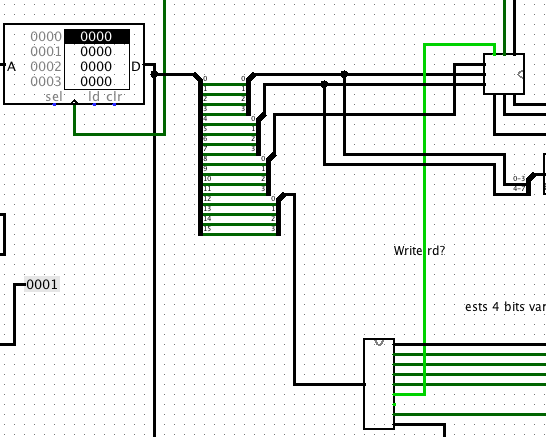
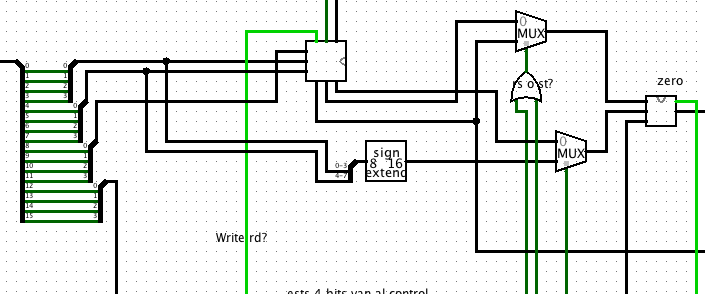
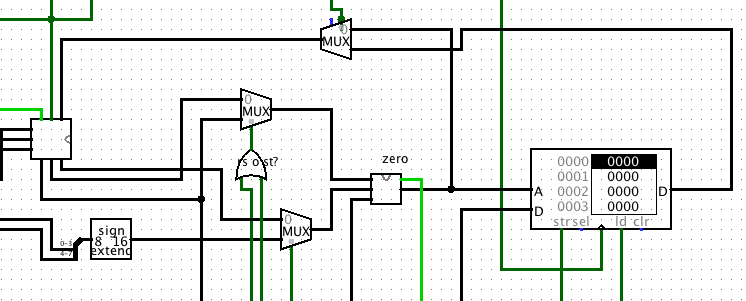
# Introduction

The purpose of this project was to implement a 16-bit CPU using Logisim. Some of the parts of this CPU are from the library of Logisim and others are from students in the class of Computer Architecture. Most of the parts I used where implemented by Christian, but still in this report I tend to explain how they work. Also I am going to explain how it was implemented.

# First the program counter, represented by a register in this CPU, sends the 16-bits to the Instruction memory. This 16-bits indicate to the Instruction memory which instruction to fetch.

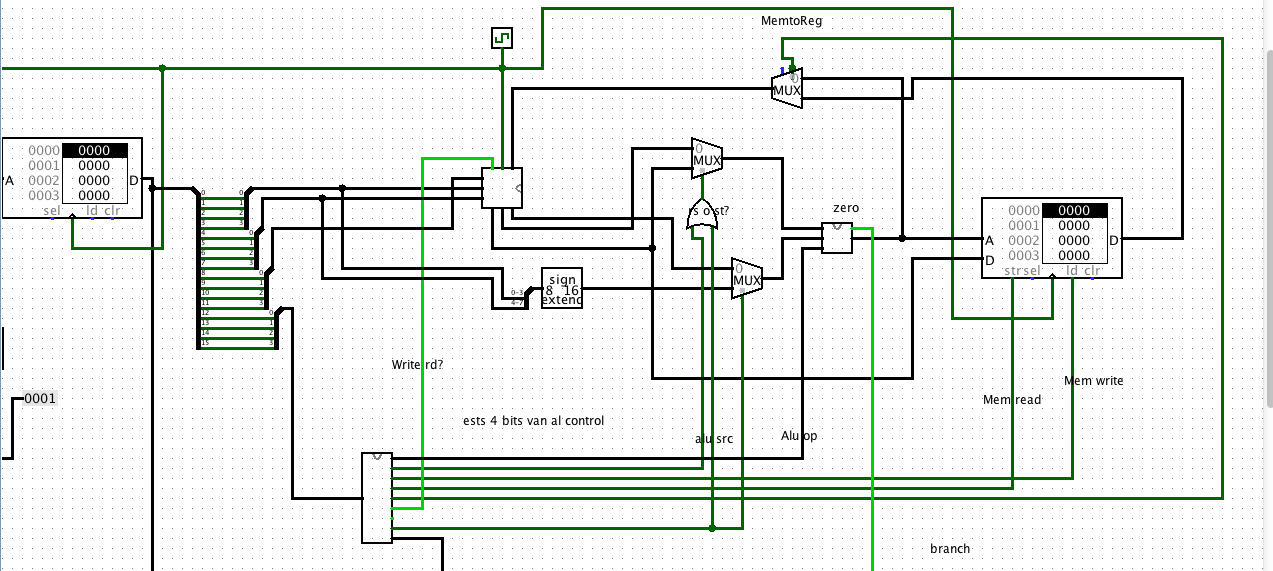
 Second the Instruction Memory outputs 16-bits representing the instruction it fetched. This 16-bit split with a splitter into 4 groups of 4-bits each. The first group represent $rs (location), the second group represent $rt (location), the third group represent $rd (location) and the last group represents the instruction to be executed. $rs (location), $rt (location) and $rt (location) along with a bit from the control (write enable) and the 4-bits of the instruction are sent to the control to se what instruction is going to be execute and how to set the other components.

 Third, the control sets the other components given the instruction. If the instruction is R-type it sends $rs (value), $rt (value) and $rt (value) to the multiplexers. If the instruction is an I-type it concatenates $rs (value) and $rt (value), then it extends it to 16-bits to form the immediate. After this the immediate is send to a multiplexer. $rt and the immediate are sent to one multiplexer and $rs and $rd are sent to another multiplexer. After all of this is set, the control given the instruction decide which data to send to the ALU, either $rs and $rd or the immediate and $rt. The OR gate connected to the multiplexer is used to send alusrcabajo or alusrcarriba. If the instruction is a branch it uses the ALU to perform the condition and a bit indicating if the conditions is valid or not is send back to multiplexers along with $rs (value), $rt (value) and $rd. Then $rt (value) and $rs (value) are concatenated, extended to 16 bits (branch) and then sent to two different multiplexers, one response to branch if not zero and the other to branch if zero. The results of these multiplexers are sent to a bigger multiplexer. Also $rs (value), $rt (value), and $rd (value) are concatenated, extended to 16-bits and sent to the big multiplexer along with the control bit, which indicate which of the three instructions, bz, bnz, j to execute or to just add a constant. Then the output of the big multiplexer goes to an adder that adds up the result of the multiplexer and the program counter.

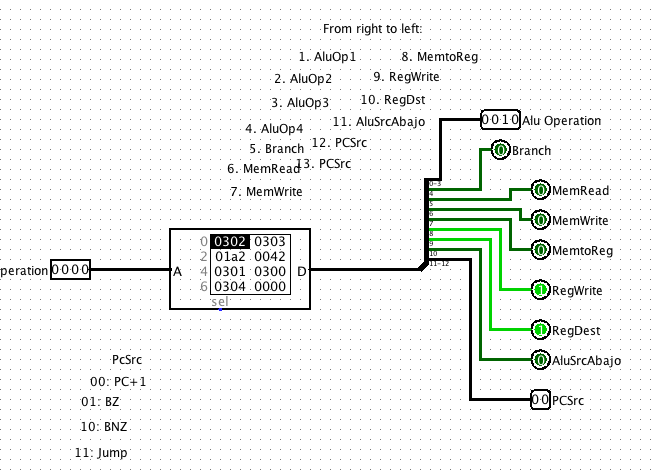


After the instructions are executed the result of the ALU goes to a multiplexer and to the Data Memory. If the instruction is a store word it writes on the memory and if it is a load word it send the value to a multiplexer that decides between the output of the memory and the output of the ALU. If it was a load word it uses the output of the data memory, else it uses the result of the ALU. The decision output is later sent to the register file to write on the registers.

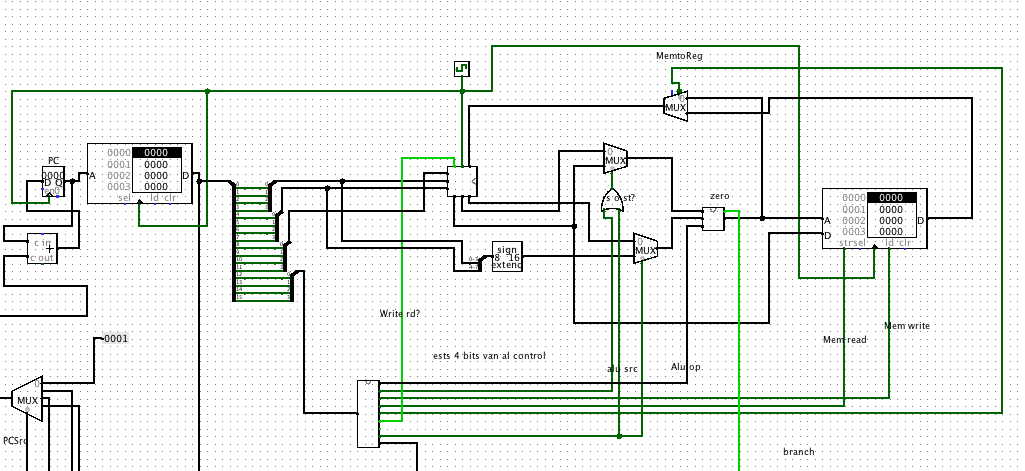
The control it’s connected to components in which decisions of which instruction to do are made like the ALU, Data memory and multiplexers.



The control was implemented with an instruction table, given 4 bits of the instructions it sets the components.



And finally with the clock we run the program



In my CPU I used Felipe Torre’s Register File and Christian Rodriguez’s ALU and Control.

Yes, this is how the CPU you created X – 2012 years ago works :).

