Lab Report #5

Name: Ruochen Duan
Student ID: 1405106

① Introduction

Lab5 asks us to add the data hazards forwarding and stalling by add the component of hazard detection unit, forwarding unit and some multiplier to control the IF/ID stage, program stall the instruction of LDUR and STUR to have the pipeline instruction by below figure.

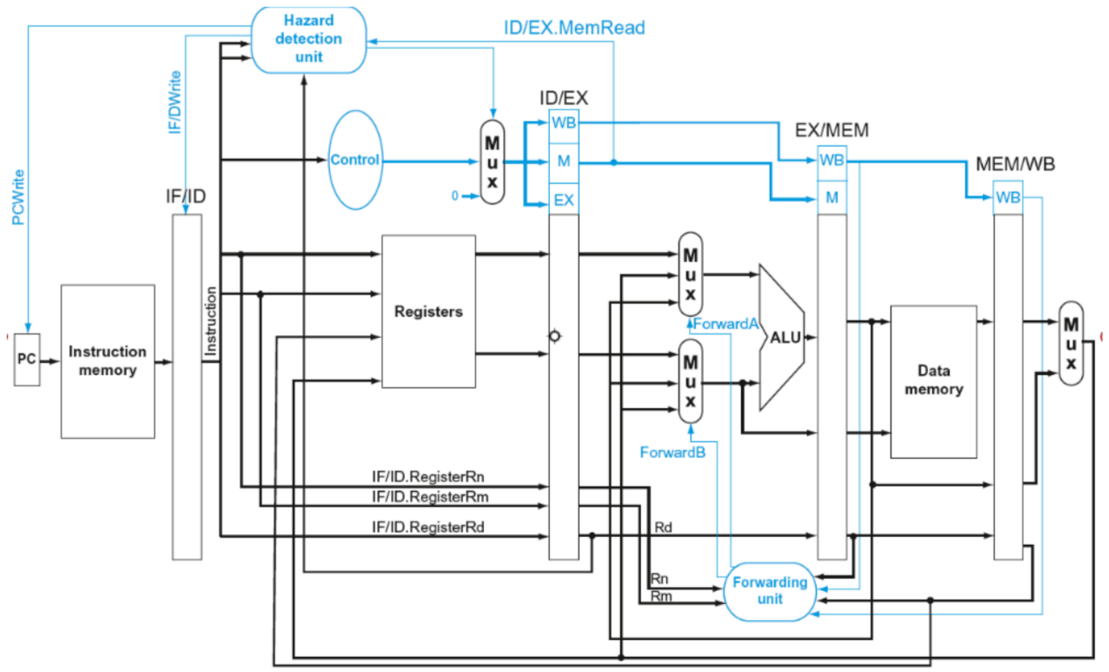
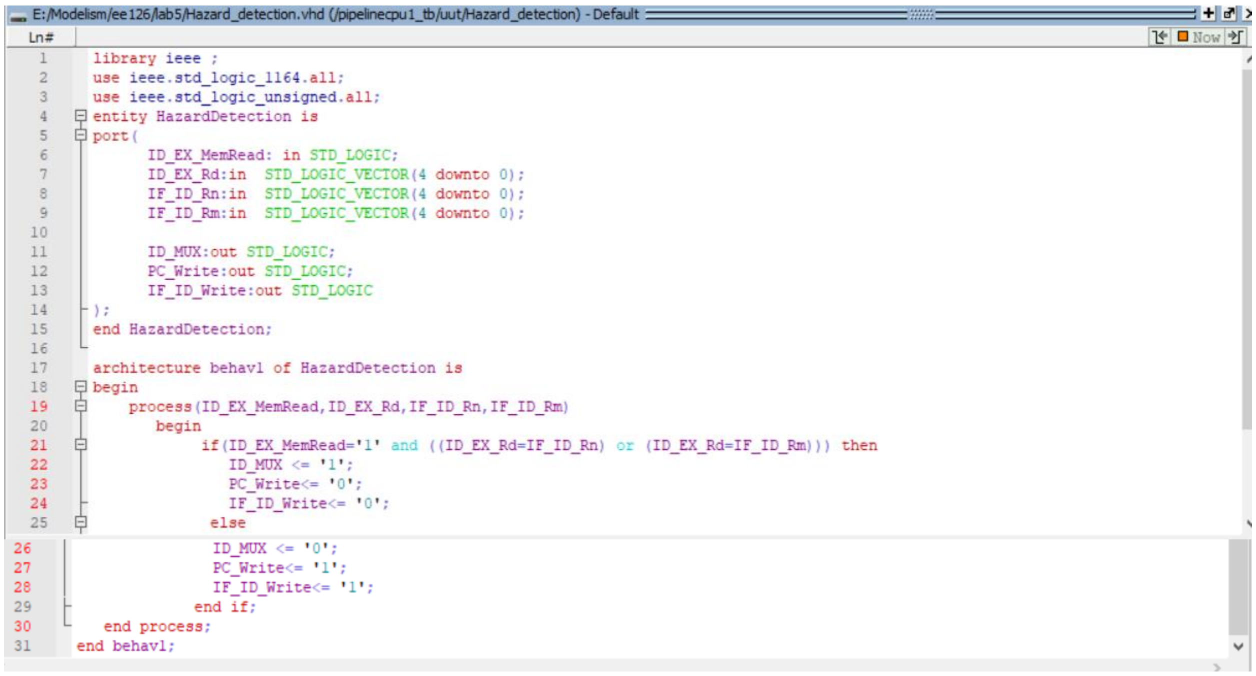


Figure 1: **Pipelined control overview showing the forwarding and hazard detection logic.** Note that this diagram leaves out some of the details of the full datapath. [Figure 4.7.4 from the textbook]

② Implementation

First, we need to create the forwarding unite and hazard detection unite. At the same time, we need to add new signal (IF/ID write_enable) to the register IF/ID. The code below is its implementation and signals(IF/ID_Rn and IF/ID_Rm) to register ID/EX to implement forwarding function.

Detection Unite:

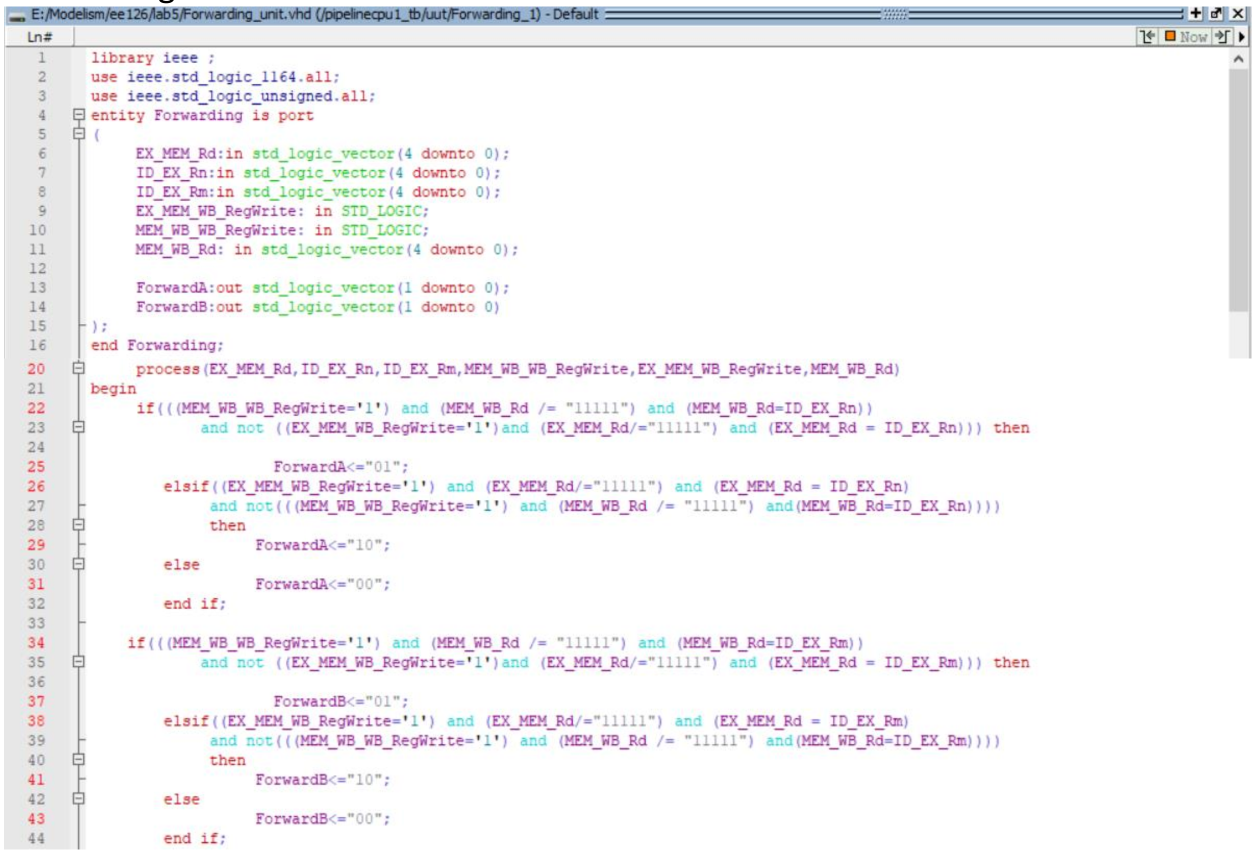

E:/Modelism/ee126/lab5/Hazard_detection.vhd (/pipelinecpu1_tb/uut/Hazard_detection) - Default
Ln#
Now

```vhdl
library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity HazardDetection is
port(
      ID_EX_MemRead: in STD_LOGIC;
      ID_EX_Rd:in  STD_LOGIC_VECTOR(4 downto 0);
      IF_ID_Rn:in  STD_LOGIC_VECTOR(4 downto 0);
      IF_ID_Rm:in  STD_LOGIC_VECTOR(4 downto 0);

      ID_MUX:out STD_LOGIC;
      PC_Write:out STD_LOGIC;
      IF_ID_Write:out STD_LOGIC
);
end HazardDetection;

architecture behavl of HazardDetection is
begin
    process(ID_EX_MemRead,ID_EX_Rd,IF_ID_Rn,IF_ID_Rm)
       begin
            if(ID_EX_MemRead='1' and ((ID_EX_Rd=IF_ID_Rn) or (ID_EX_Rd=IF_ID_Rm))) then
               ID_MUX <= '1';
               PC_Write<= '0';
               IF_ID_Write<= '0';
             else
               ID_MUX <= '0';
               PC_Write<= '1';
               IF_ID_Write<= '1';
             end if;
    end process;
end behavl;
```

Forwarding Unite:


E:/Modelism/ee126/lab5/Forwarding_unit.vhd (/pipelinecpu1_tb/uut/Forwarding_1) - Default
Ln#
Now

```vhdl
library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity Forwarding is port
(
     EX_MEM_Rd:in std_logic_vector(4 downto 0);
     ID_EX_Rn:in std_logic_vector(4 downto 0);
     ID_EX_Rm:in std_logic_vector(4 downto 0);
     EX_MEM_WB_RegWrite: in STD_LOGIC;
     MEM_WB_WB_RegWrite: in STD_LOGIC;
     MEM_WB_Rd: in std_logic_vector(4 downto 0);

     ForwardA:out std_logic_vector(1 downto 0);
     ForwardB:out std_logic_vector(1 downto 0)
);
end Forwarding;
     process(EX_MEM_Rd,ID_EX_Rn,ID_EX_Rm,MEM_WB_WB_RegWrite,EX_MEM_WB_RegWrite,MEM_WB_Rd)
begin
     if(((MEM_WB_WB_RegWrite='1') and (MEM_WB_Rd /= "11111") and (MEM_WB_Rd=ID_EX_Rn))
            and not ((EX_MEM_WB_RegWrite='1')and (EX_MEM_Rd/="11111") and (EX_MEM_Rd = ID_EX_Rn))) then

                  ForwardA<="01";
        elsif((EX_MEM_WB_RegWrite='1') and (EX_MEM_Rd/="11111") and (EX_MEM_Rd = ID_EX_Rn)
             and not(((MEM_WB_WB_RegWrite='1') and (MEM_WB_Rd /= "11111") and(MEM_WB_Rd=ID_EX_Rn))))
             then
                  ForwardA<="10";
        else
                  ForwardA<="00";
        end if;

    if(((MEM_WB_WB_RegWrite='1') and (MEM_WB_Rd /= "11111") and (MEM_WB_Rd=ID_EX_Rm))
            and not ((EX_MEM_WB_RegWrite='1')and (EX_MEM_Rd/="11111") and (EX_MEM_Rd = ID_EX_Rm))) then

                  ForwardB<="01";
        elsif((EX_MEM_WB_RegWrite='1') and (EX_MEM_Rd/="11111") and (EX_MEM_Rd = ID_EX_Rm)
             and not(((MEM_WB_WB_RegWrite='1') and (MEM_WB_Rd /= "11111") and(MEM_WB_Rd=ID_EX_Rm))))
             then
                  ForwardB<="10";
        else
                  ForwardB<="00";
        end if;
```

Then I connect each devices shown in the first figure to run the simulation.

③ Test Result

The simulation we do is:

```
LDUR  X9, [XZR, 0]          11111000010000000000001111101001
ADD   X9, X9, X9            10001011000010010000000100101001
ADD   X10, X9, X9           10001011000010010000000100101010
SUB   X11, X10, X9          11001011000010010000000101001011
STUR  X11, [XZR, 8]         11111000000000001000001111101011
STUR  X11, [XZR, 16]        11111000000000010000001111101011
NOP                         filled with zero - see imem
NOP                         filled with zero - see imem
NOP                         filled with zero - see imem
NOP                         filled with zero - see imem
```

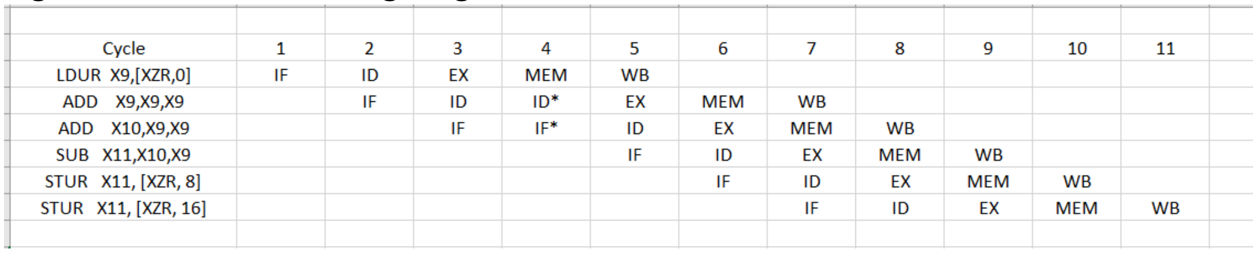
Register Values

```
$x9 =  1    $x19 = 0
$x10 = 0    $x20 = 0
$x11 = 0    $x21 = 0
$x12 = 0    $x22 = 0
```

DEME Contents

```
DMEM(0x0)   = 0x00000000
00 00 00 01
DMEM(0x8)   = 0x00000000
00 00 00 00
DMEM(0x10)  = 0x00000000
00 00 00 00
DMEM(0x18) = 0x00000000
00 00 00 00
DMEM(0x2E) = 0x00000000
00 00 00 01
```

Figure 1 shows the Timing diagram of the test.



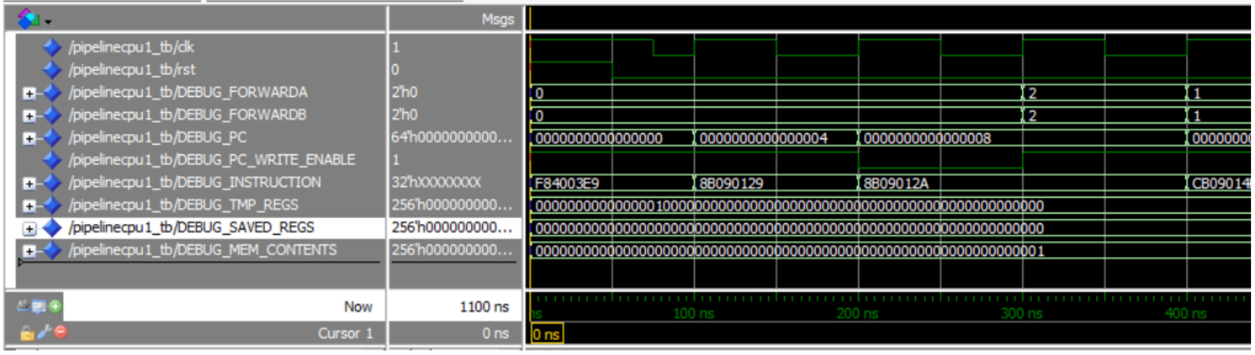
Cycle	1	2	3	4	5	6	7	8	9	10	11
LDUR X9,[XZR,0]	IF	ID	EX	MEM	WB						
ADD X9,X9,X9		IF	ID	ID*	EX	MEM	WB				
ADD X10,X9,X9			IF	IF*	ID	EX	MEM	WB			
SUB X11,X10,X9					IF	ID	EX	MEM	WB		
STUR X11, [XZR, 8]						IF	ID	EX	MEM	WB	
STUR X11, [XZR, 16]							IF	ID	EX	MEM	WB

Figure 1 Timing diagram

Test Result (I set each cycle equals to 100 ns.)

According to the Timing diagram, I will show and explain key events in the simulation (Complete figures of simulated waveforms will be attached separately in other files)

1) The stall in the cycle 3 and cycle 4.

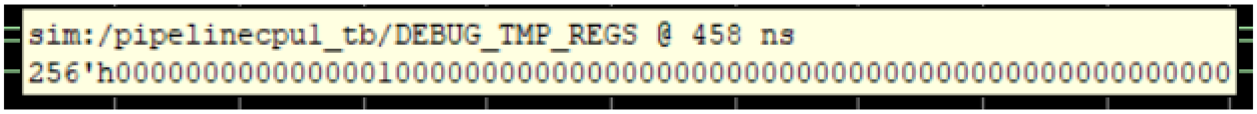


```
                                      Msgs
/pipelinecpu1_tb/clk                  1
/pipelinecpu1_tb/rst                  0
/pipelinecpu1_tb/DEBUG_FORWARDA       2'h0              0                                              2                  1
/pipelinecpu1_tb/DEBUG_FORWARDB       2'h0              0                                              2                  1
/pipelinecpu1_tb/DEBUG_PC             64'h0000000000... 0000000000000000  0000000000000004  0000000000000008            00000000
/pipelinecpu1_tb/DEBUG_PC_WRITE_ENABLE 1
/pipelinecpu1_tb/DEBUG_INSTRUCTION    32'hXXXXXXXX      F84003E9          8B090129          8B09012A                    CB09014
/pipelinecpu1_tb/DEBUG_TMP_REGS       256'h000000000... 00000000000000010000000000000000000000000000000000000000000000
/pipelinecpu1_tb/DEBUG_SAVED_REGS     256'h000000000... 00000000000000000000000000000000000000000000000000000000000000
/pipelinecpu1_tb/DEBUG_MEM_CONTENTS   256'h000000000... 00000000000000000000000000000000000000000000000000000000000001
Now      1100 ns     ns   100 ns   200 ns   300 ns   400 ns
Cursor 1    0 ns     0 ns
```

Figure2: stall in the cycle 3 and cycle4(200ns-400ns)

We can see the signal DEBUG_PC keep same, which is 8B09012A, in the cycle3 and cycle 4 means in these two cycle the CPU executes the same instruction. Because there is a data dependency(X9) between the first instruction and the second instruction. So the CPU has to wait until data in [XZR,0] is read from the DMEM. The DEBUG_PC_WRITE_ENABLE signal is '0' in 200ns to 300ns.

The result of first instruction:

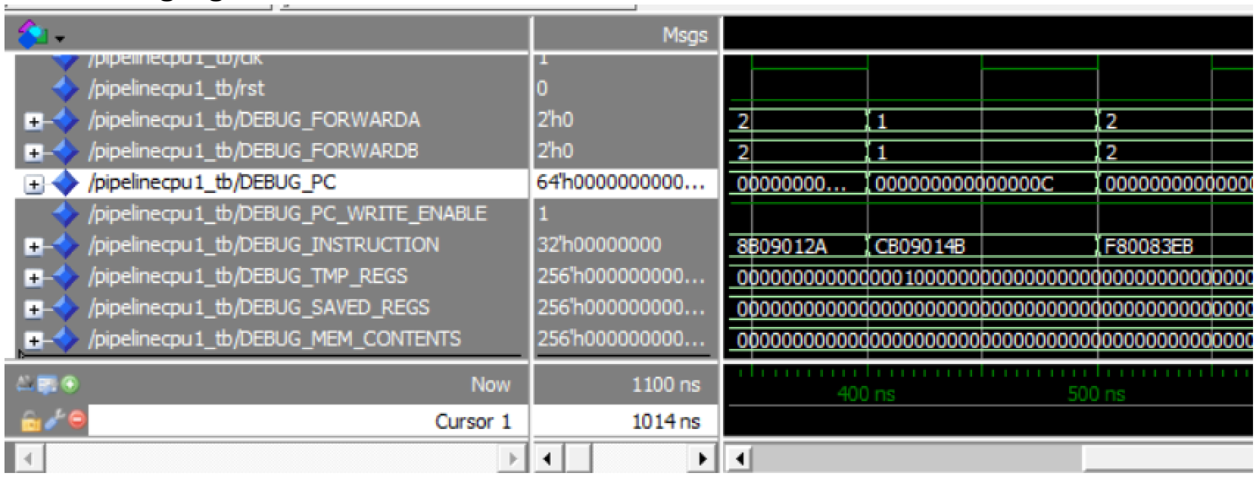


```
sim:/pipelinecpul_tb/DEBUG_TMP_REGS @ 458 ns
256'h0000000000000001000000000000000000000000000000000000000000000000
```

We can see that X9 is loaded data of DEME in address [XZR,0]. We cannot see signal change here, because the original value of X9 is same as the data of DEME in address [XZR,0].

2) Forwarding value X9 from WB stage of the first instruction to the EX-stage of the second instruction

Forwarding signal:



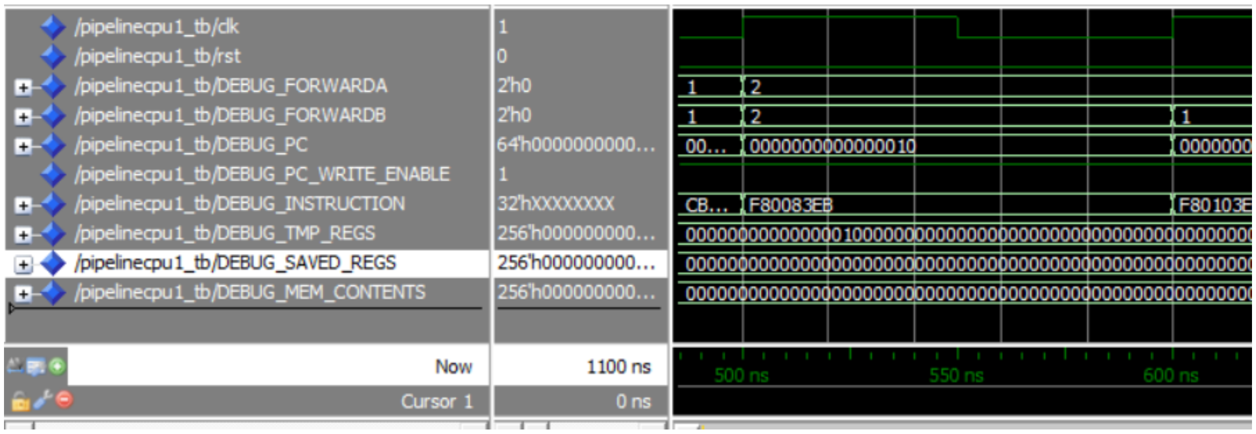
```
                                          Msgs
/pipelinecpu1_tb/clk                      1
/pipelinecpu1_tb/rst                      0
/pipelinecpu1_tb/DEBUG_FORWARDA           2'h0                 2    1    2
/pipelinecpu1_tb/DEBUG_FORWARDB           2'h0                 2    1    2
/pipelinecpu1_tb/DEBUG_PC                 64'h0000000000...    00000000...  000000000000000C  00000000000000
/pipelinecpu1_tb/DEBUG_PC_WRITE_ENABLE    1
/pipelinecpu1_tb/DEBUG_INSTRUCTION        32'h00000000         8B09012A  CB09014B  F80083EB
/pipelinecpu1_tb/DEBUG_TMP_REGS           256'h000000000...    0000000000000001000000000000000000000000000000000
/pipelinecpu1_tb/DEBUG_SAVED_REGS         256'h000000000...    0000000000000000000000000000000000000000000000000
/pipelinecpu1_tb/DEBUG_MEM_CONTENTS       256'h000000000...    0000000000000000000000000000000000000000000000000
Now                                       1100 ns              400 ns    500 ns
Cursor 1                                  1014 ns
```

Figure5: Forwarding X9 to third instruction (400ns to 500ns)

At 400ns(cycle5), We can see that the signal DEBUG_FORWARDA and signal DEBUG_FORWARD are “1” which means forward data from WB-stage MUX64.

3)Forwarding value X9 from EX stage of the second instruction to the EX-stage of the third instruction

Forwarding signal:

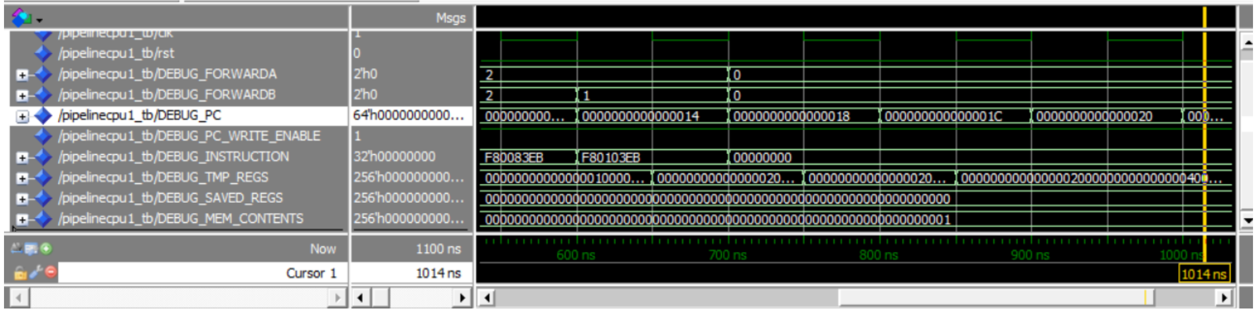


```
/pipelinecpu1_tb/clk                      1
/pipelinecpu1_tb/rst                      0
/pipelinecpu1_tb/DEBUG_FORWARDA           2'h0                 1  2
/pipelinecpu1_tb/DEBUG_FORWARDB           2'h0                 1  2    1
/pipelinecpu1_tb/DEBUG_PC                 64'h0000000000...    00...  0000000000000010  0000000
/pipelinecpu1_tb/DEBUG_PC_WRITE_ENABLE    1
/pipelinecpu1_tb/DEBUG_INSTRUCTION        32'hXXXXXXXX         CB...  F80083EB  F80103E
/pipelinecpu1_tb/DEBUG_TMP_REGS           256'h000000000...    0000000000000001000000000000000000000000000000000
/pipelinecpu1_tb/DEBUG_SAVED_REGS         256'h000000000...    0000000000000000000000000000000000000000000000000
/pipelinecpu1_tb/DEBUG_MEM_CONTENTS       256'h000000000...    0000000000000000000000000000000000000000000000000
Now                                       1100 ns              500 ns    550 ns    600 ns
Cursor 1                                  0 ns
```

Figure6: Forwarding X9 to third instruction (500ns to 600ns)

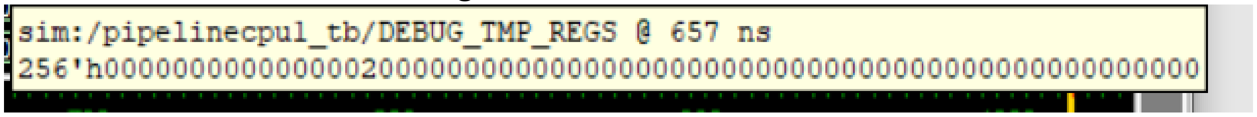
At 500ns(cycle6), We can see that the signal DEBUG_FORWARDA and signal DEBUG_FORWARD are “2” which means forward data from EX/MEM_ALU_result.

The result of second instruction:



```
/pipelinecpu1_tb/clk                   1
/pipelinecpu1_tb/rst                   0
/pipelinecpu1_tb/DEBUG_FORWARDA        2'h0
/pipelinecpu1_tb/DEBUG_FORWARDB        2'h0
/pipelinecpu1_tb/DEBUG_PC              64'h0000000000...
/pipelinecpu1_tb/DEBUG_PC_WRITE_ENABLE 1
/pipelinecpu1_tb/DEBUG_INSTRUCTION     32'h00000000
/pipelinecpu1_tb/DEBUG_TMP_REGS        256'h000000000...
/pipelinecpu1_tb/DEBUG_SAVED_REGS      256'h000000000...
/pipelinecpu1_tb/DEBUG_MEM_CONTENTS    256'h000000000...
Now      1100 ns
Cursor 1 1014 ns
```

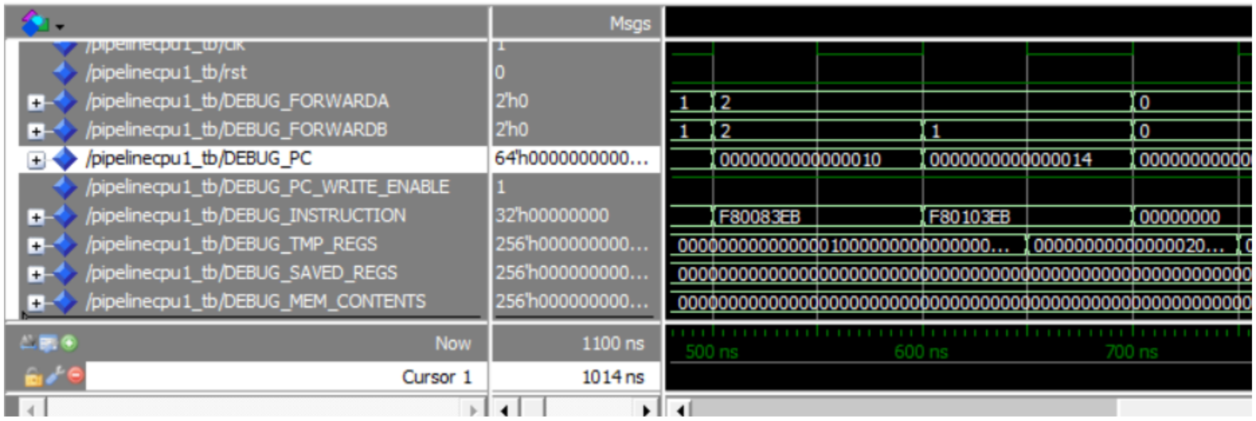
Figure7: 600ns to 700ns



```
sim:/pipelinecpu1_tb/DEBUG_TMP_REGS @ 657 ns
256'h0000000000000002000000000000000000000000000000000000000000000000
```

We can see that the value X9 becomes the sum of X9 and X9.

4) Forwarding value X10 from EX/MEM_ALU_result of third instruction to the EX-stage of the third instruction. And Forwarding value X9 from WB-stage MUX64 of second instruction to the EX-stage of the fourth instruction.

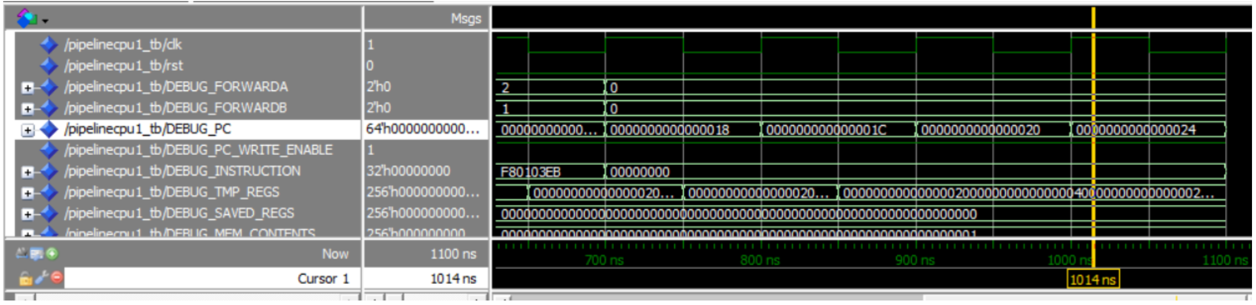


```
/pipelinecpu1_tb/clk                   1
/pipelinecpu1_tb/rst                   0
/pipelinecpu1_tb/DEBUG_FORWARDA        2'h0
/pipelinecpu1_tb/DEBUG_FORWARDB        2'h0
/pipelinecpu1_tb/DEBUG_PC              64'h0000000000...
/pipelinecpu1_tb/DEBUG_PC_WRITE_ENABLE 1
/pipelinecpu1_tb/DEBUG_INSTRUCTION     32'h00000000
/pipelinecpu1_tb/DEBUG_TMP_REGS        256'h000000000...
/pipelinecpu1_tb/DEBUG_SAVED_REGS      256'h000000000...
/pipelinecpu1_tb/DEBUG_MEM_CONTENTS    256'h000000000...
Now      1100 ns
Cursor 1 1014 ns
```

Figure8: 500ns to 700ns

We can see from figure 8 that at 600ns(cycle 7), signal DEBUG_FORWARDA is 2 means forward X9 from WB-stage MUX64 of second instruction to the EX-stage of the fourth instruction. And signal DEBUG_FORWARDB is changed to 1 means Forwarding value X10 from EX/MEM_ALU_result of third instruction to the EX-stage of the third instruction.

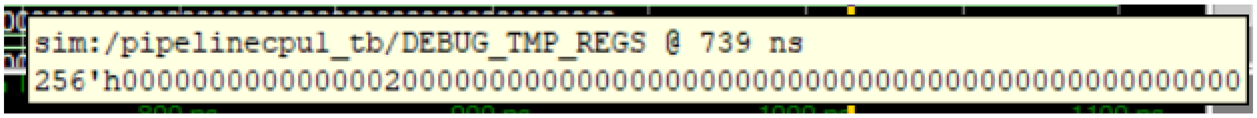
5) The result of fourth instruction:



```
                                      Msgs
/pipelinecpu1_tb/clk                  1
/pipelinecpu1_tb/rst                  0
/pipelinecpu1_tb/DEBUG_FORWARDA       2'h0              2 | 0
/pipelinecpu1_tb/DEBUG_FORWARDB       2'h0              1 | 0
/pipelinecpu1_tb/DEBUG_PC             64'h0000000000... 00000000000... | 0000000000000018 | 000000000000001C | 0000000000000020 | 0000000000000024
/pipelinecpu1_tb/DEBUG_PC_WRITE_ENABLE 1
/pipelinecpu1_tb/DEBUG_INSTRUCTION    32'h00000000      F80103EB | 00000000
/pipelinecpu1_tb/DEBUG_TMP_REGS       256'h000000000... 0000000000000020... | 0000000000000020... | 000000000000002000000000000000400000000000000002...
/pipelinecpu1_tb/DEBUG_SAVED_REGS     256'h000000000... 0000000000000000000000000000000000000000000000000000000000
/pipelinecpu1_tb/DEBUG_MEM_CONTENTS   256'h000000000    000000000000000000000000000000000000000000000000000000001
Now                                   1100 ns           700 ns  800 ns  900 ns  1000 ns  1100 ns
Cursor 1                              1014 ns           1014 ns
```

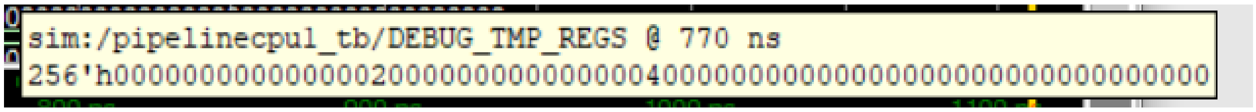
Figure8: 700nsns to 800ns

We can see from figure that at t =750ns, the signal DEBUG_TMP_REGS is changed from



```
sim:/pipelinecpul_tb/DEBUG_TMP_REGS @ 739 ns
256'h0000000000000002000000000000000000000000000000000000000000000000
```

to



```
sim:/pipelinecpul_tb/DEBUG_TMP_REGS @ 770 ns
256'h0000000000000002000000000000000400000000000000000000000000000000
```

Means the X11 is changed to the result of SUB, X11,X10,X9.

6) The result of fifth and sixth instruction:

The fifth and sixth instruction is STUR instructions. So, they should be finished in after finishing their MEM stage. So, the signal DEBUG_MEM_CONTENTS should be changed at t =800ns and t =900ns. But, in fact, DEBUG_MEM_CONTENTS does not change. Because in the lab5 we cannot forward data from EX-stage or MEM-stage to ID-stage. So, the result of fifth and sixth instruction will load original value of X11 to the DMEM, which is 0, instead of the result of SUB, X11,X10,X9. Because we have already gotten the value of X11 at cycle 7and 8 (at that time the result of SUB, X11,X10,X9 had not written back to register). And our hazard detection unite cannot detects this hazard. Because signal ID_EX_MemRead =0.

These are key events in the lab5. And we still have some problem needed to be solved.