Cameron Kennedy and Nolan Baker
Lab4
CPEG324
May 23rd, 2022

CPEG 324 Lab Report 4

Abstract**:** The purpose of the lab was to create and implement the improved calculator circuit design that we created in lab 3. Doing this we created source code for the 8-bit calculator that had the add and sub functionality that we required for the main and optional parts of the lab.

Division of Labor: We split up the calculator work between the two of us, with Nolan focusing on the extra credit portion and Cameron focusing on the main portion of the lab.

Detailed Strategy: The strategy behind the lab was to start with each individual component of the lab to work on all the individual parts of it. With a focus on the ALU and the regFile as the main part in the beginning because they would be having the most code associated with them while also being the main focal point of the lab. From there we started to make all the side parts that would be used by the before mentioned parts like the adders, mux's, signExtended, and the other logic gates needed. From there we made the calculator as a whole to put everything together and start testing what we had done previously to make sure it worked. Doing this we created all of the testbenches, which allowed us to visualize which aspects of the code were not working. We then continued this process until the code was working how we wanted it to.

Our main idea behind the calculator design was to utilize opcodes which made sense. Since printing does not involve much besides knowing which register you are going to print, we decided that it could be noted through having the first 3 bits of the instruction as 0. To make things easier, register 1 is moved when printing is enabled so that the instructions bits can be changed consecutively. The opcodes for adding and subtracting were given to us.

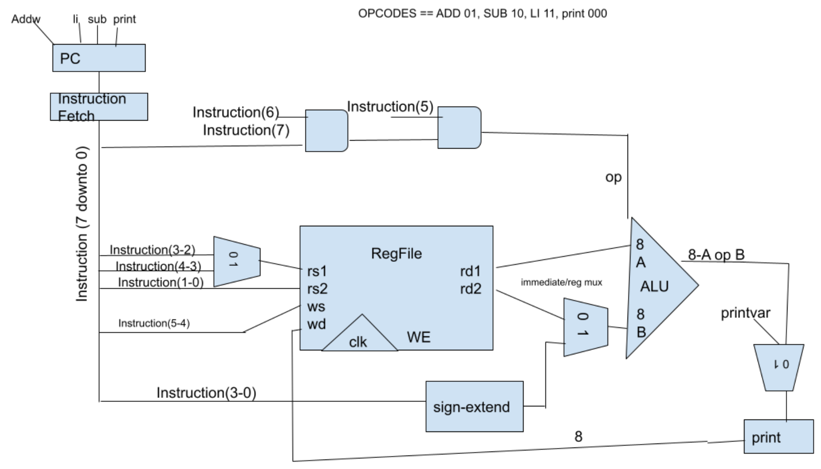
Once everything was finished we went back and changed the required parts to make the extra credit section work. For this we utilized Vivado and a Zybo Z7-10 board. Not much was required to be changed for this to work properly, as the switches worked perfectly as instruction bits and the leds properly resembled binary code. A 'display' output was needed, as Vivado requires at least one main output signal in order to function. This output was simply changed in the 'print' section of our code, where each clock signal & print instruction would change the LEDs.

Conclusion: After implementing the code and getting everything working there's most definitely room for improvement with our ode. Everything works as intended with it but since this is our first big VHDL component project there's definitely some tricks that we could implement to get a more efficient device working but we did the best that we could think of with the knowledge that we had. The registry file and ALU made the most sense, while implementing the control signals within the calculators was a challenge. Also, despite the minimal amount of change required to implement the code onto a board, it took many hours to properly function, as Vivado does not work the same as GHDL.

In the future and with more time, we could implement the skip & branch instructions to allow for an even more implemented calculator, along with fully developing our test bench to ensure it covers all edge cases.

Appendix II (VHDL files):

Schematic Design



Implementation Source Code

OPTIONAL PART

https://drive.google.com/file/d/1MtxVGxor-SKCi783wRsC3O94lcFWqFC-/view?usp=sharing

ISA Instructions

Add: 01 RD RS RT

Sub: 10 RD RS RT

LI : 11 RD RD Signed Immediate$_4$

Print: 000 RS RS 000

Register File

```
library ieee;
use ieee.std_logic_1164.all;

entity reg_file is
  port(
      clk, we         : in std_logic := '0';
      rs1, rs2, ws    : in std_logic_vector(1 downto 0);
      wd              : in std_logic_vector(7 downto 0);
      rd1, rd2        : out std_logic_vector(7 downto 0)
  );
end reg_file;

architecture structural of reg_file is

      signal reg0 : std_logic_vector(7 downto 0) := (others => '0'); --
index "00"
      signal reg1 : std_logic_vector(7 downto 0) := (others => '0'); --
index "01"
      signal reg2 : std_logic_vector(7 downto 0) := (others => '0'); --
index "10"
      signal reg3 : std_logic_vector(7 downto 0) := (others => '0'); --
index "11"

      begin
      process(clk) is
      begin

      -- rd1 definition
      register1: case rs1 is
      when "00" => rd1 <= reg0;
      when "01" => rd1 <= reg1;
      when "10" => rd1 <= reg2;
      when "11" => rd1 <= reg3;
      when others => rd1 <= (others => 'X');
      end case register1;
```

```
    -- rd2 definition
    register2: case rs2 is
    when "00" => rd2 <= reg0;
    when "01" => rd2 <= reg1;
    when "10" => rd2 <= reg2;
    when "11" => rd2 <= reg3;
    when others => rd2 <= (others => 'X');
    end case register2;

    -- set destination register
    if ((clk and WE) ='1') then
    set_wd: case ws is
         when "00" => reg0 <= wd;
         when "01" => reg1 <= wd;
         when "10" => reg2 <= wd;
         when "11" => reg3 <= wd;
         when others => null;
    end case set_wd;
    end if;

    end process;
end structural;
```

ALU

```
library ieee;
use ieee.std_logic_1164.all;
entity ALU is
     port(a, b : in std_logic_vector(7 downto 0);
     op : in std_logic; --0 for addition, 1 for substraction
     sum : out std_logic_vector(7 downto 0));
end entity ALU;

architecture structural of ALU is
component adder is
     port(a, b : in std_logic_vector(7 downto 0);
     sum : out std_logic_vector(7 downto 0));
end component adder;
```

```
signal b_sel, inv_b, neg_b : std_logic_vector(7 downto 0);
begin
adder0: adder port map(a, b_sel, sum); -- for addition/subtraction
adderneg: adder port map(inv_b, "00000001", neg_b); -- for
subtraction/addition with negatives

inv_b <= not(b);
with op select b_sel <= -- makes b negative if subtraction
     b when '0',
     neg_b when others;

end architecture structural;
--Adder--
library ieee;
use ieee.std_logic_1164.all;
entity adder is
     port(a, b : in std_logic_vector(7 downto 0);
     sum : out std_logic_vector(7 downto 0));
end entity adder;

architecture structural of adder is
component full_adder is
     port(a, b, c : in std_logic;
     sum, co : out std_logic);
end component full_adder;

signal carry : std_logic_vector(6 downto 0);
begin
     digit0: full_adder port map(a(0), b(0),'0', sum(0), carry(0));
     digit1: full_adder port map(a(1), b(1), carry(0), sum(1),
carry(1));
     digit2: full_adder port map(a(2), b(2), carry(1), sum(2),
carry(2));
     digit3: full_adder port map(a(3), b(3), carry(2), sum(3),
carry(3));
     digit4: full_adder port map(a(4), b(4), carry(3), sum(4),
carry(4));
```

```
     digit5: full_adder port map(a(5), b(5), carry(4), sum(5),
carry(5));
     digit6: full_adder port map(a(6), b(6), carry(5), sum(6),
carry(6));
     digit7: full_adder port map(a(7), b(7), carry(6), sum(7), open);
end architecture structural;

--Full Adder--
library ieee;
use ieee.std_logic_1164.all;
entity full_adder is
     port(a, b, c : in std_logic;
     sum, co : out std_logic);
end entity full_adder;

architecture data of full_adder is
begin
     sum <= a xor b xor c;
     co <= ((a and b) or (b and c) or (a and c));
end architecture data;
```

Calculator

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity calculator is
  port(
     instr : in std_logic_vector(7 downto 0);
     clk : in std_logic
  );
end entity calculator;

architecture structural of calculator is
  component reg_file is
     port(
     clk, we    : in std_logic := '0';
     rs1, rs2, ws  : in std_logic_vector(1 downto 0);
     wd         : in std_logic_vector(7 downto 0);
```

```
      rd1, rd2   : out std_logic_vector(7 downto 0)
      );
  end component reg_file;

  component alu is
      port(
           a, b : in std_logic_vector(7 downto 0);
           op : in std_logic; --0 = addition, 1 is subtraction.
           sum : out std_logic_vector(7 downto 0)
      );
  end component alu;

  component clk_RisingEdge is
      port(
      clk : in std_logic;
      clk_top : out std_logic
      );
  end component clk_RisingEdge;

signal clk_top, WE, print, wd_sel : std_logic;
signal rs1, rs2, ws : std_logic_vector(1 downto 0);
signal wd, rd1, rd2, sign_ext_imm, ALU_out: std_logic_vector(7 downto
0);

begin
  instReg : reg_file port map(clk_top, we, rs1, rs2, ws, wd, rd1,
rd2);
  iALU: ALU port map(rd1, rd2, instr(7), ALU_out);
  RE : clk_RisingEdge port map(clk, clk_top);

  -- assign control signals
  rs2 <= instr(1 downto 0);
  with print select rs1 <=
      instr(3 downto 2) when '0',
      instr(4 downto 3) when others;
  ws <= instr(5 downto 4);
  --mux for selecting ws
  wd_sel <= not(instr(7) and instr(6));
  with wd_sel select wd <=
     sign_ext_imm when '0',
     ALU_out when others;
  we <= instr(7) or instr(6);

  --sign extended immediate
  sign_ext_imm(3 downto 0) <= instr(3 downto 0);
  with instr(3) select sign_ext_imm(7 downto 4) <=
     "1111" when '1',
     "0000" when others;

  print <= not (instr(7) or instr(6) or instr(5));
  --print
  process(clk, print) is
     variable int_val : integer;
     begin
     if((clk'event and clk = '1') and (print = '1')) then
     report integer'image(to_integer(signed(rd1)));
     end if;
  end process;
end architecture structural;

-- rising edge of clk w/ slight delay
library ieee;
use ieee.std_logic_1164.all;

entity clk_RisingEdge is
  port(
     clk : in std_logic;
     clk_top : out std_logic
  );
end entity clk_RisingEdge;

architecture structural of clk_RisingEdge is
begin
  clk_top <= clk after 1 ps; --0 if skip 1, 0 if skip 2, 1 if no skip
```

```
end architecture structural;
```

Appendix II (testing):

Test Bench

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;

entity tb is
end entity tb;

architecture structural of tb is
component calculator is
  port(
     instr : in std_logic_vector(7 downto 0); --opcodes:
01->add, 10->sub, 11->li, 000->print
     clk : in std_logic
  );
end component calculator;

signal instr : std_logic_vector(7 downto 0);
signal clk : std_logic;

begin
     testbench : calculator port map(instr, clk);
     process
     begin
     clk <= '0';
     instr <= "11000011"; --li reg0 3
     wait for 1 ns;
     clk <= '1';
     wait for 1 ns;
     clk <= '0';
     instr <= "00000000"; --print reg0
     wait for 1 ns;
     clk <= '1';
```

```
wait for 1 ns;
clk <= '0';
instr <= "11100101"; --li reg2 5
wait for 1 ns;
clk <= '1';
wait for 1 ns;
clk <= '0';
instr <= "00010000"; --print reg2
wait for 1 ns;
clk <= '1';
wait for 1 ns;
clk <= '0';
instr <= "00001000"; --print reg1
wait for 1 ns;
clk <= '1';
wait for 1 ns;
clk <= '0';
instr <= "11111111"; --li reg3 -1 (twos comp)
wait for 1 ns;
clk <= '1';
wait for 1 ns;
clk <= '0';
instr <= "00011000"; --print reg3
wait for 1 ns;
clk <= '1';
wait for 1 ns;
clk <= '0';
instr <= "01010010"; --addw reg1, reg0, reg2 (3 + 5)
wait for 1 ns;
clk <= '1';
wait for 1 ns;
clk <= '0';
instr <= "00001000"; --print reg1
wait for 1 ns;
clk <= '1';
```

```
wait for 1 ns;
clk <= '0';
instr <= "01010011"; --addw reg1, reg0, reg3 (3 + -1)
wait for 1 ns;
clk <= '1';
wait for 1 ns;
clk <= '0';
instr <= "00001000"; --print reg1
wait for 1 ns;
clk <= '1';
wait for 1 ns;
clk <= '0';
instr <= "10011000"; --subw reg1, reg2, reg0 (5-3)
wait for 1 ns;
clk <= '1';
wait for 1 ns;
clk <= '0';
instr <= "00001000"; --print reg1
wait for 1 ns;
clk <= '1';
wait for 1 ns;
clk <= '0';
instr <= "10011111"; --subw reg1, reg3, reg3 (-1 -(-1) = 0)
wait for 1 ns;
clk <= '1';
wait for 1 ns;
clk <= '0';
instr <= "00001000"; --print reg1
wait for 1 ns;
clk <= '1';
wait for 1 ns;
clk <= '0';
instr <= "10001110"; --subw reg0, reg3, reg2
wait for 1 ns;
clk <= '1';
```

```
        wait for 1 ns;
        clk <= '0';
        instr <= "00000000"; --print reg1
        wait for 1 ns;
        clk <= '1';
        wait for 1 ns;
        clk <= '0';
        wait;
        end process;
end architecture structural;
```

<u>Test Bench Output:</u>

3 <= loaded

5 <= loaded

0 <= unitialized

-1 <= loaded w/ twos complement

8 <= 3 + 5

2 <= 3 + -1

2 <= 5-3

0 <= -1 - (-1)

-6<= -1 - 5