Ande Chen and Sunwoo Park
EC413
October 27th, 2021

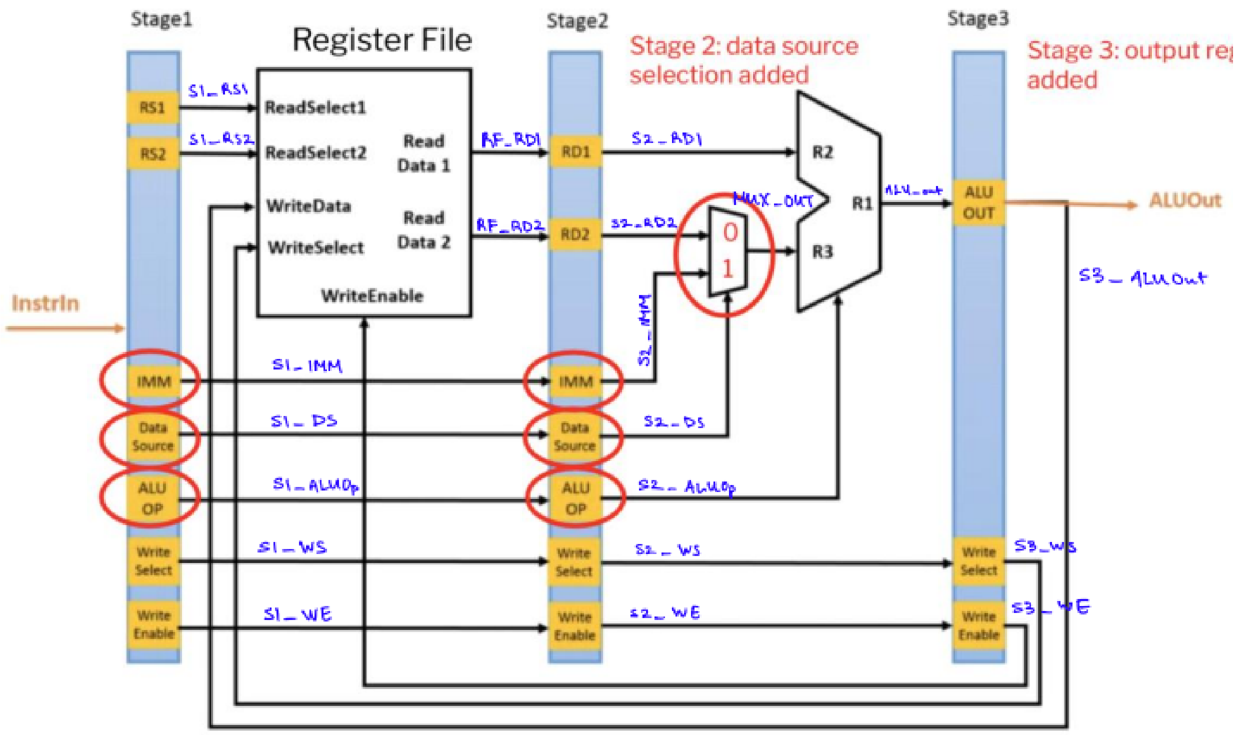
Lab 5 Report

Introduction

The purpose of this lab was to build a 4-stage 32-bit pipelined datapath given the Verilog files for the D-flip-flops used in the registers, an n-bit register, an n-bit demux used in the register file, and an n-bit register file. Our job was to create Verilog modules for the registers at each stage of the datapath and create a top module in which we instantiated each component to piece together the pipeline. We were also responsible for verifying the functionality of our pipelined datapath by passing 32-bit instruction sequences and comparing the register values at each stage to make sure our paper test cases matched the register values in our pipelined CPU.

Design

The diagram below is the design schematic of our pipelined datapath. Each wire is labeled in blue with the name we gave them in our top-level datapath module.



Stage 1 Register (Stage1Reg.v)

The 'Stage1Reg' module takes the 32-bit overall input instruction (InstrIn) as its input. The module consists of two 5-bit read select registers, a 16-bit immediate register, a 1-bit data source register, a 3-bit ALU Op-code register, a 5-bit write select register, and a 1-bit write enable register. The read select registers determine which registers will have their data read by the register file. ‘Read Select 1’ is defined in bits 20 to 16 and ‘Read Select 2’ is defined in 15 to 11 of the instruction. The immediate value (bits 15 to 0 of the instruction) is stored in a register in case the instruction is I format and uses it as an operand. The data source (bit 29 of the instruction) specifies if the instruction is I format (1) or R format (0). The ALU Op-code (bits 28 to 26 of the instruction) indicates the operation to perform on the data of the two read select registers. The write select register holds the register that will be overwritten with the output of the instruction’s ALU operation. This is defined by bits 25 to 21 of the instruction. The write enable register allows the contents of the write select register to be updated if its value is 1. The two read select values and the other five register values are passed to the register file and the Stage 2 register, respectively, on the next positive edge of the clock.

Register File (nbit_register_file.v)

The ‘RegFile’ module is a register file that takes the two 5-bit read select outputs of the ‘Stage1Reg’ module, a 32-bit write data, a 5-bit write select, and a 1-bit write enable from the ‘Stage3Reg’ module as inputs. The write select input specifies the register that should be updated with the contents of the ‘ALU OUT’ register of Stage 3. The write data input is the 32-bit value to be written into the write select input as long as the write enable input is 1. The register file takes the two read select inputs and returns their corresponding 32-bit contents as outputs to ‘Read Data 1’ and ‘Read Data 2,’ respectively. For our instantiation of the register file, all 32-registers are initialized to 1 times the register number. For example, register 1 has value 10.

Stage 2 Register (Stage2Reg.v)

The ‘Stage2Reg’ module has registers that hold the two 32-bits of data to be read from the input registers from Stage 1 (and outputted from the register file), as well as some of the same register values in Stage 1: the 16-bit immediate value (if an immediate is to be utilized), the 1-bit data source, the 3-bit ALU Op-code, the 5-bit write select, and the 5-bit write enable. The Stage 2 registers hold their values for one cycle and in the next cycle, the current register values are sent to the next respective components and registers. To be specific, ‘Read Data 1’ is sent to the ALU as one of the two inputs; ‘Read Data 2’ and the 16-bit immediate value are sent to a 2:1 multiplexer as inputs, with the 1-bit data source being the select to the multiplexer. Then, the output of the multiplexer is wired to the second input of the ALU; the ALU Op-code is the select into the ALU (and the output of the ALU is sent to the ‘ALU OUT’ register in Stage 3); the write select value is directly sent to the write select register in Stage 3; and the write enable value is also directly sent to the write enable register in Stage 3.

Multiplexer and ALU (Ideal_ALU.v)

The multiplexer takes the 32-bit value of Read Data 2, the 16-bit immediate, and the 1-bit data source outputs of the ‘Stage2Reg’ module as inputs. It selects the correct second operand for the ALU based on the data source bit. If the data source bit is 0, then the instruction is R format, and the mux will output the value of 'Read Data 2.' If the data source bit is 1, then the instruction is I format, and the mux will output the immediate value.

The 'ALU' module takes the 32-bit value of 'Read Data 1' and the 3-bit ALU Op-code from the 'Stage2Reg' module and the output of the mux as inputs. It uses a mux to select the correct operation (MOV, NOT, ADD, SUB, OR, AND, or SLT) based on the 3-bit ALU Op-code input. The ALU outputs the 32-bit result of the operation to the Stage 3 Register.

Stage 3 Register (Stage3Reg.v)

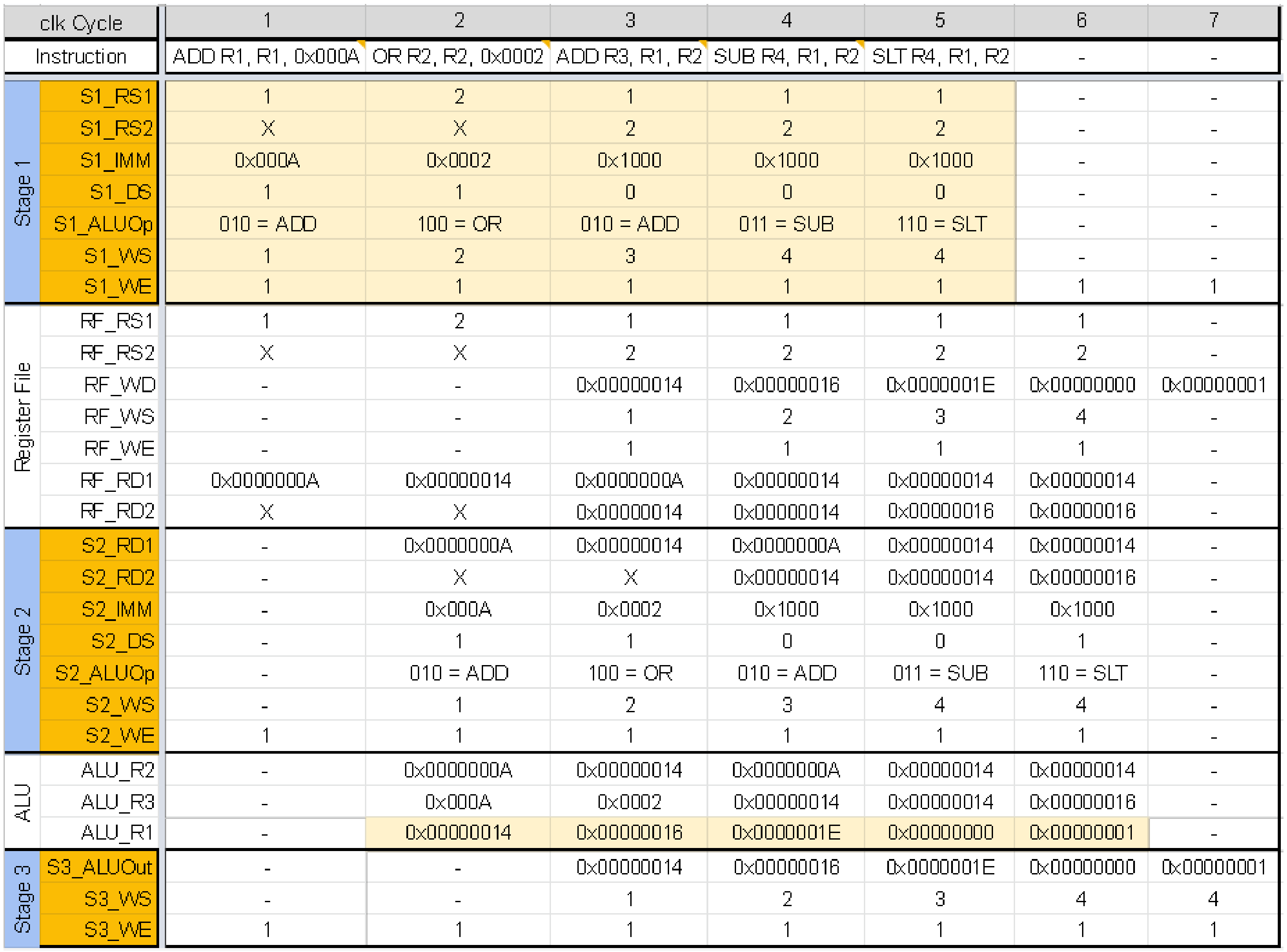
The 'Stage3Reg' module has registers that hold the 32-bit ALU output, the write select value from the Stage 2 write select register, and the write enable value from the Stage 2 write enable register. Then, at the next clock cycle, the ALU output is wired to the input of the write data register in the register file; the write select is wired to the write select register in the register file; and the write enable is wired to the write enable register select in the register file.

Testbench (Sample_Pipeline_test.v)

The 'Sample_Pipeline_test' module is the testbench we used to verify that our pipelined datapath was working properly. Within the module, we instantiated the top-level module (our pipelined datapath module) as the unit under test. Then, after performing a global reset, we set the initial 'WriteEnable' to high then passed a series of 32-bit instruction sequences to the datapath, with 10 time units (not clock cycles) between inputting each instruction. The instructions we tested are as follows:

```
ADD R1, R1, 0x0000000A
OR R2, R2, 0x00000002
ADD R3, R1, R2
SUB R4, R1, R2
SLT R4, R1, R2
```

We traced the test cases in the testbench through our pipelined datapath and organized each register and its value in a spreadsheet, similar to our task in the prelab. Below is a screenshot of our spreadsheet. (Note: - = don't-care value due to being out of scope, X = don't-care value)



	clk Cycle	1	2	3	4	5	6	7
	Instruction	ADD R1, R1, 0x000A	OR R2, R2, 0x0002	ADD R3, R1, R2	SUB R4, R1, R2	SLT R4, R1, R2	-	-
Stage 1	S1_RS1	1	2	1	1	1	-	-
	S1_RS2	X	X	2	2	2	-	-
	S1_IMM	0x000A	0x0002	0x1000	0x1000	0x1000	-	-
	S1_DS	1	1	0	0	0	-	-
	S1_ALUOp	010 = ADD	100 = OR	010 = ADD	011 = SUB	110 = SLT	-	-
	S1_WS	1	2	3	4	4	-	-
	S1_WE	1	1	1	1	1	1	1
Register File	RF_RS1	1	2	1	1	1	1	-
	RF_RS2	X	X	2	2	2	2	-
	RF_WD	-	-	0x00000014	0x00000016	0x0000001E	0x00000000	0x00000001
	RF_WS	-	-	1	2	3	4	-
	RF_WE	-	-	1	1	1	1	-
	RF_RD1	0x0000000A	0x00000014	0x0000000A	0x00000014	0x00000014	0x00000014	-
	RF_RD2	X	X	0x00000014	0x00000014	0x00000016	0x00000016	-
Stage 2	S2_RD1	-	0x0000000A	0x00000014	0x0000000A	0x00000014	0x00000014	-
	S2_RD2	-	X	X	0x00000014	0x00000014	0x00000016	-
	S2_IMM	-	0x000A	0x0002	0x1000	0x1000	0x1000	-
	S2_DS	-	1	1	0	0	1	-
	S2_ALUOp	-	010 = ADD	100 = OR	010 = ADD	011 = SUB	110 = SLT	-
	S2_WS	-	1	2	3	4	4	-
	S2_WE	1	1	1	1	1	1	-
ALU	ALU_R2	-	0x0000000A	0x00000014	0x0000000A	0x00000014	0x00000014	-
	ALU_R3	-	0x000A	0x0002	0x00000014	0x00000014	0x00000016	-
	ALU_R1	-	0x00000014	0x00000016	0x0000001E	0x00000000	0x00000001	-
Stage 3	S3_ALUOut	-	-	0x00000014	0x00000016	0x0000001E	0x00000000	0x00000001
	S3_WS	-	-	1	2	3	4	4
	S3_WE	1	1	1	1	1	1	1

As we expected (from our tracing of the series of instructions), the register values shown in the waveform of our simulated pipelined datapath match up with the values in our spreadsheet. For instance, at the first clock cycle, the Stage 1 registers hold their respective values from the parsed 32-bit instruction sequence of an ADD operation. In the same clock cycle, the register file registers read ‘ReadSelect1’ and ‘ReadSelect2’ (which is a don’t-care value since we were adding an immediate to a register) from the Stage 1 registers, and the register file’s ‘Read Data 1’ register assumes the value of the contents of R1 in memory, 0x0000000A, while the ‘Read Data 2’ register assumes a don’t-care value that is not used in the calculation. Then, at the second clock cycle, while the Stage 1 and register file registers assume the values corresponding to the new instruction sequence of an OR operation, the Stage 2 registers now hold the values passed from the Stage 1 and register file registers. Specifically, the Stage 2 ‘Read Data 1’ register holds the 32-bit R1 memory content; the ‘Read Data 2’ register holds a don’t-care value; the ‘IMM’ register holds the immediate 0x000A; the ‘Data Source’ register holds the bit 1; the ‘ALU Op’ register holds the 3-bit Op-code for ADD which is 010; the ‘Write Select’ register holds the bit 1 because we are writing to register R1; and the ‘Write Enable’ register holds the bit 1 because we are writing to memory. For the next clock cycles we observe, this process continues with the respective instruction sequence. Finally, at the seventh clock cycle, we see that the Stage 3 register values are filled with the parsed previous instruction for a SLT operation: the ‘ALU OUT’ register holds the output of the ALU following the computation of the SLT operation between the contents of R1 and R2; the ‘Write Select’ register assumes the value of the ‘Write Select’ register from Stage 2 of the previous cycle (clock cycle 6); and the ‘Write Enable’ register also assumes the value of the ‘Write Enable’ register from Stage 2 of the previous cycle (clock cycle 6). All in all, our pipelined datapath works properly and matches the manual tracing of each instruction sequence through the registers.

Note: Please refer to either the waveform screenshot (Lab5Waveform.jpg) or the waveform file (Pipeline_test_behav.wcfg) for the waveform of our simulated pipelined datapath design. When comparing with the spreadsheet, keep in mind that clock cycle 1 occurs at 135 ns and clock cycle 7 occurs at 195 ns.

Design Hierarchy

```
Sample_Pipeline_test.v
        datapath.v
                Stage1Reg.v
                nbit_register_file.v
                Stage2Reg.v
                Ideal_ALU.v
                Stage3Reg.v
```