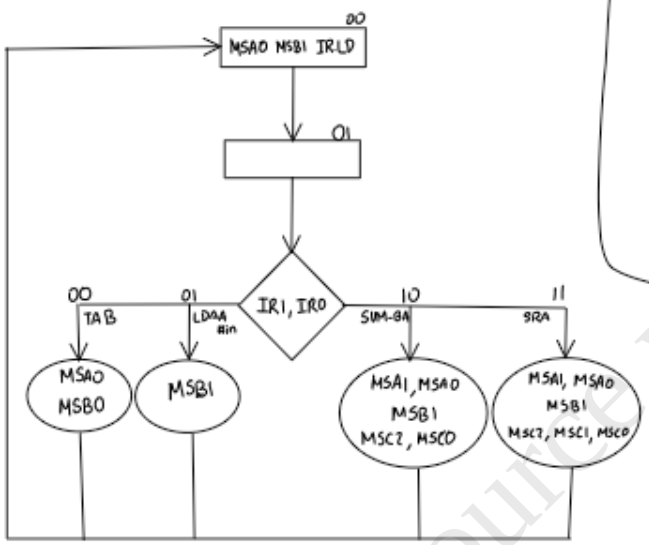
Lab 6 – Elementary CPU Design: Fetching Instructions From a ROM

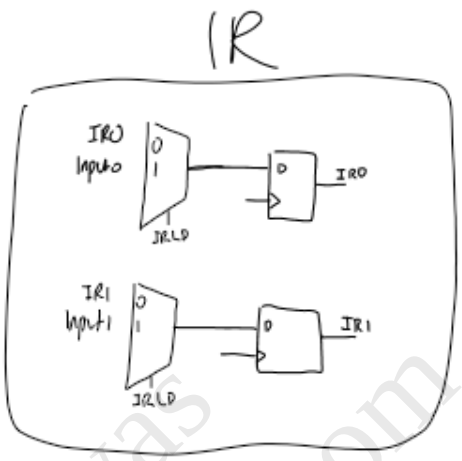
NAME: Juan Valbuena
Lab Section: 1491

Part 1

Part 1: 1st ALU Controller

1. ASM chart





TAB → MSA1 MSA0 MSB1 MSB0 MSC2 MSC1 MSC0
0 1 0 1 0 0 0

LDAA #in → MSA1 MSA0 MSB1 MSB0 MSC2 MSC1 MSC0
0 0 1 0 0 0 0

SUM_BA → MSA1 MSA0 MSB1 MSB0 MSC2 MSC1 MSC0
1 1 1 0 1 0 1

SRA → MSA1 MSA0 MSB1 MSB0 MSC2 MSC1 MSC0
1 1 1 0 1 1 1

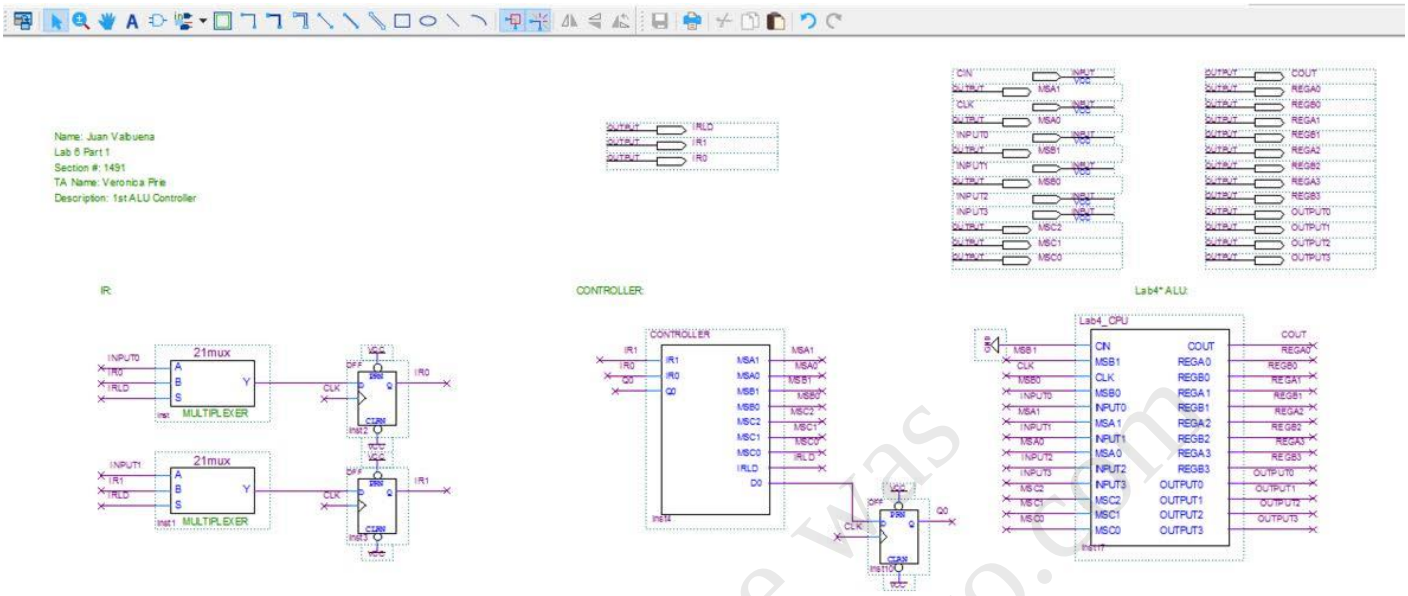
2. Next State Truth Table (controller)

IR1	IR 0	Q_0	D_0	Q_0^*	MSA1	MSA0	MSB1	MSB0	MSC2	MSC1	MSC0	IR.LD
X	X	0	1	1	0	1	1	0	0	0	0	1
0	0	1	0	0	0	1	0	1	0	0	0	0
0	1	1	0	0	0	0	1	0	0	0	0	0
1	0	1	0	0	1	1	1	0	1	0	1	0
1	1	1	0	0	1	1	1	0	1	1	1	0

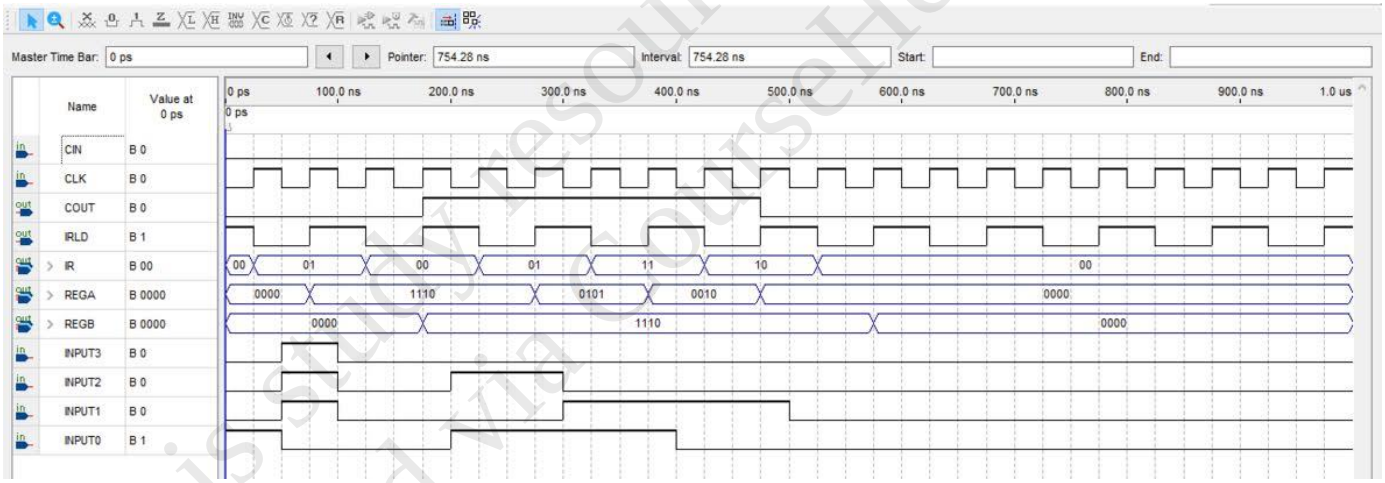
VHDL:

```
1   library ieee; use ieee.std_logic_1164.all;
2   entity CONTROLLER is port (
3         IR1, IR0, Q0: in bit;
4         MSA1, MSA0: out bit;
5         MSB1, MSB0: out bit;
6         MSC2, MSC1, MSC0, IRLD, D0: out bit
7   );
8   end CONTROLLER;
9   architecture logic OF CONTROLLER IS
10  begin
11
12  --D0 = /Q0
13
14     D0 <= (NOT Q0);
15
16  --MSA1 = (IR1*/IR0*Q0) + (IR1*IR0*Q0)
17
18     MSA1 <= (IR1 AND (NOT IR0) AND Q0)
19          OR (IR1 AND IR0 AND Q0);
20
21  --MSA0 = (IR0 + /IR0 + /Q0)
22
23     MSA0 <= (IR1 OR (NOT IR0) OR (NOT Q0));
24
25  --MSB1 = (IR1 + IR0 + /Q0)
26
27     MSB1 <= (IR1 OR IR0 OR (NOT Q0));
28
29  --MSB0 = (/IR1*/IR0*Q0)
30
31     MSB0 <= ((NOT IR1) AND (NOT IR0) AND Q0);
32
33  --MSC2 = (IR1*/IR0*Q0) + (IR1*IR0*Q0)
34
35     MSC2 <= (IR1 AND (NOT IR0) AND Q0)
36          OR (IR1 AND IR0 AND Q0);
37
38  --MSC1 = (IR1*IR0*Q0)
39
40     MSC1 <= (IR1 AND IR0 AND Q0);
41
42  --MSC0 = (IR1*/IR0*Q0) + (IR1*IR0*Q0)
43
44     MSC0 <= (IR1 AND (NOT IR0) AND Q0)
45          OR (IR1 AND IR0 AND Q0);
46
47  --IRLD = /Q0
48
49     IRLD <= (NOT Q0);
50
```

Schematic:



Simulation:

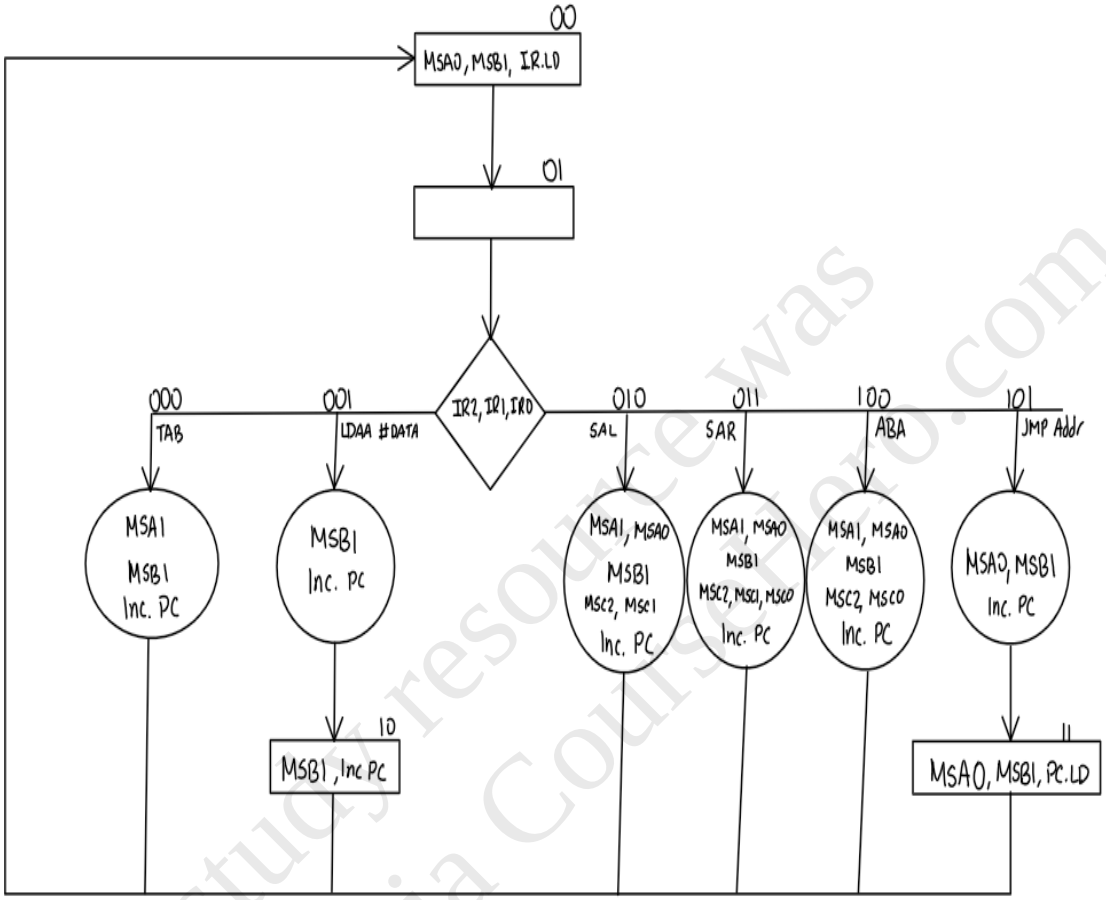


The simulation was working perfectly it was following the instructions depending on the input. For example the beginning told reg a to load a number and that number being loaded was 3 and in the next clock cycle the register now loaded 3. The next cycle reg a held its number and now reg b loaded reg a so now reg b had the number 3 loaded.

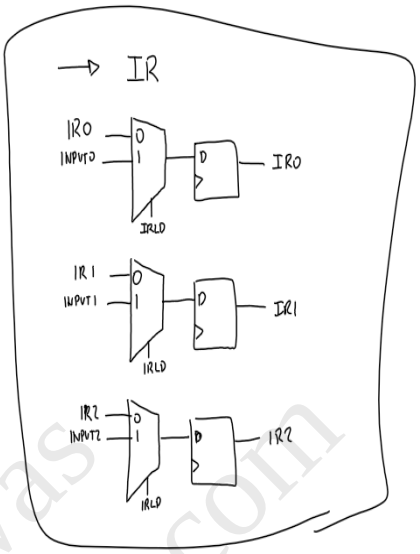
Part 2

Part 2 : 2nd ALU Controller

1. ASM Chart



	MSA1	MSA0	MSB1	MSB0	MSC2	MSC1	MSC0
TAB (000) →	0	1	0	1	0	0	0
LDAA #data (001) →	0	0	1	0	0	0	0
SAL (010) →	1	1	1	0	1	1	0
SAR (011) →	1	1	1	0	1	1	1
ABA (100) →	1	1	1	0	1	0	1
JMP Addr (101) →	0	1	1	0	0	0	0
STATE 10 →	0	0	1	0	0	0	0
STATE 11 →	0	1	1	0	0	0	0



IR2	IR1	IR0	Q_1	Q_0	D_1	D_0	Q_1^+	Q_0^+	MSA1	MSA0	MSB1	MSB0	MSC2	MSC1	MSC0	IR.LD	PC.INC	PC.LD
X	X	X	0	0	0	1	0	1	0	1	1	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0
0	0	1	0	1	1	0	1	0	0	0	1	0	0	0	0	0	1	0
0	1	0	0	1	0	0	0	0	1	1	1	0	1	1	0	0	1	0
0	1	1	0	1	0	0	0	0	1	1	1	0	1	1	1	0	1	0
1	0	0	0	1	0	0	0	0	1	1	1	0	1	0	1	0	1	0
1	0	1	0	1	1	1	1	1	0	1	1	0	0	0	0	0	1	0
X	X	X	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
X	X	X	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	1

Hand Assembled Machine Code

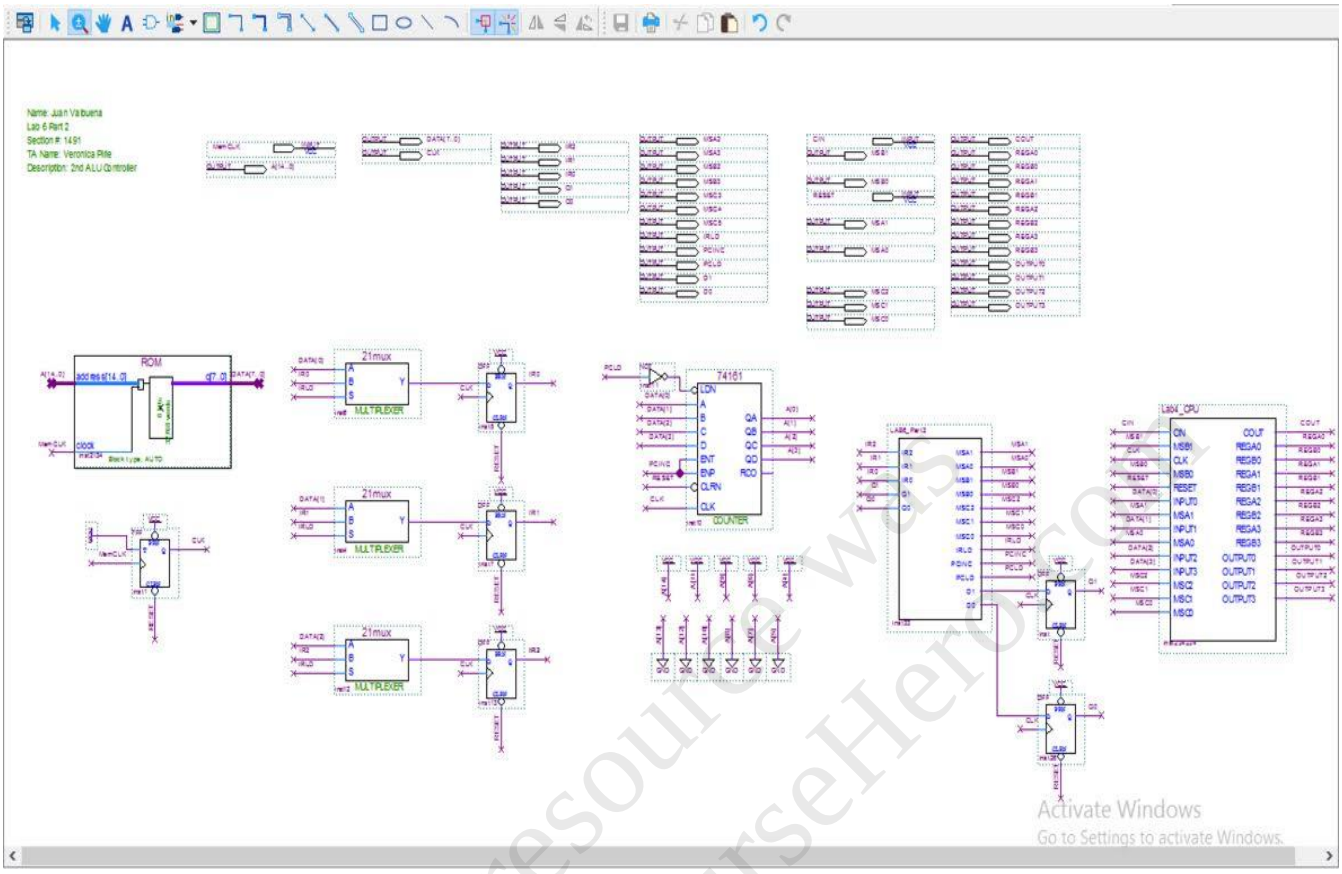
Addr		Mach. Code	A	B	A	B	A	B	A	B
$4A50-4A51	LDAA #3	001	0011	X	X	X	X	X	X	X
$4A52	TAB	000	0011	0011	X	X	X	X	X	X
$4A53-4A54	LDAA #7	001	0111	0011	0111	0000	0111	0110	0111	1000
$4A55	ABA	100	1010	0011	0111	0000	1011	0110	1111	1000
$4A56	SAR	011	0101	0011	1100	0000	0110	0110	0111	1000
$4A57	ABA	100	1000	0011	1100	0000	0110	0110	1111	1000
$4A58	SAL	010	0000	0011	1100	0000	1100	0110	1111	1000
$4A59	TAB	000	0000	0000	0110	0110	0001	0110	1111	1000
$4A5A-4A5B	JMP 3	101	0000	0000	0110	0110	0001	0110	1111	1000
	LDAA SE		X	X	X	X	X	X	X	X

VHDL:

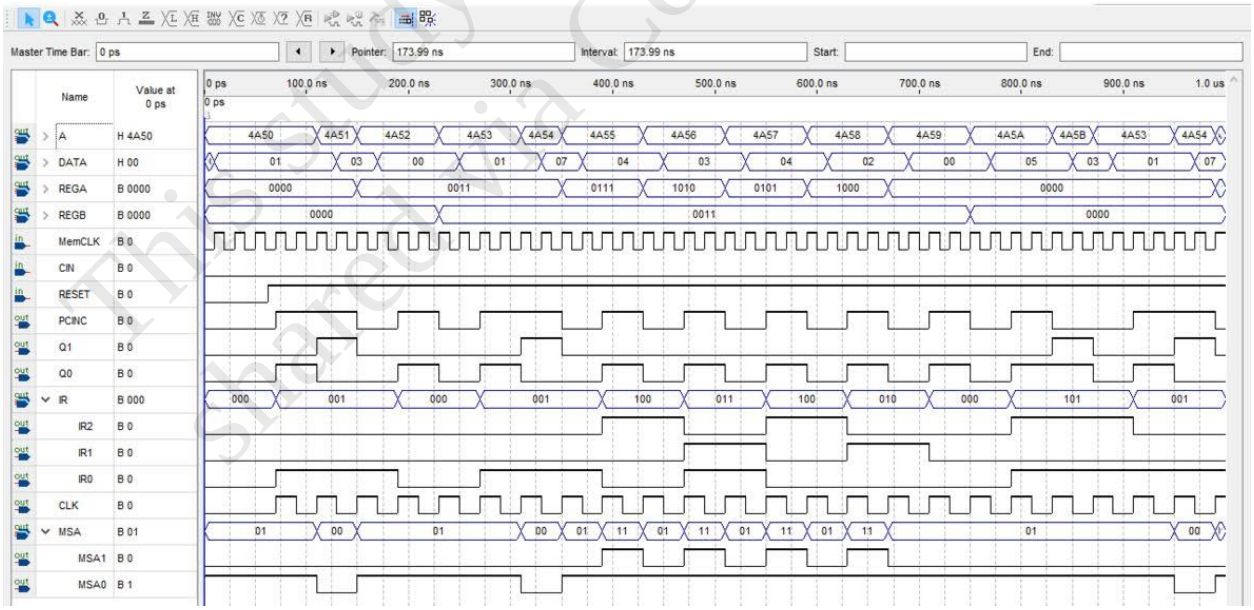
```
library ieee; use ieee.std_logic_1164.all;
entity LAB6_Part2 is port (
      IR2, IR1, IR0: IN BIT;
      Q1, Q0: IN BIT;
      MSA1, MSA0: OUT BIT;
      MSB1, MSB0: OUT BIT;
      MSC2, MSC1, MSC0: OUT BIT;
      IRLD, PCINC, PCLD, D1, D0: OUT BIT
);
end LAB6_Part2;
architecture logic OF LAB6_Part2 IS
begin

--D1 = (/IR2*/IR1*IR0*/Q1*Q0) + (IR2*/IR1*IR0*/Q1*Q0)

   D1 <= ((NOT IR2) AND (NOT IR1) AND IR0 AND (NOT Q1) AND Q0)
      OR (IR2 AND (NOT IR1) AND IR0 AND (NOT Q1) AND Q0);

--D0 = (/Q1*/Q0) + (IR2*IR0*/Q1*Q0)

   D0 <= ((NOT Q1) AND (NOT Q0))
      OR (IR2 AND IR0 AND (NOT Q1) AND Q0);

--MSA1 = (/IR2*IR1*/IR0*/Q1*Q0) + (/IR2*IR1*IR0*/Q1*Q0) + (IR2*/IR1*/IR0*/Q1*Q0)

   MSA1 <= ((NOT IR2) AND IR1 AND (NOT IR0) AND (NOT Q1) AND Q0)
        OR ((NOT IR2) AND IR1 AND IR0 AND (NOT Q1) AND Q0)
        OR (IR2 AND (NOT IR1) AND (NOT IR0) AND (NOT Q1) AND Q0);

--MSA0 = (/Q1+Q0)

   MSA0 <= ((NOT Q1) OR Q0);

--MSB1 = (IR2+IR1+IR0+Q1+/Q0)

   MSB1 <= (IR2 OR IR1 OR IR0 OR Q1 OR (NOT Q0));

--MSB0 = (/IR2*/IR1*/IR0*/Q1*Q0)

   MSB0 <= ((NOT IR2) AND (NOT IR1) AND (NOT IR0) AND (NOT Q1) AND Q0);

--MSC2 = (/IR2*IR1*/IR0*/Q1*Q0) + (/IR2*IR1*IR0*/Q1*Q0) + (IR2*/IR1*/IR0*/Q1*Q0)

   MSC2 <= ((NOT IR2) AND IR1 AND (NOT IR0) AND (NOT Q1) AND Q0)
        OR ((NOT IR2) AND IR1 AND IR0 AND (NOT Q1) AND Q0)
        OR (IR2 AND (NOT IR1) AND (NOT IR0) AND (NOT Q1) AND Q0);
```

```
--MSA1 = (/IR2*IR1*/IR0*/Q1*Q0) + (/IR2*IR1*IR0*/Q1*Q0) + (IR2*/IR1*/IR0*/Q1*Q0)

   MSA1 <= ((NOT IR2) AND IR1 AND (NOT IR0) AND (NOT Q1) AND Q0)
        OR ((NOT IR2) AND IR1 AND IR0 AND (NOT Q1) AND Q0)
        OR (IR2 AND (NOT IR1) AND (NOT IR0) AND (NOT Q1) AND Q0);

--MSA0 = (/Q1+Q0)

   MSA0 <= ((NOT Q1) OR Q0);

--MSB1 = (IR2+IR1+IR0+Q1+/Q0)

   MSB1 <= (IR2 OR IR1 OR IR0 OR Q1 OR (NOT Q0));

--MSB0 = (/IR2*/IR1*/IR0*/Q1*Q0)

   MSB0 <= ((NOT IR2) AND (NOT IR1) AND (NOT IR0) AND (NOT Q1) AND Q0);

--MSC2 = (/IR2*IR1*/IR0*/Q1*Q0) + (/IR2*IR1*IR0*/Q1*Q0) + (IR2*/IR1*/IR0*/Q1*Q0)

   MSC2 <= ((NOT IR2) AND IR1 AND (NOT IR0) AND (NOT Q1) AND Q0)
        OR ((NOT IR2) AND IR1 AND IR0 AND (NOT Q1) AND Q0)
        OR (IR2 AND (NOT IR1) AND (NOT IR0) AND (NOT Q1) AND Q0);

--MSC1 = (/IR2*IR1*/IR0*/Q1*Q0) + (/IR2*IR1*IR0*/Q1*Q0)

   MSC1 <= ((NOT IR2) AND IR1 AND (NOT IR0) AND (NOT Q1) AND Q0)
        OR ((NOT IR2) AND IR1 AND IR0 AND (NOT Q1) AND Q0);

--MSC0 = (/IR2*IR1*IR0*/Q1*Q0) + (IR2*/IR1*/IR0*/Q1*Q0)

   MSC0 <= ((NOT IR2) AND IR1 AND IR0 AND (NOT Q1) AND Q0)
        OR (IR2 AND (NOT IR1) AND (NOT IR0) AND (NOT Q1) AND Q0);

--IRLD = (/Q1*/Q0)

   IRLD <= ((NOT Q1) AND (NOT Q0));

--PCINC = (Q1+Q0) * (/Q1+/Q0)

   PCINC <= (Q1 OR Q0)
        AND ((NOT Q1) OR (NOT Q0));

--PCLD = (Q1*Q0)

   PCLD <= (Q1 AND Q0);

end logic;
```

Schematic:



Simulation:



The simulation increments in the desired address order and also follows the machine code as well as loads and does the desired operations to between reg A and B and the inputs.