

Lab Report #6

Name: Ruochen Duan

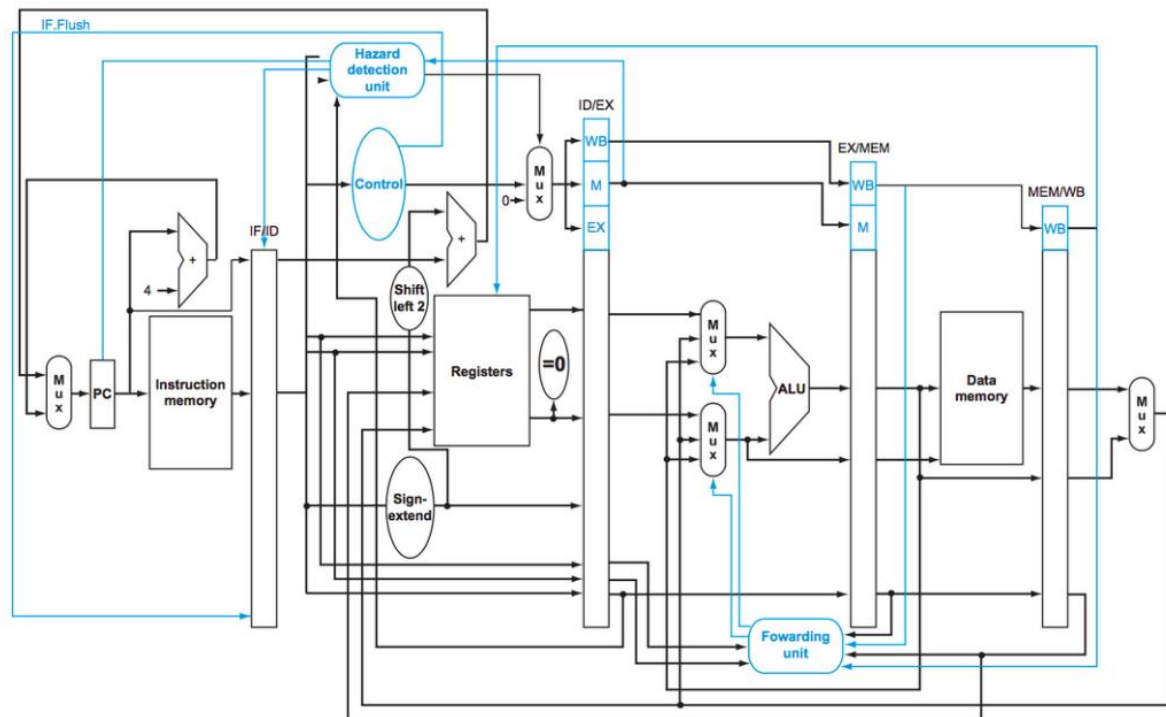
Student ID: 1405106

① Introduction

Lab6 asks us to design PipelinedCPU2 which is a modified version of PipelinedCPU1 that resolves conditional branches (CBZ) and unconditional branches (CBNZ) in the ID stage. The changes we made in the Lab6 are:

1. Add Flush signal to IF/ID register, when branches(conditional or unconditional branches) are taken, we need to flush the pipeline.
2. Move branch address adder to ID stage so that we can check whether the branch is taken or not after one cycle and flush it.
3. Add Flush signal to CPUcontrol unite to give flush signal to IF/ID register. When UBranch = 1, we flush the pipeline. When CBranch=1, we need to check the result of comparator to decide whether flush the pipeline. When instruction is CBZ and result of comparator is 1, we flush the pipeline. When instruction is CBNZ and result of comparator is 0, we flush the pipeline.
4. Add comparator to the ID stage and give their result to CPUcontrol to send the Flush signal. One input of comparator is zero and the other is the Read Data 2 of register file.

The figure shown below shows the Datapath of Lab6.



② Implementation

Below is the code of the comparator.

```

E:/Modelism/ee126/lab6/Comparator.vhd (/pipelinecpu2_tb/uut/comparator_1) - Default
Ln#
1  library ieee ;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity Comparator is
6  port(
7      in0 : in std_logic_vector(63 downto 0);
8      in1 : in std_logic_vector(63 downto 0);
9      output : out std_logic
10 );
11 end Comparator;
12
13 architecture behavl of Comparator is
14 begin
15     process(in0,in1)
16     begin
17         if in0 = in1 then
18             output <= '1';
19         else
20             output <= '0';
21         end if;
22     end process;
23 end behavl;

```

③ Test Result

The test we do is

	Binary instruction	Hexadecimal instruction
SUB X23, X20, X19	110010110001001100000001010010111	CB130297
CBZ X23, 5	10110100000000000000000010110111	B40000B7
ADD X9, X9, X9	10001011000010010000000100101001	8B090129
SUB X24, X22, X21	110010110001010100000001011011000	CB1502D8
CBZ X24, 3	10110100000000000000000001111000	B4000078
ADD X10, X10, X10	10001011000010100000000101001010	8B0A014A
ADD X11, X11, X11	10001011000010110000000101101011	8B0B016B
ADD X12, X12, X12	10001011000011000000000110001100	8B0C018C
B 2	00010100000000000000000000000010	14000002
ADD X19, X19, X19	100010110001001100000001001110011	8B130273
ADD X20, X20, X20	100010110001010000000001010010100	8B140294
nop		
nop		
nop		
nop		

Register Values

```
$X9  = 0h1      $X19 = 0h1
$X10 = 0h2      $X20 = 0h2
$X11 = 0h4
$X21 = 0x000000008BADF00D
$X12 = 0h8
$X22 = 0x000000008BADF00D
$X23 = 0x0000000000000000
$X24 = 0x0000000000000000
```

DMEM Contents

```
DMEM(0x0)  = 0x00 00 00 09
DMEM(0x8)  = 0x00 00 00 08
DMEM(0x10) = 0x00 00 00 07
DMEM(0x18) = 0x00 00 00 06
```

Figure 1 shows the order of executed instructions of test.

```
    SUB X23, X20, X19
    CBZ X23, 5
    ADD X9, X9, X9
    SUB X24, X22, X21
    CBZ X24, 3
    ADD X10, X10, X10
    ADD X11, X11, X11
    ADD X12, X12, X12
    B 2
    ADD X19, X19, X19
    ADD X20, X20, X20
    nop
    nop
    nop
    nop
```

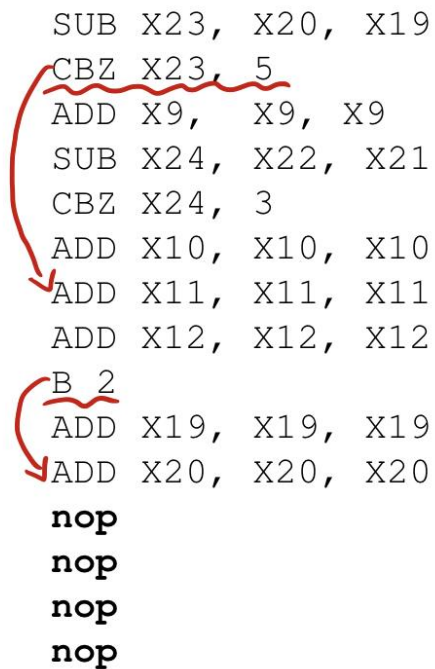


Figure 1 order of instructions

Figure 2 shows the Timing diagram of the test.

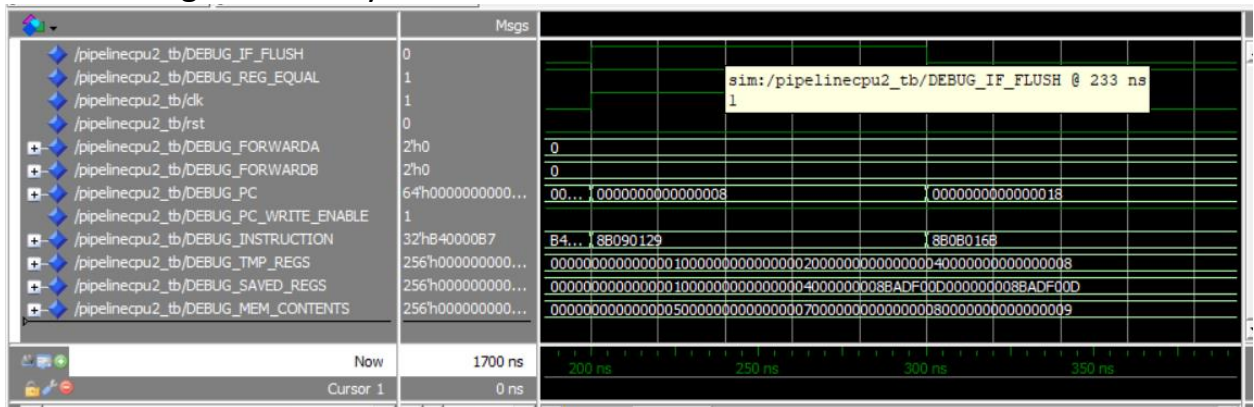
Cycle	1	2	3	4	5	6	7	8	9	10	11	12
SUB X23, X20, X19	IF	ID	EX	MEM	WB							
CBZ X23, 5		IF	ID	EX	MEM	WB						
ADD X9, X9, X9			IF	Flush								
ADD X11, X11, X11				IF	ID	EX	MEM	WB				
ADD X12, X12, X12					IF	ID	EX	MEM	WB			
B 2						IF	ID	EX	MEM	WB		
ADD X19, X19, X19							IF	Flush				
ADD X20, X20, X20								IF	ID	EX	MEM	WB

Figure 2 Timing diagram

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

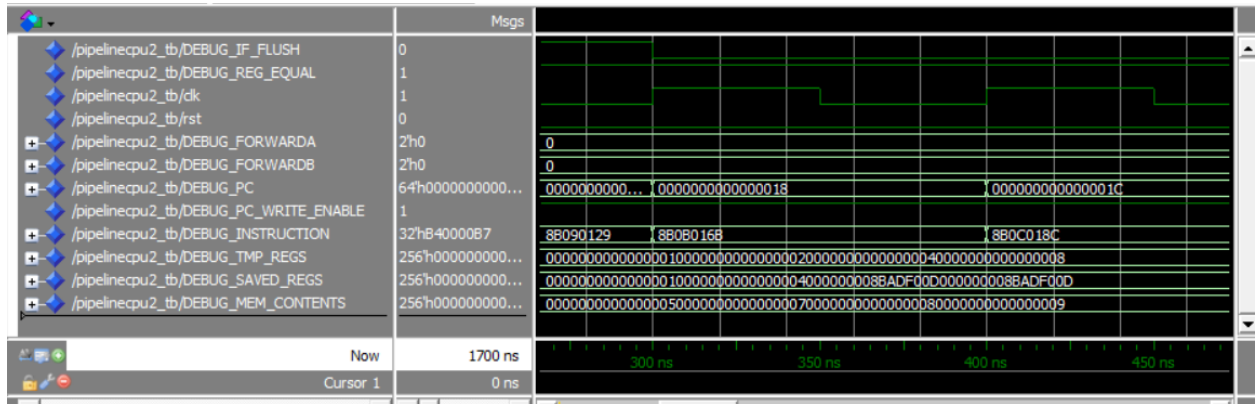
According to the Timing diagram, I will show and explain key events in the simulation (Complete figures of simulated waveforms will be attached separately in other files)

1. The Flush signal in the cycle 3



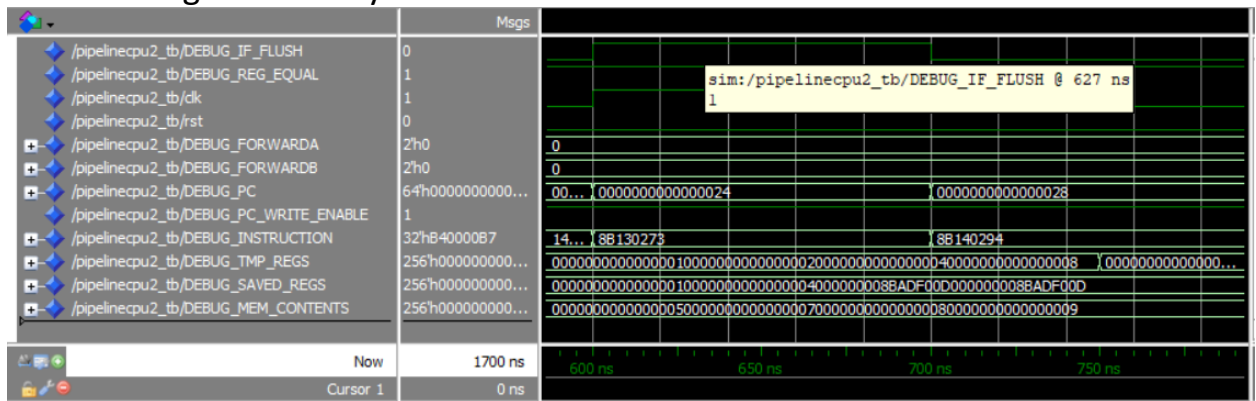
We can see that in cycle 3 (200ns – 300ns), the signal `DEBUG_IF_FLUSH = 1` which shows that the Branch is taken (CBZ X23,5). However, it is not right. Because in the first instruction, the value of X23 is changed to X20 - X19 = 1, X23 is not zero, the branch should not be taken. The reason why this branch is taken is because we do not have forwarding for the ID stage. So, we get the original value of X23 and old value is zero and branch is taken.

2. The branch to instruction ADD X11, X11, X11.



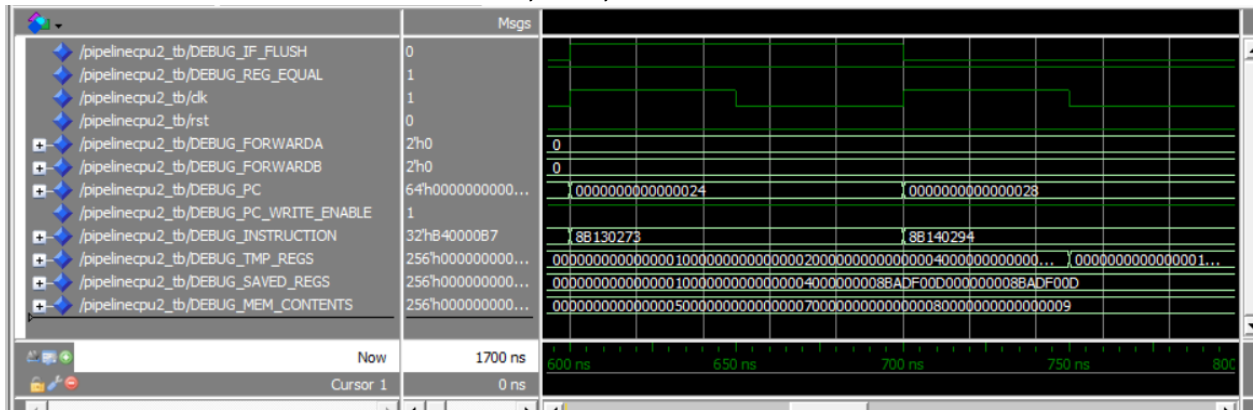
We can see from the figure that at $t=300$ ns (cycle 4), the signal `DEBUG_INSTRUCTION` is changed from `8B090129` (`ADD X9, X9, X9`) to `8B0B016B` (`ADD X11, X11, X11`).

3. The Flush signal in the cycle 8



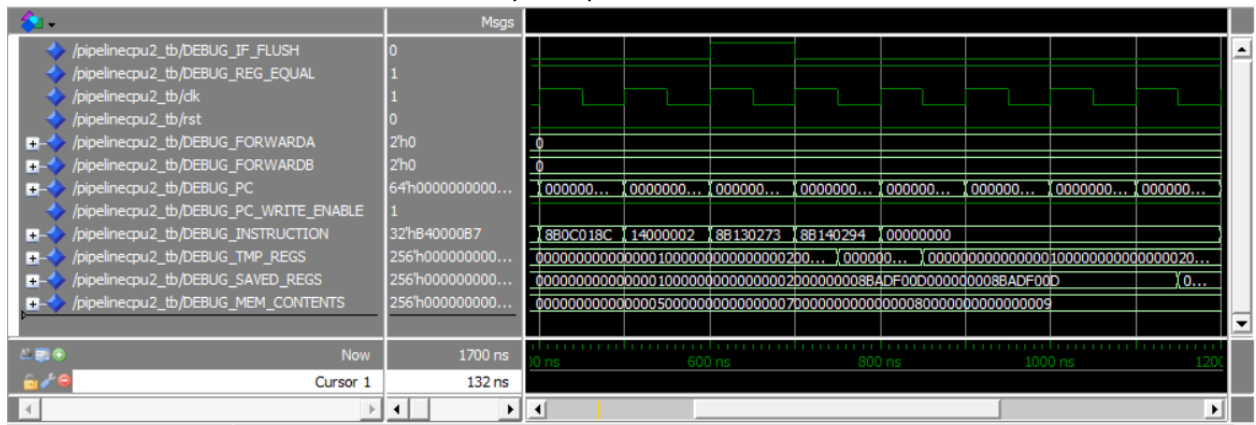
We can see that in cycle 8 (600ns – 700ns), the signal `DEBUG_IF_FLUSH` =1 which shows that the Branch is taken(B 2). This is a unconditional branch, so we need to flush the pipeline.

4. The branch to instruction `ADD X20, X20, X20`.

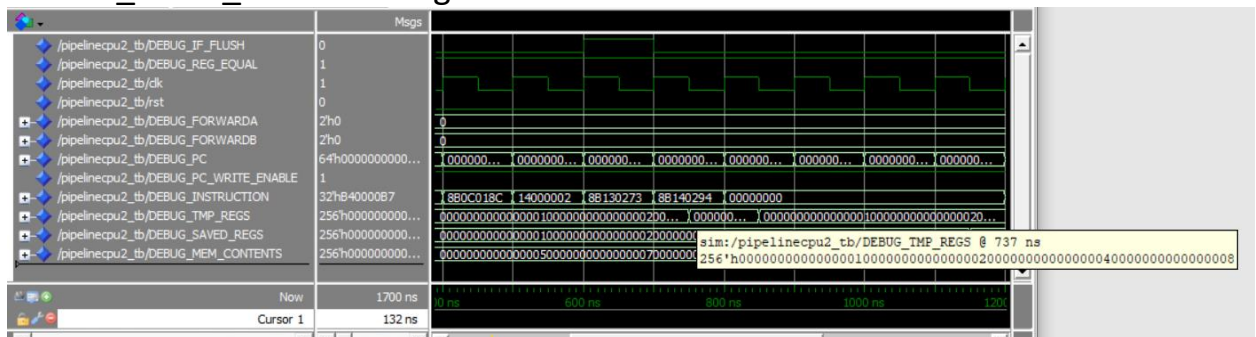


We can see from the figure that at t=700 ns (cycle 8), the signal `DEBUG_INSTRUCTION` is changed from 8B090129 (ADD X19, X19, X19) to 8B0B016B (ADD X20, X20, X20). The instruction ADD X19, X19, X19 is not taken.

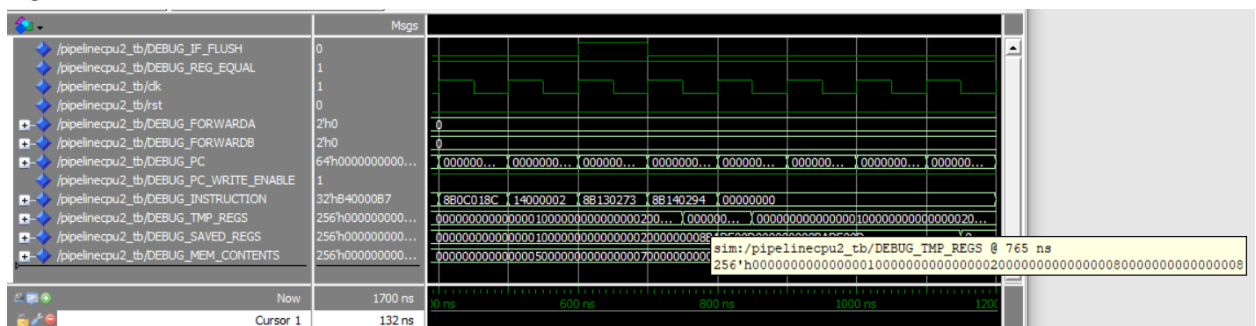
5. The result of instruction `ADD X11, X11, X11`



We can see from the figure that at t=750ns(cycle 8) the signal DEBUG_TEMP_REGS is changed from

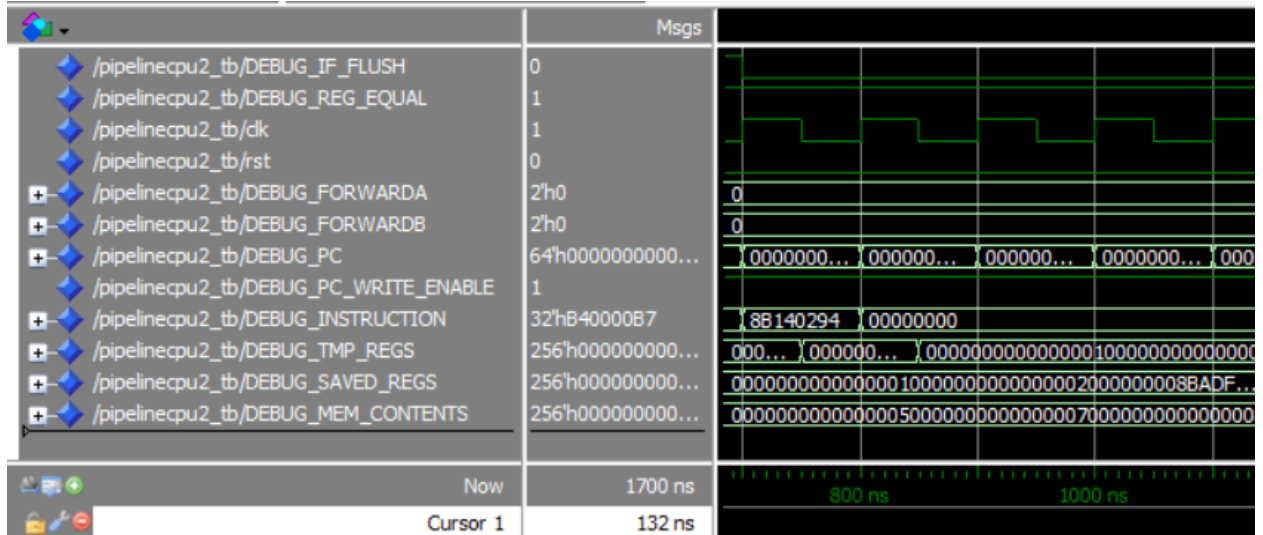


To

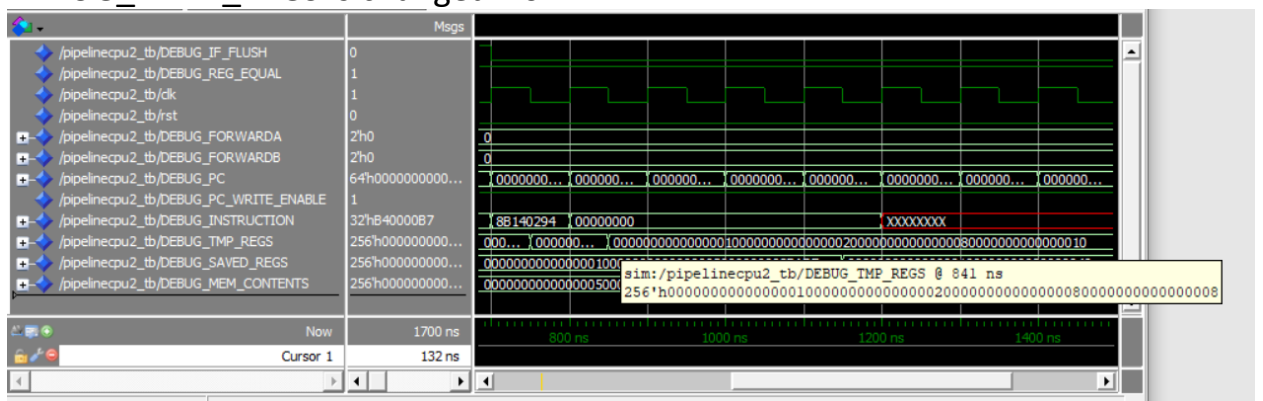


Means X11 is changed to the result $X11+X11=8$.

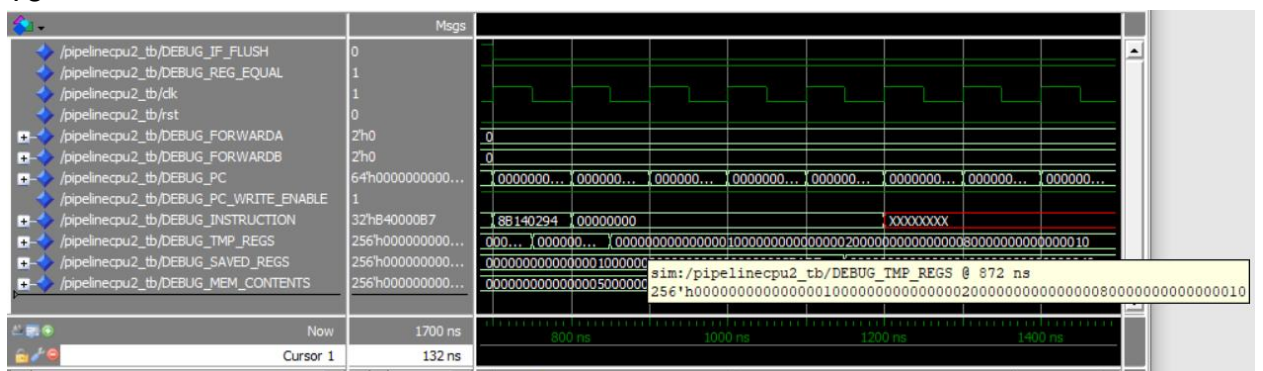
6. The result of instruction `ADD X12, X12, X12`



We can see from the figure that at t=850ns (cycle 9) the signal `DEBUG_TEMP_REGS` is changed from

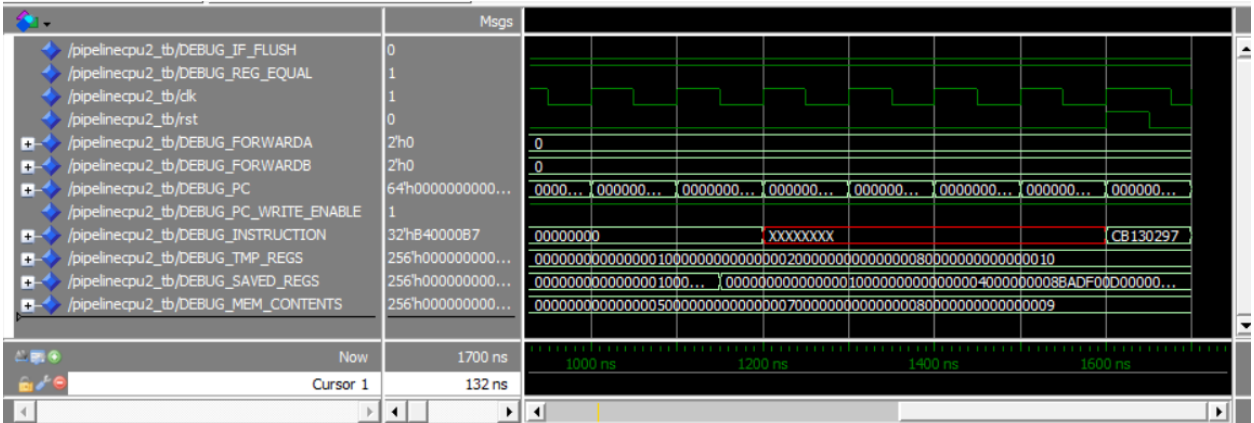


To

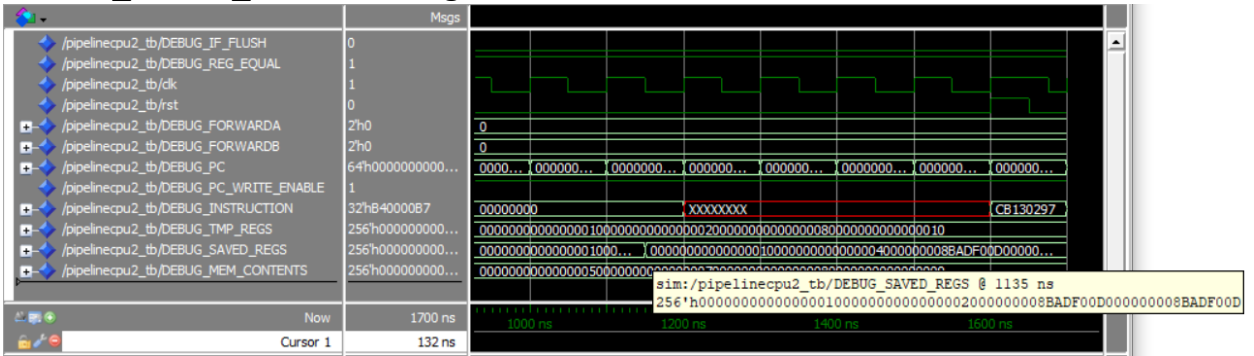


Means X12 is changed to the result $X12 + X12 = 16(0x10)$.

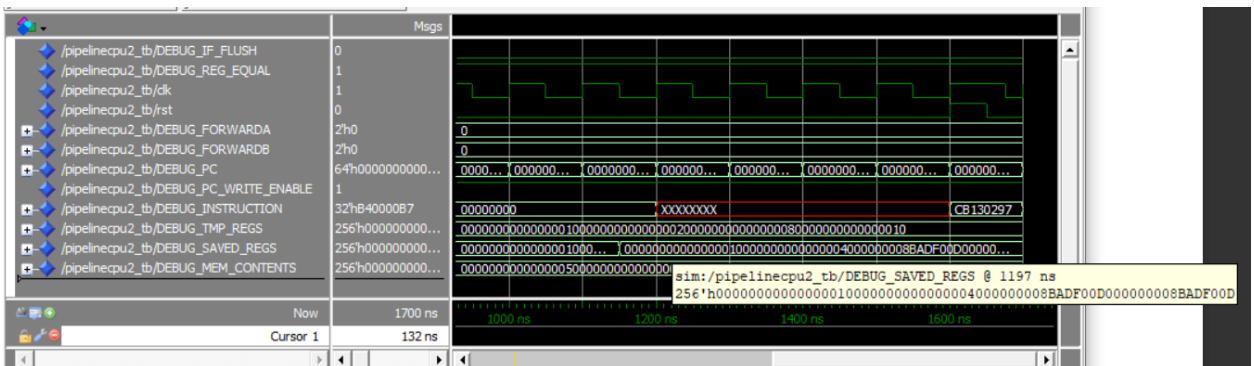
7. The result of instruction `ADD X20, X20, X20`



We can see from the figure that at t=1150ns (cycle 12) the signal `DEBUG_SAVED_REGS` is changed from



To



Means X20 is changed to the result $X20+X20=4$.

These are key events in the lab6. And we still have some problem needed to be solved.(we still use wrong data of X23)