Weifan Lin
Csc343
04/30/2015

Simple Processor

Objectives:

The goal of this lab was to design a simple processor unit that consists of several parts: a multiplexer, an adder/subtractor unit, control unit and registers. This circuit was capable to perform basic assembly instructions in each clock cycle, such as loading specific data into a register or move immediate instruction, moving data from one register to another, addition of data in registers and subtraction of data in registers.

Specifications:

Run: an input is to activate the processor before allowing certain instruction to be executed.

Resetn: This is a reset input that refreshes the data for DIN input

DIN[15..0]: This is a 16-bit-input data that uses the first 9 bits are responsible for what instructions will be performed in the processor. Then follow the IIIXXXXYYY format, that III is the instruction, with XXX as the RX registers, and YYY as the RY register

Done: This is an output that let us know the instruction has been excecuted.

BusWires[15..0]: The 16-bit output that contains the data values loaded from DIN.

Reg0Out[7..0]: This is the output contains the first 8-bits of the data value in Register 0.

Reg1Out[7..0]: This is the output contains the first 8-bits of the data value in Register 1.

Functionality:

Register – The function of this system for the purpose of this lab is to store a data value, a 16-bit number, from the D input signal when the Clock signal is transitioning from LOW to HIGH. While the clock stays at a state (HIGH or LOW) or transitioning from HIGH to LOW, the register will hold its previous state.

Instruction Register – The function of this system is to store part of a data value, 9- bits out of a 16-bit number, from the D input signal when the Clock signal is transitioning from LOW to HIGH. Thus, its function is almost identical to that of the general register seen above; however, the fact that it does not store the entire input sets it apart. It is also important to note that the part of the input that is stored is the leading or most significant 9-bits.

Multiplexer – The function of this system is to switch between multiple data input signals based on a selector signal, Sel. Thereby, the output of the multiplexer, BUS, will always be the data input selected by a selector signal.

Adder/Subtracter – The function of this system is to preform decimal arithmetic, addition or subtraction, upon two input data values based on the sign signal. When the sign signal is LOW, this system will preform addition on the two input signals, Rx and Ry, where the output signal becomes the sum. On the other hand, when the sign signal is HIGH, this system will subtract Ry from Rx. It is important to note that like the multiplexer, the operation of this system is not dependent on the state of the clock sign.

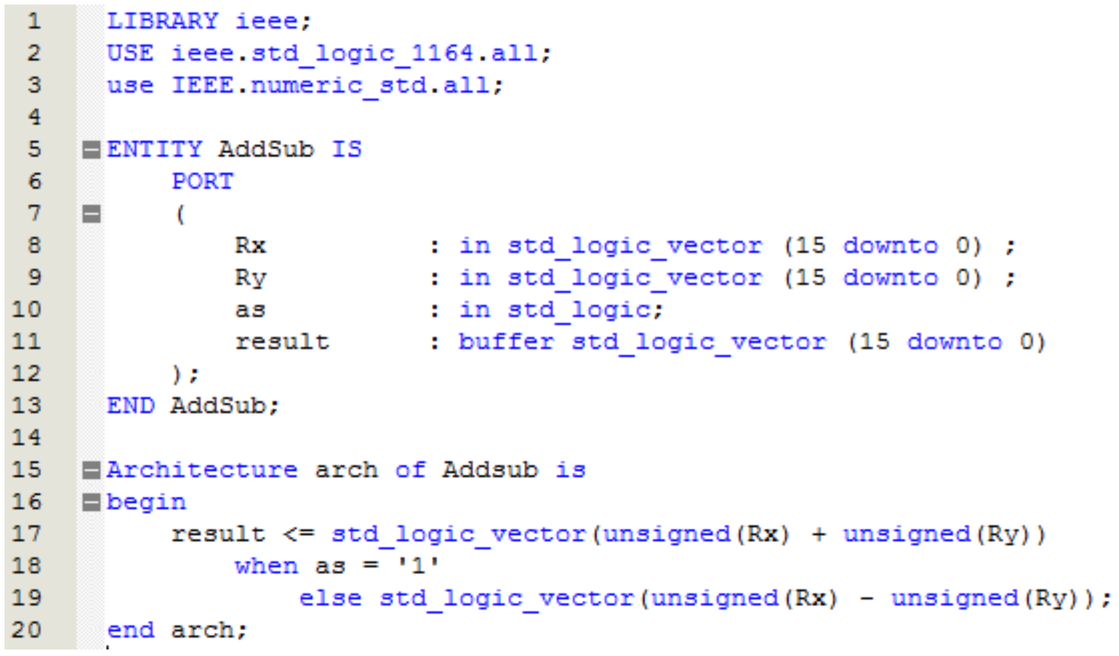
Control Unit – One of the functions of this system is to enable the appropriate registers for writing on the correct clock cycle based on the input, IRin. Another function of this system is to control which register or input is put on the bus, which it does through the

Mux output. To accomplish these tasks, this system uses a finite state machine with for states, decode, mv, add1, and add2.

CPU – The function of this system is to combine all the previous systems into a working processor that is able to interpret instructions and execute them. This system creates the mapping between components, for example, adder/subtractor requires one of the general 16-bit registers to work properly. Also, since the control unit takes instruction as a 9-bit number, Cmd, and the instruction comes in as a 16-bit number, the Instruction Register is needed between the Din input and the control unit.

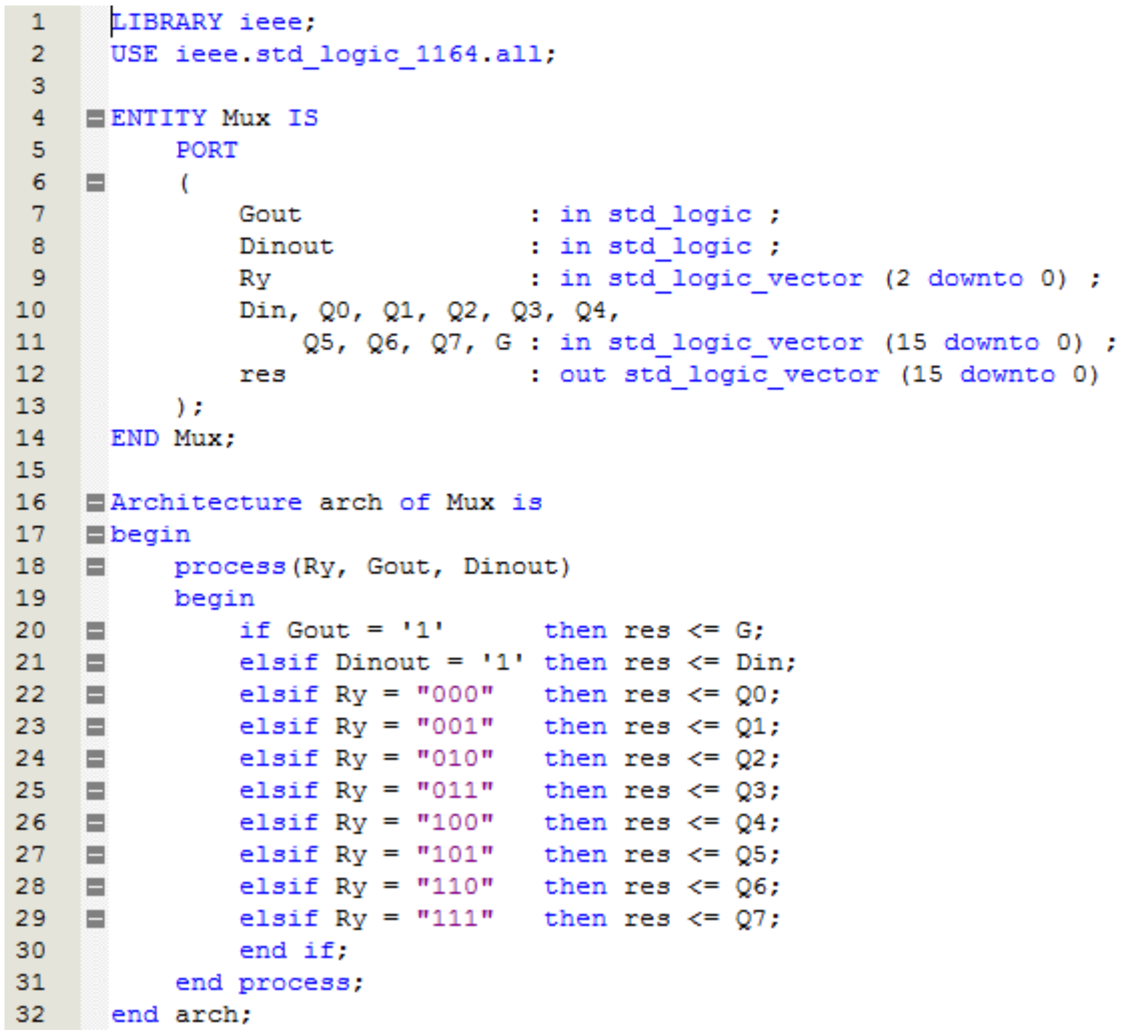
Design:

Adder/Subtractor



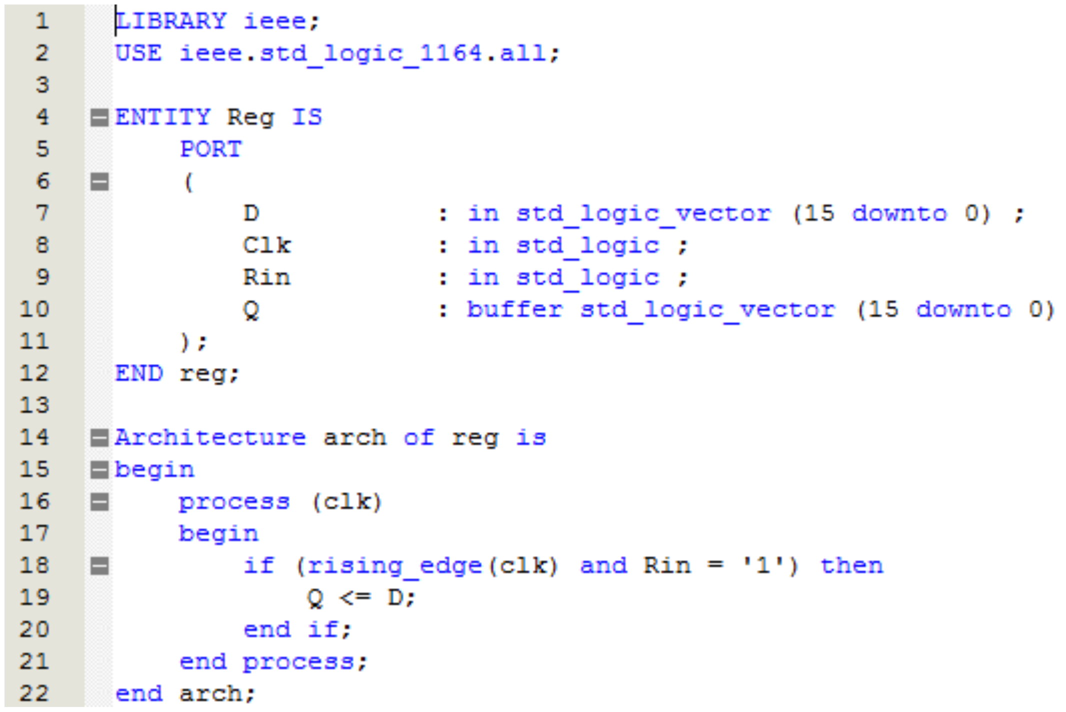
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use IEEE.numeric_std.all;

ENTITY AddSub IS
    PORT
    (
        Rx          : in std_logic_vector (15 downto 0) ;
        Ry          : in std_logic_vector (15 downto 0) ;
        as          : in std_logic;
        result      : buffer std_logic_vector (15 downto 0)
    );
END AddSub;

Architecture arch of Addsub is
begin
    result <= std_logic_vector(unsigned(Rx) + unsigned(Ry))
        when as = '1'
            else std_logic_vector(unsigned(Rx) - unsigned(Ry));
end arch;
```

Multiplexer



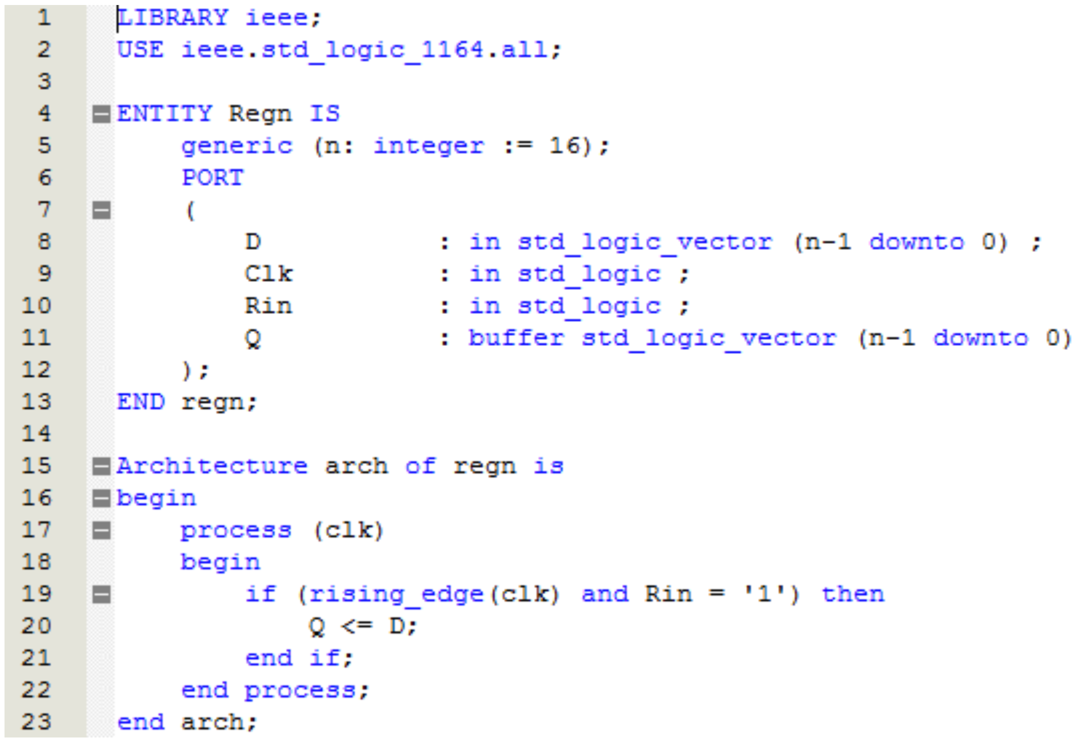
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Mux IS
    PORT
    (
        Gout              : in std_logic ;
        Dinout            : in std_logic ;
        Ry                : in std_logic_vector (2 downto 0) ;
        Din, Q0, Q1, Q2, Q3, Q4,
            Q5, Q6, Q7, G : in std_logic_vector (15 downto 0) ;
        res               : out std_logic_vector (15 downto 0)
    );
END Mux;

Architecture arch of Mux is
begin
    process(Ry, Gout, Dinout)
    begin
        if Gout = '1'      then res <= G;
        elsif Dinout = '1' then res <= Din;
        elsif Ry = "000"   then res <= Q0;
        elsif Ry = "001"   then res <= Q1;
        elsif Ry = "010"   then res <= Q2;
        elsif Ry = "011"   then res <= Q3;
        elsif Ry = "100"   then res <= Q4;
        elsif Ry = "101"   then res <= Q5;
        elsif Ry = "110"   then res <= Q6;
        elsif Ry = "111"   then res <= Q7;
        end if;
    end process;
end arch;
```

Register



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Reg IS
    PORT
    (
        D           : in std_logic_vector (15 downto 0) ;
        Clk         : in std_logic ;
        Rin         : in std_logic ;
        Q           : buffer std_logic_vector (15 downto 0)
    );
END reg;

Architecture arch of reg is
begin
    process (clk)
    begin
        if (rising_edge(clk) and Rin = '1') then
            Q <= D;
        end if;
    end process;
end arch;
```

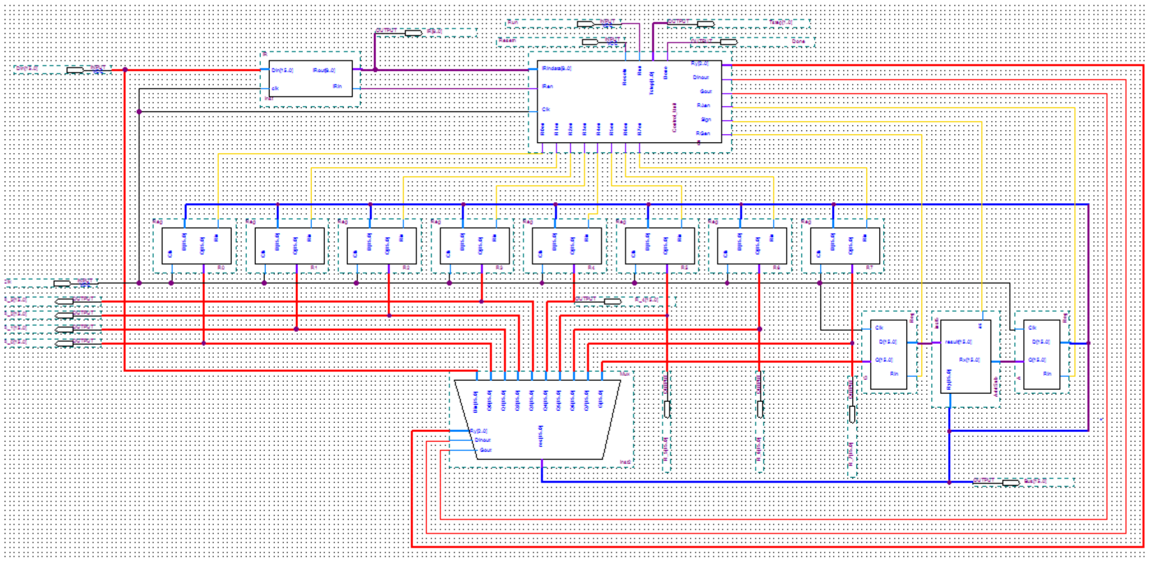
Instruction Register



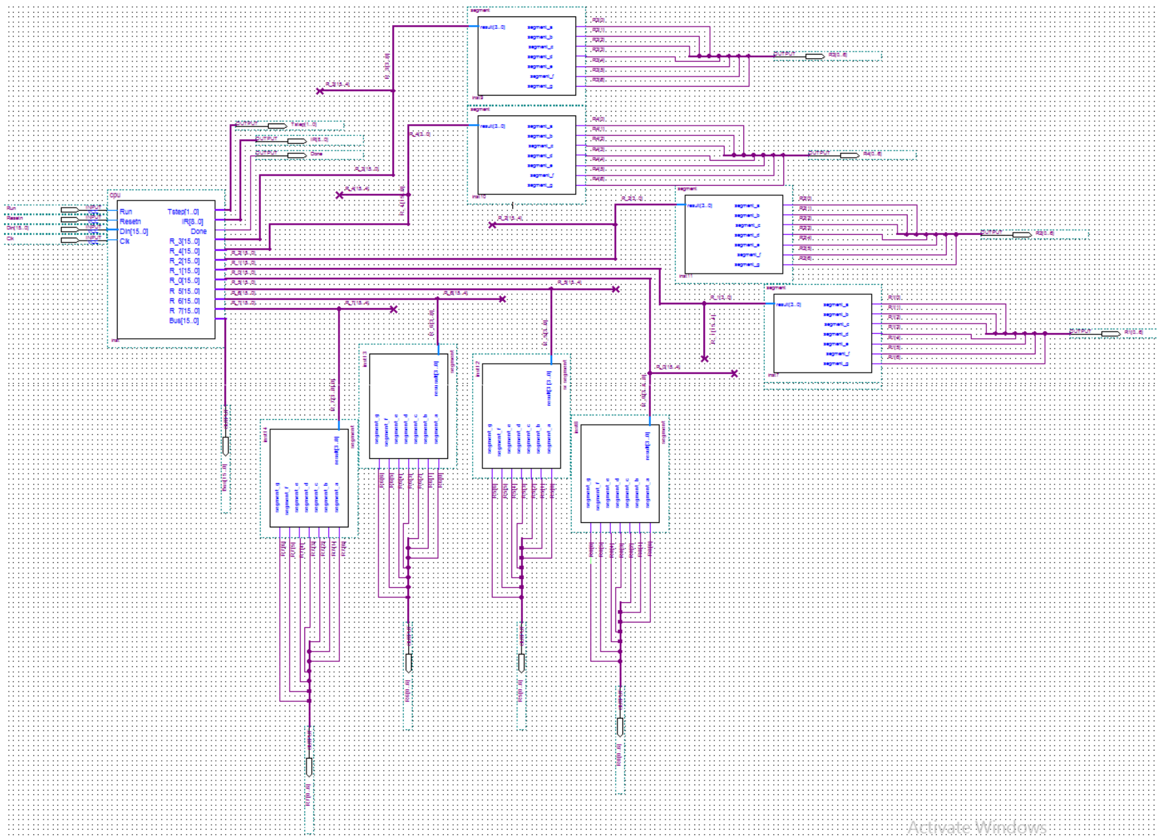
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Regn IS
    generic (n: integer := 16);
    PORT
    (
        D           : in std_logic_vector (n-1 downto 0) ;
        Clk         : in std_logic ;
        Rin         : in std_logic ;
        Q           : buffer std_logic_vector (n-1 downto 0)
    );
END regn;

Architecture arch of regn is
begin
    process (clk)
    begin
        if (rising_edge(clk) and Rin = '1') then
            Q <= D;
        end if;
    end process;
end arch;
```

Control Unit is in Appendix section

CPU

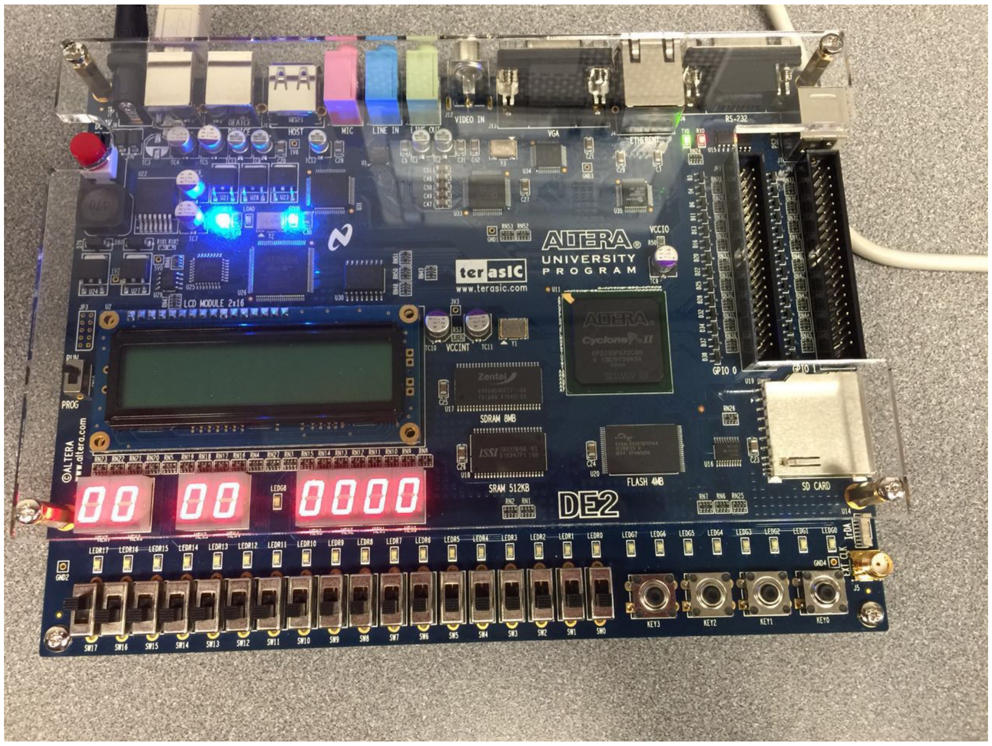


Processor

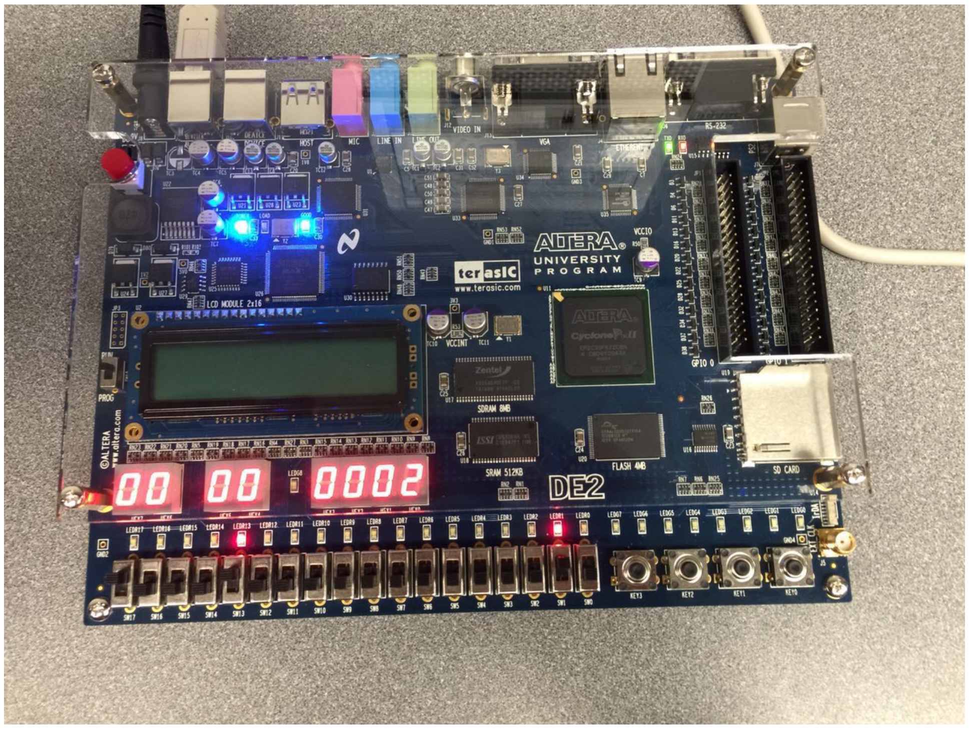


DE2 Board:

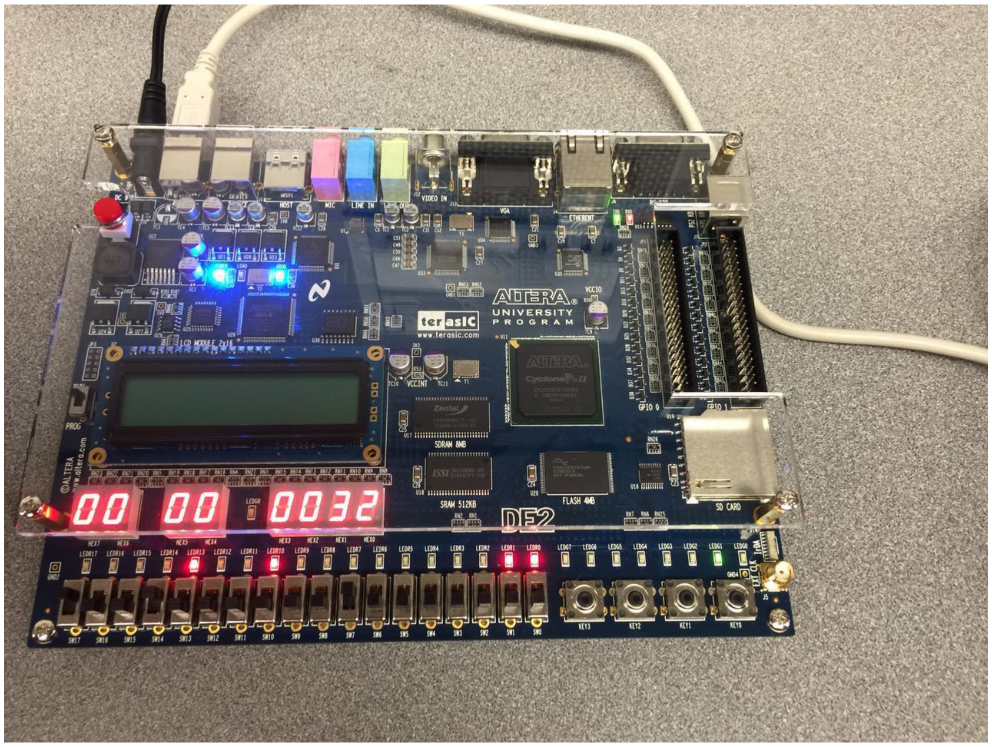
To start, set Run input to high



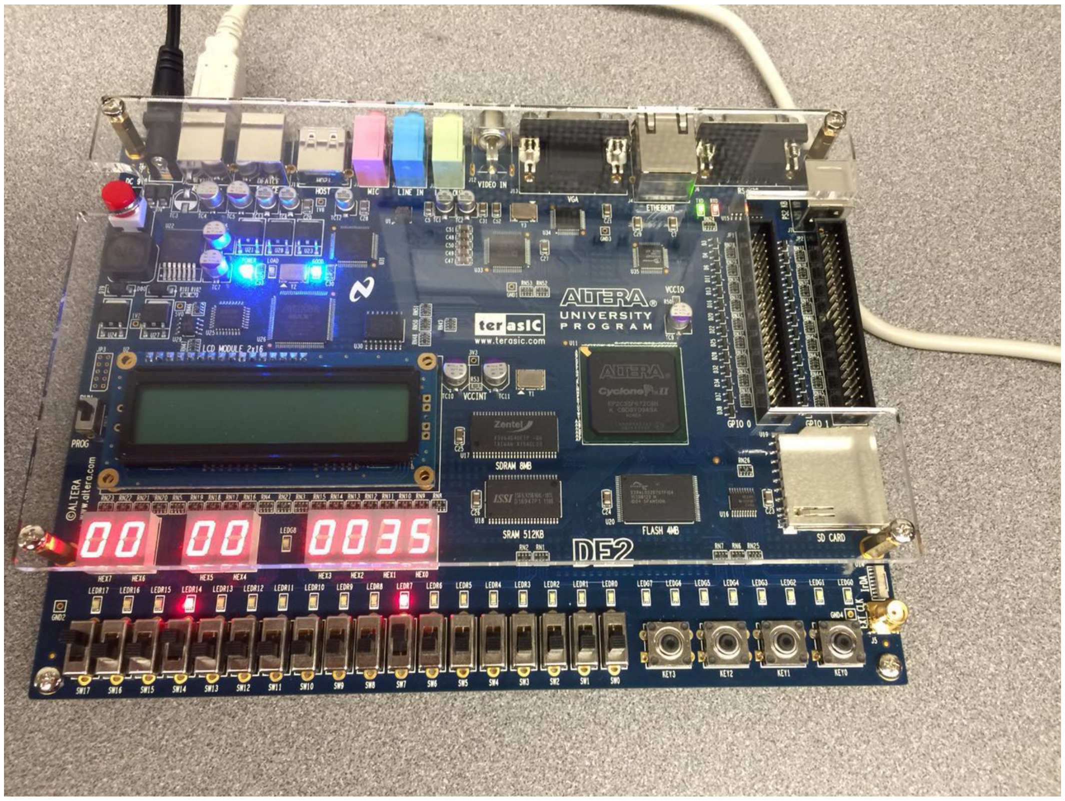
Store 2 into first register



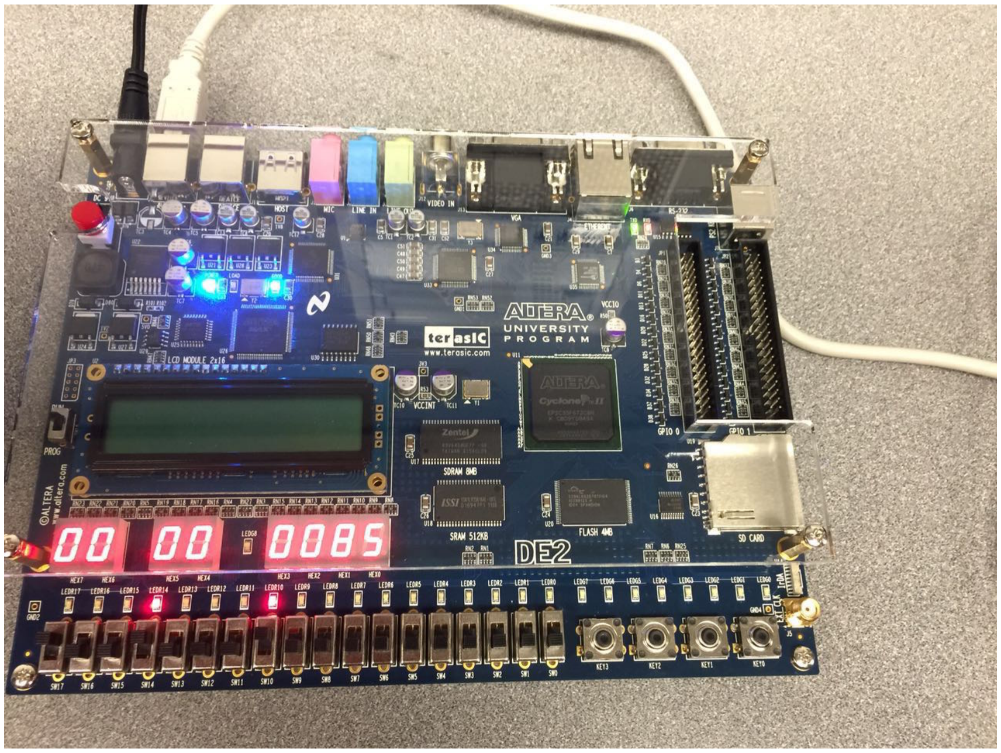
Store 3 to second register



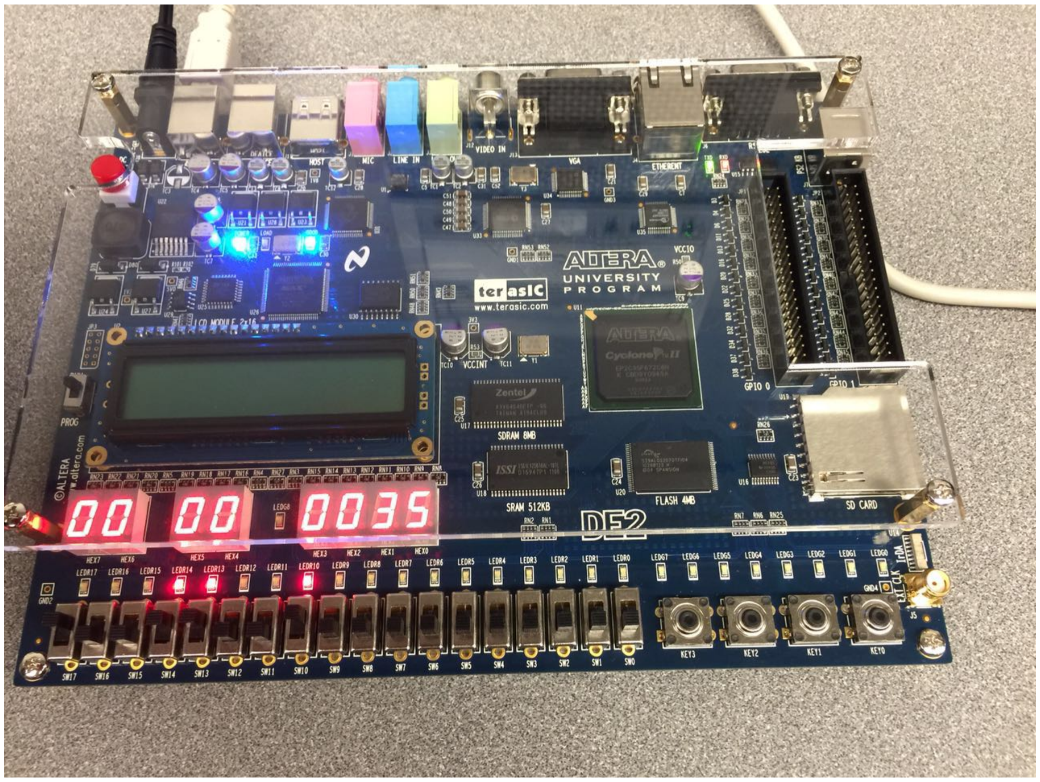
We can add 2 and 3 which result is 5 and store back to register 1



We can also add 3 and 5 which result is 8 and store back to register 2



To subtract, we can do 8-5=3 and store back to register 2



Conclusion:

This lab was much more complicated than previous labs. However, the objective of this lab was achieved because after designing the individual parts that make up the CPU, registers, multiplexer, adder, we tested the system on the DE2 board and it preformed as expected.

Appendix:

Pin Assignments

to	location
clk	PIN_N23
run	PIN_V2
resetn	PIN_G26
Done	PIN_AD12
Tstep[1]	PIN_AF22
Tstep[0]	PIN_AE22
Din[15]	PIN_U4
Din[14]	PIN_U3
Din[13]	PIN_T7
Din[12]	PIN_P2
Din[11]	PIN_P1
Din[10]	PIN_N1
Din[9]	PIN_A13
Din[8]	PIN_B13
Din[7]	PIN_C13
Din[6]	PIN_AC13
Din[5]	PIN_AD13
Din[4]	PIN_AF14
Din[3]	PIN_AE14
Din[2]	PIN_P25
Din[1]	PIN_N26
Din[0]	PIN_N25
Bus[15]	PIN_AE13
Bus[14]	PIN_AF13
Bus[13]	PIN_AE15
Bus[12]	PIN_AD15
Bus[11]	PIN_AC14
Bus[10]	PIN_AA13

```
Bus[9], PIN_Y13
Bus[8], PIN_AA14
Bus[7], PIN_AC21
Bus[6], PIN_AD21
Bus[5], PIN_AD23
Bus[4], PIN_AD22
Bus[3], PIN_AC22
Bus[2], PIN_AB21
Bus[1], PIN_AF23
Bus[0], PIN_AE23
R0[0], PIN_AF10
R0[1], PIN_AB12
R0[2], PIN_AC12
R0[3], PIN_AD11
R0[4], PIN_AE11
R0[5], PIN_V14
R0[6], PIN_V13
R1[0], PIN_V20
R1[1], PIN_V21
R1[2], PIN_W21
R1[3], PIN_Y22
R1[4], PIN_AA24
R1[5], PIN_AA23
R1[6], PIN_AB24
R2[0], PIN_AB23
R2[1], PIN_V22
R2[2], PIN_AC25
R2[3], PIN_AC26
R2[4], PIN_AB26
R2[5], PIN_AB25
R2[6], PIN_Y24
R3[0], PIN_Y23
R3[1], PIN_AA25
R3[2], PIN_AA26
R3[3], PIN_Y26
R3[4], PIN_Y25
R3[5], PIN_U22
R3[6], PIN_W24
R4[0], PIN_U9
R4[1], PIN_U1
R4[2], PIN_U2
R4[3], PIN_T4
R4[4], PIN_R7
R4[5], PIN_R6
R4[6], PIN_T3
R5[0], PIN_T2
```

```
R5[1], PIN_P6
R5[2], PIN_P7
R5[3], PIN_T9
R5[4], PIN_R5
R5[5], PIN_R4
R5[6], PIN_R3
R6[0], PIN_R2
R6[1], PIN_P4
R6[2], PIN_P3
R6[3], PIN_M2
R6[4], PIN_M3
R6[5], PIN_M5
R6[6], PIN_M4
R7[0], PIN_L3
R7[1], PIN_L2
R7[2], PIN_L9
R7[3], PIN_L6
R7[4], PIN_L7
R7[5], PIN_P9
R7[6], PIN_N9
```

Control Unit

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY Control_Unit IS
        PORT
        (
                Run                     : in std_logic ;
                Clk                     : in std_logic ;
                Resetn          : in std_logic ;
                IRindata        : in std_logic_vector (8 downto 0) ;
                IRen            : out std_logic;
                R0en            : out std_logic;
                R1en            : out std_logic;
                R2en            : out std_logic;
                R3en            : out std_logic;
                R4en            : out std_logic;
                R5en            : out std_logic;
                R6en            : out std_logic;
                R7en            : out std_logic;
                RAen            : out std_logic;
                RGen            : out std_logic;
                Sign            : out std_logic;
                Dinout          : out std_logic;
```

```
		Gout		: out std_logic;
		Ry			: out std_logic_vector(2 downto 0);
		Done		: buffer std_logic;
		Tstep		: out std_logic_vector(1 downto 0)
	);
END Control_Unit;

Architecture arch of Control_Unit is
	signal instruction, xReg, yReg  : std_logic_vector(2 downto 0);
	signal step					: std_logic_vector(1 downto 0);
begin

	instruction <= IRindata (8 downto 6);
	xReg		<= IRindata (5 downto 3);
	yReg		<= IRindata (2 downto 0);

	process (clk, resetn, Done) begin
		if resetn = '0' then
			step <= "00" ;
		elsif rising_edge(clk) then
			if Done = '1' then
				step <= "00";
			else
				step <= step + 1;
			end if;
		end if;
	end process;

	process (step, xReg) begin
		R0en <= '0'; R1en <= '0'; R2en <= '0'; R3en <= '0';  -- all registers
		R4en <= '0'; R5en <= '0'; R6en <= '0'; R7en <= '0';  -- are first unenabled.
		RAen <= '0'; RGen <= '0'; Gout <= '0';
		Dinout <= '1'; Done <= '0'; IRen <= '0';
		case step is
			when "00" =>
				IRen <= '1';
			when "01" =>
				if instruction ="000" or instruction = "001" then
					case xReg is
						when "000" => R0en <= '1';
						when "001" => R1en <= '1';
						when "010" => R2en <= '1';
						when "011" => R3en <= '1';
						when "100" => R4en <= '1';
						when "101" => R5en <= '1';
						when "110" => R6en <= '1';
```

```
                    when "111" => R7en <= '1';
                end case;
                Done <= '1';
                IRen <= '1';
                if instruction = "001" then
                    Dinout <= '1';
                elsif instruction = "000" then
                    Ry <= yReg;
                    Dinout <= '0';
                end if;
            elsif instruction = "010" or instruction = "011" then
                    Ry <= xReg;
                    Dinout <= '0';
                    RAen <= '1';
            end if;
        when "10" =>
            if instruction = "010" or instruction = "011" then
                Ry <= yReg;
                Dinout <= '0';
                RGen <= '1';
                Sign <= '0';
                if instruction = "010" then
                    Sign <= '1';
                end if;
            end if;
        when "11" =>
            if instruction = "010" or instruction = "011" then
                case xReg is
                    when "000" => R0en <= '1';
                    when "001" => R1en <= '1';
                    when "010" => R2en <= '1';
                    when "011" => R3en <= '1';
                    when "100" => R4en <= '1';
                    when "101" => R5en <= '1';
                    when "110" => R6en <= '1';
                    when "111" => R7en <= '1';
                end case;
                Gout <= '1';
                Done <= '1';
                IRen <= '1';
            end if;
        end case;
end process;

Tstep <= step;
```

```
end arch;
```