I would like to use one late token for lab 3.

1: Debug Signals

Debug Signal	Signal content
DEBUG_PC	value of PC
DEBUG_INSTRUCTION	value of 8bytes IMEM data starts at address PC_value
DEBUG_TMP_REGS	X9&X10&X11&X12
DEBUG_SAVED_REGS	X19&X20&X21&X22
DEBUG_MEM_CONTENTS	DMEM[31 DOWNTO 0]
TMP$_OPcode$	4 bits ALU control code
DEBUG_Reg2Loc	Reg2Loc
DEBUG_CBranch	CBranch
DEBUG_MemRead	MemRead
DEBUG_MemtoReg	MemtoReg
DEBUG_MemWrite	MemWrite
DEBUG_ALUSrc	ALUSrc
DEBUG_RegWrite	RegWrite
DEBUG_UBranch	UBranch
DEBUG_ALUOp	ALUOp
DEBUG_ZERO	ZERO flag

Table 1: DEBUG Signals table

These are the debug signals I used in my simulation.

2: Compilation Order and Design Decisions

a Compilation Order

0 dmem.vhd (I'm not using my dmem.vhd for testing in lab three, so you don't need to compile it, or you can let the later dmem_le.vhd to overwrite it.)

1 xorgate.vhd

2 half_adder.vhd

3 full_adder.vhd

4 adder64.vhd

5 add.vhd

6 mux5.vhd

7 mux64.vhd

8 alu.vhd

9 alucontrol.vhd

10 and2.vhd

11 cpucontrol.vhd

12 shiftleft2.vhd

14 signextend.vhd

14 zeromux.vhd

15 pc.vhd

16 registers.vhd

17 dmem_le.vhd

18 imem.vhd (To test LSL, LSR, CBNZ, ORR)

19 singlecyclecpu.vhd

20 singlecyclecpu_tb.vhd

21 imem_comp.vhd (To test computation)

22 imem_ldstr.vhd (To test communication)

23 imem_p1.vhd (To test final)

24 signextend_tb.vhd (Following tbs are for part of my updated components, you don't need to compile them.)

25 alu_tb.vhd

26 alucontrol_tb.vhd

b Design Decisions

CBNZ implementation

I create a new cpu control signal, ZEROorNOT, for CBNZ implementation. When the ZEROorNOT is '0', it will pass the value of signal 'Zero' to the and gate. And when the ZEROorNOT is '1', it will pass the value of '!Zero' to the and gate.

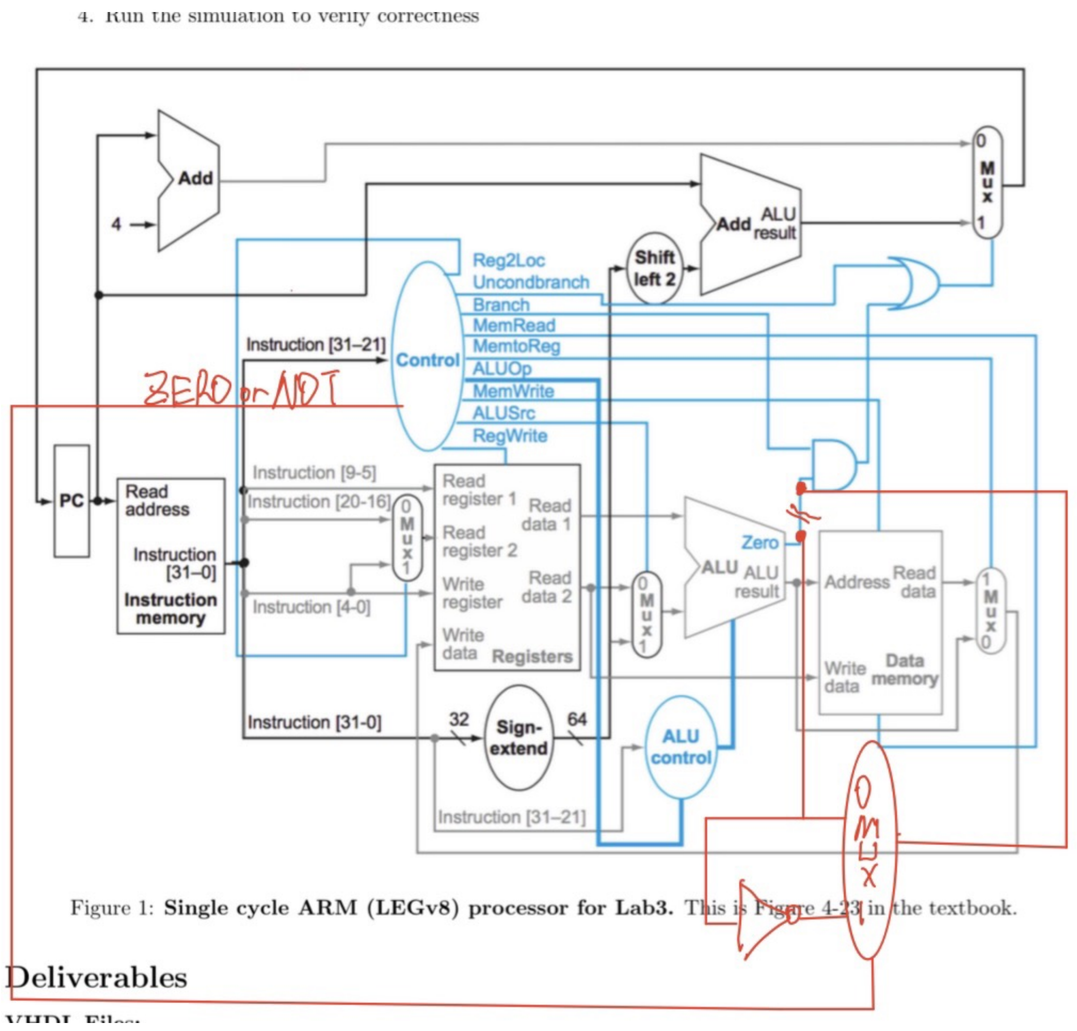

4. Run the simulation to verify correctness

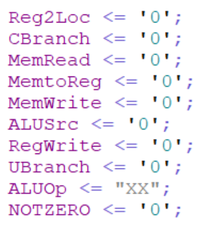
Figure 1: **Single cycle ARM (LEGv8) processor for Lab3.** This is Figure 4-23 in the textbook.

Deliverables

Figure 1: CBNZ implementation

Undecided Instructions

For undecided instructions, I set their cpu control signals like the following.



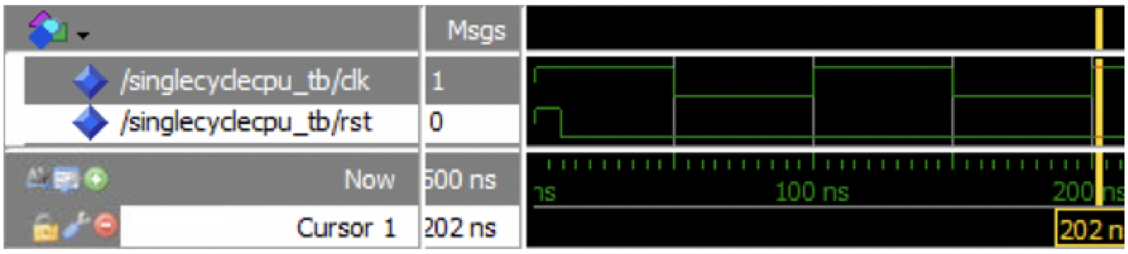
```
Reg2Loc <= '0';
CBranch <= '0';
MemRead <= '0';
MemtoReg <= '0';
MemWrite <= '0';
ALUSrc <= '0';
RegWrite <= '0';
UBranch <= '0';
ALUOp <= "XX";
NOTZERO <= '0';
```

Figure 2: CPU control signals of undecided instructions

3: Test Program with 'X9 = 0'

a Clock and Reset

The clock period is 50 ns, starts at '1'. And the reset signal is '1' during 0 - 10 ns, after 10 ns, it's always '0'.



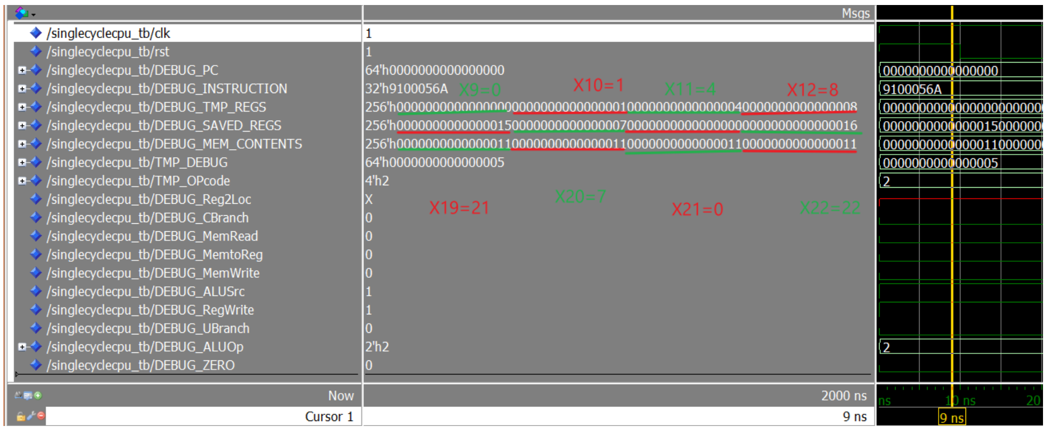
```
                          Msgs
/singlecyclecpu_tb/clk    1
/singlecyclecpu_tb/rst    0
                   Now    500 ns
              Cursor 1    202 ns
```

Figure 3: Simulation Waveform of Clock and Reset

b Registers Initialization

Temporaries: \$X9=0 \$X10=1 \$X11=4 \$X12=8
Saved: \$X19=21 \$X20=7 \$X21=0 \$X22=22
MEM[0-7] = 00000011 Hex, MEM[8-15] = 00000011 Hex
MEM[16-23] = 00000011 Hex, MEM[24-31] = 00000011 Hex

I set 'rst' to '1' to initialize registers and data memories.



```
                                                Msgs
/singlecyclecpu_tb/clk                  1
/singlecyclecpu_tb/rst                  1
/singlecyclecpu_tb/DEBUG_PC             64'h0000000000000000
/singlecyclecpu_tb/DEBUG_INSTRUCTION    32'h9100056A
/singlecyclecpu_tb/DEBUG_TMP_REGS       256'h0000000000000000000000000000000100000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS     256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS   256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/TMP_DEBUG            64'h0000000000000005
/singlecyclecpu_tb/TMP_OPcode           4'h2
/singlecyclecpu_tb/DEBUG_Reg2Loc        X
/singlecyclecpu_tb/DEBUG_CBranch        0
/singlecyclecpu_tb/DEBUG_MemRead        0
/singlecyclecpu_tb/DEBUG_MemtoReg       0
/singlecyclecpu_tb/DEBUG_MemWrite       0
/singlecyclecpu_tb/DEBUG_ALUSrc         1
/singlecyclecpu_tb/DEBUG_RegWrite       1
/singlecyclecpu_tb/DEBUG_UBranch        0
/singlecyclecpu_tb/DEBUG_ALUOp          2'h2
/singlecyclecpu_tb/DEBUG_ZERO           0
                                 Now    2000 ns
                            Cursor 1    9 ns
```

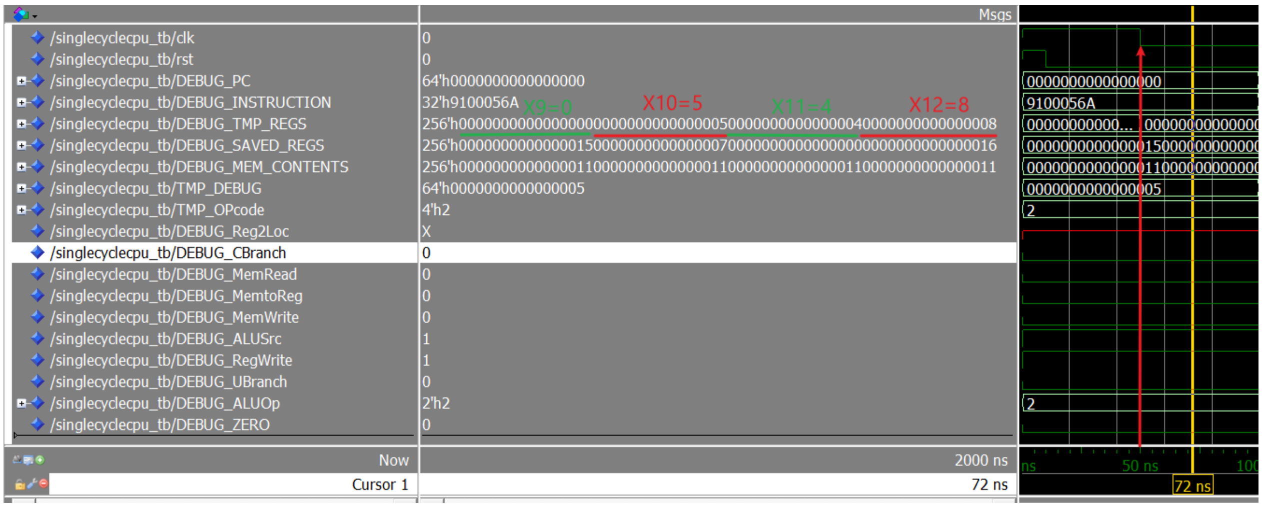
Figure 4: Simulation Waveform of initialization

From the above waveform, we can see the value of 'PC' is set to 0x0 correctly. We can also check part of the initialized registers from "DEBUG_TMP_REGS", they are correctly set.

c Computation test

Instruction one:

32'b10010001 00000000 00000101 01101010 // ADDI X10, X11, 1;
32'h9100056A



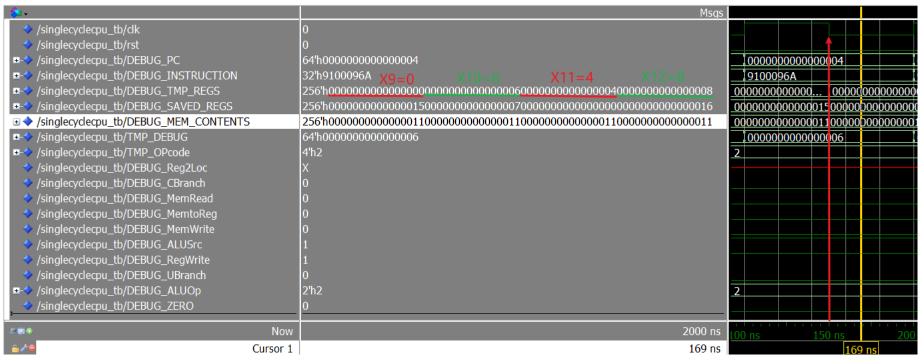
```
                                                                        Msgs
/singlecyclecpu_tb/clk                   0
/singlecyclecpu_tb/rst                   0
/singlecyclecpu_tb/DEBUG_PC              64'h0000000000000000
/singlecyclecpu_tb/DEBUG_INSTRUCTION     32'h9100056A
/singlecyclecpu_tb/DEBUG_TMP_REGS        256'h0000000000000000000000000000000500000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS      256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS    256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/TMP_DEBUG             64'h0000000000000005
/singlecyclecpu_tb/TMP_OPcode            4'h2
/singlecyclecpu_tb/DEBUG_Reg2Loc         X
/singlecyclecpu_tb/DEBUG_CBranch         0
/singlecyclecpu_tb/DEBUG_MemRead         0
/singlecyclecpu_tb/DEBUG_MemtoReg        0
/singlecyclecpu_tb/DEBUG_MemWrite        0
/singlecyclecpu_tb/DEBUG_ALUSrc          1
/singlecyclecpu_tb/DEBUG_RegWrite        1
/singlecyclecpu_tb/DEBUG_UBranch         0
/singlecyclecpu_tb/DEBUG_ALUOp           2'h2
/singlecyclecpu_tb/DEBUG_ZERO            0
Now                                      2000 ns
Cursor 1                                 72 ns
```

Figure 5: Simulation Waveform: Computation test instruction 1

In the initalization, $X11 = 4, $X10 = 1. After running this instruction, $X10 should be $X11+1=5.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h0, the instruction is 32'h9100056A, the same as instruction one.
And the value of $X10 is changed from 1 to 5 at 50 ns, a falling edge of 'clk'. The ALU control signal(TMP_OPcode) is 0010, add action of ADD.
And the last nine signals in the waveform are all the CPU control signals of this ADDI instruction.

Instruction two:

32'b10010001000000000000100101101010 // ADDI X10, X11, 2;
32'h9100096A

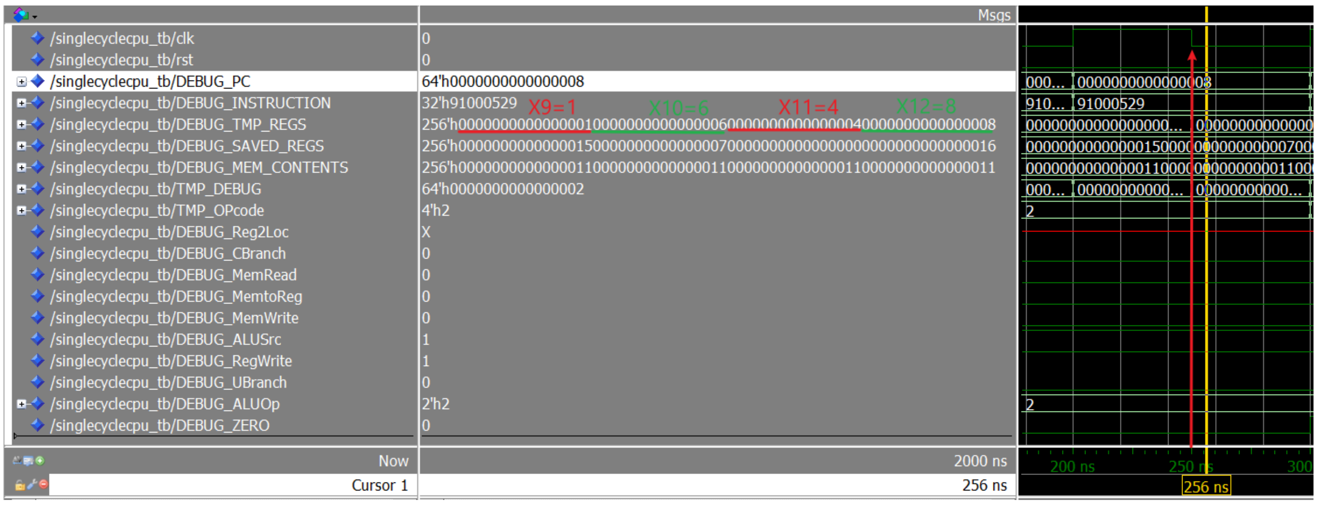


```
                                                                        Msgs
/singlecyclecpu_tb/clk                   0
/singlecyclecpu_tb/rst                   0
/singlecyclecpu_tb/DEBUG_PC              64'h0000000000000004
/singlecyclecpu_tb/DEBUG_INSTRUCTION     32'h9100096A
/singlecyclecpu_tb/DEBUG_TMP_REGS        256'h0000000000000000000000000000000600000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS      256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS    256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/TMP_DEBUG             64'h0000000000000006
/singlecyclecpu_tb/TMP_OPcode            4'h2
/singlecyclecpu_tb/DEBUG_Reg2Loc         X
/singlecyclecpu_tb/DEBUG_CBranch         0
/singlecyclecpu_tb/DEBUG_MemRead         0
/singlecyclecpu_tb/DEBUG_MemtoReg        0
/singlecyclecpu_tb/DEBUG_MemWrite        0
/singlecyclecpu_tb/DEBUG_ALUSrc          1
/singlecyclecpu_tb/DEBUG_RegWrite        1
/singlecyclecpu_tb/DEBUG_UBranch         0
/singlecyclecpu_tb/DEBUG_ALUOp           2'h2
/singlecyclecpu_tb/DEBUG_ZERO            0
Now                                      2000 ns
Cursor 1                                 169 ns
```

Figure 6: Simulation Waveform: Computation test instruction 2

After instruction one, \$X11 = 4, \$X10 = 5. After running this instruction, \$X10 should be \$X11+2=6.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h4, this instruction is 32'h9100096A, the same as instruction two.
And the value of \$X10 is changed from 5 to 6 at 150 ns, a falling edge of 'clk'. The ALU control signal(TMP_OPcode) is 0010, add action of ADDI.
And the last nine signals in the waveform are all the CPU control signals of this ADDI instruction.

Instruction three:
32'b10010001000000000000010100101001; // ADDI X9, X9, 1;
32'h91000529

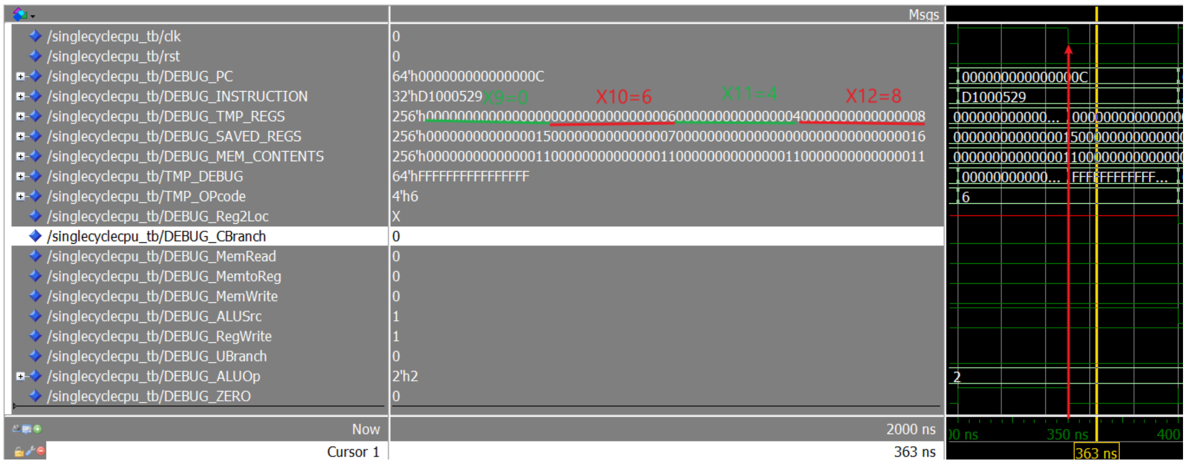


```
                                            Msgs
/singlecyclecpu_tb/clk                 0
/singlecyclecpu_tb/rst                 0
/singlecyclecpu_tb/DEBUG_PC            64'h0000000000000008
/singlecyclecpu_tb/DEBUG_INSTRUCTION   32'h91000529  X9=1   X10=6   X11=4   X12=8
/singlecyclecpu_tb/DEBUG_TMP_REGS      256'h0000000000000001000000000000000600000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS    256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS  256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/TMP_DEBUG           64'h0000000000000002
/singlecyclecpu_tb/TMP_OPcode          4'h2
/singlecyclecpu_tb/DEBUG_Reg2Loc       X
/singlecyclecpu_tb/DEBUG_CBranch       0
/singlecyclecpu_tb/DEBUG_MemRead       0
/singlecyclecpu_tb/DEBUG_MemtoReg      0
/singlecyclecpu_tb/DEBUG_MemWrite      0
/singlecyclecpu_tb/DEBUG_ALUSrc        1
/singlecyclecpu_tb/DEBUG_RegWrite      1
/singlecyclecpu_tb/DEBUG_UBranch       0
/singlecyclecpu_tb/DEBUG_ALUOp         2'h2
/singlecyclecpu_tb/DEBUG_ZERO          0

Now                                    2000 ns
Cursor 1                               256 ns
```

Figure 7: Simulation Waveform: Computation test instruction 3

After instruction two, \$X9 = 0, not changed after initialization. After running this instruction, \$X9 should be \$X9+1=1.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h8, this instruction is 32'h91000529, the same as instruction three.
And the value of \$X9 is changed from 0 to 1 at 250 ns, a falling edge of 'clk'. The ALU control signal(TMP_OPcode) is 0010, add action of ADDI.
And the last nine signals in the waveform are all the CPU control signals of this ADDI instruction.

Instruction four:
32'b11010001000000000000010100101001; // SUBI X9, X9, 1
32'hD1000529

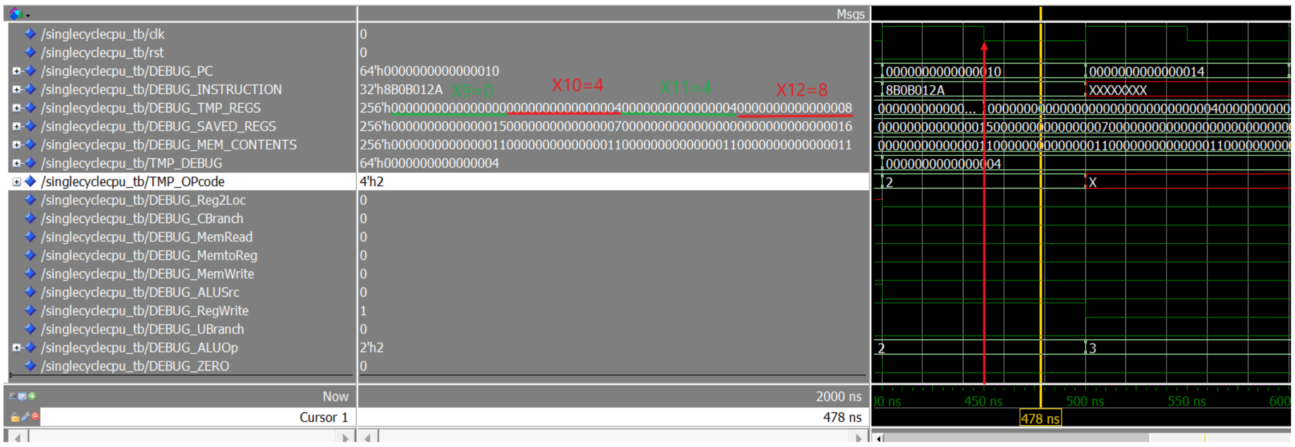


```
                                                                     Msgs
/singlecyclecpu_tb/clk                 0
/singlecyclecpu_tb/rst                 0
/singlecyclecpu_tb/DEBUG_PC            64'h000000000000000C
/singlecyclecpu_tb/DEBUG_INSTRUCTION   32'hD1000529
/singlecyclecpu_tb/DEBUG_TMP_REGS      256'h0000000000000000000000000000000600000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS    256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS  256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/TMP_DEBUG           64'hFFFFFFFFFFFFFFFF
/singlecyclecpu_tb/TMP_OPcode          4'h6
/singlecyclecpu_tb/DEBUG_Reg2Loc       X
/singlecyclecpu_tb/DEBUG_CBranch       0
/singlecyclecpu_tb/DEBUG_MemRead       0
/singlecyclecpu_tb/DEBUG_MemtoReg      0
/singlecyclecpu_tb/DEBUG_MemWrite      0
/singlecyclecpu_tb/DEBUG_ALUSrc        1
/singlecyclecpu_tb/DEBUG_RegWrite      1
/singlecyclecpu_tb/DEBUG_UBranch       0
/singlecyclecpu_tb/DEBUG_ALUOp         2'h2
/singlecyclecpu_tb/DEBUG_ZERO          0
                                  Now  2000 ns
                             Cursor 1  363 ns
```

Figure 8: Simulation Waveform: Computation test instruction 4

After instruction three, \$X9 = 1. After running this instruction, \$X9 should be \$X9-1=0. From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'hC, this instruction is 32'hD1000529, the same as instruction four.
And the value of \$X9 is changed from 1 to 0 at 250 ns, a falling edge of 'clk'. The ALU control signal(TMP_OPcode) is 0110, subtract action of SUBI.
And the last nine signals in the waveform are all the CPU control signals of this SUBI instruction.

Instruction five:
32'b10001011000010110000000100101010; // ADD X10, X9, X11
32'h8B0B012A



```
                                                                     Msgs
/singlecyclecpu_tb/clk                 0
/singlecyclecpu_tb/rst                 0
/singlecyclecpu_tb/DEBUG_PC            64'h0000000000000010
/singlecyclecpu_tb/DEBUG_INSTRUCTION   32'h8B0B012A
/singlecyclecpu_tb/DEBUG_TMP_REGS      256'h0000000000000000000000000000000400000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS    256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS  256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/TMP_DEBUG           64'h0000000000000004
/singlecyclecpu_tb/TMP_OPcode          4'h2
/singlecyclecpu_tb/DEBUG_Reg2Loc       0
/singlecyclecpu_tb/DEBUG_CBranch       0
/singlecyclecpu_tb/DEBUG_MemRead       0
/singlecyclecpu_tb/DEBUG_MemtoReg      0
/singlecyclecpu_tb/DEBUG_MemWrite      0
/singlecyclecpu_tb/DEBUG_ALUSrc        0
/singlecyclecpu_tb/DEBUG_RegWrite      1
/singlecyclecpu_tb/DEBUG_UBranch       0
/singlecyclecpu_tb/DEBUG_ALUOp         2'h2
/singlecyclecpu_tb/DEBUG_ZERO          0
                                  Now  2000 ns
                             Cursor 1  478 ns
```

Figure 9: Simulation Waveform: Computation test instruction 5

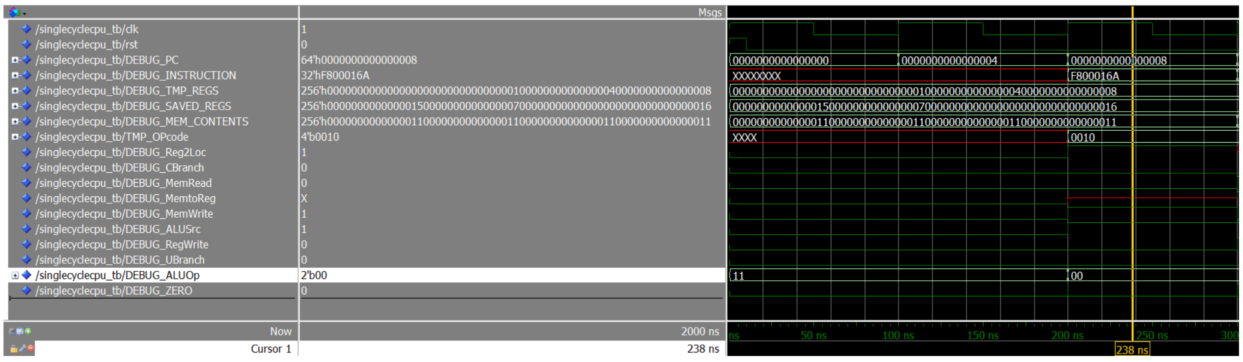
After instruction four, \$X9 = 0, \$X10 = 6 \$X11 = 4. After running this instruction, \$X10 should be \$X9+X11=4.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h10, this instruction is 32'hD1000529, the same as instruction four.
And the value of \$X10 is changed from 6 to 4 at 250 ns, a falling edge of 'clk'. The ALU control signal(TMP_OPcode) is 0010, add action of ADD.
At 500 ns, we can see there is no more instructions.
And the last nine signals in the waveform are all the CPU control signals of this ADD instruction.

d Communication test

Initialization is the same:
Temporaries: $X9=0 $X10=1 $X11=4 $X12=8
Saved: $X19=21 $X20=7 $X21=0 $X22=22
MEM[0-7] = 00000011 Hex, MEM[8-15] = 00000011 Hex
MEM[16-23] = 00000011 Hex, MEM[24-31] = 00000011 Hex

Instruction one

32'b11111000000000000000000101101010; // STUR X10, [X11, 0]
32'hF800016A



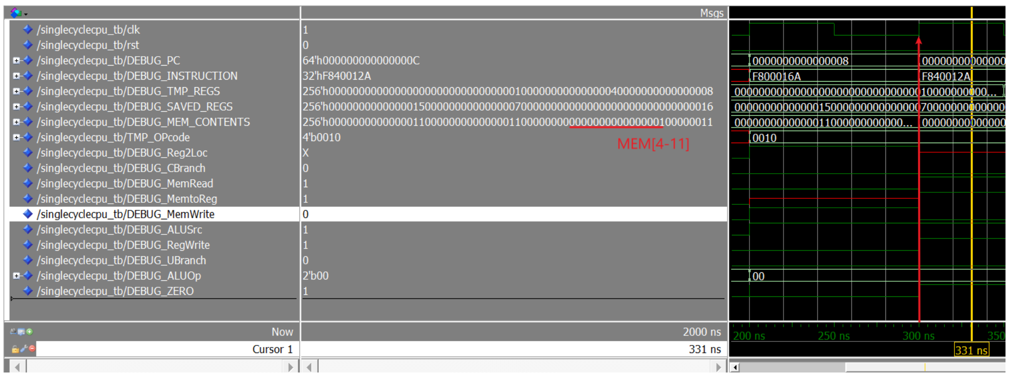
```
                                                      Msgs
/singlecyclecpu_tb/clk                 1
/singlecyclecpu_tb/rst                 0
/singlecyclecpu_tb/DEBUG_PC            64'h0000000000000008
/singlecyclecpu_tb/DEBUG_INSTRUCTION   32'hF800016A
/singlecyclecpu_tb/DEBUG_TMP_REGS      256'h0000000000000000000000000000000100000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS    256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS  256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/TMP_OPcode          4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc       1
/singlecyclecpu_tb/DEBUG_CBranch       0
/singlecyclecpu_tb/DEBUG_MemRead       0
/singlecyclecpu_tb/DEBUG_MemtoReg      X
/singlecyclecpu_tb/DEBUG_MemWrite      1
/singlecyclecpu_tb/DEBUG_ALUSrc        1
/singlecyclecpu_tb/DEBUG_RegWrite      0
/singlecyclecpu_tb/DEBUG_UBranch       0
/singlecyclecpu_tb/DEBUG_ALUOp         2'b00
/singlecyclecpu_tb/DEBUG_ZERO          0
                              Now      2000 ns
                         Cursor 1      238 ns
```

Figure 10: Simulation Waveform: Communication test: Instruction one part one

The first instruction in "imem_ldst.vhd" is start at imemBytes(8), and the PC in the waveform, which we get the first instruction is 0x8. The ALU control code here(TEP_OPcode) is 0010, add action for STUR. And all the CPU control signals for this STUR instruction. Memory write operation happens at the rising edge, so registers will change at the beginning of the next instruction.

Instruction two

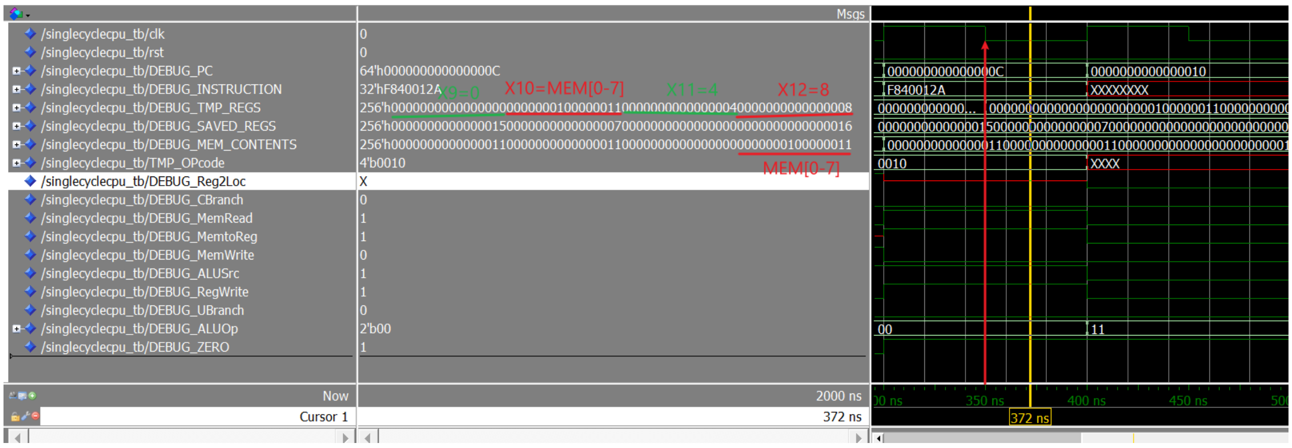
32'b11111000010000000000000100101010; // LDUR X10, [X9, 0]
32'hF840012A



```
                                                      Msgs
/singlecyclecpu_tb/clk                 1
/singlecyclecpu_tb/rst                 0
/singlecyclecpu_tb/DEBUG_PC            64'h000000000000000C
/singlecyclecpu_tb/DEBUG_INSTRUCTION   32'hF840012A
/singlecyclecpu_tb/DEBUG_TMP_REGS      256'h0000000000000000000000000000000100000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS    256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS  256'h0000000000000011000000000000001100000000000000000000000100000011
/singlecyclecpu_tb/TMP_OPcode          4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc       X
/singlecyclecpu_tb/DEBUG_CBranch       0
/singlecyclecpu_tb/DEBUG_MemRead       1
/singlecyclecpu_tb/DEBUG_MemtoReg      1
/singlecyclecpu_tb/DEBUG_MemWrite      0
/singlecyclecpu_tb/DEBUG_ALUSrc        1
/singlecyclecpu_tb/DEBUG_RegWrite      1
/singlecyclecpu_tb/DEBUG_UBranch       0
/singlecyclecpu_tb/DEBUG_ALUOp         2'b00
/singlecyclecpu_tb/DEBUG_ZERO          1
                              Now      2000 ns
                         Cursor 1      331 ns
```

Figure 11: Simulation Waveform: Communication test: Instruction one part two

The second instruction starts at imemBytes(12), and the PC in the waveform, which we get the second instruction is 0xC. The ALU control code here(TEP_OPcode) is 0010, add action for LDUR. And all the CPU control signals for this LDUR instruction. Memory write for the first instruction happens at 300 ns, a rising edge. dmemBytes(4) is changes from 00 to 01, as the instruction one, we want to do operation ”dmemBytes($X11+0)=$X10”. $X11=4, $X10=1, so we get MEM[4-11]=0000000000000001 Hex.



```
                                                    Msgs
/singlecyclecpu_tb/clk                  0
/singlecyclecpu_tb/rst                  0
/singlecyclecpu_tb/DEBUG_PC             64'h000000000000000C
/singlecyclecpu_tb/DEBUG_INSTRUCTION    32'hF840012A
/singlecyclecpu_tb/DEBUG_TMP_REGS       256'h0000000000000000000000010000001100000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS     256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS   256'h0000000000000011000000000000001100000000000000000000000100000011
/singlecyclecpu_tb/TMP_OPcode           4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc        X
/singlecyclecpu_tb/DEBUG_CBranch        0
/singlecyclecpu_tb/DEBUG_MemRead        1
/singlecyclecpu_tb/DEBUG_MemtoReg       1
/singlecyclecpu_tb/DEBUG_MemWrite       0
/singlecyclecpu_tb/DEBUG_ALUSrc         1
/singlecyclecpu_tb/DEBUG_RegWrite       1
/singlecyclecpu_tb/DEBUG_UBranch        0
/singlecyclecpu_tb/DEBUG_ALUOp          2'b00
/singlecyclecpu_tb/DEBUG_ZERO           1

Now        2000 ns
Cursor 1   372 ns
```

Figure 12: Simulation Waveform: Communication test: Instruction one part two

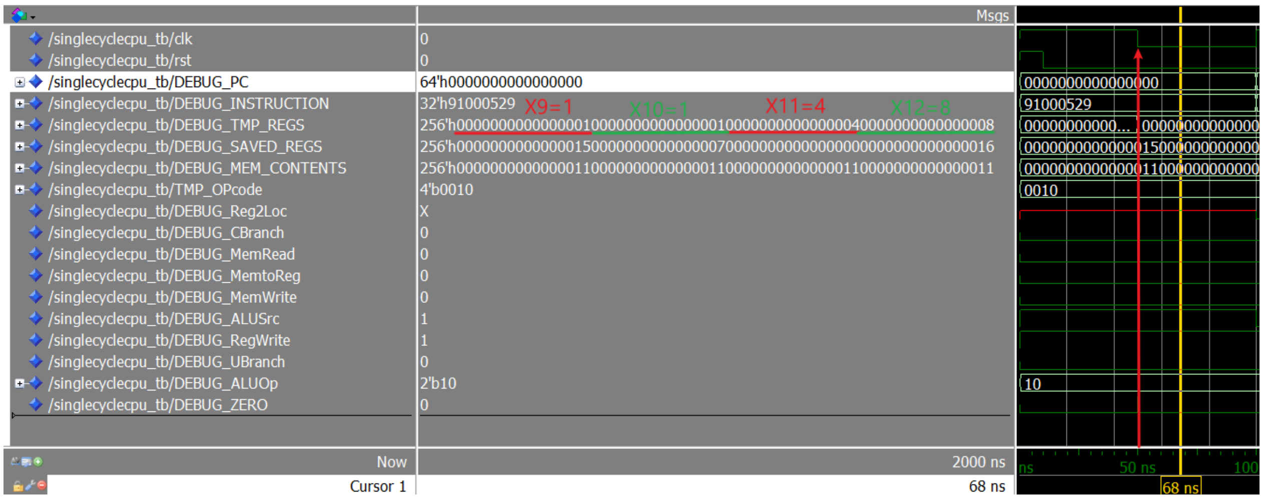
And for instruction two, $X9=0, $X10=1. The LDUR instruction makes X10 have the same value as the 8 bytes data memory starts at address X10. And we got X10 = MEM[0,7], X10 is changed at 350 ns, a falling edge of ’clk’.
There is no instruction after 400 ns, the ALU control singal(TEP_OPcode) is XXXX.

e Final test

Initialization is the same:
Temporaries: \$X9=0 \$X10=1 \$X11=4 \$X12=8
Saved: \$X19=21 \$X20=7 \$X21=0 \$X22=22
MEM[0-7] = 00000011 Hex, MEM[8-15] = 00000011 Hex
MEM[16-23] = 00000011 Hex, MEM[24-31] = 00000011 Hex

Instruction one

16'h0000: out = 32'b10010001000000000000010100101001; // ADDI X9, X9, 1
32'h91000529



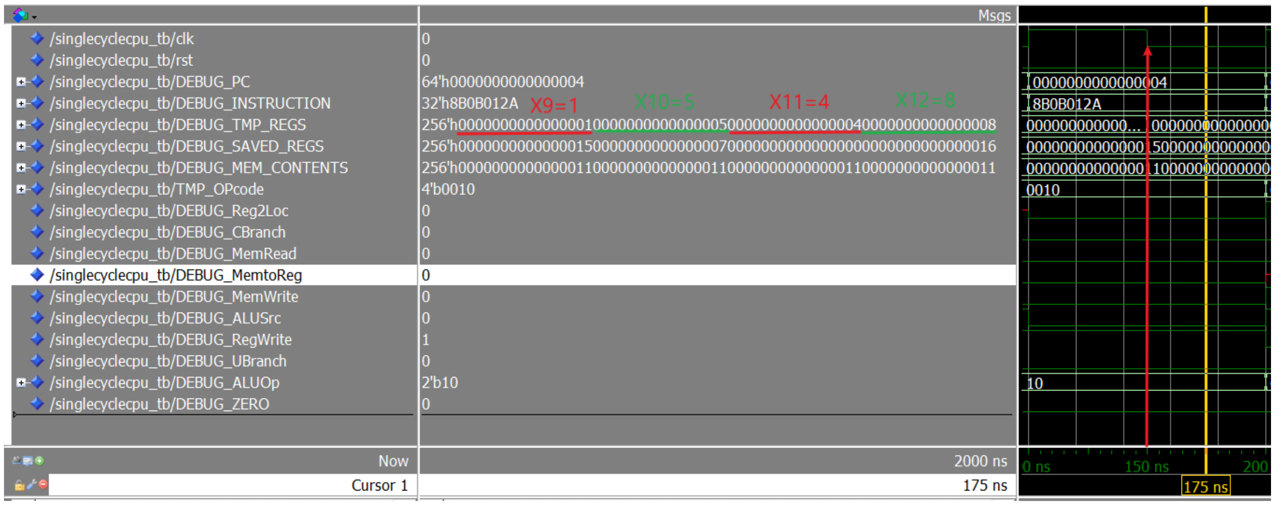
```
                                                  Msgs
/singlecyclecpu_tb/clk               0
/singlecyclecpu_tb/rst               0
/singlecyclecpu_tb/DEBUG_PC          64'h0000000000000000                       000000000000000
/singlecyclecpu_tb/DEBUG_INSTRUCTION 32'h91000529                               91000529
/singlecyclecpu_tb/DEBUG_TMP_REGS    256'h0000000000000001000000000000000100000000000000040000000000000008   00000000000...  00000000000000
/singlecyclecpu_tb/DEBUG_SAVED_REGS  256'h0000000000000015000000000000000700000000000000000000000000000016   000000000000001500000000000000
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS 256'h0000000000000011000000000000001100000000000000110000000000000011  000000000000001100000000000000
/singlecyclecpu_tb/TMP_OPcode        4'b0010                                    0010
/singlecyclecpu_tb/DEBUG_Reg2Loc     X
/singlecyclecpu_tb/DEBUG_CBranch     0
/singlecyclecpu_tb/DEBUG_MemRead     0
/singlecyclecpu_tb/DEBUG_MemtoReg    0
/singlecyclecpu_tb/DEBUG_MemWrite    0
/singlecyclecpu_tb/DEBUG_ALUSrc      1
/singlecyclecpu_tb/DEBUG_RegWrite    1
/singlecyclecpu_tb/DEBUG_UBranch     0
/singlecyclecpu_tb/DEBUG_ALUOp       2'b10                                      10
/singlecyclecpu_tb/DEBUG_ZERO        0

Now                                  2000 ns       ns     50 ns     100
Cursor 1                             68 ns                68 ns
```

Figure 13: Simulation Waveform: Final test: Instruction one

In the initalization, \$X9=0. After running this instruction, \$X9 should be \$X9+1=1.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h0, the instruction is 32'h91000529, the same as instruction one. The ALU control signal(TMP_OPcode) of this instruction is 0010, add action.
And the value of \$X9 is changed from 0 to 1 at 50 ns, a falling edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this ADDI instruction.

Instruction two

16'h0001: out = 32'b10001011000010110000000100101010; // ADD X10,X9,X11
32'h8B0B012A



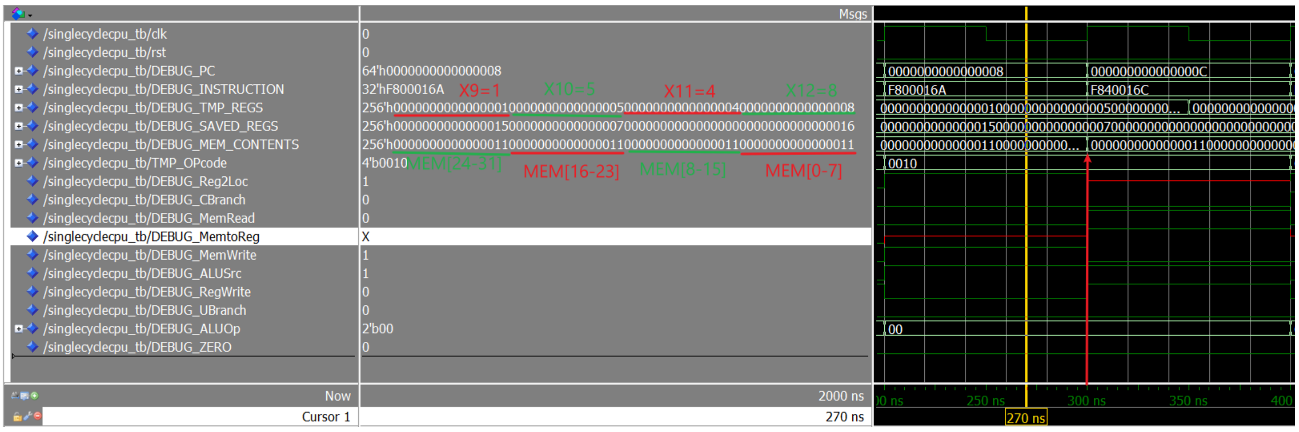
```
                                          Msgs
/singlecyclecpu_tb/clk                    0
/singlecyclecpu_tb/rst                    0
/singlecyclecpu_tb/DEBUG_PC               64'h0000000000000004
/singlecyclecpu_tb/DEBUG_INSTRUCTION      32'h8B0B012A
/singlecyclecpu_tb/DEBUG_TMP_REGS         256'h0000000000000001000000000000000500000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS       256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS     256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/TMP_OPcode             4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc          0
/singlecyclecpu_tb/DEBUG_CBranch          0
/singlecyclecpu_tb/DEBUG_MemRead          0
/singlecyclecpu_tb/DEBUG_MemtoReg         0
/singlecyclecpu_tb/DEBUG_MemWrite         0
/singlecyclecpu_tb/DEBUG_ALUSrc           0
/singlecyclecpu_tb/DEBUG_RegWrite         1
/singlecyclecpu_tb/DEBUG_UBranch          0
/singlecyclecpu_tb/DEBUG_ALUOp            2'b10
/singlecyclecpu_tb/DEBUG_ZERO             0
Now                                       2000 ns
Cursor 1                                  175 ns
```

Figure 14: Simulation Waveform: Final test: Instruction two

After instruction one, \$X9=1, \$X11=4. After running this instruction, \$X10 should be \$X11+\$X9=5.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h4, the instruction is 32'h8B0B012A, the same as instruction two. The ALU control signal(TMP_OPcode) of this instruction is 0010, add action.
And the value of \$X10 is changed from 1 to 5 at 150 ns, a falling edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this ADD instruction.

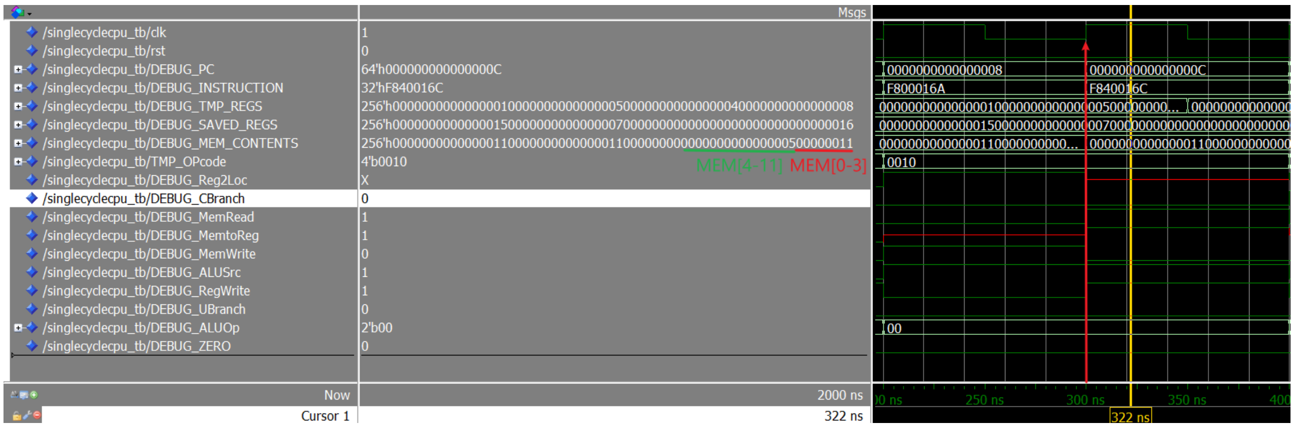
Instruction three

16'h0002: out = 32'b11111000000000000000000101101010; // STUR X10, [X11,0]
32'hF800016A



```
                                          Msgs
/singlecyclecpu_tb/clk                    0
/singlecyclecpu_tb/rst                    0
/singlecyclecpu_tb/DEBUG_PC               64'h0000000000000008
/singlecyclecpu_tb/DEBUG_INSTRUCTION      32'hF800016A
/singlecyclecpu_tb/DEBUG_TMP_REGS         256'h0000000000000001000000000000000500000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS       256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS     256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/TMP_OPcode             4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc          1
/singlecyclecpu_tb/DEBUG_CBranch          0
/singlecyclecpu_tb/DEBUG_MemRead          0
/singlecyclecpu_tb/DEBUG_MemtoReg         X
/singlecyclecpu_tb/DEBUG_MemWrite         1
/singlecyclecpu_tb/DEBUG_ALUSrc           1
/singlecyclecpu_tb/DEBUG_RegWrite         0
/singlecyclecpu_tb/DEBUG_UBranch          0
/singlecyclecpu_tb/DEBUG_ALUOp            2'b00
/singlecyclecpu_tb/DEBUG_ZERO             0
Now                                       2000 ns
Cursor 1                                  270 ns
```

Figure 15: Simulation Waveform: Final test: Instruction three, Memory has not changed



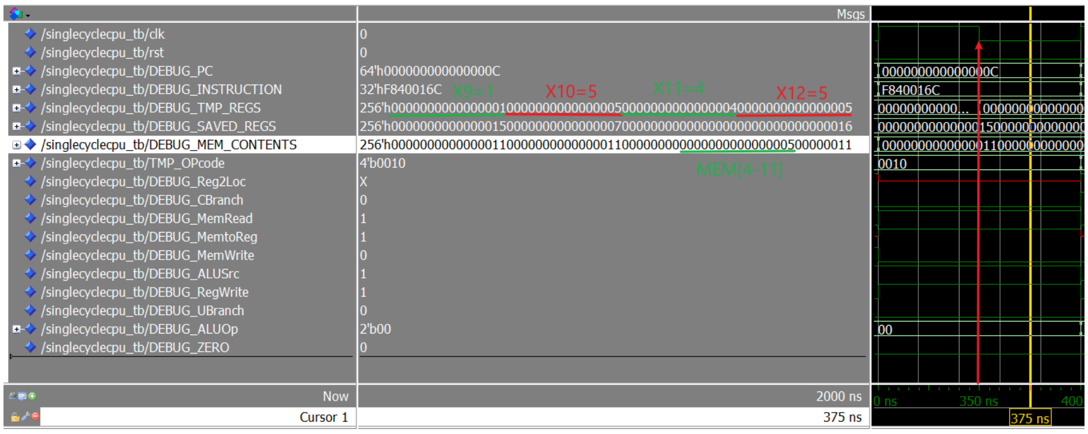
```
/singlecyclecpu_tb/clk                  1
/singlecyclecpu_tb/rst                  0
/singlecyclecpu_tb/DEBUG_PC             64'h000000000000000C
/singlecyclecpu_tb/DEBUG_INSTRUCTION    32'hF840016C
/singlecyclecpu_tb/DEBUG_TMP_REGS       256'h0000000000000001000000000000000500000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS     256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS   256'h0000000000000011000000000000001100000000000000000000000500000011
/singlecyclecpu_tb/TMP_OPcode           4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc        X
/singlecyclecpu_tb/DEBUG_CBranch        0
/singlecyclecpu_tb/DEBUG_MemRead        1
/singlecyclecpu_tb/DEBUG_MemtoReg       1
/singlecyclecpu_tb/DEBUG_MemWrite       0
/singlecyclecpu_tb/DEBUG_ALUSrc         1
/singlecyclecpu_tb/DEBUG_RegWrite       1
/singlecyclecpu_tb/DEBUG_UBranch        0
/singlecyclecpu_tb/DEBUG_ALUOp          2'b00
/singlecyclecpu_tb/DEBUG_ZERO           0
Now        2000 ns
Cursor 1   322 ns
```

Figure 16: Simulation Waveform: Final test: Instruction three, Memory has changed

After instruction two, $X10=5, $X11=4, MEM[0-7] = 00000011 Hex, MEM[8-15] = 00000011 Hex. After running this instruction, MEM[4-11] should be 00000005 Hex.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h8, the instruction is 32'hF800016A, the same as instruction three. The ALU control signal(TMP_OPcode) of this instruction is 0010, add action of STUR.
And the value of MEM[4-11] is changed from 0000001100000000 to 0000000000000005 at 300 ns, a rising edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this STUR instruction.

Instruction four

16'h0003: out = 32'b11111000010000000000000101101100; // LDUR X12, [X11, 0]
32'hF840016C



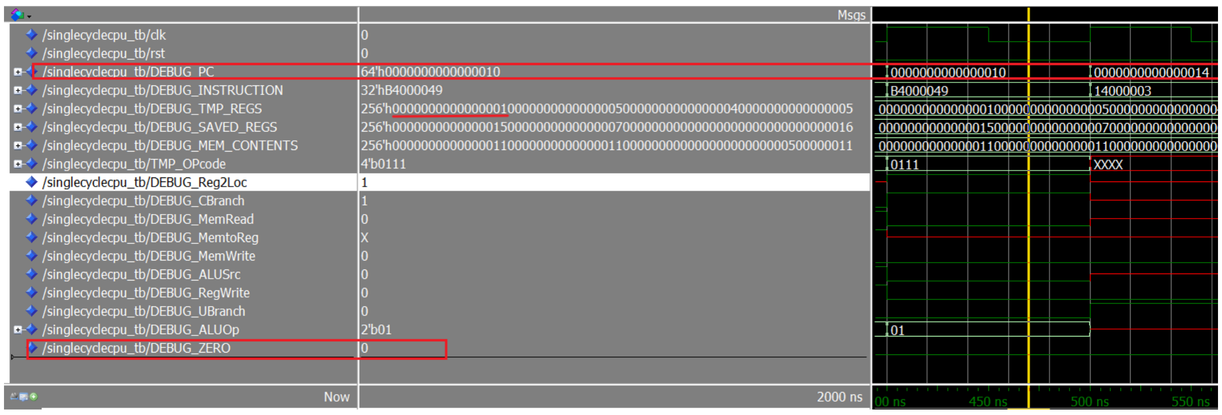
```
/singlecyclecpu_tb/clk                  0
/singlecyclecpu_tb/rst                  0
/singlecyclecpu_tb/DEBUG_PC             64'h000000000000000C
/singlecyclecpu_tb/DEBUG_INSTRUCTION    32'hF840016C
/singlecyclecpu_tb/DEBUG_TMP_REGS       256'h0000000000000001000000000000000500000000000000040000000000000005
/singlecyclecpu_tb/DEBUG_SAVED_REGS     256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS   256'h0000000000000011000000000000001100000000000000000000000500000011
/singlecyclecpu_tb/TMP_OPcode           4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc        X
/singlecyclecpu_tb/DEBUG_CBranch        0
/singlecyclecpu_tb/DEBUG_MemRead        1
/singlecyclecpu_tb/DEBUG_MemtoReg       1
/singlecyclecpu_tb/DEBUG_MemWrite       0
/singlecyclecpu_tb/DEBUG_ALUSrc         1
/singlecyclecpu_tb/DEBUG_RegWrite       1
/singlecyclecpu_tb/DEBUG_UBranch        0
/singlecyclecpu_tb/DEBUG_ALUOp          2'b00
/singlecyclecpu_tb/DEBUG_ZERO           0
Now        2000 ns
Cursor 1   375 ns
```

Figure 17: Simulation Waveform: Final test: Instruction four

After instruction three, $X11=4, $X12=8, MEM[4-11] = 00000005 Hex. After running this instruction, X12 should be the same as MEM[4-11]=5.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'hC, the instruc-

tion is 32'hF840016C, the same as instruction four. The ALU control signal(TMP_OPcode) of this instruction is 0010, add action of LDUR.
And the value of X12 is changed from 8 to 5 at 350 ns, a falling edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this LDUR instruction.

Instruction five
16'h0004: out = 32'b10110100000000000000000001001001; // CBZ X9, 2
32'hB400049

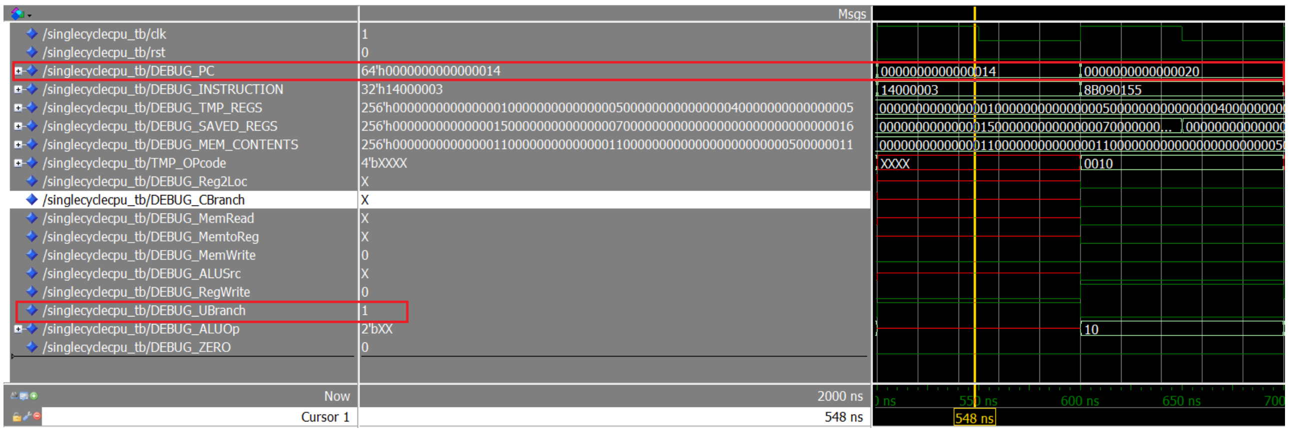


```
                                              Msgs
/singlecyclecpu_tb/clk                        0
/singlecyclecpu_tb/rst                        0
/singlecyclecpu_tb/DEBUG_PC                   64'h0000000000000010
/singlecyclecpu_tb/DEBUG_INSTRUCTION          32'hB4000049
/singlecyclecpu_tb/DEBUG_TMP_REGS             256'h0000000000000001000000000000000500000000000000040000000000000005
/singlecyclecpu_tb/DEBUG_SAVED_REGS           256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS         256'h0000000000000011000000000000001100000000000000000000000500000011
/singlecyclecpu_tb/TMP_OPcode                 4'b0111
/singlecyclecpu_tb/DEBUG_Reg2Loc              1
/singlecyclecpu_tb/DEBUG_CBranch              1
/singlecyclecpu_tb/DEBUG_MemRead              0
/singlecyclecpu_tb/DEBUG_MemtoReg             X
/singlecyclecpu_tb/DEBUG_MemWrite             0
/singlecyclecpu_tb/DEBUG_ALUSrc               0
/singlecyclecpu_tb/DEBUG_RegWrite             0
/singlecyclecpu_tb/DEBUG_UBranch              0
/singlecyclecpu_tb/DEBUG_ALUOp                2'b01
/singlecyclecpu_tb/DEBUG_ZERO                 0

Now                                           2000 ns
```

Figure 18: Simulation Waveform: Final test: Instruction five

After instruction four, \$X9=0, PC=0x10. After running this instruction, PC should be PC+4=0x14, due to the result of ALU is not ZERO.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h10, the instruction is 32'hB8400049, the same as instruction five. The ALU control signal(TMP_OPcode) of this instruction is 0111, pass input b action of CBZ. And the DEBUG_ZERO signal here is the zero flag, which is 0, do not meet the brunch condition, so we go to PC=0x14 at 500 ns, a rising edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this CBZ instruction.

Insruction six
16'h0005: out = 32'b00010100000000000000000000000011; // B 3
32'14000003

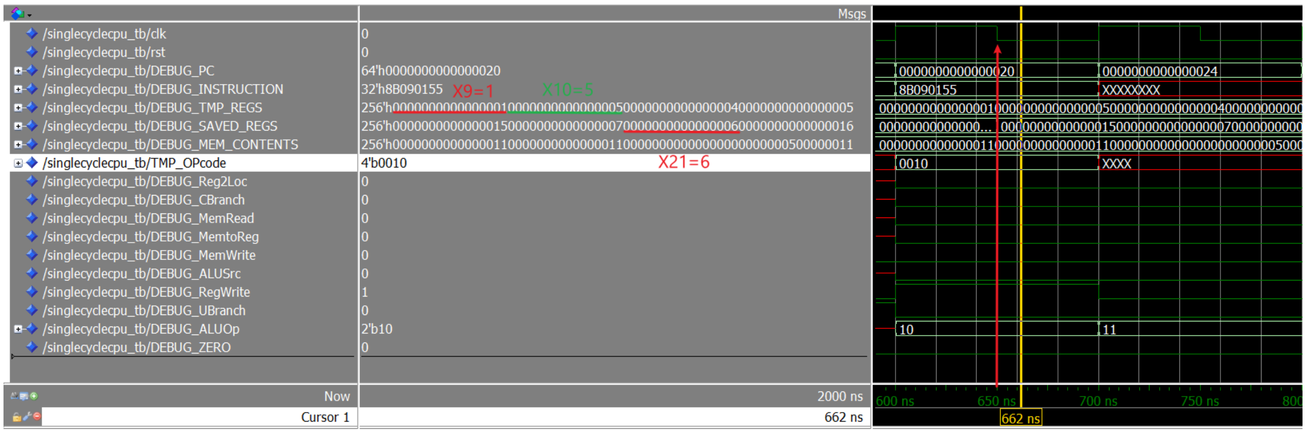


```
/singlecyclecpu_tb/clk              1
/singlecyclecpu_tb/rst              0
/singlecyclecpu_tb/DEBUG_PC         64'h0000000000000014
/singlecyclecpu_tb/DEBUG_INSTRUCTION 32'h14000003
/singlecyclecpu_tb/DEBUG_TMP_REGS   256'h0000000000000001000000000000000500000000000000040000000000000005
/singlecyclecpu_tb/DEBUG_SAVED_REGS 256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS 256'h0000000000000011000000000000001100000000000000000000000500000011
/singlecyclecpu_tb/TMP_OPcode       4'bXXXX
/singlecyclecpu_tb/DEBUG_Reg2Loc    X
/singlecyclecpu_tb/DEBUG_CBranch    X
/singlecyclecpu_tb/DEBUG_MemRead    X
/singlecyclecpu_tb/DEBUG_MemtoReg   X
/singlecyclecpu_tb/DEBUG_MemWrite   0
/singlecyclecpu_tb/DEBUG_ALUSrc     X
/singlecyclecpu_tb/DEBUG_RegWrite   0
/singlecyclecpu_tb/DEBUG_UBranch    1
/singlecyclecpu_tb/DEBUG_ALUOp      2'bXX
/singlecyclecpu_tb/DEBUG_ZERO       0
Now      2000 ns
Cursor 1 548 ns
```

Figure 19: Simulation Waveform: Final test: Instruction six

After running this instruction, PC should be PC+3*4=0x20. Unconditional branch skipped the next two instructions.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h14, the instruction here is 32'h14000003, the same as instruction seven. The ALU control signal(TMP_OPcode) of this instruction is don't care, the same as what we set in ALU control.
We go to PC=0x20 at 600 ns, a rising edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this B instruction.

Insruction nine
16'h0008: out = 32'b10001011000010010000000101010101; // ADD X21, X10, X9
32'h8B090155



```
/singlecyclecpu_tb/clk              0
/singlecyclecpu_tb/rst              0
/singlecyclecpu_tb/DEBUG_PC         64'h0000000000000020
/singlecyclecpu_tb/DEBUG_INSTRUCTION 32'h8B090155  X9=1   X10=5
/singlecyclecpu_tb/DEBUG_TMP_REGS   256'h0000000000000001000000000000000500000000000000040000000000000005
/singlecyclecpu_tb/DEBUG_SAVED_REGS 256'h0000000000000015000000000000000700000000000000060000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS 256'h0000000000000011000000000000001100000000000000000000000500000011
/singlecyclecpu_tb/TMP_OPcode       4'b0010   X21=6
/singlecyclecpu_tb/DEBUG_Reg2Loc    0
/singlecyclecpu_tb/DEBUG_CBranch    0
/singlecyclecpu_tb/DEBUG_MemRead    0
/singlecyclecpu_tb/DEBUG_MemtoReg   0
/singlecyclecpu_tb/DEBUG_MemWrite   0
/singlecyclecpu_tb/DEBUG_ALUSrc     0
/singlecyclecpu_tb/DEBUG_RegWrite   1
/singlecyclecpu_tb/DEBUG_UBranch    0
/singlecyclecpu_tb/DEBUG_ALUOp      2'b10
/singlecyclecpu_tb/DEBUG_ZERO       0
Now      2000 ns
Cursor 1 662 ns
```

Figure 20: Simulation Waveform: Final test: Instruction nine

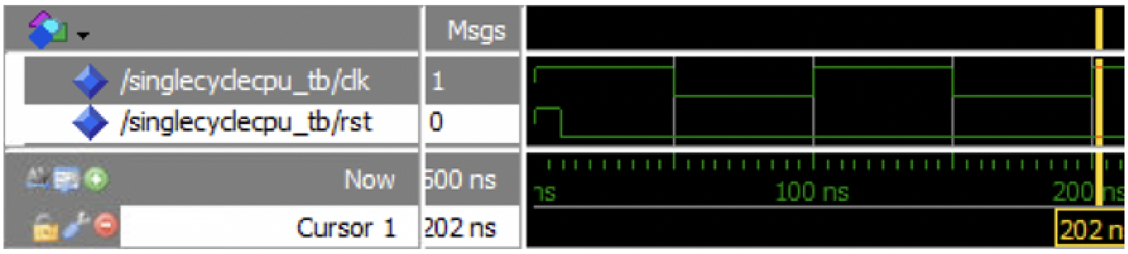
After instruction six, we skipped 2 instructions. And $X9=1, $X10=5, $X21=0. After running this instruction, $X21 should be $X10+$X9=6.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h20 because we have skipped two instructions. The instruction here is 32'h8B090155, the same as instruction nine. The ALU control signal(TMP_OPcode) of this instruction is 0010, add action of ADD.

And the value of X21 is changed from 0 to 6 at 650 ns, a falling edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this ADD instruction.
There is no more instructions after 700 ns.

4: Test Program with 'X9 = -1'

a Clock and Reset

The clock period is 50 ns, starts at '1'. And the reset signal is '1' during 0 - 10 ns, after 10 ns, it's always '0'.



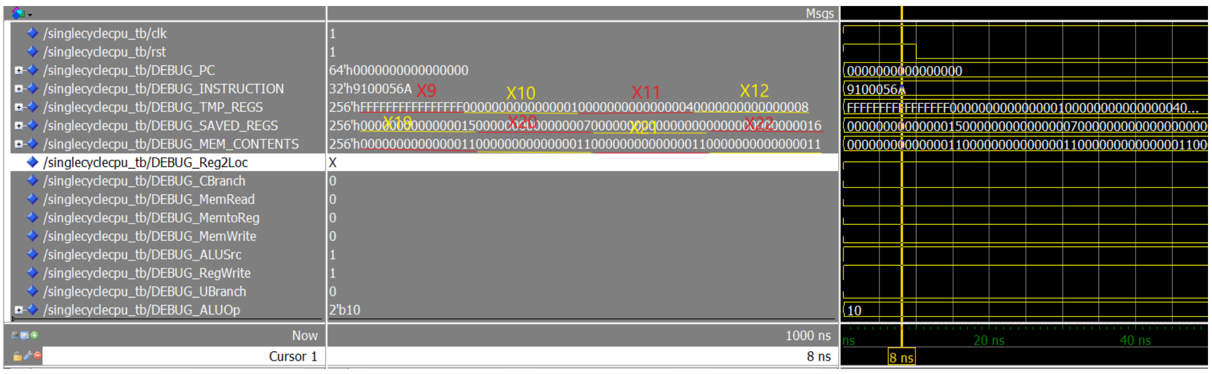
```
                          Msgs
/singlecyclecpu_tb/clk    1
/singlecyclecpu_tb/rst    0
Now                       500 ns
Cursor 1                  202 ns
```

Figure 21: Simulation Waveform of Clock and Reset

b Registers Initialization

Temporaries: \$X9=-1 \$X10=1 \$X11=4 \$X12=8
Saved: \$X19=21 \$X20=7 \$X21=0 \$X22=22
MEM[0-7] = 00000011 Hex, MEM[8-15] = 00000011 Hex
MEM[16-23] = 00000011 Hex, MEM[24-31] = 00000011 Hex

I set 'rst' to '1' to initialize registers and data memories.



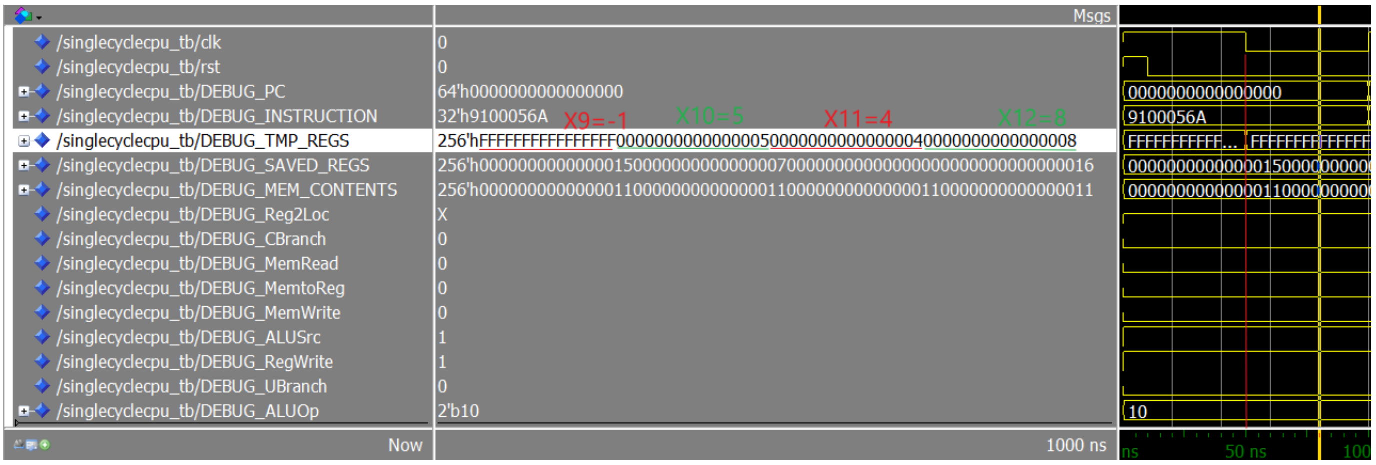
```
                                          Msgs
/singlecyclecpu_tb/clk                    1
/singlecyclecpu_tb/rst                    1
/singlecyclecpu_tb/DEBUG_PC               64'h0000000000000000
/singlecyclecpu_tb/DEBUG_INSTRUCTION      32'h9100056A
/singlecyclecpu_tb/DEBUG_TMP_REGS         256'hFFFFFFFFFFFFFFFF0000000000000001000000000000000400000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS       256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS     256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/DEBUG_Reg2Loc          X
/singlecyclecpu_tb/DEBUG_CBranch          0
/singlecyclecpu_tb/DEBUG_MemRead          0
/singlecyclecpu_tb/DEBUG_MemtoReg         0
/singlecyclecpu_tb/DEBUG_MemWrite         0
/singlecyclecpu_tb/DEBUG_ALUSrc           1
/singlecyclecpu_tb/DEBUG_RegWrite         1
/singlecyclecpu_tb/DEBUG_UBranch          0
/singlecyclecpu_tb/DEBUG_ALUOp            2'b10
Now                                       1000 ns
Cursor 1                                  8 ns
```

Figure 22: Simulation Waveform of initialization

From the above waveform, we can see the value of 'PC' is set to 0x0 correctly. We can also check part of the initialized registers from "DEBUG_TMP_REGS", they are correctly set.

c Computation test

Instruction one:
32'b10010001 00000000 00000101 01101010 // ADDI X10, X11, 1;
32'h9100056A

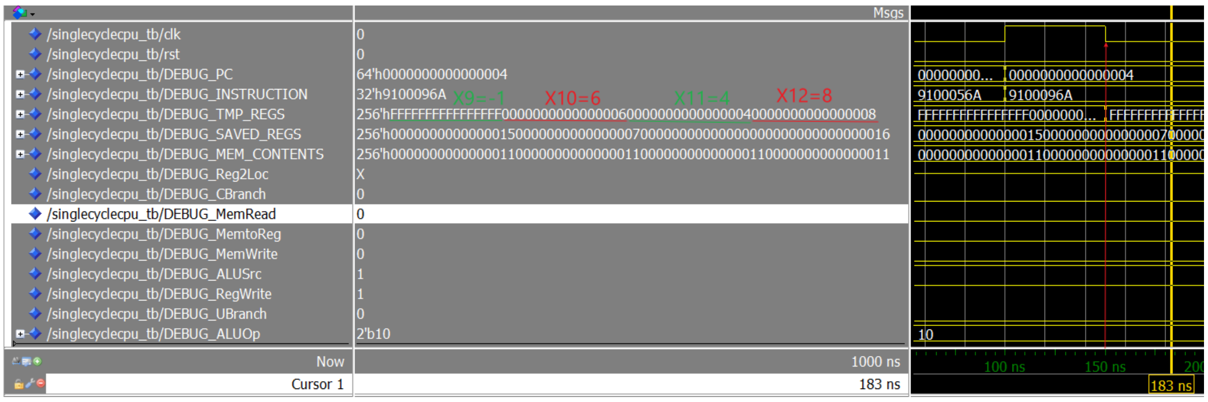


```
/singlecyclecpu_tb/clk                 0
/singlecyclecpu_tb/rst                 0
/singlecyclecpu_tb/DEBUG_PC            64'h0000000000000000
/singlecyclecpu_tb/DEBUG_INSTRUCTION   32'h9100056A  X9=-1  X10=5  X11=4  X12=8
/singlecyclecpu_tb/DEBUG_TMP_REGS      256'hFFFFFFFFFFFFFFFF0000000000000005000000000000000400000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS    256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS  256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/DEBUG_Reg2Loc       X
/singlecyclecpu_tb/DEBUG_CBranch       0
/singlecyclecpu_tb/DEBUG_MemRead       0
/singlecyclecpu_tb/DEBUG_MemtoReg      0
/singlecyclecpu_tb/DEBUG_MemWrite      0
/singlecyclecpu_tb/DEBUG_ALUSrc        1
/singlecyclecpu_tb/DEBUG_RegWrite      1
/singlecyclecpu_tb/DEBUG_UBranch       0
/singlecyclecpu_tb/DEBUG_ALUOp         2'b10
Now                                    1000 ns
```

Figure 23: Simulation Waveform: Computation test instruction 1

In the initalization, \$X11 = 4, \$X10 = 1. After running this instruction, \$X10 should be \$X11+1=5.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h0, the instruction is 32'h9100056A, the same as instruction one.
And the value of \$X10 is changed from 1 to 5 at 50 ns, a falling edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this ADDI instruction.

Instruction two:
32'b10010001000000000000100101101010 // ADDI X10, X11, 2;
32'h9100096A

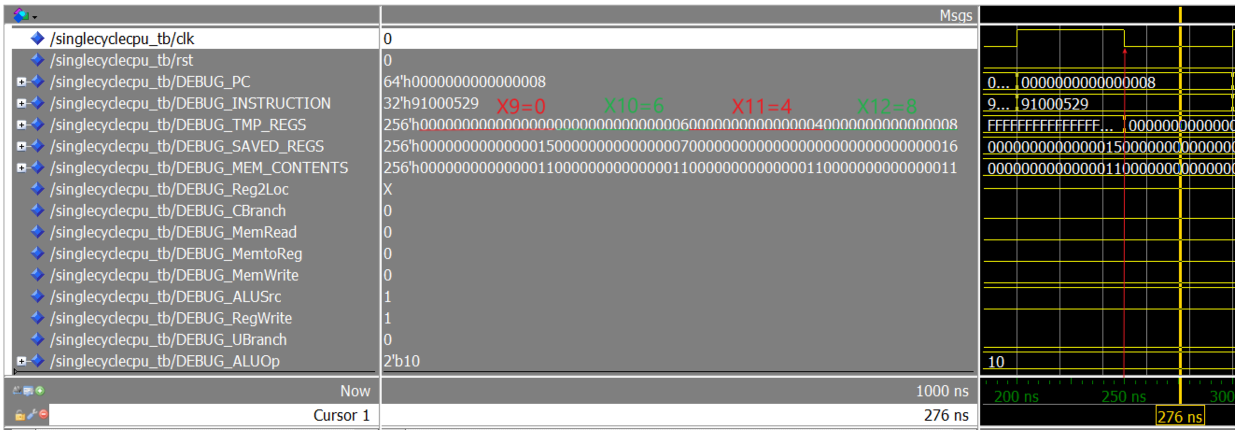


```
/singlecyclecpu_tb/clk                 0
/singlecyclecpu_tb/rst                 0
/singlecyclecpu_tb/DEBUG_PC            64'h0000000000000004
/singlecyclecpu_tb/DEBUG_INSTRUCTION   32'h9100096A  X9=-1  X10=6  X11=4  X12=8
/singlecyclecpu_tb/DEBUG_TMP_REGS      256'hFFFFFFFFFFFFFFFF0000000000000006000000000000000400000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS    256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS  256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/DEBUG_Reg2Loc       X
/singlecyclecpu_tb/DEBUG_CBranch       0
/singlecyclecpu_tb/DEBUG_MemRead       0
/singlecyclecpu_tb/DEBUG_MemtoReg      0
/singlecyclecpu_tb/DEBUG_MemWrite      0
/singlecyclecpu_tb/DEBUG_ALUSrc        1
/singlecyclecpu_tb/DEBUG_RegWrite      1
/singlecyclecpu_tb/DEBUG_UBranch       0
/singlecyclecpu_tb/DEBUG_ALUOp         2'b10
Now                                    1000 ns
Cursor 1                               183 ns
```

Figure 24: Simulation Waveform: Computation test instruction 2

After instruction one, \$X11 = 4, \$X10 = 5. After running this instruction, \$X10 should be \$X11+2=6.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h4, this instruction is 32'h9100096A, the same as instruction two.
And the value of \$X10 is changed from 5 to 6 at 150 ns, a falling edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this ADDI instruction.

Instruction three:
32'b10010001000000000000010100101001; // ADDI X9, X9, 1;
32'h91000529

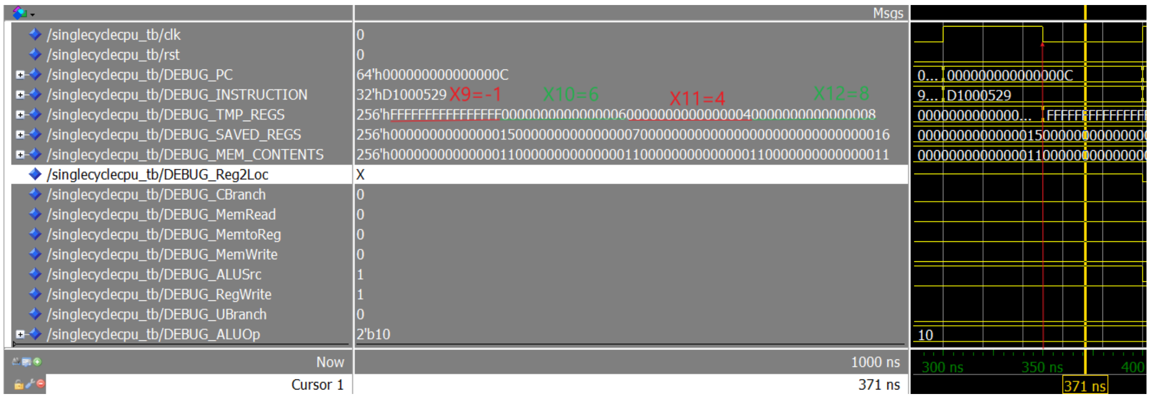


```
                                              Msgs
/singlecyclecpu_tb/clk               0
/singlecyclecpu_tb/rst               0
/singlecyclecpu_tb/DEBUG_PC          64'h0000000000000008
/singlecyclecpu_tb/DEBUG_INSTRUCTION 32'h91000529  X9=0   X10=6   X11=4   X12=8
/singlecyclecpu_tb/DEBUG_TMP_REGS    256'h0000000000000000000000000000000600000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS  256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS 256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/DEBUG_Reg2Loc     X
/singlecyclecpu_tb/DEBUG_CBranch     0
/singlecyclecpu_tb/DEBUG_MemRead     0
/singlecyclecpu_tb/DEBUG_MemtoReg    0
/singlecyclecpu_tb/DEBUG_MemWrite    0
/singlecyclecpu_tb/DEBUG_ALUSrc      1
/singlecyclecpu_tb/DEBUG_RegWrite    1
/singlecyclecpu_tb/DEBUG_UBranch     0
/singlecyclecpu_tb/DEBUG_ALUOp       2'b10
                                Now  1000 ns
                           Cursor 1  276 ns
```

Figure 25: Simulation Waveform: Computation test instruction 3

After instruction two, \$X9 = -1, not changed after initialization. After running this instruction, \$X9 should be \$X9+1=0.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h8, this instruction is 32'h91000529, the same as instruction three.
And the value of \$X9 is changed from -1 to 0 at 250 ns, a falling edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this ADDI instruction.

Instruction four:
32'b11010001000000000000010100101001; // SUBI X9, X9, 1
32'hD1000529

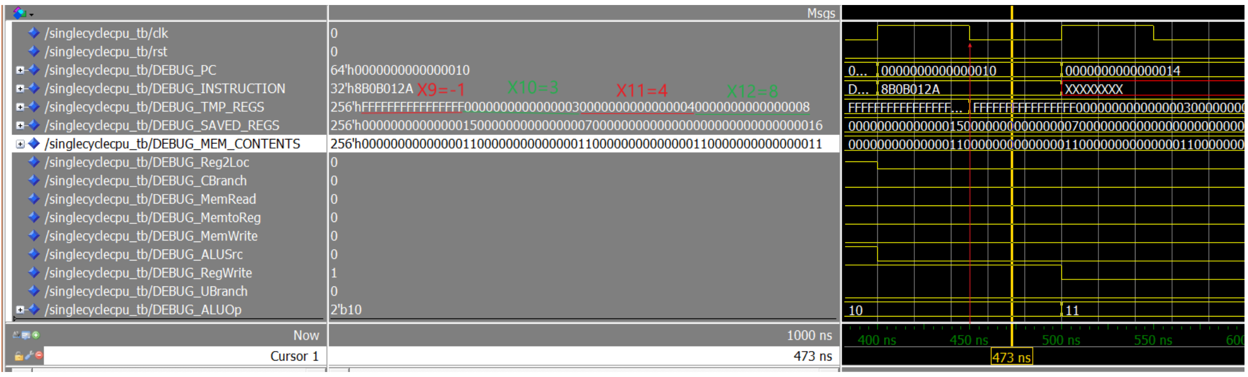


```
/singlecyclecpu_tb/clk                 0
/singlecyclecpu_tb/rst                 0
/singlecyclecpu_tb/DEBUG_PC            64'h000000000000000C
/singlecyclecpu_tb/DEBUG_INSTRUCTION   32'hD1000529
/singlecyclecpu_tb/DEBUG_TMP_REGS      256'hFFFFFFFFFFFFFFFF0000000000000006000000000000000400000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS    256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS  256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/DEBUG_Reg2Loc       X
/singlecyclecpu_tb/DEBUG_CBranch       0
/singlecyclecpu_tb/DEBUG_MemRead       0
/singlecyclecpu_tb/DEBUG_MemtoReg      0
/singlecyclecpu_tb/DEBUG_MemWrite      0
/singlecyclecpu_tb/DEBUG_ALUSrc        1
/singlecyclecpu_tb/DEBUG_RegWrite      1
/singlecyclecpu_tb/DEBUG_UBranch       0
/singlecyclecpu_tb/DEBUG_ALUOp         2'b10
Now                                    1000 ns
Cursor 1                               371 ns
```

Figure 26: Simulation Waveform: Computation test instruction 4

After instruction three, \$X9 = 0. After running this instruction, \$X9 should be \$X9-1=-1. From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'hC, this instruction is 32'hD1000529, the same as instruction four.
And the value of \$X9 is changed from 0 to -1 at 250 ns, a falling edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this SUBI instruction.

Instruction five:
32'b10001011000010110000000100101010; // ADD X10, X9, X11
32'h8B0B012A



```
/singlecyclecpu_tb/clk                 0
/singlecyclecpu_tb/rst                 0
/singlecyclecpu_tb/DEBUG_PC            64'h0000000000000010
/singlecyclecpu_tb/DEBUG_INSTRUCTION   32'h8B0B012A
/singlecyclecpu_tb/DEBUG_TMP_REGS      256'hFFFFFFFFFFFFFFFF0000000000000003000000000000000400000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS    256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS  256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/DEBUG_Reg2Loc       0
/singlecyclecpu_tb/DEBUG_CBranch       0
/singlecyclecpu_tb/DEBUG_MemRead       0
/singlecyclecpu_tb/DEBUG_MemtoReg      0
/singlecyclecpu_tb/DEBUG_MemWrite      0
/singlecyclecpu_tb/DEBUG_ALUSrc        0
/singlecyclecpu_tb/DEBUG_RegWrite      1
/singlecyclecpu_tb/DEBUG_UBranch       0
/singlecyclecpu_tb/DEBUG_ALUOp         2'b10
Now                                    1000 ns
Cursor 1                               473 ns
```

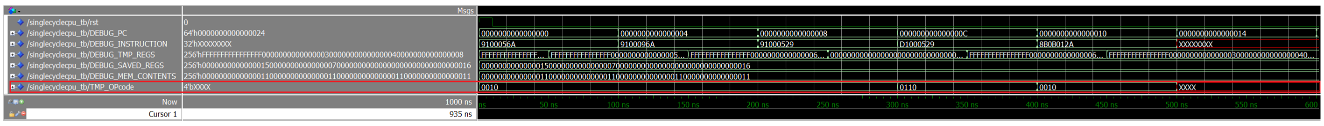
Figure 27: Simulation Waveform: Computation test instruction 5

After instruction four, \$X9 = -1, \$X10 = 6 \$X11 = 4. After running this instruction, \$X10 should be \$X9+X11=3.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h10, this instruction is 32'hD1000529, the same as instruction four.
And the value of \$X10 is changed from 6 to 3 at 250 ns, a falling edge of 'clk'.
At 500 ns, we can see there is no more instructions.

And the last nine signals in the waveform are all the CPU control signals of this ADD instruction.

ALU control Signals

I forgot to show ALU control signals in the previous waveforms. I put them all here.



```
                                                   Msgs
/singlecyclecpu_tb/rst                   0
/singlecyclecpu_tb/DEBUG_PC              64'h0000000000000024
/singlecyclecpu_tb/DEBUG_INSTRUCTION     32'hXXXXXXXX
/singlecyclecpu_tb/DEBUG_TMP_REGS        256'hFFFFFFFFFFFFFFFF0000000000000003000000000000000400000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS      256'h00000000000000150000000000000007000000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS    256'h00000000000000110000000000000011000000000000000110000000000000011
/singlecyclecpu_tb/TMP_OPcode            4'bXXXX
                                  Now    1000 ns
                             Cursor 1    935 ns
```

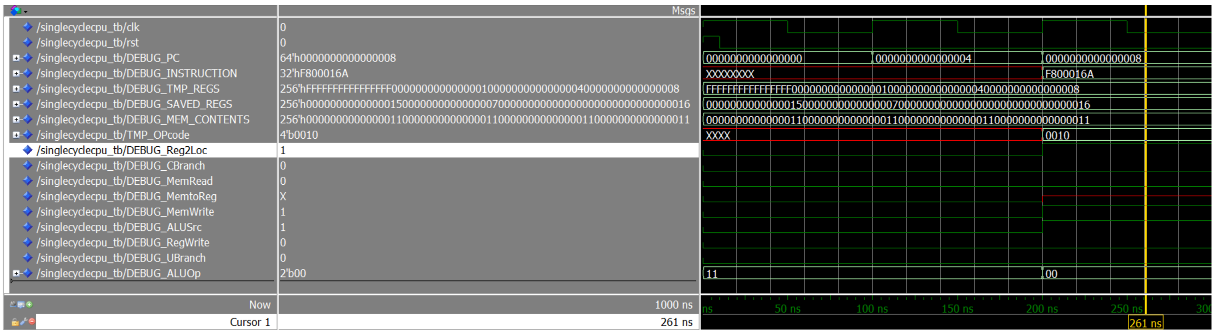
Figure 28: Simulation Waveform: Computation test ALU control signals

The ALU control signal of instruction one is 0010, add action. The ALU control signal of instruction two is 0010, add action. The ALU control signal of instruction three is 0010, add action. The ALU control signal of instruction four is 0110, subtract action. The ALU control signal of instruction five is 0010, add action. When we do not have instrctions, the ALU control signal is XXXX.

d Communication test

Initialization is the same:
Temporaries: $X9=-1 $X10=1 $X11=4 $X12=8
Saved: $X19=21 $X20=7 $X21=0 $X22=22
MEM[0-7] = 00000011 Hex, MEM[8-15] = 00000011 Hex
MEM[16-23] = 00000011 Hex, MEM[24-31] = 00000011 Hex

Instruction one
32'b11111000000000000000000101101010; // STUR X10, [X11, 0]
32'hF800016A

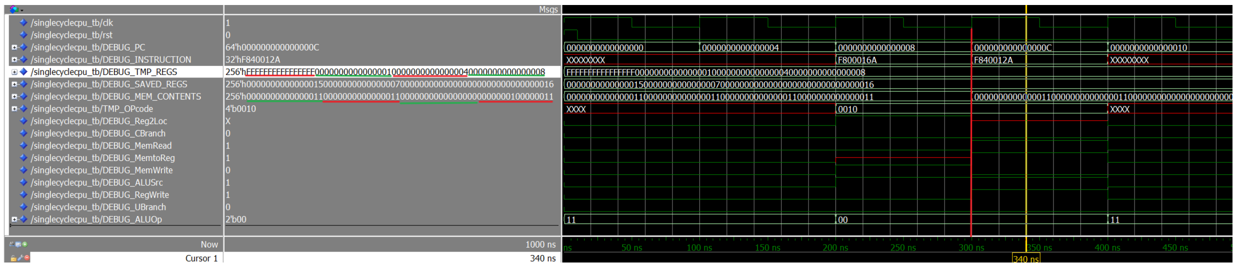


```
/singlecyclecpu_tb/clk                 0
/singlecyclecpu_tb/rst                 0
/singlecyclecpu_tb/DEBUG_PC            64'h0000000000000008
/singlecyclecpu_tb/DEBUG_INSTRUCTION   32'hF800016A
/singlecyclecpu_tb/DEBUG_TMP_REGS      256'hFFFFFFFFFFFFFFFF0000000000000001000000000000000400000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS    256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS  256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/TMP_OPcode          4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc       1
/singlecyclecpu_tb/DEBUG_CBranch       0
/singlecyclecpu_tb/DEBUG_MemRead       0
/singlecyclecpu_tb/DEBUG_MemtoReg      X
/singlecyclecpu_tb/DEBUG_MemWrite      1
/singlecyclecpu_tb/DEBUG_ALUSrc        1
/singlecyclecpu_tb/DEBUG_RegWrite      0
/singlecyclecpu_tb/DEBUG_UBranch       0
/singlecyclecpu_tb/DEBUG_ALUOp         2'b00
Now                                    1000 ns
Cursor 1                               261 ns
```

Figure 29: Simulation Waveform: Communication test: Instruction one

The first instruction in "imem_ldst.vhd" is start at imemBytes(8), and the PC in the waveform, which we get the first instruction is 0x8. The ALU control code here(TEP_OPcode) is 0010, add action for STUR. And all the CPU control signals for this STUR instruction. Memory write operation happens at the rising edge, so registers will change at the beginning of the next instruction.

Instruction two
32'b11111000010000000000000100101010; // LDUR X10, [X9, 0]
32'hF840012A

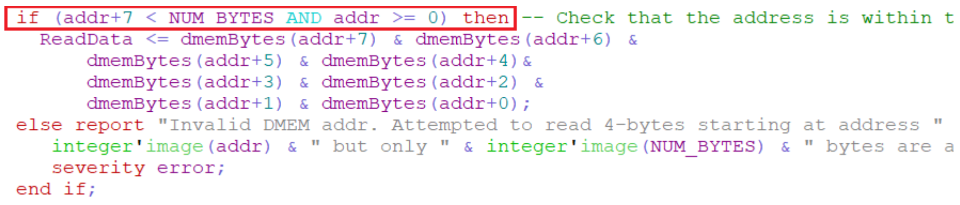


```
/singlecyclecpu_tb/clk                 1
/singlecyclecpu_tb/rst                 0
/singlecyclecpu_tb/DEBUG_PC            64'h000000000000000C
/singlecyclecpu_tb/DEBUG_INSTRUCTION   32'hF840012A
/singlecyclecpu_tb/DEBUG_TMP_REGS      256'hFFFFFFFFFFFFFFFF000000000000000100000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS    256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS  256'h0000000000000011000000000000001100000000000000000000000100000011
/singlecyclecpu_tb/TMP_OPcode          4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc       X
/singlecyclecpu_tb/DEBUG_CBranch       0
/singlecyclecpu_tb/DEBUG_MemRead       1
/singlecyclecpu_tb/DEBUG_MemtoReg      1
/singlecyclecpu_tb/DEBUG_MemWrite      0
/singlecyclecpu_tb/DEBUG_ALUSrc        1
/singlecyclecpu_tb/DEBUG_RegWrite      1
/singlecyclecpu_tb/DEBUG_UBranch       0
/singlecyclecpu_tb/DEBUG_ALUOp         2'b00
Now                                    1000 ns
Cursor 1                               340 ns
```

Figure 30: Simulation Waveform: Communication test: Instruction one

The second instruction starts at imemBytes(12), and the PC in the waveform, which we get the second instruction is 0xC. The ALU control code here(TEP_OPcode) is 0010, add action

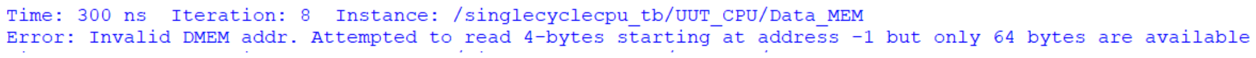
for LDUR. And all the CPU control signals for this LDUR instruction. Memory write for the first instruction happens at 300 ns, a rising edge. dmemBytes(4) is changes from 00 to 01, as the instruction one, we want to do operation ”dmemBytes($X11+0)=$X10”. $X11=4, $X10=1, so we get dmemBytes(4+0)=1.
And for instruction two, I changed DMEM a little, I added a logic to make sure address of memory is not negative, and this can let me run through the rest of the test. The address of data memory is X9, which is -1, an illegal value for address. But the ”to_integer” function is unsigned, and we are supposed to only get positives. And we get negatives here, it is possible that the integer is out of the boundary of integer.



```
 if (addr+7 < NUM_BYTES AND addr >= 0) then -- Check that the address is within t
   ReadData <= dmemBytes(addr+7) & dmemBytes(addr+6) &
       dmemBytes(addr+5) & dmemBytes(addr+4)&
       dmemBytes(addr+3) & dmemBytes(addr+2) &
       dmemBytes(addr+1) & dmemBytes(addr+0);
 else report "Invalid DMEM addr. Attempted to read 4-bytes starting at address "
    integer'image(addr) & " but only " & integer'image(NUM_BYTES) & " bytes are a
    severity error;
 end if;
```

Figure 31: Simulation Waveform: Communication test: Added logic

And we got an error at 300 ns.



```
Time: 300 ns  Iteration: 8  Instance: /singlecyclecpu_tb/UUT_CPU/Data_MEM
Error: Invalid DMEM addr. Attempted to read 4-bytes starting at address -1 but only 64 bytes are available
```

Figure 32: Simulation Waveform: Communication test: -1 address error

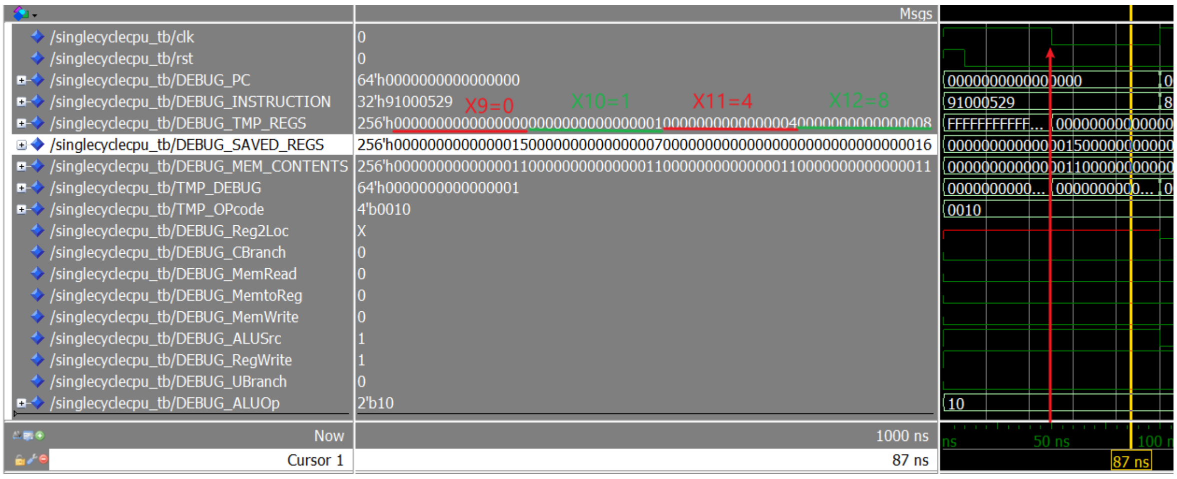
There is no instruction after 400 ns, the ALU control singal(TEP_OPcode) is XXXX.

e Final test

Initialization is the same:
Temporaries: \$X9=-1 \$X10=1 \$X11=4 \$X12=8
Saved: \$X19=21 \$X20=7 \$X21=0 \$X22=22
MEM[0-7] = 00000011 Hex, MEM[8-15] = 00000011 Hex
MEM[16-23] = 00000011 Hex, MEM[24-31] = 00000011 Hex

Instruction one

16'h0000: out = 32'b10010001000000000000010100101001; // ADDI X9, X9, 1
32'h91000529



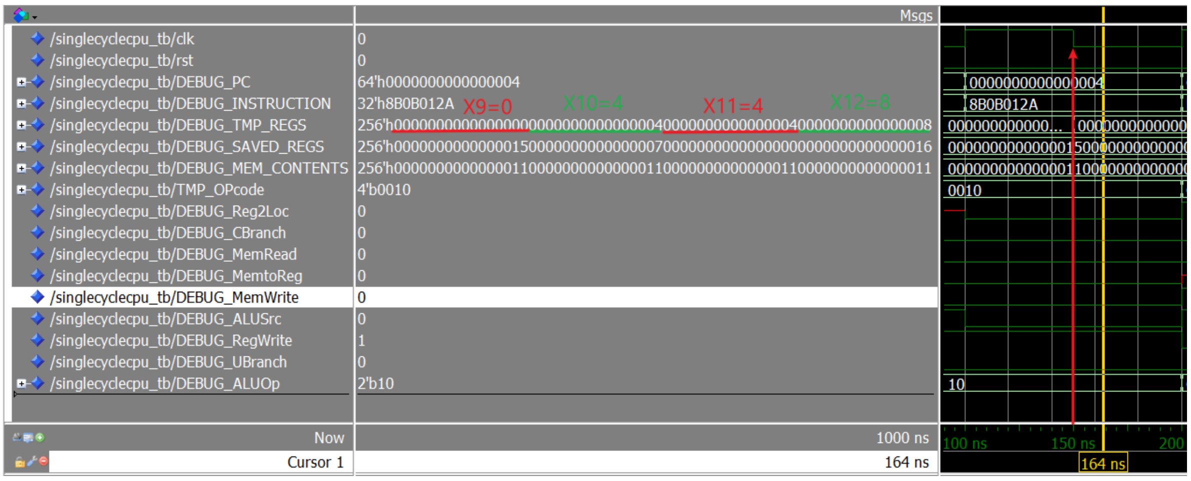
```
                                               Msgs
/singlecyclecpu_tb/clk              0
/singlecyclecpu_tb/rst              0
/singlecyclecpu_tb/DEBUG_PC         64'h0000000000000000
/singlecyclecpu_tb/DEBUG_INSTRUCTION 32'h91000529
/singlecyclecpu_tb/DEBUG_TMP_REGS   256'h0000000000000000000000000000000100000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS 256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS 256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/TMP_DEBUG        64'h0000000000000001
/singlecyclecpu_tb/TMP_OPcode       4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc    X
/singlecyclecpu_tb/DEBUG_CBranch    0
/singlecyclecpu_tb/DEBUG_MemRead    0
/singlecyclecpu_tb/DEBUG_MemtoReg   0
/singlecyclecpu_tb/DEBUG_MemWrite   0
/singlecyclecpu_tb/DEBUG_ALUSrc     1
/singlecyclecpu_tb/DEBUG_RegWrite   1
/singlecyclecpu_tb/DEBUG_UBranch    0
/singlecyclecpu_tb/DEBUG_ALUOp      2'b10
Now                                 1000 ns
Cursor 1                            87 ns
```

Figure 33: Simulation Waveform: Final test: Instruction one

In the initalization, \$X9=-1. After running this instruction, \$X9 should be \$X9+1=0.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h0, the instruction is 32'h91000529, the same as instruction one. The ALU control signal(TMP_OPcode) of this instruction is 0010, add action.
And the value of \$X9 is changed from -1 to 0 at 50 ns, a falling edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this ADDI instruction.

Instruction two

16’h0001: out = 32’b10001011000010110000000100101010; // ADD X10,X9,X11
32’h8B0B012A



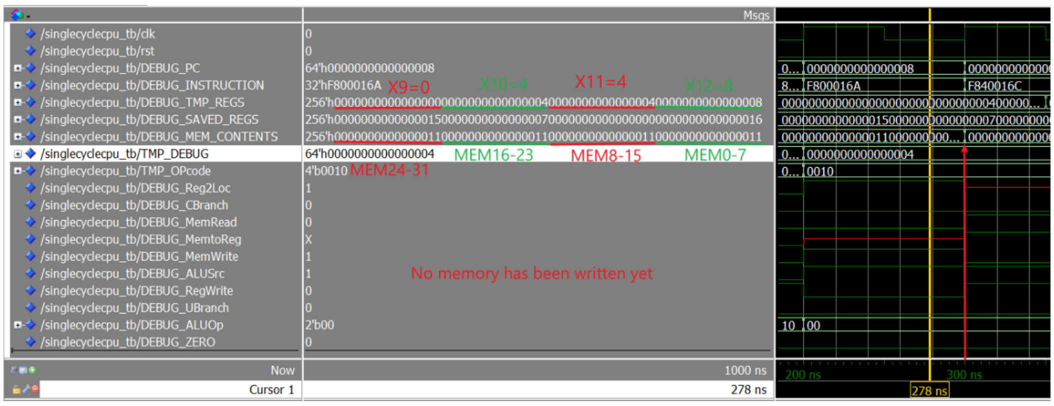
```
/singlecyclecpu_tb/clk                 0
/singlecyclecpu_tb/rst                 0
/singlecyclecpu_tb/DEBUG_PC            64'h0000000000000004
/singlecyclecpu_tb/DEBUG_INSTRUCTION   32'h8B0B012A
/singlecyclecpu_tb/DEBUG_TMP_REGS      256'h0000000000000000000000000000000400000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS    256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS  256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/TMP_OPcode          4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc       0
/singlecyclecpu_tb/DEBUG_CBranch       0
/singlecyclecpu_tb/DEBUG_MemRead       0
/singlecyclecpu_tb/DEBUG_MemtoReg      0
/singlecyclecpu_tb/DEBUG_MemWrite      0
/singlecyclecpu_tb/DEBUG_ALUSrc        0
/singlecyclecpu_tb/DEBUG_RegWrite      1
/singlecyclecpu_tb/DEBUG_UBranch       0
/singlecyclecpu_tb/DEBUG_ALUOp         2'b10
Now                                    1000 ns
Cursor 1                               164 ns
```

Figure 34: Simulation Waveform: Final test: Instruction two

After instruction one, $X9=0, $X11=4. After running this instruction, $X10 should be $X11+$X9=4.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64’h4, the instruction is 32’h8B0B012A, the same as instruction two. The ALU control signal(TMP_OPcode) of this instruction is 0010, add action.
And the value of $X10 is changed from 1 to 4 at 150 ns, a falling edge of ’clk’.
And the last nine signals in the waveform are all the CPU control signals of this ADD instruction.

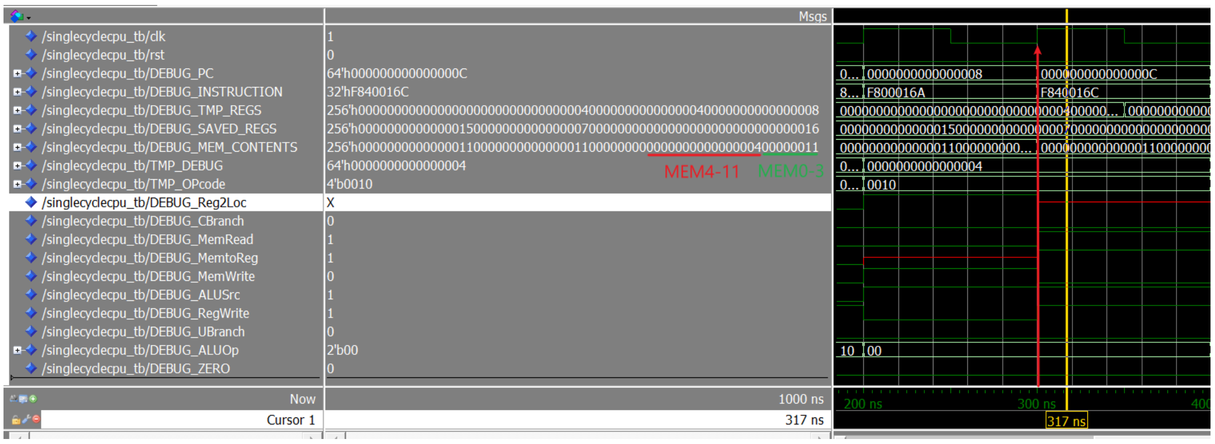
Instruction three

16’h0002: out = 32’b11111000000000000000000101101010; // STUR X10, [X11,0]
32’hF800016A



```
/singlecyclecpu_tb/clk                 0
/singlecyclecpu_tb/rst                 0
/singlecyclecpu_tb/DEBUG_PC            64'h0000000000000008
/singlecyclecpu_tb/DEBUG_INSTRUCTION   32'hF800016A
/singlecyclecpu_tb/DEBUG_TMP_REGS      256'h0000000000000000000000000000000400000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS    256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS  256'h0000000000000011000000000000001100000000000000110000000000000011
/singlecyclecpu_tb/TMP_DEBUG           64'h0000000000000004
/singlecyclecpu_tb/TMP_OPcode          4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc       1
/singlecyclecpu_tb/DEBUG_CBranch       0
/singlecyclecpu_tb/DEBUG_MemRead       0
/singlecyclecpu_tb/DEBUG_MemtoReg      X
/singlecyclecpu_tb/DEBUG_MemWrite      1
/singlecyclecpu_tb/DEBUG_ALUSrc        1
/singlecyclecpu_tb/DEBUG_RegWrite      0
/singlecyclecpu_tb/DEBUG_UBranch       0
/singlecyclecpu_tb/DEBUG_ALUOp         2'b00
/singlecyclecpu_tb/DEBUG_ZERO          0
Now                                    1000 ns
Cursor 1                               278 ns
```

Figure 35: Simulation Waveform: Final test: Instruction three, Memory has not changed

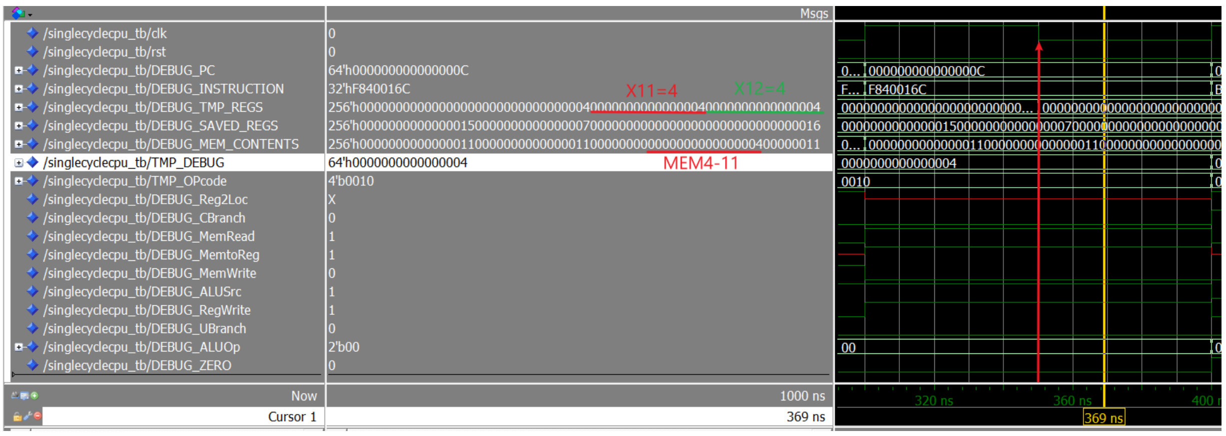


```
/singlecyclecpu_tb/clk                  1
/singlecyclecpu_tb/rst                  0
/singlecyclecpu_tb/DEBUG_PC             64'h000000000000000C
/singlecyclecpu_tb/DEBUG_INSTRUCTION    32'hF840016C
/singlecyclecpu_tb/DEBUG_TMP_REGS       256'h0000000000000000000000000000000400000000000000040000000000000008
/singlecyclecpu_tb/DEBUG_SAVED_REGS     256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS   256'h0000000000000011000000000000001100000000000000000000000400000011
/singlecyclecpu_tb/TMP_DEBUG            64'h0000000000000004
/singlecyclecpu_tb/TMP_OPcode           4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc        X
/singlecyclecpu_tb/DEBUG_CBranch        0
/singlecyclecpu_tb/DEBUG_MemRead        1
/singlecyclecpu_tb/DEBUG_MemtoReg       1
/singlecyclecpu_tb/DEBUG_MemWrite       0
/singlecyclecpu_tb/DEBUG_ALUSrc         1
/singlecyclecpu_tb/DEBUG_RegWrite       1
/singlecyclecpu_tb/DEBUG_UBranch        0
/singlecyclecpu_tb/DEBUG_ALUOp          2'b00
/singlecyclecpu_tb/DEBUG_ZERO           0
Now                                     1000 ns
Cursor 1                                317 ns
```

Figure 36: Simulation Waveform: Final test: Instruction three, Memory has changed

After instruction two, \$X10=4, \$X11=4, MEM[0-7] = 00000011 Hex, MEM[8-15] = 00000011 Hex. After running this instruction, MEM[4-11] should be 00000004 Hex.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h8, the instruction is 32'hF800016A, the same as instruction three. The ALU control signal(TMP_OPcode) of this instruction is 0010, add action of STUR.
And the value of MEM[4-11] is changed from 0000001100000000 to 0000000000000004 at 300 ns, a rising edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this STUR instruction.

Instruction four
16'h0003: out = 32'b11111000010000000000000101101100; // LDUR X12, [X11, 0]
32'hF840016C



```
/singlecyclecpu_tb/clk                  0
/singlecyclecpu_tb/rst                  0
/singlecyclecpu_tb/DEBUG_PC             64'h000000000000000C
/singlecyclecpu_tb/DEBUG_INSTRUCTION    32'hF840016C
/singlecyclecpu_tb/DEBUG_TMP_REGS       256'h0000000000000000000000000000000400000000000000040000000000000004
/singlecyclecpu_tb/DEBUG_SAVED_REGS     256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS   256'h0000000000000011000000000000001100000000000000000000000400000011
/singlecyclecpu_tb/TMP_DEBUG            64'h0000000000000004
/singlecyclecpu_tb/TMP_OPcode           4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc        X
/singlecyclecpu_tb/DEBUG_CBranch        0
/singlecyclecpu_tb/DEBUG_MemRead        1
/singlecyclecpu_tb/DEBUG_MemtoReg       1
/singlecyclecpu_tb/DEBUG_MemWrite       0
/singlecyclecpu_tb/DEBUG_ALUSrc         1
/singlecyclecpu_tb/DEBUG_RegWrite       1
/singlecyclecpu_tb/DEBUG_UBranch        0
/singlecyclecpu_tb/DEBUG_ALUOp          2'b00
/singlecyclecpu_tb/DEBUG_ZERO           0
Now                                     1000 ns
Cursor 1                                369 ns
```

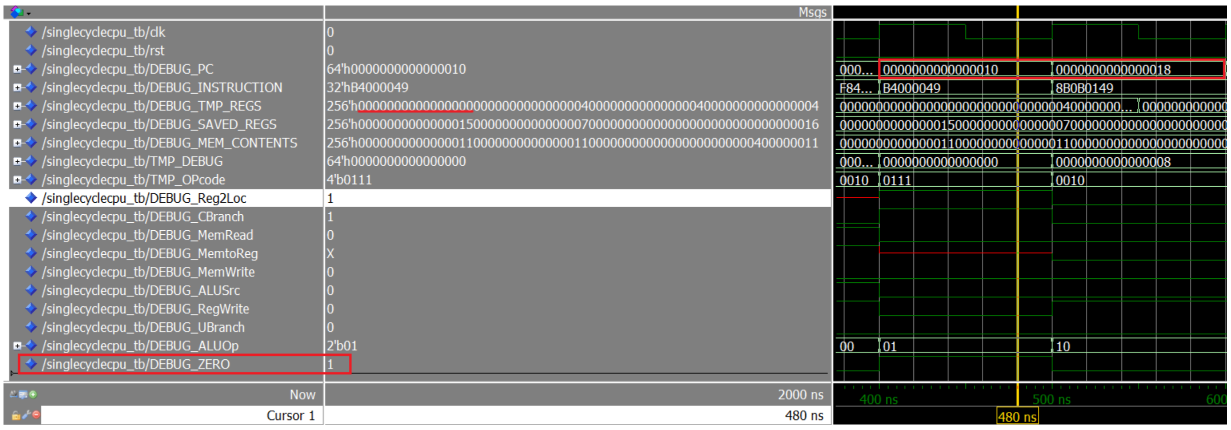
Figure 37: Simulation Waveform: Final test: Instruction four

After instruction three, \$X11=4, \$X12=8, MEM[4-11] = 00000004 Hex. After running this instruction, X12 should be the same as MEM[4-11]=4.

From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'hC, the instruction is 32'hF840016C, the same as instruction four. The ALU control signal(TMP_OPcode) of this instruction is 0010, add action of LDUR.
And the value of X12 is changed from 8 to 4 at 350 ns, a falling edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this LDUR instruction.

Instruction five

16'h0004: out = 32'b10110100000000000000000001001001; // CBZ X9, 2
32'hB8400049



```
                                            Msgs
/singlecyclecpu_tb/clk                      0
/singlecyclecpu_tb/rst                      0
/singlecyclecpu_tb/DEBUG_PC                 64'h0000000000000010
/singlecyclecpu_tb/DEBUG_INSTRUCTION        32'hB4000049
/singlecyclecpu_tb/DEBUG_TMP_REGS           256'h0000000000000000000000000000000400000000000000040000000000000004
/singlecyclecpu_tb/DEBUG_SAVED_REGS         256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS       256'h0000000000000011000000000000001100000000000000000000000400000011
/singlecyclecpu_tb/TMP_DEBUG                64'h0000000000000000
/singlecyclecpu_tb/TMP_OPcode               4'b0111
/singlecyclecpu_tb/DEBUG_Reg2Loc            1
/singlecyclecpu_tb/DEBUG_CBranch            1
/singlecyclecpu_tb/DEBUG_MemRead            0
/singlecyclecpu_tb/DEBUG_MemtoReg           X
/singlecyclecpu_tb/DEBUG_MemWrite           0
/singlecyclecpu_tb/DEBUG_ALUSrc             0
/singlecyclecpu_tb/DEBUG_RegWrite           0
/singlecyclecpu_tb/DEBUG_UBranch            0
/singlecyclecpu_tb/DEBUG_ALUOp              2'b01
/singlecyclecpu_tb/DEBUG_ZERO               1
                                       Now  2000 ns
                                  Cursor 1  480 ns
```

Figure 38: Simulation Waveform: Final test: Instruction five

After instruction four, $X9=0, PC=0x10. After running this instruction, PC should be PC+2*4=0x18.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h10, the instruction is 32'hB8400049, the same as instruction five. The ALU control signal(TMP_OPcode) of this instruction is 0111, pass input b action of CBZ. And the DEBUG_ZERO signal here is the zero flag, which is 1, meet the brunch condition, so we jump to PC=0x18 at 500 ns, a rising edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this CBZ instruction.

Insruction seven

16'h0006: out = 32'b10001011000010110000000101001001; // ADD X9, X10, X11
32'h8B0B0149

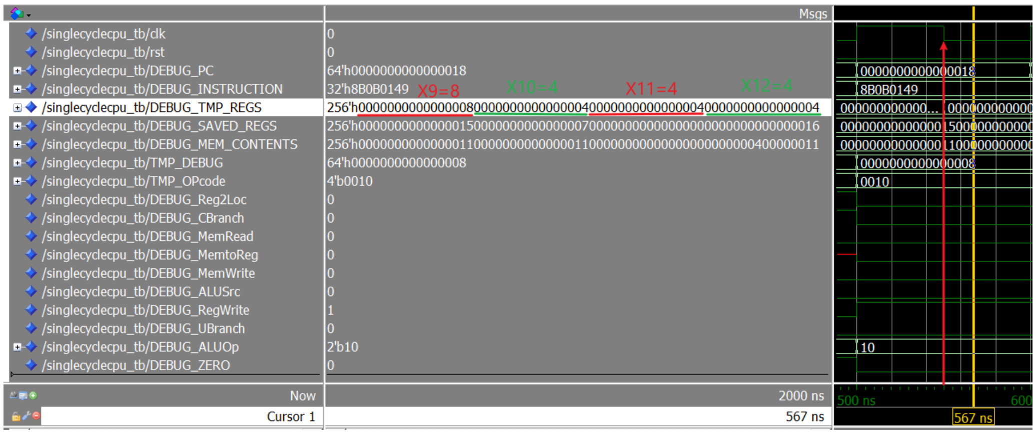

```
                                          Msgs
/singlecyclecpu_tb/clk                    0
/singlecyclecpu_tb/rst                    0
/singlecyclecpu_tb/DEBUG_PC               64'h0000000000000018
/singlecyclecpu_tb/DEBUG_INSTRUCTION      32'h8B0B0149
/singlecyclecpu_tb/DEBUG_TMP_REGS         256'h0000000000000008000000000000000400000000000000040000000000000004
/singlecyclecpu_tb/DEBUG_SAVED_REGS       256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS     256'h0000000000000011000000000000001100000000000000000000000400000011
/singlecyclecpu_tb/TMP_DEBUG              64'h0000000000000008
/singlecyclecpu_tb/TMP_OPcode             4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc          0
/singlecyclecpu_tb/DEBUG_CBranch          0
/singlecyclecpu_tb/DEBUG_MemRead          0
/singlecyclecpu_tb/DEBUG_MemtoReg         0
/singlecyclecpu_tb/DEBUG_MemWrite         0
/singlecyclecpu_tb/DEBUG_ALUSrc           0
/singlecyclecpu_tb/DEBUG_RegWrite         1
/singlecyclecpu_tb/DEBUG_UBranch          0
/singlecyclecpu_tb/DEBUG_ALUOp            2'b10
/singlecyclecpu_tb/DEBUG_ZERO             0
Now                                       2000 ns
Cursor 1                                  567 ns
```

Figure 39: Simulation Waveform: Final test: Instruction seven

After instruction five, $X9=0, $X10=4, $X11=4, $X12=4. After running this instruction, $X9 should be $X10+$X11=8.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h18, since we skipped an instruction due to the CBZ instruction. The instruction here is 32'h8B0B0149, the same as instruction seven. The ALU control signal(TMP_OPcode) of this instruction is 0010, add action of ADD.
And the value of X9 is changed from 0 to 8 at 550 ns, a falling edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this ADD instruction.

Insruction eight

16'h0007: out = 32'b10010001000000000000010100101001; // ADDI X9, X9, 1
32'h91000529

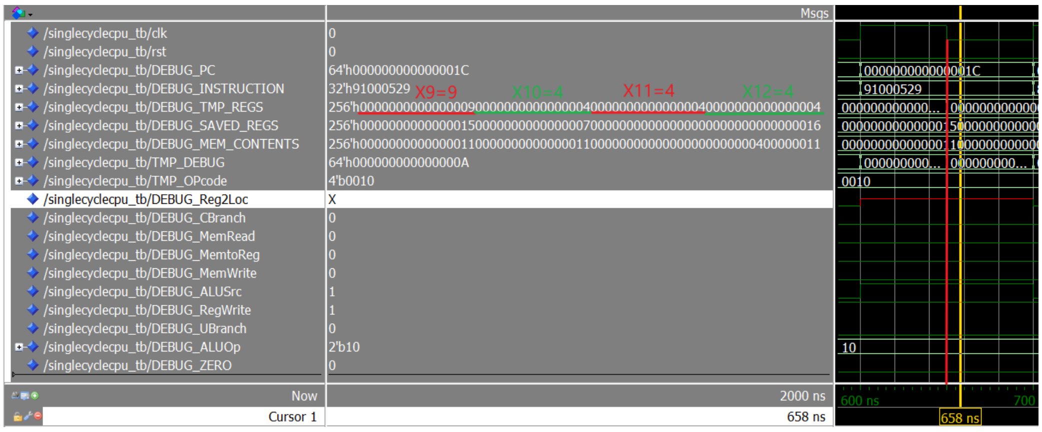
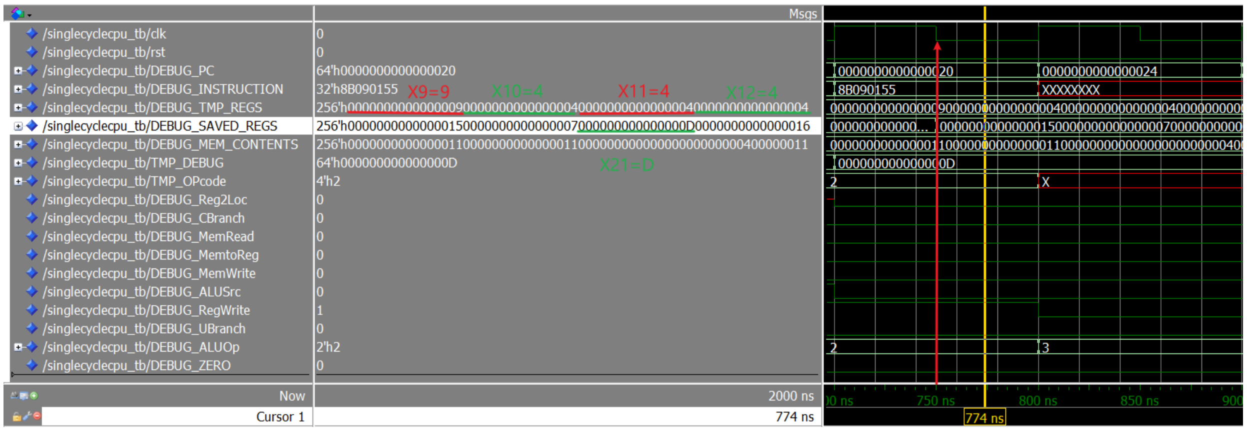

```
                                          Msgs
/singlecyclecpu_tb/clk                    0
/singlecyclecpu_tb/rst                    0
/singlecyclecpu_tb/DEBUG_PC               64'h000000000000001C
/singlecyclecpu_tb/DEBUG_INSTRUCTION      32'h91000529
/singlecyclecpu_tb/DEBUG_TMP_REGS         256'h0000000000000009000000000000000400000000000000040000000000000004
/singlecyclecpu_tb/DEBUG_SAVED_REGS       256'h0000000000000015000000000000000700000000000000000000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS     256'h0000000000000011000000000000001100000000000000000000000400000011
/singlecyclecpu_tb/TMP_DEBUG              64'h000000000000000A
/singlecyclecpu_tb/TMP_OPcode             4'b0010
/singlecyclecpu_tb/DEBUG_Reg2Loc          X
/singlecyclecpu_tb/DEBUG_CBranch          0
/singlecyclecpu_tb/DEBUG_MemRead          0
/singlecyclecpu_tb/DEBUG_MemtoReg         0
/singlecyclecpu_tb/DEBUG_MemWrite         0
/singlecyclecpu_tb/DEBUG_ALUSrc           1
/singlecyclecpu_tb/DEBUG_RegWrite         1
/singlecyclecpu_tb/DEBUG_UBranch          0
/singlecyclecpu_tb/DEBUG_ALUOp            2'b10
/singlecyclecpu_tb/DEBUG_ZERO             0
Now                                       2000 ns
Cursor 1                                  658 ns
```

Figure 40: Simulation Waveform: Final test: Instruction eight

After instruction seven, $X9=8, $X10=4, $X11=4, $X12=4. After running this instruction, $X9 should be $X9+1=9.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h1C. The instruction here is 32'h91000529, the same as instruction eight. The ALU control signal(TMP_OPcode)

of this instruction is 0010, add action of ADDI.
And the value of X9 is changed from 8 to 9 at 650 ns, a falling edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this ADDI instruction.

Insruction nine
16'h0008: out = 32'b10001011000010010000000101010101; // ADD X21, X10, X9
32'h8B090155



```
                                   Msgs
/singlecyclecpu_tb/clk             0
/singlecyclecpu_tb/rst             0
/singlecyclecpu_tb/DEBUG_PC        64'h0000000000000020
/singlecyclecpu_tb/DEBUG_INSTRUCTION 32'h8B090155
/singlecyclecpu_tb/DEBUG_TMP_REGS  256'h0000000000000009000000000000000400000000000000040000000000000004
/singlecyclecpu_tb/DEBUG_SAVED_REGS 256'h000000000000001500000000000000070000000000000000D0000000000000016
/singlecyclecpu_tb/DEBUG_MEM_CONTENTS 256'h0000000000000011000000000000001100000000000000000000000400000011
/singlecyclecpu_tb/TMP_DEBUG       64'h000000000000000D
/singlecyclecpu_tb/TMP_OPcode      4'h2
/singlecyclecpu_tb/DEBUG_Reg2Loc   0
/singlecyclecpu_tb/DEBUG_CBranch   0
/singlecyclecpu_tb/DEBUG_MemRead   0
/singlecyclecpu_tb/DEBUG_MemtoReg  0
/singlecyclecpu_tb/DEBUG_MemWrite  0
/singlecyclecpu_tb/DEBUG_ALUSrc    0
/singlecyclecpu_tb/DEBUG_RegWrite  1
/singlecyclecpu_tb/DEBUG_UBranch   0
/singlecyclecpu_tb/DEBUG_ALUOp     2'h2
/singlecyclecpu_tb/DEBUG_ZERO      0
Now                                2000 ns
Cursor 1                           774 ns
```

Figure 41: Simulation Waveform: Final test: Instruction nine

After instruction eight, \$X9=9, \$X10=4, \$X11=4, \$X12=4, \$X21=0. After running this instruction, \$X21 should be \$X10+\$X9=13=0xD.
From the DEBUG_TMP_REGS in the above waveform, we can see the PC is 64'h20. The instruction here is 32'h8B090155, the same as instruction nine. The ALU control signal(TMP_OPcode) of this instruction is 0010, add action of ADD.
And the value of X21 is changed from 0 to D at 750 ns, a falling edge of 'clk'.
And the last nine signals in the waveform are all the CPU control signals of this ADD instruction.
There is no more instructions after 800 ns.