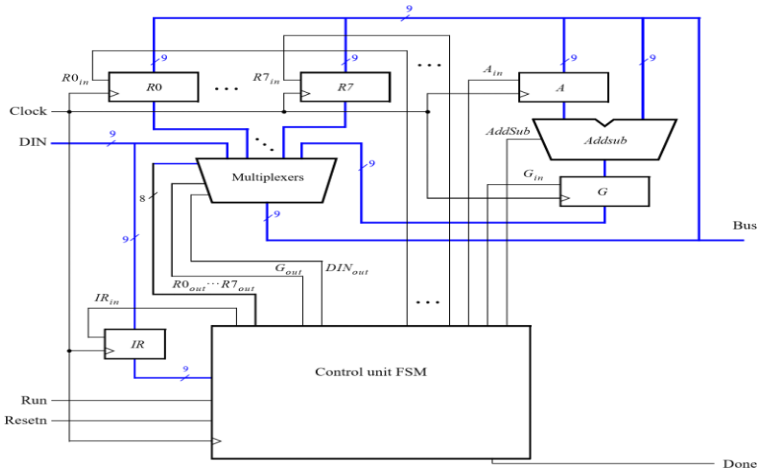
Simple Processor

Overview:

A processor, also known as a Central Processing Unit (CPU), is the main component responsible for executing a computer program by performing arithmetic and logical operations, controlling input and output devices, and executing other instructions. The design and construction of the processor must be done using an appropriate hardware description language (HDL), such as Verilog.

In this work, we will design and build a processor using Verilog.



Main components of the processor:

- **Registers:** Temporary storage locations within the CPU that hold data and instructions currently being used. They are accessed quickly and used to store intermediate results and control information.
- **Control Unit:** Manages the execution of instructions by directing the various components of the CPU to perform the necessary operations in the correct sequence.
- **Arithmetic Logic Unit (ALU):** Performs arithmetic and logical operations, such as addition, subtraction, multiplication, division, and logical operations like AND, OR, and NOT.
- **Memory:** Stores the program code and data being processed by the CPU.

The processor we will design will follow these principles and will be implemented using Verilog, a hardware description language commonly used in the design and simulation of digital systems.

Design Phase:

Table 1: Input and output signals of "subunit MUX"

input signal	Meaning	output	Meaning
Source selector	Bits represent which source to output there are 10 of these	Output source	The source selected by "Source Selector"
source	the mux has 10 sources		

Table 2: Input and output signals of "decoder subunit"

input signal	Meaning	output	Meaning
Activation	This input signal enables the unit to work	Register	Register using one - hot method
Register	Register by binary number		

This unit receives a binary number (the number of a register to which you want to contact) and converts the number into a representation using the "hot one" method. This representation helps prevent leakage of information from other registers because it is only defined when there is only one.

Note: In digital design, particularly in the context of a finite state machine (FSM) or other state full components, "register using one-hot method" refers to encoding the state of the FSM such that only one bit is set to '1' (hot) at any given time, while all other bits are '0'. This is known as one-hot encoding.

Table 3: Input and output signals of "Register Subunit"

input signal	Meaning	output	Meaning
clock	system clock	Reserved information	The information stored in the register
incoming information [8:0]	The information entered into the register		

This unit stores information with a size of 9 bits.

Table 4: Input and output signals of the "ALU" unit

input signal	Meaning	output	Meaning
incoming information [8:0]	The information entered into the unit (Bus)	output information	The result of the computational operation that was performed
incoming information [8:0]	The information that comes out of register A		
Action selection	Chooses which action to perform		

Table 5: Input and output signals of the "state machine" unit

input signal	Meaning	output	Meaning
Clock	system clock	finish	Shows that the last operation has finished
reset	Resets the state machine	the data channel	The internal data channel of a state machine is set to be visible and therefore found as well
Running a command	Activates the state machine for the entered operation		
incoming information	the input channel of the state machine		

This unit is a state machine that represents the "brain" of our processor. Each command will be executed over several different clock cycles. Shift operations will take two clock cycles, while operations requiring the logical unit will take four clock cycles. In general, the first clock cycle will be for receiving the command from the incoming information and saving it in a dedicated register, and then the operations will be performed as follows:

	T_1	T_2	T_3
move	RY - Output mux RX -save register DONE = 1		
Move immediate	Output mux - information entered RX - save register 1 = DONE		
add	RY - Output mux A - save register	RX - Output mux G - save register logic unit - add	G - Output mux RX - save register 1 = DONE
sub	RY - Output mux A - save register	RX - Output mux G - save register logic unit sub	G - Output mux RX - save register 1 = DONE
Mult by 3.5	RY - Output mux A - save register	RY - Output mux G - save register logic unit – mult by 3.5	G - Output mux RX - save register 1 = DONE

In the first step, which is the same for all states, the machine will save the input signal in the IR register until a "command activation" signal is received. This signal will activate the machine according to the entered command.

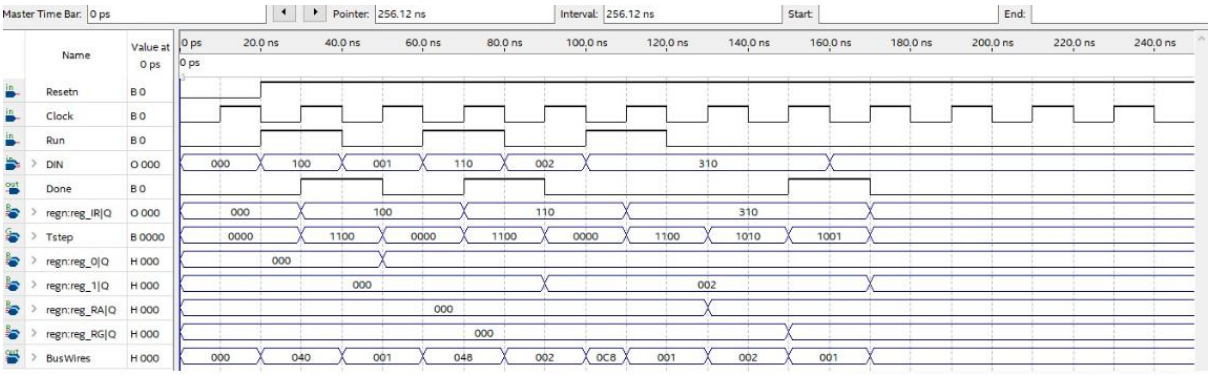
Incoming command format - An incoming command is nine bits in size and is divided into three parts of three bits each. The right three bits are the source register RY (represented as a binary number) the central three bits are the destination register RX and the left three bits are the instruction to execute.

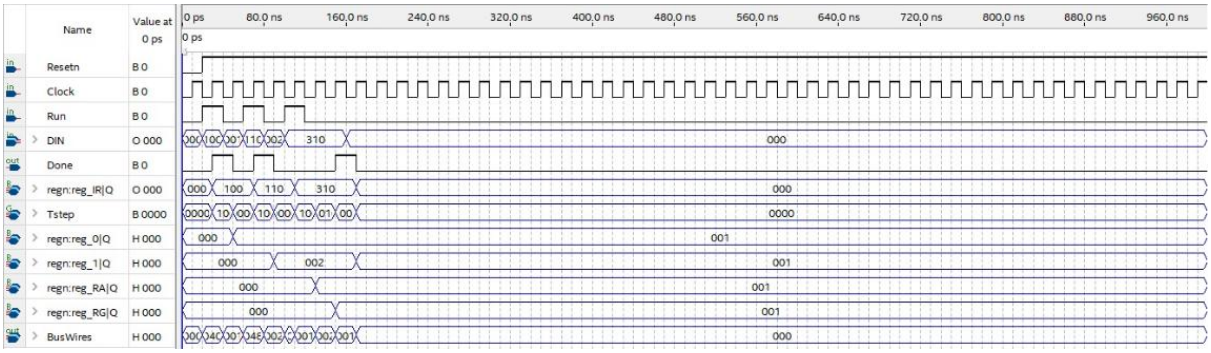
000 represents move register, 001 represents move number,010 add, 011 sub,101 mult by 3.5

test phase:

we tested all the operation the processor can do one at a time to make sure everything was working according the rules that was set in the lap instruction.

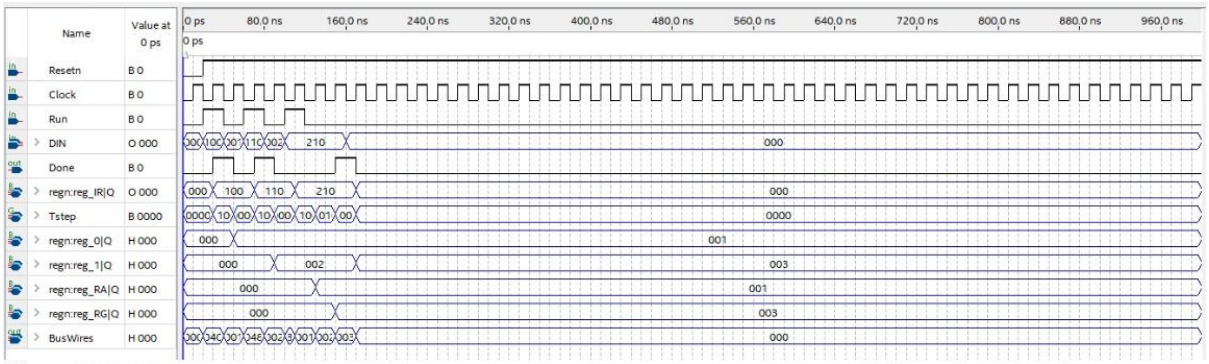
sub:

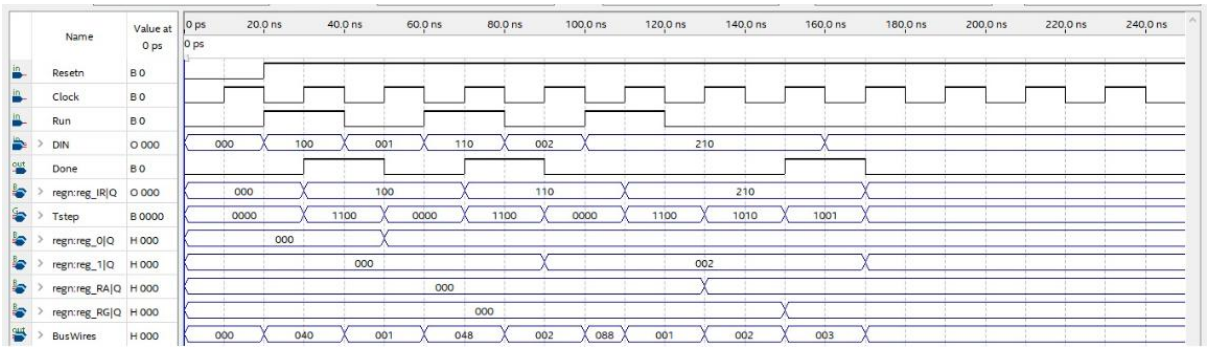




We used MVI to put the value 1 in the register 0 and then we used MVI again to put the value 2 into the register 1 and then we did the sub instruction, we took register 1 and subtracted register 0 from it and saved the final result (which is 1) in register 1 after the done signal.

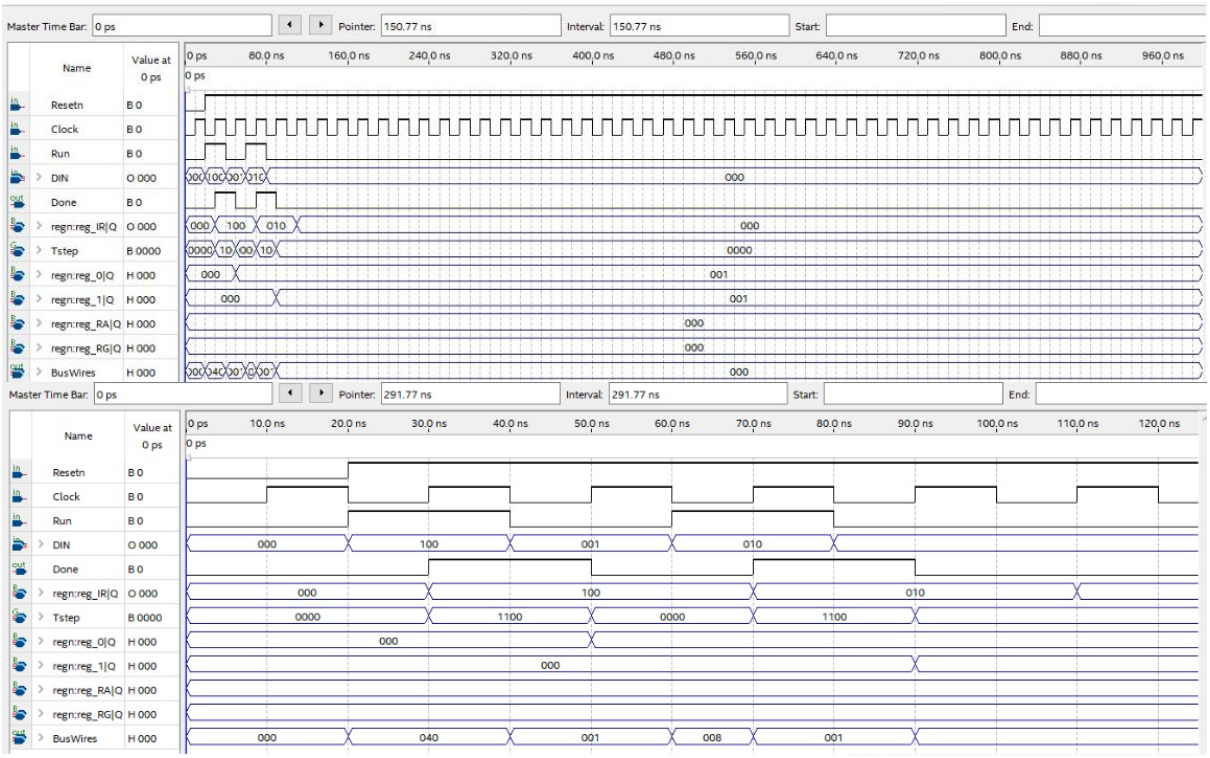
Add:





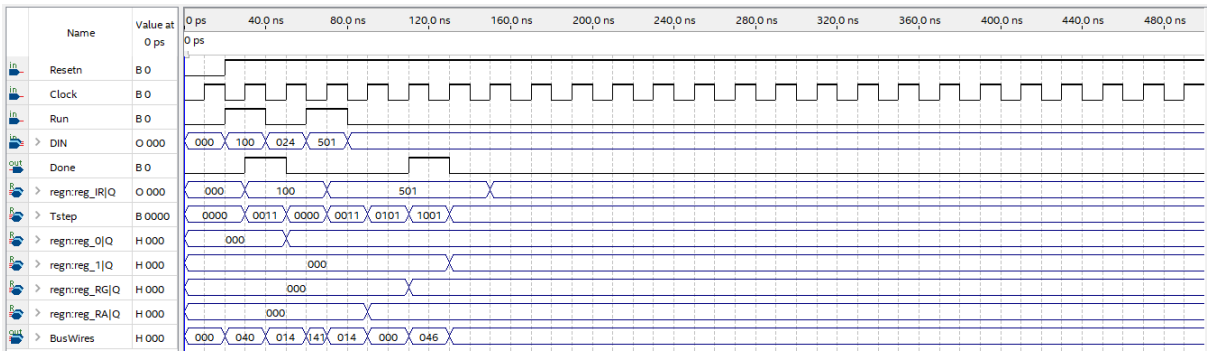
We used MVI to put the value 1 in the register 0 and then we used MVI again to put the value 2 into the register 1 and then we did the add instruction, we took the two registers and add them up together and saved the final (which is 3) result in register 1 after the done signal.

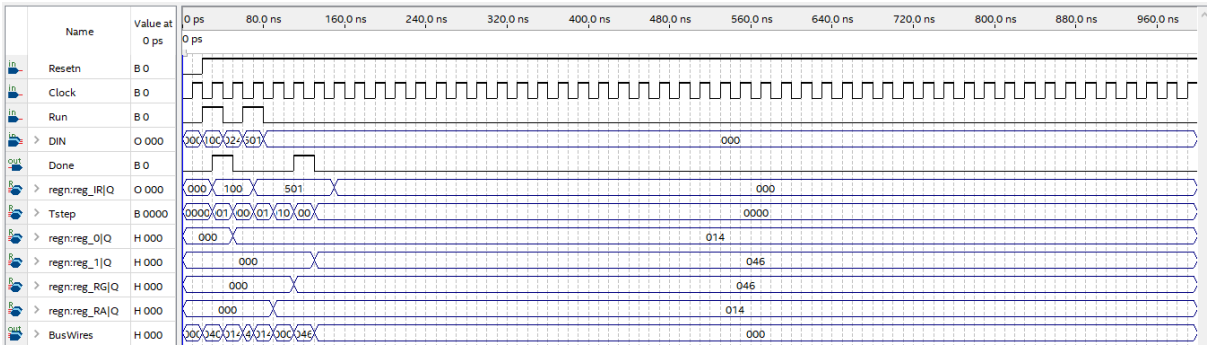
MV:



We used the MVI instruction to put 1 in register 0 and then we used the MV instruction to move the value of register 0 to register 1 after the done signal.

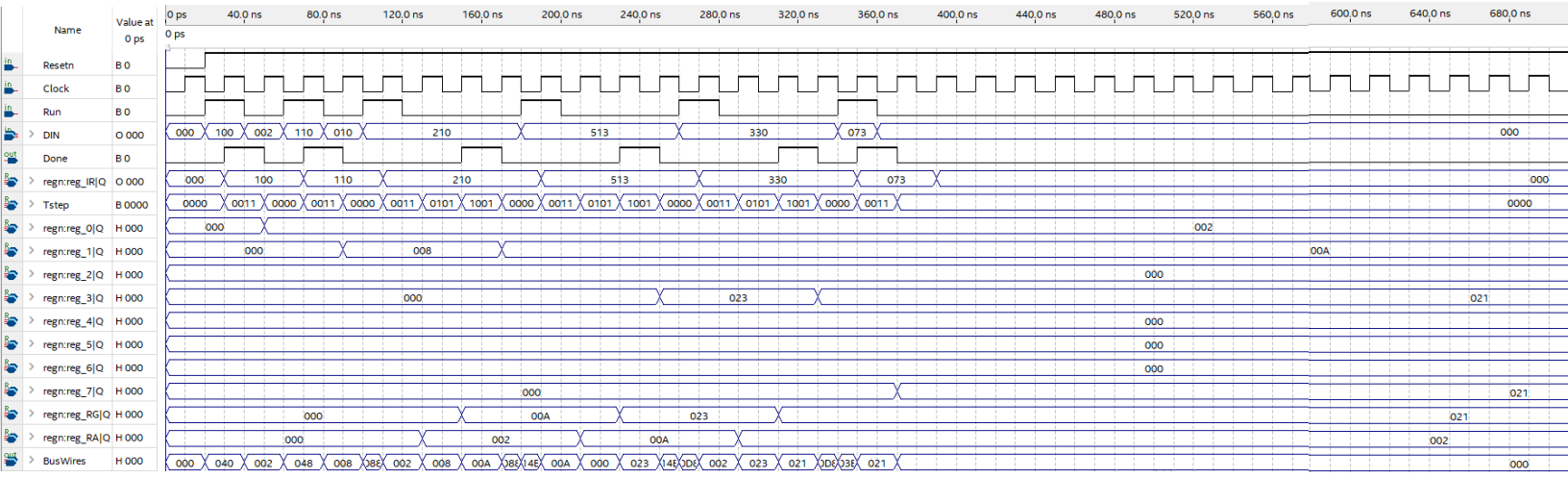
Special mult:





We used the MVI instruction to put the value 20 in register 0 and then we did the special mult instruction that will multiply the 20 by 3.5 and will put the final result into register 1 (which is 70) after the done signal.

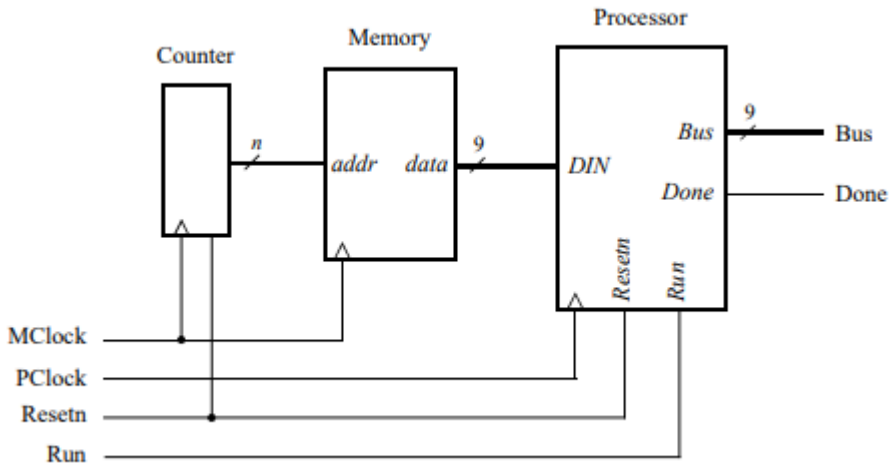
Now let's test all the instruction at once:



We started used the MVI instruction to put the value 2 in register 0 and the value 8 in register 1 then we used the add instruction to add register 0 and 1 and save the final result in register 1 then we used the special mult to mult register 1 by 3.5 and we got 35 and saved it into register 3 and then we did the sub instruction and subtracted register 0 from register 3 and saved the final result into register 3 (which is 33) and then we used the MV instruction to move the 33 to register 7.the MV and MVI instruction took 2 cycles to finish and the add, subc and mult took 4 cycles to finish . we did a vedio for this simulation https://drive.google.com/file/d/14kSA2327gG_4DtvX3HMasfiHiJe4agZJ/view?usp=drivesdk

Part 2:

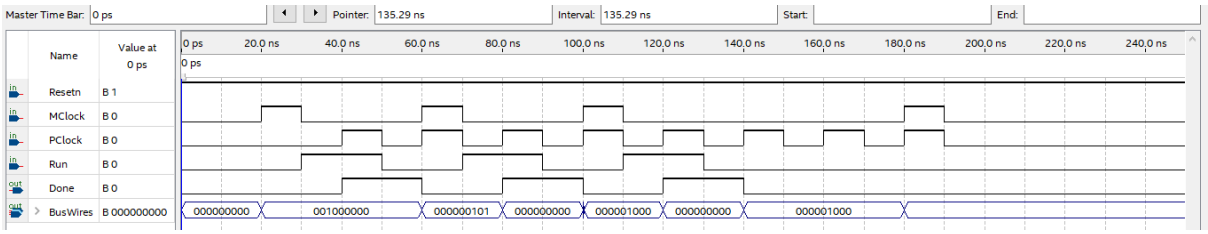
Memory:



explanation of the relevant parts:

1. **Processor and Memory Interface:**
    - The processor interfaces with memory using a 9-bit address (addr) and data lines (data). The DIN signal is used for input data.
    - The Memory block uses these signals to read or write data based on the processor's requests.
2. **Control Signals:**
    - **Resetn**: This signal is used to reset the processor and other components.
    - **Run**: This signal starts the processor's operation.
    - **Done**: This signal indicates the completion of the processor's operation. When Done is high, it means the processor has finished its task.
3. **Clock Signals:**
    - **MClock**: This is the memory clock signal, which synchronizes the operations related to memory.
    - **PClock**: This is the processor clock signal, which synchronizes the processor's operations.
4. **Counter:**
    - The Counter block is used to count clock cycles or other events. It interacts with the reset signal (Resetn) and the clock signal (MClock).

Something went wrong with the wave form and we didn’t know how to fix it



Here's a video for lap 1 part 2

We did the following sequence:

1010110110

01010110

101110110

1011 reset 010110

https://drive.google.com/file/d/1jSxJ8sZcDnfIQWQrdA4je1Z-06DxiBFG/view?usp=drivesdk