CPEG324 Lab 3: Calculator Components and Design

By: James David and Ethan Conway

Abstract: The objective of this lab is to develop and implement several key components that are needed in a single-cycle calculator. This includes an 8-bit register model, an ALU model, and a final single-cycle datapath+controller circuit for a calculator. We are also to write a truth table for a certain signal in the 8-bit register, and a truth table for a translation logic from instruction to "op" values for the ALU.

Division of Labor: The division of labor for this project was split mainly between component design, and testbench design. James was responsible for most of the component design and the coding for the components, and Ethan was responsible for most of the testbench design and the coding for the testbenches. The final calculator datapath design was worked on by both James and Ethan, as well as the final lab report.

Detailed Strategy/Results:

Problem 1:

- Explanation:
    - To create the behavioral model for the 8 bit register, we took the register from Lab 2 Problem 2a and changed the input and output length to 8 bits. For the Register File, we have a demux, 2 muxes, and 4 registers. The demux sends the WE signal to the register specified by WS. The two muxes output the register contents to rd1 and rd2 that were selected by rs1 and rs2. A register is written to only when it is selected by ws and WE is 1. Below is our truth table for WE in our ISA.
- "WE" Signal Truth Table:

Opcode	**WE**
Load Word, opcode: 00	1
Add, opcode: 01	1
Negate, opcode 10	1
Print, opcode: 10, switch bits: 01	0
Jump, opcode: 11, switch bits: 00	0
Branch Greater than, opcode: 11, switch bits: 01	0

Problem 2:

- Explanation:
    - For this ALU, we used the 4-bit adder and subtractor from Lab 2 and extended the input and output to 8 bits, extended the operation code to 2 bits, and added more functions. The ALU takes in a 2-bit long opcode that decides between adding, subtracting, doing nothing, and setting on equal. The ALU takes in 2 8-bit long inputs (A and B), and does the operation on the 2 inputs decided by the opcode. The 2 outputs are A op B, and equal. The A op B output will be A+B when op is 00, A-B when op is 01, and A when op is 10 or 11. The equal output will only be 1 when A=B and op is 11. To translate from our ISA opcodes to the ALU opcodes, we designed a logic component and truth table below.
- Explanation of Translation Logic:
    - Our translation logic for going from our ISA opcodes to the ALU opcode is as follows:
        - The logic takes in the first 4 bits from an instruction in our ISA format
        - If the first two bits are 00, then the operation the ALU is not needed, so the opcode for nothing is sent to the ALU (10)
        - If the first two bits are 01, then the operation the ALU needs to do is addition, so the opcode sent to the ALU is 00
        - If the first two bits are 10, and the next two bits are:
            - 00: then the ALU needs to subtract, so the opcode sent to the ALU is 01
            - 01: then the ALU does nothing, so the opcode sent to the ALU is 10
        - If the first two bits are 11, and the next two bits are:
            - 00: then the ALU does nothing, so the opcode sent to the ALU is 10
            - 01: then the ALU would need to do an equal operation, so the opcode sent to the ALU would be 11
- Translation Logic Truth Table:

Our ISA Opcode	**ALU Opcode**
Load Word, opcode: 00	10 (nothing)
Add, opcode: 01	00 (add)
Negate, opcode 10, switch bits: 00	01 (subtract)
Print, opcode: 10, switch bits: 01	10 (nothing)
Jump, opcode: 11, switch bits: 00	10 (nothing)
Branch Greater than, opcode: 11, switch bits: 01	10 (nothing)

Problem 3:

- Explanation:
    - Our single-cycle datapath uses many different components: Instruction Fetch, Controller, Register File, ALU, Muxes and Demuxes, and a Logic component. The instruction is fetched from PC, then is outputted to many different places. First, the Opcode section is sent to the Controller, which determines the logic to choose which operations the Register File, ALU, and the Print Module do. The Register File has an input of the last 6 digits of the instruction for rs1, rs2, and ws. The Output of the ALU is connected to wd. If the operation is lw, the middle 4 bits of the instruction are sent to INT, but otherwise those 4 bits are sent to rs1 and rs2. The last 2 bits of the instruction are sent to ws. The register is only written to in specific instructions determined by the Controller Logic, which the truth table for which is shown below, along with the truth table for the ALU opcode logic. Print will print the contents of rd1 only if the opcode for print is selected, otherwise it will print nothing. Jump and Branch Greater than are handled by the BGE Logic and the Mux on the bottom. The Mux will output 1 if the jump instruction is selected, output the result of the BGE Logic if the bge instruction is selected, and output 0 otherwise.

- Truth Tables

Truth Tables for Controller Logic

WE Truth Table

ISA Opcode	WE
00 (lw)	1
01 (add)	1
1000 (negate)	1
1001 (print)	0
1010 (sub)	1
1100 (jump)	0
1101 (bge)	0

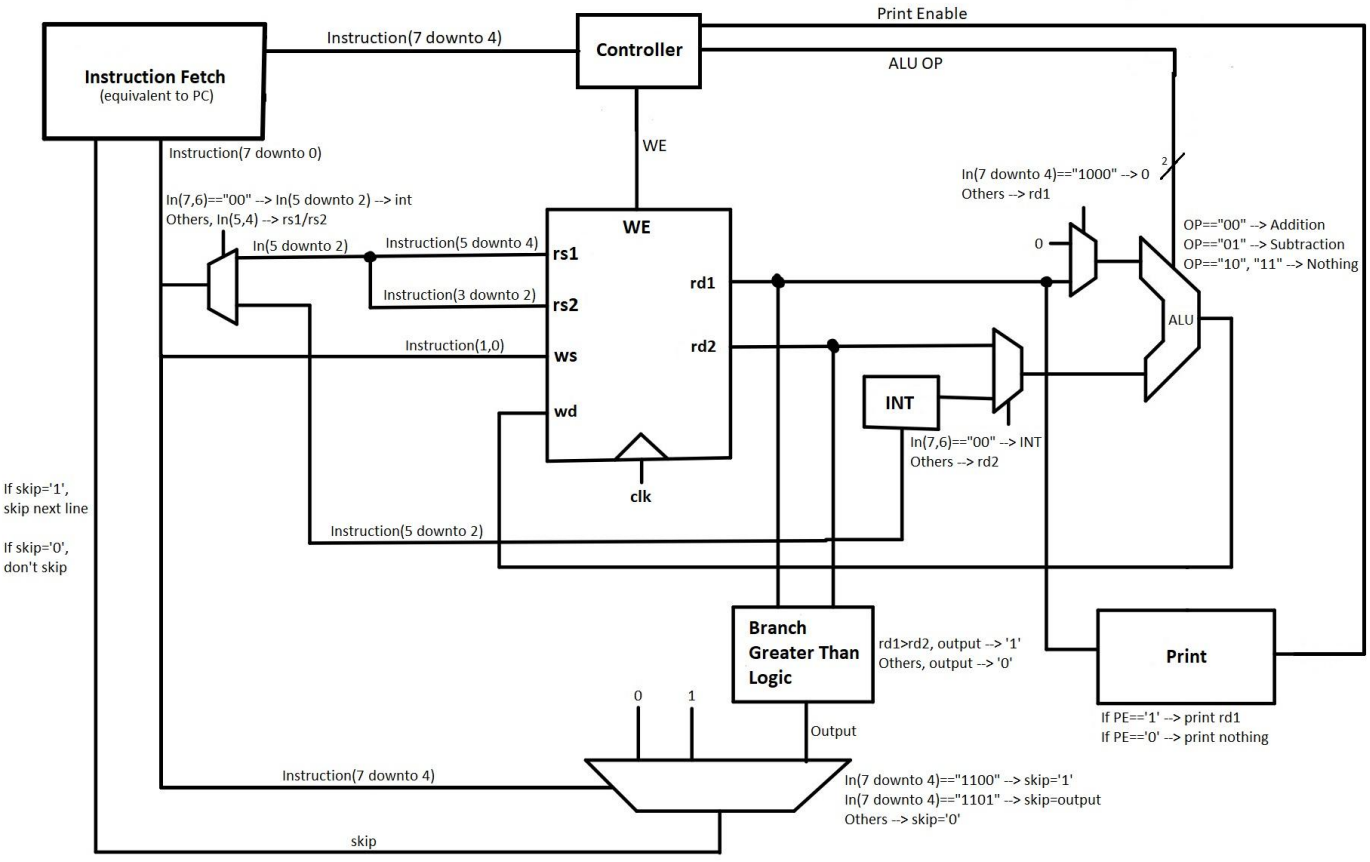
ALU Operation Truth Table

ISA Opcode	ALU Op
00 (lw)	00 (addition)
01 (add)	00 (addition)
1000 (negate)	01 (subtraction)
1001 (print)	10 (nothing)
1010 (sub)	01 (subtraction)
1100 (jump)	10 (nothing)
1101 (bge)	10 (nothing)

Print Enable Truth Table

ISA Opcode	Print Enable
1001 (print)	1
Other	0

- Circuit Design:



Full size image of datapath (to read the small text)

Conclusion: Overall, I believe that we completed this project to the best of our ability. I believe that we successfully designed and developed a working 8-bit register and ALU, and also successfully designed a circuit datapath for the calculator. I also believe that we successfully developed working testbenches for each of our components that are able to thoroughly test each of our components and each of their capabilities. Given more time and resources for this project, I think it would have been feasible to implement more calculator functions that would be on a modern day calculator, and perhaps also make the calculator a multi-cycle datapath. The area of the project that likely gave us the most difficulty was the actual design of the calculator datapath. It was tough trying to decide on a final design that would be efficient, and one that both members agreed was the best. Though, in the end, I believe that we ended on a good design that works.

Appendix I: Notebooks

- James David: 12 Total Hours Spent
- Ethan Conway: 10 Hours Spent

Appendix II: VHDL Files

Problem 1 files:

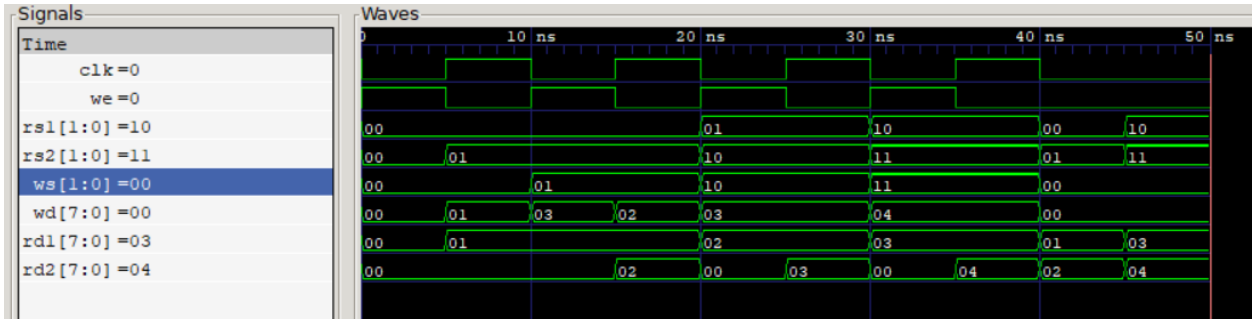
- reg8.vhdl
- reg8n4.vhdl
- reg8n4.vcd
- reg8n4_tb.vhdl
- mux_4to1.vhdl
- demux_4to1.vhdl

Problem 2 files:

- ALU8bit.vhdl
- ALU8bit_tb.vhdl
- full_adder.vhdl
- ALU8bit.vcd

Appendix III: Testing

Problem 1 Waveform:



Problem 2 Waveform: