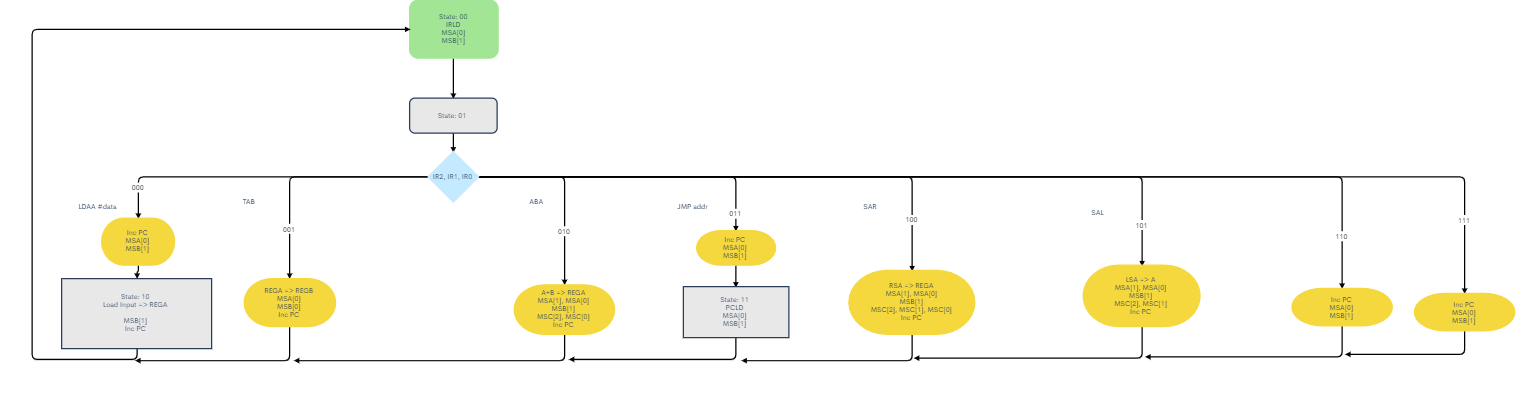
**CPU Design with ROM – Arham Khan**

This lab teaches state machine creation as a part of an overall system involving memory and has also given me a strong understanding of how manufactures actually define an architecture and instruction set for their processor and has equipped me to expand on this instruction set in the future. In addition, the lab teaches how to utilize memory such as a hardcoded ROM to execute a pre-assembled program and gives an intuition for how a processor might execute instructions that are not hardcoded. For instance if we wanted to processor to execute instructions in some arbitrary address space, we could (instead of hardwiring $73A into the first 3 bytes of the address space) take as input some address location and load the program counter with some predetermined value so that we could then load instructions from any address space. This would be much simpler of course if there was some limit to a program’s address space which in most real world operating systems holds true. Specifically this lab teaches instruction set design, reading and executing programs from memory, and how to utilize a controller alongside a clock to execute instructions using many different components for both execution and memory purposes.

Below is an implemented CPU with an associated ALU which it can control with certain control words, these words are generated in response to input from the ROM which fills the instruction register of the CPU.



In the ASM above, state 0 represents the clock cycle where we read input from the ROM into the input bus and load the instruction register. The next clock cycle, we perform a certain operation depending on the instruction we have read. Certain instructions such as IR=000 require a second clock cycle to execute, in this case because we want to read from an arbitrary memory location, state 10 is added. In IR=011 we jump addresses and so need an additional clock cycle in order to load the program counter with the jump address.

MSA, MSB, and MSC are all bit strings which control the operation of the ALU. The ALU design is also uploaded to my GitHub in a separate folder.

NSTT:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Q1** | **Q0** | **IR2** | **IR1** | **IR0** | **MSA1+** | **MSA0+** | **MSB1+** | **MSB0+** | **MSC2+** | **MSC1+** | **MSC0+** | **PCInc** | **PCLD** | **IRLD** | **Q1+** | **Q0+** |
| 0 | 0 | \* | \* | \* | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | \* | \* | \* | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | \* | \* | \* | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

MSA[1]+ = (/Q1 \* Q0 \* /IR2 \* IR1 \* /IR0) + (/Q1 \* Q0 \* IR2 \* /IR1)

MSA[0]+ = (/Q1 + Q0)

MSB[1]+ = (Q1 + /Q0 + IR2 + IR1 + /IR0)

MSB[0]+ = (/Q1 \* Q0 \* /IR2 \* /IR1 \* IR0)

MSC[2]+ = (/Q1 \* Q0 \* /IR2 \* IR1 \* /IR0) + (/Q1 \* Q0 \* IR2 \* /IR1)

MSC[1]+ = (/Q1 \* Q0 \* IR2 \* /IR1)

MSC[0]+ = (/Q1 \* Q0 \* /IR2 \* IR1 \* /IR0) + (/Q1 \* Q0 \* IR2 \* /IR1 \* /IR0)

PCInc = (Q1 + Q0) \* (/Q1 + /Q0)

PCLD = (Q1 \* Q0)

IRLD = (/Q1 \* /Q0)

Q1+ = (/Q1 \* Q0 \* /IR2 \* /IR1 \* /IR0) + (/Q1 \* Q0 \* /IR2 \* IR1 \* IR0)

Q0+ = (/Q1 \* /Q0) + (/Q1 \* Q0 \* /IR2 \* IR1 \* IR0)

PC Counter NSTT:

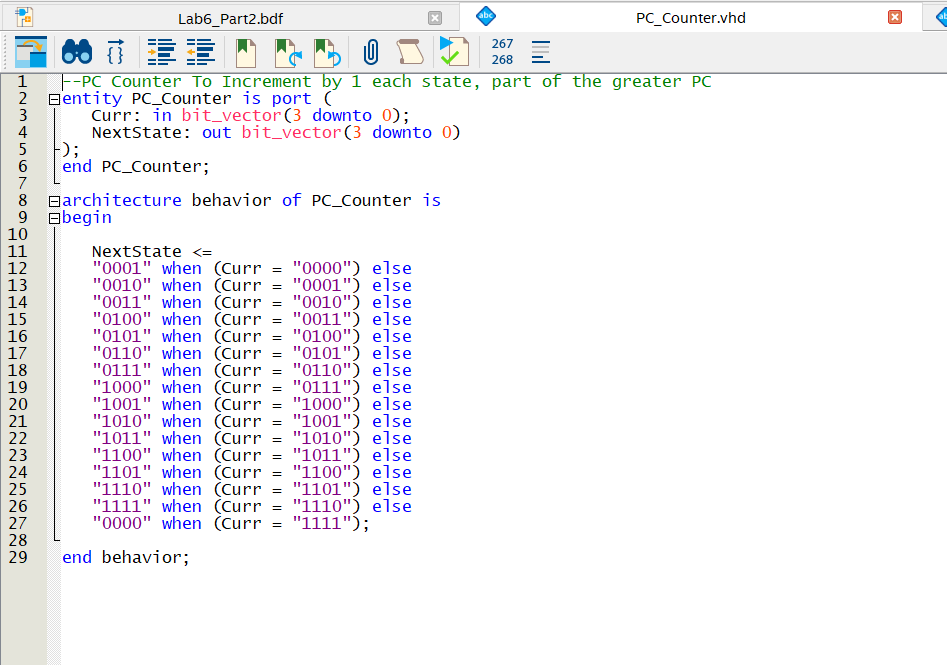
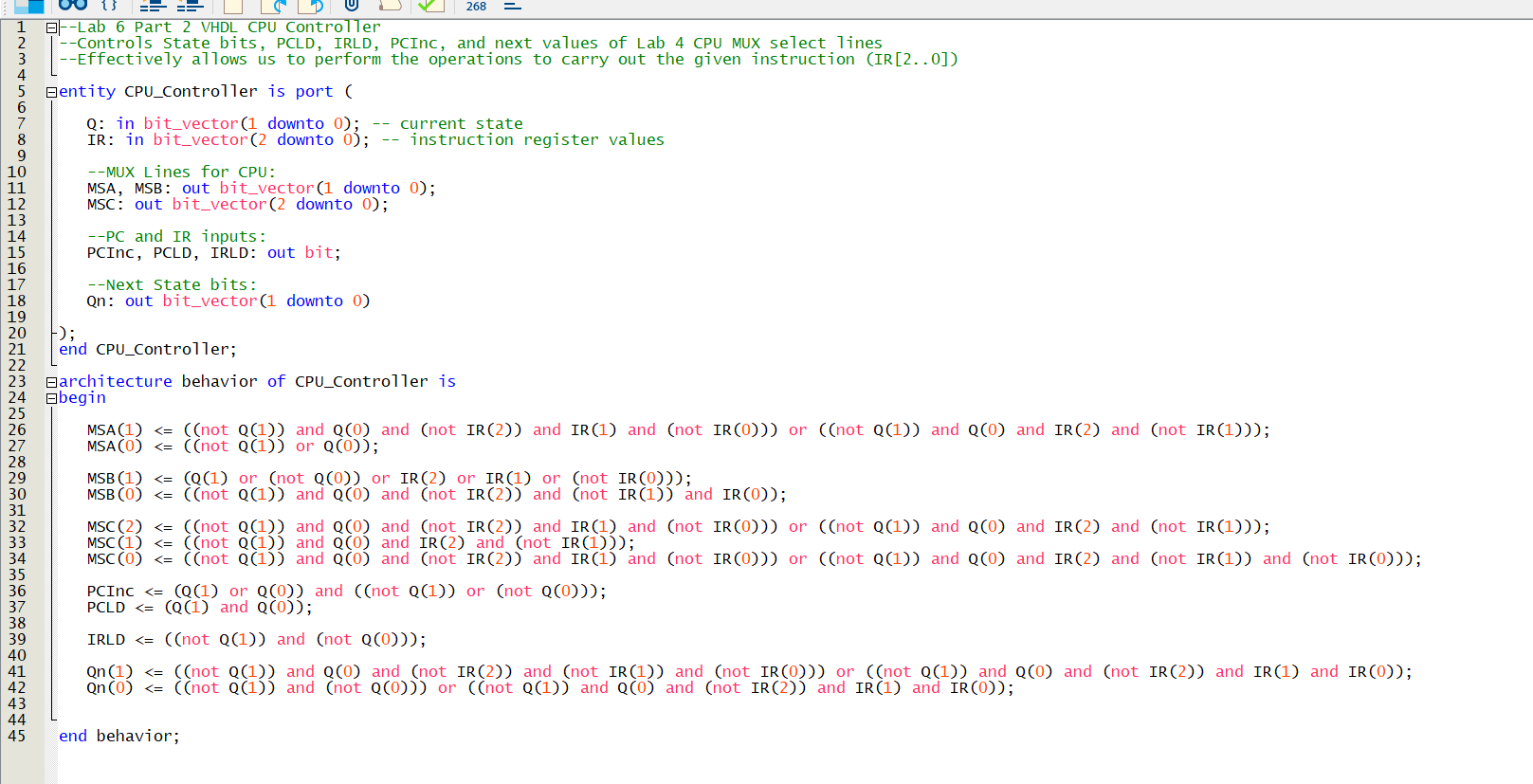
(Assumes PCInc True as this will be implemented with a MUX to allow/disallow incrementing)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **PC3** | **PC2** | **PC1** | **PC0** | **PC3+** | **PC2+** | **PC1+** | **PC0+** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

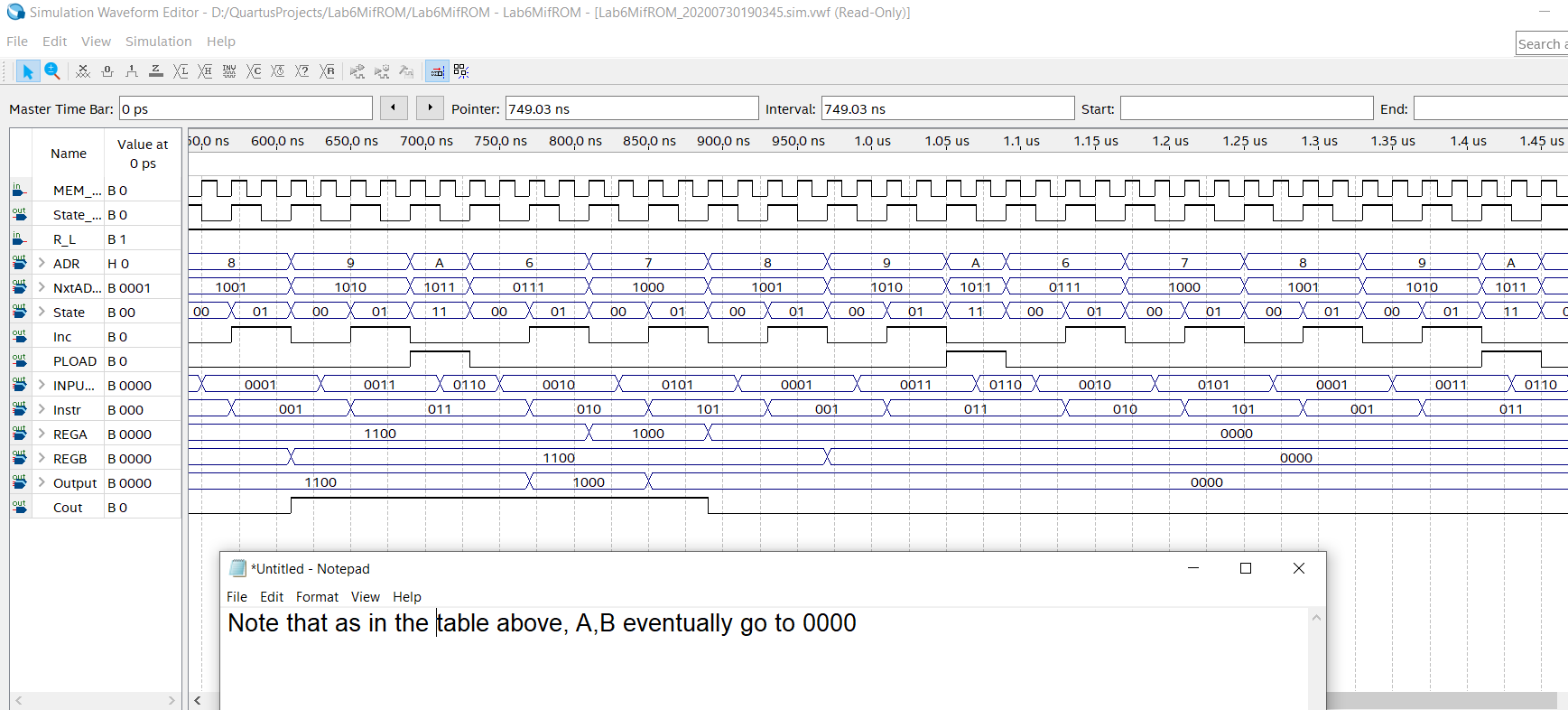
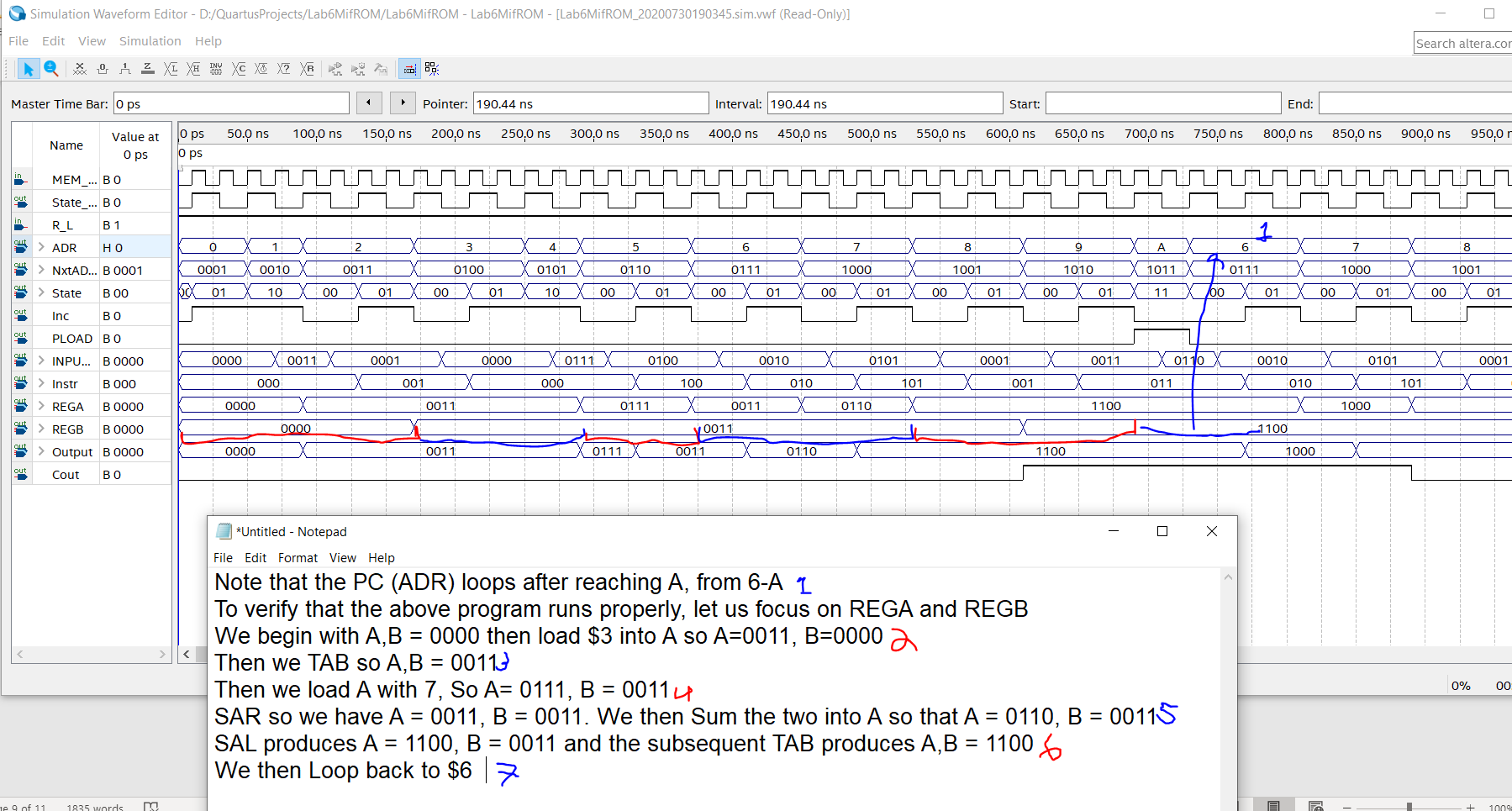
ROM Example Program:

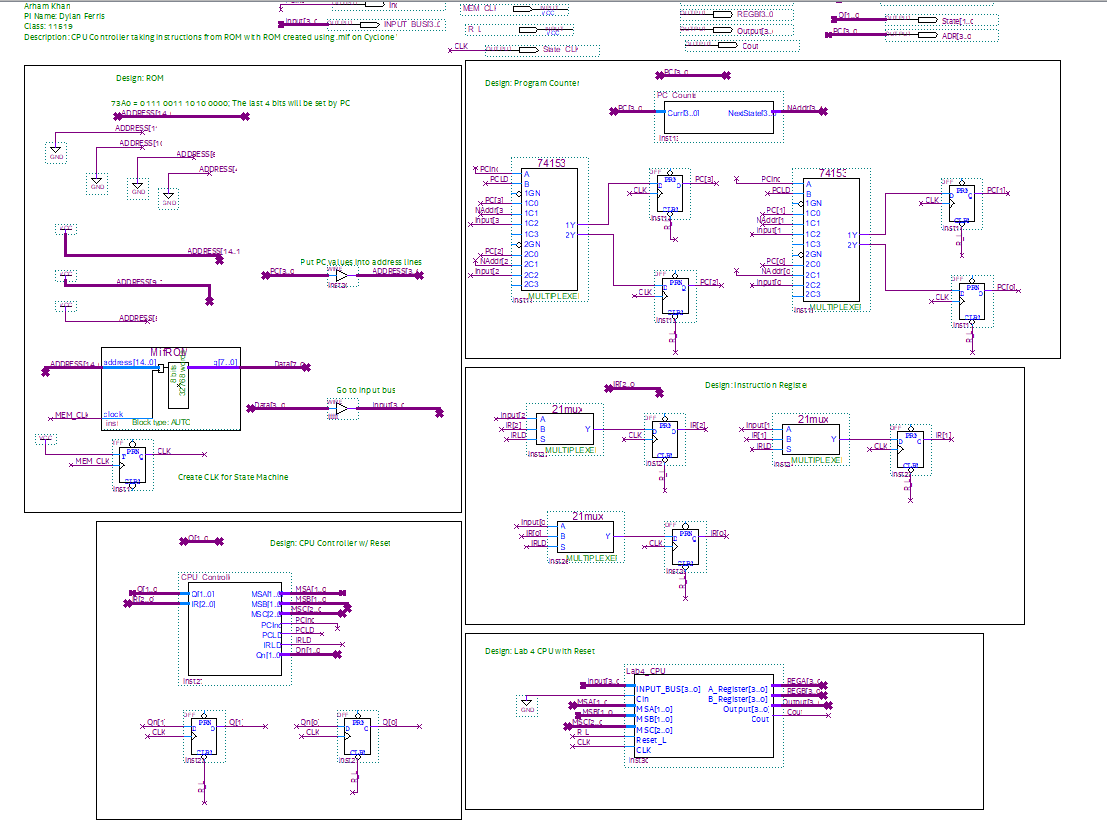
(IR is 3 bits, but Input Bus is 4 bits, Mach Codes shows the input bus value)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Addr** | **Instr** | **Mach Codes** | **A** | **B** | **A** | **B** | **A** | **B** | **A** | **B** |
| $73A0 | LDAA #3 | 0000 | 0000 | 0000 |  |  |  |  |  |  |
| $73A1 |  | 0011 | 0011 | 0000 |  |  |  |  |  |  |
| $73A2 | TAB | 0001 | 0011 | 0011 |  |  |  |  |  |  |
| $73A3 | LDAA #7 | 0000 | 0011 | 0011 |  |  |  |  |  |  |
| $73A4 |  | 0111 | 0111 | 0011 |  |  |  |  |  |  |
| $73A5 | SAR | 0100 | 0011 | 0011 |  |  |  |  |  |  |
| $73A6 | ABA | 0010 | 0110 | 0011 | 1000 | 1100 | 0000 | 0000 | 0000 | 0000 |
| $73A7 | SAL | 0101 | 1100 | 0011 | 0000 | 1100 | 0000 | 0000 | 0000 | 0000 |
| $73A8 | TAB | 0001 | 1100 | 1100 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| $73A9 | JMP 6 | 0011 | 1100 | 1100 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| $73AA |  | 0110 | 1100 | 1100 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |



Mif file as ROM:





VHDL ROM Simulation:

