Weifan Lin
Csc343
04/17/2015

Simplified Single Cycle CPU

Objectives:

The goal of this lab was to design and test a single cycle CPU consisting of a number of 16-bit registers, a multiplexer, an adder-subtractor. Data was loaded using a 16-bit input into the multiplexer, which was connected to many different registers. The CPU was capable of four different instructions: moving data between registers, moving data directly to registers, adding data between registers, and subtracting data between registers.

Specifications:

Zero extend: has a 2-bit input and a 16-bit output.

Multiplexor: has two 16-bit inputs and a selector and a 16-bit output.

16x16 register: has a 16-bit data input, three registers Rw Ra Rb, a Write-enable input and three 16-bit outputs.

Adder and subtractor: has a cin input, two 16-bit data inputs, one 16-bit output as results, and cout output and overflow detection.

Opcode: has a 2-bit inputs and a 2-bit output.

Seven segment display: has a 16-bit input and seven 4-bit outputs.

Functionality:

Zero extend

Zero extend is to extend 2-bit data input to a 16-bit data input which to be used in 16-bit Adder/Subtractor.

Multiplexor

Multiplexor selector selects either the data input or the result from adder/subtractor.

16x16 register

Instruction Register Stores the instruction during the execution time.

Combined Circuit

When selector is high, input the 2-bit data and zero-extend extends the 2-bit data to 16-bit, and high write-enable to save it into Rw with an address. And do again with second input and save it to another address. Set selector to low and assign the two addresses to Ra and Rb. Then high write-enable to do addiction/subtraction and save the result into Rw with an address.

Design:

Opcode

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.all;
3   entity opcode is
4    port(
5     op : in std_logic_vector(1 downto 0);
6     dec : out std_logic_vector(1 downto 0));
7   end opcode;
8   architecture arch of opcode is
9   begin
10  with op select
11    dec <= "00" when "00",
12    "01" when "01",
13    "10" when "10",
14    "11" when "11";
15  end arch;
```

Zero Extend

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   entity zero_extend is
7     port ( a : in std_logic_vector(1 downto 0);
8            a_out : out std_logic_vector(15 downto 0));
9
10  end zero_extend;
11
12  architecture behavioral of zero_extend is
13  begin
14    process (a) is
15  begin
16
17    a_out <= "00000000000000" & a;
18  end process;
19  end behavioral;
```

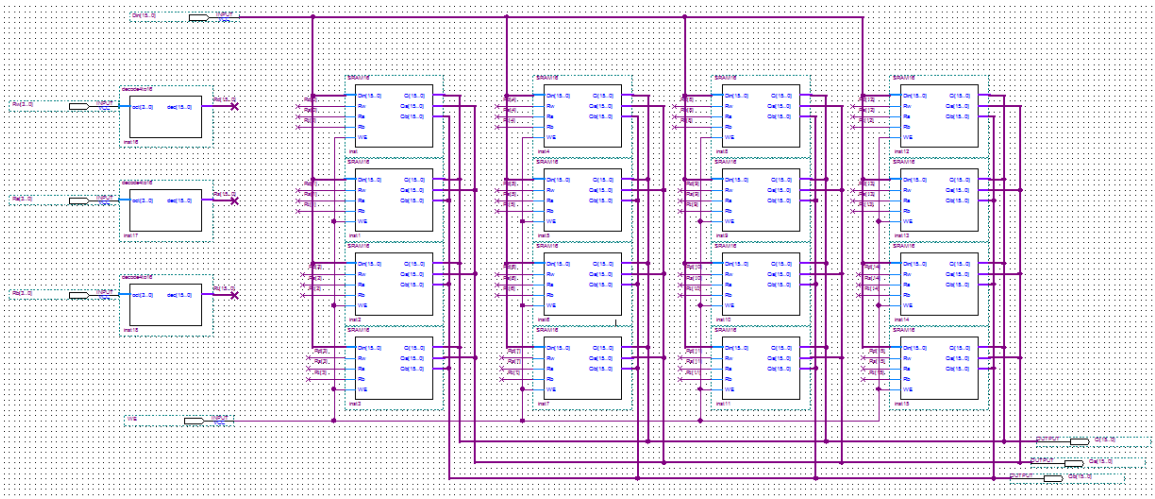
SRAM16

```
1   LIBRARY IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity SRAM16 is
5       port( Din : in STD_LOGIC_VECTOR(15 downto 0);
6             Rw, Ra, Rb, WE    : in STD_LOGIC;
7             Q : out STD_LOGIC_VECTOR(15 downto 0);
8             Qa : out STD_LOGIC_VECTOR(15 downto 0);
9             Qb : out STD_LOGIC_VECTOR(15 downto 0));
10  end SRAM16 ;
11
12  architecture behav of SRAM16  is
13  component SRAM is
14      port( Din, CS, WE    : in STD_LOGIC;
15            Q : out STD_LOGIC );
16  end component;
17
18     signal ans : std_logic_vector(15 downto 0);
19
20  begin
21
22
23  s16_s0: SRAM port map (Din(0), Rw, WE, ans(0));
24      s16_s1: SRAM port map (Din(1), Rw, WE, ans(1));
25      s16_s2: SRAM port map (Din(2), Rw, WE, ans(2));
26      s16_s3: SRAM port map (Din(3), Rw, WE, ans(3));
27      s16_s4: SRAM port map (Din(4), Rw, WE, ans(4));
28      s16_s5: SRAM port map (Din(5), Rw, WE, ans(5));
29      s16_s6: SRAM port map (Din(6), Rw, WE, ans(6));
30      s16_s7: SRAM port map (Din(7), Rw, WE, ans(7));
31      s16_s8: SRAM port map (Din(8), Rw, WE, ans(8));
32      s16_s9: SRAM port map (Din(9), Rw, WE, ans(9));
33      s16_s10: SRAM port map (Din(10), Rw, WE, ans(10));
34      s16_s11: SRAM port map (Din(11), Rw, WE, ans(11));
35      s16_s12: SRAM port map (Din(12), Rw, WE, ans(12));
36      s16_s13: SRAM port map (Din(13), Rw, WE, ans(13));
37      s16_s14: SRAM port map (Din(14), Rw, WE, ans(14));
38      s16_s15: SRAM port map (Din(15), Rw, WE, ans(15));
39
40      Q(15) <= ans(15) when (Rw = '1') else 'Z';
41      Q(14) <= ans(14) when (Rw = '1') else 'Z';
42      Q(13) <= ans(13) when (Rw = '1') else 'Z';
```

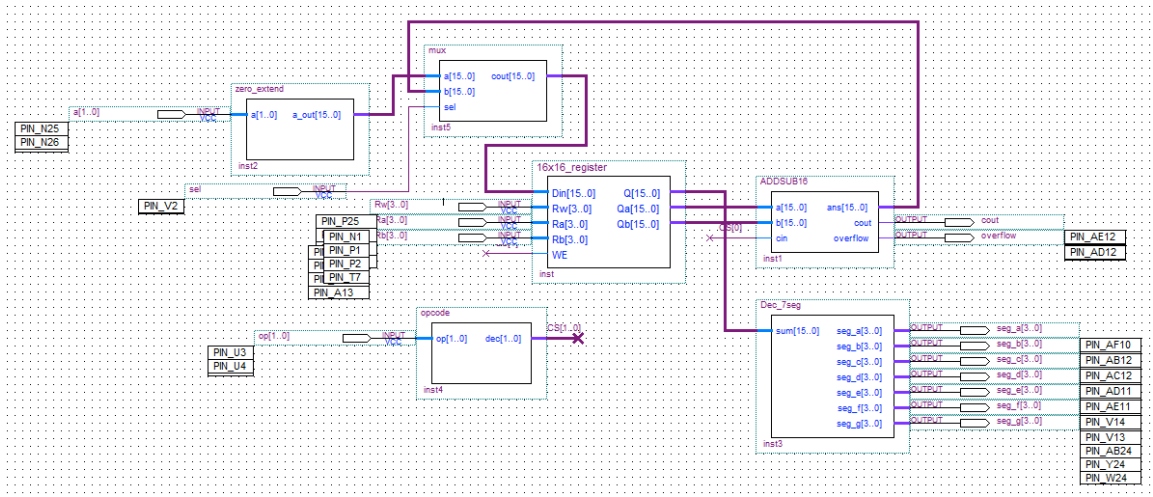
```
43      Q(12) <= ans(12) when (Rw = '1') else 'Z';
44      Q(11) <= ans(11) when (Rw = '1') else 'Z';
45      Q(10) <= ans(10) when (Rw = '1') else 'Z';
46      Q(9) <= ans(9) when (Rw = '1') else 'Z';
47      Q(8) <= ans(8) when (Rw = '1') else 'Z';
48      Q(7) <= ans(7) when (Rw = '1') else 'Z';
49      Q(6) <= ans(6) when (Rw = '1') else 'Z';
50      Q(5) <= ans(5) when (Rw = '1') else 'Z';
51      Q(4) <= ans(4) when (Rw = '1') else 'Z';
52      Q(3) <= ans(3) when (Rw = '1') else 'Z';
53      Q(2) <= ans(2) when (Rw = '1') else 'Z';
54      Q(1) <= ans(1) when (Rw = '1') else 'Z';
55      Q(0) <= ans(0) when (Rw = '1') else 'Z';
56
57      Qa(15) <= ans(15) when (Ra = '1') else 'Z';
58      Qa(14) <= ans(14) when (Ra = '1') else 'Z';
59      Qa(13) <= ans(13) when (Ra = '1') else 'Z';
60      Qa(12) <= ans(12) when (Ra = '1') else 'Z';
61      Qa(11) <= ans(11) when (Ra = '1') else 'Z';
62      Qa(10) <= ans(10) when (Ra = '1') else 'Z';
63      Qa(9) <= ans(9) when (Ra = '1') else 'Z';
64      Qa(8) <= ans(8) when (Ra = '1') else 'Z';
65      Qa(7) <= ans(7) when (Ra = '1') else 'Z';
66      Qa(6) <= ans(6) when (Ra = '1') else 'Z';
67      Qa(5) <= ans(5) when (Ra = '1') else 'Z';
68      Qa(4) <= ans(4) when (Ra = '1') else 'Z';
69      Qa(3) <= ans(3) when (Ra = '1') else 'Z';
70      Qa(2) <= ans(2) when (Ra = '1') else 'Z';
71      Qa(1) <= ans(1) when (Ra = '1') else 'Z';
72      Qa(0) <= ans(0) when (Ra = '1') else 'Z';
73
74      Qb(15) <= ans(15) when (Rb = '1') else 'Z';
75      Qb(14) <= ans(14) when (Rb = '1') else 'Z';
76      Qb(13) <= ans(13) when (Rb = '1') else 'Z';
77      Qb(12) <= ans(12) when (Rb = '1') else 'Z';
78      Qb(11) <= ans(11) when (Rb = '1') else 'Z';
79      Qb(10) <= ans(10) when (Rb = '1') else 'Z';
80      Qb(9) <= ans(9) when (Rb = '1') else 'Z';
81      Qb(8) <= ans(8) when (Rb = '1') else 'Z';
82      Qb(7) <= ans(7) when (Rb = '1') else 'Z';
83      Qb(6) <= ans(6) when (Rb = '1') else 'Z';

84      Qb(5) <= ans(5) when (Rb = '1') else 'Z';
85      Qb(4) <= ans(4) when (Rb = '1') else 'Z';
86      Qb(3) <= ans(3) when (Rb = '1') else 'Z';
87      Qb(2) <= ans(2) when (Rb = '1') else 'Z';
88      Qb(1) <= ans(1) when (Rb = '1') else 'Z';
89      Qb(0) <= ans(0) when (Rb = '1') else 'Z';
90  end behav;
```

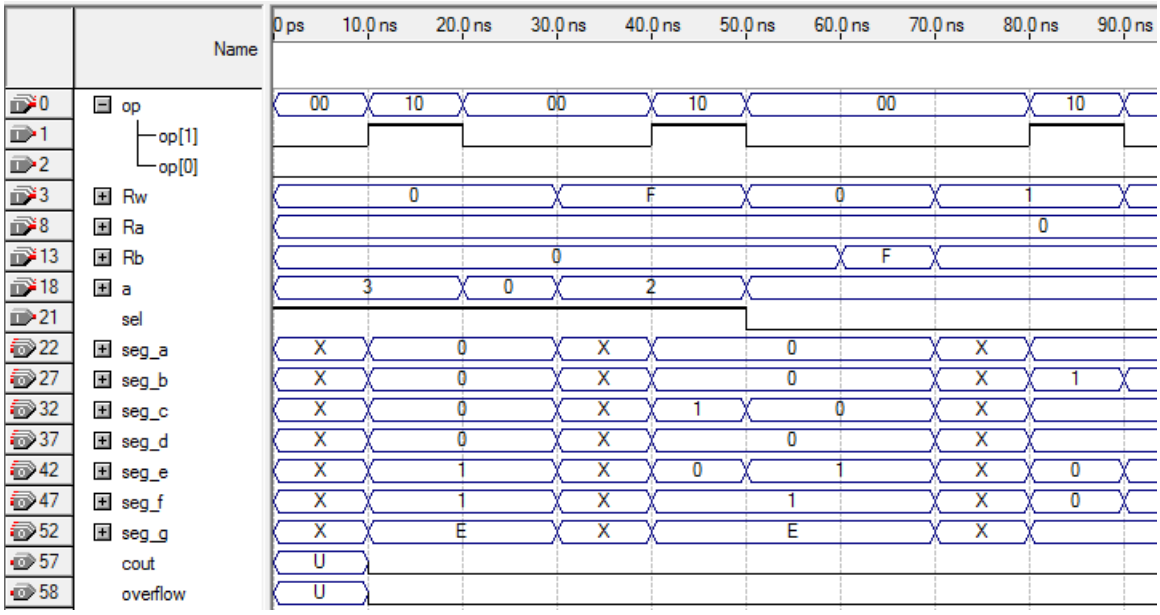
16x16 register

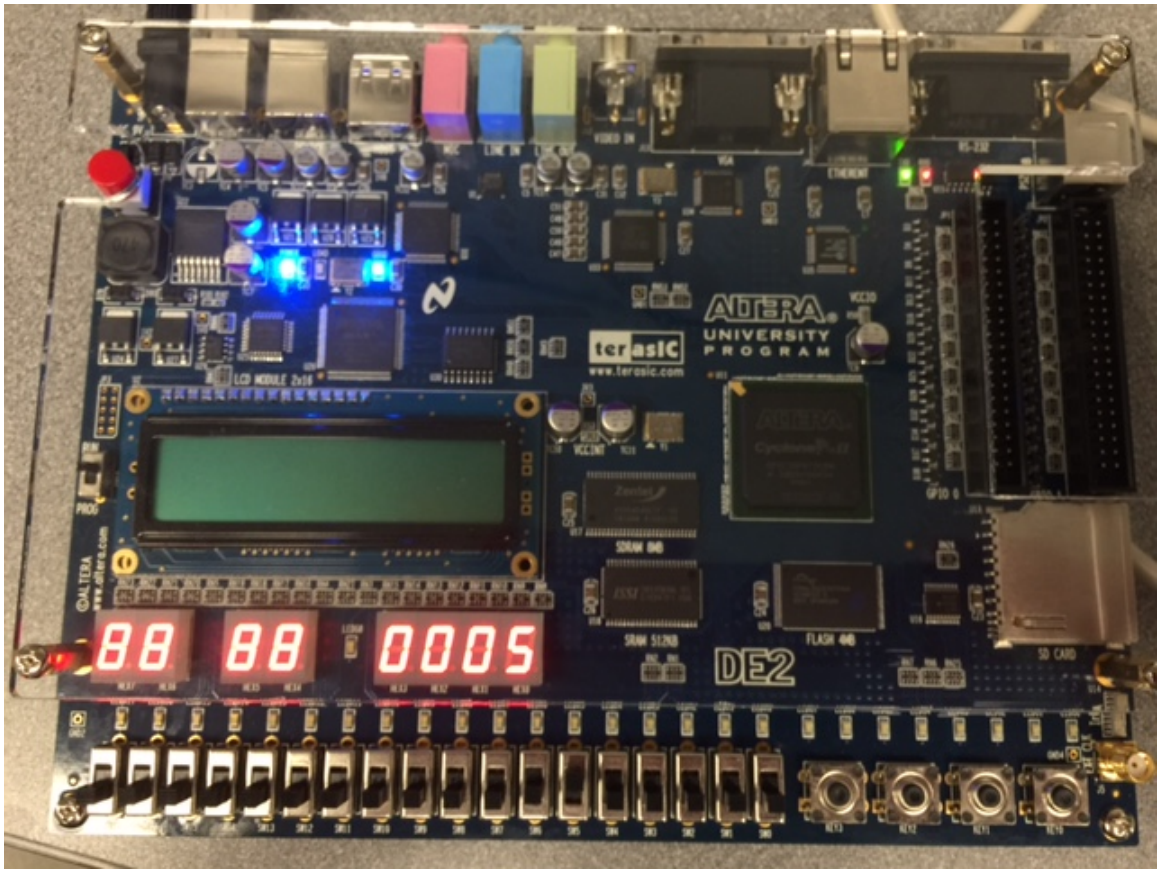


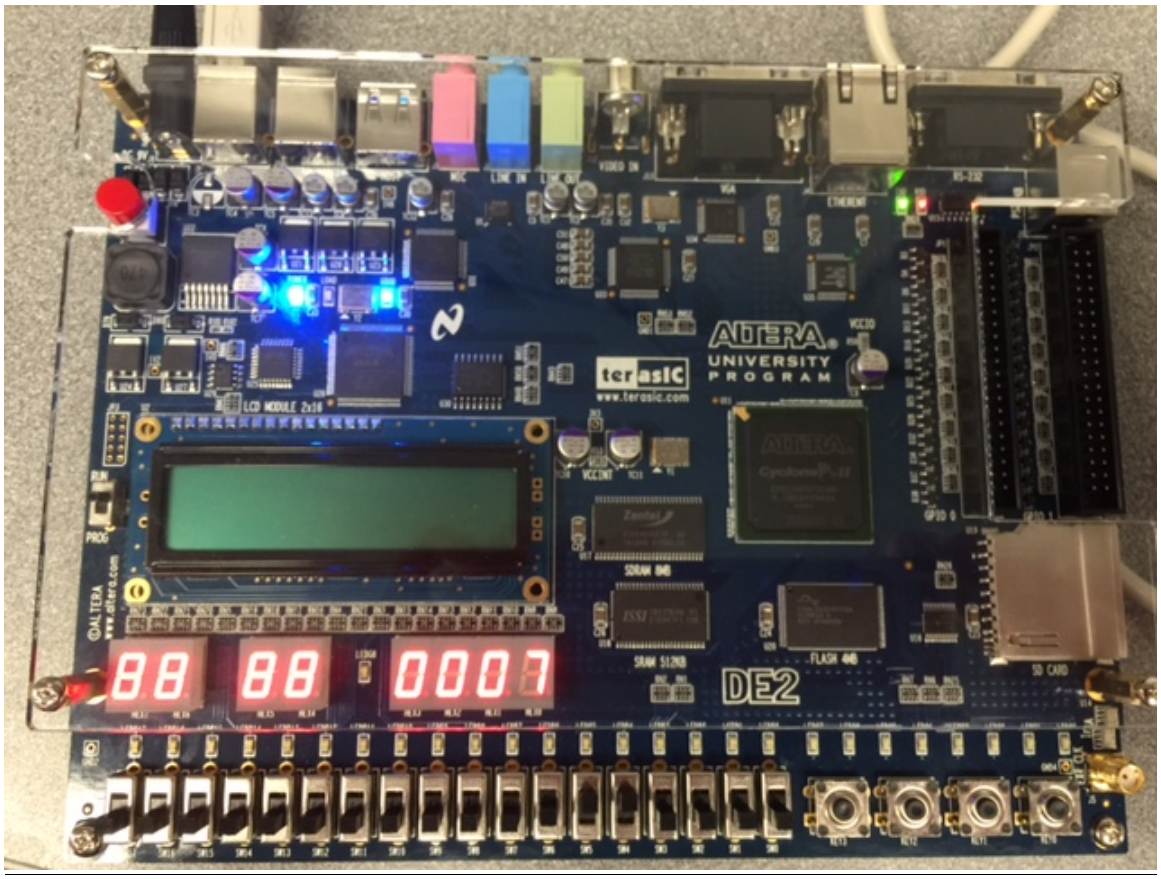
CPU

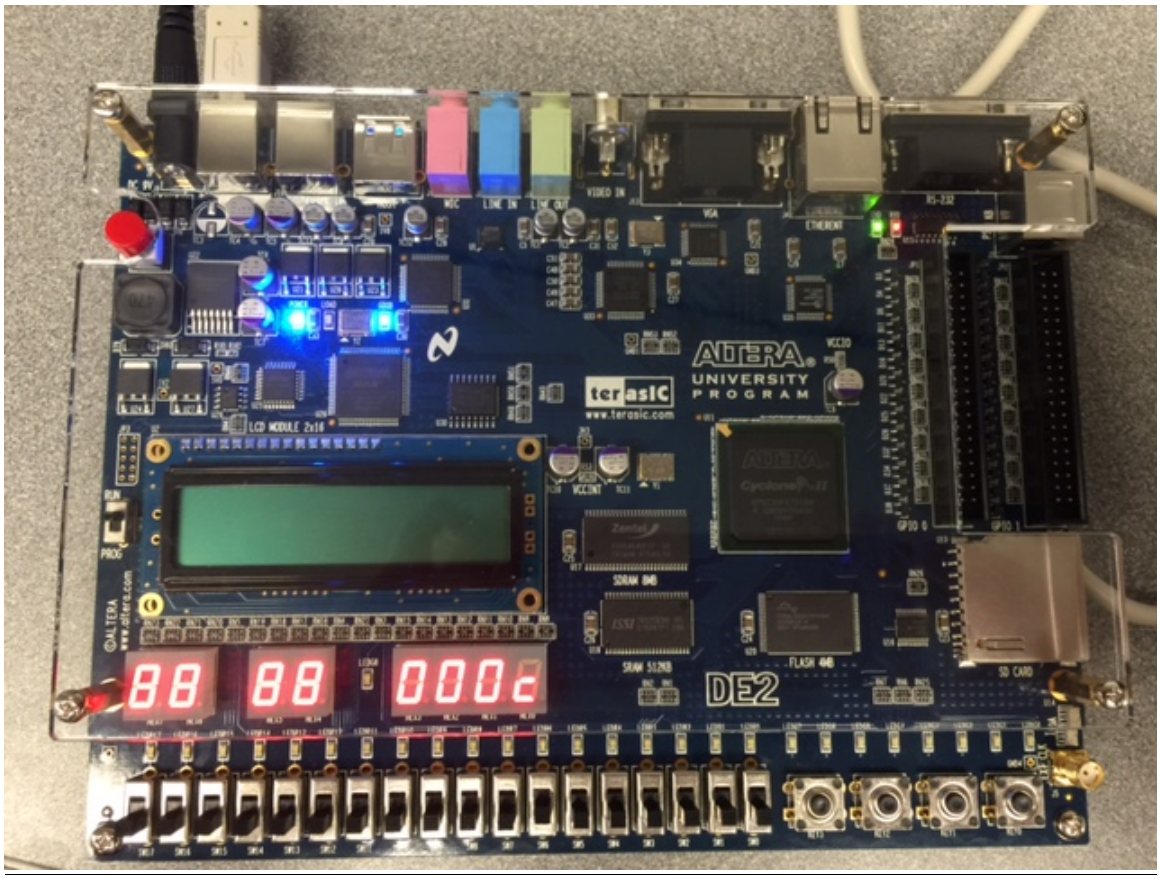


Simulation:


Name
0 ps
10.0 ns
20.0 ns
30.0 ns
40.0 ns
50.0 ns
60.0 ns
70.0 ns
80.0 ns
90.0 ns
op
op[1]
op[0]
Rw
Ra
Rb
a
sel
seg_a
seg_b
seg_c
seg_d
seg_e
seg_f
seg_g
cout
overflow


ALTERA
UNIVERSITY
PROGRAM
terasIC
DE2


ALTERA
UNIVERSITY
PROGRAM
terasIC
www.terasic.com
DE2
LCD MODULE 2x16
SDRAM 8MB
SRAM 512KB
FLASH 4MB
SD CARD
VGA
ETHERNET
RS-232
VIDEO IN
MIC
LINE IN
LINE OUT
VCCIO
VCCINT
GPIO 0
GPIO 1
PROG
RUN
LEDR4
88
88
0007



Testing CPU circuit on DE2 board proven to be successful, and it behaved exactly as demonstrated in the waveform table. Above pictures were one of tests I made. I assigned value 5 to address 1000 and value 7 to 1100, and I added them and save result to address 0000 which was value c (12).

Conclusion:

This lab was quite advanced and more complex than any of the other labs we have completed so far, as the CPU was composed of several different components. However, because we were provided the skeleton for the CPU, we were just needed to implement

the instructions. Testing it on the DE2 board was not too problematic, as I was successfully able to have it working and comprehended after a few attempts.

Appendix:

Pin Assignments:

to, location
a[0], PIN_N25
a[1], PIN_N26
Rw[0], PIN_P25
Rw[1], PIN_AE14
Rw[2], PIN_AF14
Rw[3], PIN_AD13
Rb[0], PIN_AC13
Rb[1], PIN_C13
Rb[2], PIN_B13
Rb[3], PIN_A13
Ra[0], PIN_N1
Ra[1], PIN_P1
Ra[2], PIN_P2
Ra[3], PIN_T7
op[0], PIN_U3
op[1], PIN_U4

sel, PIN_V2

cout, PIN_AE12
overflow, PIN_AD12

seg_a[0], PIN_AF10
seg_b[0], PIN_AB12
seg_c[0], PIN_AC12
seg_d[0], PIN_AD11
seg_e[0], PIN_AE11
seg_f[0], PIN_V14
seg_g[0], PIN_V13
seg_a[1], PIN_V20
seg_b[1], PIN_V21
seg_c[1], PIN_W21

seg_d[1], PIN_Y22
seg_e[1], PIN_AA24
seg_f[1], PIN_AA23
seg_g[1], PIN_AB24
seg_a[2], PIN_AB23
seg_b[2], PIN_V22
seg_c[2], PIN_AC25
seg_d[2], PIN_AC26
seg_e[2], PIN_AB26
seg_f[2], PIN_AB25
seg_g[2], PIN_Y24
seg_a[3], PIN_Y23
seg_b[3], PIN_AA25
seg_c[3], PIN_AA26
seg_d[3], PIN_Y26
seg_e[3], PIN_Y25
seg_f[3], PIN_U22
seg_g[3], PIN_W24

Zero Extend

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity zero_extend is
  port ( a : in std_logic_vector(1 downto 0);
        a_out : out std_logic_vector(15 downto 0));

end zero_extend;

architecture behavioral of zero_extend is
begin
  process (a) is
begin

  a_out <= "00000000000000" & a;
end process;
end behavioral;
```

Opcode

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity opcode is
 port(
 op : in std_logic_vector(1 downto 0);
 dec : out std_logic_vector(1 downto 0));
end opcode;
architecture arch of opcode is
begin
with op select
 dec <= "00" when "00",
 "01" when "01",
 "10" when "10",
 "11" when "11";
end arch;
```

Mux

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux is
        port( a : in STD_LOGIC_VECTOR(15 downto 0);
              b    : in STD_LOGIC_VECTOR(15 downto 0);
              sel : in STD_LOGIC;
              cout : out STD_LOGIC_VECTOR(15 downto 0));
end mux;

architecture BHV of mux is
begin

        cout <= a when (sel='1') else b;
end BHV;
```

4to16 Decoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

```
entity decode4to16 is
 port(
 oct : in std_logic_vector(3 downto 0);
 dec : out std_logic_vector(15 downto 0));
end decode4to16;
architecture arch of decode4to16 is
begin
with oct select
 dec <= "0000000000000001" when "0000",
 "0000000000000010" when "0001",
 "0000000000000100" when "0010",
 "0000000000001000" when "0011",
 "0000000000010000" when "0100",
 "0000000000100000" when "0101",
 "0000000001000000" when "0110",
 "0000000010000000" when "0111",
 "0000000100000000" when "1000",
 "0000001000000000" when "1001",
 "0000010000000000" when "1010",
 "0000100000000000" when "1011",
 "0001000000000000" when "1100",
 "0010000000000000" when "1101",
 "0100000000000000" when "1110",
 "1000000000000000" when "1111",
 "0000000000000000" when others;
end arch;
```

SRAM

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SRAM is
        port( Din, CS, WE    : in STD_LOGIC;
              Q : out STD_LOGIC );
end SRAM ;

architecture behav of SRAM  is
component Positive_Dff is
                  port( D, C    : in STD_LOGIC;
```

```
            Q, notQ : buffer STD_LOGIC );
end component;

  signal Clk, Q0 : std_logic;

begin
        Clk <= CS and WE;

        s0: Positive_Dff port map (Din, Clk, Q0);

        Q <= Q0;

end behav;
```

16bit Adder/Subtractor

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ADDSUB16 is
 port( a, b              : in    STD_LOGIC_VECTOR(15 downto 0);
      cin : in STD_LOGIC;
            ans         : out   STD_LOGIC_VECTOR(15 downto 0);

            cout, overflow              : out   STD_LOGIC           );
end ADDSUB16;

architecture struct of ADDSUB16 is
component ADDER2 is
 port( a0, a1 : in      STD_LOGIC;
      b0, b1            : in    STD_LOGIC;
      cin : in STD_LOGIC;
            ans0, ans1          : out   STD_LOGIC;
            cout : out  STD_LOGIC           );
end component;

signal c0, c1, c2, c3, c4, c5, c6, c7 : STD_LOGIC;
signal MP: std_logic_vector(15 downto 0);
begin
```

```
MP(0) <= cin XOR b(0);
MP(1) <= cin XOR b(1);
MP(2) <= cin XOR b(2);
MP(3) <= cin XOR b(3);
MP(4) <= cin XOR b(4);
MP(5) <= cin XOR b(5);
MP(6) <= cin XOR b(6);
MP(7) <= cin XOR b(7);
MP(8) <= cin XOR b(8);
MP(9) <= cin XOR b(9);
MP(10) <= cin XOR b(10);
MP(11) <= cin XOR b(11);
MP(12) <= cin XOR b(12);
MP(13) <= cin XOR b(13);
MP(14) <= cin XOR b(14);
MP(15) <= cin XOR b(15);


b_adder0: ADDER2 port map (a(0), a(1), MP(0), MP(1), cin, ans(0), ans(1), c0);

b_adder1: ADDER2 port map (a(2), a(3), MP(2), MP(3), c0, ans(2), ans(3), c1);

b_adder2: ADDER2 port map (a(4), a(5), MP(4), MP(5), c1, ans(4), ans(5), c2);

b_adder3: ADDER2 port map (a(6), a(7), MP(6), MP(7), c2, ans(6), ans(7), c3);

b_adder4: ADDER2 port map (a(8), a(9), MP(8), MP(9), c3, ans(8), ans(9), c4);

b_adder5: ADDER2 port map (a(10), a(11), MP(10), MP(11), c4, ans(10), ans(11), c5);

b_adder6: ADDER2 port map (a(12), a(13), MP(12), MP(13), c5, ans(12), ans(13), c6);

b_adder7: ADDER2 port map (a(14), a(15), MP(14), MP(15), c6, ans(14), ans(15), c7);

overflow <= ((a(14) and (MP(14) or c6)) or (c6 and MP(14))) XOR c7;

cout <= not cin and c7;

end struct;
```

SRAM16

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SRAM16 is
        port( Din : in STD_LOGIC_VECTOR(15 downto 0);
            Rw, Ra, Rb, WE    : in STD_LOGIC;
            Q : out STD_LOGIC_VECTOR(15 downto 0);
            Qa : out STD_LOGIC_VECTOR(15 downto 0);
            Qb : out STD_LOGIC_VECTOR(15 downto 0));
end SRAM16 ;

architecture behav of SRAM16  is
component SRAM is
        port( Din, CS, WE    : in STD_LOGIC;
            Q : out STD_LOGIC );
end component;

  signal ans : std_logic_vector(15 downto 0);

begin


s16_s0: SRAM port map (Din(0), Rw, WE, ans(0));
        s16_s1: SRAM port map (Din(1), Rw, WE, ans(1));
        s16_s2: SRAM port map (Din(2), Rw, WE, ans(2));
        s16_s3: SRAM port map (Din(3), Rw, WE, ans(3));
        s16_s4: SRAM port map (Din(4), Rw, WE, ans(4));
        s16_s5: SRAM port map (Din(5), Rw, WE, ans(5));
        s16_s6: SRAM port map (Din(6), Rw, WE, ans(6));
        s16_s7: SRAM port map (Din(7), Rw, WE, ans(7));
        s16_s8: SRAM port map (Din(8), Rw, WE, ans(8));
        s16_s9: SRAM port map (Din(9), Rw, WE, ans(9));
        s16_s10: SRAM port map (Din(10), Rw, WE, ans(10));
        s16_s11: SRAM port map (Din(11), Rw, WE, ans(11));
        s16_s12: SRAM port map (Din(12), Rw, WE, ans(12));
        s16_s13: SRAM port map (Din(13), Rw, WE, ans(13));
        s16_s14: SRAM port map (Din(14), Rw, WE, ans(14));
```

```
s16_s15: SRAM port map (Din(15), Rw, WE, ans(15));

Q(15) <= ans(15) when (Rw = '1') else 'Z';
Q(14) <= ans(14) when (Rw = '1') else 'Z';
Q(13) <= ans(13) when (Rw = '1') else 'Z';
Q(12) <= ans(12) when (Rw = '1') else 'Z';
Q(11) <= ans(11) when (Rw = '1') else 'Z';
Q(10) <= ans(10) when (Rw = '1') else 'Z';
Q(9) <= ans(9) when (Rw = '1') else 'Z';
Q(8) <= ans(8) when (Rw = '1') else 'Z';
Q(7) <= ans(7) when (Rw = '1') else 'Z';
Q(6) <= ans(6) when (Rw = '1') else 'Z';
Q(5) <= ans(5) when (Rw = '1') else 'Z';
Q(4) <= ans(4) when (Rw = '1') else 'Z';
Q(3) <= ans(3) when (Rw = '1') else 'Z';
Q(2) <= ans(2) when (Rw = '1') else 'Z';
Q(1) <= ans(1) when (Rw = '1') else 'Z';
Q(0) <= ans(0) when (Rw = '1') else 'Z';

Qa(15) <= ans(15) when (Ra = '1') else 'Z';
Qa(14) <= ans(14) when (Ra = '1') else 'Z';
Qa(13) <= ans(13) when (Ra = '1') else 'Z';
Qa(12) <= ans(12) when (Ra = '1') else 'Z';
Qa(11) <= ans(11) when (Ra = '1') else 'Z';
Qa(10) <= ans(10) when (Ra = '1') else 'Z';
Qa(9) <= ans(9) when (Ra = '1') else 'Z';
Qa(8) <= ans(8) when (Ra = '1') else 'Z';
Qa(7) <= ans(7) when (Ra = '1') else 'Z';
Qa(6) <= ans(6) when (Ra = '1') else 'Z';
Qa(5) <= ans(5) when (Ra = '1') else 'Z';
Qa(4) <= ans(4) when (Ra = '1') else 'Z';
Qa(3) <= ans(3) when (Ra = '1') else 'Z';
Qa(2) <= ans(2) when (Ra = '1') else 'Z';
Qa(1) <= ans(1) when (Ra = '1') else 'Z';
Qa(0) <= ans(0) when (Ra = '1') else 'Z';

Qb(15) <= ans(15) when (Rb = '1') else 'Z';
Qb(14) <= ans(14) when (Rb = '1') else 'Z';
Qb(13) <= ans(13) when (Rb = '1') else 'Z';
Qb(12) <= ans(12) when (Rb = '1') else 'Z';
```

```
        Qb(11) <= ans(11) when (Rb = '1') else 'Z';
        Qb(10) <= ans(10) when (Rb = '1') else 'Z';
        Qb(9) <= ans(9) when (Rb = '1') else 'Z';
        Qb(8) <= ans(8) when (Rb = '1') else 'Z';
        Qb(7) <= ans(7) when (Rb = '1') else 'Z';
        Qb(6) <= ans(6) when (Rb = '1') else 'Z';
        Qb(5) <= ans(5) when (Rb = '1') else 'Z';
        Qb(4) <= ans(4) when (Rb = '1') else 'Z';
        Qb(3) <= ans(3) when (Rb = '1') else 'Z';
        Qb(2) <= ans(2) when (Rb = '1') else 'Z';
        Qb(1) <= ans(1) when (Rb = '1') else 'Z';
        Qb(0) <= ans(0) when (Rb = '1') else 'Z';
end behav;
```