MILESTONE 2 DOCUMENT

Date Created: 2/4/2022

Date Last Updated: 2/9/2022

Group Members: Daisuke Chon, Angelica Consengco, Matthew Larkins

PIDs: A15388691, A14113566, A16052530

Component 1: Changelog

1. Specified that addresses are absolute rather than relative under Component 3, Part iv. of Milestone 1: Machine Specification - Control Flow
2. Added assembly language instruction to machine code translation example under Component 4 of Milestone 1: Programmer's Model

Component 2: ALU Operations

Our ALU should be able to support the following operations:

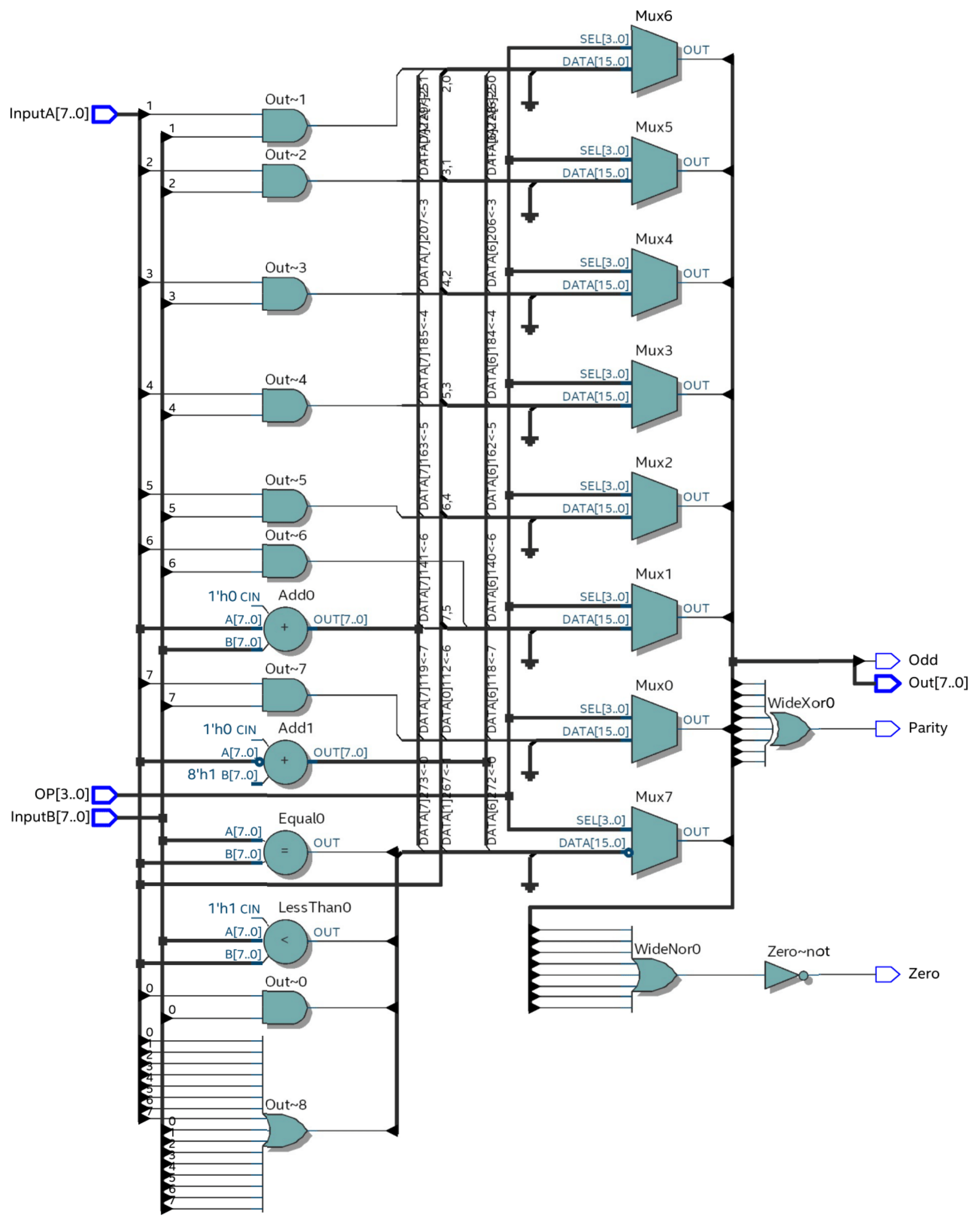
- logical left shift
- logical right shift
- bitwise and
- bitwise or
- Greater than or equal comparison
- Equals comparison
- 2's Comp Negation
- Addition
- Not Equals comparison

Our register file can read and write into registers. The basic implementation suits our design so we left it as is.

Component 3: Verilog Models

ALU Diagram

Date: February 08, 2022 Project: cse141L_ALU



Page 1 of 1 Revision: ALU

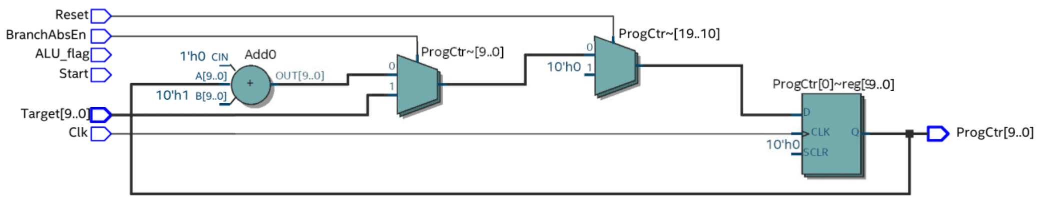
ALU Code

```
// Create Date:    2018.10.15
// Module Name:    ALU
// Project Name:   CSE141L
```

```
//
// Revision 2021.07.27
// Additional Comments:
//   combinational (unclocked) ALU
import definitions::*;                    // includes package "definitions"
module ALU #(parameter W=8, Ops=4)(
  input        [W-1:0]   InputA,      // data inputs
                         InputB,
  input        [Ops-1:0] OP,              // ALU opcode, part of microcode
  output logic [W-1:0]   Out,             // data output
  output logic           Zero,        // output = zero flag  !(Out)
                         Parity,      // outparity flag  ^(Out)
                         Odd                // output odd flag (Out[0])
// you may provide additional status flags, if desired
  );

  op_mne op_mnemonic;                     // type enum: used for convenient
waveform viewing

  always_comb begin
    Out = 0;                              // No Op = default
    case (OP)
      ADD : Out = InputA + InputB;        // add
      LSH : Out = {InputA[6:0], 1'b0};    // shift left, fill in with zeroes
        RSH : Out = {1'b0, InputA[7:1]};    // shift right
      AND : Out = InputA & InputB;        // bitwise AND
      OR  : Out = InputA || InputB;       // bitwise OR
      NEG : Out = ~InputA + 1;
      GEQ : Out = (InputA >= InputB);         // Greater than or Equal to
      EQ  : Out = (InputA == InputB);        // Equals to
      NEQ : Out = (InputA != InputB);         // Not Equals to
    endcase
  end

  assign Zero   = !Out;                   // reduction NOR
  assign Parity = ^Out;                   // reduction XOR
  assign Odd    = Out[0];                 // odd/even -- just the value of the LSB

  always_comb
    op_mnemonic = op_mne'(OP);            // displays operation name in waveform
viewer

endmodule
```

Program Counter Diagram:

Date: February 09, 2022 Project: cse141L_ProgCtr



Page 1 of 1 Revision: ProgCtr

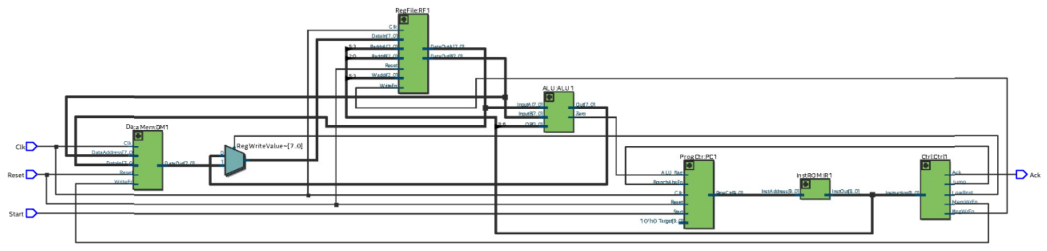
Program Counter Code:

```
// Design Name:    basic_proc
// Module Name:    InstFetch
// Project Name:   CSE141L
// Description:    instruction fetch (pgm ctr) for processor
//
// Revision:  2019.01.27
//
module ProgCtr #(parameter L=10) (
  input                 Reset,      // reset, init, etc. -- force PC to 0
                        Start,      // Signal to jump to next program; currently
unused
                        Clk,        // PC can change on pos. edges only
                        BranchAbsEn,  // jump to Target
                        ALU_flag,   // Sometimes you may require signals from other
modules, can pass around flags if needed
  input        [L-1:0] Target,     // jump ... "how high?"
  output logic [L-1:0] ProgCtr     // the program counter register itself
  );


  // program counter can clear to 0, increment, or jump
  always_ff @(posedge Clk)         // or just always; always_ff is a linting
construct
    if(Reset)
      ProgCtr <= 0;
    else if(BranchAbsEn)                    // unconditional absolute jump
      ProgCtr <= Target;               //   how would you make it conditional
and/or relative?
    else
      ProgCtr <= ProgCtr+'b1;          // default increment (no need for ARM/MIPS
+4 -- why?)
endmodule
```

Top Level Diagram

Date: February 09, 2022

Project: TopLevel



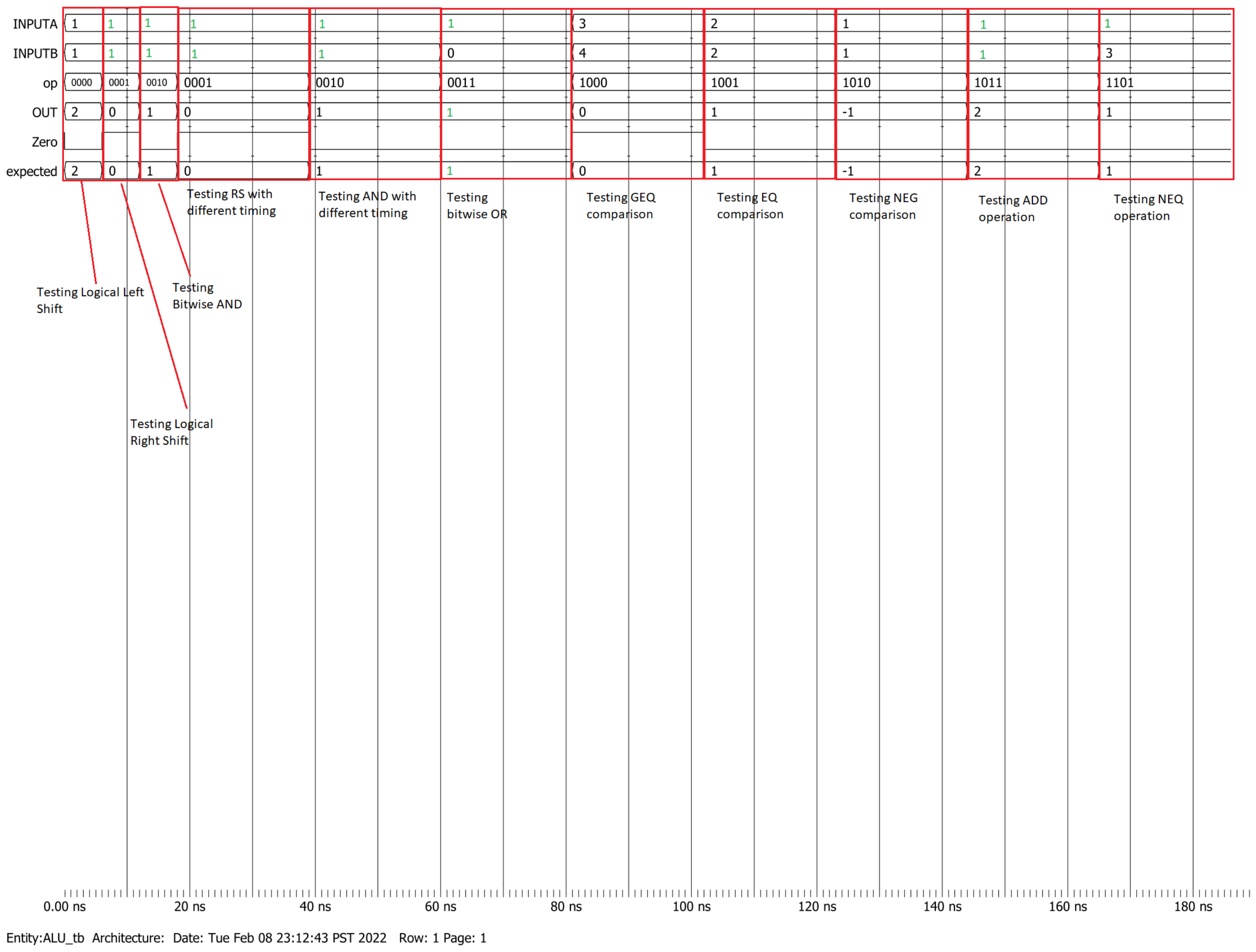
Page 1 of 1

Revision: TopLevel

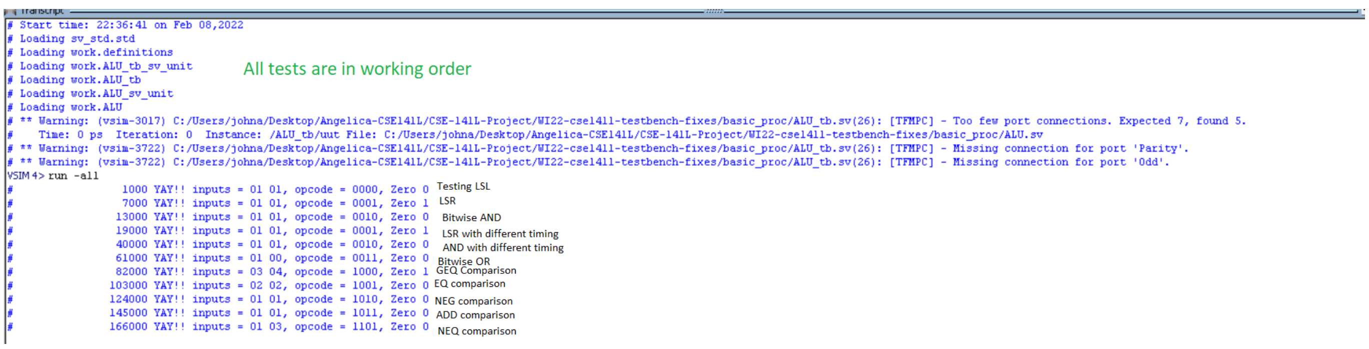
Component 4: Timing Diagrams

[comment]: #Provide well-annotated timing diagrams or transcript listings from your module level Questa/ModelSim runs. It should be clear that your program counter / instruction memory (fetch unit) and ALU works. If your presentation leaves doubt, we'll assume it doesn't.

ALU Timing Diagram



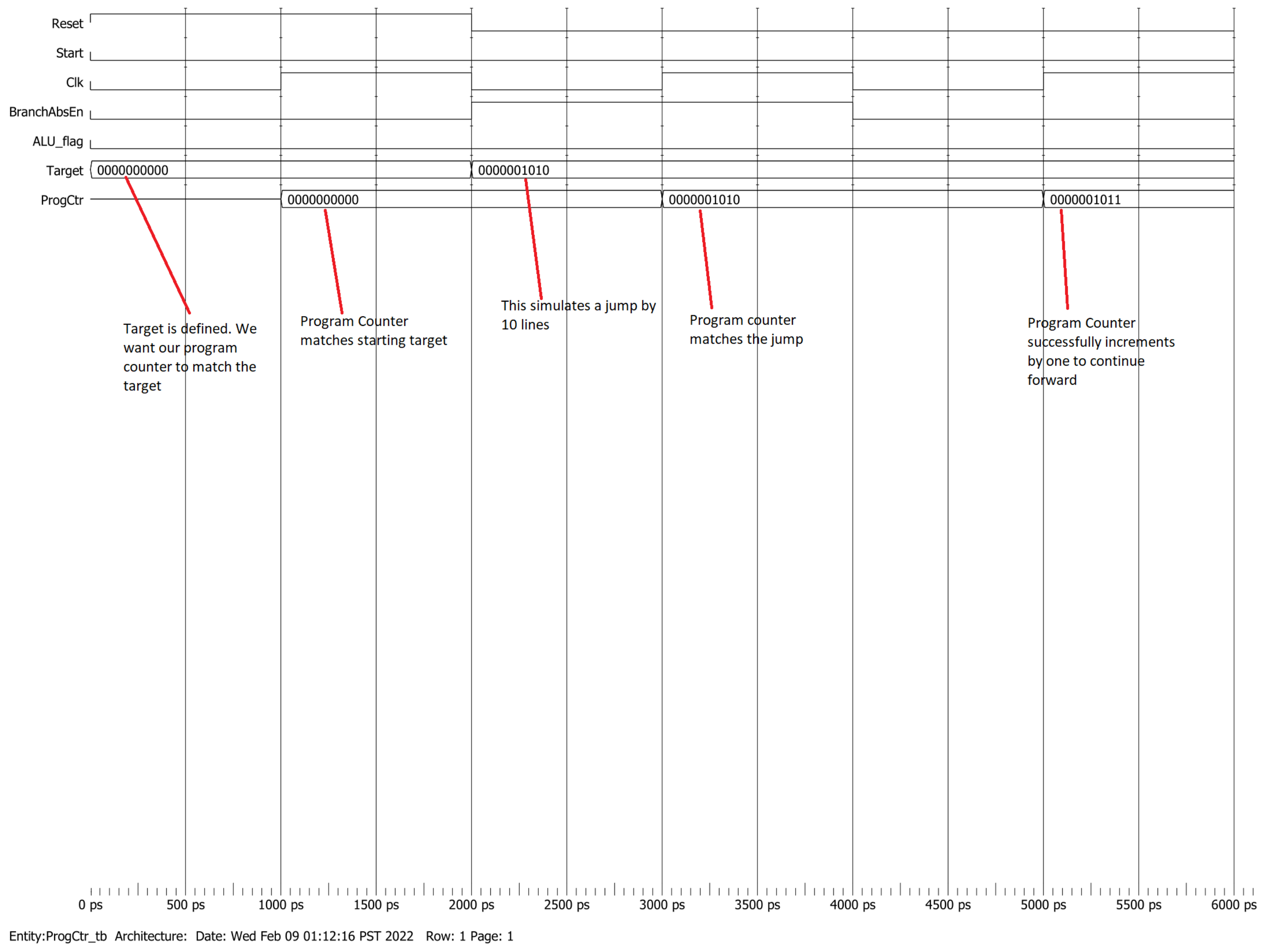
ALU Transcript



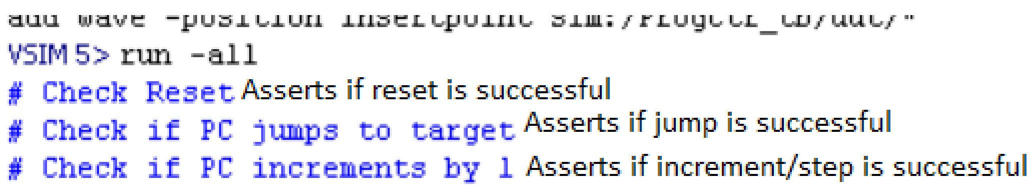
```
# Start time: 22:36:41 on Feb 08,2022
# Loading sv_std.std
# Loading work.definitions
# Loading work.ALU_tb_sv_unit
# Loading work.ALU_tb
# Loading work.ALU_sv_unit
# Loading work.ALU
# ** Warning: (vsim-3017) C:/Users/johna/Desktop/Angelica-CSE141L/CSE-141L-Project/WI22-cse141l-testbench-fixes/basic_proc/ALU_tb.sv(26): [TFMPC] - Too few port connections. Expected 7, found 5.
#    Time: 0 ps  Iteration: 0  Instance: /ALU_tb/uut File: C:/Users/johna/Desktop/Angelica-CSE141L/CSE-141L-Project/WI22-cse141l-testbench-fixes/basic_proc/ALU.sv
# ** Warning: (vsim-3722) C:/Users/johna/Desktop/Angelica-CSE141L/CSE-141L-Project/WI22-cse141l-testbench-fixes/basic_proc/ALU_tb.sv(26): [TFMPC] - Missing connection for port 'Parity'.
# ** Warning: (vsim-3722) C:/Users/johna/Desktop/Angelica-CSE141L/CSE-141L-Project/WI22-cse141l-testbench-fixes/basic_proc/ALU_tb.sv(26): [TFMPC] - Missing connection for port 'Odd'.
VSIM 4> run -all
#                  1000 YAY!! inputs = 01 01, opcode = 0000, Zero 0  Testing LSL
#                  7000 YAY!! inputs = 01 01, opcode = 0001, Zero 1  LSR
#                 13000 YAY!! inputs = 01 01, opcode = 0010, Zero 0  Bitwise AND
#                 19000 YAY!! inputs = 01 01, opcode = 0001, Zero 1  LSR with different timing
#                 40000 YAY!! inputs = 01 01, opcode = 0010, Zero 0  AND with different timing
#                 61000 YAY!! inputs = 01 00, opcode = 0011, Zero 0  Bitwise OR
#                 82000 YAY!! inputs = 03 04, opcode = 1000, Zero 1  GEQ Comparison
#                103000 YAY!! inputs = 02 02, opcode = 1001, Zero 0  EQ comparison
#                124000 YAY!! inputs = 01 01, opcode = 1010, Zero 0  NEG comparison
#                145000 YAY!! inputs = 01 01, opcode = 1011, Zero 0  ADD comparison
#                166000 YAY!! inputs = 01 03, opcode = 1101, Zero 0  NEQ comparison
```

All tests are in working order

Program Counter Timing Diagram



Program Counter Transcript



```
add wave -position insertpoint sim:/ProgCtr_tb/uut/*
VSIM 5> run -all
# Check Reset Asserts if reset is successful
# Check if PC jumps to target Asserts if jump is successful
# Check if PC increments by 1 Asserts if increment/step is successful
```

Component 5: Answering the Question

Our ALU will *indirectly* be used for non-arithmetic instructions such as load and store. While we do indeed need to make address pointer calculations, the programmer will be responsible for calculating these addresses via manual shift operations of 3 bit immediates into 8 bit values. As such, the complexity of the design is unaffected.