Simple As Possible 8 Bit Computer Design

Akhilesh Vadde
National Institute of Technology, Surathkal, Karnataka
2nd year, BTech, Electronics and Communication Engineering

Abstract---Computers have become one of the most important parts of our lives. Many of the activities have been automated and have become easier to complete because of computers. It is almost impossible to imagine technological growth without computers. But there are still many obstacles such as performance, power consumption, efficiency, speed, etc. that need to be addressed to make computers more reliable and useful. To make this happen, it is necessary to understand the architecture and organization of computers. This project aims at understanding and implementing computer architecture using Logisim-evolution to increase our understanding about computers.

Keywords---Computer Organization and Architecture, Logisim-Evolution
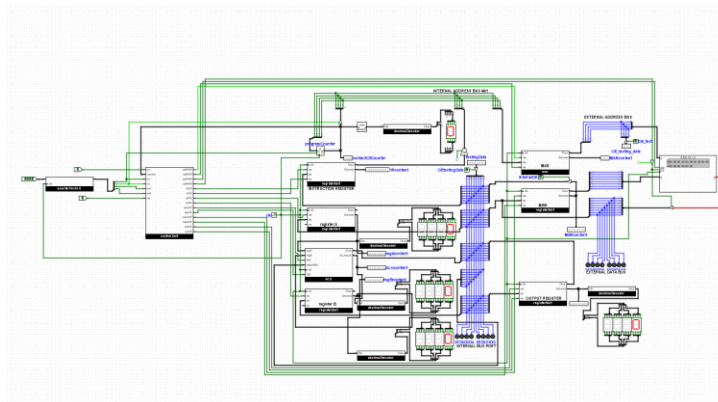
Overview

Figure 000

To have an overview, the final design of Simple-As-Possible 8 Bit Computer will look as shown in *figure 000.* The major objectives of this project are

- To conduct an in-depth exploration of fundamentals of computer architecture, focusing on synthesis of control units, 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 a better experience for the reader, we will divide this report into 3 basic parts.

I.   LEARNING PHASE   II.   DESIGNING PHASE   III.   TESTING PHASE.

I. LEARNING PHASE

COMPUTER

A computer is a unique machine which can perform almost infinite number of Tasks on single fixed hardware. This is done by running different *soft wares,* and these *soft wares* 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, while buses move instructions and data between processor and memory. Let us into these components individually.

A. PROCESSOR

The processor is the component of a computer that performs all the necessary operations on data we get. We can call it a black box or a system that takes inputs, performs operations, and produces outputs. The processor consists of 3 basic components, and they are a control unit, arithmetic logic unit (ALU) and input/output peripherals. Let us investigate each of these components in detail.

 a. *Control Unit*---Control unit is one of the major components of a processor. It is responsible to guide necessary instructions to a desired location per clock cycle. It also ensures proper data transfer via buses.

 b. *ALU*---Arithmetic Logic Unit is the main and important component of the processor. It takes data from memory and performs operations on it as specified by instructions from the Instruction Set. After performing the operations, processed data is sent to the desired destination via buses.

 c. *Input/Output Peripherals*---Input peripherals of processor are designed to work differently. They take instructions and data from other registers or memory block as input only when a pin called *write enable* is one. Output peripherals of processor take care of sending processed data through output peripheral only when a pin called out enable is on/one.

You can conclude from this that the processor is that part of your computer which is responsible for processing inputs.

B. MEMORY

Memory is a component of a computer which is responsible for storing data and instructions. We can study about memory into four major parts.

 a. *Registers*---These are the memory blocks from which processors directly access the data. There are many types of registers. Registers that store address is called memory address register. The one which stores instructions is called instruction set register, the one which stores processed data or unprocessed data before reaching its desired destination is called memory data register. Like these, there are many types of registers.

 b. *RAM*---Random Access Memory is a fast and volatile memory block that basically stores instructions from which processor fetches instructions and data.

 c. *ROM*---Read Only Memory is jsut like RAM except for the fact that it is nonvolatile and it already has instructions which are implemented with the purpose of proper functionality of a computer.

 d. *External Memory*---These need not be always connected to a computer. It is simply used to store data, and it does not vanish if power is cut. It is not as fast as RAM or ROM.

C. BUSES

It is not resource efficient to connect every input and output preipherals to individual wires. So, we create buses.
Bus is a single path flow or set of path flows through which data flows to every component it is connected, but it is accessed only be desired component as we can control enable pins. This control is established by a control unit.
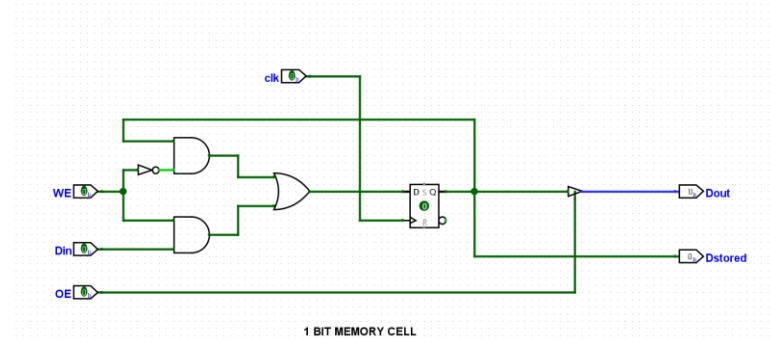
II. DESIGNING PHASE

Let us divide the design of 8-bit computers into five parts

1. REGISTERS 2. MEMORY 3.ALU 4. CONTROL UNIT 5. BUSES
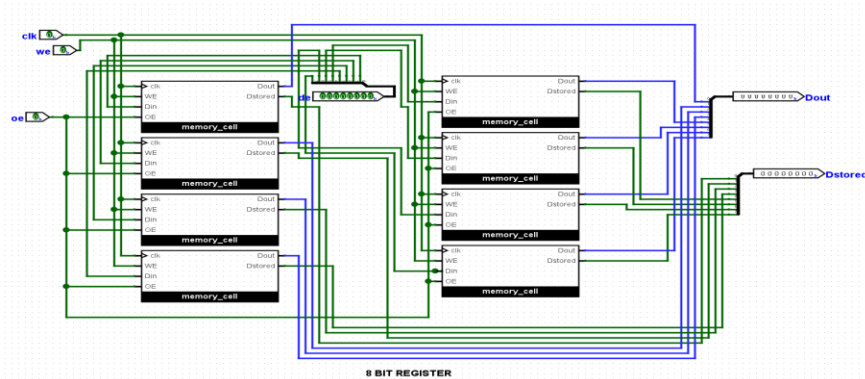
REGISTERS

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

1 BIT MEMORY CELL

This is a one-bit memory cell. Here, I have used a D flip flop. WE are a write enable pin, that means, it can only store input data when WE are on. Din is the input data, and OE is the 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	0	0
0	1	0	0	-	0
1	1	0	0	0	0

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


8 BIT REGISTER

This is an 8-bit register designed using a 1-bit memory cell. de is an 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 the WE pin of all memory cells are connected 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, 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 the 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. (Note that we are not using it to store data from RAM for the sake of simplicity)
5. Just like 8-bit registers, we make a 4 bit register and name it a memory address register (MAR).

MEMORY ADDRESS FOR REGISTER
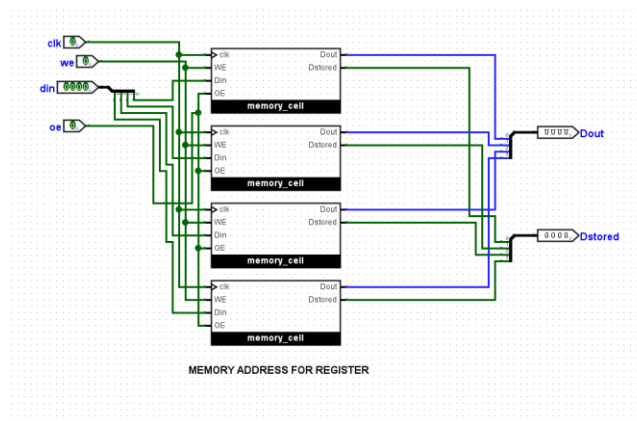
Memory is the one that will store instructions that need to be performed. To understand this, we first need to begin from the start.

First, we store the instructions that need 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 in 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 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 those 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 for the instructions). 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 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 first sends four bits of 0000. MAR will store it. And then it will be sent 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. When the clock is connected, it gives pre-stored data, and it is nonvolatile. There are many 8 bits of testing connections connected and also display LEDs connected to get data displayed on LED.
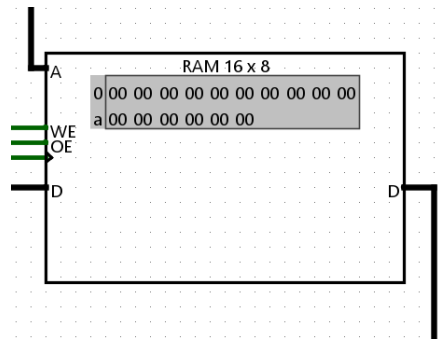
ALU

This is the logical part of the computer. It stores operations on given inputs regA and regB. The figure given below 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. To know which operation to be done, we connect to 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.
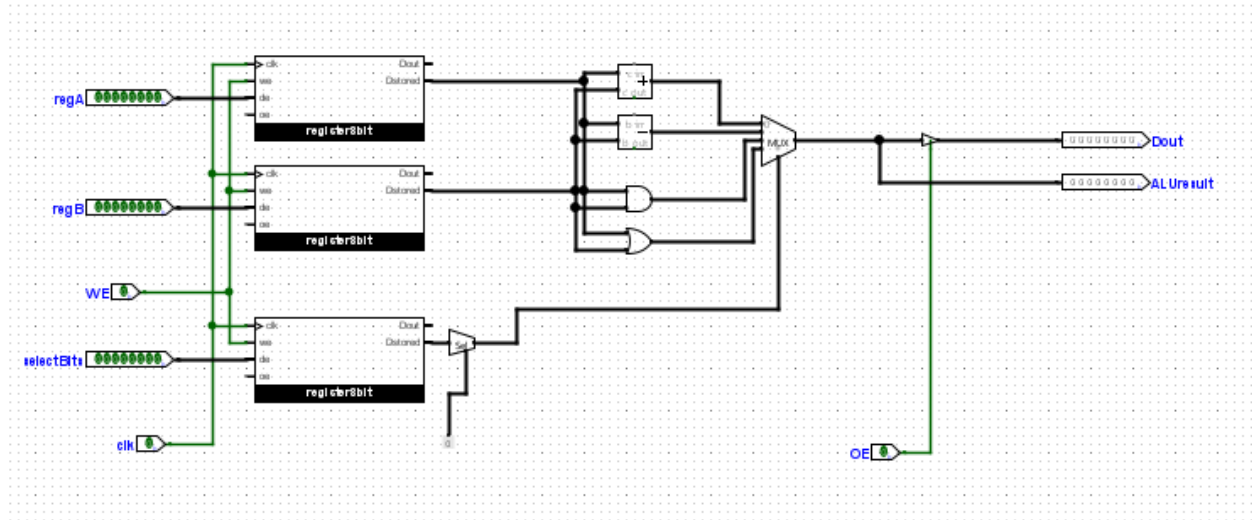
CONTROL UNIT

       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 the number of pins is 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. 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.

The truth table for this is given in the src folder in GitHub link attached to this report.

After implementing it, we get the following circuit.

(NOTE: Memory addressing modes: Addressing modes are techniques used by the CPU to identify the location of the operand(s) needed for executing an instruction. They provide rules for interpreting the address field in an instruction, helping the CPU fetch operands correctly. There are many types of memory addresses but we are using *direct addressing* mode. The instruction contains the memory address of the operand. The CPU accesses the data directly from that address.)
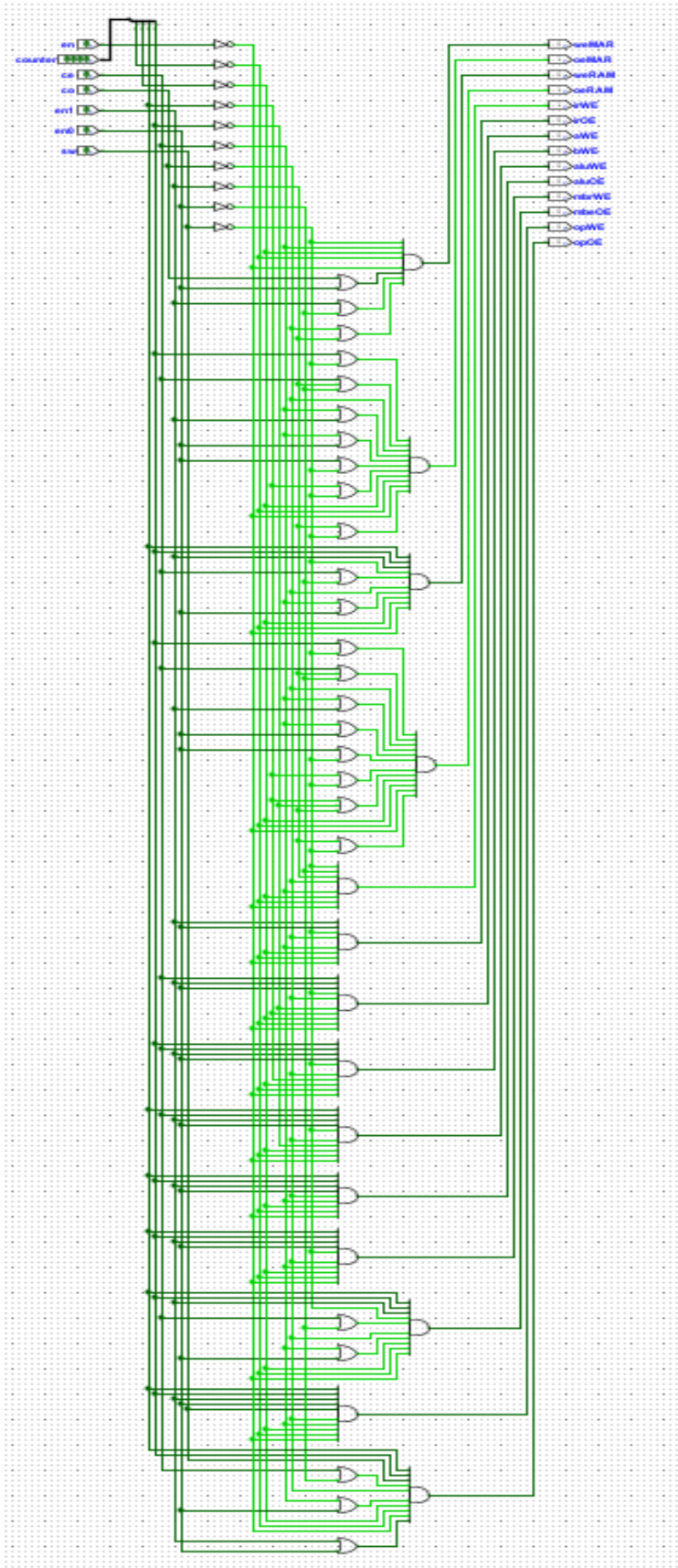
BUSES

To avoid confusion and debugging headaches, 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. We have an internal data bus and an external data bus. Internal data bus connects ALU, regA, regB, Instruction set registers while external data bus connects RAM, memory buffer register etc. There is one more bus that takes care of addresses from program counter and instruction set register. So, we call it the address bus.

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. Counter that can do this is implemented down here:
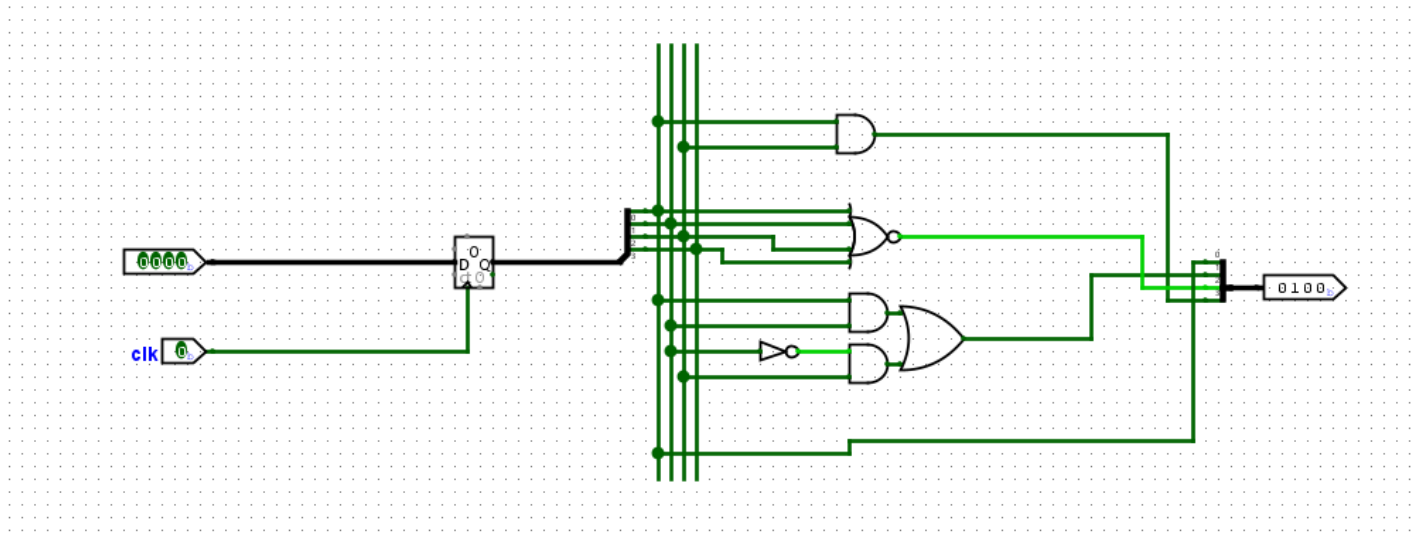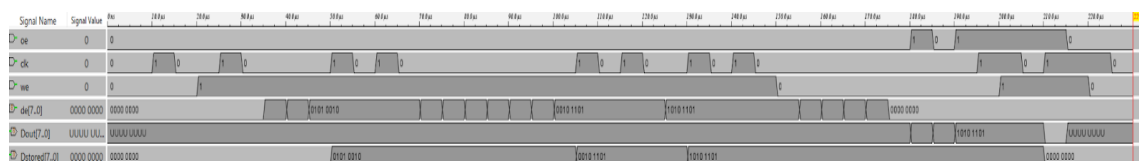

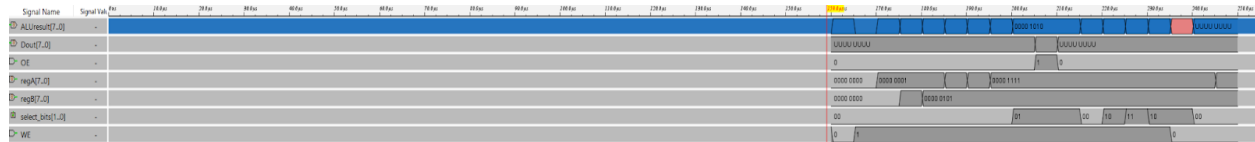## III. TESTING PHASE

a. Simulation for 1 bit memory:
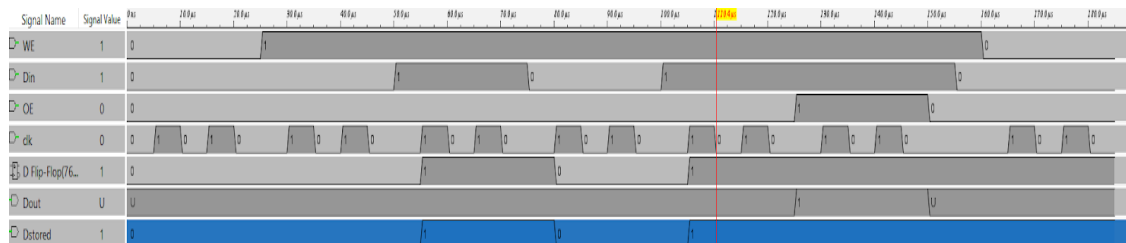
b. Simulation for register:

c. Simulation for ALU:

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.

Github link : https://github.com/Dr-Fhinx/ECE_YourCupOfChai/tree/main/SAP_8_bit_computer

Conclusion

We have successfully implemented a computer design that can perform addition subtraction AND and OR operation on given 2 8-bit numbers. We had an in-depth exploration of fundamentals of computer architecture, focusing on synthesis of control units, ALU, and memory. We got to know the complexity of building a complex system just by starting with a simple memory block. We have learnt the practical implementation of digital system design, simulation, and testing.

Resources

- https://youtube.com/playlist?list=PLsHOxmMgAiQzTWKWKHYHSnYNPBb2dqmQv&si=8FzfQkA3p_XLoGjc
- https://youtu.be/JPfG0UQd3x4?si=BB1dpmefsV_Gzis4
- https://youtu.be/gYmDpcV0X7k?si=4Ue8MvtyH0pW9KTn
- https://www.geeksforgeeks.org/computer-organization-architecture/addressing-modes-1/
- https://en.wikipedia.org/wiki/Central_processing_unit
- https://www.geeksforgeeks.org/computer-organization-architecture/difference-between-register-and-memory/
- https://youtu.be/dXdoim96v5A?si=vdvd4lPkRkknkYvT
- And many other resources.