Kaicheng Yang

I would like to use one late token for lab 6.

1: Compilation Order and Design Decisions

a Compilation Order

1. xorgate.vhd
2. half_adder.vhd
3. full_adder.vhd
4. adder64.vhd
5. add.vhd
6. mux5.vhd
7. mux64.vhd
8. mux64_3to1.vhd
9. muxcontrol.vhd
10. alu.vhd
11. alucontrol.vhd
12. cpucontrol.vhd
13. shiftleft2.vhd
14. signextend.vhd
15. pc.vhd
16. forwarding.vhd
17. hazarddetction.vhd
18. equalzero.vhd
19. reg_IFID.vhd
20. reg_IDEX.vhd
21. reg_EXMEM.vhd
22. reg_MEMWB.vhd
23. registers.vhd
24. lab5_dmem.vhd
25. lab5_imem.vhd
26. pipelinedcpu2.vhd
27. pipelinedcpu2_tb.vhd

b Design Decisions

IF.flush and PCSrc

I use (Unconditional_Branch OR (Conditional_Branch AND Readdata2)) as the IF.flush and PCSrc signal. IF.flush and PCSrc is '1' when Unconditional_Branch is '1', or Conditional_Branch is '1' and Readdata2 is '0'.

Prediction

I always predict not-taken in my pipeline. The next instruction of a branch instruction would always be fetched first. And if the branch is taken, the instruction would be flushed to not affect the program.

Pipeline Registers

For all pipeline registers(IF/ID, ID/EX, EX/MEM, MEM/WB), I initialize their output singals to all zero. The input signals are assigned to output signals at the rising edge of 'clk'. The output signals are set to zero when 'rst' is '1'.

2: Cycle One

a IF

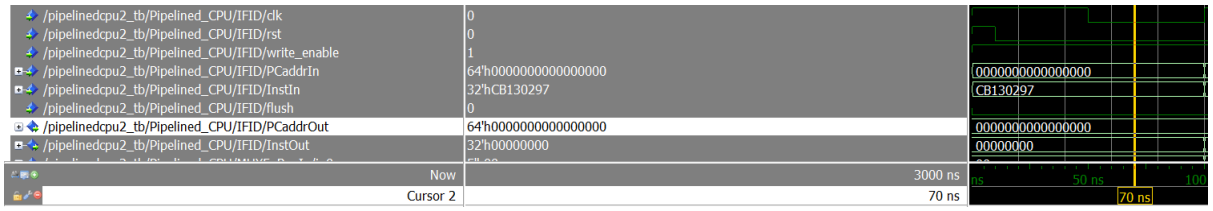

Figure 1: IF Signals of Cycle one

In cycle one, IF fetches instruction one
(SUB X23, X20, X19: 1100 1011 0001 0011 0000 0010 1001 0111). We can see PC address is 0x00 and nothing is passed to ID stage.

3: Cycle Two

a IF

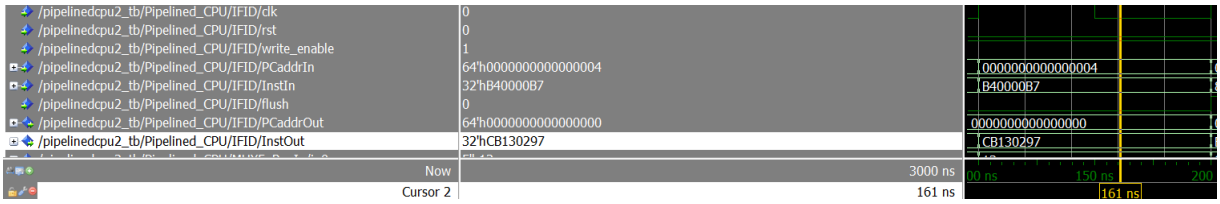

Figure 2: IF Signals of Cycle two

In cycle two, IF fetches instruction two
(CBZ X23, 5: 1011 0100 0000 0000 0000 0000 1011 0111). We can see PC address is 0x04 and instruction one is passed to ID stage.
This is a CBZ instruction, we predict it will not be taken(always not taken in my implementation).

b ID

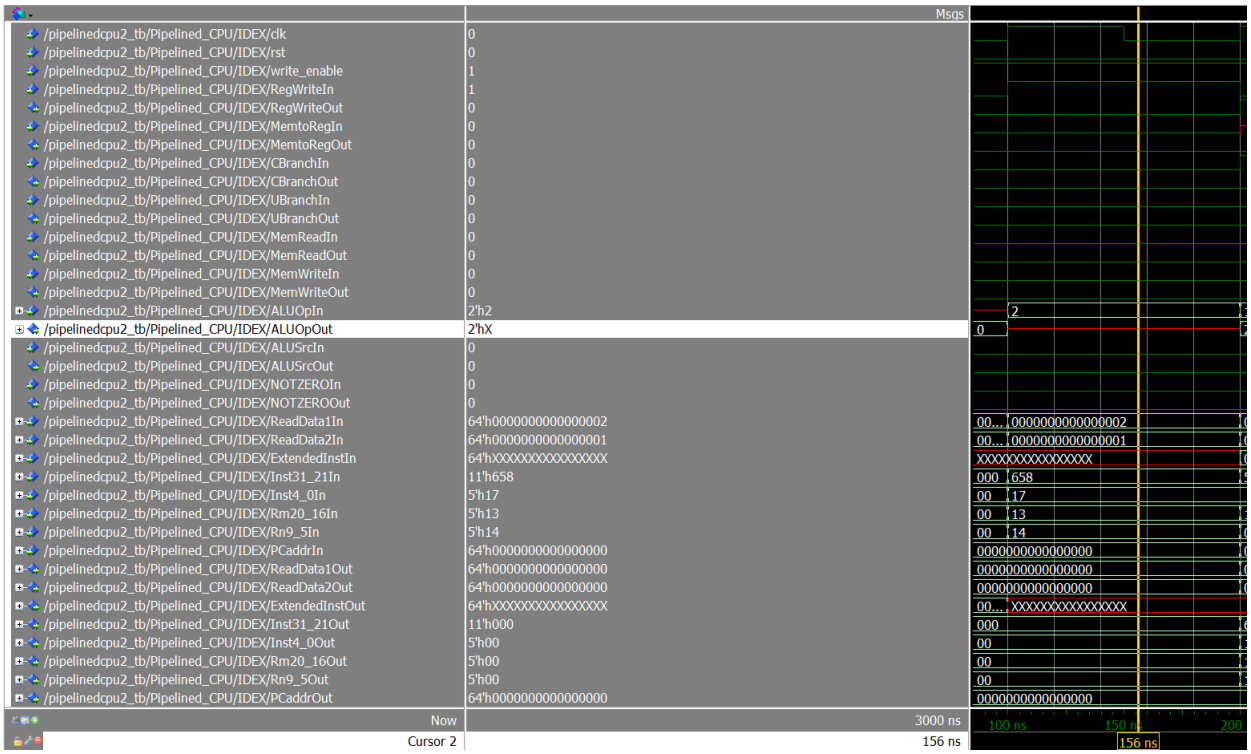

Figure 3: ID/EX Signals of Cycle two

In cycle two, ID decodes instruction one
(SUB X23, X20, X19: 1100 1011 0001 0011 0000 0010 1001 0111). Signals are changed according to instruction one at 100 ns, we can see its CPU control signals in the above figure. And Rn is 'h14=20; Rm is 'h13=19; Rd is 'h17=23. No signal value is passed to EX stage.

4: Cycle Three

a IF

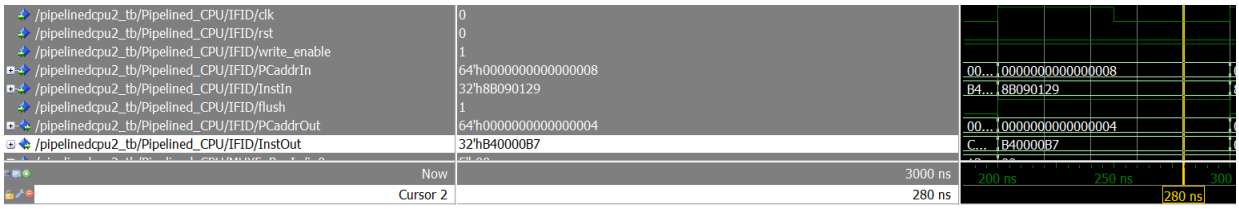

Figure 4: IF Signals of Cycle three

In cycle three, IF fetches instruction three
(ADD X9, X9, X9 1000 1011 0000 1001 0000 0001 0010 1001). We can see PC address is 0x08 and instruction two is passed to ID stage.
Actually instruction two shouldn't be taken, however, a new data hazard (data dependencies from a previous instruction to a branch) leads to using the old value of $X23=0. Our prediction is wrong, and instruction two should be taken.
And IFFLUSH here is '1', we will flush instruction three at the beginning of the next cycle so that it will pass NOP to ID stage, and not have any impact on our program.

b ID

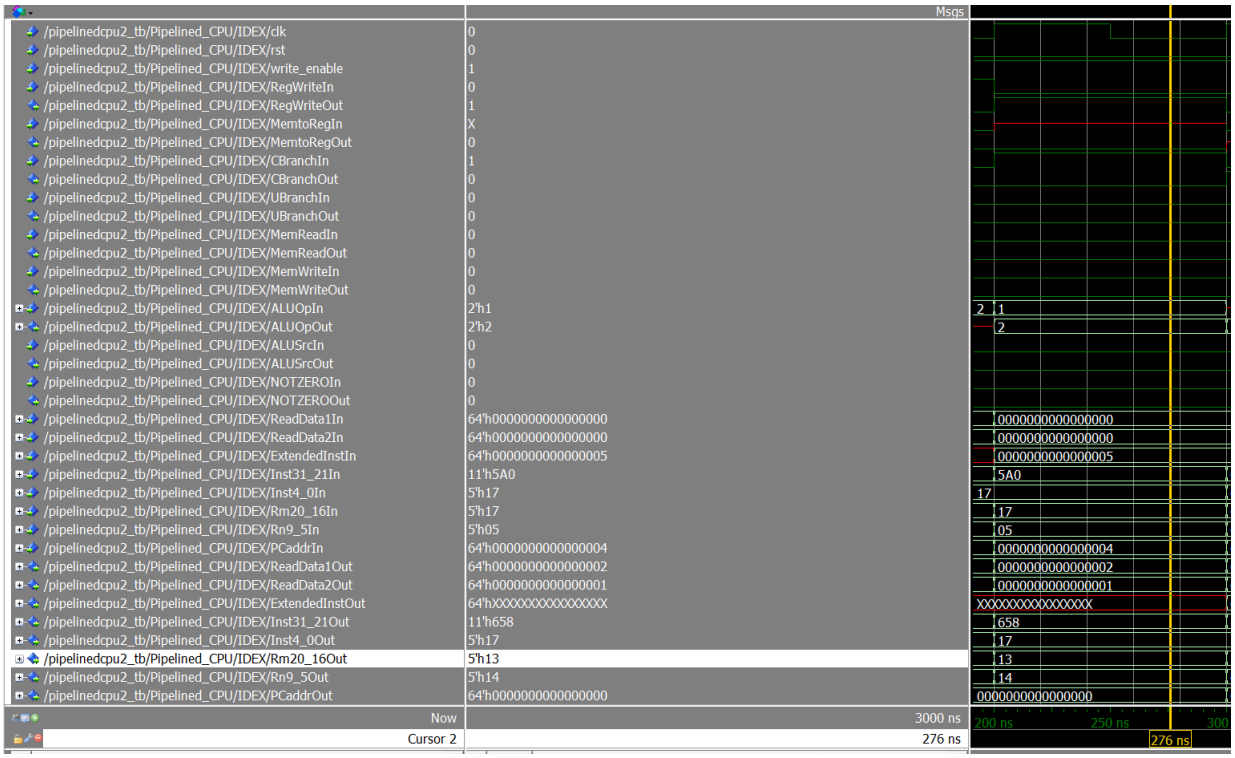

Figure 5: ID/EX Signals of Cycle three

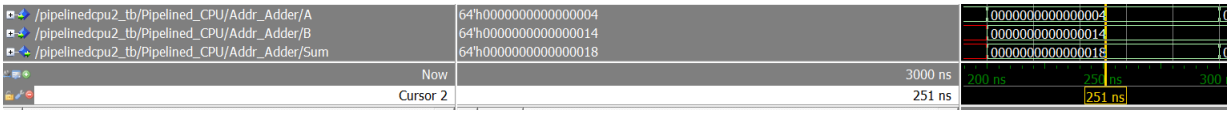

Figure 6: Address Counting Signals of Cycle three

In cycle three, ID decodes instruction two
(CBZ X23, 5: 1011 0100 0000 0000 0000 0000 1011 0111). Signals are changed according to instruction two at 200 ns, we can see its CPU control signals in the above figure. And Rt is 'h17=23; COND_BR_address=5. Signal values of instruction one are passed to EX stage.
This is a conditional branch instruction, $X23 equals 0, we take the branch, and the next address is 4 + 5*4 = 24 = 'h18.

c EX

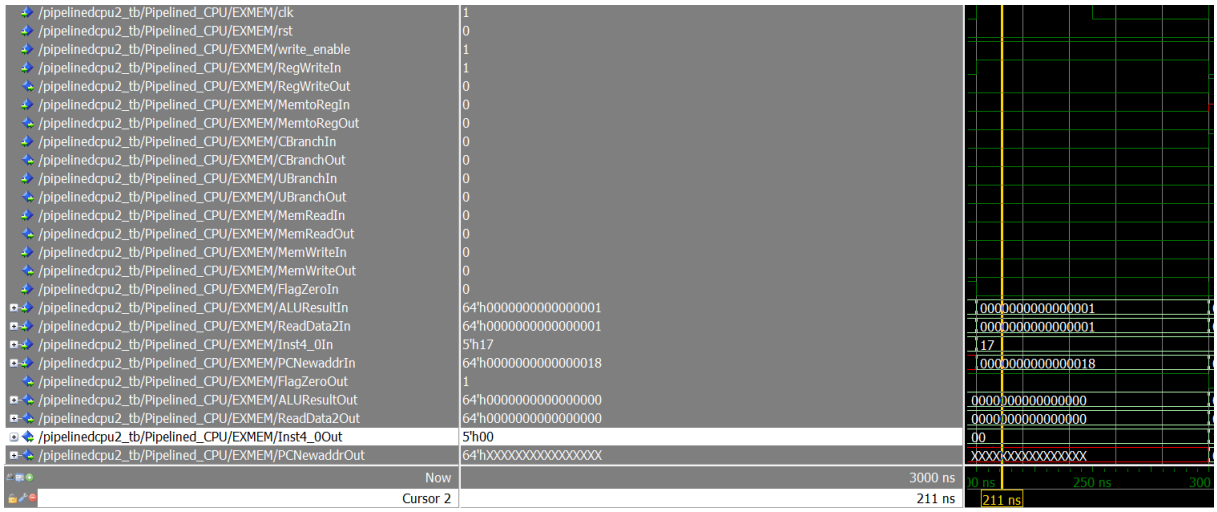

Figure 7: EX/MEM Signals of Cycle three

In cycle three, EX executes operations of instruction one
(SUB X23, X20, X19: 1100 1011 0001 0011 0000 0010 1001 0111). Signals are changed according to instruction one at 200 ns, we can see the ALUResult is 'h01=1, which is $X20-$X19=2-1=1.
Nothing is passed to MEM stage.

5: Cycle Four

a IF

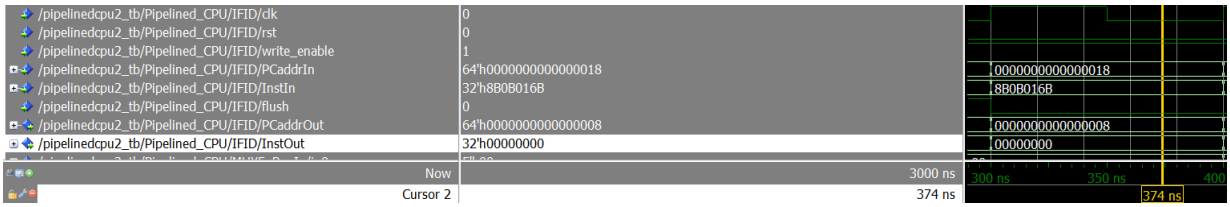

Figure 8: IF/ID Signals of Cycle four

In cycle three, IF fetches instruction seven
(ADD X11, X11, X11: 1000 1011 0000 1011 0000 0001 0110 1011). We can see PC address is 0x18 and flushed instruction three is passed to ID stage.

b ID

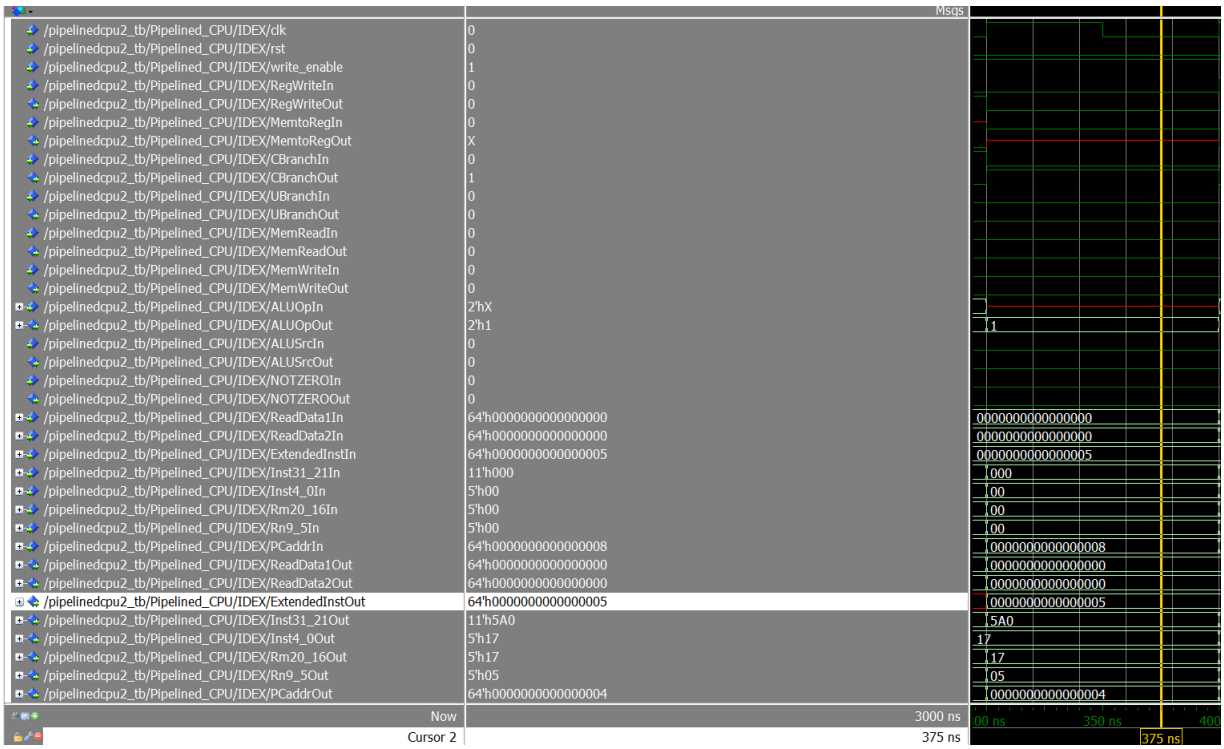

Figure 9: ID/EX Signals of Cycle four

In cycle four, ID decodes flushed instruction three(NOP).
Signal values of instruction two are passed to EX stage.

c EX

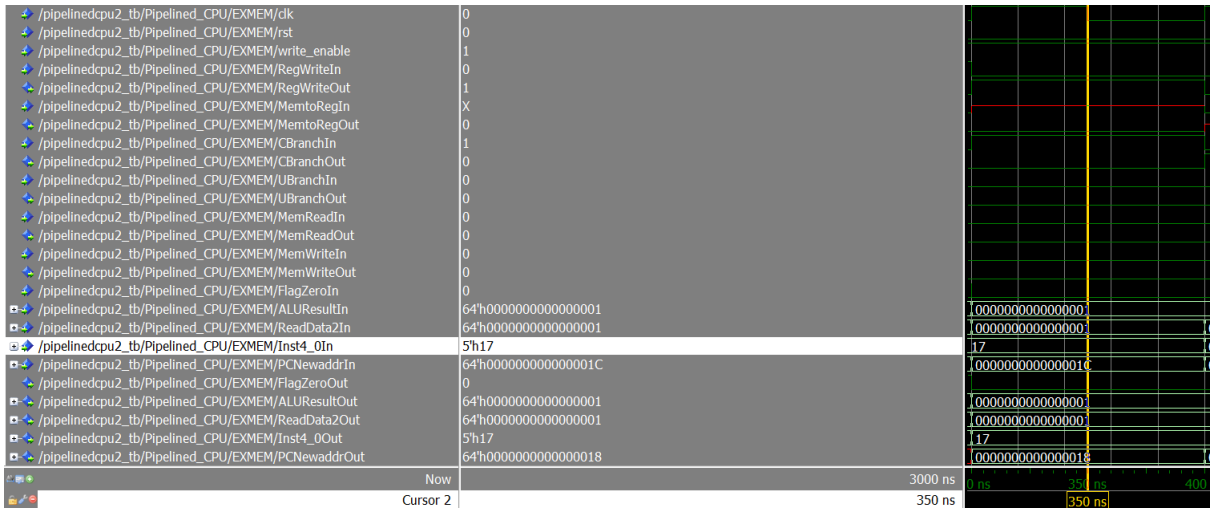

Figure 10: EX/MEM Signals of Cycle four

In cycle four, EX executes operations of instruction two
(CBZ X23, 5: 1011 0100 0000 0000 0000 0000 1011 0111). This is a CBZ instruction, we actually do nothing here.
Results of instruction one are passed to MEM stage.

d MEM

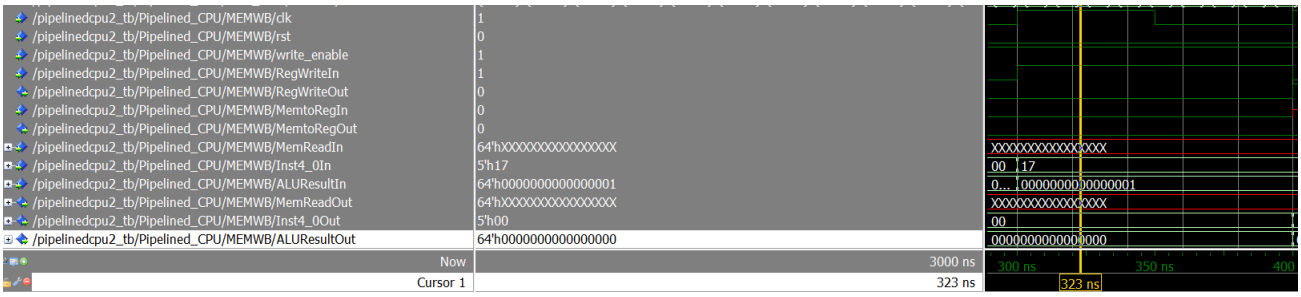

Figure 11: EX/MEM Signals of Cycle four

In cycle four, MEM executes for instruction one
(SUB X23, X20, X19: 1100 1011 0001 0011 0000 0010 1001 0111). Since instruction one is a SUB instruction, we do nothing in MEM stage, however, we get the ALUResult of subtraction here, which is 'h01=1.
Nothing is passed to WB stage.

6: Cycle Five

a IF

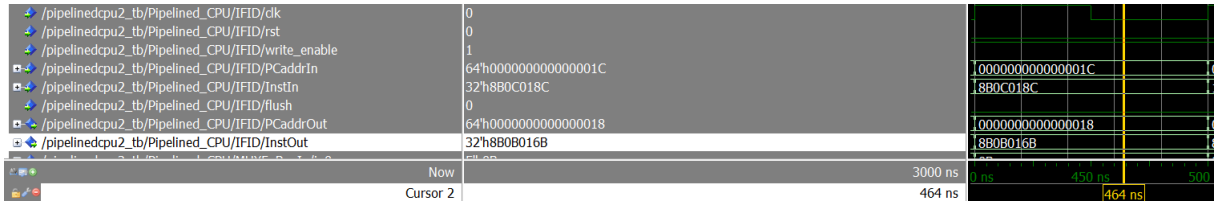

Figure 12: IF/ID Signals of Cycle five

In cycle five, IF fetches instruction eight
(ADD X12, X12, X12: 1000 1011 0000 1100 0000 0001 1000 1100). We can see PC address is 0x1C and instruction seven is passed to ID stage.

b ID

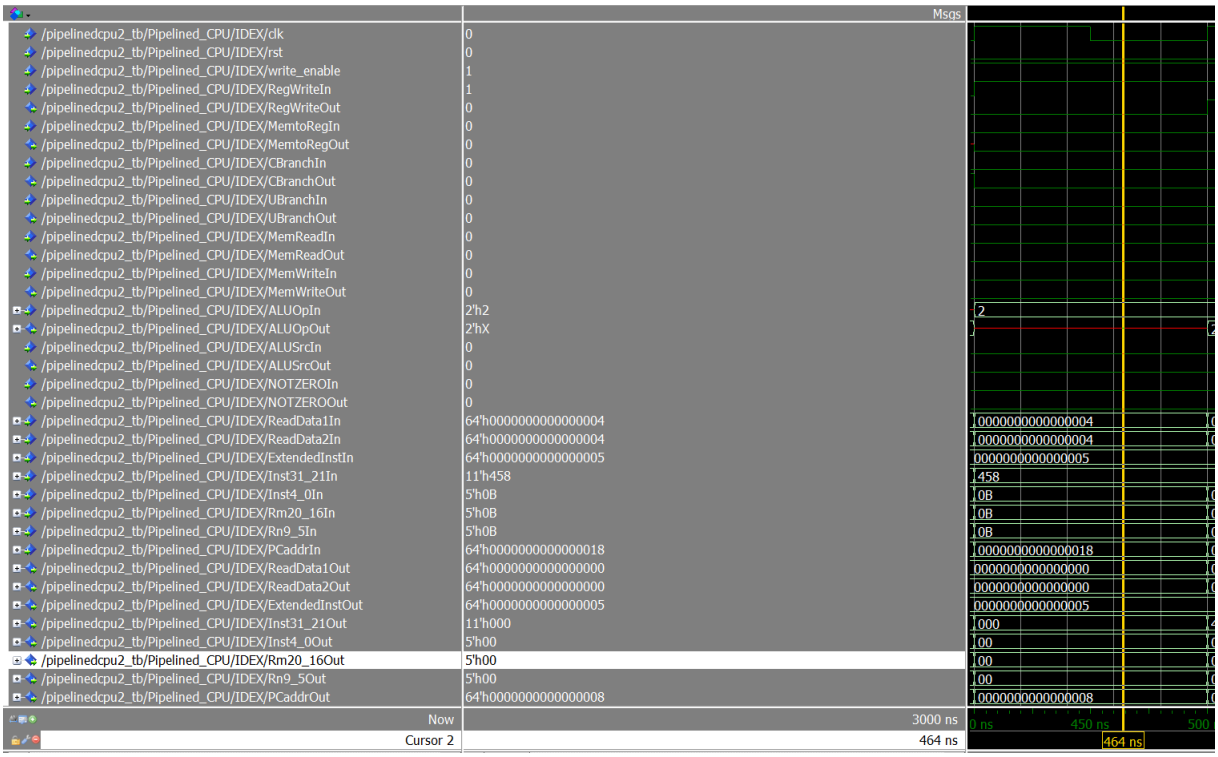

Figure 13: ID/EX Signals of Cycle five

In cycle five, ID decodes instruction seven
(ADD X11, X11, X11: 1000 1011 0000 1011 0000 0001 0110 1011). Signals are changed according to instruction seven at 400 ns, we can see its CPU control signals in the above figure.

And Rm is 'h0B=11; Rn is 'h0B=11; Rd is 'h0B=11. Signal values of flushed instruction three are passed to EX stage.

c EX

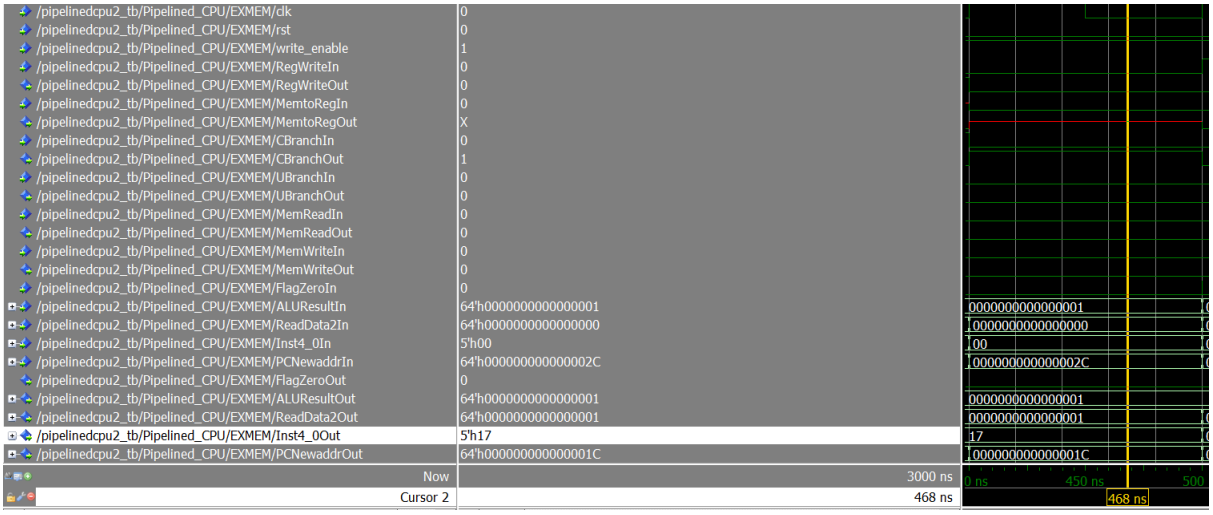

Figure 14: EX/MEM Signals of Cycle five

In cycle five, EX executes operations of flushed instruction three. Signals are changed according to flushed instruction three at 400 ns.
Results of instruction two are passed to MEM stage.

d MEM

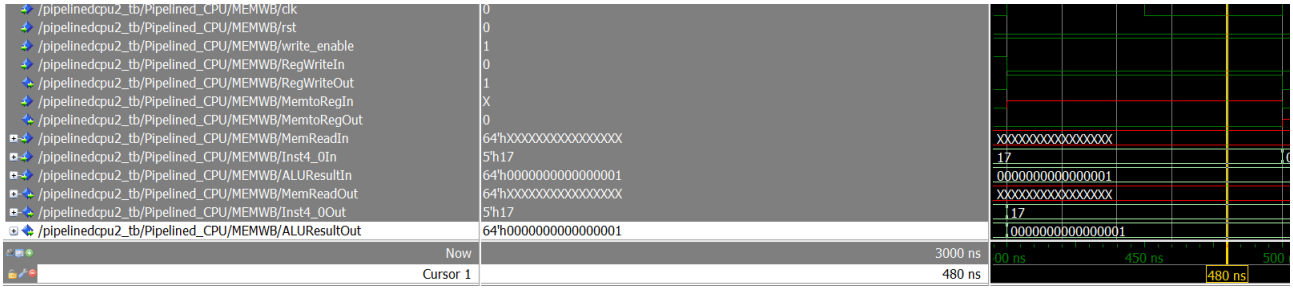

Figure 15: EX/MEM Signals of Cycle five

In cycle five, MEM executes for instruction two
(CBZ X23, 5: 1011 0100 0000 0000 0000 0000 1011 0111). Since instruction one is a CBZ instruction, we do nothing in MEM stage.
The value of data memory related to instruction one is passed to WB stage.

e WB

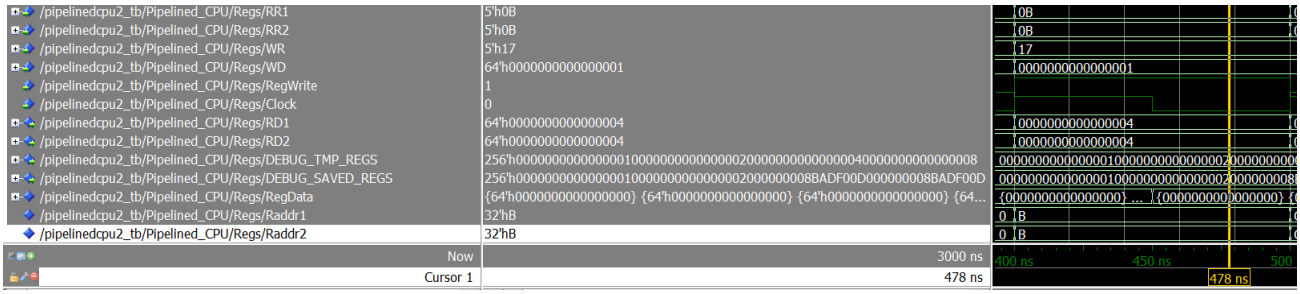

Figure 16: Registers Signals of Cycle five

In cycle five, WB writes value into registers according to instruction one. Control signal RegWrite is '1', write register(WR) is 'h17=23, write data(WD) is 'h01=1. We are writing the result of subtraction in instruction one into $X23.

f Debug Signals

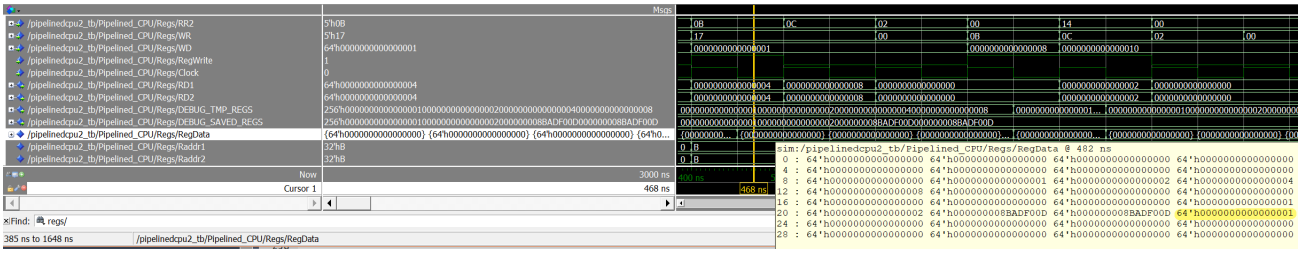

Figure 17: Debug Signals of Cycle five

During cycle five, $X23 is changed from 'h00 to 'h01 at 450 ns.

7: Cycle Six

a IF

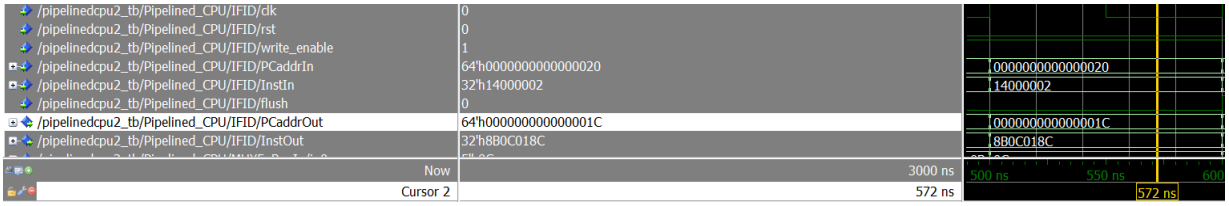

Figure 18: IF/ID Signals of Cycle six

In cycle six, IF fetches instruction nine
(B 2: 0001 0100 0000 0000 0000 0000 0000 0010). We can see PC address is 0x20 and instruction eight is passed to ID stage.

b ID

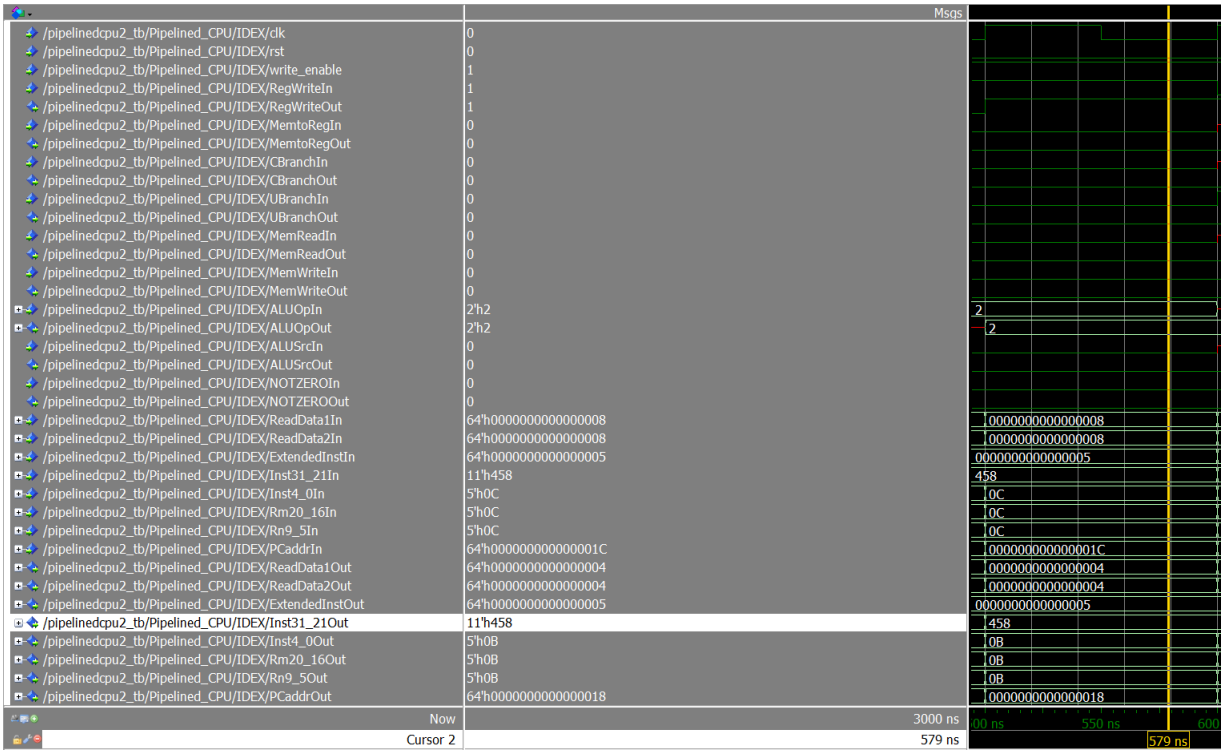

Figure 19: ID/EX Signals of Cycle six

In cycle six, ID is decoding instruction eight
(ADD X12, X12, X12: 1000 1011 0000 1100 0000 0001 1000 1100). Signals are changed according to instruction eight at 500 ns, we can see its CPU control signals in the above figure.

And Rm is 'h0C=12; Rn is 'h0C=12; Rd is 'h0C=12. Signal values of instruction seven are passed to EX stage.

c EX

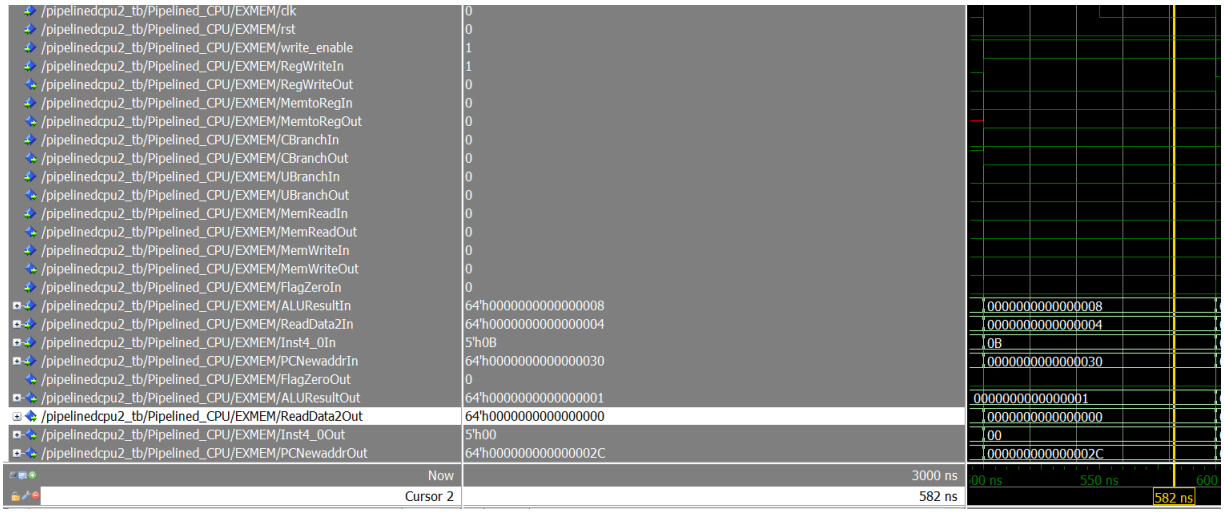

Figure 20: EX/MEM Signals of Cycle six

In cycle six, EX is executing operations of instruction seven
(ADD X11, X11, X11: 1000 1011 0000 1011 0000 0001 0110 1011). Signals are changed according to instruction seven at 500 ns, we can see the ALUResult is 'h08=8, which is $X11+$X11 = 4+4 = 8
Results of instruction flushed instruction three are passed to MEM stage.

d MEM

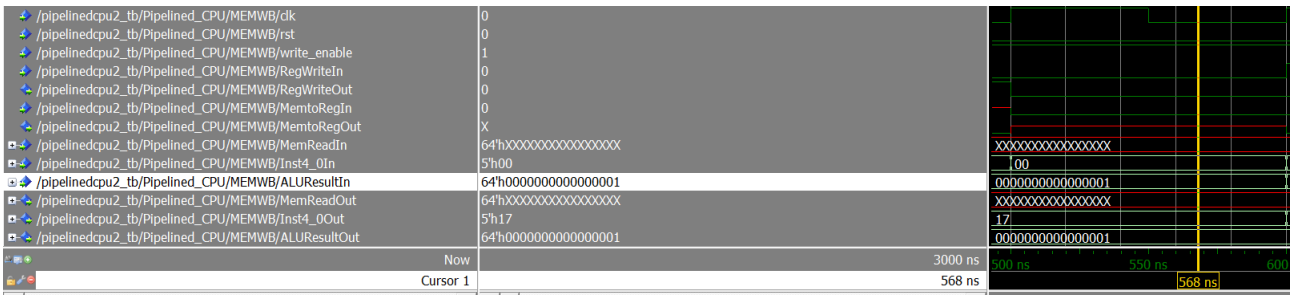

Figure 21: EX/MEM Signals of Cycle six

In cycle six, MEM is executing operations of flushed instruction three. We do nothing here. The value of data memory related to instruction two is passed to WB stage.

e WB

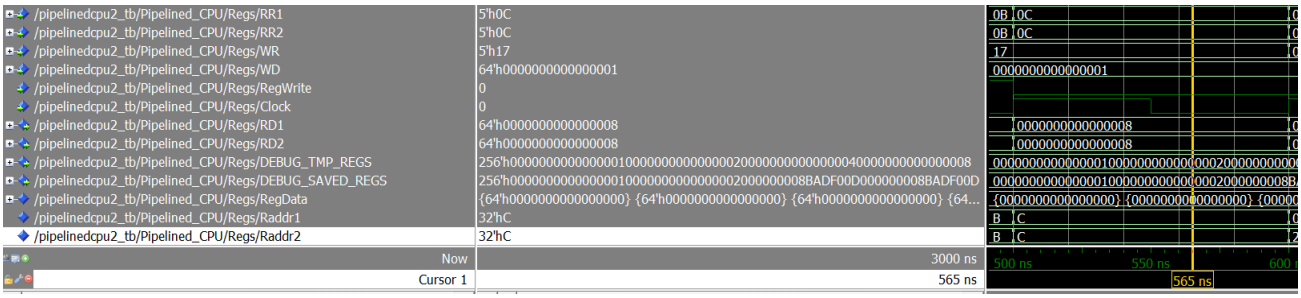

Figure 22: Registers Signals of Cycle six

In cycle six, WB writes value into registers according to instruction two. This is a B instruction, Control signal RegWrite is '0', we do not write into registers.

8: Cycle Seven

a IF

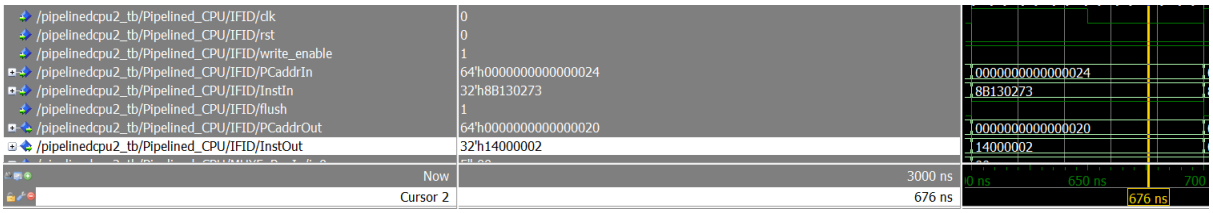

Figure 23: IF/ID Signals of Cycle seven

In cycle seven, IF fetches instruction ten
(ADD X19, X19, X19: 1000 1011 0001 0011 0000 0010 0111 0011). We can see PC address is 0x24 and instruction nine is passed to ID stage.
Although instruction nine is an unconditional branch instruction, we still predict instruction ten is not being taken. It's kind of a compromise for the conditional branch. Our prediction is wrong, and instruction nine should be taken.

And IFFLUSH here is '1', we will flush instruction ten at the beginning of the next cycle so that it will pass NOP to ID stage, and not have any impact on our program.

b ID

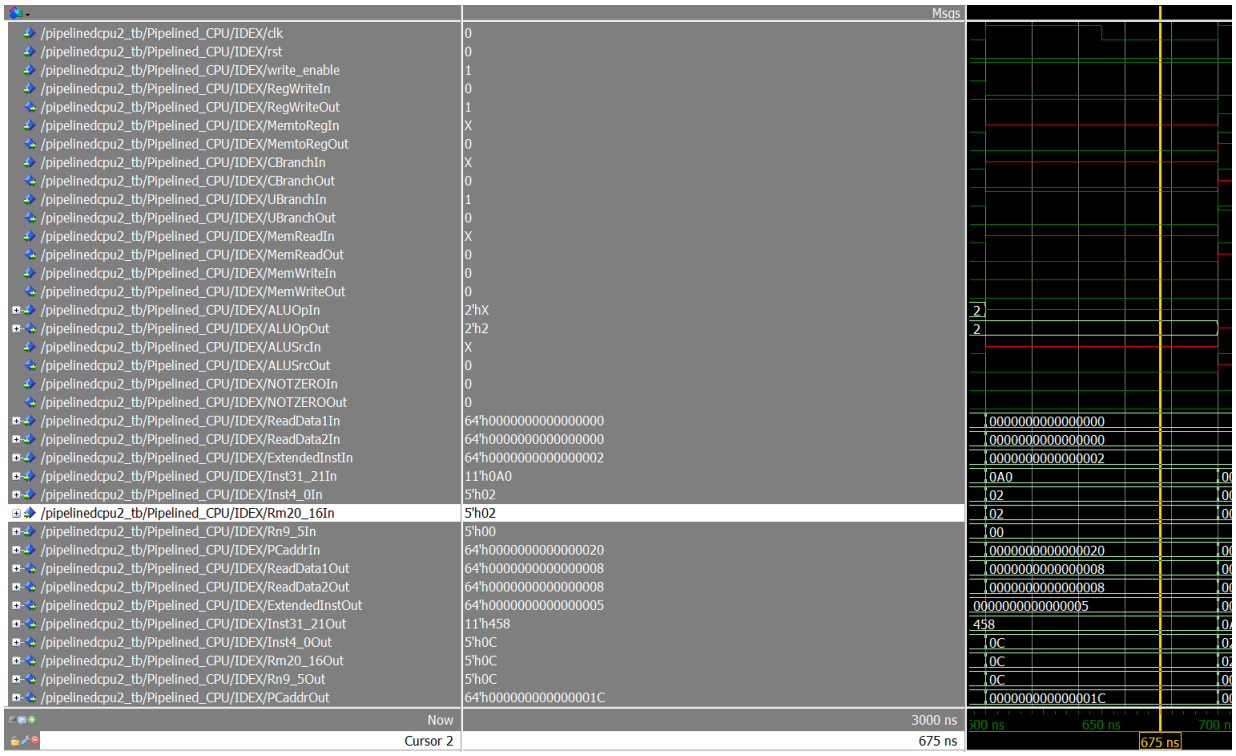

Figure 24: ID/EX Signals of Cycle seven

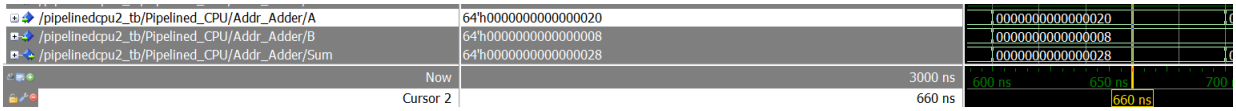

Figure 25: Address Counting Signals of Cycle seven

In cycle seven, ID is decoding instruction nine
(B 2: 0001 0100 0000 0000 0000 0000 0000 0010). Signals are changed according to instruction nine at 600 ns, we can see its CPU control signals in the above figure. And BR_address=2. Signal values of instruction eight are passed to EX stage.
This is an unconditional branch instruction, we take the branch, and the next address is 32 + 2*4 = 40 = 'h28.

c EX

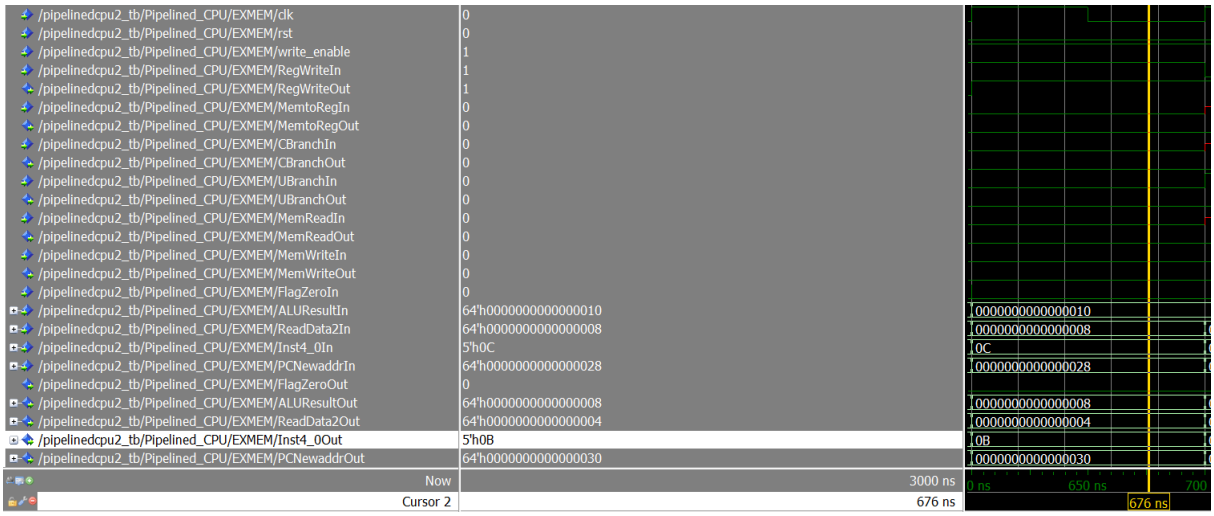

Figure 26: EX/MEM Signals of Cycle seven

In cycle seven, EX executes operations of instruction eight
(ADD X12, X12, X12: 1000 1011 0000 1100 0000 0001 1000 1100). Signals are changed according to instruction seven at 600 ns, we can see the ALUResult is 'h10=16, which is $X12+$X12 = 8+8 = 16.
Results of instruction seven are passed to MEM stage.

d MEM

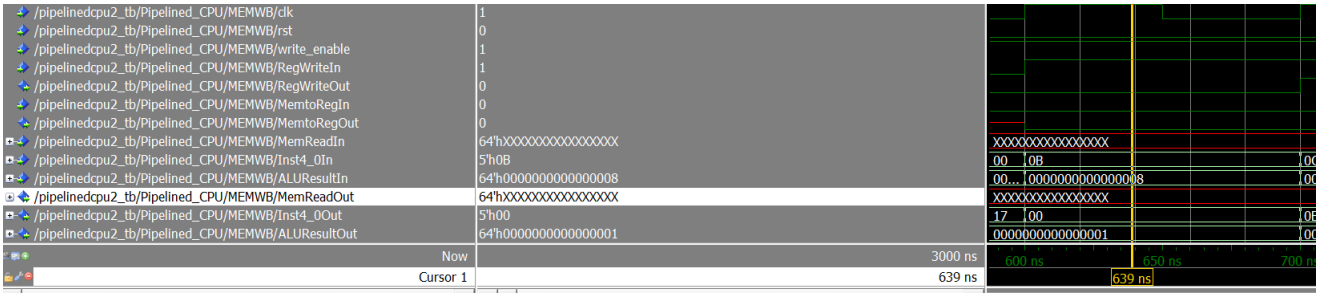

Figure 27: EX/MEM Signals of Cycle seven

In cycle seven, MEM executes for instruction seven. Since instruction seven is an ADD instruction, we do nothing in MEM stage, however we get the ALUResult of addition here, which is 'h08=8.
Results of flushed instruction three are passed to WB stage.

e WB

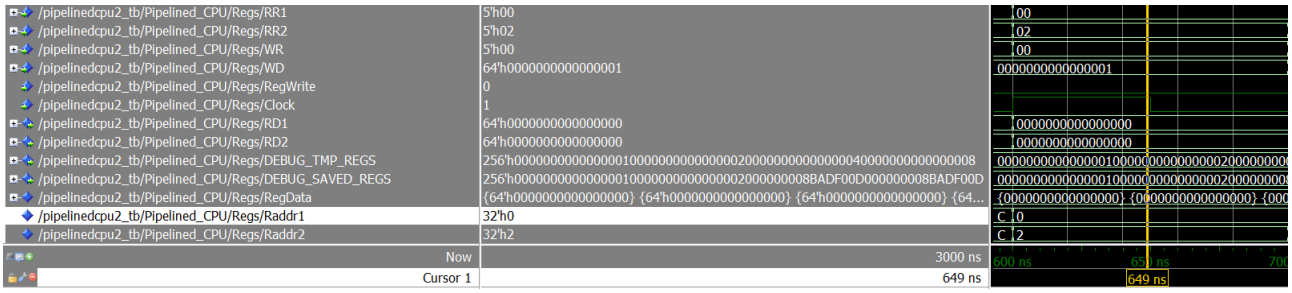

Figure 28: Registers Signals of Cycle seven

In cycle seven, WB writes value into registers according to flushed instruction three. We do nothing here.

9: Cycle Eight

a IF

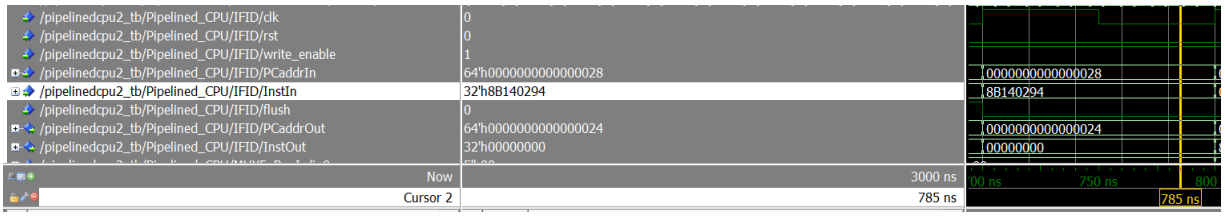

Figure 29: IF/ID Signals of Cycle eight

In cycle eight, IF fetches instruction eleven(ADD X20, X20, X20: 1000 1011 0001 0100 0000 0010 1001 0100). We can see PC address is 0x28 and flushed instruction ten is passed to ID stage.

b ID

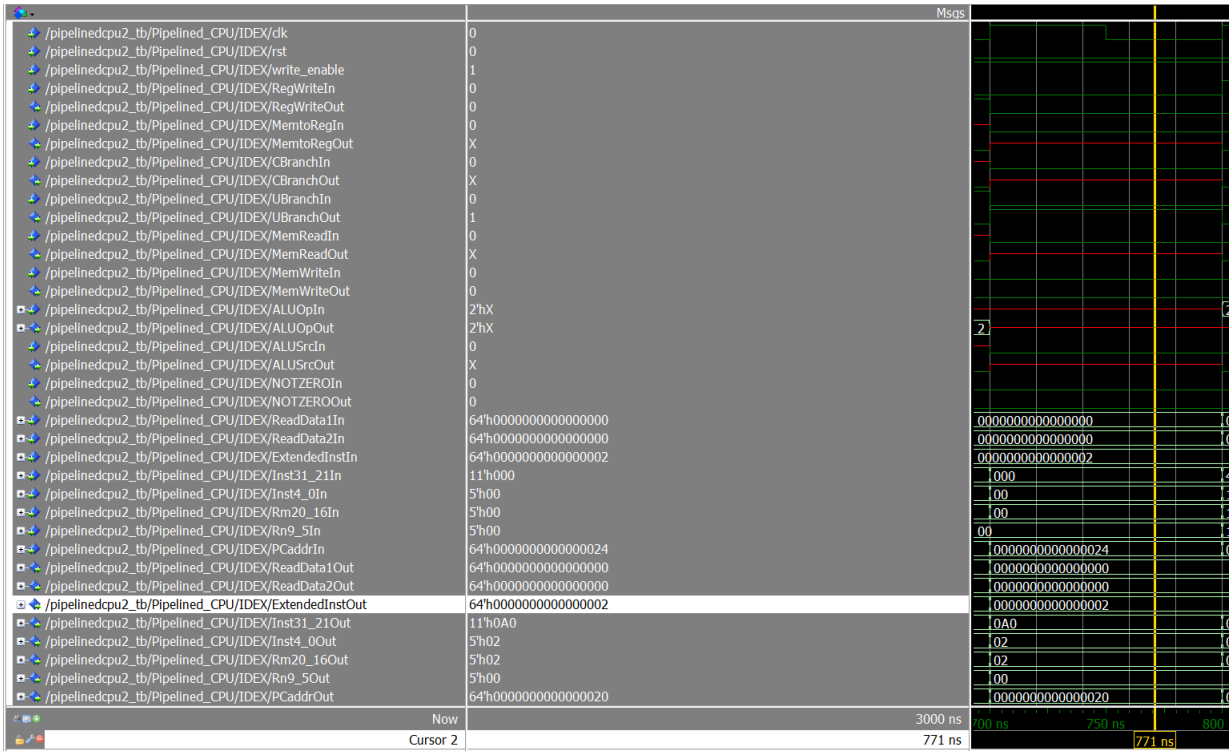

Figure 30: ID/EX Signals of Cycle eight

In cycle eight, ID decodes flushed instruction ten(NOP).
Signals are changed according to flushed instruction ten at 700 ns, we can see its CPU control signals in the above figure.

Signal values of instruction nine are passed to EX stage.

c EX

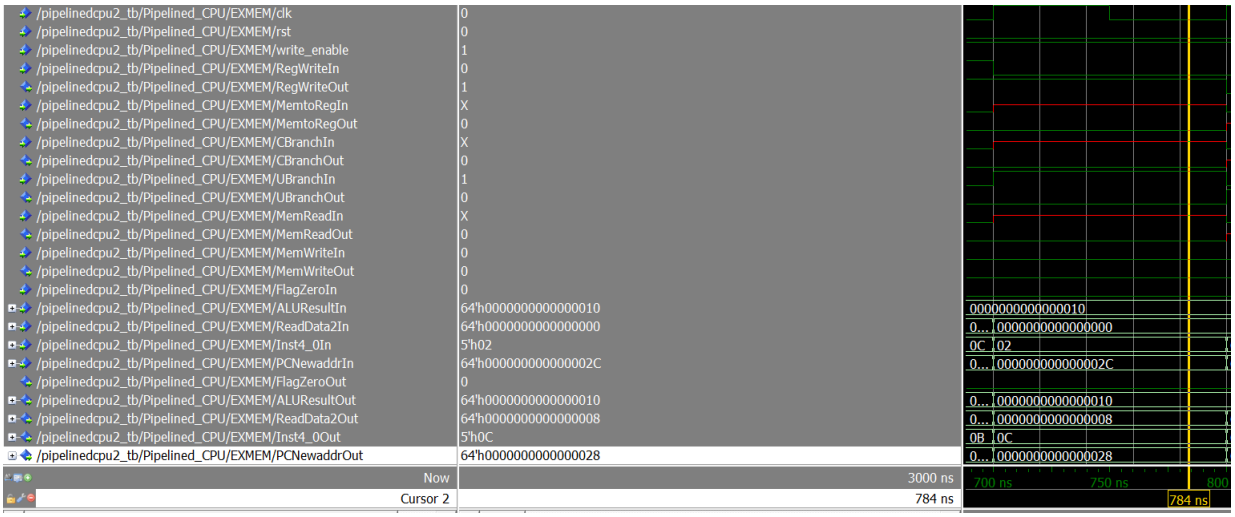

Figure 31: EX/MEM Signals of Cycle eight

In cycle eight, EX is executing operations of instruction nine
(B 2: 0001 0100 0000 0000 0000 0000 0000 0010). Signals are changed according to instruction nine at 700 ns.
Results of instruction eight are passed to MEM stage.

d MEM

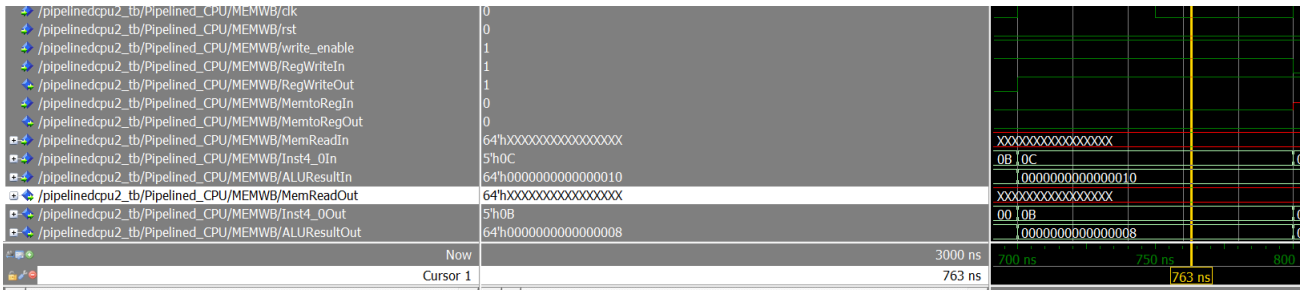

Figure 32: EX/MEM Signals of Cycle eight

In cycle eight, MEM executes for instruction eight. Since instruction eight is a ADD instruction, we do nothing in MEM stage, however we get the ALUResult of additon here, which is 'h10=16.
Results of instruction seven are passed to WB stage.

e WB

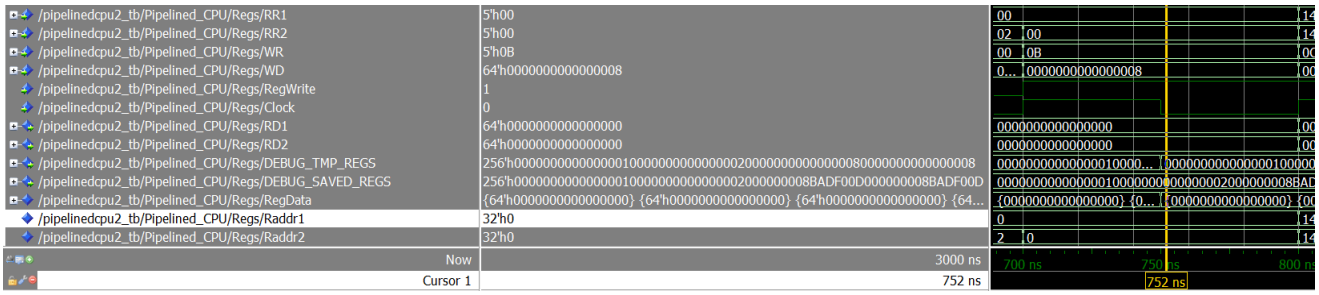

Figure 33: Registers Signals of Cycle eight

In cycle eight, WB writes value into registers according to instruction seven. Control signal RegWrite is '1', write register(WR) is 'h0B=$X11, write data(WD) is 'h08=8. We are writing the result of addition in instruction seven into $X11.

f Debug Signals

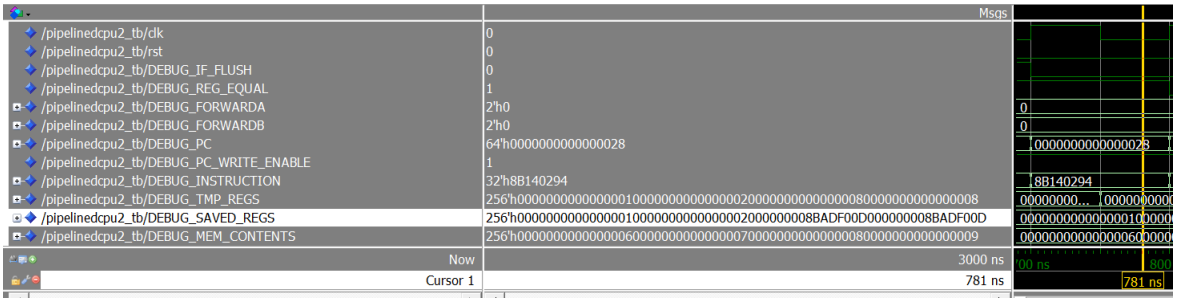

Figure 34: Debug Signals of Cycle eight

During cycle eight, $X11 is changed from 'h04 to 'h08 at 750 ns.

10: Cycle Nine

a IF

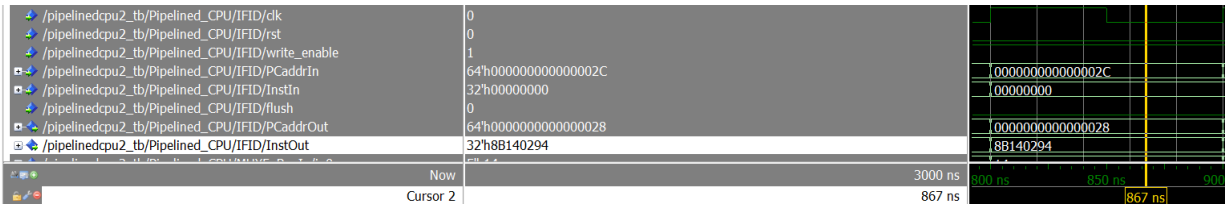

Figure 35: IF/ID Signals of Cycle nine

In cycle nine, IF fetches NOP. We can see PC address is 0x2C and instruction eleven is passed to ID stage.
No work for IF stage.

b ID

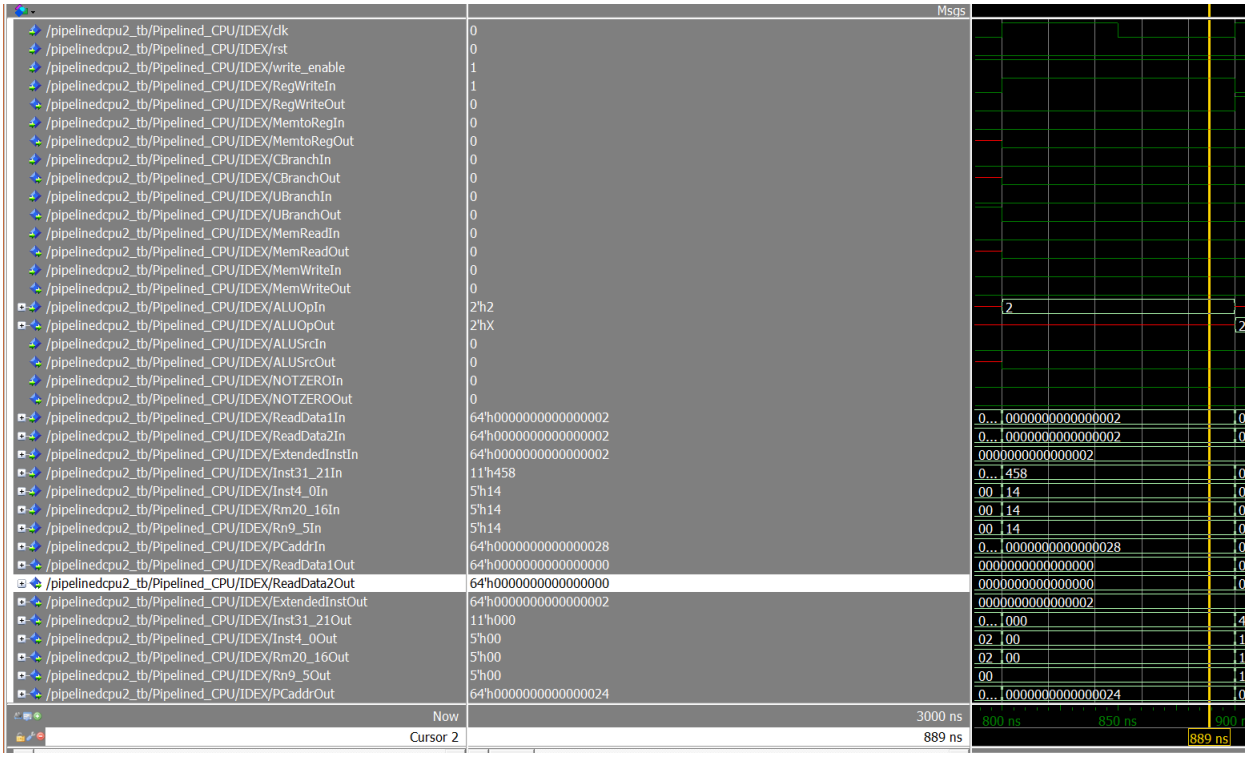

Figure 36: ID/EX Signals of Cycle nine

In cycle nine, ID decodes instruction eleven(ADD X20, X20, X20: 1000 1011 0001 0100 0000 0010 1001 0100). Signals are changed according to instruction eleven at 800 ns. Signal values

of flushed instruction ten are passed to EX stage.

c EX

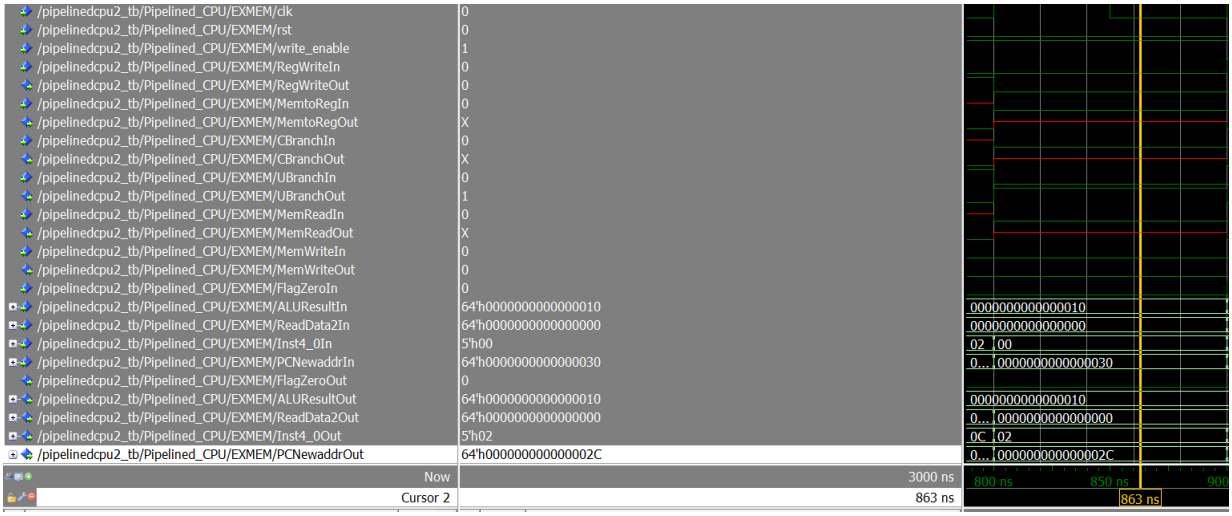

Figure 37: EX/MEM Signals of Cycle nine

In cycle nine, EX is executing operations of flushed instruction ten.
Signals are changed according to flushed instruction ten at 800 ns.
Results of instruction nine are passed to MEM stage.

d MEM

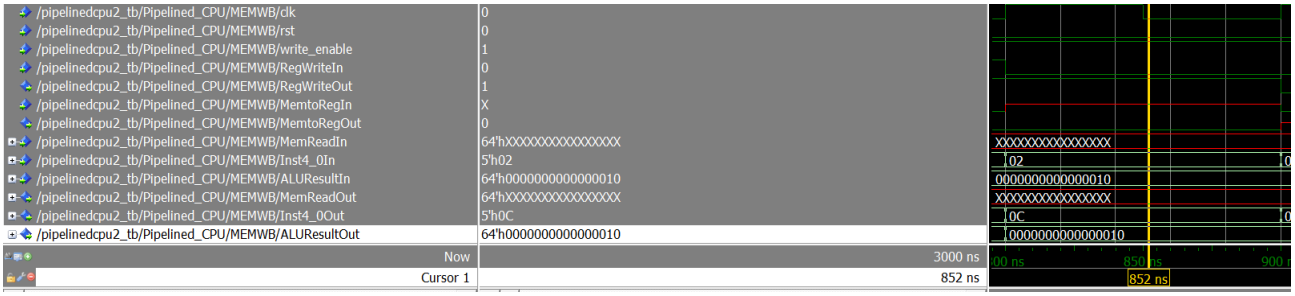

Figure 38: Data Memory Signals of Cycle nine

In cycle nine, MEM is executing operations of instruction nine
(B 2: 0001 0100 0000 0000 0000 0000 0000 0010). Since instruction nine is a B instruction, we do nothing in MEM stage.
Results of instruction eight are passed to WB stage.

e WB

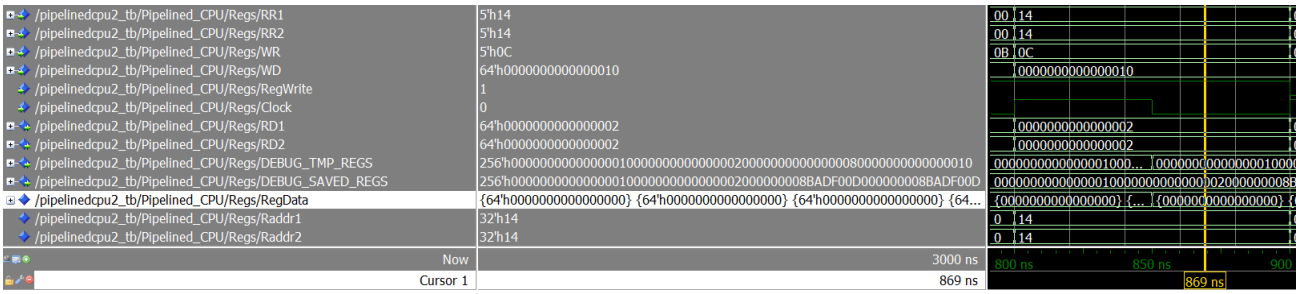

Figure 39: Registers Signals of Cycle nine

In cycle nine, WB writes value into registers according to instruction eight. Control signal RegWrite is '1', write register(WR) is 'h0C=$X12, write data(WD) is 'h10=16. We are writing the result of addition in instruction eight into $X12.

f Debug Signals

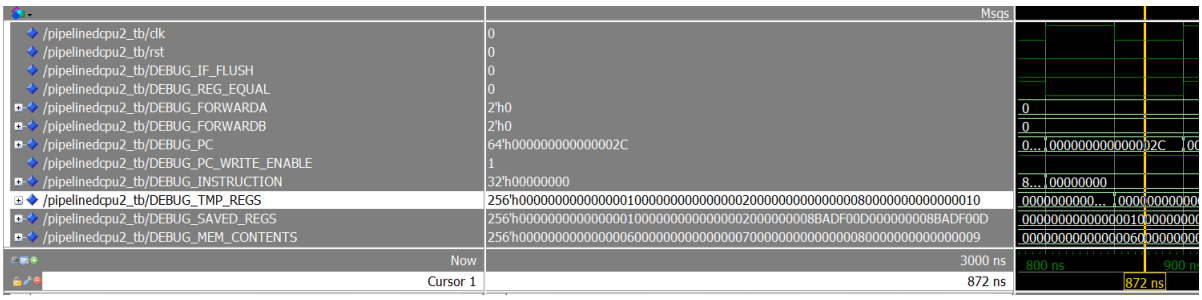

Figure 40: Debug Signals of Cycle nine

During cycle nine, $X12 is changed from 'h8 to 'h10 at 850 ns.

11: Cycle Ten

No work for IF stage.

a ID

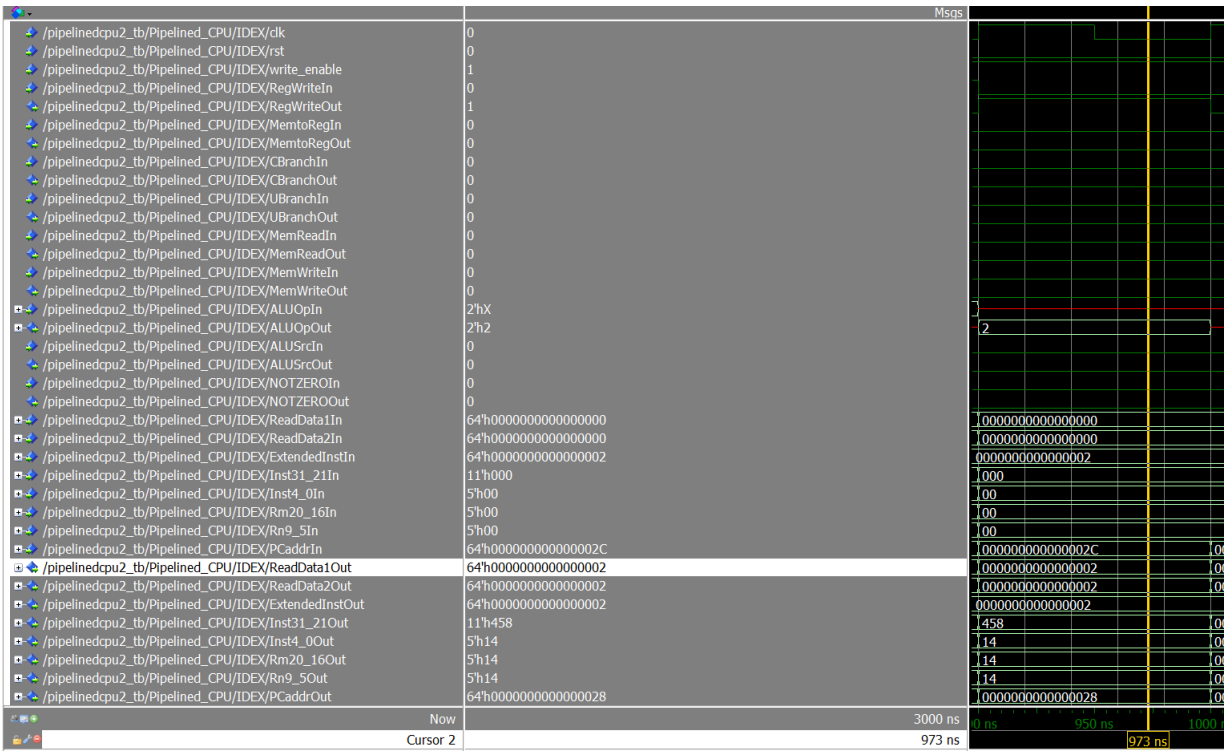

Figure 41: ID/EX Signals of Cycle ten

In cycle ten, ID decodes instruction twelve(NOP). Signals are changed according to instruction twelve at 900 ns. Signal values of instruction eleven are passed to EX stage.

b EX

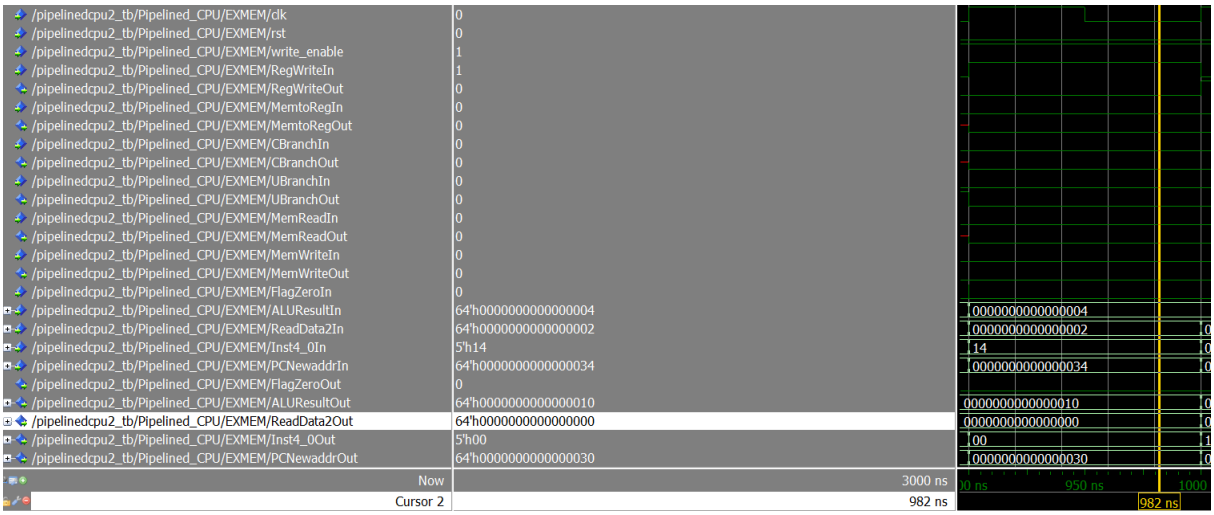

Figure 42: EX/MEM Signals of Cycle ten

In cycle ten, EX is executing operations of instruction eleven
(ADD X20, X20, X20: 1000 1011 0001 0100 0000 0010 1001 0100). Signals are changed according to instruction eleven at 900 ns, we can see the ALUResult is 'h04=4, which is $X20+$X20 = 2+2 = 4.
Results of flushed instruction ten are passed to MEM stage.

c MEM

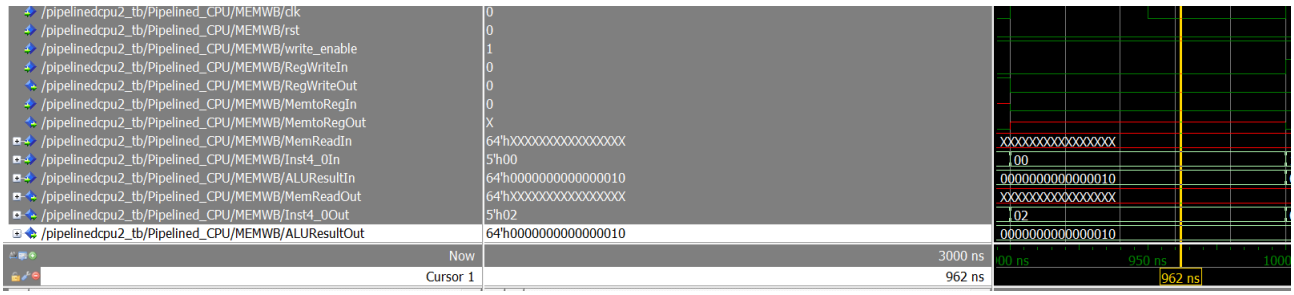

Figure 43: Data Memory Signals of Cycle ten

In cycle ten, MEM executes for flushed instruction ten. We do nothing here.
Results of instruction nine are passed to WB stage.

d WB

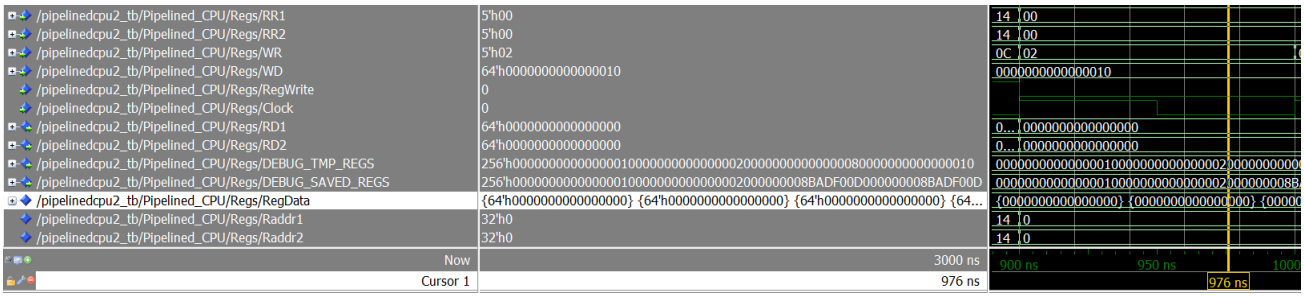

Figure 44: Registers Signals of Cycle ten

In cycle ten, WB writes value into registers according to instruction nine. This is a B instruction, Control signal RegWrite is '0', we do not write into registers.

12: Cycle Eleven

No work for IF and ID stages.

a EX

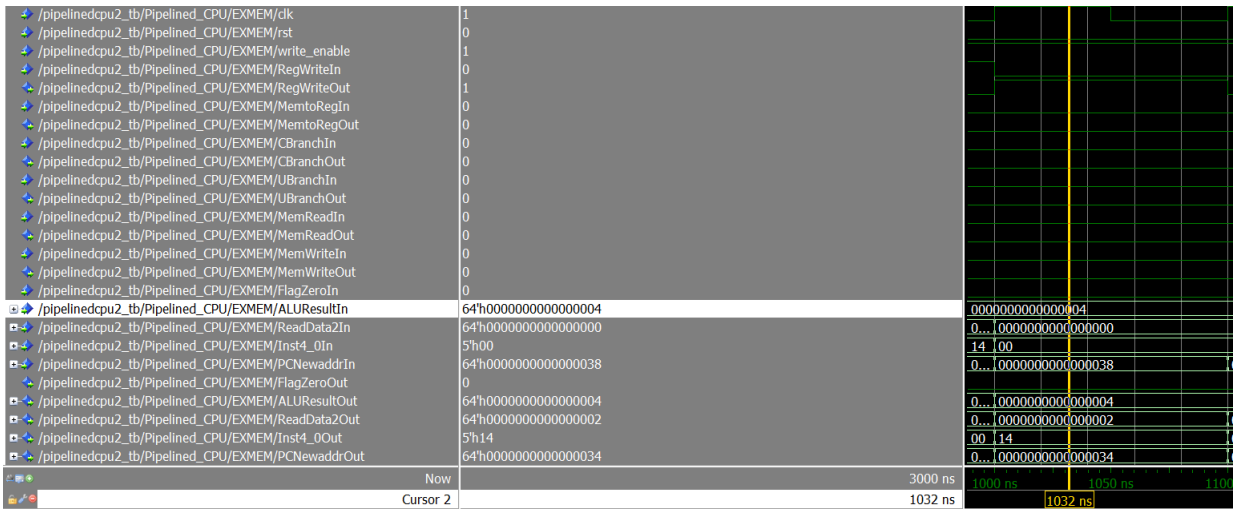

Figure 45: EX/MEM Signals of Cycle ten

In cycle ten, EX executes operations of instruction twelve(NOP). Signals are changed according to instruction eleven at 1000 ns.
Results of flushed instruction eleven are passed to MEM stage.

b MEM

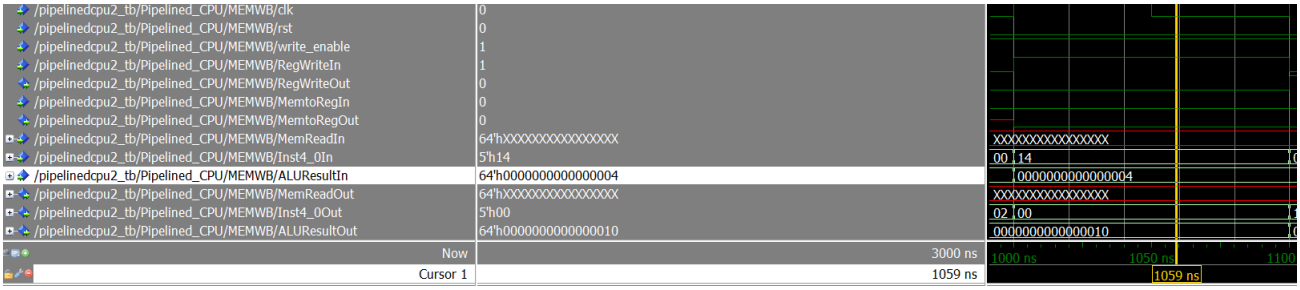

Figure 46: Data Memory Signals of Cycle eleven

In cycle eleven, MEM executes for instruction eleven(ADD X20, X20, X20: 1000 1011 0001 0100 0000 0010 1001 0100). Since instruction eleven is a ADD instruction, we do nothing in MEM stage, however we get the ALUResult of additon here, which is 'h04=4.
Results of flushed instruction ten are passed to WB stage.

c WB

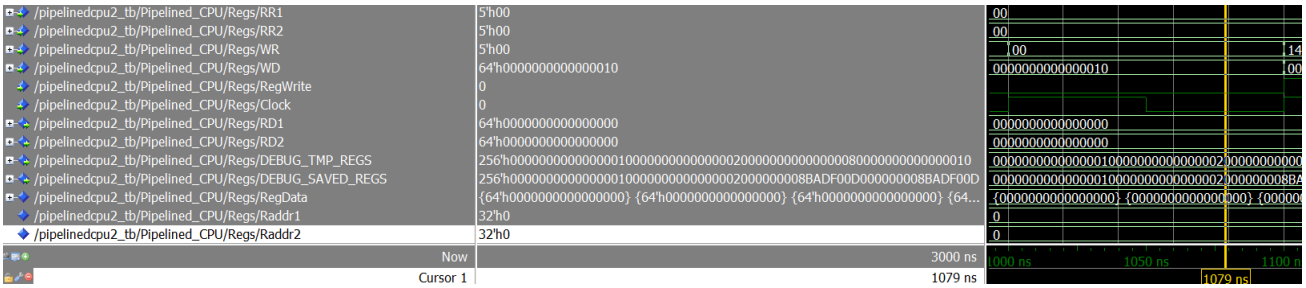

Figure 47: Registers Signals of Cycle eleven

In cycle eleven, WB writes value into registers according to flushed instruction ten. We do nothing here.

13: Cycle Twelve

No work for IF, ID and EX stages.

a MEM

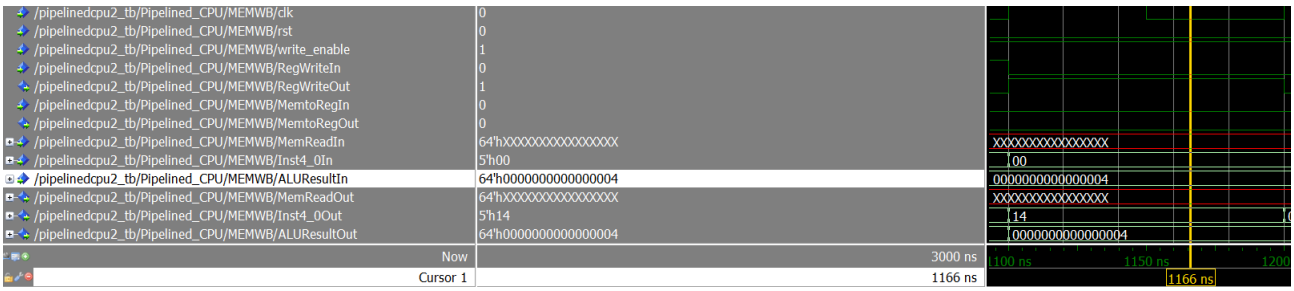

Figure 48: Data Memory Signals of Cycle twelve

In cycle twelve, MEM is executing operations of instruction twelve(NOP). We do nothing here. Results of instruction eleven are passed to WB stage.

b WB

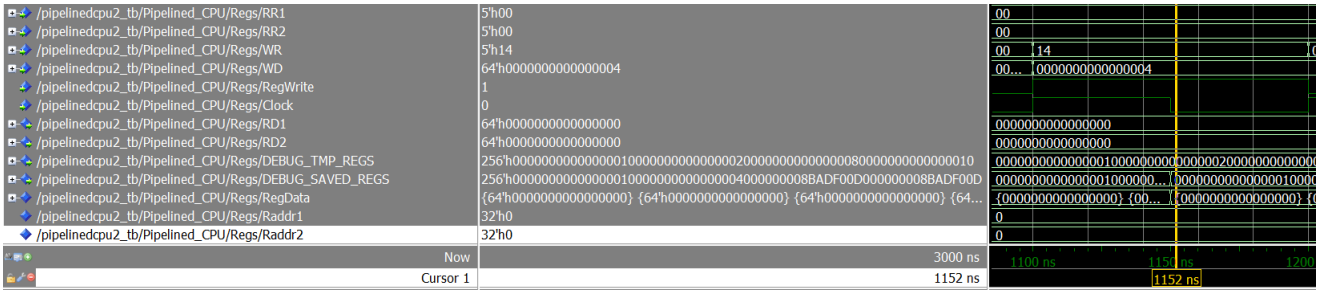

Figure 49: Registers Signals of Cycle twelve

In cycle twelve, WB writes value into registers according to instruction eleven. Control signal RegWrite is '1', write register(WR) is 'h14=$X20, write data(WD) is 'h04=4. We are writing the result of addition in instruction eleven into $X20.

c Debug Signals

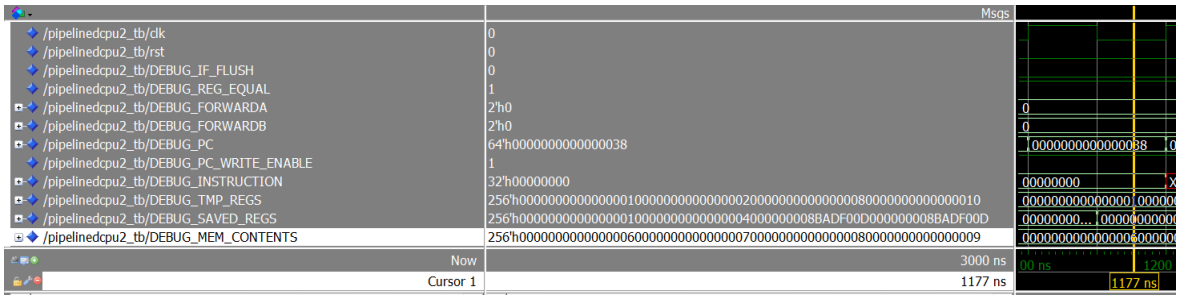

Figure 50: Debug Signals of Cycle twelve

During cycle twelve, $X20 is changed from 'h02 to 'h04 at 1050 ns.

14: Sum

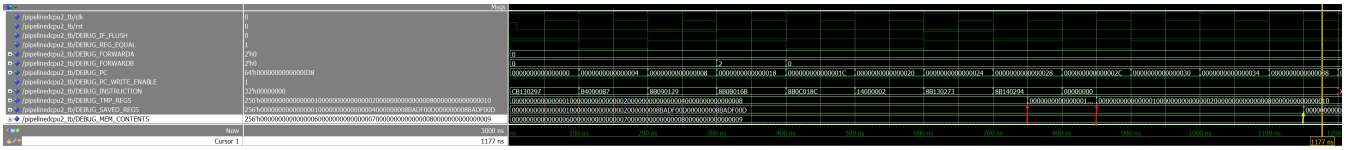

Figure 51: Debug Signals

TMP_REGS changes two times, at 750ns and 850ns(During cycle 8 and cycle 9).
Saved_REGS changes one time, at 1150 ns(During cycle 12).
MEM does not change.
The final states are, $X9 = 1, $X10 = 2, $X11 = 8, $X12 = 'h10=16, $X19 = 1, $X20 = 4, $X21 = 0x8BADF00D, $X22 = 0x8BADF00D, DMEM(0x0) = 0x09, DMEM(0x8) = 0x08, DMEM(0x10) = 0x07, DMEM(0x18) = 0x06.