**REQUIREMENTS NOT MET**

N/A

**VIDEO FILE LINK**

https://youtu.be/xayLNdAygm0

**PROBLEMS ENCOUNTERED**

Loading REGA with a desired value in part 1 proved to be very difficult due to timing of the CPU. I solved this problem to be able to get a functional unit when I moved the IR\_LD output being true in my ASM from state 0 to state 1.

**FUTURE WORK/APPLICATIONS**

This lab covers very essential components necessary to build any type of CPUs. There is the development of state machines for use as controllers for a RALU as well as instruction register and timing of instructions that are crucial to having a proper CPU (over just say an RALU). The elements of making the elementary CPU will translate in further understanding of more complicated CPU processes like advanced pipeline and staging, as well as practical use in development of simple CPUs for a range of possible devices. With the incorporation of making our own programs, we also get introduced to the beginnings of assembly language.

**PRE-LAB QUESTIONS OR EXERCISES**

Part 1:

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

**The instruction register provides a way to control the flow of instructions to the CPU in order to avoid conflict in possible access time errors for MUXA, MUXB, and their corresponding registers. The IR does this by essentially stretching out (doubling) the CLK time.**

1. 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?

**You can store this value in a memory device such as a ROM.**

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

**To add more instructions, the size of the IR register would need to be increased to hold more possible combinations of instruction values. For 8 instruction, 3 D-Flip Flops would be used instead of the 2 that would be used for 4 instructions.**

Part 2:

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

**The extra states are used to specify and load what inputs to go into A for the LDAA command and how many instructions to jump in the JMP command.**

1. What do you need to do to the address lines to get your program to start at address $2C50 (instead of $6B70)?

**To start at $2C50 instead of $6B70, each of the addresses would have to be decremented by the difference between these two starting values. As for changing the values in an actual schematic, the [14..4] bits would have to be set to different HI and LO values than for $6B70.**

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

Part 1:

1. Controller ASM Flowchart:

A diagram of a flowchart

Description automatically generated

1. Next-State Truth Table:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| IR1 | IR0 | Q | Q+ | MSA1 | MSA0 | MSB1 | MSB0 | MSC2 | MSC1 | MSC0 | IR.LD |
| X | X | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |

1. VHDL for Combinational Portion of Controller:

A screenshot of a computer program

Description automatically generated

Quartus BDF Schematic:

A white paper with black dots and a white background

Description automatically generatedA close up of a diagram

Description automatically generated

A diagram of a computer

Description automatically generated4. Quartus Simulation:

A screenshot of a computer

Description automatically generated

\*Note: Q signal is current state. Last value (cutoff) for REGA and output signal is 0100

Column Number:

**0**: IR\_LD is false at this stage and thus no instructions inputs are being recorded so there is a delay until the next rising CLK edge. *MSA, 01 and MSB, 10 are set at this point to retain their values, which will always be the for the case for their respective registers unless otherwise commanded by a instruction from IR*.

**1**: Once IR\_LD becomes high (which corresponds to the Q, or state output) the input value is decoded to make IR 01. This is the TAB function which loads REGA so MSA changes to 00. Within the next CLK cycle the input changes to 1001 which is then loaded into REGA. IR\_LD changes to low after being high for exactly one cycle.

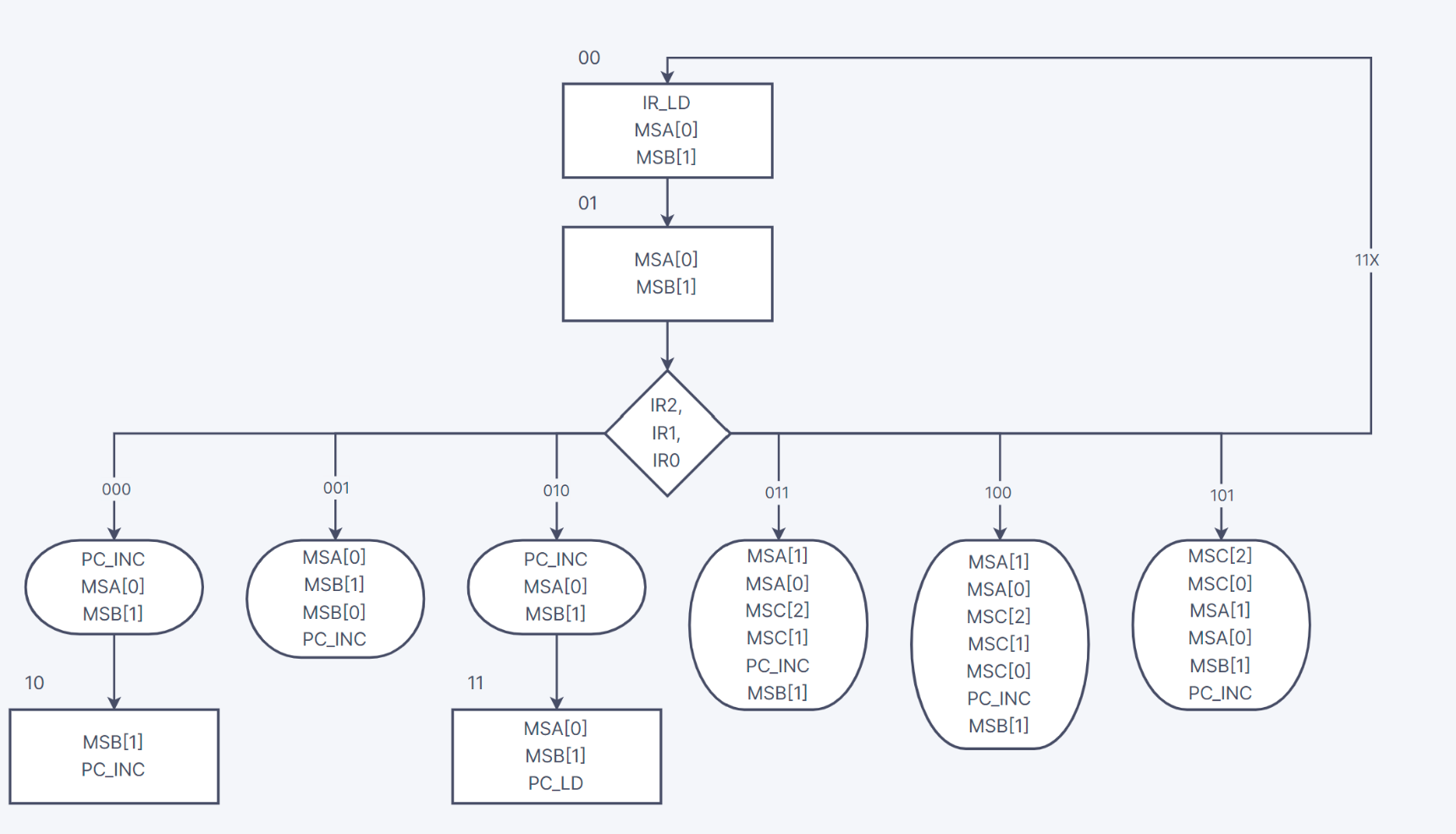
**2**: (IR\_LD changes to high from low). The input is changed to 0000 which is decoded to make IR 00. (The input stays the same once IR is loaded since it is no longer needed). IR 00 is the LDAA #In function which causes MSB to change to 11 to read the output values. The output values become 1001, the contents of REGA, since MSC is 000 due to IR being 00. Thus, REGB becomes 1001. (IR\_LD changes to low)

**3**: (IR\_LD changes to high). The input changes to 0010 which makes IR 10. This triggers the SUM function, causing REGA, 1001, to be added with REGB, also 1001. This output becomes 0010, as MSC is set to 101, and that is then sent to REGA since MSA also changed to 11. The Cout flag also becomes 1. (IR\_LD changes to low)

**4**: (IR\_LD changes to high). The input is 0011 which is decoded to make IR 11. This is the SLA function so MSA changes to 11 and MSC changes to 110. This makes the output 0100, or the contents of REGA shifted to the left one bit. This causes REGA to change to 0100. (The Cout flag is changed back to low).

Part 2:

1. ASM Controller Flowchart:



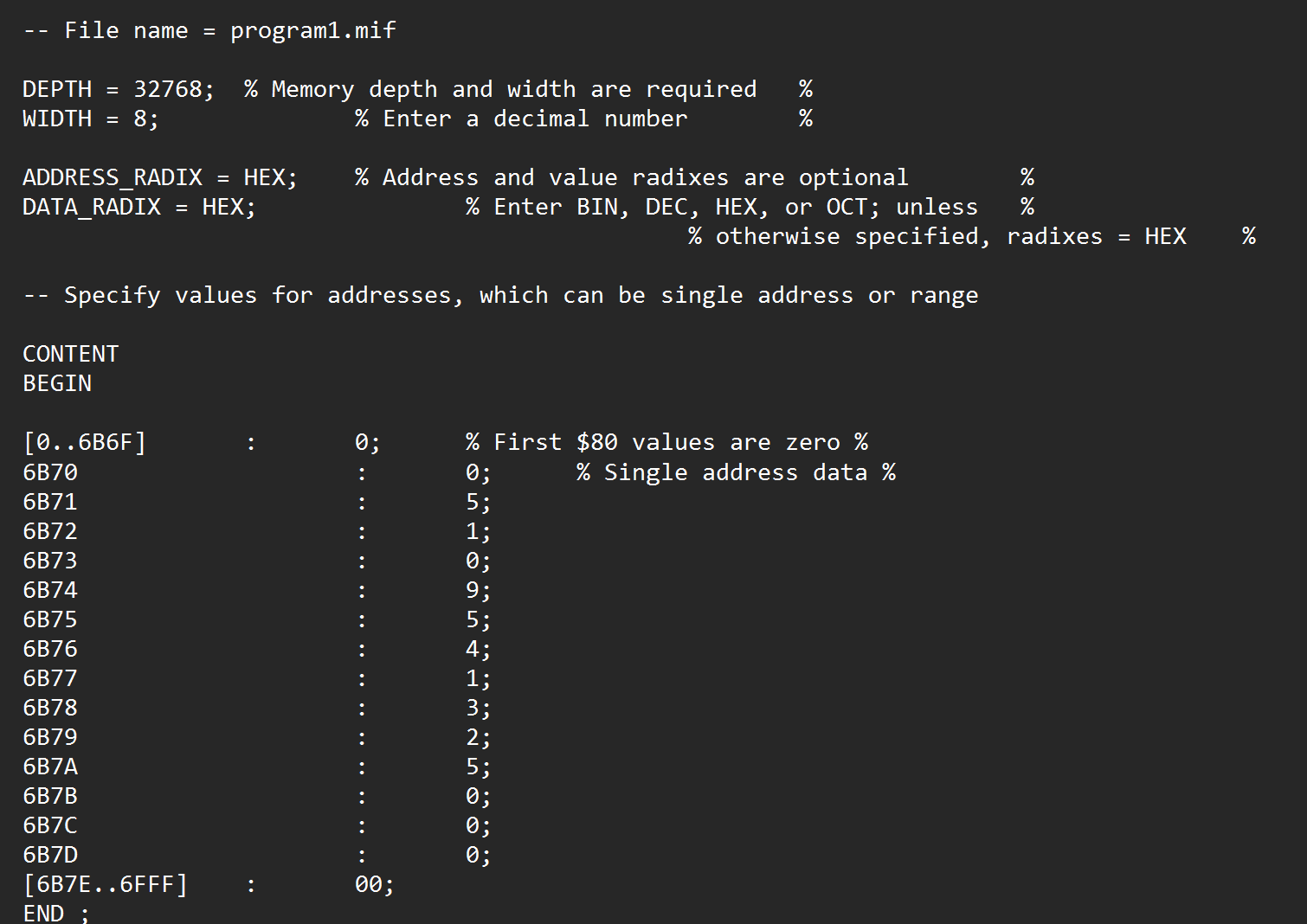
1. Next-State Truth Table:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| IR2 | IR1 | IRO | Q1 | Q0 | Q1+ | Q0+ | MSA1 | MSA0 | MSB1 | MSB0 | MSC2 | MSC1 | MSC0 | IR\_LD | PC\_INC | PC\_LD |
| X | X | X | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | X | 0 | 1 | 0 | 0 | X | X | X | X | X | X | X | X | X | X |
| X | X | X | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| X | X | X | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

1. Program Table:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Addr** |  | **Match Codes** | **A** | **B** | **A** | **B** | **A** | **B** | **A** | **B** |
| $**6B7**0 | LDAA #$5 | 0000 | X | X | X | X | X | X | X | X |
| $6B71 |  | 0101 | $5 | X | X | X | X | X | X | X |
| $6B72 | TAB | 0001 | $5 | $5 | X | X | X | X | X | X |
| $6B73 | LDAA #$9 | 0000 | $5 | $5 | X | X | X | X | X | X |
| $6B74 |  | 1001 | $9 | $5 | X | X | X | X | X | X |
| $6B75 | ABA | 0101 | $E | $5 | $5 | $7 | $6 | $2 | $9 | $3 |
| $6B76 | SAR | 0100 | $7 | $5 | $2 | $7 | $3 | $2 | $4 | $3 |
| $6B77 | TAB | 0001 | $7 | $7 | $2 | $2 | $3 | $3 | $4 | $4 |
| $6B78 | SAL | 0011 | $E | $7 | $4 | $2 | $6 | $3 | $8 | $4 |
| $6B79 | JMP 5 | 0010 | $E | $7 | $4 | $2 | $6 | $3 | $8 | $4 |
| $6B7A |  | 0101 | $E | $7 | $4 | $2 | $6 | $3 | $8 | $4 |
| $6B7B | LDAA #$D | 0000 | X | X | X | X | X | X | X | X |
| $6B7C |  | 1101 | X | X | X | X | X | X | X | X |
| $6B7D | ABA | 0101 | X | X | X | X | X | X | X | X |

MIF File:

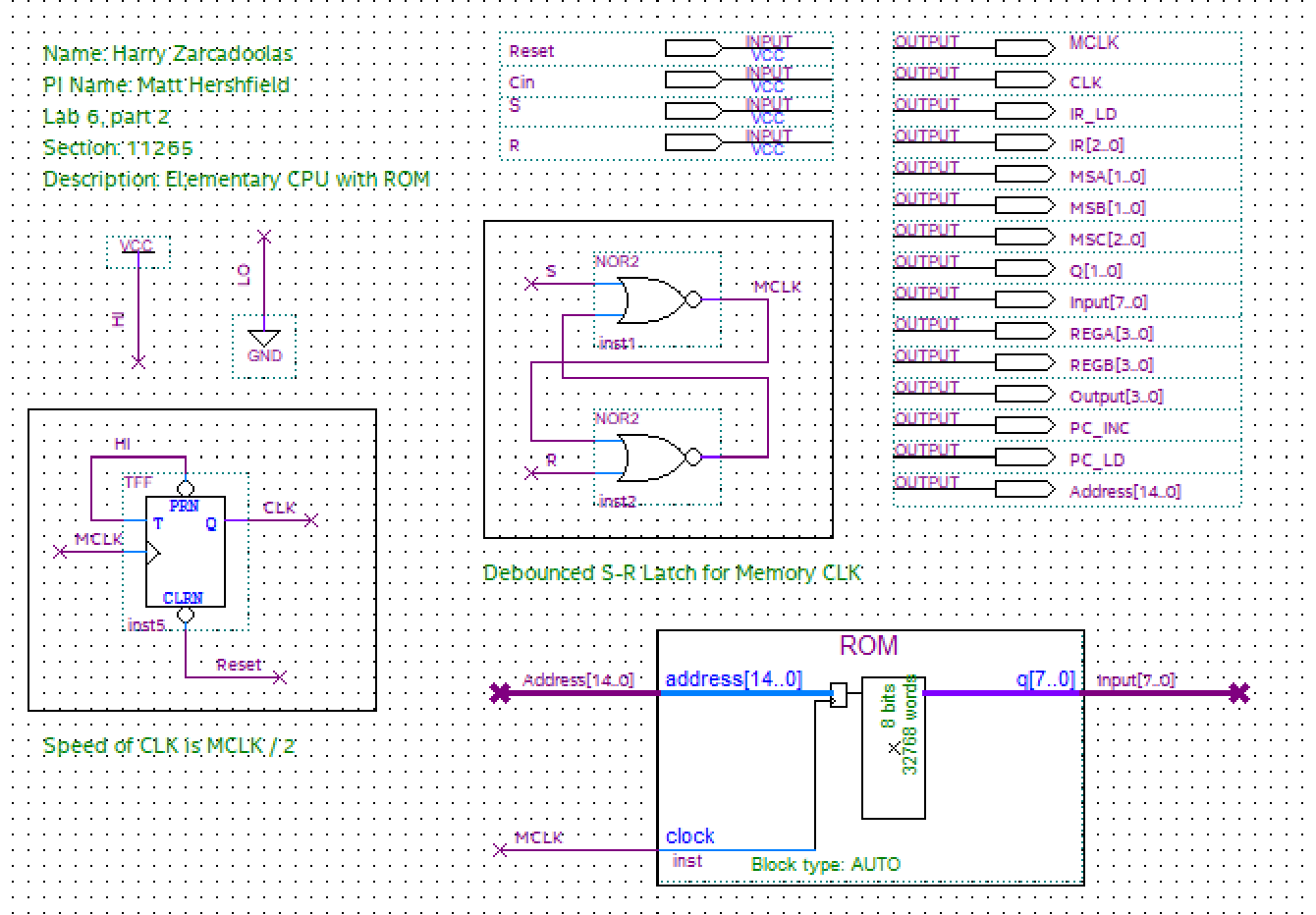


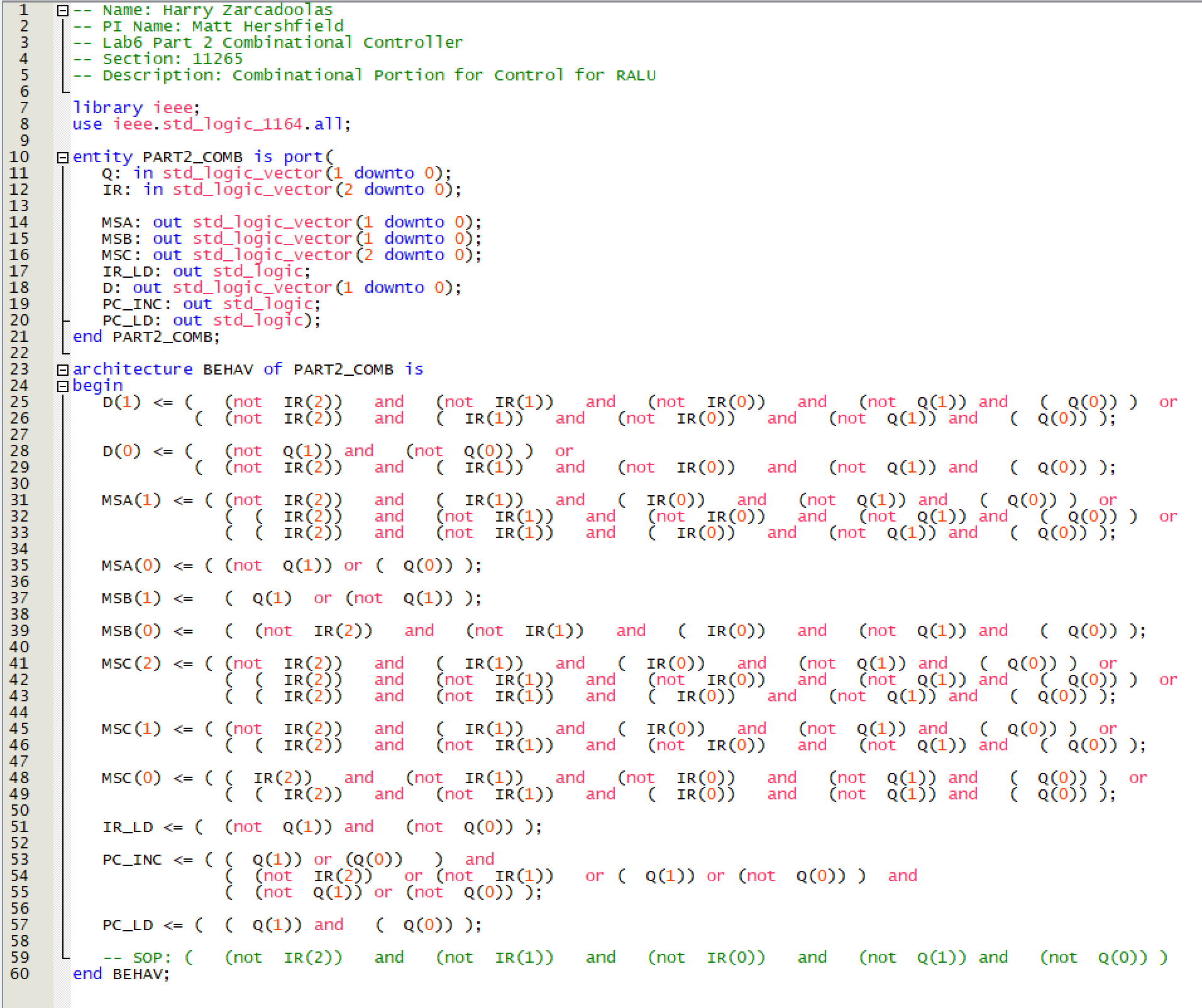
Simulation for functioning ROM:

A close-up of a graph

Description automatically generated

LAB6\_Part2 Design:



VHDL for Combinational Portion:

LAB6\_Part2\_VHDL\_ROM Design: A screenshot of a computer screen

Description automatically generated

A diagram of a circuit

Description automatically generatedA screenshot of a computer program

Description automatically generated

Note: Same VHDL file Combinational Portion

Functional Simulation (same results for both LAB6\_Part2 & LAB6\_VHDL\_ROM\_Part2):

A screenshot of a graph

Description automatically generated

LED Legend, for REGA and REGB that utilize 7 segment decoders:

