A blue and white logo

Description automatically generated

**MARMARA UNIVERSITY**

**FACULTY OF ENGINEERING**

**CSE3215 DIGITAL LOGIC DESIGN**

**TERM PROJECT**

**Group Members**

Mustafa Emir Uyar – 150120007

Muhammed Hayta – 150121068

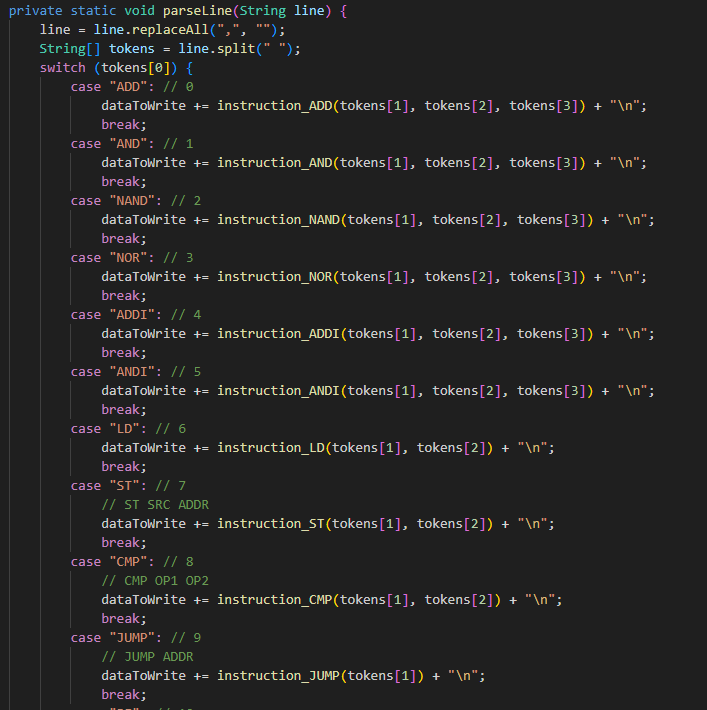
Yusuf Demir – 150120032

Burak Karayağlı - 150121824

# **Step I**

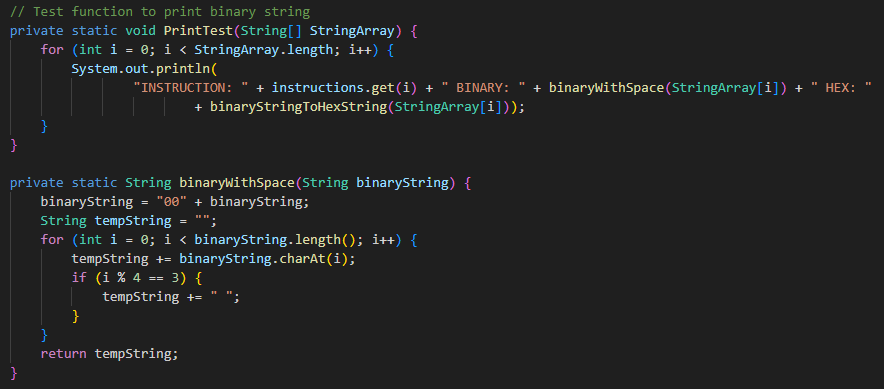
In first step of the Project, we designed the ISA (Instruction Set Architecture). ISA is part of the abstract model of a computer that defines how the CPU is controlled by the software. So, it is something like we are defining format of our instructions for CPU. In ISA we decided to order of Source Register, Destination Registers or Immediate values, Which bits should be empty etc. We reviewed Intel documentations to get ideas. We got an insight into how Intel designs its processors. We also decided to opcodes of instructions. We directly ordered opcodes according to given document. After designed the ISA we wrote Assembler in java. In java we wrote functions for reading, parsing input. Then we implemented every single instruction in our ISA.

Later we read and parsed the input. After deciding which method to call we called that method with necessary tokens.



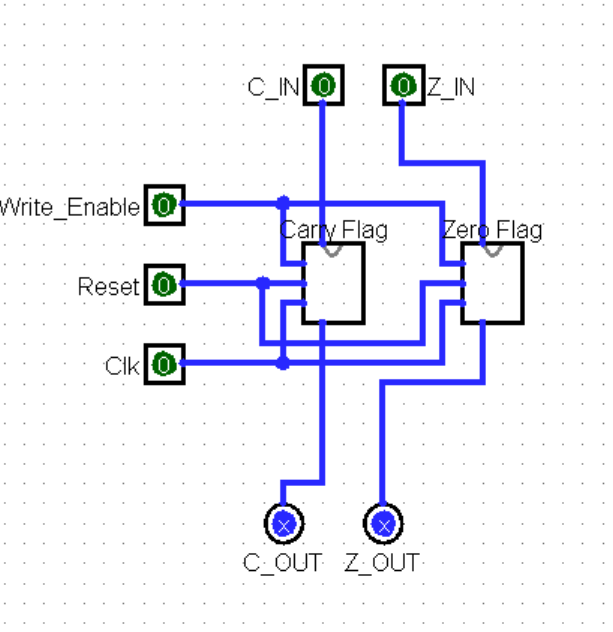
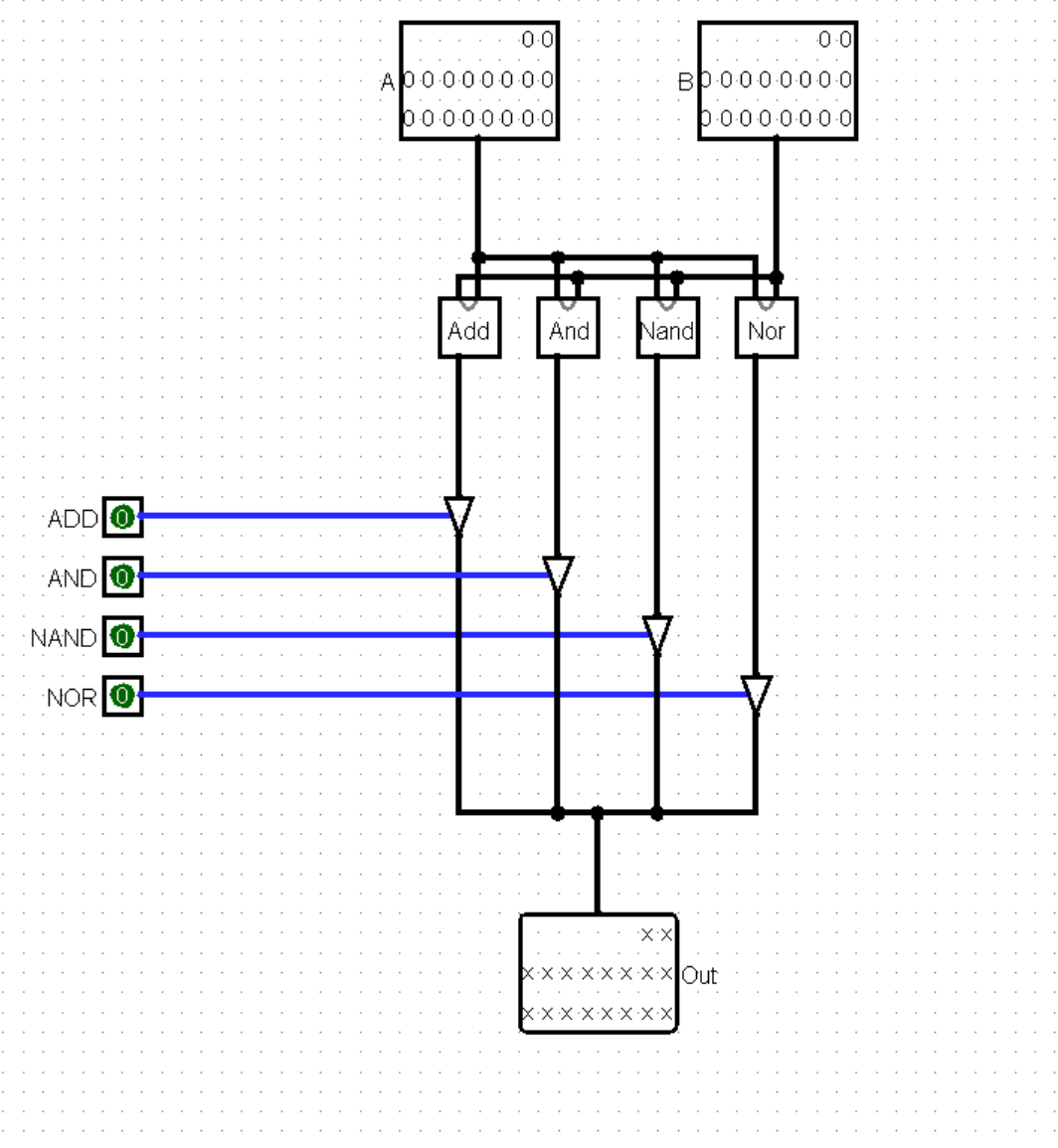
In every instruction we converted the input strings to binary. Then after the read all input we converted all binary string to hex to write output file.

We also have test function for ensure that our outputs are correct.



# **Step II – Logisim Component Design**

In this part of the project, we had to design all the components which we would use on the Datapath except for the control unit. The main components we made are: ALU, flag file, Register file, Program Counter, Comparator, 6- and 10-bit sign extender. We of course had to design several smaller circuits for our bigger components. For the design of these components, we were inspired by the components mentioned on our lectures. We made several Manuel unit tests for all these components, and we concluded them to be successful. Sometimes we encountered a bug on the Logisim program which caused the components to behave not as they should with no evidence of a false design. We could reliably fix this bug with cutting and pasting the buggy component.

A diagram of a circuit

Description automatically generated

ALU Flag File Program Counter

A blue lines with white squares

Description automatically generated A diagram of a computer

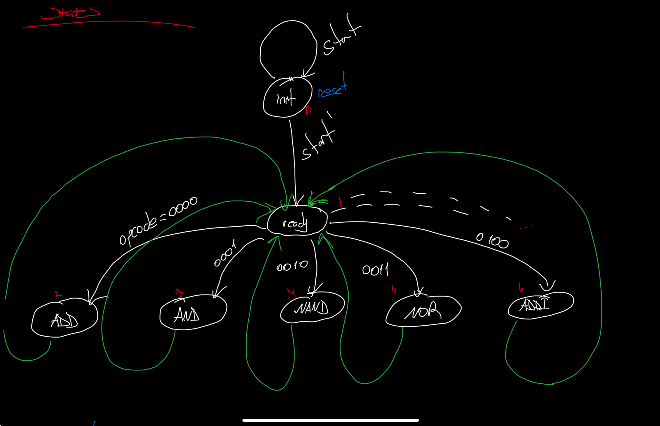
Description automatically generated

18-bit comparator 18-Bit Register File

# **Step III – Control Unit Design**

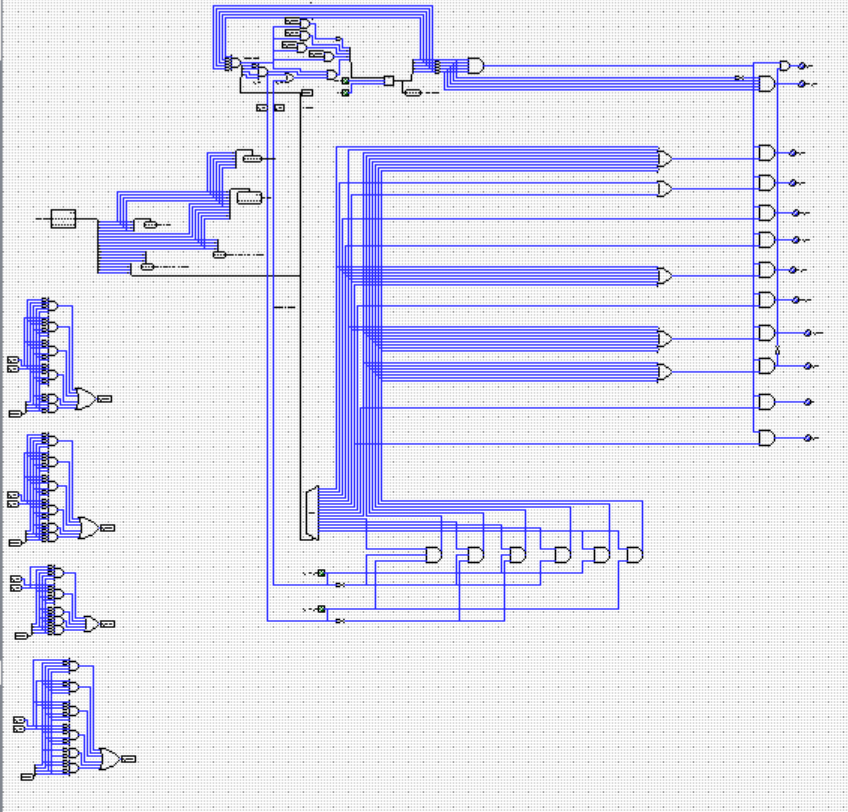
In this part of the project, we are now ready to design our control unit. Control unit is the brain of the computer. Basically, our computer is a finite state machine. Control unit is responsible for generating the correct signals and next state by examining the current state and instruction. Best way to do this is drawing the finite state machine on the paper and finding the logic circuit for all the outputs (signals and next state).

We first determined our states on paper. There are 17 states, one for init state, one for ready state and the others are for instructions. Each instruction has its own state.

A screenshot of a spreadsheet

Description automatically generated

After we found the logical circuit, we implemented the control unit on Logisim. Output signals of the control unit are connected to the appropriate components and multiplexers. We did some tests by using the assembler that we write in the first step and the results were perfect. Our processor is fully operational.



Complete design of the control unit

A diagram of a computer

Description automatically generated

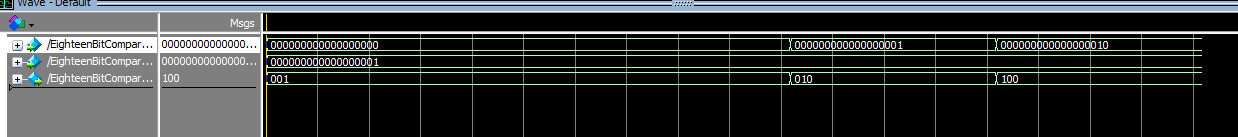
Whole Datapath

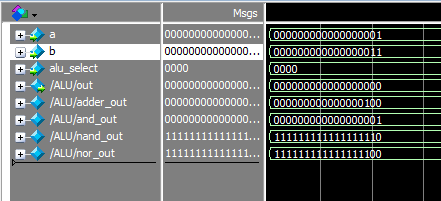
# **Step IV – Verilog Design**

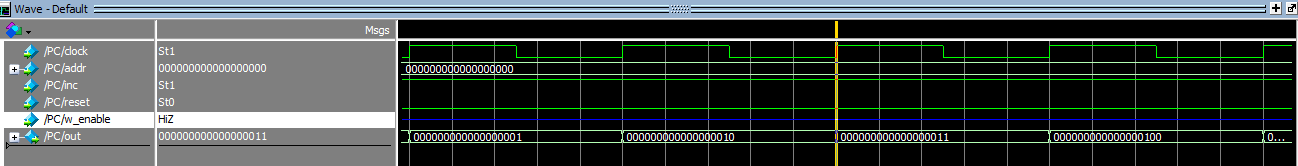
In this step, we are now ready to implement the process in Verilog. We first design our components as modules in the Verilog. Since we use instances of those components in other components, we begin from the simplest component to more complex ones.

There is an “always” keyword in the modules which is triggered by any change. The problem is that while there is a change in any part, the module triggers repeatedly. This might be unwanted scenario especially for registers. Registers are responsible for holding data. Therefore, in case of a change during the clock cycle, we have to avoid running the module. This is done by using “posedge clk” term. This means that run the module when a clock rising edge comes.

We created our components similar to the previous Logisim design. We added comparator, ALU, Register File, Program counter, etc. After that we simulated our component to test their behavior.







We didn’t know how to test the CPU because we don’t have any Rom or Ram modules. We tried to implement it, but we couldn’t read the file using Verilog.

