Egypt-Japan University of Science and Technology

Department of Computer Science Engineering

CSE 431 - Advanced Computer Architecture

16-bit MIPS Processor Design

Amr Tarek Zaki Abdelrahman

Student ID: 120220212

Submitted to:

Dr. Hossam Kasem

Submitted on: December 25, 2025

Contents

List of Figures

List of Tables

Listings

1 Abstract

This technical report presents the design and implementation of a **16-bit MIPS** (Million Instructions Per Second) processor, in both **single-cycle** and **pipelined architectures**. The report details the design specifications, including the **instruction set architecture (ISA)** and **control signal encoding**. It also includes **datapath schematics** for both architectures along with **waveform simulations** for testing and verification. Finally, a **performance comparison** with analysis summarizes the advantages and trade-offs between the two designs.

2 Introduction

MIPS architecture is a RISC (Reduced Instruction Set Computer) architecture that is widely used in computer engineering education and research. The 16-bit MIPS processor introduced in this report is a simplified version of the standard MIPS architecture. The processor supports very basic set of instructions with minimal specifications. The main differences between both processor will be highlighted throughout the report.

2.1 Objectives

The primary objectives of this project are:

- **Design and implement** a simple 16-bit MIPS processor in both single-cycle and pipelined architectures.
- **Simulate and verify** both processor designs using simple tests with waveform analysis.
- **Compare the performance** of single-cycle and pipelined architectures.
- **Provide insights** into the trade-offs between single-cycle and pipelined designs.
- **Document the design process, specifications, and performance analysis** in this technical report.

2.2 Development Environment

The development environment for this project includes:

- Hardware Description Language (HDL): **Verilog**
- Compilation Tool: **Quartus (Web Edition)**
- Simulation Tool: **ModelSim** (included with Quartus)

3 Design Specifications

3.1 Memory Organization

The 16-bit MIPS processor utilizes two separate memory interfaces for instruction and data memory, respectively. This eliminates *structural hazards*[1] and allow for simultaneous instruction fetch and data access in the pipelined architecture.

While the standard MIPS architecture uses the Big-Endian format for memory storage, this 16-bit MIPS processor employs the Little-Endian format. In Little-Endian format, the least significant byte (LSB) is stored at the lowest memory address, while the most significant byte (MSB) is stored at the highest memory address. This choice simplifies certain operations and aligns with common practices in various computing systems.

[1] Structural hazards occur when multiple instructions need to use the same resources simultaneously

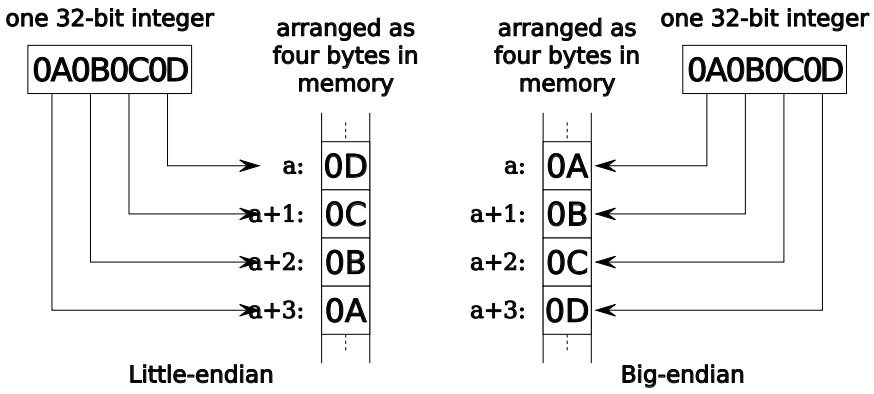


Figure 1: Illustration of Little-Endian and Big-Endian memory layouts. Source: Wikimedia Commons

3.1.1 Instruction Memory

The instruction memory is a *byte-addressable*[2], read-only memory module with **64 bytes** of storage (addresses 0x00 to 0x3F). It is implemented in **little-endian** format and stores program instructions.

0x00	0x00
0x01	0x81
0x02	0x42
0x03	0x81
⋮	
0x3F	0x00

Figure 2: Instruction Memory Layout (64 bytes)

[2]Meaning that each location stores one byte (8 bits) of data

3.1.2 Data Memory

Like the instruction memory, the data memory is also a byte-addressable, little-endian memory module with **64 bytes** of storage. It supports both read and write operations. Figure 3 shows an abstract representation example of the memory layout.

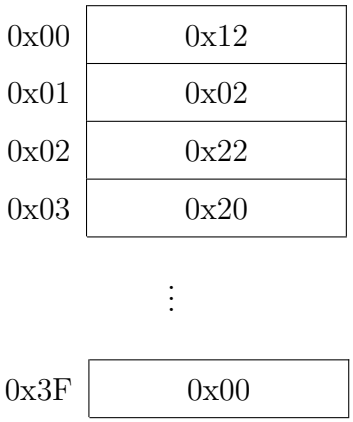


Figure 3: Data Memory Layout (64 bytes)

Key Feature:

- Supports both read and write operations.
- Byte-addressable with little-endian format.
- **Synchronous write:** Data is written to memory on the rising edge of the clock when `MemWrite = 1` signal is asserted.
- **Asynchronous read:** Data can be read from memory at any time without needing for a read signal.

3.1.3 Memory Interfaces

Both memories follow the following interface specifications:

Parameter	**Specification**
Data Width	16-bit
Instruction Width	16-bit
Address Width	16-bit
Memory Model	Byte-Addressable

Table 1: Design Specifications

Instruction Memory Interface:

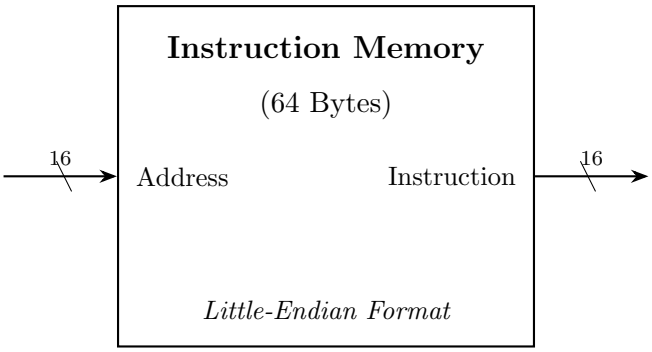


Figure 4: Instruction Memory Block Diagram

Data Memory Interface:

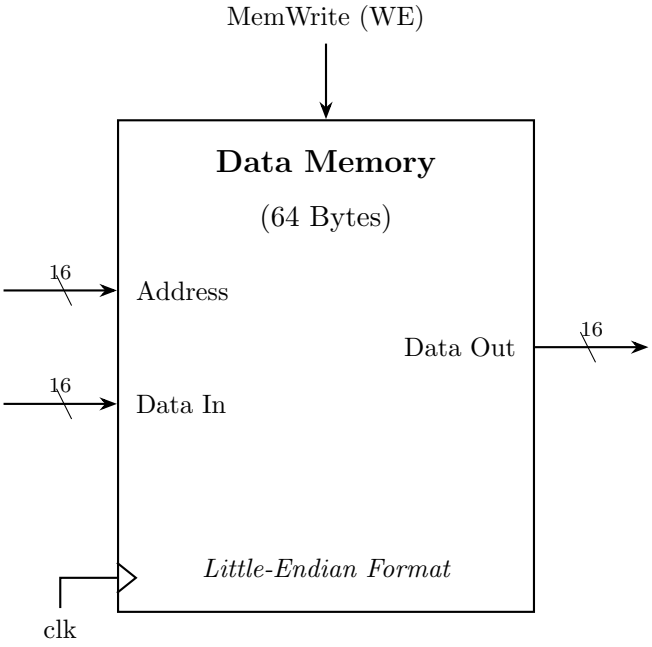


Figure 5: Data Memory Block Diagram

3.2 Instruction Set Architecture (ISA)

The 16-bit MIPS processor supports a simplified ISA with a minimal set of instructions. The processor implements a 16-bit instruction word with support for R-Type and I-Type instructions, along with branch instructions for the single-cycle.

The instruction formats, supported instructions, register file, control signals, and ALU control encoding are detailed in the following subsections.

3.2.1 Instruction Formats

R-Type Instruction Format:

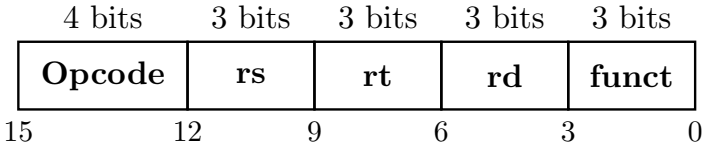


Figure 6: R-Type Instruction Format (16-bit)

I-Type Instruction Format:

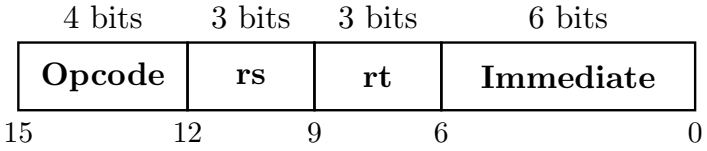


Figure 7: I-Type Instruction Format (16-bit)

3.2.2 Supported Instructions

The 16-bit MIPS processor supports the following instructions:

Instruction	Format	Opcode	Funct	Description
R-Type Instructions (Opcode = 0000)				
AND	R	0000	000	rd = rs & rt
OR	R	0000	001	rd = rs | rt
ADD	R	0000	010	rd = rs + rt
XOR	R	0000	011	rd = rs $\oplus$ rt
NOR	R	0000	100	rd = $\sim$(rs | rt)
Reserved	R	0000	101	*Reserved for future use*
SUBTRACT	R	0000	110	rd = rs - rt
SLT	R	0000	111	rd = (rs < rt) ? 1 : 0
I-Type Instructions				
ADDI	I	0001	—	rt = rs + SignExt(Imm)
SUBI	I	0010	—	rt = rs - SignExt(Imm)
LW	I	1000	—	rt = Mem[rs + SignExt(Imm)]
SW	I	1010	—	Mem[rs + SignExt(Imm)] = rt
BEQ	I	1100	—	if (rs == rt) PC += SignExt(Imm)

Table 2: Supported Instructions

3.3 Register File

3.3.1 Register File Specifications

The register file is responsible for storing and providing access to the processor's registers. It contains 8 general-purpose registers, each is 16-bit wide. The register `$0` is hardwired to the constant value 0. The remaining registers (`$1` to `$7`) can be used for general-purpose data storage and manipulation.

Register	Number	Description
$0	000	Constant value 0
$v0	001	General-purpose register 1
$a0	010	General-purpose register 2
$a1	011	General-purpose register 3
$t0	100	General-purpose register 4
$t1	101	General-purpose register 5
$s0	110	General-purpose register 6
$s1	111	General-purpose register 7

Table 3: Available Registers

3.3.2 Register File Interface

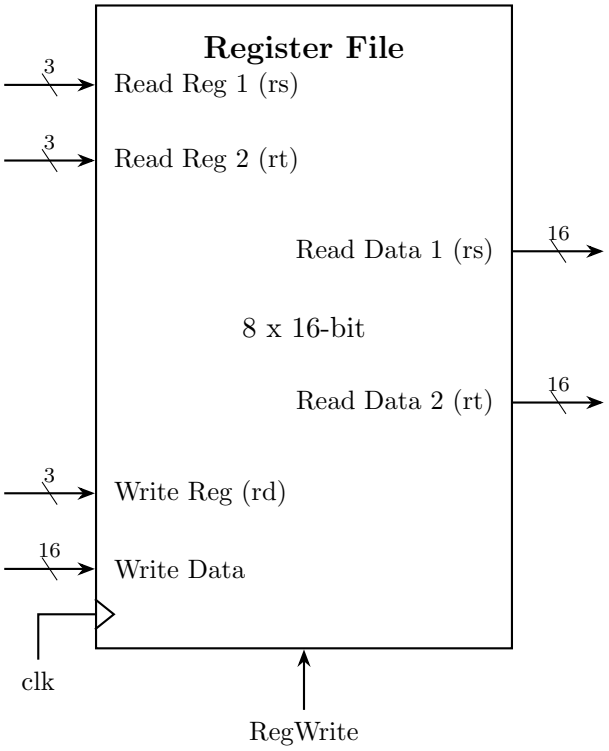


Figure 8: Register File Block Diagram

3.4 Control Signals

3.4.1 Control Unit

The control unit generates the necessary control signals based on the **opcode** and **funct** fields of the instruction. These control signals dictate the operation of various components within the processor, such as the ALU, register file, and memory. The control unit interface is illustrated in Figure 9.

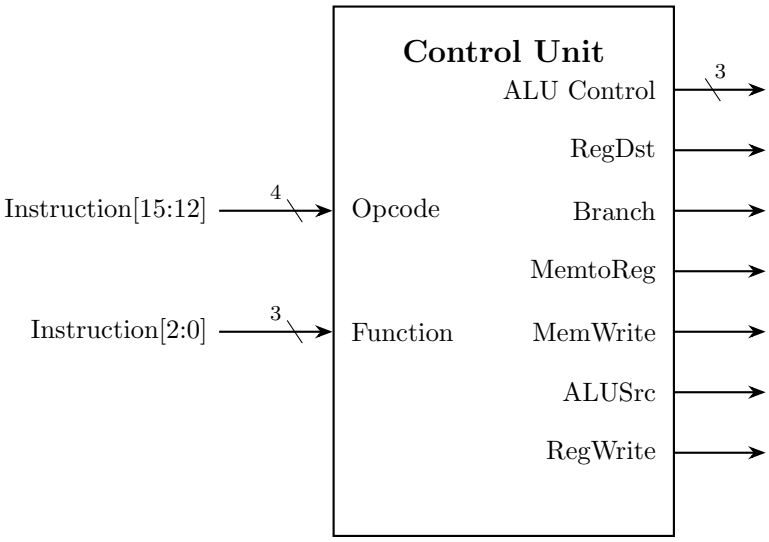


Figure 9: Control Unit Block Diagram

3.4.2 Control Signals Truth Table

Instruction	**Reg Dst**	**Branch**	**Mem To Reg**	**Mem Write**	**ALU Src**	**Reg Write**	**ALU Ctrl**
R-Type	1	0	0	0	0	1	funct
ADDI	0	0	0	0	1	1	010
SUBI	0	0	0	0	1	1	110
LW	0	0	1	0	1	1	010
SW	X	0	X	1	1	0	010
BEQ	X	1	X	0	0	0	110

Table 4: Instructions Control Signals

Note that 'X' indicates a *"don't care"*[3] condition for that particular control signal.

3.5 ALU Control Encoding

The 3-bit `ALUCtrl` signal specifies the operation to be performed by the ALU. The encoding is as follows:

ALUCtrl	**Operation**	**Description**
000	AND	Bitwise AND
001	OR	Bitwise OR
010	ADD	Addition
011	XOR	Bitwise XOR
100	NOR	Bitwise NOR
101	—	Reserved
110	SUBTRACT	Subtraction
111	SLT	Set on Less Than (signed)

Table 5: ALU Control Signal Encoding

Note that ALUCtrl takes the same value as the `funct` field for R-Type instructions (see Table 2).

4 Datapath Schematics

4.1 Single-Cycle Datapath

The single-cycle architecture executes each instruction in one clock cycle. The entire datapath is designed to accommodate the longest instruction execution time, which may lead to inefficiencies for simpler instructions.

[3] A condition where the value of the control signal does not affect the operation of the instruction.

4.1.1 Datapath Modules

- **Program Counter (PC):** Holds the address of the next instruction to be fetched.
- **Instruction Memory (IM):** Stores the program instructions.
- **Register File (RF):** Holds the general-purpose registers.
- **Control Unit:** Generates control signals based on the opcode of the instruction.
- **Arithmetic Logic Unit (ALU):** Performs arithmetic and logical operations.
- **Sign Extend:** Extends 6-bit immediate values to 16 bits.
- **Data Memory (DM):** Stores data for load and store instructions.
- **2-to-1 Multiplexers:** Selects between two data sources based on control signals.

4.1.2 Single-Cycle Datapath Schematic

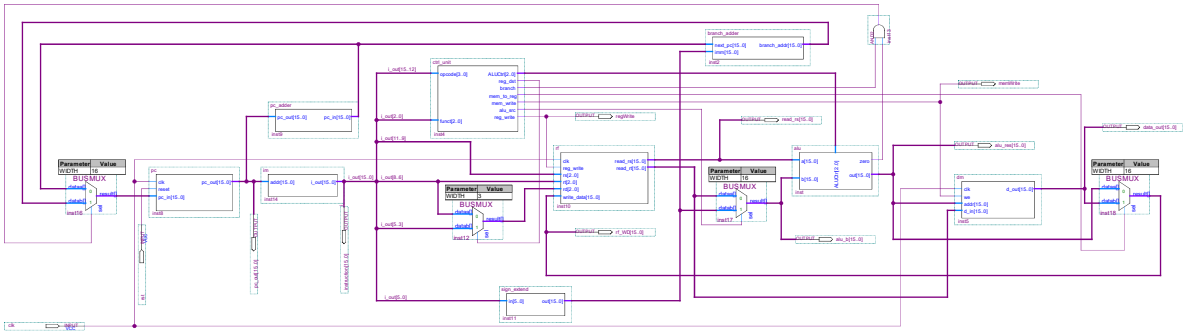


Figure 10: Single-Cycle Datapath Schematic

4.2 Pipelined Datapath

The pipelined architecture divides instruction execution into multiple stages, allowing for overlapping execution of instructions. This improves overall throughput but introduces complexity in handling hazards. It uses the same modules as the single-cycle architecture, but with additional pipeline registers between stages. For simplicity, no branch prediction or hazard detection mechanisms are implemented in this design.

4.2.1 Pipeline Stages

- **Instruction Fetch (IF):** Fetches the instruction from instruction memory.
- **Instruction Decode (ID):** Decodes the instruction and reads registers.
- **Execute (EX):** Performs ALU operations or calculates addresses.
- **Memory Access (MEM):** Accesses data memory for load/store instructions.
- **Write Back (WB):** Writes results back to the register file.

To separate these stages, pipeline registers are used to hold intermediate values between stages. This ensures that each stage can operate independently and concurrently. The registers used are as follows:

- **IF/ID Register:** Holds the instruction fetched from instruction memory between the IF and ID stages.
- **ID/EX Register:** Holds decoded instruction data, control signals, register values, and immediate values between the ID and EX stages.
- **EX/MEM Register:** Holds the ALU result, memory write data, control signals, and destination register between the EX and MEM stages.
- **MEM/WB Register:** Holds data from memory, ALU result, control signals, and destination register between the MEM and WB stages.

4.2.2 Pipelined Datapath Schematic

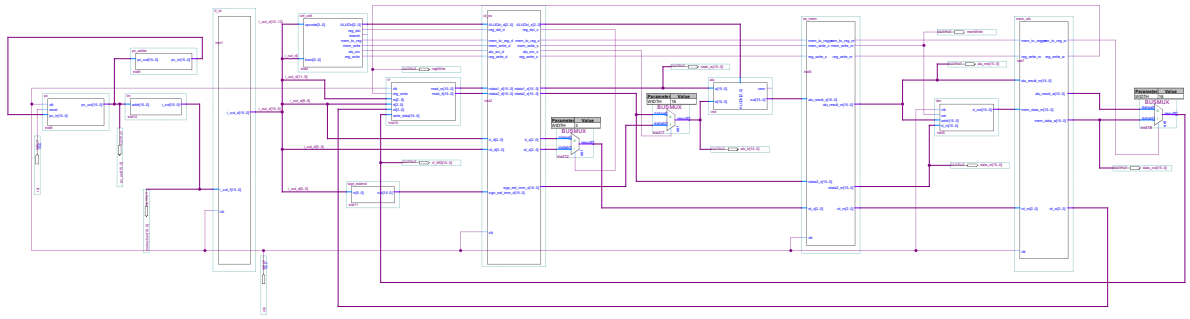


Figure 11: Pipelined Datapath Schematic

5 Waveform Simulation and Verification

Both processor designs were simulated and verified using `Quartus` with the help of `ModelSim`. Vector Waveform Files (VWF) were created to test the functionality of each processor design. The following subsections present the test instructions and the waveform results for both architectures.

The data memory is initialized to contain the following values before simulation:

Address	**Value**
0x00	0x12
0x01	0x02
0x02	0x22
0x03	0x20

Table 6: Initial Data Memory Values

Meaning, `Mem[0] = 0x0212` and `Mem[2] = 0x2022`, since it is a little-endian architecture. The rest of the memory is initialized to 0x00. The following sections detail the test instructions and waveform results for both processor architectures. Note that the introduced codes are assembly instructions that were converted to machine code (binary) before being loaded into the instruction memory.

5.1 Single-Cycle Verification

5.1.1 Single-Cycle Test Instructions

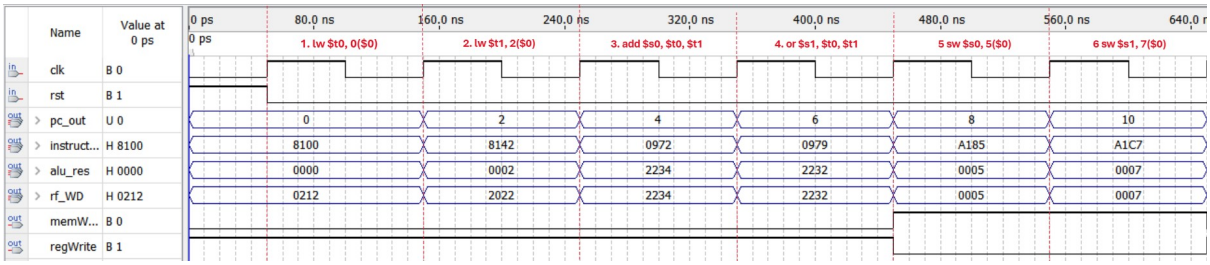
The following assembly code was used to test the single-cycle processor:

```
// Initialize Registers
lw   $t0, 0($0)      // 1. t0 = 0x0212
lw   $t1, 2($0)      // 2. t1 = 0x2022

// Perform Arithmetic and Logical Operations
add  $s0, $t0, $t1   // 3. s0 = 0x2234
or   $s1, $t0, $t1   // 4. s1 = 0x2232

```

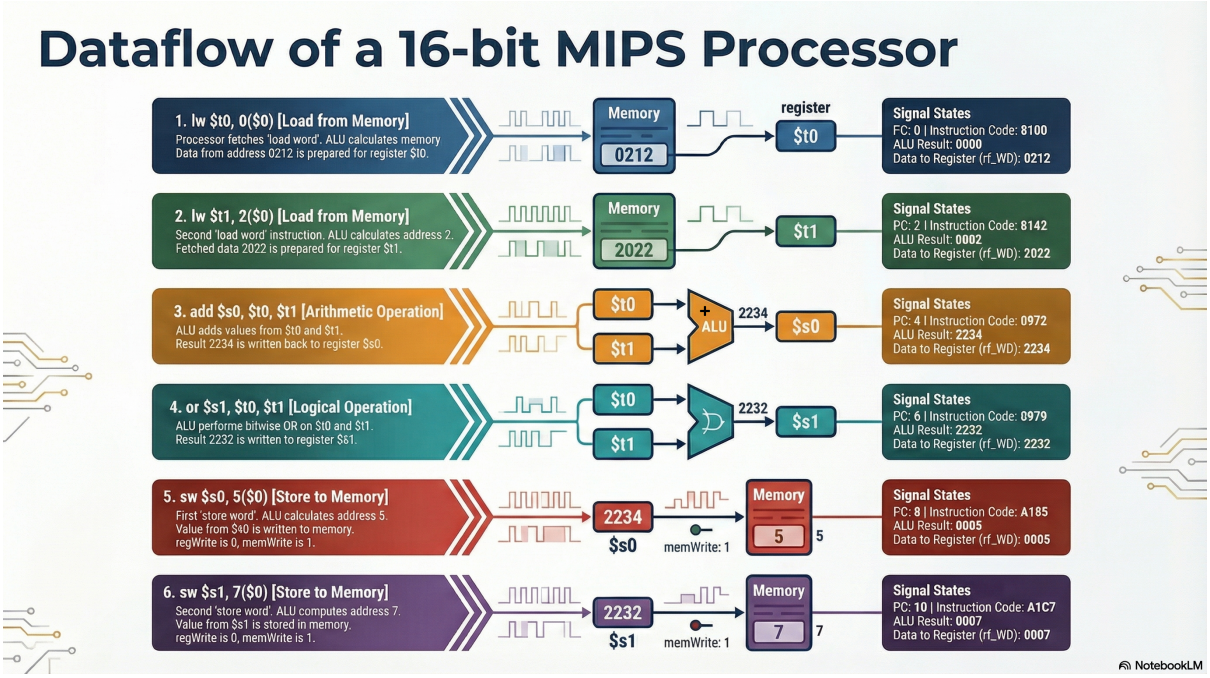
```
// Store Results Back to Memory
sw   $s0, 5($0)      // 5. Memory[0x05, 0x06] = 0x2234
sw   $s1, 7($0)      // 6. Memory[0x07, 0x08] = 0x2232
```

Listing 1: Single-Cycle Test Instructions

5.1.2 Single-Cycle Simulation Waveform



(a) Original waveform



(b) Waveform with annotations

Figure 12: Single-Cycle Simulation Waveform

5.2 Pipelined Verification

5.2.1 Pipelined Test Instructions

The following assembly code was used to test the pipelined processor:

```
// Initialize Registers
lw   $t0, 0($0)      // 1. t0 = 0x0212
lw   $t1, 2($0)      // 2. t1 = 0x2022
nop
nop
nop
// Perform Arithmetic and Logical Operations
add  $s0, $t0, $t1   // 3. s0 = 0x2234
or   $s1, $t0, $t1   // 4. s1 = 0x2232
// Pipeline RAW Hazards
nop
nop
// Store Results Back to Memory
sw   $s0, 5($0)     // 5. Memory[0x05, 0x06] = 0x2234
sw   $s1, 7($0)     // 6. Memory[0x07, 0x08] = 0x2232
```

Listing 2: Pipelined Test Instructions

Note that `nop` instructions were added between actual instructions to avoid *data hazards* in the pipelined architecture (stalling). **Data hazards** are situations where an instruction depends on the result of a previous instruction that has not yet completed its execution in the pipeline. In this test case, several data hazards were identified that required stalling to ensure correct execution.

The first hazard that is encountered is a *load-use hazard*[4] between the `LW` and `ADD` instructions. The rest of the hazards are *RAW hazards*[5] between the `ADDI` and `SW` instructions.

[4]A load-use hazard occurs when an instruction tries to use data that is being loaded by a previous instruction that has not yet completed.

[5]A read-after-write (RAW) hazard occurs when an instruction depends on the **result** of a previous instruction that has not yet completed.

5.2.2 Pipelined Simulation Waveform

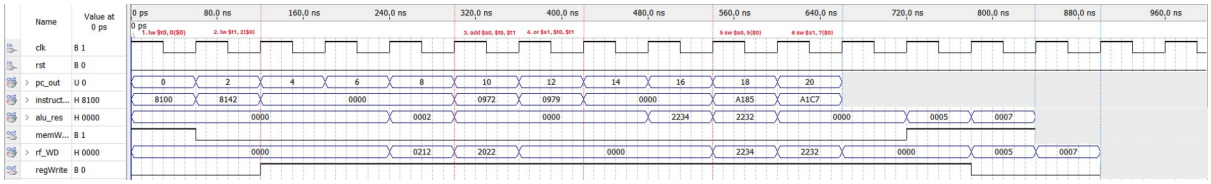


Figure 13: Pipelined Simulation Waveform

6 Timing Analysis (Performance Comparison)

6.1 Timing Analysis Methodology

Three main metrics were extracted from Quartus timing analysis reports for both architectures:

- **Maximum Frequency ($\mathbf{F_{max}}$):** The highest clock frequency at which the processor can operate without timing violations. This metric is reported from the Quartus compilation report.
- **Total Execution Time:** Execution Time = IC $\times$ CPI $\times$ Clock Cycle Time, where IC is the instruction count, CPI is cycles per instruction, and Clock Cycle Time is $1/F_{\text{max}}$.

For simplicity, no critical path analysis was performed. Instead, the maximum frequency reported by Quartus was used to calculate the clock cycle time for both architectures.

6.2 Compilation Summary

6.2.1 Timing Comparison

The timing analysis comparison between the single-cycle and pipelined architectures is summarized in Table 7.

Performance Metrics:

- **Maximum Frequency:** Was reported directly from Quartus compilation reports.
- **Clock Cycle Time:** Calculated as the inverse of the maximum frequency.
- **Instruction Count (IC):** Both architectures executed the same number of instructions (6 instructions).
- **Cycles Per Instruction (CPI):** The single-cycle architecture has a CPI of 1, while the pipelined architecture has a CPI of 2.5[6] due to the inserted stalls (`nop`) instructions to handle data hazards.
- **Total Execution Time:** Calculated using the formula mentioned in the methodology section.

Metric	Single-Cycle	Pipelined
Maximum Frequency (MHz)	61.55	93.18
Clock Cycle Time (ns)	16.25	10.73
Instruction Count	6	6
Cycles Per Instruction	1	2.5 (with stalls)
Total Execution Time (ns)	97.5	160.98

Table 7: Timing Analysis Comparison

6.3 Performance Comparison Summary

The pipelined processor exhibits a 65% increase in execution time (160.98 ns vs. 97.5 ns) despite a 51% higher clock frequency. This counterintuitive result stems from the absence of hazard detection and data forwarding mechanisms.

Root Cause: The 6 useful instructions require 5 manual `nops` to resolve data hazards:

[6] $\text{CPI}_{\text{pipe}} = \dfrac{\text{Total Cycles}}{\text{Instruction Count (IC)}} = \dfrac{\text{PipelineDepth} + N - 1}{6} = \dfrac{15}{6} = 2.5$

- 3 NOPs for load-use hazards (after `lw` instructions)
- 2 NOPs for RAW hazards (before `sw` instructions to wait for `$s0`/`$s1` writes)

Performance Impact:

- CPI increases from 1.0 to 2.5 (150% penalty)
- Pipeline efficiency: 54.5% (6 useful / 11 total instructions)
- Speedup: $\frac{97.5}{160.98} = 0.606$ (39.4% performance loss)

7 Conclusion

This report presented the design and implementation of a 16-bit MIPS processor in both single-cycle and pipelined architectures. The processor implements a 12-instruction ISA with separate 64-byte instruction and data memories in little-endian format, an 8-register file, and complete datapath components including PC, control unit, ALU, sign extension, and multiplexers. The single-cycle design executes each instruction in one clock cycle with straightforward control flow, while the pipelined architecture divides execution into five stages (IF, ID, EX, MEM, WB) separated by pipeline registers to enable instruction overlap.

Both designs were verified through waveform simulations. The performance analysis revealed that without hazard detection and data forwarding mechanisms, the pipelined processor exhibited 65% longer execution time despite higher clock frequency, requiring manual `nop` insertions that increased CPI from 1.0 to 2.5. This demonstrates that pipelining benefits are only realized with proper hazard management.

Future work should include implementing forwarding units, hazard detection hardware, branch prediction, an expanded instruction set, and larger memory spaces to fully leverage pipelined execution advantages.