IITB-RISC Pipelined Processor Design

By

Siddhesh Sali

Electrical Engineering Department, IIT Bombay

Indian Institute of Technology Bombay

Abstract

IITB-RISC is a 6-stage pipelined processor. It is a 16-bit very simple computer that is based on the Little Computer Architecture. The IITB-RISC is a 16-bit computer system with 8 registers. It follows the standard 6 stages of pipeline (Instruction Fetch (IF), Instruction Decode (ID), Register Read (RR), Execute (EX), Memory Access (MEM), and Write Back (WB)). The architecture is optimized for performance, i.e. it includes hazard mitigation techniques. Therefore, a forwarding mechanism has been implemented.

Code and design available at https://github.com/sid4sal/pipelined_processor

Keywords: Microprocessor, Instruction Set Architecture (ISA), RISC, Pipelining.

Table of Contents

1 Introduction

Aim of the project is to design and implement a pipelined processor, IITB-RISC. It is a very simple computer that is based on the Little Computer Architecture. The IITB-RISC is an 8-register, 16-bit computer system. The IITB-RISC is very simple, but the properties, instructions are chosen so that it is general enough to solve complex problems. The architecture allows predicated instruction execution and multiple load and store execution.

1.1 Objectives:

- To select instructions and properties.
- To design a 6-stage pipelined processor, IITB-RISC as mentioned above.
- Implement the processor in VHDL.
- To include hazard mitigation techniques for optimized performance by implementing forwarding mechanism.
- To Implement the branch predictor and make optimizations for faster execution(Not done yet).

2 Instruction and Processor Properties

2.1 Processor Properties

- 16-bit
- 8 general purpose registers (R0 to R7)
- R0 stores the PC (Program Counter)
- All addresses are byte addresses and instructions. And it fetches two bytes for instruction and data.
- This architecture uses a condition code register which has two flags: Carry flag (C) and Zero flag (Z).
- There are three machine-code instruction formats (R, I, and J type) and a total of 14 instructions.

2.2 Instruction Format

- R Type Instruction Format

Opcode	Register A (RA)	Register B (RB)	Register C (RC)	Complement	Condition (CZ)
4 bit	3 bit	3 bit	3 bit	1 bit	2 bit

- I Type Instruction Format

Opcode	Register A (RA)	Register C (RC)	Immediate
4 bit	3 bit	3 bit	6 bit signed

- J Type Instruction Format

Opcode	Register A (RA)	Immediate
4 bit	3 bit	9 bit signed

3 Literature Review

3.1 RISC

The widespread adoption and success of RISC computer architecture stem from its utilization of a simplified instruction set to accelerate processing. Since its inception in the 1980s, RISC architecture has become a cornerstone in microprocessor development, embedded systems, and various computing hardware applications. A key distinguishing feature of RISC architecture is its reliance on a concise set of basic instructions, executed swiftly. This stands in sharp contrast to the intricate instruction sets of Complex Instruction Set Computing (CISC) designs. This streamlined approach not only boosts execution speed and overall performance but also minimizes the hardware overhead required for instruction decoding and execution. As a result, RISC-based CPUs are characterized by their compactness and superior energy efficiency.

3.2 Pipeline

CPU performance is greatly improved through the implementation of pipeline architecture, which facilitates the simultaneous execution of multiple instructions. Within the pipeline, instructions progress through a series of stages, each assigned a specific task to complete during execution.The core principle of pipeline design is to break down the execution of instructions into smaller phases, each handled by a separate component within the CPU. This allows for multiple instructions to be in various stages of execution concurrently, thanks to the pipeline structure. As one instruction concludes a stage, another can enter, enabling the overlap of instruction executions and enhancing overall efficiency. While overlapping instructions in a pipeline architecture enhances execution speed and performance compared to sequential processing, it also introduces challenges, notably hazards that can impede instruction execution.

Data hazards arise when an instruction must await the outcome of a preceding one, while control hazards occur when an instruction is stalled by a conditional branch's outcome. To mitigate these issues, various optimization techniques have been devised, including forwarding, which directly transmits results to dependent instructions, and stalling, which temporarily halts the pipeline until data or control dependencies are resolved.

3.2.1 Instruction Fetch (IF)

In the initial stage of the pipeline architecture within the IITB CPU project, known as the Instruction Fetch stage, the CPU retrieves the subsequent instruction from memory and readies it for execution. This involves accessing the memory location specified by the program counter and retrieving the stored instruction. Subsequently, the instruction is placed into the instruction register (IR), which serves as a temporary register accessible to subsequent pipeline steps. Notably, it's crucial to increment the program counter within this same stage to ensure continuous operation and prevent any idle stages in subsequent steps of the pipeline.

3.2.2 Instruction Decode (ID)

The second stage within the 6-stage pipeline architecture of the IITB CPU project assumes a pivotal role in the processor's overall operation. Its primary function, as previously mentioned, is to decode the instruction fetched in the preceding stage and prepare it for subsequent pipeline stages. Once the instruction is stored in the instruction register, it traverses through the pipeline registers and enters the controller. Often likened to the "brain" of the processor, the controller orchestrates the activities of all other components. Its initial task during the instruction decode

stage is to identify the instruction type received, crucial for determining the specific operations and operands required. For instance, arithmetic instructions like addition or subtraction entail distinct operations and operands compared to jump instructions. Leveraging this understanding of instruction types, the controller then dispatches control signals to all processor components. It's essential that these control signals propagate through the pipeline registers, ensuring each pipeline stage is equipped with the requisite controls tailored to the current instruction under processing.

3.2.3 Register Read (RR)

The Register Read stage, positioned as the third step within the IITB CPU project's 6-stage pipeline architecture, follows the Instruction Decode stage. In this phase, the operands required for the current instruction are retrieved from the register file. The instruction provides the register addresses for these operands, which are then forwarded to the register file. Essentially, a register file comprises a collection of registers capable of storing and accessing data values, with each register accessible via its unique address. After receiving the register addresses, the register file retrieves data from the specified registers and transfers it to the subsequent step of the pipeline. This data can serve various purposes, such as being used as memory access addresses, inputs for arithmetic or logical operations, or both.

3.2.4 Execute (EX)

The Execution stage, situated as the fourth step within the 6-stage pipeline architecture of the IITB CPU project, assumes a critical role in performing the actual calculations and operations dictated by the instruction set. This phase integrates various components, including an

Arithmetic Logic Unit (ALU) and multiple adders, to execute the required operations efficiently. The ALU handles a diverse range of operations specified by the instructions, while the adders primarily serve to calculate addresses and support complex calculations by feeding output into each other. Outputs from this stage, such as those from the ALU and adders, are directed to the subsequent stage. The inputs to these components are controlled by multiplexers, enabling selection from different sources, with control signals managed by the controller.

3.2.5 Memory Access (MEM)

In the 6-stage pipeline architecture of the IITB CPU project, the fifth stage is dedicated to Memory Access. Here, the CPU retrieves the necessary information from memory as instructed by the current instruction. The address, calculated in the preceding stage, determines the specific memory location required for the operation. Subsequently, this address is forwarded to the memory unit, facilitating the retrieval of data stored at that particular location. Upon obtaining the data, it is temporarily stored in the next pipeline register until the Write Back stage.

3.2.6 Write Back (WB)

In the six-stage pipeline architecture of the IITB CPU project, the final stage is termed Write Back. Here, the outcomes of instructions are recorded back into the register file. The results, temporarily held in the pipeline register, are now transferred to the designated destination register specified in the instruction. With the completion of this phase, the pipeline is ready to accept the next command following the successful writing back of data into the register file.

3.3 Hazards

A hazard in computer architecture, particularly in the context of pipelined processors, refers to a situation where the execution of an instruction is affected by the status or outcome of another instruction. Hazards can occur due to various dependencies between instructions and can lead to incorrect results or stalls in the pipeline, reducing overall performance. Common types of hazards include data hazards, control hazards, and structural hazards.

3.3.1 Data Hazards

These occur when an instruction depends on the result of a previous instruction that has not yet completed. For example, if one instruction writes to a register while another instruction reads from the same register, the second instruction must wait for the first one to complete before proceeding.

3.3.2 Control Hazards

These arise when the control flow of the program changes, such as due to branch instructions. If a branch instruction is followed by instructions that are in the pipeline, those instructions may need to be flushed if the branch is taken, resulting in wasted cycles.

3.3.3 Structural Hazards

These occur when hardware resources are not available to execute instructions simultaneously. For instance, if two instructions require access to the same hardware unit at the same time, a structural hazard arises, causing one instruction to stall until the resource becomes available.

3.4 Hazard Mitigation Techniques

Following are the techniques to mitigate hazards in a pipeline processor:

1. Forwarding : Directly pass data from one stage to another to resolve data hazards without waiting for write-back.

2. Stalling : Insert idle cycles into the pipeline to resolve hazards by delaying the execution of dependent instructions.

3. Branch Prediction: Predict the outcome of branch instructions to speculatively execute instructions and minimize the impact of control hazards.

4 Datapath Design

The datapath can be accessed by using the following link:

https://github.com/sid4sal/Pipelined_Processor/blob/main/Datapath_IITB_RISC_Pipeline.svg

The VHDL implementation of the processor can be found on

https://github.com/sid4sal/Pipelined_Processor/