Pipelined RISC-V Implementation

Project Mentors : Keerthi Bhushan M, Ajinkya Galegave, Srinath Seshadri

Team Members : Pratik Sabne, Chinmay G S, Ashna Gaude, Aneesh Bharadwaj, Rishab Bathwal

Introduction

Our project proposal focuses on creating a pipelined RISC-V architecture using Verilog. RISC-V is an open-standard instruction set architecture, and pipelining enhances microprocessor performance. The pursuit of optimal performance and efficiency is an ongoing endeavour in the field of computer architecture. New methods are needed to solve these issues as the need for processors that are faster and more power-efficient keeps growing. The goal of this research is to achieve new levels of efficiency and performance by optimising pipelining techniques inside the architecture of the RISC-V instruction set. Being an open-source ISA, RISC-V offers a conducive environment for research, experimentation, and teamwork. This research aims to improve our knowledge of computer architectural principles and provide useful solutions that can improve the performance of RISC-V processors in a variety of applications by exploring the complexities of pipelining in the context of RISC-V.

Hence our project statement is to implement pipelined RISC-V and identify the key challenges, bottlenecks, and inefficiencies that hinder its operations, with the goal of proposing recommendations and strategies to enhance the efficiency of operation.

Inspiration

Primarily, it offers an opportunity for a deep dive into computer architecture, appealing to those interested in understanding the fundamental workings of computers. Given RISC-V's status as an open-source instruction set architecture, engaging with such projects provides access to a vibrant community and fosters collaborative learning. Additionally, the educational value is significant, as it allows students and enthusiasts to apply theoretical knowledge to practical implementations, solidifying their understanding of computer architecture principles. Moreover, with RISC-V gaining traction as an

alternative to proprietary ISAs, involvement in RISC-V projects can enhance one's industry relevance, potentially leading to career opportunities in computer architecture or related fields. Beyond that, the challenges inherent in optimising pipelining techniques for RISC-V processors provide ample opportunities for problem-solving and innovation, contributing to both personal growth and the advancement of the broader computing community.

Literature Review

1. RISC - V

RISC-V is an open-source ISA (Instruction Set Architecture) that has fostered innovation in computer architecture. The ISA is defined with a base integer instruction set, divided into several standard extensions catering to diverse application domains. Its instruction set comprises a small number of instructions, which simplifies both hardware implementation and software development. Furthermore, RISC-V supports both 32-bit and 64-bit address spaces, accommodating a wide range of memory configurations. Over the years, RISC-V has gained traction across various industries, with companies like SiFive, Western Digital, and NVIDIA leveraging the architecture in their products. Additionally, educational institutions worldwide have embraced RISC-V for teaching computer architecture and fostering innovation. It was constructed in a way that it reduces the hardware needed for minimum implementation. It follows a little endian format, where the lowest address contains the LSB part of the particular word. RISC V was mainly chosen because it can be easily pipelined and consumes less hardware and power.

2. Pipelining

The process of gathering instructions from the processor through a pipeline is known as pipelining. It provides for the systematic storage and execution of instructions. Pipeline processing is another name for it. Pipelining increases the overall instruction throughput. In the pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and a combinational circuit performs operations on it. The output of the combinational circuit is applied to the input register of the next segment. All objects go through the same stages and there is no sharing of resources between the stages. The propagation delay through all pipeline stages is equal and the

scheduling of an object entering the pipeline is not affected by the objects in other stages. Pipelining doesn't help latency of a single task, it helps the throughput of the whole workload. Multiple tasks operate simultaneously using different resources. However, pipelining introduces challenges such as hazards, including data hazards, control hazards, and structural hazards, which can lead to stalls or incorrect execution if not managed properly. Techniques such as forwarding, branch prediction, and hazard detection and resolution are employed to mitigate these hazards and optimise pipeline efficiency.

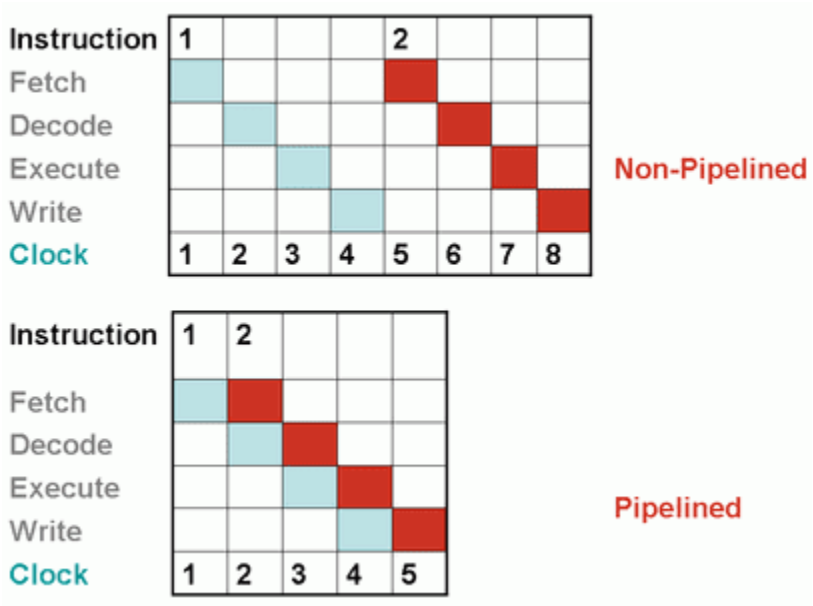

Fig 1 : Non-Pipelined and Pipelined Instruction Cycle for 4 Instructions

Stages of RISC-V Pipelining

This RISC processor design has been constructed using five pipeline stages. The used pipeline stages are the Instruction Fetch stage (IF), Instruction Decode stage (ID), Execution stage (EX), Memory Access stage (MEM) and Write Back stage (WB). Pipeline registers or latches are used to separate the stages of the processor into 5 parts, so there is no contradictory data due to the execution of multiple instructions. Other blocks include instruction memory (IR_MEM), Data memory (DATA_MEM), and General purpose registers. In order to pipeline, we separate the datapath into 5 discrete stages, each completing a different function and accessing different resources on the way to executing an entire instruction.

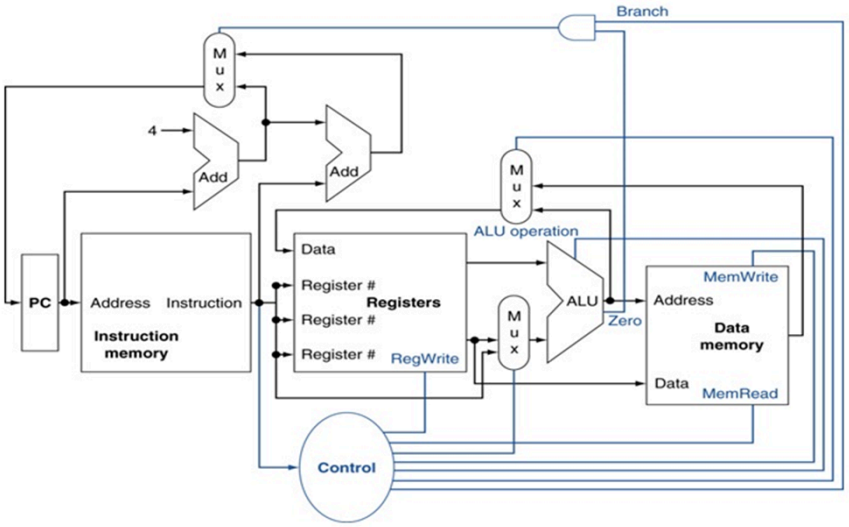

Fig 2 : Unpipelined RISC-V processor

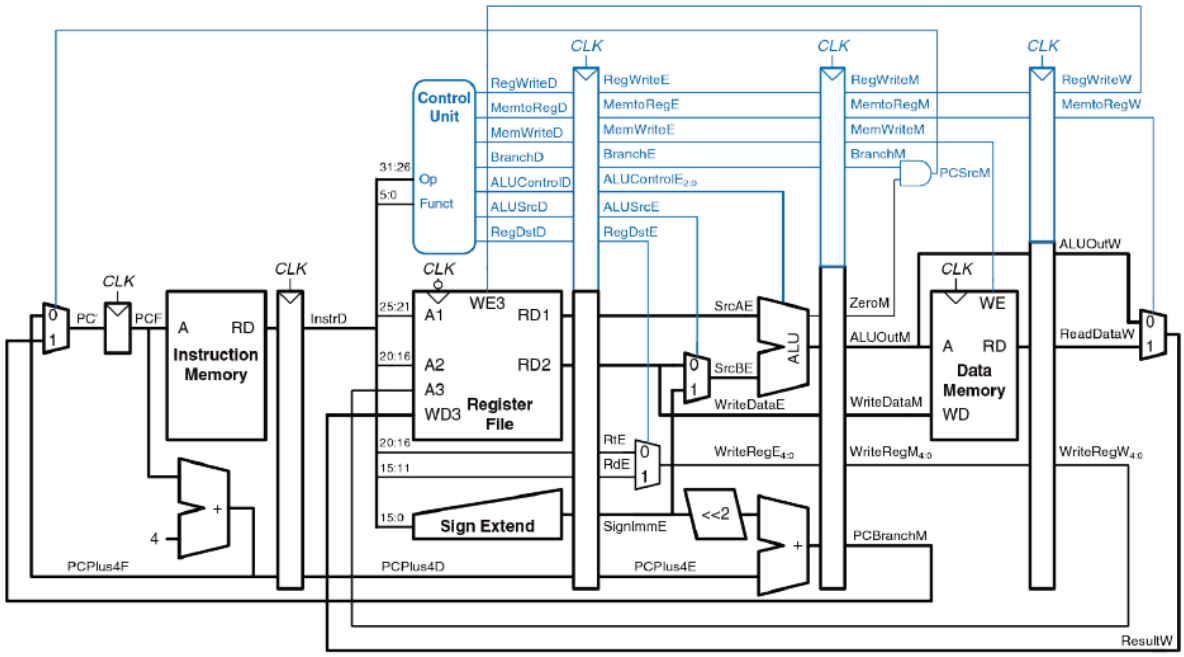

Fig 3 : Pipelined RISC-V processor

The stages in order are as follows :

1. Fetch Cycle
2. Decode Cycle
3. Execute Cycle
4. Memory Cycle
5. Writeback Cycle

1. Instruction Fetch Cycle

In the RISC-V architecture, the instruction fetch cycle is a fundamental stage in the execution pipeline, responsible for retrieving instructions from memory for subsequent processing. At the beginning of the fetch cycle, the program counter (PC) holds the address of the next instruction to be fetched. The PC is then used to access the instruction memory, typically implemented as a cache or directly connected to the processor. The instruction memory returns the instruction located at the address specified by the PC. In RISC-V, instructions are fixed-length, typically 32 bits or 64 bits, making it straightforward to fetch and decode them. Once the instruction is fetched, the PC is updated to point to the address of the next sequential instruction, preparing for the next fetch cycle. Efficient handling of the instruction fetch cycle is essential for overall processor performance, as it sets the stage for subsequent stages of instruction processing, including decoding, execution, and memory access.

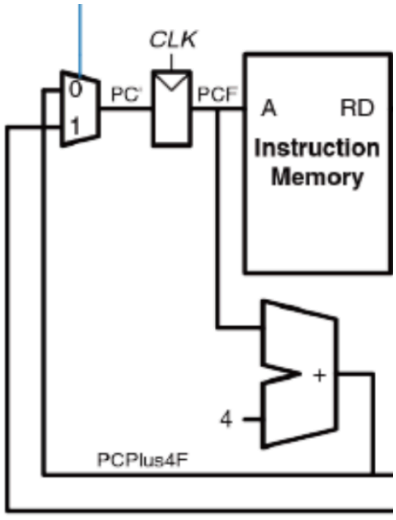

Fig 4 : Instruction Fetch Cycle

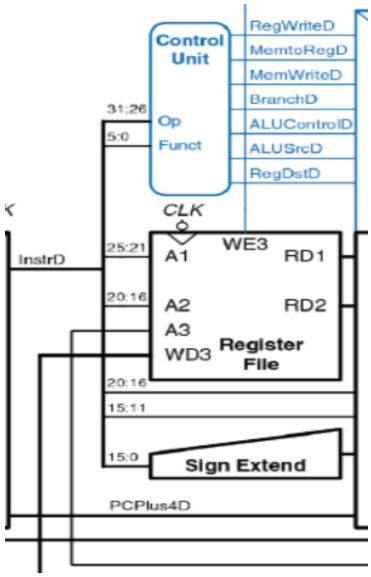

Fig 5 : Decode Cycle

2. Decode Cycle

The fetched instruction is interpreted and decoded to determine the operation to be performed. During this phase, the opcode and operands of the instruction are extracted, and any necessary control signals are generated to facilitate the subsequent stages of the pipeline. RISC-V's simplified instruction set architecture contributes to the efficiency of the decode cycle, as each instruction is designed to have a fixed format, making decoding relatively straightforward. Additionally, RISC-V employs a uniform encoding scheme across instructions, further

simplifying the decoding process. Moreover, the decode cycle may involve handling of immediate values, register addresses, and instruction formats, ensuring that the processor is properly configured to execute the instruction accurately. Through efficient decoding, RISC-V processors can effectively prepare instructions for execution in subsequent pipeline stages, optimising overall performance and throughput.

3. Execute Cycle

The execute cycle represents the stage in the instruction execution pipeline where the actual computation or data manipulation takes place. The decoded instruction from the previous stage undergoes execution, which typically involves arithmetic or logic operations specified by the instruction. RISC-V's simple and streamlined instruction set facilitates efficient execution, with each instruction designed to perform a specific operation in a single clock cycle. Additionally, the execute cycle may involve accessing registers or memory locations as necessary to retrieve operands or store results.

4. Memory Cycle

The memory access cycle is a critical stage in the instruction execution pipeline where data is transferred between the processor and memory hierarchy. Instructions that involve accessing memory, such as load and store operations, are executed. In the case of a load operation, the processor sends a request to the memory hierarchy to retrieve the data from the specified memory location. Conversely, for a store operation, the processor sends the data along with the address to the memory hierarchy for storage. RISC-V processors typically employ techniques such as memory hierarchy optimizations, caching, and memory access pipelining to enhance memory access efficiency. By efficiently managing memory access, RISC-V processors aim to minimise memory access latency and maximise overall system performance.

5. Writeback Cycle

It is the final stage in the instruction execution pipeline where the results of the executed instruction are written back to the appropriate destination. The computed result is typically written back to either a register or memory location, depending on the nature of the instruction. In the case of arithmetic or logic operations, the result is often stored in a register, while memory access instructions may involve writing data to a specificied memory address. Additionally, the writeback cycle may involve updating various processor state information, such as program counters or status flags, to accurately reflect the outcome of the executed instruction. By efficiently completing the writeback cycle, RISC-V processors

ensure that instructions are executed correctly and that the results are properly propagated for subsequent instructions, ultimately contributing to overall system performance and responsiveness.

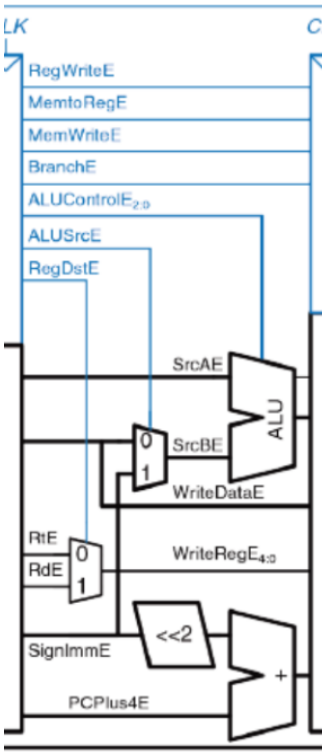

Fig 6: Execute Cycle

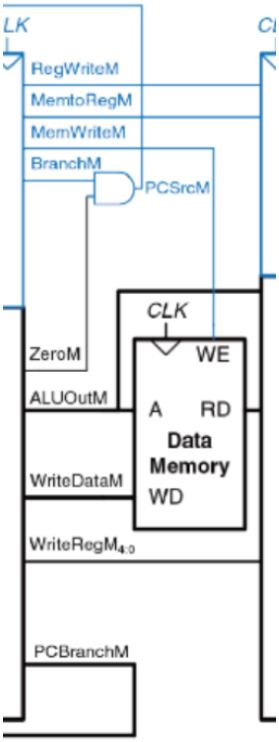

Fig 7: Memory Cycle

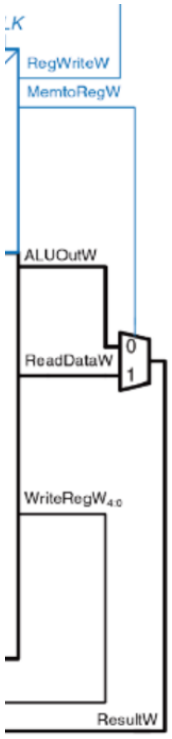

Fig 8: Writeback Cycle

Implementation on FPGA

Field Programmable Gate Arrays (FPGAs) are integrated circuits often sold off-the-shelf. They're referred to as 'field programmable' because they provide customers the ability to reconfigure the hardware to meet specific use case requirements after the manufacturing process. This allows for feature upgrades and bug fixes to be performed in situ, which is especially useful for remote deployments. FPGAs contain configurable logic blocks (CLBs) and a set of programmable interconnects that allow the designer to connect blocks and configure them to perform everything from simple logic gates to complex functions.

We are using the Basys 3 Artix 7 FPGA board to implement the cycle. The Basys 3 FPGA is a development board centred around the Xilinx Artix-7 FPGA, designed by Digilent for educational and hobbyist use. It features various onboard peripherals like

switches, buttons, LEDs, and connectors for expansion modules, facilitating experimentation and project development. Powered and programmed via USB, it offers ease of setup and connectivity to a computer for development purposes. With support for Xilinx's Vivado Design Suite, it provides a comprehensive environment for FPGA design, making it an ideal platform for learning digital logic design, FPGA programming, and embedded systems development at both beginner and advanced levels.
We implemented the ALU module on the FPGA board. It shows the basic functionalities of an ALU which are addition, subtraction, multiplication and an XOR gate. It was a task to run the whole project on the FPGA board hence we implemented single modules on the FPGA.

Results

Our pipelined RISC-V implementation demonstrated substantial performance improvements over the baseline architecture. Through rigorous experimentation, we observed a significant reduction in execution time and an increase in throughput, validating the efficacy of pipelining techniques. We successfully put together a five stage pipelined system that would increase the efficiency of executing instructions. We were able to generate the GDS file of the project and also implement the ALU module used in the cycles on the FPGA board. The working of the ALU can be seen by giving respective inputs to the FPGA. Hence we have a working RISC-V pipelined single cycle module as well as an ALU implemented on the FPGA.

Obstacles Faced

We had originally planned to implement this whole RISC-V cycle on an FPGA (Field Programmable Gate Array), specifically the Digilent Basys 3 Artix - 7 FPGA Board. The major issue that we faced was being able to generate bitstream. Without the bitstream we were unable to run the single cycle on the FPGA and hence couldn’t carry out implementation. In the beginning learning phase, difficulties were also encountered with the Verilog programming but were easily overcome by studying the language more and practising. After successfully coding all the cycles and connecting all register banks as well as running the synthesis and mapping the inputs and outputs to specific board pins we were faced with this problem right in the last step where the bitstream would not generate.

Conclusion

In conclusion, our RISC-V pipelining project has demonstrated the tangible benefits of implementing pipelining techniques in processor design. Through systematic experimentation and analysis, we have shown significant improvements in performance metrics such as execution time, throughput, CPI, and IPC compared to a non-pipelined baseline. The incorporation of advanced pipelining strategies, including instruction forwarding and branch prediction, has played a pivotal role in enhancing instruction-level parallelism and mitigating performance bottlenecks. We also learnt how to implement Verilog codes on an FPGA and learnt in detail about OpenLane Flow and its varMoving forward, the insights gained from this project can inform future advancements in processor design, paving the way for more efficient and high-performance computing systems.

Future Works

In the future we plan to work on successfully generating the bitstream and being able to implement the whole RISC-V pipelined system on the FPGA board. Although we optimised the whole process of instruction execution as much as we possibly could there is always scope for improvement. We can always work towards putting more effort into increasing the efficiency and functionality of this processor as it has a lot of scope in helping future creations.

References

- "Verilog HDL: A Guide to Digital Design and Synthesis" by Samir Palnitkar
- “Computer Organization and Design” by David Patterson and John Hennessy
- https://www.arm.com/
- https://digilent.com/reference/programmable-logic/basys-3/start