Lab Report #3

Name: Ruochen Duan
Student ID: 1405106

① Introduction

In this lab, I implement a SingalCycleCPU by using the components designed in previous lab and test it.

First, we need to connect each data path. I use the figure given by the file"assignment3.pdf" to implement a SingalCycleCPU. Figure 1 is the SingalCycle ARM for lab3.

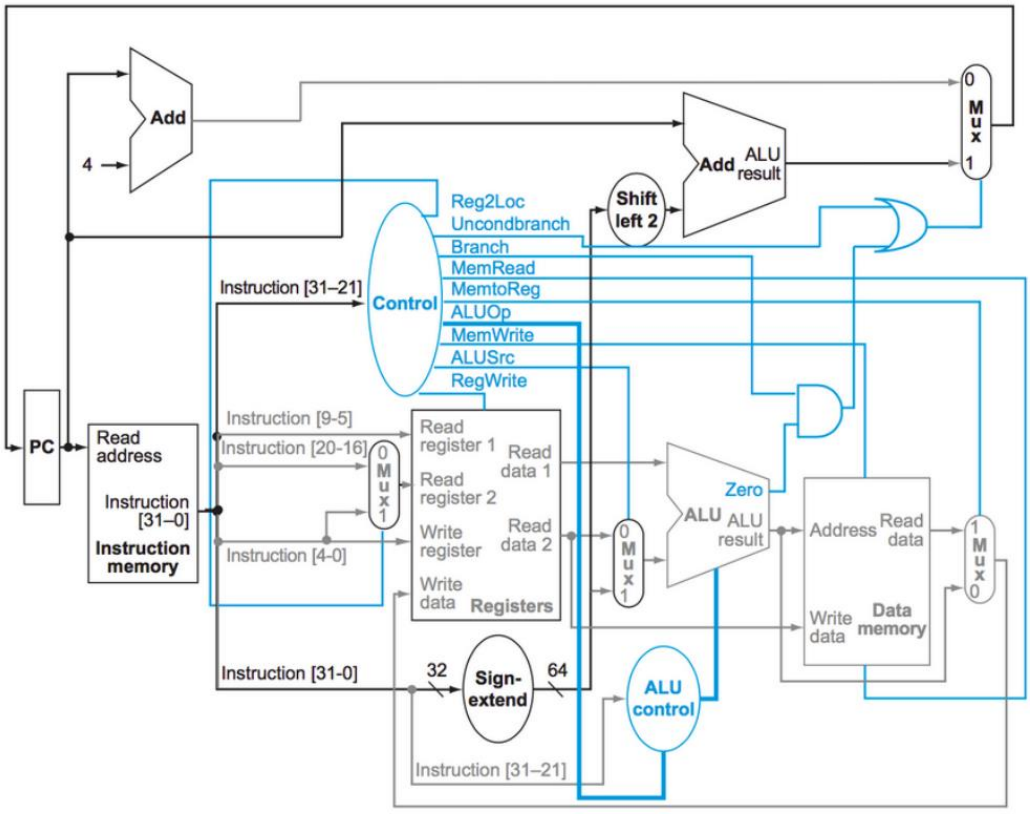


Figure 1: **Single cycle ARM (LEGv8) processor for Lab3.** This is Figure 4-23 in the textbook.

In order to wire different component, we need to create different signals. Below are signal I create.

```
end component;
signal MUX64_out_PC : std_logic_vector(63 downto 0);
signal MUX64_out_ALU : std_logic_vector(63 downto 0);
signal MUX64_out_WD : std_logic_vector(63 downto 0);
signal MUX5_out: std_logic_vector(4 downto 0);
signal PC_out : std_logic_vector(63 downto 0);
signal add_out_PC : std_logic_vector(63 downto 0);
signal IMEM_out : std_logic_vector(31 downto 0);
signal ReadData01_out : std_logic_vector(63 downto 0);
signal ReadData02_out : std_logic_vector(63 downto 0);
signal SignExtend_out : std_logic_vector(63 downto 0);
signal Shiftleft2_out : std_logic_vector(63 downto 0);
signal add_out_shiftleft2 : std_logic_vector(63 downto 0);
signal ALU_result : std_logic_vector(63 downto 0);
signal ALU_zero : STD_LOGIC;
signal ALUControl_out : std_logic_vector(3 downto 0);
signal DMEM_out : std_logic_vector(63 downto 0);
signal Reg2Loc : STD_LOGIC;
signal Regwrite : STD_LOGIC;
signal ALUSrc : STD_LOGIC;
signal CBranch : STD_LOGIC;
signal UBranch : STD_LOGIC;
signal ALUOp: std_logic_vector(1 downto 0);
signal MemWrite : STD_LOGIC;
signal MemRead : STD_LOGIC;
signal MemtoReg : STD_LOGIC;
signal OR2_out : STD_LOGIC;
signal AND2_out : STD_LOGIC;
signal DEBUG_TMP_REGS_out :  STD_LOGIC_VECTOR(64*4 - 1 downto 0);
signal DEBUG_SAVED_REGS_out:  STD_LOGIC_VECTOR(64*4 - 1 downto 0);
signal DEBUG_MEM_CONTENTS_out :  STD_LOGIC_VECTOR(64*4 - 1 downto 0);
```

② Introduction

However, in the lab3 we cannot simply wire each component. Because before lab3, we create components under simple condition which cannot be directly used for lab3. We need to add some new functionality for some components.

First of all, we need to change the ALU control and CPU control to generate new operation code for the instructions which are not used in the previous lab. For example, I-type, D-type and CBZ instructions.

This is my CPU control for lab3:

E:/Modelism/ee126/lab3/cpucontrol.vhd - Default

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity CPUControl is
-- Functionality should match the truth table shown in Figure 4.22 of the textbook, inlcuding the
--    output 'X' values.
-- The truth table in Figure 4.22 omits the unconditional branch instruction:
--    UBranch = '1'
--    MemWrite = RegWrite = '0'
--    all other outputs = 'X'
port(Opcode   : in  STD_LOGIC_VECTOR(10 downto 0);
     Reg2Loc  : out STD_LOGIC;
     CBranch  : out STD_LOGIC;  --conditional
     MemRead  : out STD_LOGIC;
     MemtoReg : out STD_LOGIC;
     MemWrite : out STD_LOGIC;
     ALUSrc   : out STD_LOGIC;
     RegWrite : out STD_LOGIC;
     UBranch  : out STD_LOGIC; -- This is unconditional
     ALUOp    : out STD_LOGIC_VECTOR(1 downto 0)
);
end CPUControl;

ARCHITECTURE Behavl OF CPUControl IS
BEGIN
        PROCESS (opcode)
        BEGIN
                regWrite <= '0';
```

E:/Modelism/ee126/lab3/cpucontrol.vhd - Default

```
29  if(Opcode(10 downto 5)="000101")then
30        Reg2Loc  <='X';
31        CBranch  <='X';
32        MemRead  <='X';
33        MemtoReg <='X';
34        MemWrite <='0';
35        ALUSrc   <='X';
36        RegWrite <='0';
37        UBranch  <='1';
38        ALUOp    <="XX";
39
40           ELSif(Opcode(10 downto 0)="11111000010")then
41        Reg2Loc  <='X';
42        CBranch  <='0';
43        MemRead  <='1';
44        MemtoReg <='1';
45        MemWrite <='0';
46        ALUSrc   <='1';
47        RegWrite <='1';
48        UBranch  <='0';
49        ALUOp    <="00";
50         ELSif(Opcode(10 downto 0)="11111000000")then
51        Reg2Loc  <='1';
52        CBranch  <='0';
53        MemRead  <='0';
54        MemtoReg <='X';
55        MemWrite <='1';
56        ALUSrc   <='1';
57        RegWrite <='0';
58        UBranch  <='0';
59        ALUOp    <="00";
```

E:/Modelism/ee126/lab3/cpucontrol.vhd - Default

```
60  ELSif(Opcode(10 downto 3)="10110100")then
61        Reg2Loc  <='1';
62        CBranch  <='1';
63        MemRead  <='0';
64        MemtoReg <='X';
65        MemWrite <='0';
66        ALUSrc   <='0';
67        RegWrite <='0';
68        UBranch  <='0';
69        ALUOp    <="01";
70  ELSif(Opcode(10 downto 3)="10110101")then
71        Reg2Loc  <='1';
72        CBranch  <='1';
73        MemRead  <='0';
74        MemtoReg <='X';
75        MemWrite <='0';
76        ALUSrc   <='0';
77        RegWrite <='0';
78        UBranch  <='0';
79        ALUOp    <="01";
80  ELSif(Opcode(10)='1' AND Opcode(7 downto 4)="0101" and Opcode(2 downto 0) ="000")then
81        Reg2Loc  <='0';
82        CBranch  <='0';
83        MemRead  <='0';
84        MemtoReg <='0';
85        MemWrite <='0';
86        ALUSrc   <='0';
87        RegWrite <='1';
88        UBranch  <='0';
89        ALUOp    <="10";
```

```
Ln#
 90   ELSif(Opcode(10)='1' AND Opcode(7 downto 5)="100" and Opcode(2 downto 1) ="00")then
 91        Reg2Loc  <='0';
 92        CBranch  <='0';
 93        MemRead  <='0';
 94        MemtoReg <='0';
 95        MemWrite <='0';
 96        ALUSrc   <='1';
 97        RegWrite <='1';
 98        UBranch  <='0';
 99        ALUOp    <="10";
100  ELSif(Opcode(10 downto 2) ="1101001l0")then
101        Reg2Loc  <='0';
102        CBranch  <='0';
103        MemRead  <='0';
104        MemtoReg <='0';
105        MemWrite <='0';
106        ALUSrc   <='1';
107        RegWrite <='1';
108        UBranch  <='0';
109        ALUOp    <="10";
110  else
111     Reg2Loc  <='0';
112        CBranch  <='0';
113        MemRead  <='0';
114        MemtoReg <='0';
115        MemWrite <='0';
116        ALUSrc   <='0';
117        RegWrite <='0';
118        UBranch  <='0';
119        ALUOp    <="11";
120  end if;
```

This is my ALU control for lab3:

```
E:/Modelism/ee126/lab3/alucontrol.vhd - Default
Ln#
11        Opcode    : in  STD_LOGIC_VECTOR(10 downto 0);
12        Operation : out STD_LOGIC_VECTOR(3 downto 0)
13       );
14   end ALUControl;
15
16   architecture behavl of ALUControl is
17   begin
18   col : process(ALUOp,Opcode)
19   begin
20   if Opcode="10010001000" then
21      operation <= "0010";
22
23   elsif Opcode="10001011000" then
24       operation <= "0010";
25   elsif Opcode = "11010001000" then
26      operation <= "0110";
27   elsif Opcode = "10001010000" then
28      operation <= "0000";
29   elsif Opcode = "10101010000" then
30      operation <= "0001";
31   elsif Opcode = "11111000000" then
32      operation <= "0010";
33   elsif Opcode = "11111000010" then
34      operation <= "0010";
35   elsif Opcode = "11001011000" then
36      operation <= "0110";
37   else
38       operation <= "1111";
39   end if;
40   end process;
41   end behavl;
```

Second, we need change the signextend to get the constant for all types of instructions. We know that the constant in each type is in the different position. So we need to figure out which type of the instruction is and then extend the sign of the constant in the instruction.

CORE INSTRUCTION FORMATS

R	opcode	Rm	shamt	Rn	Rd
	31 21 20	16 15	10 9	5 4	0
I	opcode	ALU_immediate		Rn	Rd
	31 22 21		10 9	5 4	0
D	opcode	DT_address	op	Rn	Rt
	31 21 20	12 11	10 9	5 4	0
B	opcode	BR_address			
	31 26 25				0
CB	Opcode	COND_BR_address			Rt
	31 24 23			5 4	0
IW	opcode	MOV_immediate			Rd
	31 21 20			5 4	0

So the signextend in the lab3 is:

E:/Modelism/ee126/lab3/signextend.vhd - Default

```
architecture behv1 of SignExtend is
begin
process(x)
 begin
      if x(28 downto 22) = "1000100" then
           if x(21) = '0' then
              y <= x"0000000000000" & x(21 downto 10);
              else
              y <= x"FFFFFFFFFFFFF" & x(21 downto 10);
           end if;
       elsif   x(31 downto 27)= "11111" then

              if x(20) = '0'  then
              y <= "00000000000000000000000000000000000000000000000000000000000000000000" & x(20 downto 12);
              else
               y <= "11111111111111111111111111111111111111111111111111111111111111111111" & x(20 downto 12) ;
              end if;
      elsif  x(31 downto 24) = "10110100" then

             if x(23) = '0' then
               y <= "00000000000000000000000000000000000000000000000000" & x(23 downto 5) ;
               ELSE
                y <= "11111111111111111111111111111111111111111111111111" & x(23 downto 5);
              end if;

       elsif (x(31 downto 26) = "000101") then

                if x(25) = '0' then
                y <= "00000000000000000000000000000000000000000000000000" & x(25 downto 0);
                ELSE
                 y <= "11111111111111111111111111111111111111111111111111" & x(25 downto 0);
                end if;
                y <= "11111111111111111111111111111111111111111111111111" & x(25 downto 0);
              end if;
       else
                  y <= x"0000_0000_0000_0001";
end if;
end process;
end behv1;
```

③ Test Result

For the test part, we need to write the testbench, for this lab we use one test bench to finish all three test, but we use three different IMEM which have different group of instructions. Blew is my test bench for the lab3:

E:/Modelism/ee126/lab3/Singlecyclecpu_tb.vhd - Default

```
Ln#
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4
5   entity Singlecyclecpu_tb is
6   end Singlecyclecpu_tb;
7
8   architecture Singlecyclecpu_tb of Singlecyclecpu_tb is
9
10  component SingleCycleCPU
11      port(clk :in STD_LOGIC;
12  rst :in STD_LOGIC;
13  DEBUG_PC : out STD_LOGIC_VECTOR(63 downto 0);
14  DEBUG_INSTRUCTION : out STD_LOGIC_VECTOR(31 downto 0);
15  DEBUG_TMP_REGS : out STD_LOGIC_VECTOR(64*4 - 1 downto 0);
16  DEBUG_SAVED_REGS : out STD_LOGIC_VECTOR(64*4 - 1 downto 0);
17  DEBUG_MEM_CONTENTS : out STD_LOGIC_VECTOR(64*4 - 1 downto 0)
18  );
19  end component;
20  signal rst : std_logic;
21  signal clk :   STD_LOGIC:='0';
22  signal DEBUG_MEM_CONTENTS : std_logic_vector(64*4 - 1 downto 0);
23  signal DEBUG_TMP_REGS : std_logic_vector(64*4 - 1 downto 0);
24  signal DEBUG_PC : STD_LOGIC_VECTOR(63 downto 0);
25  signal DEBUG_INSTRUCTION : STD_LOGIC_VECTOR(31 downto 0);
26
```

```
begin
uut: SingleCycleCPU port map (clk => clk, rst=>rst, DEBUG_MEM_CONTENTS => DEBUG_MEM_CONTENTS,
                              DEBUG_TMP_REGS => DEBUG_TMP_REGS,DEBUG_PC => DEBUG_PC ,DEBUG_INSTRUCTION => DEBUG_INSTRUCTION);


stim_proc: process
begin
     rst <= '1';

    wait for 45 ns;
    rst <= '0';
wait;
end process;
  clk <= not clk after 50 ns;
end;
```

A. Computation Test

Test code:

```
ADDI X10, X11, 1
ADDI X10, X11, 2
ADDI X9, X9, 1
SUBI X9, X9, 1
ADD  X10, X9, X11
-------- Assembly Translation ---------
16'h0000:  out = 32'b10010001000000000000010101101010; // ADDI X10, X11, 1
16'h0001:  out = 32'b10010001000000000000100101101010; // ADDI X10, X11, 2
16'h0002:  out = 32'b10010001000000000000010100101001; // ADDI X9, X9, 1
16'h0003:  out = 32'b11010001000000000000010100101001; // SUBI X9, X9, 1
16'h0004:  out = 32'b10001011000010110000000100101010; // ADD  X10, X9, X11
```

Initial value of X9,X10,X11:

```
Registers(9) <= x"0000_0000_0000_0000"; --X9
Registers(10) <= x"0000_0000_0000_0001"; --X10
Registers(11) <= x"0000_0000_0000_0002"; --X11
Registers(12) <= x"0000_0000_0000_0004"; --X12
```

Test result:
The figure below just shows the result of six signals:
Clk,rst,
DEBUG_MEM_CONTENTS,
DEBUG_TMP_REGS,
DEBUG_PC,
DEBUG_INSTRUCTION
The complete waveform will be stored separately in the file.

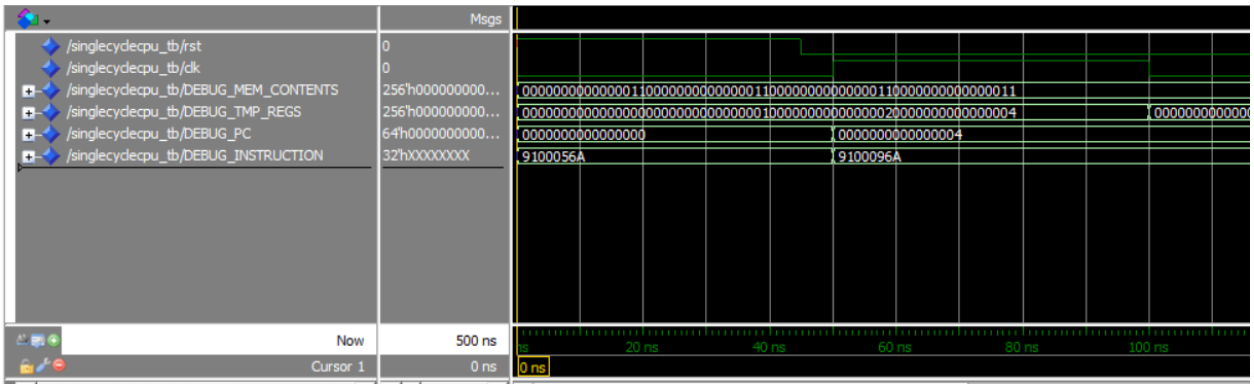
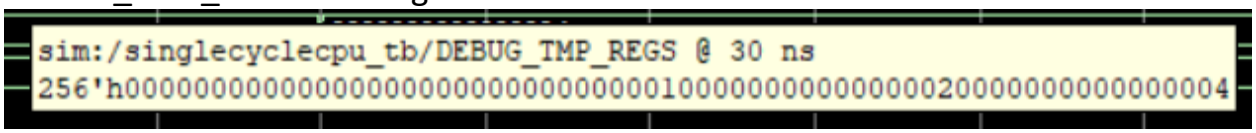


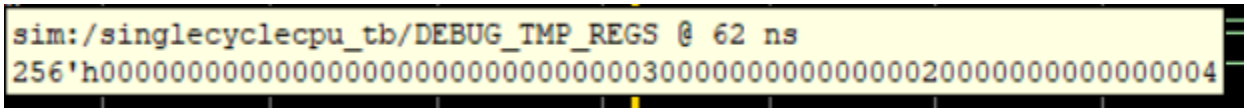
Figure1 computation test (0-100ns)

During this time, the instruction in IMEM = 910056A is excuted, we can see that

DEBUG_TMP_REGS is changed from



To



Because

```
DEBUG_SAVED_REGS <= Registers(19)&Registers(20)&Registers(21)&Registers(22);
DEBUG_TMP_REGS   <= Registers(9)&Registers(10)&Registers(11)&Registers(12);
```

So X10 is changed to 3(X11+1).

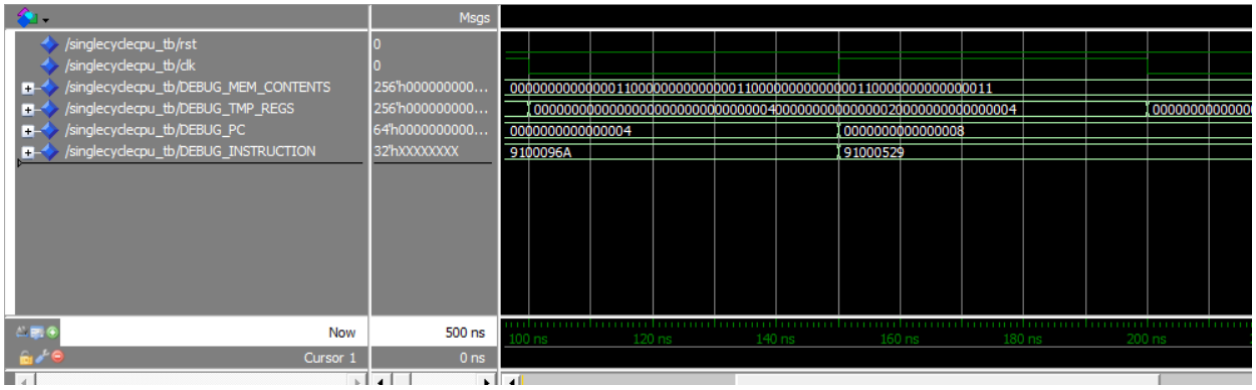


Figure2 computation test (100-200ns)

During this time, the instruction in IMEM = 9100096A is excuted, we can see that DEBUG_TMP_REGS is changed from

```
sim:/singlecyclecpu_tb/DEBUG_TMP_REGS @ 140 ns
256'h0000000000000000000000000000000300000000000000020000000000000004
```

To

```
sim:/singlecyclecpu_tb/DEBUG_TMP_REGS @ 171 ns
256'h0000000000000000000000000000000400000000000000020000000000000004
```

So X10 is changed to 4(X11+2).

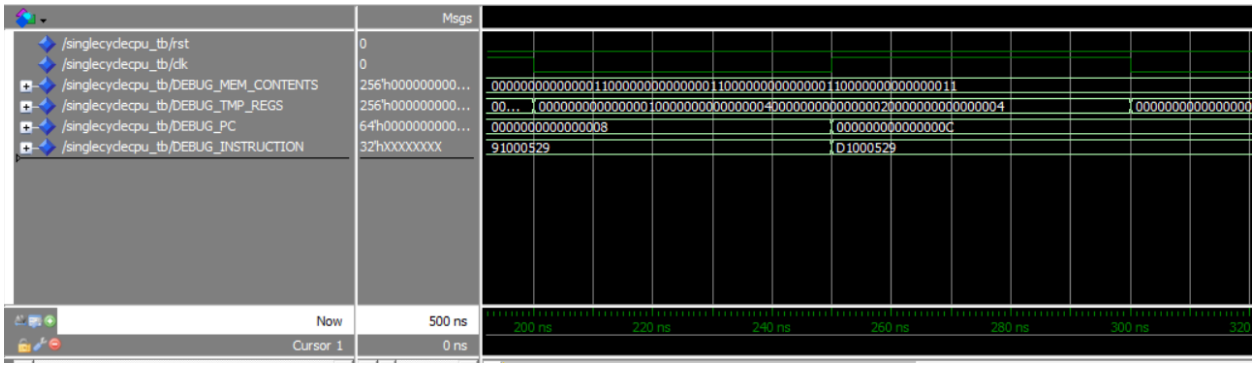


Figure3 computation test (200-300ns)

During this time, the instruction in IMEM = 91000529 is executed, we can see that

DEBUG_TMP_REGS is changed from

```
sim:/singlecyclecpu_tb/DEBUG_TMP_REGS @ 233 ns
256'h0000000000000000000000000000000400000000000000020000000000000004
```

To

```
sim:/singlecyclecpu_tb/DEBUG_TMP_REGS @ 287 ns
256'h0000000000000001000000000000000400000000000000020000000000000004
```

So X9 is changed to 1(X9+1).

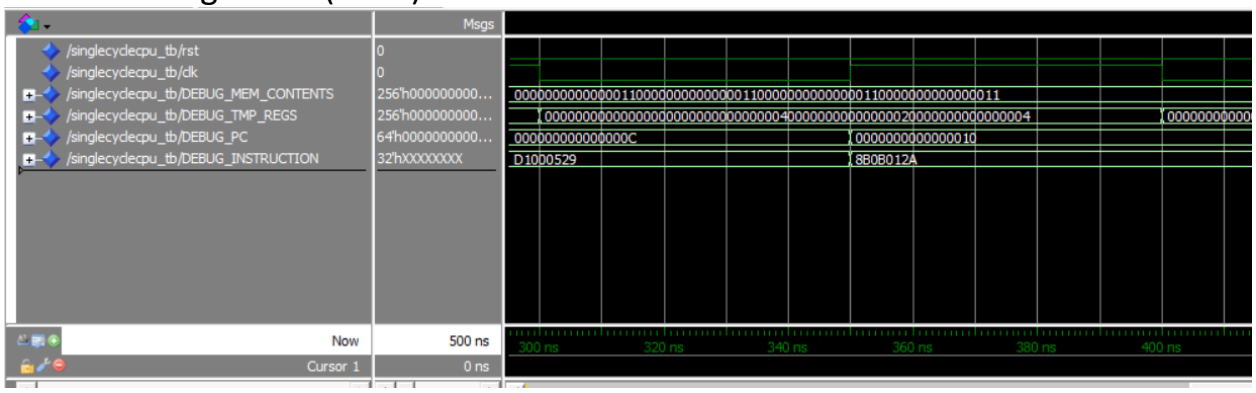


Figure4 computation test (300-400ns)

During this time, the instruction in IMEM = D1000529 is executed, we can see that

DEBUG_TMP_REGS is changed from

```
sim:/singlecyclecpu_tb/DEBUG_TMP_REGS @ 330 ns
256'h0000000000000001000000000000000400000000000000020000000000000004
```

To

```
sim:/singlecyclecpu_tb/DEBUG_TMP_REGS @ 380 ns
256'h0000000000000000000000000000000400000000000000020000000000000004
```

So X9 is changed to 0(X9-1).

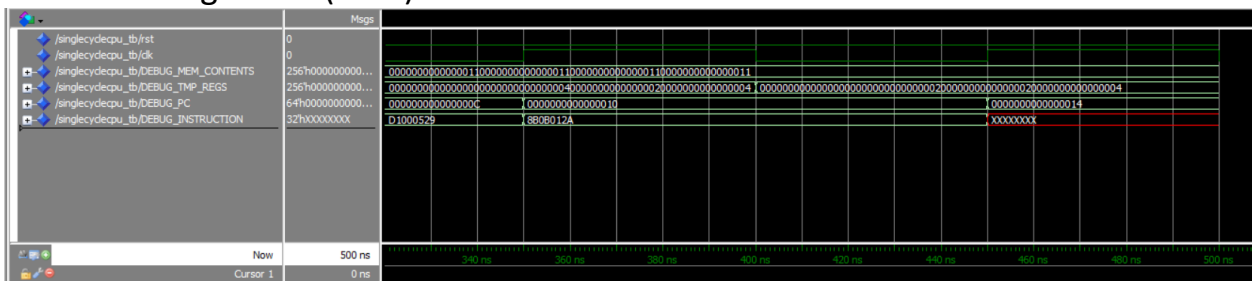


Figure5 computation test (400-500ns)

During this time, the instruction in IMEM = 8B0B012A is executed, we can see that

DEBUG_TMP_REGS is changed from

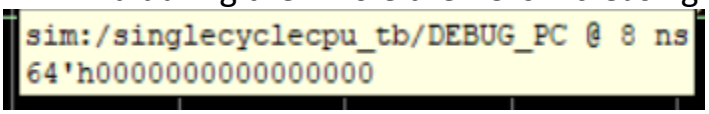
```
sim:/singlecyclecpu_tb/DEBUG_TMP_REGS @ 434 ns
256'h0000000000000000000000000000000400000000000000020000000000000004
```

To

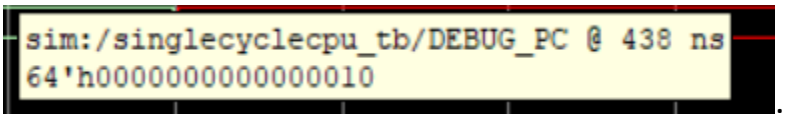
```
sim:/singlecyclecpu_tb/DEBUG_TMP_REGS @ 467 ns
256'h0000000000000000000000000000000200000000000000020000000000000004
```

So X10 is changed to 2(X9+x11).

And during the whole the PC is increasing each cycle, from



To



.

Because there is no fifth instruction in the IMEM, so DEBUG_INSTRUCTION become xxxxxxxx after 450 ns and the result of the test is correct.

B. Communication Test

Test code:

```
STUR X10, [X11, 0]
LDUR X10, [X9, 0]
----- Assembly Trasnlation ----
16'h0000:  out = 32'b11111000000000000000000101101010; // STUR X10, [X11, 0]
16'h0001:  out = 32'b11111000010000000000000100101010; // LDUR X10, [X9, 0]
```

Test result:

The figure below just shows the result of six signals:

Clk,rst,

DEBUG_MEM_CONTENTS,

DEBUG_TMP_REGS,

DEBUG_PC,

DEBUG_INSTRUCTION

The complete waveform will be stored separately in the file.

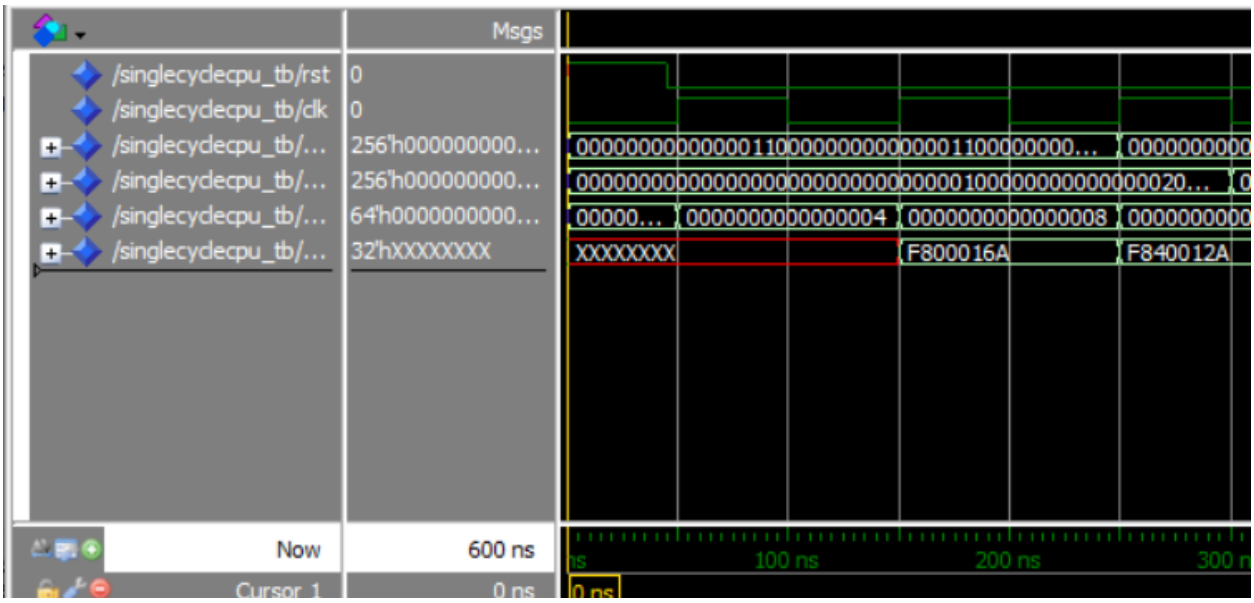
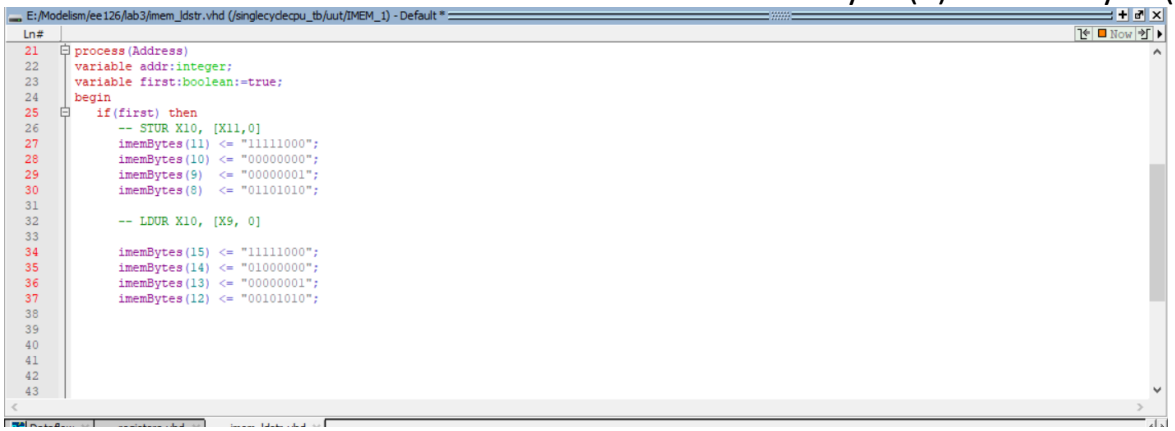


Figure6 communication test(0-250ns)

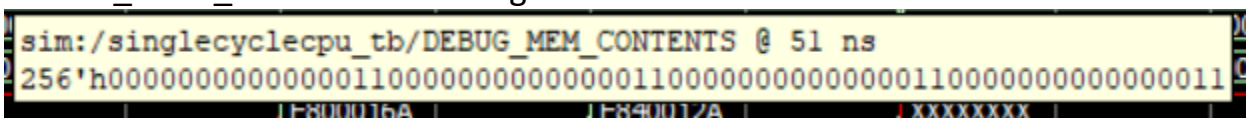
Because there is no instruction in the IMEM from imemBytes(0) to imemBytes(7):



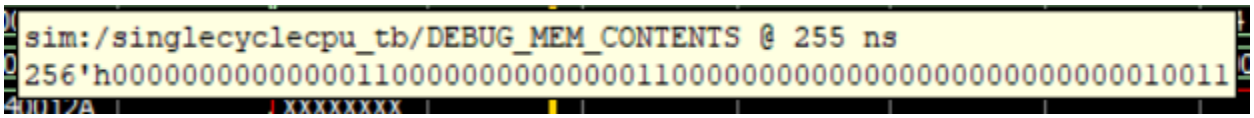
So, the DEBUG_INSTRUCTION is xxxxxxx before 150 ns

During time t =150ns to 200 ns, the instruction in IMEM = F800 016A is executed, we can see that

DEBUG_MEM_CONTENTS is changed from



To



Because

```
-- Conntect the signals that will be used for testing
DEBUG_MEM_CONTENTS <=
    dmemBytes(31) & dmemBytes(30) & dmemBytes(29) & dmemBytes(28)&  --DMEM(28)
    dmemBytes( 27) & dmemBytes( 26) & dmemBytes(25) & dmemBytes(24) & --DMEM(24)
    dmemBytes( 23) & dmemBytes( 22) & dmemBytes( 21) & dmemBytes( 20) & --DMEM(20)
    dmemBytes( 19) & dmemBytes( 18) & dmemBytes( 17) & dmemBytes( 16) & --DMEM(16)
    dmemBytes(15) & dmemBytes(14) & dmemBytes(13) & dmemBytes(12)&  --DMEM(12)
    dmemBytes( 11) & dmemBytes( 10) & dmemBytes(9) & dmemBytes(8) & --DMEM(8)
    dmemBytes( 7) & dmemBytes( 6) & dmemBytes( 5) & dmemBytes( 4) & --DMEM(4)
    dmemBytes( 3) & dmemBytes( 2) & dmemBytes( 1) & dmemBytes( 0) ; --DMEM(0)
```

So we can know dmemBytes(2) is changed to 1.

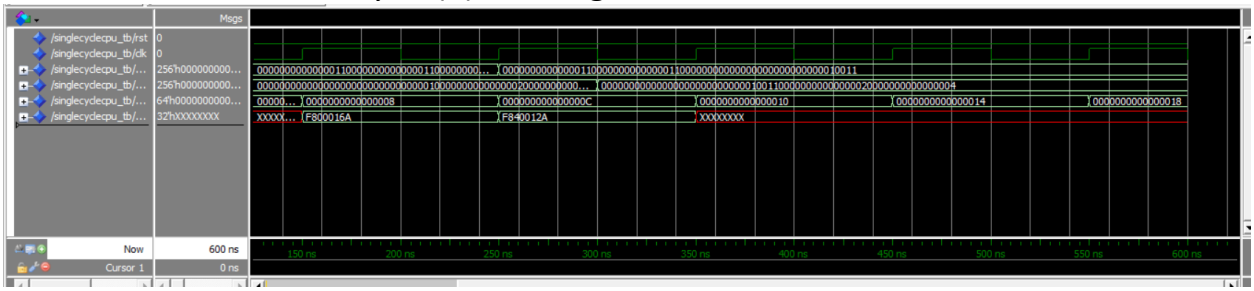
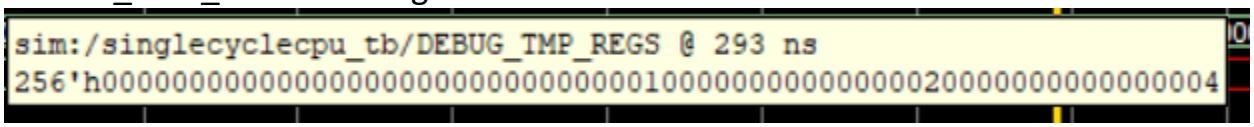
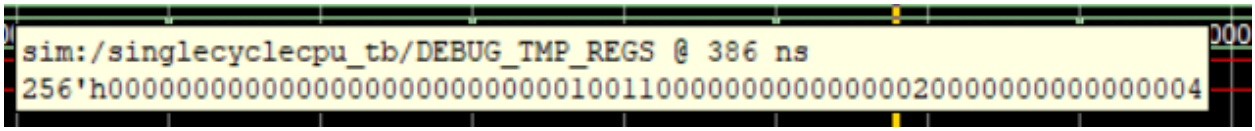


Figure7 communication test(250ns-600ns)

During this time, the instruction in IMEM = F840 012Ais executed, we can see that
DEBUG_TMP_REGS is changed from



To



So X10 is loaded date from dmemBytes(0).

C. Final Test

Test code:

```
 ADDI X9, X9, 1
ADD  X10,X9,X11
STUR X10, [X11,0]
LDUR X12, [X11, 0]
CBZ X9, 2
B 3
ADD X9, X10, X11
ADDI  X9, X9, 1
ADD   X21, X10, X9
```

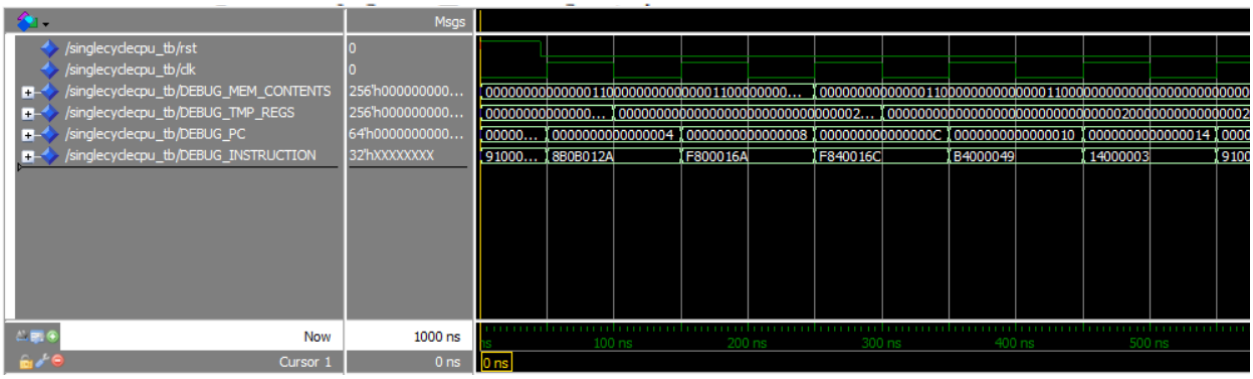
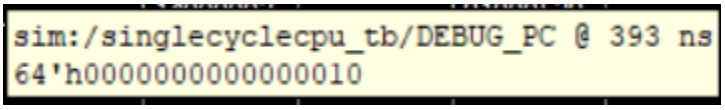


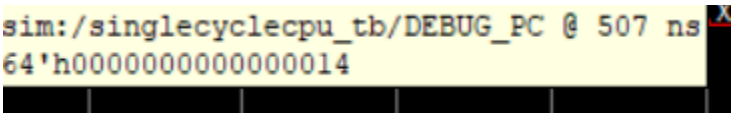
Figure8 final test(0-500ns)

For the 0-500ns, the figure 8 shows the result of final test, Since four of these instructions have already been discussed above, they are not discussed.

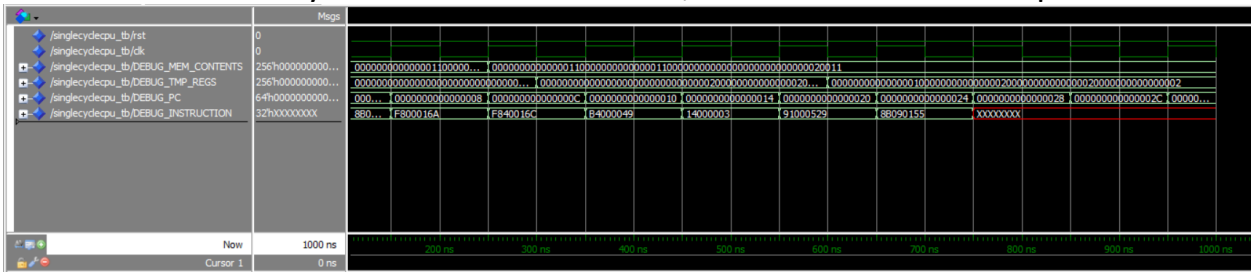
But we should notice that address of PC is changed from



To



Means not branch by the instruction “CBZ X9,2” because X9 is not equal to 0.



Figre9 final test(500-1000ns)

During this time t=450ns to 550ns, the instruction in IMEM = 1400 0003is executed, we can see that

DEBUG_PC is changed from

```
sim:/singlecyclecpu_tb/DEBUG_PC @ 515 ns
64'h0000000000000014
```

To

```
sim:/singlecyclecpu_tb/DEBUG_PC @ 621 ns
64'h0000000000000020
```

Hex 20 = 32

Hex 14 = 20

So the address is increase 12 not 4. So the instruction “B 3”.

And we can see next instruction we execute is 9100 0529 which is

E:/Modelism/ee126/lab3/imem_p1.vhd (/singlecyclecpu_tb/uut/IMEM_1) - Default

```
Ln#
62
63          -- ADD X9, X10, X11
64          imemBytes(27) <= "10001011";
65          imemBytes(26) <= "00001011";
66          imemBytes(25) <= "00000001";
67          imemBytes(24) <= "01001001";
68          -- ADD X9, X10, X11
69          imemBytes(31) <= "10001011";
70          imemBytes(30) <= "00001011";
71          imemBytes(29) <= "00000001";
72          imemBytes(28) <= "01001001";
73
74          -- ADDI  X9, X9, 1
75          imemBytes(35) <= "10010001";
76          imemBytes(34) <= "00000000";
77          imemBytes(33) <= "00000101";
78          imemBytes(32) <= "00101001";
79
80          -- ADD   X21, X10, X9
81          imemBytes(39) <= "10001011";
82          imemBytes(38) <= "00001001";
83          imemBytes(37) <= "00000001";
84          imemBytes(36) <= "01010101";
85
86
```

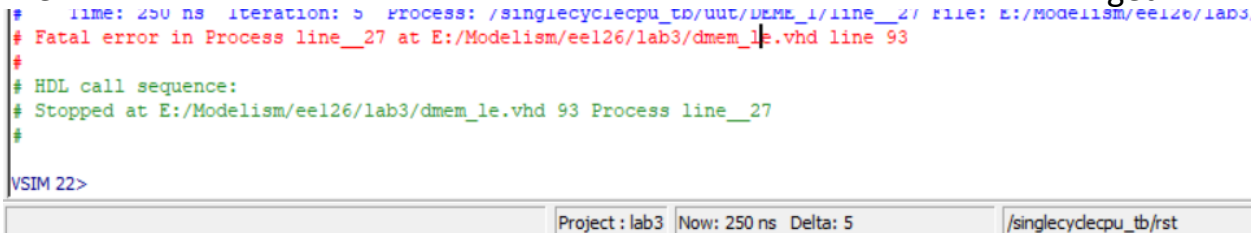
So we skipped two instructions. The result is correct.

④ Question

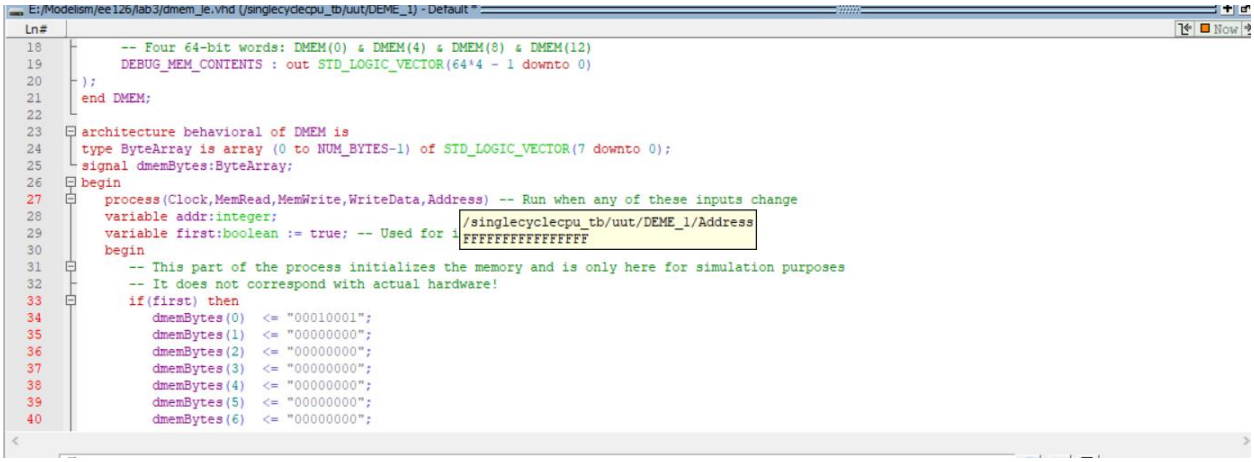
If we change X9=-1, then we execute code:

```
STUR X10, [X11, 0]
LDUR X10, [X9, 0]
Registers(8) <= x"0000_0000_0000_0000"; --X8
Registers(9) <= x"ffff_ffff_ffff_ffff"; --X9
```

We get



The test is failed. In the DMEM file we have:



The address = “FFFFFFFFFFFFFFFF”, I the address is too big to find.