**EEL 3701C: Digital Logic & Computer Systems**
Revision **0**

# Lab 6 Report: **Elementary CPU Design**

Poche, Natalie
Class #: 11198
PI Name: Jaiden Magnan
April 10, 2025

## REQUIREMENTS NOT MET

N/A

## VIDEO FILE LINK

https://youtu.be/leiCw0wOtsg?si=MGPtSdBVkfT43ET1

## PROBLEMS ENCOUNTERED

Had some trouble with the simulation.

## FUTURE WORK/APPLICATIONS

Lab 6 Builds and elementary CPU using the RALU from Lab4. It adds an instruction register, program counter, and ROM to store data and instructions. Using memory to perform simple arithmetic is one of the very basic things that can be done for computers, which make way for more complex algorithms that can be based on arithmetic.

## PRE-LAB QUESTIONS OR EXERCISES

Part 1 Pre-Lab Questions

1. Why did we require the new instruction register in this design?
   We need a new instruction register in the design to automate the code that go into the MUX select outputs.
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?
   If you want to read or fetch the INPUT from memory, you could add a ROM to store the value every CLK cycle automatically.
3. How would you add more instructions (i.e., 8 instead of 4) to the controller?
   To increase the number of instructions, add another bit to IR from two to three.

Part 2 Pre-Lab Questions

1. Why do we need the extra states in the LDAA and JMP instruction paths?
   We need the extra states in the LDAA and JMP instruction paths to set the address value to the PC.
2. What do you need to do to the address lines to get your program to start address $37D0 (instead of $2B70)?
   To get to address $37D0, you need to directly connect the Adress[14..4] wire to the hex representation of $37D0

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

Each section of the pre-lab requirements should be completed separately, and in order. Include each of the following items in order. Note that some of these items will not apply to every lab. Anything scanned or copied ***must be clear and legible***.

- Logic equations. (Note that logic equations do not contain activation levels.)
- **All** tables and figures should have captions with Figure/Table numbers and a description of for which part of the lab it references, e.g., ***Table 3****: Truth Table for Part B*.
- Truth tables and voltage tables (and/or next-state truth tables).
- When applicable, include Karnaugh Maps (i.e., K-Maps).
- Include hand-drawn circuits (when required). Label all input and output activation-levels and intermediate equations in the circuits.
- Include screenshots of the BDF designs of circuits.
    - Label all input and output activation-levels, i.e., use _L suffix for active-low signals and no suffix for active-high signals. **Add chip and pin numbers to any schematic that will be constructed.**
    - Images should be large enough so that inputs, outputs, labels, and parts are clearly visible and distinguishable to any reader.
    - Each BDF should have the following info on the top left corner (similar to the top right of this page):

      *Last Name, First Name*
      *Lab #, Part #*
      *Class #*
      *PI Name:*
      *Description:* (short description of what is to be accomplished in the design; perhaps an equation)

    - In Windows, I use the ***Snipping Tool***, which is now built into Windows. Just type "snip" in the Windows search box and then select ***Snipping Tool***.
- When necessary, include ASM Charts. These can be hand-drawn, but clear and legible. We recommend that you use resources like https://www.draw.io/ to create computer-generated ASMs.
- Truth tables or next-state truth tables should have the following characteristics.
    - Can be either typed or hand-written and scanned (must be clear and legible)
    - Must be in **counting order** (i.e., inputs of 000, 001, 010, 011, …, 111)
    - Clearly distinguish inputs from outputs (see the example below that uses a thick line)
    - If you are designing a state machine or a controller, clearly indicate and separate signal values both before the clock and after the clock (i.e., Q1 and Q1$^+$, respectively)
    - Tip: divide rows into consecutive groups of 4 (or 2 or 8) to make it easier for both you and your PI to read.
    - **Example**

| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Table: Caption for above table. It should reference the part of the lab, e.g., "Table 3: Truth Table for Part B."

- Voltage tables should have the following characteristics.
  - Must be in counting order (i.e., inputs of LLL, LLH, LHL, … , HHH)
  - Use similar formatting to truth tables (described above)

- **Example**

| A(H) | B(H) | C(L) | Y(L) |
|---|---|---|---|
| L | L | L | H |
| L | L | H | H |
| L | H | L | L |
| L | H | H | H |
| H | L | L | H |
| H | L | H | H |
| H | H | L | H |
| H | H | H | L |

- Include **meaningfully annotated** functional simulations.
  - Using the grouping tool, ***group as many signals together as possible*** (when it makes sense to group them)! If you are simulating a basic logic equation, group all the inputs together. If you are simulating a circuit that includes MSI elements, group signals of the form $X_{N-0}$. The most-significant bit should appear first, ending with the least-significant bit. If you are simulating an ALU, group the buses together as just described.
  - Not every row of your voltage table must be annotated in the waveform simulation, but your choices of rows that you annotate must be ***encompassing***
  - If you are designing a state machine or a controller, your CLK signal should appear ***at the top*** of your inputs and outputs. The general order is CLK -> Reset -> state bits -> inputs -> outputs.
  - ***Tip***: Use Microsoft Paint to annotate your waveforms. An alternative is to print out the waveforms, annotate them by hand, then scan and upload
  - ***Hint***: Consider a truth table where the output is true in significantly less cases than it is false (or vice versa). If the output signal is active-high, it would be wise to annotate only the cases where the output voltage is HIGH.
- Include every line of VHDL programs, including both ***architecture*** and ***behavior*** sections.
- Include every line of any **MIF** files. If these are associated with assembly language programs, you can either put the assembly code as comments or separately include assembly language programs

1. First RALU Controller

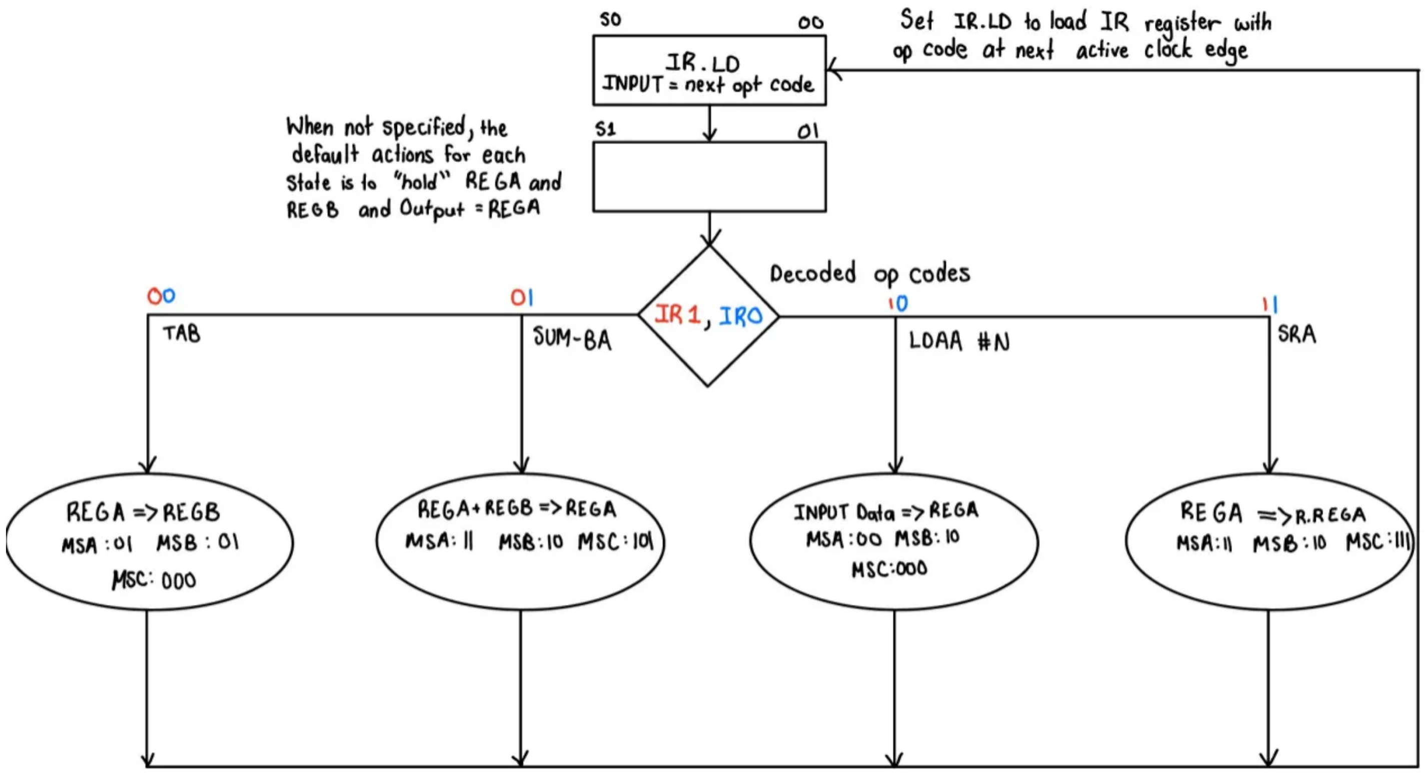


*Table 1: Lab 6 Part 1.1 - ASM of RALU Controller*

| $Q_0$ | $IR_1$ | $IR_0$ | $Q_0^+$ | $MSA_1$ | $MSA_0$ | $MSB_1$ | $MSB_0$ | $MSC_2$ | $MSC_1$ | $MSC_0$ | IR.LD |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | - | - | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |

*Table 2: Lab 6 Part 1.2 - Next State Truth Table for RALU Controller*

$IR.LD = \overline{Q_0}$

$Q^+/D = \overline{Q_0}$

$MSA(1) = \overline{IR(1)}\, IR(0)\, Q_0 + IR(1)\, IR(0)\, Q_0$

$MSA(0) = \overline{IR(1)}\, \overline{IR(0)}\, Q_0 + \overline{IR(1)}\, IR(0)\, Q_0 + IR(1)\, IR(0)\, Q_0$

$MSB(1) = \overline{IR(1)}\, IR(0)\, Q_0 + IR(1)\, \overline{IR(0)}\, Q_0 + IR(1)\, IR(0)\, Q_0$

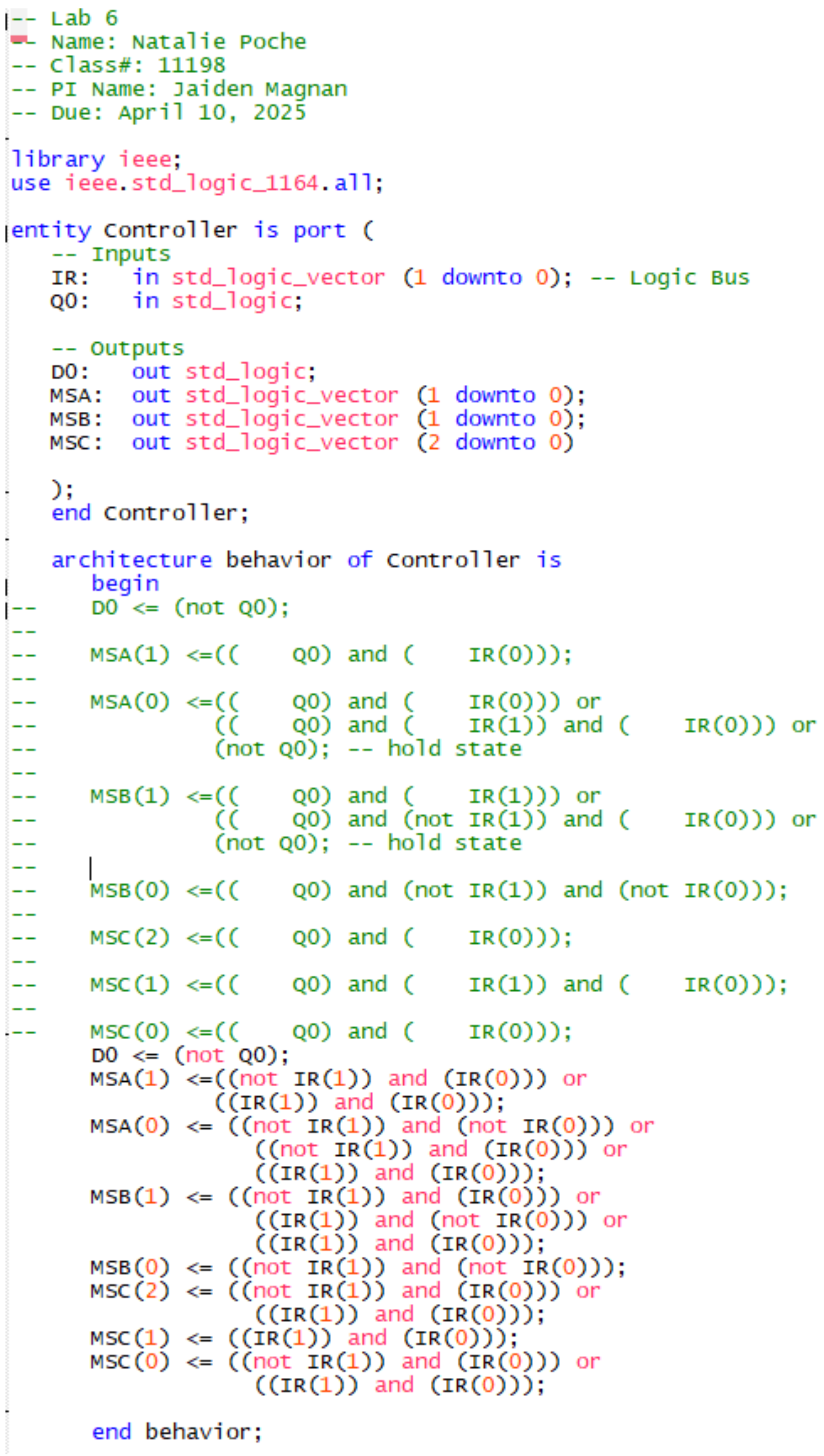
$MSB(0) = \overline{IR(1)}\, \overline{IR(0)}\, Q_0$

$MSC(2) = \overline{IR(1)}\, IR(0)\, Q_0 + IR(1)\, IR(0)\, Q_0$

$MSC(1) = IR(1)\, IR(0)\, Q_0$

$MSC(0) = \overline{IR(1)}\, IR(0)\, Q_0 + IR(1)\, IR(0)\, Q_0$

*Table 3: Lab 6 Part 1 - Logic Equations for RALU Controller*



```
-- Lab 6
-- Name: Natalie Poche
-- Class#: 11198
-- PI Name: Jaiden Magnan
-- Due: April 10, 2025

library ieee;
use ieee.std_logic_1164.all;

entity Controller is port (
   -- Inputs
   IR:   in std_logic_vector (1 downto 0); -- Logic Bus
   Q0:   in std_logic;

   -- Outputs
   D0:   out std_logic;
   MSA:  out std_logic_vector (1 downto 0);
   MSB:  out std_logic_vector (1 downto 0);
   MSC:  out std_logic_vector (2 downto 0)

   );
   end Controller;

   architecture behavior of Controller is
      begin
--    D0 <= (not Q0);
--
--    MSA(1) <=((    Q0) and (    IR(0)));
--
--    MSA(0) <=((    Q0) and (    IR(0))) or
--             ((    Q0) and (    IR(1)) and (    IR(0))) or
--             (not Q0); -- hold state
--
--    MSB(1) <=((    Q0) and (    IR(1))) or
--             ((    Q0) and (not IR(1)) and (    IR(0))) or
--             (not Q0); -- hold state
--
--    MSB(0) <=((    Q0) and (not IR(1)) and (not IR(0)));
--
--    MSC(2) <=((    Q0) and (    IR(0)));
--
--    MSC(1) <=((    Q0) and (    IR(1)) and (    IR(0)));
--
--    MSC(0) <=((    Q0) and (    IR(0)));
      D0 <= (not Q0);
      MSA(1) <=((not IR(1)) and (IR(0))) or
               ((IR(1)) and (IR(0)));
      MSA(0) <= ((not IR(1)) and (not IR(0))) or
                ((not IR(1)) and (IR(0))) or
                ((IR(1)) and (IR(0)));
      MSB(1) <= ((not IR(1)) and (IR(0))) or
                ((IR(1)) and (not IR(0))) or
                ((IR(1)) and (IR(0)));
      MSB(0) <= ((not IR(1)) and (not IR(0)));
      MSC(2) <= ((not IR(1)) and (IR(0))) or
                ((IR(1)) and (IR(0)));
      MSC(1) <= ((IR(1)) and (IR(0)));
      MSC(0) <= ((not IR(1)) and (IR(0))) or
                ((IR(1)) and (IR(0)));

      end behavior;
```

*Table 4: Lab 6 Part 1.3 - VHDL Code for RALU Controller*

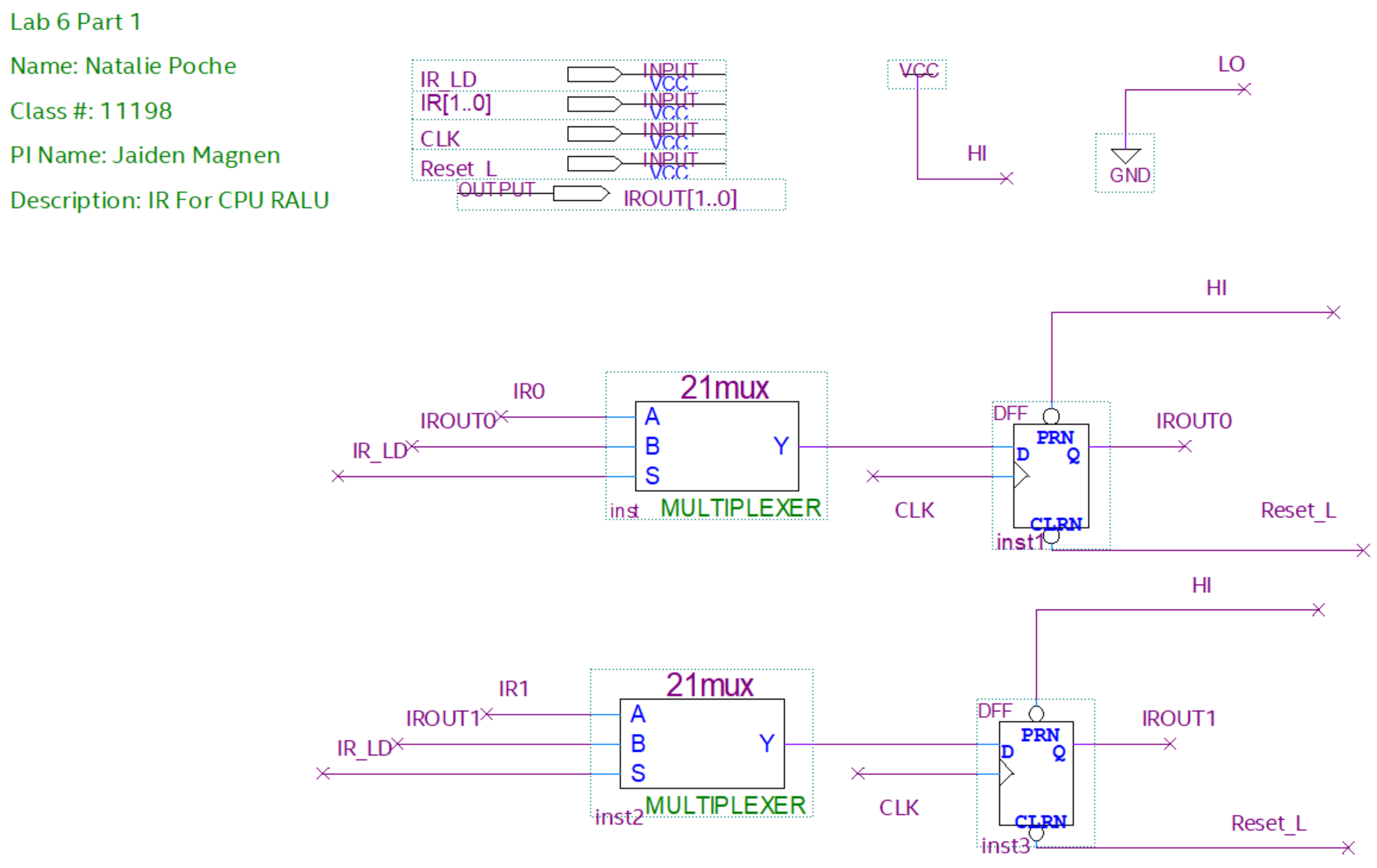


*Table 5: Lab 6 Part 1 - BDF of IR Part of RALU*

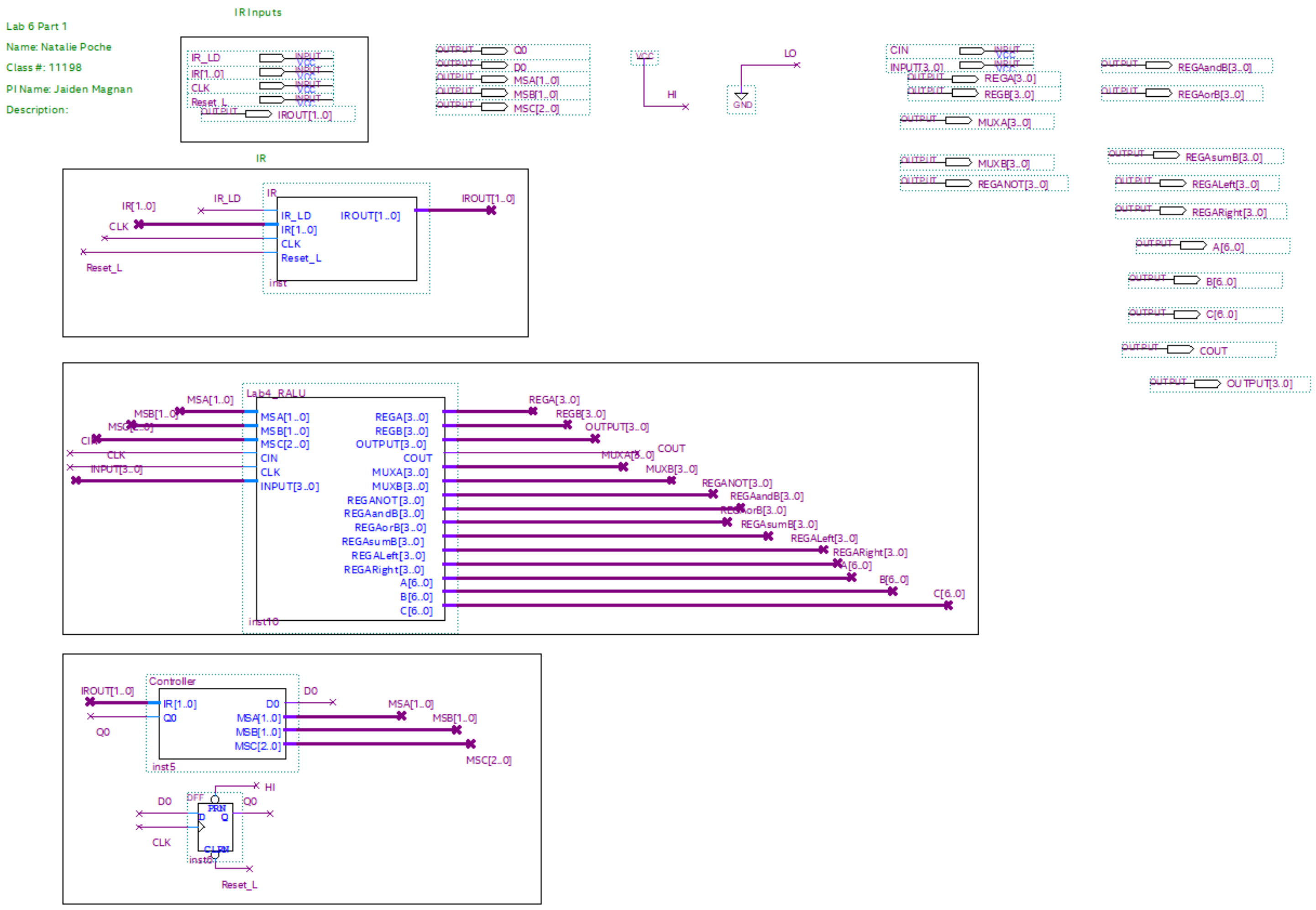


*Table 6: Lab 6 Part 1 - BDF of RALU*

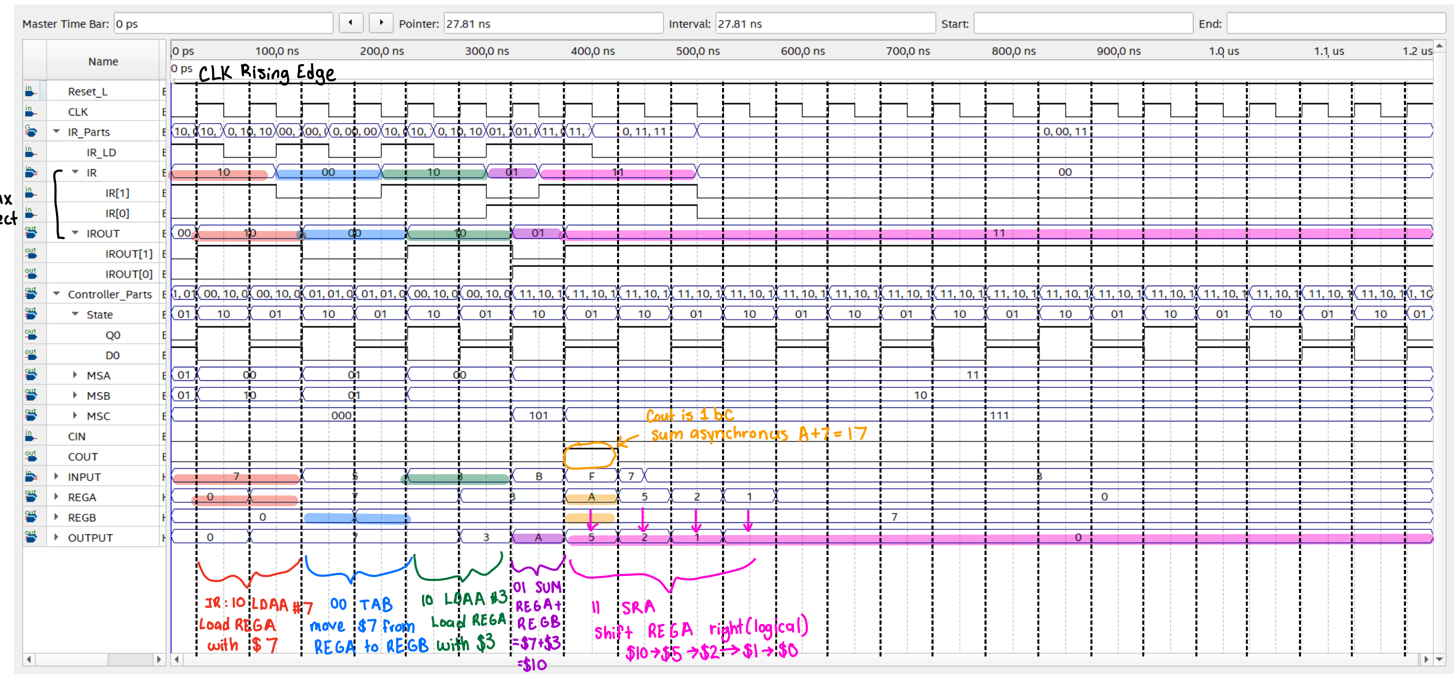


*Table 7: Lab 6 Part 1 - Functional Simulation of RALU*

## 2. Second RALU Controller with ROM

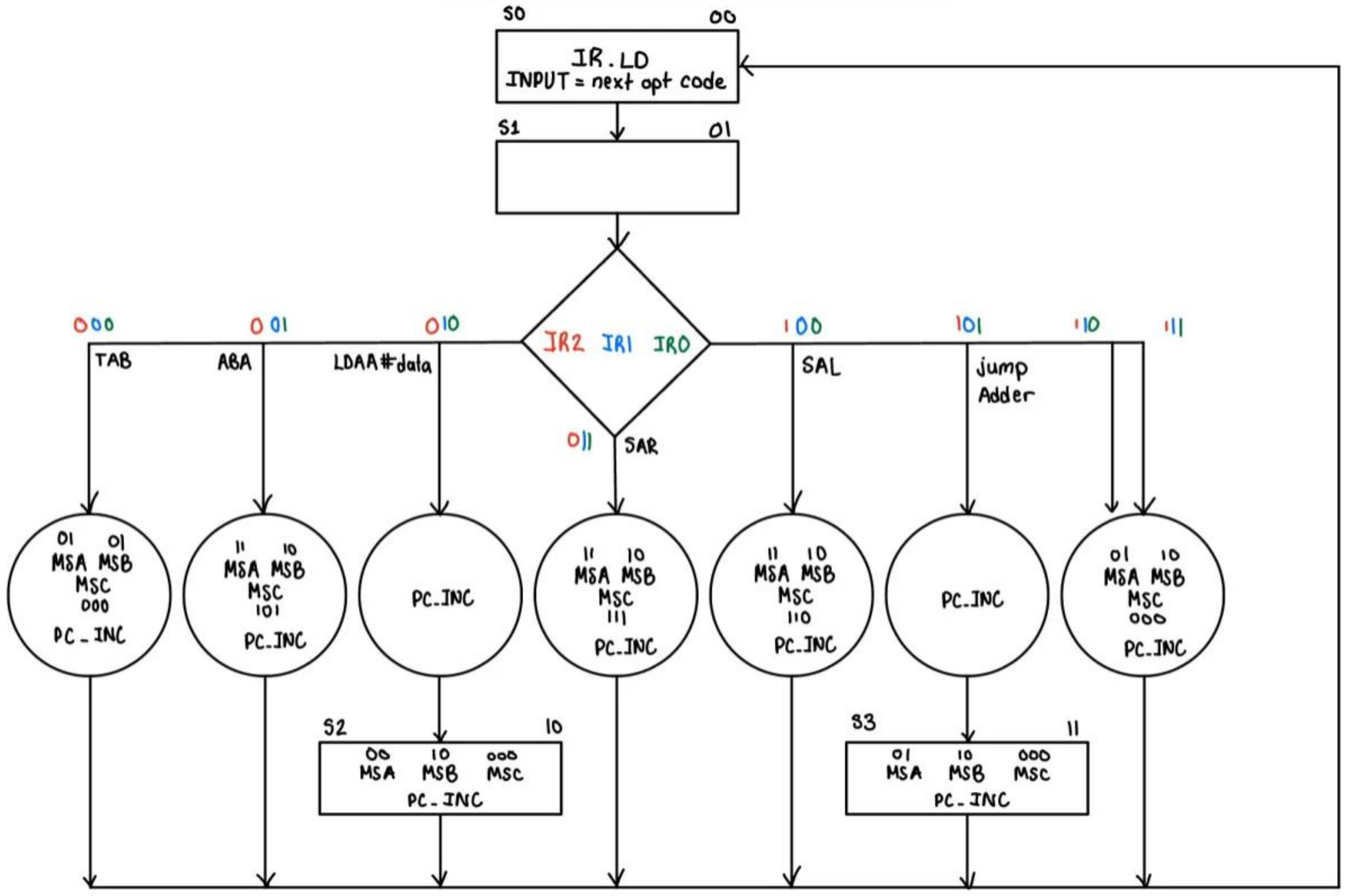
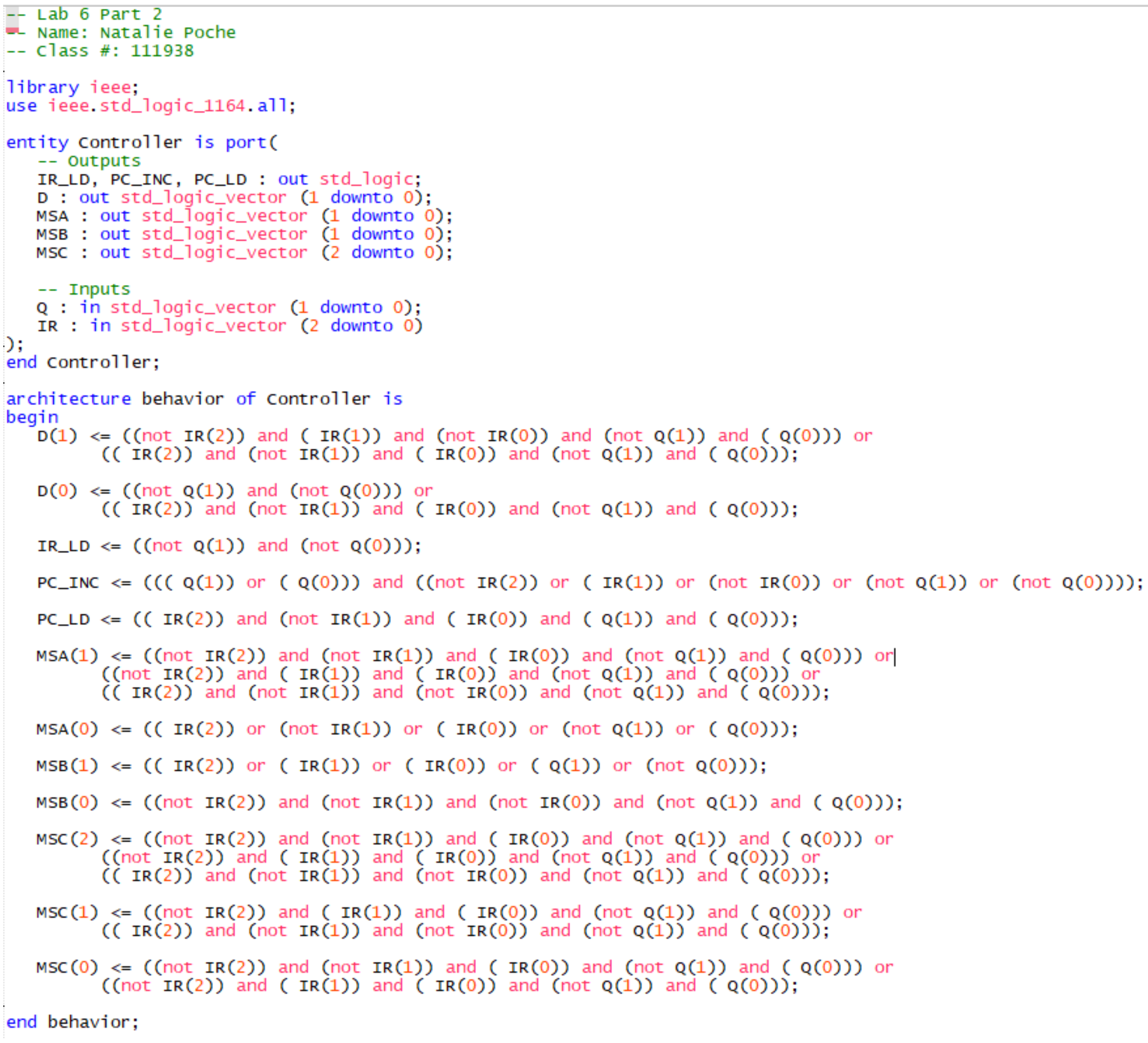

S0 00
IR.LD
INPUT = next opt code
S1 01
IR2 IR1 IR0
000 TAB
001 ABA
010 LDAA#data
011 SAR
100 SAL
101 jump Adder
110
111
01 01 MSA MSB MSC 000 PC_INC
11 10 MSA MSB MSC 101 PC_INC
PC_INC
11 10 MSA MSB MSC 111 PC_INC
11 10 MSA MSB MSC 110 PC_INC
PC_INC
01 10 MSA MSB MSC 000 PC_INC
S2 10
00 MSA 10 MSB 000 MSC PC_INC
S3 11
01 MSA 10 MSB 000 MSC PC_INC

*Table 8: Lab 6 Part 2.1 - ASM for RALU with ROM*

| $IR_2$ | $IR_1$ | $IR_0$ | $Q_1$ | $Q_0$ | $Q_1^+$ | $Q_0^+$ | $MSA_1$ | $MSA_0$ | $MSB_1$ | $MSB_0$ | $MSC_2$ | $MSC_1$ | $MSC_0$ | IR.LD | PC.INC | PC.LD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | - | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0* | 1* | 0* | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1* | 0* | 1* | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

*Table 9: Lab 6 Part 2.2 - Next State Truth Table for RALU with ROM*

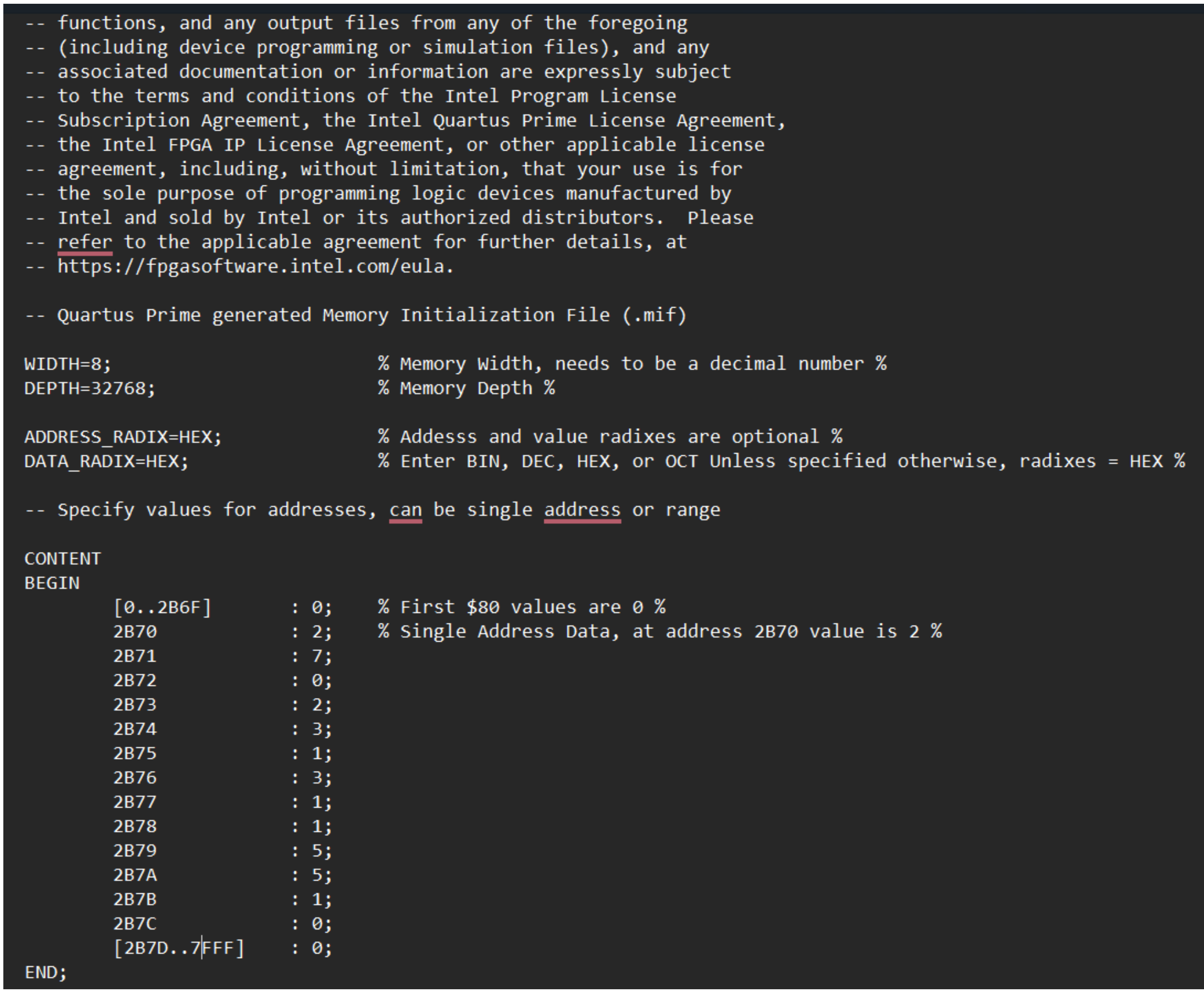


```
-- Lab 6 Part 2
-- Name: Natalie Poche
-- Class #: 111938

library ieee;
use ieee.std_logic_1164.all;

entity Controller is port(
   -- Outputs
   IR_LD, PC_INC, PC_LD : out std_logic;
   D : out std_logic_vector (1 downto 0);
   MSA : out std_logic_vector (1 downto 0);
   MSB : out std_logic_vector (1 downto 0);
   MSC : out std_logic_vector (2 downto 0);

   -- Inputs
   Q : in std_logic_vector (1 downto 0);
   IR : in std_logic_vector (2 downto 0)
);
end Controller;

architecture behavior of Controller is
begin
   D(1) <= ((not IR(2)) and ( IR(1)) and (not IR(0)) and (not Q(1)) and ( Q(0))) or
         (( IR(2)) and (not IR(1)) and ( IR(0)) and (not Q(1)) and ( Q(0)));

   D(0) <= ((not Q(1)) and (not Q(0))) or
         (( IR(2)) and (not IR(1)) and ( IR(0)) and (not Q(1)) and ( Q(0)));

   IR_LD <= ((not Q(1)) and (not Q(0)));

   PC_INC <= ((( Q(1)) or ( Q(0))) and ((not IR(2)) or ( IR(1)) or (not IR(0)) or (not Q(1)) or (not Q(0))));

   PC_LD <= (( IR(2)) and (not IR(1)) and ( IR(0)) and ( Q(1)) and ( Q(0)));

   MSA(1) <= ((not IR(2)) and (not IR(1)) and ( IR(0)) and (not Q(1)) and ( Q(0))) or
         ((not IR(2)) and ( IR(1)) and ( IR(0)) and (not Q(1)) and ( Q(0))) or
         (( IR(2)) and (not IR(1)) and (not IR(0)) and (not Q(1)) and ( Q(0)));

   MSA(0) <= (( IR(2)) or (not IR(1)) or ( IR(0)) or (not Q(1)) or ( Q(0)));

   MSB(1) <= (( IR(2)) or ( IR(1)) or ( IR(0)) or ( Q(1)) or (not Q(0)));

   MSB(0) <= ((not IR(2)) and (not IR(1)) and (not IR(0)) and (not Q(1)) and ( Q(0)));

   MSC(2) <= ((not IR(2)) and (not IR(1)) and ( IR(0)) and (not Q(1)) and ( Q(0))) or
         ((not IR(2)) and ( IR(1)) and ( IR(0)) and (not Q(1)) and ( Q(0))) or
         (( IR(2)) and (not IR(1)) and (not IR(0)) and (not Q(1)) and ( Q(0)));

   MSC(1) <= ((not IR(2)) and ( IR(1)) and ( IR(0)) and (not Q(1)) and ( Q(0))) or
         (( IR(2)) and (not IR(1)) and (not IR(0)) and (not Q(1)) and ( Q(0)));

   MSC(0) <= ((not IR(2)) and (not IR(1)) and ( IR(0)) and (not Q(1)) and ( Q(0))) or
         ((not IR(2)) and ( IR(1)) and ( IR(0)) and (not Q(1)) and ( Q(0)));

end behavior;
```

*Table 10: Lab 6 Part 2 - VHDL Code for Controller for RALU with ROM*

| Address | | Mach Codes | A | B | A | B | A | B | A | B |
|---|---|---|---|---|---|---|---|---|---|---|
| $2B70 | LDAA #7 | $2, #7 | 0111 | 0000 | | | | | | |
| $2B72 | TAB | $0 | 0111 | 0111 | | | | | | |
| $2B73 | LDAA #3 | $2, #3 | 0011 | 0111 | | | | | | |
| $2B75 | ABA | $1 | 1010 | 0111 | 1010 | 0111 | 1010 | 0111 | 1010 | 0111 |
| $2B76 | SAR | $3 | 0101 | 0111 | 0101 | 0111 | 0101 | 0111 | 0101 | 0111 |
| $2B77 | ABA | $1 | 1100 | 0111 | 1100 | 0111 | 1100 | 0111 | 1100 | 0111 |
| $2B78 | ABA | $1 | 0011 | 0111 | 0011 | 0111 | 0011 | 0111 | 0011 | 0111 |
| $2B79 | JMP 5 | $5, #5 | 0011 | 0111 | 0011 | 0111 | 0011 | 0111 | 0011 | 0111 |
| $2B7B | ABA | $1 | | | | | | | | |
| $2B7C | TAB | $0 | | | | | | | | |

*Table 11: Lab 6 Part 2 - Program for ROM MIF file*



```
-- functions, and any output files from any of the foregoing
-- (including device programming or simulation files), and any
-- associated documentation or information are expressly subject
-- to the terms and conditions of the Intel Program License
-- Subscription Agreement, the Intel Quartus Prime License Agreement,
-- the Intel FPGA IP License Agreement, or other applicable license
-- agreement, including, without limitation, that your use is for
-- the sole purpose of programming logic devices manufactured by
-- Intel and sold by Intel or its authorized distributors.  Please
-- refer to the applicable agreement for further details, at
-- https://fpgasoftware.intel.com/eula.

-- Quartus Prime generated Memory Initialization File (.mif)

WIDTH=8;                        % Memory Width, needs to be a decimal number %
DEPTH=32768;                    % Memory Depth %

ADDRESS_RADIX=HEX;              % Addesss and value radixes are optional %
DATA_RADIX=HEX;                 % Enter BIN, DEC, HEX, or OCT Unless specified otherwise, radixes = HEX %

-- Specify values for addresses, can be single address or range

CONTENT
BEGIN
        [0..2B6F]       : 0;    % First $80 values are 0 %
        2B70            : 2;    % Single Address Data, at address 2B70 value is 2 %
        2B71            : 7;
        2B72            : 0;
        2B73            : 2;
        2B74            : 3;
        2B75            : 1;
        2B76            : 3;
        2B77            : 1;
        2B78            : 1;
        2B79            : 5;
        2B7A            : 5;
        2B7B            : 1;
        2B7C            : 0;
        [2B7D..7FFF]    : 0;
END;
```

*Table 12: Lab 6 Part 2 MIF file for ROM Component for RALU with ROM*

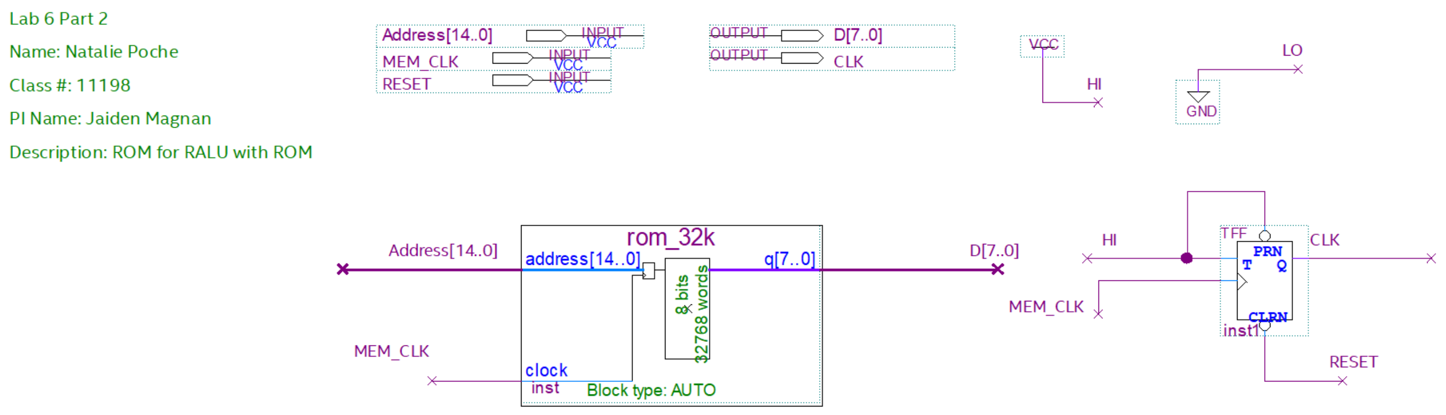


*Table 13: Lab 6 Part 2 - BDF of ROM Component 32K x 8*

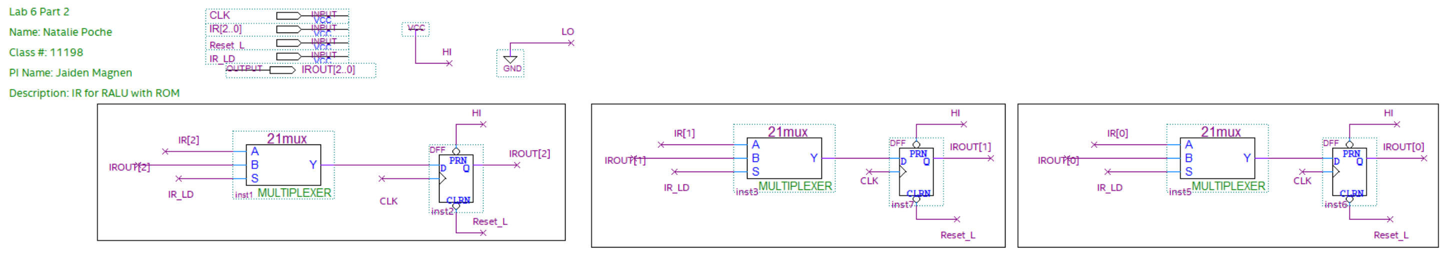


*Table 14: Lab 6 Part 2 - BDF of IR Component with IR2, IR1, and IR0*

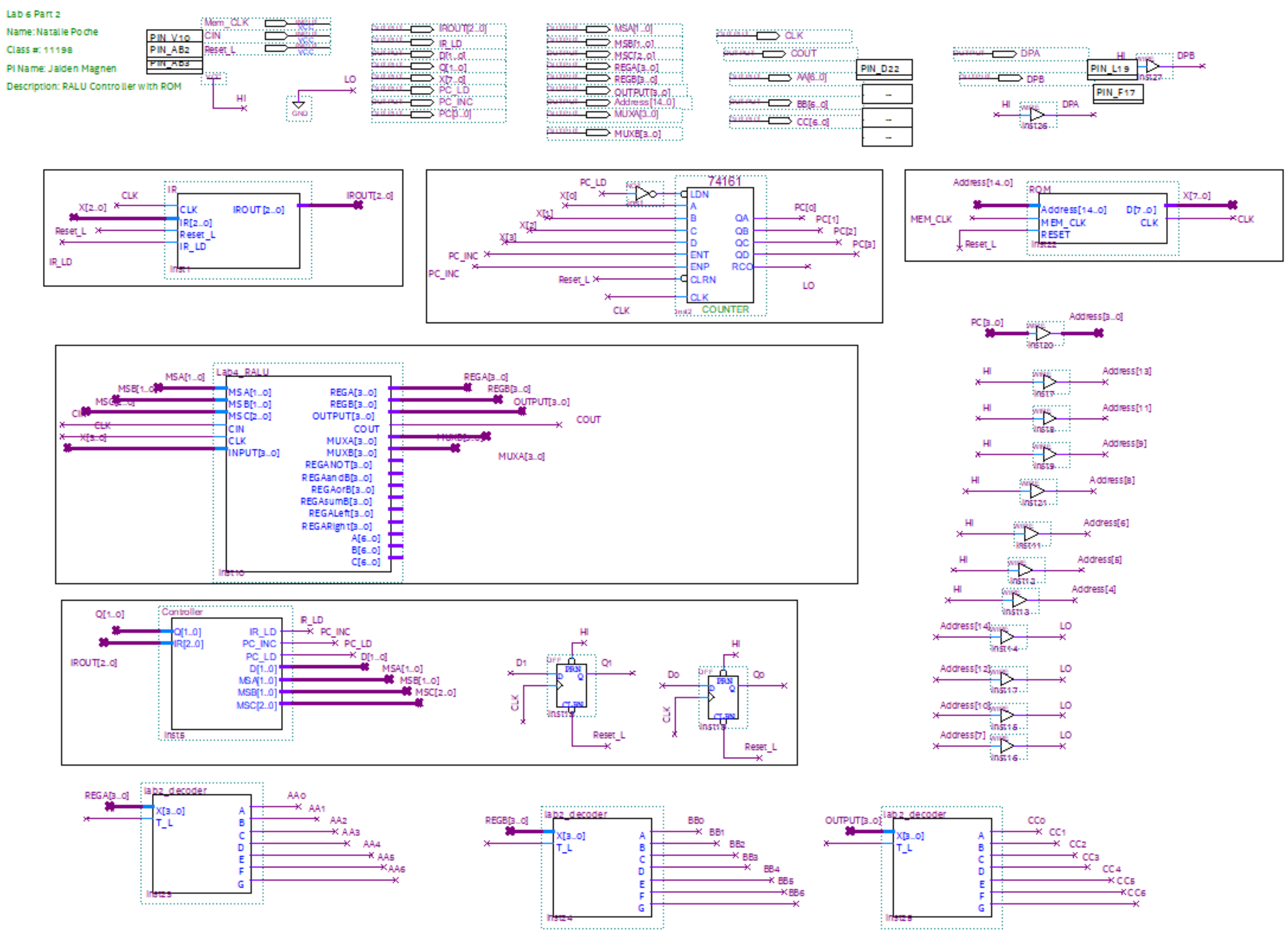
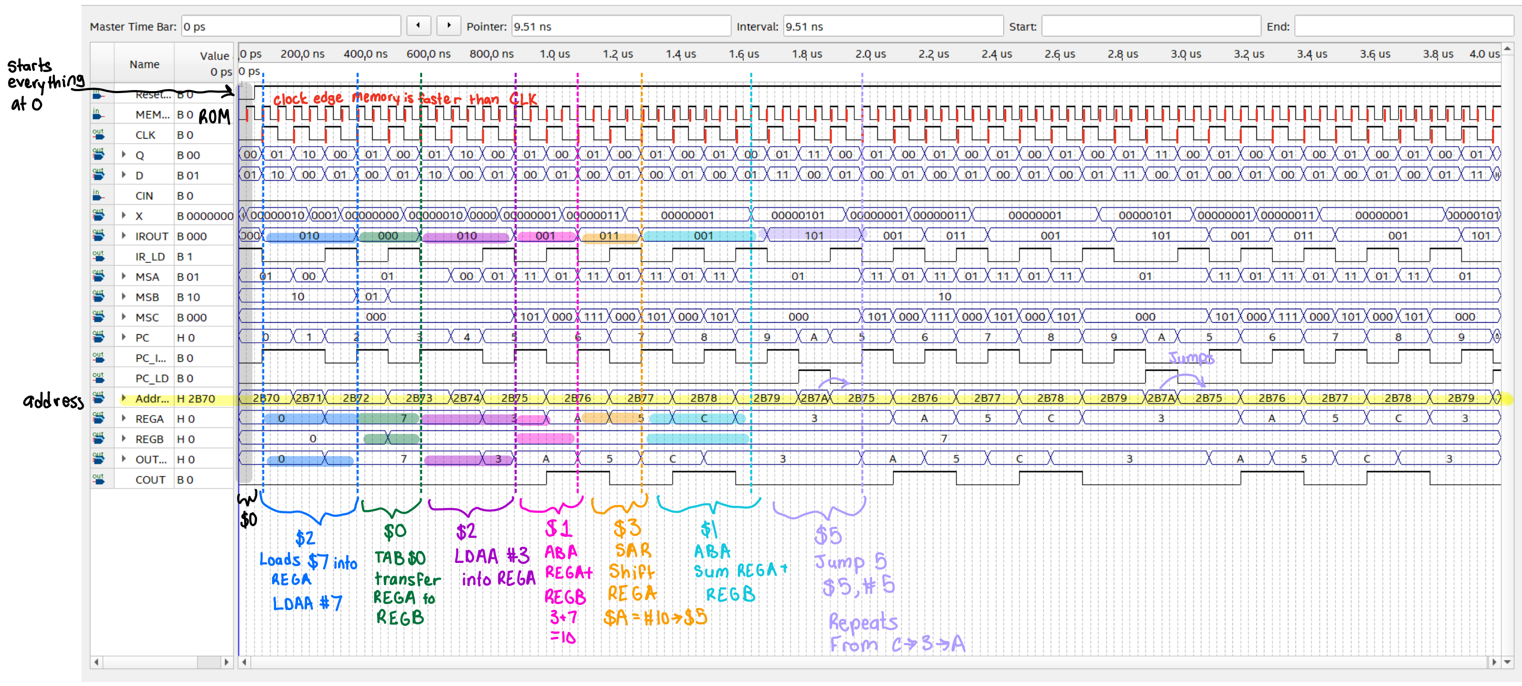


*Table 15: Lab 6 Part 2 - BDF of RALU with ROM*

**EEL 3701C: Digital Logic & Computer Systems**
Revision 0

# Lab 6 Report: **Elementary CPU Design**



Master Time Bar: 0 ps
Pointer: 9.51 ns
Interval: 9.51 ns
Start:
End:

| Name | Value 0 ps |
|---|---|
| Reset... | B 0 |
| MEM... | B 0 |
| CLK | B 0 |
| Q | B 00 |
| D | B 01 |
| CIN | B 0 |
| X | B 0000000 |
| IROUT | B 000 |
| IR_LD | B 1 |
| MSA | B 01 |
| MSB | B 10 |
| MSC | B 000 |
| PC | H 0 |
| PC_I... | B 0 |
| PC_LD | B 0 |
| Addr... | H 2B70 |
| REGA | H 0 |
| REGB | H 0 |
| OUT... | H 0 |
| COUT | B 0 |

Starts everything at 0
ROM
address
clock edge memory is faster than CLK
$0
$2 Loads $7 into REGA LDAA #7
$0 TAB $0 transfer REGA to REGB
$2 LDAA #3 into REGA
$1 ABA REGA+ REGB 3+7 =10
$3 SAR Shift REGA $A=#10→$5
$1 ABA Sum REGA+ REGB
$5 Jump 5 $5, #5
Repeats From C→3→A
Jumps

*Table 16: Lab 6 Part 2 - Functional Simulation of RALU with ROM Component*