Lab 2 Report

CSE 141L - Hadi Esmaeilzadeh

0. Team Members

Akshay Kamal
Aabjeet Grewal
Rachel Wong

1. Introduction

This report contains a brief summary of the **$NAME** instruction set architecture, a list of ALU operations, and test outputs to demonstrate the functionality of the ALU and program counter/instruction memory.

2. ISA Summary

The $NAME instruction uses a modified load-store architecture, with the use of a special $ops register which allows the programmer to specify a range of 8-bit register values and addresses, etc. within 9 bits.

Instructions:

R-Type: add, logical shift left, logical shift right, and, or, xor, set less than
Example: add

Description	Adds two registers and stores the result in the specified register.
Operation	$d = $s + $t; advance_pc(4);
Syntax	add $d
$ops usage	ssss tttt
Encoding	0 000d ddd0
Example	ldh $r1 ldl $r2 add $r8 *Adds the value of $r1 to $r2 and stores the result in $r8.*

I-Type: move, load, store, load high, load low
Example: mov

Description	Stores the sign-extended immediate value in a register.
Operation	$s = imm; advance_pc(4);
Syntax	mov $i
$ops usage	ssss XXXX
Encoding	0_111i_iiii
Example	ldh $r2 mov 4 *Sets the value of $r2 to 4. The bottom four bits of the $ops register are unused for this instruction.*

B-Type: jump, *added the halt instruction
Example: hlt

Description	Halts the CPU from performing any more instructions until the processor receives a software interrupt.
Operation	stalls pc
Syntax	hlt
$ops usage	x xxxx xxxx
Encoding	1 111x xxxx
Example	hlt *Signifies the end of the current program. Stops the processor from performing more work until another interrupt occurs.*

3. Operations tested

All instructions in the $NAME instruction set are dependent on the ALU in some form. The following operations cover the ALU's involvement with all of the relevant instruction types.

ALU behavior

1. Addition (add, ldb, str)
2. Shift left (lsl)
3. Shift right (lsr)
4. And (and)
5. Or (lor)
6. Xor (xor)
7. Less than (slt)
8. Move (mov)

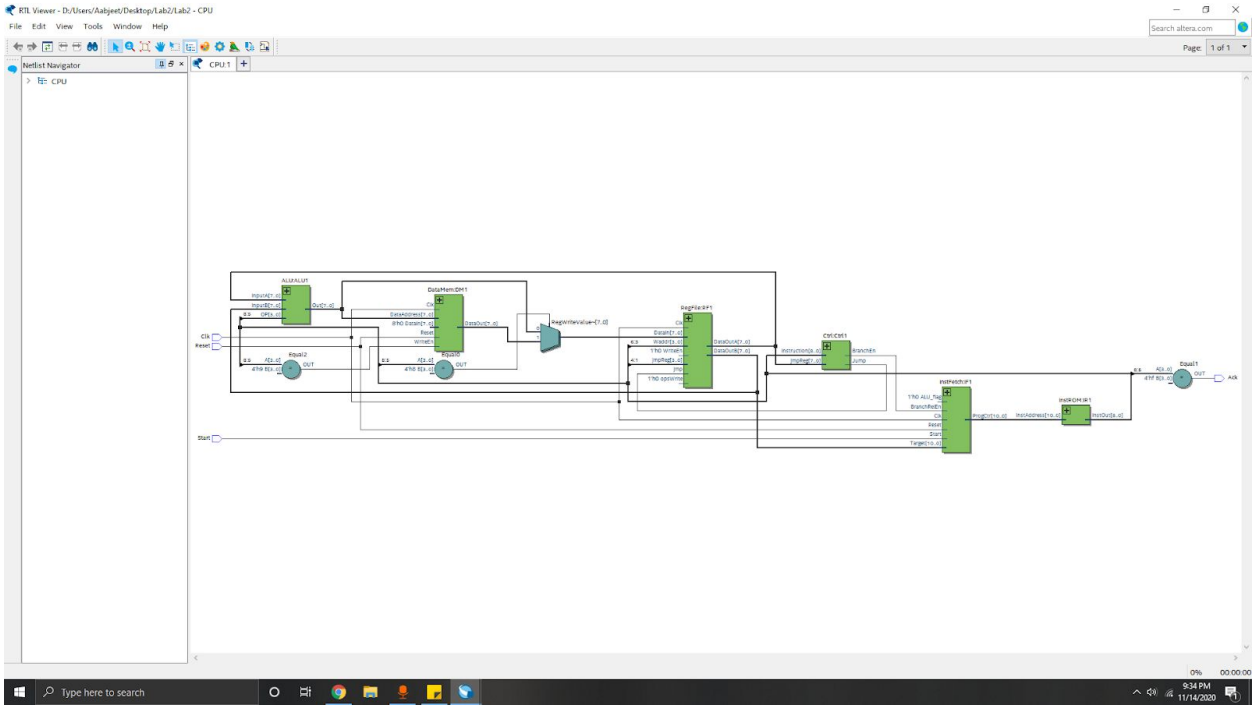
The instruction fetch module was tested on the relevant behaviors defined in the program logic.

Instruction fetch behavior

1. Reset the program counter to 0
2. Hold the program counter at the current position (start)
3. Add an offset to the program counter (jump)
4. Increment the program counter (default)
5.

4. Verilog Model

Top-Level Verilog Model: Complete Processor



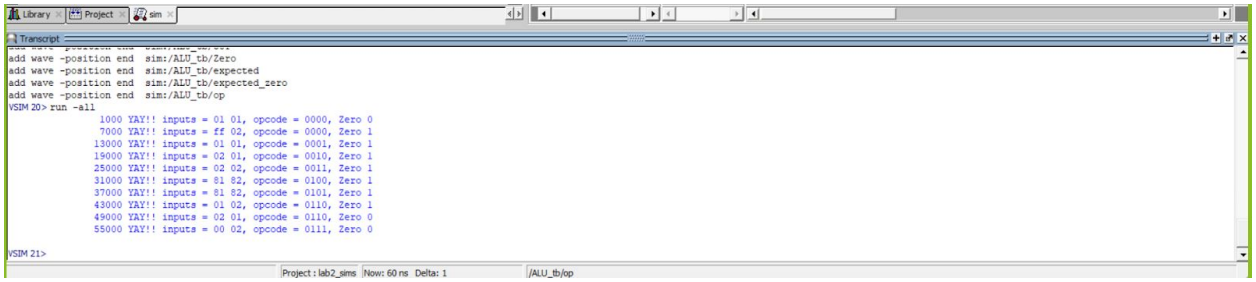
5. Test Cases

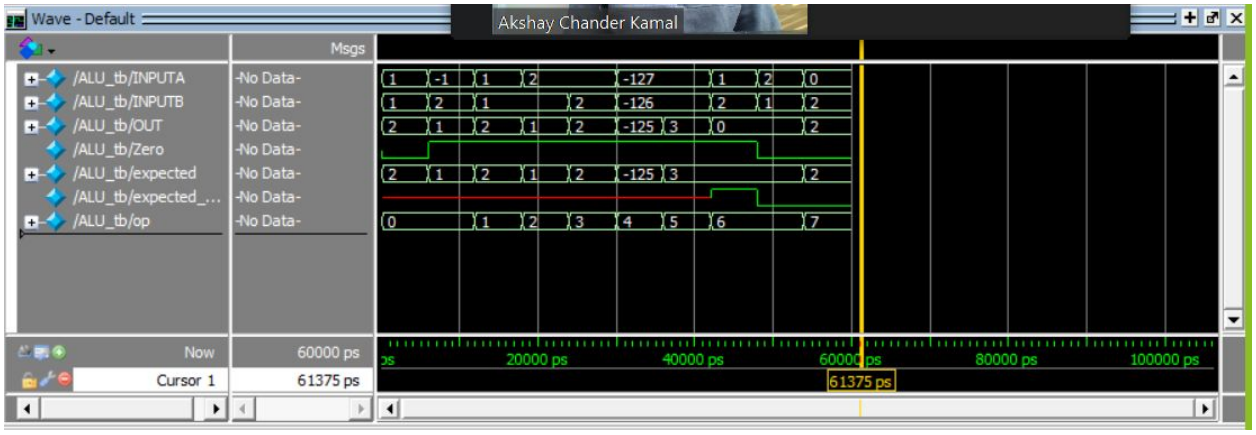
We tested our ALU by extending the provided ALU testbench, and created a similar testbench for the instruction fetch module.

We modified the ALU testbench to use our ISA-specific opcodes, and also verified that the zero/overflow/carry bit was functioning correctly when applicable.

We delineated the instruction fetch test cases based on three scenarios: resetting the PC, branching the PC (adding/subtracting an offset), and the default increment. The specifics of the test cases can be found in the appropriately labeled testbench files for each module.

ModelSim output: ALU testbench





ModelSim output: InstFetch testbench