

**Computer Architecture (ENCS4370)**

**Project #2**

**Multi-cycle MIPS RISC Processor in Verilog**

**Prepared by:**

**Christina Saba – 1201255**

**Joud Hijaz – 1200342**

**Pierre Backleh – 1201296**

**Supervised by:**

**Dr. Ayman Hroub**

**January 2024**

# Abstract

# Table of Content

# 1. Design & Implementation

Our method for this project was to create a multi-cycle CPU using the MIPS Reduced Instruction Set Computer (RISC) architecture. The data route is divided into separate phases in the processor's architecture, each of which is completed in a single clock cycle. FETCH, DECODE, EXECUTE, MEMORY, and WRITEBACK are the five essential phases.

## 1.1 Processor Architecture

A section of the MIPS instruction set, which is known for its effectiveness and simplicity, is used in this project. Every instruction follows the same format and has a length of 32 bits. The following categories are included in the instruction set:

1. ***R- Type Instructions:***

These instructions define the operation and register operands in a specified syntax and involve register operations.

1. ***J- Type Instructions:***

Jump instructions are used to change the program's control flow. They consist of calls to functions and unconditional jumps.

1. ***I- Type Instructions:***

Instructions for tasks like loading, saving, and branching that need immediate values.

1. ***S- Type Instructions:***

Guidelines for adding and removing values from the stack that are relevant to stacks.

## 1.2 Multi-cycle

## 

An instruction is executed in several separate stages, each taking one clock cycle to finish, in a multi-cycle processor design. These stages make it easier to execute instructions in a more organized and modular method.

1. ***Fetch Instruction:***

Using the Program Counter, the CPU retrieves the instruction from memory and stores it in an instruction memory.

1. ***Decode Instruction:***

To identify the kind of instruction and the operands involved, the fetched instruction is decoded at the instruction decode stage. The type and purpose of the instruction determine which control signals are generated in order to carry out the instruction.

1. ***Execution Stage:***

During the execution phase, the instruction's stated computation or action is carried out. The relevant arithmetic or logical operation is carried out once the operands needed for the operation are retrieved from memory or the register file.

1. ***Memory Access Stage:***

If the instruction calls for accessing or changing data in memory, memory operations are carried out during the memory access stage. Instructions for load (LW) and store (SW) are mostly employed at this step. When a load instruction is used, information is taken from memory and put into a register. Data from a register is written into memory at a designated memory location in order to execute a store instruction.

1. ***Write-back Stage:***

The process of executing an instruction ends with the write-back stage. Depending on the kind of instruction, it involves writing the computation's results back to memory or the destination register.

## 1.3 Instruction Types

### 1.3.1 Register Type



* 6-bit function, to determine the specific operation of the instruction
* 4-bit Rd: destination register
* 4-bit Rs1: first source register
* 4-bit Rs2: second source register
* 14-bit unused

Instruction Decoding for R-Type Register:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function | RTL | | | | |
| **FETCH** | **DECODE** | **EXECUTE** | **MEMORY** | **WRITEBACK** |
| AND(000000) | PC ← PC + 1 | A = Rs1 B = Rs2 | Rd ← Rs1 AND | - | Rd = ALU Out |
| ADD(00001) | PC ← PC + 1 | A = Rs1 B = Rs2 | Rd ← Rs1 + Rs2 | - | Rd = ALU Out |
| SUB(000010) | PC ← PC + 1 | A = Rs1 B = Rs2 | Rd ← Rs1 - Rs2 | - | Rd = ALU Out |

### 1.3.2 Instruction Decoding for I-Type Register



* 4-bit Rd: destination register
* 4-bit Rs1: first source register
* 16-bit immediate: unsigned for logic instructions, and signed otherwise.
* 2-bit mode: this is used with load/store instructions only, such that:

00: no increment, 01: post increment the base register 10-11: unused

Instruction Decoding for I-Type Register:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function | RTL | | | | |
| **FETCH** | **DECODE** | **EXECUTE** | **MEMORY** | **WRITEBACK** |
| ANDI(000011) | PC ← PC + 1 | A = Rs1  B = Imm | Rd ← Rs1 AND Rs2 | - | ALU  output |
| ADDI(000100) | PC ← PC + 1 | A = Rs1  B = Imm | Rd ← Rs1 + Imm | - | ALU output |
| LW(000101) | PC ← PC + 1 | A = Rs1  B = Imm | Rd ← Rs1 + Imm | Rd ← M[Rs1 + Imm] | ALU output |
| LW.POI(000110) | PC ← PC + 1 | A = Rs1  B = Imm | Rd ← Rs1 + Imm | Rd ← M[Rs1 + Imm]  Rs1 ← Rs1 + 1 | ALU output |
| SW(000111) | PC ← PC + 1 | A = Rs1  B = Imm | Rd ← Rs1 + Imm | M[Rs1 + Imm] → Rd | - |
| BGT(001000) | PC ← PC + Imm | A = Rs1  B = Rd | If Rs1 < Rd | - | - |
| BLT(001001) | PC ← PC + Imm | A = Rs1  B = Rd | If Rs1 > Rd | - | - |
| BEQ(001010) | PC ← PC + Imm | A = Rs1  B = Rd | If Rs1 == Rd | - | - |
| BNE(001011) | PC ← PC + Imm | A = Rs1  B = Rd | If Rs1 != Rd | - | - |

### 1.3.3 Instruction Decoding for J-Type Register

1. Jump (jmp) & call:

* 6-bit function, to determine the specific operation of the instruction
* 26-bit Offset: signed immediate

2. Return (ret)

* 6-bit function, to determine the specific operation of the instruction
* 26-bit: unused

Instruction Decoding for J-Type Register:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function | RTL | | | | |
| **FETCH** | **DECODE** | **EXECUTE** | **MEMORY** | **WRITEBACK** |
| JMP(001100) | PC ← PC + Immediate26 | PC + Imm26 | - | - | - |
| CALL(001101) | PC ← PC + Immediate26 | PC + Imm26 |  |  |  |
| RET(001110) | PC ← pop(stack)  Take top of the stack | PC = pop(stack) | - | - | - |

**1.3.4 Instruction Decoding for S-Type Register**

* 4-bit Rd: destination register
* 22-bit: unused

Instruction Decoding for S-Type Register:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function | RTL | | | | |
| **FETCH** | **DECODE** | **EXECUTE** | **MEMORY** | **WRITEBACK** |
| PUSH (001111) | PC ← PC + 1 | PUSH(Rd)  Rd is pushed on the top of the stack | PUSH (Rd) | - | TOP of the Stack = ALU  Out |
| POP(010000) | PC ← PC + 1 | Rd = POP()  The top element of the stack is popped, and it is stored in the Rd register | Rd = Fetch(top of the stack) | - | - |

## 

## 1.4 Data path

## 1.5 Components used to build the processor

1. ***Program Counter (PC):***

Responsible for monitoring the address of the subsequent command to be retrieved.

1. ***Instruction Memory:***

Keeps program instructions in memory, which the PC retrieves when needed.

1. ***Register File:***

A group of registers used to store data temporarily while computation is being done.

1. ***ALU (Arithmetic Logic Unit):***

Manipulating data using logical and arithmetic processes.

1. Data Memory