Kaicheng Yang

I would like to use one late token for lab 5.

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. reg_IFID.vhd
19. reg_IDEX.vhd
20. reg_EXMEM.vhd
21. reg_MEMWB.vhd
22. registers.vhd
23. lab5_dmem.vhd
24. lab5_imem.vhd
25. pipelinedcpu1.vhd
26. pipelinedcpu1_tb.vhd
27. test_nop.vhd(testing inserting nop)

b Design Decisions

CBNZ implementation

I create a new cpu control signal, NOTZERO, for CBNZ implementation. When the NOTZERO is '0', it will pass the value of signal 'Zero' to the and gate. And when the NOTZERO is '1', it will pass the value of '!Zero' to the and gate. I use (NOTZERO XOR ALU_Zero) to implement this logic.

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

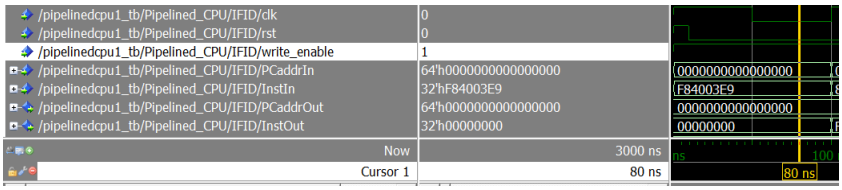

Figure 1: IF Signals of Cycle one

In cycle one, IF is fetching instruction one
(LDUR X9, [XZR, 0]: 11111000010000000000001111101001). We can see PC address is 0x00 and nothing is passed to ID stage.

3: Cycle Two

a IF

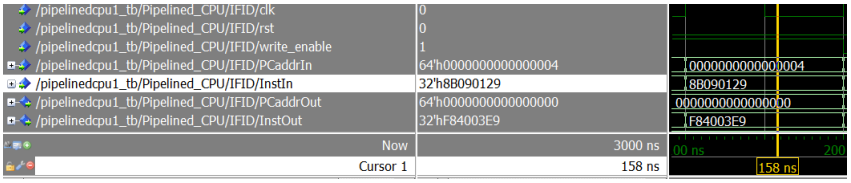

Figure 2: IF Signals of Cycle two

In cycle two, IF is fetching instruction two
(ADD X9, X9, X9: 10001011000010010000000100101001). We can see PC address is 0x04 and instruction one is passed to ID stage.

b ID

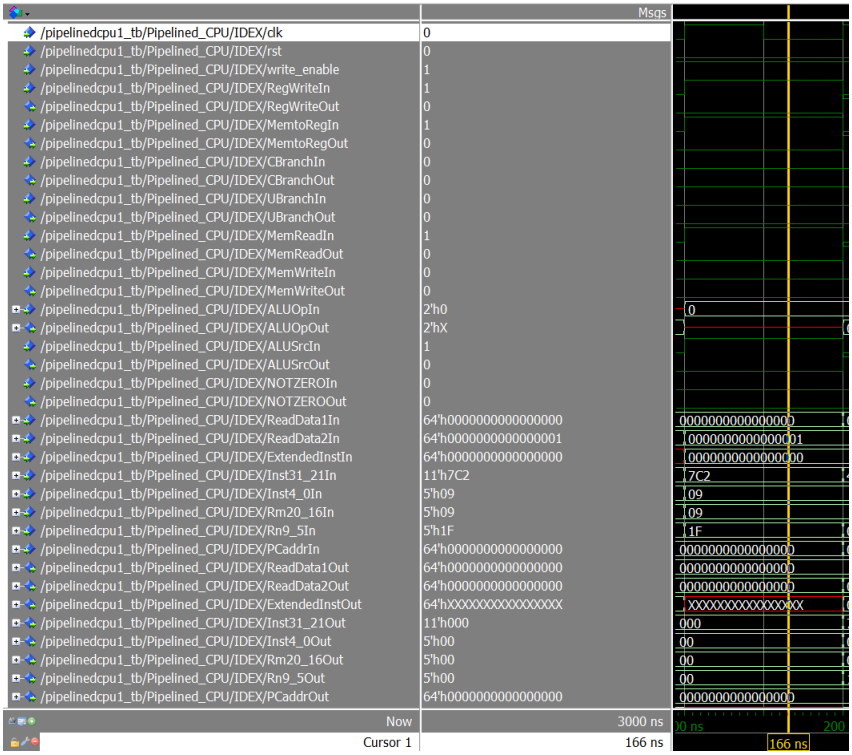

Figure 3: ID/EX Signals of Cycle two

In cycle two, ID is decoding instruction one
(LDUR X9, [XZR, 0]: 11111000010000000000001111101001). Signals are changed according to instruction one at 100 ns, we can see its CPU control signals in the above figure. And Rn is 'h1F=31, the address of XZR; Rt is 'h09=9, the address of $X9; offset is 0. No signal value is passed to EX stage.

4: Cycle Three

a IF

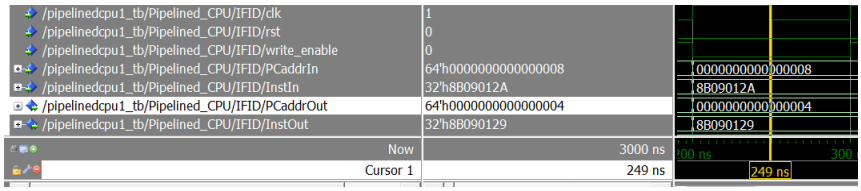

Figure 4: IF/ID Signals of Cycle three

In cycle three, IF is fetching instruction three
(ADD X10, X9, X9: 10001011000010010000000100101010). We can see PC address is 0x08 and instruction two is passed to ID stage.

b ID

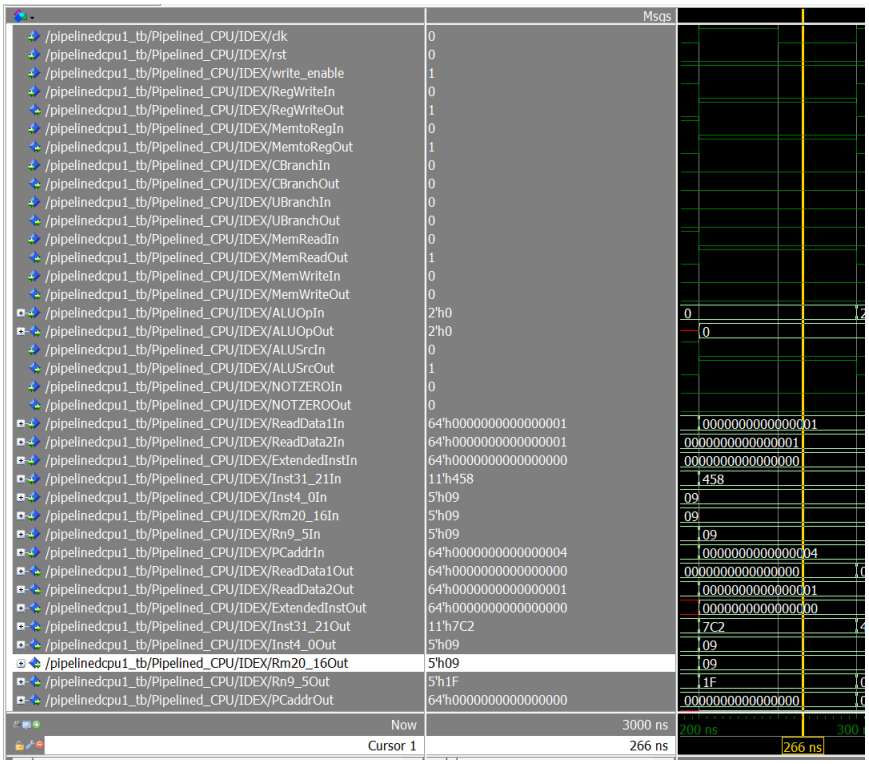

Figure 5: ID/EX Signals of Cycle three

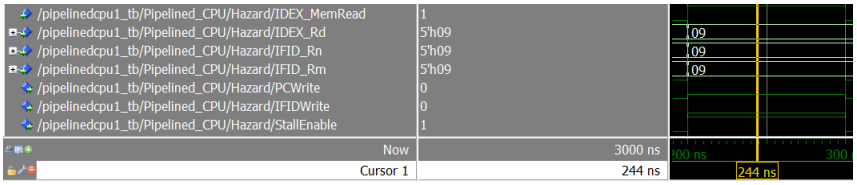

Figure 6: Hazard Control Signals of Cycle three

In cycle three, ID is decoding instruction two
(ADD X9, X9, X9: 10001011000010010000000100101001). Signals are changed according to instruction two at 200 ns, we can see its CPU control signals in the above figure. And Rm is 'h09=9; Rn is 'h09=9; Rd is 'h09=9. Signal values of instruction one are passed to EX stage. And we detect data hazard here, so input signals of ID/EX registers are all '0'. The next cycle would be stall.

c EX

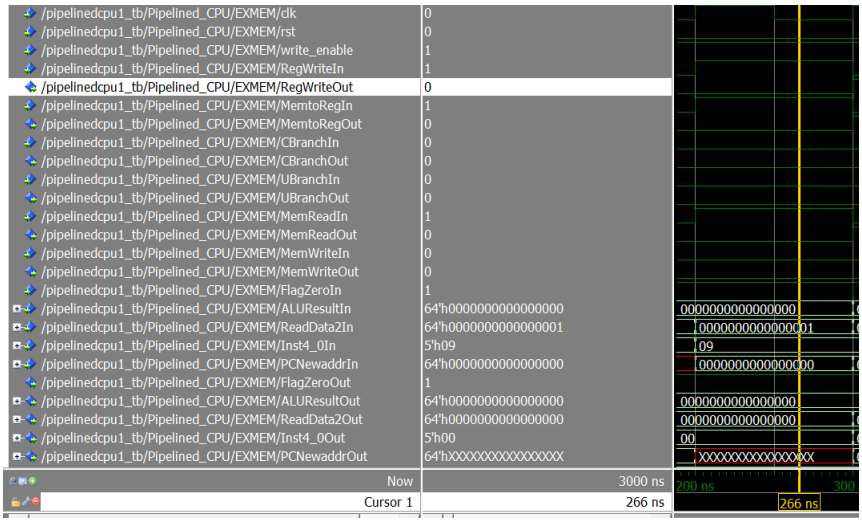

Figure 7: EX/MEM Signals of Cycle three

In cycle three, EX is executing operations of instruction one
(LDUR X9, [XZR, 0]: 11111000010000000000001111101001). Signals are changed according to instruction one at 200 ns, we can see the ALUResult is 'h00=0, which is the Address of Data Memory to write in.
Nothing is passed to MEM stage.

5: Cycle Four

a IF

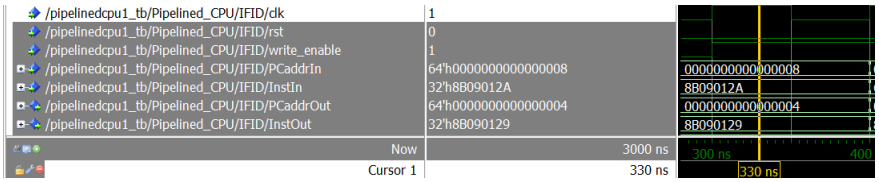

Figure 8: IF/ID Signals of Cycle four

In cycle four, pipeline is stall. We can see PC address is still 0x08.
And we can see write_enable is changed from 0 to 1 at 300 ns, which means instruction two is still passed to ID stage in this cycle.

b ID

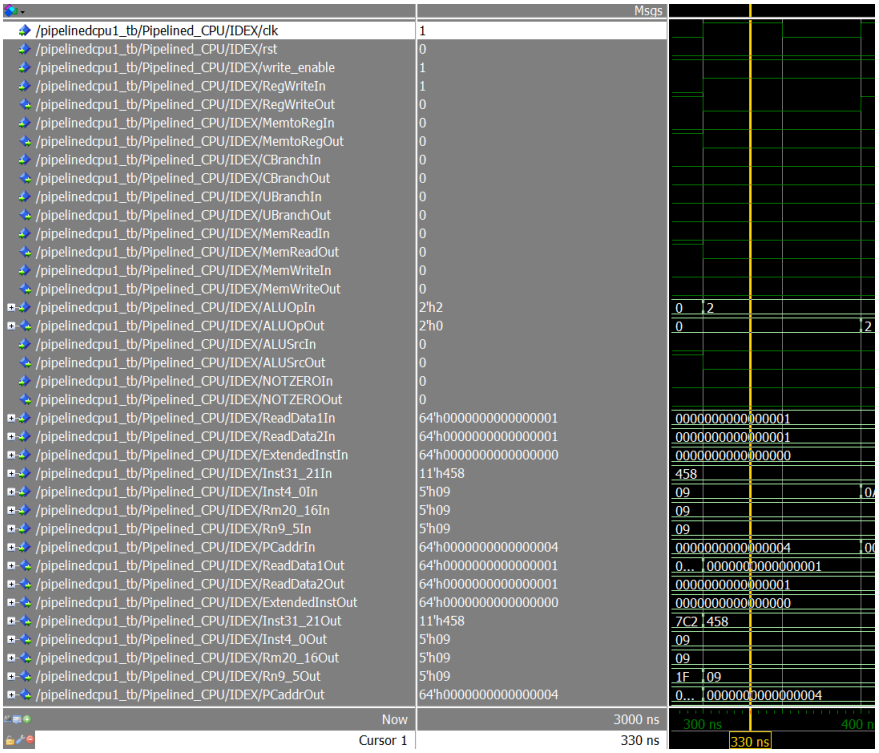

Figure 9: ID/EX Signals of Cycle four

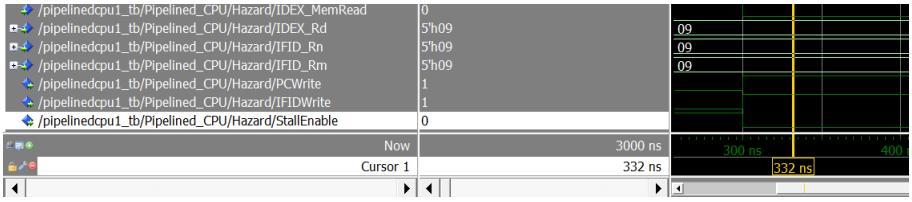

Figure 10: Hazard Control Signals of Cycle four

In cycle four, pipeline is stall. ID is still decoding instruction two
(ADD X9, X9, X9: 10001011000010010000000100101001). We can see the StallEnable is '1' during cycle three, and is changed to '0' at the beginning of cycle four.
All the control signals passed to ID/EX register are set according to instruction two. Signal values of NOP are passed to EX stage.

c EX

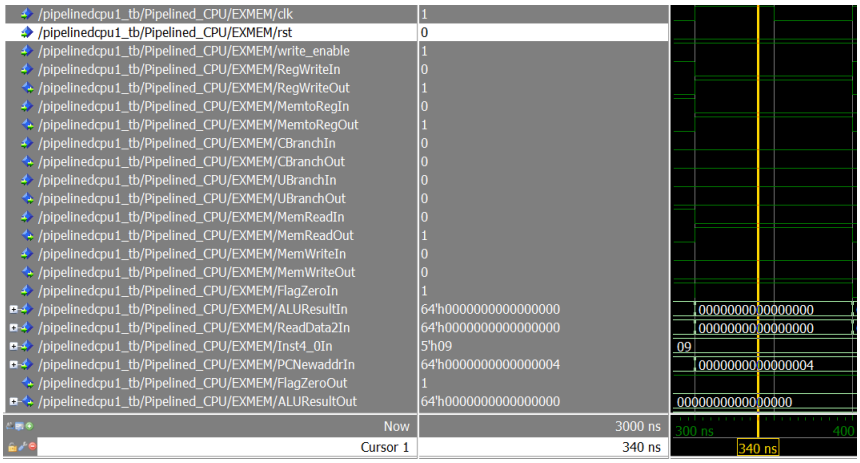

Figure 11: EX/MEM Signals of Cycle four

In cycle four, EX is executing operations of instruction two
(ADD X9, X9, X9: 10001011000010010000000100101001). However due to data hazard, we flushed instruction two in cycle four, so we are actually doing nothing here.
Results of instruction one are passed to MEM stage.

d MEM

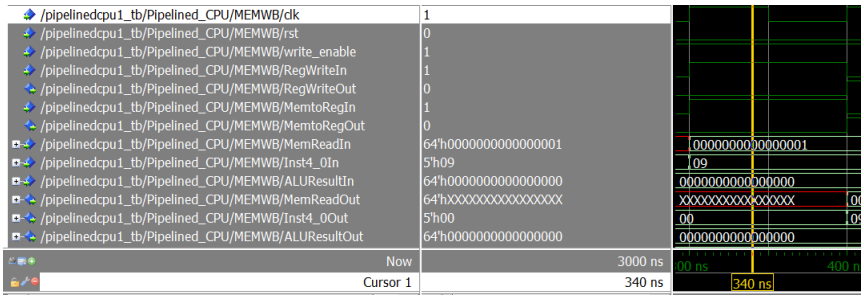

Figure 12: EX/MEM Signals of Cycle four

In cycle four, MEM is executing operations of instruction one
(LDUR X9, [XZR, 0]: 11111000010000000000001111101001). We load DMEM[0-7], which is MemReadIn='h01, from the data memory.
Nothing is passed to WB stage.

6: Cycle Five

a IF

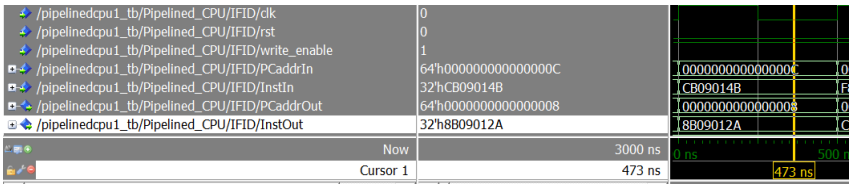

Figure 13: IF/ID Signals of Cycle five

In cycle five, IF is fetching instruction four
(SUB X11, X10, X9: 11001011000010010000000101001011). We can see PC address is 0x0C and instruction three is passed to ID stage.

b ID

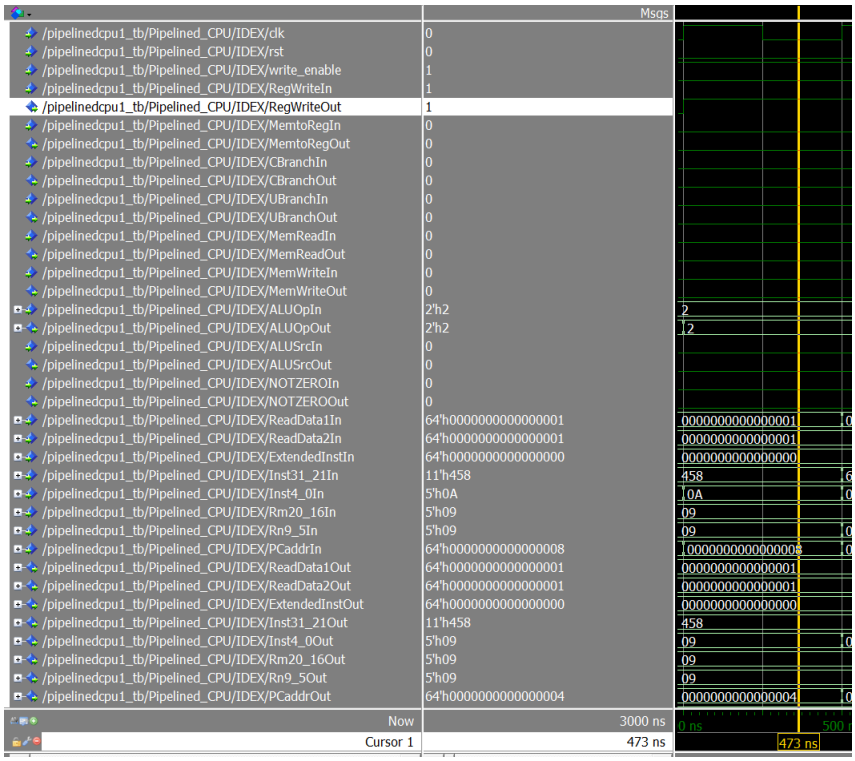

Figure 14: ID/EX Signals of Cycle five

In cycle five, ID is decoding instruction three
(ADD X10, X9, X9: 10001011000010010000000100101010). Signals are changed according to instruction three at 400 ns, we can see its CPU control signals in the above figure. And Rm

is 'h09=9; Rn is 'h09=9; Rd is 'h0A=10. Signal values of instruction two are passed to EX stage.

c EX

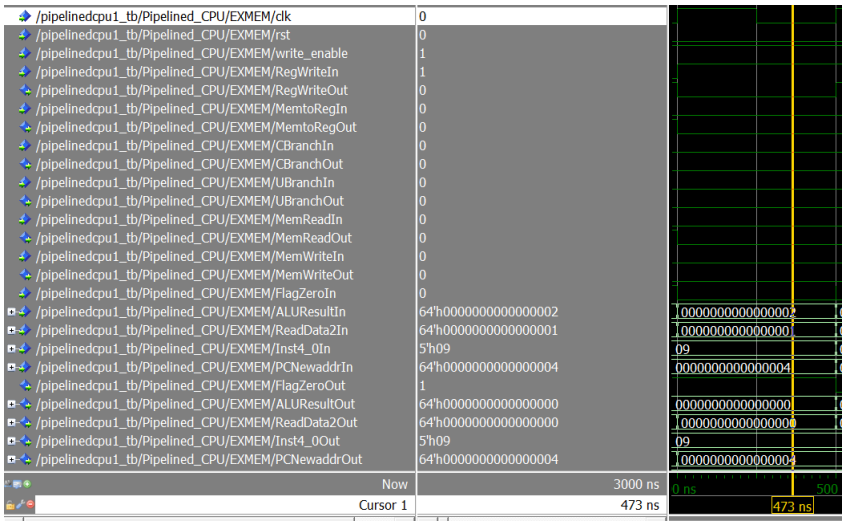

Figure 15: EX/MEM Signals of Cycle five

In cycle five, EX is executing operations of instruction two
(ADD X9, X9, X9: 10001011000010010000000100101001). Signals are changed according to instruction two at 400 ns, we can see the ALUResult is 'h02=2, which is \$X9+\$X9 = 1+1 = 2
Results of flushed instruction two are passed to MEM stage.

d MEM

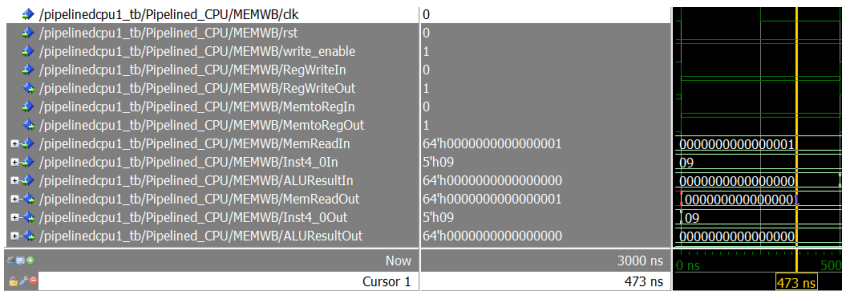

Figure 16: EX/MEM Signals of Cycle five

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

e WB

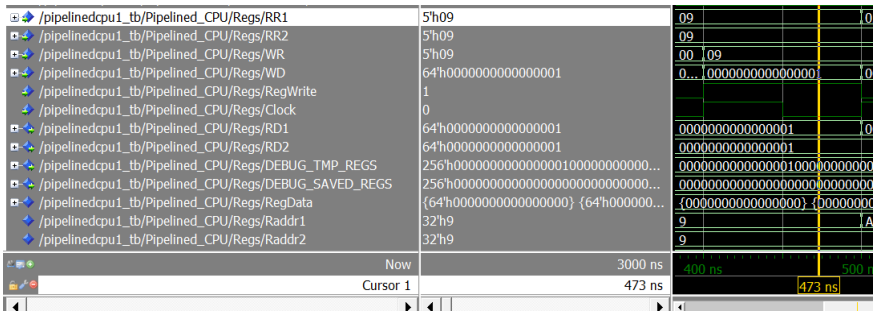

Figure 17: Registers Signals of Cycle five

In cycle five, WB is writing value into registers according to instruction one. Control signal RegWrite is '1', write register(WR) is 'h09, write data(WD) is 'h01. We are writing the value of DMEM[0-7] into $X9.

7: Cycle Six

a IF

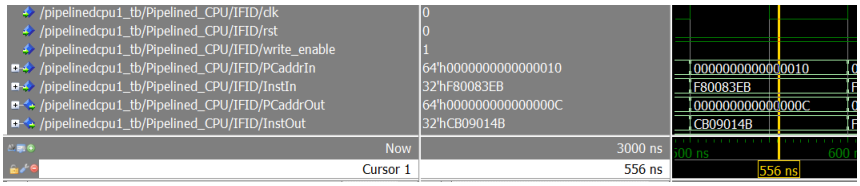

Figure 18: IF/ID Signals of Cycle six

In cycle six, IF is fetching instruction five
(STUR X11, [XZR, 8]: 11111000000000001000001111101011). We can see PC address is 0x10 and instruction four is passed to ID stage.

b ID

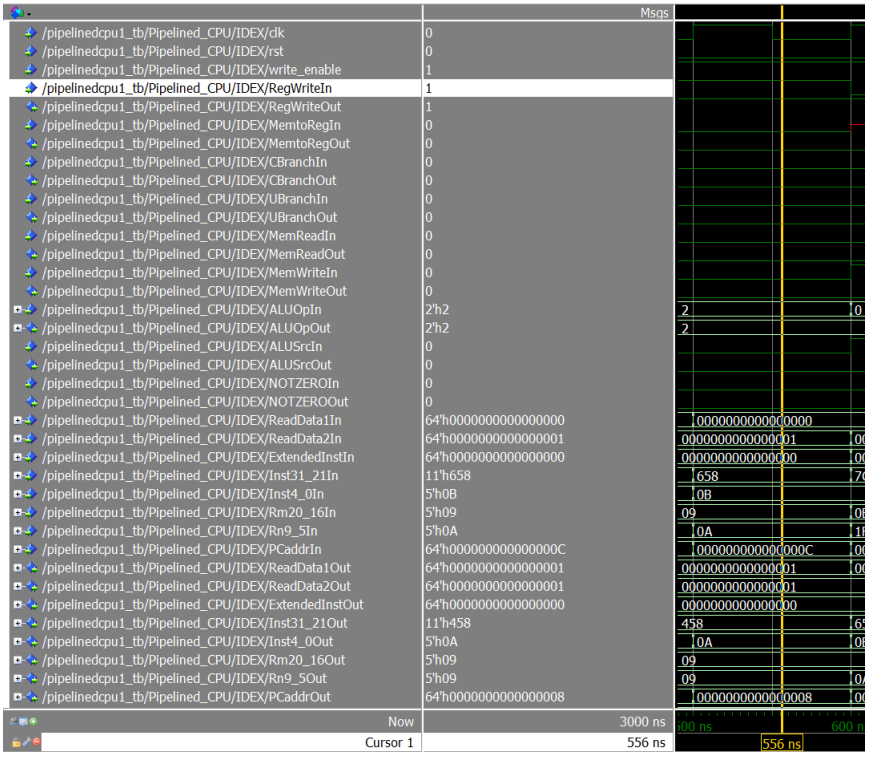

Figure 19: ID/EX Signals of Cycle six

In cycle six, ID is decoding instruction four
(SUB X11, X10, X9: 11001011000010010000000101001011). Signals are changed according to instruction four at 500 ns, we can see its CPU control signals in the above figure. And Rm is

'h09=9; Rn is 'h0A=10; Rd is 'h0B=11. Signal values of instruction three are passed to EX stage.

c EX

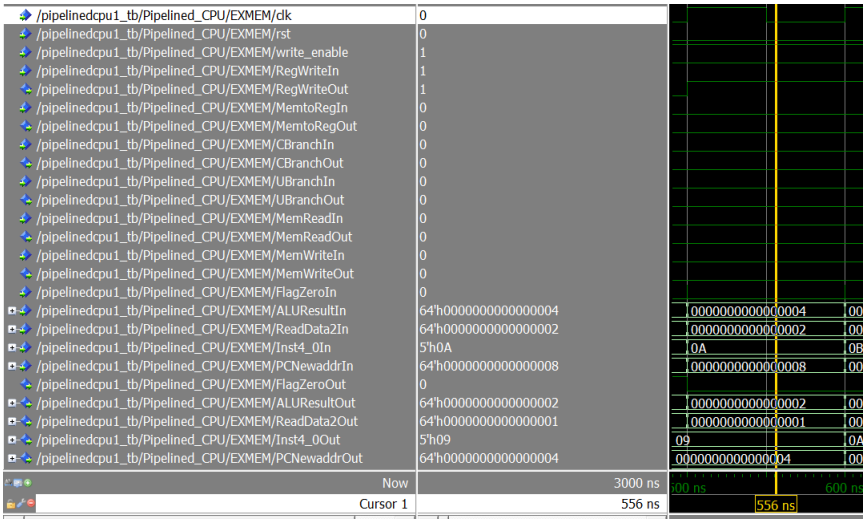

Figure 20: EX/MEM Signals of Cycle six

In cycle six, EX is executing operations of instruction three
(ADD X10, X9, X9: 10001011000010010000000100101010). Signals are changed according to instruction three at 500 ns, we can see the ALUResult is 'h04=4, which is $X9+$X9 = 2+2 = 4
Results of instruction two are passed to MEM stage.

d MEM

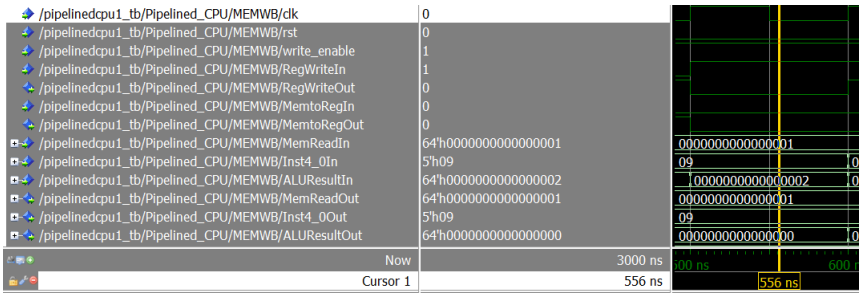

Figure 21: EX/MEM Signals of Cycle six

In cycle six, MEM is executing operations of instruction two. Since instruction two is an ADD instruction, we do nothing in MEM stage, however we get the ALUResult of addition here,

which is 'h02=2.
Nothing is passed to WB stage, last instruction is flushed.

e WB

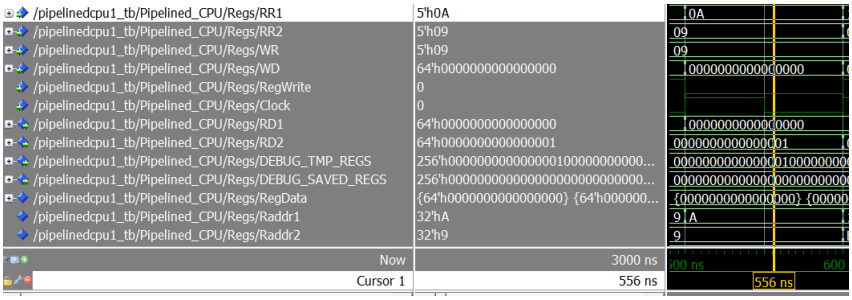

Figure 22: Registers Signals of Cycle six

In cycle six, WB is writing value into registers according to flushed instruction two. Control signal RegWrite is '0', we actually do nothing here.

8: Cycle Seven

a IF

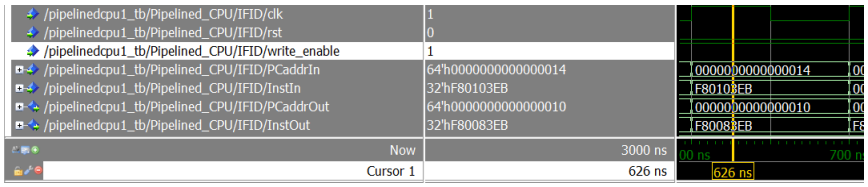

Figure 23: IF/ID Signals of Cycle seven

In cycle seven, IF is fetching instruction six
(STUR X11, [XZR, 16]: 11111000000000100000001111101011). We can see PC address is 0x14 and instruction five is passed to ID stage.

b ID

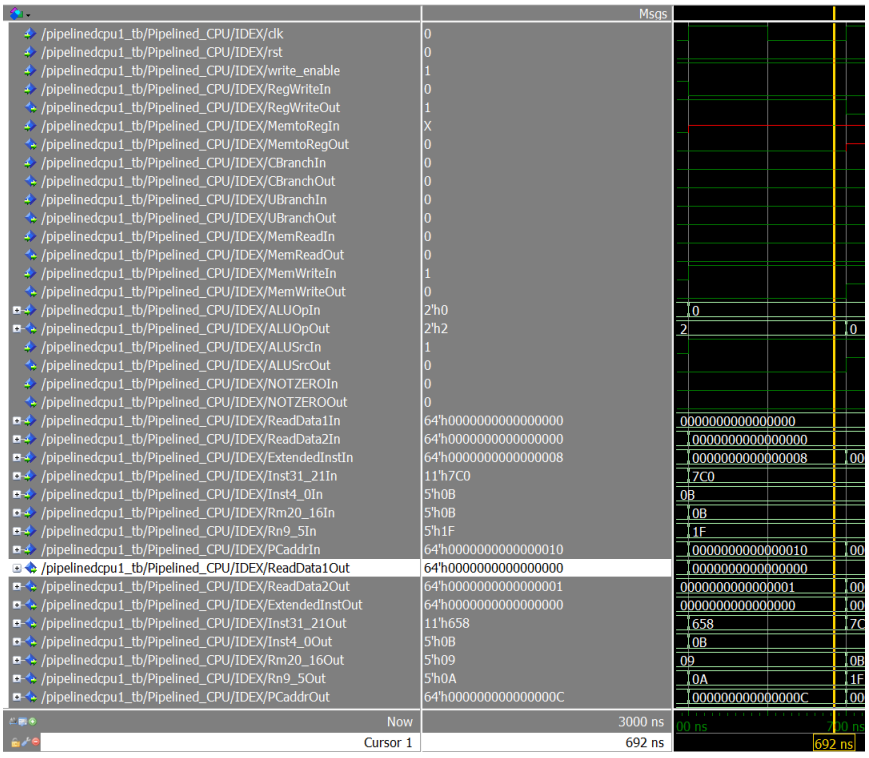

Figure 24: ID/EX Signals of Cycle seven

In cycle seven, ID is decoding instruction five
(STUR X11, [XZR, 8]: 11111000000000001000001111101011). Signals are changed according to instruction five at 600 ns, we can see its CPU control signals in the above figure. And Rn

is 'h1F=31; Rt is 'h0B=11; offset is 8. Signal values of instruction four are passed to EX stage.

c EX

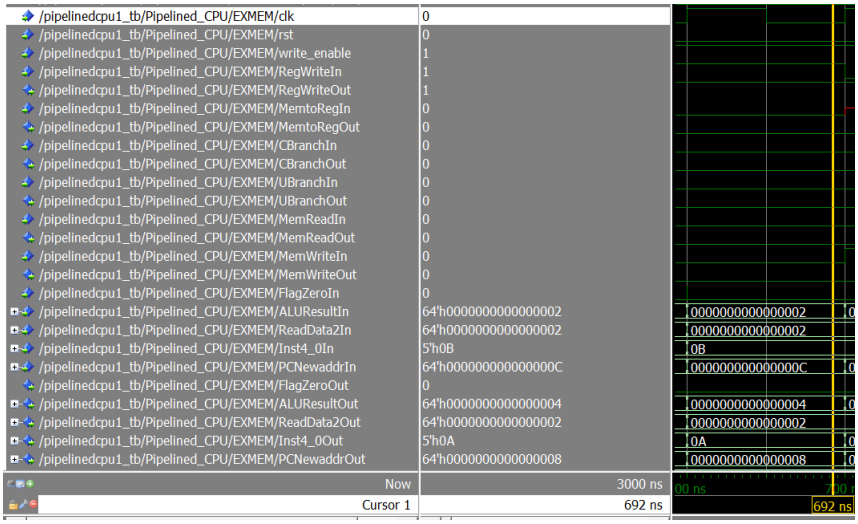

Figure 25: EX/MEM Signals of Cycle seven

In cycle seven, EX is executing operations of instruction four
(SUB X11, X10, X9: 11001011000010010000000101001011). Signals are changed according to instruction four at 600 ns, we can see the ALUResult is 'h02=2, which is $X10-$X9 = 4-2 = 2.
Results of instruction three are passed to MEM stage.

d MEM

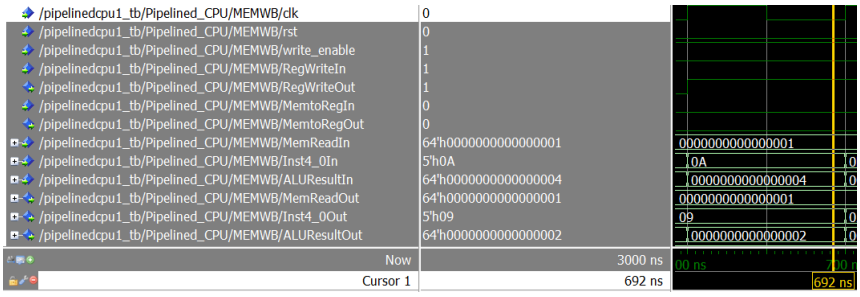

Figure 26: EX/MEM Signals of Cycle seven

In cycle seven, MEM is executing operations of instruction three. Since instruction three is an ADD instruction, we do nothing in MEM stage, however we get the ALUResult of addition here, which is 'h04=4.

Results of instruction two are passed to WB stage.

e WB

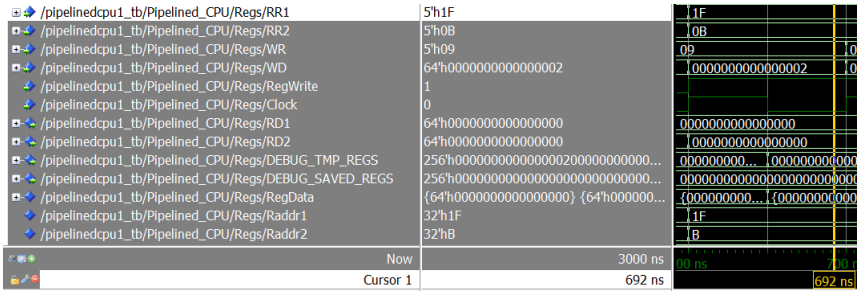

Figure 27: Registers Signals of Cycle seven

In cycle seven, WB is writing value into registers according to instruction two. Control signal RegWrite is '1', write register(WR) is 'h09, write data(WD) is 'h02. We are writing the result of addition in instruction two into $X9.

f Debug Signals

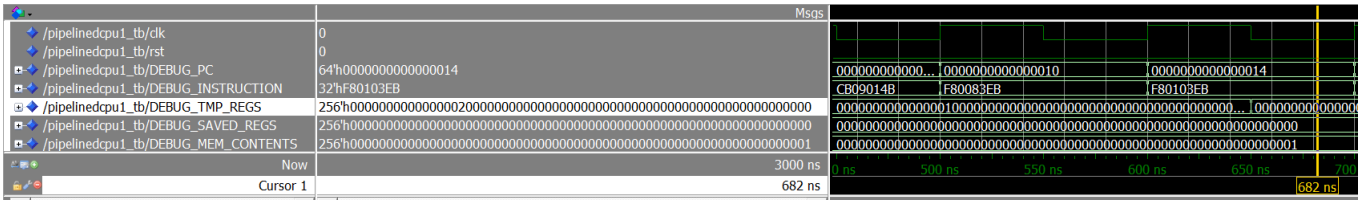

Figure 28: Debug Signals of Cycle seven

During cycle seven, $X9 is changed from 1 to 2 at 650 ns.

9: Cycle Eight

a IF

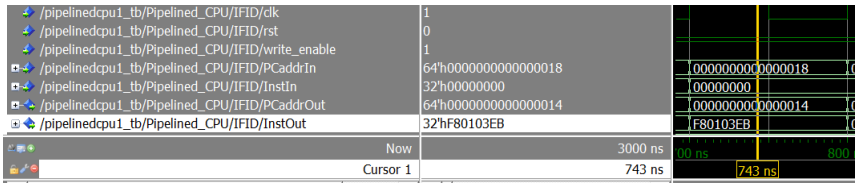

Figure 29: IF/ID Signals of Cycle eight

In cycle eight, IF is fetching instruction seven(NOP). We can see PC address is 0x18 and instruction six is passed to ID stage.

b ID

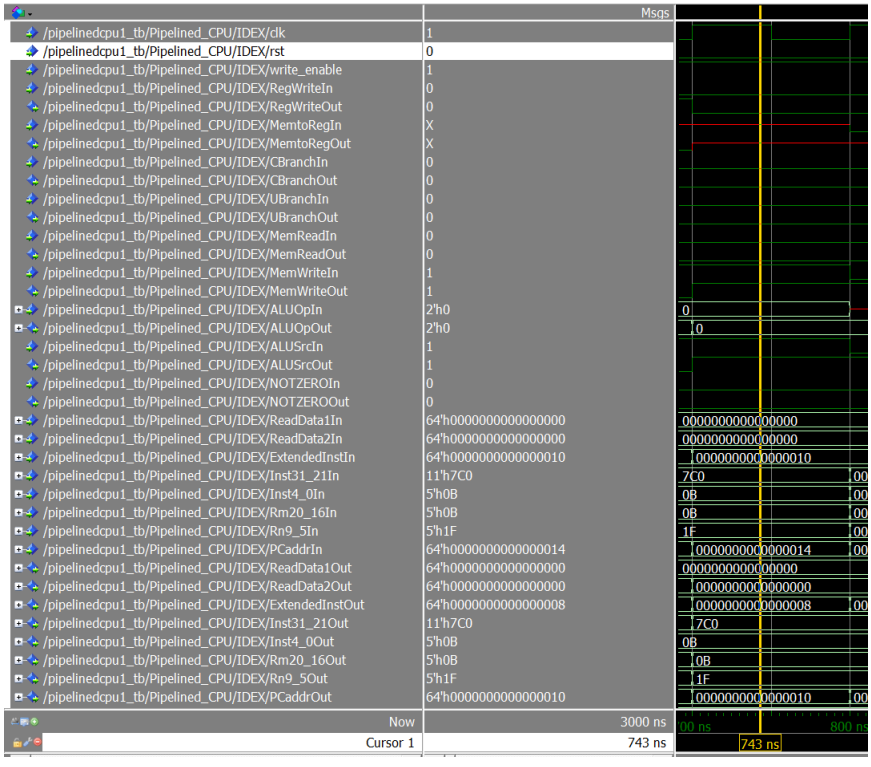

Figure 30: ID/EX Signals of Cycle eight

In cycle eight, ID is decoding instruction six
(STUR X11, [XZR, 16]: 11111000000000100000001111101011). Signals are changed according to instruction six at 700 ns, we can see its CPU control signals in the above figure. And Rn is 'h1F=31; Rt is 'h0B=11; offset is 'h10=16. Signal values of instruction five are passed to

EX stage.

c EX

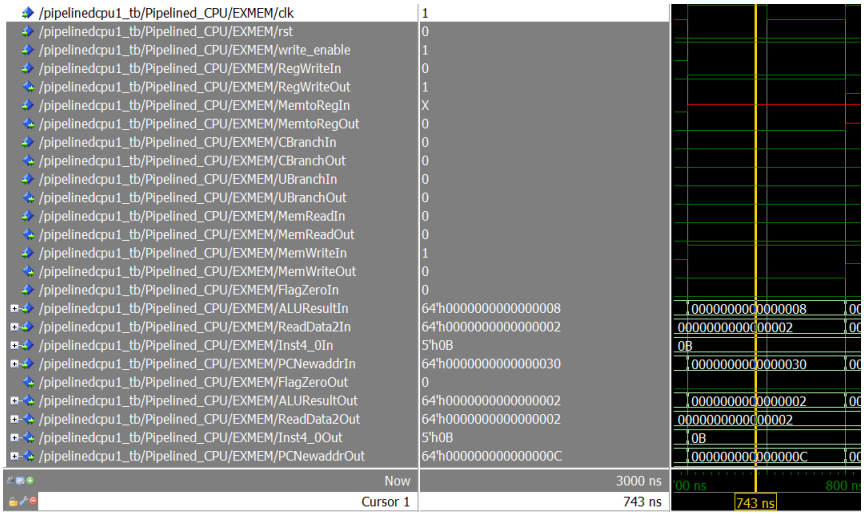

Figure 31: EX/MEM Signals of Cycle eight

In cycle eight, EX is executing operations of instruction five
(STUR X11, [XZR, 8]: 11111000000000001000001111101011). Signals are changed according to instruction five at 700 ns, we can see the ALUResult is 'h08=8, which is $XZR+8 = 0+8 = 8, the address of data memory we would access.
Results of instruction four are passed to MEM stage.

d MEM

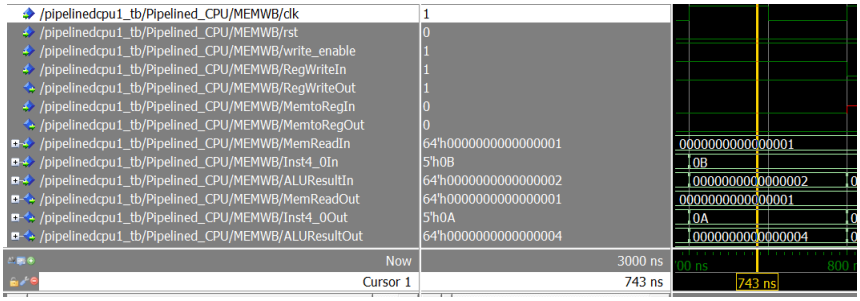

Figure 32: EX/MEM Signals of Cycle eight

In cycle eight, MEM is executing operations of instruction four. Since instruction four is a SUB instruction, we do nothing in MEM stage, however we get the ALUResult of subtraction here, which is 'h02=2.

Results of instruction three are passed to WB stage.

e WB

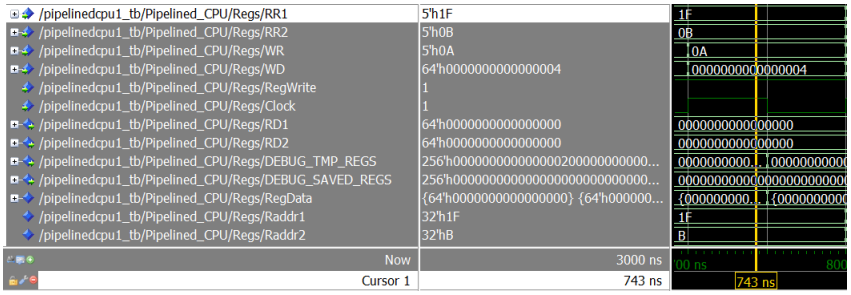

Figure 33: Registers Signals of Cycle eight

In cycle eight, WB is writing value into registers according to instruction three. Control signal RegWrite is '1', write register(WR) is 'h0A=$X10, write data(WD) is 'h04=4. We are writing the result of addition in instruction three into $X10.

f Debug Signals

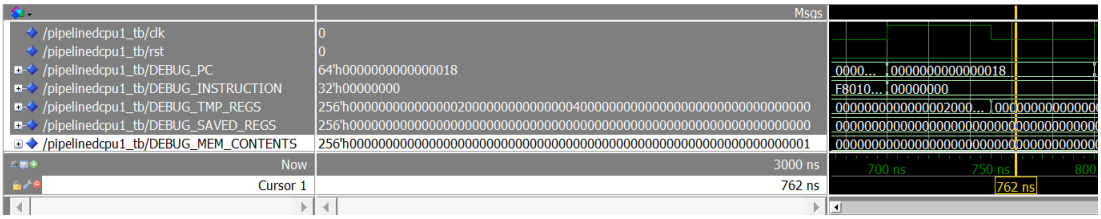

Figure 34: Debug Signals of Cycle eight

During cycle eight, $X10 is changed from 0 to 4 at 750 ns.

10: Cycle Nine

a IF

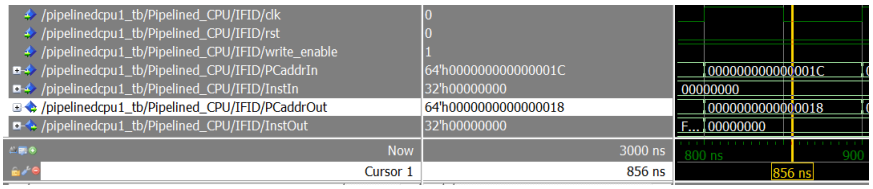

Figure 35: IF/ID Signals of Cycle nine

In cycle nine, IF is fetching instruction seven(NOP). We can see PC address is 0x1C and instruction seven is passed to ID stage.
No work for IF stage.

b ID

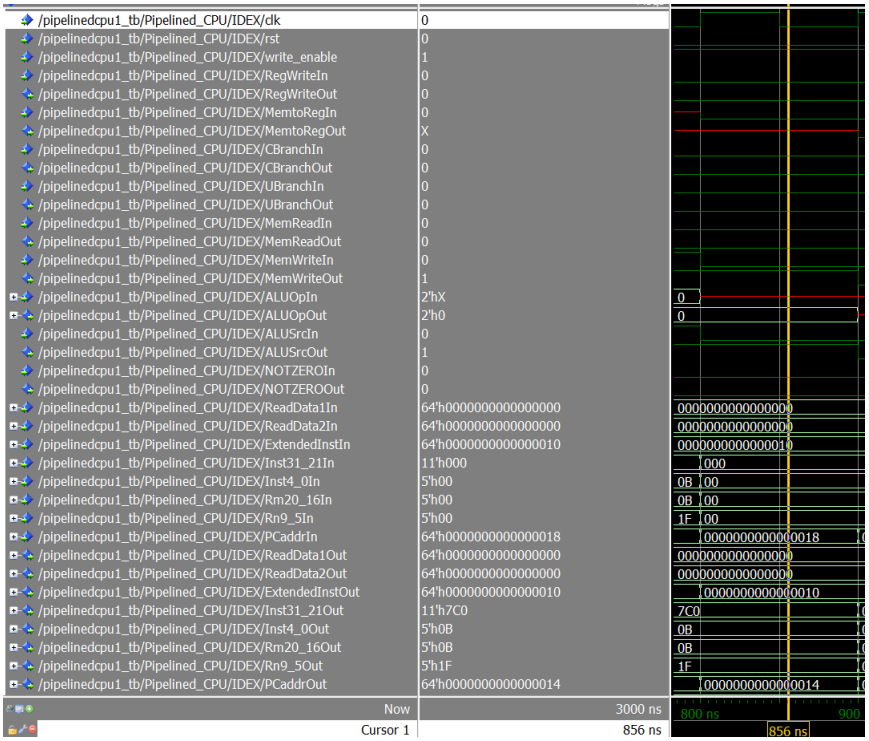

Figure 36: ID/EX Signals of Cycle nine

In cycle nine, ID is decoding instruction seven(NOP). Signals are changed according to instruction seven at 800 ns. Signal values of instruction six are passed to EX stage.

c EX

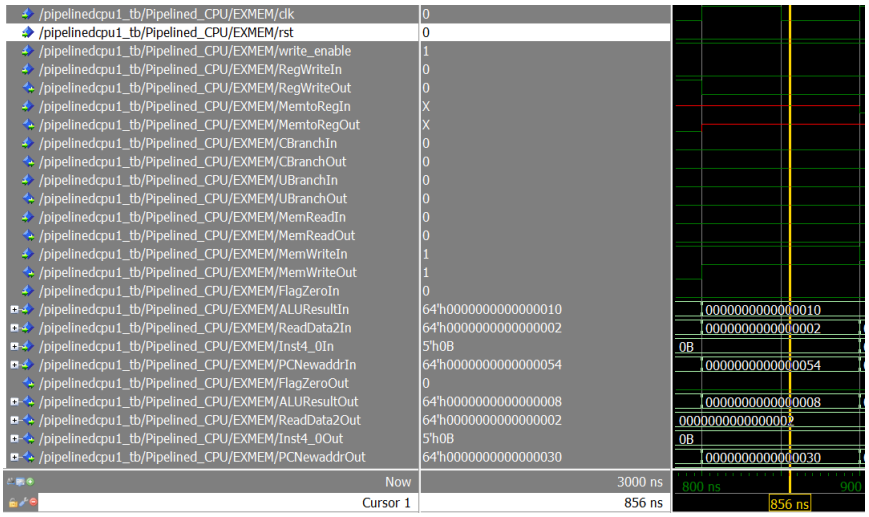

Figure 37: EX/MEM Signals of Cycle nine

In cycle nine, EX is executing operations of instruction six
(STUR X11, [XZR, 16]: 11111000000000100000001111101011). Signals are changed according to instruction six at 800 ns, we can see the ALUResult is 'h10=16, which is $XZR+16 = 0+16 = 16, the address of data memory we would access.
Results of instruction five are passed to MEM stage.

d MEM

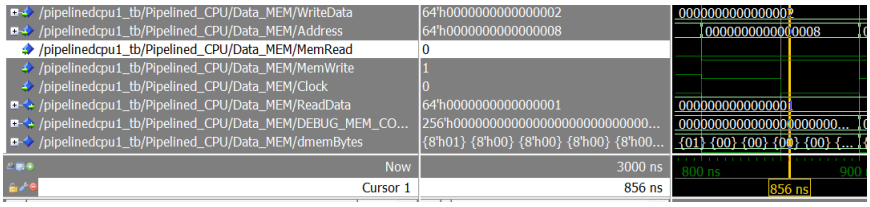

Figure 38: Data Memory Signals of Cycle nine

In cycle nine, MEM is executing operations of instruction five
(STUR X11, [XZR, 8]: 11111000000000001000001111101011). We store $X11 into DMEM[8-15], control signal MemWrite is '1', WriteData is 'h02, Address is 'h08
Results of instruction four are passed to WB stage.

e WB

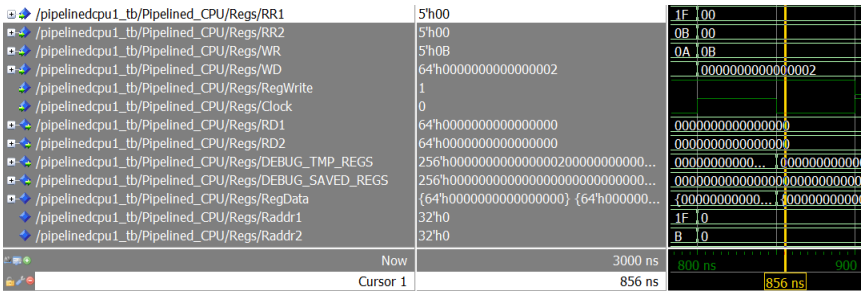

Figure 39: Registers Signals of Cycle nine

In cycle nine, WB is writing value into registers according to instruction four. Control signal RegWrite is '1', write register(WR) is 'h0B=$X11, write data(WD) is 'h02=2. We are writing the result of subtraction in instruction four into $X11.

f Debug Signals

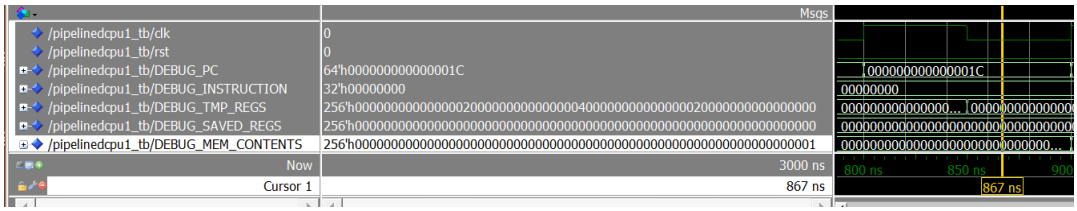

Figure 40: Debug Signals of Cycle nine

During cycle nine, $X11 is changed from 0 to 2 at 850 ns.

11: Cycle Ten

No work for IF and ID stages.

a EX

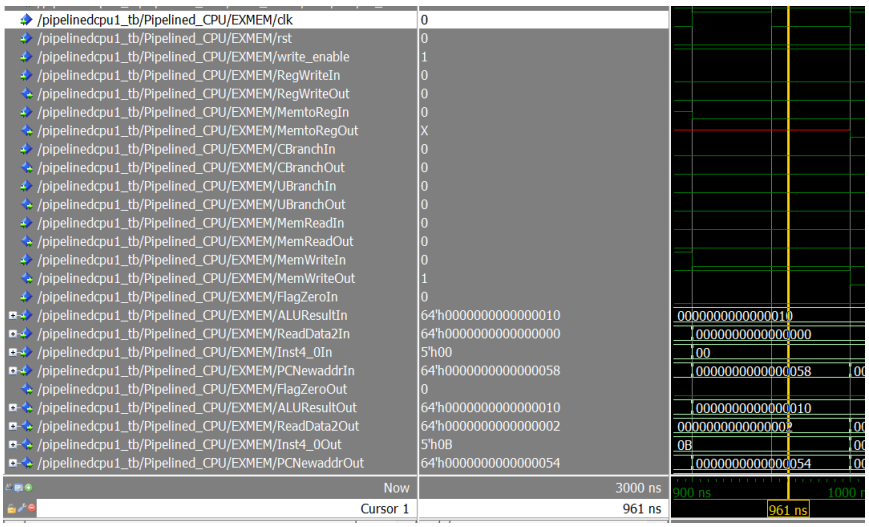

Figure 41: EX/MEM Signals of Cycle ten

In cycle ten, EX is executing operations of instruction seven(NOP). Signals are changed according to instruction seven at 900 ns.
Results of instruction six are passed to MEM stage.

b MEM

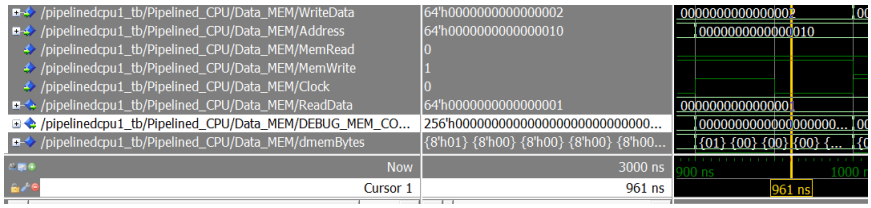

Figure 42: Data Memory Signals of Cycle ten

In cycle ten, MEM is executing operations of instruction six
(STUR X11, [XZR, 16]: 11111000000000100000001111101011). We store $X11 into DMEM[16-23], control signal MemWrite is '1', WriteData is 'h02, Address is 'h10=16
Results of instruction five are passed to WB stage.

c WB

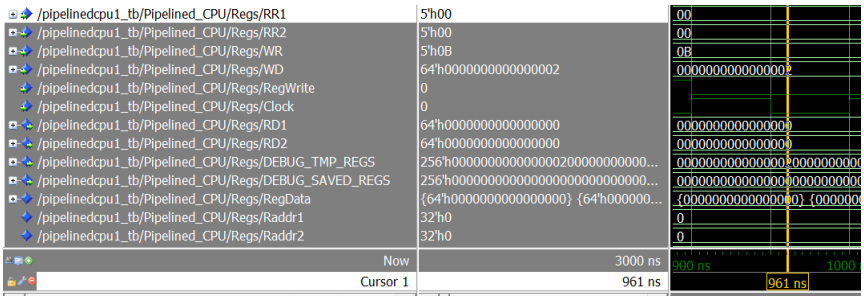

Figure 43: Registers Signals of Cycle ten

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

d Debug Signals

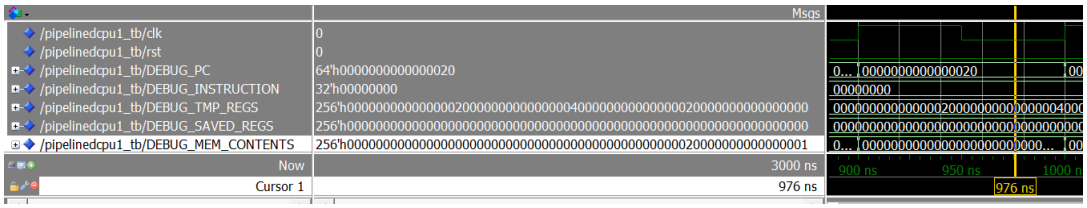

Figure 44: Debug Signals of Cycle ten

During cycle ten, DMEM(0x08) is changed from 0x00 to 0x02 at 900 ns.

12: Cycle Eleven

No work for IF, ID and EX stages.

a MEM

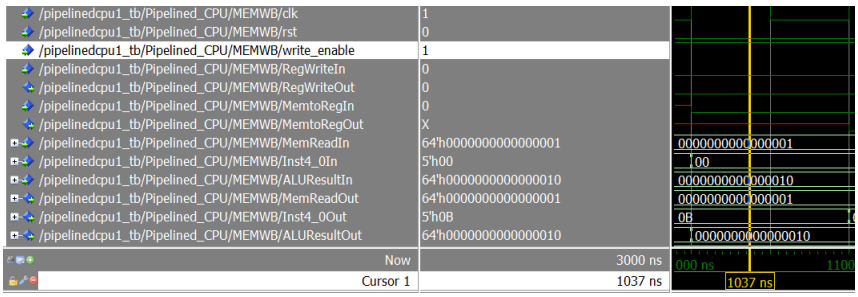

Figure 45: Data Memory Signals of Cycle eleven

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

b WB

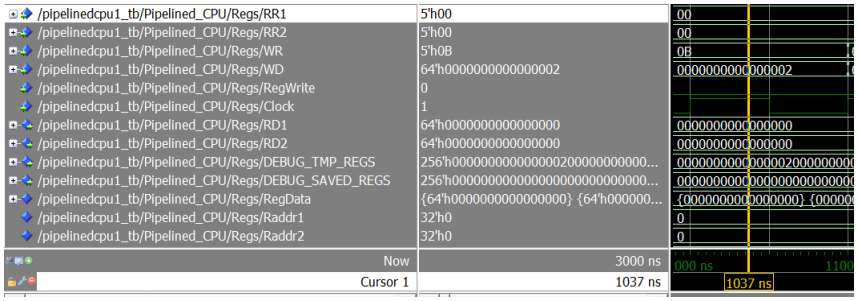

Figure 46: Registers Signals of Cycle ten

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

c Debug Signals

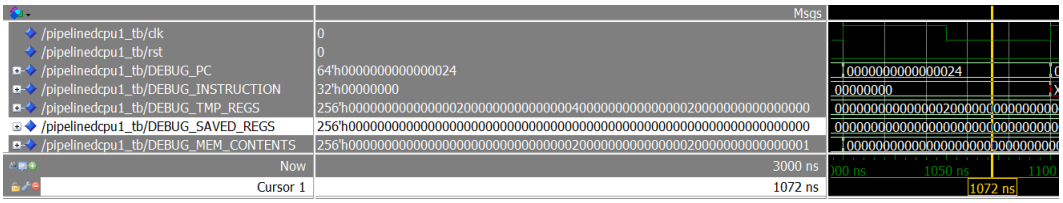

Figure 47: Debug Signals of Cycle eleven

During cycle eleven, DMEM(0x10) is changed from 0x00 to 0x02 at 1000 ns.

13: Sum

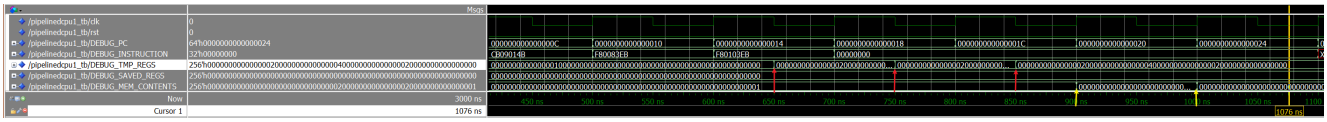

Figure 48: Debug Signals

TMP_REGS changes three times, at 650ns, 750ns, and 850ns(During cycle 7, cycle 8, and cycle 9).
MEM changes two times, at 900ns and 1000ns(the beginning of cycle 10 and cycle 11).
The final states are, $X9 = 2, $X10 = 4, $X11 = 2, $X12 = 0, $X19 = 0, $X20 = 0, $X21 = 0, $X22 = 0, DMEM(0x0) = 0x01, DMEM(0x8) = 0x02, DMEM(0x10) = 0x02, DMEM(0x18) = 0x00.

14: Extra Credit

```
if(first) then
   --1st cycle LDUR  X9, [XZR, 0]
   imemBytes(3) <= "11111000";
   imemBytes(2) <= "01000000";
   imemBytes(1) <= "00000011";
   imemBytes(0) <= "11101001";

   --nop

   imemBytes(7) <= "00000000";
   imemBytes(6) <= "00000000";
   imemBytes(5) <= "00000000";
   imemBytes(4) <= "00000000";

   -- 2nd cycle ADD X9, X9, X9
   imemBytes(11) <= "10001011";
   imemBytes(10) <= "00001001";
   imemBytes(9) <= "00000001";
   imemBytes(8) <= "00101001";
```

Figure 49: Inserting NOP Implementation

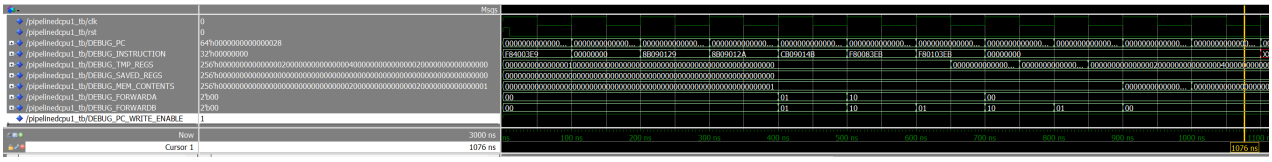

Figure 50: Inserting NOP Signals

Pipeline does not stall after I inserted a nop after the first instruction, control signal 'stallEnable' is '0' during 0 ns to 1100 ns.
We still need 11 cycles to finish the program, inserting a nop is similar to flushing instruction two. The difference is that the '0' control signals of inserting nop comes from nop instruction it self, and the '0' control signals of flushing comes from the hazard controlling.
Forwarding signals are the same as stalling, except for instruction two.
The PC value when the final STUR occurs of inserting nop is 'h18=24, which is instruction seven. And the PC value when the final STUR occurs of stalling is 'h14=20, which is instruction six. Stalling do not create a new instruction, but let a instruction wait until the next

cycle to work. In this way, stalling has one instruction less than inserting nop.

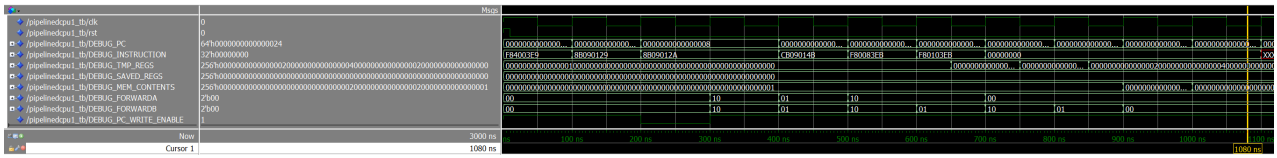

Figure 51: Final States

The final states are the same as stalling.