EC413 Lab 6: 3-Stage 32-bit Pipelined Datapath

Austin M. Welch

October 26, 2015

Abstract

The main objective of this lab was to gain a better understanding of the timing that goes on in a pipelined datapath by simulating MIPS instruction codes and tracing them through three stages of registers.

1 Introduction

Figure 1 shows the structure of the entire 32-bit pipelined datapath. Each of these six components is contained in my top module.

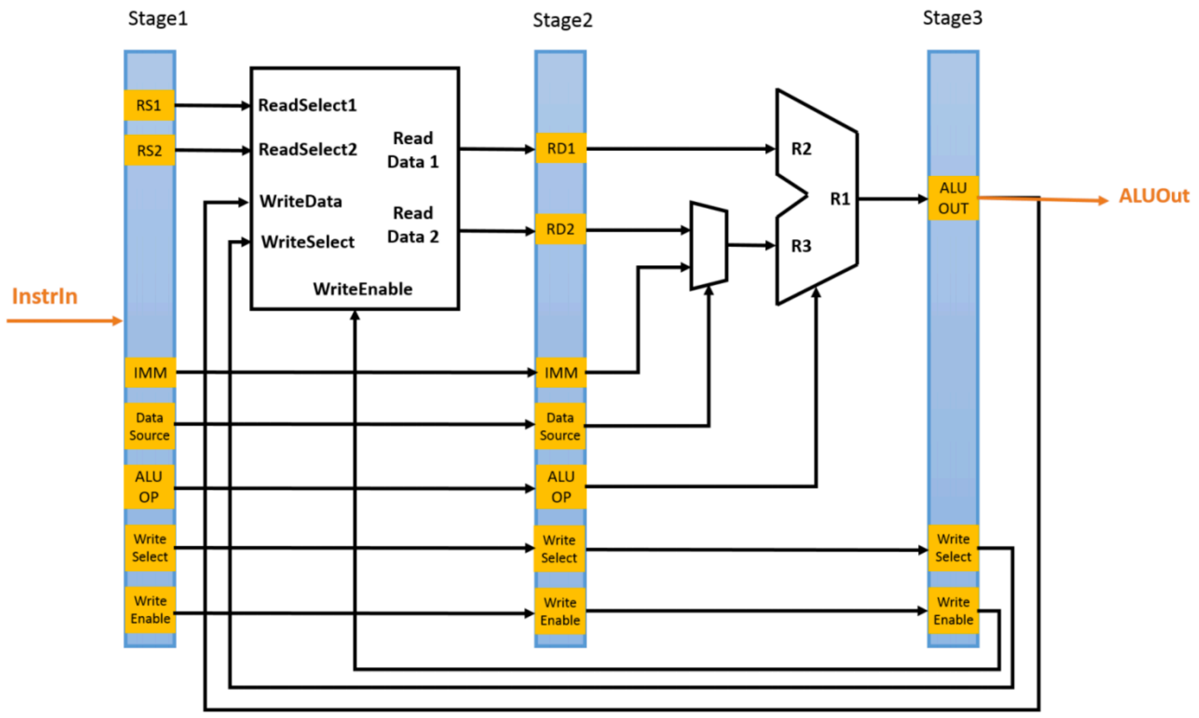
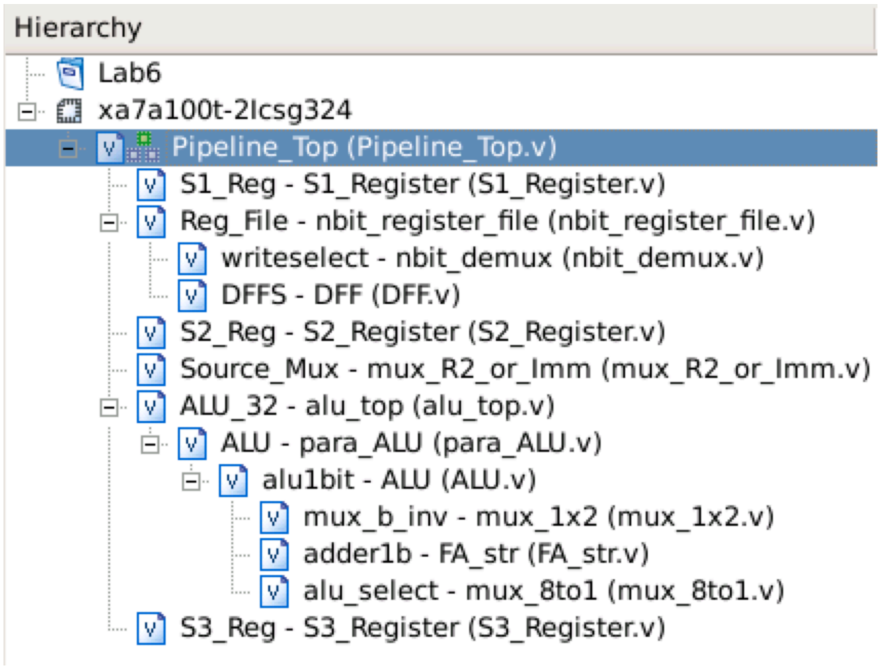


Figure 1: **Top structure for 3-stage pipeline.** I have six modules within my top module, corresponding to each of the pieces in this diagram: S1_Reg, Reg_File, S2_Reg, Source_Mux, Para_ALU and S3_Reg.

The hierarchy of my final design is shown in figure 2.



```
Hierarchy
  Lab6
  xa7a100t-2lcsg324
    Pipeline_Top (Pipeline_Top.v)
      S1_Reg - S1_Register (S1_Register.v)
      Reg_File - nbit_register_file (nbit_register_file.v)
        writeselect - nbit_demux (nbit_demux.v)
        DFFS - DFF (DFF.v)
      S2_Reg - S2_Register (S2_Register.v)
      Source_Mux - mux_R2_or_Imm (mux_R2_or_Imm.v)
      ALU_32 - alu_top (alu_top.v)
        ALU - para_ALU (para_ALU.v)
          alu1bit - ALU (ALU.v)
            mux_b_inv - mux_1x2 (mux_1x2.v)
            adder1b - FA_str (FA_str.v)
            alu_select - mux_8to1 (mux_8to1.v)
      S3_Reg - S3_Register (S3_Register.v)
```

Figure 2: **Top module hierarchy.** The six modules inside my top module, along with all of their submodules.

1.1 Pipeline Overview

A 32-bit instruction very similar to MIPS instruction encoding is taken as an input to the stage one register module along with the clock (clk) and reset (rst) inputs. The instruction, labeled InstrIn, is parsed inside stage one as follows: InstrIn[31:30] can be ignored. If InstrIn[29] == 1 it is I-type, if InstrIn[29] == 0 it is R-type. InstrIn[28:26] is the 3-bit ALU op-code, InstrIn[25:21] is the destination register, InstrIn[20:16] is the first source register, InstrIn[15:11] is the second source register, and InstrIn[15:0] is the immediate field. Each of the three register stages will take new values at positive clock edges only.

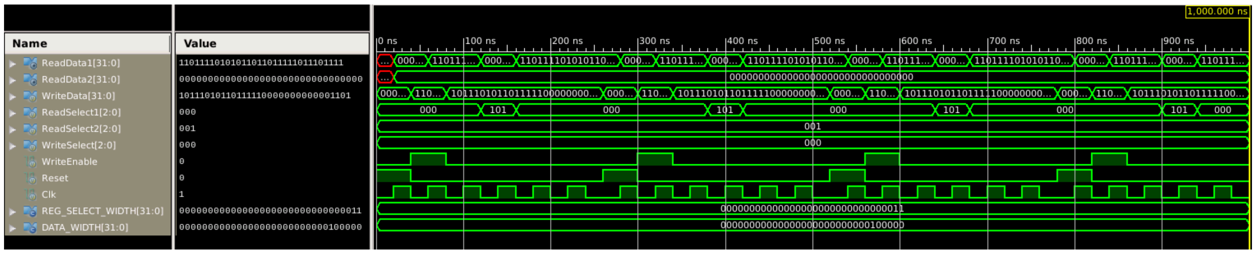
The values for the first and second source registers, labeled ReadSelect1 and ReadSelect2, respectively, are passed to the register file module called nbit_register_file, while all the other values including Immediate, DataSource, AluOp, WriteSelect and WriteEnable, are passed to the stage two register module.

Along with the outputs from stage one, stage two also takes as inputs the data read from the two source register locations, which is labeled ReadData1 and ReadData2. At the next positive clock edge all of the inputs of stage two are passed as outputs. ReadData1 is passed directly to the first input of a parameterized ALU sized to 32-bit, while ReadData2, Immediate and DataSource are all passed to a multiplexer. This multiplexer looks at the value of InstrIn[29] and decides whether to pass ReadData2 or Immediate to the second input of the ALU depending on whether it is R-type (0) or I-type (1). If the instruction is I-type, 16'd0 must be concatenated onto the front of the 16-bit immediate so that the ALU will get a 32-bit value. The ALU is combinational logic so it does not have to wait for a clock cycle. After it calculates an output based on its inputs and its opcode it passes the output to the stage three module. The remaining values, WriteSelect and WriteEnable, are passed directly to the stage three module.

The stage three module is the simplest of the three, and doesn't do anything but receive values and pass them along at the next positive clock edge. AluOut is the final output, but it also gets passed to WriteData of the register file. WriteSelect and WriteEnable are also passed to the register file. This nbit_register_file module generates a 32 by 32-bit array of D flip flops to create the 32 registers. Finally, on the next positive clock edge WriteData is written to the register selected by WriteSelect.

2 Register File

The first part of this lab to be tested was the nbit_register_file module. This was easy to do because a test bench for this module was already provided. The resulting simulation waveform for the register file using the provided test bench is shown in figure 3. The only thing required to do here was to look at the sixteen instructions listed in the test bench and follow the simulation waveform to make sure everything matched up correctly.

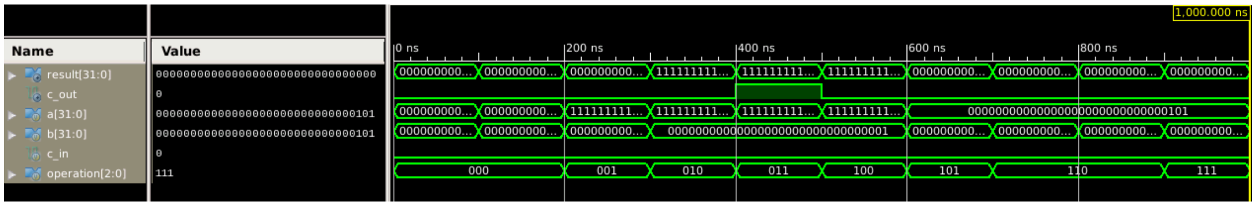


Name	Value
ReadData1[31:0]	11011110101011011011111011101111
ReadData2[31:0]	00000000000000000000000000000000
WriteData[31:0]	10111010110111110000000000001101
ReadSelect1[2:0]	000
ReadSelect2[2:0]	001
WriteSelect[2:0]	000
WriteEnable	0
Reset	0
Clk	1
REG_SELECT_WIDTH[31:0]	00000000000000000000000000000011
DATA_WIDTH[31:0]	00000000000000000000000000100000

Figure 3: **nbit_register_file_test waveform.** This image is also included separately.

3 Bit-sliced ALU

The next section that required resting was the ALU. This was also simple because we built the entire ALU in the previous lab. I copied every module from the previous lab into this project minus the registers and verification logic. After adding behavioral logic for the 6th opcode, SLT, I created a test bench with one to two test cases for each operation and followed the simulation waveform to confirm that I ended up with the correct results. The waveform for the ALU is shown in figure 4.



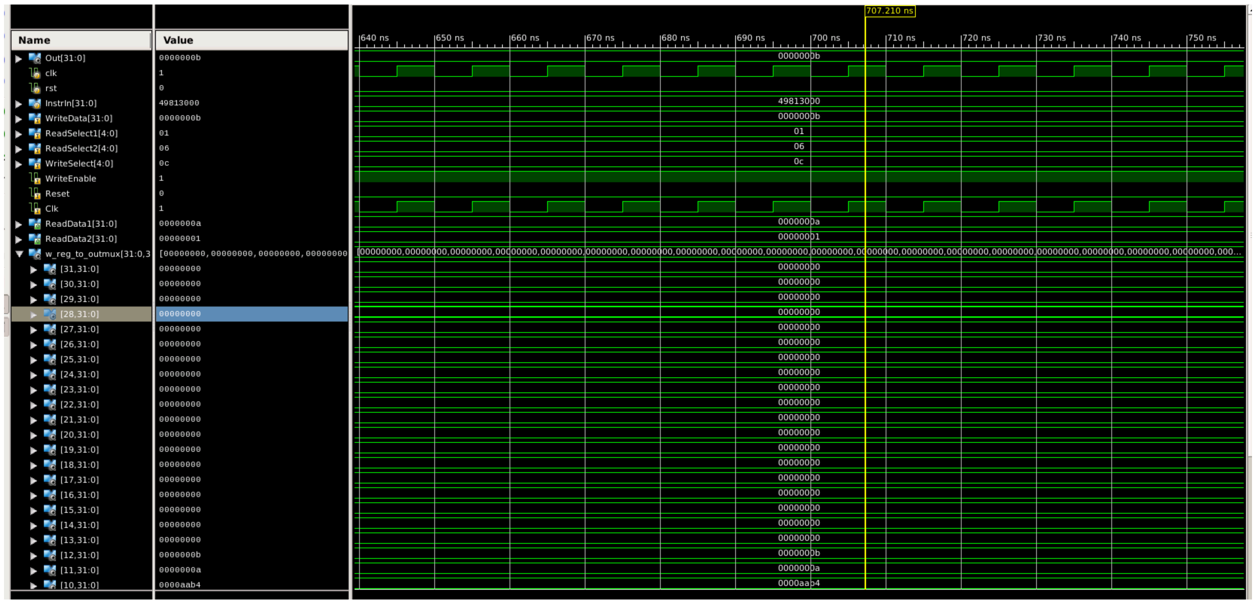
Name	Value
result[31:0]	00000000000000000000000000000000
c_out	0
a[31:0]	00000000000000000000000000000101
b[31:0]	00000000000000000000000000000101
c_in	0
operation[2:0]	111

Figure 4: **alu_top waveform.** This image is also included separately.

4 Datapath

I tested my final datapath design in two ways. First, I used the four sample instructions included in the lab procedure for my test bench. I checked the simulated waveform against the provided timing spreadsheet to make sure all of my values matched correctly.

Next, I changed my instruction test bench to the provided list of thirteen instructions. These are the instructions that will be included in the test bench I submit. First off, I checked that the final output was correct. This is shown in figure 5. After confirming the final output, I started at the beginning and followed each instruction while also checking the individual registers R0-R31 by examining the values for the 32-bit wire w_reg_to_outmux. All of the registers took the values I expected them to take. The final values for R0-R31 can be seen in figure 6.

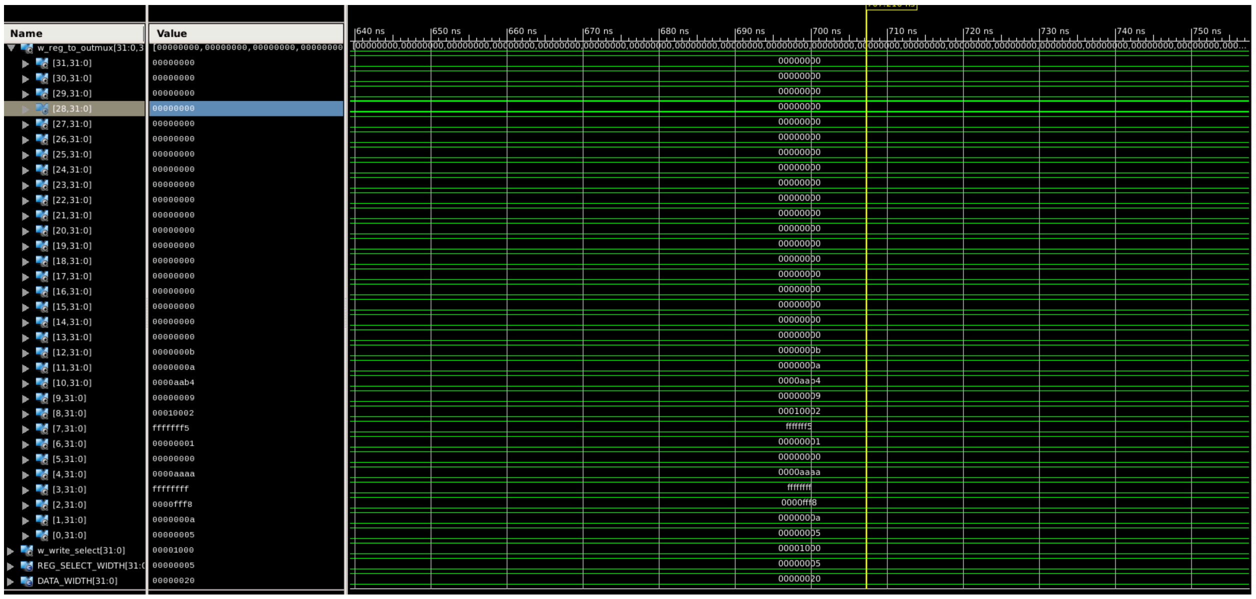


Name	Value
Out[31:0]	0000000b
clk	1
rst	0
InstrIn[31:0]	49813000
WriteData[31:0]	0000000b
ReadSelect1[4:0]	01
ReadSelect2[4:0]	06
WriteSelect[4:0]	0c
WriteEnable	1
Reset	0
Clk	1
ReadData1[31:0]	0000000a
ReadData2[31:0]	00000001
w_reg_to_outmux[31:0,3	[00000000,00000000,00000000,00000000
[31,31:0]	00000000
[30,31:0]	00000000
[29,31:0]	00000000
[28,31:0]	00000000
[27,31:0]	00000000
[26,31:0]	00000000
[25,31:0]	00000000
[24,31:0]	00000000
[23,31:0]	00000000
[22,31:0]	00000000
[21,31:0]	00000000
[20,31:0]	00000000
[19,31:0]	00000000
[18,31:0]	00000000
[17,31:0]	00000000
[16,31:0]	00000000
[15,31:0]	00000000
[14,31:0]	00000000
[13,31:0]	00000000
[12,31:0]	0000000b
[11,31:0]	0000000a
[10,31:0]	0000aab4

Figure 5: **Final output waveform.** This image is mainly used to show that the final output after all instructions have passed is 0x0B as expected.

5 Included Files

Source Files
Pipeline_Top.v
S1_Register.v
nbit_reg_file.v
nbit_demux.v
DFF.v
S2_Register.v



```
Name                        Value
w_reg_to_outmux[31:0,3      [00000000,00000000,00000000,00000000
  [31,31:0]                 00000000
  [30,31:0]                 00000000
  [29,31:0]                 00000000
  [28,31:0]                 00000000
  [27,31:0]                 00000000
  [26,31:0]                 00000000
  [25,31:0]                 00000000
  [24,31:0]                 00000000
  [23,31:0]                 00000000
  [22,31:0]                 00000000
  [21,31:0]                 00000000
  [20,31:0]                 00000000
  [19,31:0]                 00000000
  [18,31:0]                 00000000
  [17,31:0]                 00000000
  [16,31:0]                 00000000
  [15,31:0]                 00000000
  [14,31:0]                 00000000
  [13,31:0]                 00000000
  [12,31:0]                 0000000b
  [11,31:0]                 0000000a
  [10,31:0]                 0000aab4
  [9,31:0]                  00000009
  [8,31:0]                  00010002
  [7,31:0]                  fffffff5
  [6,31:0]                  00000001
  [5,31:0]                  00000000
  [4,31:0]                  0000aaaa
  [3,31:0]                  ffffffff
  [2,31:0]                  0000fff8
  [1,31:0]                  0000000a
  [0,31:0]                  00000005
w_write_select[31:0]        00001000
REG_SELECT_WIDTH[31:(       00000005
DATA_WIDTH[31:0]            00000020
```

Figure 6: **Final Register File Data.** The data within the registers R0-R31 can be seen by looking at the values of the outgoing wire array labeled w_reg_to_outmux.

mux_R2_or_Imm.v
alu_top.v
para_alu.v
ALU.v
mux_1x2.v
FA_str.v
mux_8to1.v
S3_Register.v

Test Benches
Pipeline_Top_tb.v
nbit_register_file_test.v
alu_top_tb.v

Images
pipeline_pic.pdf
Hierarchy.pdf
Reg_file_waveform.pdf
ALU_waveform.pdf
final_output_waveform.pdf
Final_reg_data_waveform.pdf