Project Report – RISC-V (v0.1)

Introduction:

This project aims to create an RTL- design of "**single cycle RISC-V**" core using Verilog HDL.

This project implements 5-types of instructions from RISC-V ISA:

1. **R-Type**: Register instructions, that operate on the core's registers.
2. **I-Type**: Immediate instructions, that also include immediate operands along with registers in their operations.
3. **B-Type**: Branch instructions, that are used branching (conditional) to different parts of the program being executed.
4. **L-Type**: Load instructions, that are used to load data from the data-memory to the core's registers.
5. **S-Type**: Store instructions, that are used for storing the data from the core's registers to the data-memory.

Details of Implementation:

The implementation is mainly divided into two main sections that realise the functionality of the core that I built.

1. **Datapath:** This section involves the data processing and storing elements present in the core. This mainly defines the modules that are present in the core along with their arrangement thereby defining how the data flows in the core.

   The elements in the core are as follows:
   1. **Program Counter (PC):** It is a 32-bit wide register that contains the address of the instruction that is to be fetched. (no separate module defined; it is implicit to Datapath).
   2. **Immediate Generator:** It is used to Sign extend or Zero extend the immediate data decoded from the instruction fetched.
   3. **Register Bank:** This is a collection of 32 registers each registers 32-bit wide that are used for temporary storage of the data being processed.
   4. **Arithmetic Logic Unit (ALU):** This performs various Arithmetic and Logical operations the operations performed are provided in the later part of the report.
   5. **Data Memory*:** This is the element of the Datapath that stores the all the data processed and unprocessed.
   6. **Instruction Memory*:** This is another storage element that stores the instructions present in a computer program that needs to be executed.

*These elements are not truly present inside the core hence they are not included the Datapath module of the project which only includes the elements present in the core. But still they are an integral part of the Datapath hence they are instantiated in a separate top module.

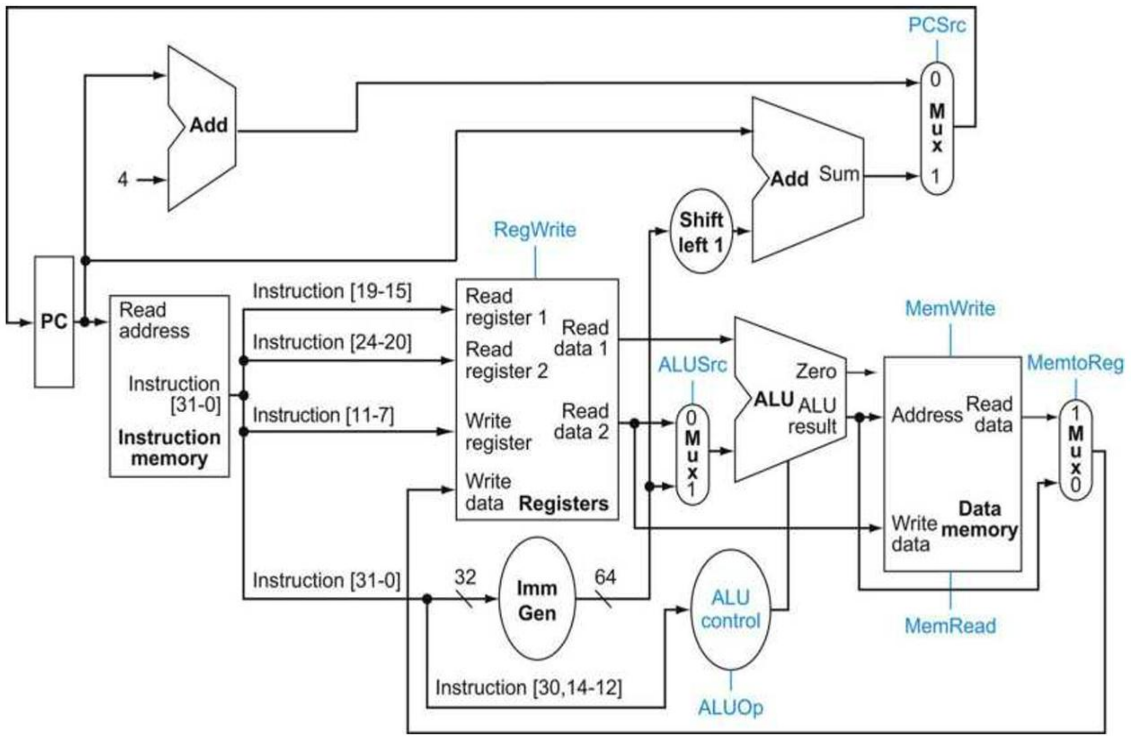

Fig1. Datapath of the RISC-V core single cycle. Ref from Computer-Organization-Design-RISC-V-Architecture

2. **Control Path/Unit:** This section contains the control logic for controlling the execution of the instructions of types mentioned above. It produces various control signals that control the functioning of the various elements present in the Datapath.

   The control signals used in this implementation are as follows:

   1. **write_ctrl:** This control signal when enabled allows write operation into one of the registers in one clock cycle.
   2. **operand_ctrl:** This signal controls whether the second operand for ALU (when logic 0) is from the immediate generator (when logic1) or one of the registers.
   3. **load_ctrl:** This signal controls the writeback to the register. If logic 1, writeback occurs from data fetched from memory. If logic 0, writeback occurs from the ALU result.
   4. **branch_flag:** This signal is set high when branch instructions are to be executed.
   5. **mem_read:** This signal is set when data is to be read from memory (load instructions).
   6. **mem_write:** This signal is set when the data is to be written into the memory (store instruction).
   7. **ctrl_op:** This is a 4-bit wide signal bus that carries the decoded and re-encoded signal that tells ALU what operation to perform.

To most extent I have made the code self-commenting. This document aims to provide the information regarding the Signals used in the design so that it becomes easy to recreate or understand the implementation.

Tools used:

1. Xilinx Vivado for simulation and synthesis.
2. VS code for editing the code.

The following image shows the synthesis result of the design all connected together in the "top_module".

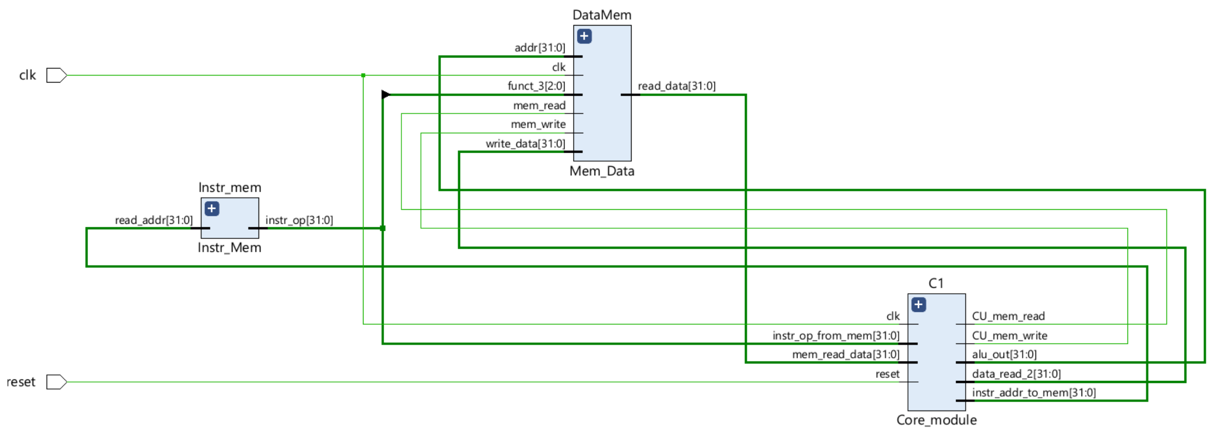

Fig 2. Synthesis result of the Top module

- In the above synthesis result the **Core_module** is the module that encapsulates both the **Control path and Data path**.

The following image shows the expanded schematic of the Core_module.

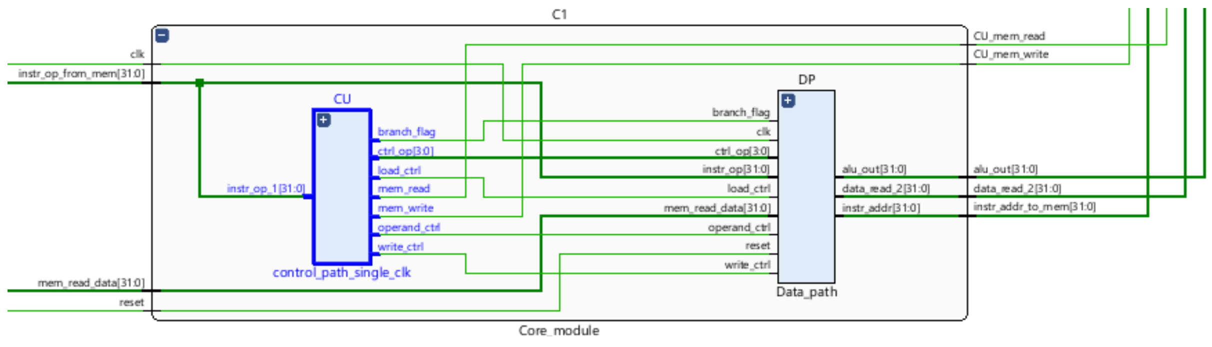

Fig 3. Expanded Schematic of Core_module.

- The above image shows all the signals shared between the **Control Unit and Data path.**

The following image shows the expanded schematic of the Control unit.

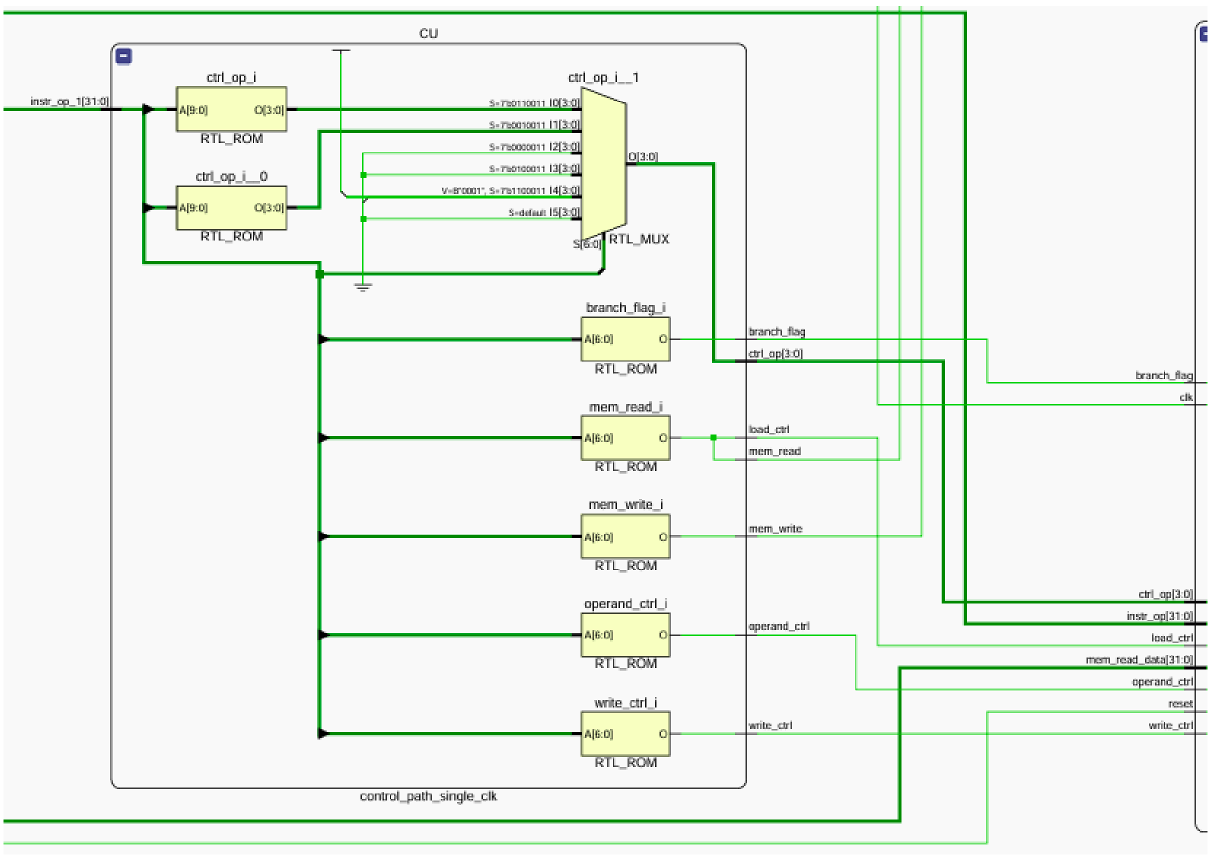

Fig 4. Expanded Schematic of Control Unit.

- The above image shows the logical realisation of the Control logic for facilitating the execution of all the instructions mentioned above.

The following image shows the expanded schematic of the Data path.

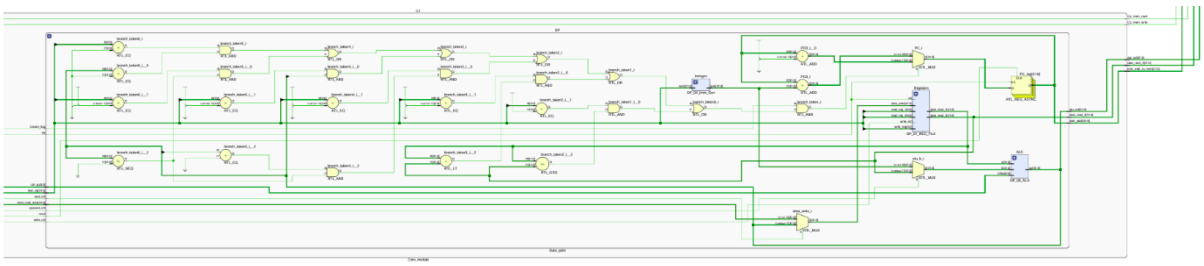

Fig 5. Expanded Schematic of Data_path.

- The above image shows the RTL logic realisation of the various Data path elements and their interconnection.

References:

- PDF file of the Schematic in the GitHub repository.
- Instruction Set Architecture (ISA) reference PDF file in the GitHub repository.
- For Textbook reference refer to "Computer-Organization-Design-RISC-V-Architecture".

Thank You for reading.

Regards CHANNABASAVA H