Labs 5 & 6: Single Cycle Implementation of RISC-V (part 3)

Objectives

The objective of this lab is to complete the implementation of the RISC-V single cycle datapath and to test it on the Nexys A7 trainer board.

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 all non-memory components shown in Figure 1 assuming that we only need 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 the memory components (both the instruction memory and the data memory) and to construct the entire datapath out of previously modelled components.

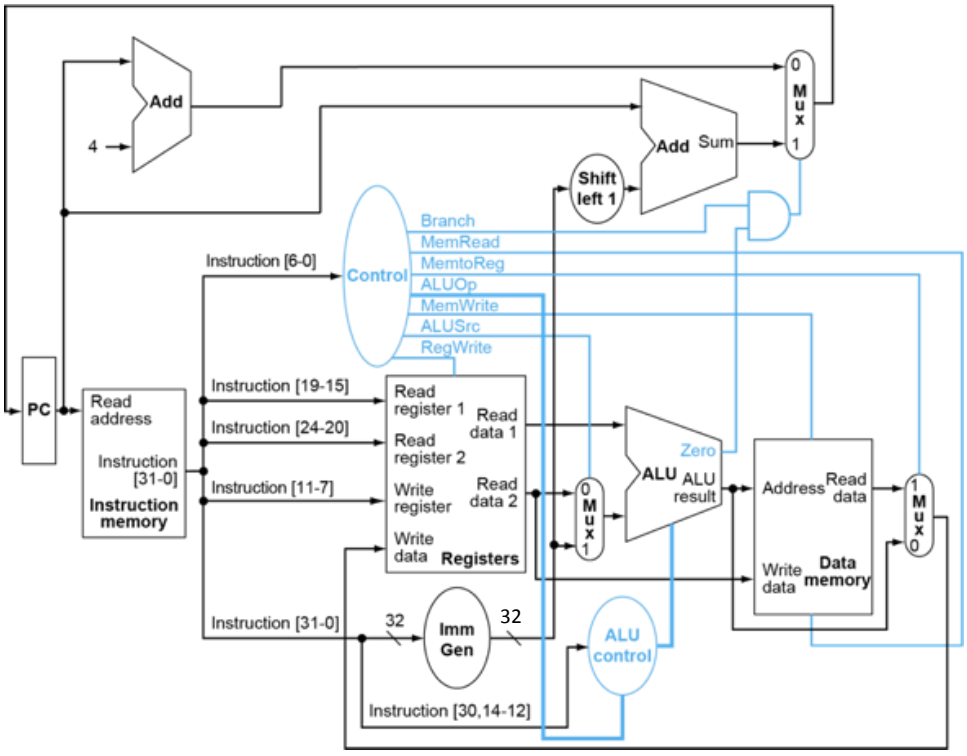

Figure 1 RISC-V Single Cycle Datapath

Table 1 shows the instruction encoding for the RV32I instructions highlighting the supported instructions:

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 conduct the following experiments:

1. Instruction Memory Modelling and Simulation
2. Data Memory Modelling and Simulation
3. Constructing the Full Datapath and Testing it on the Nexys A7 Board
4. Testing Using Your Own Program

Experiments

Experiment 1: Instruction Memory Modelling and Simulation

RISC-V processor has a byte addressable instruction memory with a maximum capacity of 1 Giga Words (4 GBytes) with 32 address bits. Figure 2 shows the RISC-V instruction memory. This is not suitable for our experimental setup since we cannot implement this amount of memory on the FPGA board. Also, since there is an implicit assumption that only entire words can be read out of the instruction memory, we will implement **a word addressable** instruction memory with a maximum capacity of 64 words (256 bytes) with 6 address bits; however, since the PC contains the byte address of the instruction to be executed, we must divide it by 4 (discard the least significant 2 bits) before connecting it to the read address input of the instruction memory to convert it to a word address.

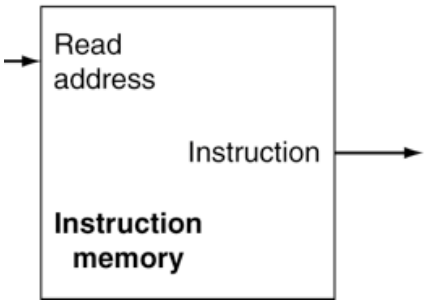

Figure 2 RISC-V Instruction Memory

The instruction memory module can be defined as follows:

```
module InstMem (input [5:0] addr, output [31:0] data_out);
    reg [31:0] mem [0:63];
    assign data_out = mem[addr];
endmodule
```

Develop a testbench module to simulate the instruction memory you defined. Since the instruction memory as defined above is essentially a read only memory (ROM), you will need to initialize some of the entries of the memory to test it. You can do so by adding an initial block inside the InstMem module.

Your testbench needs to include at least 5 test cases and verification codes.

Experiment 2: Data Memory Modelling and Simulation

Figure 3 shows the RISC-V data memory. For similar reasons to the instruction memory, we will implement **a word addressable** data memory with a maximum capacity of 64 words (256 bytes) with 6 address bits. Similarly, since the ALU computes the byte address of the data item to be loaded or stored, we must divide it by 4 (discard the least significant 2 bits) before connecting it to the address input of the data memory to convert it to a word address. Please note that the data memory also has a clock input that is not shown in the diagram.

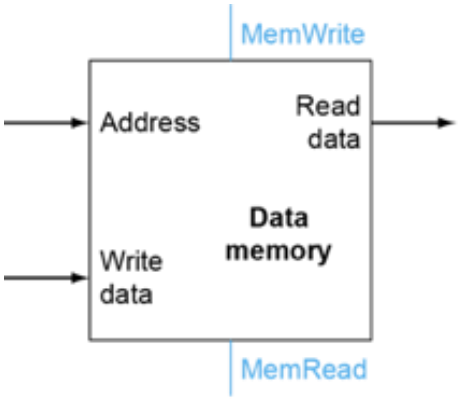

Figure 3 RISC-V Data Memory

The data memory module can be defined as follows:

```
module DataMem
    (input clk, input MemRead, input MemWrite,
    input [5:0] addr, input [31:0] data_in, output [31:0] data_out);

    reg [31:0] mem [0:63];

    always @(posedge clk)
    begin
        if (MemWrite)
            mem[addr] <= data_in;
    end

    assign data_out = MemRead ? mem[addr]:0;
endmodule
```

Develop a testbench module to simulate the data memory you defined. Please note that the data memory allows for asynchronous reading and synchronous writing and can be tested without being initialized. **(why?, what does asynchronous read means?)**

Experiment 3: Constructing the Full Datapath and simulating it

Now that all the components of Figure 1 has been defined as Verilog modules, you can put them all together to construct the full datapath. Please note that as shown in the figure the datapath does not have any input or output ports (except for implicit clock and reset input signals).

To be able to test the RISC-V implementation on a program, we need to initialize the instruction and data memories. This can be done by adding initial blocks in each of them.

Here is a sample set of data items to initialize the data memory:

```
initial begin
    mem[0]=32'd17;
    mem[1]=32'd9;
    mem[2]=32'd25;
end
```

And below is a sample program that uses the above data that can be used to initialize the instruction memory.

```
initial begin
    mem[0]=32'b0000000_00000_00000_000_00000_0110011 ;      //add x0, x0, x0
    mem[1]=32'b000000000000_00000_010_00001_0000011 ;       //lw x1, 0(x0)
    mem[2]=32'b000000000100_00000_010_00010_0000011 ;       //lw x2, 4(x0)
    mem[3]=32'b000000001000_00000_010_00011_0000011 ;       //lw x3, 8(x0)
    mem[4]=32'b0000000_00010_00001_110_00100_0110011 ;      //or x4, x1, x2
```

```
    mem[5]=32'b0_000000_00011_00100_000_0100_0_1100011 ;    //beq x4, x3, L
    mem[6]=32'b0000000_00010_00001_000_00011_0110011 ;      //add x3, x1, x2
    mem[7]=32'b0000000_00010_00011_000_00101_0110011 ;      //L: add x5,x3,x2
    mem[8]=32'b0000000_00101_00000_010_01100_0100011;       //sw x5, 12(x0)
    mem[9]=32'b000000001100_00000_010_00110_0000011 ;       //lw x6, 12(x0)
    mem[10]=32'b0000000_00001_00110_111_00111_0110011 ;     //and x7, x6, x1
    mem[11]=32'b0100000_00010_00001_000_01000_0110011 ;     //sub x8, x1, x2
    mem[12]=32'b0000000_00010_00001_000_00000_0110011 ;     //add x0, x1, x2
    mem[13]=32'b0000000_00001_00000_000_01001_0110011 ;     //add x9, x0, x1
end
```

Do not forget to reset your circuit to initialize the PC and all the registers in the register file to 0.

Did the program above change the contents of register x0? In case it did, **fix your register file module implementation to prevent this.**

You can start by simulating your program. And after confirming the simulation is working correctly you can start testing it on the board as described later in Experiment 4.

Experiment 4: Testing the Datapath on the Nexys A7 Board

The datapath does not have any input or output ports (except for implicit clock and reset input signals). If the datapath is implemented without any ports, we will not be able to verify that it is actually executing instructions, so we need to add a monitoring mechanism allowing us to peek inside the processor and check some of the values on the wire.

In order to do so we will define the following 2 output ports:

1. **An 8-bit output port connected to 8 status LEDs on the Nexys A7 board.**
   Based on a 2-bit led selection input port (e.g., ledSel) connected to 2 of the Nexys A7 input switches, these LEDs will be used to view either:
   a. Some of the control signals. When ledSel = 00, the following signals RegWrite, ALUSrc, 2-bit ALUOp, MemRead, MemWrite, MemtoReg, Branch should be presented on the status LEDs.
   b. Remaining control signals. When ledSel = 01, the following signals 4-bit ALU selection lines, the zero flag, the output of the branching AND gate should be presented on the status LEDs.
   c. When ledSel = 10 or 11, choose any other internal signals you need to use for monitoring the datapath.

2. **A 13-bit output port connected to one of the two 4-digits seven segment displays (SSD) on the Nexys A7 board.** Based on a 4-bit SSD selection input port (e.g., ssdSel) connected to 4 of the Nexys A7 input switches, the SSD will be used to view either:
   a. The PC output (when ssdSel = 0000)
   b. The PC+4 adder output (when ssdSel = 0001)
   c. The branch target adder output (when ssdSel = 0010)
   d. The PC input (when ssdSel = 0011)
   e. The data read from the register file based on RS1 (when ssdSel = 0100)
   f. The data read from the register file based on RS2 (when ssdSel = 0101)

g. The data provided as an input to the register file (when ssdSel = 0110)
h. The immediate generator output (when ssdSel = 0111)
i. The shift left 1 output (when ssdSel = 1000)
j. The output of the ALU 2nd source multiplexer (when ssdSel = 1001)
k. The output of the ALU (when ssdSel = 1010)
l. The memory output (when ssdSel = 1011)

Note that all the above values are 32-bit values; however, we will only be displaying the least significant 13-bits of each which is enough for this lab's purposes.

As a result, the RISC-V module should have the following list of ports:

1. Clock Input (to be connected to a switch)
2. Reset Input (to be connected to a switch)
3. ledSel 2-bit Input (to be connected to 2 switches)
4. ssdSel 4-bit Input (to be connected to 4 switches)
5. SSD Clock Input (to be connected to pin E3)
6. leds 8-bit output (to be connected to 8 LEDs)
7. ssd 13-bit output (to be connected to the 4-digits SSD)

Please note that in order to use the SSD, you will need to **define a top-level module instantiating both the RISC-V module and the SSD driver module of lab 1** and connect them together. You will also need to connect them to the UART receiver module (provided on lab 2) to supply the 8 inputs remotely and display these inputs values on the other 8 leds of the FPGA board.

Please note that the RISC-V module clock is different from the SSD driver clock. The SSD clock is connected to built-in 100 MHz clock available on PIN E3. The RISC-V clock will be provided manually using a switch to allow us to step through the execution of a program cycle by cycle and inspect all the signals.

Figure 4 shows the names of all the FPGA package pins needed to implement the above. Please note that to be able to connect the RISC-V clock to the one of the switches, you will need to add the following line in the constraints file:

```
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets rclk]
```

where rclk is the RISC-V clock (as opposed to the SSD driver clock).

Do not forget to reset your circuit using the switch to initialize the PC and all the registers in the register file to 0.

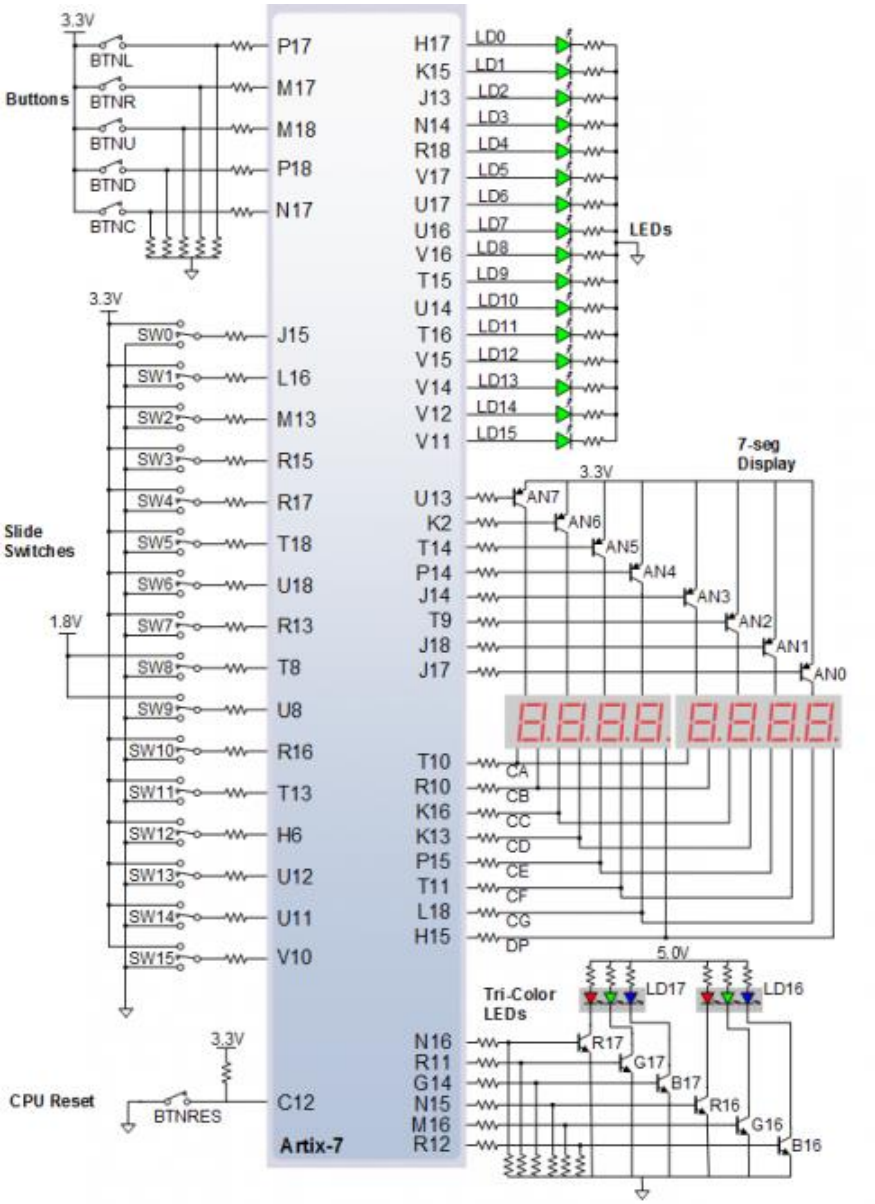

Figure 4 General purpose I/O devices on the Nexys A7 (reproduced from the Nexys A7 reference manual)

Experiment 5: Testing Using Your Own Program

Write a different RISC-V program using only the instructions supported and test it on the FPGA kit. The program must feature a simple loop.

To avoid having to manually assemble the program, you can use RARS (a RISC-V Assembler and Runtime Simulator available on GitHub) or use the Oak online assembler/simulator (which was developed by AUC students) (available on https://donn.github.io/Oak.js/). Note that these simulators are not perfect and may produce errors in results.

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 5 Report [10 pts]

1. [0 pts] Your name and student ID.
2. [2 pts] A technical summary of experiments 1 and 2 as conducted in the lab (steps, results, components, code functionality, etc.).
3. [3 pts] All Verilog code written (including testbenches) for experiments 1 and 2.
4. [3 pts] Snapshot of simulation output corresponding to each submitted testbench with a **brief interpretation** of the snapshot.
5. [2 pts] Verilog code written for experiment 3 (even if incomplete, please provide your progress in it).

Lab 6 Report [10 pts]

1. [0 pts] Your name and student ID.
2. [2 pts] All Verilog code of your final RISC-V implementation (experiment 4) in addition to the corresponding FPGA constraint file.
3. [1 pts] The test program you wrote for experiment 5 (both in assembly and binary) with a brief description of what it does.
4. [4 pts] The values seen on the 8 status LEDs and on the 4-digits SSD each clock cycle (for all relevant ledSel and ssdSel values of each instruction) for both experiments 4 and 5. You can provide that as one large table or in an Excel sheet if you prefer.
6. [3 pts] At least one photo of the Nexys A7 board for each different instruction supported. The photo should be showing relevant information on the LEDs and the 4-digits SSD. A clear description of the instruction being executed and the value of the selection lines (ledSel and ssdSel) when the photo was taken should be included.

Congratulations ☺ you have your first Processor Working ☺