**COLTIUM INDUSTRIES**

**TASK**

**INTERNSHIP PROGRESS REPORT 2**

NAME

**BRIAN ONZERE**

STUDENT REGISTRATION NUMBER

**M064/KE23**

**TABLE OF CONTENT**

1.0. OBJECTIVES 1

2.0. PROGRESS 2

8 – BIT MCU DESIGN 2

CENTRAL PROCESSING UNIT 2

MEMORY 3

CLOCK 4

INPUT AND OUTPUT (I/O) 4

PROGRAM COUNTER 5

STATUS FLAGS 5

REGISTERS 5

8-BIT PROCESSOR MCU BLOCK DIAGRAM 6

8-BIT PROCESSOR MCU STATE TRANSITION DIAGRAM 6

alu.v 8

register\_file.v 9

reg\_a.v 10

reg\_b.v 11

reg\_c.v 12

program\_counter.v 13

instruction\_memory.v 14

instruction\_register.v 16

decoder.v 17

flags.v 19

control\_unit.v 20

mux1.v 21

pc\_mux.v 21

data\_register.v 21

data\_memory.v 21

io\_module.v 22

datapath.v 23

mcu\_top.v 23

tb\_mcu\_top.v 23

clock\_gen.v 24

COMBINED SIMULATION 24

SYNTHESIS 24

3.0. Pending 25

# 

# OBJECTIVES

The main objective of this task is to build my understanding of HDL.

Below are the specific objectives

* Familiarize with HDL.
* Design a basic 8-bit CPU architecture (registers, ALU, control unit).
* Write a basic instruction decoder module in Verilog.
* Simulate and test it in Icarus Verilog.
* Implement your simple microcontroller design on an FPGA using nextpnr.
* Run synthesis using Yosys.
* Design atmega32

# PROGRESS

## 8 – BIT MCU DESIGN

I went though various material (books and videos) to gain an understanding of MCU and processor architectures and design. Below are the basic elements of an MCU

* + Central Processing Unit
  + Memory
  + Clock
  + Input and Output (I/O)
  + Program Counter
  + Status flags e.t.c.

### CENTRAL PROCESSING UNIT

This is the processing element of the MCU. Its is the brain of the MCU. The CPU is the main part of the MCU that reads and runs instructions. It controls the operation of the entire microcontroller. The CPU includes the following important parts:

* **Arithmetic Logic Unit (ALU)**

The ALU performs all basic math operations like addition and subtraction. It also handles logic operations like AND and OR. The ALU receives inputs from registers or memory and sends the result back to a register or memory.

* **Control Unit**

The Control Unit reads the instruction from the program and decides what the CPU should do. It sends control signals to the ALU, memory, and registers based on the instruction. It also handles which register to use and when to read or write data.

* **Registers**

Registers are small storage units inside the CPU. They hold temporary data that the CPU is working with. For example, when adding two numbers, both numbers are taken from registers, and the result is also stored in a register.

The registers here are not accessible to the programmer directly. They are used to move and hold data during instruction execution.

### MEMORY

The MCU needs memory to store both the program and data. There are two main types of memory, RAM and ROM.

* **Program Memory (ROM)**

ROM stands for Read-Only Memory. It stores the program that the MCU runs. This memory does not change during operation. The CPU reads instructions from ROM using the Program Counter.

* **Data Memory (RAM)**

RAM stands for Random Access Memory. It is used to store variables and temporary data. RAM can be read and written during operation. When power is turned off, RAM is cleared.

### CLOCK

A clock signal is used to control how fast the MCU operates. Each instruction takes a certain number of clock cycles. In a simple design, the clock can be an external oscillator or a fixed internal signal. The clock ensures that the CPU and all parts work in sync.

### INPUT AND OUTPUT (I/O)

The MCU needs to communicate with the outside world using I/O ports.

* Input Ports

Used to read data to MCU.

* Output Ports

Used to send signals from MCU.

The I/O ports can be simple digital pins that are turned ON or OFF by the program. Some MCUs use memory-mapped I/O, special memory addresses are connected to I/O ports.

### PROGRAM COUNTER

The Program Counter keeps track of the current instruction address. It increases by one after each instruction so that the next instruction can be read from memory.

### STATUS FLAGS

1-bit register set that indicate the status of an operation e.g. when result is zero, when there is a carry/borrow or in the event of an overflow

### REGISTERS

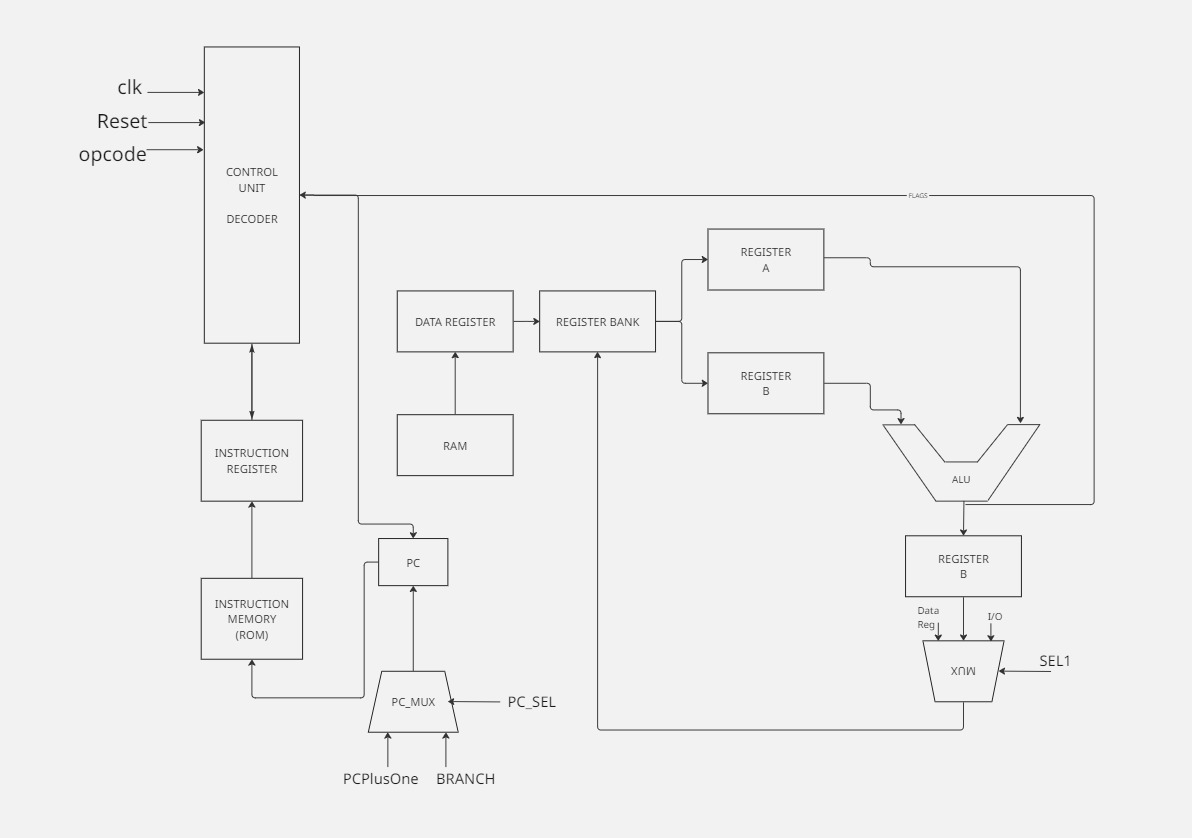
The MCU will have 8 general purpose registers R0 to R7 to hold data for program operations

To be able to design an MCU to the basic level, the above listed elements have to be developed.

We also need to develop an instruction set.

### 8-BIT PROCESSOR MCU BLOCK DIAGRAM

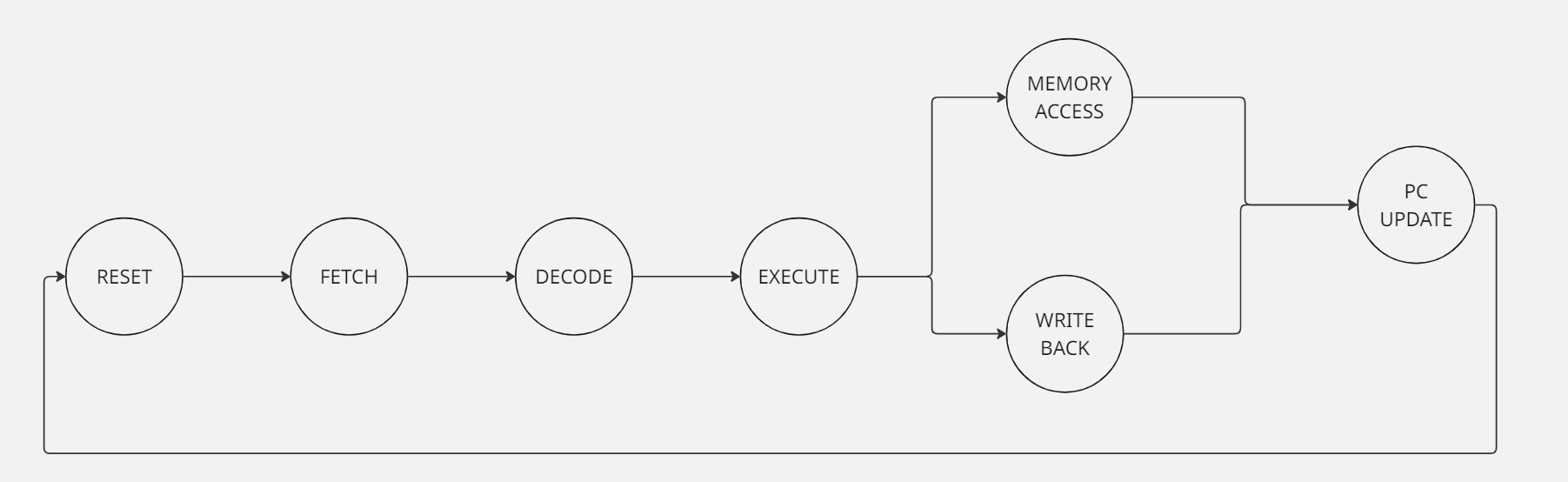
Below is a block diagram of the designed MCU



The CU unit implements our FSM approach.

### 8-BIT PROCESSOR MCU STATE TRANSITION DIAGRAM

Below is the state transition diagram.



In my design I identified the following 7 states in the operation of the MCU

* RESET - system initialization
* FETCH - get instruction from ROM
* DECODE - identify opcode and operands
* EXECUTE - perform ALU or prepare memory access
* MEMORY ACCESS - handle LOAD/STORE instructions
* WRITE BACK - write result to register file
* PC UPDATE - increment or load a new PC address

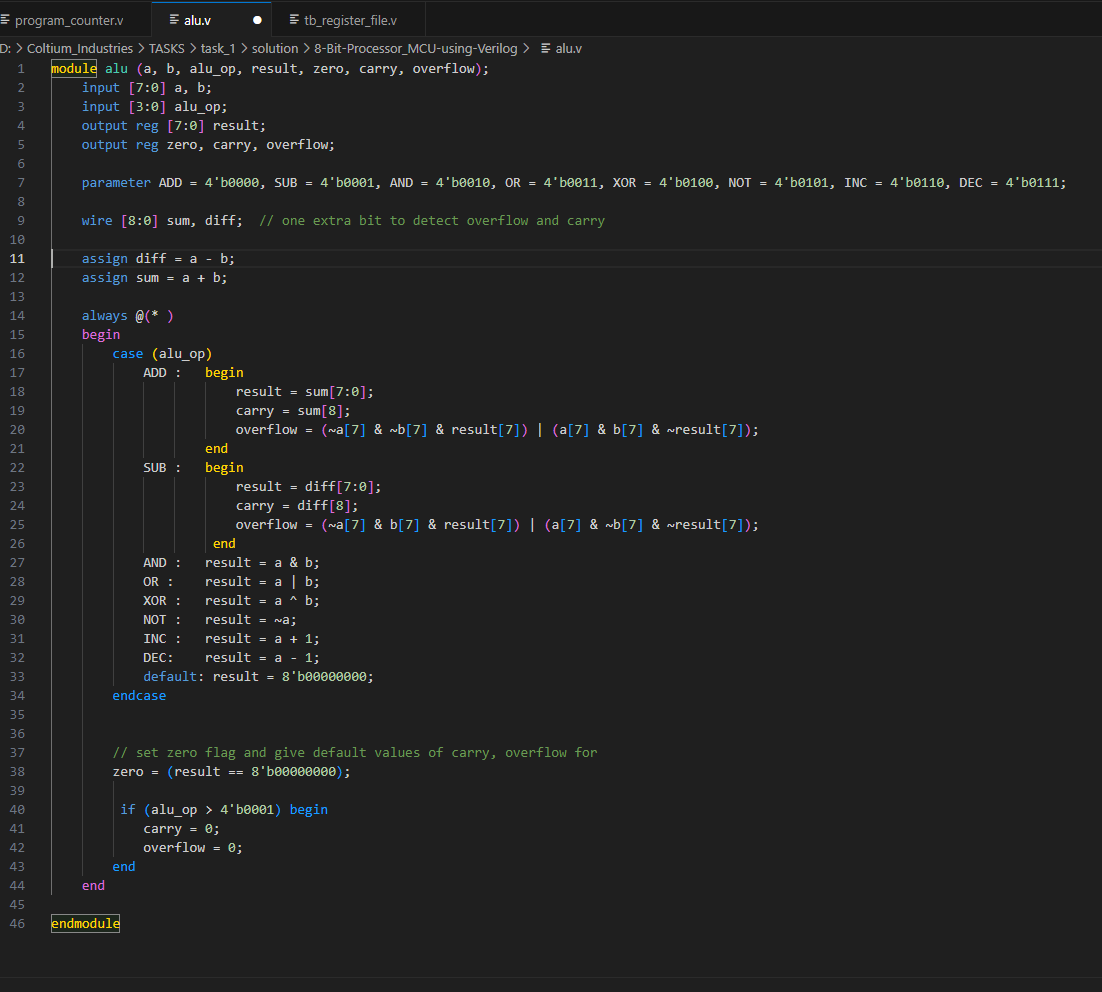
I used the following 19 modules to develop the MCU

* alu.v
* register\_file.v
* reg\_a.v
* reg\_b.v
* reg\_c.v
* program\_counter.v
* instruction\_memory.v
* instruction\_register.v
* decoder.v
* flags.v
* control\_unit.v
* mux1.v
* pc\_mux.v
* data\_register.v
* data\_memory.v
* io\_module.v
* datapath.v
* mcu\_top.v
* tb\_mcu\_top.v
* clock\_gen.v

Below are the code representation of each Verilog module

### alu.v

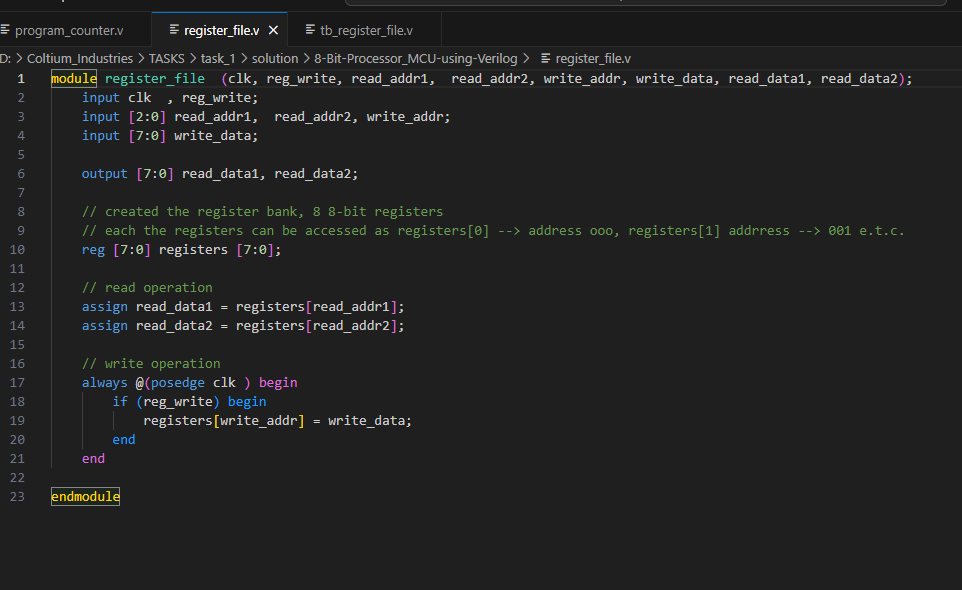
Below is the code written for this module



Below is the test results

### register\_file.v

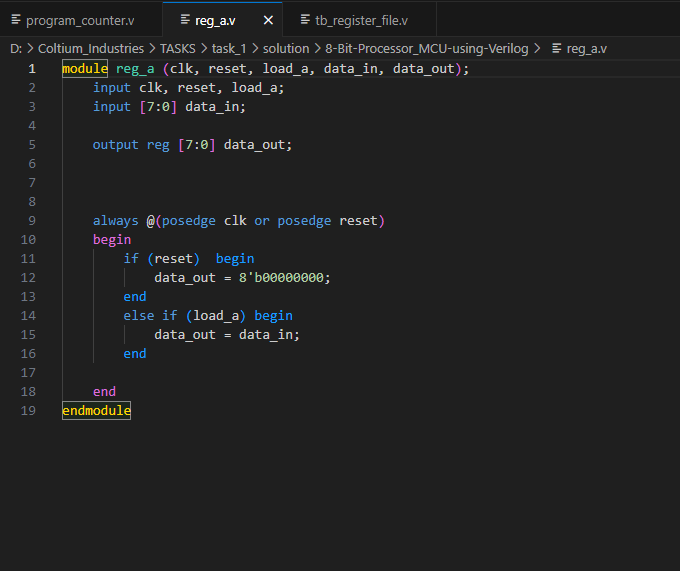
Below is the code written for this module



Below is the test results

### reg\_a.v

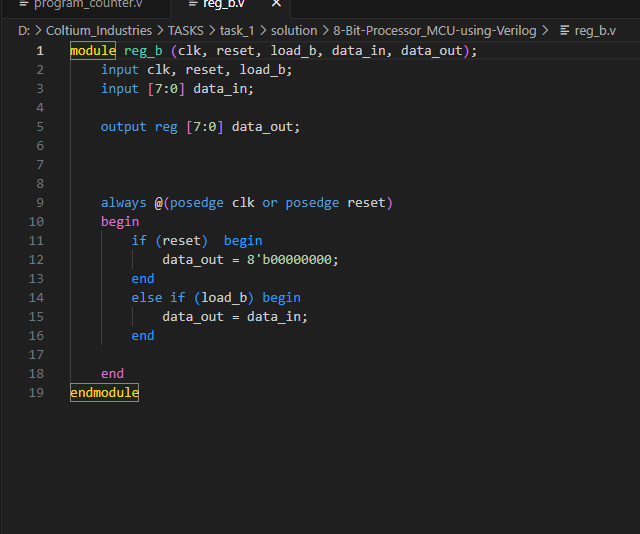
Below is the code written for this module



Below is the test results

### reg\_b.v

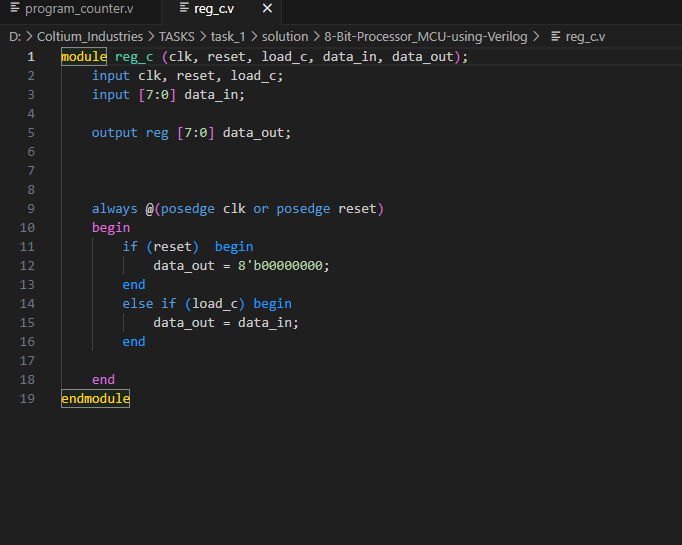
Below is the code written for this module



Below is the test results

### reg\_c.v

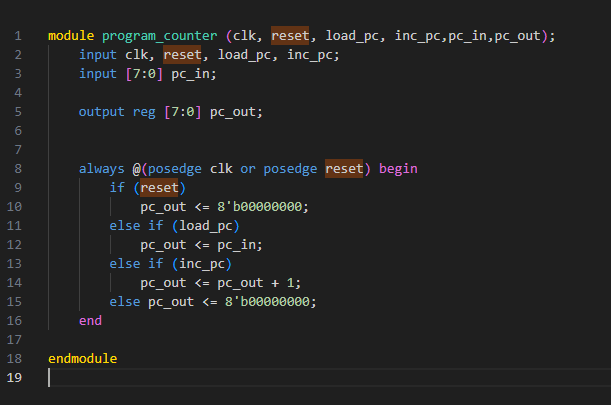
Below is the code written for this module



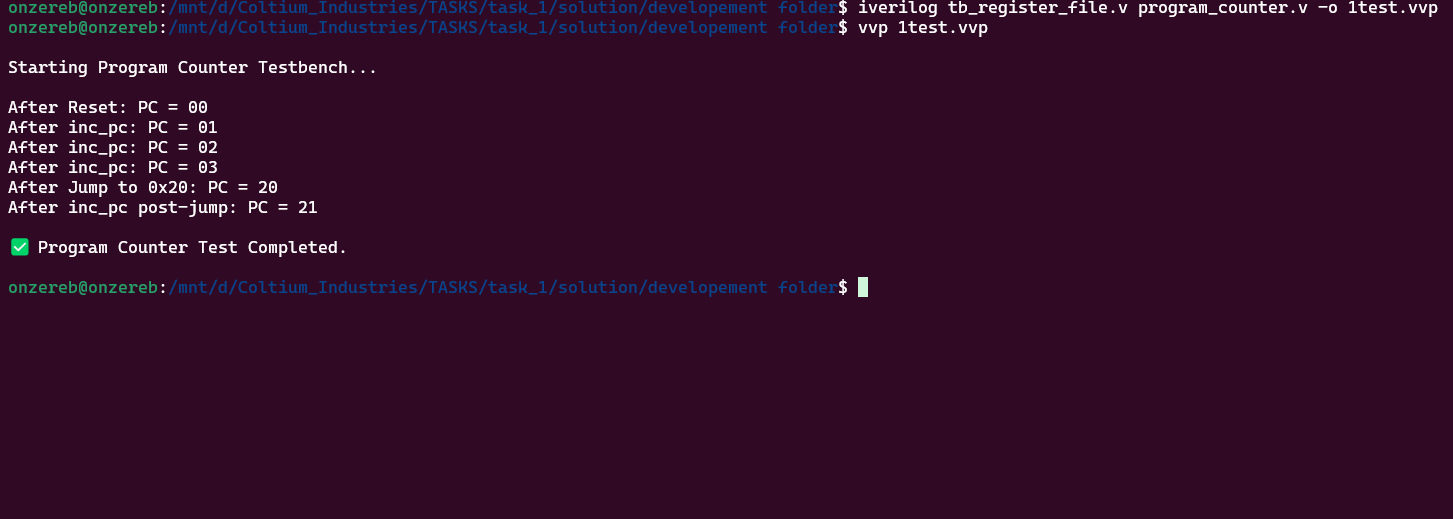
Below is the test results

### program\_counter.v

Below is the code written for this module

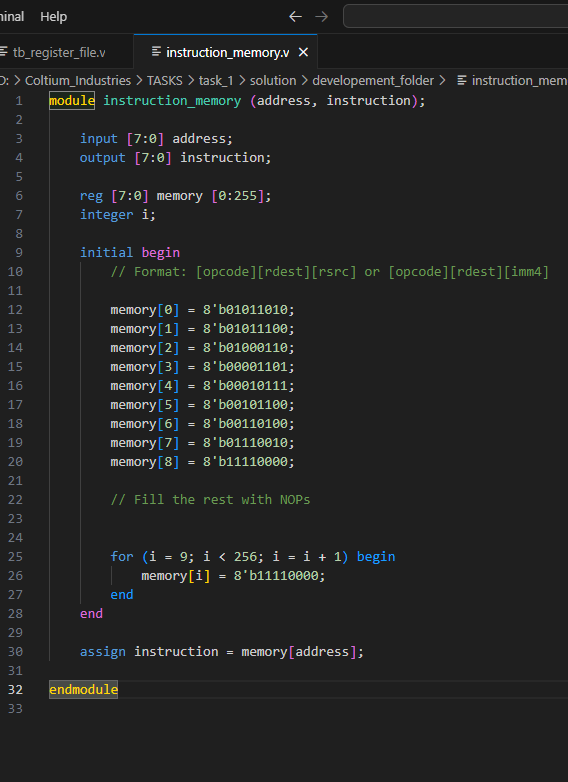


Below is the test results



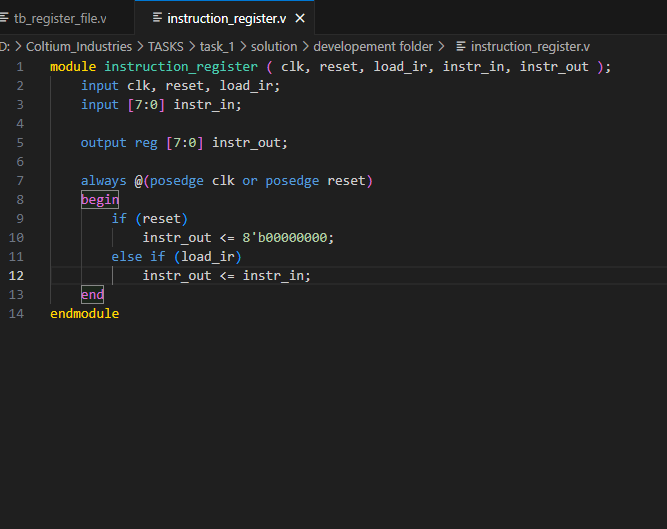
### instruction\_memory.v

Below is the code written for this module

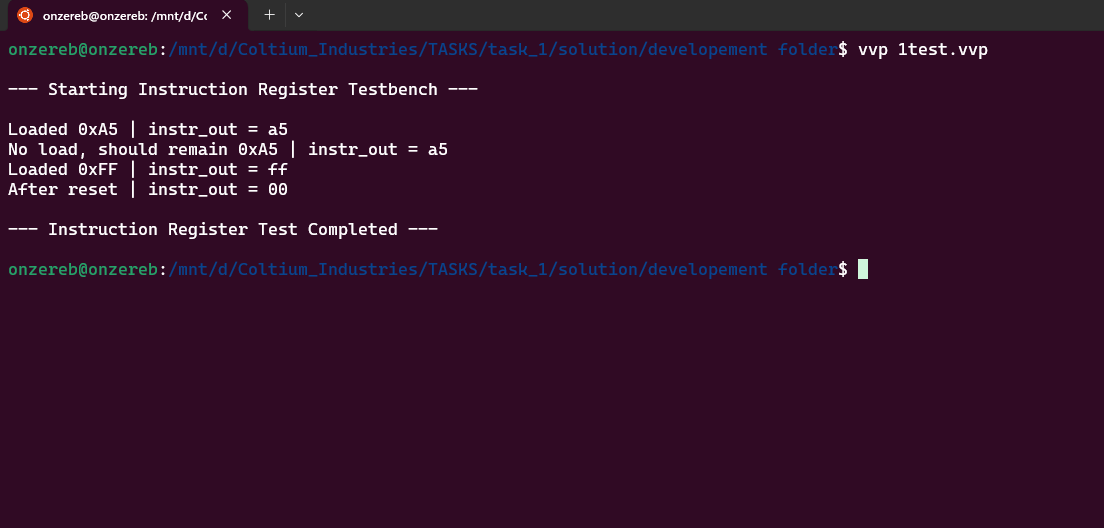


### instruction\_register.v

Below is the code written for this module

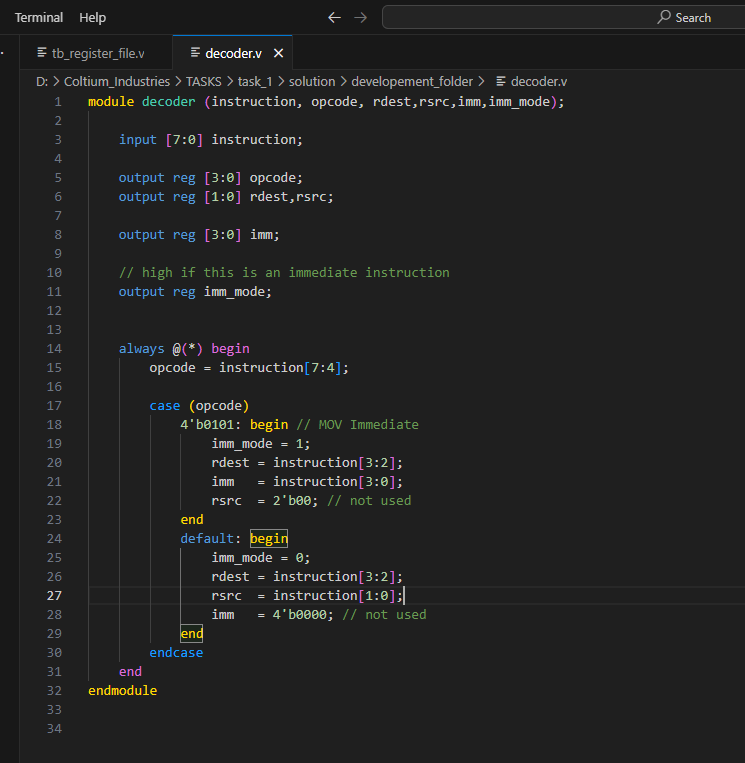


Below is the test results

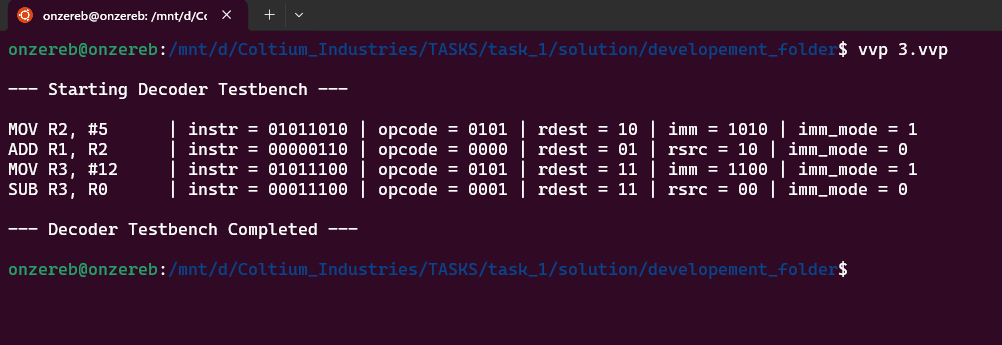


### decoder.v

Below is the code written for this module

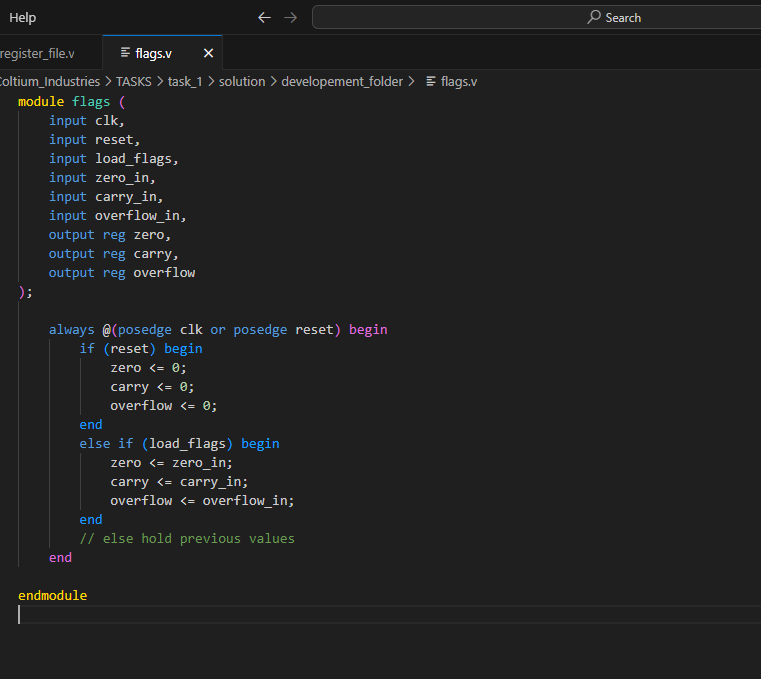


Below is the test results



### flags.v

Below is the code written for this module



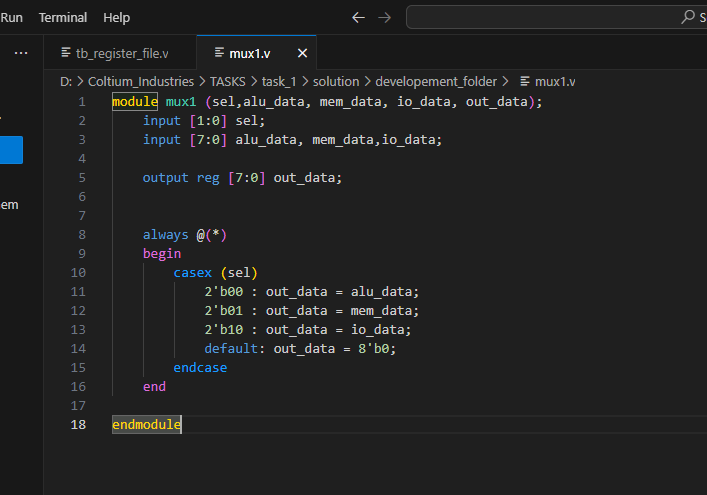
### control\_unit.v

Below is the code written for this module

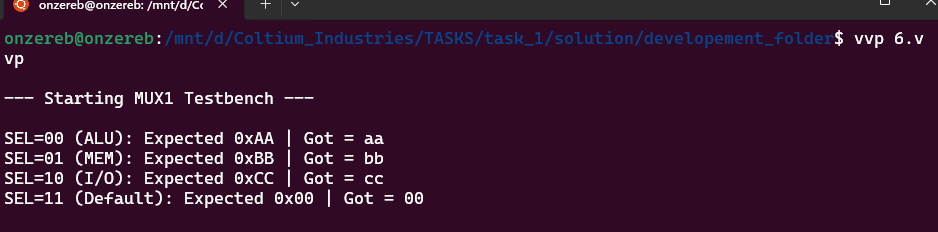
Below is the test results

### mux1.v

Below is the code written for this module

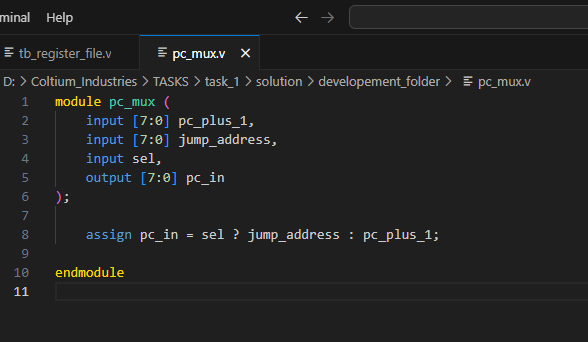


Below is the test results



### pc\_mux.v

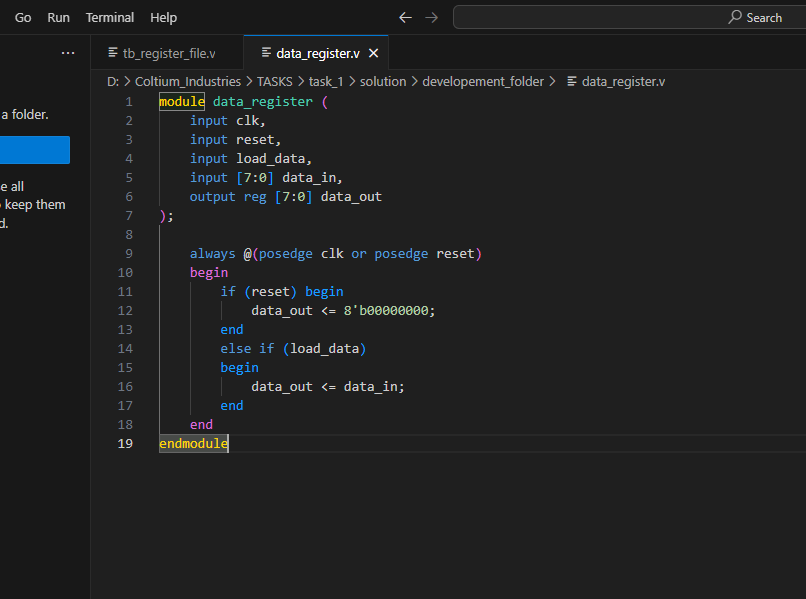
Below is the code written for this module



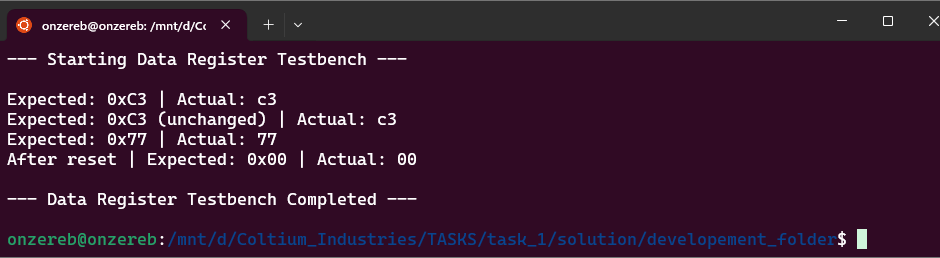
Below is the test results

### data\_register.v

Below is the code written for this module

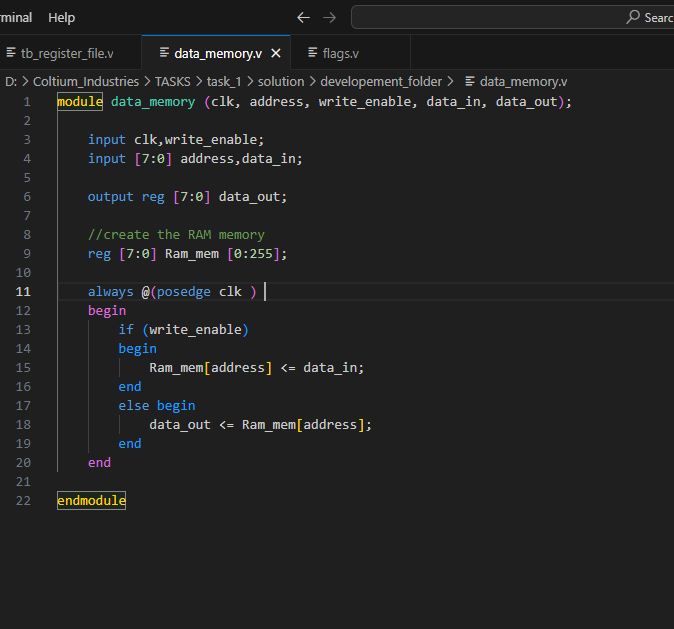


Below is the test results



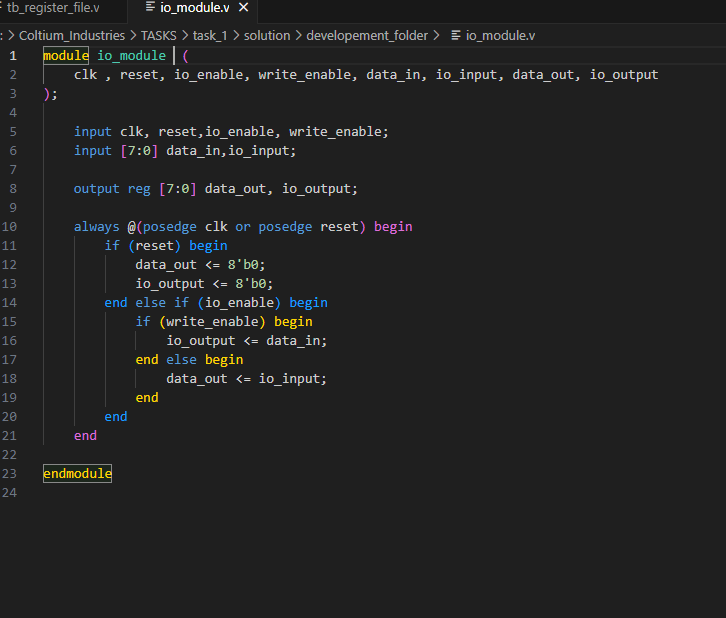
### data\_memory.v

Below is the code written for this module

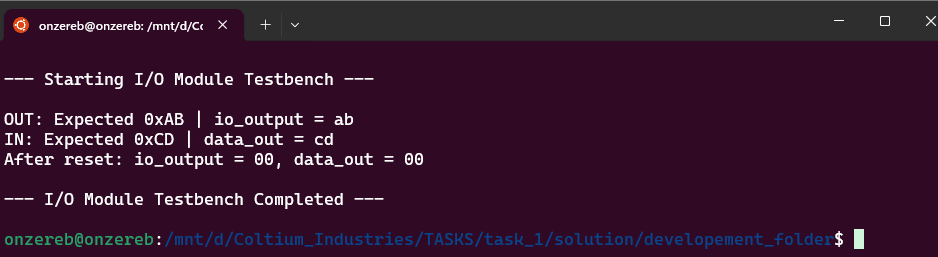


### io\_module.v

Below is the code written for this module

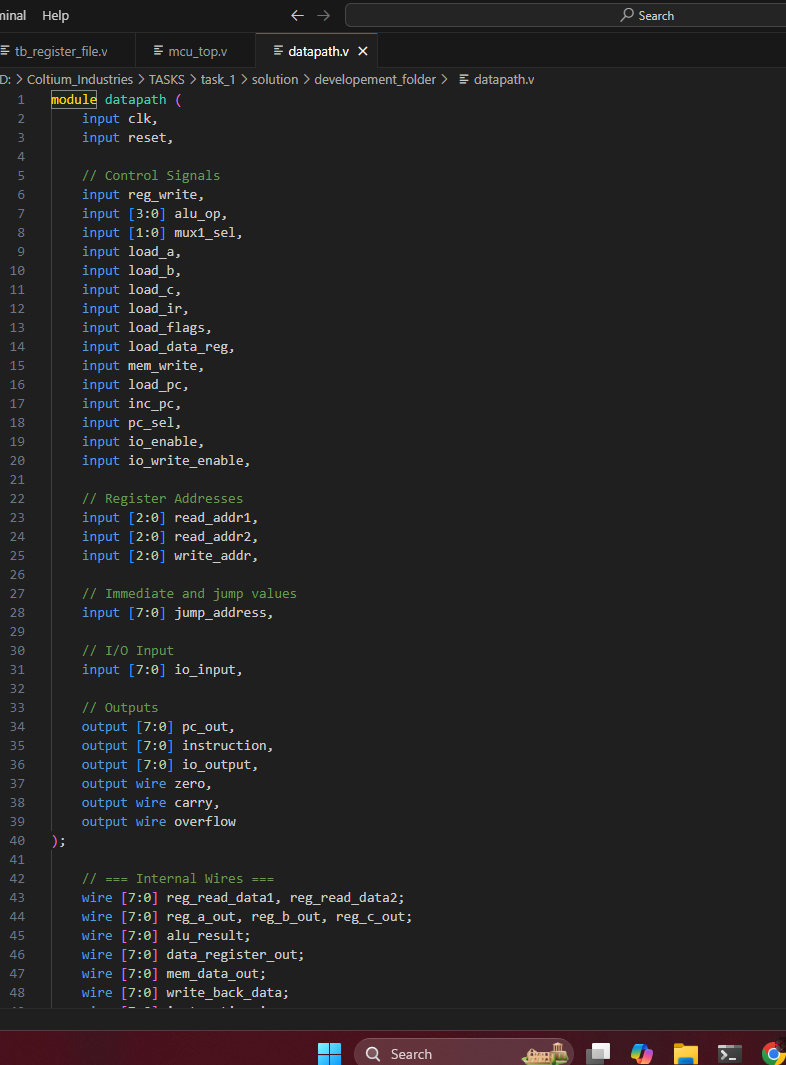


Below is the test results



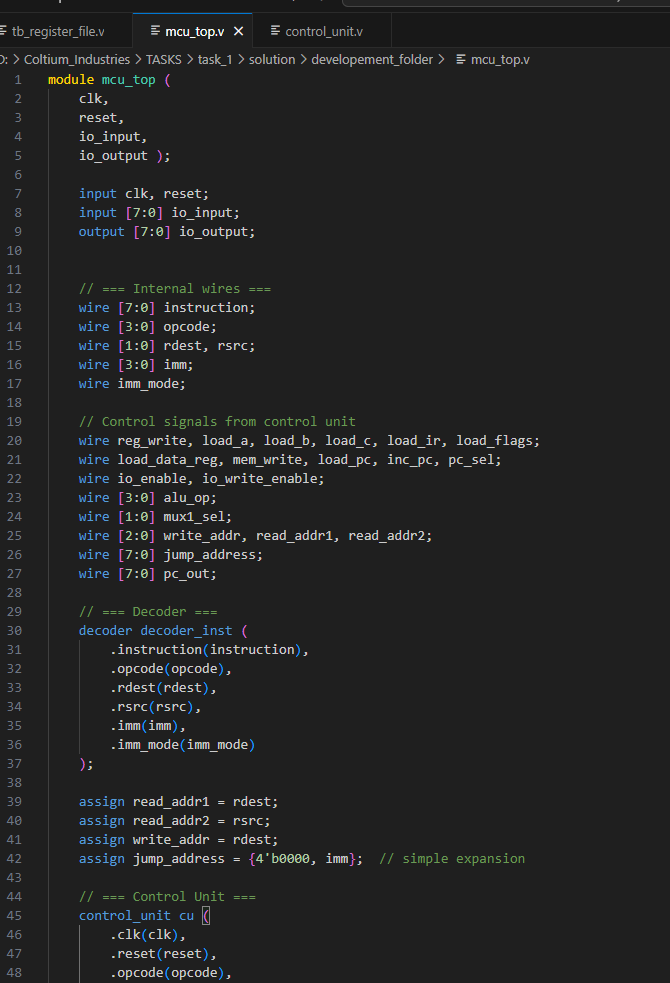
### datapath.v

Below is the code written for this module



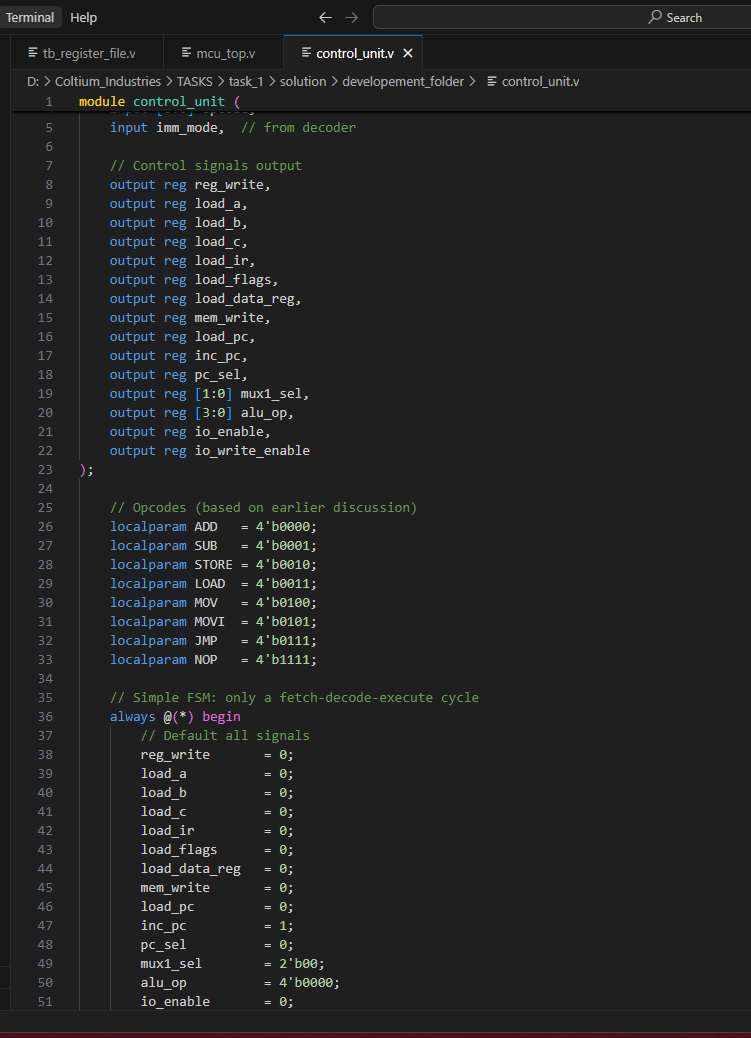
### mcu\_top.v

Below is the code written for this module



### Control\_unit.v

Below is the code written for this module



### clock\_gen.v

Below is the code written for this module

# Pending

The following modules are pending development:

* datapath.v
* mcu\_top.v
* clock\_gen.v

The following tasks are also pending:

* COMBINED SIMULATION
* SYNTHESIS