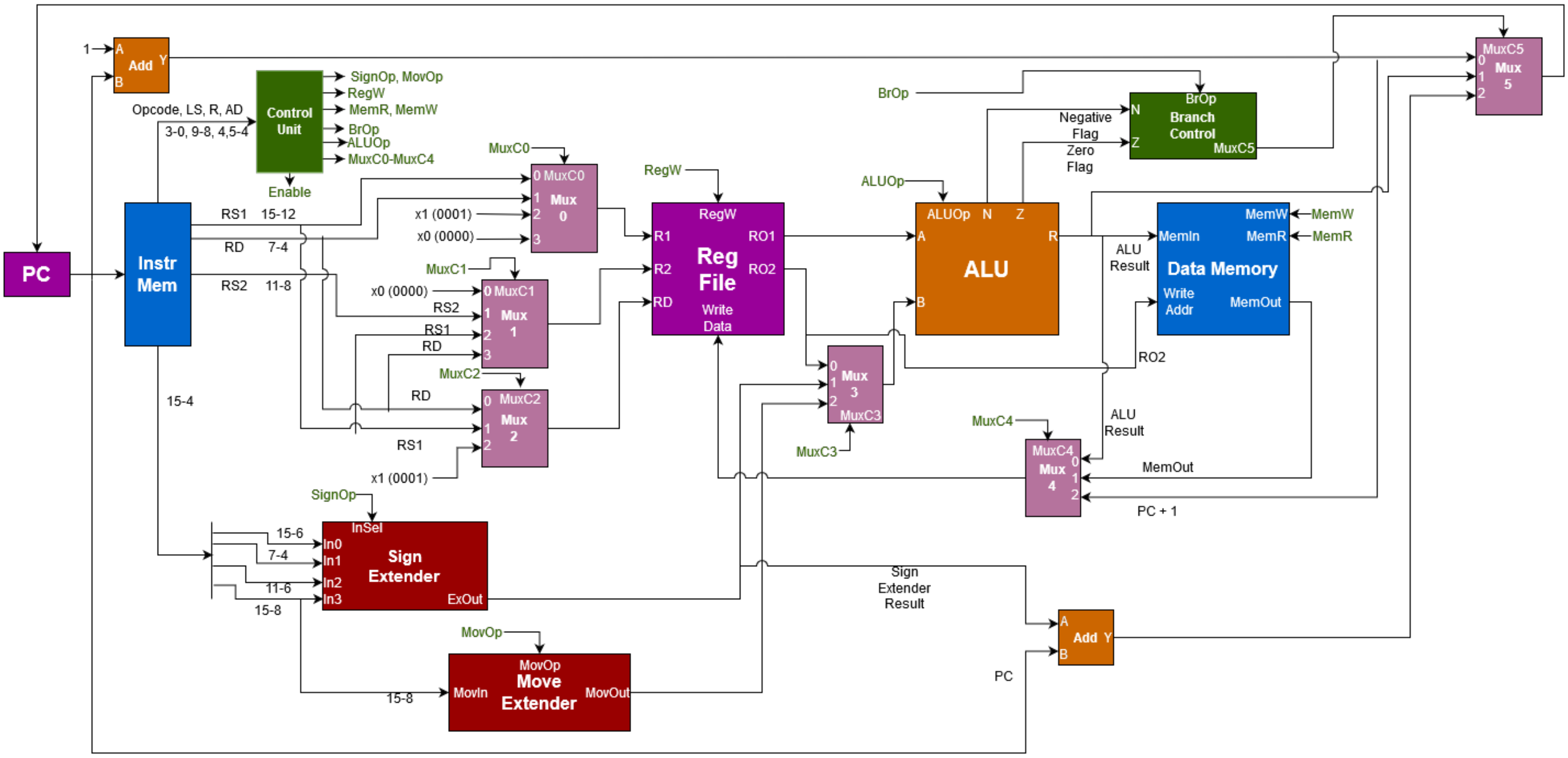
Top Level Component

1 Overview

- The control and datapath must be combined with the instruction and data memories.
- A program using all types of instructions must be written, compiled into binary, and simulated.
- The top-level component and memories code can be seen in Appendix A.
- The datapath can be seen below.



2 Test Program

A test program including all instructions was written. Different instruction results are also tested, such as branches being both taken and not taken. The assembly and binary instructions can be seen below.

Assembly:

```
       movu x5, 1
       addi x5, 1
       movl x6, 5
       str x5, x6
       jmp func
       halt



func:  ld x7, x5
       addr x8, x7, x6
       subr x9, x8, x6
       bne x0, x9, b1
       addr x9, x0, x0
b1:    nand x10, x0, x0
       andr x10, x10, x0
       orr x10, x10, x9
       beq x10, x0, b2
       shl x8, 2
b2:    sal x8, 2
       bgt x8, x0, b3
       addr x8, x0, x0
b4:    addr x9, x0, x0
b3:    shr x8, 1
       sar x8, 1
       blt x8, x0, b4
       ret
```

Machine Code:

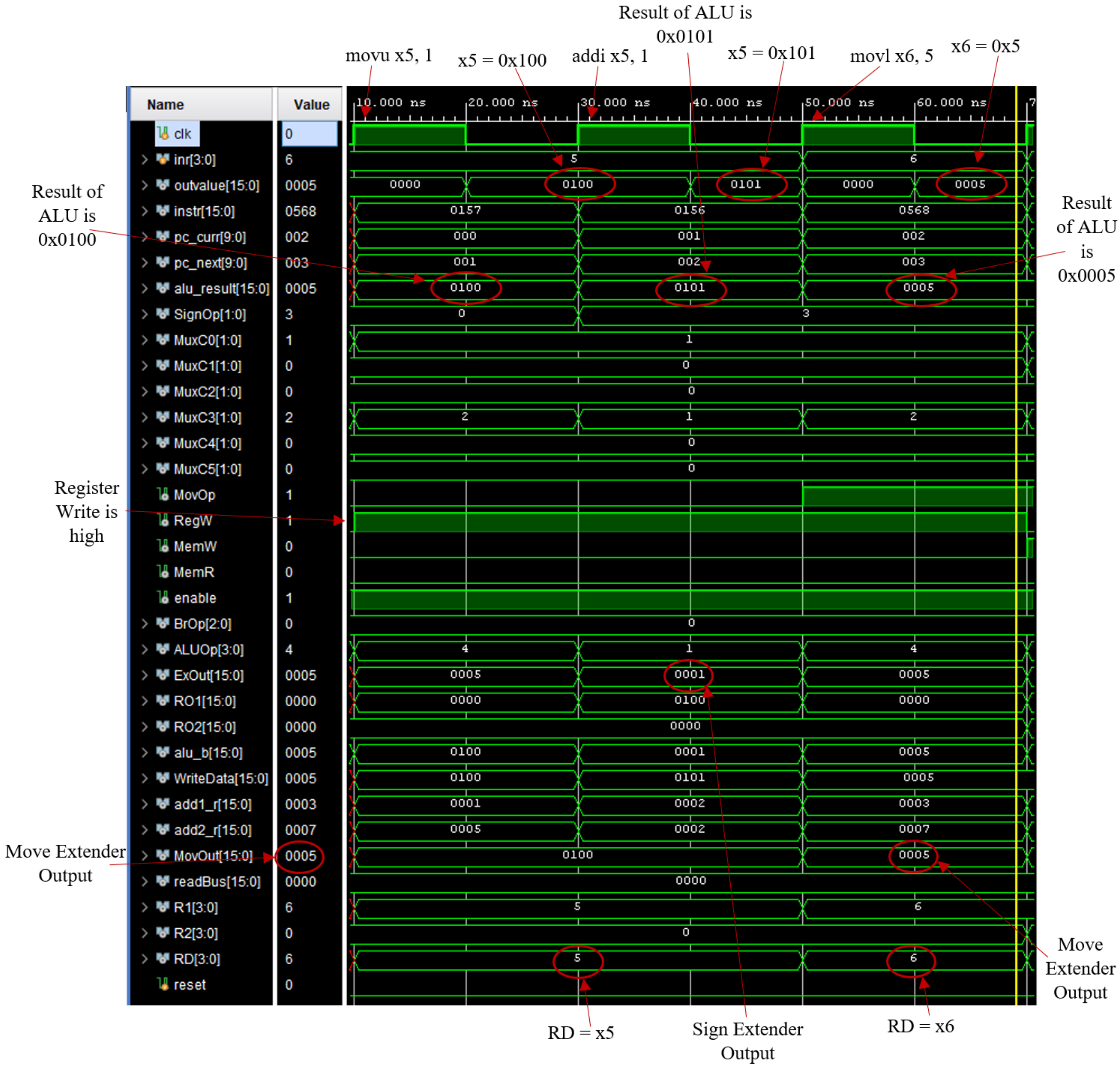
```
0000000101010111
0000000101010110
0000010101101000
0110000101011110
0000000101001001
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0101001001111110
0111011010000001
1000011010010101
0000100100101011
0000000010010001
0000000010100011
1010000010100010
1010100110100100
1010000000101010
1000000010001111
1000000010101111
1000000000111100
0000000010000001
0000000010010001
1000000001011111
1000000001111111
1000000011001101
0000000000011001
```

The table below details the expected results for each instruction.

Instr. Addr.	Binary	Instr.	Expected Result
0	0000000101010111	movu x5, 1	x5[15:8] = 0x01
1	0000000101010110	addi x5, 1	x5 = 0x0100 + 1 = 0x0101
2	0000010101101000	movl x6, 5	x6[7:0] = 0x05
3	0110000101011110	str x5, x6	data[x5] = x6
4	0000000101001001	jmp func	x1 = 0x5, PC = PC + 5 = 0x9
9	0101001001111110	ld x7, x5	x7 = data[x5] = 0x5
10	0111011010000001	addr x8, x7, x6	x8 = 0x5 + 0x5 = 0xA
11	1000011010010101	subr x9, x8, x6	x9 = 0xA - 0x5 = 0x5
12	0000100100101011	bne x0, x9, 2	Branch Taken, PC = PC + 2 = 0xE
14	0000000010100011	nand x10, x0, x0	x10 = x0 ∼ & x0 = 0xFFFF
15	1010000010100010	andr x10, x10, x0	x10 = 0xFFFF & 0x0 = 0
16	1010100110100100	orr x10, x10, x9	x10 = 0x0 | 0x5 = 0x5
17	1010000000101010	beq x10, x0, 2	Branch Not Taken, PC = PC + 1 = 0x12
18	1000000010001111	shl x8, 2	x8 = 0xA << 2 = 0x28
19	1000000010101111	sal x8, 2	x8 = 0x28 <<< 2 = 0xA0
20	1000000000111100	bgt x8, x0, 3	Branch Taken, PC = PC + 3 = 0x17
23	1000000001011111	shr x8, 1	x8 = 0xA0 >> 1 = 0x50
24	1000000001111111	sar x8, 1	x8 = 0x50 >>> 1 = 0x28
25	1000000011001101	blt x8, x0, -4	Branch Not Taken, PC = PC + 1 = 0x1A
26	0000000000011001	ret	PC = x1 = 0x5
5	0000000000000000	halt	Enable = 0, Program stops

In the simulation, the instructions above were placed into the instruction memory. The resulting simulation waveform can be seen below.

The simulation waveform was observed to match the expected results in the above table. Thus, the program was confirmed to function correctly.



Name	Value
clk	0
inr[3:0]	6
outvalue[15:0]	0005
instr[15:0]	0568
pc_curr[9:0]	002
pc_next[9:0]	003
alu_result[15:0]	0005
SignOp[1:0]	3
MuxC0[1:0]	1
MuxC1[1:0]	0
MuxC2[1:0]	0
MuxC3[1:0]	2
MuxC4[1:0]	0
MuxC5[1:0]	0
MovOp	1
RegW	1
MemW	0
MemR	0
enable	1
BrOp[2:0]	0
ALUOp[3:0]	4
ExOut[15:0]	0005
RO1[15:0]	0000
RO2[15:0]	0000
alu_b[15:0]	0005
WriteData[15:0]	0005
add1_r[15:0]	0003
add2_r[15:0]	0007
MovOut[15:0]	0005
readBus[15:0]	0000
R1[3:0]	6
R2[3:0]	0
RD[3:0]	6
reset	0

movu x5, 1

x5 = 0x100

addi x5, 1

Result of ALU is 0x0101

x5 = 0x101

movl x6, 5

x6 = 0x5

Result of ALU is 0x0100

Result of ALU is 0x0005

Register Write is high

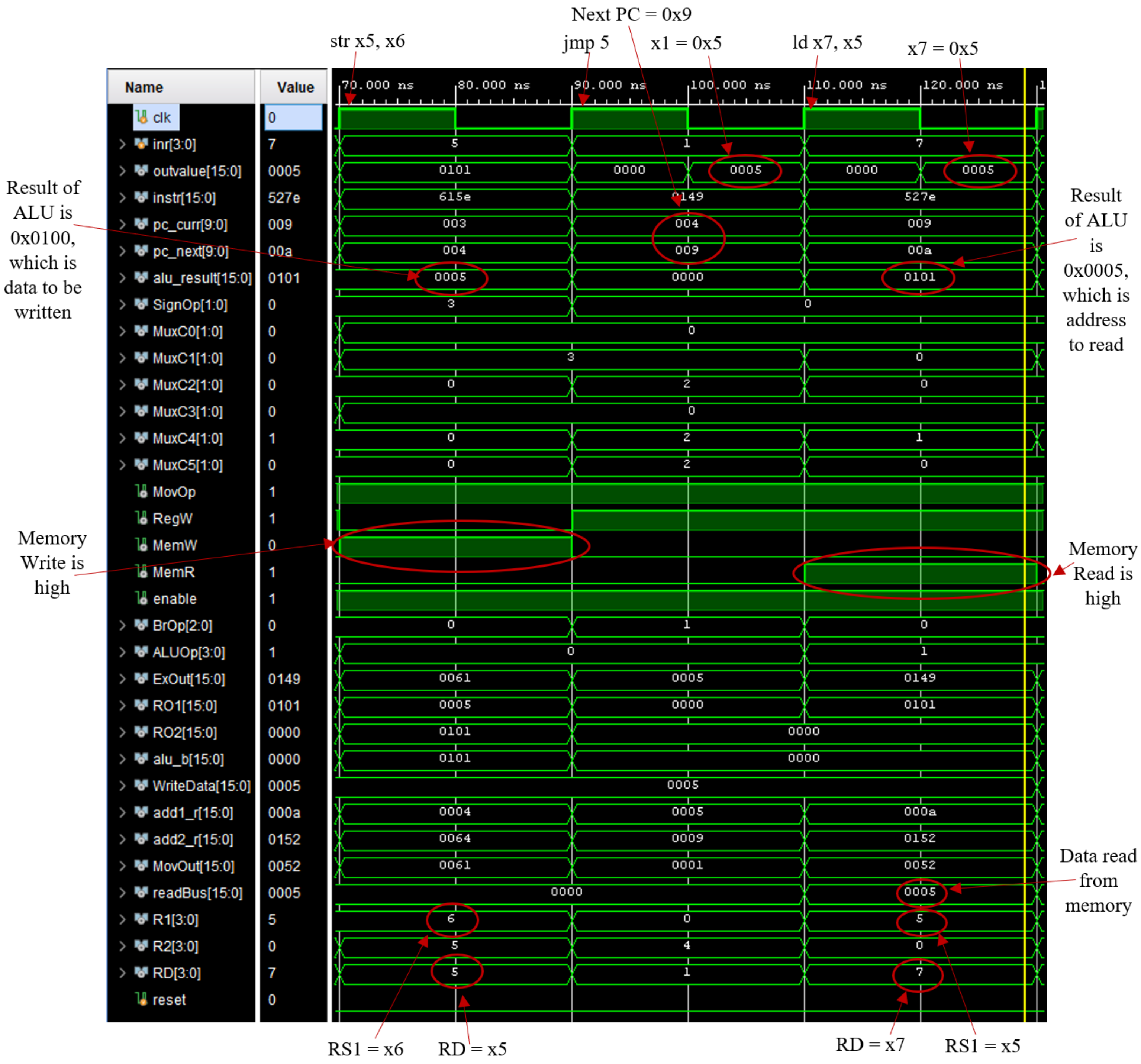
Move Extender Output

Move Extender Output

RD = x5

Sign Extender Output

RD = x6



Name	Value
clk	0
inr[3:0]	7
outvalue[15:0]	0005
instr[15:0]	527e
pc_curr[9:0]	009
pc_next[9:0]	00a
alu_result[15:0]	0101
SignOp[1:0]	0
MuxC0[1:0]	0
MuxC1[1:0]	0
MuxC2[1:0]	0
MuxC3[1:0]	0
MuxC4[1:0]	1
MuxC5[1:0]	0
MovOp	1
RegW	1
MemW	0
MemR	1
enable	1
BrOp[2:0]	0
ALUOp[3:0]	1
ExOut[15:0]	0149
RO1[15:0]	0101
RO2[15:0]	0000
alu_b[15:0]	0000
WriteData[15:0]	0005
add1_r[15:0]	000a
add2_r[15:0]	0152
MovOut[15:0]	0052
readBus[15:0]	0005
R1[3:0]	5
R2[3:0]	0
RD[3:0]	7
reset	0

str x5, x6

Next PC = 0x9

jmp 5

x1 = 0x5

ld x7, x5

x7 = 0x5

Result of ALU is 0x0100, which is data to be written

Result of ALU is 0x0005, which is address to read

Memory Write is high

Memory Read is high

Data read from memory

RS1 = x6

RD = x5

RD = x7

RS1 = x5

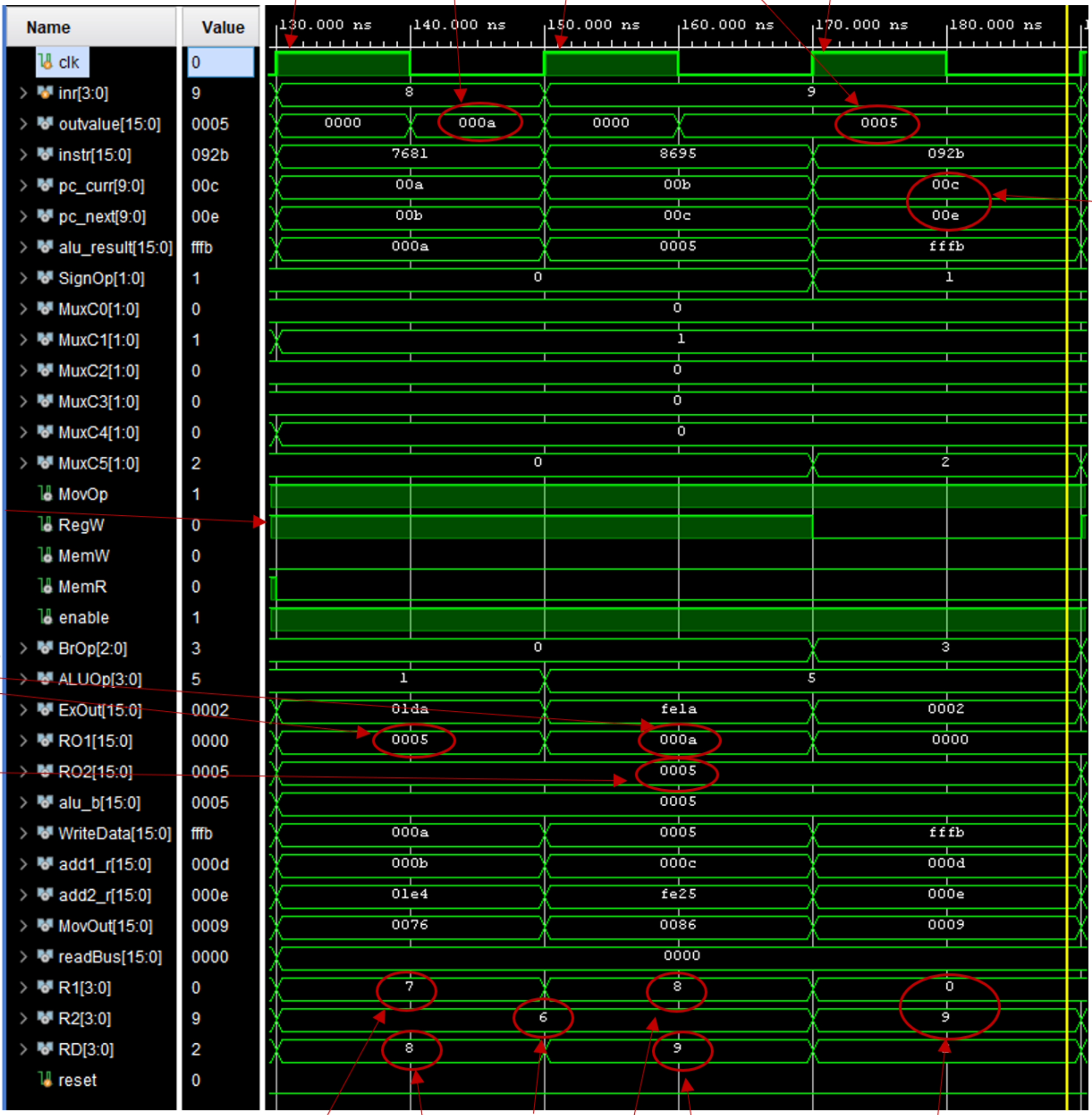
x8 = 0xA

addr x8, x7, x6

x9 = 0x5

subr x9, x8, x6

bne x0, x9, 2



Name	Value
clk	0
inr[3:0]	9
outvalue[15:0]	0005
instr[15:0]	092b
pc_curr[9:0]	00c
pc_next[9:0]	00e
alu_result[15:0]	fffb
SignOp[1:0]	1
MuxC0[1:0]	0
MuxC1[1:0]	1
MuxC2[1:0]	0
MuxC3[1:0]	0
MuxC4[1:0]	0
MuxC5[1:0]	2
MovOp	1
RegW	0
MemW	0
MemR	0
enable	1
BrOp[2:0]	3
ALUOp[3:0]	5
ExOut[15:0]	0002
RO1[15:0]	0000
RO2[15:0]	0005
alu_b[15:0]	0005
WriteData[15:0]	fffb
add1_r[15:0]	000d
add2_r[15:0]	000e
MovOut[15:0]	0009
readBus[15:0]	0000
R1[3:0]	0
R2[3:0]	9
RD[3:0]	2
reset	0

Branch Taken, PC = PC + 2 = 0xE

Register Write is high

Output of RS1

Output of RS2

RS1 = x7

RD = x8

RS2 = x6

RD = x9

RS1 = x8

RS1 = x0
RS2 = x9

x10 = 0xFFFF

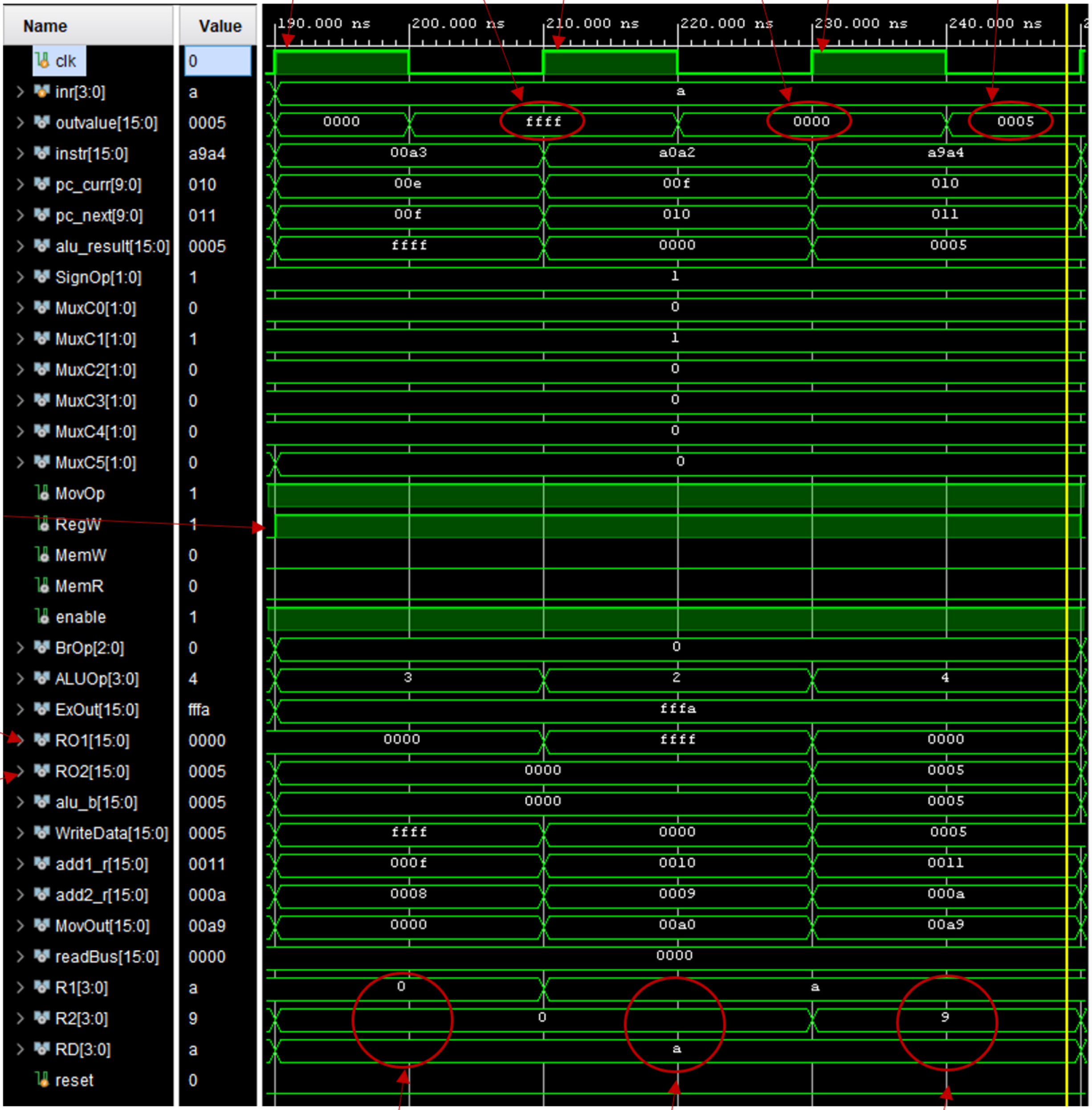
nand x10, x0, x0

andr x10, x10, x0

x10 = 0x0

orr x10, x10, x9

x10 = 0x5



Name	Value
clk	0
inr[3:0]	a
outvalue[15:0]	0005
instr[15:0]	a9a4
pc_curr[9:0]	010
pc_next[9:0]	011
alu_result[15:0]	0005
SignOp[1:0]	1
MuxC0[1:0]	0
MuxC1[1:0]	1
MuxC2[1:0]	0
MuxC3[1:0]	0
MuxC4[1:0]	0
MuxC5[1:0]	0
MovOp	1
RegW	1
MemW	0
MemR	0
enable	1
BrOp[2:0]	0
ALUOp[3:0]	4
ExOut[15:0]	fffa
RO1[15:0]	0000
RO2[15:0]	0005
alu_b[15:0]	0005
WriteData[15:0]	0005
add1_r[15:0]	0011
add2_r[15:0]	000a
MovOut[15:0]	00a9
readBus[15:0]	0000
R1[3:0]	a
R2[3:0]	9
RD[3:0]	a
reset	0

Register Write is high

Output of RS1

Output of RS2

RS1 = x0
RS2 = x0
RD = x10

RS1 = x10
RS2 = x0
RD = x10

RS1 = x10
RS2 = x9
RD = x10

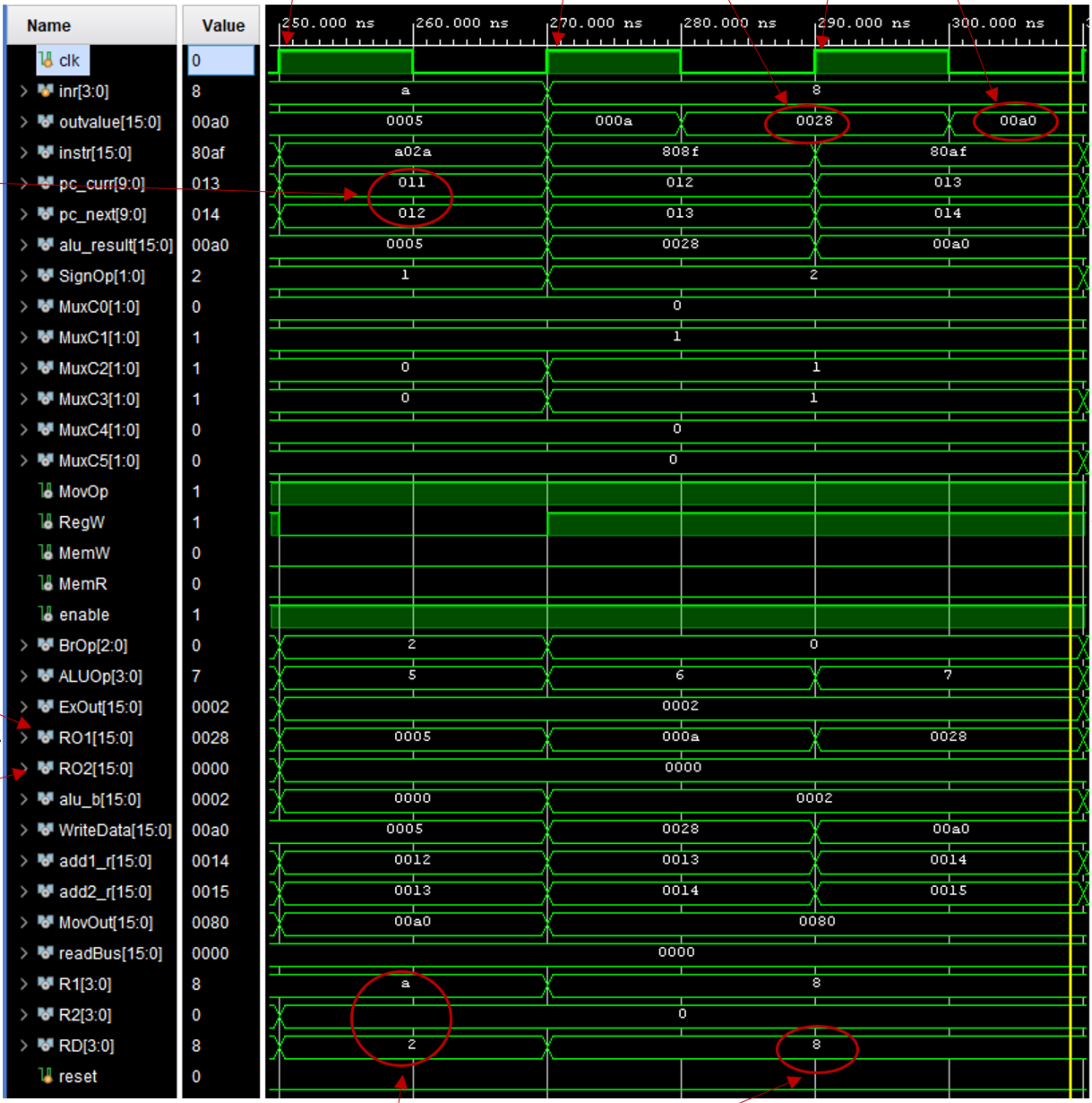
x8 = 0x28

x8 = 0xA0

beq x10, x0, 2

shl x8, 2

sal x8, 2



Name	Value
clk	0
inr[3:0]	8
outvalue[15:0]	00a0
instr[15:0]	80af
pc_curr[9:0]	013
pc_next[9:0]	014
alu_result[15:0]	00a0
SignOp[1:0]	2
MuxC0[1:0]	0
MuxC1[1:0]	1
MuxC2[1:0]	1
MuxC3[1:0]	1
MuxC4[1:0]	0
MuxC5[1:0]	0
MovOp	1
RegW	1
MemW	0
MemR	0
enable	1
BrOp[2:0]	0
ALUOp[3:0]	7
ExOut[15:0]	0002
RO1[15:0]	0028
RO2[15:0]	0000
alu_b[15:0]	0002
WriteData[15:0]	00a0
add1_r[15:0]	0014
add2_r[15:0]	0015
MovOut[15:0]	0080
readBus[15:0]	0000
R1[3:0]	8
R2[3:0]	0
RD[3:0]	8
reset	0

Branch Not Taken, PC = PC + 1 = 0x12

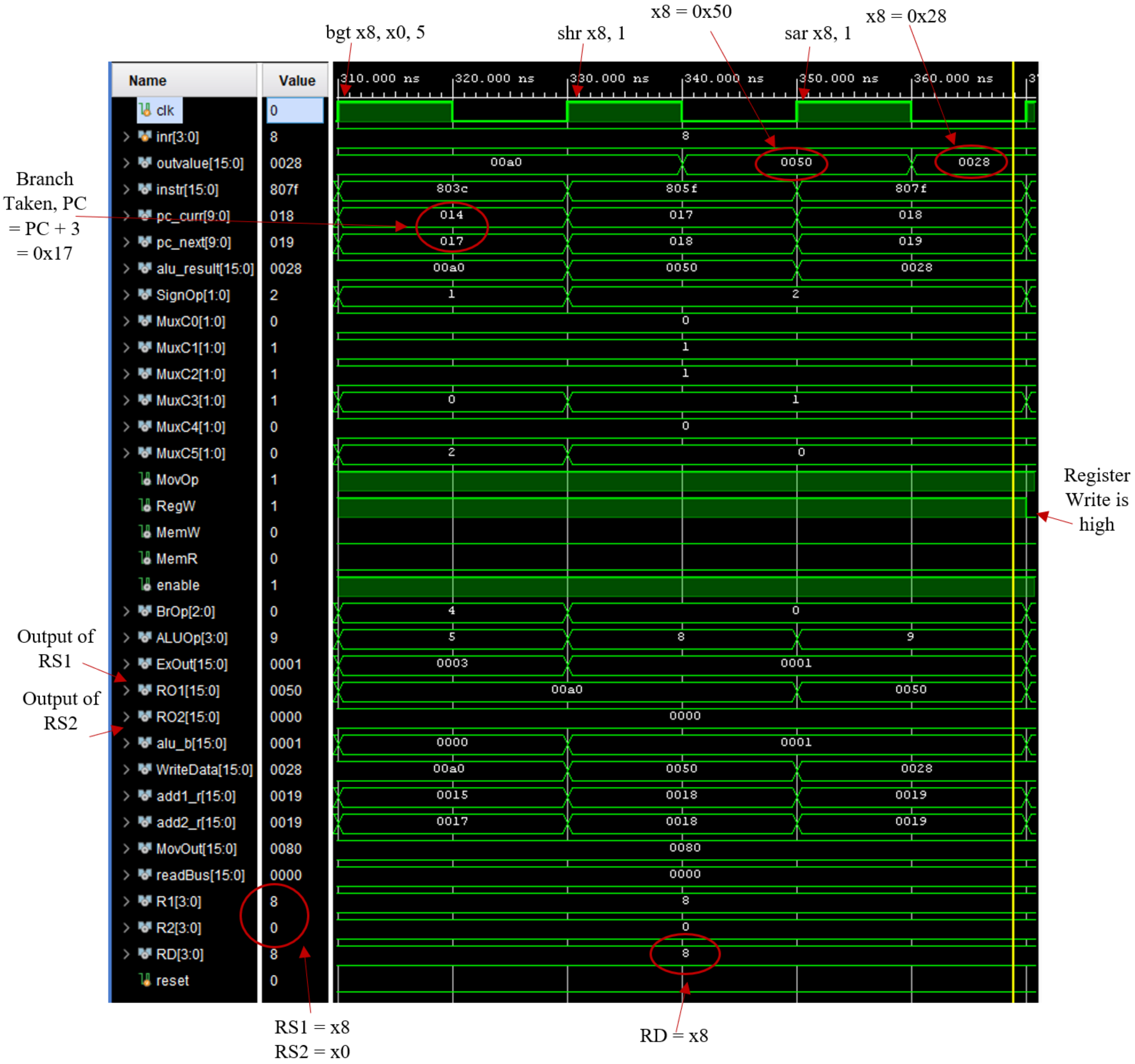
Output of RS1

Output of RS2

Register Write is high

RS1 = x10
RS2 = x0

RD = x8



Name	Value
clk	0
inr[3:0]	8
outvalue[15:0]	0028
instr[15:0]	807f
pc_curr[9:0]	018
pc_next[9:0]	019
alu_result[15:0]	0028
SignOp[1:0]	2
MuxC0[1:0]	0
MuxC1[1:0]	1
MuxC2[1:0]	1
MuxC3[1:0]	1
MuxC4[1:0]	0
MuxC5[1:0]	0
MovOp	1
RegW	1
MemW	0
MemR	0
enable	1
BrOp[2:0]	0
ALUOp[3:0]	9
ExOut[15:0]	0001
RO1[15:0]	0050
RO2[15:0]	0000
alu_b[15:0]	0001
WriteData[15:0]	0028
add1_r[15:0]	0019
add2_r[15:0]	0019
MovOut[15:0]	0080
readBus[15:0]	0000
R1[3:0]	8
R2[3:0]	0
RD[3:0]	8
reset	0

bgt x8, x0, 5

shr x8, 1

x8 = 0x50

sar x8, 1

x8 = 0x28

Branch Taken, PC = PC + 3 = 0x17

Register Write is high

Output of RS1

Output of RS2

RS1 = x8
RS2 = x0

RD = x8

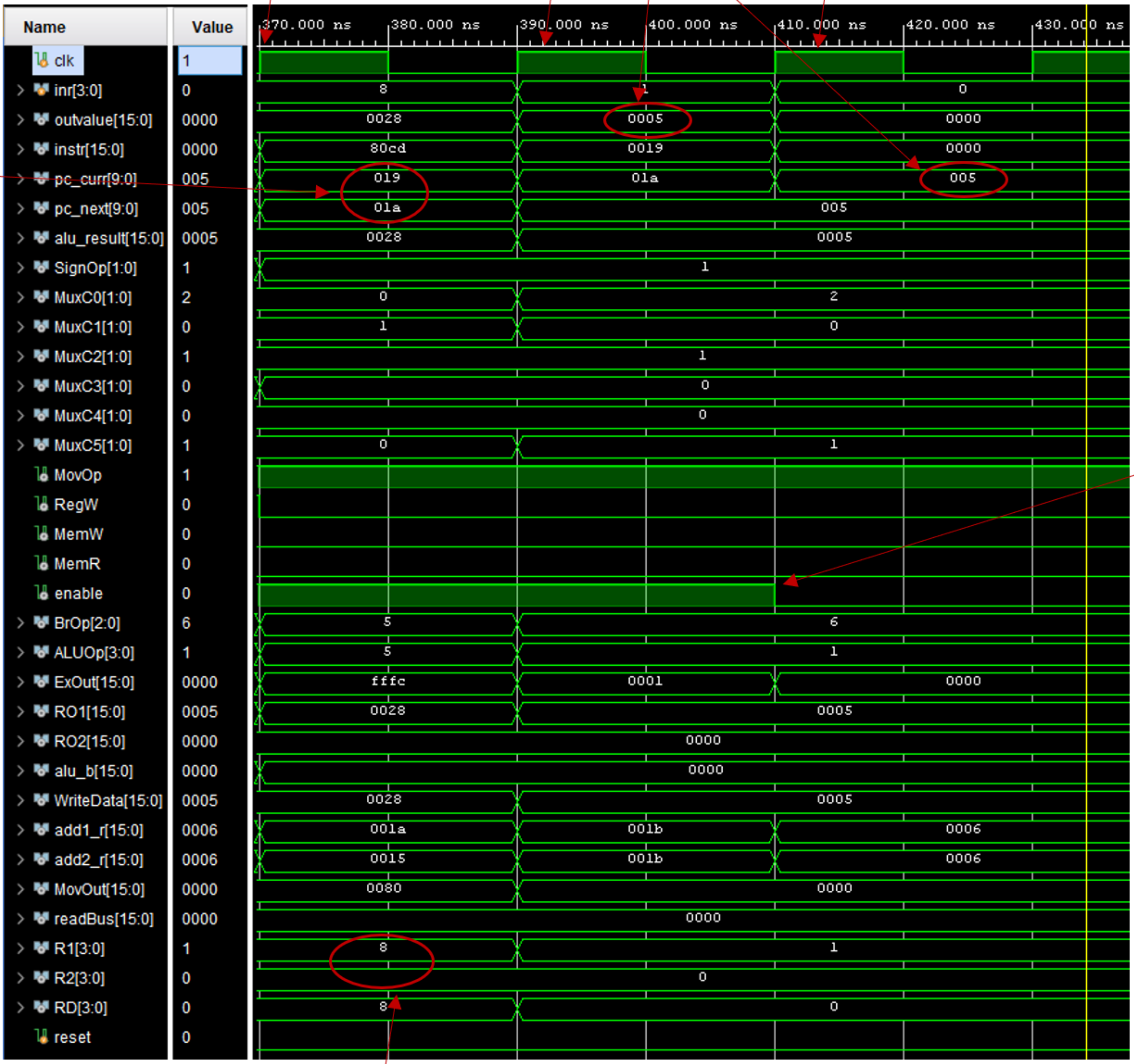
blt x8, x0, -4

ret

x1 = 0x5, which becomes next PC

halt

Branch Not Taken, PC = PC + 1 = 0x1A



Name	Value
clk	1
inr[3:0]	0
outvalue[15:0]	0000
instr[15:0]	0000
pc_curr[9:0]	005
pc_next[9:0]	005
alu_result[15:0]	0005
SignOp[1:0]	1
MuxC0[1:0]	2
MuxC1[1:0]	0
MuxC2[1:0]	1
MuxC3[1:0]	0
MuxC4[1:0]	0
MuxC5[1:0]	1
MovOp	1
RegW	0
MemW	0
MemR	0
enable	0
BrOp[2:0]	6
ALUOp[3:0]	1
ExOut[15:0]	0000
RO1[15:0]	0005
RO2[15:0]	0000
alu_b[15:0]	0000
WriteData[15:0]	0005
add1_r[15:0]	0006
add2_r[15:0]	0006
MovOut[15:0]	0000
readBus[15:0]	0000
R1[3:0]	1
R2[3:0]	0
RD[3:0]	0
reset	0

Enable goes low, execution stops

RS1 = x8
RS2 = x0

Appendix A Component Code

A.1 Top-level Component Code

```
module top_CPU(clk, reset, inr, outvalue);
    input clk;
    input [3:0] inr;
    input reset;
    output [15:0] outvalue;

    wire [15:0] pc_in, pc_out, readBus, RO2, result;
    wire [15:0] instr, ExOut, RO1, alu_b, WriteData, add1_r, add2_r, MovOut,
                R1, R2, RD;
    wire [1:0] SignOp, MuxC0, MuxC1, MuxC2, MuxC3, MuxC4, MuxC5;
    wire MovOp, RegW, enable, MemW, MemR;
    wire [2:0] BrOp;
    wire [3:0] ALUOp;

    instr_mem INSTR (enable, pc_out[9:0], instr);
    data_mem DATA (MemW, MemR, result, RO2, readBus);
    control_and_datapath CAD (clk, reset, inr, readBus, outvalue, instr,
                              pc_out, result, RO2, MemW, MemR, enable);
endmodule
```

A.2 Instruction Memory

```
module instr_mem(enable, pc, out);
    input [9:0] pc;
    output reg [15:0] out;
    reg [15:0] instructions [999:0];
    input enable;

    initial
    begin
        $readmemb("instructions.mem", instructions);
    end

    always @ (pc)
    begin
        if (enable)
            out = instructions[pc];
    end
endmodule
```

A.3 Data Memory

```
module data_mem(MemW, MemR, MemIn, WriteAddr, MemOut);
    input MemW, MemR;
    input [15:0] MemIn, WriteAddr;
    output reg [15:0] MemOut;
    reg [15:0] data [999:0];

    integer i;
    initial
    begin
       for (i=0;i<=999;i=i+1)
            data[i] = 0;
    end

    always @ (MemW or MemR or MemIn or WriteAddr)
    begin
        if(MemW)
            data[WriteAddr[9:0]] <= MemIn;
        if(MemR)
            MemOut <= data[MemIn[9:0]];
        else
            MemOut <= 0;
    end
endmodule
```