**Lab 4: Basic Computer Organization**

**Fall 2018**

**School of Electrical Engineering and Computer Science**

**University of Ottawa**

**Professor: Dr. Fadi Malek**

**Group 6**

**Khang Nguyen 300007277**

**Syed Haider Rizvi 6842303**

**Experiment Date: November 12 2018**

**Submission Date: December 10 2018**

**Introduction:**

The goal of this lab is to analyse the structure of a basic computer. We are going to devise and design its control unit, and will use opcodes to write simple programs in machine code. The design must function in simulation, and also on the DE2-115 Altera development board.

**Lab Problem:**

During the lab we will be building a basic computer. The memory space of the computer is divided into two. 80-FF will be the operation code and 00-7F will be where the operands are stored. In this lab we will be working with 3 different types of reference instructions. Register reference instruction uses 1 read cycle as it simply gets the information in the given address. A memory reference instruction with direct addresses uses 3 memory read cycles, first to get the operation code, second to get the operand address, and third to retrieve the operand and to execute the instruction. A memory reference instruction with indirect addresses will take 4 memory read cycles, the reason for that being it will do the same steps except the operand address will not contain the operand but another address which then contains the information. Below is our solution to the circuit design.

Memory

memwrite = T9Y4 + T10Y6

CPU Registers

AR\_Load = T0 + T2 + T5(IR6)’ + T6(IR6)’ + T7X2

PC\_Load = T8Y5

PC\_Inc = T2(S)’ + T5(IR6)’(S)’ + S’Y6(T11 + T12)(DR0 + DR1 + DR2 + DR3 + DR4 + DR5 + DR6 + DR7)’

DR\_Load = T8(Y0 + Y1 + Y2 + Y3 + Y6)

DR\_Inc = T9Y6

IR\_Load = T3

AC\_Clear = T5X1IR0

AC\_Load = T5X1(IR1 + IR2 + IR3) + T9(Y0 + Y1 + Y2 + Y3)

AC\_Inc = T5X1IR4

OUTD\_Load = T1

CPU ALU

ALU\_Sel2 = T9(Y0 + Y3) + T5X1IR1

ALU\_Sel1 = T9Y3 + T10Y6 + T5X1(IR1 + IR2 + IR3)

ALU\_Sel0 = T9Y2 + T5X1(IR1 + IR3)

Bus

BusSel2 = T0 + T9Y4

BusSel1 = T2 + T5 + T0 + T10Y6

BusSel0 = T9Y4 + T8Y5 + T10Y6

Control Unit

SC\_Clear = T5X1 + T8Y5 + T12Y6 + T9(Y0 + Y1 + Y2 + Y3 + Y4)

Halt = T5X1IR5

**Software Problem:**

Analysis of the prelab program:

1. Load AC with value from a0
2. Compliment what's in AC
3. Store AC to a0
4. Increment a0 and if it is 0 skip next line
5. Branch to 20
6. Halt
7. Load AC with the value from the address in a1
8. Add the value from the address in a2 to AC
9. Store AC to the address in a3
10. Load AC with a1
11. Increment AC
12. Store AC into a1
13. Increment AC
14. Store AC into a2
15. Increment AC
16. Store AC into a3
17. Branch to 05

**Pseudo code:**

int x=0, y=1, z=2, counter = -10;

array a = new array[12];

a[x]=0;

a[y]=1;

while (counter != 0) {

a[z] = a[x] + a[y];

x++;

y++;

z++;

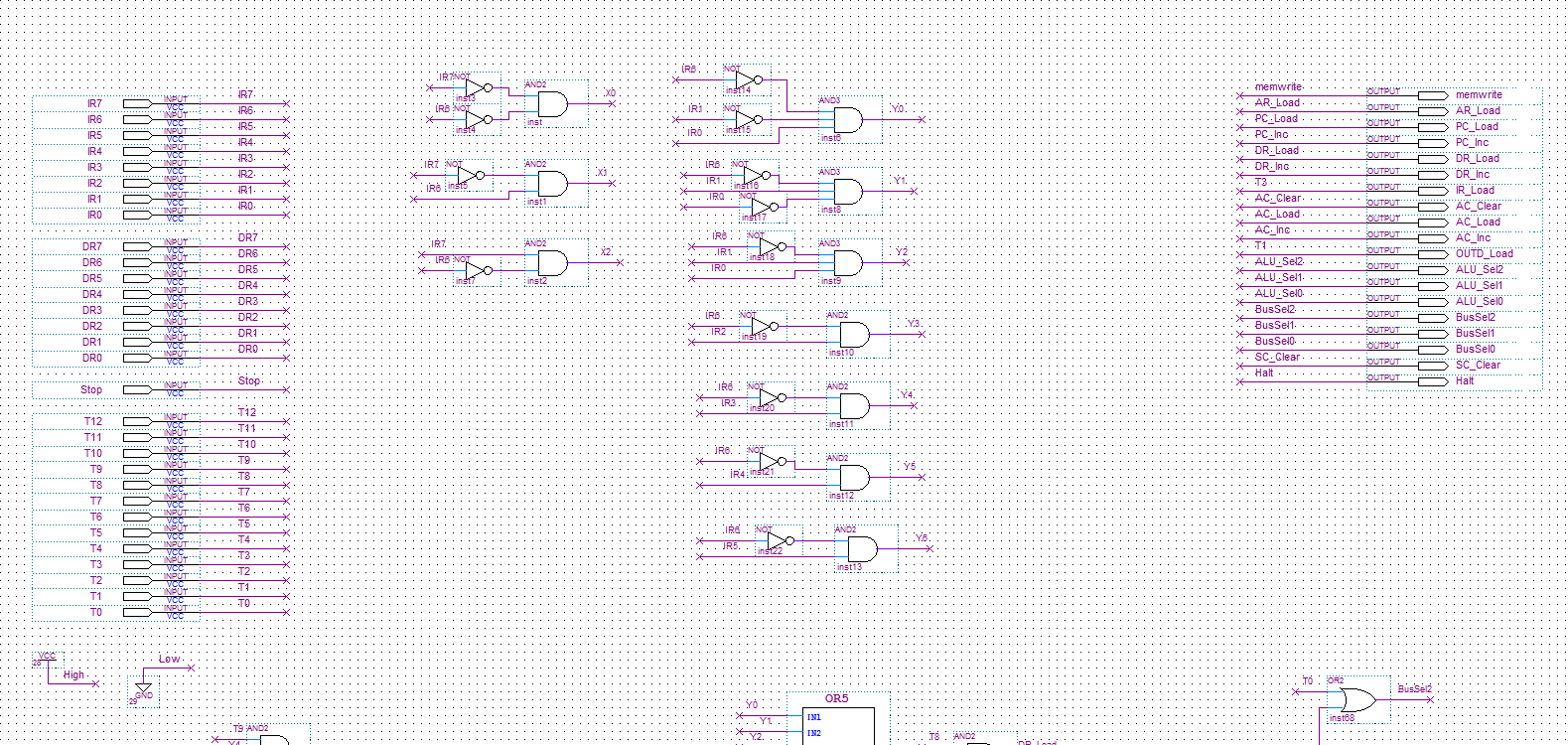
counter++;

}

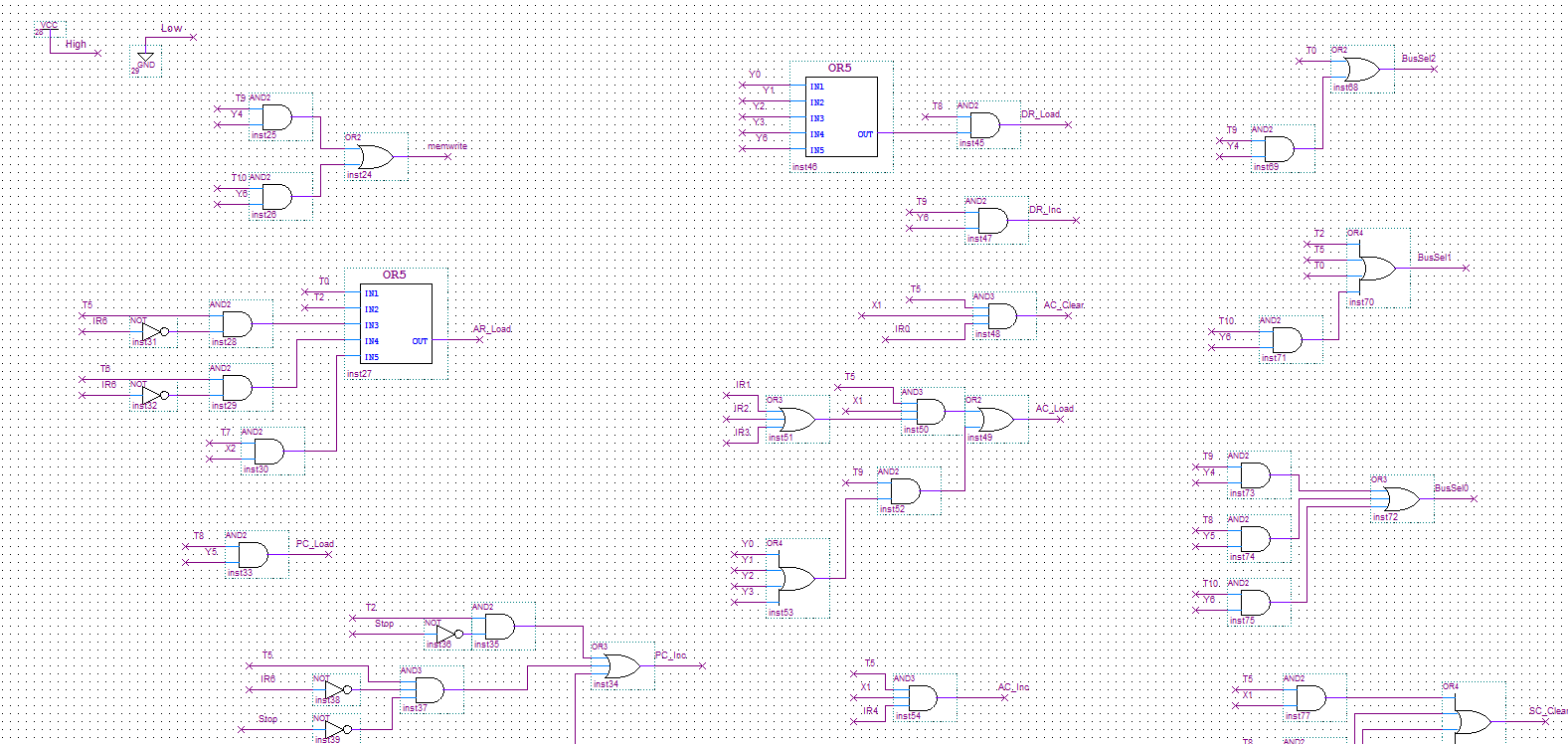
(indirect addressing is very useful to increment the actual address containing values)

**Solution:**

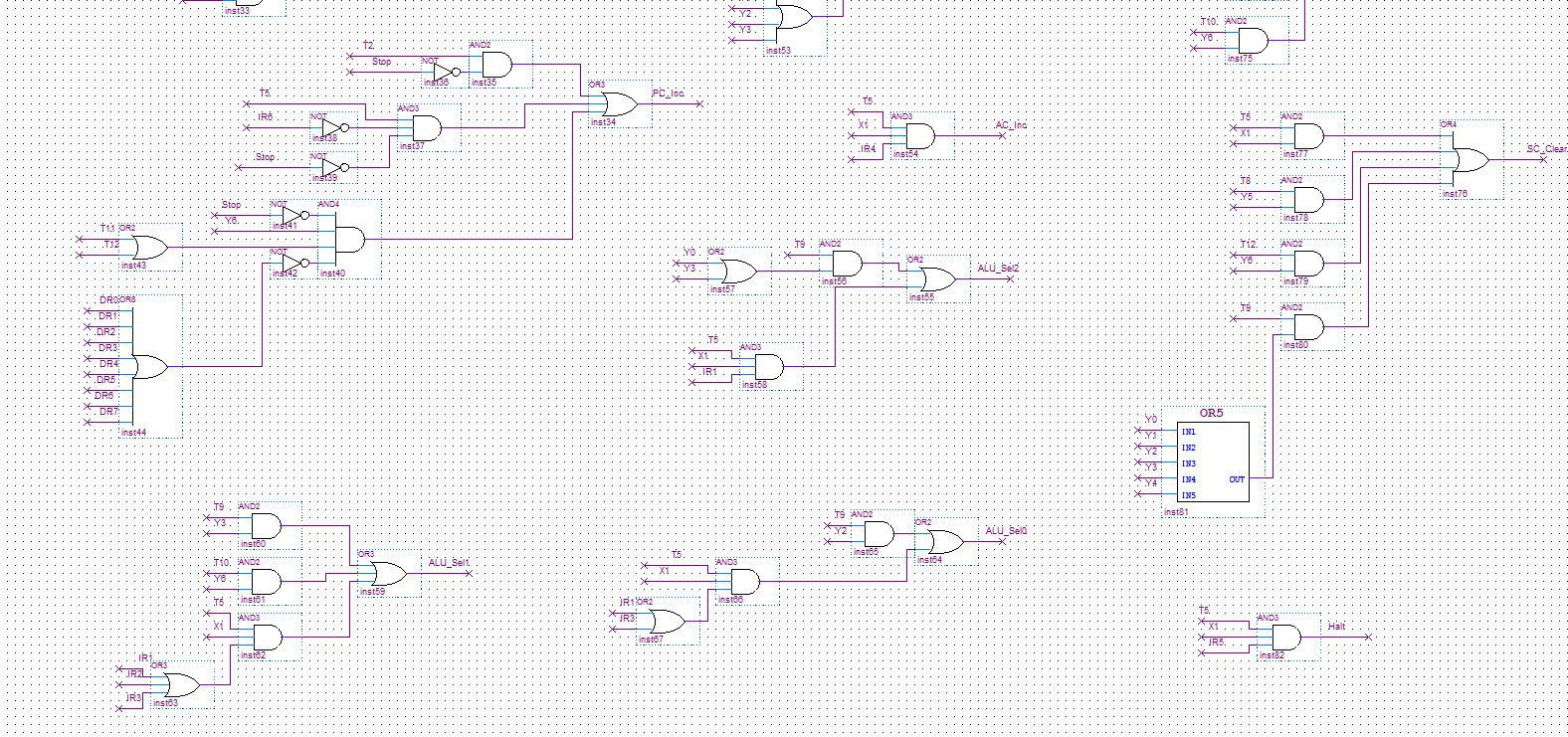
To solve the problem we first had to create a circuit for every equation that we had (see figure)



**Figure 1: Screen capture of lab3controller Part 1**

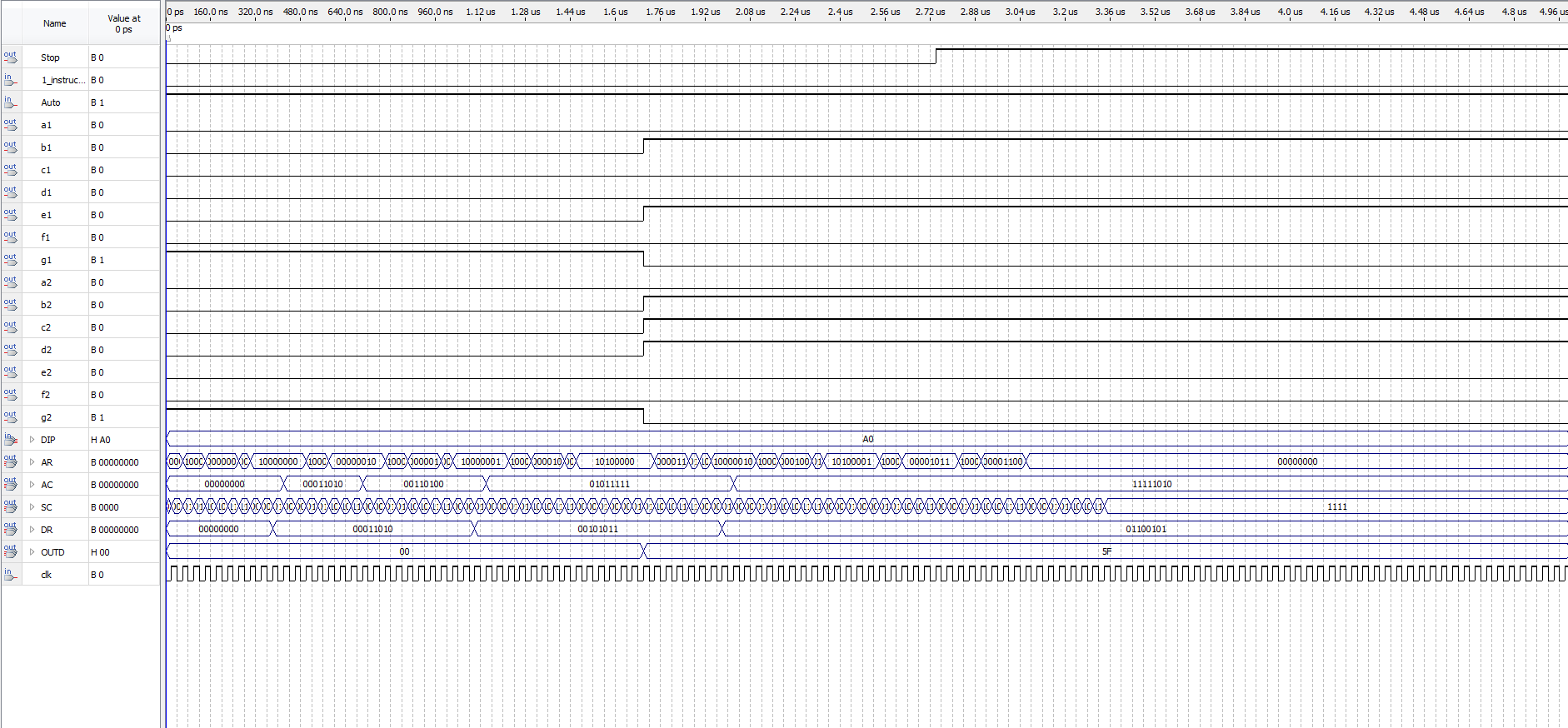


**Figure 2: Screen capture of lab3controller Part 2**

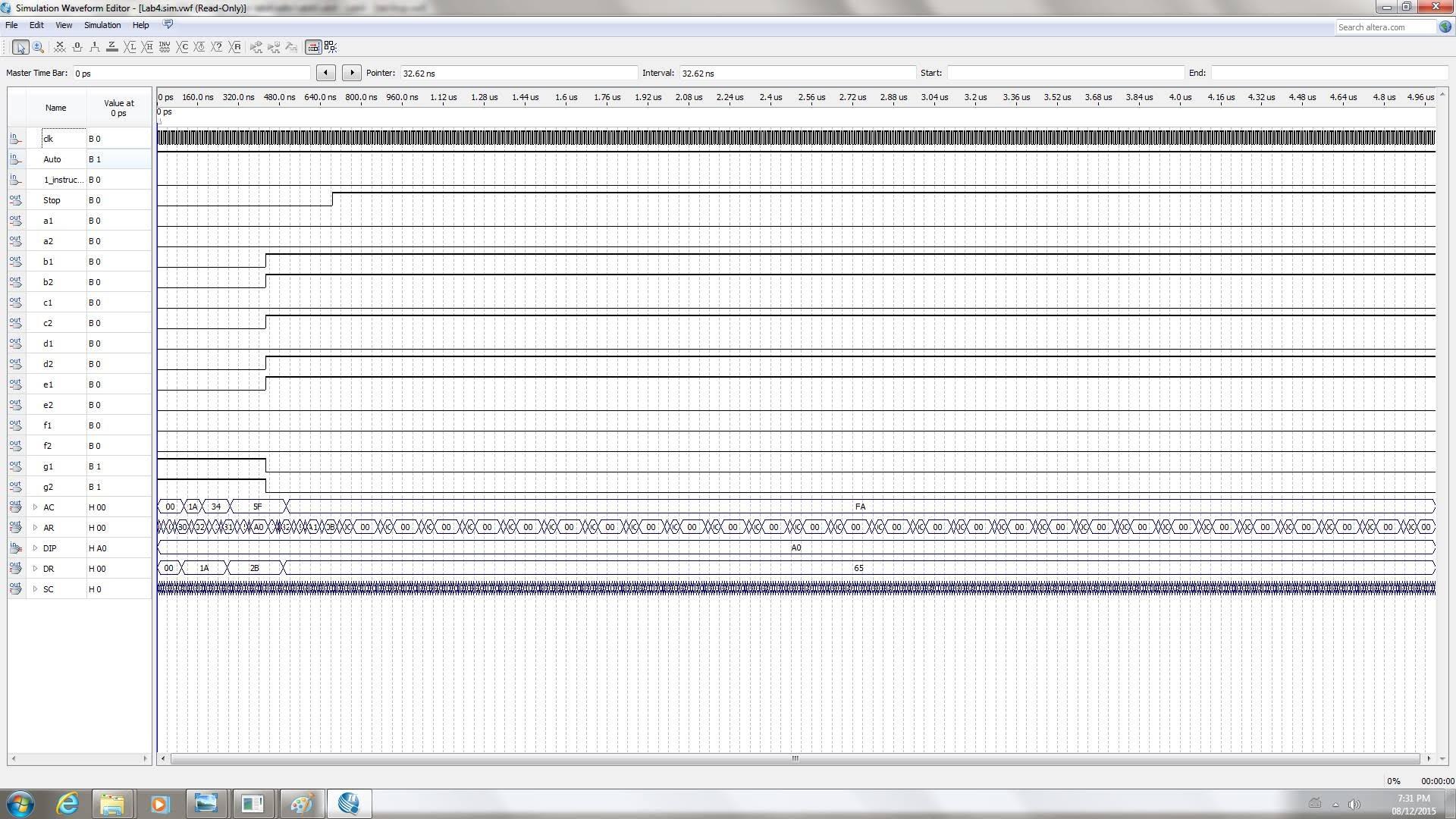


**Figure 3: Screen capture of lab3controller Part 3**

We then had to add parts to lab3top.bdf to match the pdf (see CEG2136\_17Lab4.docx page 15)



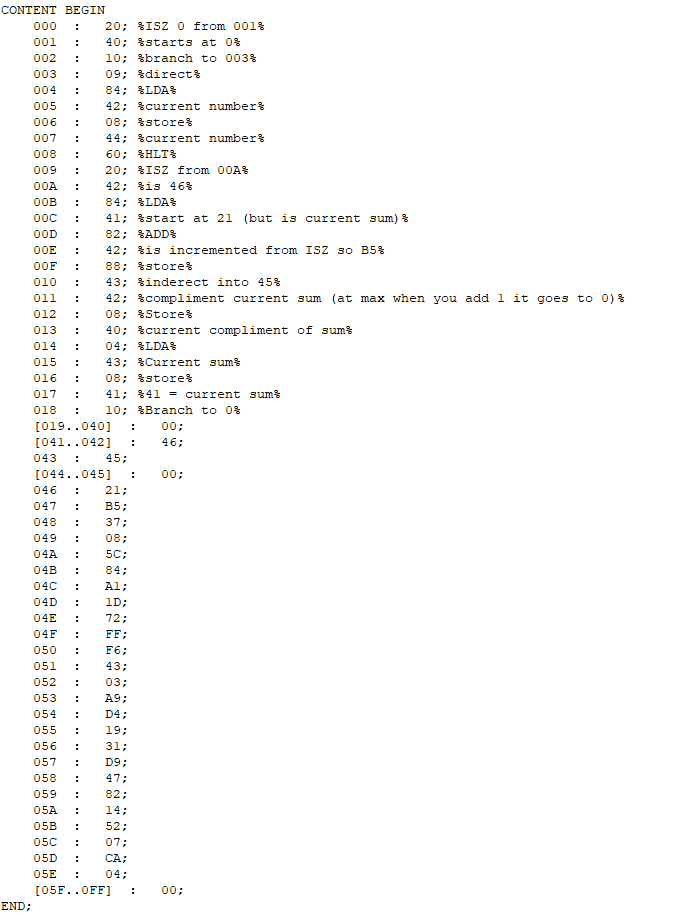
**Figure 4: Screen capture of Hardware Simulation**



**Figure 5: Screen capture of Software Simulation**

Next we had to machine code a solution to the software problem.

Start off by incrementing 40 to check if 40 (complement of sum) is FF if so it will become 0 and will store the current number (address found at 42) at address 44. If not it will increment the value at 42 (which goes to the next number to be added) and load 41 then add value at 42 to 41. It will then save 41 in the address at 43 (so 45) and will then compliment AC and store it in 40. Set 41 to current sum and restart at 0.

****

**Figure 6: Screen capture of mif File**

**Discussion of used Components:** The circuits in the lab3controller circuit were constructed using NOT, AND, OR, OR3, OR4, OR4, OR8, AND3, and AND4 gates. The inputs used were: T0, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, Stop, DR0, DR1, DR2, DR3, DR4, DR5, DR6, DR7, IR0, IR1, IR2, IR3, IR4, IR5, IR6, and IR7. The outputs used were: memwrite, AR\_Load, PC\_Load, PC\_Inc, DR\_Load, DR\_Inc, IR\_Load, AC\_Clear, AC\_Load, AC\_Inc, OUTD\_Load, ALU\_Sel2, ALU\_Sel1, ALU\_Sel0, BusSel2, BusSel1, BusSel0, SC\_Clear, and Halt. X0, X1, X3, Y0, Y1, Y2, Y3, Y4, Y5, and Y6 were used both as inputs and as outputs.

**Discussion of Tool:** The tools we used were the Quartus II software and card. During the simulation process we were able to determine if we had derived correct logic functions. In the simulation (figure 4/5). Every 1 and 0 are indicated by the bumps up and down. With this method we were able to see that our pre-lab was correct.

**Experimental verification of Operation of the Circuit:** As seen above, our theoretical data was a match to the experimental data, achieved by loading our circuit onto the Altera board and testing each combination, meaning that our design was correct and the outcome expected. Our simulation followed a path that was described in our pre-lab.

**Discussion of Challenging Problems:** A very challenging problem was the fact that we were not told in the lab manual to change the lab3top.bdf, because of that we were not getting some outputs that were necessary. After having fixed that we got another problem that we did not understand. The board was not loading our new machine code nor was the code on it working. Because of that we had to create a new project and copy everything over and then restart the pin assignments

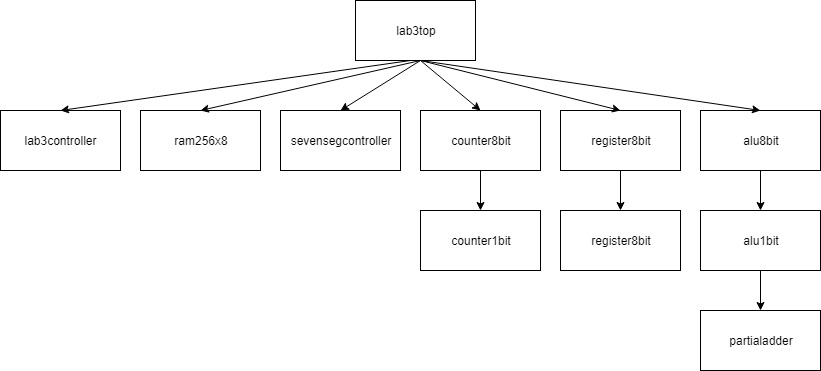
**Conclusion:** This lab did not contain many major problems for us, as the ALU functioned as we had expected it to as shown in Table 2. Initially, our board was not working for an unknown reason, with a tip from the TA we had copied everything into a new project and got it to work. This lab taught us how to devise and design a computer's control unit, and use opcodes to write simple programs in machine code.

**Appendix:**

**Pre-Lab:**

**5.1**

1.



2. Only one register will place its output on the bus at a time because an 8x1 multiplexer will be used (therefore only one output) and because the select inputs will determine the operation to be performed.

3. The reset (clear) signals are synchronous because all of the inputs are triggered/cleared by the same clock.

4. The register will be cleared to 0. This is because the value for both AND gates in the register1bit logic circuit will be 0, therefore the output of the register will be 0.

5. The address register is connected directly to the memory for efficiency purposes (easier to store and fetch).

6. They are implemented as counters because their values need to be incremented in order to access the next instruction that is stored in memory.

7. Reset has the highest priority because if Reset is 1, all of the AND gates go to 0 regardless of the other inputs. Increment has the lowest priority.

8. No, because the value is first sent from the memory to the DR before it is sent to the accumulator.

9.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S2** | **S1** | **S0** | **Operation** | **Description** |
| 0 | 0 | 0 | AC + DR | Addition |
| 0 | 0 | 1 | AC + DR’ + 1 | Subtraction |
| 0 | 1 | 0 | Ashl AC | Arithmetic shift left |
| 0 | 1 | 1 | Ashr AC | Arithmetic shift right |
| 1 | 0 | 0 | AC AND DR | Logic AND |
| 1 | 0 | 1 | AC OR DR | Logic OR |
| 1 | 1 | 0 | DR | DR Transfer |
| 1 | 1 | 1 | AC’ | AC Complement |

The first four operations are arithmetic. The last four operations are logical.

**5.2**

Memory

memwrite = T9Y4 + T10Y6

CPU Registers

AR\_Load = T0 + T2 + T5(IR6)’ + T6(IR6)’ + T7X2

PC\_Load = T8Y5

PC\_Inc = T2(S)’ + T5(IR6)’(S)’ + S’Y6(T11 + T12)(DR0 + DR1 + DR2 + DR3 + DR4 + DR5 + DR6 + DR7)’

DR\_Load = T8(Y0 + Y1 + Y2 + Y3 + Y6)

DR\_Inc = T9Y6

IR\_Load = T3

AC\_Clear = T5X1IR0

AC\_Load = T5X1(IR1 + IR2 + IR3) + T9(Y0 + Y1 + Y2 + Y3)

AC\_Inc = T5X1IR4

OUTD\_Load = T1

CPU ALU

ALU\_Sel2 = T9(Y0 + Y3) + T5X1IR1

ALU\_Sel1 = T9Y3 + T10Y6 + T5X1(IR1 + IR2 + IR3)

ALU\_Sel0 = T9Y2 + T5X1(IR1 + IR3)

Bus

BusSel2 = T0 + T9Y4

BusSel1 = T2 + T5 + T0 + T10Y6

BusSel0 = T9Y4 + T8Y5 + T10Y6

Control Unit

SC\_Clear = T5X1 + T8Y5 + T12Y6 + T9(Y0 + Y1 + Y2 + Y3 + Y4)

Halt = T5X1IR5

Software Prelab

2.

While counter > 01

M[z] <- M[x] + M[y]

M[A1] <- x + 1

M[A2] <- M[A1] + 1

M[A3] <- M[A2] + 1

Counter--

Loop

3. Fibonacci sequence

4. The utilization of indirect addresses is useful because it is possible to increment the address by using pointers.