RISC CPU

For this project, we had to implement a simple yet functional CPU with the major elements of a general-purpose computer within a span of two weeks. The project involves adhering to a specified instruction set and having the leeway to try out personalized CPU architectures and machine languages. The project requires the establishment of RAM and ROM memory files, creation and verification of an ALU, defining a CPU entity as a 9-state state machine, and assembling and verifying all components.

ALU Top Level Design:

The ALU, contained in the "alu.vhd" file, is an asynchronous circuit and operates under four condition bits. The Zero Condition (cr(0)) is set when an operation results in all zeros in bits 15 down to 0. The Arithmetic Overflow Condition (cr(1)) is set on addition or subtract when the two operands are the same sign, and the result is of different sign, indicating an overflow. The Negative Condition (cr(2)) is set when the sign bit of the result (bit 15) is '1' and the result is considered negative. The Carry Out Condition (cr(3)) is set after either a bit shift out of the input or a carry-out from an arithmetic operation, with tdest(16) containing the carry bit. This method of design indicates the ALU indicates separation and distinction of the ALU's different operation states and outcomes, allowing for reliable computation from the ALU.

CPU: Top Level Design

The CPU's top-level design in the "cpu.vhd" file is based on RISC architecture and contains registers A-E; Stack Pointer; Memory Buffer Register (MBR); Instruction Register; Program Counter (PC) and a 4-bit Condition Register. The construction of the CPU depends on the ALU circuit as two 16-bit source registers and an opcode input to the ALU create a 16-bit destination value and condition flags for the CPU. The CPU contains a nine-state machine comprised of Start, Fetch, Execute-Setup, Execute-ALU, Execute-MemoryWait, Execute-Write, Execute-ReturnPause1, Execute-ReturnPause2, and Halt states. The Start state powers-up the CPU, waits, then switches to the Fetch state to fetch instructions and update the PC. In Fetch, the CPU fetches the opcode from memory and updates the Program Counter. The Execute-Setup goes to status fetching and setup to carry out the instructions, then will transition the CPU to: the Execute-ALU state to process the ALU operation; or the Execute- MemoryWait for a Waitfor instruction use; or the Execute-Write if data storage was requested and return to Fetch for most instructions. In the case for AFUNCTION with a RETURN instruction, a Execute-ReturnPause1 and a second Execute-ReturnPause2 state will be carried out before looping back to the Fetch state. Finally, the Halt state stops the CPU until it is reset. As described here in the effectively state transitioned CPU will carry out instructions and properly generating storing data inside the CPU memory/registers.

Below are the required Gtkwave forms for the required testing. Each diagram is labelled as required.

All of the images are also attached in the zip file

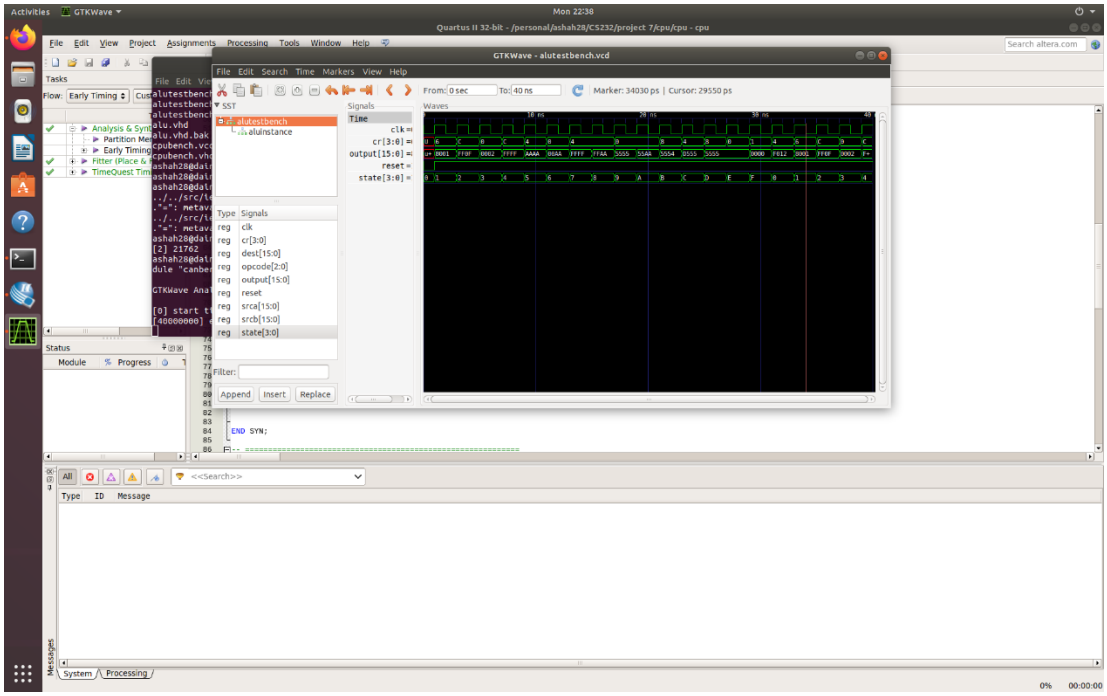

Figure: gtkwave for alutestbench.vhd

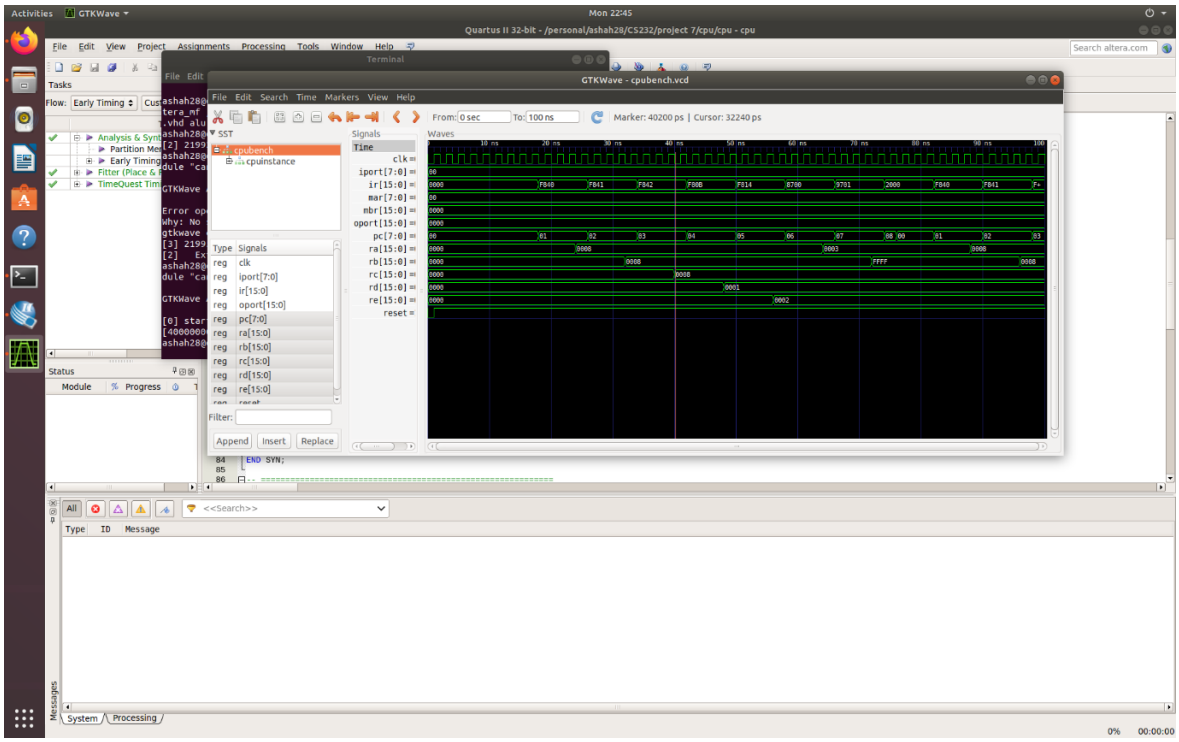

Figure: gtkwave for cpubench.vhd (1/4)

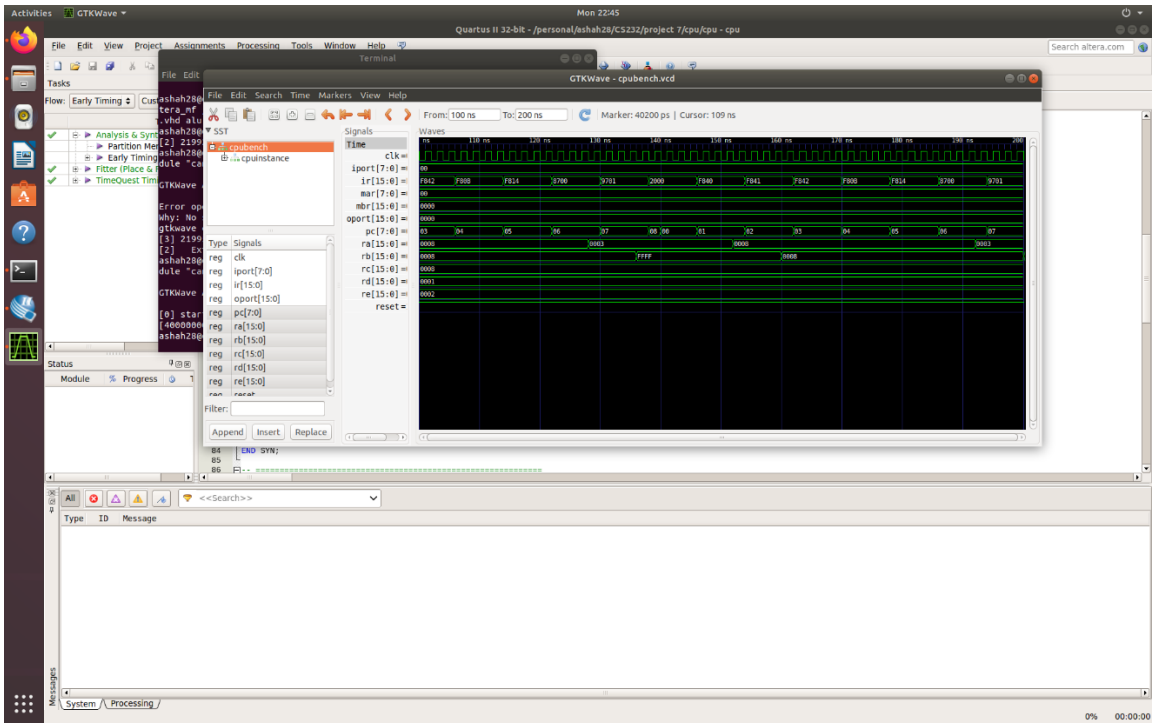

Figure: gtkwave for cpubench.vhd (2/4)

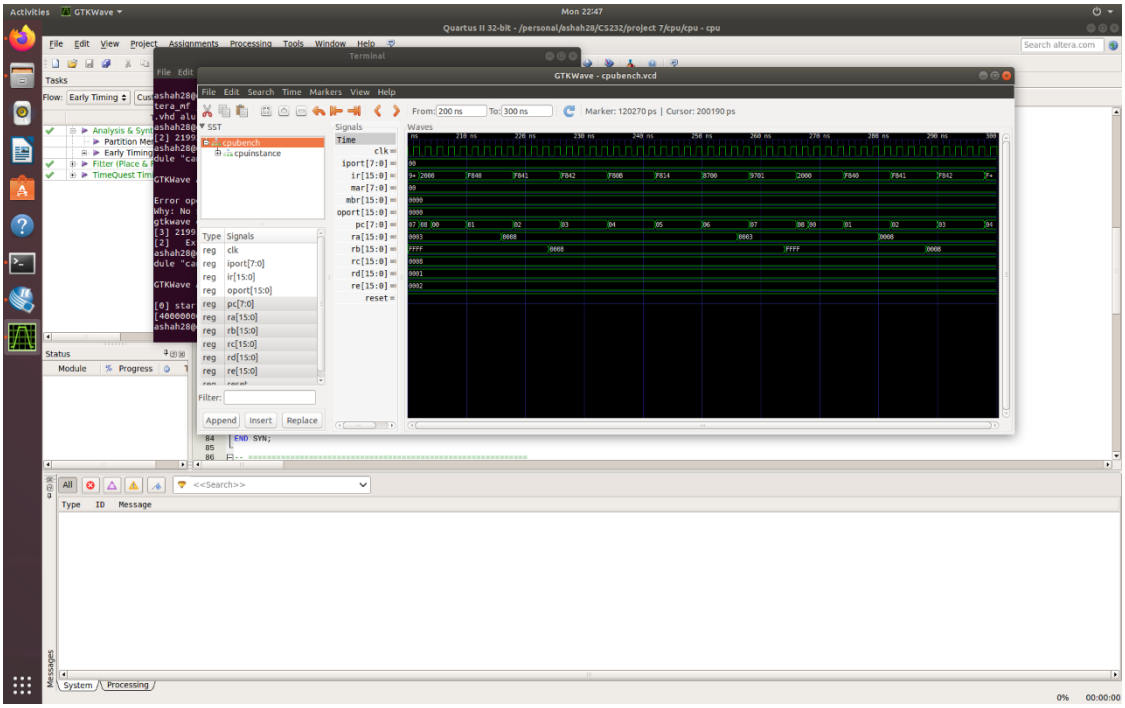

Figure: gtkwave for cpubench.vhd (3/4)

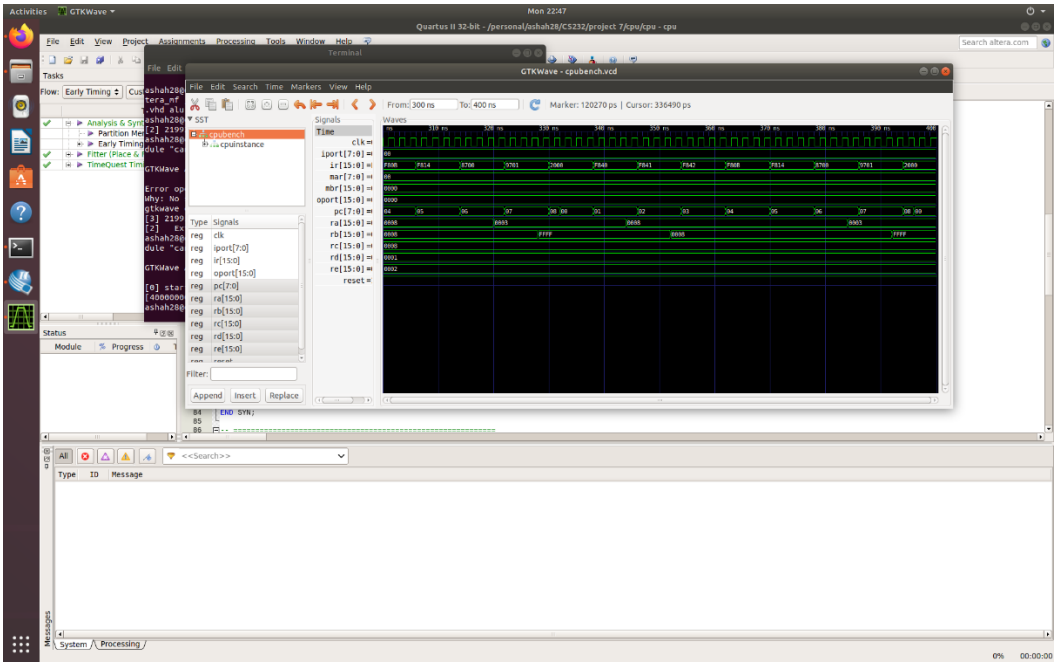

Figure: gtkwave for cpubench.vhd (4/4)

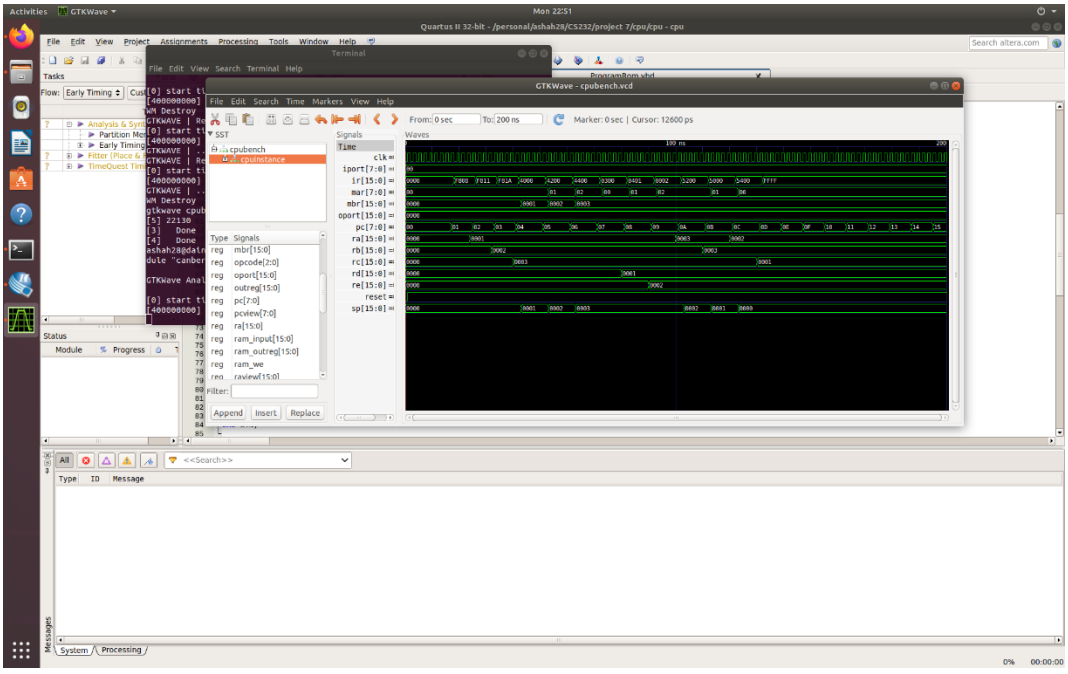

Figure: gtkwave for testpush.mif

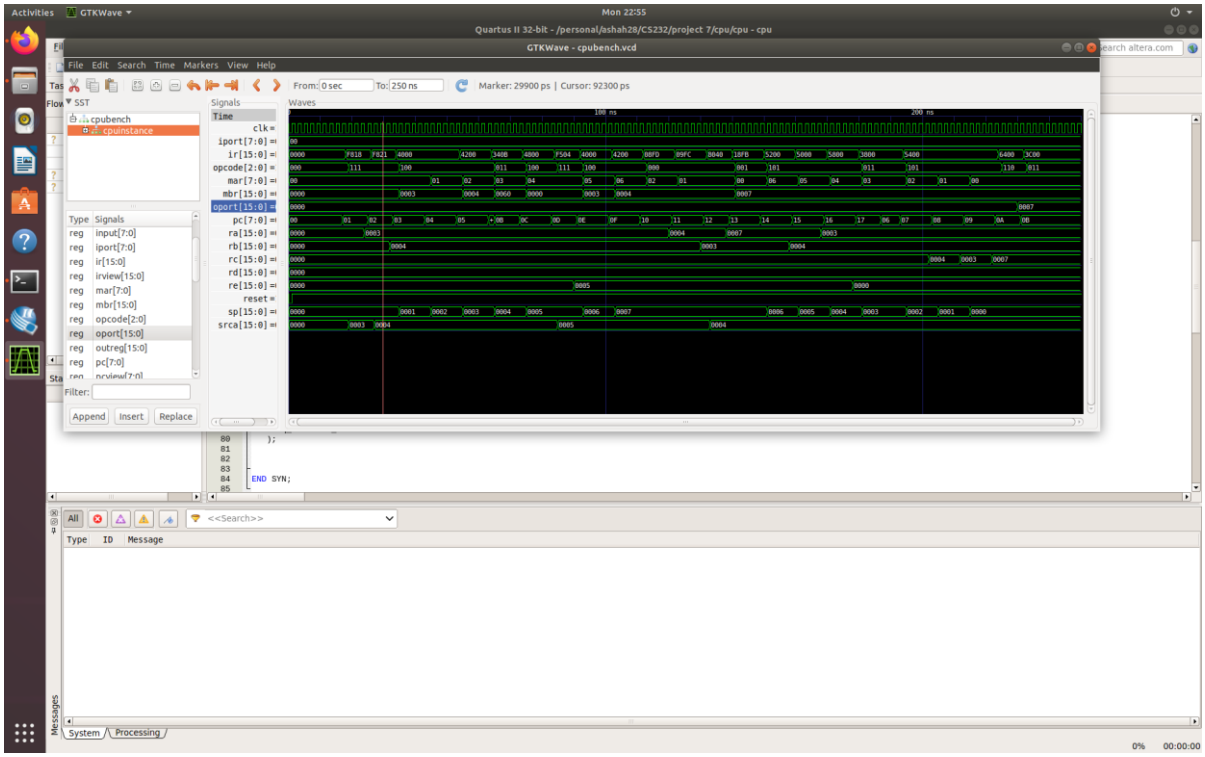

Figure: gtkwave for testcall.mif

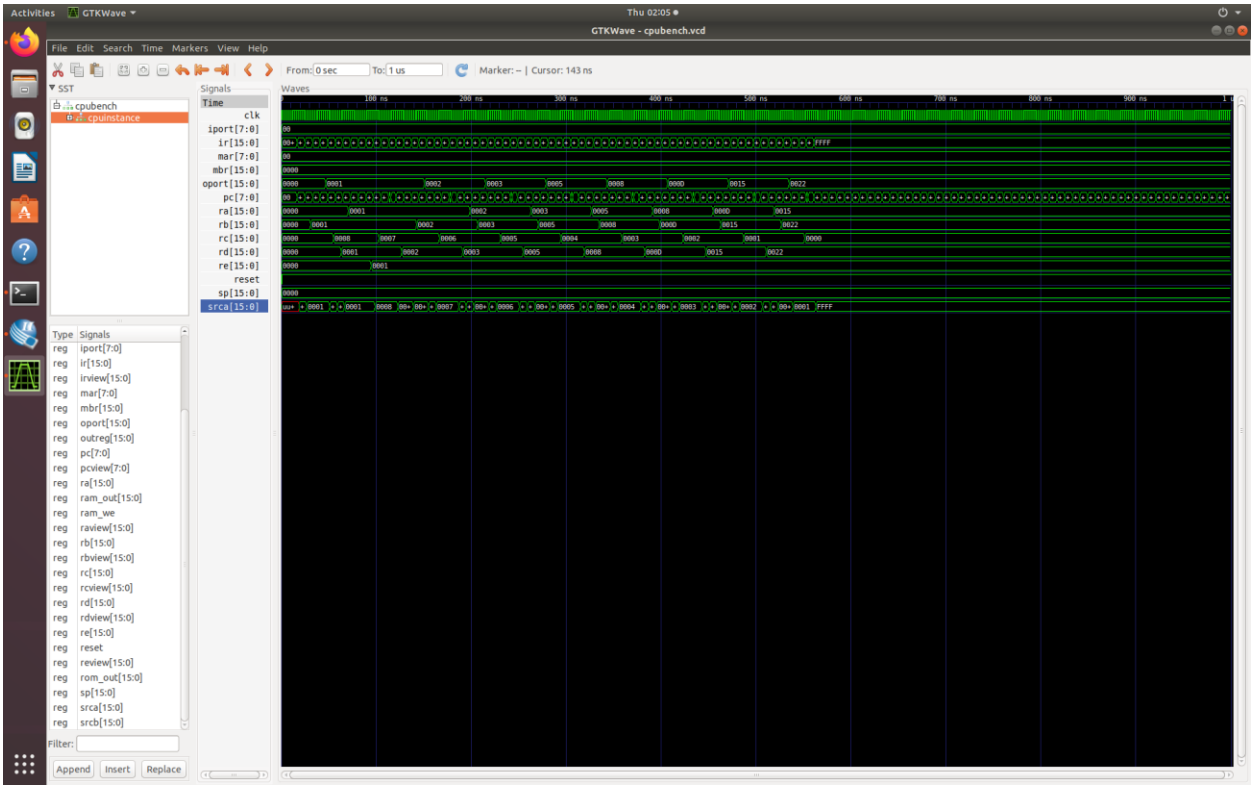

Figure: gtkwave for testfibo.mif

Here, the output of the Fibonacci series is presented in the output port in hexadecimal format.

Acknowledgements:

This project would not have been possible without the help of amazing classmates Azam, Maya, Aleksandra, online articles and forums such as Reddit and Stack Overflow and Professors Dr. Stephanie and Dr. Tahiya Chowdhury, who provided us with their expertise, guidance, and support.