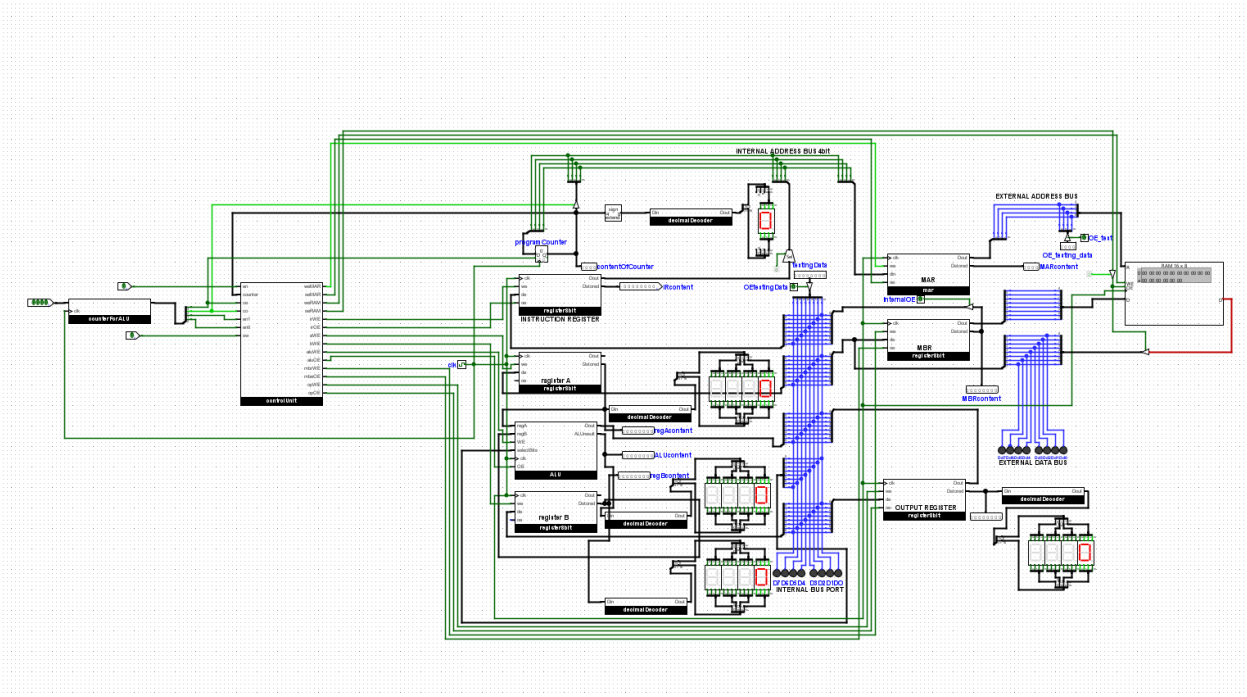
DESIGNING A 8 BIT COMPUTER

This project implements a Simple-As-Possible 8_bit computer and control unit on logisim-evolution.



Objective

The objective of this project is to

- To conduct an in-depth exploration of fundamentals of computer architecture, focusing on synthesis of control unit, ALU, and memory.
- To experience the complexity by building a complex system just by starting with a simple memory block.
- To learn the practical implementation of digital system design, simulation and testing.

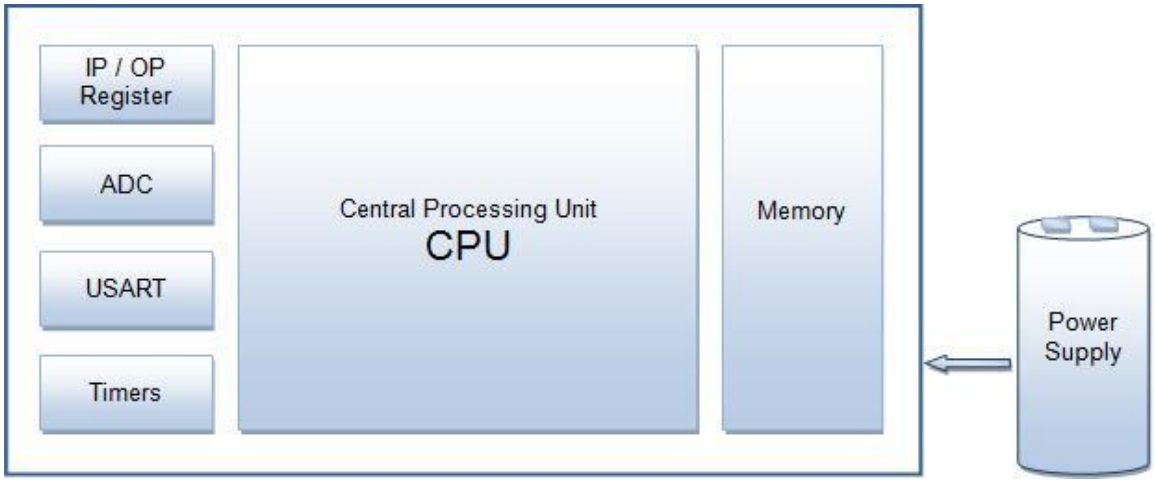
For better experience to the reader, we will divide this report into 3 basic parts.

1.LEARNING PHASE 2.DESIGNING PHASE 3.TESTING PHASE.

LEARNING PHASE

COMPUTER ARCHITECTURE

Microcontrollers are the systems used to perform specific tasks and often include interfacing with sensors and displays. They basically consist of CPU, memory and several peripherals. Embedded software is programmed on a separate computer then compiled into a list of instructions. This list of instructions is often known as firmware. The microcontroller executes the instructions present in this firmware.



The architecture of microcontroller can be divided into 4 parts :

1. INSTRUCTION SET ARCHITECTURE.
2. MICROARCHITECTURE.
3. SYSTEM ARCHITECTURE.
4. LOAD-STORE ARCHITECTURE.

(NOTE : ARCHITECTURE MEANS DESIGN)

ISA(INSTRUCTION SET ARCHITECTURE)

ISA describes the instructions that the computer can execute. This describes how the CPU is controlled by software. If you assume a computer to be a kitchen then ISA is a recipe book.

MICROARCHITECTURE

Microarchitecture is the actual physical design and logical organisation of processors that makes commands of ISA possible. There are two major parts of this architecture. One is THE DATATPATH : It includes ALU(arithmetic Logic Unit), registers, Buses, MUXes, etc. the other one is THE CONTROL UNIT : It tells the datapath what to do. It decodes the instruction and sends out electrical " control signals ". This can be done is two ways :

1. HARDWIRED CONTROL : Uses fixed logic gates for maximum speed(common in RISC philosophy).
2. MICROPROGRAMMED CONTROL : Uses a tiny internal memory(control store) to look up a sequence of signals.

Microarchitecture also takes responsibility for pipelining.

SYSTEM ARCHITECTURE

It describes the complete system including the processor, memory buses and peripherals.

LOAD-STORE ARCHITECTURE

It is a particular type of processor design and refers to the way in which data is operated on and moved around.

COMPUTER

A computer is a fairly unique machine which can perform almost infinite number of

Tasks on a single fixed hardware. This is done by running different softwares and these software contain instructions. This is called the STORED-PROGRAM concept. The computer then fetches these instructions from memory and executes them.

You can divide a computer into three basic components : PROCESSOR, MEMORY AND BUSES. The processor executes instructions, memory stores instructions and required data, and buses move instructions and data between processor and memory.

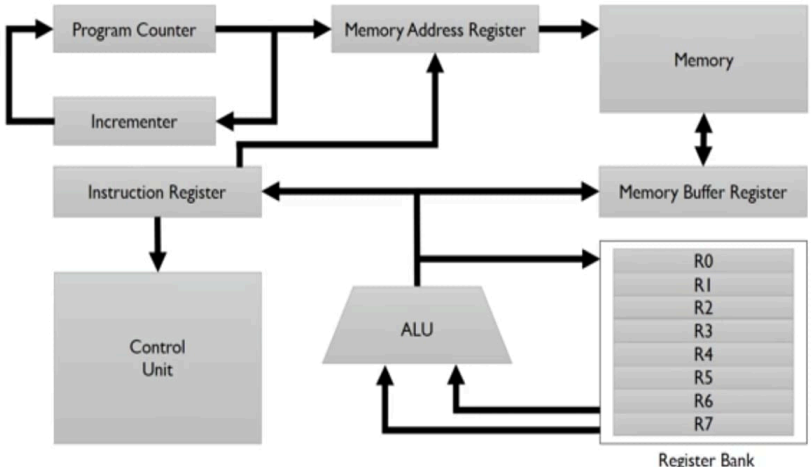
Let us try to understand everything part by part and combine the whole thing together at the end.

DESIGNING and TESTING PHASE

Since we have no clue of any of the components, let us try to understand computers in parts and then combine all of them to make a workable 8-bit computer.

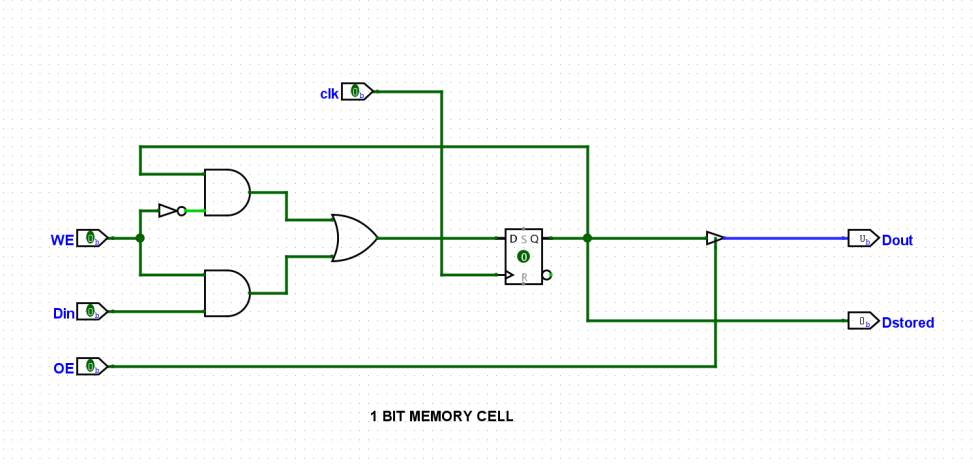
Let us divide the study of 8 bit computers into six parts.

1. REGISTERS, 2. MEMORY (RAM) 3. ALU 4. CONTROL UNIT 5.BUSES (IMPORTANCE OF ENABLE PINS) 6. TESTING



REGISTERS

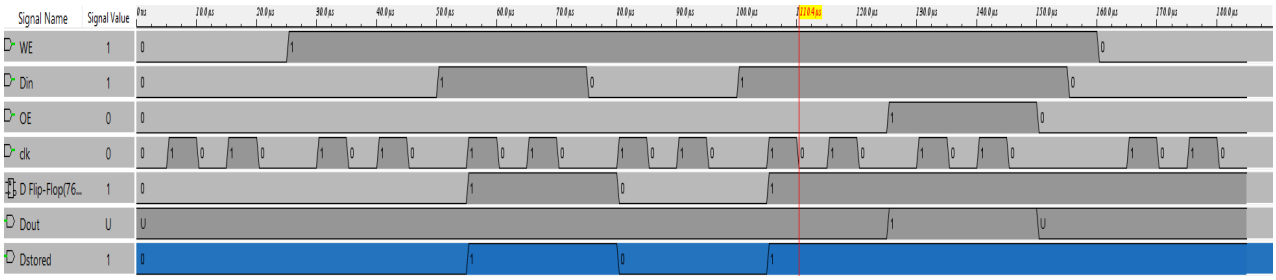
Registers are any bit memory cells that can store any bit memory. In our case it is 8 bit, but it could be 16 but 32 bit etc. We are designing an 8 bit computer, so we are using an 8 bit register.



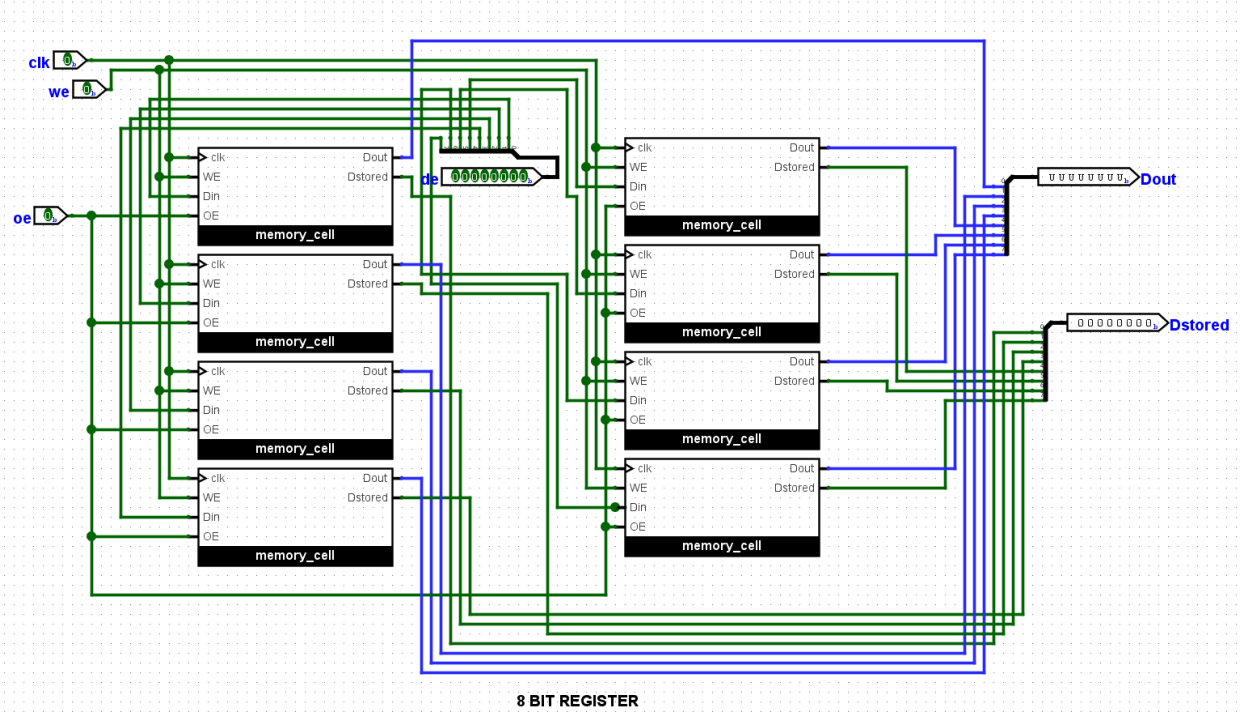
This is a one bit memory cell. Here, I have used a D flip flop. WE is a write enable pin, that means, it can only store input data when WE is on. Din is the input data and OE is output enable pin. You can share the output via Dout only when OE is on. To do this we could have used an AND gate as well but I have used a controlled buffer. D-stored is attached for us to know whether the data is actually being stored. The logic that has been connected to the D pin of the D flip flop can easily be written using the truth table and K-map. Truth Table is mentioned below :

clk	WE	Din	OE	Dout	Dstored
0	0	x	1	**1**	**1**
0	0	x	0	**-**	**1**
1	1	0	1	**0**	**0**

The following data will prove if the given circuit is working.

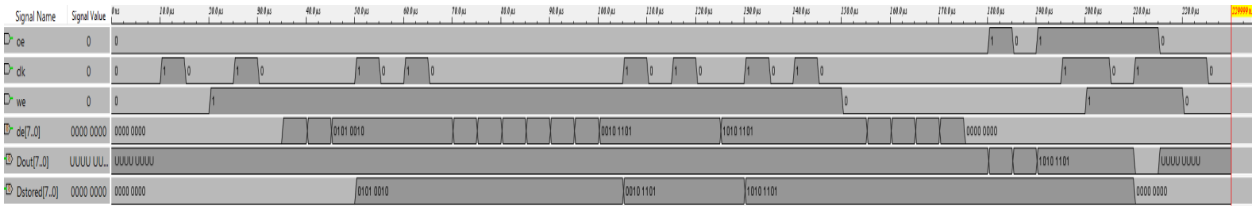


Now, using this 1 bit memory cell, we will design a 8_bit memory cell which we call an 8 bit register.



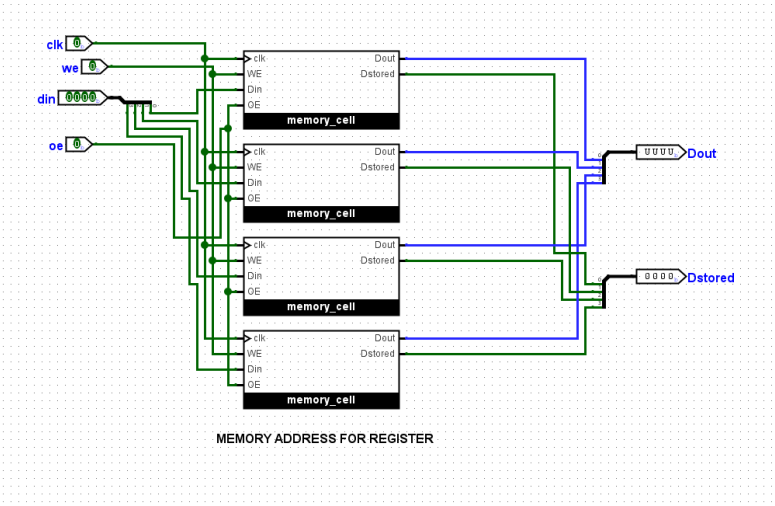
This is an 8 bit register designed using a 1 bit memory cell. de- is 8 bit input. Using a splitter, we split 8 bit data into individual bits which are sent to a memory cell where each bit is stored and then sent to output whenever required. You can see that WE pins of all memory cells are connected together to a common WE pin. The same is done with OE pins and clocks. The mechanism is the same as a 1 bit memory cell, the only difference is that it stores 8 bits.

Now, the simulation shown will prove its working.



Now, registers have many uses in our computer.

1. It is used to store our inputs we call regA and regB.
2. It will act as an instruction register, which we will discuss in the memory section.
3. It is used to store final output before displaying it in the output register.
4. We use it to store output in the memory buffer register(MBR) if we want to use it again by storing it in RAM.
5. Just like 8 bit registers we make a 4 bit register and name it a memory address register (MAR).



MEMORY

Memory is the one that will store instructions that are needed to be performed.

To understand this we first need to begin from the start.

First, we store the instructions that are needed to be performed in RAM. Now RAM is a random access memory that stores volatile data that burns off when reset or restarted.

We will not get too deep into how RAM is made because it is not our current priority but it is important to know how RAM stores instructions and how it sends the required data.

So, how are instructions saved in RAM ?

Basically, there are two types of architecture when it comes to data storage and address storage.

1. Harvard Architecture : In this architecture, instructions and data are stored on different memory compartments.

2.Von Neumann Architecture : In this architecture, both instructions and data are stored on the same memory. We are going to implement a 16x8 RAM following this architecture.

When we say 16x8, it means that RAM has 16 ,1 byte divisions and the first 8 bytes are used to store instructions and the next 8 are used to store data. Now, instructions are given by following some set of rules and this is what we call as Instruction Set Architecture.

We will define it this way :

INSTRUCTIONS : FIRST 4 BITS : OPCODE ->This is a code that decides the operation your computer will do. Let me explain it using the RAM that we are going to use as an example.

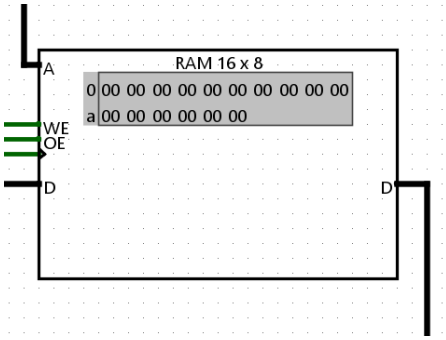
Our RAM has 16 bytes of space. Among that first 8 bytes are used to store instructions(This is not a hard and fast rule. You can choose how much of the space you want to give to instructions,etc.).

A pin brings a 4 bit address. For example 0000. Then we

Will enter first space. It will contain 8 bits of data.

The first 4 bits are OPCODE. OPCODE are basically bits

That decides the operation.



For example : 0000 => fetch regA data.

Next 4 bits store address of data that is needed to

Perform operation specified by OPCODE. In this case the last 4 bits will have the address of regA data. Let's say it is stored in 11th space. This means the overall data that is stored at first space is (OPCODE)0000(NXT-ADDRESS)1011 =>data that is present at first space is 00001011.

Now, this data is made to be stored in the instruction register. From there the last four bits are fetched and sent via pin A again, this will take us to 11th space. From there we transfer data to regA register by keeping pin OE on.

Now there is something called a Program counter. This is what sends OPCODE. It is basically the counter that decides the number of cycles in which a program runs. The following paragraph will explain how the computer works.

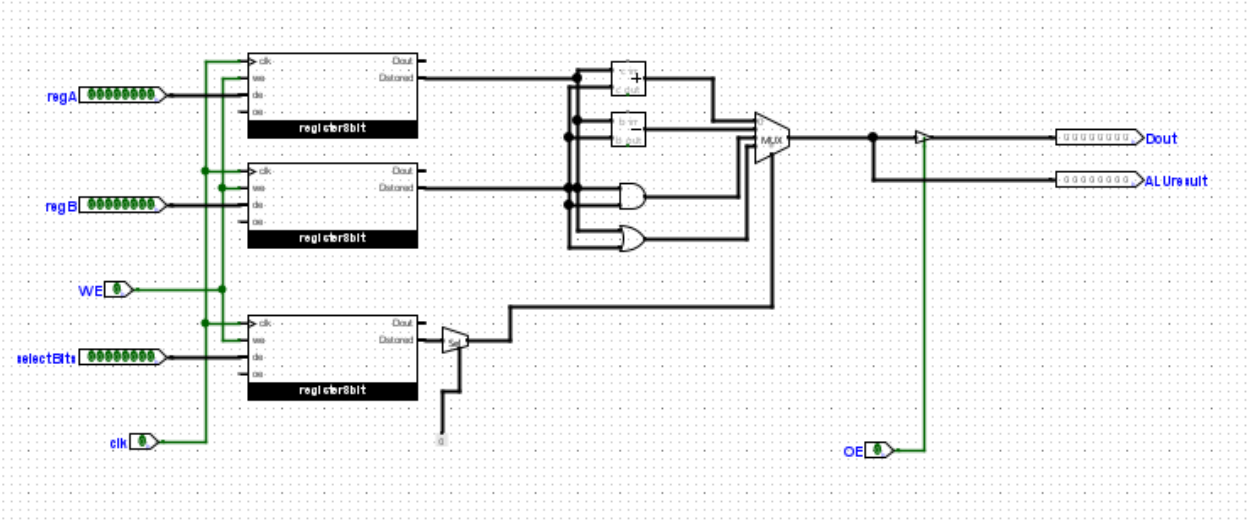
The program counter at first sends four bits 0000. MAR will store it. And then it will send it to RAM. RAM will enter first space. OE pin will be on, the data from there will be sent to the instruction set. The last four bits of it will be sent back to MAR. It is then sent to RAM and that specific space will open. From there data is sent. The WE pin of the required register will be on and data will get stored in it. By this time the counter reaches 0001. Next it will enter 0010. The process will continue. If you want to send data to RAM to store it, enable WE and data will be stored from MBR at whatever space the RAM is in.

ROM is basically a circuit with predefined enablers,etc, when the clock is connected, it gives out pre-stored data and it is non volatile.

There are many 8 bits testing connections connected and also display LEDs are connected to get data displayed on LED.

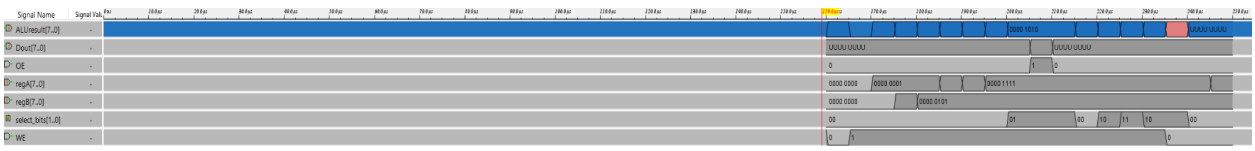
ALU

This is the logical part of the computer. It stores operations on given inputs regA and regB.



This is the design of ALU which can perform addition, subtraction(does not take care of overflow) , bitwise AND and bitwise OR. ALU is basically an Arithmetic Logic Unit that can do operations based on the inputs given. We use an 8 bit adder, subtractor, AND gate and OR gate for our ALU. In order to know which operation to be done, we connect a demux which gets input from the Instruction Set. Since, we get input as 8 bits and we are using 4x2 demux, we need to do bit selection using bit selector. Let us stick with the first 2 bits of all the 8 bits as selector line inputs for demux. Here, whenever, selector line indicates 00, addition is performed. If it shows 01, subtraction is done. For 10, AND operation and for 11, OR operation is done. The following simulation proves that ALU is working.

The simulation here proves that ALU is working fine. Also, MUX will take care of which operation to perform.



CONTROL UNIT

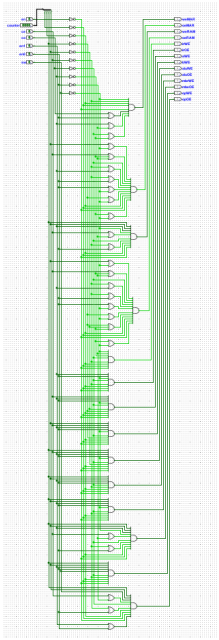
In order to make the circuit almost automated without changing WE and OE pins manually, we need a system that can do it. In-order to make that happen, we need to know what is the number of pins we need to change, at what counter, which input is changing,etc. This has been explained through the truth table given here very clearly. Hence we create a control unit using the truth table shown. This control unit can perform almost any operation you want to perform on this computer. In order to automate the control of WE and OE pins, we make a control unit by writing a truth table as shown. Here, en0, en1, sw, en are the external input pins that are implemented to make sure that 1 instruction transfer can be done by the end of 1 count itself. If these pins were not used then, for the completion of one instruction it could have taken 3 clock counts.

en	counter	ce	co	en1	en0	sw	weMAR	oeMAR	weRAM	oeRAM	irWE	irOE	aWE	bWE	aluWE	aluOE	mbrWE	mbrOE	opWE	opOE
0	0000	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0
0	0000	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
0	0000	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
0	0000	0	0	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0000	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0000	1	0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0
0	0001	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0

0	0001	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
0	0001	0	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
0	0001	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
0	0001	0	0	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0001	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0001	1	0	1	1	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0
0	0010	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0
0	0010	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
0	0010	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
0	0010	0	0	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0010	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0010	1	0	1	1	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0
0	0011	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0
0	0011	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0

0	0011	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0
0	0011	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0011	0	0	1	1	0	1	0	0	0	0	1	0	0	0	1	1	0	0	0
0	0011	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0
0	0011	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0011	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0011	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0
0	0011	1	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0

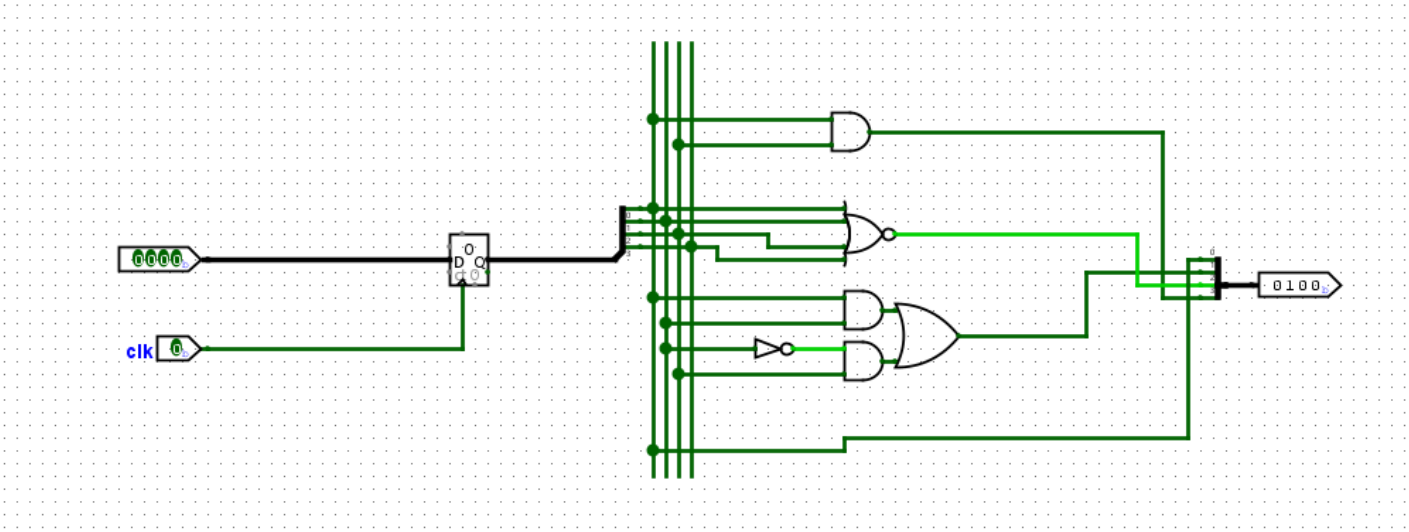
Once this is done, you get a circuit as shown here



BUSES

In order to avoid confusion and debugging headache, we use buses. River flow and Enable pins are like dams. Until and Unless enable is on, data won't flow into anything. When enable is On, data flows into that specific register. So we just need to control WE and OE pins.

Now, if you observe ce, co, en0, en1 repeat themselves for every count. So, if we make a count of this for every cycle, then all we will be left with is to control sw and en pin. En pin is a pin when all the we and oe pins shut and only the OE pin of the output registers becomes one at 6th cycle. TCounter that can do this is implemented down here :



TESTING

(a) ADD any 2 values. Checked. Link :

https://drive.google.com/file/d/19wfwTI_0S5VwjcUkGofyiqHUEaDmW-UB/view?usp=sharing

(b) PERFORM SUBTRACTION FOR THE SAME. Checked.

(c) PERFORM BITWISE AND FOR THE SAME. Checked.

(d) PERFORM BITWISE OR FOR THE SAME. Checked.

Resources link :

https://drive.google.com/drive/folders/1sMrMSK0UKaPnKEJpVvSIdTlXUXKOJJhb?usp=sharing

NOTE THAT YOU NEED LOGISIM-EVOLUTION TO RUN THIS.

THANK YOU.