EEL 3701C: Digital Logic & Computer Systems
Revision **0**
Lab **6** Report: **Elementary CPU Design**

Stern, Arion
Class #: 10844
Erick Zayas Ramos
July 24, 2025

REQUIREMENTS NOT MET

N/A

VIDEO FILE LINK

https://youtube.com/shorts/TeFolGsbdYU?feature=share

PROBLEMS ENCOUNTERED

I had very minor issues with my first attempts at my ASMs, but my PI helped me fix them. A major problem I had was creating the ROM and MIF files; I found the tutorials confusing, and it took me much longer than it should have. One issue I had was that I flipped the PC's MSB and LSB connections, which caused incorrect ROM access. The final issue I had was with creating the simulations, as it involved a lot of troubleshooting and minor corrections before they were correct.

FUTURE WORK/APPLICATIONS

The CPU designed in Lab 6 shows how a state machine can execute instructions using a custom instruction set. This basic architecture can be extended to support features like branching, conditional logic, and memory addressing. In Lab 7, these concepts are applied to the G-CPU, a more advanced processor capable of running full assembly programs. The skills developed in Lab 6 provide the foundation for writing, assembling, and simulating code in a complete CPU system.

PRE-LAB QUESTIONS OR EXERCISES

PART 1 PRE-LAB QUESTIONS

1. Why did we require the new instruction register in this design?

The instruction register is required to temporarily store the opcode of the current instruction, allowing the controller to decode it and generate the necessary control signals for the data path. This design allows the CPU to execute instructions in a synchronized and efficient manner across clock cycles, ensuring consistent operation.

2. In this section of the lab, you are setting the INPUT bus by hand. If you wanted to read or fetch this value from memory, what could you add to do this automatically for you every CLK cycle?

By adding a program counter and a ROM module, the system could automatically fetch opcodes and data from memory on each clock cycle. The PC would sequentially supply memory addresses, and the ROM would provide the stored instructions, eliminating the need for manual input.

3. How would you add more instructions (i.e., 8 instead of 4) to the controller?

To accommodate eight instructions, the instruction register would need to be expanded from 2 bits to 3 bits, allowing for eight unique opcodes. The controller's state machine logic would also need to be updated to decode the additional opcodes and produce the appropriate control signals for each operation.

PART 2 PRE-LAB QUESTIONS

1. Why do we need the extra states in the LDAA and JMP instruction paths?

The LDAA and JMP instructions require extra states because they each involve two separate memory reads, which must occur over two clock cycles. For LDAA, the first cycle fetches the opcode, and the second retrieves the immediate data to load into REGA. For JMP, the first receives the opcode, and the second cycle retrieves the target address to load into the Program Counter.

2. What do you need to do to the address lines to get your program to start at address $2CD0 (instead of $3A10)?

You would have to tie the upper address lines, from 14 to 4, to $2CD, which is binary 0010 1100 1101. Addresses 4, 6, 7, 10, 11, and 13 would be tied to VCC, and the rest would be tied to ground.

PRE-LAB REQUIREMENTS (Design, Schematic, ASM Chart, VHDL, etc.)

1. FIRST RALU CONTROLLER:

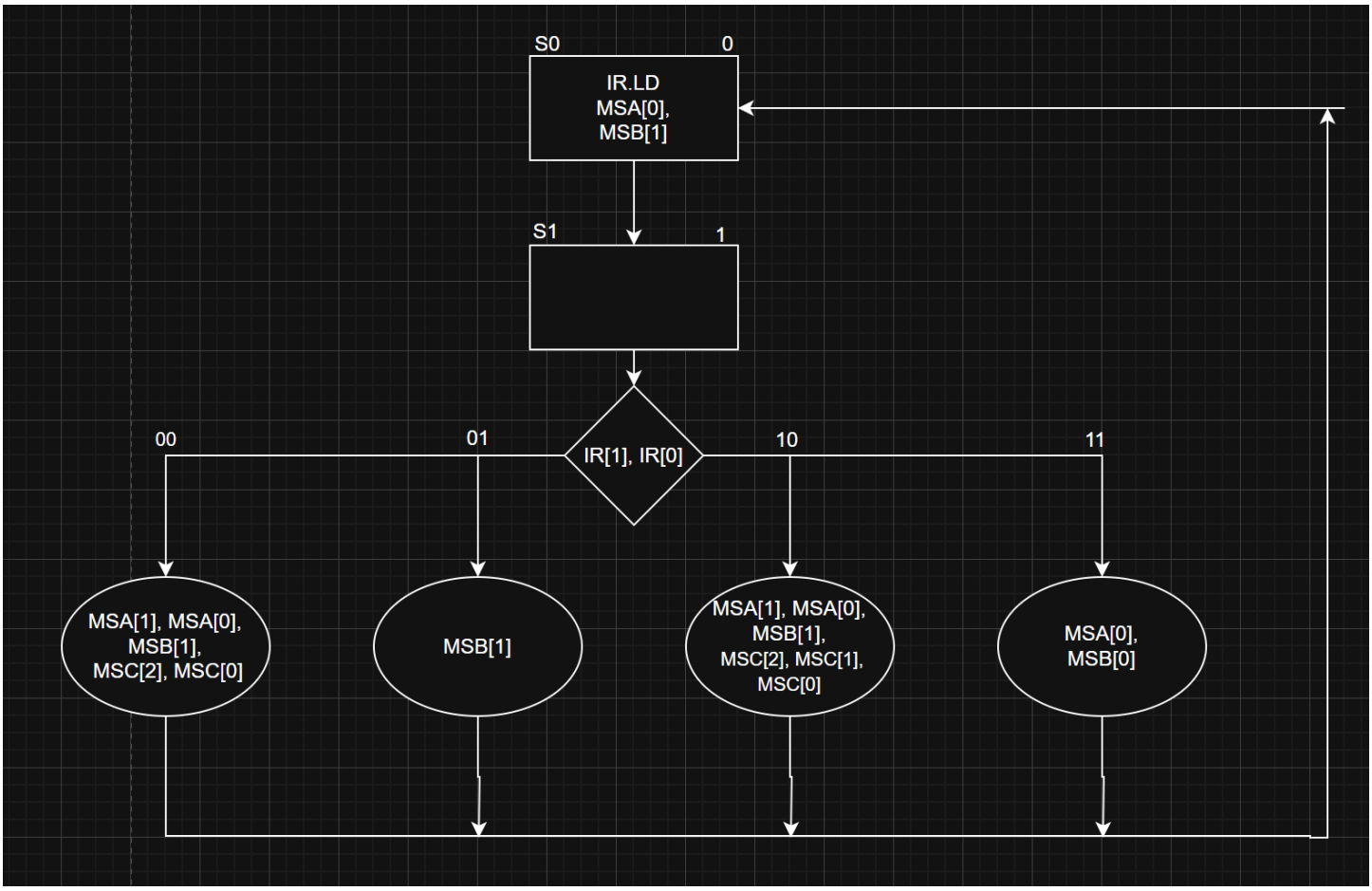


Figure 1: ASM for First RALU Controller (Part 1)

Q0	IR[1]	IR[0]	IR.LD	MSA[1]	MSA[0]	MSB[1]	MSB[0]	MSC[2]	MSC[1]	MSC[0]	Q0+
0	-	-	1	0	1	1	0	0	0	0	1
1	0	0	0	1	1	1	0	1	0	1	0
1	0	1	0	0	0	1	0	0	0	0	0
1	1	0	0	1	1	1	0	1	1	1	0
1	1	1	0	0	1	0	1	0	0	0	0

Table 1: NSTT for First RALU Controller (Part 1)

Text Editor - C:/QuartusProjects/Lab6/LAB6_Part1/LAB6_Part1 - LAB6_Part1 - [Controller.vhd]

File Edit View Project Processing Tools Window Help

```
1   --controller1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5   entity Controller is port(
6
7    IR: in std_logic_vector(1 downto 0);
8    q0: in std_logic;
9
10   MSA, MSB: out std_logic_vector(1 downto 0);
11   MSC:  out  std_logic_vector(2 downto 0);
12   IRLD, D0: out std_logic
13
14
15   );
16   end Controller;
17
18
19   architecture behavior of Controller is begin
20
21   with (q0 & IR) select
22
23     MSA(1) <= '1' when "100" | "110",
24               '0' when others;
25   with (q0 & IR) select
26     MSA(0) <= '0' when "101",
27               '1' when others;
28
29   with (q0 & IR) select
30     MSB(1) <= '0' when "111",
31               '1' when others;
32   with (q0 & IR) select
33     MSB(0) <= '1' when "111",
34               '0' when others;
35
36   with (q0 & IR) select
37     MSC(2) <= '1' when "100" | "110",
38               '0' when others;
39   with (q0 & IR) select
40     MSC(1) <= '1' when "110",
41               '0' when others;
42   with (q0 & IR) select
43     MSC(0) <= '1' when "100" | "110",
44               '0' when others;
45
46     IRLD <= not Q0;
47     D0 <= not Q0;
48  end behavior;
49
50
```

Figure 2: VHDL for First RALU Controller (Part 1)

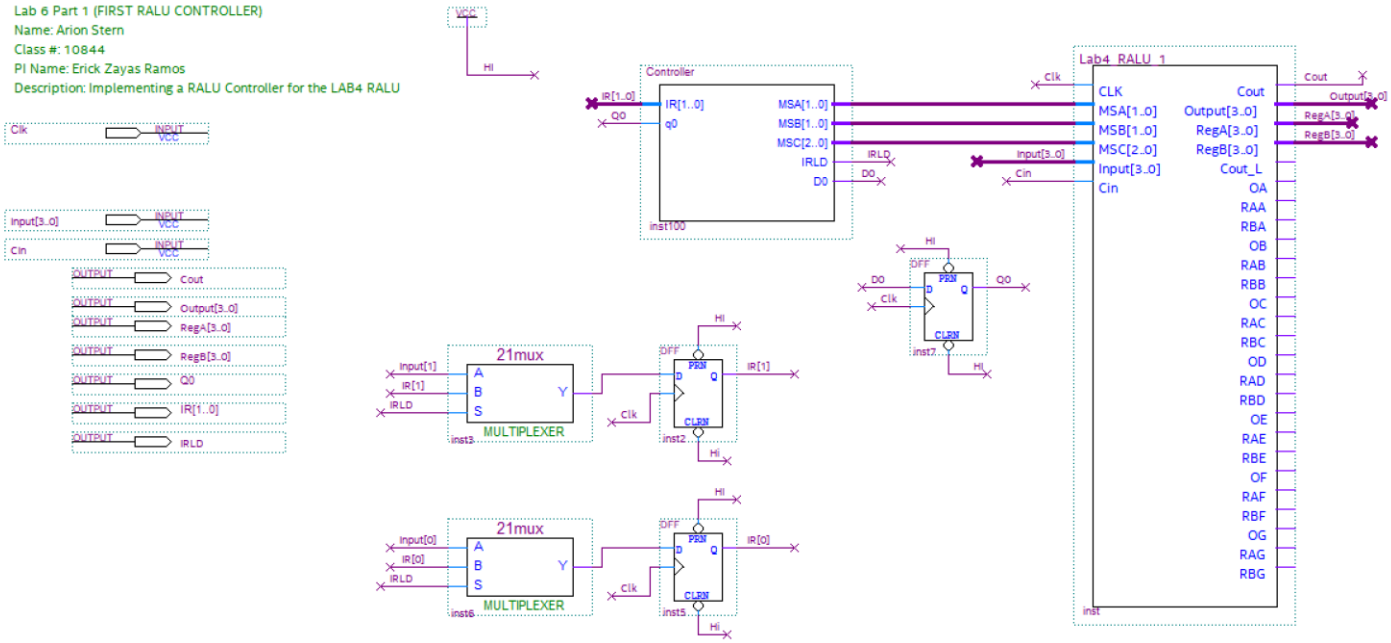


Figure 3: BDF Schematic for LAB6_Part1 (Part 1)

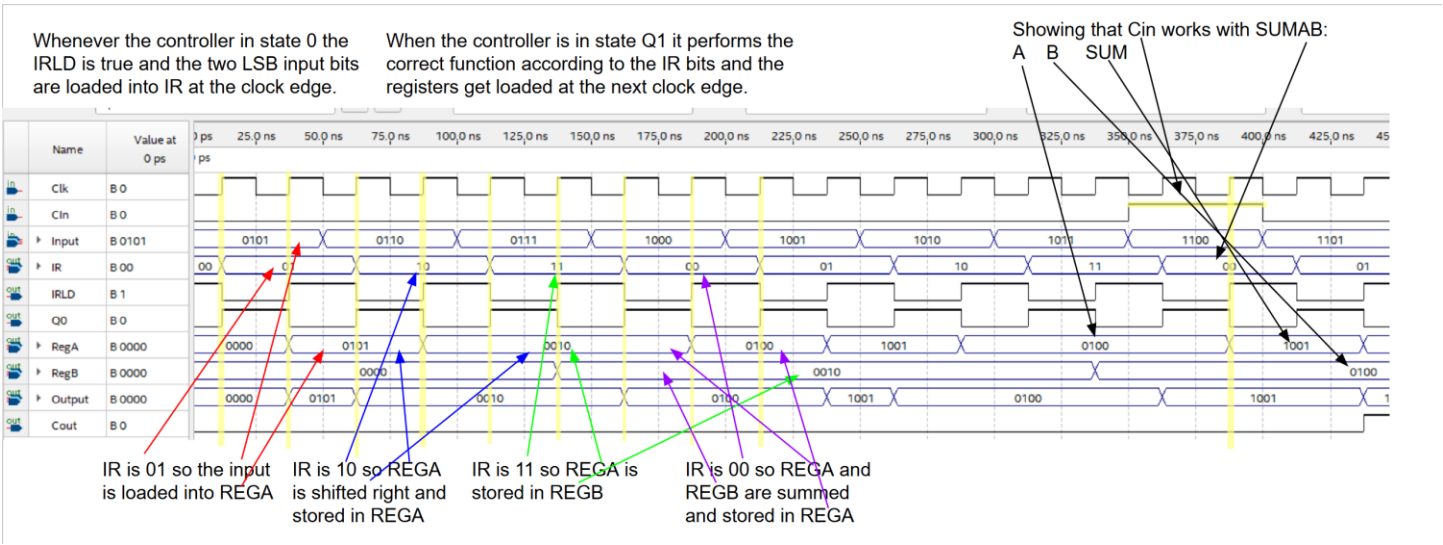


Figure 4: Annotated Simulation for Lab6_Part1 (Part 1)

2. SECOND RALU CONTROLLER WITH ROM :

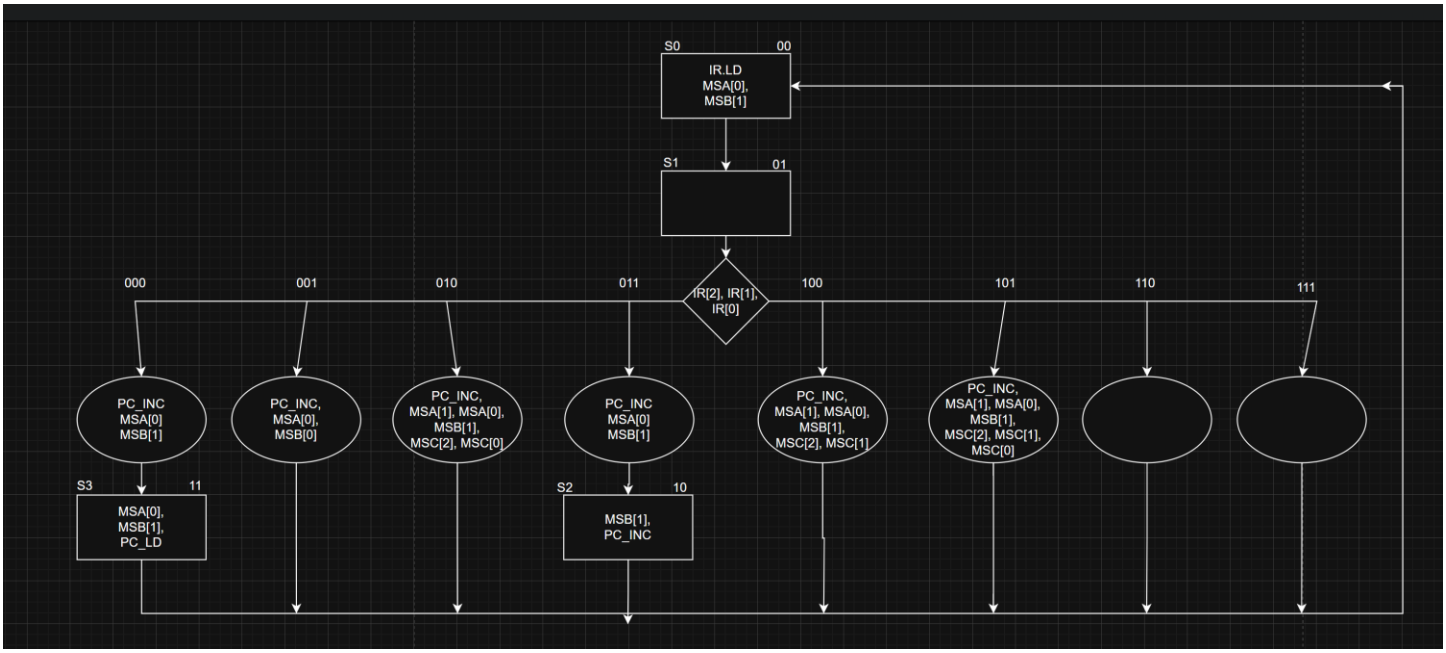


Figure 5: ASM for Second RALU Controller (Part 2)

Q1	Q0	IR[2]	IR[1]	IR[0]	IR.LD	PC.LD	PC.INC	MSA[1]	MSA[0]	MSB[1]	MSB[0]	MSC[2]	MSC[1]	MSC[0]	Q1+	Q0+
0	0	-	-	-	1	0	0	0	1	1	0	0	0	0	0	1
0	1	0	0	0	0	0	1	0	1	1	0	0	0	0	1	1
0	1	0	0	1	0	0	1	0	1	0	1	0	0	0	0	0
0	1	0	1	0	0	0	1	1	1	1	0	1	0	1	0	0
0	1	0	1	1	0	0	1	0	1	1	0	0	0	0	1	0
0	1	1	0	0	0	0	1	1	1	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1	1	1	1	0	1	1	1	0	0
1	0	-	-	-	0	0	1	0	0	1	0	0	0	0	0	0
1	1	-	-	-	0	1	0	0	1	1	0	0	0	0	0	0

Table 2: NSTT for Second RALU Controller (Part 2)

Text Editor - C:/QuartusProjects/Lab6/Lab6_Part2/Lab6_Part2 - Lab6_Part2 - [controller2.vhd]*

File Edit View Project Processing Tools Window Help

```
1   --controller2
2
3   library IEEE;
4   use IEEE.STD_LOGIC_1164.ALL;
5
6   entity controller2 is port (
7
8   Q1, Q0: in std_logic;
9   Ir: in std_logic_vector(2 downto 0);
10
11  IRLD, PCLD, PCINC, D1, D0: out std_logic;
12
13  MSA, MSB: out std_logic_vector (1 downto 0);
14  MSC: out std_logic_vector (2 downto 0)
15
16  ); end controller2;
17
18  architecture behavior of controller2 is
19
20  signal state : std_logic_vector(1 downto 0);
21
22
23  begin
24
25      state <= Q1 & Q0;
26
27     IRLD <= not (Q0 or Q1);
28     PCLD <= q1 and q0;
29
30     --pos 00 and 11
31     PCINC <= (Q1 or Q0) and ((not q1) or (not q0));
32
33     with (q1 & q0 & IR) select
34        MSA(1) <= '1' when "01010" | "01100" | "01101",
35                  '0' when others;
36
37
38     with state select
39     MSA(0) <= '0' when "10",
40               '1' when others;
41
42     --MSA(0) <= '0' when (q1 = '1' and q0 = '0')
43        --  else '1';
44
45
46     with (q1 & q0 & IR) select
47        MSB(1) <= '0' when "01001",
48                  '1' when others;
49
50     MSB(0) <= (not q1) and Q0 and (not IR(2)) and (not IR(1)) and IR(0);
51
52
53     with (q1 & q0 & IR) select
54        MSC(2) <= '1' when "01010" | "01100" | "01101",
55                  '0' when others;
56
57     with (q1 & q0 & IR) select
58        MSC(1) <= '1' when "01100" | "01101",
59                  '0' when others;
```

Figure 6: VHDL for Second RALU Controller 1/2 (Part 2)

Text Editor - C:/QuartusProjects/Lab6/Lab6_Part2/Lab6_Part2 - Lab6_Part2 - [controller2.vhd]*

File Edit View Project Processing Tools Window Help

```
25      state <= Q1 & Q0;
26
27     IRLD <= not (Q0 or Q1);
28     PCLD <= q1 and q0;
29
30     --pos 00 and 11
31     PCINC <= (Q1 or Q0) and ((not q1) or (not q0));
32
33     with (q1 & q0 & IR) select
34        MSA(1) <= '1' when "01010" | "01100" | "01101",
35                  '0' when others;
36
37
38     with state select
39     MSA(0) <= '0' when "10",
40               '1' when others;
41
42     --MSA(0) <= '0' when (q1 = '1' and q0 = '0')
43        --  else '1';0
44
45
46     with (q1 & q0 & IR) select
47        MSB(1) <= '0' when "01001",
48                  '1' when others;
49
50     MSB(0) <= (not q1) and Q0 and (not IR(2)) and (not IR(1)) and IR(0);
51
52
53     with (q1 & q0 & IR) select
54        MSC(2) <= '1' when "01010" | "01100" | "01101",
55                  '0' when others;
56
57     with (q1 & q0 & IR) select
58        MSC(1) <= '1' when "01100" | "01101",
59                  '0' when others;
60
61     with (q1 & q0 & IR) select
62        MSC(0) <= '1' when "01010" | "01101",
63                  '0' when others;
64
65     with(q1 & q0 & IR) select
66        D1 <= '1' when "01000" | "01011",
67              '0' when others;
68
69     D0 <= '1' when (q1 = '0' and q0 = '0') or
70                    (q1 = '0' and q0 = '1' and IR = "000")
71      else '0';
72
73
74  end behavior;
75
```

Figure 7: VHDL for Second RALU Controller 2/2 (Part 2)

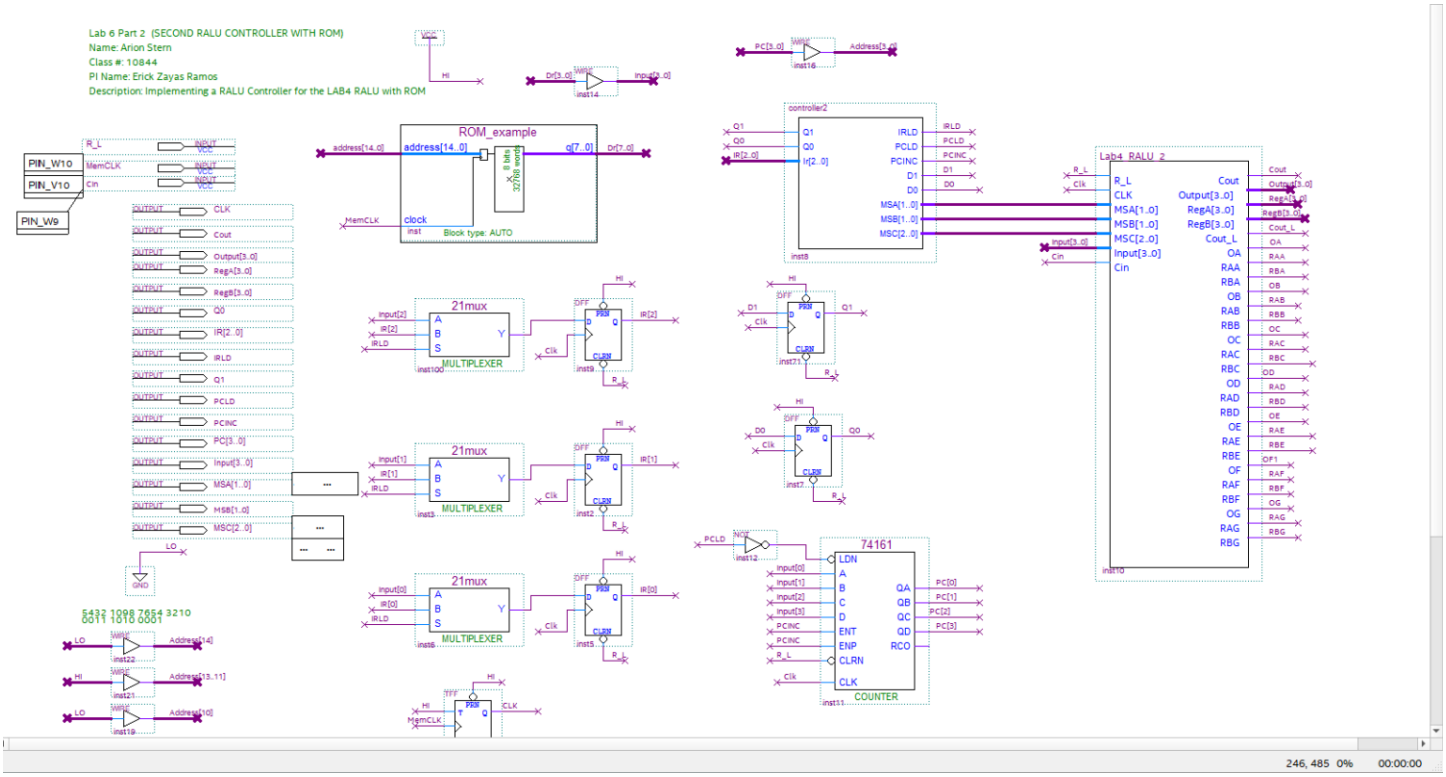


Figure 8: BDF Schematic for LAB6_Part2 1/2 (Part 2)

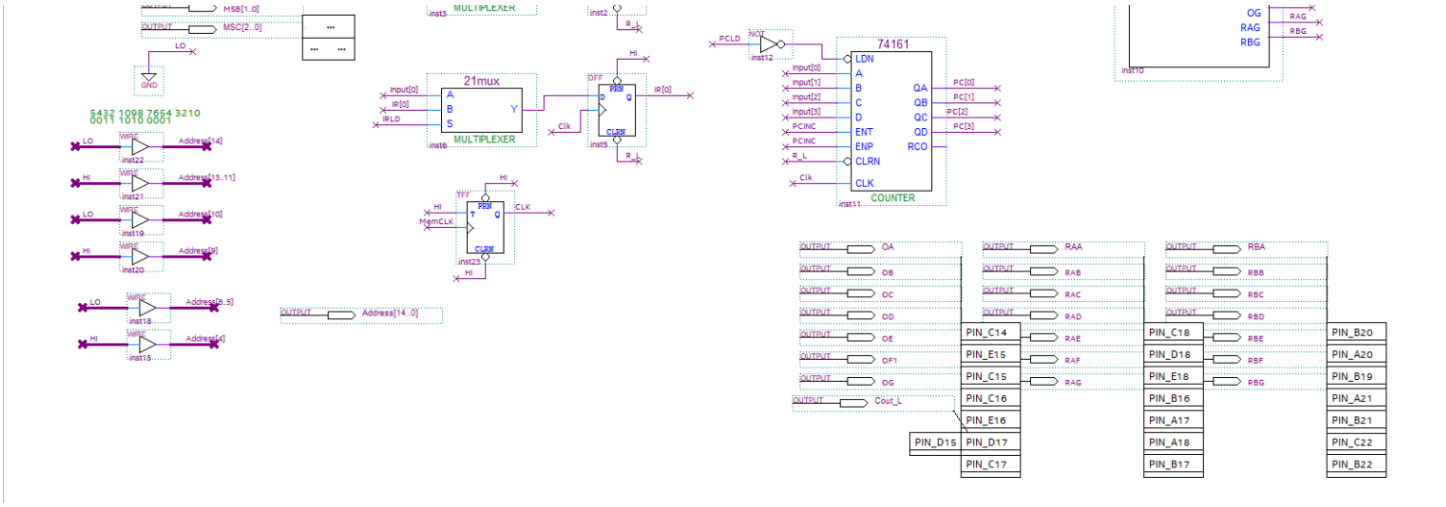


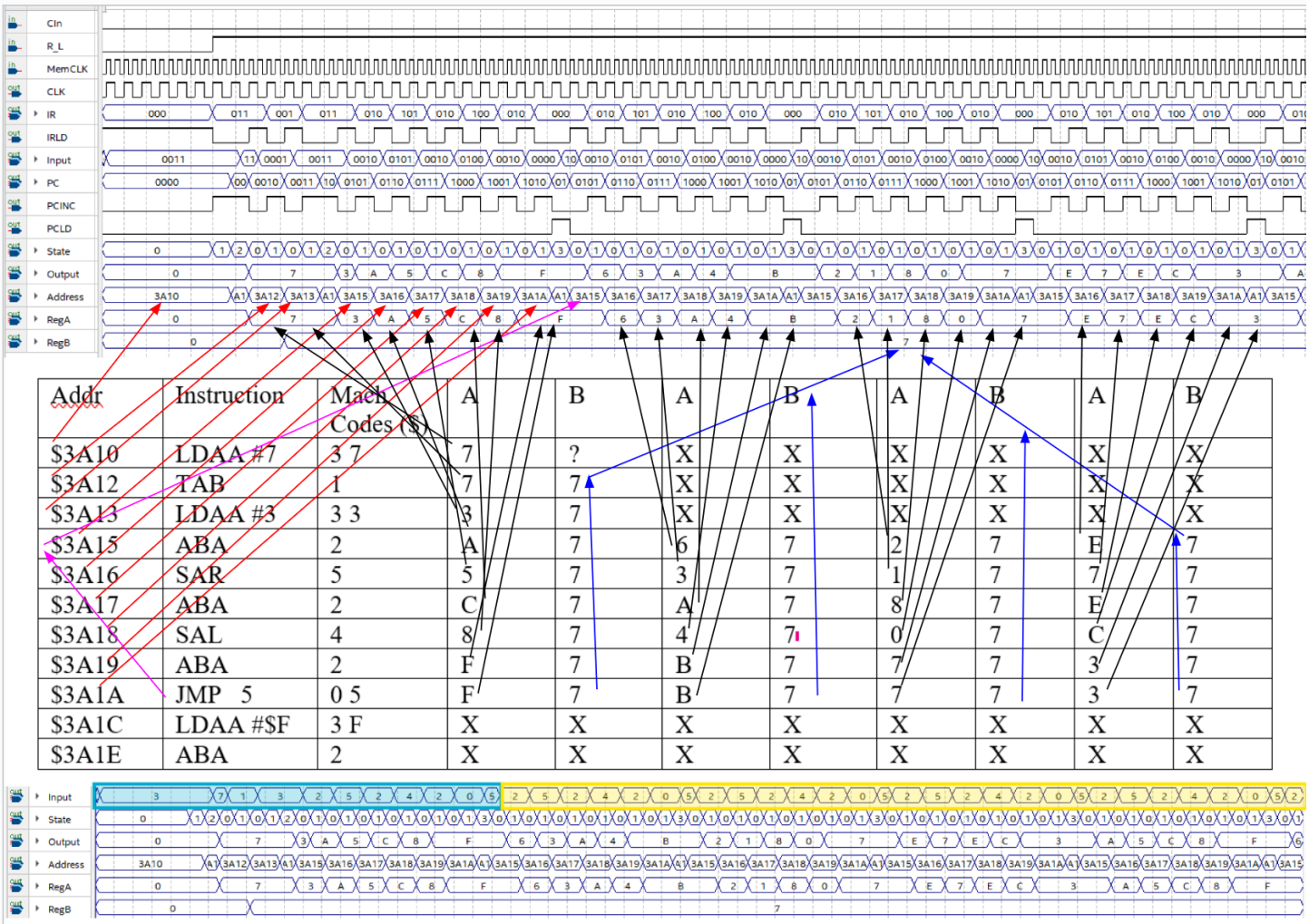
Figure 9: BDF Schematic for LAB6_Part2 2/2 (Part 2)

Addr	Instruction	Mach Codes ($)	A	B	A	B	A	B	A	B
$3A10	LDAA #7	3 7	7	?	X	X	X	X	X	X
$3A12	TAB	1	7	7	X	X	X	X	X	X
$3A13	LDAA #3	3 3	3	7	X	X	X	X	X	X
$3A15	ABA	2	A	7	6	7	2	7	E	7
$3A16	SAR	5	5	7	3	7	1	7	7	7
$3A17	ABA	2	C	7	A	7	8	7	E	7
$3A18	SAL	4	8	7	4	7	0	7	C	7
$3A19	ABA	2	F	7	B	7	7	7	3	7
$3A1A	JMP 5	0 5	F	7	B	7	7	7	3	7
$3A1C	LDAA #$F	3 F	X	X	X	X	X	X	X	X
$3A1E	ABA	2	X	X	X	X	X	X	X	X

Table 3: Program to Assemble (Part 2)

```
1   -- Copyright (C) 2023  Intel Corporation. All rights reserved.
2   -- Your use of Intel Corporation's design tools, logic functions
3   -- and other software and tools, and any partner logic
4   -- functions, and any output files from any of the foregoing
5   -- (including device programming or simulation files), and any
6   -- associated documentation or information are expressly subject
7   -- to the terms and conditions of the Intel Program License
8   -- Subscription Agreement, the Intel Quartus Prime License Agreement,
9   -- the Intel FPGA IP License Agreement, or other applicable license
10  -- agreement, including, without limitation, that your use is for
11  -- the sole purpose of programming logic devices manufactured by
12  -- Intel and sold by Intel or its authorized distributors.  Please
13  -- refer to the applicable agreement for further details, at
14  -- https://fpgasoftware.intel.com/eula.
15
16  -- Quartus Prime generated Memory Initialization File (.mif)
17
18  WIDTH=8;
19  DEPTH=32768;
20
21  ADDRESS_RADIX=HEX;
22  DATA_RADIX=HEX;
23
24  CONTENT BEGIN
25     [0000..3A0F]  :   00;
26     3A10  :   03;
27     3A11  :   07;
28     3A12  :   01;
29     [3A13..3A14]  :   03;
30     3A15  :   02;
31     3A16  :   05;
32     3A17  :   02;
33     3A18  :   04;
34     3A19  :   02;
35     3A1A  :   00;
36     3A1B  :   05;
37     3A1C  :   03;
38     3A1D  :   0F;
39     3A1E  :   02;
40     [3A1F..7FFF]  :   00;
41  END;
42
```

Figure 10: MIF File for LAB6_Part2 ROM (Part 2)



Addr	Instruction	Mach Codes ($)	A	B	A	B	A	B	A	B
$3A10	LDAA #7	3 7	7	?	X	X	X	X	X	X
$3A12	TAB	1	7	7	X	X	X	X	X	X
$3A13	LDAA #3	3 3	3	7	X	X	X	X	X	X
$3A15	ABA	2	A	7	6	7	2	7	E	7
$3A16	SAR	5	5	7	3	7	1	7	7	7
$3A17	ABA	2	C	7	A	7	8	7	E	7
$3A18	SAL	4	8	7	4	7	0	7	C	7
$3A19	ABA	2	F	7	B	7	7	7	3	7
$3A1A	JMP 5	0 5	F	7	B	7	7	7	3	7
$3A1C	LDAA #$F	3 F	X	X	X	X	X	X	X	X
$3A1E	ABA	2	X	X	X	X	X	X	X	X

As shown above, the simulation perfectly matches my table. Also the inputs align with the machine code pattern of 3 7 1 3 3 2 5 2 4 2 0 5, with the highlighted sequence of numbers repeating due to the jump.

Figure 11: Annotated Simulation for LAB6_Part2 (Part 2)