wr,

input mem\_rd,

input [31:0] addr,

input [31:0] wr\_data,

output [31:0] rd\_data

);

reg [7:0] dmem [0:4095];

// write

always@(posedge clk) begin

if(mem\_wr) begin

dmem[addr[11:0] + 2'd3] <= wr\_data[7:0];

dmem[addr[11:0] + 2'd2] <= wr\_data[15:8];

dmem[addr[11:0] + 2'd1] <= wr\_data[23:16];

dmem[addr[11:0] + 2'd0] <= wr\_data[31:24];

end

end

//read

assign rd\_data = (mem\_rd) ? { dmem[addr[11:0] + 2'd0],

dmem[addr[11:0] + 2'd1],

dmem[addr[11:0] + 2'd2],

dmem[addr[11:0] + 2'd3] }

: 32'hz;

Endmodule

module Mux32x1(

input [31:0] zero,

input [31:0] one,

input ctrl,

output reg [31:0] out

);

always @(zero or one or ctrl) begin

case(ctrl)

1'b0: out <= zero;

1'b1: out <= one;

//default: $display("Error in Mux32x1");

endcase

end

endmodule

module Datapath(input clk, input reset, output [31:0] Dout);

wire [31:0] ALUout;

wire [31:0]Cout;

wire [31:0]Cin;

wire [31:0]Inst\_out,S,T;

wire [3:0]ALUCntl;

wire RegWrite,C,N,Z,V;

wire [4:0] wr;

wire [1:0] br;

wire RegDst;

wire mr;

wire mtr;

wire mw;

wire ALUSrc;

wire jump;

wire [31:0] ALUin;

wire [31:0] SEval;

wire [31:0] rd;

wire [31:0] shl2;

wire [31:0] Co4;

wire [27:0] jse;

wire [31:0] jAddr;

wire [31:0] PCUpdate;

ProgramCounter pc(.clk(clk),.reset(reset),.Din(Cin),.Dout(Cout));

assign Co4 = Cout + 3'd4;

ShiftLeftTwo sll(.in(SEval), .out(shl2));

Mux32x1 adder(.zero(Co4), .one(shl2 + Co4), .ctrl(br[0] & Z | br[1] & ~Z), .out(PCUpdate));

Instruction\_Memory imem(.Addr(Cout),.Inst\_out(Inst\_out));

Mux4x1 m(.zero(Inst\_out[20:16]), .one(Inst\_out[15:11]), .ctrl(RegDst), .out(wr));

regfile32 rf32(.clk(clk),.reset(reset),.D\_En(RegWrite),.D\_Addr(wr),

.S\_Addr(Inst\_out[25:21]),.T\_Addr(Inst\_out[20:16]), .D(Dout),.S(S),.T(T) );

SignExtend se(.in(Inst\_out[15:0]), .out(SEval));

Control cntl(.Op(Inst\_out[31:26]),.Func(Inst\_out[5:0]), .ALUCntl(ALUCntl), .RegWrite(RegWrite),

.Branch(br), .MemRead(mr), .MemToReg(mtr), .MemWrite(mw), .ALUSrc(ALUSrc), .RegDst(RegDst), .Jump(jump));

Mux32x1 e(.zero(T), .one(SEval), .ctrl(ALUSrc), .out(ALUin));

ALU alu(.A(S),.B(ALUin),.ALUCntl(ALUCntl),.ALUout(ALUout),.C(C),.N(N),.Z(Z),.V(V));

DataMem d(.clk(clk), .addr(ALUout), .wr\_data(T), .mem\_wr(mw), .mem\_rd(mr), .rd\_data(rd));

Mux32x1 dMux(.one(rd), .zero(ALUout), .ctrl(mtr), .out(Dout));

shiftLeft2\_26 sladdr(.in(Inst\_out[25:0]), .out(jse));

assign jAddr = {Co4[31:28], jse};

Mux32x1 jumpMux(.one(jAddr), .zero(PCUpdate), .ctrl(jump), .out(Cin));

Endmodule

.DAT files:

DataMem.dat:

00 00 00 01

00 00 00 02

00 00 00 03

00 00 00 04

00 00 00 05

Imem.dat:

00 00 08 20

00 00 09 20

05 00 0a 34

00 00 0b 20

2a 60 0a 01

05 00 80 11

00 00 6c 8d

20 48 2c 01

01 00 08 21

04 00 6b 21

04 0c 00 08

00 00 68 ad

04 00 69 ad

08 00 6a ad

04 00 6b 21

04 0c 00 08

00 00 68 ad

04 00 69 ad

08 00 6a ad

0c 00 6b ad

Conclusion:

We were successfully able to execute the program. We utilized the waveform diagram extensively to debug our program, and were able to get it to function properly.