Lab 7 Post lab – Simplified DLX

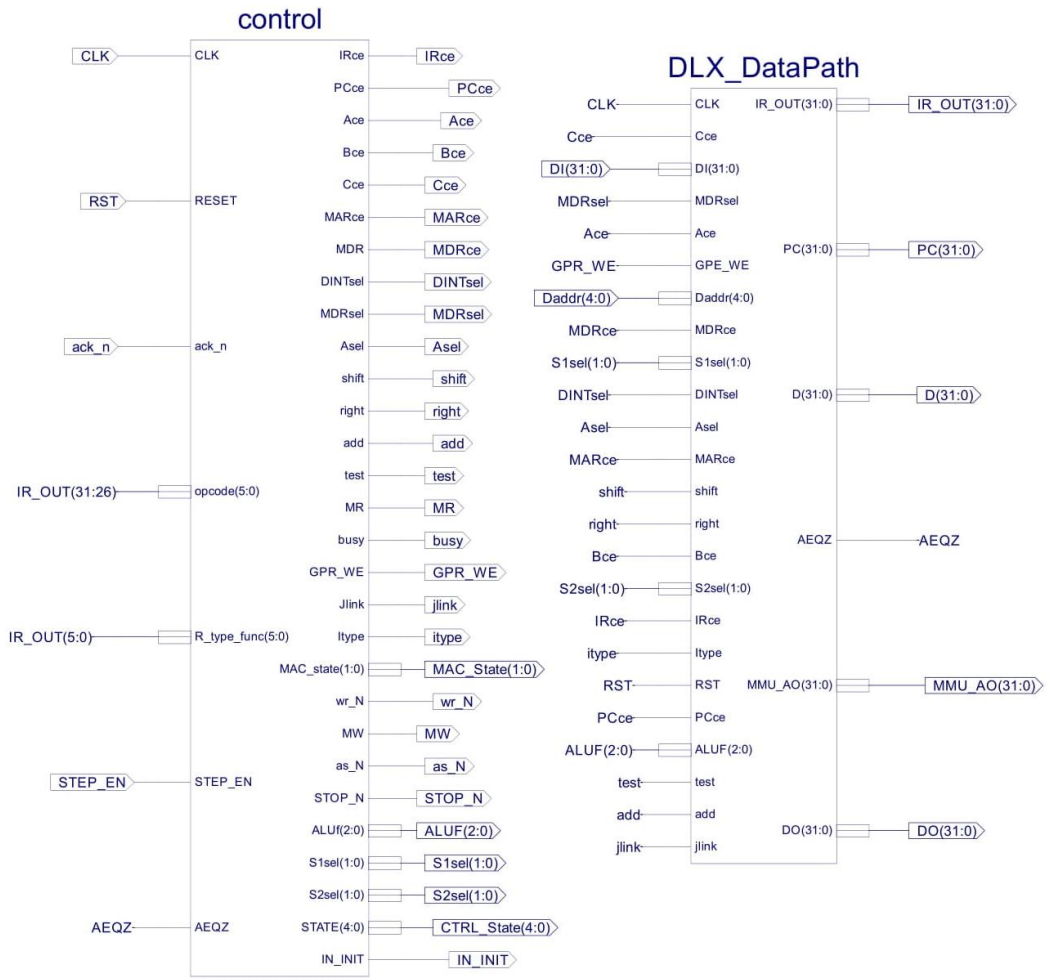
Yohai Shiloh

Yarin Koren

Group: B2

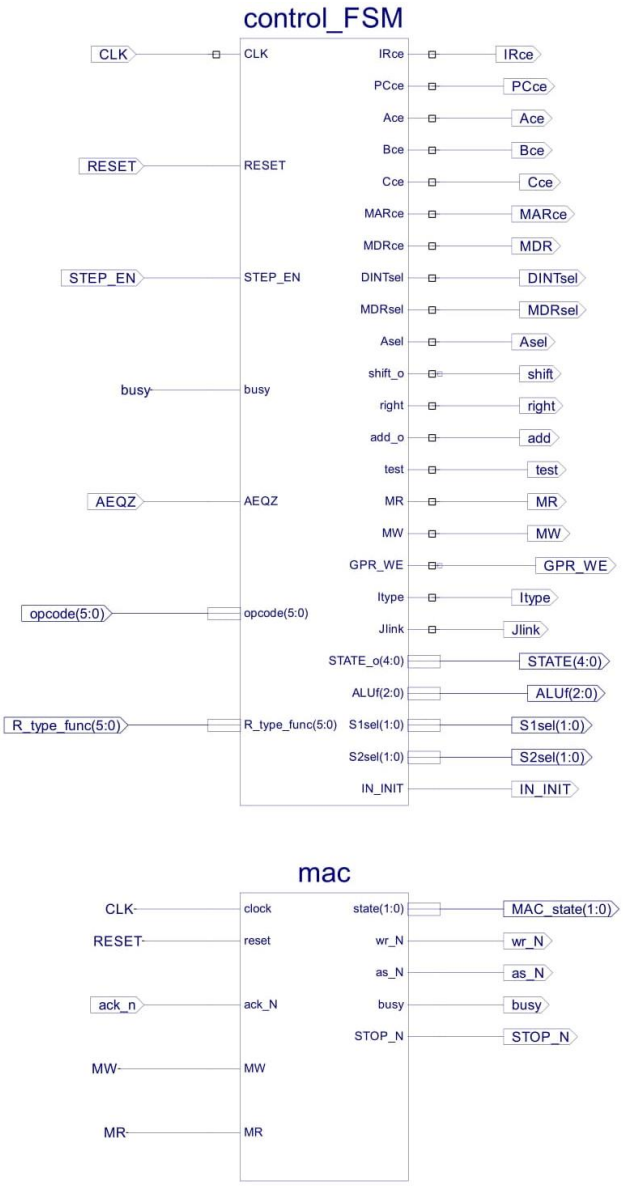
DLX – Top Level

The scheme of the connections between the data path of the processor and the control unit(including the MAC state machine as part of its symbol) attached below:



The Control

Control Top level schematic:



Note: The MAC was implemented the same as in the previous lab and therefore will not be detailed again.

Finite State Machine Verilog code:

The Finite State Machine of the control was written in Verilog(code attached below). The FSM code is built of four main parts – parameters definitions, states transition definition, sequential part and combinational part. The first defines toe parameters we would use in the code to make it more readable, and includes the states, paths, instructions etc. The second includes the conditions for transitions from one state to another, the third take care of resets and updating the states according to the clock and the fourth define the signals the FSM would have to generate to control the MAC, the data-path and ensure its right functionality.

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date:    23:39:45 07/06/2024
7   // Design Name:
8   // Module Name:    control_FSM
9   // Project Name:
10  // Target Devices:
11  // Tool versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////
21  module control_FSM(
22      input CLK,
23      input RESET,
24      input STEP_EN,
25      input busy,
26      input [5:0] opcode,
27      input [5:0] R_type_func, // Instruction's 6 LSB
28      input AEQZ,
29      output IN_INIT,
30      output [4:0] STATE_o,
31      output [2:0] ALUf,
32      output IRce,
33      output PCce,
34      output Ace,
35      output Bce,
36      output Cce,
37      output MARce,
38      output MDRce,
39      output [1:0] S1sel,
40      output [1:0] S2sel,
41      output DINTsel,
42      output MDRsel,
43      output Asel,
44      output shift_o,
45      output right,
46      output add_o,
47      output test,
48      output MR,
49      output MW,
50      output GPR_WE,
51      output Itype,
52      output Jlink
53      );
54
55  // Supporting registers
56  reg [4:0] next_state;
57  reg [4:0] STATE;
58  reg hold; // Small FSM letch indicator. Equal to one only if
59            // it's in MEM transaction state for one cycle or more
60  wire bt; // 'Branch taken', determines either will branch or not
61
62  // states parameters definitions:
63  parameter INIT = 5'h00;
64  parameter fetch = 5'h01;
65  parameter decode = 5'h02;
66  parameter halt = 5'h1f;
67  parameter ALU = 5'h03;
68  parameter shift_state = 5'h04;
69  parameter WBR = 5'h05;
70  parameter ALUI= 5'h06;
71  parameter testI = 5'h07;
72  parameter WBI = 5'h08;
73  parameter addessCMP = 5'h09;
74  parameter load = 5'h0a;
75  parameter copyMDR2C = 5'h0b;
76  parameter copyGPR2MDR = 5'h0c;
77  parameter store = 5'h0d;
78  parameter JR = 5'h0e;
79  parameter savePC = 5'h0f;
80  parameter JALR = 5'h10;
81  parameter branch = 5'h11;
82  parameter Btaken = 5'h12;
83
84  // R type instructions' Function code
85  parameter R_inst_opcode = 6'b000000;
86  parameter slli = 3'b000;
87  parameter srli = 3'b010;
88  parameter add_func= 3'b011;
89  parameter sub = 3'b010;
90  parameter and_logic = 3'b110;
91  parameter or_logic = 3'b101;
92  parameter xor_logic = 3'b100;
93
94  // I type instructions' Opcode
95  parameter D1 = 3'b110; // special nop - go to INIT/fetch
96  parameter D5 = 3'b001;
97  parameter D6 = 3'b011;
98  parameter D7 = 2'b10; // lw and sw instruction flow
99  parameter D8 = 6'b010110;
100 parameter D9 = 6'b010111;
101 parameter D12 = 5'b00010;
102 parameter lw = 4'b0011;
103 parameter sw = 4'b1011;
104 parameter addi =6'b001011;
105
106 always @(STATE or busy or STEP_EN) begin // State Machine flow
107     if(~STEP_EN) next_state = INIT;
108     case(STATE)
109         INIT:
110             if(STEP_EN) next_state = fetch;
111             else next_state = INIT;
112         fetch:
113             if(~busy & hold) next_state = decode;
114             else next_state = fetch;
```

```
115          decode:
116              if (opcode == R_inst_opcode) // R type instruction was received
117                  if(R_type_func[5]==1) next_state = ALU;
118                  else if(R_type_func[5] == 0) next_state = shift_state;
119                  else next_state = halt; // Undefined R type instruction was detected
120              else // I type instruction
121                  if(opcode[5:3] == D1) // special nop
122                      next_state = (STEP_EN) ? fetch : INIT;
123                  else if(opcode == addi) next_state = ALUI;
124                  else if(opcode[5:3] == D6) next_state = testI;
125                  else if(opcode[5:4]== D7) next_state = addessCMP;
126                  else if(opcode == D8) next_state = JR;
127                  else if(opcode == D9) next_state = savePC;
128                  else if(opcode[5:1] == D12) next_state = branch;
129                  else next_state = halt;
130          ALU, shift_state:
131              next_state = WBR;
132          ALUI, testI, copyMDR2C:
133              next_state = WBI;
134          WBI, WBR, JR, JALR, Btaken:
135              next_state = (STEP_EN) ? fetch : INIT;
136          addessCMP:
137              next_state = (opcode[3]) ? copyGPR2MDR : load;
138          copyGPR2MDR:
139              next_state = store;
140          store:
141              if(~busy & hold) next_state = INIT;
142              else next_state = store;
143          load:
144              if(~busy & hold) next_state = copyMDR2C;
145              else next_state = load;
146          savePC:
147              next_state = JALR;
148          branch:
149              if(bt) next_state = Btaken; // Branch at the next step
150              else next_state = (STEP_EN) ? fetch : INIT; // Don't branch and read the
    next instruction
151          halt:
152              next_state = halt;
153
154          default:
155              next_state = INIT;
156
157          endcase
158      end
159
160  always @(posedge CLK) begin // Sequential part: checks reset and pass STATE forward
161      if (RESET)
162          STATE <= INIT;
163      else begin
164        STATE <= #1 next_state;
165        hold <= ((STATE == fetch)|(STATE == store)|(STATE == load)) ? 1 : 0;
166        end
167      end
168
169  // Combinational part - compute output signals and conditions for the next state
170  assign IN_INIT = ((STATE == INIT) || (STATE == halt)) ? 1 : 0;
171  assign STATE_o = STATE;
172  assign ALUf = ((opcode[5:3] == 3'b001)||(opcode[5:3] == 3'b011)) ? opcode[2:0] : (
    opcode == R_inst_opcode)&(R_type_func[5:3] == 3'b100) ? R_type_func[2:0] : 3'b000;
173  assign IRce = (STATE == fetch) ? 1:0;
174  assign PCce = (STATE == decode)|(STATE == Btaken)|(STATE == JR)|(STATE == JALR) ? 1:0;
175  assign Ace = (STATE == decode) ? 1:0;
176  assign Bce = Ace;
177  assign Cce = (STATE == ALU)|(STATE == testI)|(STATE == ALUI)|(STATE == shift_state)|(
    STATE == copyMDR2C)|(STATE == savePC) ? 1:0;
178  assign MARce = (STATE == addessCMP) ? 1:0;
179  assign MDRce = (STATE == load)|(STATE == copyGPR2MDR) ? 1:0;
180  assign S1sel[1] = (STATE == copyGPR2MDR)|(STATE == copyMDR2C) ? 1:0;
181  assign S1sel[0] = ((Cce)&(STATE != savePC))|(STATE == addessCMP)|(STATE == JR)|(STATE
    == JALR) ? 1:0;
182  assign S2sel = (STATE == decode) ? 2'b11 : ((opcode[5:3] == 3'b010)|(STATE ==
    copyGPR2MDR)|(STATE==copyMDR2C)) ? 2'b10 : ((STATE == addessCMP)||((~GPR_WE)&Itype)||(
    STATE == Btaken)) ? 2'b01 : 2'b0;
183  assign DINTsel = (shift_o)|(STATE == copyGPR2MDR)|(STATE == copyMDR2C) ? 1:0;
184  assign MDRsel = (STATE == load) ? 1 : 0;
185  assign Asel = (STATE == load)|(STATE == store) ? 1 : 0;
186  assign shift_o = (STATE == shift_state) ? 1 : 0;
187  assign right = (shift_o & R_type_func[1]);
188  assign add_o = (STATE ==decode)|(STATE == ALUI)|(STATE == addessCMP)|(STATE == Btaken
    )| (STATE ==JR)|(STATE == savePC)|(STATE == JALR) ? 1 : 0;
189  assign test = (STATE == testI) ? 1 : 0;
190  assign MR = (STATE == fetch)|(STATE == load) ? 1 : 0;
191  assign MW = (STATE == store) ? 1 : 0;
192  assign GPR_WE = (STATE ==WBI)|(STATE == WBR)|(STATE == JALR) ? 1 : 0;
193  assign Itype = (STATE == ALUI)|(STATE == testI)|(STATE == WBI) ? 1 : 0;
194  assign Jlink = (STATE == JALR) ? 1 : 0;
195  assign bt = (AEQZ ^ opcode[0]);
196
197  endmodule
```

Test Vectors and Control Validation

Test vectors

To improve the readability of the test vectors, we marked the different groups of signals in separated colors(light blue for inputs, orange for MAC related signals, purple for the select signals of the muxes and so on). In addition, the state of the control is written in hexadecimal for similar reasons. Table of states' numbering is attached below:

State	Number(Hex)	State	Number(Hex)
INIT	0	load	a
Fetch	1	copyMDR2C	b
Decode	2	copyGPR2MDR	c
ALU	3	store	d
Shift	4	JR	e
WBR	5	savePC	f
ALUI	6	JALR	10
testI	7	branch	11
WBI	8	Btaken	12
addressCAMP	9	halt	1f

ALU:

	Clock	0	1	2	3	4	5	6	7	8	9
Input	STEP_EN	0	0	1	0	0	0	0	0	0	0
Input	Reset	1	0	0	0	0	0	0	0	0	0
Input	Busy	0	0	0	1	1	0	0	0	1	0
Output	STATE Ctrl	0	0	1	1	1	1	2	3	5	0
Output	IN_INIT	1	1	0	0	0	0	0	0	0	1
Output	S1sel	00	00	00	00	00	00	00	01	00	00
Output	S2sel	00	00	00	00	00	00	11	00	00	00
Output	Asel	0	0	0	0	0	0	0	0	0	0
Output	DINTsel	0	0	0	0	0	0	0	0	0	0
Output	MDRsel	0	0	0	0	0	0	0	0	0	0
Output	IRce	0	0	1	1	1	1	0	0	0	0
Output	PCce	0	0	0	0	0	0	1	0	0	0
Output	Ace	0	0	0	0	0	0	1	0	0	0
Output	Bce	0	0	0	0	0	0	1	0	0	0
Output	Cce	0	0	0	0	0	0	0	1	0	0
Output	MARce	0	0	0	0	0	0	0	0	0	0
Output	MDRce	0	0	0	0	0	0	0	0	0	0
Output	GPR_WE	0	0	0	1	0	0	0	0	1	0
Output	Itype	0	0	0	0	0	0	0	0	0	0
Output	Jlink	0	0	0	0	0	0	0	0	0	0
Output	test	0	0	0	0	0	0	0	0	0	0
Output	add	0	0	0	0	0	0	1	0	0	0
Output	shift	0	0	0	0	0	0	0	0	0	0
Output	right	0	0	0	0	0	0	0	0	0	0
Output	MR	0	0	1	1	1	1	0	0	0	0
Output	MW	0	0	0	0	0	0	0	0	0	0
Output	STATE MAC	00	00	00	01	01	10	00	00	00	00
Input(MAC)	ACK_N	1	1	1	1	1	0	1	1	1	1

TESTI:

	Clock	0	1	2	3	4	5	6	7	8	9
Input	STEP_EN	0	0	1	0	0	0	0	0	0	0
Input	Reset	1	0	0	0	0	0	0	0	0	0
Input	Busy	0	0	0	1	1	0	0	0	0	0
Output	STATE Ctrl	0	0	1	1	1	1	2	7	8	0
Output	IN_INIT	1	1	0	0	0	0	0	0	0	1
Output	S1sel	00	00	00	00	00	00	00	01	00	0
Output	S2sel	00	00	00	00	00	00	11	01	00	0
Output	Asel	0	0	0	0	0	0	0	0	0	0
Output	DINTsel	0	0	0	0	0	0	0	0	0	0
Output	MDRsel	0	0	0	0	0	0	0	0	0	0
Output	IRce	0	0	1	1	1	1	0	0	0	0
Output	PCce	0	0	0	0	0	0	1	0	0	0
Output	Ace	0	0	0	0	0	0	1	0	0	0
Output	Bce	0	0	0	0	0	0	1	0	0	0
Output	Cce	0	0	0	0	0	0	0	1	0	0
Output	MARce	0	0	0	0	0	0	0	0	0	0
Output	MDRce	0	0	0	0	0	0	0	0	0	0
Output	GPR_WE	0	0	0	1	0	0	0	0	1	0
Output	Itype	0	0	0	0	0	0	0	1	1	0
Output	Jlink	0	0	0	0	0	0	0	0	0	0
Output	test	0	0	0	0	0	0	0	1	0	0
Output	add	0	0	0	0	0	0	1	0	0	0
Output	shift	0	0	0	0	0	0	0	0	0	0
Output	right	0	0	0	0	0	0	0	0	0	0
Output	MR	0	0	1	1	1	1	0	0	0	0
Output	MW	0	0	0	0	0	0	0	0	0	0
Output	STATE MAC	00	00	00	01	01	10	00	00	00	0
Input(MAC)	ACK_N	1	1	1	1	1	0	1	1	1	1

LOAD:

	Clock	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Input	STEP_EN	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Input	Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Input	Busy	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0
Output	STATE Ctrl	0	0	1	1	1	1	2	9	a	a	a	a	b	8	0
Output	IN_INIT	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
Output	S1sel	00	00	00	00	00	00	00	01	00	00	00	00	11	00	00
Output	S2sel	00	00	00	00	00	00	11	01	00	00	00	00	10	00	00
Output	Asel	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Output	DINTsel	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
Output	MDRsel	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0
Output	IRce	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
Output	PCce	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Output	Ace	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Output	Bce	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Output	Cce	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Output	MARce	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Output	MDRce	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0
Output	GPR_WE	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0
Output	Itype	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Output	Jlink	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Output	test	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Output	add	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
Output	shift	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Output	right	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Output	MR	0	0	1	1	1	1	0	0	1	1	1	1	0	0	0
Output	MW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Output	STATE MAC	0	0	0	1	1	2	0	0	0	1	1	2	0	0	0
Input(MAC)	ACK_N	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1

STORE:

	Clock	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Input	STEP_EN	0	0	1	0	0	0	0	0	0	0	0	0	0	0
Input	Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Input	Busy	0	0	0	1	1	0	0	0	0	0	1	1	0	0
Output	STATE Ctrl	0	0	1	1	1	1	2	9	c	d	d	d	d	0
Output	IN_INIT	1	1	0	0	0	0	0	0	0	0	0	0	0	1
Output	S1sel	00	00	00	00	00	00	00	01	10	00	00	00	00	00
Output	S2sel	00	00	00	00	00	00	11	01	10	00	00	00	00	00
Output	Asel	0	0	0	0	0	0	0	0	00	1	1	1	1	0
Output	DINTsel	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Output	MDRsel	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Output	IRce	0	0	1	1	1	1	0	0	0	0	0	0	0	0
Output	PCce	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Output	Ace	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Output	Bce	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Output	Cce	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Output	MARce	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Output	MDRce	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Output	GPR_WE	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Output	Itype	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Output	Jlink	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Output	test	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Output	add	0	0	0	0	0	0	1	1	0	0	0	0	0	0
Output	shift	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Output	right	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Output	MR	0	0	1	1	1	1	0	0	0	0	0	0	0	0
Output	MW	0	0	0	0	0	0	0	0	0	1	1	1	1	0
Output	STATE MAC	0	0	0	1	1	2	0	0	0	0	1	1	2	0
Input(MAC)	ACK_N	1	1	1	1	1	0	1	1	1	1	1	1	0	1

JALR:

	Clock	0	1	2	3	4	5	6	7	8	9
Input	STEP_EN	0	0	1	0	0	0	0	0	0	0
Input	Reset	1	0	0	0	0	0	0	0	0	0
Input	Busy	0	0	0	1	1	0	0	0	0	0
Output	STATE Ctrl	0	0	1	1	1	1	2	f	10	0
Output	IN_INIT	1	1	0	0	0	0	0	0	0	1
Output	S1sel	00	00	00	00	00	00	00	00	01	0
Output	S2sel	00	00	00	00	00	00	11	10	10	0
Output	Asel	0	0	0	0	0	0	0	0	0	0
Output	DINTsel	0	0	0	0	0	0	0	0	0	0
Output	MDRsel	0	0	0	0	0	0	0	0	0	0
Output	IRce	0	0	1	1	1	1	0	0	0	0
Output	PCce	0	0	0	0	0	0	1	0	1	0
Output	Ace	0	0	0	0	0	0	1	0	0	0
Output	Bce	0	0	0	0	0	0	1	0	0	0
Output	Cce	0	0	0	0	0	0	0	1	0	0
Output	MARce	0	0	0	0	0	0	0	0	0	0
Output	MDRce	0	0	0	0	0	0	0	0	0	0
Output	GPR_WE	0	0	0	1	0	0	0	0	1	0
Output	Itype	0	0	0	0	0	0	0	0	0	0
Output	Jlink	0	0	0	0	0	0	0	0	1	0
Output	test	0	0	0	0	0	0	0	0	0	0
Output	add	0	0	0	0	0	0	1	1	1	0
Output	shift	0	0	0	0	0	0	0	0	0	0
Output	right	0	0	0	0	0	0	0	0	0	0
Output	MR	0	0	1	1	1	1	0	0	0	0
Output	MW	0	0	0	0	0	0	0	0	0	0
Output	STATE MAC	00	00	00	01	01	10	00	00	00	0
Input(MAC)	ACK_N	1	1	1	1	1	0	1	1	1	1

BTAKEN(both for taking the branch and skipping it):

	Clock	0	1	2	3	4	5	6	7	8a(bt)	9a(bt)	8b(~bt)
Input	STEP_EN	0	0	1	0	0	0	0	0	0	0	0
Input	Reset	1	0	0	0	0	0	0	0	0	0	0
Input	Busy	0	0	0	1	1	0	0	0	1	1	0
Output	STATE Ctrl	0	0	1	1	1	1	2	11	12	0	0
Output	IN_INIT	1	1	0	0	0	0	0	0	1	1	1
Output	S1sel	00	00	00	00	00	00	00	00	00	00	00
Output	S2sel	00	00	00	00	00	00	11	00	01	00	00
Output	Asel	0	0	0	0	0	0	0	0	0	0	0
Output	DINTsel	0	0	0	0	0	0	0	0	0	0	0
Output	MDRsel	0	0	0	0	0	0	0	0	0	0	0
Output	IRce	0	0	1	1	1	1	0	0	0	0	0
Output	PCce	0	0	0	0	0	0	1	0	1	0	0
Output	Ace	0	0	0	0	0	0	1	0	0	0	0
Output	Bce	0	0	0	0	0	0	1	0	0	0	0
Output	Cce	0	0	0	0	0	0	0	0	0	0	0
Output	MARce	0	0	0	0	0	0	0	0	0	0	0
Output	MDRce	0	0	0	0	0	0	0	0	0	0	0
Output	GPR_WE	0	0	0	1	0	0	0	0	0	0	0
Output	Itype	0	0	0	0	0	0	0	0	0	0	0

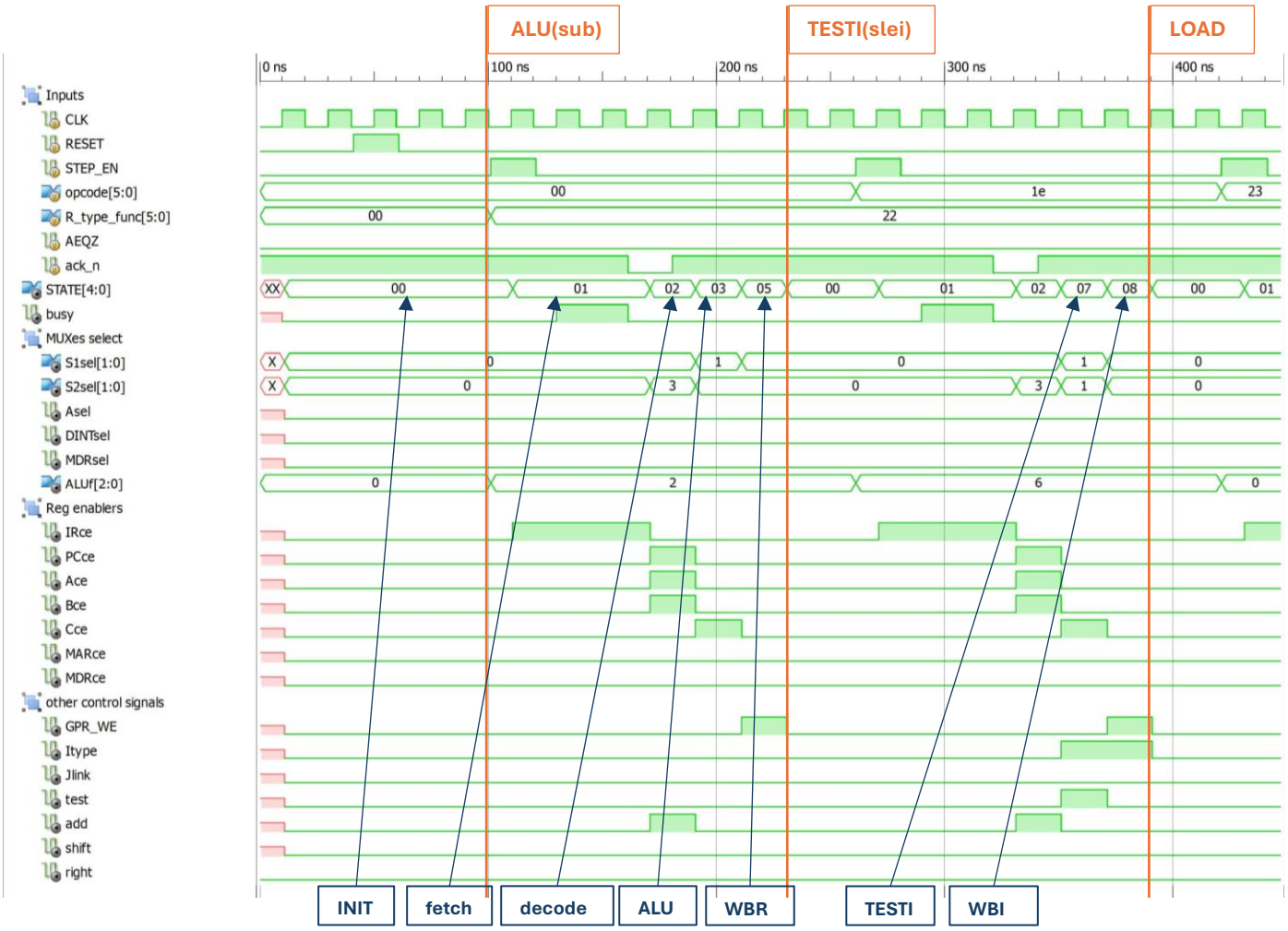
Output	Jlink	0	0	0	0	0	0	0	0	0	0	0
Output	test	0	0	0	0	0	0	0	0	0	0	0
Output	add	0	0	0	0	0	0	1	0	1	0	0
Output	shift	0	0	0	0	0	0	0	0	0	0	0
Output	right	0	0	0	0	0	0	0	0	0	0	0
Output	MR	0	0	1	1	1	1	0	0	0	0	0
Output	MW	0	0	0	0	0	0	0	0	0	0	0
Output	STATE MAC	0	0	0	1	1	2	0	0	0	0	0
Input(MAC)	ACK_N	1	1	1	1	1	0	1	1	1	1	1

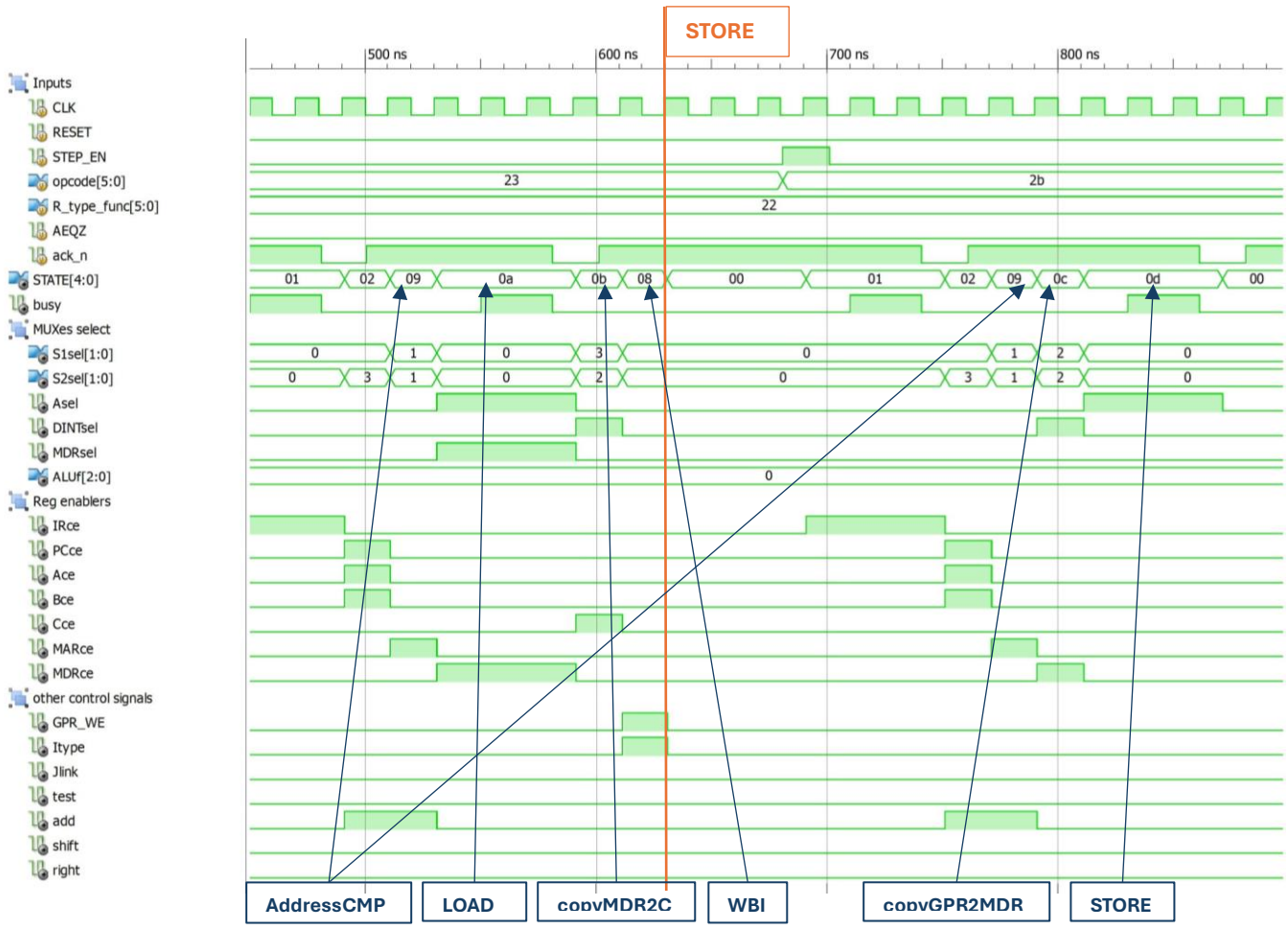
Control test-bench

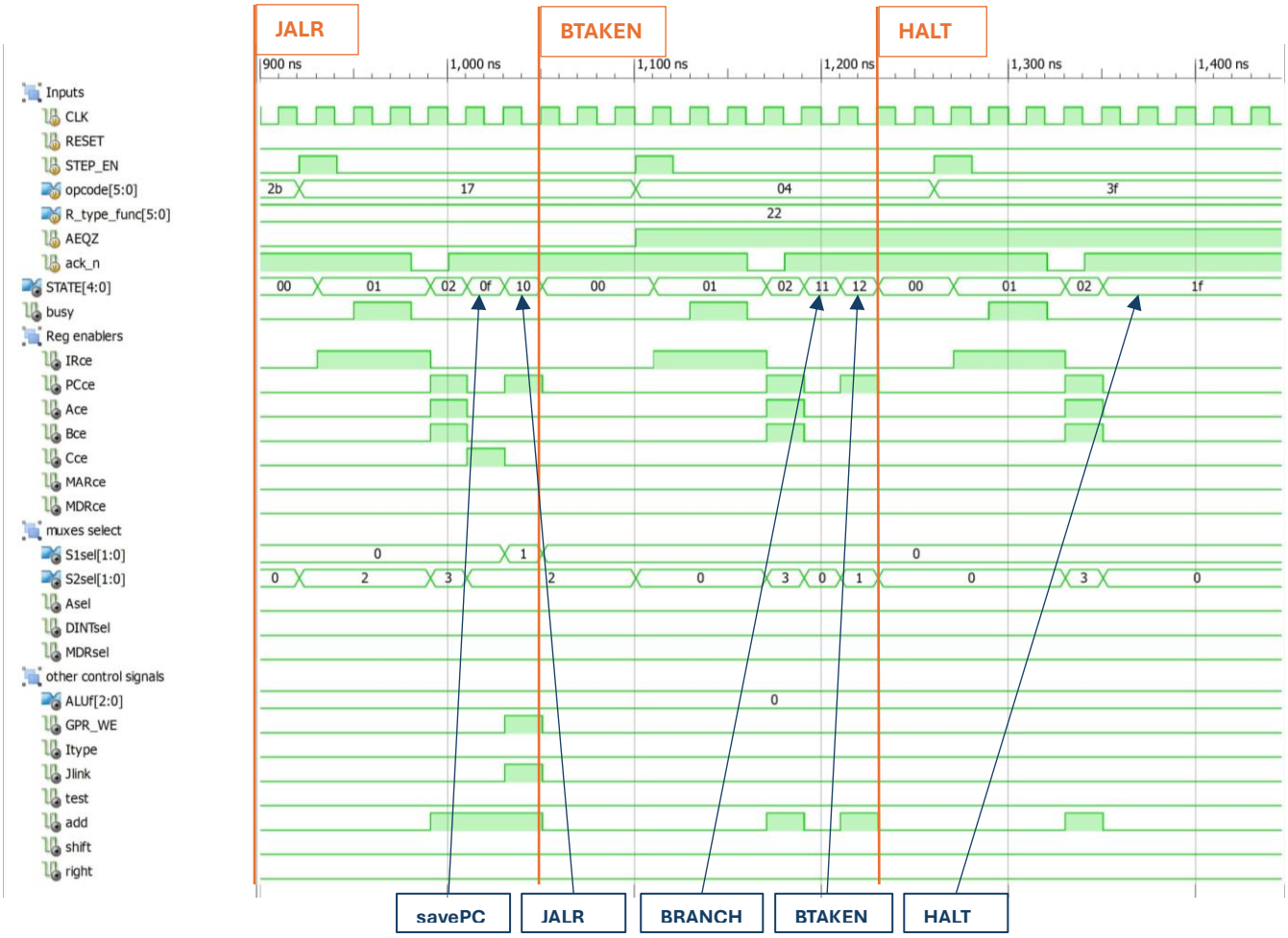
After designing the control unit, we used the next test-bench code to validate its functionality :

```
`timescale 1ns / 1ps

module control_control_sch_tb();

// Inputs
   reg CLK;
   reg RESET;
   reg STEP_EN;
   reg [5:0] opcode;
   reg [5:0] R_type_func;
   reg ack_n;
   reg AEQZ;

// Output
   wire IRce;
   wire PCce;
   wire Ace;
   wire Bce;
   wire Cce;
   wire MARce;
   wire MDR;
   wire DINTsel;
   wire MDRsel;
   wire Asel;
   wire shift;
   wire right;
   wire add;
   wire test;
   wire MR;
   wire busy;
   wire GPR_WE;
   wire [2:0] ALUf;
   wire Jlink;
   wire Itype;
   wire [1:0] S1sel;
   wire [1:0] S2sel;
   wire [1:0] MAC_state;
   wire wr_N;
   wire MW;
   wire as_N;
   wire STOP_N;
   wire [4:0] STATE;

// Bidirs

// Instantiate the UUT
   control UUT (
      .CLK(CLK),
      .IRce(IRce),
      .PCce(PCce),
      .Ace(Ace),
      .Bce(Bce),
      .Cce(Cce),
      .RESET(RESET),
      .MARce(MARce),
      .MDR(MDR),
      .DINTsel(DINTsel),
      .MDRsel(MDRsel),
      .Asel(Asel),
      .shift(shift),
      .right(right),
      .add(add),
      .test(test),
      .MR(MR),
      .busy(busy),
      .STEP_EN(STEP_EN),
      .GPR_WE(GPR_WE),
      .opcode(opcode),
      .R_type_func(R_type_func),
      .ALUf(ALUf),
      .Jlink(Jlink),
      .Itype(Itype),
      .S1sel(S1sel),
      .S2sel(S2sel),
      .MAC_state(MAC_state),
      .wr_N(wr_N),
      .ack_n(ack_n),
      .MW(MW),
      .as_N(as_N),
      .STOP_N(STOP_N),
      .STATE(STATE),
      .AEQZ(AEQZ)

   );

      initial
      CLK = 0;
      always #10 CLK = ~CLK;

// Initialize Inputs
      initial begin
      STEP_EN = 0;
      CLK = 0;
      RESET = 0;
      opcode = 6'b000000;
      ack_n = 1;
      R_type_func = 6'b000000;
      AEQZ = 0;


   //wating 2 clock cycles for reset
   #41;
   RESET = 1;
   #20;
   RESET = 0;
   #40;

// start ALU(sub) operation:
   opcode = 6'b000000;
   R_type_func = 6'b100010;
   STEP_EN = 1;
   #20;
   STEP_EN = 0;
   #40;
   ack_n = 0;
   #20;
   ack_n = 1;
   #80;

// start TESTI(slei) operation:
   opcode = 6'b011110;
   STEP_EN = 1;
   #20;
   STEP_EN = 0;
   #40;
   ack_n = 0;
   #20;
   ack_n = 1;
   #80;

// start load operation:
   opcode = 6'b100011;
   STEP_EN = 1;
   #20;
   STEP_EN = 0;
   #40;
   ack_n = 0;
   #20;
   ack_n = 1;
   #80;
   ack_n = 0;
   #20;
   ack_n = 1;
   #80;


// start store operation:
   opcode = 6'b101011;
   STEP_EN = 1;
   #20;
   STEP_EN = 0;
   #40;
   ack_n = 0;
   #20;
   ack_n = 1;
   #100;
   ack_n = 0;
   #20;
   ack_n = 1;
   #40;

// start JALR operation:
   opcode = 6'b010111;
   STEP_EN = 1;
   #20;
   STEP_EN = 0;
   #40;
   ack_n = 0;
   #20;
   ack_n = 1;
   #100;

// start BTAKEN(beqz) operation:
   opcode = 6'b000100;
   AEQZ = 1'b1;
   STEP_EN = 1;
   #20;
   STEP_EN = 0;
   #40;
   ack_n = 0;
   #20;
   ack_n = 1;
   #80;

// start halt operatoin
   opcode = 6'b111111;
   STEP_EN = 1;
   #20;
   STEP_EN = 0;
   #40;
   ack_n = 0;
   #20;
   ack_n = 1;
   #80;

   end
endmodule
```

After running the code and simulating the control, we received the following waveforms, which matching the test vectors:

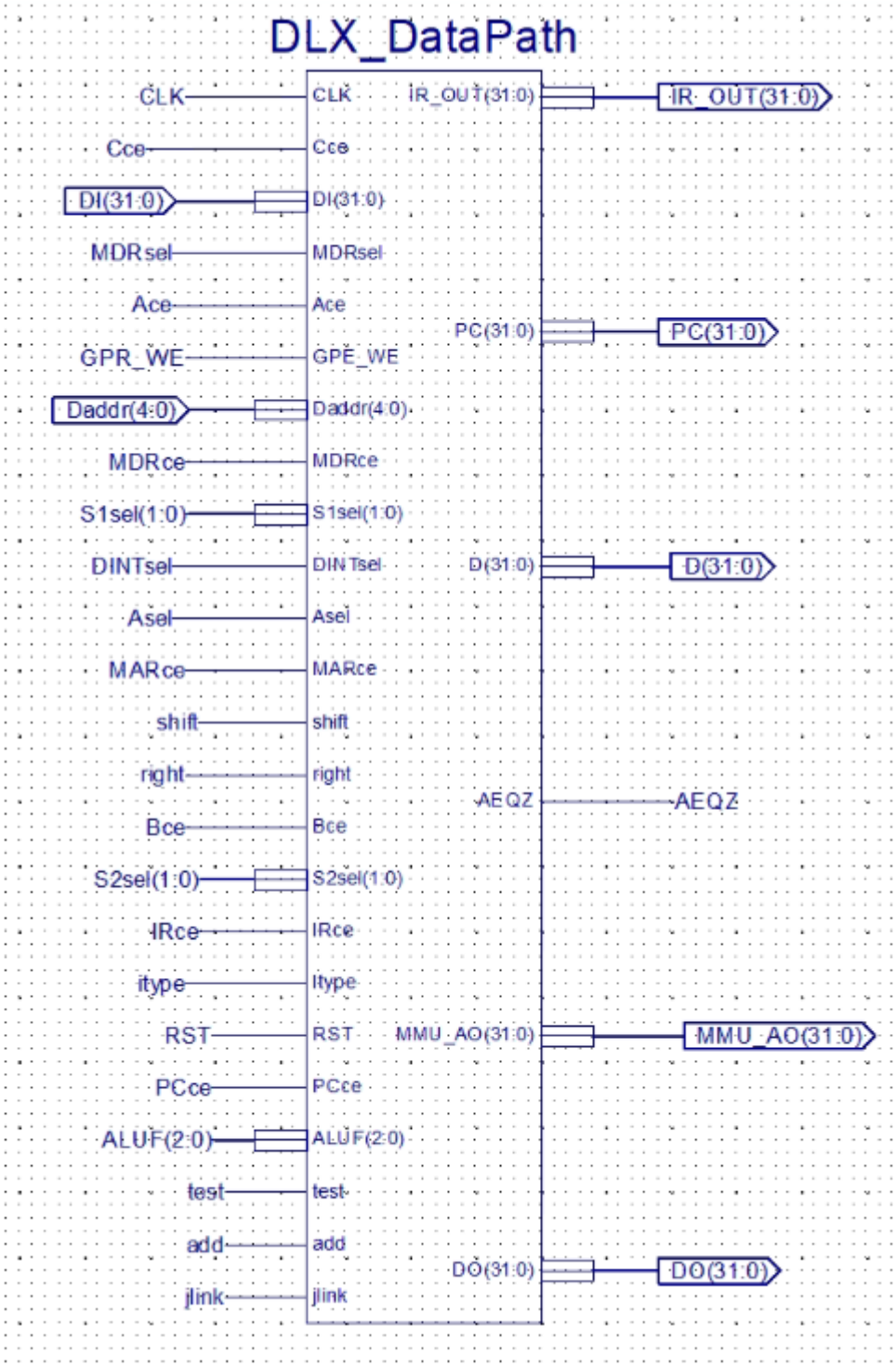



STORE
500 ns
600 ns
700 ns
800 ns
Inputs
CLK
RESET
STEP_EN
opcode[5:0]
R_type_func[5:0]
AEQZ
ack_n
STATE[4:0]
busy
MUXes select
S1sel[1:0]
S2sel[1:0]
Asel
DINTsel
MDRsel
ALUf[2:0]
Reg enablers
IRce
PCce
Ace
Bce
Cce
MARce
MDRce
other control signals
GPR_WE
Itype
Jlink
test
add
shift
right
AddressCMP
LOAD
copyMDR2C
WBI
copyGPR2MDR
STORE


JALR
BTAKEN
HALT
900 ns
1,000 ns
1,100 ns
1,200 ns
1,300 ns
1,400 ns
Inputs
CLK
RESET
STEP_EN
opcode[5:0]
R_type_func[5:0]
AEQZ
ack_n
STATE[4:0]
busy
Reg enablers
IRce
PCce
Ace
Bce
Cce
MARce
MDRce
muxes select
S1sel[1:0]
S2sel[1:0]
Asel
DINTsel
MDRsel
other control signals
ALUf[2:0]
GPR_WE
Itype
Jlink
test
add
shift
right
savePC
JALR
BRANCH
BTAKEN
HALT

The Datapath

Datapath schematics:

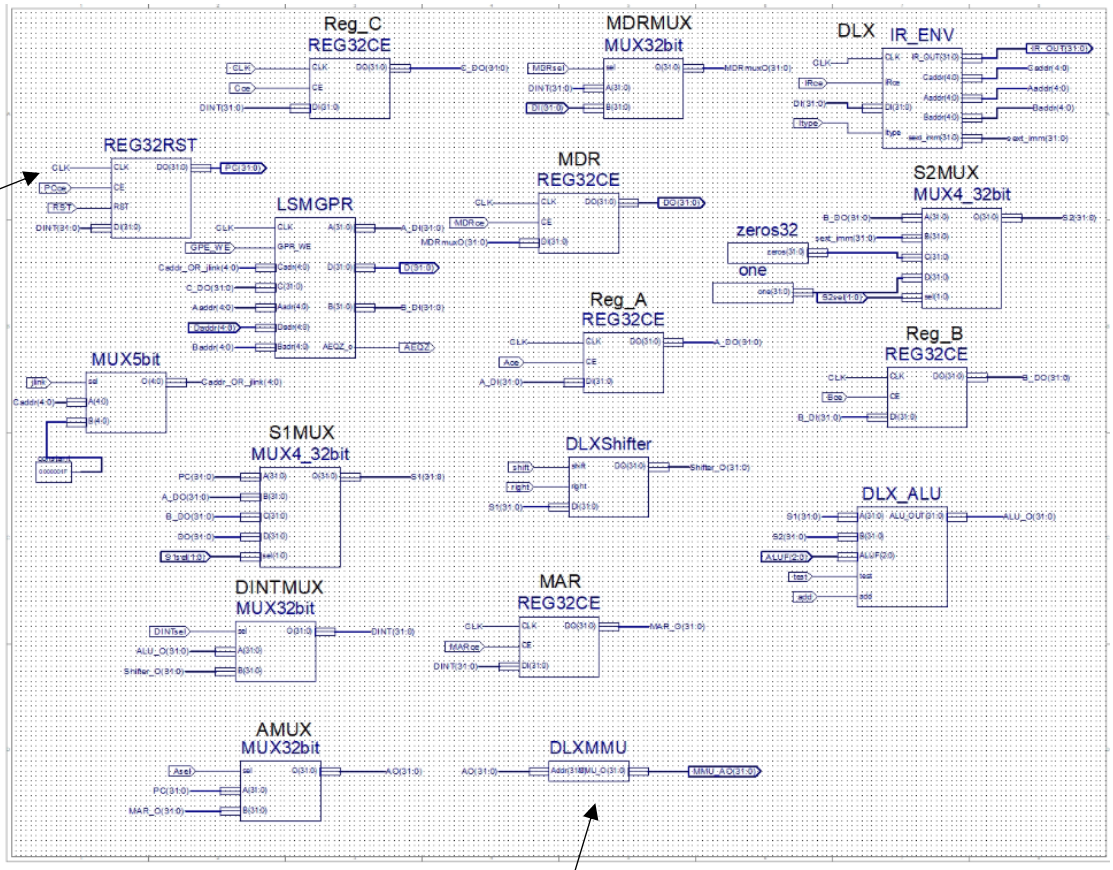


The datapath contains the following units:

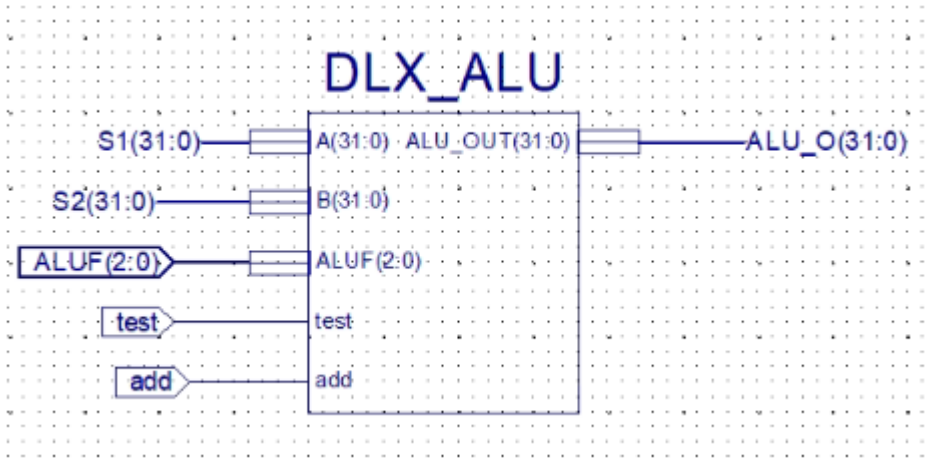
Registers: A, B, C, PC (register with reset), MDR, MAR

Muxes: MDR, DINT, S1, S2, A, jlink mux (chooses between Caddr to R31 in case of jlink).

Shifter, GPR (from lab 6), ALU, MMU, IR environment.

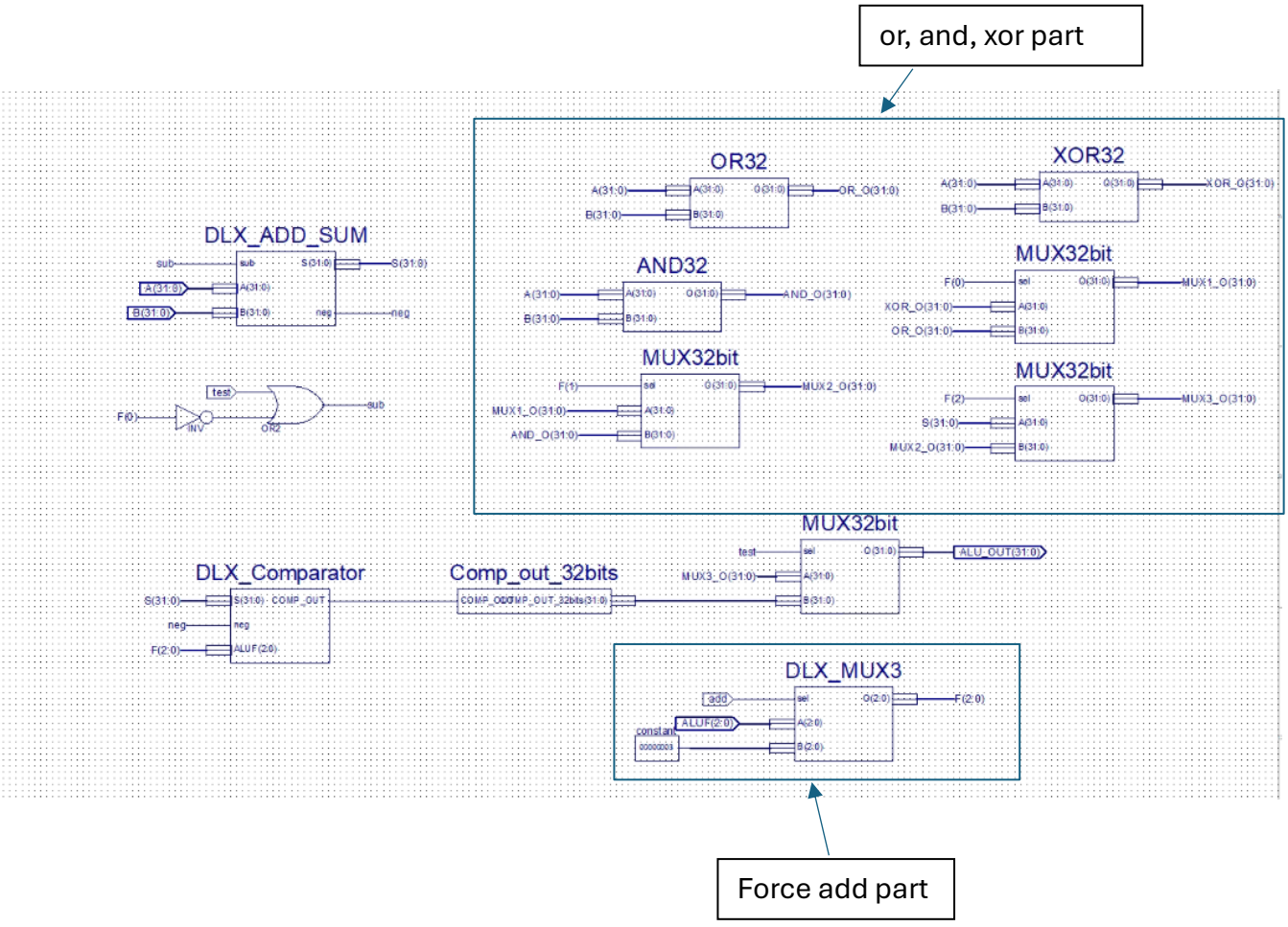


The ALU:



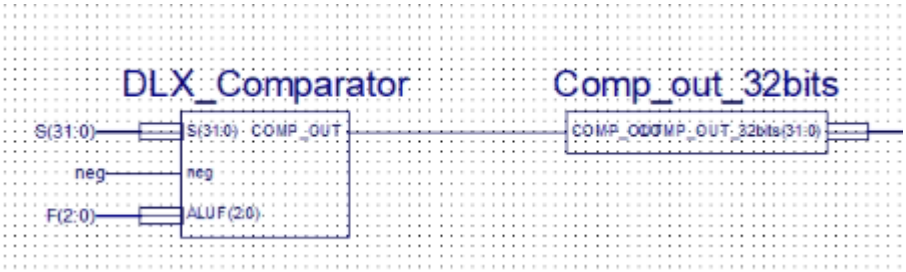
The ALU responsible of arithmetic and comparation operation

Inside the ALU:

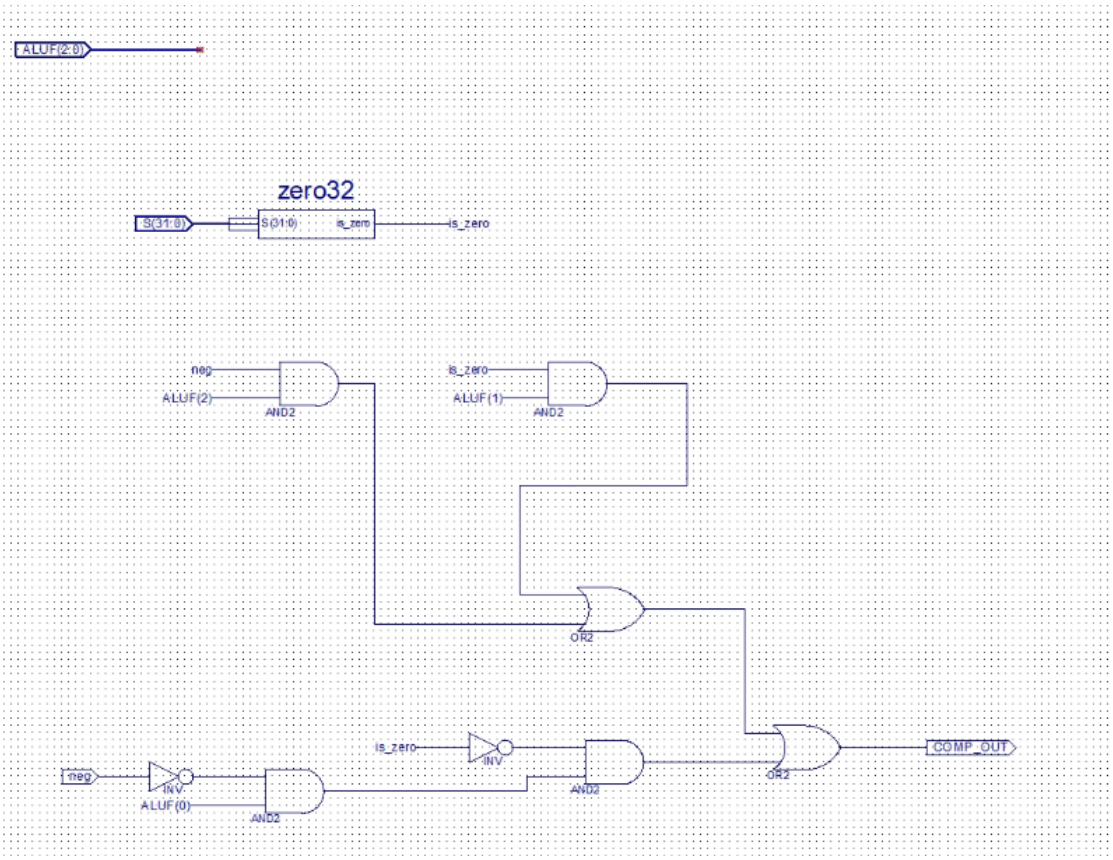


The comparator:

inside the ALU we have a comparator unit which responsible of the comparation operation

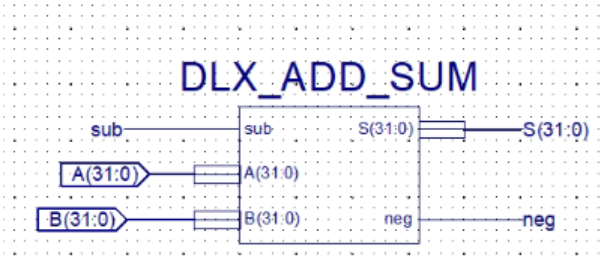


inside the comparator:

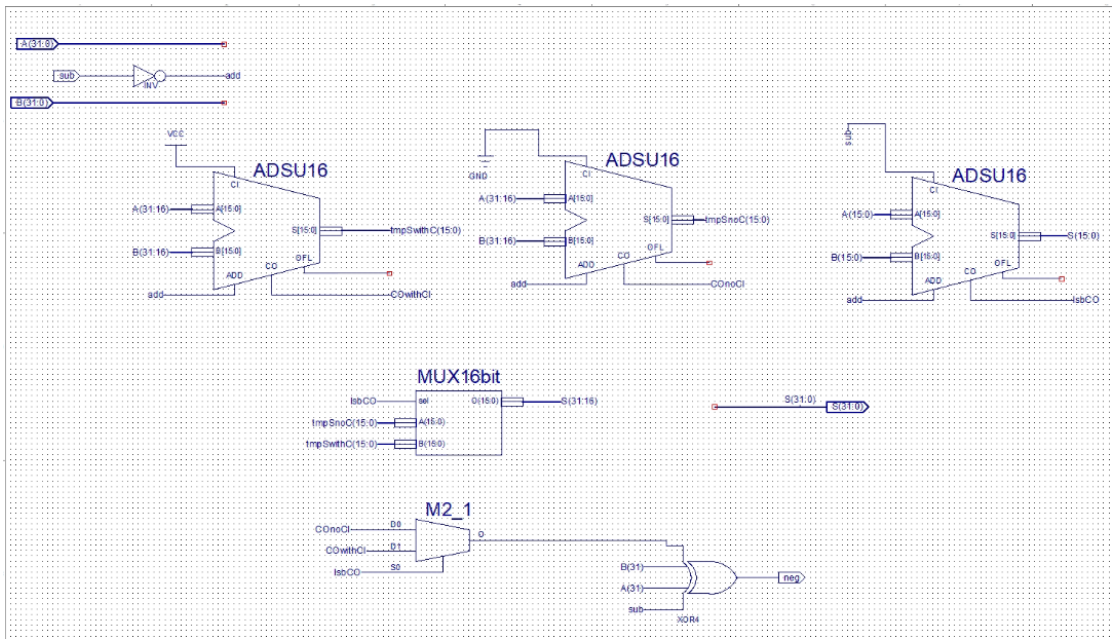


The ADD-SUB unit:

inside the ALU we have ADD-SUB unit which responsible for add and subtract operations

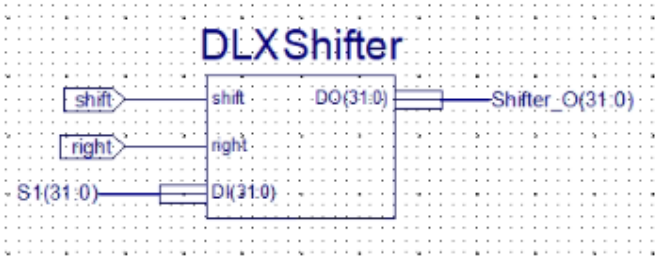


inside the ADD-SUB:



The shifter:

the shifter responsible of shifting operations

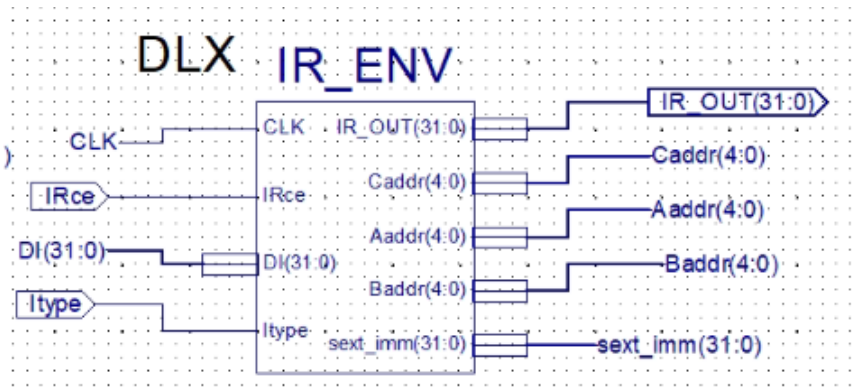


Inside the shifter:

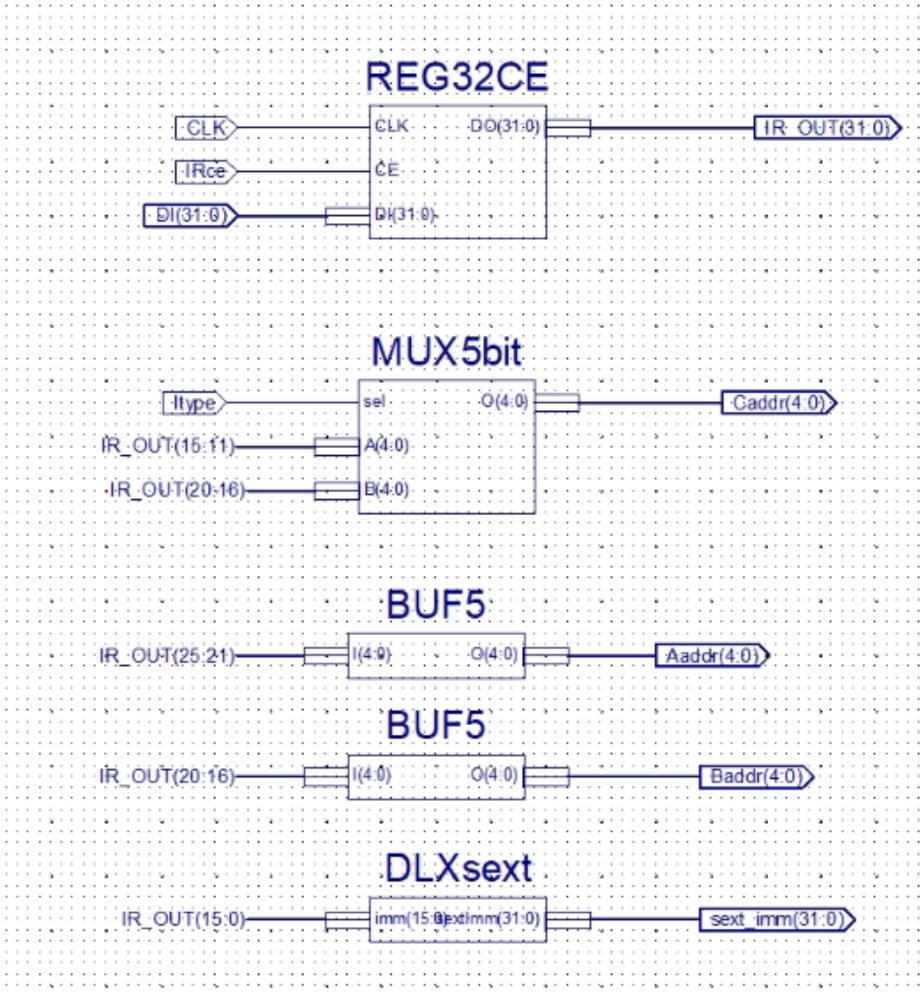
```
21  module DLXShifter(
22      input [31:0] DI,
23      input shift,
24      input right,
25      output [31:0] DO
26      );
27
28
29
30  assign DO = (shift && right) ? (DI>>1) : (shift) ? (DI<<1) : DI;
31
32  endmodule
33
```

The IR environment:

The IR environment responsible for breaking the instruction into it's RD RS1 RS2 IMM parts depends on the type of it.



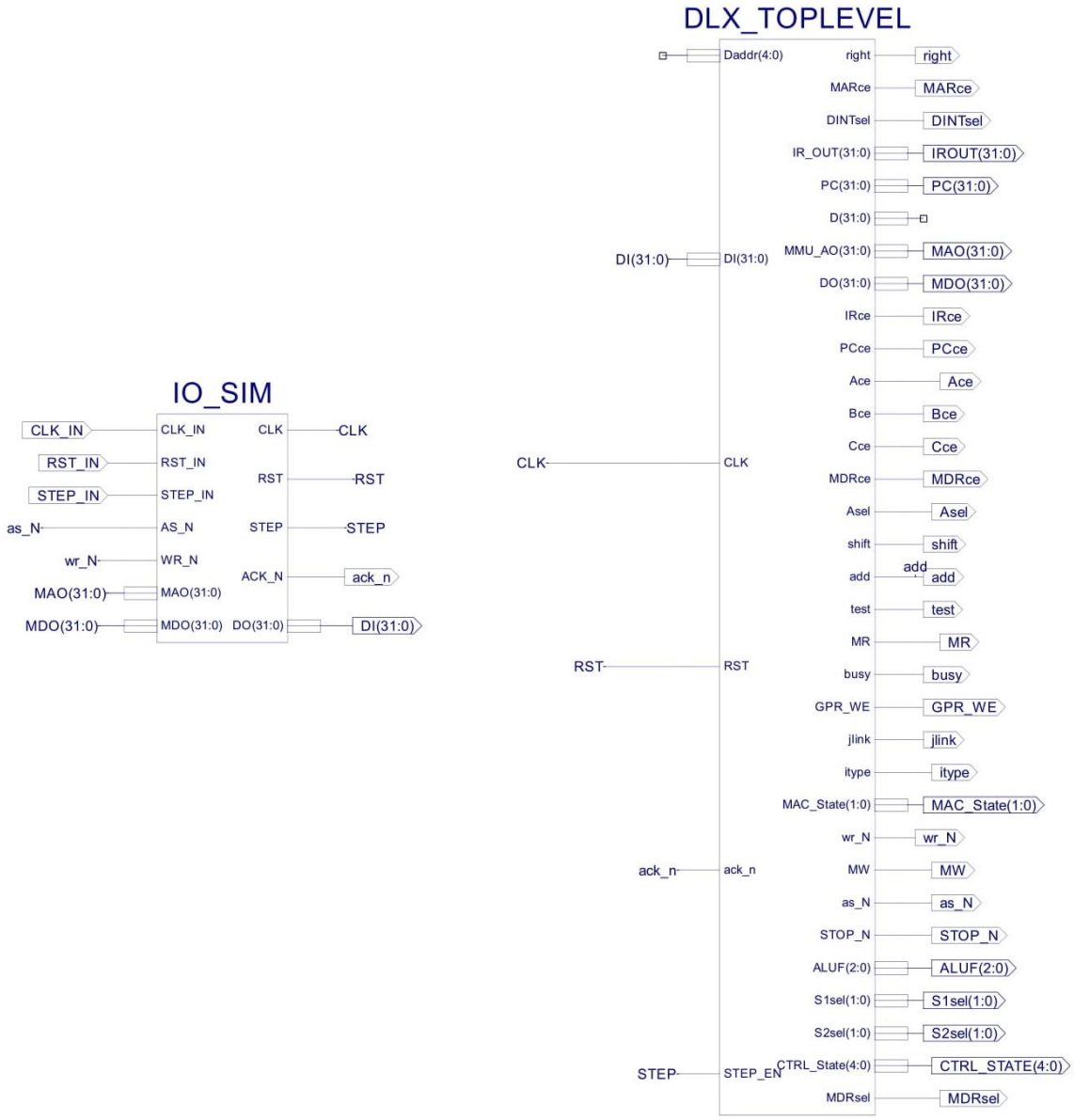
Inside the IR environment:



DLX Simulation and The IOsimul

Scheme of the connections to the DLX:

In order to simulate the entire DLX processor functionality and behavior, after finishing the design we connected the DLX to the IOsimul unit the way it shown in the scheme below.



After connecting the two systems, we built a test-bench and simulated the execution of DLX program and we received its waveforms, which all attached in the next pages.

DLX program:

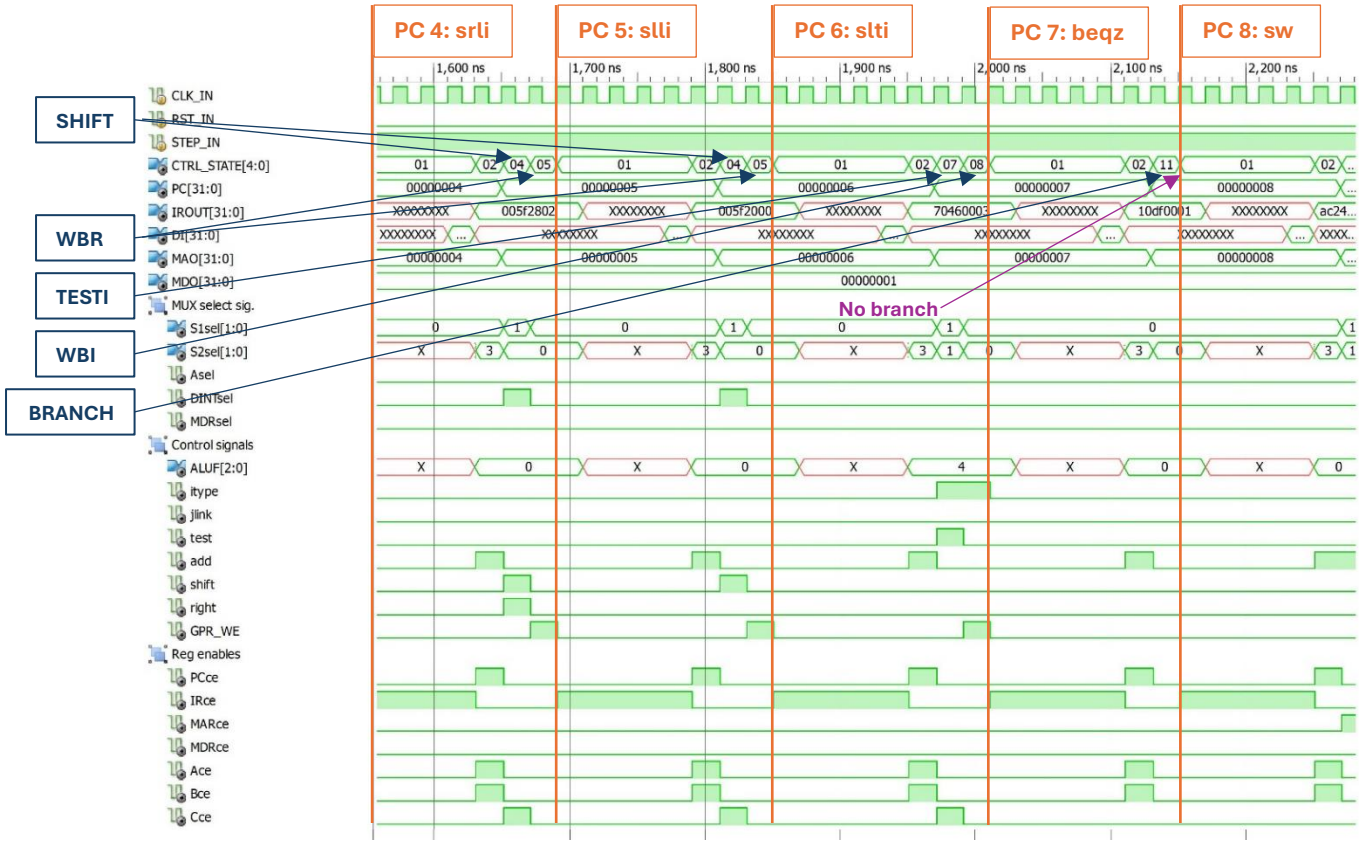
The code that was loaded into the IOsimul, as it was given by the course stuff, can be seen below, with the traversed states for each instruction detailed right to it:

```
pc=0x0
lw R1 R0 data1      INIT → fetch → decode → AddressCMP → LOAD → copyMDR2C → WBI → INIT/fetch
sw   R1 R0 adr2     INIT → fetch → decode → AddressCMP → copyGPR2MDR → STORE → INIT/fetch
addi R2 R1 0x1      INIT → fetch → decode → ALUI → WBI → INIT/fetch
add  R3 R1 R2       INIT → fetch → decode → ALU → WBR → INIT/fetch
srli R5 R2          INIT → fetch → decode → SHIFT → WBR → INIT/fetch
slli R4 R2          INIT → fetch → decode → SHIFT → WBR → INIT/fetch
slti R6 R2 3        INIT → fetch → decode → TESTI → WBI → INIT/fetch
beqz R6 jump1       INIT → fetch → decode → BRANCH (no branch) → INIT/fetch
sw   R4 R1 adr1     INIT → fetch → decode → AddressCMP → copyGPR2MDR → STORE → INIT/fetch
jump1: lw   R7 R1 adr1   INIT → fetch → decode → AddressCMP → LOAD → copyMDR2C → WBI → INIT/fetch
sgei R8 R4 0        INIT → fetch → decode → TESTI → WBI → INIT/fetch
bnez R8 jump2       INIT → fetch → decode → BRANCH → Btaken → INIT/fetch
or   R4 R2 R4       INIT → fetch → decode → ALU → WBR → INIT/fetch
jump2: special-nop  INIT → fetch → decode → INIT/fetch
addi R3 R0 jump3    INIT → fetch → decode → ALUI → WBI → INIT/fetch
jalr R3             INIT → fetch → decode → savePC → JALR → INIT/fetch
sub  R3 R5 R1       INIT → fetch → decode → ALU → WBR → INIT/fetch
beqz R3 jump4       INIT → fetch → decode → BRANCH → Btaken → INIT/fetch
jump3: jr   R31     INIT → fetch → decode → JR → INIT/fetch
jump4: seqi R3 R0 -1     INIT → fetch → decode → TESTI → WBI → INIT/fetch
lw   R2 R0 data2    INIT → fetch → decode → AddressCMP → LOAD → copyMDR2C → WBI → INIT/fetch
snei R5 R2 0        INIT → fetch → decode → TESTI → WBI → INIT/fetch
slei R6 R4 6        INIT → fetch → decode → TESTI → WBI → INIT/fetch
xor R6 R4 R0        INIT → fetch → decode → ALU → WBR → INIT/fetch
or R2 R6 R4         INIT → fetch → decode → ALU → WBR → INIT/fetch
halt                INIT → fetch → HALT
data1: dc 0x1
data2: dc 0xEFFFFFFF
adr1:  dc 0x0
adr2:  dc 0x0
```

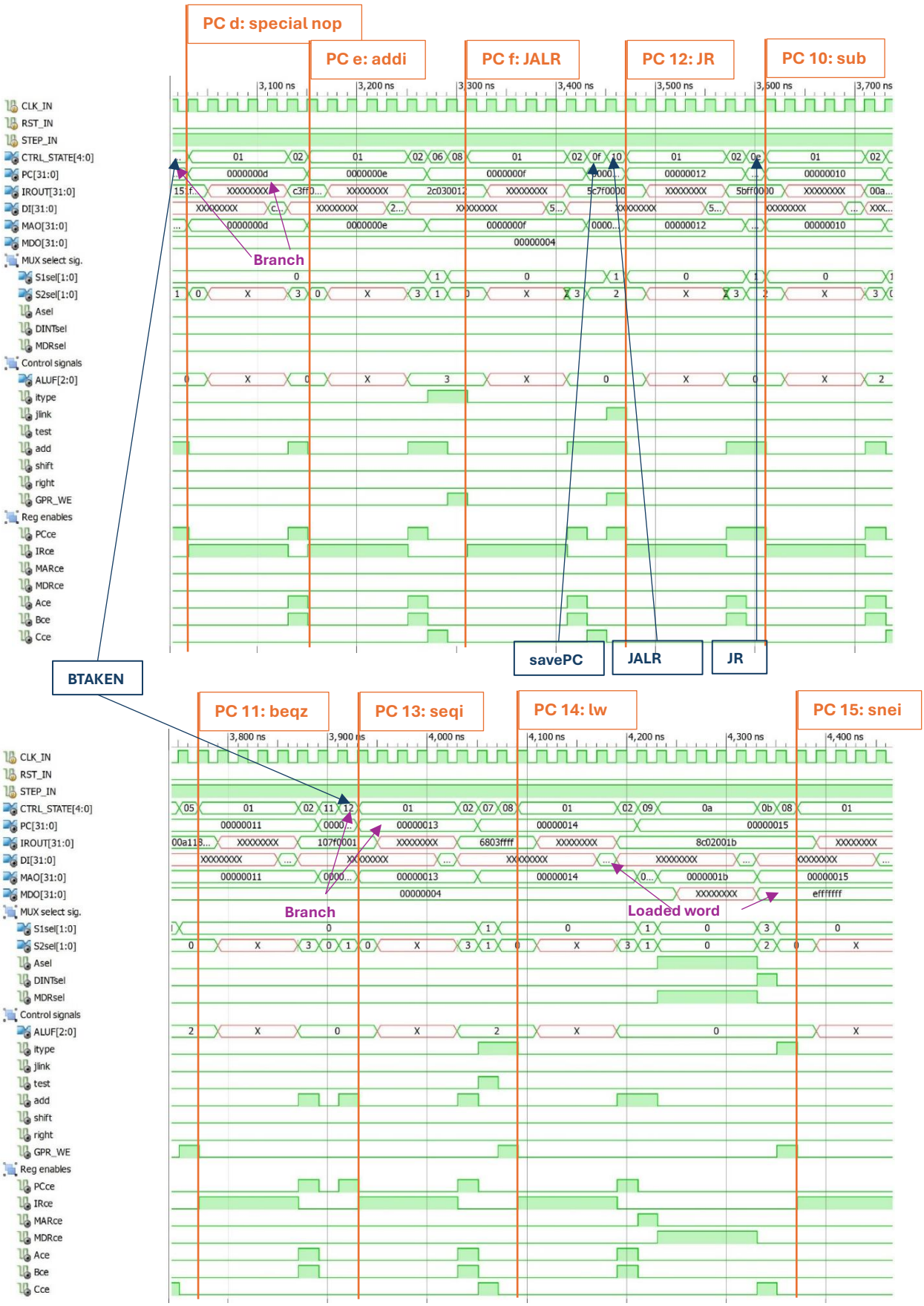
DLX test-bench with the IOsimul:

```
`timescale 1ns / 1ps

module DLX_IOSIM_DLX_IOSIM_sch_tb();

// Inputs
   reg CLK_IN;
   reg RST_IN;
   reg STEP_IN;

// Output
   wire right;
   wire MARce;
   wire DINTsel;
   wire [31:0] IROUT;
   wire [31:0] PC;
   wire [31:0] MDO;
   wire IRce;
   wire PCce;
   wire Ace;
   wire Bce;
   wire Cce;
   wire MDRce;
   wire Asel;
   wire shift;
   wire add;
   wire test;
   wire MR;
   wire busy;
   wire GPR_WE;
   wire jlink;
   wire itype;
   wire [1:0] MAC_State;
   wire MW;
   wire STOP_N;
   wire [2:0] ALUF;
   wire [1:0] S1sel;
   wire [1:0] S2sel;
   wire [4:0] CTRL_STATE;
   wire MDRsel;
   wire as_N;
   wire wr_N;
   wire [31:0] MAO;
   wire ack_n;
   wire [31:0] DI;

// Bidirs

// Instantiate the UUT
   DLX_IOSIM UUT (
      .right(right),
      .MARce(MARce),
      .DINTsel(DINTsel),
      .IROUT(IROUT),
      .PC(PC),
      .MDO(MDO),
      .IRce(IRce),
      .PCce(PCce),
      .Ace(Ace),
      .Bce(Bce),
      .Cce(Cce),
      .MDRce(MDRce),
      .Asel(Asel),
      .shift(shift),
      .add(add),
      .test(test),
      .MR(MR),
      .busy(busy),
      .GPR_WE(GPR_WE),
      .jlink(jlink),
      .itype(itype),
      .MAC_State(MAC_State),
      .MW(MW),
      .STOP_N(STOP_N),
      .ALUF(ALUF),
      .S1sel(S1sel),
      .S2sel(S2sel),
      .CTRL_STATE(CTRL_STATE),
      .MDRsel(MDRsel),
      .CLK_IN(CLK_IN),
      .RST_IN(RST_IN),
      .STEP_IN(STEP_IN),
      .as_N(as_N),
      .wr_N(wr_N),
      .MAO(MAO),
      .ack_n(ack_n),
      .DI(DI)
   );

      initial
      CLK_IN = 0;
      always #10 CLK_IN = ~CLK_IN;

      // Initialize Inputs
      initial begin
      STEP_IN = 0;
      CLK_IN = 0;
      RST_IN = 0;
      #41;

      // System set
      RST_IN = 1;
      #100;
      RST_IN = 0;
      #60;

      // Reset test
      STEP_IN = 1;
      #300;
      STEP_IN = 0;
      #20;
      RST_IN = 1;
      #40;
      RST_IN = 0;
      #40;

      // Start Execution
      STEP_IN = 1;
      #400;
      STEP_IN = 0;
      #200;
      STEP_IN = 1;
      #20;
      STEP_IN = 0;
      #100;
      STEP_IN = 1;

      end

endmodule
```

Simulation waveform results:


PC 4: srli
PC 5: slli
PC 6: slti
PC 7: beqz
PC 8: sw
SHIFT
WBR
TESTI
WBI
BRANCH
No branch
CLK_IN
RST_IN
STEP_IN
CTRL_STATE[4:0]
PC[31:0]
IROUT[31:0]
DI[31:0]
MAO[31:0]
MDO[31:0]
MUX select sig.
S1sel[1:0]
S2sel[1:0]
Asel
DINTsel
MDRsel
Control signals
ALUF[2:0]
itype
jlink
test
add
shift
right
GPR_WE
Reg enables
PCce
IRce
MARce
MDRce
Ace
Bce
Cce


PC 9: lw
PC a: sgei
PC b: bnez
Saved word
Loaded word
CLK_IN
RST_IN
STEP_IN
CTRL_STATE[4:0]
PC[31:0]
IROUT[31:0]
DI[31:0]
MAO[31:0]
MDO[31:0]
MUX select sig.
S1sel[1:0]
S2sel[1:0]
Asel
DINTsel
MDRsel
Control signals
ALUF[2:0]
itype
jlink
test
add
shift
right
GPR_WE
Reg enables
PCce
IRce
MARce
MDRce
Ace
Bce
Cce


PC d: special nop
PC e: addi
PC f: JALR
PC 12: JR
PC 10: sub
CLK_IN
RST_IN
STEP_IN
CTRL_STATE[4:0]
PC[31:0]
IROUT[31:0]
DI[31:0]
MAO[31:0]
MDO[31:0]
MUX select sig.
S1sel[1:0]
S2sel[1:0]
Asel
DINTsel
MDRsel
Control signals
ALUF[2:0]
itype
jlink
test
add
shift
right
GPR_WE
Reg enables
PCce
IRce
MARce
MDRce
Ace
Bce
Cce
Branch
BTAKEN
savePC
JALR
JR
PC 11: beqz
PC 13: seqi
PC 14: lw
PC 15: snei
Branch
Loaded word

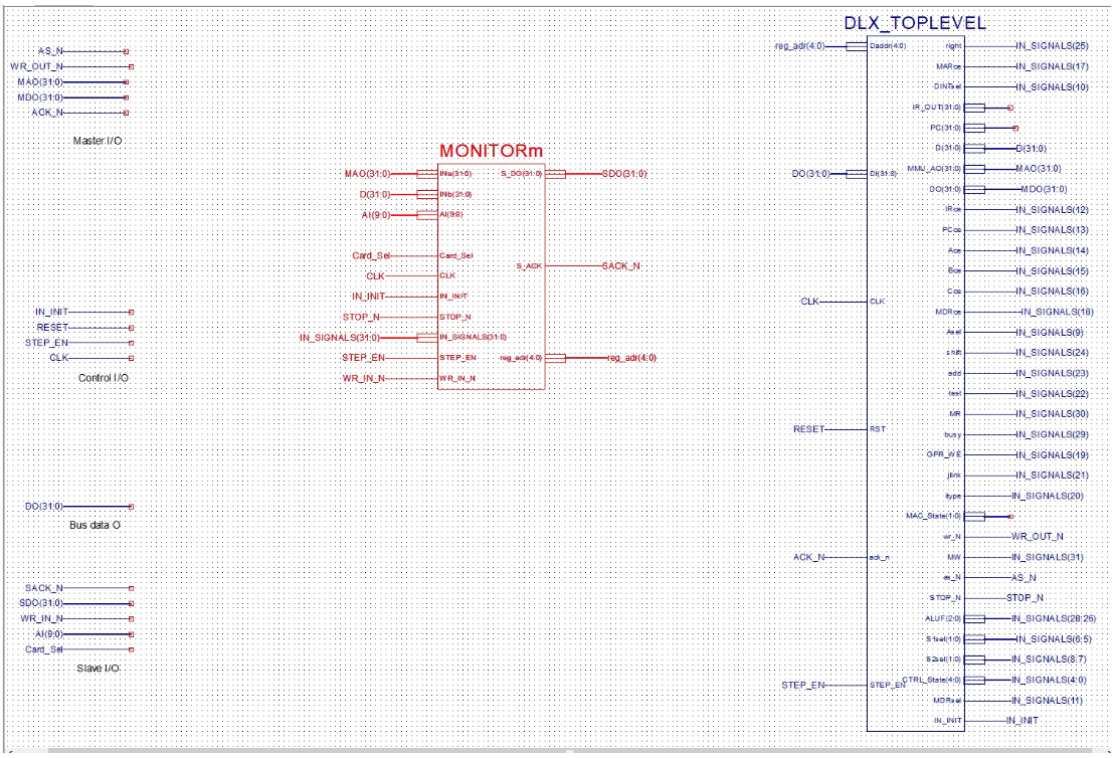

PC 16: slei
PC 16: xor
PC 18: or
PC 19: halt
4,500 ns
4,600 ns
4,700 ns
4,800 ns
4,900 ns
5,000 ns
5,100 ns
CLK_IN
RST_IN
STEP_IN
CTRL_STATE[4:0]
PC[31:0]
IROUT[31:0]
DI[31:0]
MAO[31:0]
MDO[31:0]
MUX select sig.
S1sel[1:0]
S2sel[1:0]
Asel
DINTsel
MDRsel
Control signals
ALUF[2:0]
itype
jlink
test
add
shift
right
GPR_WE
Reg enables
PCce
IRce
MARce
MDRce
Ace
Bce
Cce
HALT

7.2)

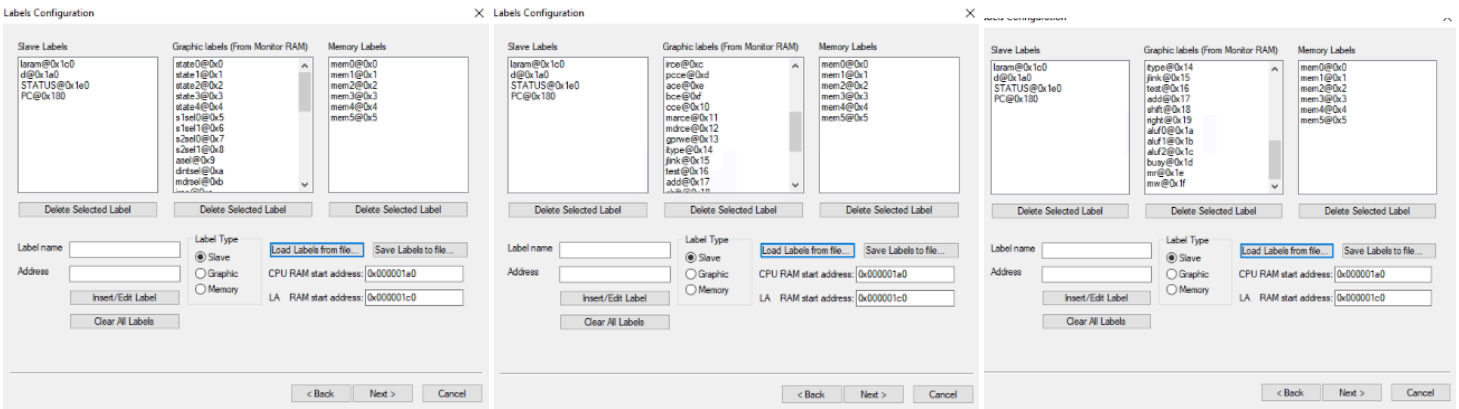
The RESA

After the simulation of the DLX machine with the i/o simul we connect the DLX to the i/o logic to upload it to the fpga and simulate it with the RESA program.
We used the same assembly program as we used for the i/o simul simulation.

DLX to i/o logic connection schematic:

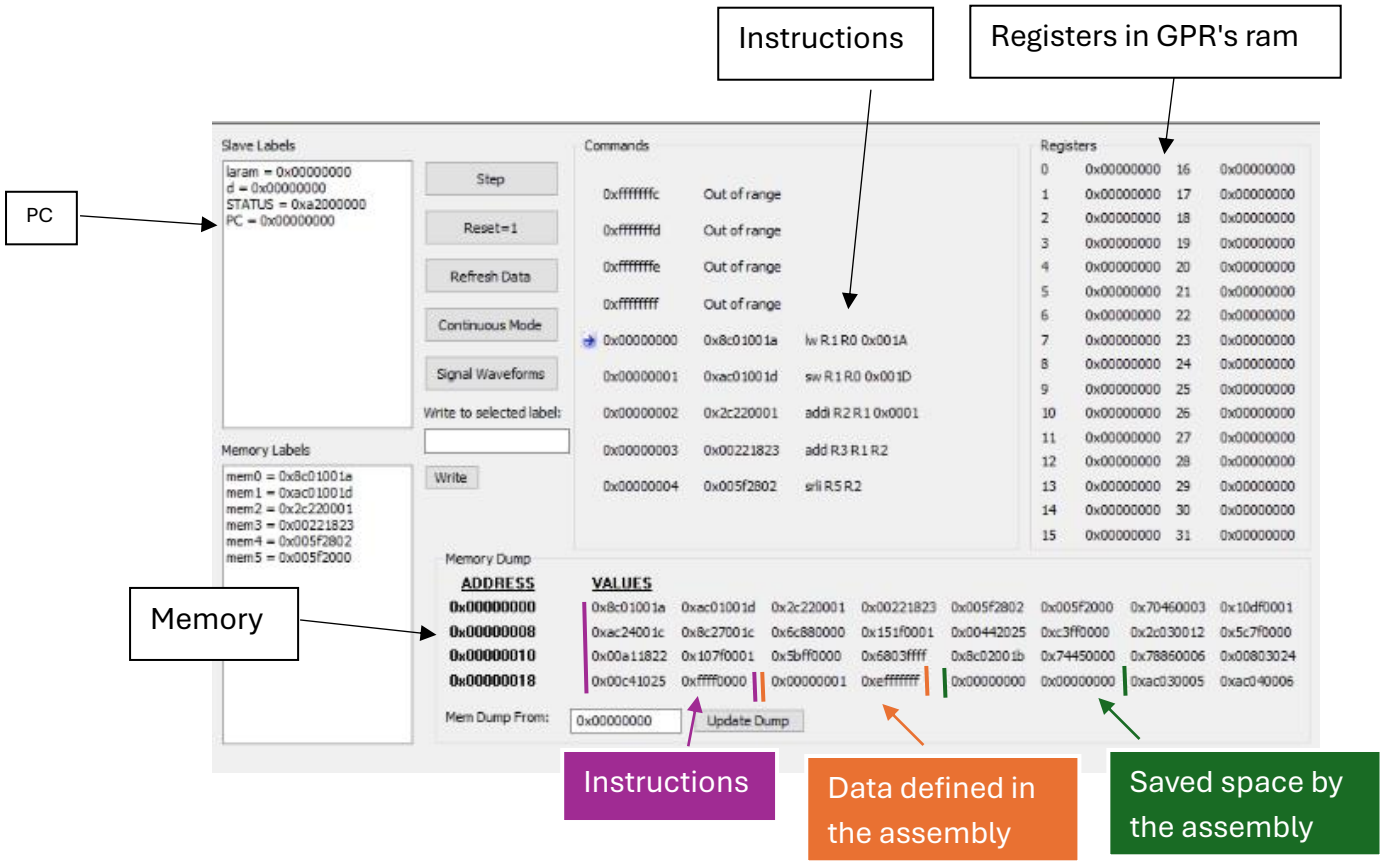


After the connection of the DLX to i/o logic we uploaded it to the fpga using the following label configuration:



In the Graphic labels window, the IN_SIGNALS connections are detailed.

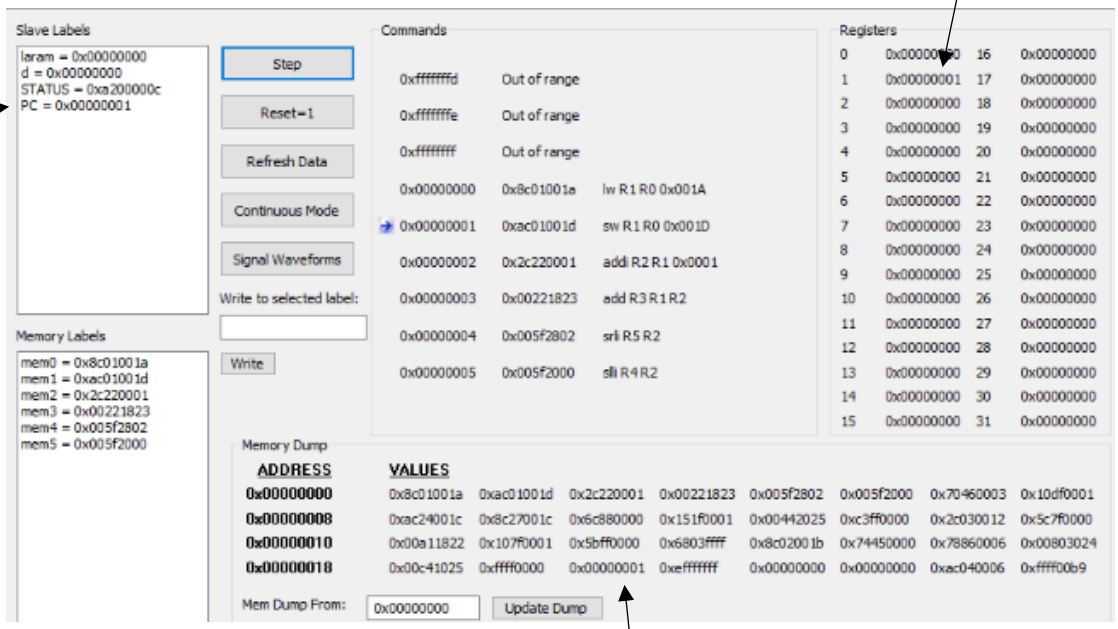
Next, we loaded the relevant bit file and .cod file. The following screen shows the initial values of the fpga memory registers and the gpr's ram memory registers before running and instruction:



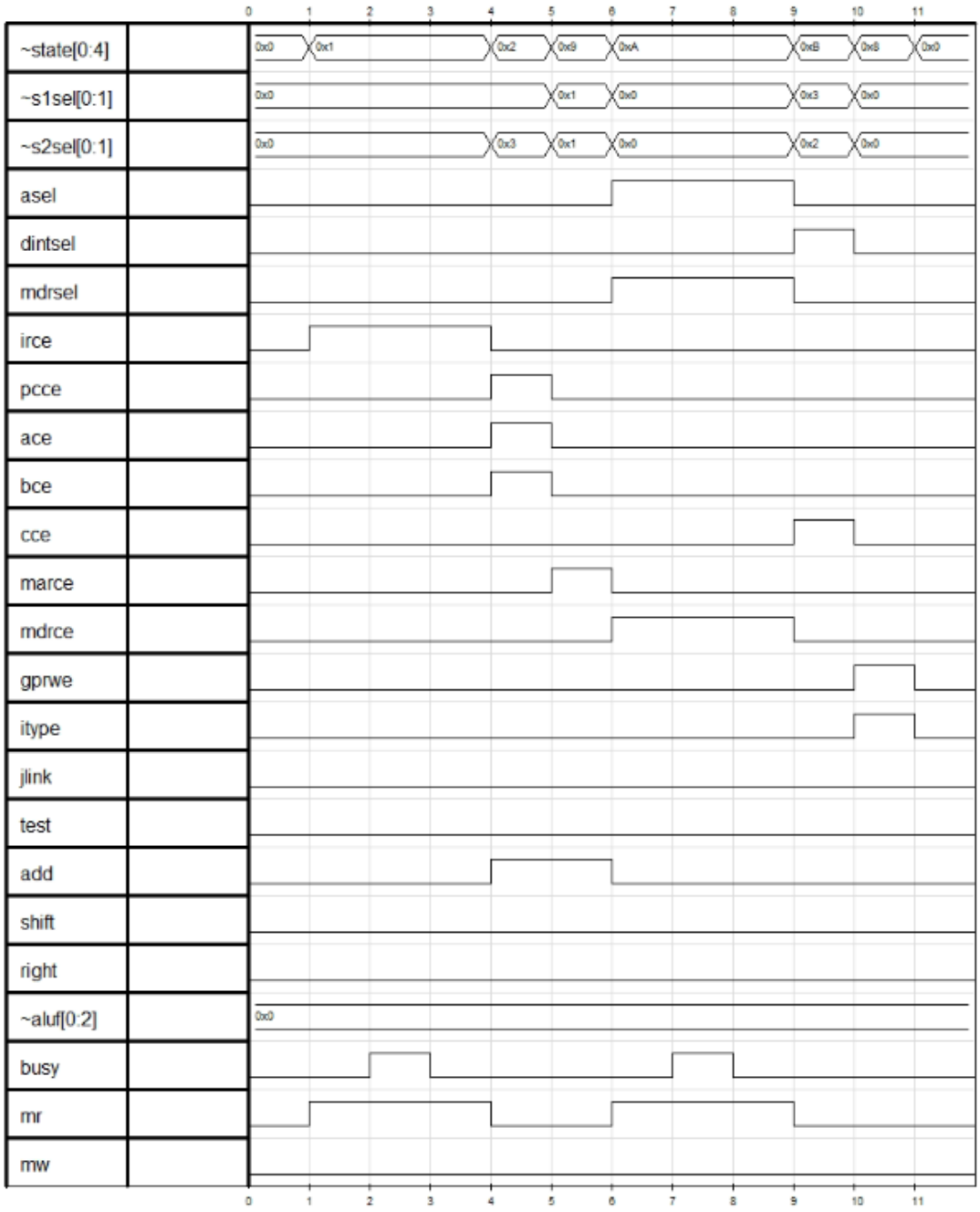
Now for each instruction (disregarding repeating instructions) we will show the formula, the values of the fpga and GPR memory registers before and after the running of the instruction and the signal waveforms after executing the instruction and compare them to the test vectors.

PC = 0x0
Instruction: lw R1 R0 0x001a
Formula: R1 = Mem[0x001a] = 1
PC = PC + 1 = 1
*The before values the same as the initial values above.

After step enable:



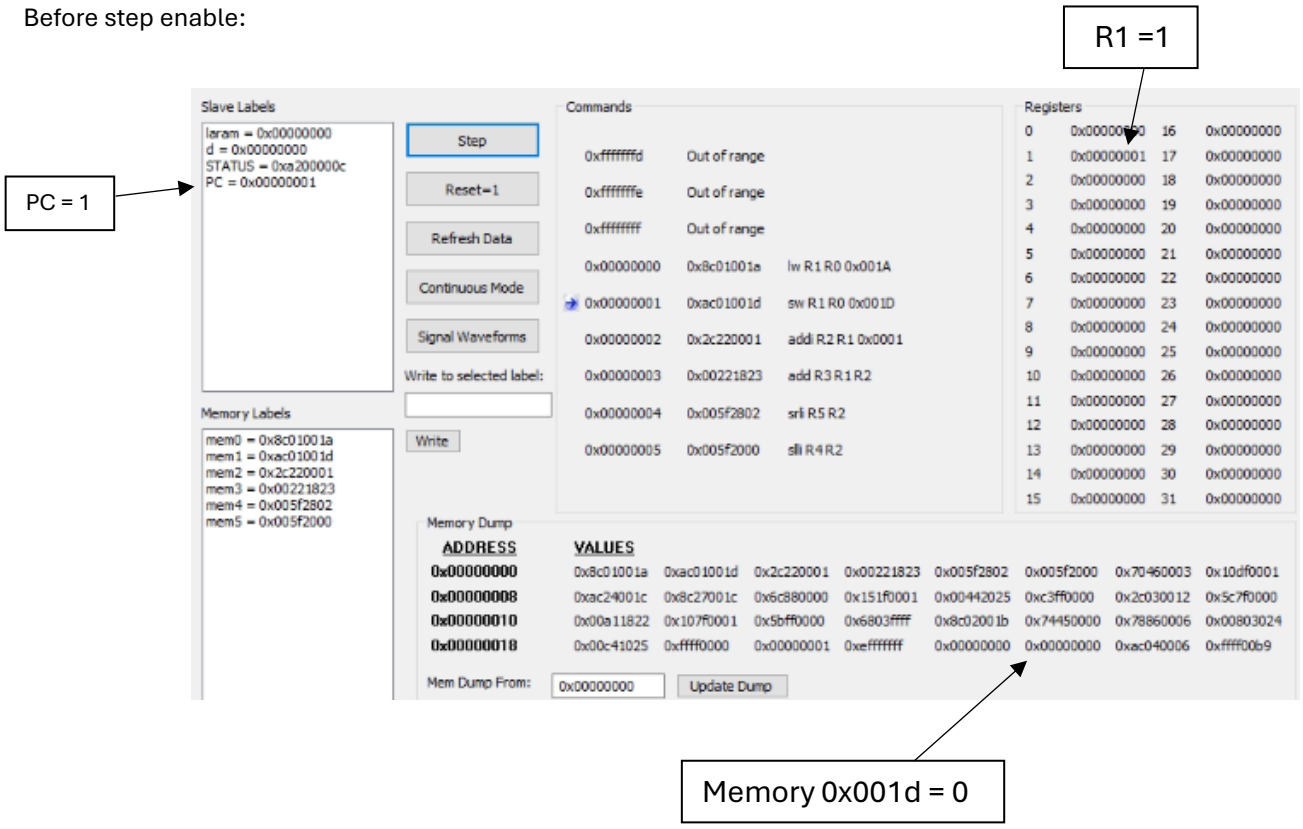
Waveforms:



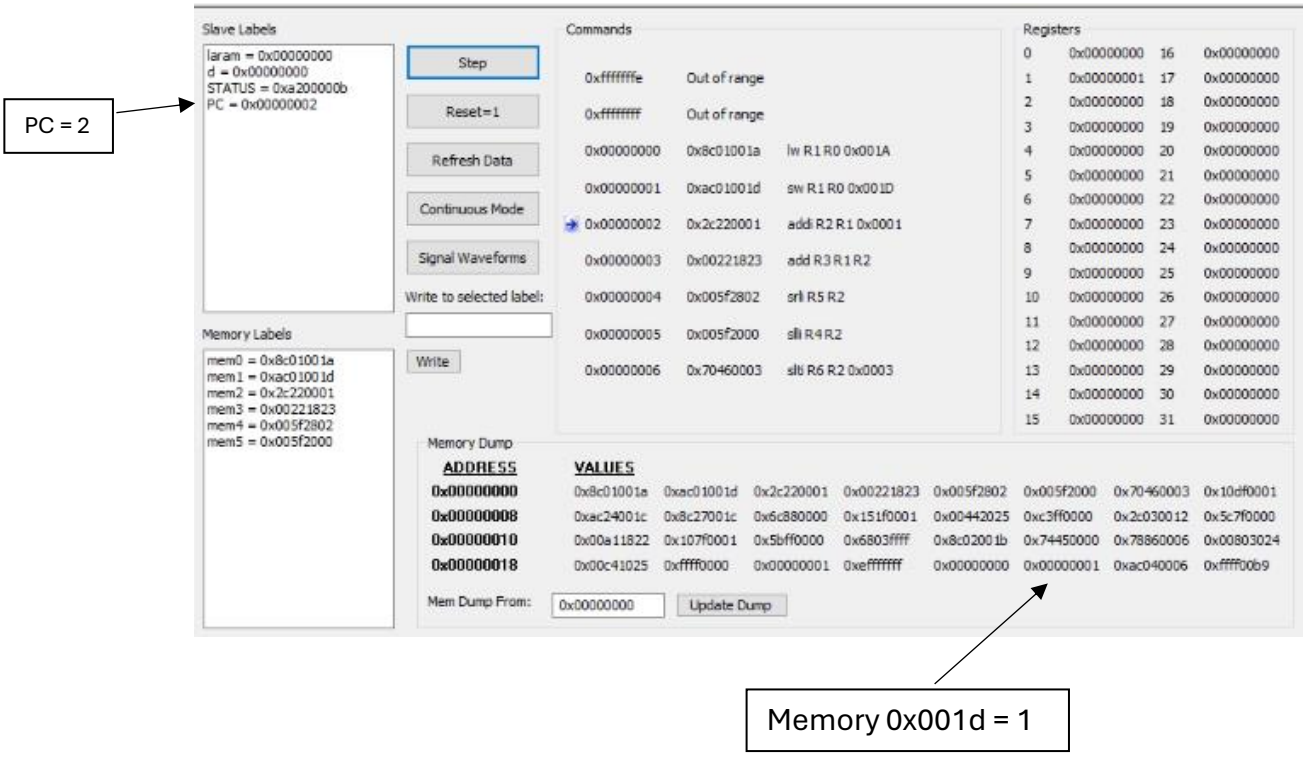
PC = 0x1

Instruction: sw R1 R0 0x001d
Formula: Mem[0x001d] = R1 = 1
PC = PC + 1 = 2

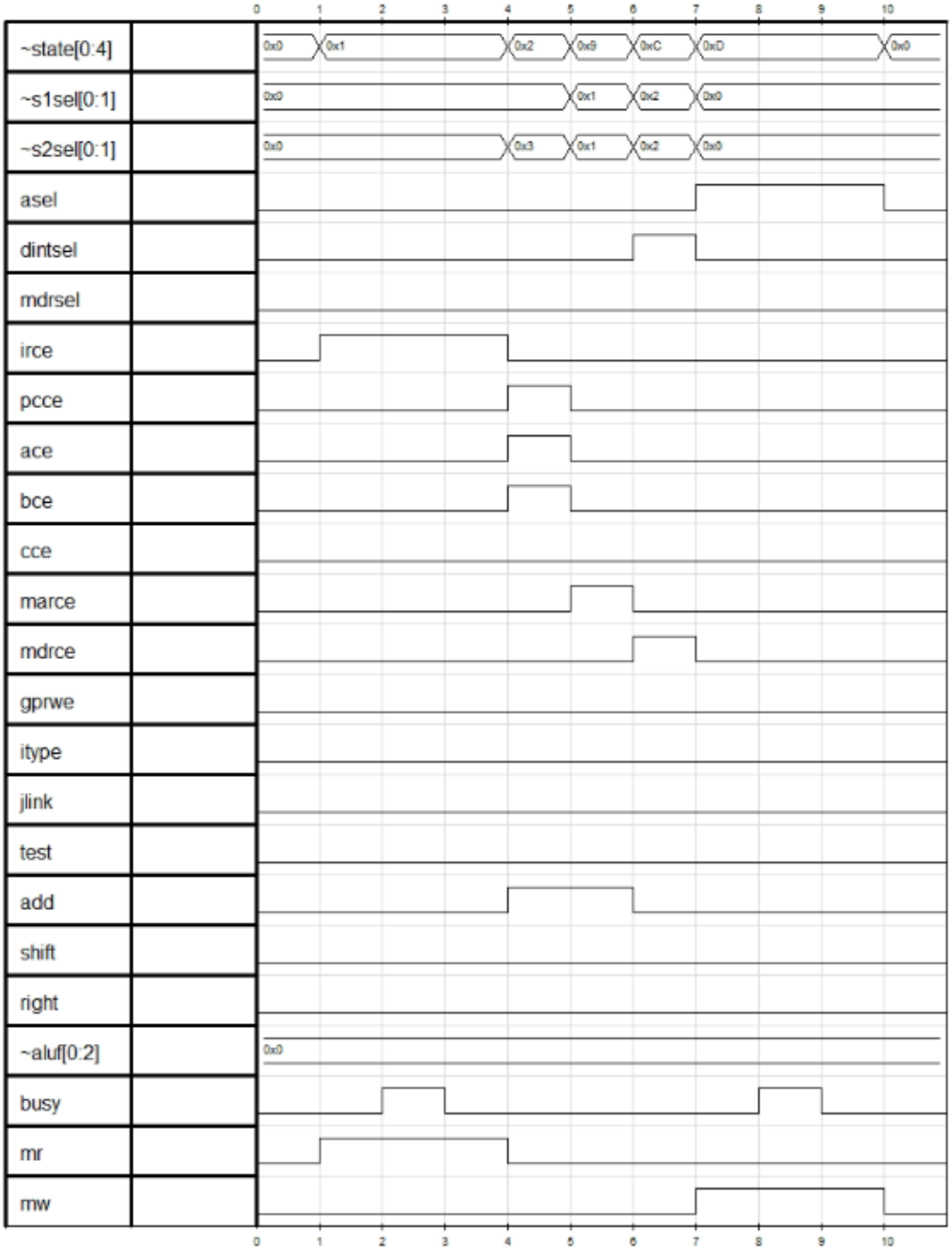
Before step enable:



After step enable:

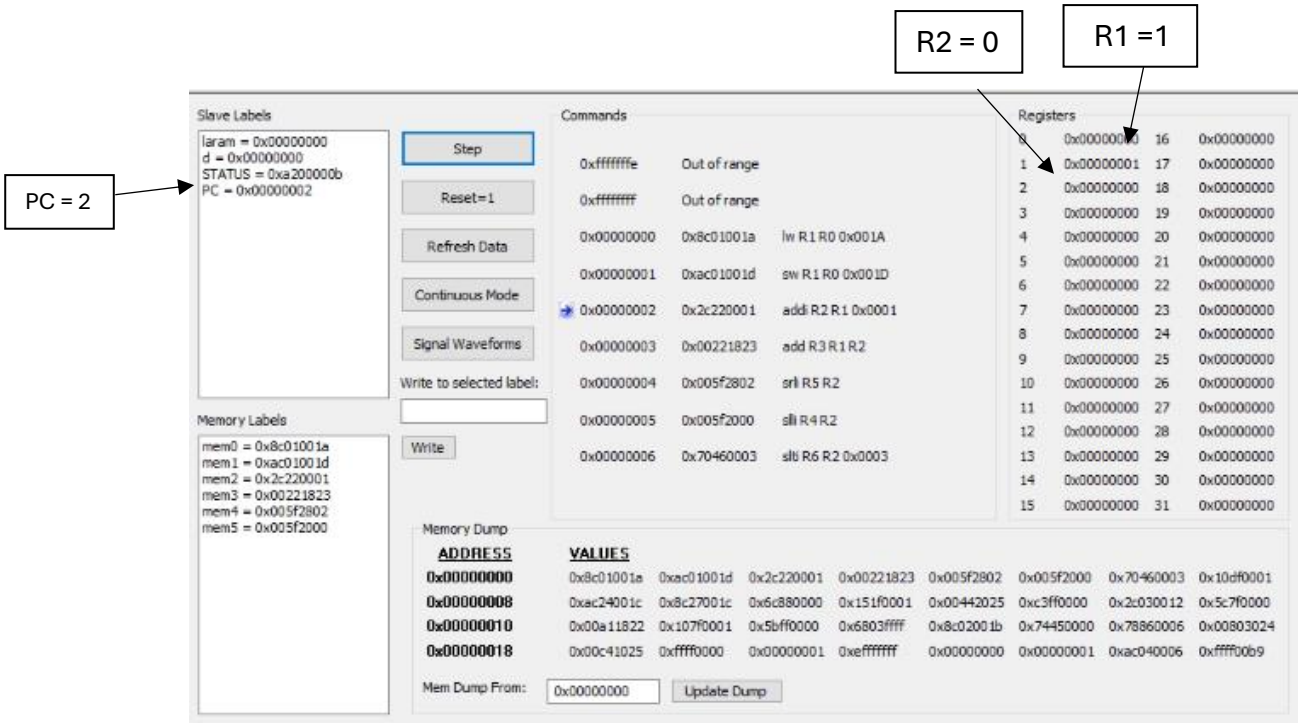


Waveforms:

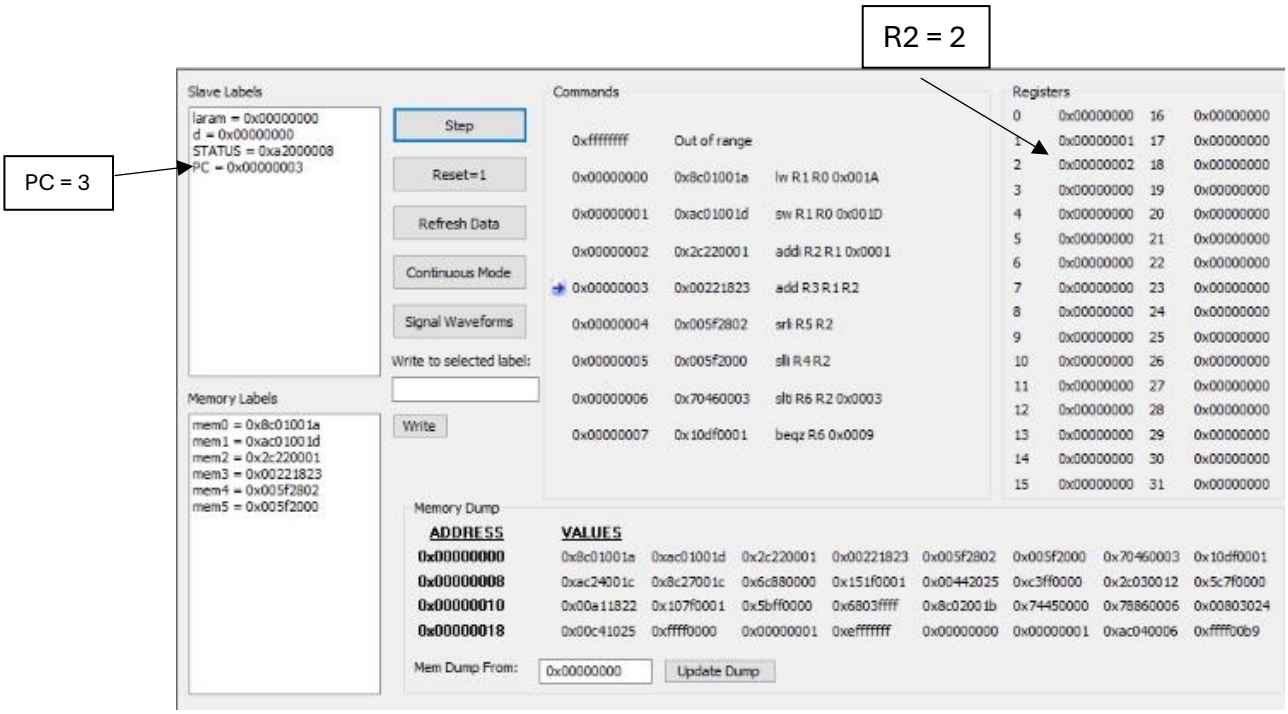

0
1
2
3
4
5
6
7
8
9
10
~state[0:4]
0x0
0x1
0x2
0x9
0xC
0xD
0x0
~s1sel[0:1]
0x0
0x1
0x2
0x0
~s2sel[0:1]
0x0
0x3
0x1
0x2
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x0
busy
mr
mw

PC = 0x2
Instruction: addi R2 R1 0x0001
Formula: R2 = R1 + 1 = 2
PC = PC + 1 = 3

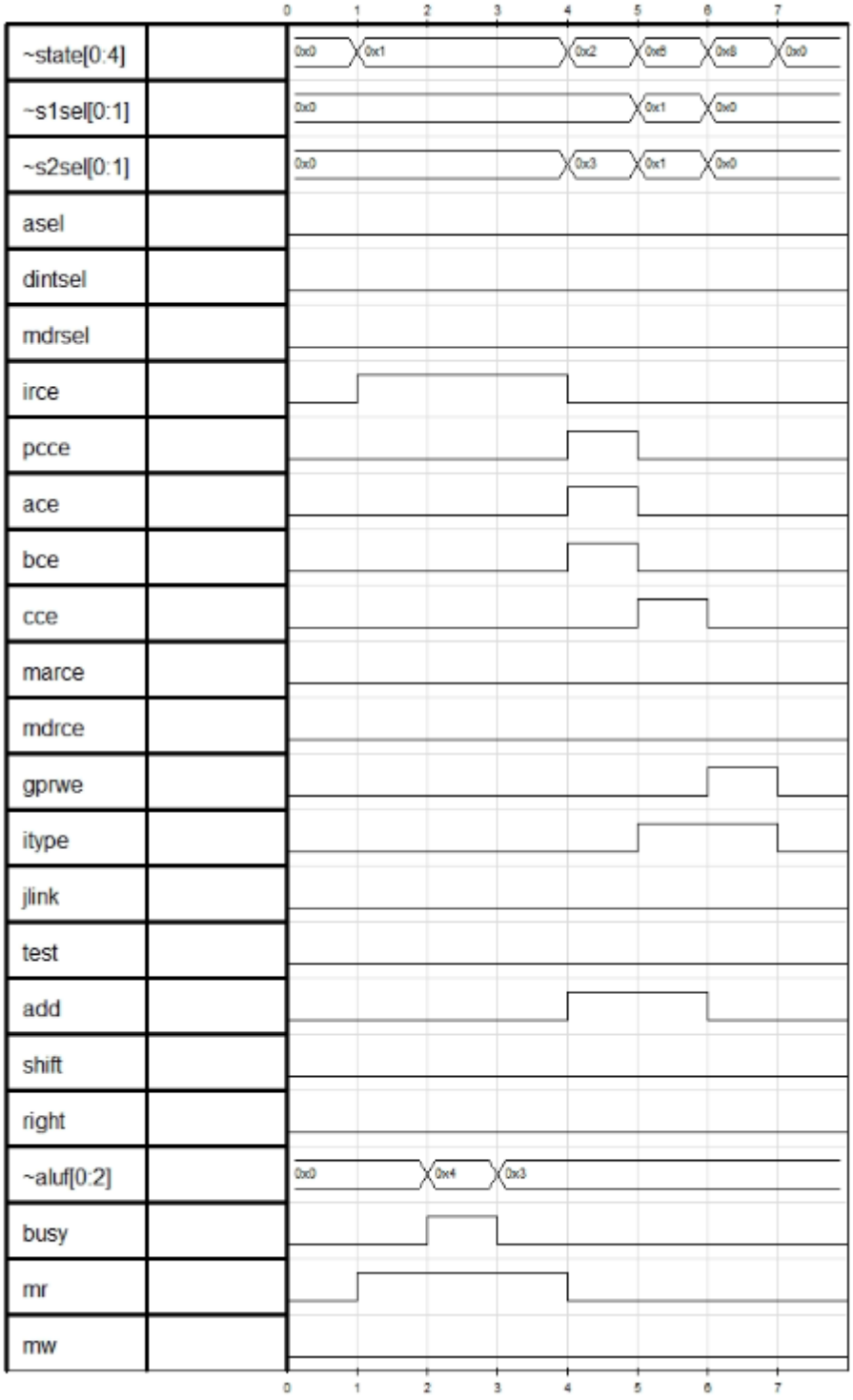
Before step enable:



After step enable:

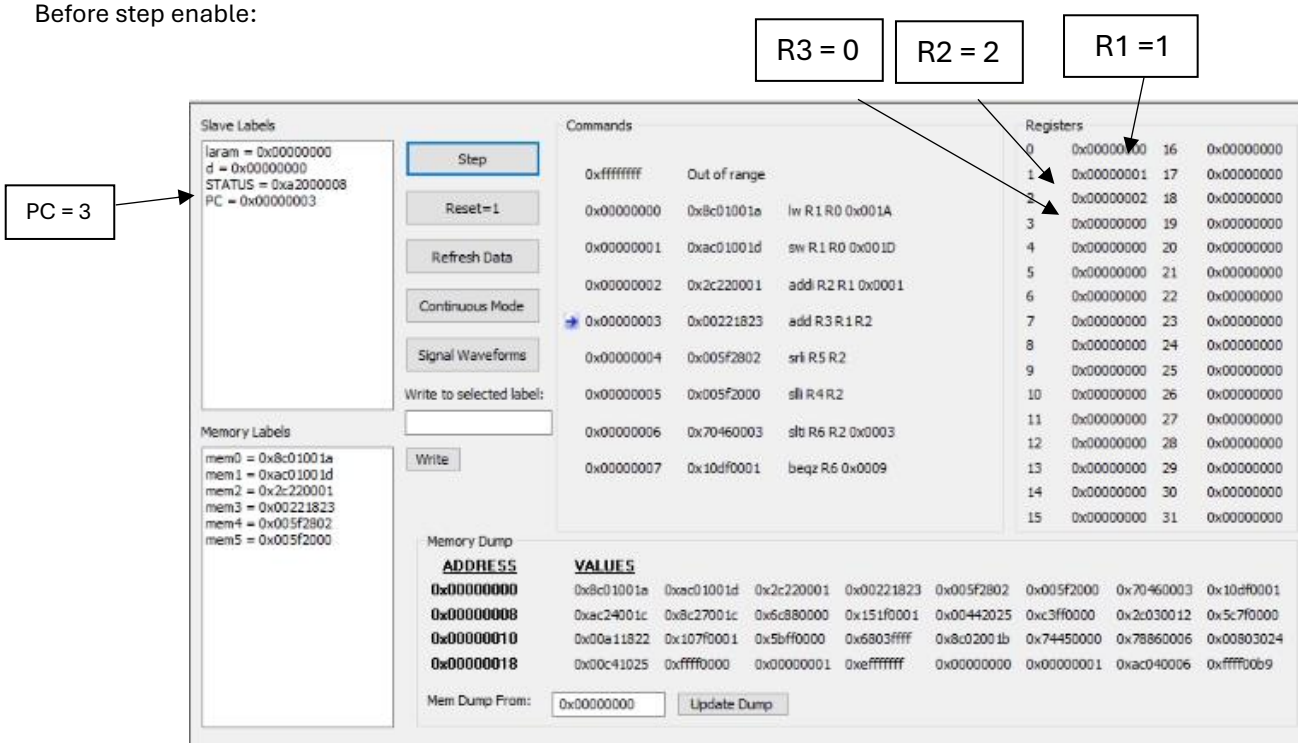


Waveforms:

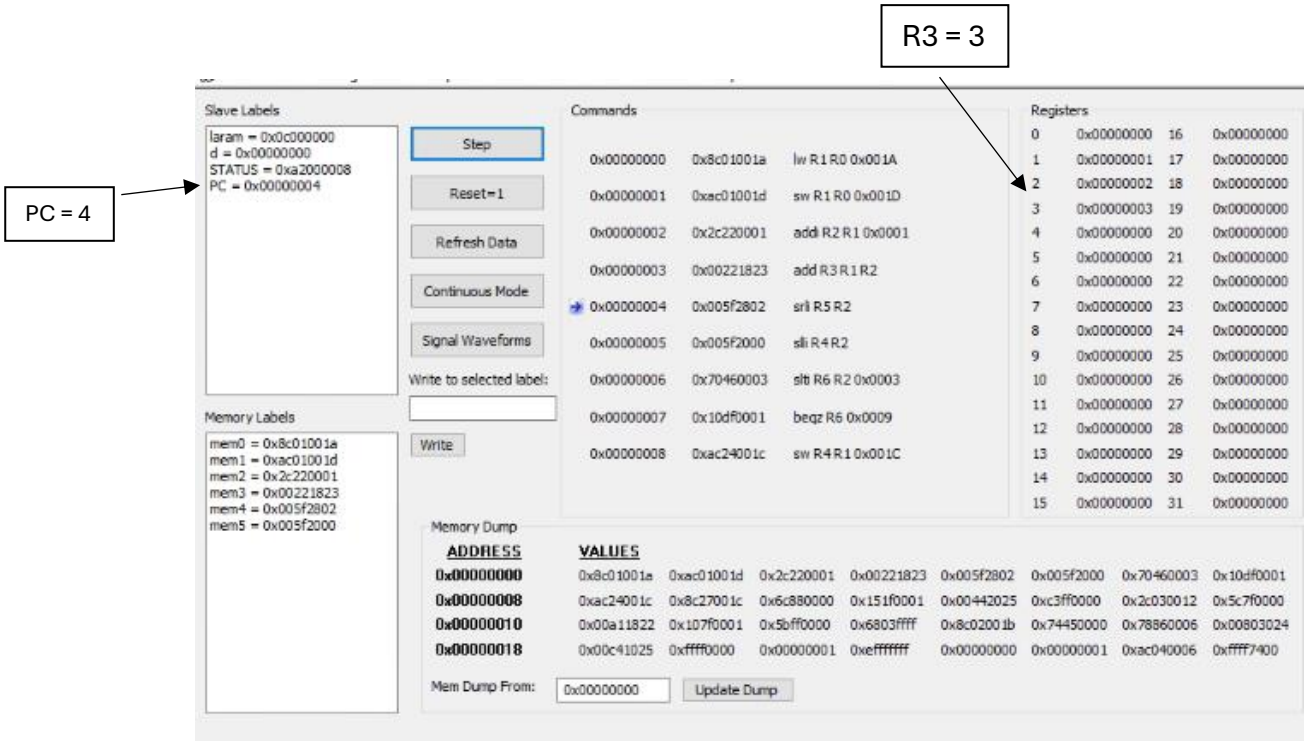

0
1
2
3
4
5
6
7
~state[0:4]
0x0
0x1
0x2
0x5
0x8
0x0
~s1sel[0:1]
0x0
0x1
0x0
~s2sel[0:1]
0x0
0x3
0x1
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x0
0x4
0x3
busy
mr
mw

PC = 0x3
Instruction: add R3 R1 R2
Formula: R3 = R1 + R2 = 3
PC = PC + 1 = 4

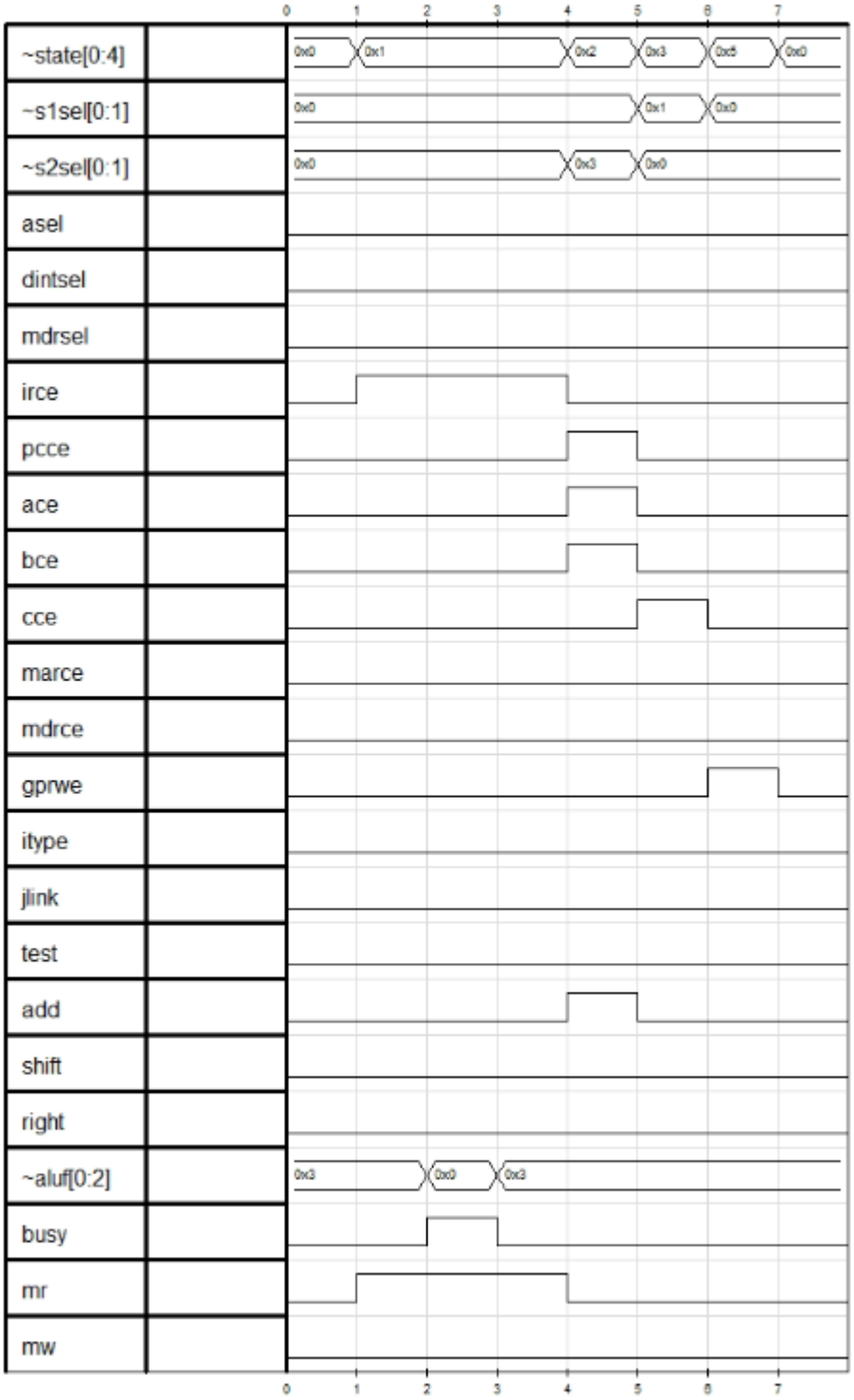
Before step enable:



After step enable:

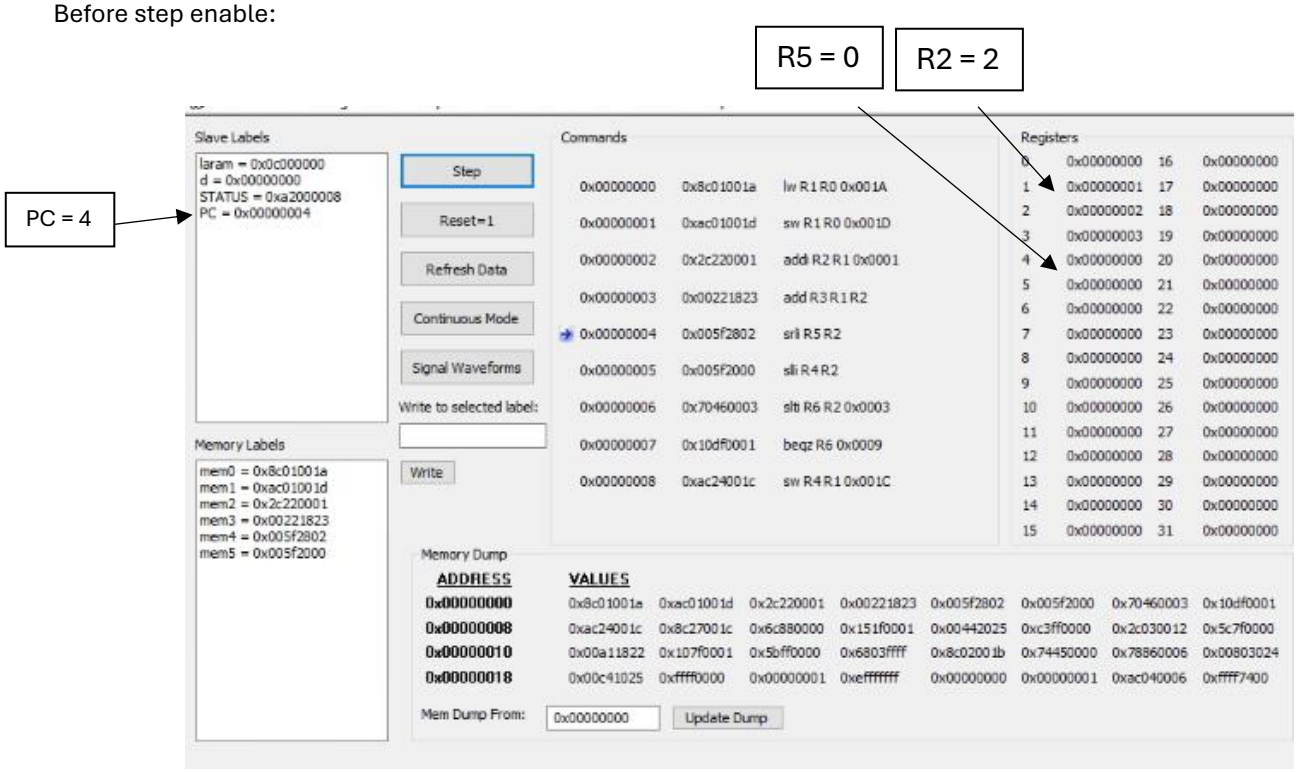


Waveforms:

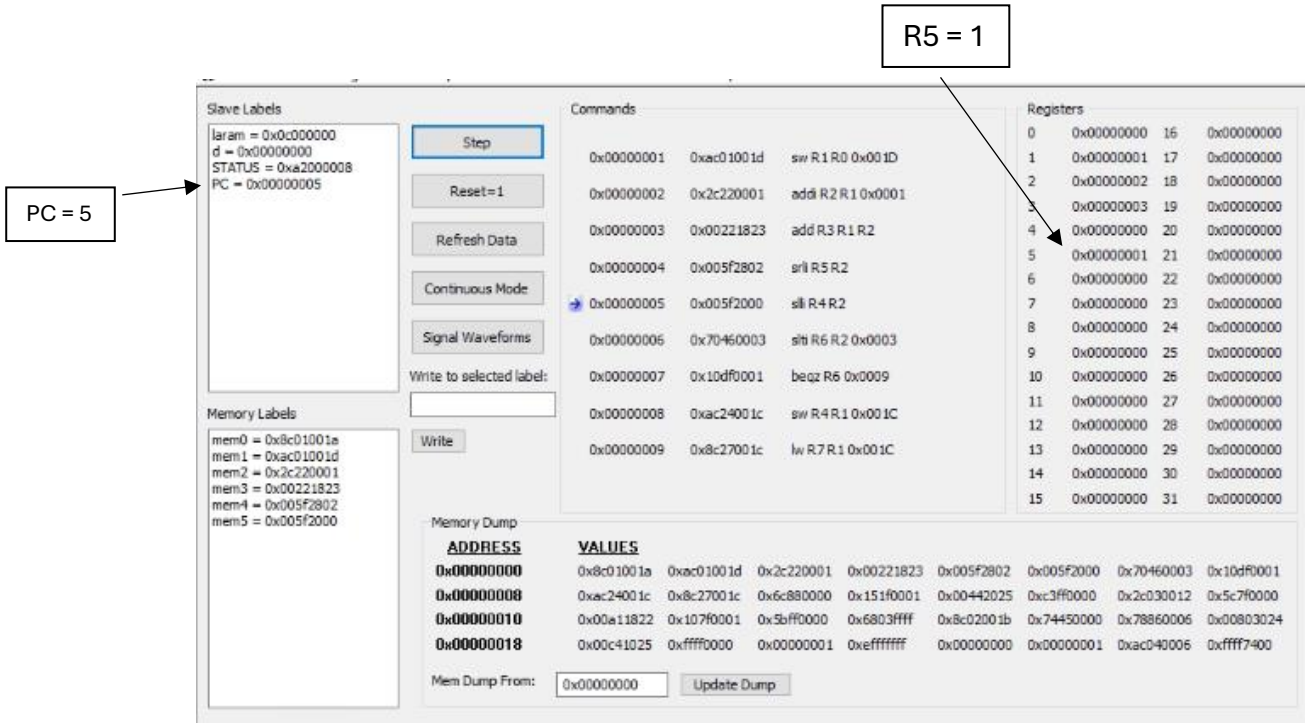

0
1
2
3
4
5
6
7
~state[0:4]
0x0
0x1
0x2
0x3
0x5
0x0
~s1sel[0:1]
0x0
0x1
0x0
~s2sel[0:1]
0x0
0x3
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x3
0x0
0x3
busy
mr
mw
0
1
2
3
4
5
6
7

PC = 0x4
Instruction: srli R5 R2
Formula: R5 = R2 >> 1 = 1
PC = PC + 1 = 5

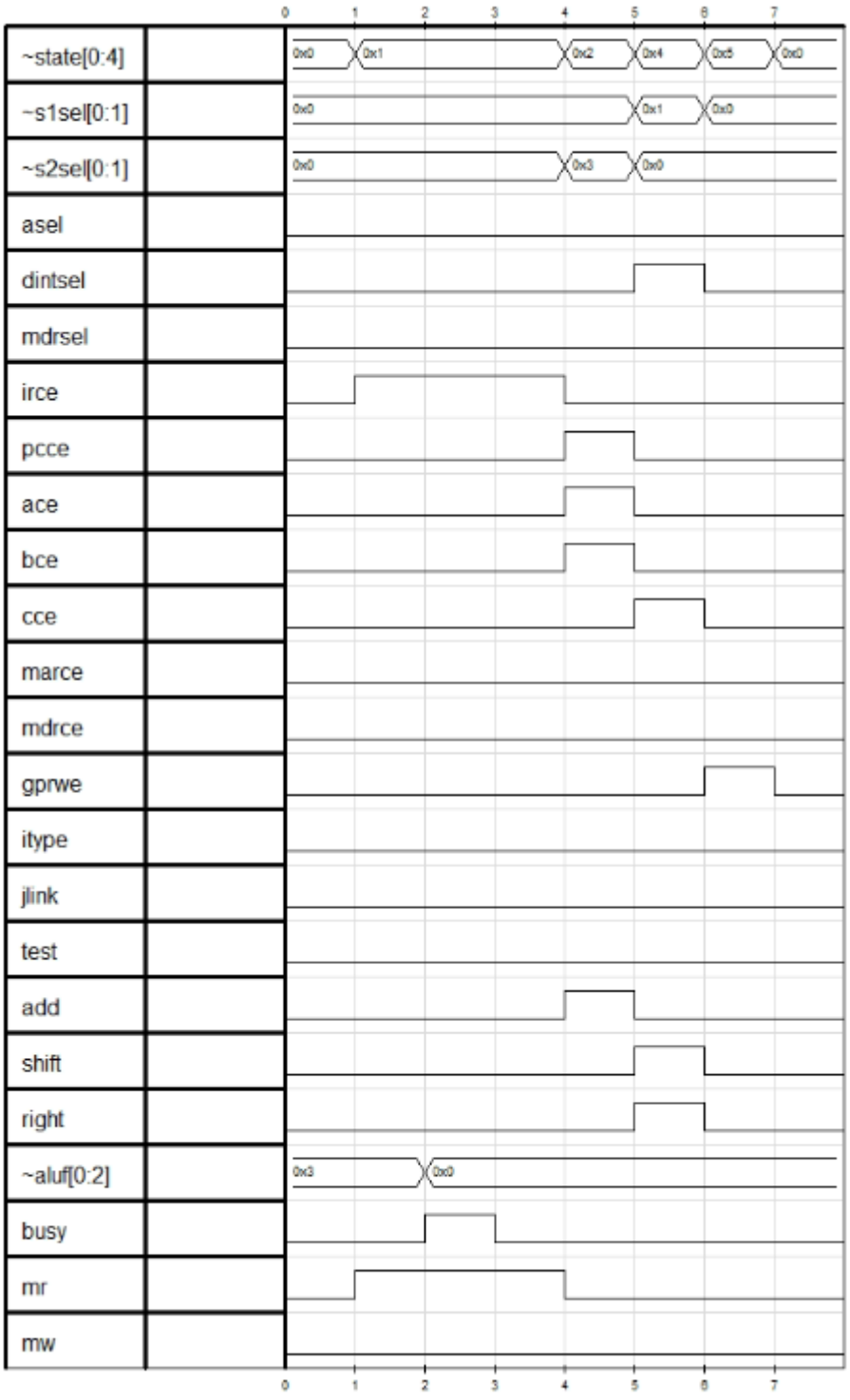
Before step enable:



After step enable:

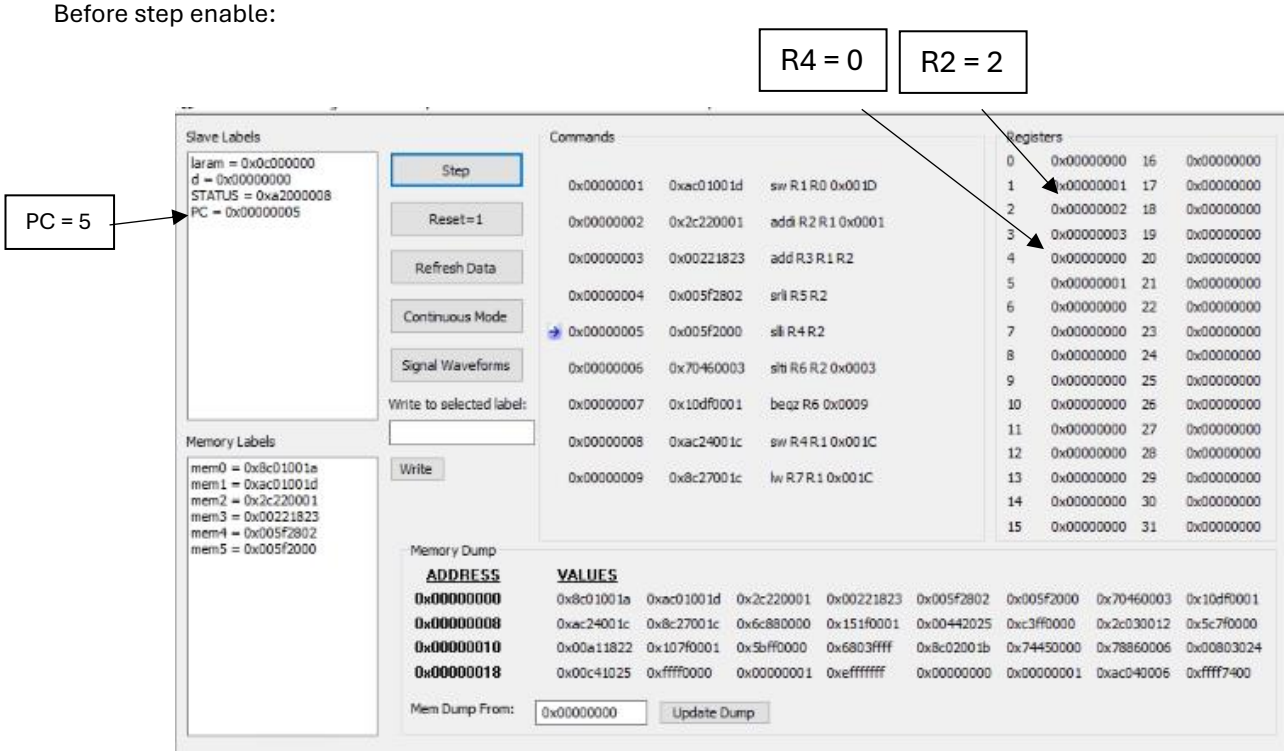


Waveforms:

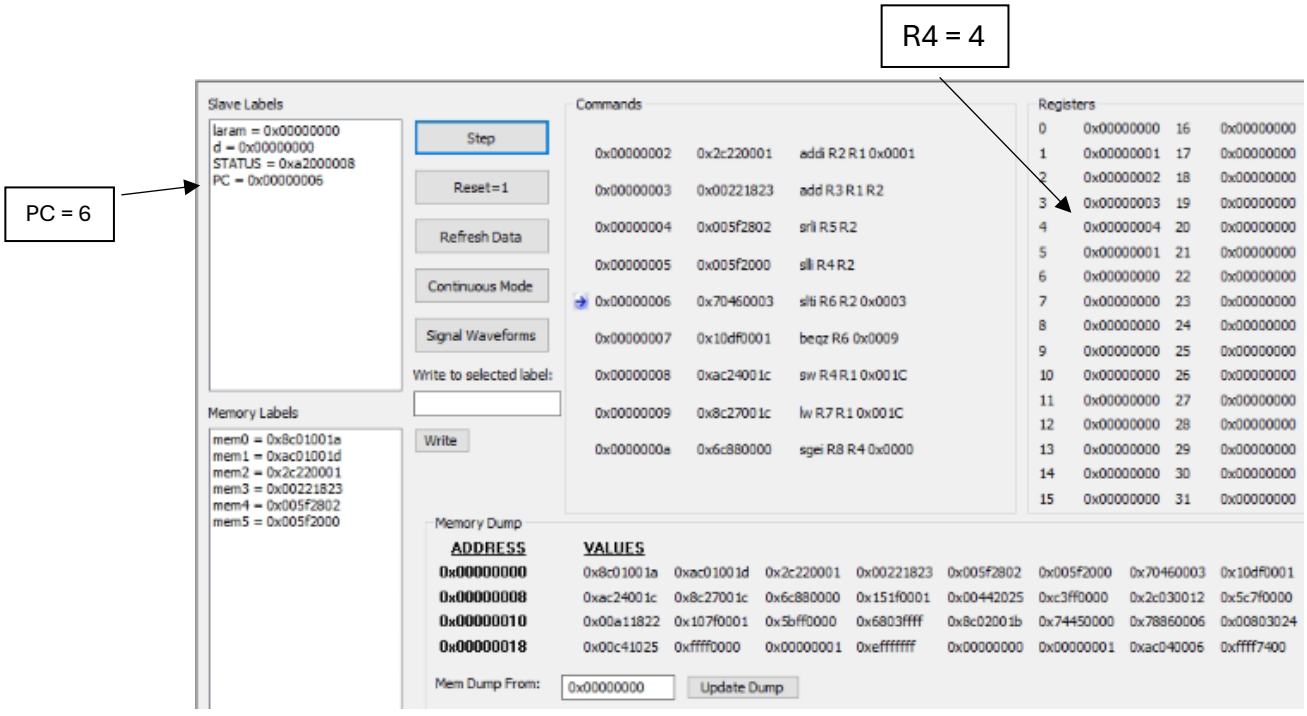

0
1
2
3
4
5
6
7
~state[0:4]
0x0
0x1
0x2
0x4
0x5
0x0
~s1sel[0:1]
0x0
0x1
0x0
~s2sel[0:1]
0x0
0x3
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x3
0x0
busy
mr
mw

PC = 0x5
Instruction: slli R4 R2
Formula: R4 = R2 << 1 = 4
PC = PC + 1 = 6

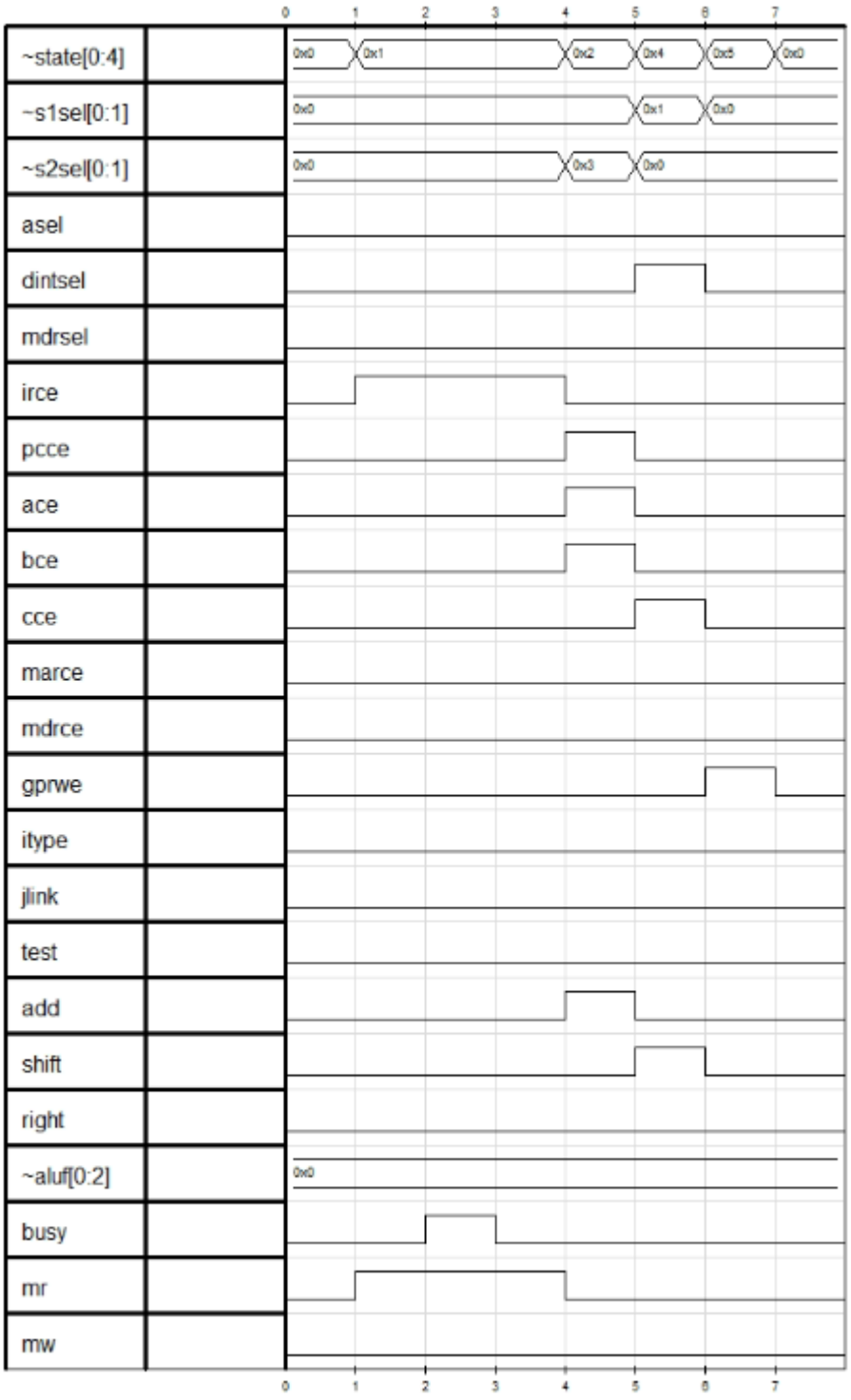
Before step enable:



After step enable:

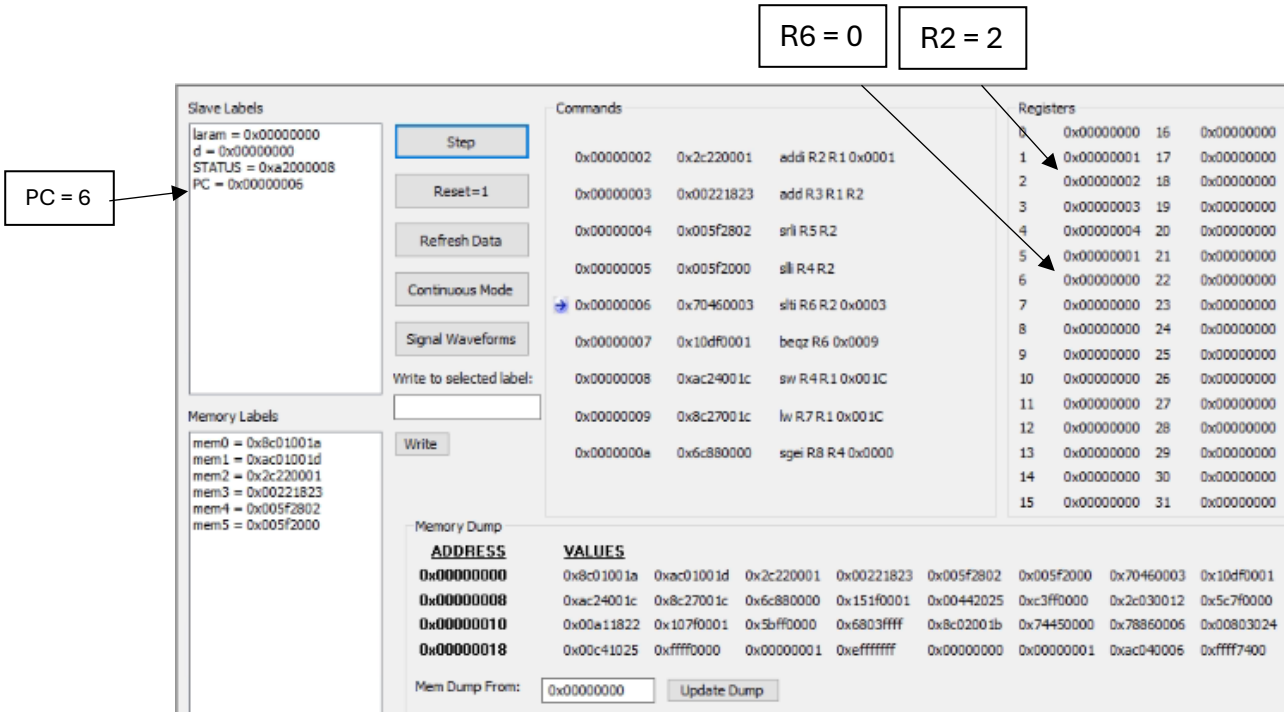


Waveforms:

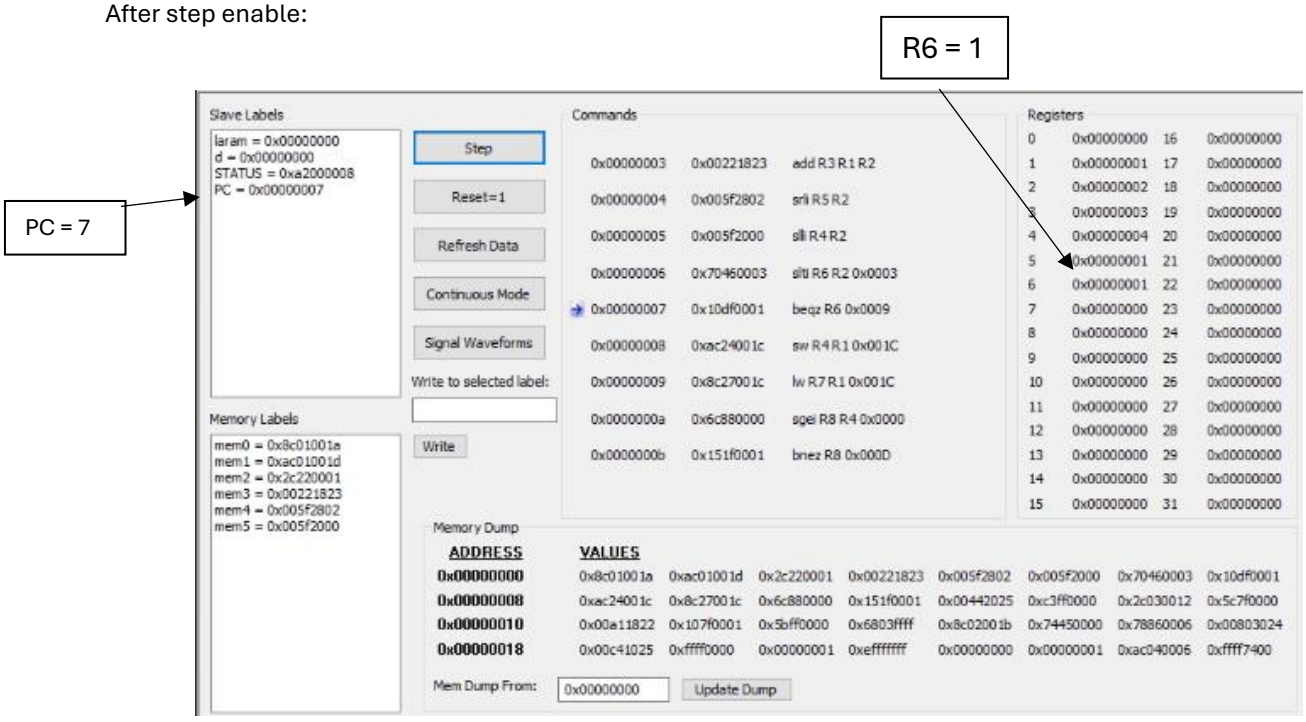

0
1
2
3
4
5
6
7
~state[0:4]
0x0
0x1
0x2
0x4
0x5
0x0
~s1sel[0:1]
0x0
0x1
0x0
~s2sel[0:1]
0x0
0x3
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x0
busy
mr
mw

PC = 0x6
Instruction: slti R6 R2 0x003
Formula: R6 = (R2 < 3) ? 1:0 = > R6 = 1
PC = PC + 1 = 7

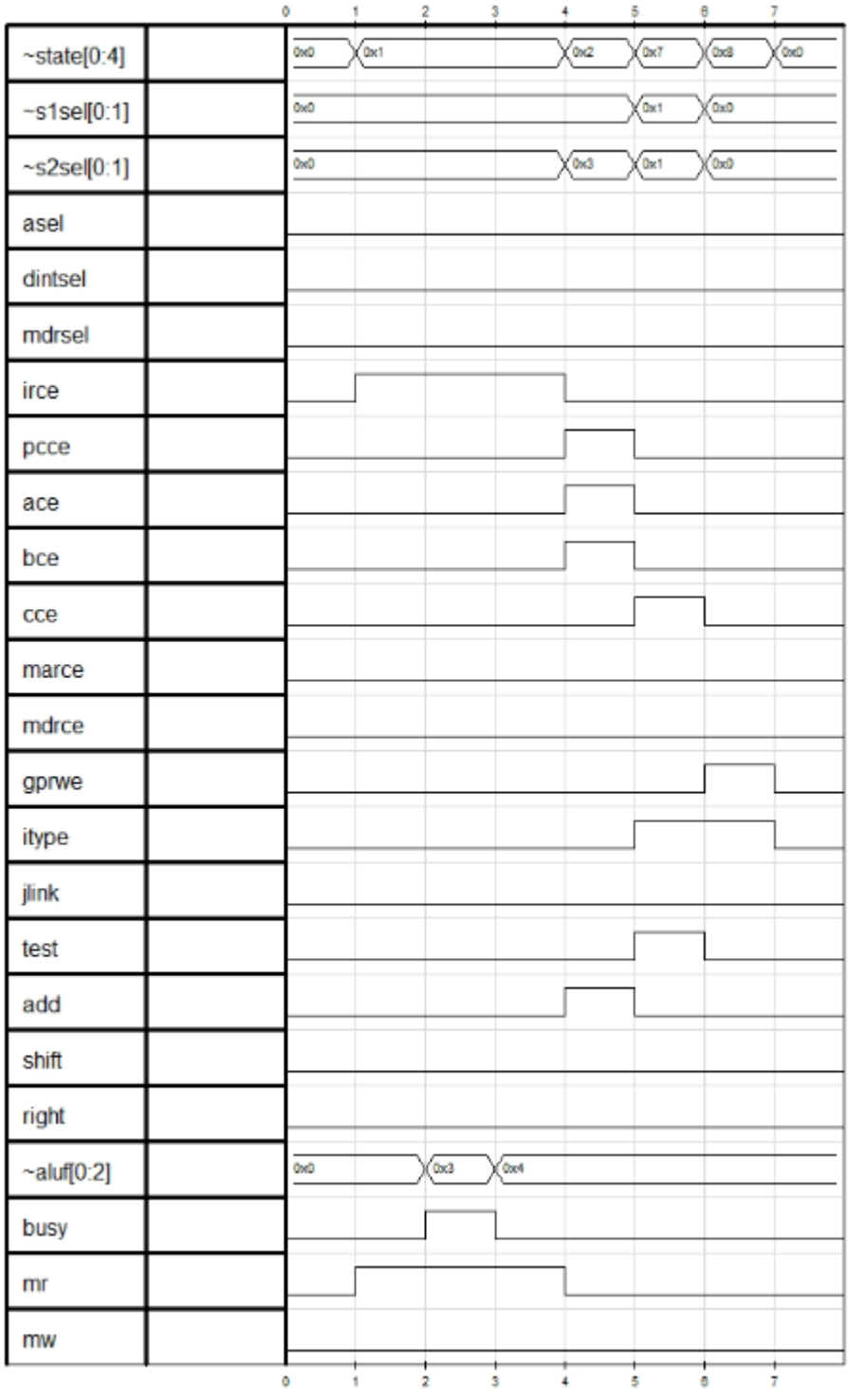
Before step enable:



After step enable:

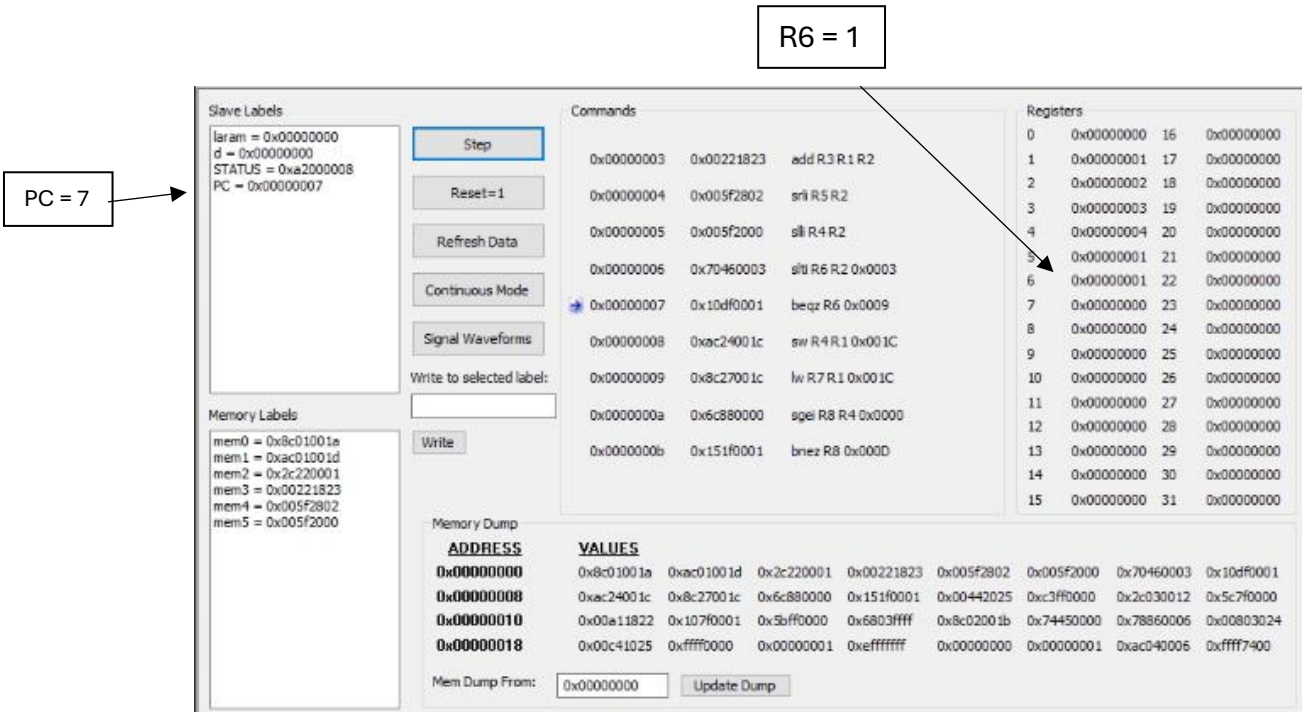


Waveforms:

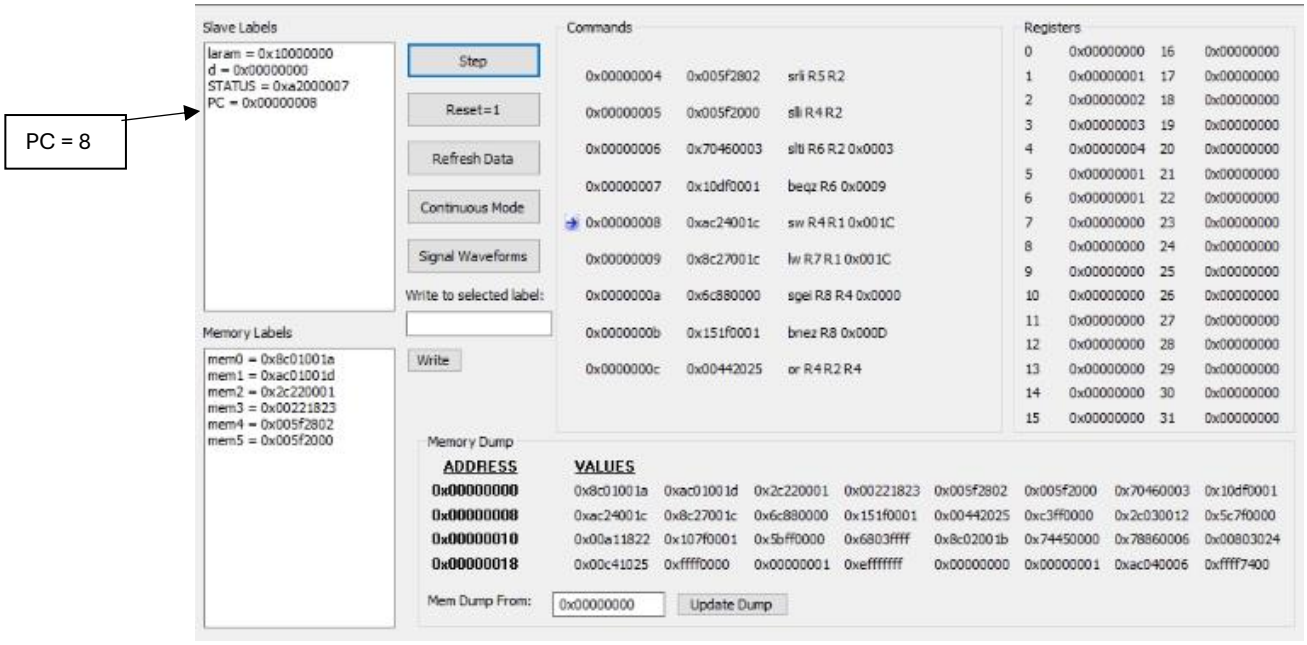

0
1
2
3
4
5
6
7
~state[0:4]
0x0
0x1
0x2
0x7
0x8
0x0
~s1sel[0:1]
0x0
0x1
0x0
~s2sel[0:1]
0x0
0x3
0x1
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x0
0x3
0x4
busy
mr
mw

PC = 0x7
Instruction: beqz R6 R2 0x009
Formula: PC = (!R6) ? 0x009 : (pc+1)
PC = PC + 1 = 8

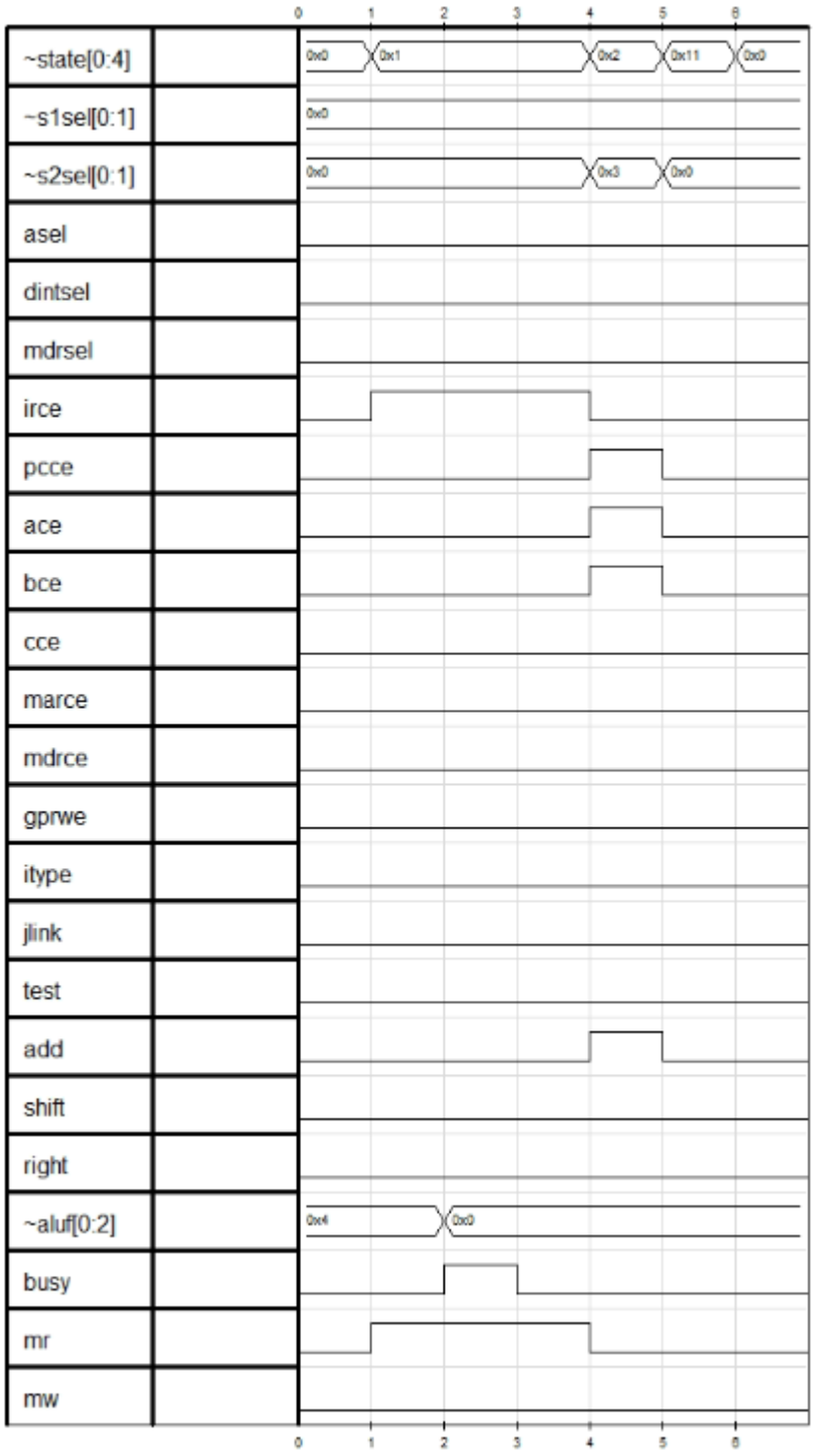
Before step enable:



After step enable:

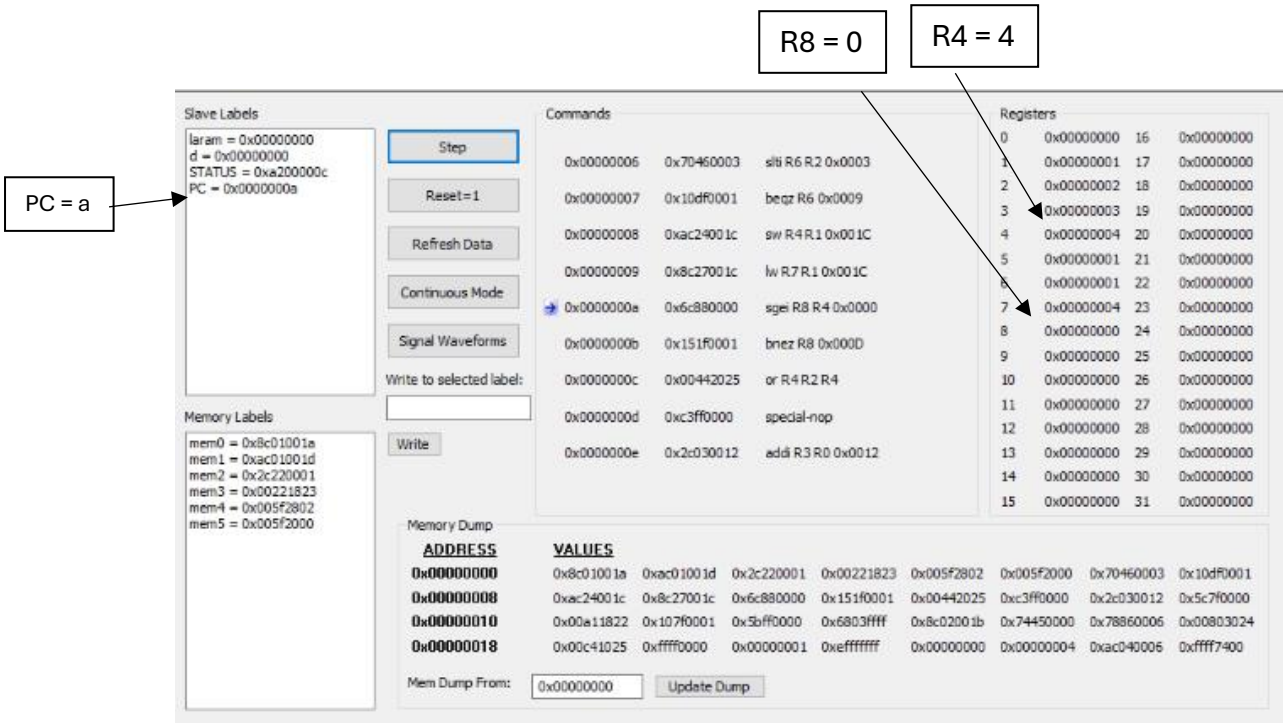


Waveforms:

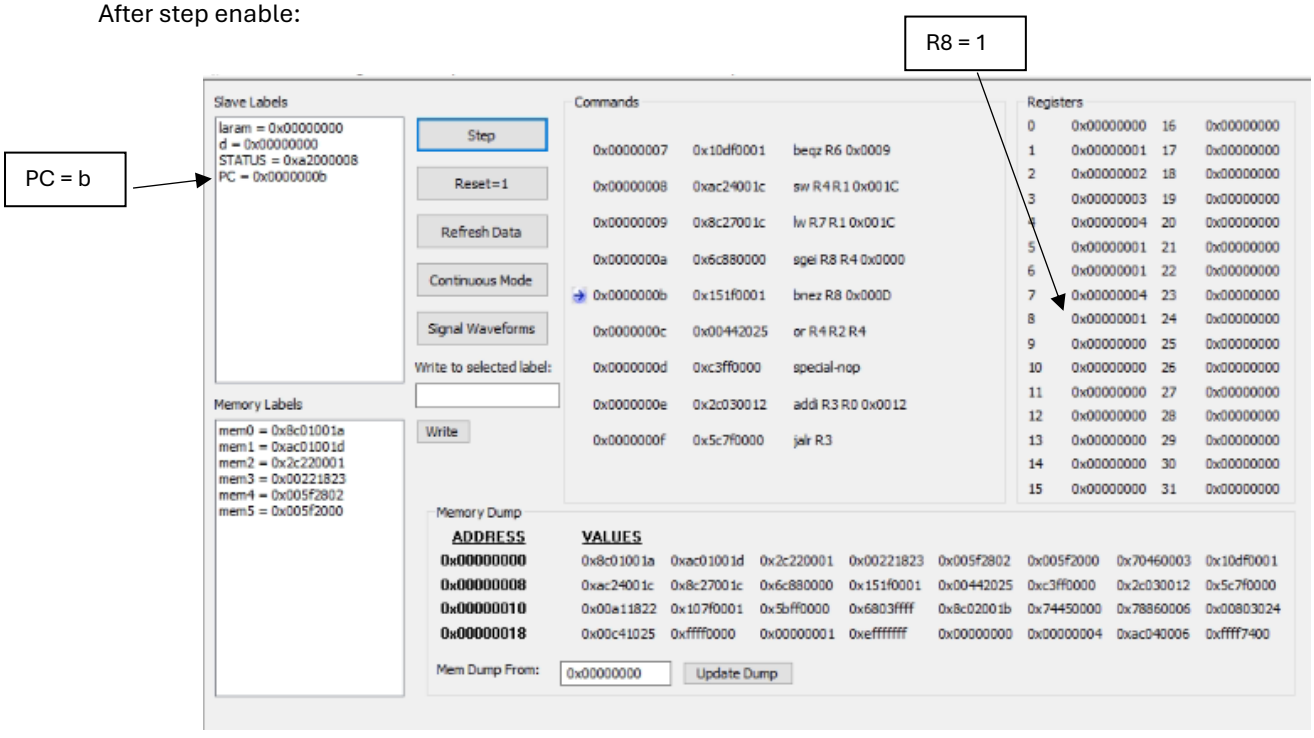

0
1
2
3
4
5
6
~state[0:4]
0x0
0x1
0x2
0x11
0x0
~s1sel[0:1]
0x0
~s2sel[0:1]
0x0
0x3
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x4
0x0
busy
mr
mw

PC = 0xa
Instruction: sgei R8 R4 0x000
Formula: R8 = (R4 >= 0) ? 1 : 0 => R8 = 1
PC = PC + 1 = b

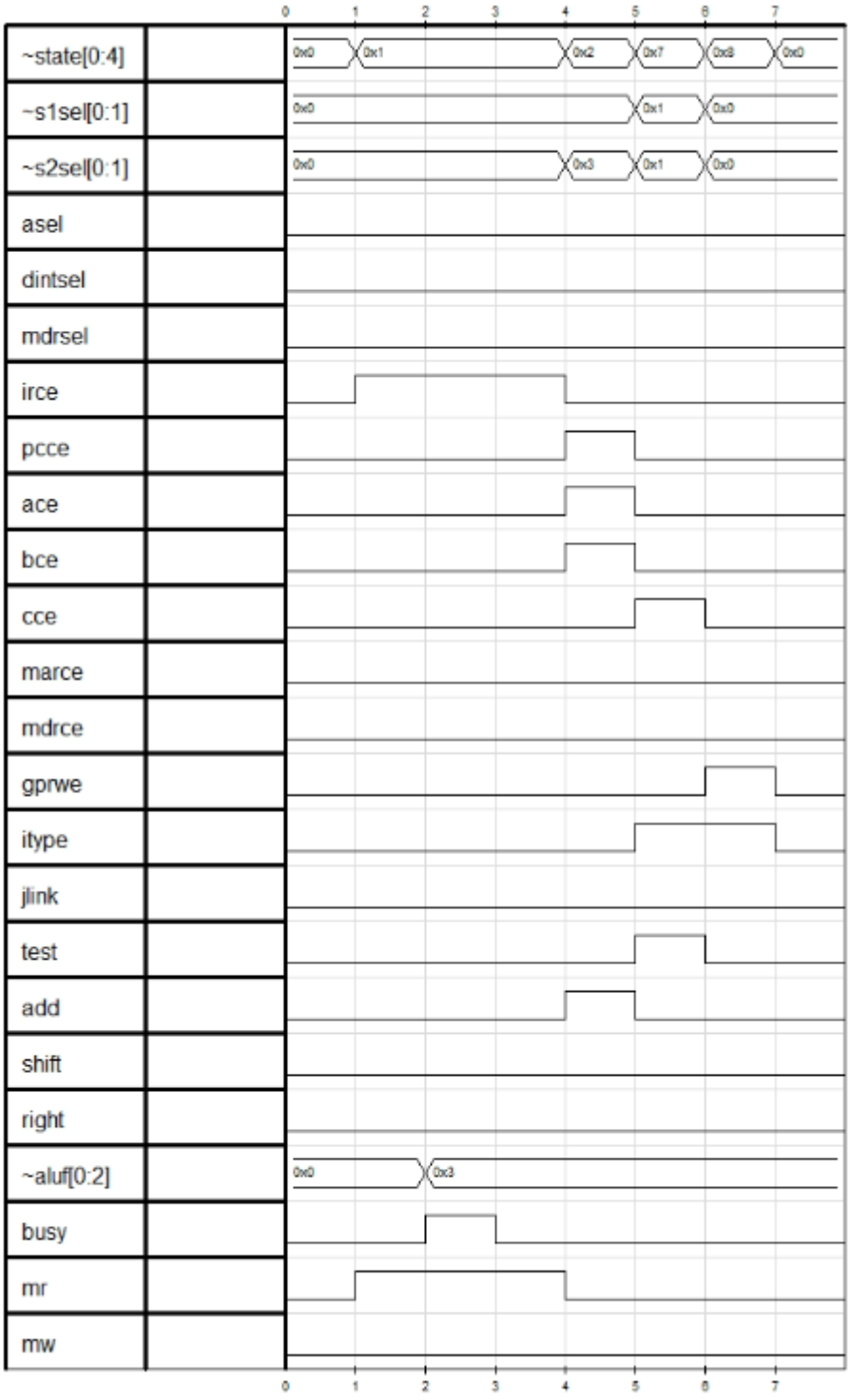
Before step enable:



After step enable:

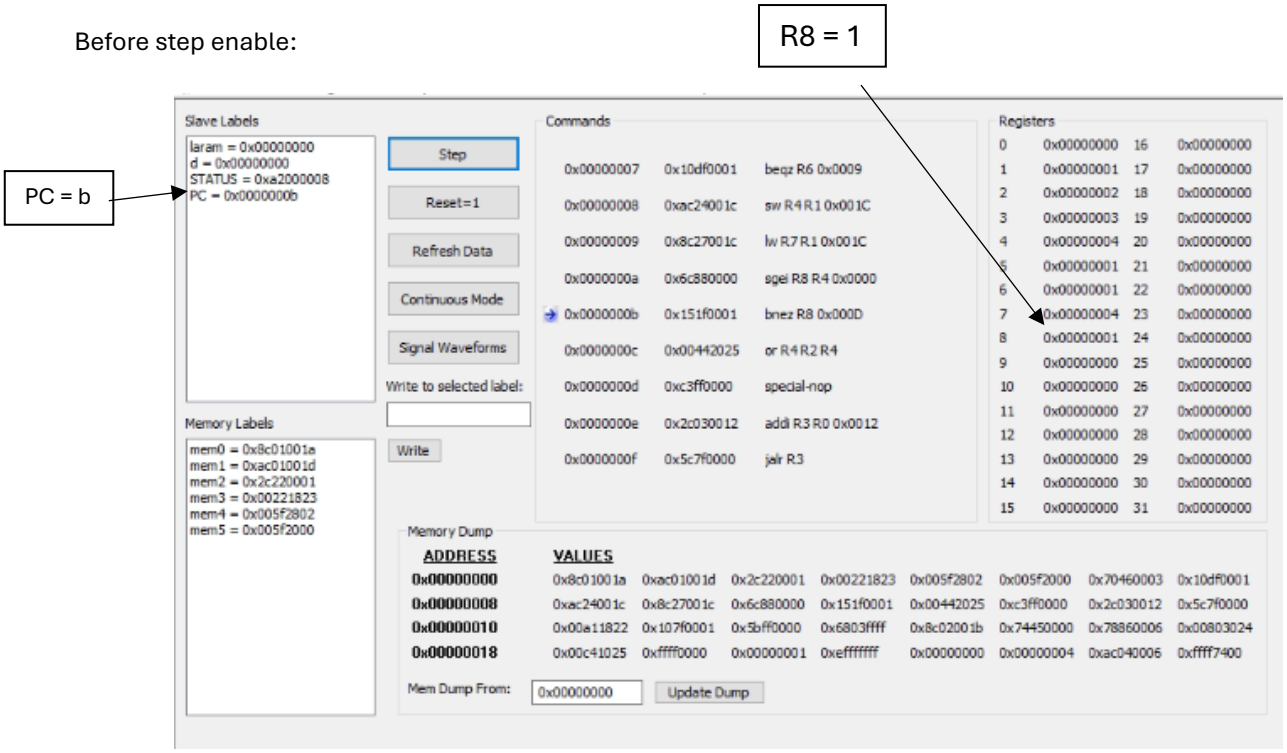


Waveforms:

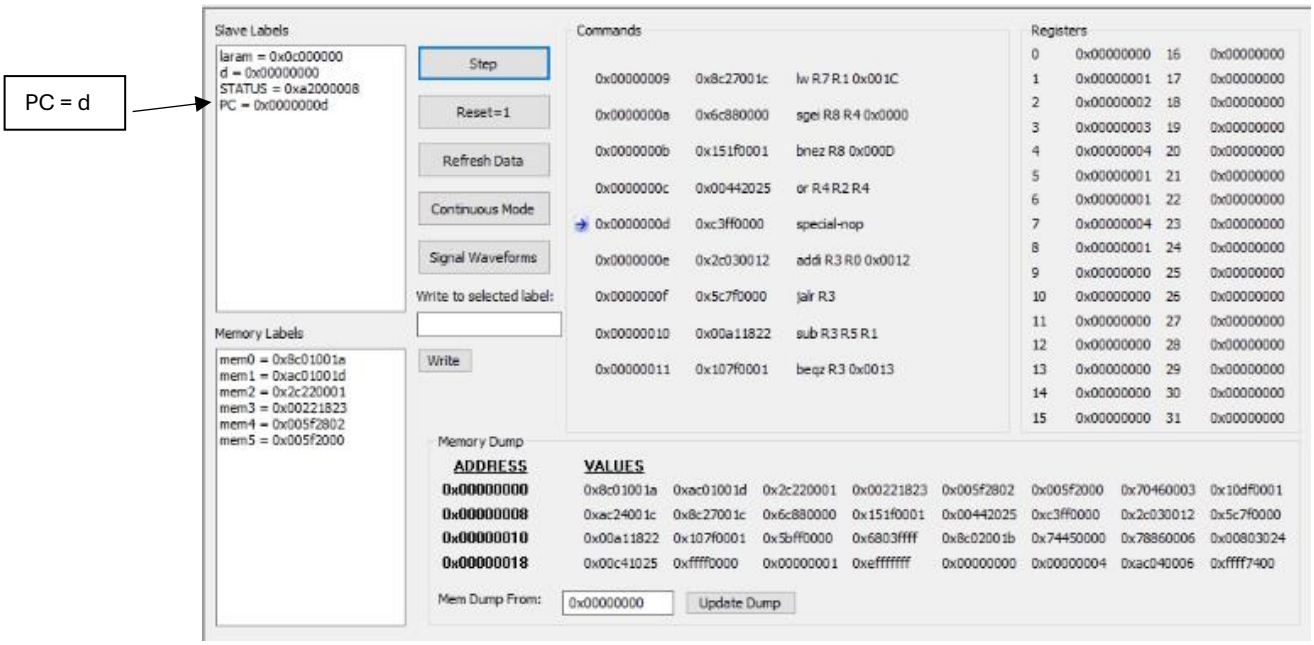

0
1
2
3
4
5
6
7
~state[0:4]
0x0
0x1
0x2
0x7
0x8
0x0
~s1sel[0:1]
0x0
0x1
0x0
~s2sel[0:1]
0x0
0x3
0x1
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x0
0x3
busy
mr
mw

PC = 0xb
Instruction: bnez R8 0x000d
Formula: PC = (R8) ? 0x000d : (pc+1)
PC = 0x000d

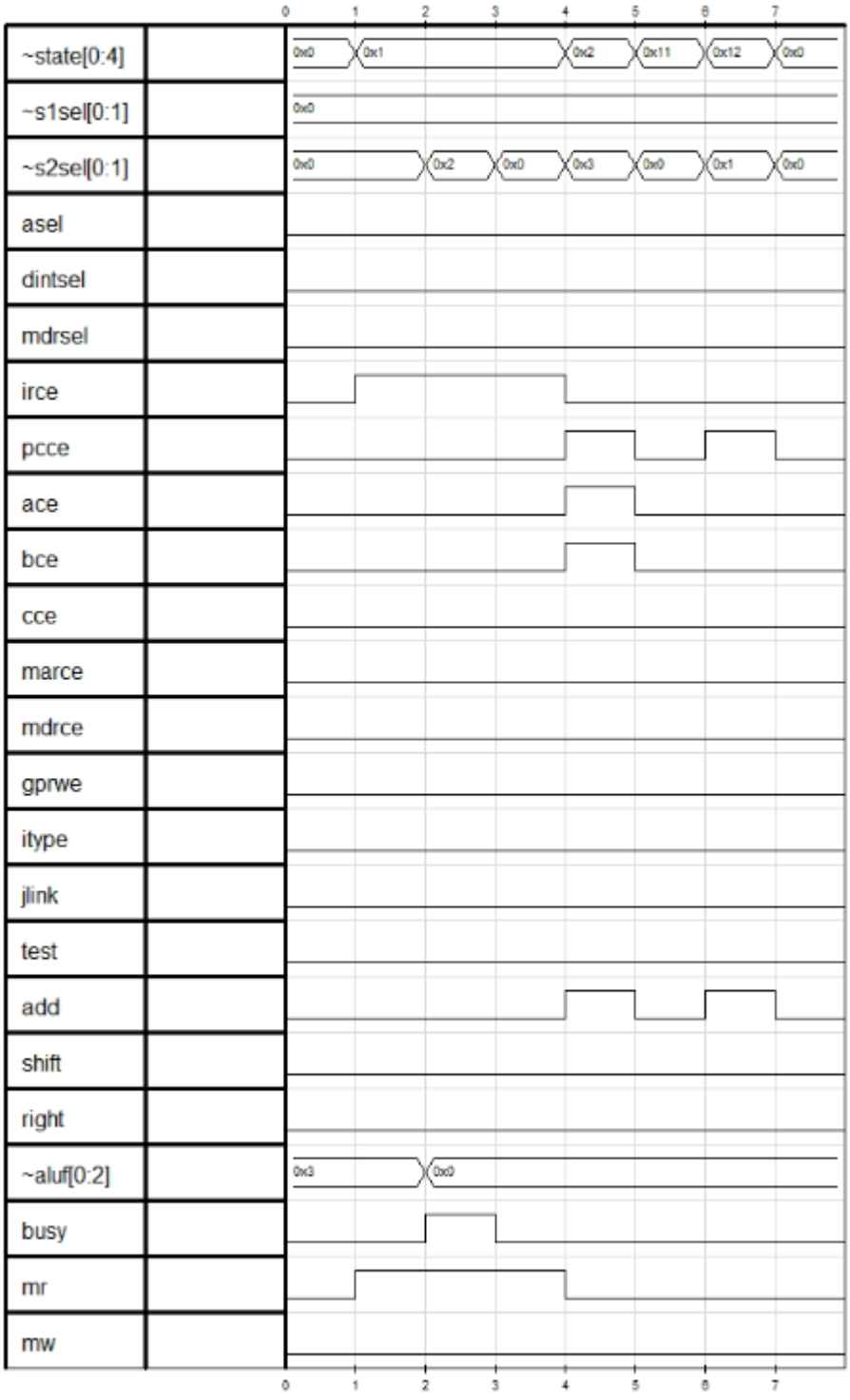
Before step enable:



After step enable:

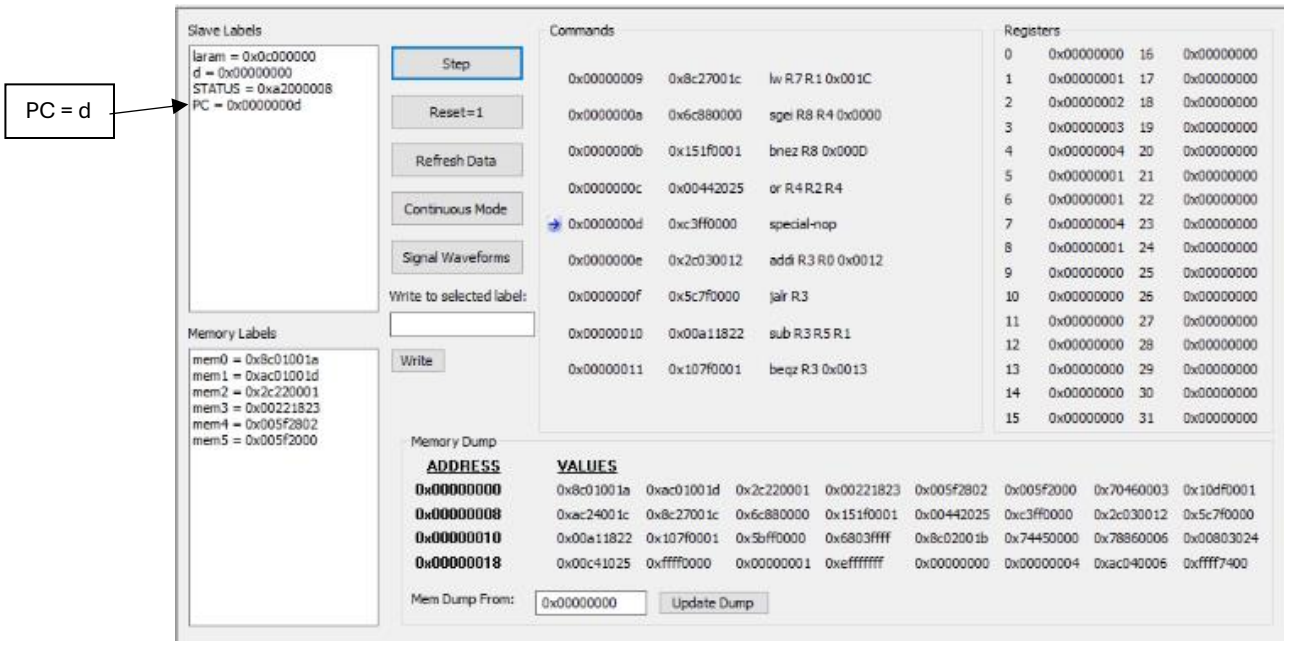


Waveforms:

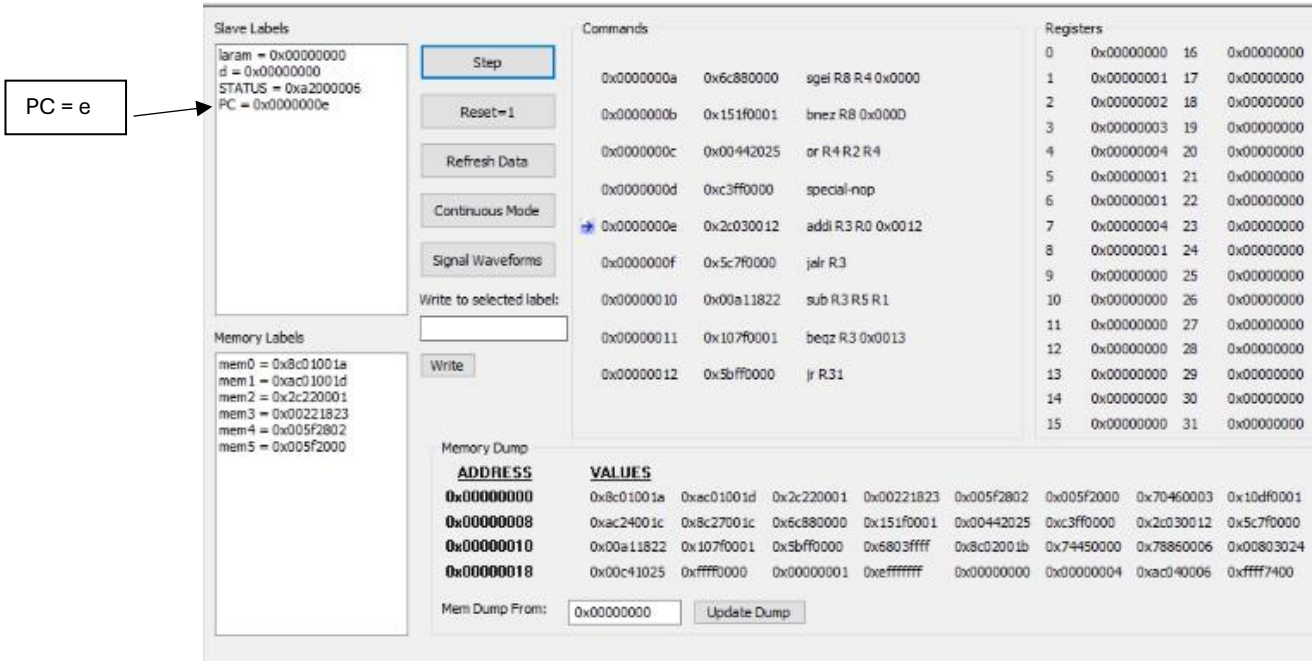

0
1
2
3
4
5
6
7
~state[0:4]
0x0
0x1
0x2
0x11
0x12
0x0
~s1sel[0:1]
0x0
~s2sel[0:1]
0x0
0x2
0x0
0x3
0x0
0x1
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x3
0x0
busy
mr
mw

PC = 0xd
Instruction: special-nop
Formula: -
PC = PC + 1 = e

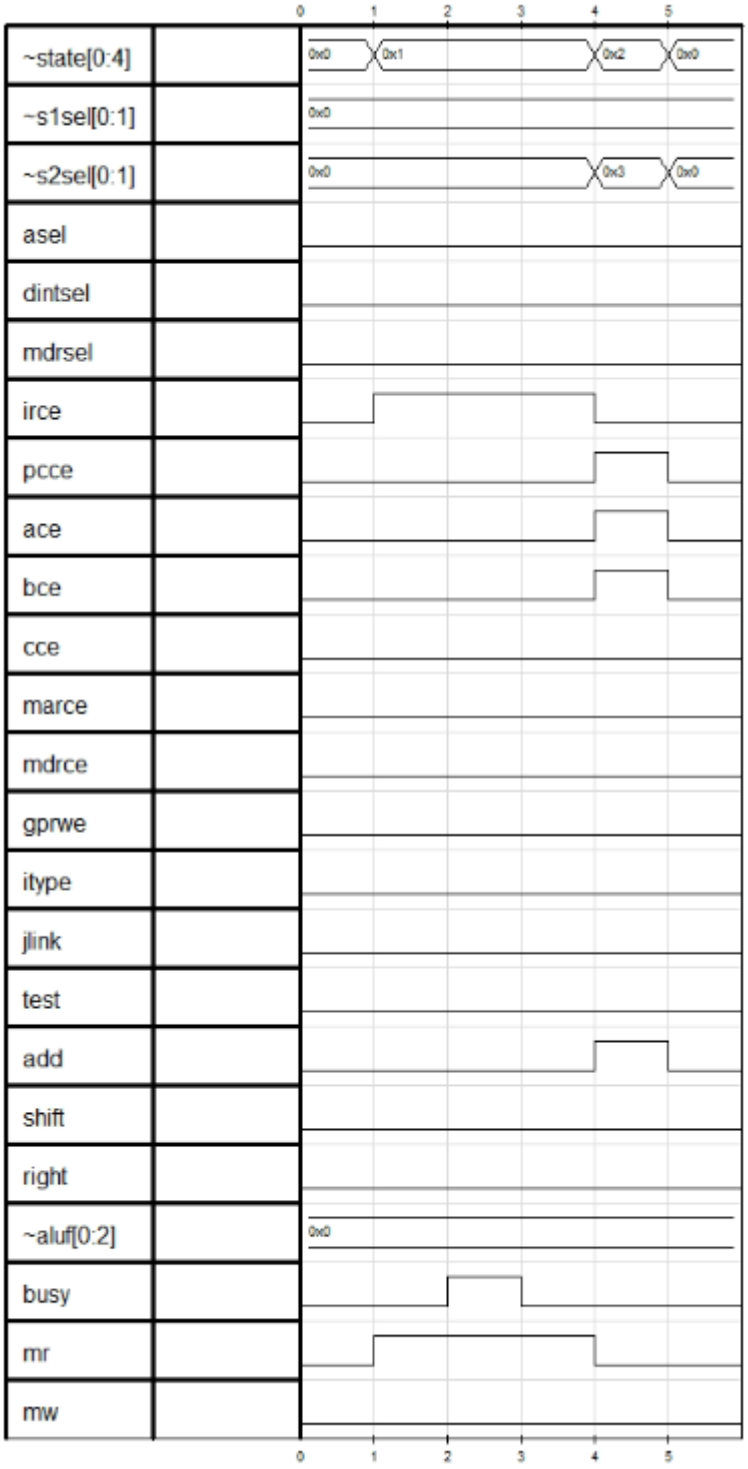
Before step enable:



After step enable:

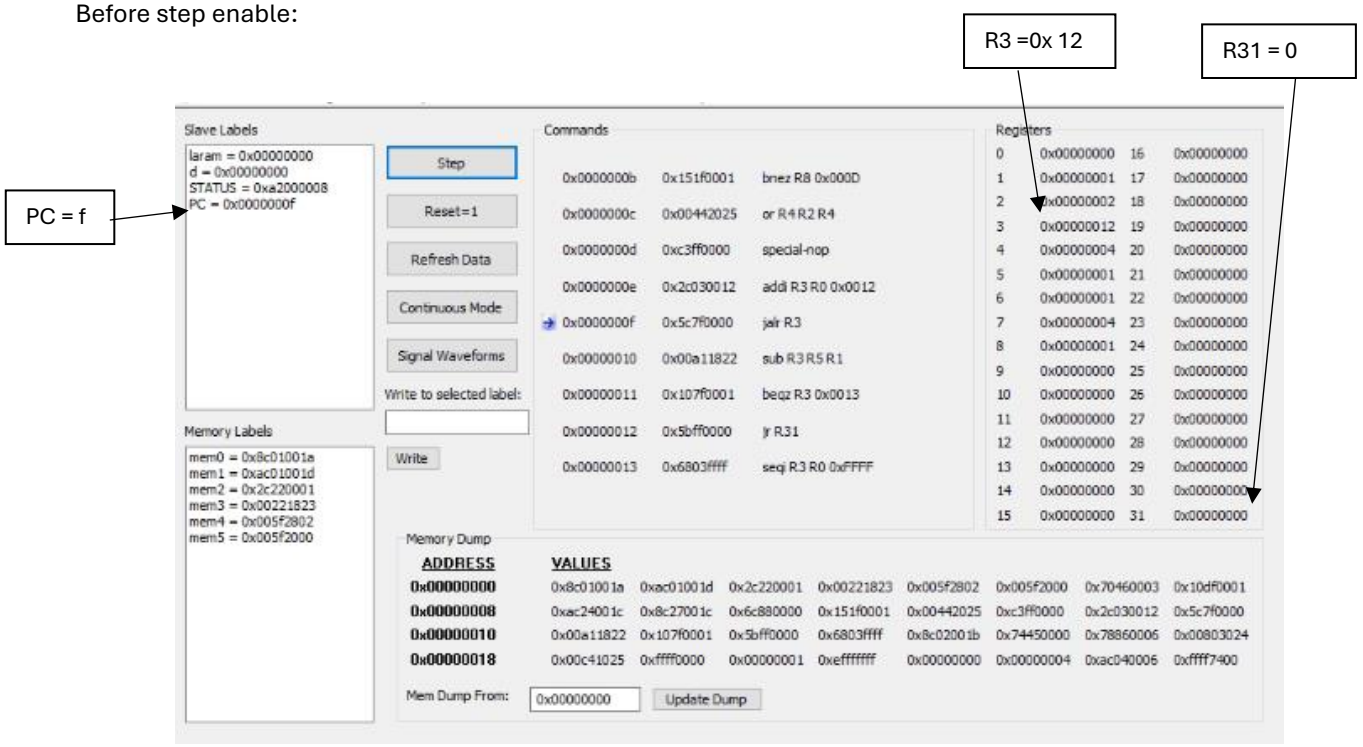


Waveforms:

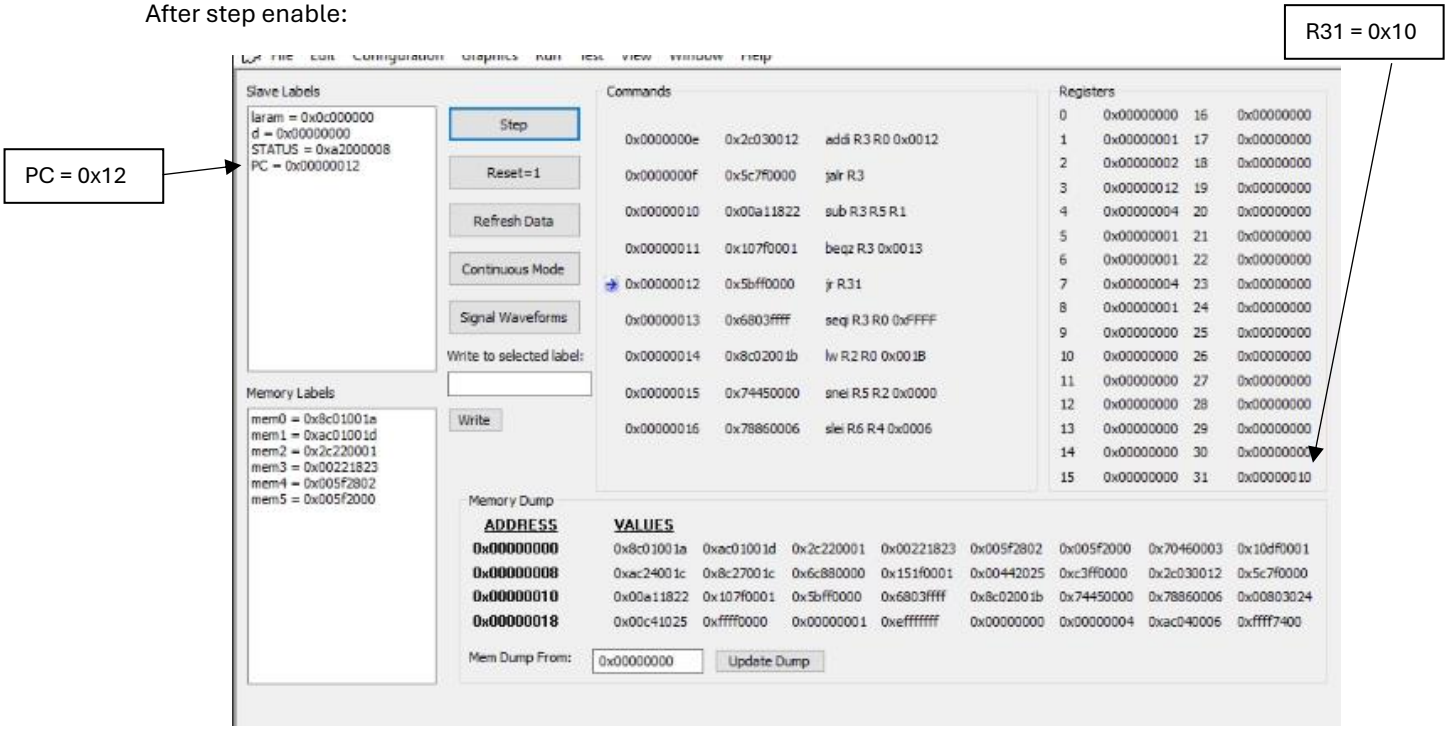

0
1
2
3
4
5
~state[0:4]
0x0
0x1
0x2
0x0
~s1sel[0:1]
0x0
~s2sel[0:1]
0x0
0x3
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x0
busy
mr
mw
0
1
2
3
4
5

PC = 0xf
Instruction: jalr R3
Formula: R31 = PC + 1 = 0x10
PC = R3 = 0x12

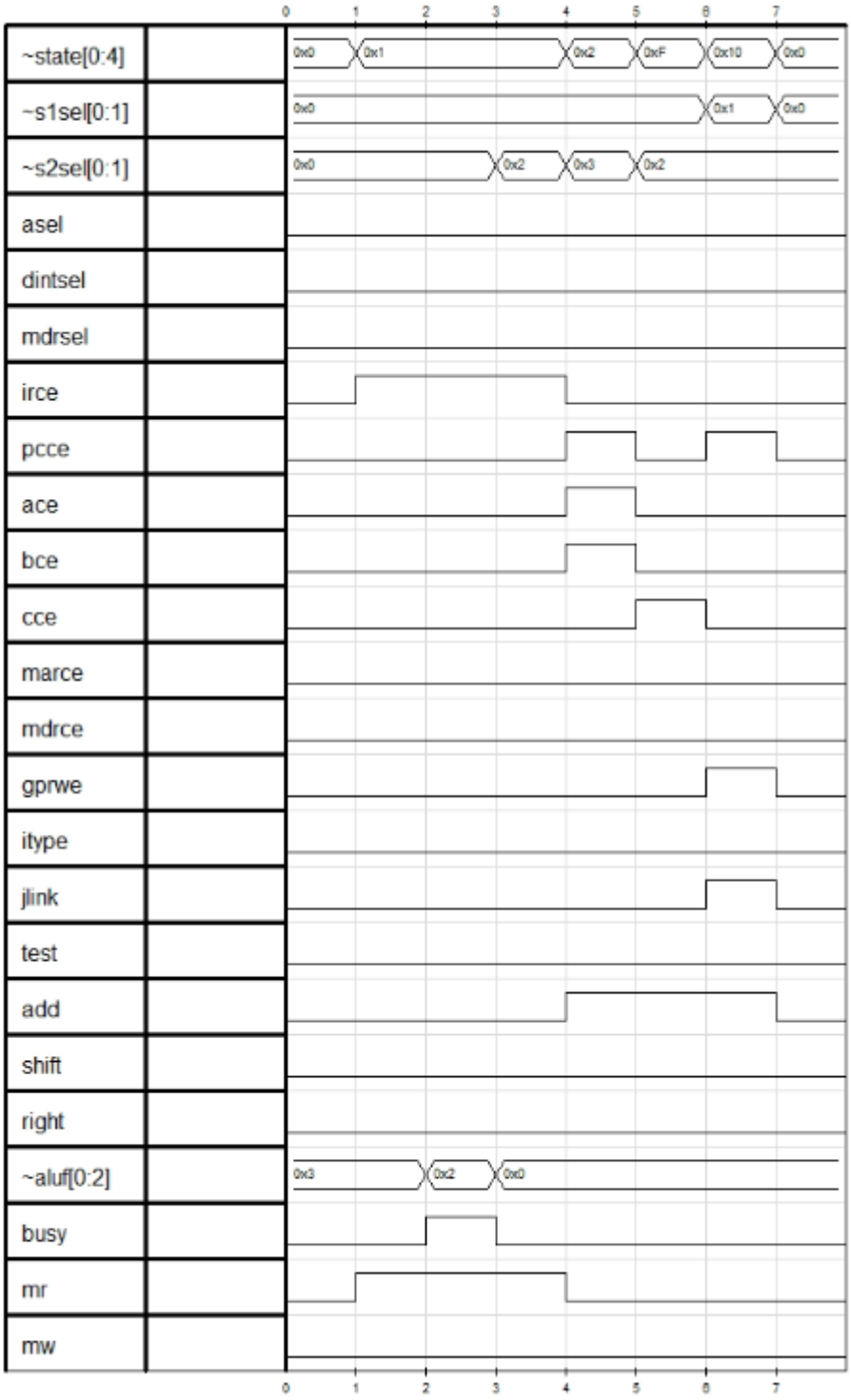
Before step enable:



After step enable:

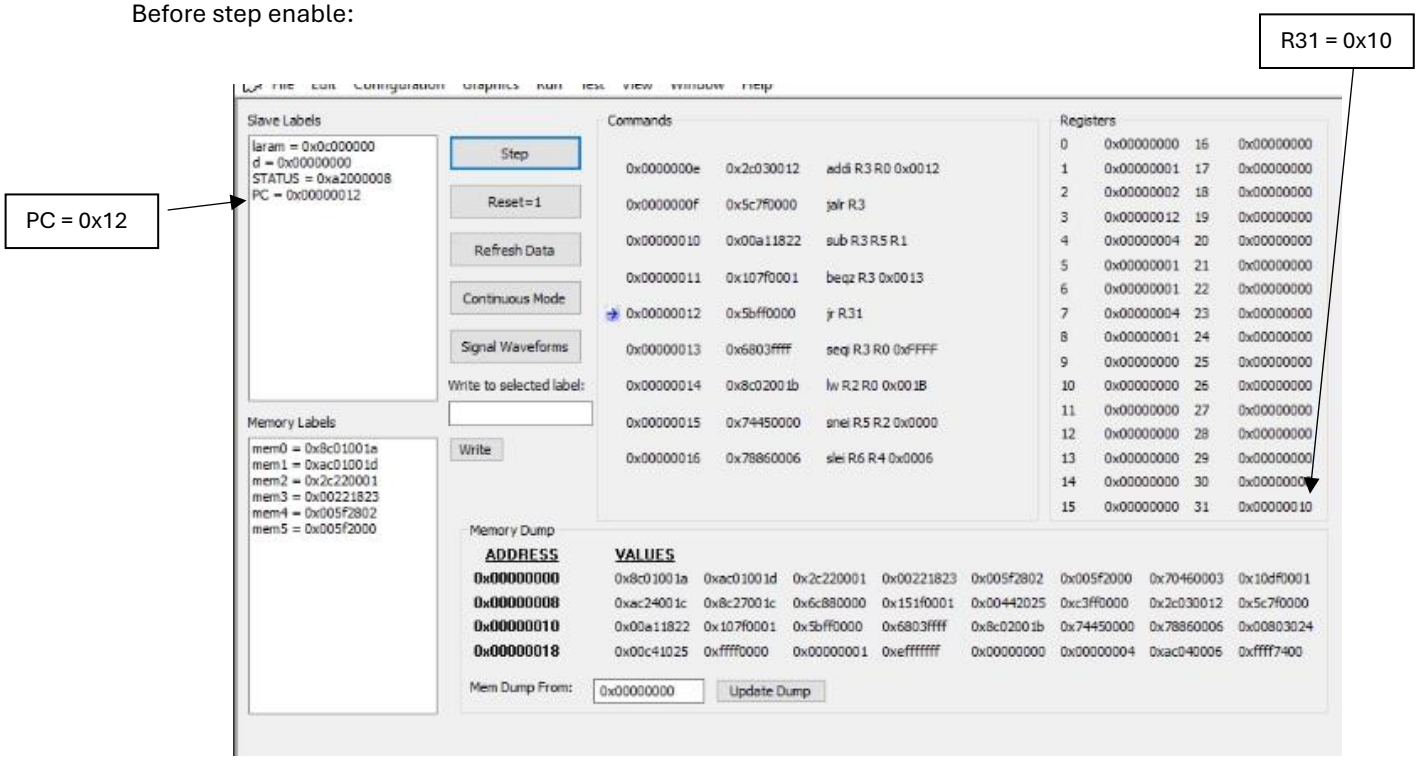


Waveforms:

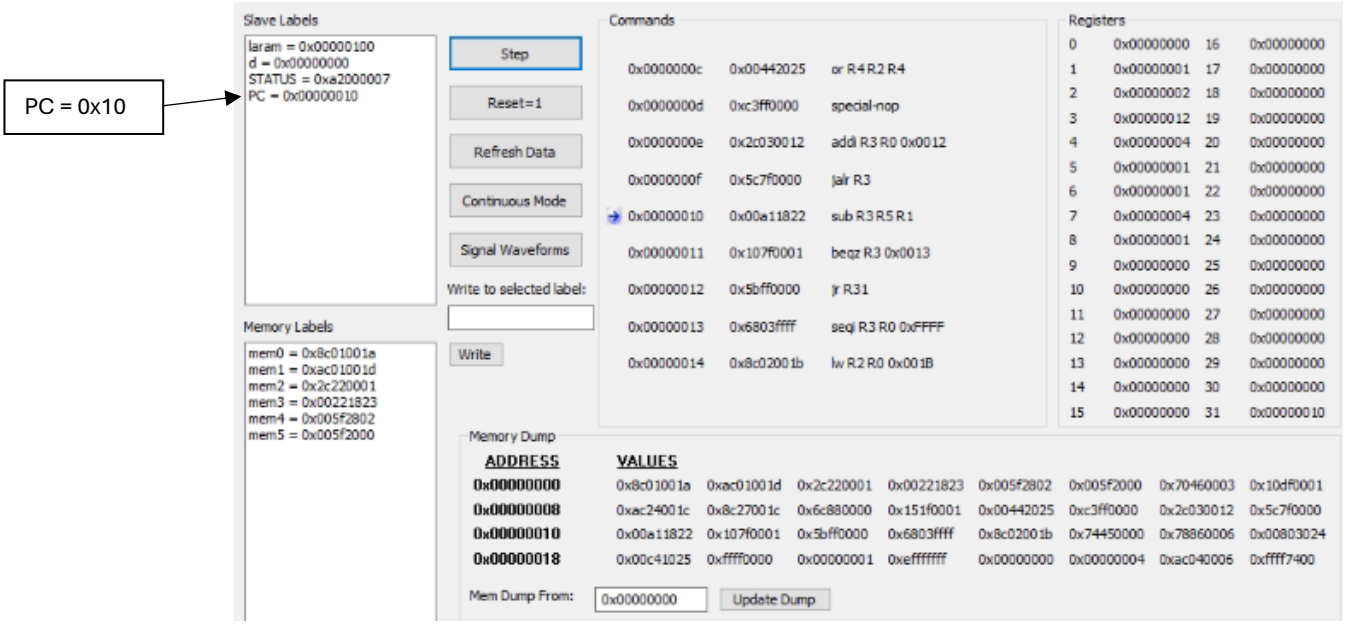

0
1
2
3
4
5
6
7
~state[0:4]
0x0
0x1
0x2
0xF
0x10
0x0
~s1sel[0:1]
0x0
0x1
0x0
~s2sel[0:1]
0x0
0x2
0x3
0x2
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x3
0x2
0x0
busy
mr
mw

PC = 0x12
Instruction: jr R31
Formula: -
PC = R31 = 0x10

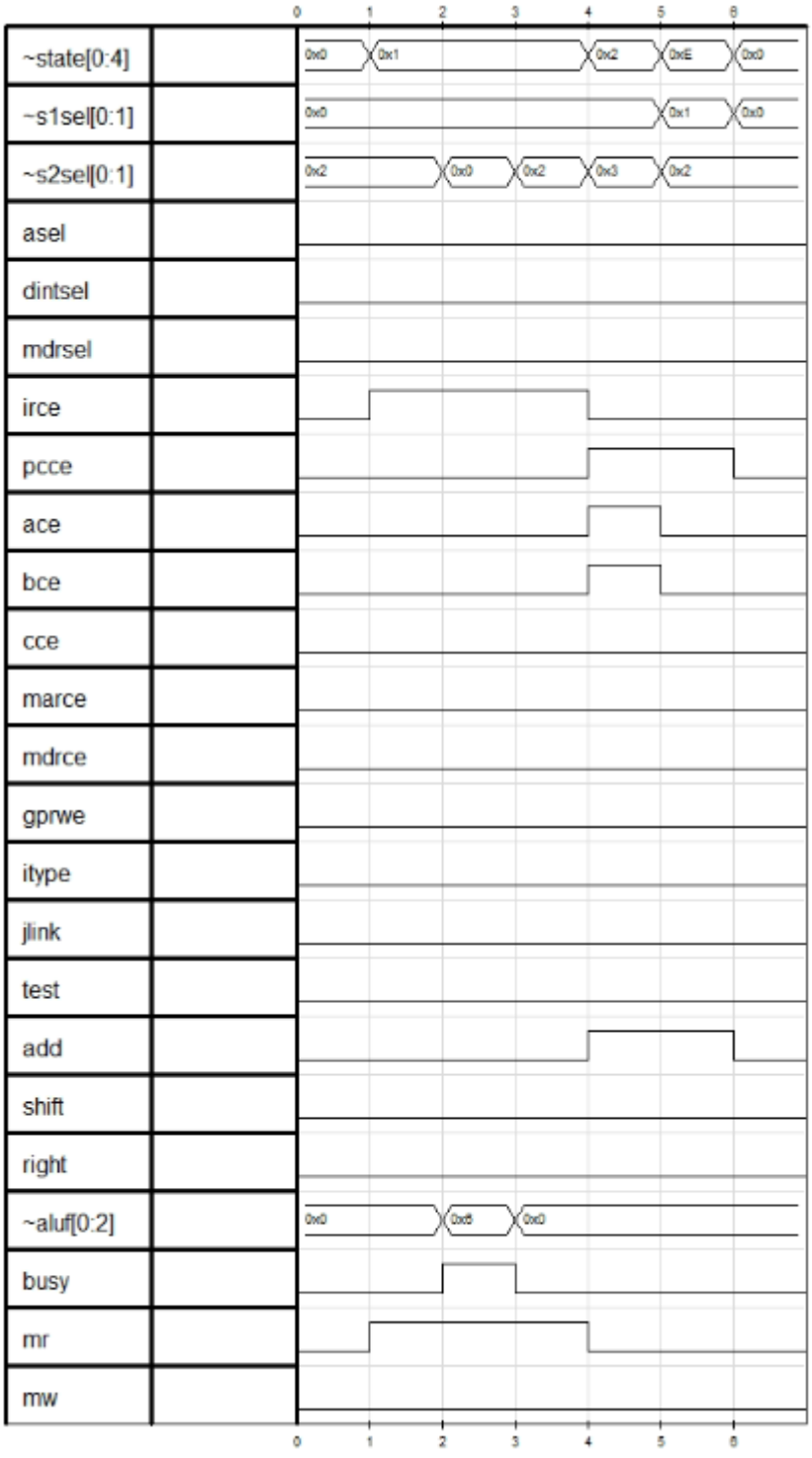
Before step enable:



After step enable:

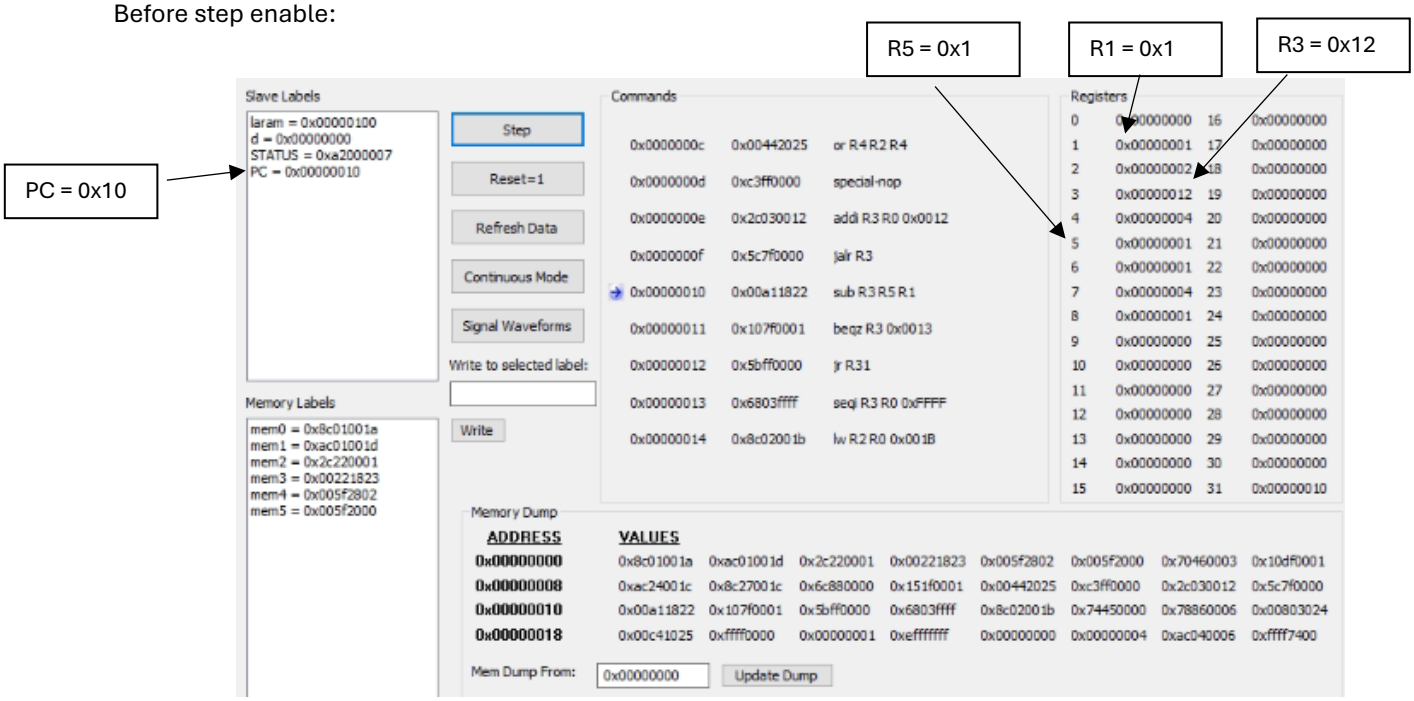


Waveforms:

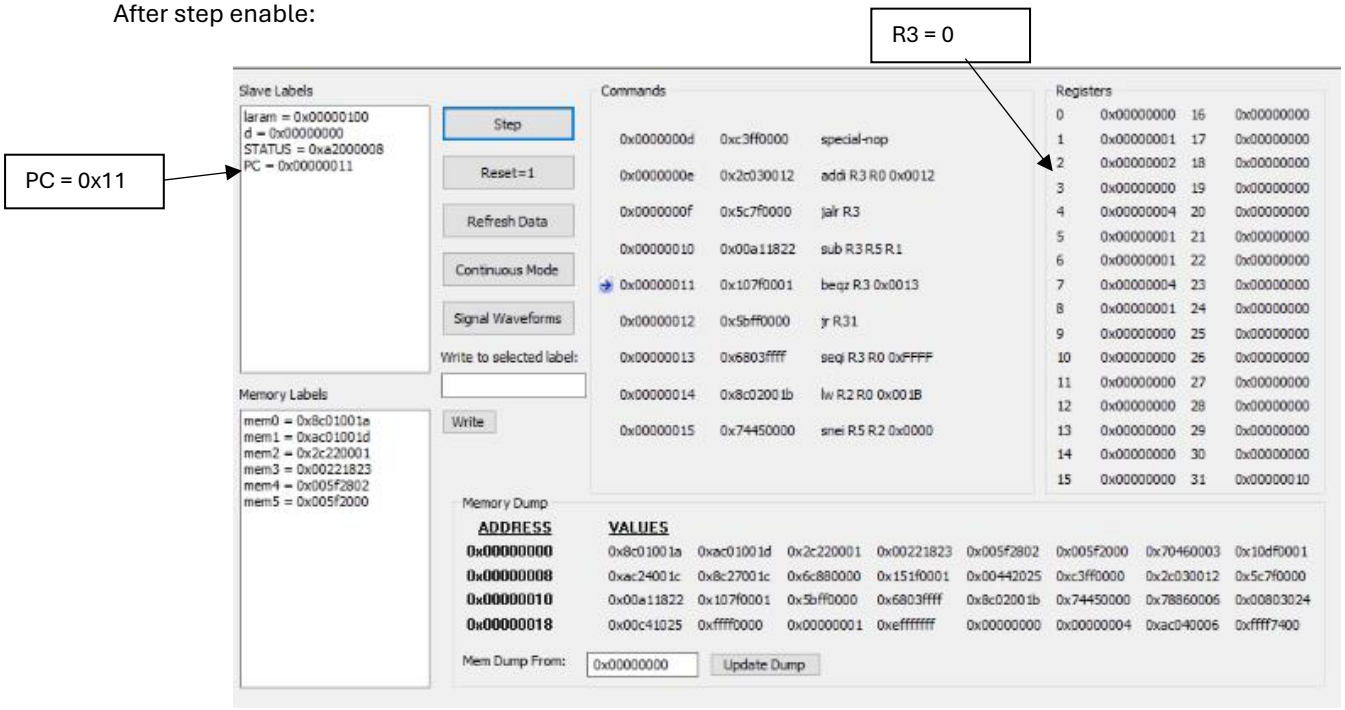

0
1
2
3
4
5
6
~state[0:4]
0x0
0x1
0x2
0xE
0x0
~s1sel[0:1]
0x0
0x1
0x0
~s2sel[0:1]
0x2
0x0
0x2
0x3
0x2
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x0
0x6
0x0
busy
mr
mw

PC = 0x10
Instruction: sub R3 R5 R1
Formula: R3 = R5 – R1 = 0
PC = PC + 1 = 0x11

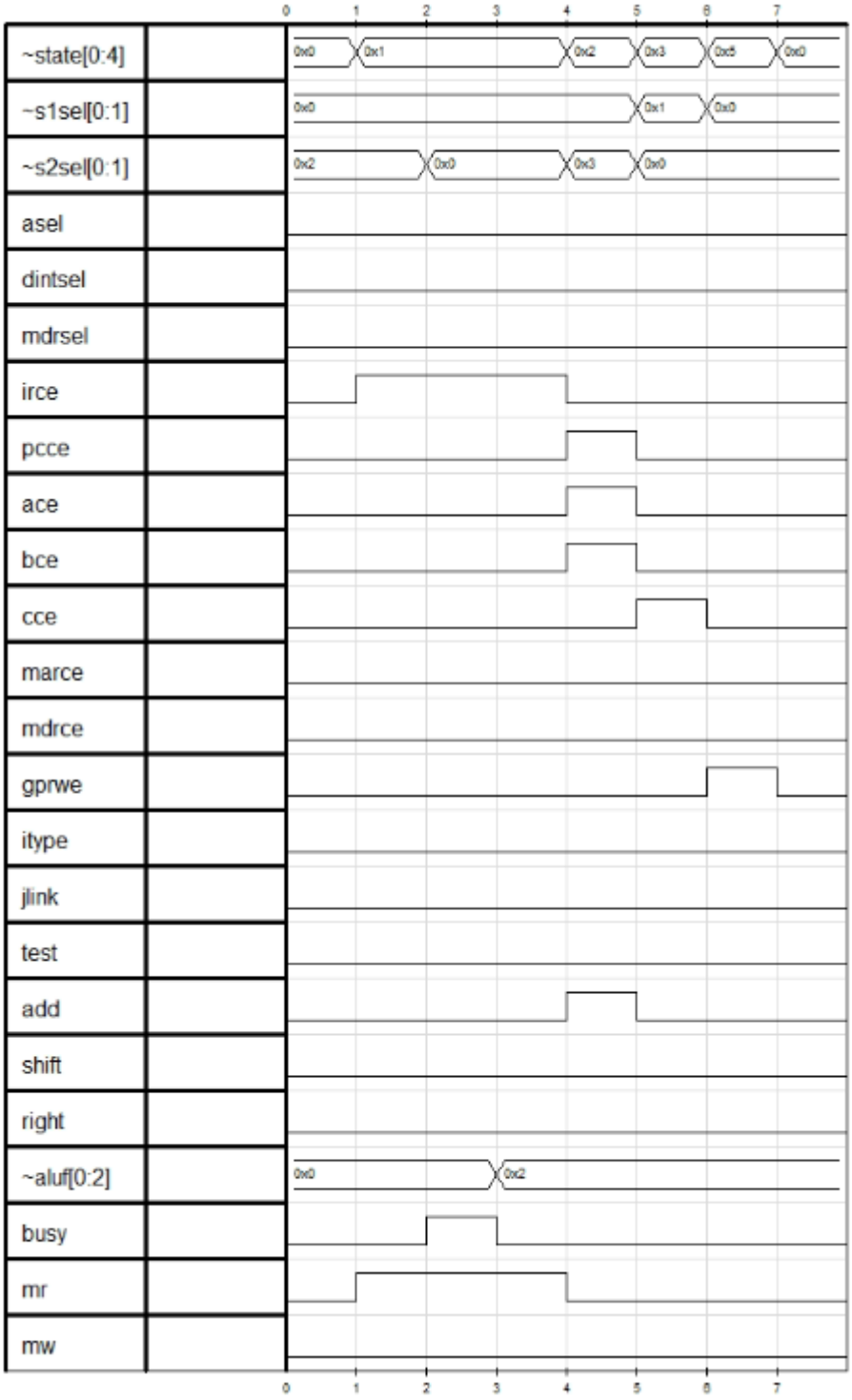
Before step enable:



After step enable:

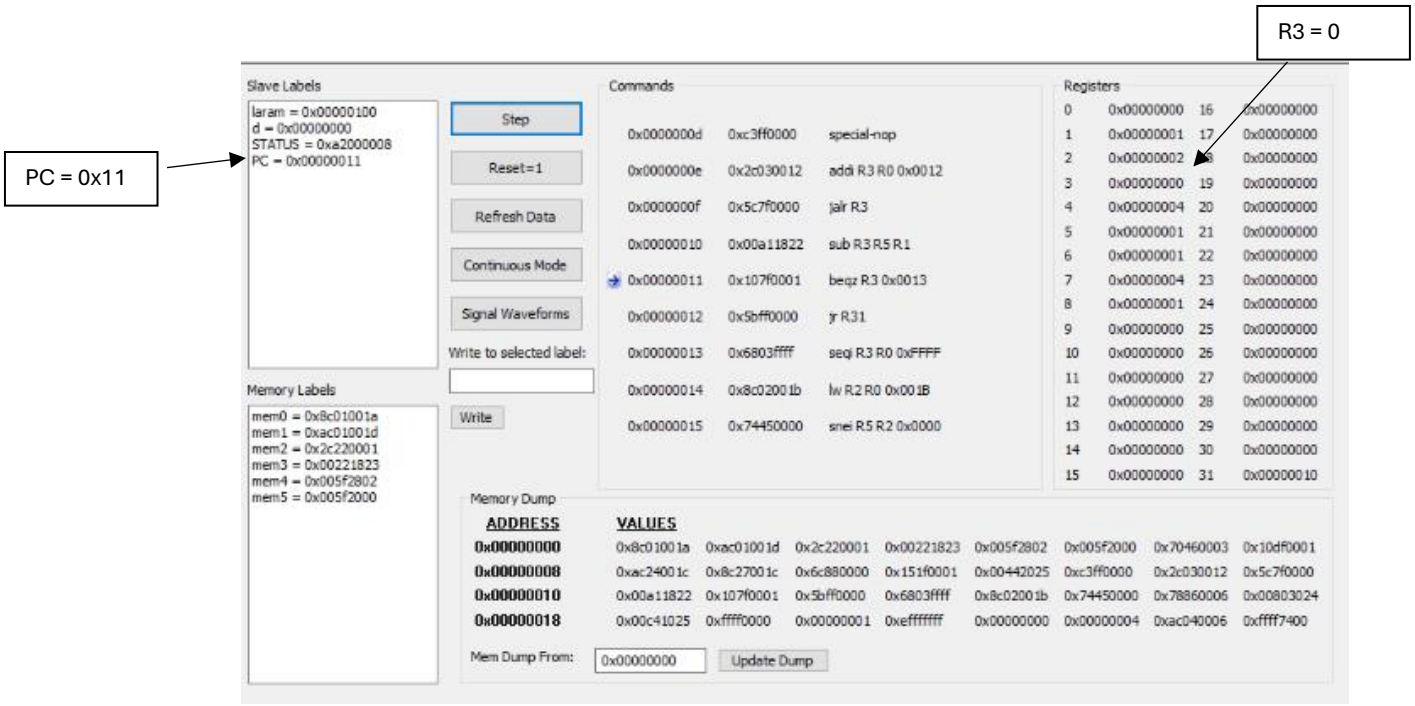


Waveforms:

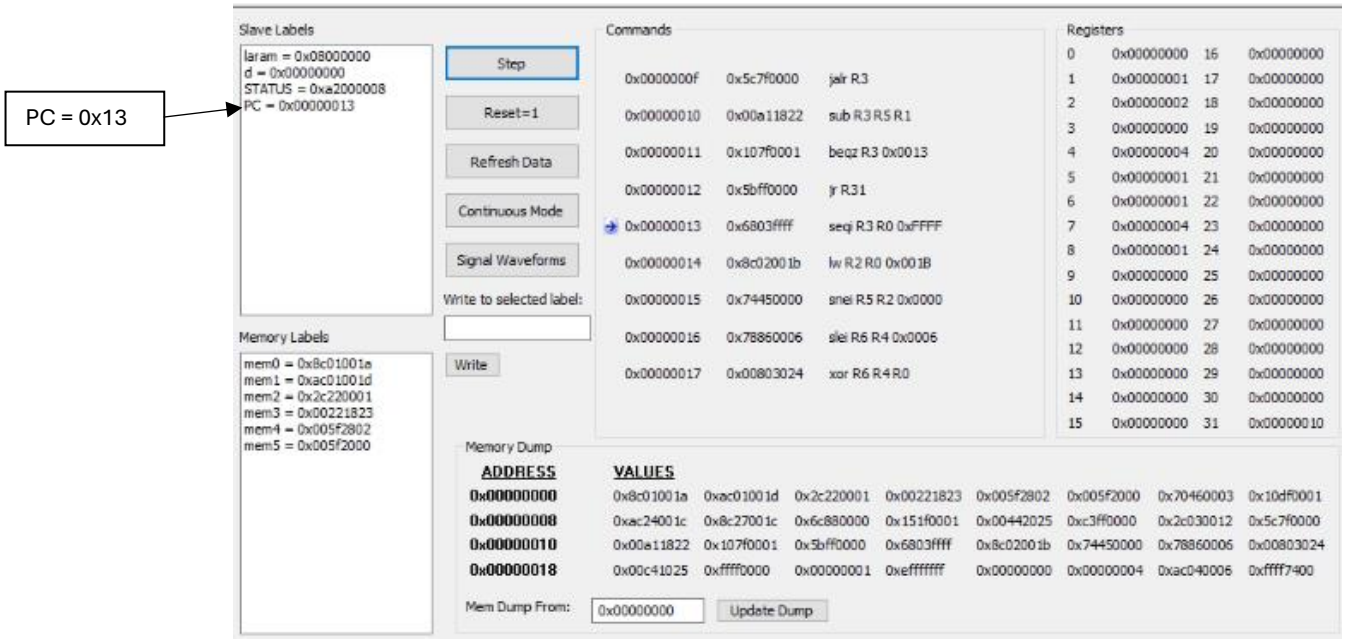

0
1
2
3
4
5
6
7
~state[0:4]
0x0
0x1
0x2
0x3
0x5
0x0
~s1sel[0:1]
0x0
0x1
0x0
~s2sel[0:1]
0x2
0x0
0x3
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x0
0x2
busy
mr
mw

PC = 0x11
Instruction: beqz R3 0x0013
Formula: PC = (!R3) ? 0x0013 : (PC + 1)
PC = 0x0013

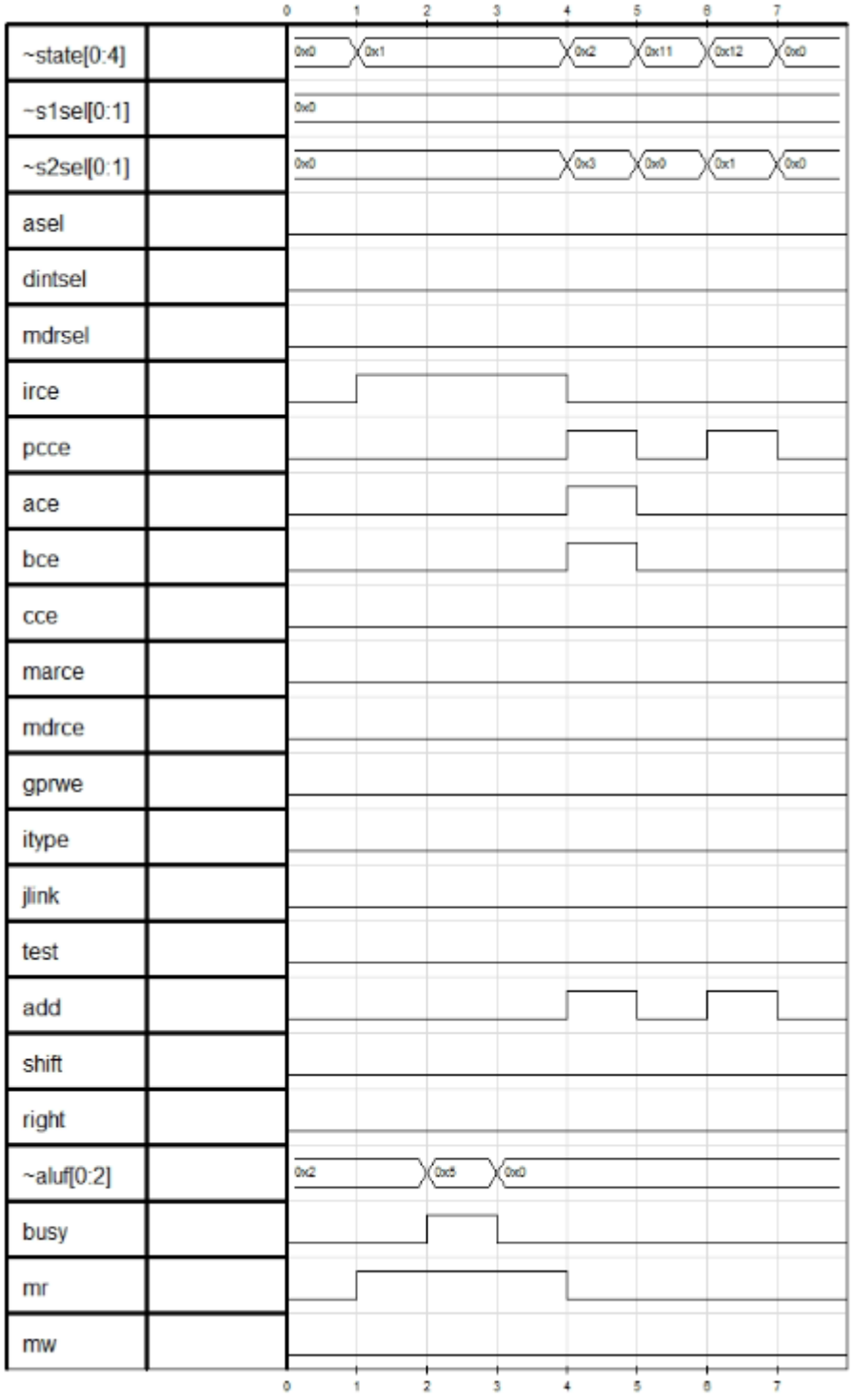
Before step enable:



After step enable:

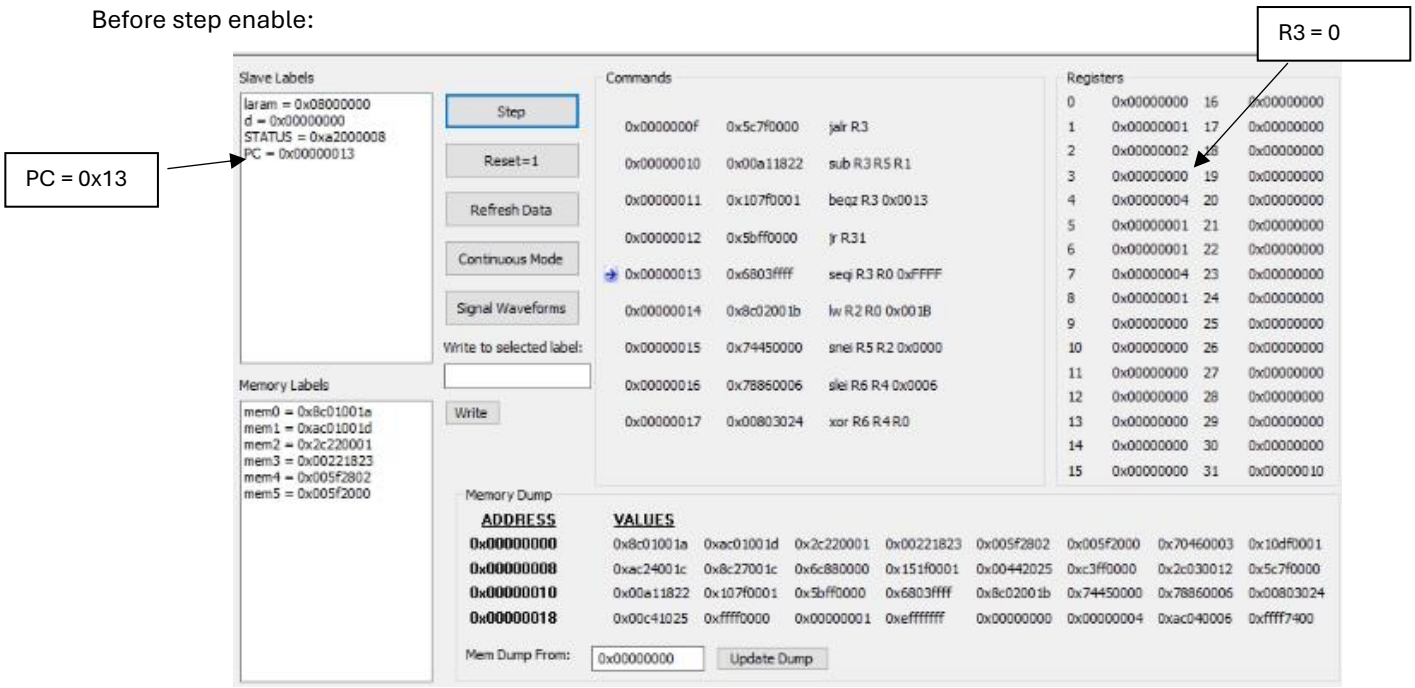


Waveforms:

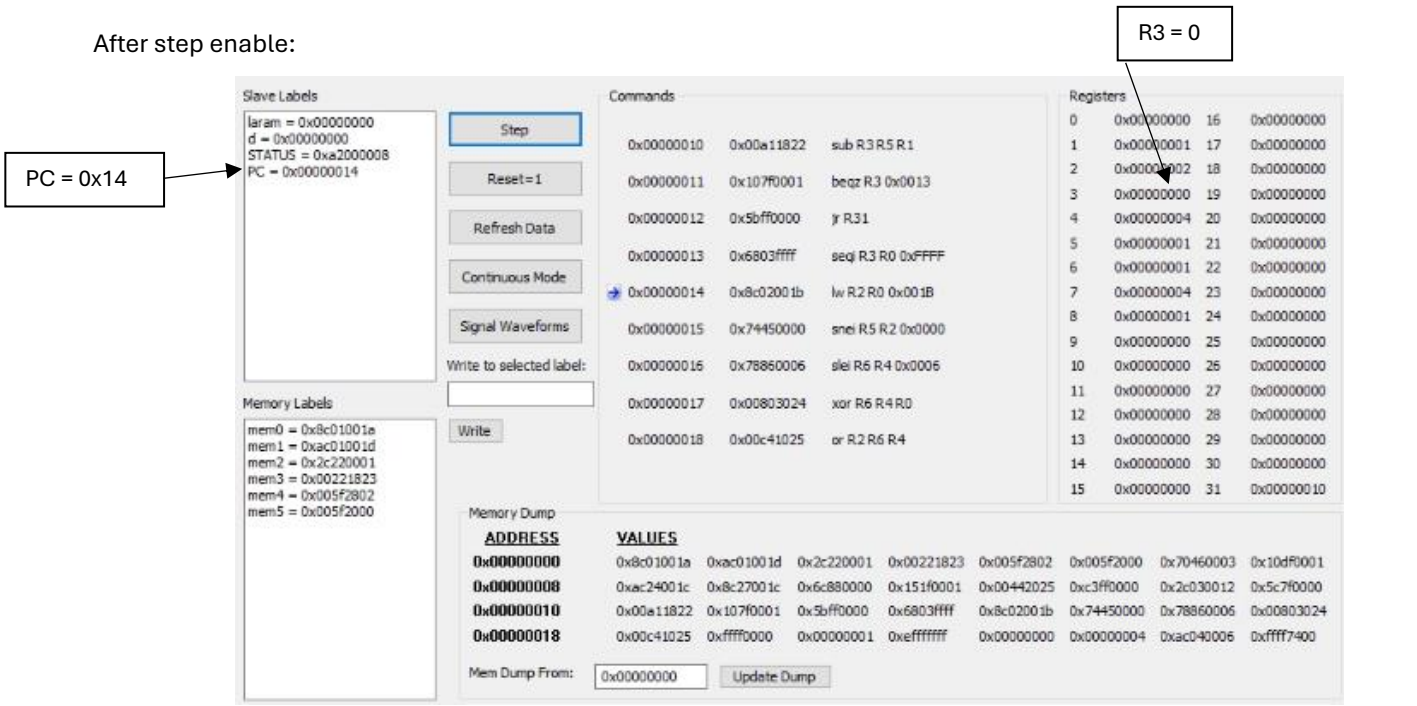

0
1
2
3
4
5
6
7
~state[0:4]
0x0
0x1
0x2
0x11
0x12
0x0
~s1sel[0:1]
0x0
~s2sel[0:1]
0x0
0x3
0x0
0x1
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x2
0x5
0x0
busy
mr
mw

PC = 0x13
Instruction: seqi R3 R0 0xFFFF
Formula: R3 = (R0 == 0xFFFF) ? 1 : 0 => R3 = 0
PC = PC + 1 = 0x14

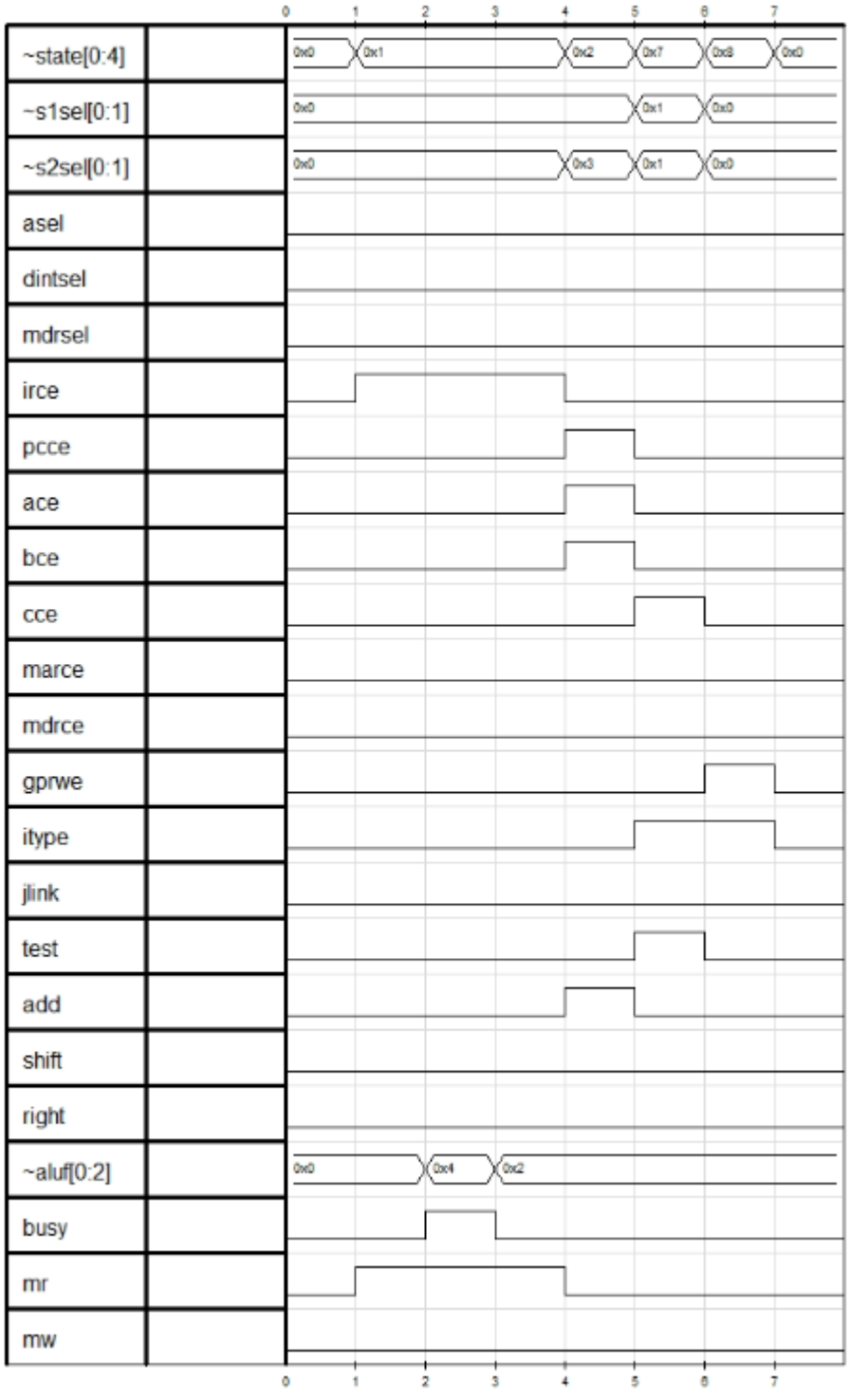
Before step enable:



After step enable:

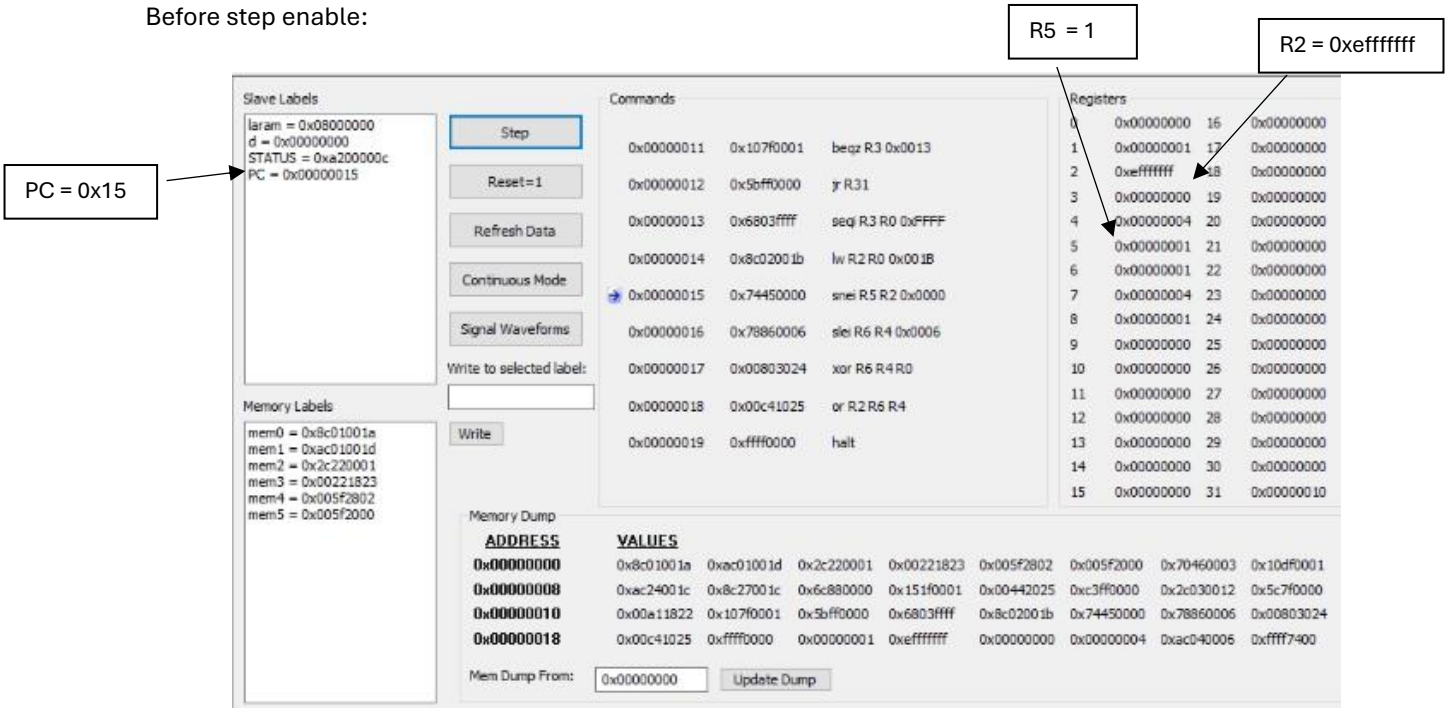


Waveforms:

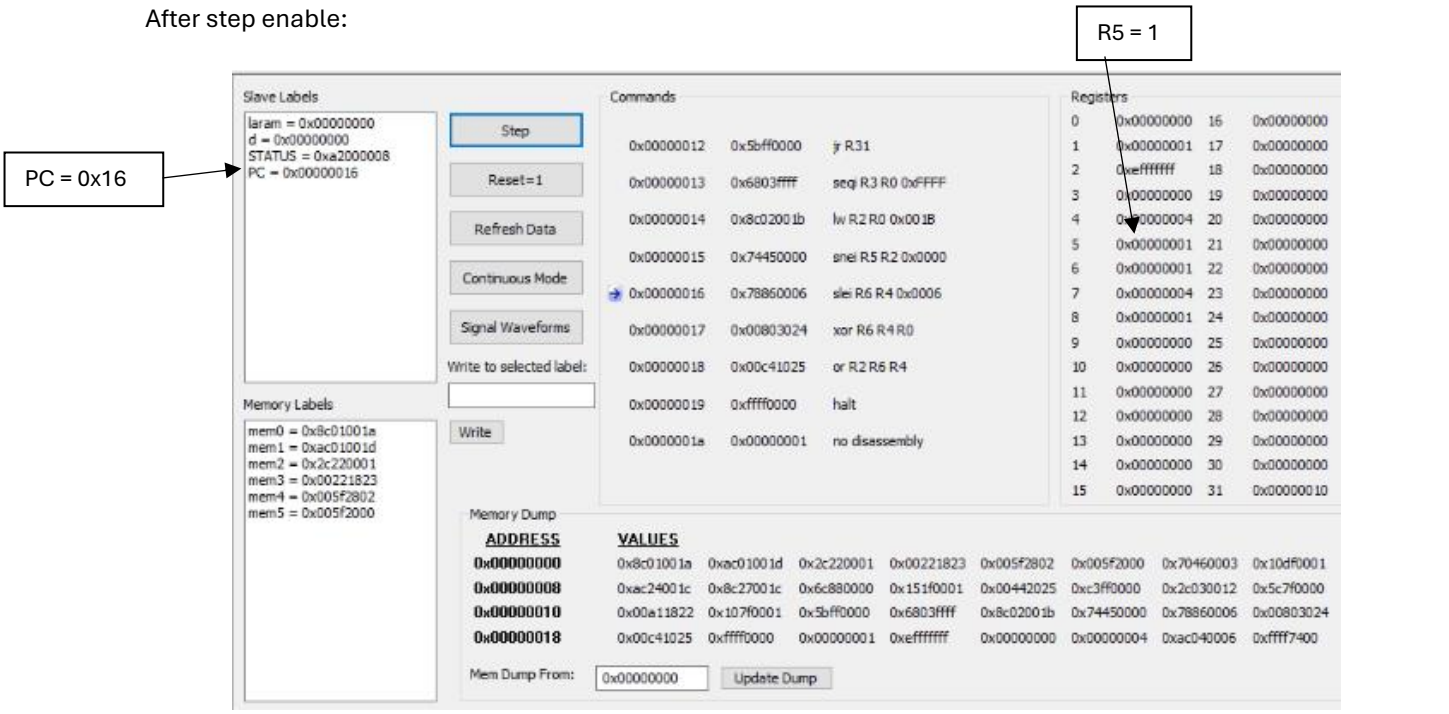

0
1
2
3
4
5
6
7
~state[0:4]
0x0
0x1
0x2
0x7
0x8
0x0
~s1sel[0:1]
0x0
0x1
0x0
~s2sel[0:1]
0x0
0x3
0x1
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x0
0x4
0x2
busy
mr
mw

PC = 0x15
Instruction: snei R5 R2 0x0000
Formula: R5 = (R2 != 0x0000) ? 1 : 0 => R5 = 1
PC = PC + 1 = 0x16

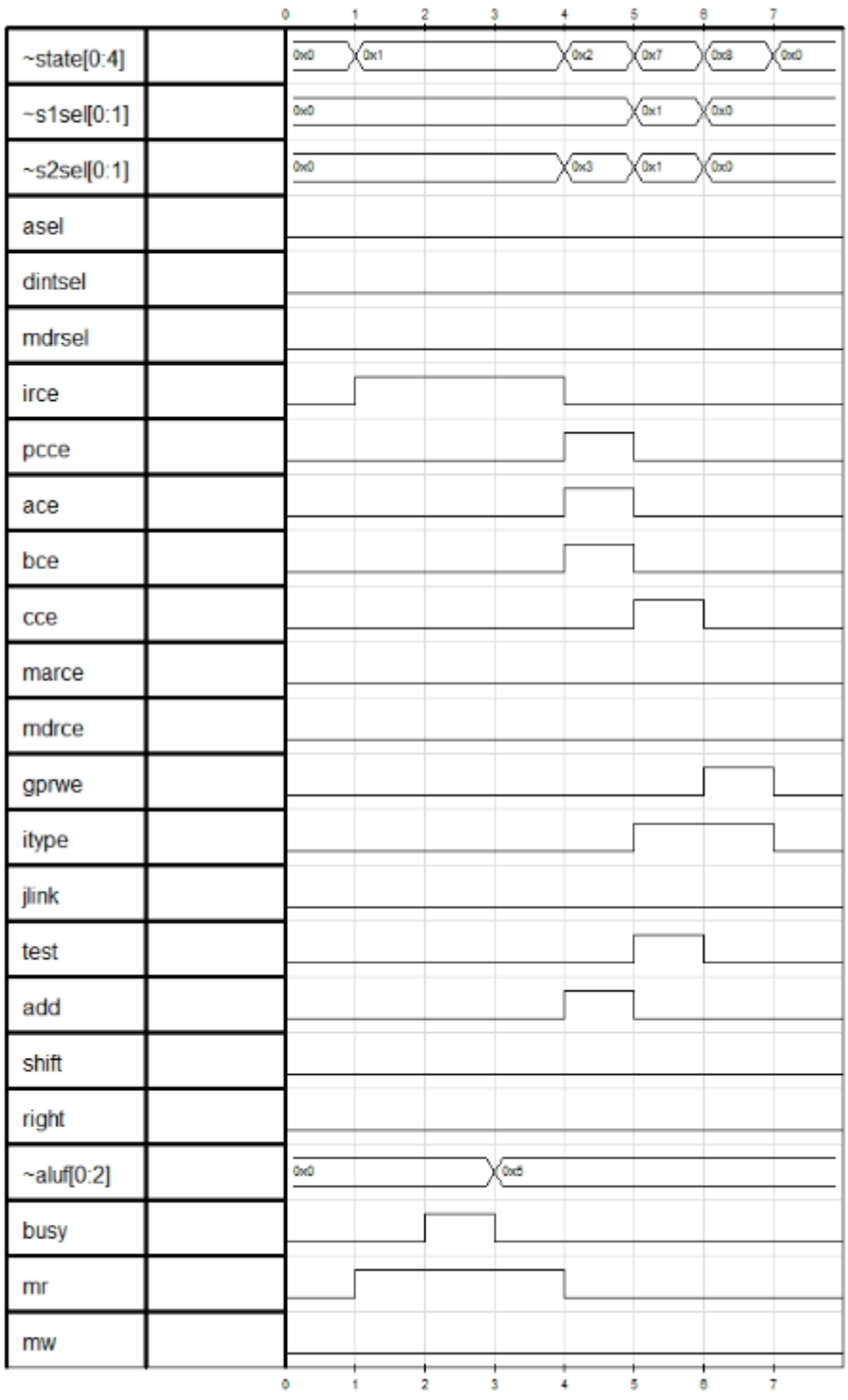
Before step enable:



After step enable:



Waveforms:


0
1
2
3
4
5
6
7
~state[0:4]
0x0
0x1
0x2
0x7
0x8
0x0
~s1sel[0:1]
0x0
0x1
0x0
~s2sel[0:1]
0x0
0x3
0x1
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x0
0x5
busy
mr
mw

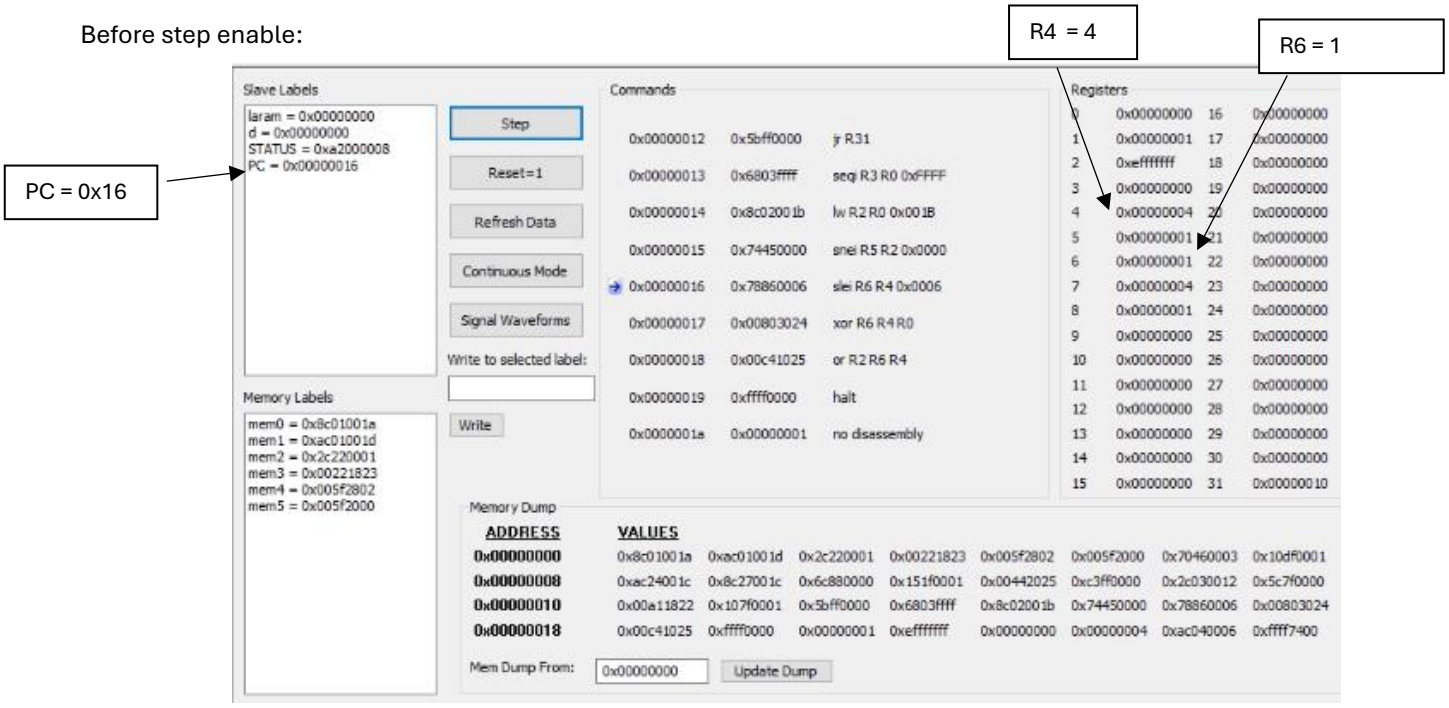
PC = 0x16

Instruction: slei R6 R4 0x0006

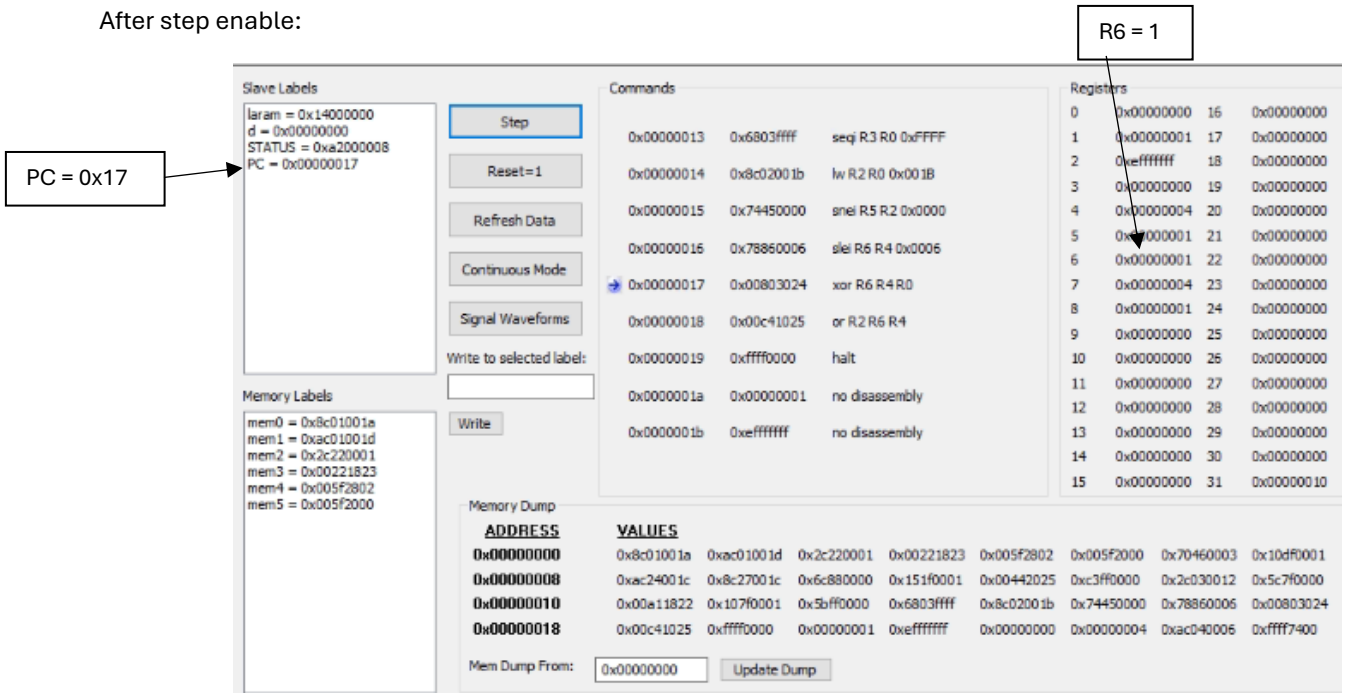
Formula: R6 = (R4 <= 0x0006) ? 1 : 0 => R6 = 1

PC = PC + 1 = 0x17

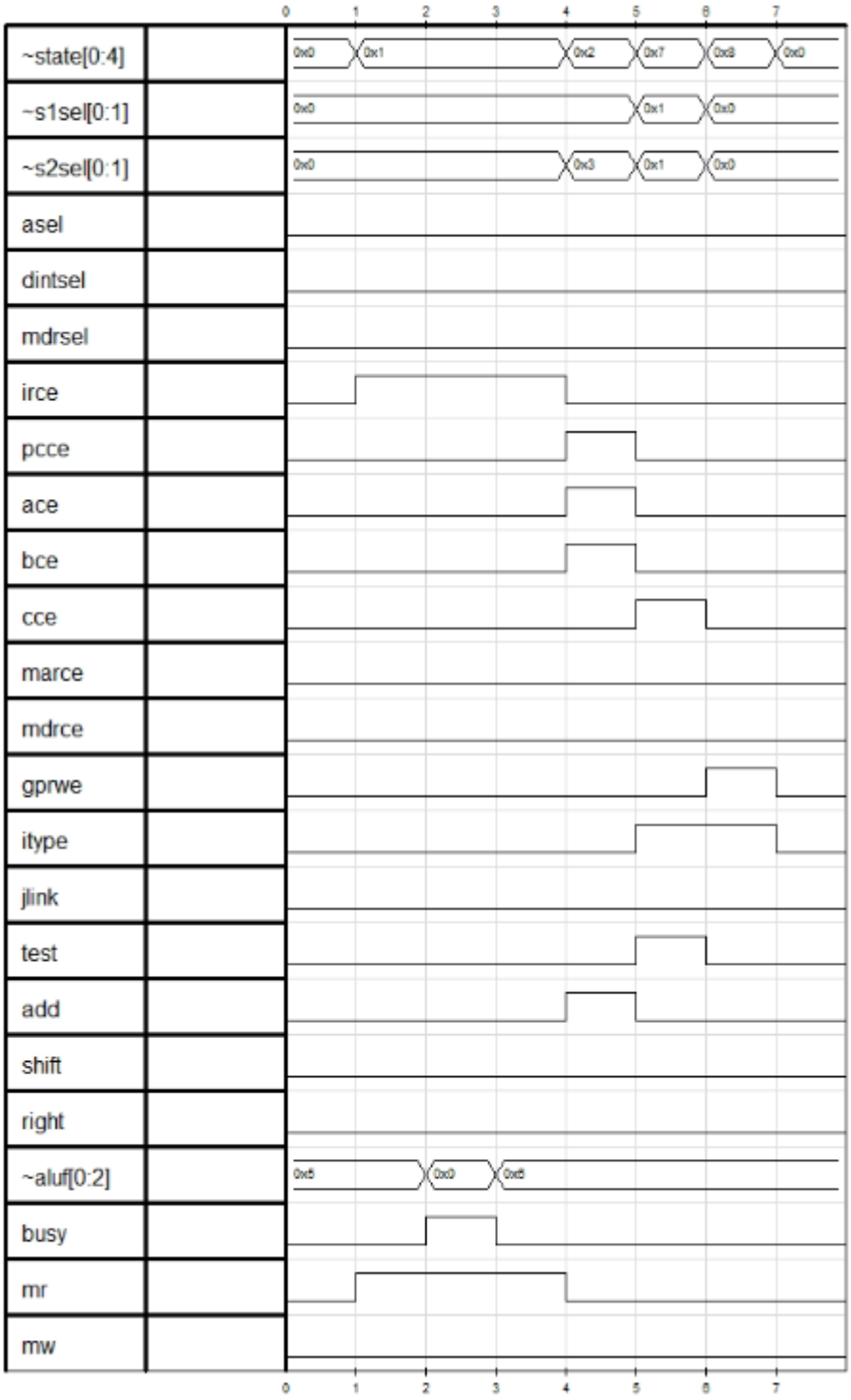
Before step enable:



After step enable:

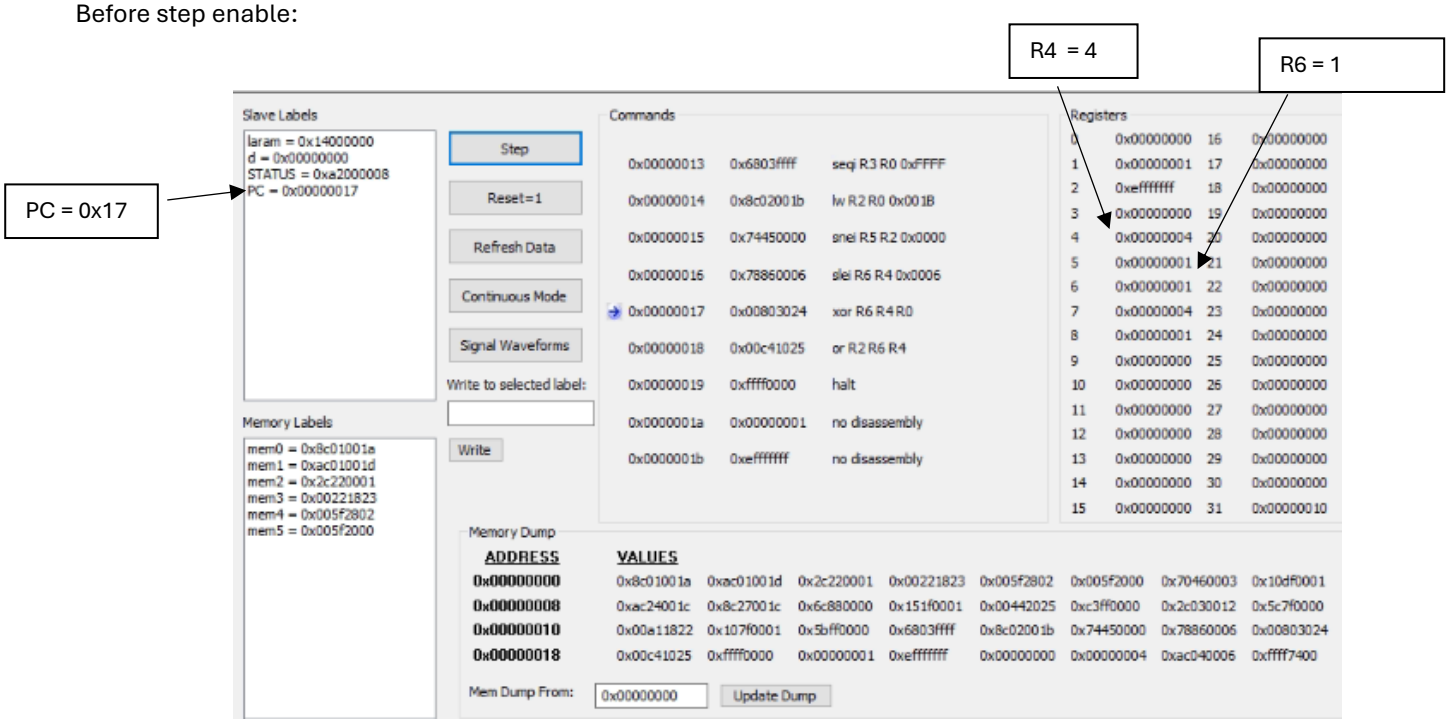


Waveforms:

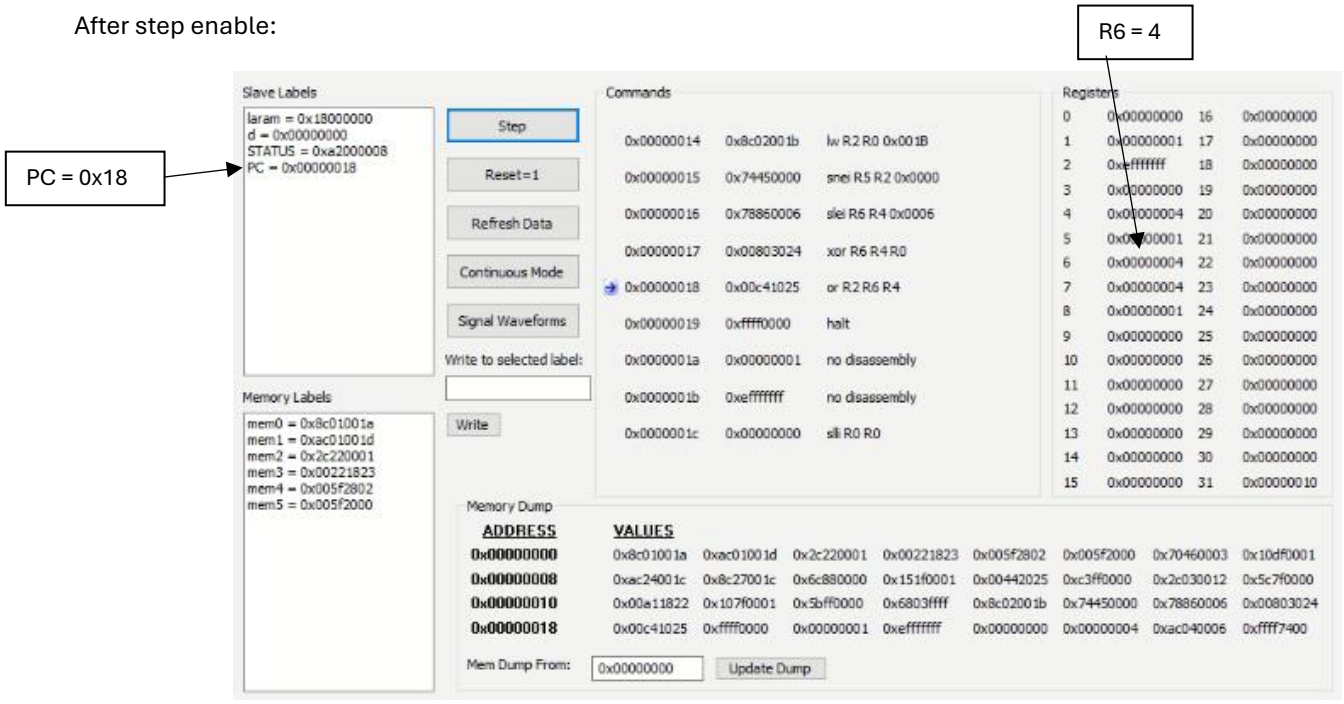

0
1
2
3
4
5
6
7
~state[0:4]
0x0
0x1
0x2
0x7
0x8
0x0
~s1sel[0:1]
0x0
0x1
0x0
~s2sel[0:1]
0x0
0x3
0x1
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x5
0x0
0x5
busy
mr
mw

PC = 0x17
Instruction: xor R6 R4 R0
Formula: R6 = R4 xor R0 = R4 = 4
PC = PC + 1 = 0x18

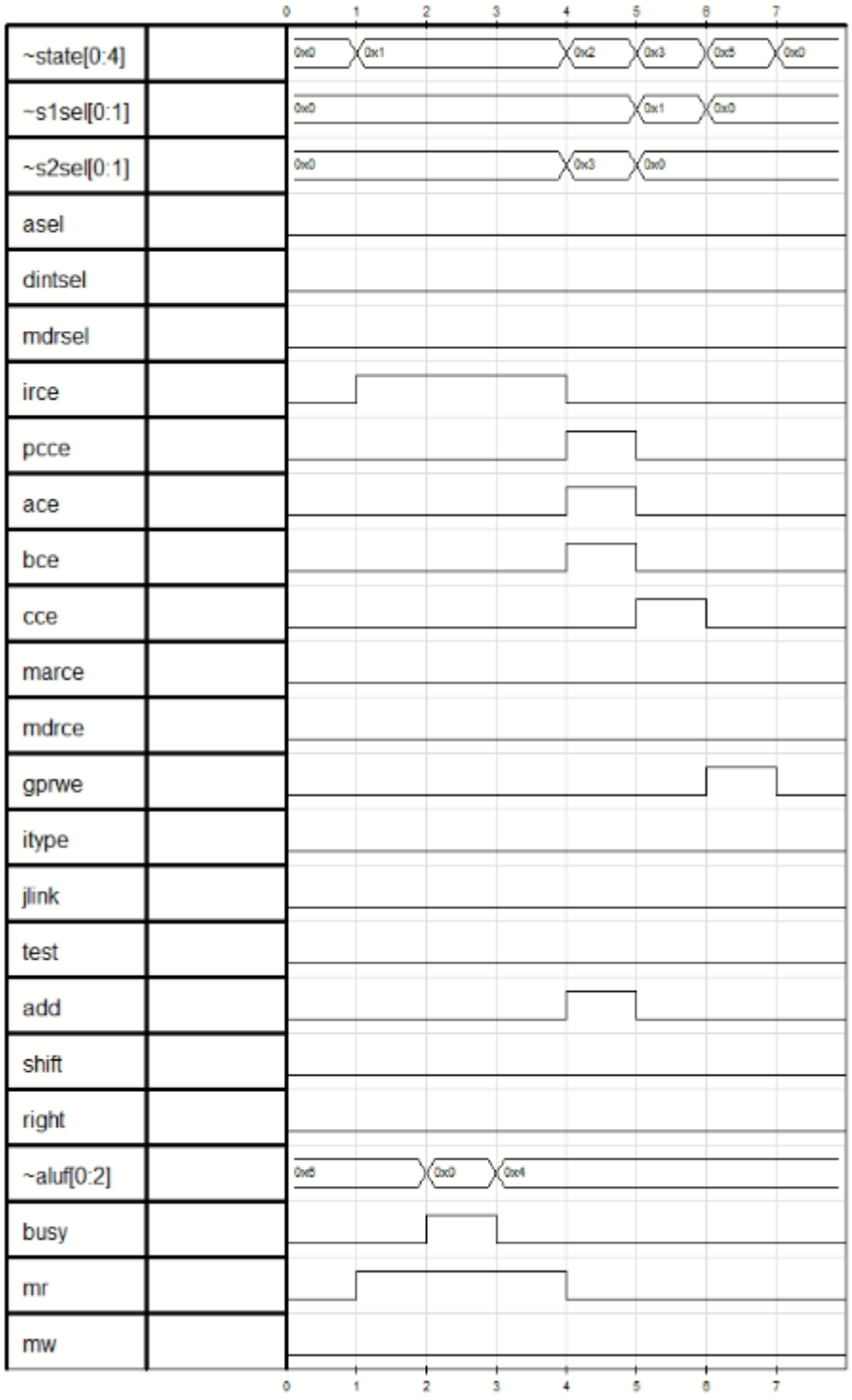
Before step enable:



After step enable:

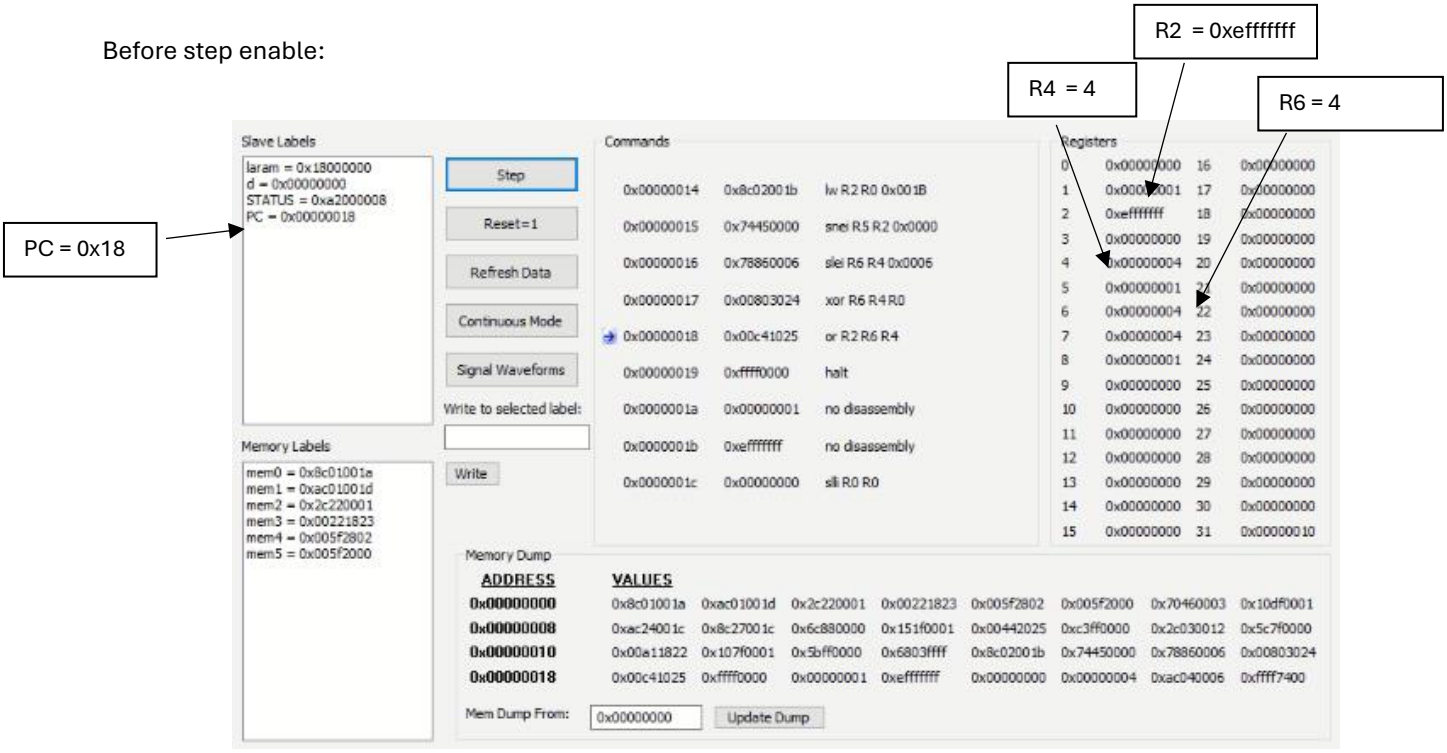


Waveforms:

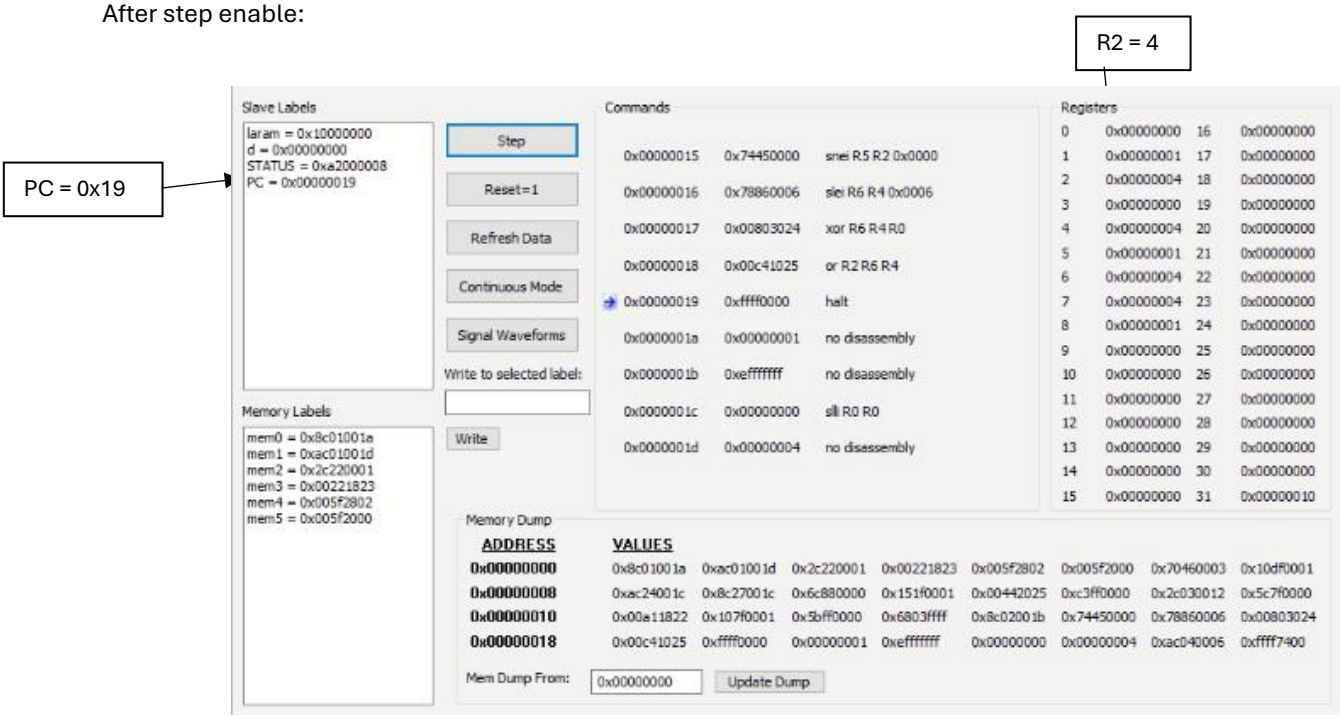

0
1
2
3
4
5
6
7
~state[0:4]
0x0
0x1
0x2
0x3
0x5
0x0
~s1sel[0:1]
0x0
0x1
0x0
~s2sel[0:1]
0x0
0x3
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x5
0x0
0x4
busy
mr
mw

PC = 0x18
Instruction: or R2 R6 R4
Formula: R2 = R6 or R4 = 4
PC = PC + 1 = 0x19

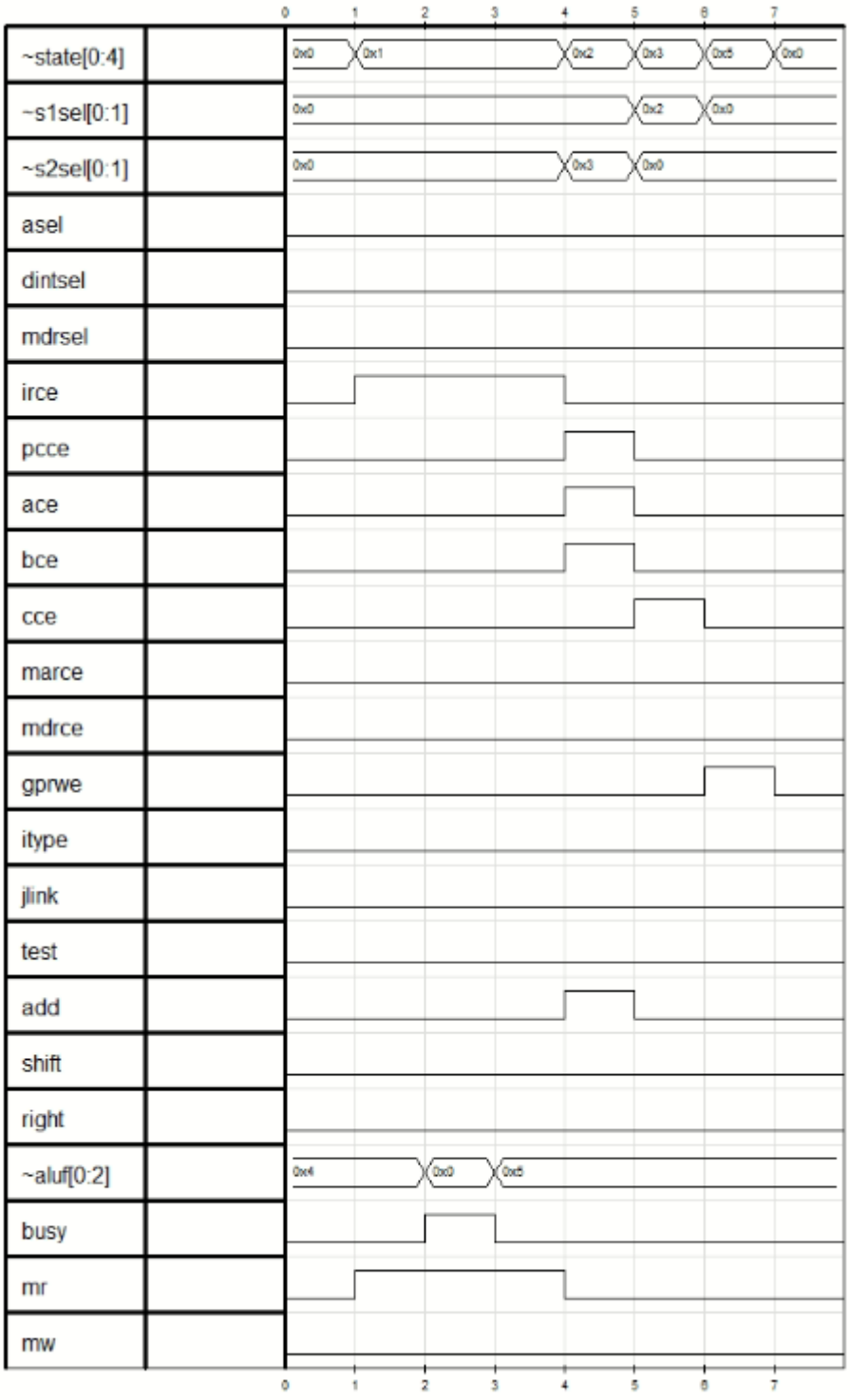
Before step enable:



After step enable:



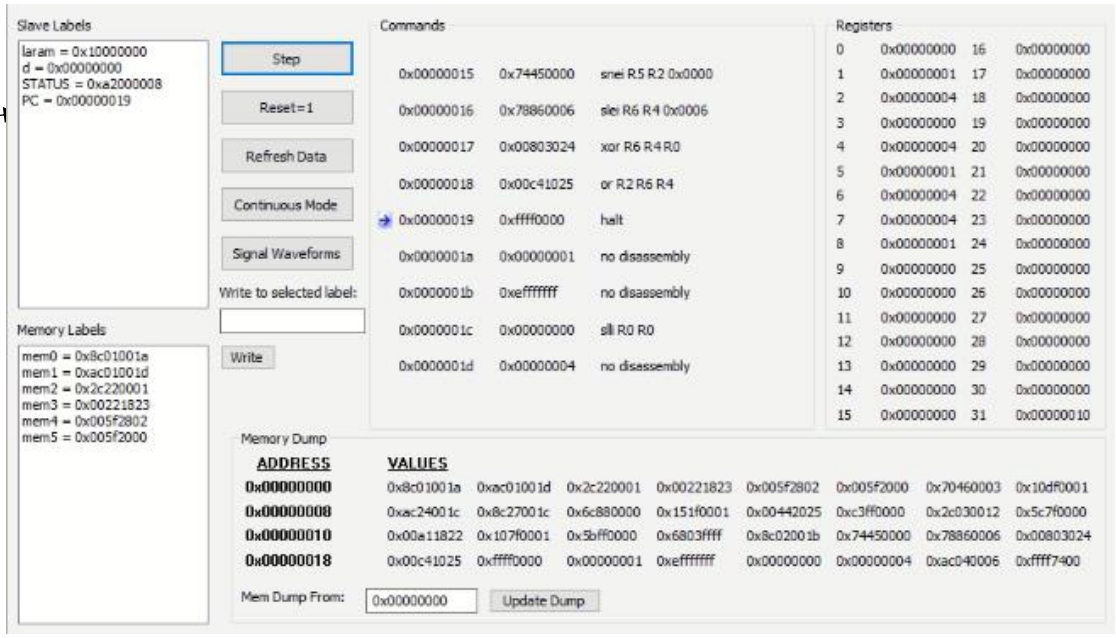
Waveforms:


0
1
2
3
4
5
6
7
~state[0:4]
0x0
0x1
0x2
0x3
0x5
0x0
~s1sel[0:1]
0x0
0x2
0x0
~s2sel[0:1]
0x0
0x3
0x0
asel
dintsel
mdrsel
irce
pcce
ace
bce
cce
marce
mdrce
gprwe
itype
jlink
test
add
shift
right
~aluf[0:2]
0x4
0x0
0x5
busy
mr
mw

PC = 0x19
Instruction: halt
Formula: -
PC = PC + 1 = 0x1a

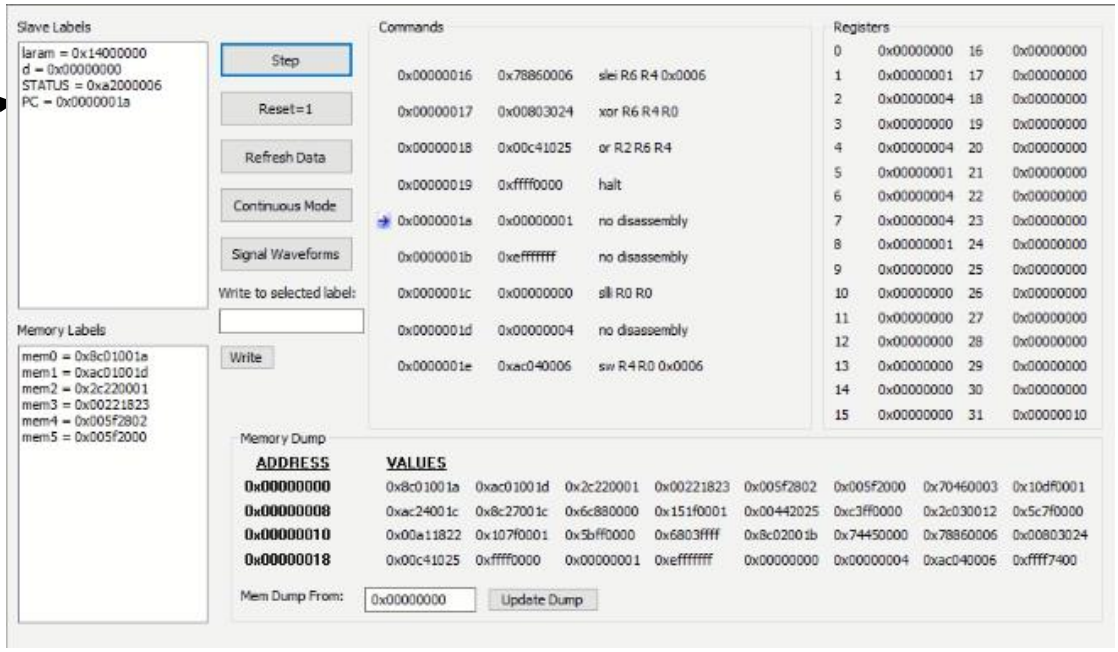
Before step enable:

PC = 0x19

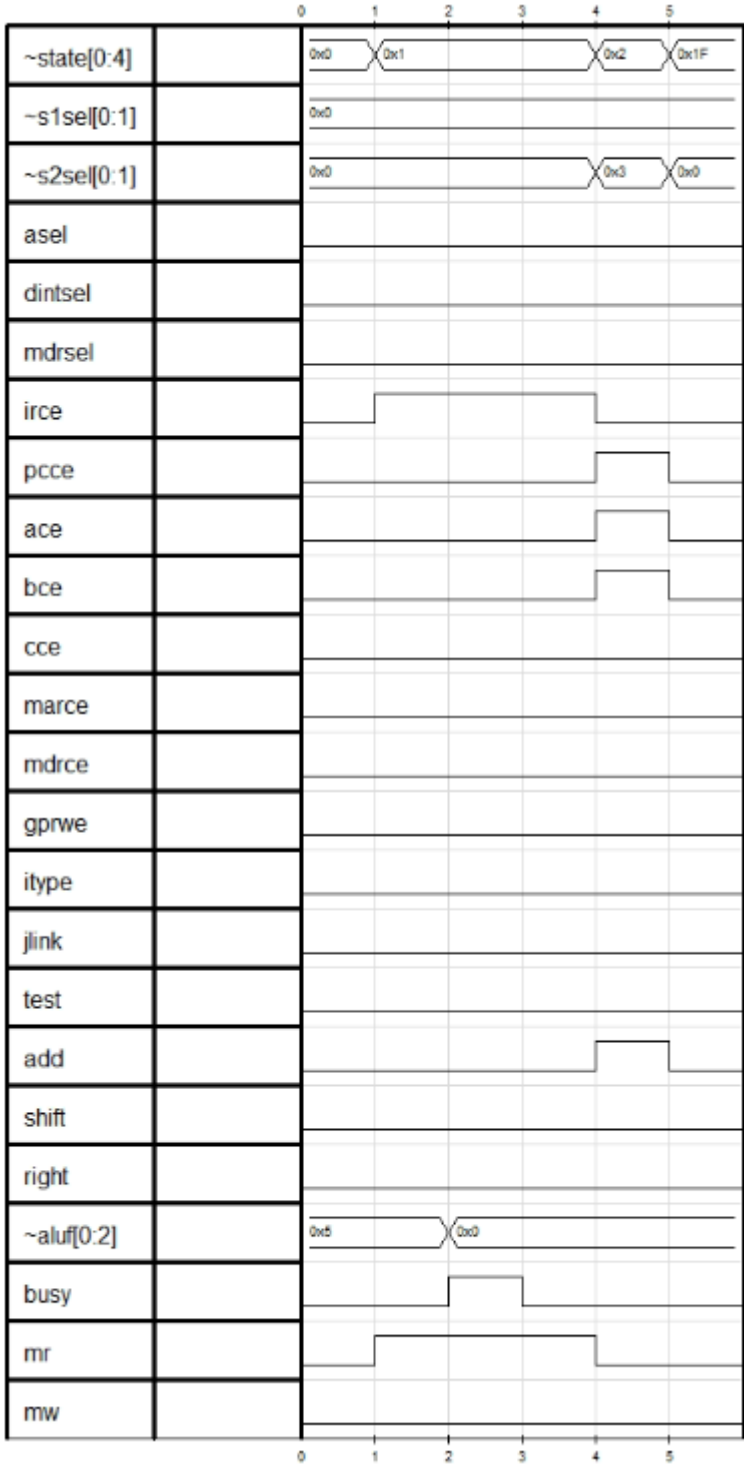


After step enable:

PC = 0x1a

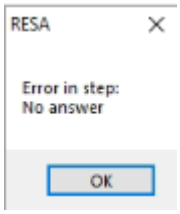


Waveforms:

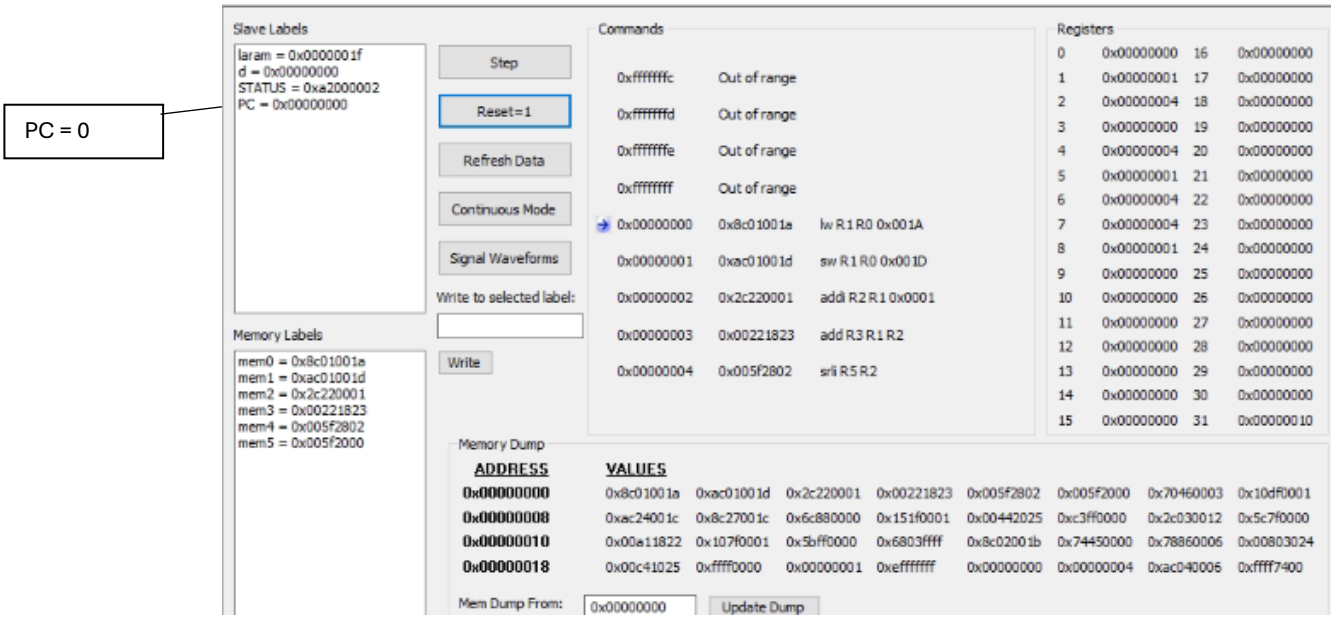


*Now the program is stuck until we press reset and pc will set to 0 if we try to press step we will receive an error.

After step enable:



After reset:



Conclusion:

All the instructions works as expected in addition all the waveforms are as expected comparing to the test vector we can see that the states travers as expected and the outputs in each state are as expected as well therefore we can conclude our DLX machine works as defined.