**ECEN 323 – Winter 2020**

Lab 6: RISC-V Datapath

Johnson, Ryan

Section 1

Preliminary

**Simulation Tutorials**

What signals are placed on the waveform viewer by default when you first start the simulator?

The ports, internal signals, and parameters from the top-level module.

What is the purpose of “dividers” in the waveform window?

They allow us to group and title related signals for clarity of viewing

How do you move the yellow waveform cursor to a specific location in time?

You right click in the waveform window, select “Go To…” and type in the specific time you want to go to.

**Immediate ALU Instructions**

Immediate Instruction Table

| **32-bit instruction** | **neumonic** | **rs1** | **rd** | **32-bit Immediate Value: hex (dec)** |
| --- | --- | --- | --- | --- |
| 0xff918113 | addi | 3 | 2 | 0xfffffff9 (-7) |
| 0x01922193 | slti | 4 | 3 | 0x00000019 (25) |
| 0x0ff17213 | andi | 2 | 4 | 0x000000ff (255) |
| 0xf9c22193 | slti | 4 | 3 | 0xffffff9c (-100) |
| 0x7ff0c293 | xori | 1 | 5 | 0x000007ff (2047) |

How can you tell from the binary instruction that each of these instructions are immediate instructions?

You can tell because the opcode corresponds to the I instruction type.

Provide a Verilog dataflow statement that determines a 32-bit sign-extended immediate value from an immediate ALU instruction defined by the signal 'instruction[31:0]'

Assign immediate = {20{instruction[31]}, instruction[31:19]};

Branch ALU Instructions

Branch Instruction Table

|  |
| --- |
|  |
| **32-bit** | **Current PC** | **Branch** | **Offset** | **Branch Target** |
| **instruction** |  | **Type** | **hex (dec)** | **(hex)** |
| 0x02600c63 | 0x00000004 | beq | 0x038 (56) | 0x0000003C |
| 0xfe139ee3 | 0x00000008 | bne | 0xfffc (-4) | 0x00000004 |
| 0x01bfd263 | 0x00000020 | bge | 0x004 (4) | 0x00000024 |
| 0xfe5a42e3 | 0x00000024 | blt | 0xffe4 (-28) | 0x00000008 |

How can you tell that each of these instructions are branch instructions?

The opcode corresponds to the fact that they are branch instructions.

Provide a Verilog dataflow statement that determines a 32-bit sign-extended branch offset from a branch instruction defined by the signal 'instruction[31:0]'

assign offset = {20{instruction[31]},instruction[7],instruction[30:25],instruction[11:8]};

Exercise #1

For each of the following single-bit control signals, provide the following: A) a sentence or two describing the purpose of the control signal, B) identify an instruction that requires the corresponding control signal to be set to a “1”, and C) identify an instruction that requires the corresponding control signal to be set to a “0”.

|  |
| --- |
|  |
| **Control Signal** | **Summary of Purpose of Signal** | **Instruction '0'** | **Instruction '1'** |
| PCSrc | Multiplexer for the PC input. Selects either PC+4 or the PC plus an immediate offset | add | beq |
| ALUSrc | Multiplexer for the ALU input. Selects either a register’s value or the immediate value provided. | add | addi |
| RegWrite | Designates if the register on the Write register input is written with the new value from the write data input | beq | add |
| MemtoReg | Designates if the value in the memory location will be taken to the register | add | ld |
| MemRead | Designates if the value in the memory location given is accessed | add | ld |
| MemWrite | Designates if the memory location is written with the new value | beq | sd |

What is the purpose of the Zero status signal that goes from the datapath to the control unit?

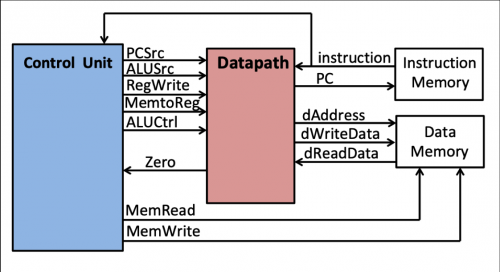
The PC is replaced by the branch target if the Zero output of the ALU is high.

| **ALU Control Lines** | **Function** |
| --- | --- |
| 0000 | AND |
| 0001 | OR |
| 0010 | add |
| 0110 | subtract |
| 0111 | set less than |
| 1100 | XOR |

Create a table and describe what type of instruction is executing in the datapath for each of the following conditions. Use the ALU table above to determine the instruction that is being executed according to the control signals provided. i.e. add, xori, beq.

| **PCSrc** | **ALUSrc** | **RegWrite** | **MemtoReg** | **MemRead** | **MemWrite** | **ALUCtrl** | **instruction** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 0 | 0 | 0000 | and |
| 0 | 0 | 1 | 0 | 0 | 0 | 0001 | or |
| 0 | 0 | 1 | 0 | 0 | 0 | 0010 | add |
| 0 | 0 | 1 | 0 | 0 | 0 | 0110 | sub |
| 0 | 0 | 1 | 0 | 0 | 0 | 0111 | slt |
| 0 | 0 | 1 | 0 | 0 | 0 | 1100 | xor |
| 0 | 1 | 1 | 0 | 0 | 0 | 0010 | addi |
| 0 | 1 | 1 | 0 | 0 | 0 | 0111 | slti |
| 0 | 1 | 1 | 0 | 0 | 0 | 0001 | ori |
| 0 | 1 | 1 | 0 | 0 | 0 | 0000 | andi |
| 0 | 1 | 1 | 0 | 0 | 0 | 1100 | xori |
| 0 | 1 | 1 | 1 | 1 | 0 | 0010 | ld |
| 0 | 1 | 0 | 0 | 0 | 1 | 0010 | sd |
| 0 | 0 | 0 | 0 | 0 | 0 | 0110 | branch not taken |
| 1 | 0 | 0 | 0 | 0 | 0 | 0110 | branch taken |

Note: the last two functions are conditional 'branches' but they do not behave the same. Try to identify the operational differences between these two branches.



Exercise #2

Include a copy of your Verilog datapath in your laboratory report

`timescale 1ns / 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Module: riscv\_simple\_datapath

\*

\* Author: Ryan Johnson

\* Class: ECEN 323, Section 01, Winter 2020

\* Date: 11 Feb 2020

\*

\* Description: acts as a datapath to process instructions

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

module riscv\_simple\_datapath #(parameter INITIAL\_PC = 32'h00400000) (

input wire logic clk,

input wire logic rst,

input wire logic [31:0] instruction,

input wire logic PCSrc,

input wire logic ALUSrc,

input wire logic RegWrite,

input wire logic [3:0] ALUCtrl,

output logic [31:0] PC,

output logic Zero,

output logic [31:0] dAddress,

output logic [31:0] dWriteData,

input wire logic [31:0] dReadData,

input wire logic MemtoReg

);

logic [31:0] RF[31:0];

integer i; // i needs to be declared before it is used

initial

for (i = 0; i < 32; i=i+1)

RF[i] = 0; // initializes each register to 0

logic [4:0] rs1;

logic [4:0] rs2;

logic [4:0] rd;

assign rs1 = instruction[19:15];

assign rs2 = instruction[24:20];

assign rd = instruction[11:7];

logic [31:0] writeDataR;

logic [31:0] aluResult;

assign writeDataR = (MemtoReg) ? dReadData : aluResult;

//writes data to the specified register if RegWrite is asserted

always\_ff@ (posedge clk)

if (RegWrite && rd != 0) RF[rd] <= writeDataR;

logic [31:0] data1;

logic [31:0] data2;

logic [31:0] imm;

logic [31:0] immI;

logic [31:0] immS;

localparam SOpCode = 7'b0100011;

assign immI = {{20{instruction[31]}},instruction[31:20]};

assign immS = {{20{instruction[31]}},instruction[31:25],instruction[11:7]};

assign imm = (instruction[6:0] == SOpCode) ? immS : immI;

assign data1 = RF[rs1];

assign data2 = (ALUSrc) ? imm:

RF[rs2];

logic [31:0] writeDataM;

assign writeDataM = RF[rs2];

localparam AND = 4'b0000;

localparam OR = 4'b0001;

localparam ADD = 4'b0010;

localparam SUB = 4'b0110;

localparam SLT = 4'b0111;

localparam XOR = 4'b1100;

assign aluResult = (ALUCtrl == AND) ? data1 & data2 :

(ALUCtrl == OR) ? data1 | data2 :

(ALUCtrl == ADD) ? data1 + data2 :

(ALUCtrl == SUB) ? data1 - data2 :

(ALUCtrl == SLT) ? ($signed(data1) < $signed(data2)) :

(ALUCtrl == XOR) ? data1 ^ data2 : 0;

assign Zero = (aluResult == 0) ? 1'b1 : 1'b0;

localparam usual\_offset = 4;

logic [31:0] branch\_offset;

assign branch\_offset = {{20{instruction[31]}},instruction[7],

instruction[30:25],instruction[11:8],1'b0};

// updates the pc to pc + 4 or to pc + branch\_offset

always\_ff@ (posedge clk)

if (rst) PC <= INITIAL\_PC;

else if (PCSrc) PC <= PC + branch\_offset;

else PC <= PC + usual\_offset;

always\_comb

begin

dWriteData = writeDataM;

dAddress = aluResult;

end

endmodule

Exercise #3

Copy the last 10 lines of your testbench console output and add it to your lab report

===== RISCV DATAPATH TB V1.04 =====

Runs for 11180 ns or use run -all

[0ns]Testing Reset

[100ns]Testing x0

[120ns]Loading Initial Values

INFO: [USF-XSim-96] XSim completed. Design snapshot 'datapath\_tb\_behav' loaded.

INFO: [USF-XSim-97] XSim simulation ran for 1000ns

launch\_simulation: Time (s): cpu = 00:00:04 ; elapsed = 00:00:23 . Memory (MB): peak = 1307.242 ; gain = 0.000

run all

[1080ns]Testing 1000 random commands

You Passed!

$finish called at time : 11180 ns : File "J:/ECEN 323 Labs/project\_6/datapath\_tb.sv" Line 712

Exercise #4

Summarize and justify any warnings you had during synthesis. If you did not have any warnings, say such in your laboratory report.

Design riscv\_simple\_datapath has unconnected port instruction[14:12]

* These warnings state that there are bits in the instruction port that aren’t connected. However, this is fine because I don’t need those, and I use the rest of the bits in the instruction input.

Summarize the estimated resources for your synthesized logic in the table below.

| **Resource** | **Estimation** |
| --- | --- |
| LUT | 323 |
| FF | 32 |
| IO | 168 |
| BUFG | 1 |

How many hours did you work on the lab?

5 hours

Please provide any suggestions for improving this lab in the future:

None. Great stuff.