Unit 3: Microprocessor I: Instruction Data Set, Machine Language

3.1. Introduction

Microprocessor Overview: A microprocessor is the brain of a computer system that executes instructions to perform tasks. It interprets and processes data based on a set of machine instructions, called the instruction set.

- Instruction Set: The collection of commands that a microprocessor can execute. The instruction set defines the operations the microprocessor can perform and the format for communicating these instructions.
- Machine and Assembly Languages: Programming microprocessors can be done at various levels:
- Machine Language: Binary code that the microprocessor directly executes.
- Assembly Language: A low-level representation of machine instructions, using symbolic code for easier human understanding.

3.2. Assembly Language

Low-Level Programming Language: Assembly language is closer to machine code but easier for humans to read and write. It uses mnemonics like `MOV`, `ADD`, `SUB` to represent instructions.

Assembly Syntax:

Mnemonics: Represent operations (e.g., `MOV`, `ADD`).

Operands: Represent data to be processed (registers, memory addresses, or constants).

Example: `MOV AX, 5` (Moves the value 5 into register AX).

Assembler: A software tool that converts assembly language code into machine code.

Advantages of Assembly:

Greater control over hardware resources.

Efficient memory and CPU utilization.

3.3. Machine Language

Binary Code: The set of binary instructions directly executed by the CPU (composed of 0s and 1s).

Opcode: Part of a machine language instruction specifying the operation (e.g., addition, subtraction).

Operands: Specifies the data or memory location involved in the operation.

Instruction Format: Machine language instructions have a fixed structure including opcode and operand(s).

Programming in Machine Language: While possible, writing directly in machine language is cumbersome and error-prone. It is done by advanced programmers or when extreme optimization is required.

Examples:

Binary: `10110000 00000001` (Move the value 1 into register AL).

Hexadecimal: `B001` (Same operation as above, in hex form).

3.4. Programming

Programming in Assembly and Machine Language:

- ➢ Requires an understanding of the underlying hardware architecture.
- ➢ Consists of loading data into registers, performing arithmetic or logical operations, storing results back to memory.

Debugging and Testing:

- ➢ Assembly language programs are debugged using tools like debuggers or simulators to inspect register contents, memory, and step through code.
- ➢ Machine code is tested at the hardware level.

Efficiency:

Writing efficient assembly code requires knowledge of hardware-specific details like instruction timing, pipeline usage, and memory access patterns.

3.5. Addressing Modes

Definition: Addressing modes define how the operand for an instruction is chosen or accessed by the microprocessor. They specify whether the operand is a constant, a register, or a memory location.

Types of Addressing Modes:

1. Immediate Addressing: The operand is a part of the instruction itself (e.g., `MOV AX, 5`).

2. Register Addressing: The operand is located in a register (e.g., `MOV AX, BX`).

3. Direct Addressing: The operand is located at a memory address provided in the instruction (e.g., `MOV AX, [1000h]`).

4. Indirect Addressing: The operand is located at a memory address held in a register (e.g., `MOV AX, [BX]`).

5. Indexed Addressing: The operand address is computed by adding a constant to a base address held in a register (e.g., `MOV AX, [BX+SI]`).

6. Relative Addressing: Used primarily in branch or jump instructions; the address is specified relative to the current program counter.

3.6. Lights, Camera, Action: Compiling, Assembling, and Loading

Compiling: Transforms high-level language code into assembly code.

High-level languages like C/C++ are compiled into assembly language, which is easier to convert to machine language.

Compilers optimize the code for speed and memory usage.

Assembling: Converts assembly language code into machine code (binary).

Assembler: A program that translates assembly language code to machine language.

Generates object files that contain machine code and symbol tables.

Linking and Loading:

Linker: Combines object files and resolves symbols between them to create an executable file Loader: Loads the executable into memory for the CPU to execute.

Final executable is loaded into memory, and the CPU begins executing the instructions starting from the entry point.

3.7. Odds and Ends

Endianness: Refers to how bytes are ordered in memory:

Big-endian: Most significant byte is stored at the lowest memory address.

Little-endian: Least significant byte is stored at the lowest memory address.

Instruction Pipelining: A CPU architecture technique where multiple instruction phases (fetch, decode, execute) are overlapped to improve efficiency.

Interrupts and Exceptions: Signals that alter the normal flow of execution, allowing the processor to respond to events like hardware signals or software errors.

Flag Registers: Store the status of various conditions (e.g., zero flag, carry flag) that affect decision-making in branching or arithmetic operations.

4. Microprocessor II: Control and Datapath Design – Single-Cycle Processor

4.1. Introduction

Datapath and Control: In a microprocessor, the datapath is the part of the CPU that performs all data processing operations (e.g., arithmetic, logical operations). The control unit manages the flow of data and the sequence of operations by generating control signals based on the current instruction.

Single-Cycle Processor: A processor design where each instruction is executed in exactly one clock cycle. This means that all operations (fetch, decode, execute, memory access, and write-back) are completed within a single clock cycle.

Trade-offs:

- Advantages: Simplicity of design and predictable instruction timing.
- Disadvantages: Inefficiency in handling complex instructions, leading to longer clock cycles to accommodate slower operations.

4.2. Performance Analysis

- Clock Cycle Time: The amount of time taken to complete a single clock cycle. In a single-cycle processor, this is determined by the instruction that takes the longest time to execute (critical path).
- Clock Rate (Frequency): The inverse of the clock cycle time. A shorter clock cycle time results in a higher clock rate and potentially better performance.
- CPI (Cycles per Instruction): In a single-cycle processor, CPI is always 1 since each instruction is executed in one cycle.

Performance Formula: Performance is generally measured using the equation:

$$\text{CPU Time} = \frac{\text{Number of Instructions} \times \text{Cycles per Instruction}}{\text{Clock Rate}}$$

For a single-cycle processor:

$$\text{CPU Time} = \frac{\text{Number of Instructions}}{\text{Clock Rate}}$$

Instruction Mix: Even though the CPI is 1, different types of instructions (e.g., ALU operations, memory access) may take different amounts of time to complete. This affects the critical path and overall clock speed.

Critical Path and Bottlenecks:

The critical path is the longest delay in the processor's datapath, which determines the overall clock cycle time. Instructions that involve memory access, multiplication, or division can create bottlenecks.
Optimizing the critical path is essential for improving the clock rate and overall performance of the processor.

4.3. Single-Cycle Processor

Structure of a Single-Cycle Processor:

1. Instruction Fetch (IF): The processor fetches the instruction from memory, using the Program Counter (PC) to identify the address.
2. Instruction Decode (ID): The fetched instruction is decoded to determine the operation and operands.
3. Execute (EX): The processor performs the arithmetic, logical, or memory operation specified by the instruction. The Arithmetic Logic Unit (ALU) typically performs this.
4. Memory Access (MEM): If the instruction involves a memory operation (load/store), the memory is accessed in this stage.
5. Write Back (WB): The result of the operation is written back to the register file if required.

Datapath Components:

Program Counter (PC): Holds the address of the current instruction being executed.

Instruction Memory: Stores the program instructions.

Register File: Stores operands and results. It contains general-purpose registers (e.g., R0, R1, R2, etc.).

ALU (Arithmetic Logic Unit): Performs arithmetic and logical operations.

Data Memory: Used to store and retrieve data for load and store instructions.

Muxes (Multiplexers): Used to select between multiple inputs, such as deciding which data to pass to the ALU.

Control Unit: Responsible for generating control signals based on the instruction. These signals control the flow of data and the operations performed by the datapath components.

Control Signals: Enable or disable specific parts of the processor (e.g., reading/writing to registers or memory, selecting the ALU operation).

Example control signals:

- `RegWrite`: Controls writing to the register file.
- `MemRead`, `MemWrite`: Controls reading and writing to memory.
- `ALUOp`: Specifies the operation to be performed by the ALU (e.g., addition, subtraction).

Example of Instruction Execution in Single-Cycle Processor:

1. R-type instruction (e.g., ADD R1, R2, R3):

- Fetch instruction.
- Decode operands from register file.
- Perform addition in the ALU.
- Write the result back to the register file.

2. Load Word (LW R1, 0(R2)):

- Fetch instruction.
- Decode base address from R2.
- Access data memory.
- Write the loaded word to register R1.

Limitations of Single-Cycle Processor:

- **Inefficiency for Complex Instructions:** The clock cycle must be long enough to accommodate the slowest instruction. This leads to inefficient execution of faster instructions.
- **Wasted Resources:** Many functional units (e.g., ALU, memory) are idle for part of the cycle when they are not being used by certain instructions.
- **Scalability:** As the instruction set becomes more complex, a single-cycle processor becomes less efficient because the critical path gets longer, leading to longer clock cycles.

In summary, while single-cycle processors are simple and straightforward in design, they are not efficient for handling complex instructions or modern instruction sets due to the fixed clock cycle time based on the slowest instruction. The need for a faster and more efficient design leads to the development of multi-cycle and pipelined processors.

5. Microprocessor III: Control and Datapath Design – MultiCycle and Pipelined Processors

5.1. Introduction

SingleCycle Limitation: In a singlecycle processor, all instructions are forced to complete in one clock cycle, which leads to inefficiency. The cycle time is determined by the longest instruction (e.g., memory operations), making it less efficient for faster instructions (e.g., ALU operations).

MultiCycle Processor: Solves this problem by breaking down the instruction execution into multiple stages. Each stage is executed in a separate clock cycle, allowing shorter cycle times.

Pipelining: A further optimization where multiple instructions are overlapped in execution, allowing for faster throughput.

5.2. Performance Analysis

Clock Cycle Time: In a multicycle processor, the clock cycle time can be shorter because each step (fetch, decode, execute, etc.) is performed in its own cycle. This reduces the critical path length.

CPI (Cycles per Instruction): Unlike singlecycle processors (where CPI = 1), multicycle processors have a CPI greater than 1 because each instruction takes several cycles to complete. However, the cycle time is shorter, which often leads to better overall performance.

Performance Formula: The CPU performance is evaluated using the formula:

$$\text{CPU Time} = \frac{\text{Number of Instructions} \times \text{CPI}}{\text{Clock Rate}}$$

In a multicycle processor, CPI varies depending on the type of instruction (e.g., load/store may take more cycles than ALU operations).

Instruction Mix: Different instructions take a different number of cycles, affecting overall performance. The more complex the instruction mix, the more cycles the processor may need per instruction.

5.3. MultiCycle Processor

Instruction Execution Breakdown:

The instruction execution process is divided into multiple smaller steps, with each step completed in a separate cycle.

The common steps for each instruction are:

1. **Instruction Fetch (IF):** Fetch the instruction from memory.
2. **Instruction Decode (ID):** Decode the instruction and read operands from the register file.
3. **Execution (EX):** Perform the operation (arithmetic or logical) or calculate the memory address.
4. **Memory Access (MEM):** Access data from memory for load/store instructions.
5. **Write Back (WB):** Write the result back to the register file.

Control and Datapath:

- In a multicycle processor, the control signals vary in each stage. The control unit generates different signals in each cycle to activate specific parts of the processor.
- Datapath components are reused across different cycles. For instance, the ALU may be used in both the address calculation step and the arithmetic operation step.
- Control Unit: More complex than in a singlecycle processor, as it must generate different control signals for each step of the instruction.

Advantages:

- **Shorter Clock Cycle Time:** Since each stage is designed to take a short time, the overall clock cycle time is reduced.
- **Efficient Resource Usage:** Functional units (like the ALU or memory) can be reused in different stages of instruction execution, reducing the need for multiple instances of these units.

Disadvantages:

- **Increased Control Complexity:** The control unit is more complicated because it has to manage multiple stages and different operations in each cycle.
- **Variable CPI:** Depending on the instruction, the number of cycles required varies, making the performance less predictable compared to a singlecycle processor.

5.4. Pipelined Processor

- Pipeline Concept: Pipelining involves breaking the execution of instructions into several stages and executing multiple instructions simultaneously in different stages of the pipeline. This leads to a significant increase in throughput.
- Analogy: Similar to an assembly line in a factory, where different parts of a product are assembled at different stages, pipelining allows new instructions to start before the previous ones have finished.

Stages in a Pipelined Processor:

1. Instruction Fetch (IF): Fetch the instruction from memory.
2. Instruction Decode (ID): Decode the instruction and read registers.
3. Execution (EX): Perform arithmetic/logical operation or calculate memory address.
4. Memory Access (MEM): Access data from memory (for load/store instructions).
5. Write Back (WB): Write the result back to the register file.

Pipeline Throughput: The primary advantage of pipelining is that it allows for the overlap of instruction execution. Once the pipeline is full, one instruction can be completed every cycle (ignoring stalls).

Latency: Time taken to complete a single instruction from start to finish (this remains constant).

Throughput: The rate at which instructions are completed (significantly increased with pipelining).

Control and Hazards:

- **Control Unit:** Generates control signals for each pipeline stage. A pipelined processor requires careful coordination of control signals to ensure smooth execution across multiple instructions.
- **Hazards:** Challenges in pipelining include dealing with different types of hazards:

1. Data Hazards: Occur when an instruction depends on the result of a previous instruction that hasn’t been completed yet.
   Solution: Techniques like data forwarding or stalling are used to resolve data hazards.
2. Control Hazards: Occur due to branch instructions, where the next instruction is not known until the current instruction completes.
   Solution: Branch prediction or pipeline flushing.
3. Structural Hazards: Occur when two instructions need the same hardware resource at the same time.
   Solution: Can be mitigated by adding more hardware or using resource scheduling.

Performance of Pipelined Processors:

- CPI: Ideally, a pipelined processor has a CPI close to 1. However, due to stalls and hazards, CPI can increase slightly.
- Clock Rate: The clock rate is determined by the time taken for the slowest pipeline stage. This is usually much faster than a singlecycle or multicycle processor, as each stage is designed to take a minimal amount of time.

- Pipeline Depth: More stages in a pipeline (deeper pipelines) generally increase throughput, but also make the processor more vulnerable to hazards.

Advantages of Pipelining:

1. Increased Throughput: Multiple instructions are in progress at the same time, leading to higher instruction completion rates.
2. Efficient Utilization of Resources: Pipelining allows better use of processor resources by keeping functional units busy.

Disadvantages:

1. Pipeline Hazards: The need to handle data, control, and structural hazards introduces complexity in both hardware and control.
2. Branch Penalty: The processor needs extra cycles to recover from a mispredicted branch, which lowers performance.

Pipeline Optimization:

Techniques like outoforder execution, superscalar pipelines, and speculative execution further optimize pipelined processors, allowing even more parallelism and improving performance.

In summary, both multicycle and pipelined processors improve upon the limitations of singlecycle designs by allowing more efficient use of the clock cycle and processor resources. Multicycle processors break instruction execution into smaller, more manageable stages, while pipelined processors allow for concurrent execution of instructions, maximizing throughput. Pipelining is a key technique used in modern microprocessor design.