



Advanced Computer Structure Lab

Report 7: DLX

Saleh Yasin - 212108187
Ibrahim Hware - 322505926

Design and test Assignment

We will start by defining the different modules and partitions that we used in our implementation:

Control Unit Partitions:

DLX state machine Verilog code:

```

21 module DLX_state_machine(
22     input clk,
23     input reset,
24     input step_en,
25     input busy,
26     input [31:0] IR, // Instruction Register
27     input AEQZ,
28     output reg [4:0] state,
29     output reg [4:0] next_state,
30     output reg MR, IRce, Ace, Bce, PCce, add, Cce, test, itype,
31     output reg DINTsel, MARce, MDRce, MDRsel, MW, GPR_WE, jlink,
32     output reg [1:0] shift_right,
33     output reg in_init, Asel,
34     output reg [1:0] Sisel,
35     output reg [1:0] S2sel
36 );
37
38 // State Encoding
39 parameter [4:0] INIT = 5'b00000,
40     FETCH = 5'b00001,
41     DECODE = 5'b00010,
42     HALT = 5'b00011,
43     ALU = 5'b00100,
44     SHIFT = 5'b00101,
45     ALUI = 5'b00110,
46     TESTI = 5'b00111,
47     WBR = 5'b10100,
48     WBI = 5'b10101,
49     ADDRESSCMP = 5'b01010,
50     LOAD = 5'b01011,
51     COPYMDR2C = 5'b01100,
52     STORE = 5'b01101,
53     COPYGPR2MDR = 5'b01110,
54     JR = 5'b01111,
55     SAVEPC = 5'b10000,
56     JALR = 5'b10001,
57     BRANCH = 5'b10010,
58     BTAKEN = 5'b10011;
59
60 reg bt;
61
62 // bt Calculation
63 always @(*) begin
64     bt = AEQZ ^ IR[26];
65 end
66
67 // State Transition Logic
68 always @(*) begin
69     case (state)
70         INIT: next_state = step_en ? FETCH : INIT;
71         FETCH: next_state = busy ? FETCH : DECODE;
72         DECODE: begin
73             if ((IR[31:26] & 6'b111000) == 6'b1110000) next_state = step_en ? FETCH : INIT;
74             else if ((IR[31:26] & 6'b111100) == 6'b000000 && IR[5] == 1'b1) next_state = ALU;
75             else if ((IR[31:26] & 6'b111100) == 6'b000000 && (IR[5:0] & 6'b100000) == 6'b000000) next_state = SHIFT;
76             else if ((IR[31:26] & 6'b111000) == 6'b000000) next_state = ALUI;
77             else if ((IR[31:26] & 6'b111000) == 6'b011000) next_state = TESTI;
78             else if ((IR[31:26] & 6'b111000) == 6'b100000) next_state = ADDRESSCMP;
79             else if ((IR[31:26] & 6'b111001) == 6'b011000) next_state = JR;
80             else if ((IR[31:26] & 6'b111001) == 6'b0110001) next_state = SAVEPC;
81             else if ((IR[31:26] & 6'b111100) == 6'b000100) next_state = BRANCH;
82             else next_state = HALT;
83         end
84         HALT: next_state = HALT;
85         ALU: next_state = WBR;
86         SHIFT: next_state = WBR;
87         ALUI: next_state = WBI;
88         TESTI: next_state = WBI;
89         WBR: next_state = step_en ? STORE : (step_en ? FETCH : INIT);
90         WBI: next_state = step_en ? FETCH : INIT;
91         ADDRESSCMP: begin
92             if (IR[31:26] == 6'b101011) next_state = COPYGPR2MDR;
93             else if (IR[31:26] == 6'b100011) next_state = LOAD;
94         end
95         LOAD: next_state = busy ? LOAD : COPYMDR2C;
96         COPYMDR2C: next_state = WBI;
97         COPYGPR2MDR: next_state = STORE;
98         STORE: next_state = busy ? STORE : (step_en ? FETCH : INIT);
99         JR: next_state = step_en ? FETCH : INIT;
100        SAVEPC: next_state = JALR;
101        JALR: next_state = step_en ? FETCH : INIT;
102        BRANCH: next_state = bt ? BTAKEN : (step_en ? FETCH : INIT);
103        BTAKEN: next_state = step_en ? FETCH : INIT;
104        default: next_state = INIT;
105    endcase
106 end
107
108 // State Update Logic
109 always @ (posedge clk or posedge reset) begin
110     if (reset) begin
111         state <= INIT;
112     end else begin
113         state <= next_state;
114     end
115 end
116
117 // Output Logic
118 always @(*) begin
119     // Default values for all control signals
120     MR = 0; IRce = 0; Ace = 0; Bce = 0; PCce = 0; add = 0; Cce = 0; test = 0; itype = 0;
121     DINTsel = 0; shift_right = 2'b00; DINTsel = 0; MARce = 0; MDRce = 0; MDRce = 0; MW = 0;
122     GPR_WE = 0; jlink = 0; in_init = 1; MDRsel = 0;
123     Asel = 0;
124     Sisel = 2'b00;
125     S2sel = 2'b00;
126
127     case (state)
128         INIT: begin
129             in_init = 1;
130         end
131         FETCH: begin
132             MR = 1;
133             IRce = 1;
134             in_init = 0;
135         end
136         DECODE: begin
137             Ace = 1;
138             Bce = 1;
139             S2sel = 2'b11;
140             PCce = 1;
141             add = 1;
142             in_init = 0;
143         end
144         ALU: begin
145             Sisel = 2'b01;
146             Cce = 1;
147             in_init = 0;
148         end
149         TESTI: begin
150             Sisel = 2'b01;
151             S2sel = 2'b01;
152             Cce = 1;
153             test = 1;
154             itype = 1;
155             in_init = 0;
156         end
157         ALUI: begin
158             Sisel = 2'b01;
159             S2sel = 2'b01;
160             Cce = 1;
161             add = 1;
162             itype = 1;
163             in_init = 0;
164         end
165         SHIFT: begin
166             Sisel = 2'b01;
167             Cce = 1;
168             DINTEsel = 1;
169             shift_right = 2'b01;
170             if (IR[1])
171                 shift_right = 2'b11;
172             in_init = 0;
173         end
174         ADDRESSCMP: begin
175             itype = 1;
176             Sisel = 2'b01;
177             S2sel = 2'b01;
178             MARce = 1;
179             add = 1;
180             in_init = 0;
181         end
182         LOAD: begin
183             IRce = 1;
184             Ace = 1;
185             MR = 1;
186             MDRsel = 1;
187             in_init = 0;
188         end
189         STORE: begin
190             type = 1;
191             Asel = 1;
192             MW = 1;
193             in_init = 0;
194         end
195         COPYMDR2C: begin
196             Sisel = 2'b11;
197             S2sel = 2'b10;
198             DINTEsel = 1;
199             Cce = 1;
200             in_init = 0;
201         end
202         COPYGPR2MDR: begin
203             Sisel = 2'b10;
204             S2sel = 2'b10;
205             DINTEsel = 1;
206             MDRce = 1;
207             in_init = 0;
208             itype = 1;
209         end
210         WBR: begin
211             GPR_WE = 1;
212             in_init = 0;
213         end
214         WBI: begin
215             GPR_WE = 1;
216             itype = 1;
217             in_init = 0;
218         end
219         BRANCH: begin
220             in_init = 0;
221             if (bt) begin
222                 S2sel = 2'b01;
223             end
224         end
225     end
226 end

```

```

223         add = 1;
224         PCce = 0;
225     end
226
227 BTAKEN: begin
228     in_init = 0;
229     S2sel = 2'b01;
230     add = 1;
231     PCce = 1;
232 end
233 JR: begin
234     in_init = 0;
235     S1sel = 2'b01;
236     S2sel = 2'b10;
237     add = 1;
238     Cce = 1;
239 end
240 SAVEPC: begin
241     in_init = 0;
242     S2sel = 2'b10;
243     add = 1;
244     Cce = 1;
245 end
246 JALR: begin
247     in_init = 0;
248     S1sel = 2'b01;
249     S2sel = 2'b10;

```

```

250         add = 1;
251         PCce = 1;
252         GPR_WE =1;
253         jlink = 1;
254     end
255 default: begin
256     // Default case, all control signals remain 0
257 end
258 endcase
259 end
260
261 endmodule
262

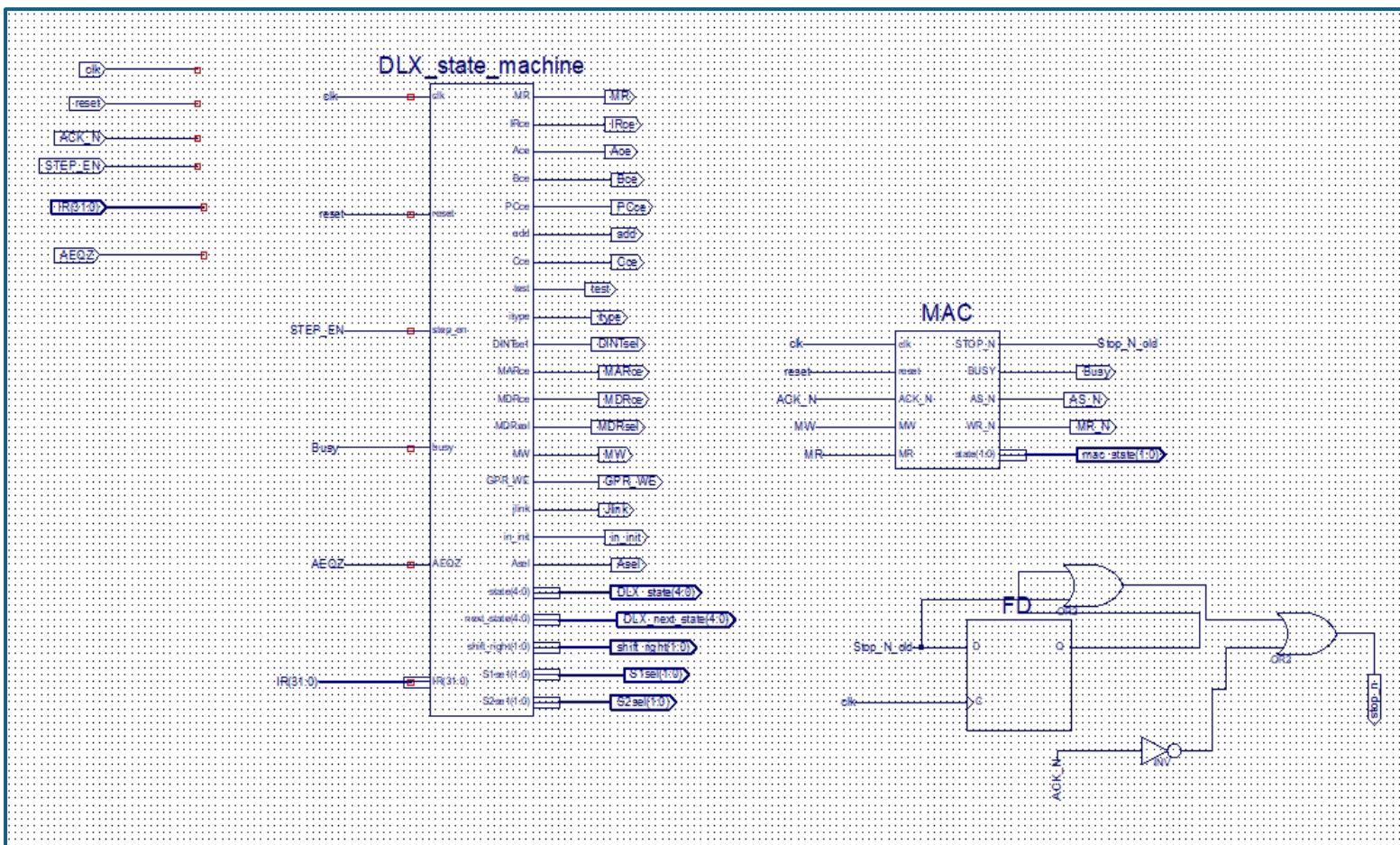
```

DLX_State_control unit:

Unit describe:

This unit contain all control partitions for our DLX connected together, including the aforementioned dlx_state_machine, MAC (from the previous lab) and the stop_n logic that allow us to record the important signals in RESA when the ack received from the external memory (also mentioned in the previous labs).

Unit schematic:



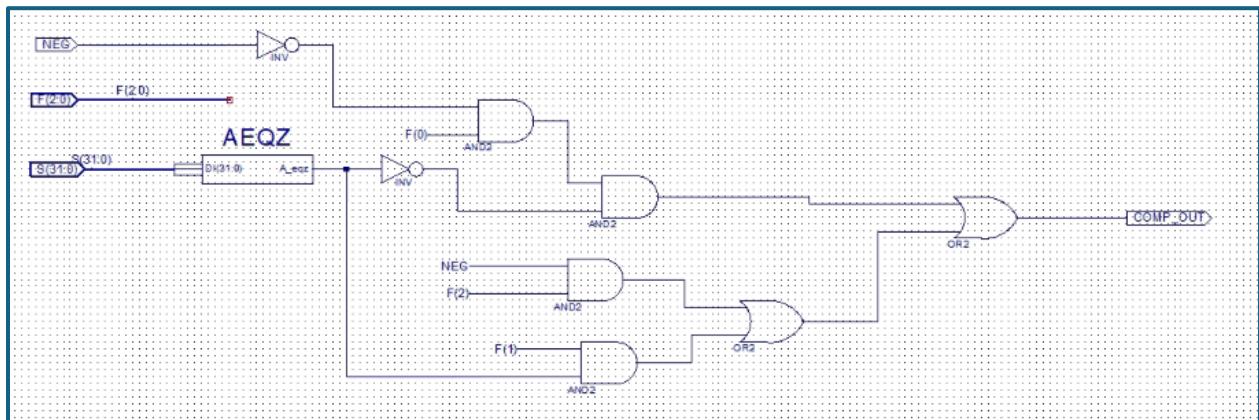
Data Path Unit Partitions:

Comparator:

Unit describe:

This module used to compare different inputs (A, B) according to the performed instruction (SLEI, SGTI.....), and its part of the ALU.

Unit schematic:



Add_sub_32:

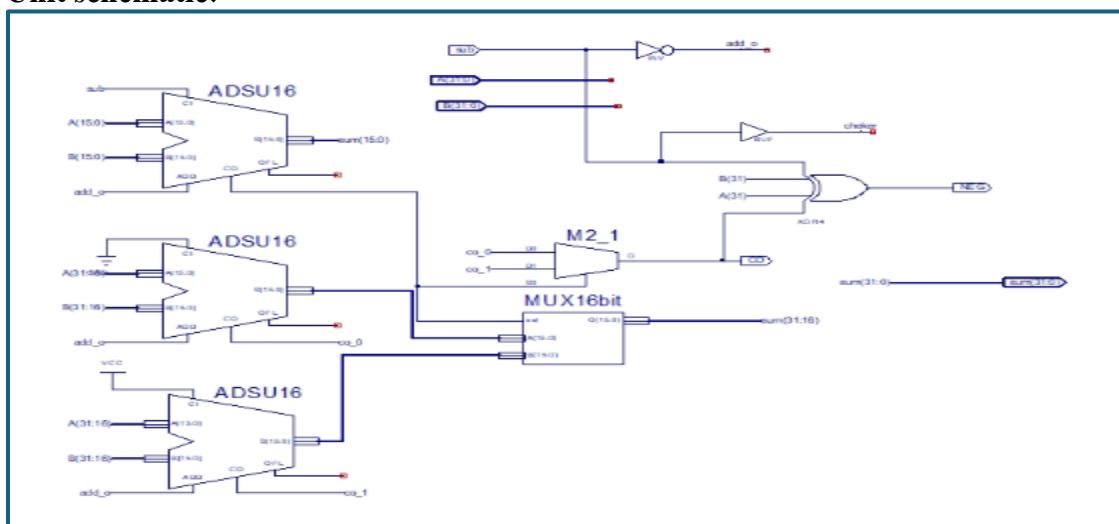
Unit describe:

This module is a 2's-complement adder that can handle both add, sub instructions for 32-bit inputs.

This adder is constructed of 3 ADSU_16 units from xilinx library, one ADSU used to add/sub the 16 LSBs bit and the other two ADSU's to add/sub the 16 MSBs.

We use two ADSU's to add/sub the 16 MSBs to have a better performance and use parallel approach instead of series which cost more delay.

Unit schematic:



ALU:

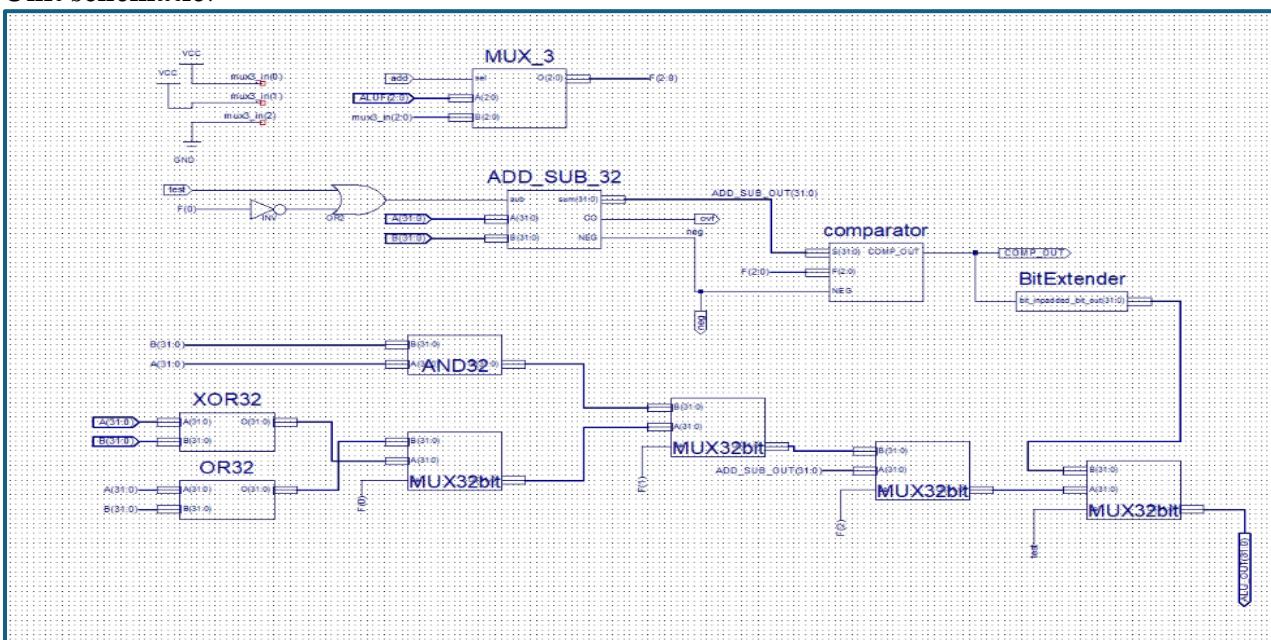
Unit describe:

The ALU of our DLX, this unit is responsible to perform all the arithmetic operations including: ADD, SUB , COMP, AND, OR, and XOR.

This unit use the following blocks:

- MUX_3: used to choose between the add and the ALUF[2:0] (the function in the OPCODE).
- BitExtender: padding the 1 bit input with zero's.
- AND32: bitwise and.
- OR32: bitwise or.
- XOR32: bitwise xor.
- 3 32-bit MUX'S: used to choose between the comparator , AND, OR, XOR outputs.

Unit schematic:



bit_shifter:

Unit describe:

This unit operates one bit shift to the right or the left according to the input signals: shift, right.

Unit Verilog module:

```

21 module bit_shifter(
22     input wire shift_en,
23     input wire right,
24     input wire [31:0] data_in,
25     output reg [31:0] data_out
26 );
27
28 always @(*) begin
29     if (shift_en) begin
30         if (right) begin

```

```

31         // Logical shift right by 1
32         data_out = data_in >> 1;
33     end else begin
34         // Logical shift left by 1
35         data_out = data_in << 1;
36     end
37     end else begin
38         // No shift
39         data_out = data_in;
40     end
41 end
42
43 endmodule

```

DLX datapath:

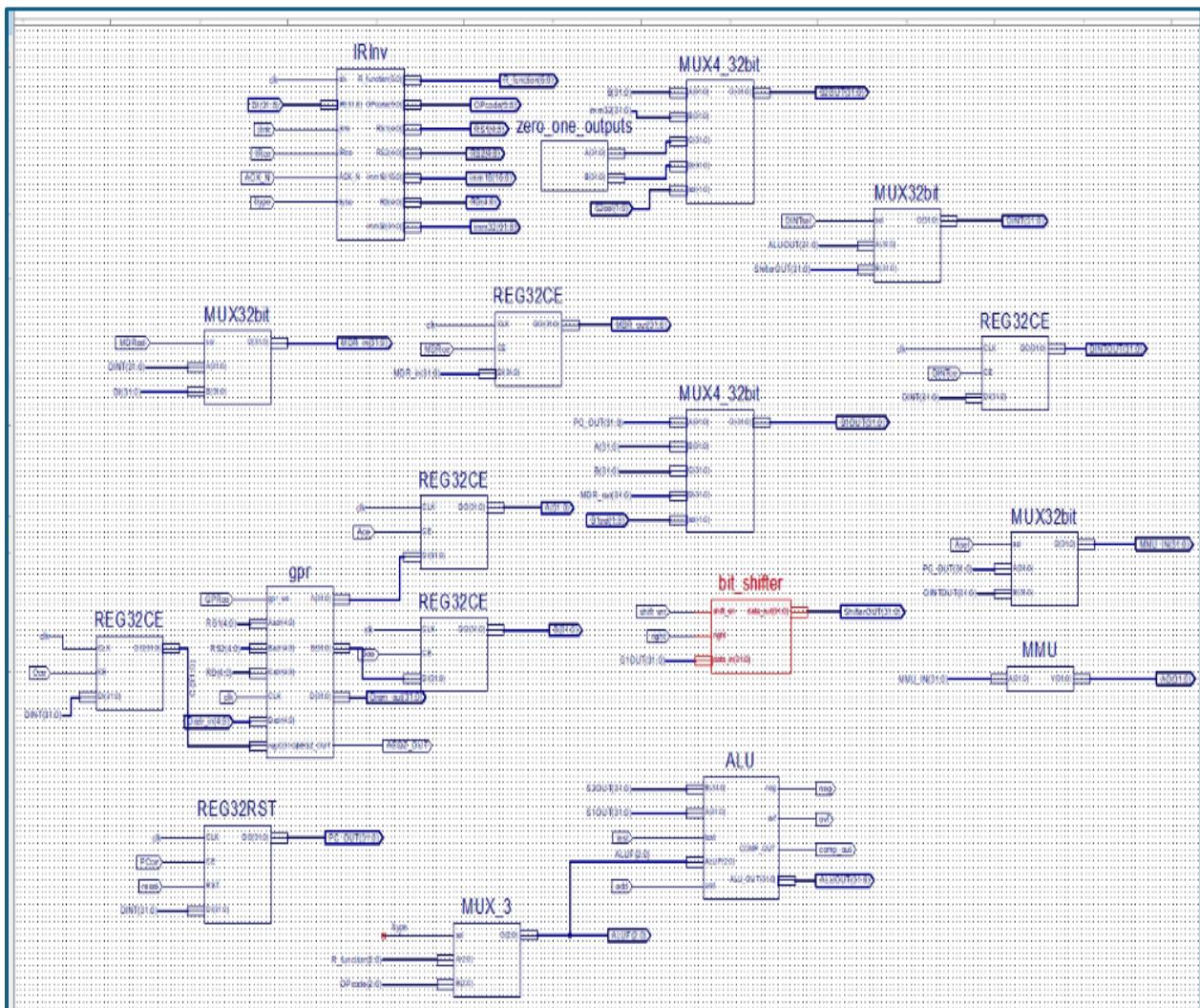
Unit describe:

This unit connects all the sub-unites of the DLX datapath together, In addition it use some mux's to choose between the different path's according to the performed operation, and it use also some registers with ce's to save a local copy of the data at specific states.

This unit use the following blocks:

- GPR: from the previous lab.
- MMU: 24 to 32 bit extension.
- IRInv: subset the loaded instruction into different signals, and extends the immediate width .

Unit schematic:

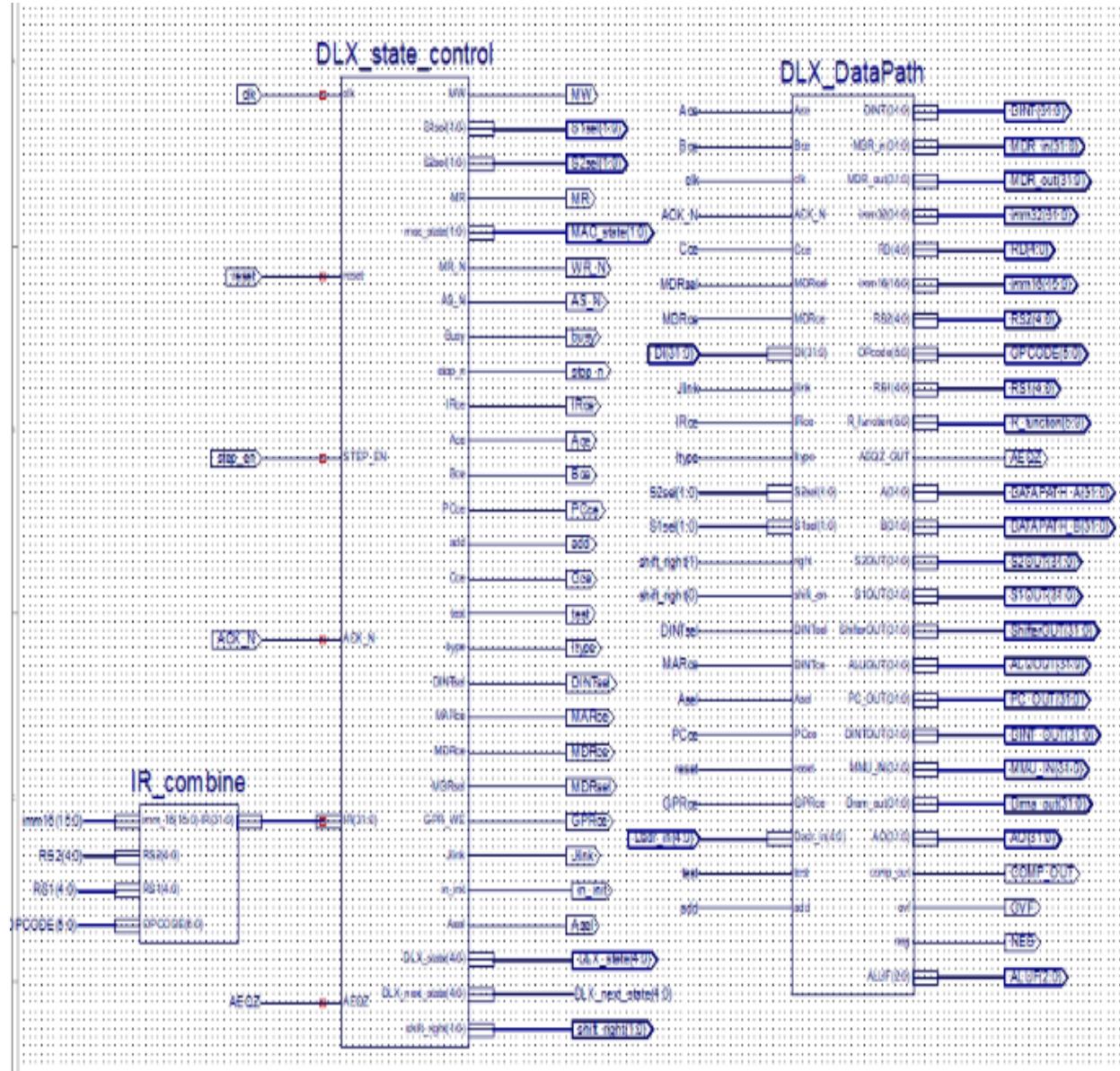


DLX:

Unit describe:

The DLX contains the control unit connected to the data path unit all in one unit.

Unit schematic:



- The IR_combine used to combine all the IR env signals in one 32 bit bus.

Top module:

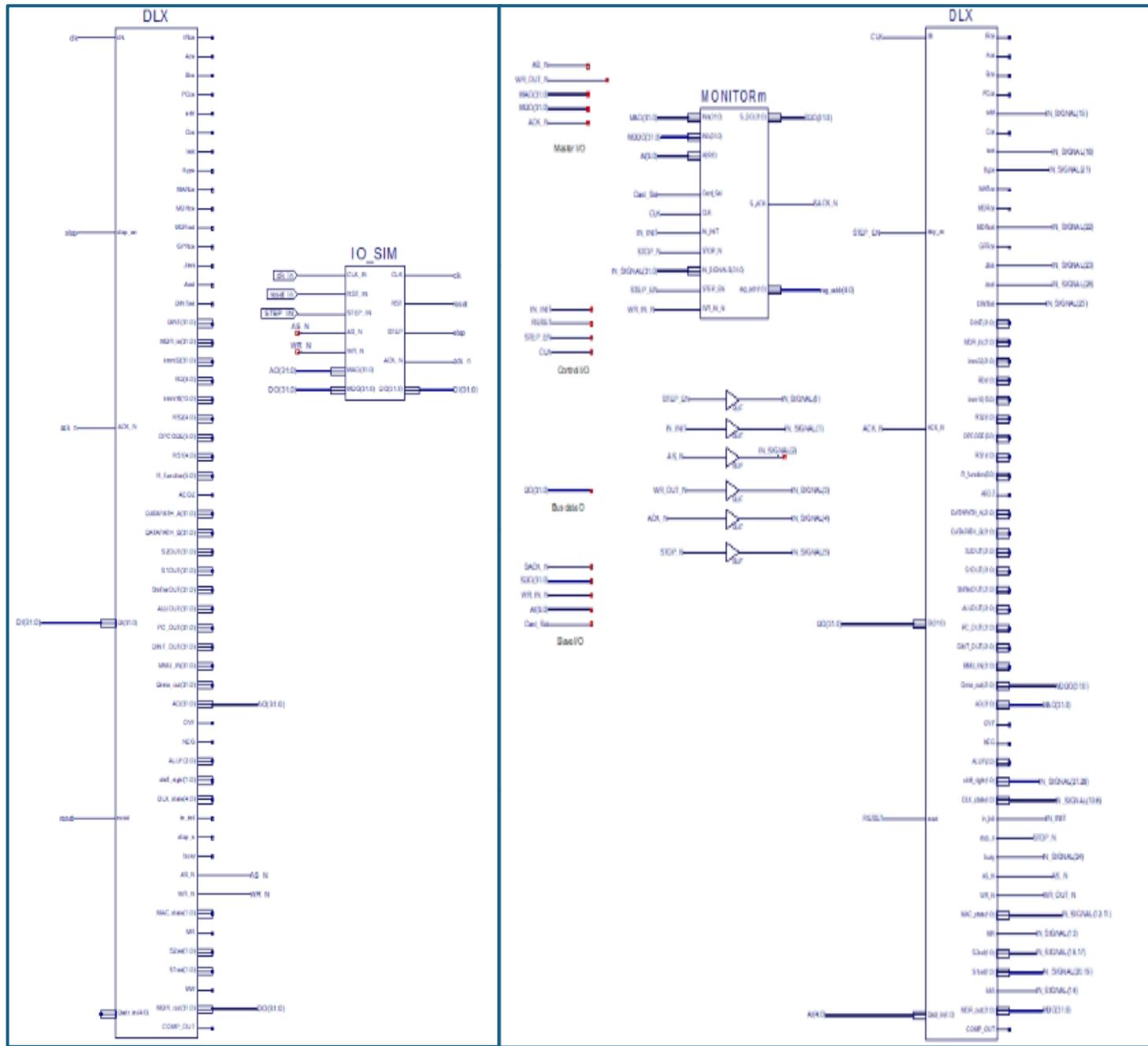
Unit describe:

The top_module for L/O stimul contains L/O stimul unit connected to the DLX.
The top_module for RESA contains L/O_logic unit connected to the DLX.

Unit's schematic:

IO_SIM TOP LEVEL:

RESA TOP LEVEL:



Control Unit Test:

We will start by determining the number of each state in both the MAC (memory access controller) and the state machine and the OPCODE's:

State Machine states:

State	State Enumeration
INIT	00000
FETCH	00001
DECODE	00010
	00011
HALT	
ALU	00100
SHIFT	00101
ALUI	00110
TESTI	00111
WBR	01000
WBI	01001
ADDRESSCMP	01010
LOAD	01011
COPYMDR2C	01100
STORE	01101
COPYGPR2MDR	01110
JR	01111
SAVEPC	10000
JALR	10001
BRANCH	10010
BTAKEN	10011

OPCODE:

OPCODE	operation
IR[31 : 26]	Mnemonic
Data Transfer	
100 011	lw
101 011	sw
Arithmetic, Logical Operation	
001 011	addi
Test Set Operation	
011 rel	s rel i
011 001	sgti
011 010	seqi
011 011	sgei
011 100	slti
011 101	snei
011 110	slei
Control Operation	
000 100	beqz
000 101	bnez
010 110	jr
010 111	jalr
Miscellaneous Instructions	
110 000	special-nop
111 111	halt
IR[5 : 0]	Mnemonic
Shift Operation	
000 000	Slli
000 010	Srli
Arithmetical and Logical Operations	
100 011	Add
100 010	Sub
100 110	And
100 101	Or
100 100	Xor

MAC states:

State	State Enumeration (in Decimal)
WAIT4REQ	0
WAIT4ACK	1
NEXT	2

Test Vectors:

STORE Instruction:

I/O	Signals	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
input	#CC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
input	RESET	1														
input	STEP_EN		1													
input	IN_INIT			1												
output	DLX state	INIT	INIT	FETCH	FETCH	FETCH	FETCH	DECODE	ADRCOMP	COPYGPR2MDR	STORE	STORE	STORE	STORE	INIT	INIT
output	OPCODE (5:0)	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	101***	101***	101***	000000	000000	000000	000000	000000	000000
output	STATE (MAC)	Wait4req	Wait4req	Wait4req	Wait4ack	Wait4ack	Wait4ack	Next	Wait4req	Wait4req	Wait4ack	Wait4ack	Wait4ack	Wait4ack	Next	Wait4req
output	GPR_CE															
output	MAC_BUSY		1								1					
output	W_R	1														
output	AS_N			1												
output	STOP_N				1											
output	ACK_N					1										
output	IR_CE		1													
output	A_CE															
output	B_CE															
output	C_CE															
output	MDR_SEL									1						
output	MDR_CE										1					
output	MAR_CE										1					
output	SHIFT															
output	ADD									1						
output	S1SEL (1:0)	00	00	00	00	00	00	00	01	10	00	00	00	11	00	00
output	S2SEL (1:0)	00	00	00	00	00	00	11	01	10	00	00	00	10	00	00
output	DINTSEL									1						
output	PC_CE										1					
output	ASEL											1				
output	TEST															
output	JLINK															
output	MW										1					
output	MR		1													

LOAD Instruction:

I/O	Signals	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
input	#CC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
input	RESET	1														
input	STEP_EN		1													
input	IN_INIT			1												
output	STATE (DLX_CTRL)	INIT	INIT	FETCH	FETCH	FETCH	FETCH	DECODE	ADRCOMP	LOAD	LOAD	LOAD	LOAD	COPYMDR2C	WBI	INIT
output	OPCODE (5:0)	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	100***	100***	100***	100***	100***	100***	100***	100***	000000
output	ITYPE															
output	GPR_CE															
output	STATE (MAC)	Wait4req	Wait4req	Wait4req	Wait4ack	Wait4ack	Wait4ack	Next	Wait4req	Wait4req	Wait4ack	Wait4ack	Wait4ack	Wait4ack	Next	Wait4req
output	MAC_BUSY															
output	W_R		1													
output	AS_N			1												
output	STOP_N				1											
output	ACK_N					1										
output	IR_CE			1												
output	A_CE															
output	B_CE									1						
output	C_CE										1					
output	MDR_SEL										1					
output	MDR_CE											1				
output	MAR_CE											1				
output	SHIFT															
output	ADD									1						
output	S1SEL (1:0)	00	00	00	00	00	00	00	01	00	00	00	00	11	00	00
output	S2SEL (1:0)	00	00	00	00	00	00	11	01	00	00	00	00	10	00	00
output	DINTSEL															
output	PC_CE															
output	ASEL															
output	TEST															
output	JLINK															
output	MW															
output	MR			1												

TESTI Instruction:

I/O	Signals	0	1	2	3	4	5	6	7	8	9	10	11	12
input	#CC	0												
input	RESET		1											
input	STEP_EN			1										
input	IN_INIT				1									
output	TATE (DLX_CTRL)	INIT	INIT	FETCH	FETCH	FETCH	FETCH	DECODE	TESTI	WBI	INIT	INIT	INIT	INIT
output	OPCODE (5:0)	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	011***	011***	011***	000000	000000	000000	000000
output	ITYPE													
output	GPR_CE													
output	STATE (MAC)	Wait4req	Wait4req	Wait4req	Wait4ack	Wait4ack	Wait4ack	Next	Wait4req	Wait4req	Wait4req	Wait4req	Wait4req	Wait4req
output	MAC_BUSY													
output	W_R													
output	AS_N													
output	STOP_N													
output	ACK_N													
output	IR_CE													
output	A_CE													
output	B_CE													
output	C_CE													
output	MDR_SEL													
output	MDR_CE													
output	MAR_CE													
output	SHIFT													
output	ADD													
output	S1SEL (1:0)	00	00	00	00	00	00	00	01	00	00	00	00	00
output	S2SEL (1:0)	00	00	00	00	00	00	11	01	00	00	00	00	00
output	DINTSEL													
output	PC_CE													
output	ASEL													
output	TEST													
output	JLINK													
output	MW													
output	MR													

JALR Instruction:

I/O	Signals	0	1	2	3	4	5	6	7	8	9	10	11	12
input	#CC	0												
input	RESET		1											
input	STEP_EN			1										
input	IN_INIT				1									
output	TATE (DLX_CTRL)	INIT	INIT	FETCH	FETCH	FETCH	FETCH	DECODE	SAVEPC	JALR	INIT	INIT	INIT	INIT
output	OPCODE (5:0)	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	011***	011***	011***	000000	000000	000000	000000
output	GPR_CE													
output	STATE (MAC)	Wait4req	Wait4req	Wait4req	Wait4ack	Wait4ack	Wait4ack	Next	Wait4req	Wait4req	Wait4req	Wait4req	Wait4req	Wait4req
output	MAC_BUSY													
output	W_R													
output	AS_N													
output	STOP_N													
output	ACK_N													
output	IR_CE													
output	A_CE													
output	B_CE													
output	C_CE													
output	MDR_SEL													
output	MDR_CE													
output	MAR_CE													
output	SHIFT													
output	ADD													
output	S1SEL (1:0)	00	00	00	00	00	00	00	00	01	00	00	00	00
output	S2SEL (1:0)	00	00	00	00	00	00	11	10	10	00	00	00	00
output	DINTSEL													
output	PC_CE													
output	ASEL													
output	TEST													
output	JLINK													
output	MW													
output	MR													

ALU Instruction:

input	#CC	0	1	2	3	4	5	6	7	8	9	10	11	12
input	RESET													
input	STEP_EN													
input	IN_INIT													
output	TATE (DLX_CTRL)	INIT	INIT	FETCH	FETCH	FETCH	FETCH	DECODE	ALU	WBR	INIT	INIT	INIT	INIT
output	OPCODE (5:0)	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	0000**	0000**	0000**	000000	000000	000000	000000
output	GPR_CE													
output	STATE (MAC)	Wait4req	Wait4req	Wait4req	Wait4ack	Wait4ack	Wait4ack	Next	Wait4req	Wait4req	Wait4req	Wait4req	Wait4req	Wait4req
output	MAC_BUSY													
output	W_R													
output	AS_N													
output	STOP_N													
output	ACK_N													
output	IR_CE													
output	A_CE													
output	B_CE													
output	C_CE													
output	MDR_SEL													
output	MDR_CE													
output	MAR_CE													
output	SHIFT													
output	ADD													
output	S1SEL (1:0)	00	00	00	00	00	00	00	01	00	00	00	00	00
output	S2SEL (1:0)	00	00	00	00	00	00	11	00	00	00	00	00	00
output	DINTSEL													
output	PC_CE													
output	ASEL													
output	TEST													
output	JLINK													
output	MW													
output	MR													

BTAKEN Instruction:

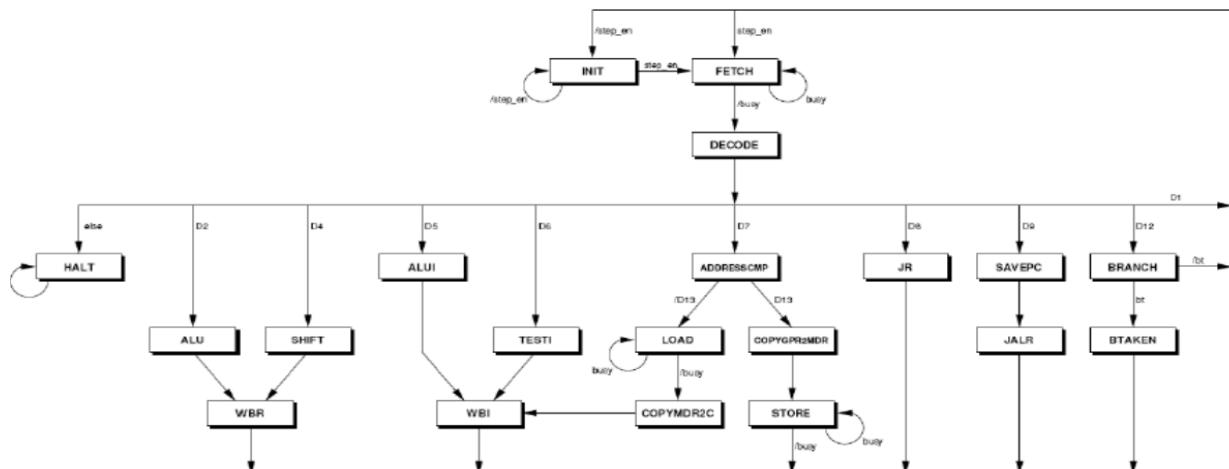
I/O	Signals	0	1	2	3	4	5	6	7	8	9	10	11	12
input	#CC													
input	RESET													
input	STEP_EN													
input	IN_INIT													
output	STATE(DLX_CTRL)	INIT	INIT	FETCH	FETCH	FETCH	FETCH	DECODE	BRANCH	BTAKEN	INIT	INIT	INIT	INIT
output	OPCODE(5:0)	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	XXXXXX	0001**	0001**	0001**	000000	000000	000000	000000
output	GPR_CE													
output	STATE(MAC)	Wait4req	Wait4req	Wait4req	Wait4ack	Wait4ack	Wait4ack	Wait4ack	Next	Wait4req	Wait4req	Wait4req	Wait4req	Wait4req
output	MAC_BUSY													
output	W_R													
output	AS_N													
output	STOP_N													
output	ACK_N													
output	IR_CE													
output	A_CE													
output	B_CE													
output	C_CE													
output	MDR_SEL													
output	MDR_CE													
output	MAR_CE													
output	SHIFT													
output	ADD													
output	S1SEL (1:0)	00	00	00	00	00	00	00	01	00	00	00	00	00
output	S2SEL (1:0)	00	00	00	00	00	00	11	00	00	00	00	00	00
output	DINTSEL													
output	PC_CE													
output	ASEL													
output	TEST													
output	JLINK													
output	MW													
output	MR													

Explanations:

- To check our control unit and to simulate the Test Vectors we used the table that we saw in the recitation, the table shows which signals we need to enable in each state, the rest of the signals must be nullified/grounded or the default value of the signal (in_init=1 for example).
 - We tested the control unit by comparing the waveform that we got with the same table that we mentioned before, and to test the DLX state transitions we used the state diagram that we saw in the recitation.
 - The aforementioned table:

Name	RTL Instruction	Active Control Signals
Fetch	$IR = M(PC)$	MR, IRce
Decode	$A = R(RSI)$ $B = R(RS2)$ $PC = PC + 1$	Ace, Bce, S2sel[1], S2sel[0] PCce, add
Alu	$C = A \text{ op } B$	S1sel[0], Cce
TestI	$C = (A \text{ rel } imm)$	S1sel[0], S2sel[0], Cce, test, Itype
Alu1(add)	$C = A + sext(imm)$	S1sel[0], S2sel[0], Cce, add, Itype
Shift	$C = A \text{ shift } sa$ $Sa = 1, IR[1]$	S1sel[0], Cce DINTsel, shift (.right)
Adr.Comp	$MAR = A + sext(imm)$	S1sel[0], S2sel[0].MARce, add
Load	$MDR = M(MAR)$	MDRce, Ase1, MR, MDRsel
Store	$M(MAR) = MDR$	Ase1, MW
CopyMDR2C	$C = MDR (>> 0)$	S1sel[0], S1sel[1], S2sel[1], DINTsel, Cce
CopyGPR2MDR	$MDR = B (<< 0)$	S1sel[1], S2sel[1], DINTsel, MDRce
WBR	$R(RD) = C$ (R-type)	GPR_WE
WBI	$R(RD) = C$ (I-type)	GPR_WE, Itype
Branch	Branch taken?	
Btaken	$PC = PC + sext(imm)$	S2sel[0], Add, PCce
JR	$PC = A$	S1sel[0], S2sel[1], add, PCce
Save PC	$C = PC$	S2sel[1], add, Cce
JALR	$PC = A$ $R(31) = C$	S1sel[0], S2sel[1], add, PCce GPR_WE, jlink

- The aforementioned state diagram:



I/O Simul Test:

To simulate the DLX we connected it to the I/O simul from the previous lab, and we ran the following test:

```
1  `timescale 1ns / 1ps
2
3  module IO_LS_IO_LS_sch_tb();
4
5  // Inputs
6  reg CLK_IN;
7  reg RESET_IN;
8  reg STEP_EN_IN;
9
10 // Output
11
12 // Bidirs
13
14 // Instantiate the UUT
15   DLX_STIMUL UUT (
16     .clk_in(CLK_IN),
17     .reset_in(RESET_IN),
18     .STEP_IN(STEP_EN_IN)
19   );
20 // Initialize Inputs
21 initial
22 CLK_IN=0;
23 always #10 CLK_IN=~CLK_IN;
24 initial
25 begin
26 #10;
27 RESET_IN=0;
28 STEP_EN_IN=0;
29 #20;
30 RESET_IN=1;
31 #20;
32 RESET_IN=0;
33 #20;
34 STEP_EN_IN=1;
35 #20;
36 STEP_EN_IN=0;
37 #300;
38 STEP_EN_IN=1;
39 #20;
40 STEP_EN_IN=0;
41 #300;
42 STEP_EN_IN=1;
43 #20;
44 STEP_EN_IN=0;
45 #300;
46 STEP_EN_IN=1;
47 #20;
48 STEP_EN_IN=0;
49 #300;
50 STEP_EN_IN=1;
51 #20;
52 STEP_EN_IN=0;
53 #300;
54 STEP_EN_IN=1;
55 #20;

      56  STEP_EN_IN=0;
      57  #300;
      58  STEP_EN_IN=1;
      59  #20;
      60  STEP_EN_IN=0;
      61  #300;
      62  STEP_EN_IN=1;
      63  #20;
      64  STEP_EN_IN=0;
      65  #300;
      66  STEP_EN_IN=1;
      67  #20;
      68  STEP_EN_IN=0;
      69  #300;
      70  STEP_EN_IN=1;
      71  #20;
      72  STEP_EN_IN=0;
      73  #300;
      74  STEP_EN_IN=1;
      75  #20;
      76  STEP_EN_IN=0;
      77  #300;
      78  STEP_EN_IN=1;
      79  #20;
      80  STEP_EN_IN=0;
      81  #300;
      82  STEP_EN_IN=1;
      83  #20;
      84  STEP_EN_IN=0;
      85  #300;
      86  STEP_EN_IN=1;
      87  #20;
      88  STEP_EN_IN=0;
      89  #300;
      90  STEP_EN_IN=1;
      91  #20;
      92  STEP_EN_IN=0;
      93  #300;
      94  STEP_EN_IN=1;
      95  #20;
      96  STEP_EN_IN=0;
      97  #300;
      98  STEP_EN_IN=1;
      99  #20;
     100  STEP_EN_IN=0;
     101  #300;
     102  STEP_EN_IN=1;
     103  #20;
     104  STEP_EN_IN=0;
     105  #300;
     106  STEP_EN_IN=1;
     107  #20;
     108  STEP_EN_IN=0;
     109  #300;
     110  STEP_EN_IN=1;

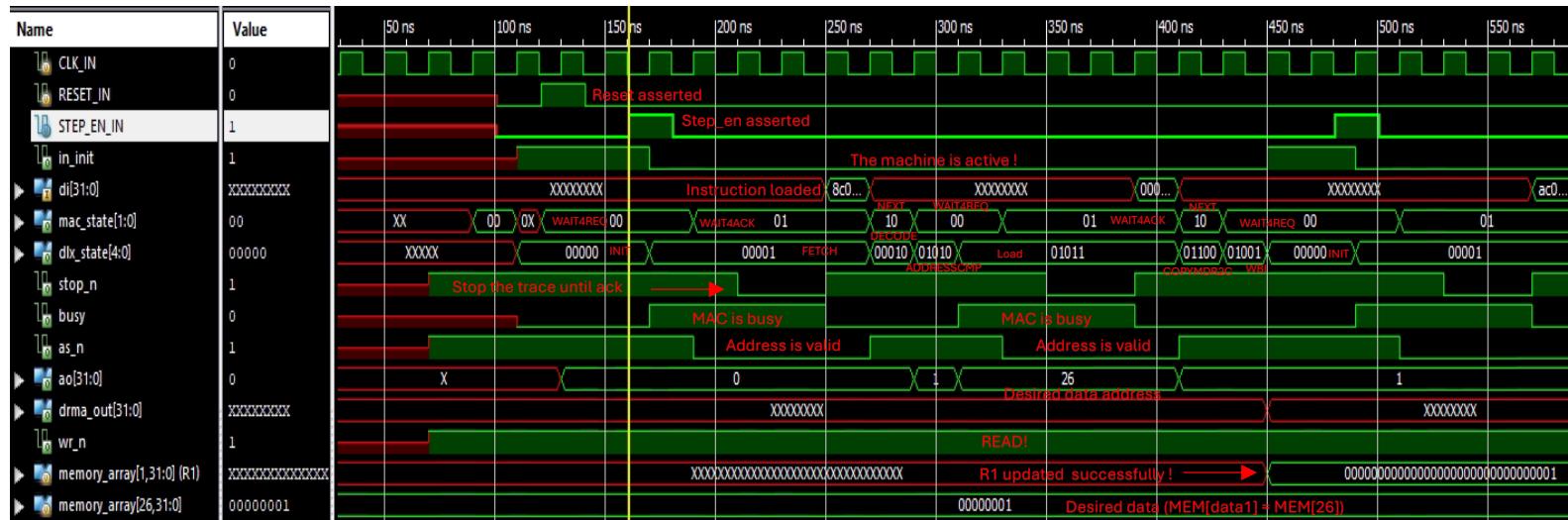
      174  STEP_EN_IN=1;
      175  #20;
      176  STEP_EN_IN=0;
      177  #300;
      178  STEP_EN_IN=1;
      179  #20;
      180  STEP_EN_IN=0;
      181  #300;
      182  STEP_EN_IN=1;
      183  #20;
      184  STEP_EN_IN=0;
      185  #300;
      186  STEP_EN_IN=1;
      187  #20;
      188  STEP_EN_IN=0;
      189  #300;
      190  STEP_EN_IN=1;
      191  #20;
      192  STEP_EN_IN=0;
      193  #300;
      194  STEP_EN_IN=1;
      195  #20;
      196  STEP_EN_IN=0;
      197  #300;
      198  STEP_EN_IN=1;
      199  #20;
      200  STEP_EN_IN=0;
      201  #300;
      202  STEP_EN_IN=1;
      203  #20;
      204  STEP_EN_IN=0;
      205  #300;
      206  STEP_EN_IN=1;
      207  #20;
      208  STEP_EN_IN=0;
      209  #300;
      210  STEP_EN_IN=1;
      211  #20;
      212  STEP_EN_IN=0;
      213  #300;
      214  STEP_EN_IN=1;
      215  #20;
      216  STEP_EN_IN=0;
      217  end
      218  endmodule
```

So basically we just reset the DLX and the I/O simul, and then we just give it step enables one after another, and in each step the DLX will execute one instruction.

The results of the test shown in the next pages.

LW simulation waveform:

Operation used for this simulation: **8C01001A // lw R1 R0 data1**



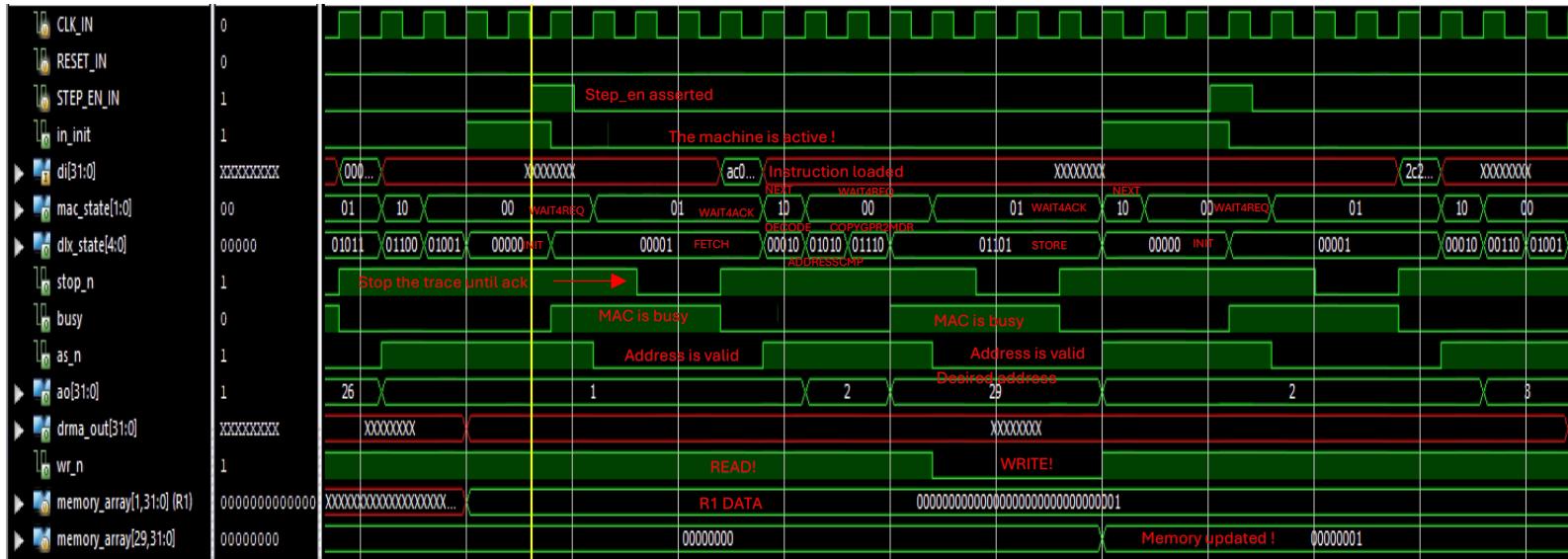
Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 0 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the ADDRESSCMP state.
- Cycle after that the state machine moves to the LOAD state.
- As_n = 0, WR_N = 1 to load the data from the external memory.
- AO = DATA0 = 26 to load the desired data from the external memory.
- Cycle after that the state machine moves to the COPYMDR2C state.
- Cycle after that the state machine moves to the WBI state.
- Cycle after we can see that the register 1 updated with the desired data (0x1) that was in the desired address in the memory (data1).
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

SW simulation waveform:

Operation used for this simulation: **AC01001D //**

sw R1 R0 adr2



Waveform explanation:

- We can see that when the reset asserted the machine start in the INIT state.
 - We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
 - AO = PC = 1 to load the relevant instruction.
 - We can see that cycle after the state moves to DECODE state.
 - Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
 - When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully
 - Cycle after that the state machine move to the ADDRESSCMP state.
 - Cycle after that the state machine move to the COPYGPR2MDR state.
 - Cycle after that the state machine move to the STORE state.
 - As_n = 0 , WR_N=0 to store the data to the external memory.
 - AO = Adr2 = 29 to store the desired data from to external memory from R1.
 - Cycle after we can see that the memory updated with the desired data (0x1) that was in register 1.
 - Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

ADDI simulation waveform:

Operation used for this simulation: **2C220001 // addi R2 R1 1**

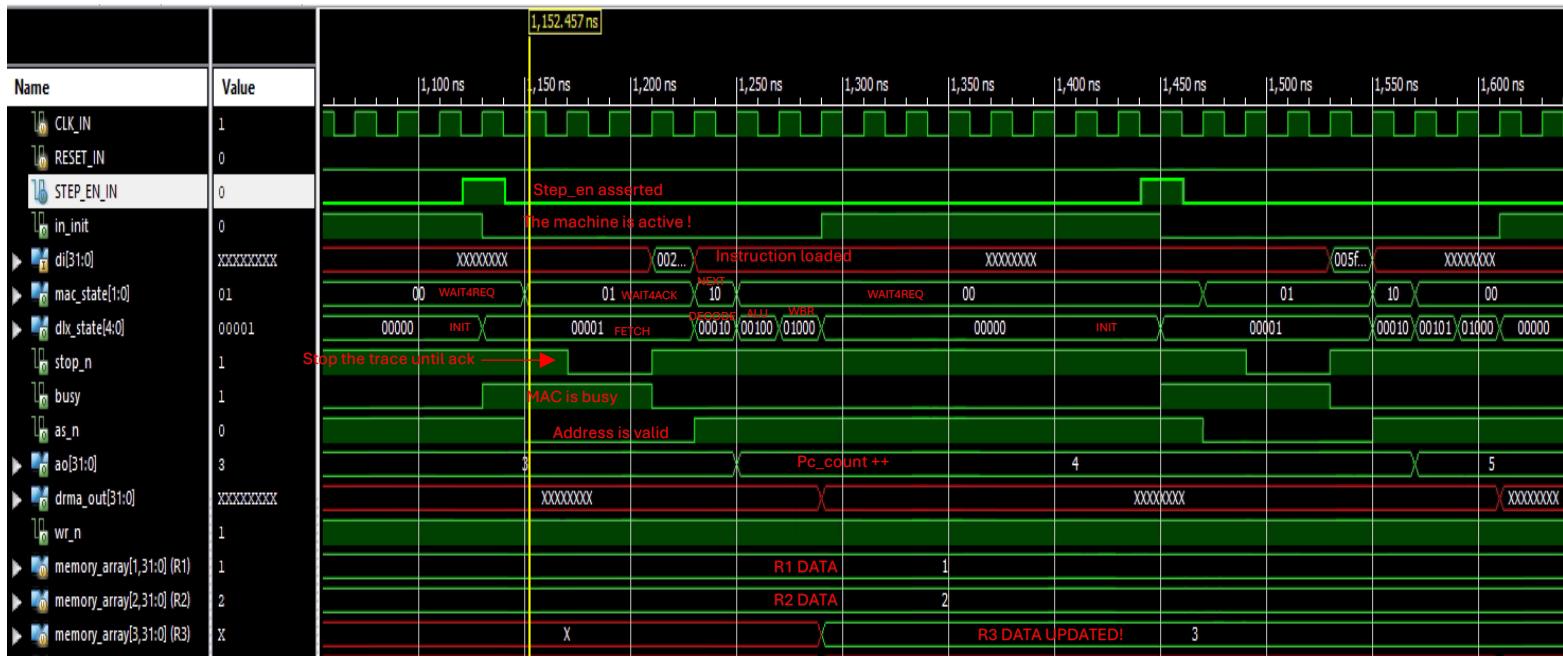


Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 2 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the ALUI state.
- Cycle after that the state machine moves to the WBI state.
- Cycle after we can see that the R2 updated with the value: immediate + R1 = 1 + 1 = 2 successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

ADD simulation waveform:

Operation used for this simulation: 00221823 // add R3 R1 R2

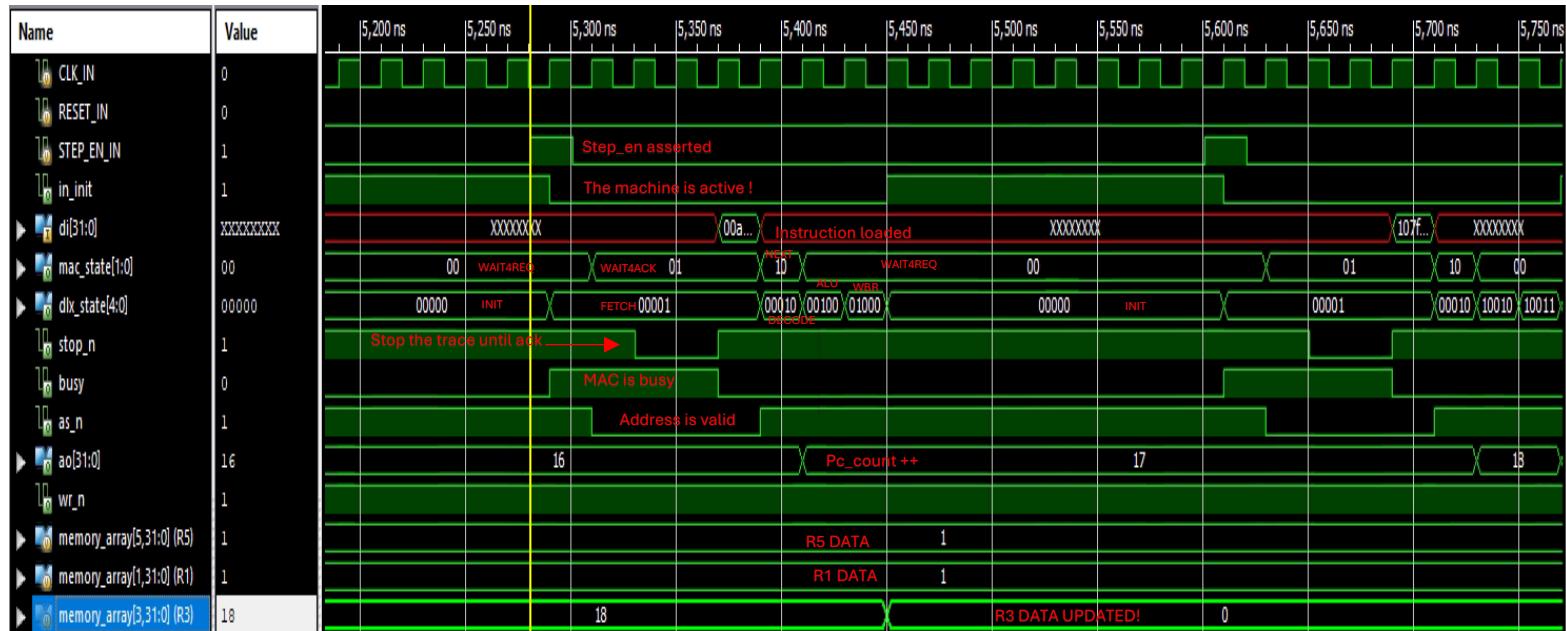


Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 2 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the ALU state.
- Cycle after that the state machine moves to the WBR state.
- Cycle after we can see that the R3 updated with the value: $R1 + R2 = 1 + 2 = 3$ successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

SUB simulation waveform:

Operation used for this simulation: **00A11822 // sub R3 R5 R1**



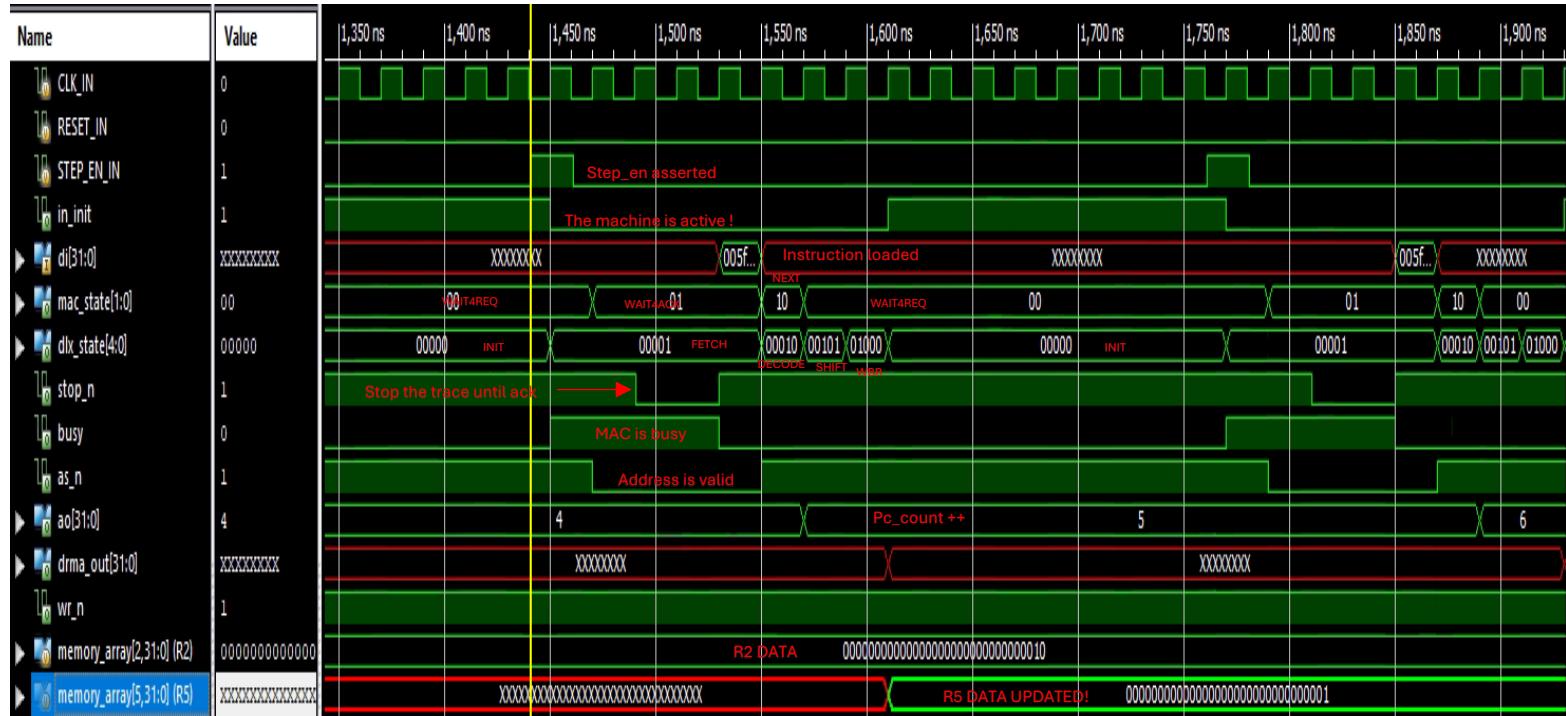
Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 2 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully
- Cycle after that the state machine moves to the ALU state.
- Cycle after that the state machine moves to the WBR state.
- Cycle after we can see that the R3 updated with the value: $R5 - R1 = 1 - 1 = 0$ successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

SRLI simulation waveform:

Operation used for this simulation: **005F2802 //**

srl R5 R2

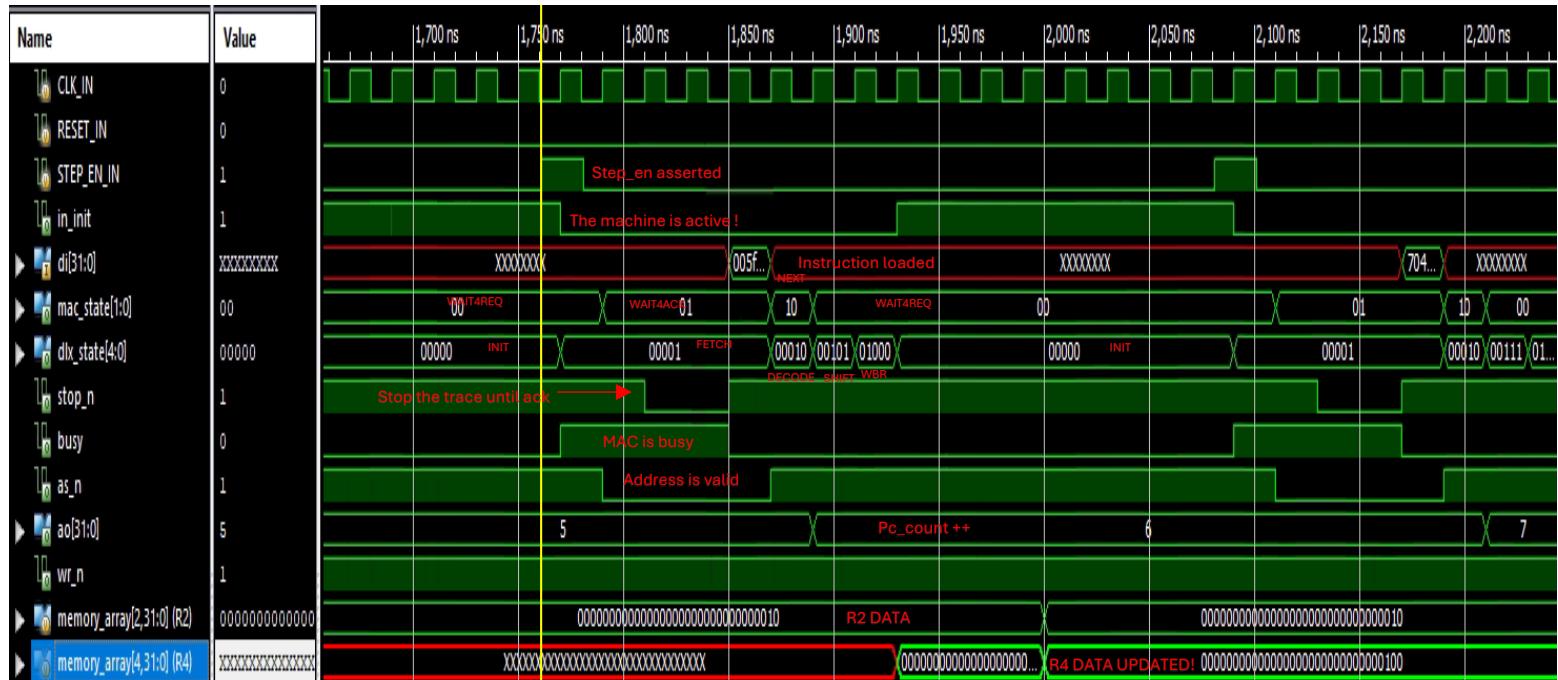


Waveform explanation:

- We can see that when the reset asserted the machine start in the INIT state.
 - We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
 - AO = PC = 2 to load the relevant instruction.
 - We can see that cycle after the state moves to DECODE state.
 - Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
 - When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully
 - Cycle after that the state machine move to the SHIFT state.
 - Cycle after that the state machine move to the WBR state.
 - Cycle after we can see that the R5 updated with the value: $R5 = 2'b10>>1 = 2'b01$ successfully.
 - Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

SLLI simulation waveform:

Operation used for this simulation: **005F2000 // slli R4 R2**



Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 5 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the SHIFT state.
- Cycle after that the state machine moves to the WBR state.
- Cycle after we can see that the R4 updated with the value: R4 = 2'b10<<1 = 3'b100 successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

SLTI simulation waveform:

Operation used for this simulation: 70460003 // slti R6 R2 3

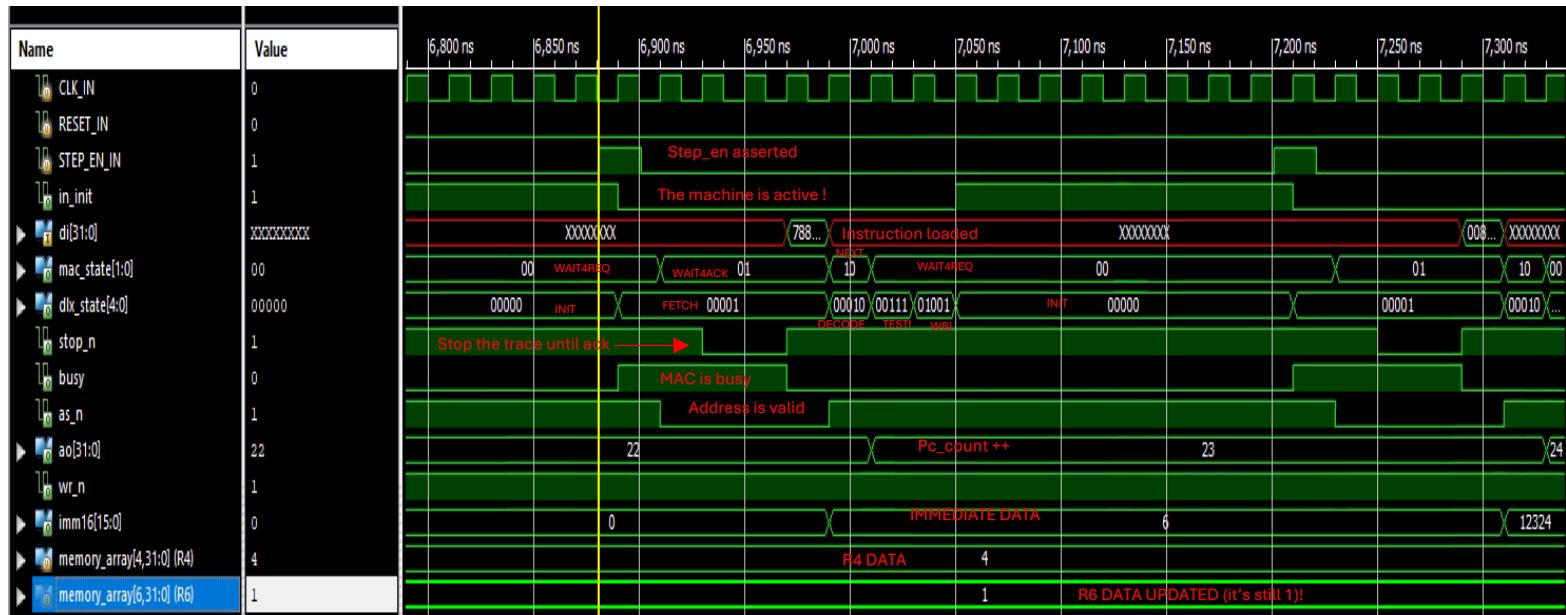


Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 6 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the TESTI state.
- Cycle after that the state machine moves to the WBI state.
- Cycle after we can see that the R6 updated with the value: R6 = (2<3) = 1 successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

SLEI simulation waveform:

Operation used for this simulation: **78860006 // slei R6 R4 6**

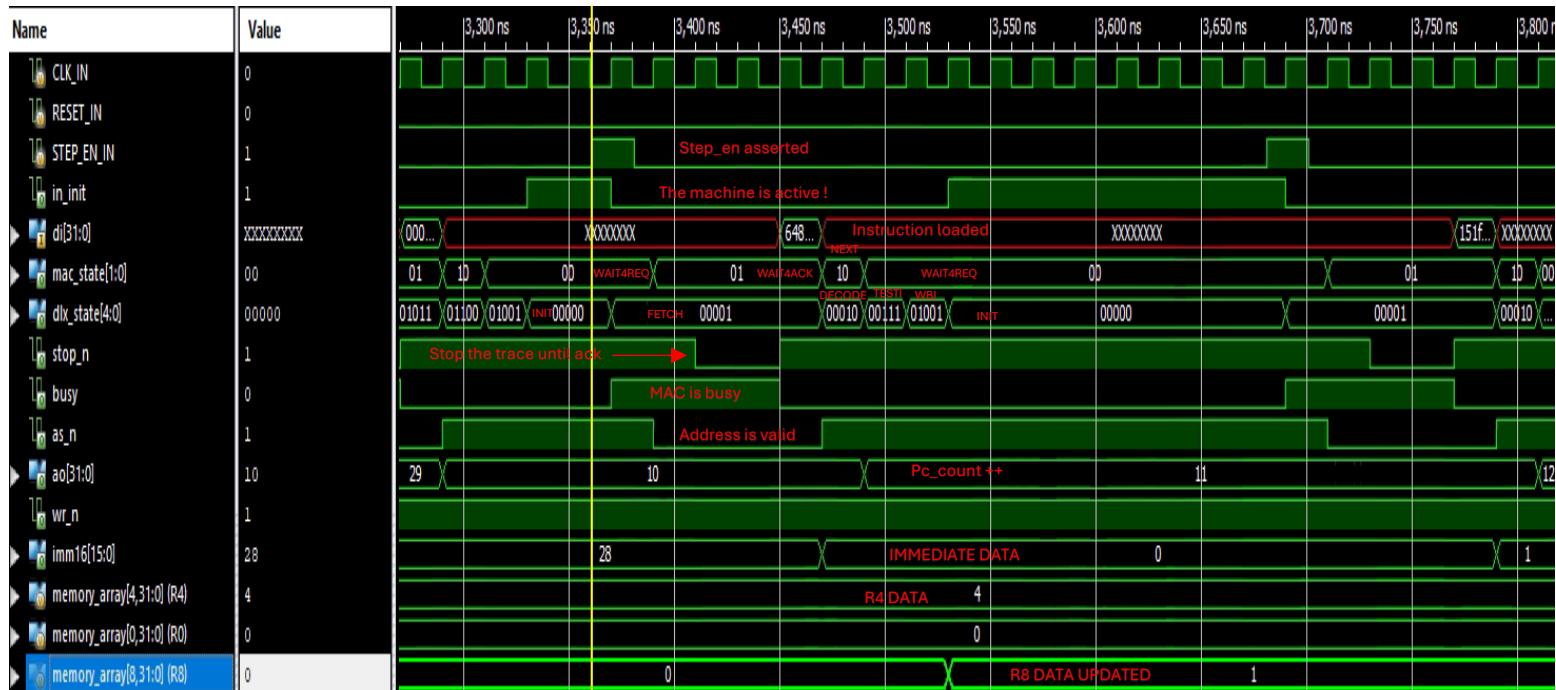


Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 22 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the TESTI state.
- Cycle after that the state machine moves to the WBI state.
- Cycle after we can see that the R6 updated with the value: R6 = (4 <= 6) = 1 successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

SGTI simulation waveform:

Operation used for this simulation: **64880000 // sgti R8 R4 0**



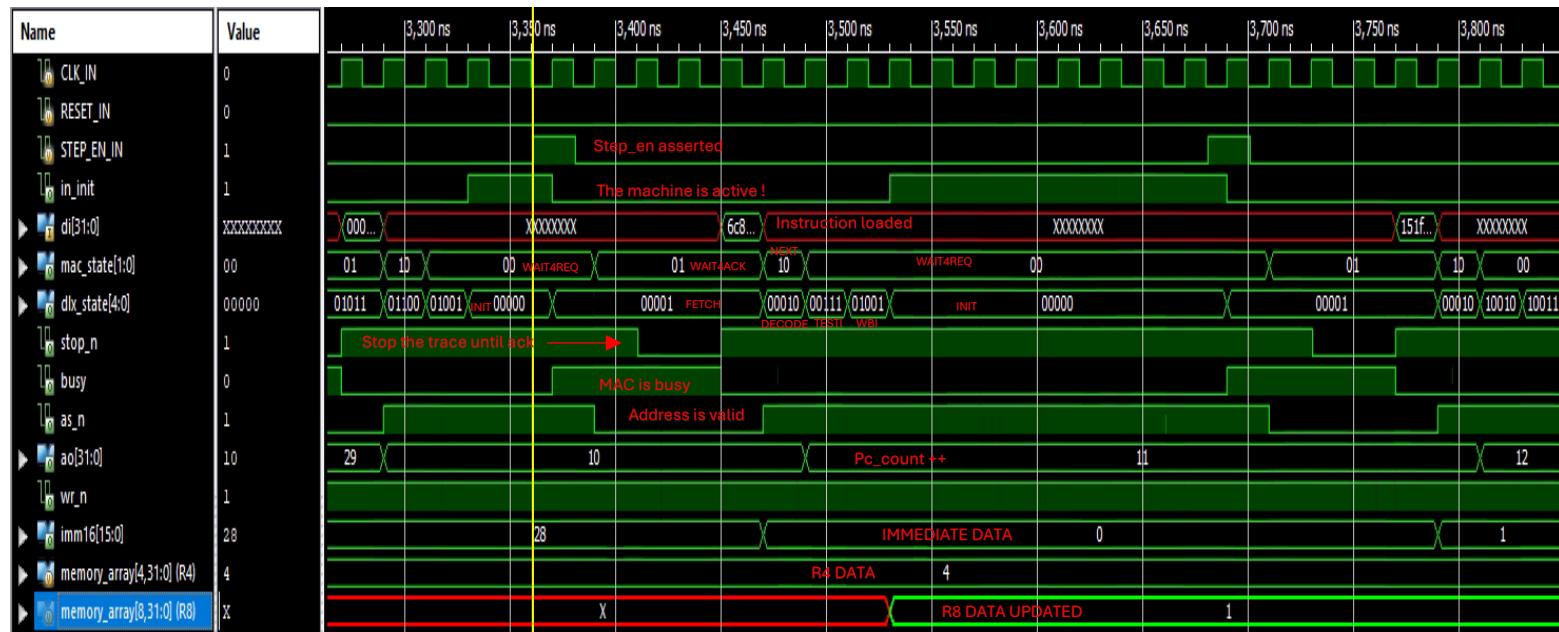
Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 10 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the TESTI state.
- Cycle after that the state machine moves to the WBI state.
- Cycle after we can see that the R8 updated with the value: R8 = (4>=0) = 1 successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

SGEI simulation waveform:

Operation used for this simulation: **6C880000 //**

sgei R8 R4 0

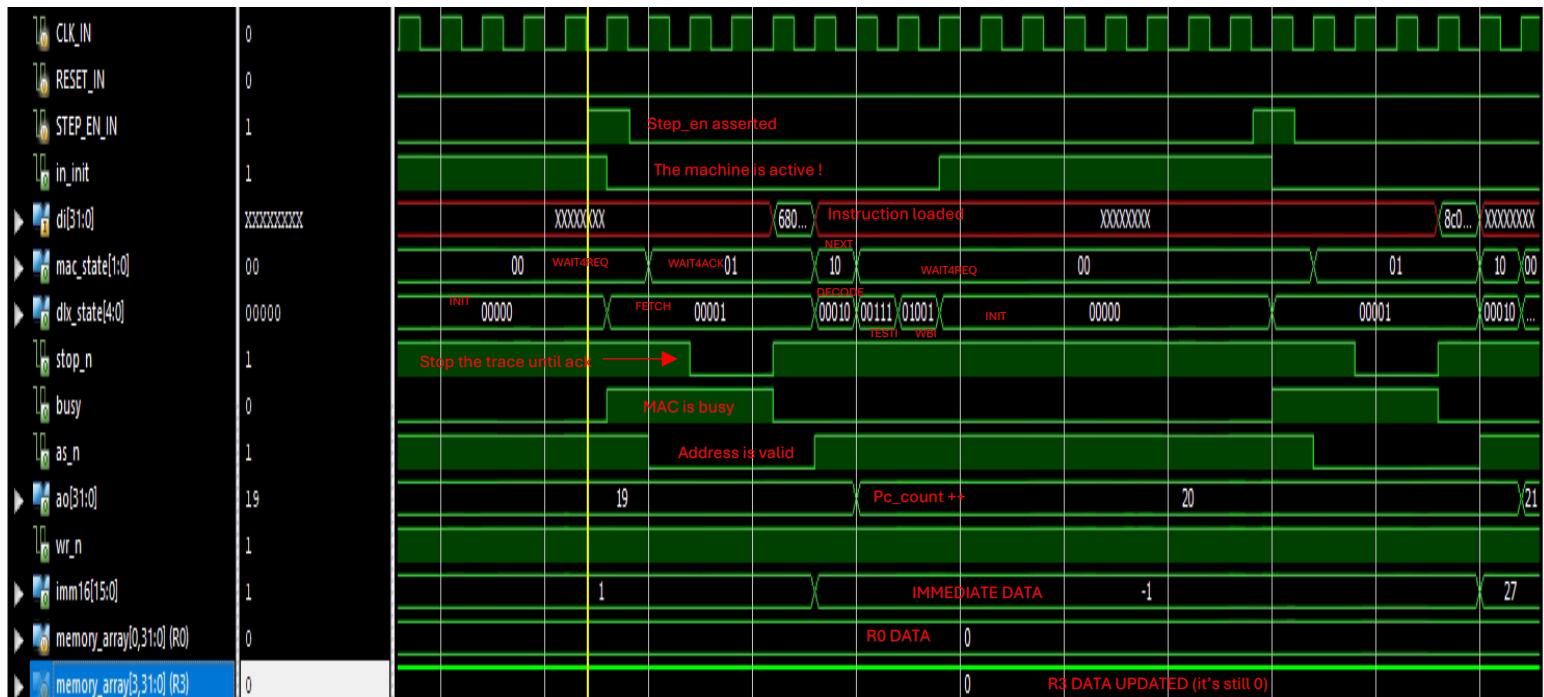


Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 10 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the TESTI state.
- Cycle after that the state machine moves to the WBI state.
- Cycle after we can see that the R8 updated with the value: R8 = (4>=0) = 1 successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

SEQI simulation waveform:

Operation used for this simulation: **6803FFFF // jump4: seqi R3 R0 -1**

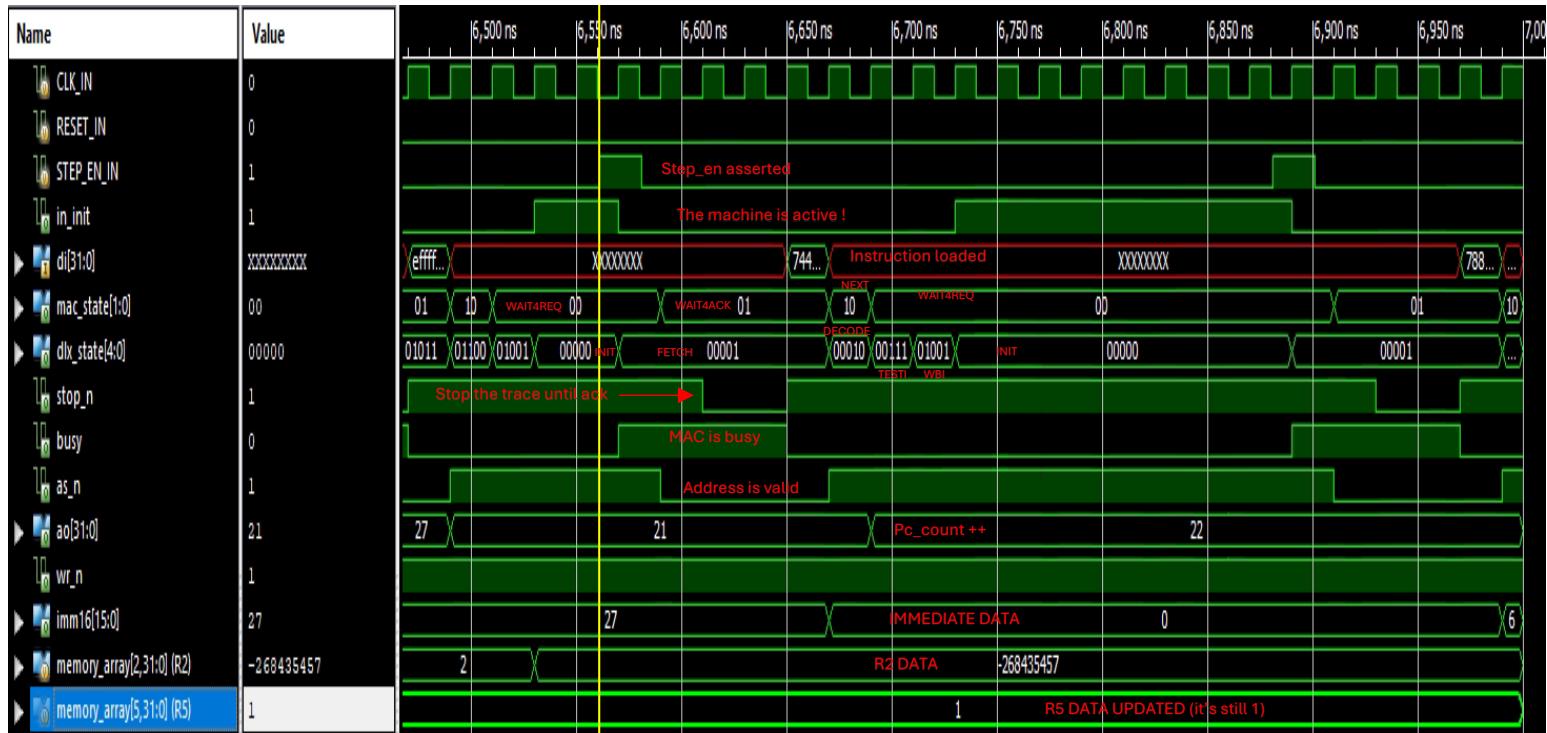


Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 19 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ACK received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the TESTI state.
- Cycle after that the state machine moves to the WBI state.
- Cycle after we can see that the R3 updated with the value: R3 = (-1 == 0) = 0 successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

SNEI simulation waveform:

Operation used for this simulation: 74450000 // snei R5 R2 0

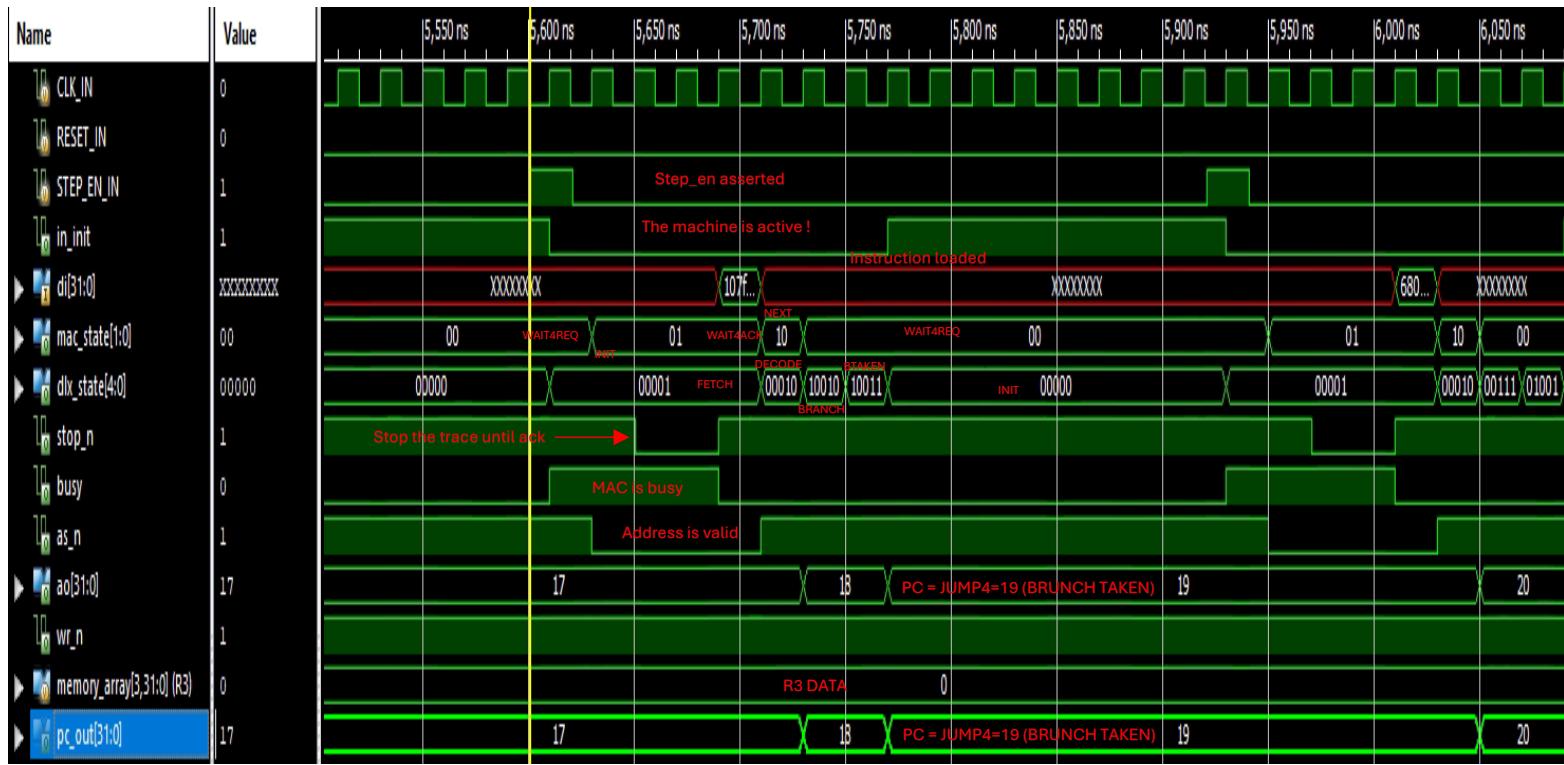


Waveform explanation:

- We can see that when the reset asserted the machine start in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 21 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully
- Cycle after that the state machine move to the TESTI state.
- Cycle after that the state machine move to the WBI state.
- Cycle after we can see that the R5 updated with the value: $R5 = (-268435457 \neq 0) = 1$ successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

BEQZ simulation waveform:

Operation used for this simulation: **107F0001 // beqz R3 jump4**

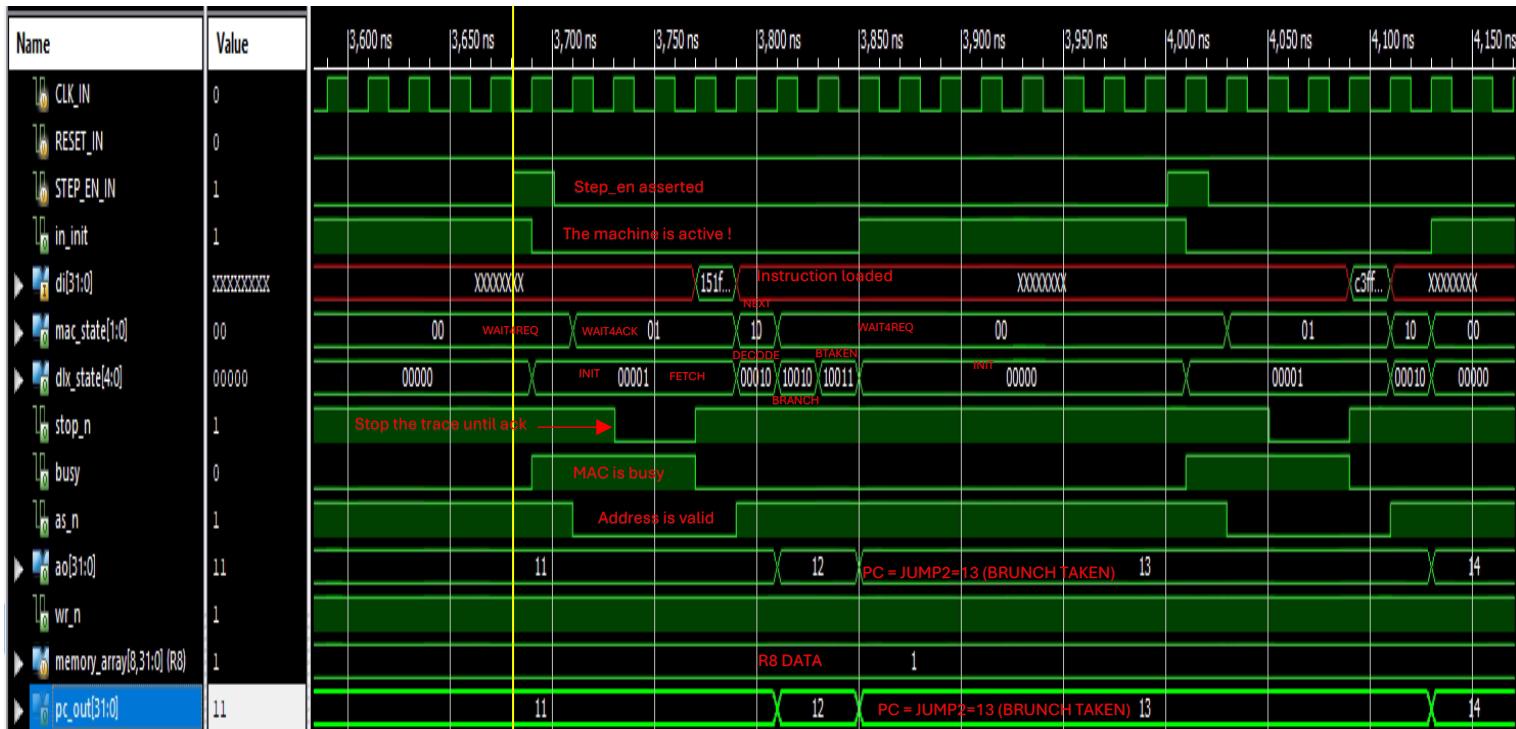


Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 17 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the BRANCH state.
- Since the condition met (R3==0) cycle after that the state machine will move to the BTAKEN state.
- Cycle after we can see that the pc updated with the value: pc = jump4=19 successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

BNEZ simulation waveform:

Operation used for this simulation: **151F0001 // bnez R8 jump2**



Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 11 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the BRANCH state.
- Since the condition met (R8!=0) cycle after that the state machine will move to the BTAKEN state.
- Cycle after we can see that the PC updated with the value: pc = jump4=19 successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

JR simulation waveform:

Operation used for this simulation: **587F0000 // jr R3**

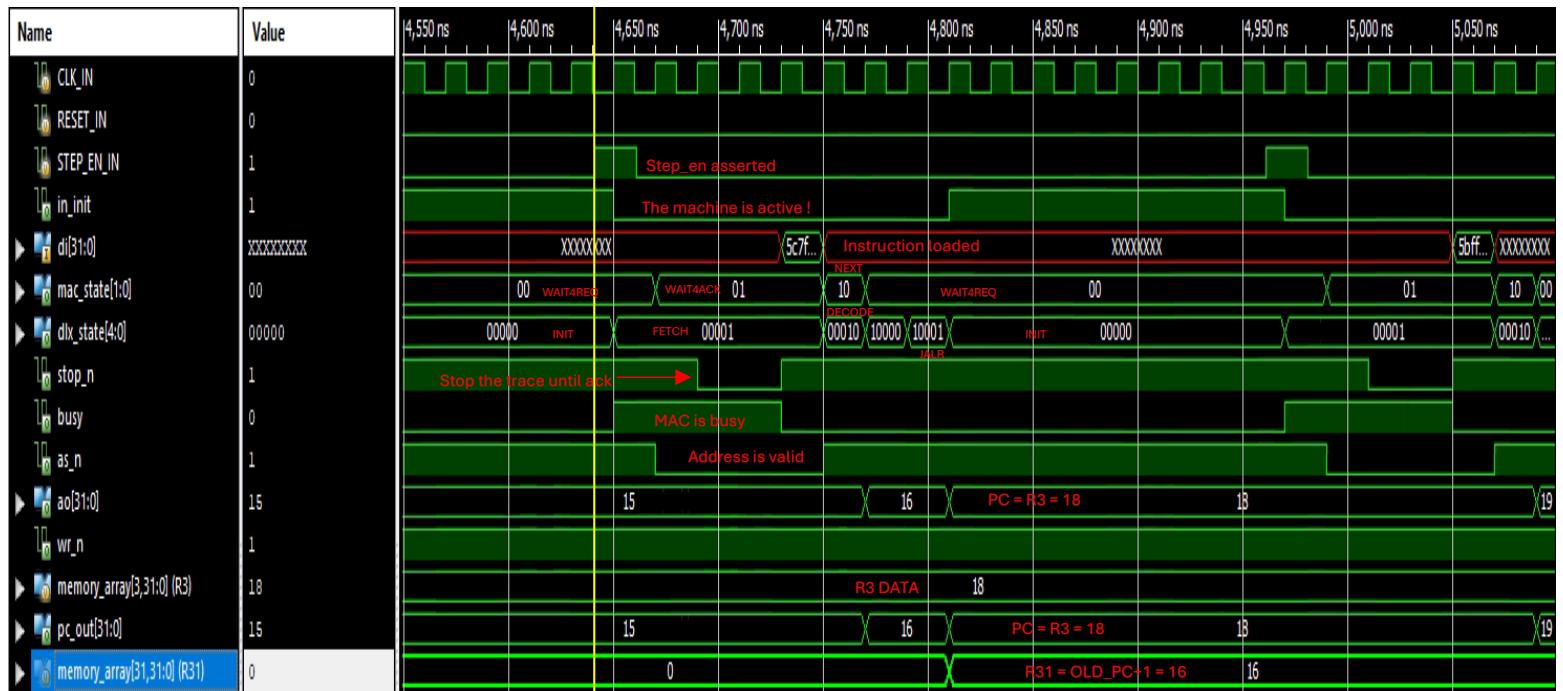


Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 15 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the JR state.
- Cycle after we can see that the PC updated with the value: pc = R3=18 successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

JALR simulation waveform:

Operation used for this simulation: **5C7F0000 // jalr R3**

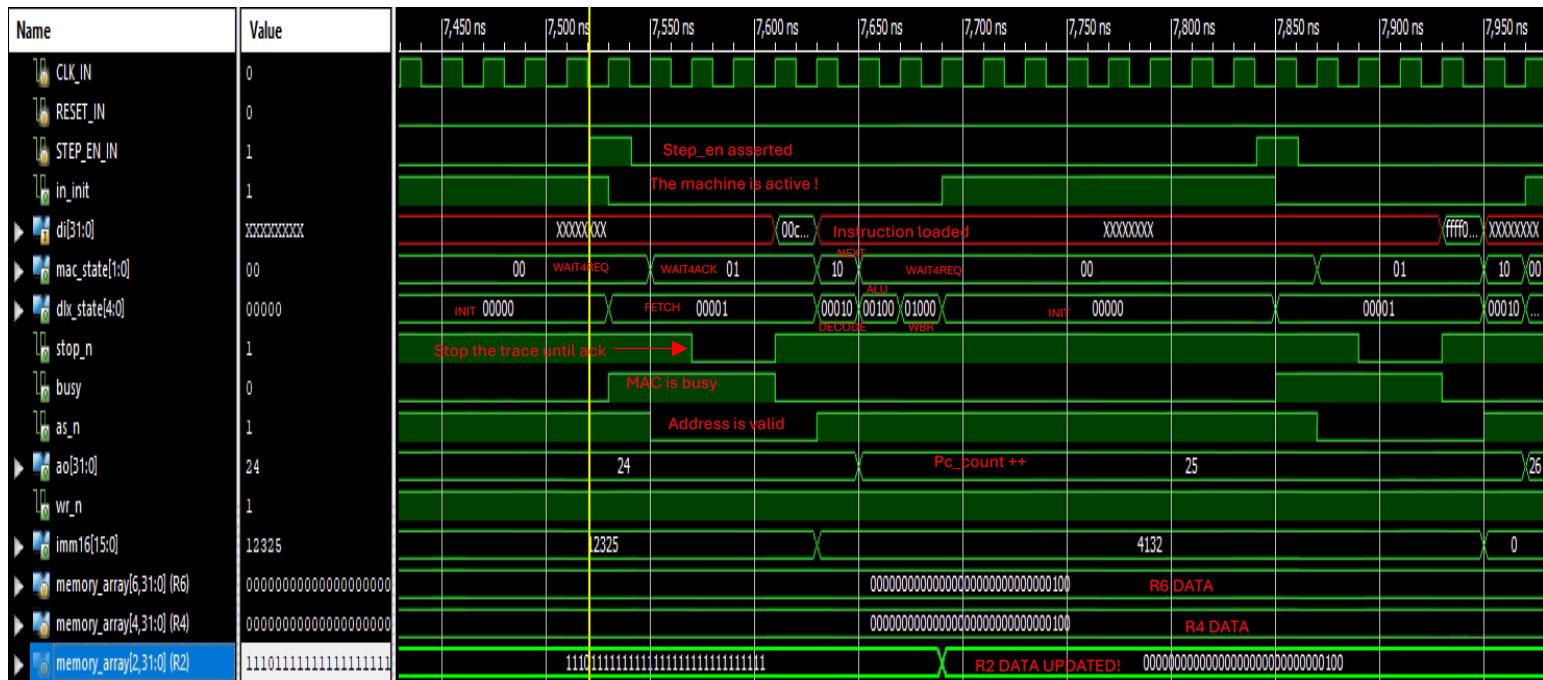


Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 15 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the SAVEPC state.
- Cycle after that the state machine moves to the JALR state.
- Cycle after we can see that the PC updated with the value: PC = R3=18 successfully and R31 saved the old PC + 1 which is 16.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

OR simulation waveform:

Operation used for this simulation: **00C41025 // or R2 R6 R4**

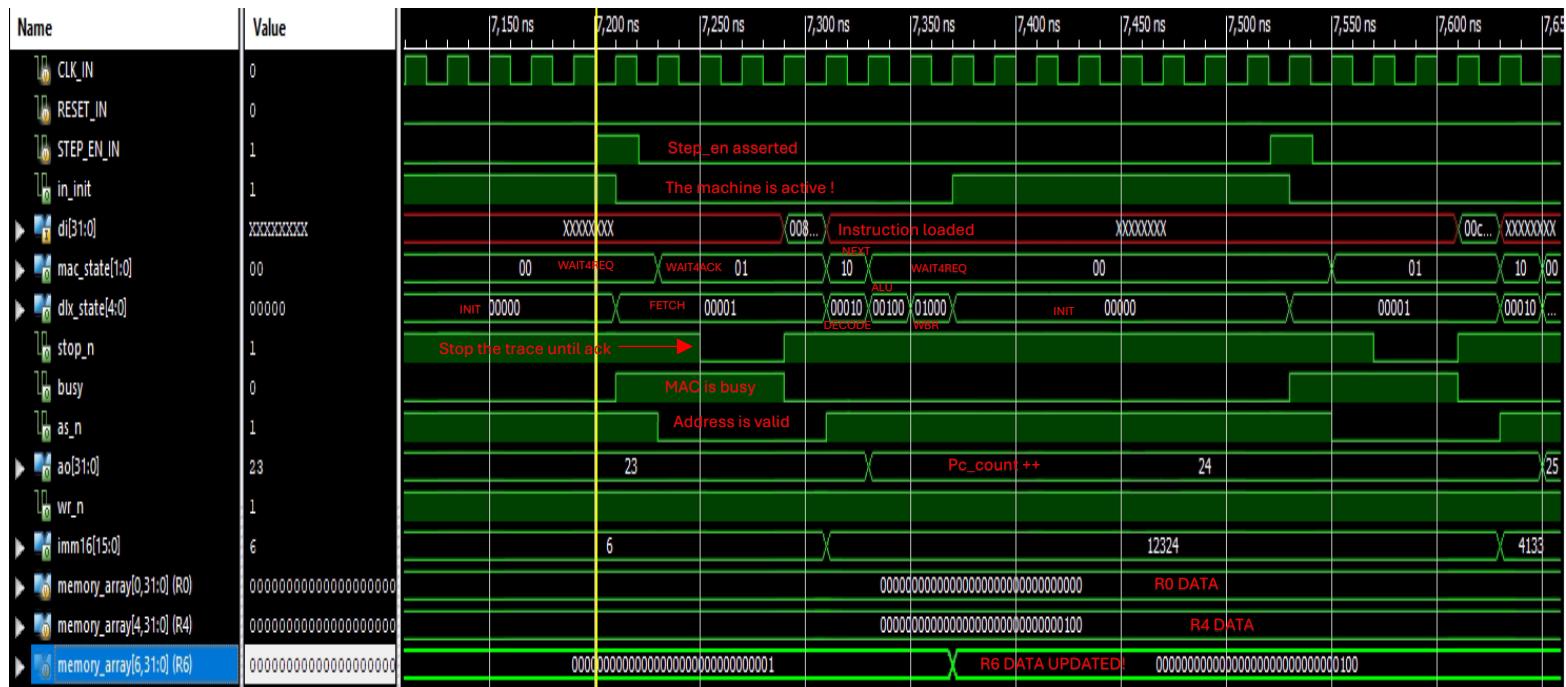


Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 24 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the ALU state.
- Cycle after that the state machine moves to the WBR state.
- Cycle after we can see that the R3 updated with the value: R5 or R1= 3'b100 successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

XOR simulation waveform:

Operation used for this simulation: **00803024 //** xor R6 R4 R0

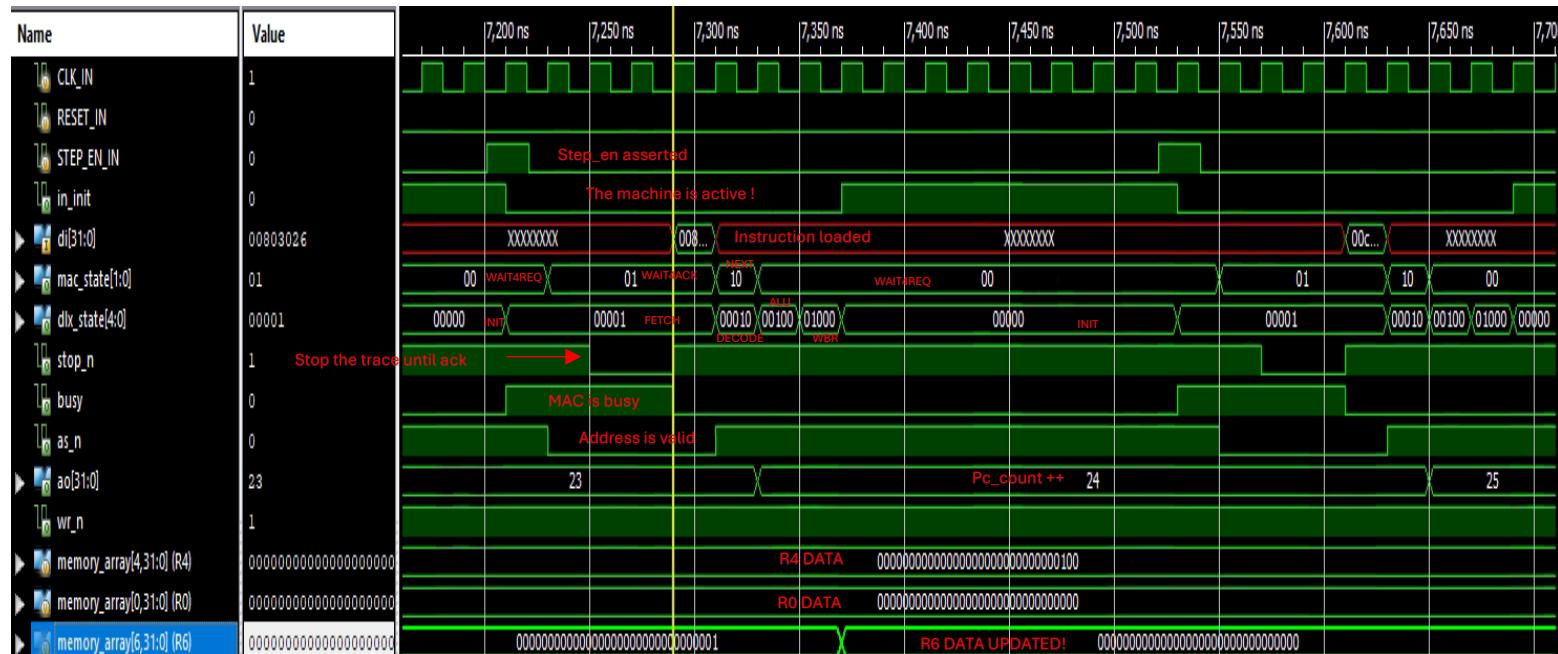


Waveform explanation:

- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 23 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the ALU state.
- Cycle after that the state machine moves to the WBR state.
- Cycle after we can see that the R3 updated with the value: R0=0 xor R4= R4=3'B100 successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

AND simulation waveform:

Operation used for this simulation: **00803026 // and R6 R4 R0**

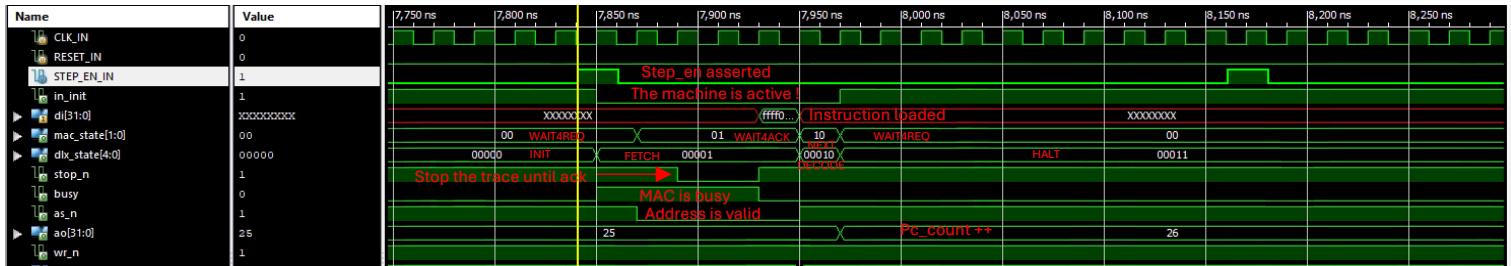


Waveform explanation:

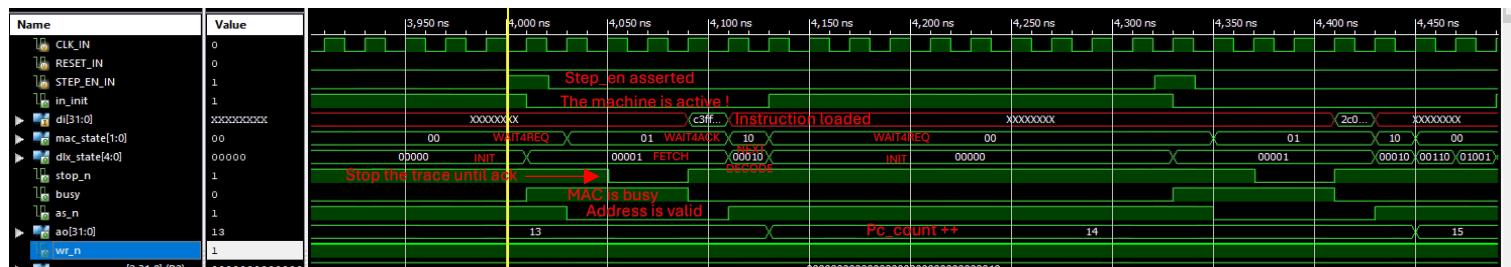
- We can see that when the reset asserted the machine starts in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 23 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully.
- Cycle after that the state machine moves to the ALU state.
- Cycle after that the state machine moves to the WBR state.
- Cycle after we can see that the R3 updated with the value: R0=0 AND R4= R0 = 0 successfully.
- Cycle after the state machine is in INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

Miscellaneous Instructions:

Halt:



Special-nop:



Waveform explanation for both waveforms:

- We can see that the when the reset asserted the machine start in the INIT state.
- We can see that when the step_en asserted the state moves to 01 state (Fetch), the in_init, AS_N moves to 0 while WR_N moves to 1 the machine in Reading active mode.
- AO = PC = 23 to load the relevant instruction.
- We can see that cycle after the state moves to DECODE state.
- Cycle after the MAC state moves to wait4ack, stop_N de-asserted to tell the monitor to stop the recording until the ACK received.
- When the ack received MAC state moves to the NEXT, the stop_N asserted again and the instruction loaded successfully
- Only for the Halt waveform:** Cycle after that the halt waveform state machine move to the HALT state, the machine will stuck in this state until there reset asserted.
- Only for the Special-nop waveform:** Cycle after that the state machine move to the INIT state, MAC in WAIT4REQ state, in_init asserted and the machine in idle state until the next step_en.

RESA Test And Simulations:

Labels data :

```
Label and addresses Report:  
CPU RAM Address: 0x1a0  
LA RAM Address: 0x1c0  
  
Slave Labels:  
laram @ 0x1c0  
STATUS @ 0x1e0  
mdo @ 0x1a0  
PC @ 0x180  
  
Graphic Labels:  
stepen @ 0x0  
ininit @ 0x1  
asn @ 0x2  
wrout @ 0x3  
ackn @ 0x4  
stopen @ 0x5  
dixstate0 @ 0x6  
dixstate1 @ 0x7  
dixstate2 @ 0x8  
dixstate3 @ 0x9  
dixstate4 @ 0xa  
macstate0 @ 0xb  
macstate1 @ 0xc  
mr @ 0xd  
mw @ 0xe  
add @ 0xf  
test @ 0x10  
ltype @ 0x15  
jlink @ 0x1c  
shift @ 0x1a  
right @ 0x1b  
busy @ 0x18  
s1sel0 @ 0x13  
s1sel1 @ 0x14  
s2sel0 @ 0x11  
s2sel1 @ 0x12  
mdrsel @ 0x16  
asel @ 0x1c  
dintsel @ 0x19  
  
Memory Labels:  
rd @ 0xd
```

The four slave labels are:

1. **LARAM:** Stores sampled values in the Logic Analyzer's RAM during instruction execution.
2. **STATUS:** Indicates the number of rows with relevant data in the Logic Analyzer's RAM.
3. **mdo:** Master Data Out bus, used by the master device to send data to slave devices.
4. **PC:** Program Counter, holds the address of the next instruction to be fetched and executed.

TEST:

Next, we have run a test on the FPGA and showed the waveform and registers for each instruction in the instruction list using RESA (the list appears in the control part).

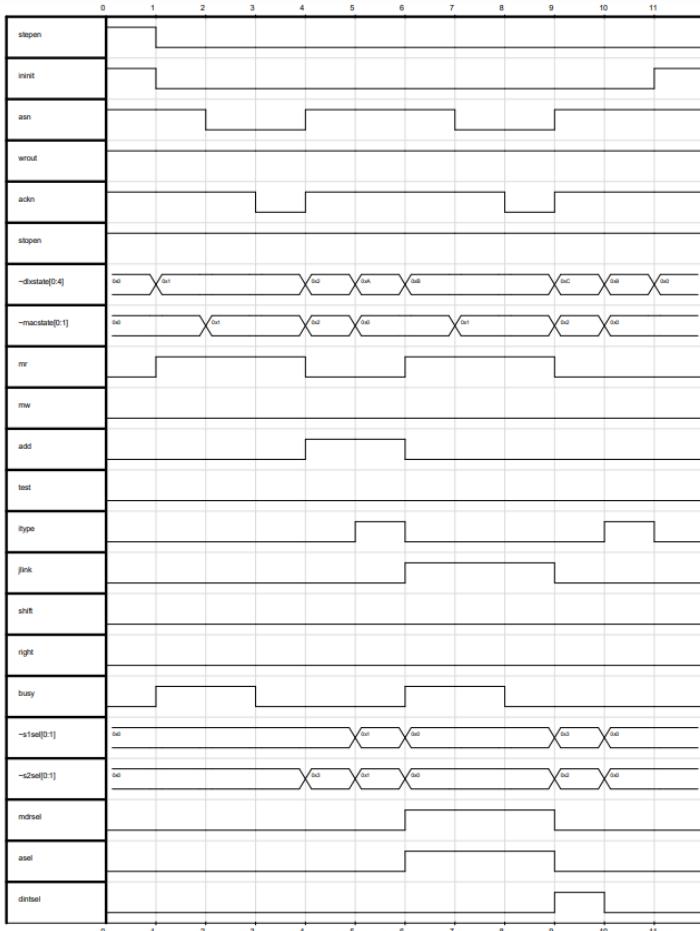
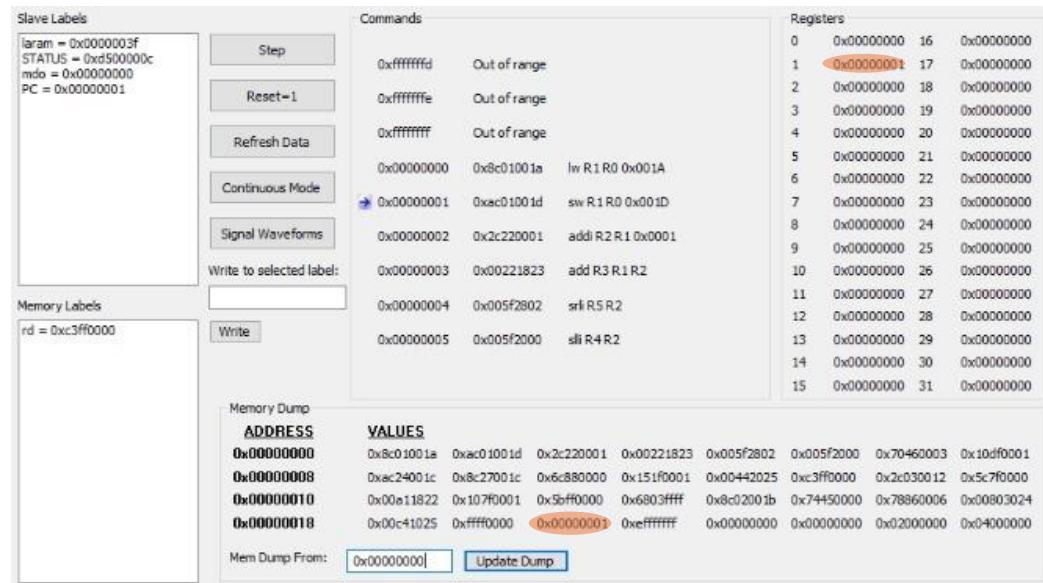
In this part we did not add notes and explanations like we did before, and that's because we had the same result that we got in the simulations, we can see that the state machine behaves as expected and as described in the state diagram (in the control part), the signals behave like the table that we added also in the control part.

we showed how the design work on FPGA and how the register change according to the asserted instruction.

Starting from the next page we can see the test results.

##Note: we had a technical issue with our monitor, so we copied a new one from the other group on our pc (D5), and accidentally we did not change the group id (and that's why D5 appears in the RESA status instead of C5).

Lw R1 R0 0x001



the instruction is **Lw R1 R0 0x001**

which translates to :

$$R(RD) = m(imm + R(RS1))$$

in our case RS1 = 0 and the imm = 0x001A

so the dlx will load address 0x001A from the memory to R1

we see the dlx loaded the data : 0x00000001 from the memory to R1 highlighted in orange

sw R1 R0 0x001A

Slave Labels

```

laram = 0x0000003f
STATUS5 = 0xd500000b
mido = 0x00000000
PC = 0x00000002

```

Commands

	Step	Reset=1	Refresh Data	Continuous Mode	Signal Waveforms	Write to selected label:
0xffffffffe	Out of range					
0xfffffffff		Out of range				
0x00000000	0x8c01001a lw R1 R0 0x001A					
0x00000001		0xac01001d sw R1 R0 0x001D				
0x00000002			0x2c220001 addi R2 R1 0x0001			
0x00000003				0x00221823 add R3 R1.R2		
0x00000004					0x005f2802 srl R5 R2	
0x00000005						0x005f2000 sli R4 R2
0x00000006						0x70460003 sli R6.R2 0x0003

Registers

0	0x00000000	16	0x00000000
1	0x00000001	17	0x00000000
2	0x00000000	18	0x00000000
3	0x00000000	19	0x00000000
4	0x00000000	20	0x00000000
5	0x00000000	21	0x00000000
6	0x00000000	22	0x00000000
7	0x00000000	23	0x00000000
8	0x00000000	24	0x00000000
9	0x00000000	25	0x00000000
10	0x00000000	26	0x00000000
11	0x00000000	27	0x00000000
12	0x00000000	28	0x00000000
13	0x00000000	29	0x00000000
14	0x00000000	30	0x00000000
15	0x00000000	31	0x00000000

Memory Labels

```

rd = 0xc3ff0000

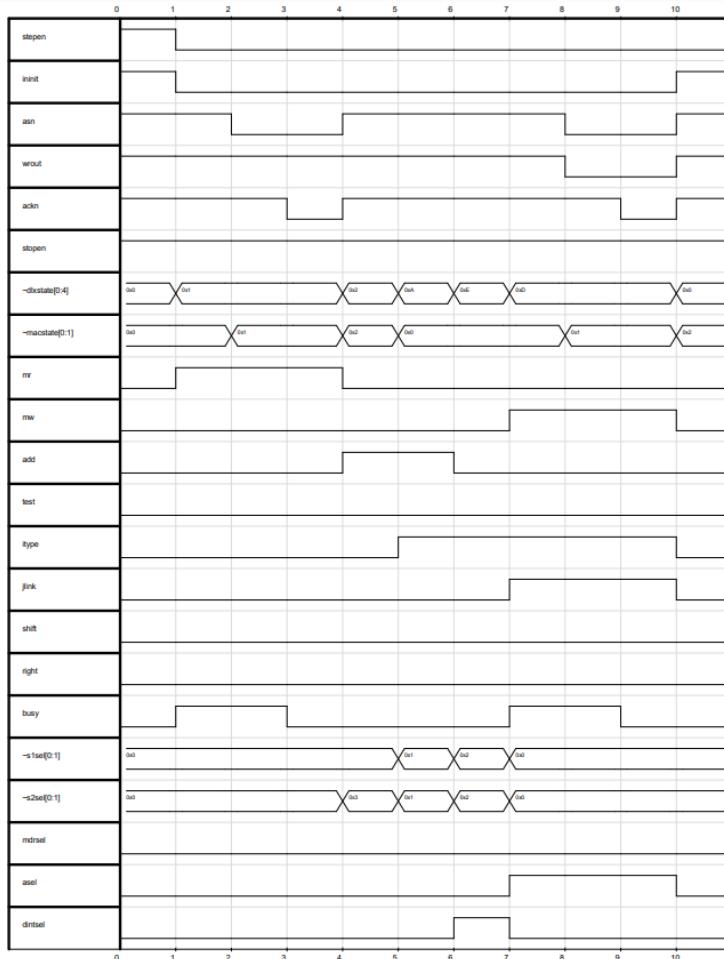
```

Write

Memory Dump

ADDRESS	VALUES
0x00000000	0x8c01001a 0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001
0x00000008	0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x00442025 0xc3ff0000 0x2c030012 0x5c7f0000
0x00000010	0x00a11822 0x107f0001 0x5ffff0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024
0x00000018	0x00c41025 0xfffff0000 0x00000001 0xffffffff 0x00000000 0x00000001 0x02000000 0x04000000

Mem Dump From: 0x00000000 Update Dump



the instruction is **sw R1 R0**

0x001A

which translates to :

$$M(\text{imm} + R1) = RD$$

$$\text{imm} = 0x001D$$

which means the dlx should save the data of R1 in memory slot 0x001D

0x001D

we see that after the step the address 0x001D have similar data to R1 which is 1

Addi R2 R1 0x0001

Slave Labels

lalarm = 0x0000003f
STATUS = 0xd5000008
mdo = 0x00000000
PC = 0x00000003

Memory Labels

rd = 0xc3ff0000

Commands

Step	0xffffffff	Out of range
Reset-1	0x00000000	0x8c01001a lw R1 R0 0x001A
Refresh Data	0x00000001	0xac01001d sw R1 R0 0x001D
Continuous Mode	0x00000002	0x2c220001 addi R2 R1 0x0001
Signal Waveforms	0x00000003	0x00221823 add R3 R1 R2
Write to selected label:	0x00000004	0x005f2802 srl R5 R2
	0x00000005	0x005f2000 sll R4 R2
	0x00000006	0x70460003 sfb R6 R2 0x0003
	0x00000007	0x10df0001 beqz R6 0x0009

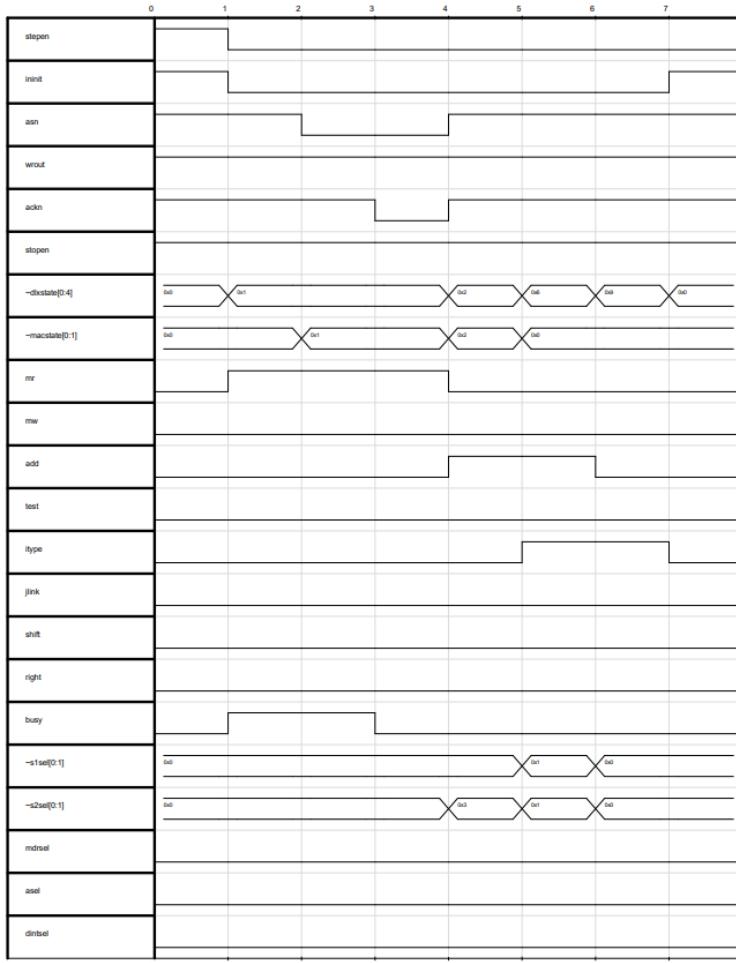
Registers

0	0x00000000	16	0x00000000
1	0x00000001	17	0x00000000
2	0x00000002	18	0x00000000
3	0x00000000	19	0x00000000
4	0x00000000	20	0x00000000
5	0x00000000	21	0x00000000
6	0x00000000	22	0x00000000
7	0x00000000	23	0x00000000
8	0x00000000	24	0x00000000
9	0x00000000	25	0x00000000
10	0x00000000	26	0x00000000
11	0x00000000	27	0x00000000
12	0x00000000	28	0x00000000
13	0x00000000	29	0x00000000
14	0x00000000	30	0x00000000
15	0x00000000	31	0x00000000

Memory Dump

ADDRESS	VALUES
0x00000000	0x8c01001a 0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001
0x00000008	0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x00442025 0xc3ff0000 0x2c030012 0x5c7f0000
0x00000010	0x00a11822 0x107f0001 0x5eff0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024
0x00000018	0x00c41025 0xfffff0000 0x00000001 0xffffffff 0x00000000 0x00000001 0x02000000 0x04000000

Mem Dump From:



the instruction is **Addi R2 R1 0x0001**

which translates to :

RD = RS1 + imm

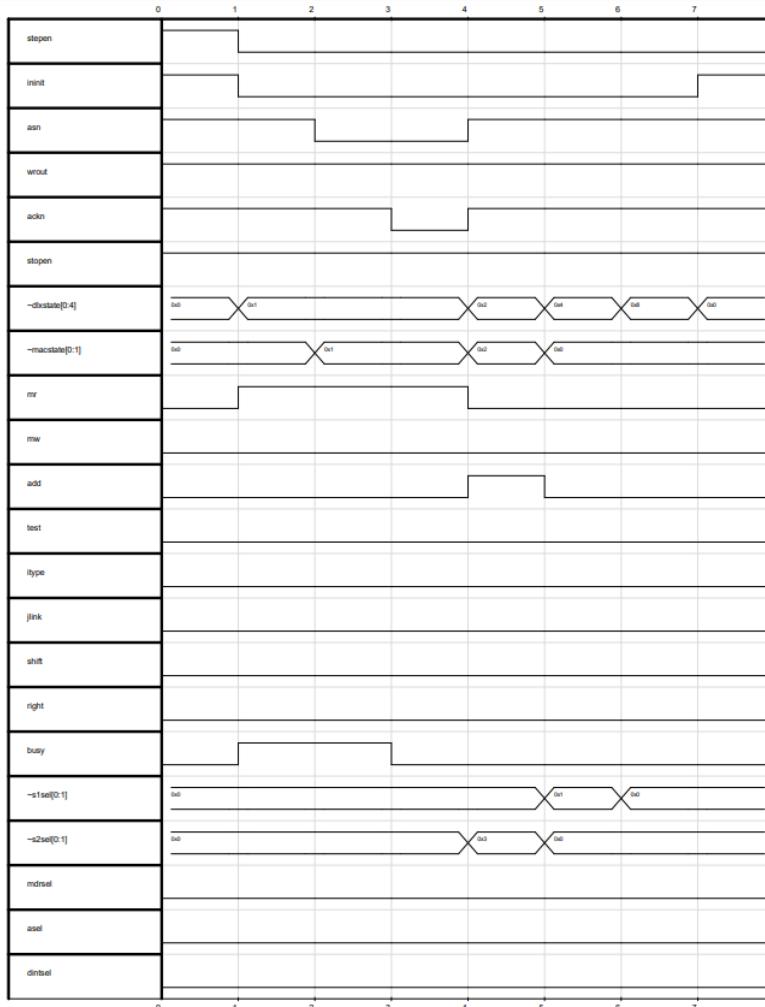
imm = 0x0001

R1 has 0x00000001 in it

We see that the destination register R2 after the step now have 0x00000002 in it.

Add R3 R1 R2

Slave Labels	Commands	Registers
laram = 0x0000003f STATUS = 0xd5000008 mdr = 0x00000000 PC = 0x00000004	Step Reset=1 Refresh Data Continuous Mode Signal Waveforms Write to selected label: rd = 0xc3ff0000 Write	0 0x00000000 16 0x00000000 1 0x00000001 17 0x00000000 2 0x00000002 18 0x00000000 3 0x00000003 19 0x00000000 4 0x00000000 20 0x00000000 5 0x00000000 21 0x00000000 6 0x00000000 22 0x00000000 7 0x00000000 23 0x00000000 8 0x00000000 24 0x00000000 9 0x00000000 25 0x00000000 10 0x00000000 26 0x00000000 11 0x00000000 27 0x00000000 12 0x00000000 28 0x00000000 13 0x00000000 29 0x00000000 14 0x00000000 30 0x00000000 15 0x00000000 31 0x00000000
Memory Labels	Memory Dump ADDRESS 0x00000000 0x00000008 0x00000010 0x00000018 VALUES 0x8c01001a 0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001 0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x00442025 0xc3ff0000 0x2c030012 0x5c7f0000 0x000e11822 0x107f0001 0x5bfff0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024 0x000c41025 0xfffff0000 0x00000001 0xffffffff 0x00000000 0x00000001 0x02000000 0x40000000	
	Mem Dump From: 0x00000000 Update Dump	



the instruction is **Add R3 R1 R2**

which translates into:

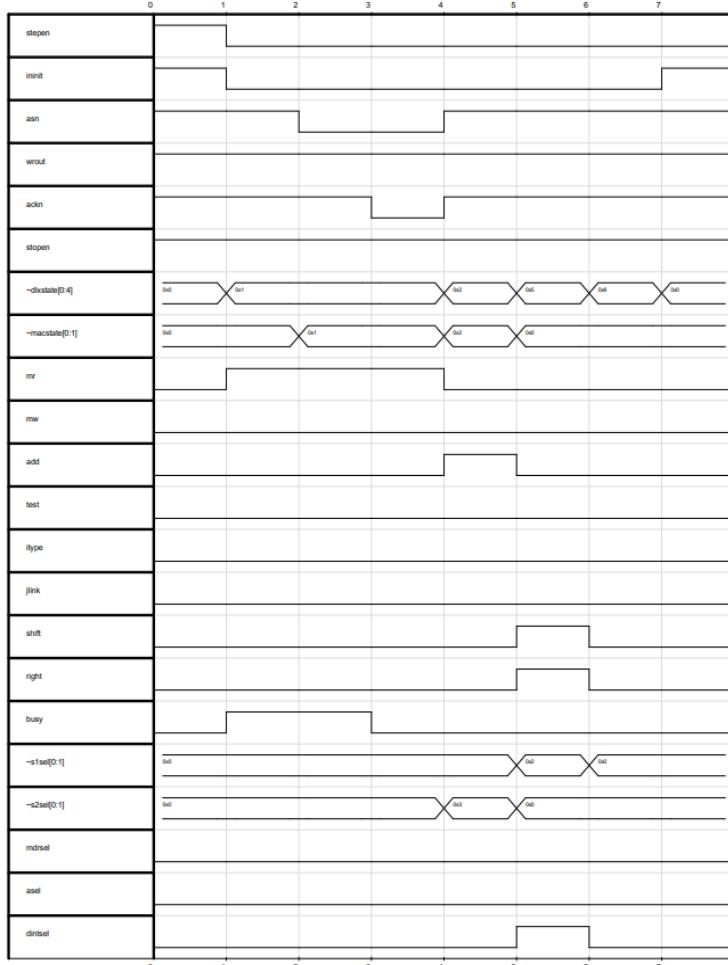
$$RD = RS1 + RS2$$

$$RS1 = R1 \text{ has } 0x01 \text{ in it,}$$

$$RS2 = R2 \text{ has } 0x02 \text{ in it,}$$

As we can see after the step the RD=R3 has 0x03 which is the sum of R1 and R2

Srli R5 R2



the instruction is **Srli R5 R2**
which translates to:
 $RD = (RS1 >> 1)$ which is a Right shift operation
RS1 is R2
The RD = R5 we can see is the value of R2 shifter one to the right and loaded to R5.

Slli R4 R2

Slave Labels

```

laram = 0x0000003f
STATUS = 0xd5000008
mdo = 0x00000000
PC = 0x00000006

```

Memory Labels

```

rd = 0xc3ff0000

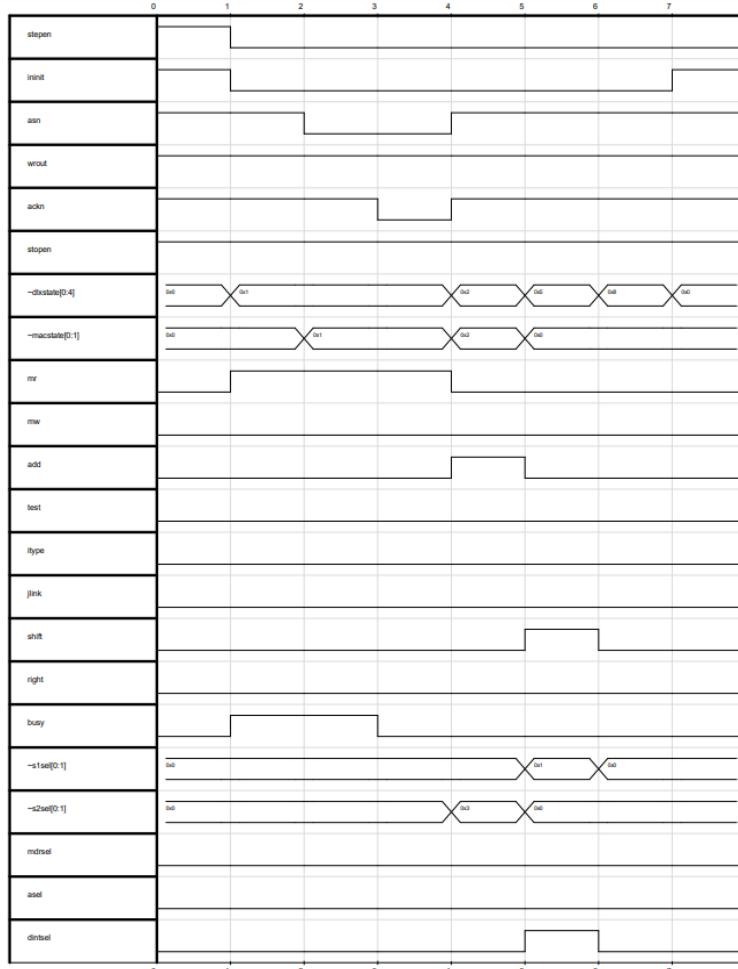
```

Step	Commands	Registers
Reset-1	0x00000002 0x2c220001 addi R2 R1 0x0001	0 0x00000000 16 0x00000000
Refresh Data	0x00000003 0x00221823 add R3 R1 R2	1 0x00000001 17 0x00000000
Continuous Mode	0x00000004 0x005f2802 srl R5 R2	2 0x00000002 18 0x00000000
Signal Waveforms	0x00000005 0x005f2000 slli R4 R2	3 0x00000003 19 0x00000000
Write to selected label:	0x00000006 0x70460003 sll R6 R2 0x0003	4 0x00000004 20 0x00000000
	0x00000007 0x10df0001 beqz R6 0x0009	5 0x00000001 21 0x00000000
	0x00000008 0xac24001c sw R4 R1 0x001C	6 0x00000000 22 0x00000000
	0x00000009 0x8c27001c lw R7 R1 0x001C	7 0x00000000 23 0x00000000
Write	0x0000000a 0x6c880000 sgei R8 R4 0x0000	8 0x00000000 24 0x00000000
		9 0x00000000 25 0x00000000
		10 0x00000000 26 0x00000000
		11 0x00000000 27 0x00000000
		12 0x00000000 28 0x00000000
		13 0x00000000 29 0x00000000
		14 0x00000000 30 0x00000000
		15 0x00000000 31 0x00000000

Memory Dump

ADDRESS	VALUES
0x00000000	0xb01001a 0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001
0x00000008	0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x00442025 0xc3ff0000 0x2c030012 0x5c7f0000
0x00000010	0x00a11822 0x107f0001 0x5eff0000 0x6803ffff 0x8c2001b 0x74450000 0x78860006 0x0803024
0x00000018	0x00c41025 0xfffff0000 0x00000001 0xffffffff 0x00000000 0x00000001 0x02000000 0x04000000

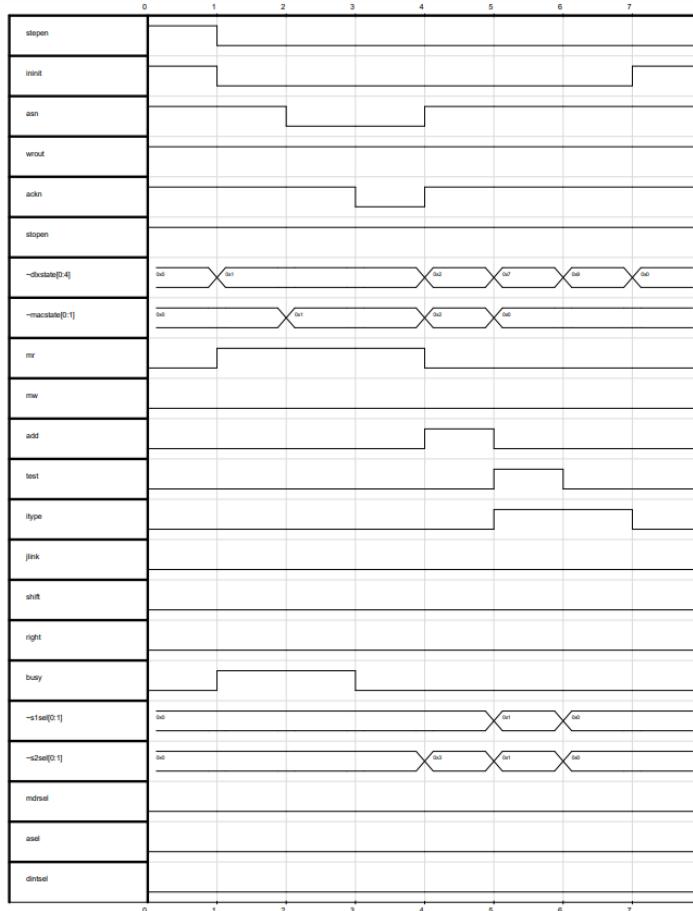
Mem Dump From:



the instruction is **Slli R4 R2**
which translates to:
 $RD = (RS1 \ll 1)$ which is a left shift operation
RS1 is R2
The RD = R4 we can see is the value of R2 shifter one to the left

Slti R6 R2 0x0003

Slave Labels	Commands	Registers																																																																
<pre> laram = 0x0000003f STATUS = 0xd5000008 mdo = 0x00000000 PC = 0x00000007 </pre>	<input type="button" value="Step"/> <input type="button" value="Reset-1"/> <input type="button" value="Refresh Data"/> <input type="button" value="Continuous Mode"/> <input checked="" type="button" value="Signal Waveforms"/> <input type="button" value="Write to selected label:"/> <input type="button" value="Write"/>	<table border="1"> <tr><td>0</td><td>0x00000000</td><td>16</td><td>0x00000000</td></tr> <tr><td>1</td><td>0x00000001</td><td>17</td><td>0x00000000</td></tr> <tr><td>2</td><td>0x00000002</td><td>18</td><td>0x00000000</td></tr> <tr><td>3</td><td>0x00000003</td><td>19</td><td>0x00000000</td></tr> <tr><td>4</td><td>0x00000004</td><td>20</td><td>0x00000000</td></tr> <tr><td>5</td><td>0x00000001</td><td>21</td><td>0x00000000</td></tr> <tr><td>6</td><td>0x00000001</td><td>22</td><td>0x00000000</td></tr> <tr><td>7</td><td>0x00000000</td><td>23</td><td>0x00000000</td></tr> <tr><td>8</td><td>0x00000000</td><td>24</td><td>0x00000000</td></tr> <tr><td>9</td><td>0x00000000</td><td>25</td><td>0x00000000</td></tr> <tr><td>10</td><td>0x00000000</td><td>26</td><td>0x00000000</td></tr> <tr><td>11</td><td>0x00000000</td><td>27</td><td>0x00000000</td></tr> <tr><td>12</td><td>0x00000000</td><td>28</td><td>0x00000000</td></tr> <tr><td>13</td><td>0x00000000</td><td>29</td><td>0x00000000</td></tr> <tr><td>14</td><td>0x00000000</td><td>30</td><td>0x00000000</td></tr> <tr><td>15</td><td>0x00000000</td><td>31</td><td>0x00000000</td></tr> </table>	0	0x00000000	16	0x00000000	1	0x00000001	17	0x00000000	2	0x00000002	18	0x00000000	3	0x00000003	19	0x00000000	4	0x00000004	20	0x00000000	5	0x00000001	21	0x00000000	6	0x00000001	22	0x00000000	7	0x00000000	23	0x00000000	8	0x00000000	24	0x00000000	9	0x00000000	25	0x00000000	10	0x00000000	26	0x00000000	11	0x00000000	27	0x00000000	12	0x00000000	28	0x00000000	13	0x00000000	29	0x00000000	14	0x00000000	30	0x00000000	15	0x00000000	31	0x00000000
0	0x00000000	16	0x00000000																																																															
1	0x00000001	17	0x00000000																																																															
2	0x00000002	18	0x00000000																																																															
3	0x00000003	19	0x00000000																																																															
4	0x00000004	20	0x00000000																																																															
5	0x00000001	21	0x00000000																																																															
6	0x00000001	22	0x00000000																																																															
7	0x00000000	23	0x00000000																																																															
8	0x00000000	24	0x00000000																																																															
9	0x00000000	25	0x00000000																																																															
10	0x00000000	26	0x00000000																																																															
11	0x00000000	27	0x00000000																																																															
12	0x00000000	28	0x00000000																																																															
13	0x00000000	29	0x00000000																																																															
14	0x00000000	30	0x00000000																																																															
15	0x00000000	31	0x00000000																																																															
Memory Labels	Memory Dump																																																																	
rd = 0xc3ff0000	<table border="1"> <thead> <tr> <th>ADDRESS</th> <th>VALUES</th> </tr> </thead> <tbody> <tr><td>0x00000000</td><td>0xb01001a 0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001</td></tr> <tr><td>0x00000008</td><td>0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x0442025 0xc3ff0000 0x2c030012 0x5c7f0000</td></tr> <tr><td>0x00000010</td><td>0x00e11822 0x107f0001 0x5ff0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024</td></tr> <tr><td>0x00000018</td><td>0x00c41025 0xfffff0000 0x00000001 0xffffffff 0x00000000 0x00000001 0x02000000 0x04000000</td></tr> </tbody> </table>	ADDRESS	VALUES	0x00000000	0xb01001a 0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001	0x00000008	0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x0442025 0xc3ff0000 0x2c030012 0x5c7f0000	0x00000010	0x00e11822 0x107f0001 0x5ff0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024	0x00000018	0x00c41025 0xfffff0000 0x00000001 0xffffffff 0x00000000 0x00000001 0x02000000 0x04000000																																																							
ADDRESS	VALUES																																																																	
0x00000000	0xb01001a 0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001																																																																	
0x00000008	0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x0442025 0xc3ff0000 0x2c030012 0x5c7f0000																																																																	
0x00000010	0x00e11822 0x107f0001 0x5ff0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024																																																																	
0x00000018	0x00c41025 0xfffff0000 0x00000001 0xffffffff 0x00000000 0x00000001 0x02000000 0x04000000																																																																	
	<input type="button" value="Mem Dump From:"/> <input type="text" value="0x00000000"/> <input type="button" value="Update Dump"/>																																																																	



the instruction is **Slti R6 R2 0x0003**
which translates to:

RD = RS1 < imm

We can see that the value of R2 is smaller than 0x03 thus we see 0x01 in the RD which is R6

Beqz R6 0x0009

Slave Labels

```

laram = 0x0000003f
STATUS = 0xd5000007
mdio = 0x00000000
PC = 0x00000008

```

Commands

Step	0x00000004	0x005f2802	srl R5 R2	
Reset=1	0x00000005	0x005f2000	sll R4 R2	
Refresh Data	0x00000006	0x70460003	slt R6 R2 0x0003	
Continuous Mode	0x00000007	0x10df0001	beqz R6 0x0009	
Signal Waveforms	0x00000008	0xac24001c	sw R4 R1 0x001C	
Write to selected label:	0x00000009	0x8c27001c	lw R7 R1 0x001C	
	0x0000000a	0x6c880000	sgel R8 R4 0x0000	
	0x0000000b	0x151f0001	bnez R8 0x000D	
	0x0000000c	0x00442025	or R4 R2 R4	

Registers

0	0x00000000	16	0x00000000	
1	0x00000001	17	0x00000000	
2	0x00000002	18	0x00000000	
3	0x00000003	19	0x00000000	
4	0x00000004	20	0x00000000	
5	0x00000001	21	0x00000000	
6	0x00000001	22	0x00000000	
7	0x00000000	23	0x00000000	
8	0x00000000	24	0x00000000	
9	0x00000000	25	0x00000000	
10	0x00000000	26	0x00000000	
11	0x00000000	27	0x00000000	
12	0x00000000	28	0x00000000	
13	0x00000000	29	0x00000000	
14	0x00000000	30	0x00000000	
15	0x00000000	31	0x00000000	

Memory Labels

```

rd = 0xc3ff0000

```

Commands

Step	0x00000004	0x005f2802	srl R5 R2	
Reset=1	0x00000005	0x005f2000	sll R4 R2	
Refresh Data	0x00000006	0x70460003	slt R6 R2 0x0003	
Continuous Mode	0x00000007	0x10df0001	beqz R6 0x0009	
Signal Waveforms	0x00000008	0xac24001c	sw R4 R1 0x001C	
Write to selected label:	0x00000009	0x8c27001c	lw R7 R1 0x001C	
	0x0000000a	0x6c880000	sgel R8 R4 0x0000	
	0x0000000b	0x151f0001	bnez R8 0x000D	
	0x0000000c	0x00442025	or R4 R2 R4	

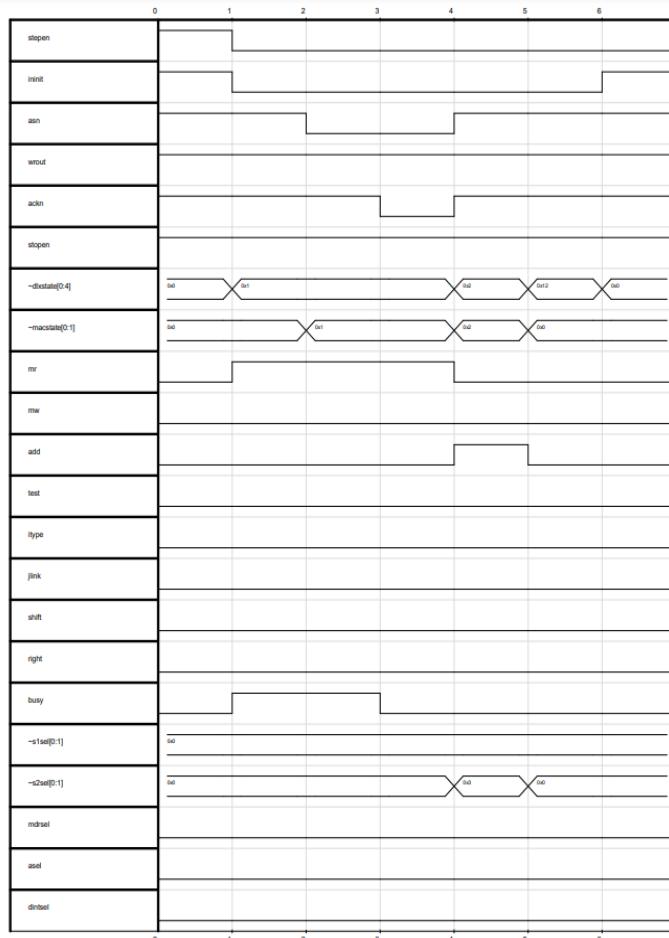
Registers

0	0x00000000	16	0x00000000	
1	0x00000001	17	0x00000000	
2	0x00000002	18	0x00000000	
3	0x00000003	19	0x00000000	
4	0x00000004	20	0x00000000	
5	0x00000001	21	0x00000000	
6	0x00000001	22	0x00000000	
7	0x00000000	23	0x00000000	
8	0x00000000	24	0x00000000	
9	0x00000000	25	0x00000000	
10	0x00000000	26	0x00000000	
11	0x00000000	27	0x00000000	
12	0x00000000	28	0x00000000	
13	0x00000000	29	0x00000000	
14	0x00000000	30	0x00000000	
15	0x00000000	31	0x00000000	

Memory Dump

ADDRESS	VALUES
0x00000000	0x8c01001a 0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001
0x00000008	0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x00442025 0xc3ff0000 0x2c030012 0x5c7f0000
0x00000010	0x00e11822 0x107f0001 0x5ff00000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024
0x00000018	0x00c41025 0xfffff0000 0x00000001 0xffffffff 0x00000000 0x02000000 0x04000000

Mem Dump From: 0x00000000 Update Dump



the instruction is Beqz R6 0x0009 which is a branch instruction checks if the value of R6 is equal to zero the PC change it address in our dlx we can see that the value of R6 is not zero thus the instruction didn't jump

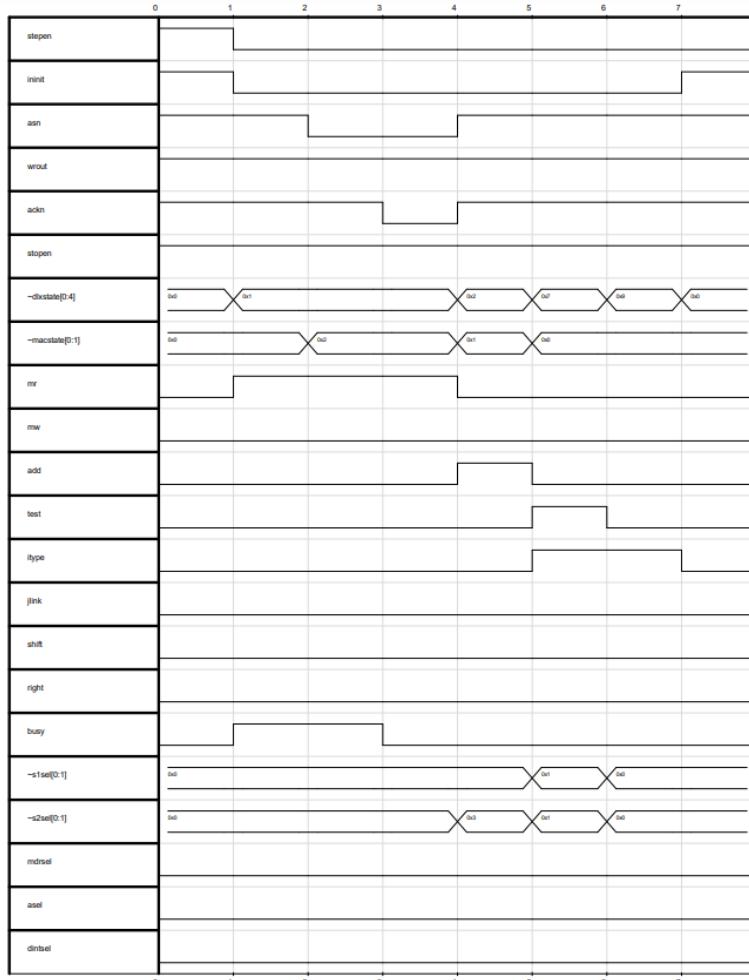
Sgei R8 R4 0x0000

Slave Labels	Commands	Registers
laram = 0x0000003f STATUS = 0xd5000008 mdo = 0x00000000 PC = 0x0000000b		
<input type="button" value="Step"/>	0x00000007 0x10df0001 beqz R6 0x0009	0 0x00000000 16 0x00000000
<input type="button" value="Reset=1"/>	0x00000008 0xac24001c srl R4 R1 0x001C	1 0x00000001 17 0x00000000
<input type="button" value="Refresh Data"/>	0x00000009 0x8c27001c lw R7 R1 0x001C	2 0x00000002 18 0x00000000
<input type="button" value="Continuous Mode"/>	0x0000000a 0x6c880000 sgei R8 R4 0x0000	3 0x00000003 19 0x00000000
<input type="button" value="Signal Waveforms"/>	0x0000000b 0x151f0001 bnez R8 0x0000	4 0x00000004 20 0x00000000
<input type="button" value="Write to selected label:"/>	0x0000000c 0x04442025 or R4 R2 R4	5 0x00000001 21 0x00000000
<input type="button" value="Write"/>	0x0000000d 0xc3ff0000 special-nop	6 0x00000001 22 0x00000000
	0x0000000e 0x2c030012 addi R3 R0 0x0012	7 0x00000004 23 0x00000000
	0x0000000f 0x5c7f0000 jalr R3	8 0x00000001 24 0x00000000
		9 0x00000000 25 0x00000000
		10 0x00000000 26 0x00000000
		11 0x00000000 27 0x00000000
		12 0x00000000 28 0x00000000
		13 0x00000000 29 0x00000000
		14 0x00000000 30 0x00000000
		15 0x00000000 31 0x00000000

Memory Labels

ADDRESS	VALUES
0x00000000	0x8c01001a 0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001
0x00000008	0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x04442025 0xc3ff0000 0x2c030012 0x5c7f0000
0x00000010	0x00a11822 0x10700001 0x5bffff0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024
0x00000018	0x00c41025 0xfffff0000 0x00000001 0xffffffff 0x00000000 0x00000004 0x02000000 0x40000000

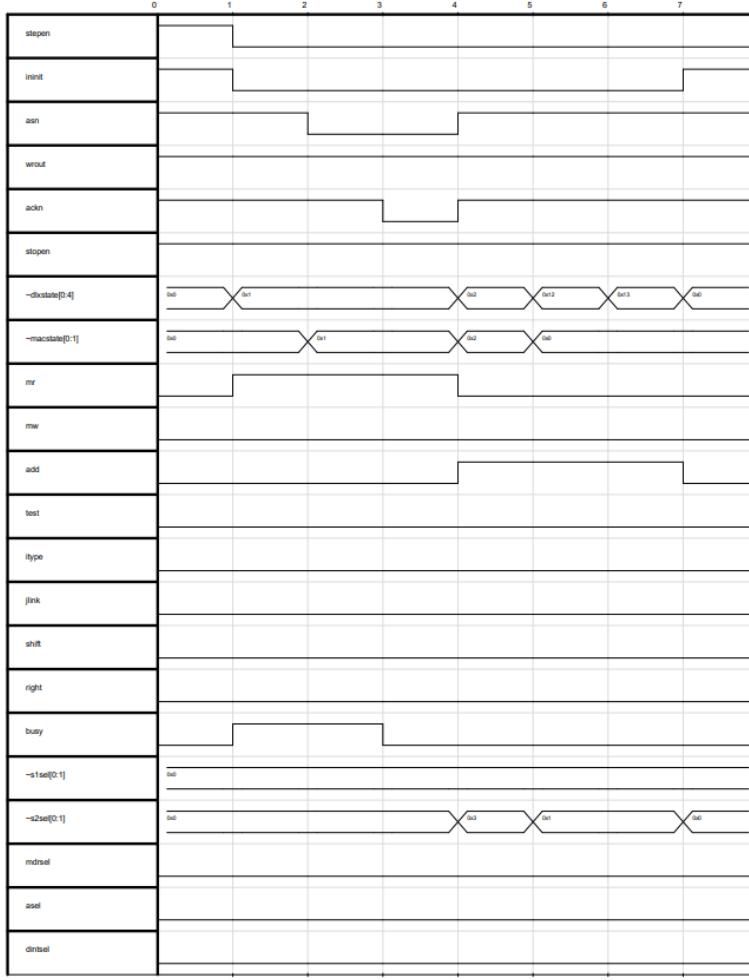
Mem Dump From:



the instruction is Sgei R8 R4 0x0000
which translates to:
 $RD = RS1 \geq imm$
We can see that R4 have the value 0x04 which is greater than the imm = 0 thus the value of the destination register R8 is 1 now

Bnez R8 0x000D

Slave Labels	Commands	Registers
larm = 0x0000003f STATUS = 0xd5000008 mdo = 0x00000000 PC = 0x0000000d	Step	0 0x00000000 16 0x00000000
	Reset=1	1 0x00000001 17 0x00000000
	Refresh Data	2 0x00000002 18 0x00000000
	Continuous Mode	3 0x00000003 19 0x00000000
	Signal Waveforms	4 0x00000004 20 0x00000000
Write to selected label:		5 0x00000001 21 0x00000000
rd = 0xc3ff0000	Write	6 0x00000001 22 0x00000000
		7 0x00000004 23 0x00000000
		8 0x00000001 24 0x00000000
		9 0x00000000 25 0x00000000
		10 0x00000000 26 0x00000000
		11 0x00000000 27 0x00000000
		12 0x00000000 28 0x00000000
		13 0x00000000 29 0x00000000
		14 0x00000000 30 0x00000000
		15 0x00000000 31 0x00000000



the instruction is Bnez R8 0x000D which is a branch instruction which conduct a jump if the register is not equal to zero we can see that R8 doesn't have the value zero in it so the dlx PC jumped to 0x000D and skipped 0x000C

Jalr R3

Slave Labels

```
laram = 0x0000003f
STATUS = 0xd5000008
mdio = 0x00000000
PC = 0x00000012
```

Memory Labels

```
rd = 0xc3ff0000
```

Step
Reset-1
Refresh Data
Continuous Mode
Signal Waveforms

Write to selected label:

Write

Commands

0x0000000e	0x2c030012	addi R3 R0 0x0012
0x0000000f	0x5c7f0000	Jalr R3
0x00000010	0x00a11822	sub R3 R5 R1
0x00000011	0x107f0001	beqz R3 0x0013
0x00000012	0x5bff0000	jr R31
0x00000013	0x6803ffff	seqz R3 R0 0xFFFF
0x00000014	0x8c02001b	lw R2 R0 0x001B
0x00000015	0x74450000	snei R5 R2 0x0000
0x00000016	0x78860006	slei R6 R4 0x0006

Registers

0	0x00000000	16	0x00000000
1	0x00000001	17	0x00000000
2	0x00000002	18	0x00000000
3	0x00000012	19	0x00000000
4	0x00000004	20	0x00000000
5	0x00000001	21	0x00000000
6	0x00000001	22	0x00000000
7	0x00000004	23	0x00000000
8	0x00000001	24	0x00000000
9	0x00000000	25	0x00000000
10	0x00000000	26	0x00000000
11	0x00000000	27	0x00000000
12	0x00000000	28	0x00000000
13	0x00000000	29	0x00000000
14	0x00000000	30	0x00000000
15	0x00000000	31	0x00000010

Memory Dump

ADDRESS	VALUES
0x00000000	0xac01001a 0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001
0x00000008	0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x00442025 0xc3ff0000 0x2c030012 0x5c7f0000
0x00000010	0x00a11822 0x107f0001 0x5bff0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024
0x00000018	0x00c41025 0xfffff0000 0x00000001 0xffffffff 0x00000000 0x00000004 0x02000000 0x04000000

Mem Dump From:

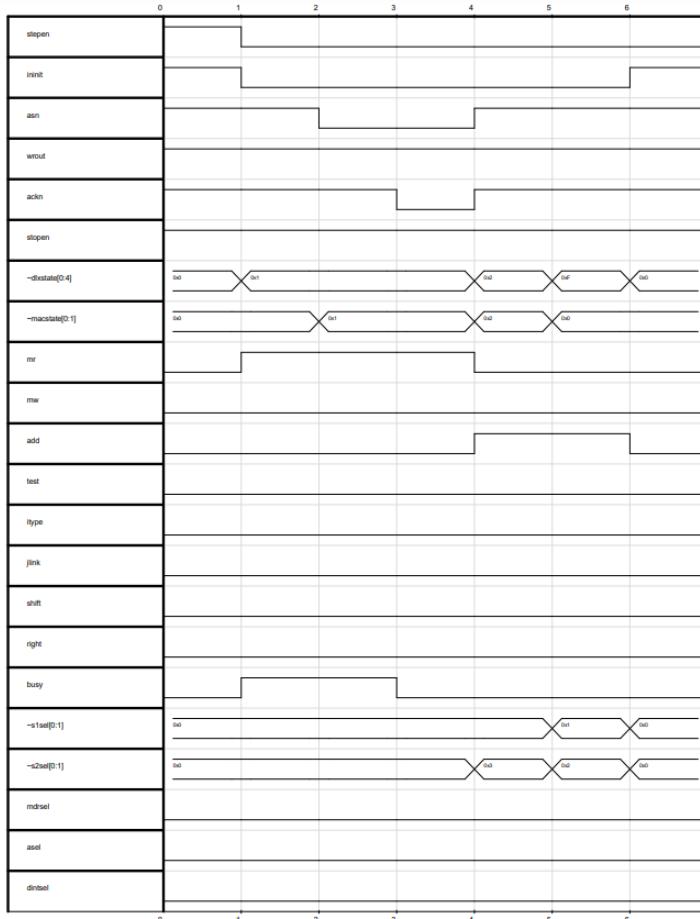
Update Dump

the instruction is **Jalr R3** which is a control operation that save the (PC +1) in R31 and jump to address in R3 we can see in R3 we have 0x12 so we were in address 0x0f and now jumped to 0x12 and in R31 we saved the PC +1 which is 0x10

46

Jr R31

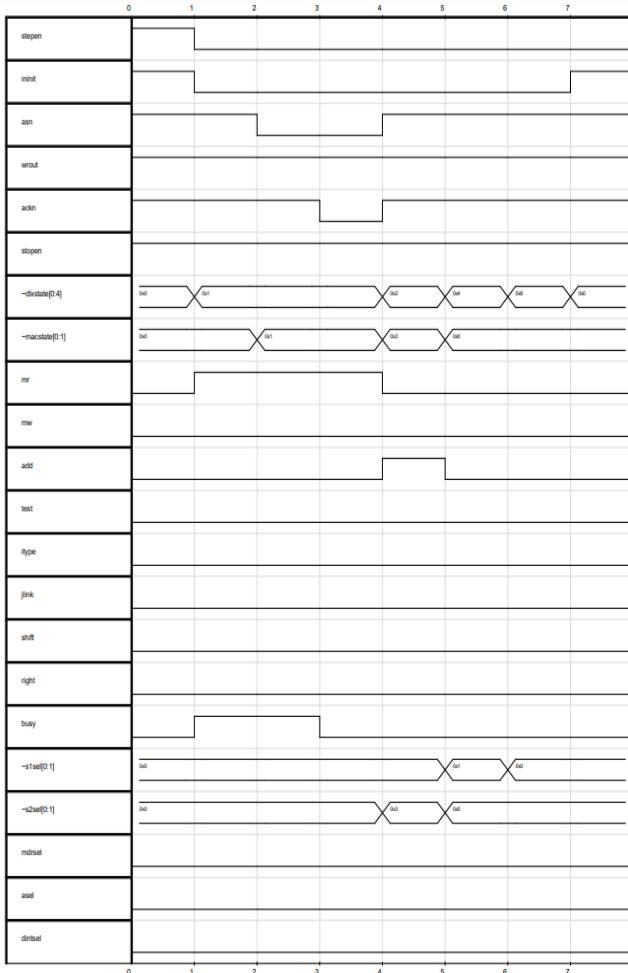
Slave Labels	Commands	Registers
laram = 0x0000003f STATUS = 0xd5000007 mido = 0x00000000 PC = 0x00000010		
<input type="button" value="Step"/>	0x0000000c 0x00442025 or R4 R2 R4	0 0x00000000 16 0x00000000
<input type="button" value="Reset-1"/>	0x0000000d 0xc3ff0000 special-nop	1 0x00000001 17 0x00000000
<input type="button" value="Refresh Data"/>	0x0000000e 0x2c030012 addi R3 R0 0x0012	2 0x00000002 18 0x00000000
<input type="button" value="Continuous Mode"/>	0x0000000f 0x5c7f0000 jalr R3	3 0x00000012 19 0x00000000
<input type="button" value="Signal Waveforms"/>	0x00000010 0x00a11822 sub R3 R5 R1	4 0x00000004 20 0x00000000
Write to selected label:	0x00000011 0x107f0001 beqz R3 0x0013	5 0x00000001 21 0x00000000
<input type="button" value="Write"/>	0x00000012 0x5bff0000 Jr R31	6 0x00000001 22 0x00000000
rd = 0xc3ff0000	0x00000013 0x6803ffff seqi R3 R0 0xFFFF	7 0x00000004 23 0x00000000
	0x00000014 0x8c02001b lw R2 R0 0x001B	8 0x00000001 24 0x00000000
		9 0x00000000 25 0x00000000
		10 0x00000000 26 0x00000000
		11 0x00000000 27 0x00000000
		12 0x00000000 28 0x00000000
		13 0x00000000 29 0x00000000
		14 0x00000000 30 0x00000000
		15 0x00000000 31 0x00000010
Memory Labels	Memory Dump	
	ADDRESS	VALUES
	0x00000000	0x8c01001a 0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001
	0x00000008	0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x00442025 0xc3ff0000 0x2c030012 0x5c7f0000
	0x00000010	0x00a11822 0x107f0001 0x5bff0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024
	0x00000018	0x00c41025 0xfffff0000 0x00000001 0xffffffff 0x00000000 0x00000004 0x02000000 0x04000000
	Mem Dump From:	<input type="text" value="0x00000000"/> <input type="button" value="Update Dump"/>



the instruction is **Jr R31**
which is a control operation that
jumps to the address in RS1
we can see that that RS1 = R31 and
the value in that register is 0x10 from
the previous instruction we can see
that jr R31 is in address 0x12 and
after the jump the PC address is
0x10 which is the value in R31

Sub R3 R5 R1

Slave Labels	Commands	Registers
laram = 0x0000003f STATUS = 0xd5000008 mdo = 0x00000000 PC = 0x00000011	Step	0 0x00000000 16 0x00000000
	Reset=1	1 0x00000001 17 0x00000000
	Refresh Data	2 0x00000002 18 0x00000000
	Continuous Mode	3 0x00000000 19 0x00000000
	Signal Waveforms	4 0x00000004 20 0x00000000
	Write to selected label:	5 0x00000001 21 0x00000000
rd = 0xc3ff0000		6 0x00000000 22 0x00000000
	Write	7 0x00000004 23 0x00000000
		8 0x00000001 24 0x00000000
		9 0x00000000 25 0x00000000
		10 0x00000000 26 0x00000000
		11 0x00000000 27 0x00000000
		12 0x00000000 28 0x00000000
		13 0x00000000 29 0x00000000
		14 0x00000000 30 0x00000000
		15 0x00000000 31 0x00000010
Memory Labels	Memory Dump	
	ADDRESS	VALUES
	0x00000000	0xac01001a 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001
	0x00000008	0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x00442025 0xc3ff0000 0x2c030012 0x5c7f0000
	0x00000010	0x00a1822 0x107f0001 0x5bff0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024
	0x00000018	0x00c41025 0xfffff0000 0x00000001 0xfefffff 0x00000000 0x00000004 0x02000000 0x04000000
	Mem Dump From:	0x00000000 Update Dump



the instruction is **Sub R3 R5 R1**

which translates to:

$$R3 = R5 - R1$$

We can see that the value in both R5 and R1 is 0x01 now if subtract we can see that the value in R3 now is 0

Seqi R3 R0 0xFFFF

Slave Labels:

```
lram = 0x0000003f
STATUS = 0xd5000008
mido = 0x00000000
PC = 0x00000014
```

Commands

Step	0x00000010	0x00e11822	sub R3 R5 R1
Reset=1	0x00000011	0x107f0001	beqz R3 0x0013
Refresh Data	0x00000012	0x5bff0000	jr R31
Continuous Mode	0x00000013	0x6803ffff	seqi R3 R0 0xFFFF
Signal Waveforms	0x00000014	0x8c02001b	lw R2 R0 0x001B
Write to selected label:	0x00000015	0x74450000	slei R5 R2 0x0000
	0x00000016	0x78860006	slei R6 R4 0x0006
	0x00000017	0x00803024	xor R6 R4 R0
	0x00000018	0x00c41025	or R2 R6 R4

Registers

0	0x00000000	16	0x00000000
1	0x00000001	17	0x00000000
2	0x00000002	18	0x00000000
3	0x00000000	19	0x00000000
4	0x00000004	20	0x00000000
5	0x00000001	21	0x00000000
6	0x00000001	22	0x00000000
7	0x00000004	23	0x00000000
8	0x00000001	24	0x00000000
9	0x00000000	25	0x00000000
10	0x00000000	26	0x00000000
11	0x00000000	27	0x00000000
12	0x00000000	28	0x00000000
13	0x00000000	29	0x00000000
14	0x00000000	30	0x00000000
15	0x00000000	31	0x00000000

Memory Labels:

```
rd = 0xc3ff0000
```

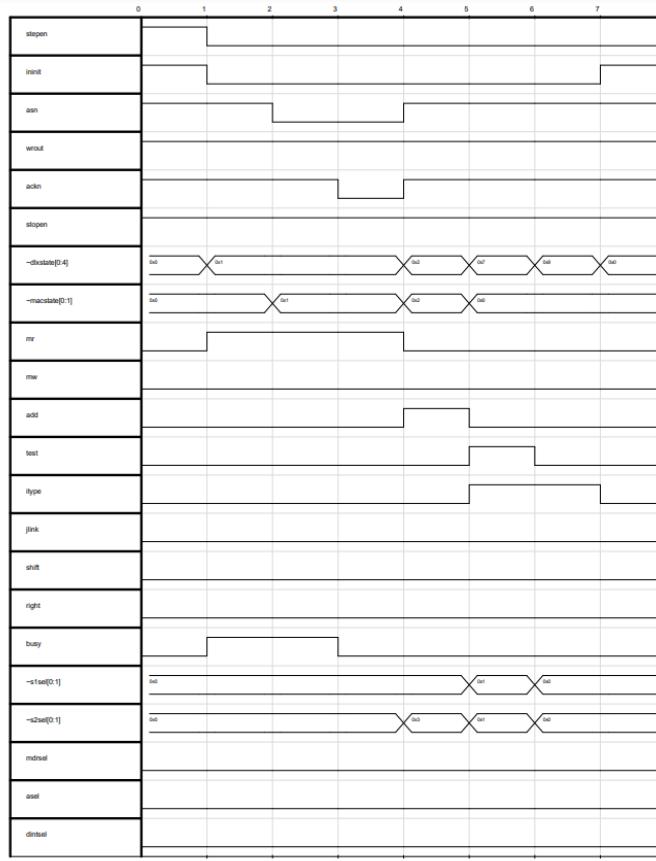
Write

Memory Dump

ADDRESS	VALUES
0x00000000	0xb0c01001a 0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001
0x00000008	0xac24001c 0x8c27001c 0x6d880000 0x151f0001 0x00442025 0xc3ff0000 0x2c030012 0x5c7f0000
0x00000010	0x00e11822 0x107f0001 0x5bff0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024
0x00000018	0x00c41025 0xfffff0000 0x00000001 0xffffffff 0x00000000 0x00000004 0x02000000 0x40000000

Mem Dump From:

Update Dump



the instruction is Seqi R3 R0 0xFFFF
which translates to:

$$R3 = (R0 == 0xFFFF)$$

We can see that the condition is not met so the value of R3 is now zero

Snei R6 R4 0x0006

Slave Labels

```

laram = 0x0000003f
STATUS = 0x50000008
mdio = 0x00000000
PC = 0x00000016

```

Commands

Step	0x00000012	0xbffff0000	jr R31
Reset=1	0x00000013	0x6803ffff	seqj R3 R0 0xFFFF
Refresh Data	0x00000014	0x8c02001b	lw R2 R0 0x0018
Continuous Mode	0x00000015	0x74450000	snei R5 R2 0x0000
Signal Waveforms	0x00000016	0x78860006	slei R6 R4 0x0006
Write to selected label:	0x00000017	0x00803024	xor R6 R4 R0
	0x00000018	0x00c41025	or R2 R6 R4
	0x00000019	0xfffff000	halt
	0x0000001a	0x00000001	no disassembly

Registers

0	0x00000000	16	0x00000000
1	0x00000001	17	0x00000000
2	0xffffffff	18	0x00000000
3	0x00000000	19	0x00000000
4	0x00000004	20	0x00000000
5	0x00000001	21	0x00000000
6	0x00000001	22	0x00000000
7	0x00000004	23	0x00000000
8	0x00000001	24	0x00000000
9	0x00000000	25	0x00000000
10	0x00000000	26	0x00000000
11	0x00000000	27	0x00000000
12	0x00000000	28	0x00000000
13	0x00000000	29	0x00000000
14	0x00000000	30	0x00000000
15	0x00000000	31	0x00000010

Memory Labels

```

rd = 0xc3ff0000

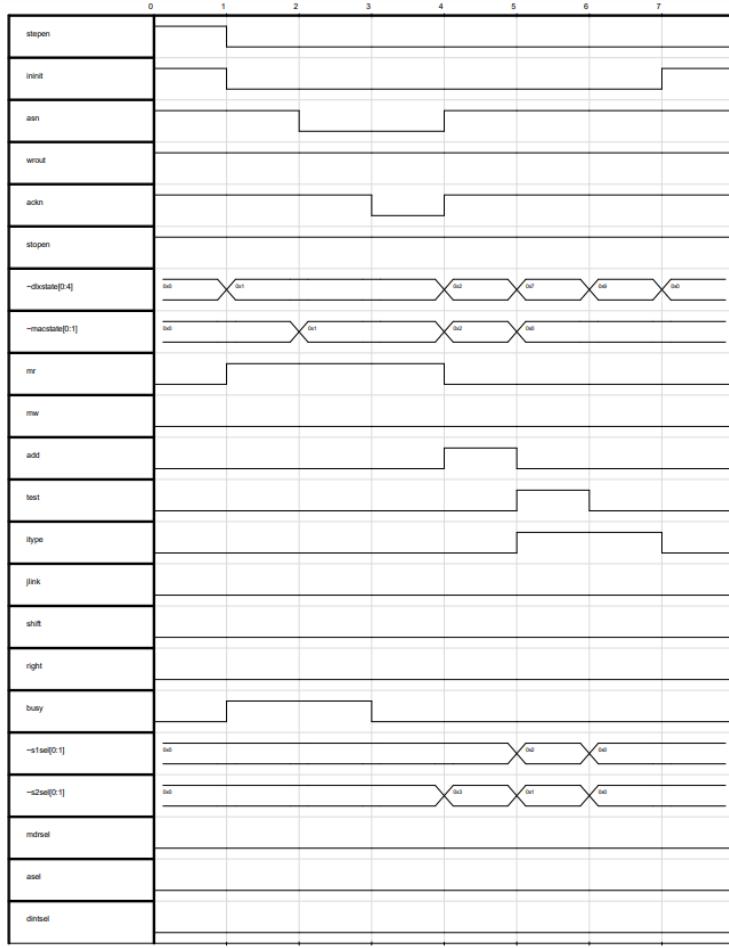
```

Write

Memory Dump

ADDRESS	VALUES
0x00000000	0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001
0x00000008	0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x00442025 0xc3ff0000 0x2c030012 0x5c7f0000
0x00000010	0x00e11822 0x1070001 0x5ff0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024
0x00000018	0x00c41025 0xfffff000 0x00000001 0xefffffff 0x00000000 0x00000004 0x02000000 0x40000000

Mem Dump From: 0x00000000 Update Dump



the instruction is Snei R6 R4
0x0006

which translates to:
 $R6 = (R4 \neq 0x0006)$

We can see that the value of R4 is 0x04 and is not equal to the imm = 0x0006 so the value of R6 is now 1

Slei R6 R4 0x0006

Slave Labels

laram = 0x0000003F
STATUS = 0xd5000008
mdo = 0x00000000
PC = 0x00000017

Memory Labels

rd = 0xc3ff0000

Commands

Step
Reset=1
Refresh Data
Continuous Mode
Signal Waveforms
Write to selected label:
Write

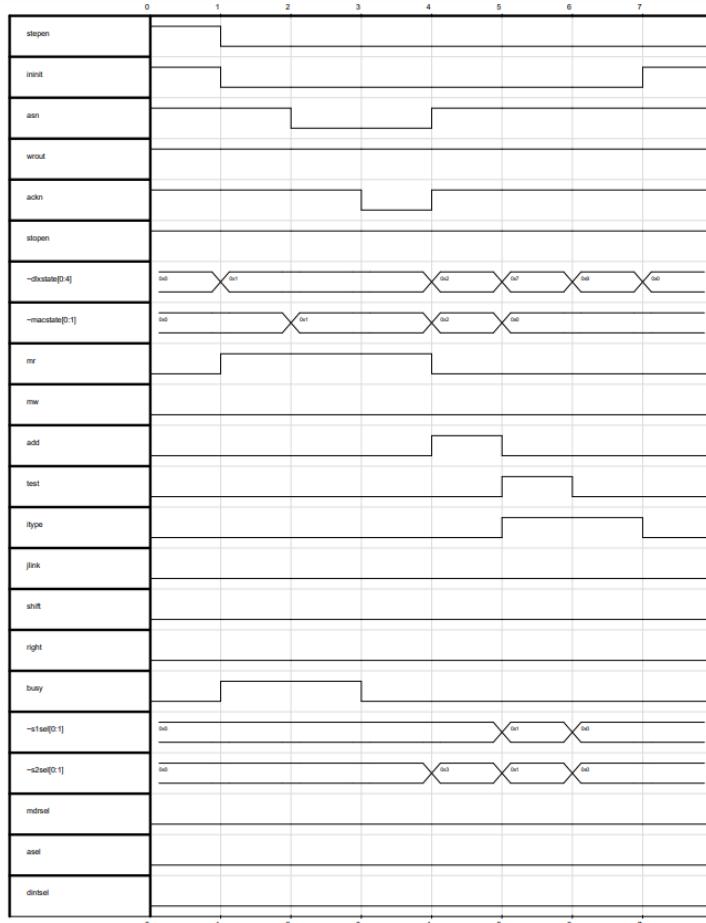
Registers

0	0x00000000	16	0x00000000
1	0x00000001	17	0x00000000
2	0xffffffff	18	0x00000000
3	0x00000000	19	0x00000000
4	0x00000004	20	0x00000000
5	0x00000001	21	0x00000000
6	0x00000001	22	0x00000000
7	0x00000004	23	0x00000000
8	0x00000001	24	0x00000000
9	0x00000000	25	0x00000000
10	0x00000000	26	0x00000000
11	0x00000000	27	0x00000000
12	0x00000000	28	0x00000000
13	0x00000000	29	0x00000000
14	0x00000000	30	0x00000000
15	0x00000000	31	0x00000010

Memory Dump

ADDRESS	VALUES
0x00000000	0x8c01001a 0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001
0x00000008	0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x00442025 0xc3ff0000 0x2c030012 0x5c7f0000
0x00000010	0x00e11822 0x107f0001 0x5bff0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024
0x00000018	0x00c41025 0xfffff0000 0x00000001 0xffffffff 0x00000000 0x00000004 0x02000000 0x04000000

Mem Dump From:



the instruction is Slei R6 R4 0x0006
which translates to:

RD = R4 <= 0x0006

We can see that the value of R4 is 0x04 which is smaller than the imm = 0x0006 so the condition is met and the value of R6 is now 1

Xor R6 R4 R0

Slave Labels

```

laram = 0x0000003f
STATUS = 0xd5000008
mdo = 0x00000000
PC = 0x00000018

```

Memory Labels

```

rd = 0xc3ff0000

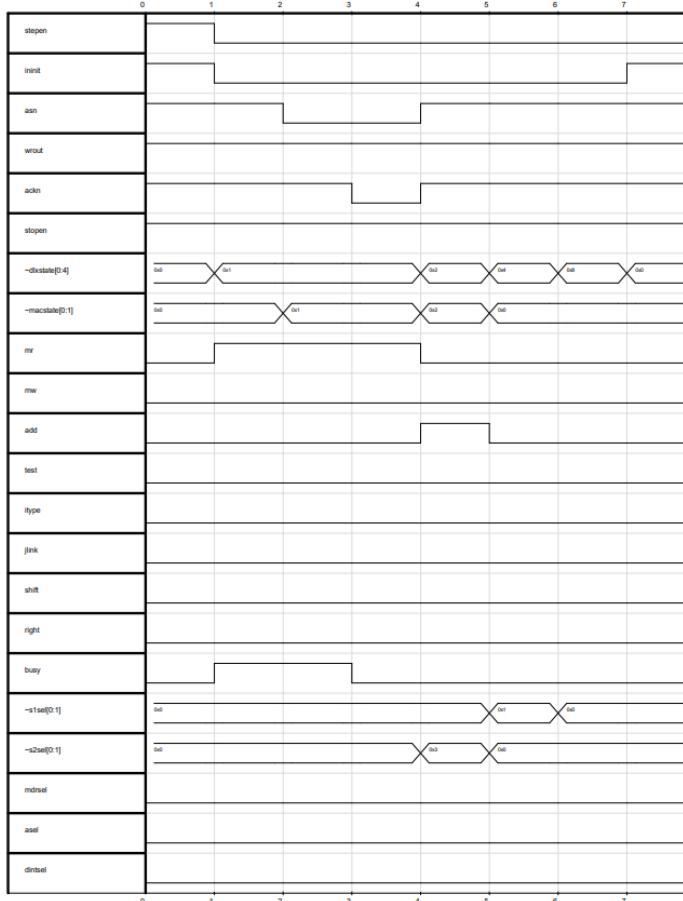
```

Commands		Registers	
Step	0x00000014 0x8c02001b lw R2 R0 0x001B	0 0x00000000 16 0x00000000	
Reset=1	0x00000015 0x74450000 slei R5 R2 0x0000	1 0x00000001 17 0x00000000	
Refresh Data	0x00000016 0x78860006 slsl R6 R4 0x0006	2 0xfffffff 18 0x00000000	
Continuous Mode	0x00000017 0x00803024 xor R6 R4 R0	3 0x00000000 19 0x00000000	
Signal Waveforms	0x00000018 0x0c41025 or R2 R6 R4	4 0x00000004 20 0x00000000	
Write to selected label:	0x00000019 0xffff0000 halt	5 0x00000001 21 0x00000000	
	0x0000001a 0x00000001 no disassembly	6 0x00000004 22 0x00000000	
	0x0000001b 0xffffffff no disassembly	7 0x00000004 23 0x00000000	
Write	0x0000001c 0x00000000 sli R0 R0	8 0x00000001 24 0x00000000	
		9 0x00000000 25 0x00000000	
		10 0x00000000 26 0x00000000	
		11 0x00000000 27 0x00000000	
		12 0x00000000 28 0x00000000	
		13 0x00000000 29 0x00000000	
		14 0x00000000 30 0x00000000	
		15 0x00000000 31 0x00000000	

Memory Dump

ADDRESS	VALUES
0x00000000	0xac01001a 0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001
0x00000008	0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x00442025 0xc3f0000 0x2c030012 0x5c7f0000
0x00000010	0x00e11822 0x107f0001 0x5bf0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024
0x00000018	0x00c41025 0xffff0000 0x00000001 0xffffffff 0x00000000 0x00000004 0x02000000 0x04000000

Mem Dump From:



the instruction is Xor R6 R4 R0
which translates to:
R6 = R4 Xor R0
Which means R6 = 0x04 Xor 0x00
Which we can see now R2 have the result of the equation which is 0x04

Or R2 R6 R4

Slave Labels

```

laram = 0x0000003f
STATUS = 0xc5000008
mdo = 0x00000000
PC = 0x00000019

```

Commands

Step	0x00000015	0x74450000	slei R5 R2 0x0000
Reset-1	0x00000016	0x78860006	slei R6 R4 0x0006
Refresh Data	0x00000017	0x00803024	xor R6 R4 R0
Continuous Mode	0x00000018	0x00c41025	or R2 R6 R4
Signal Waveforms	0x00000019	0xffff0000	halt
Write to selected label:	0x0000001a	0x00000001	no disassembly
	0x0000001b	0xffffffff	no disassembly
	0x0000001c	0x00000000	sli R0 R0
	0x0000001d	0x00000004	no disassembly

Registers

0	0x00000000	16	0x00000000
1	0x00000001	17	0x00000000
2	0x00000004	18	0x00000000
3	0x00000000	19	0x00000000
4	0x00000004	20	0x00000000
5	0x00000001	21	0x00000000
6	0x00000004	22	0x00000000
7	0x00000004	23	0x00000000
8	0x00000001	24	0x00000000
9	0x00000000	25	0x00000000
10	0x00000000	26	0x00000000
11	0x00000000	27	0x00000000
12	0x00000000	28	0x00000000
13	0x00000000	29	0x00000000
14	0x00000000	30	0x00000000
15	0x00000000	31	0x00000010

Memory Labels

```

rd = 0xc3ff0000

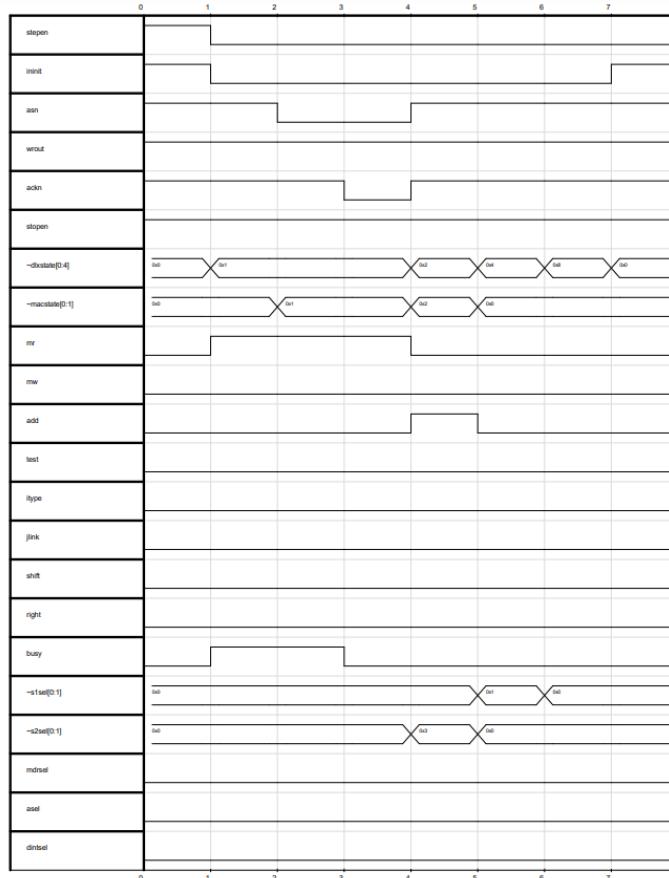
```

Write

Memory Dump

ADDRESS	VALUES
0x00000000	0xac01001a 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001
0x00000008	0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x00442025 0xc3ff0000 0x2c030012 0x5c7f0000
0x00000010	0x00a11822 0x107f0001 0x5bff0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006 0x00803024
0x00000018	0x00c41025 0xffff0000 0x00000001 0xffffffff 0x00000000 0x00000004 0x02000000 0x40000000

Mem Dump From: 0x00000000 Update Dump



the instruction is **Or R2 R6 R4**

which translates to:

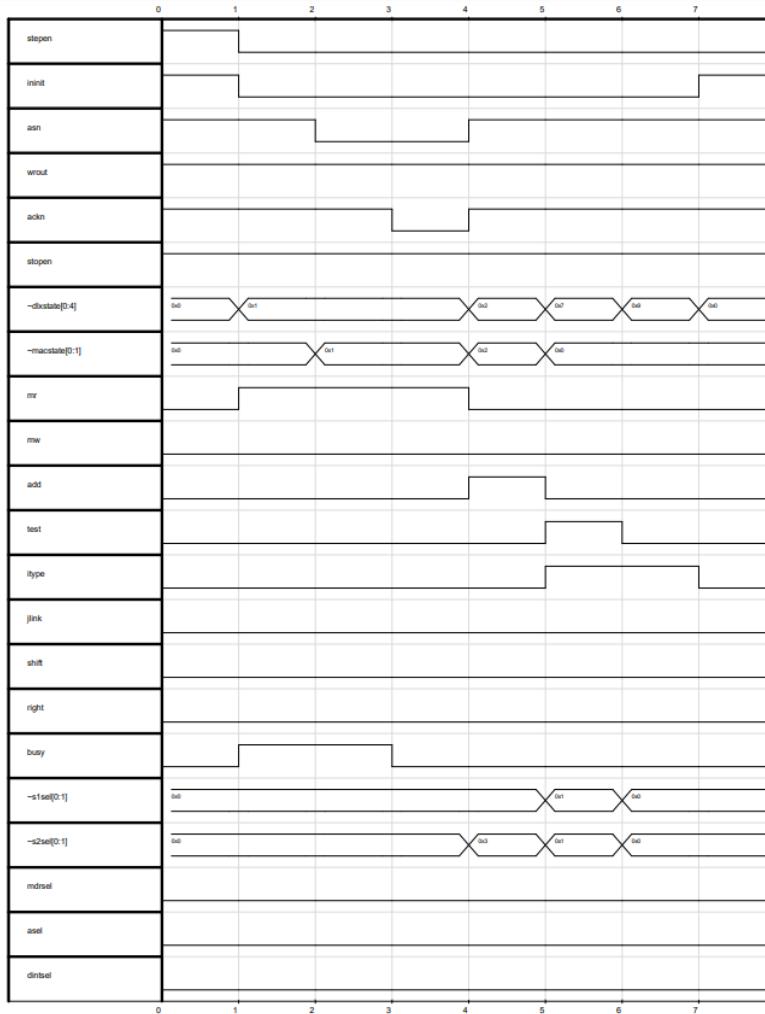
R2 = R6 or R4 we can see that R6 and R4 have the same value in it which should result in the same value of the register after the OR operation which we can see in R2

Sgti R4 R1 0x0000

Slave Labels	Commands	Registers
laram = 0x0000003f STATUS = 0xd5000008 mdo = 0x00000000 PC = 0x00000229		
<input type="button" value="Step"/>	0x00000225 0x10800607 beqz R4 0x082D	0 0x00000000 16 0x00008000
<input type="button" value="Reset=1"/>	0x00000226 0x6844ffff seqi R4 R2 0xFFFF	1 0x00000001 17 0x00010000
<input type="button" value="Refresh Data"/>	0x00000227 0x10800605 beqz R4 0x082D	2 0xffffffff 18 0x00200000
<input type="button" value="Continuous Mode"/>	0x00000228 0x64240000 sgti R4 R1 0x0000	3 0xffffffff 19 0x00400000
<input checked="" type="button" value="Signal Waveforms"/>	0x00000229 0x10800604 beqz R4 0x082E	4 0x00000001 20 0x00800000
<input type="button" value="Write to selected label:"/>	0x0000022a 0x64240001 sgti R4 R1 0x0001	5 0x00000001 21 0x00100000
<input type="button" value="Write"/>	0x0000022b 0x14800602 bnez R4 0x082E	6 0x00000020 22 0x00200000
	0x0000022c 0x6444ffff sgti R4 R2 0xFFFF	7 0x00000040 23 0x00400000
	0x0000022d 0x14800600 bnez R4 0x082E	8 0x00000080 24 0x00800000
		9 0x00000100 25 0x01000000
		10 0x00000200 26 0x02000000
		11 0x00000400 27 0x04000000
		12 0x00000800 28 0x08000000
		13 0x00001000 29 0x10000000
		14 0x00002000 30 0x20000000
		15 0x00004000 31 0x80000000

Memory Dump	
ADDRESS	VALUES
0x00000000	0xc0000000 0x2c000000 0x8c10026 0x58200000 0xfc000000 0x00000001 0x00000002 0x00000004
0x00000008	0x00000008 0x00000010 0x00000020 0x00000040 0x00000080 0x00000100 0x00000200 0x00000400
0x00000010	0x00000080 0x00001000 0x00002000 0x00004000 0x00008000 0x00100000 0x00200000 0x00400000
0x00000018	0x00080000 0x00100000 0x00200000 0x00400000 0x00800000 0x01000000 0x02000000 0x04000000

Mem Dump From:



the instruction is **Sgti R4 R1 0x0000**

which translates to:

R4 = (R1 > 0x00)

We can see that R1 have the value 0x01 in it which is greater than the imm = 0x00

So now the value of R4 is 1

Srli R7 R6 0x0001

Slave Labels

```
laram = 0x0000003f
STATUS = 0xd5000008
mdo = 0x00000000
PC = 0x0000051f
```

Commands

Step	0x0000051b	0x00203000	slli R6 R1
Reset=1	0x0000051c	0x68c70002	seqj R7 R6 0x0002
Refresh Data	0x0000051d	0x10e0031c	beqz R7 0x083A
Continuous Mode	0x0000051e	0x00c03002	srl R6 R6
Signal Waveforms	0x0000051f	0x68c70001	seqj R7 R6 0x0001
Write to selected label:	0x00000520	0x10e00319	beqz R7 0x083A
rd = 0x00000100	0x00000521	0x000003000	slli R6 R0
Write	0x00000522	0x68c70000	seqj R7 R6 0x0000
	0x00000523	0x10e00316	beqz R7 0x083A

Registers

0	0x00000000	16	0x00008000
1	0x00000001	17	0x00010000
2	0xffffffff	18	0x00020000
3	0x7fffffff	19	0x00040000
4	0x80000000	20	0x00080000
5	0x00000001	21	0x00100000
6	0x00000001	22	0x00200000
7	0x00000001	23	0x00400000
8	0x00000001	24	0x00800000
9	0x00000100	25	0x01000000
10	0x00000200	26	0x02000000
11	0x00000400	27	0x04000000
12	0x00000800	28	0x08000000
13	0x00001000	29	0x10000000
14	0x00002000	30	0x20000000
15	0x00004000	31	0x80000000

Memory Labels

```
rd = 0x00000100
```

Memory Dump

ADDRESS	VALUES
0x00000000	0xc0000000 0x2c000000 0x8c010026 0x58200000 0xfc000000 0x00000001 0x00000002 0x00000004
0x00000008	0x00000008 0x00000010 0x00000020 0x00000040 0x00000080 0x00000100 0x00000200 0x00000400
0x00000010	0x00000800 0x00001000 0x00002000 0x00004000 0x00008000 0x00010000 0x00020000 0x00040000
0x00000018	0x00080000 0x00100000 0x00200000 0x00400000 0x00800000 0x01000000 0x02000000 0x04000000

Mem Dump From:

the instruction is **Srli R7 R6 0x0001**

which translates to:

R6 = R6 >> 1

Which is a shift right operation the value in R6 was previously 0x02 now we can see that the value of R6 is 0x01 which is 0x02 shifted to the right by 1

55

And R5 R4 R3

Slave Labels

```

lram = 0x0000003f
STATUS = 0xd5000008
mdo = 0x00000000
PC = 0x0000000e

```

Commands

Step	0x0000000a	0x6c880000	sgei R8 R4 0x0000
Reset=1	0x0000000b	0x151f0001	bnez R8 0x000D
Refresh Data	0x0000000c	0x00442025	or R4 R2 R4
Continuous Mode	0x0000000d	0x00832826	and R5 R4 R3
Signal Waveforms	0x0000000e	0xc3ff0000	special-mop
Write to selected label:	rd = 0x00832825	0x0000000f	addi R3 R0 0x0012
		0x00000010	jalr R3
		0x00000011	sub R3 R5 R1
		0x00000012	beqz R3 0x0014

Registers

0	0x00000000	16	0x00000000
1	0x00000001	17	0x00000000
2	0x00000002	18	0x00000000
3	0x00000003	19	0x00000000
4	0x00000004	20	0x00000000
5	0x00000000	21	0x00000000
6	0x00000001	22	0x00000000
7	0x00000004	23	0x00000000
8	0x00000001	24	0x00000000
9	0x00000000	25	0x00000000
10	0x00000000	26	0x00000000
11	0x00000000	27	0x00000000
12	0x00000000	28	0x00000000
13	0x00000000	29	0x00000000
14	0x00000000	30	0x00000000
15	0x00000000	31	0x00000000

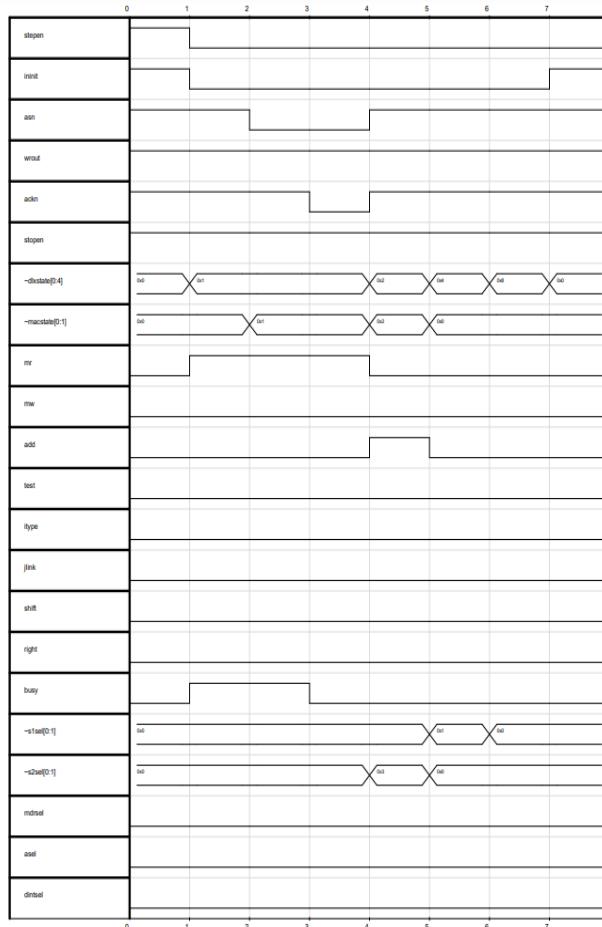
Memory Labels

```
rd = 0x00832825
```

Memory Dump

ADDRESS	VALUES
0x00000000	0x8c01001a 0xac01001d 0x2c220001 0x00221823 0x005f2802 0x005f2000 0x70460003 0x10df0001
0x00000008	0xac24001c 0x8c27001c 0x6c880000 0x151f0001 0x00442025 0x00832826 0xc3ff0000 0x2c030012
0x00000010	0x5c7f0000 0x00a11822 0x107f0001 0x5bfff0000 0x6803ffff 0x8c02001b 0x74450000 0x78860006
0x00000018	0x00803024 0x00c41025 0x00000001 0xffffffff 0x00000000 0x00000004 0x00000000 0x00000000

Mem Dump From: 0x00000000 Update Dump



the instruction is **And R5 R4 R3**

which translates to:

R5 = R4 and R3

R3 is 0x03 and R4 is 0x04 which should result in zero after the and operation which we can see that the value of R5 is now 0 as we expected

The instructor signature :

C 5 : 23.7.24 AF
Half reached.
P A F T or 31.07. GP