Single Cycle Processor

Madiba Hudson-Quansah

Contents

Chapter 1

Instructions

Each instruction in MIPS is the same size (32 bits). There are three types of instructions in MIPS:

- R-Type: Register, example `add $t0, $t1, $t2`
- I-Type: Immediate, example `addi $t0, $t1, 100`
- J-Type: Jump, example `j 1000`

1.1 R-Type Instructions

Register operations. Arithmetic and logical operations directly on registers.

1.1.1 Structure of R-Type Instructions

opcode	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

For R-type instructions, the `opcode` field is always 0, i.e. 000000 as the `funct` determines the operation type. This is only for R-type instructions. Where:

`opcode` The operation code. Used to specify the operation.

`rs` The source register. Used to store the first operand.

`rt` The target register. Used to store the second operand.

`rd` The destination register. Used to store the result of the operation.

`shamt` The shift amount. Used to specify the number of bits to shift.

`funct` The function code. Used to specify the operation, example `add`, `sub`, `and`, `or`, etc.

1.2 I-Type Instructions

Memory access and immediate values.

opcode	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

Where:

`opcode` The operation code. Used to specify the operation.

`rs` The source register. Used to store the register operand

`rt` The target register. The desination register.

`immediate` The immediate value. Used to store the immediate value operand

1.3 J-Type Instructions

Branching and jumping.

`opcode`	`address`
6 bits	26 bits

Where:

`opcode` The operation code. Used to specify the operation.

`address` The address. Target address of the jump.

Chapter 2

Processor Design

In designing a processor for an ISA one goes through the following steps

Analyse the Instruction Set - To determine data path requirements and control signals

Select Data path components and clocking methodology - Based on the instruction set

Assemble the datapath meeting the requirements

Analyse the implementation of each instruction - To determine the needed control signals

Assemble the control logic - To generate the control signals

Taking into account only a subset of the MIPS instructions:

ALU Instructions (R-Type) - `add, sub, and, or, slt`

Immediate Instructions (I-Type) - `addi, lw, sw, beq`

Load and Store (I-Type) - `lw, sw`

Branch (I-Type) - `beq, bne`

Jump (J-Type) - `j`

2.1 Register Transfer Level (RTL)

Definition 2.1.1: Register Transfer Level (RTL)

A description of data flow between registers in a processor. It is a way of describing the implementation of a processors instructions at the register level.

Instructions are first fetched from memory using the **Program Counter (PC)** to get the address. The instruction is then decoded to determine the operation type and then executed using accordingly.

ADD - Reg (rd) $\leftarrow$ Reg (rs) + Reg (rt); $PC \leftarrow PC + 4$

In terms of processor design a hybrid Von-Nuemann Harvard architecture is used, where instruction memory is separate from data memory. In terms of MIPS there are 31 general purpose registers each 32-bit wide with $R0$ always being 0. The program counter is stored in the PC registers and an adder is used to increment the PC.

2.2 Data Path Components and Clocking Methodology

2.2.1 Data Path Components

Definition 2.2.1: Data Path

The part of the processor that is responsible for executing instructions by performing arithmetic operations, data movement and logical operations.

The data path is made up of the following components:

Registers - Small fast memory within the processor. Used to store data temporarily during execution. Registers can include

General Purpose Registers - Used to hold data and intermediate results during execution

Special-Purpose Registers - Used for specific purposes such as keeping track of the program counter (PC), instruction register (IR), stack pointer (SP), status flags, etc.

Arithmetic Logic Unit (ALU) - Performs arithmetic and logical operations on data. Arithmetic operations include addition, subtraction, multiplication and logical operations include AND, OR, NOT, XOR, etc.

Data Paths - The connections between registers, the ALU and other functional units of the processor.

Memory Interface - Enables communication between the processor and system memory (RAM). Includes components such as the memory address register (MAR) for storing memory addresses, and memory data register (MDR) for storing data read from or written to memory.

Multiplexers and Demultiplexers - For selecting between multiple data paths between different components such as registers and ALUs based on control signals generated by the control unit

Buses - Communication pathways that transfer data and control signals between different components of the CPU and system memory. Buses include the data bus, address bus, and control bus.

Instruction Fetch and Decode Logic - The instruction fetch unit retrieves instructions from memory based on the program counter (PC) and send them to the instruction decode unit which decodes the instruction to determine appropriate operation to execute.

Pipeline Stages - The data path can also be divided in to pipeline stages to improve throughput and performance. Each pipeline stage performs a specific task such as instruction fetch, decode, execute, and write-back.

Cache Interface - For storing frequently accessed data and instruction closer to the CPU cores for faster access.

Control Signals - Generated by the control unit to coordinate the operation of different components within the data path. Control signals determine which operations are performed by the ALU, which data paths are selected and when data is transferred between registers and memory.

The design of the Data path is critical to processor design as it determines how fast the processor can execute instructions and how efficiently it can perform computations. It must be optimized to ensure that instructions are executed correctly and efficiently while minimizing latency and maximizing throughput. Modern processors employ complex data paths with some of the following features:

- Pipelining
- Out-of-order execution
- Speculative execution

2.2.1.1 Pipelining

Definition 2.2.2: Pipelining

The execution of an instruction is broken down into multiple stages, i.e. fetch, decode, etc, then stages of instruction execution are executed in an overlapping manner allowing multiple instructions to executed simultaneously.

2.2.1.2 Out-of-order Execution

Definition 2.2.3: Out-of-order Execution

Instructions are executed in an order different from the order they appear in the program, based on data dependencies and resource availability. This allows the processor to execute instructions more efficiently by avoiding stalls and keeping the execution units busy.

2.2.1.3 Speculative Execution

Definition 2.2.4: Speculative Execution

The processor executes instructions under an execution branch before the result of the branch is known, i.e. if the branch is taken or not. This potentially allows the processor to be more performant by executing instructions that are likely to be executed.

2.2.2 Clocking Methodology

Definition 2.2.5: Clocking Methodology

The method used to synchronize the operations of the different components of the processor. This defines when data can be written to and read from.

We assume edge-triggered clocking where all state changes happen on the same clock edge. Data must be valid and stable before the clock edge and the output must be valid after the clock edge. Edge-trigged clocking allows data to be read and written during the same clock cycle, for example data could be read during the rising edge and written during the falling edge.

With edge-triggered clocking the clock cycle must be long enough to allow all operations to complete, therefore the clock cycle speed is determined by the time taken by slowest instruction. This shown by the equation:

$$T_{\text{cycle}} \geqslant T_{\text{clk-q}} + T_{\text{max\_comb}} + T_s$$

Where:

$T_{\textbf{cycle}}$ - The clock cycle time

$T_{\textbf{clk}}$ - The clock to output delay through register, i.e time taken for data to be read from a register

$T_{\textbf{max\_comb}}$ - The longest delay through combinatorial logic, i.e. the longest time taken for data to be processed (instruction execution)

T_s - Setup time, the time data must be stable before the clock edge

T_h - Hold time, the time data must be stable after the clock edge, normally satisfied since $T_{\text{clk-q}} > T_h$

2.2.2.1 Clock Skew

Definition 2.2.6: Clock Skew

The absolute difference in time between when two storage elements see a clock edge.

Clock skew arises from the fact that the clock signal uses different paths each with differing delays to reach state elements, like flip-flops. This has to be accounted for to ensure that the cycle time can accommodate all read and write operation irrespective of the time the registers they are clocked. This results in the new equation:

$$T_{\text{cycle}} \geqslant T_{\text{clk-q}} + T_{\text{max\_comb}} + T_s + T_{\text{skew}}$$

This skew amount can however be minimized by balancing the clock delays to all storage elements.

Chapter 3

Instruction Type Data Paths

3.1 R-Type Instructions

3.2 I-Type Instructions

3.3 J-Type Instructions

Chapter 4

Limitations of Single Cycle Processors

Each instruction execution can be categorized into the following categories based on their common stages:

ALU	Instruction Fetch	Decode / Register Read	ALU	Register Write

Load	Instruction Fetch	Decode / Register Read	Compute Address	Memory Read	Register Write

Store	Instruction Fetch	Decode / Register Read	Compute Address	Memory Write

Branch	Instruction Fetch	Register read / Branch Target	Compare and Update PC

Jump	Instruction Fetch	Decode / Update PC

With the Load instruction category being the slowest, meaning the cycle time must be long enough to accommodate the load instruction. This has several implications:

Clock Speed Constraint - The clock cycle time must be long enough to accommodate the longest instruction, i.e. the load instruction. This means that shorter instructions waste time waiting for the longer ones to complete and in the case where a shorter one completes before the cycle is over, the processor is idle for the rest of the cycle.

Resource Utilization - All resources such as ALUs, memory and registers are only used once per cycle. This means that the processor is not fully utilising its resources most of the time.

Complexity in Control - The control unit must manage all the operations in a single cycle increasing the complexity of design.

Power Consumption - Due to the need to complete all operations in a single cycle, the processor will need to consume more power drive all the functional units simultaneously.

Scalabity - As more instructions are added to the ISA, the cycle time will need to be adjusted to accommodate the longest instruction, placing a limit on the ability to scale the processor for better performance.

Peformance - Since the clock cycle time is constrained by the slowest instruction, the processor is not able to achieve the best performance by executing multiple instructions simultaneously.

4.1 Overcoming Limitations

To overcome the limitations of the single cycle processor, the following techniques can be employed:

Pipelining 2.2.1.1

Multi-Cycle Design - Breaks down an instruction execution into multiple stages, with each stage taking one cycle to execute. This allows the clock cycle time to be reduced as it is no longer constrained by the slowest instruction rather the slowest stage.

Superscalar Architecture - Involves the use of multiple execution units, i.e. multiple ALUs, allowing multiple instructions to be executed simultaneously. By fetching and dispatching multiple instructions per cycle, superscalar processors increase instruction throughput.

Out-of-order Execution 2.2.1.2

Branch Prediction and Speculative Execution - Predicts the outcome of a branch instruction before it is executed, allowing the processor to speculatively execute 2.2.1.3

Instruction Level Parallelism - Techniques such as loop unrolling, software pipelining, vectorization and instruction reordering can allow for more parallelism in instruction execution.

Caching - Improves performance by storing frequently accessed data and instructions closer to the processor cores reducing the latency of memory access.

Register Renaming - Dynamically allocates physical registers to logical registers, i.e. dynamically mapping concurrently used logical registers to avoid what is the hardware equivalent of a race condition , to avoid hazards and dependencies between instructions.

4.2 Multi-Cycle vs Pipelining

Feature	Multi-Cycle Processor Design	Pipelined Processor Design
Execution Phases	Instructions are executed over several cycles in the form of stages	Instructions are divided into stages that overlap execution
Hardware Utilization	Resources are reused across different cycles for different stages	More hardware resources are needed to execute multiple stages simultaneously
Control Complexity	Complex control unit normally implemented as a finite state machine (FSM)	Complex control logic needed to manage simultaneous execution of multiple stages
Cycle Time	Clock cycle time is determined by the slowest stage	Clock cycle time is determined by the longest stage
Performance	Instructions are completed in several cycles, but each cycle is faster than in a single-cycle processor	Instruction throughput can approach one instruction per cycle due to stage overlapping

4.3 Multi-Cycle Implementation

First break each instruction into five stages:

- Instruction Fetch (IF)
- Instruction Decode / Register Read or Target Address for jump/branch. (ID)
- Execution / Memory address computation or branch outcome. (EX)
- Memory Access / ALU instruction completion (MEM)
- Load instruction completion (WB)

Each stage takes one clock cycle to complete, i.e. the categories of instructions break down into the following cycle counts

ALU - 4

Load - 5

Store -4

Branch - 3

Jump - 2

Example 4.3.1

Question 1

Assuming the following operation times for components:

- Access time for instruction and data memories is 200 ps
- Delay in ALU and adders is 180 ps
- Delay in decode and register file access (read or write) is 150 ps
- Ignore all other delays in PC, mux, extender, and wires

Which of the following designs would be faster and by how much?

1. Single cycle processor
2. Multi-cycle processor

Assume the following instruction mix: 40% ALU, 20% Load, 10% Store, 20% Branch, 10% Jump

Solution:

ALU - 200 + 150 + 180 + 150 = 680ps
Load - 200 + 150 + 180 + 200 + 150 = 880ps
Store - 200 + 150 + 180 + 150 = 680ps
Branch - 200 + 150 + 180 = 530ps
Jump - 200 + 180 = 280ps

1. Each single cycle processor cycle has to be as long as the longest instruction, i.e. 880ps
2. Each multi cycle processor cycle has to be as long as the longest stage i.e. 200ps.

$$\begin{aligned}\text{CPI} &= 0.4 \times 4 + 0.2 \times 5 + 0.1 \times 4 + 0.2 \times 3 + 0.1 \times 2 \\ &= 3.8\end{aligned}$$

$$\therefore \text{Clock Period} = 3.8 \times 200 = 760$$

$$\frac{880}{760} = 1.16$$

$\therefore$ the multi-cycle design is faster by about 16 %

4.4 Pipelining Implementation

Consider a task that can be divided into k subtasks, where k subtasks are executed on k different stages. Each subtask requires one time unit and the total execution time of the task is k time units. Pipelining is to overlap the execution such that the k stages work in parallel on k different tasks, where each task leaves and enters the pipeline at a rate of one task per time unit.

Let τ_i be time delay in stage S_i. The Clock Cycle is the maximum stage delay, i.e. $\tau = \max(\tau_i)$, and accordingly the clock rate/frequency is $\frac{1}{\tau}$ or $\frac{1}{\max(\tau_i)}$. A pipeline can process n tasks in $k + n - 1$ cycles where k cycles are needed to complete the first task and $n - 1$ cycles are needed to complete the remaining tasks. This gives the following equation:

$$S_k = \frac{\text{Serial execution in cycles}}{\text{Pipelined execution in cycles}} = \frac{nk}{k + n - 1} \quad S_k = k \text{ for large } n$$

Given the five stages:

- Instruction Fetch (IF)
- Instruction Decode (ID)
- Execution (EX)
- Memory Access (MEM)
- Write Back (WB)

Example 4.4.1

Question 2

Consider a 5-stage instruction execution in which
- IF = ALU = Data Memory Access = 200 ps
- Register Read = Register Write = 150 ps

1. What is the clock cycle time for a single cycle processor
2. What is the clock cycle time for a pipelined processor
3. What is the speed-up factor of the pipelined processor

Solution:

1. 200 + 200 + 200 + 150 + 150 = 900ps
2. 200ps, i.e. max stage delay
3.

$$S_k = \frac{900}{200} = 4.5$$