Lab 4: Single Cycle Implementation of RISC-V (part 2)

Objectives

The objective of this lab is to continue modeling and simulating all the non-memory components of the RISC-V processor as a step towards completing its single cycle implementation.

This lab is organized as follows:

1. An introduction
2. Experiments to be conducted
3. Lab Report Requirements

Introduction

In the previous labs, you should have completed the Verilog modeling and simulation of some of the non-memory components shown in Figure 1. In this lab, you should resume the modeling and simulation of all the remaining non-memory components. Again, we are only interested to support the following RV32I ISA subset:

- Memory reference: LW (I-Format), SW (S-Format)
- Arithmetic/logical: ADD, SUB, AND, OR (R-Format)
- Control transfer: BEQ (SB-Format)

The goal of this lab is to model and simulate the remaining non-memory components necessary for this implementation.

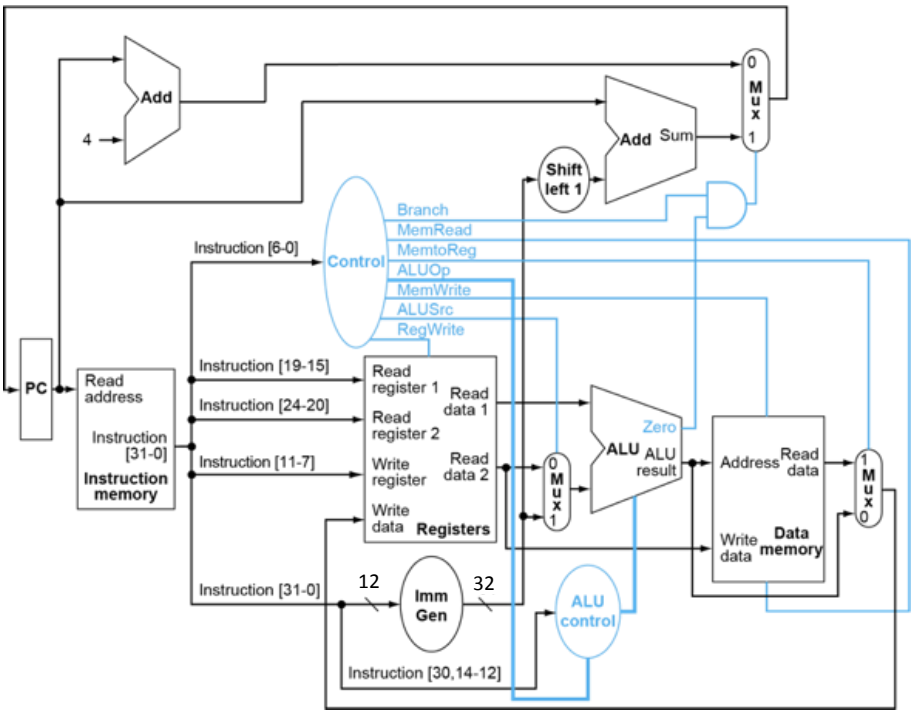

Figure 1 RISC-V Single Cycle Datapath

Table 1 shows the instruction encoding for the RV32I instructions highlighting the instructions we will implement in this lab.

Table 1 RV32I instructions encoding

imm 31:12			rd	0110111	LUI
imm 31:12			rd	0010111	AUIPC
imm 20 10:1 11 19:12			rd	1101111	JAL
imm 11:0	rs1	000	rd	1100111	JALR
imm 12 10:5	rs2	rs1	000	imm 4:1 11	1100011	BEQ
imm 12 10:5	rs2	rs1	001	imm 4:1 11	1100011	BNE
imm 12 10:5	rs2	rs1	100	imm 4:1 11	1100011	BLT
imm 12 10:5	rs2	rs1	101	imm 4:1 11	1100011	BGE
imm 12 10:5	rs2	rs1	110	imm 4:1 11	1100011	BLTU
imm 12 10:5	rs2	rs1	111	imm 4:1 11	1100011	BGEU
imm 11:0		rs1	000	rd	0000011	LB
imm 11:0		rs1	001	rd	0000011	LH
imm 11:0		rs1	010	rd	0000011	LW
imm 11:0		rs1	100	rd	0000011	LBU
imm 11:0		rs1	101	rd	0000011	LHU
imm 11:5	rs2	rs1	000	imm 4:0	0100011	SB
imm 11:5	rs2	rs1	001	imm 4:0	0100011	SH
imm 11:5	rs2	rs1	010	imm 4:0	0100011	SW
imm 11:0		rs1	000	rd	0010011	ADDI
imm 11:0		rs1	010	rd	0010011	SLTI
imm 11:0		rs1	011	rd	0010011	SLTIU
imm 11:0		rs1	100	rd	0010011	XORI
imm 11:0		rs1	110	rd	0010011	ORI
imm 11:0		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

In this lab, we are going to model and test the following components:

1. A register file with reset
2. A 32-bit ALU with zero output flag
3. The control unit
4. The ALU control unit

Note: Students should be working in pairs to implement the following experiments

Experiments

Experiment 1: A Register File with Reset

Create and simulate the RISC-V register file. It should contain 32 registers each having 32-bit with 2 reading ports and one writing port that is active only if RegWrite signal is active as shown in Figure 2. The register file should use two 32-bit 32x1 multiplexers for the reading ports and a 5x32 decoder for writing port. Please note that the register file should also have clock and reset inputs (not shown in the figure).

Figure 2 RISC-V Register File

Here is the Register File module skeleton that you should complete and simulate.

```
module RegFile (
      input clk, rst,
      input [4:0] readreg1, readreg2, writereg,
      input [31:0] writedata,
      input regwrite
      output [31:0] readdata1, readdata2 );
    ...
    ...
    ...
endmodule
```

Write a good testbench that performs specific test cases along with their verification, similar to the one shown below:

```
module RegFile_tb();

reg clk, rst;
reg [4:0] readreg1, readreg2, writereg;
reg [31:0] writedata;
reg regwrite;
wire [31:0] readdata1, readdata2;

// instantiate device under test
RegFile dut(clk, rst, readreg1, readreg2, writereg, writedata, regwrite,
readdata1, readdata2);

// apply clock
initial begin
      clk = 0;
      forever #50 clk = ~clk;
end


// apply inputs one at a time
// checking results
```

```
initial begin
	// check that registers contents = 0 when rst = 1
	$display("test 1 ");
	rst = 1; readreg1 = 10; readreg2 = 20;
	regwrite = 0;
	#100
	if(readdata1 == 0 && readdata2 == 0)
		$display("passed");
	else
		$display("failed");

	// check that data is not written and read on readreg1 if regwrite = 0
	$display("test 2 ");
	rst = 0;
	writedata = 78894131; writereg = 15;
	readreg1 = 15;
	#100
	if(readdata1 == 0 && readdata2 == 0)
		$display("passed");
	else
		$display("failed");

	// check that new data is written and read on readreg1 if regwrite = 1
	$display("test 3 ");
	regwrite = 1;
	#100
	if(readdata1 == 78894131 && readdata2 == 0)
		$display("passed");
	else
		$display("failed");

	// check that data is not written and read on readreg2 if regwrite = 0
	$display("test 4 ");
	writedata = 98765; writereg = 25;
	readreg2 = 25;
	regwrite = 0;
	#100
	if(readdata1 == 78894131 && readdata2 == 0)
		$display("passed");
	else
		$display("failed");

	 // check that new data is written and read on readreg2 if regwrite = 1
	$display("test 5 ");
	regwrite = 1;
	#100
	if(readdata1 == 78894131 && readdata2 == 98765)
		$display("passed");
	else
		$display("failed");

end
endmodule
```

Experiment 2: A 32-bit ALU with zero output flag

Create and simulate a 32-bit ALU with 4-bit selection lines allowing up to 16 different arithmetic and logic operations. We will actually implement the first 4 operations only as addition, subtraction, ANDing, and ORing while the remaining 12 operations will produce zero output regardless from the input. The selections line for adding, subtraction, ANDing, and ORing should be 0010, 0110, 0000, and 0001 respectively. All other selection line combinations should produce a zero output. Please note that you should use the same circuit for adding and subtraction. You can use any of the adders you developed during Lab1 after extending it to 32-bits and modifying it to become an adder/subtractor. The ALU should have a 32-bit 16x1 multiplexer to select one out of the 16 possible outputs generated by the ALU. The ALU should have a 32-bit output in addition to a single bit output representing a zero flag that should only be equal to 1 when all 32 bits are zero.

Here is the 32-bit ALU module skeleton that you should complete and simulate.

```
module ALU32bit (
      input [31:0] in1, in2,
      input [3:0] aluSel,
      output reg [31:0] result, output zero );
    ...
    ...
    ...
endmodule
```

Experiment 3: The control unit

Create and simulate a module representing the control unit. The control unit is a combinational circuit with the following truth table (enough to support to the 7 instructions we are interested in).

Instruction	Inst[6-2]	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegWrite
R-Format	01100	0	0	0	10	0	0	1
LW	00000	0	1	1	00	0	1	1
SW	01000	0	0	X	00	1	1	0
BEQ	11000	1	0	X	01	0	0	0

Experiment 4: The ALU control unit

Create and simulate a module representing the ALU control unit. Use the following truth table:

ALUOp	Inst[14-12]	Inst[30]	ALU Selection
00	X	X	0010 (ADD)
01	X	X	0110 (SUB)
10	000	0	0010 (ADD)
10	000	1	0110 (SUB)
10	111	0	0000 (AND)
10	110	0	0001 (OR)

Deliverables

General Report Submission Notes:

- Each student is required to submit an individual report.
- The submission should be composed of **two files only**:
  1. YourID_Report.pdf/doc:
     A single pdf/word file having the solution of the lab report questions in correct sequence.
  2. YourID_labwork.zip/rar/…:
     A zipped folder containing your codes for all the lab experiments organized in sequence.
- **Submission deadline**: half an hour before the start of next week's session.
- **Submission method**: On Blackboard

Lab Report [10 pts]

1. [0 pts] Your name and student ID.
2. [2 pts] A technical summary of experiments conducted in the lab (steps, results, components, code functionality, etc…).
3. [1 pts] Verilog code for all modules of each experiment and Verilog code for the testbench module of each experiment's top-level module.
4. [1 pts] Snapshot of simulation output corresponding to the submitted testbench with a brief interpretation of the snapshot.
5. [1.5 pts] Develop a Verilog module for a 64-bit 8x1 multiplexer using array indexing. (Don't use if/case/gate level modeling)
6. [1.5 pts] Research Q: discuss the difference between a processor register file and cache memory. Mention the functions and pros and cons of each in a tabular form.
7. [3 pts] Develop an ALU module similar to that of experiment 2 that includes the following extra functions:
   a) Support for XORing the two ALU inputs together.
   b) Support for logical shifting of the 1st ALU input to the right with a number of bits equal to the value on the 2nd ALU input.
   c) Support for logical shifting of the 1st ALU input to the left with a number of bits equal to the value on the 2nd ALU input.
   d) Support for arithmetic shifting of the 1st ALU input to the right with a number of bits equal to the value on the 2nd ALU input.
   e) An output *overflow* flag to indicate whether the addition/subtraction caused an overflow.

   Modify the original Verilog code of experiment 2 and highlight the changes.