Princess Sumaya University for Technology

King Abdullah II Faculty of Engineering

Computer Engineering Department



|  |
| --- |
| **Computer Architecture II**  **Verilog Project**  **Group No.** |

|  |  |  |
| --- | --- | --- |
| *Authors:* |  | *Supervisor:* |
|  |  |  |

*December 24, 2019*

***Abstract***

*A processor pipeline is the process of accumulating instructions from the processor through a pipeline. It allows storing and executing instructions in an orderly process, it is also attempts to keep every part of the processor busy by dividing instructions into a set of sequential steps called stages.*

*In this project we aimed to design a limited version MIPS processor, with a set of chosen instructions, some of which were custom made for the sake of the project, the design includes the pipeline of the processor, in addition to the Verilog code to get the processor to run, tested by some test benches.*

**Table of Contents**

[**1.** **Introduction** 2](#_Toc60302698)

[**1.1 Objectives** 2](#_Toc60302699)

[**2.** **Processor Design** 2](#_Toc60302700)

[**2.1 IF Stage** 2](#_Toc60302701)

[**2.2 ID Stage** 2](#_Toc60302702)

[**2.3 EXE Stage** 3](#_Toc60302703)

[**2.4 MEM Stage** 3](#_Toc60302704)

[**2.5 WB Stage** 3](#_Toc60302705)

[**2.6 Control Unit** 4](#_Toc60302706)

[**3.** **Design Analysis** 4](#_Toc60302707)

[**4.** **Tests runs and discussion** 4](#_Toc60302708)

[**4.1 Test Run #1** 4](#_Toc60302709)

[**4.2 Test Run #2** 4](#_Toc60302710)

[**4.3 Test Run #3** 4](#_Toc60302711)

[**4.4 Test Run #4** 4](#_Toc60302712)

[**4.5 Test Run #5** 4](#_Toc60302713)

[**4.6 Additional Test runs** 4](#_Toc60302714)

[**5.** **Conclusion** 4](#_Toc60302715)

[**References** 4](#_Toc60302716)

# **Introduction**

The processor is an integrated circuit that performs a set of calculations that run a computer. It performs arithmetical, logical, input/output and other basic instructions that are passed from the operating system.  
Pipelining increases the CPU instruction throughput, the number of instructions completed per unit of time, however; it does not reduce the execution time of an individual instruction.  
In this project we will be aiming to design a simple processor with its pipeline, using Verilog language to code each block of the pipeline, after each block is coded individually, it will be connected to the rest of the blocks using a “Top” module, in which the whole image of the pipeline will show.  
The module is going to be tested using a set of test benches provided with the project requirements, and some blocks will also be tested individually with separate test benches.

## **1.1 Objectives**

The main objective of this project is to design a simple MIPS processor that works on a set of instructions, the design will include the pipeline stages, in addition to the code that runs this pipeline.

The processor should receive an instruction, decode it, execute it properly and saves the data calculated in its proper register, it should also allow forwarding and flush stages depending on the hazard detection unit.

# **Processor Design**

The design is separated into different stages as follows:

## **2.1 IF Stage**

IF stands for instruction fetch, and the main blocks of this stage are the Instruction Memory & the PC (Program Counter).

The instruction Memory will take the output the of PC as it’s input, then save it in the IF/ID register in order to send it to the next stage, The PC’s input will depend on a set of 2-to-1 Multiplexers, those multiplexers will choose their output depending on the Control Unit, which will send signals depending on the instruction decoded, those signals will decide whether the PC will be added by 4 (to go to the next instruction), or branch/jump to a whole new address if the instruction was branch or jump.

The IF/ID register will be separating the bits of the instruction received and sending them to the next stage.

## **2.2 ID Stage**

The ID (Instruction decoding) stage is responsible for decoding the instruction received from the previous stage, putting the bits of the instruction received into their proper place, then passing it to the following stage.

The main blocks in this stage are the Register file and the Control Unit; the Register file contains the registers used in the processor, and it will receive its input from the IF/ID register, it will be responsible for assigning the registers that will be reading the data, and the registers that we will be writing the data onto (destination registers) using a set of 2-to-1 multiplexers to choose which register will take which role. Then sending the output to the ID/EXE register to pass the data onto the next stage.  
The Control Unit will be taking its data as bits that were separated in the IF/ID register, the input received will be deciding the operation of the instruction, the operation of the instruction depends mainly on the OP code and the Function ID of the instruction (or only the OP code in the I-type instructions).  
The Control unit will then separate those bits even more and sending them to each assigned block to distinguish which wire will be used in this particular instruction, and which will not be needed, it’ll also pass some of those bits to the ID/EXE register, which will pass them into the next stage.This stage also contains the Hazard detection unit, which will decide whether we are going to be stalling/flushing the previous stage, depending on the input received to it.

## **2.3 EXE Stage**

The EXE (Execution) stage is responsible for doing the arithmetic operations of the data, whether it is to calculate a mathematical operation, choose which register is the destination register, or to calculate a branch instruction address, before passing it in the next stage to be saved in the memory. The main block in this stage is the ALU (Arithmetic Logic Unit).

In the ALU, registers are operated on using mainly their OP code and/or their Function ID, the ALU receives these data as bits from the Control Unit, to decide which instruction is going to be executed, the inputs of the ALU are registers decided from two 3-to-1 multiplexers, as well as two inputs that will act as most significant bits in case of a 64-bit operation. Giving two outputs, one is 32-bits, and the other is 64-bits. It will then pass the output to the EXE/MEM register.

This stage also contains an adder; this adder is used to calculate the branch address in case of a branch instruction.

The Forwarding Unit is responsible for detecting dependencies in the instructions, and then forwarding the needed data to a said instruction in order for it to be executed properly.

## **2.4 MEM Stage**

The Memory stage is responsible for storing the data and the addresses while the program is being executed.

The Data Memory is the main block in this stage, it takes it’s inputs from the EXE/MEM register as following: an input for the address, and two for the data being written, separated into 32-bits and 64-bits, we control which of those two inputs to take based on signals sent from the Control Unit. The output of the Data Memory is then passed to the Write Back stage using the MEM/WB register.

Not all data passing through this stage needs to be stored in the memory. Some data, like the destination register, is passed immediately to the Write Back stage.

## **2.5 WB Stage**

During the WB (Write Back) stage, the instructions write their results into the register file in the ID stage and stored in the destination register.

The data being written back is passed into this stage using the MEM/WB register; the data is chosen using two 2-to-1 multiplexers, where one multiplexer is responsible for the 32-bits results while the other is responsible for the 64-bits result controlled by signals coming from the Control Unit.

## **2.6 Control Unit**

The Control Unit is where each block receives its signals from, to decide which instruction is being executed, depending mainly on the OP code and the Function ID of the instruction.

The Control Unit gets its input from the IF/ID register, then, depending on the inputs, separate signals will be sent to multiplexers, the Register File, the ALU and the Data Memory to select the proper wires that should be used for the instruction being executed.

It is very crucial for the Control Unit to send all the signals correctly, as one wrong signal, even the 1-bit signals, can cause the whole pipeline to fail and the instruction to not work.

## Hazard Detection

khaled

## Forwarding Unit

ahmad

# **Design Analysis**

## IF

ahmad

## ID

ahmad

## EXE

Khaled

## WB

khaled

## Register File

ahmad

## ALU

khaled

## Data Memory

ahmad

## Control Unit

khaled

# **Tests runs and discussion**

## **4.1 Test Run #1**

## **4.2 Test Run #2**

## **4.3 Test Run #3**

## **4.4 Test Run #4**

## **4.5 Test Run #5**

## **4.6 Additional Test runs**

# **Conclusion**

# **References**