16-Bit RISC PIPELINE MICROPROCESSOR

May 10, 2024

Group No 10

Hrishikesh S 22B4217
Jatin Kumar 22B3922
Suyash Jitendra Majhi 22B3924
Arnav Agrawal 22B3917

Contents

1. Introduction

The project aims to design and develop a Central Processing Unit (CPU) using Reduced Instruction Set Computing (RISC) and a 6-stage Pipeline Architecture. The project involves the design and implementation of various components of the CPU, such as the Instruction Decoder, Control Unit, Arithmetic and Logic Unit (ALU), and Memory Management Unit (MMU), among others. The use of RISC architecture simplifies the design of the CPU, making it easier to optimize its performance and efficiency.

The 6-stage Pipeline Architecture is an essential component of the project, which allows the CPU to execute multiple instructions simultaneously, thus enhancing its processing speed. The IITB-RISC-23 is a 16 -bit computer system with 8 registers. The pipeline stages include Instruction Fetch, Instruction Decode, Register Read, Execute, Memory Access, and Write Back.

2. Core Concepts Implemented

2.1. PIPELINE

Pipeline enhances the CPU by allowing multiple instructions to be executed at once. Pipeline can be considered as a collection of steps the instruction needs to be completed,and each step gives the instruction a specific task to be completed.

In Pipeline,an instruction is executed by dividing it into multiple stages and then executing them simultaneously using different parts of the CPU.By doing so, multiple instructions can overlap,as one instruction is entering into a stage that is finishing up,and another instruction is exiting it.On comparison to running the instruction in sequential order,pipeline provides great advantage by executing multiple instructions at the same time,thereby saving a huge amount of time and performance.

But the pipeline architecture can introduce us with some difficulties,because not all the stages within the same instructions are independent from each other,unlike from that of the assembly line concept,in which all the stages are independent of each other. These are known as pipeline hazards.

The dependencies/hazards can occur due to various reasons.As an example,let us take the case in which an instruction need to wait for the result obtained by the previous instruction,and this is known as data hazard.Likewise,a control hazard can occur for a branching instruction as it needs to wait for the outcome of the branching condition before it can proceed.

We will effectively address the problem by introducing the concepts of forwarding and stalling.

- **Forwarding :** Forwarding involves passing the result of an instruction directly to the next instruction that needs it.
- **Stalling :** Stalling involves pausing the pipeline for the data or control dependencies to be resolved.

2.2. RISC

The RISC architecture uses a fixed length instructions,which are basic and quick-to-execute.The Cycles Per Instructions (CPI) of RISC is 1. In contrast to the CISC architecture,which uses complex set of instructions which is variable in length. Thus RISC architecture is more efficient and enhances the performance.Moreover it is easier to implement pipeline in RISC as it is using only simple to execute and fixed length instructions,whereas in CISC it is comparatively more difficult to implement Pipeline as it involves more of complex instruction set to be implemented and variable instruction length.In addition to this,the RISC uses less hardware for decoding and executing instructions thus leading to more powerful CPUs which are smaller in size.

3. The 6 stages of Pipelining

The IITB-RISC-23 is a 16 -bit computer system with 8 registers. It should follow the standard 6 -stage pipelines.

- Instruction fetch,
- Instruction decode,
- Register read,
- Execute,
- Memory access,
- Write back.

3.1 Instruction Fetch

First stage of the Pipeline is instruction fetch.During this stage the processor fetches the next instruction from memory and prepares it for execution.

The CPU fetches the memory location of the next instruction from the Program Counter (PC) and then fetches the instructions specified by the PC and then stores it into the Instruction Register (IR).The IR is accessible by the further stages of the Pipeline.

The core point here is that we need to increment the PC at the same stage so that the next stage will not be idle.

3.2 Instruction Decode

Instruction decode stage is responsible for decoding the instruction fetched in the previous stage.

After the instruction enters the IR,it travels through the pipeline registers and enter the Controller,often referred to as the Brain of the CPU.

The first task of the instruction decode stage is to identify the type of instruction.This is important as different types of instructions will require different operations and different operands.

Once again it is important that the control signals are supposed to be going through the Pipeline registers so that it is provided with the proper instructions for executing the given signal.

The Controller coordinates the activities of all the components of the CPU.

3.3 Register Read

In this stage, the read and write addresses are first updated, and then the appropriate forwarding logic is used to get the correct register value in case of any data hazards. In case of branch or jump instructions, the branch-rr control signal is set and the proper PC-branch is used to branch to the correct instruction, along with flushing any previous instructions running.

3.4 Execute

This is the 4th stage of the pipeline.This stage is responsible for performing all the required calculations specified by the instruction. The ALU will perform the required operations on the operands depending on the type of instructions given to the CPU.

3.5 Memory Access

The memory address is used from the alu output and depending on the nature of instruction, i.e, load/store , the input is given or data is read from the memory.

3.6 Write Back

The sixth stage of the IITB CPU project's six-stage pipeline architecture is called Write Back. The instruction results are written back to the register file during this phase. The instruction's result, which had been momentarily stored in the pipeline register, is now written back to the destination register that was indicated in the instruction. The pipeline is prepared to receive the next command after the data has been written back to the register file.

4 Handiling hazards

1. Data Hazards

Data Forwarding :

- The technique of Data Forwarding helps us to find the correct Register value in the Register read stage,as opposed to stalling which will increase the number of cycles required (CPI)
- The correct value of the register may be in any of the execute,memory access or write back stage
- We will first check if there is any match of any read register address from register read stage to any write register address from execute address,then memory access,then memory write-back stage,in that order as the latest instruction close to register stage will have the updated value.
- If there's a match for execute stage, the possible instructions before the instruction where the value is read, in the execute stage are :-
  1. ADA, ADC, ADZ, AWC, etc or ADI or NDU, NDC, NDZ, etc (take the output of the ALU)
  2. LLI (take the immediate value)
- Else If there's a match for memory access :-
  1. Take the ALU output if instruction in memory stage is ADD, NAND
  2. Take immediate value if instruction in memory stage is LLI,
  3. Take the data read from memory if the instruction in memory stage is LW, LM
  4. Take the PC+2 value in the pipeline register if the instruction in memory stage is JAL, JLR
- Else if there's a match for writeback stage :-
  1. Take the ALU output if instruction in memory stage is ADD, NAND
  2. Take immediate value if instruction in memory stage is LLI,
  3. Take the data read from memory if the instruction in memory stage is LW, LM
  4. Take the PC+2 value in the pipeline register if the instruction in memory stage is JAL, JLR
- Thus, it is ensured that you get the right value in your pipeline register at the end of the register read stage.

2. Control Hazards

Branch Instructions :

- If the opcode is that of a branch instruction in stage 2,we set the branch-id flag to 1, we use a dynamic branch predictor in stage 2 to check if our branch is taken is PC+ Imm*2 OR PC+2
- So, the branch controller takes the history bit as an input alongwith PC+2 as one input and PC+ Imm*2 as the other input, the branch controller selects PC+ Imm*2 if the history bit is one and selects PC+2 if the history bit is zero.

- Now it stores the other input into a temporary register
- Now since we have set the branch flag, we use this flag as the control signal to the mux in instruction fetch stage and it takes the PC from the branch controller in stage 2.
- Now as the instruction moves to stage 3, we have the register values, so we now use a comparator which gives a single bit output as 1 if the branch should've been to PC + imm*2 and output as 0 if the branch should've been to PC+2.
- If the branch taken is to the correct PC, the instruction goes on as it is, else a hazard flag is set in the RR stage. If this flag is set we set our branch-rr flag to 1 and the PC to be branched to is taken from the temporary register which has the alternate PC.
- If the hazard flag is set, the instruction in current ID stage is flushed and the PC-branch from RR stage is given to the instruction memory in stage 1.

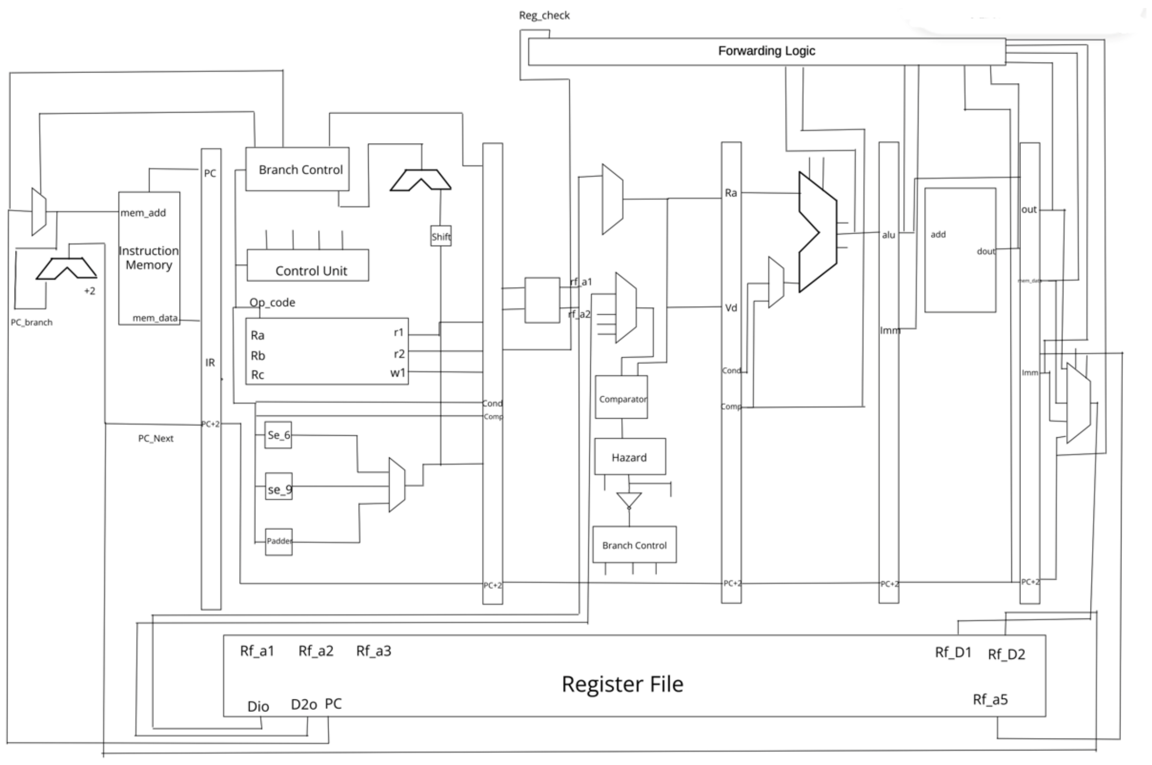
Jump Instructions :

- If the instruction in stage 2 turns out to be a jump instruction, we flush the instruction in if stage running right now,because we are jumping and we will not need it.
- We then move to stage 3, where it calculates the branch location using another ALU used in RR stage, and it puts this value into PC-branch-RR and sets the branch-rr flag which allows stage 1 to take the updated PC.

This is the logic we have used to make the components necessary for mitigating control hazards. The components involved are branch-controller(in stage 2), control-hazard-unit, comparator, and another branch-controller(in stage 3). We believe this is the best possible solution, in lieu of this course, to minimize the CPI increase associated with control hazards. If the branch predictor is correct, there are no loss of cycles If the branch predictor is wrong, one cycle is lost. Jump instructions lead to a loss of one cycle always.

5. Datapath

The Datapath of the CPU is given in the following figure :



6. Work Contribution of team members

Arnav Agrawal	Pipelined registers, Forwarding unit, Pipeline Stages 1 to 3, Comparator
Hrishikesh S	Pipeline Stages 4 to 6, Padder, Main, Forwarding unit, ALU
Jatin Kumar	SIGN extenders, Data Memory, Hazard Control, Branch Control
Suyash Jitendra Majhi	Hazard Control, Branch Control, Instruction memory, Register File, Register Address