Computer Abstractions and Technology

Madiba Hudson-Quansah

Contents

Chapter 1

Eight Great Ideas In Computer Architecture

- Design for Moore's Law
- Use abstraction to simplify design
- Make the common case fast
- Performance via parallelism
- Performance via pipelining
- Performance via prediction
- Hierarchy of memories
- Dependability via redundancy

1.1 Design for Moore's Law

Moore's law states that the number of transistors that can be placed on a chip doubles every two years. As computer designs can take years to complete and implement, it is important to have this in mind when designing to exploit increased transistor count for better performance, rather than relying solely on increasing clock speeds.

1.2 Use Abstraction to Simplify Design

Abstractions allow us to represent a design at different levels of representation with details irrelevant to the current level being hidden. This allows for greater productivity as designers can focus on the current level of abstraction.

1.3 Make the Common Case Fast

Optimize the design for the most common operations to achieve better performance in typical scenarios. This encourages focusing on optimizing the execution of frequently used operations / instructions

1.4 Performance via Parallelism

Parallelism involves executing multiple operations simultaneously to enhance performance. This idea underscores the importance of exploiting both instruction-level parallelism (ILP) and data-level parallelism (DLP) to achieve better overall system performance.

1.5 Performance via Pipelining

Pipelining is s technique that divides the execution of instruction into stages, allowing multiple instructions to be processed concurrently. This idea promotes the use of pipelining to improve instruction throughput and overall system performance.

1.6 Performance via Prediction

This idea involves the use of prediction mechanisms to foresee events and make intelligent decisions to optimize performance.

1.7 Hierarchy of Memories

The memory hierarchy involves organizing different levels of memory with varying speeds and sizes. This idea emphasizes the importance of using faster, smaller and more expensive memory close to the processor and larger, slower and cheaper memory farther away.

1.8 Dependability via Redundancy

This idea focuses on enhancing system reliability and dependability through redundancy. Techniques such as error detection and correction, redundant components, and fault-tolerant designs are employed to minimize the impact of hardware failures.

Chapter 2

Building processors

2.1 Semiconductors

Definition 2.1.1: Semiconductor

A material that has an electrical conductivity between that of a conductor and an insulator.

Definition 2.1.2: P-Type

A semiconductor material that has been doped with an electron acceptor element. This creates a material that has a net positive charge.

Definition 2.1.3: N-Type

A semiconductor material that has been doped with an electron donor element. This creates a material that has a net negative charge.

Definition 2.1.4: PNP Transistor

A type of transistor that consists of a layer of N-type semiconductor between two layers of P-type semiconductor.

Definition 2.1.5: NPN Transistor

A type of transistor that consists of a layer of P-type semiconductor between two layers of N-type semiconductor.

Semiconductors are used in the construction of transistors, which are the building blocks of modern computers. Transistors are made up of three parts:

- Gate
- Drain
- Source

Definition 2.1.6: Von Neumann Process

- Fetch instruction
- Decode instruction
- Determine Operand Address(es)
- Fetch Operands
- Perform operation
- Store result
- Repeat for next instruction

(F D E S)

Chapter 3

Performance

3.1 Defining Performance

Definition 3.1.1: Execution Time

The total time required for the computer to complete a task, including disk accesses, memory accesses, I/O operations, operating system overhead, and so on.

Definition 3.1.2: Performance

A measure of how quickly a computer system completes a task.

$$\text{Performance} = \frac{1}{\text{Execution Time}}$$

Or

$$\text{Performance} = \frac{\text{Number of Instructions}}{}$$

Definition 3.1.3: Throughput / Bandwidth

The number of tasks completed per unit time.

A decrease in execution time almost always results in an increase in throughput as more tasks can be completed in the same amount of time. In discussing the performance of individual computers such as desktops we are mostly concerned with execution time, and thus if we had computers X and Y we can determine the performance of X relative to Y:

$$n = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

This conveys that "X is n times faster than Y". Due to the definition of performance this can also be written:

$$n = \frac{\text{Execution time}_Y}{\text{Execution time}_X}$$

Note:-

The smaller execution time is always the denominator.

3.2 Measuring Performance

Time is the measure of computer performance, i.e. the computer that performs the same amount of work in the least amount of time is the fastest / most performant.

Definition 3.2.1: CPU (Execution) Time

The actual amount of time the CPU spends computing for a specific task. Does not include things like overhead from waiting for I/O or running other programs.

Definition 3.2.2: User CPU Time

The CPU time spent in a program itself, i.e. the time the CPU spends executing all the instructions in the program.

Definition 3.2.3: System CPU Time

The CPU time spent in the operating system performing tasks on behalf of the program.

Definition 3.2.4: Clock Cycle / Tick

The time for one clock period, usually of the processor clock, which runs at a constant rate. Where the clock period is the time taken to complete a clock cycle (e.g. 4 gigahertz / GHz). Designers derive the clock rate from this clock cycle as the inverse of the clock period, i.e.:

$$\text{Clock Rate} = \frac{1}{\text{Clock Period}}$$

3.3 Performance and it's Factors

Users and Designers examine performance differently, thus it is important to relate the two to determine the effect of a design change on user performance. As we are focusing on CPU performance we can relate the number of clock cycles a program takes to complete with the clock period, i.e.:

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time / Clock period}$$

Or

$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\text{Clock Rate}}$$

3.4 Instruction Performance

Another way to think of CPU performance is in instructions. As the program will require the CPU to execute instructions the execution time will depend on the number of instructions generated by the program and the average clock cycles needed to run all those instructions, i.e.:

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \text{Average Clock cycles per instruction}$$

Definition 3.4.1: Clock Cycles Per Instruction (CPI)

Average number of clock cycles per instruction for a program or program fragment

Since different instructions may take different amounts of time depending on what they do, CPI is an average of all the instructions executed in the program.

Example 3.4.1

Question 1

Suppose we have two implementations of the same instruction set architecture. Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program. Which computer is faster for this program and by how much?

Solution: Let I be the number of instructions in each program as they have the same ISA.

$$\begin{aligned}\text{CPU Clock Cycles}_A &= 2.0 \times I \\ \text{CPU Clock Cycles}_B &= 1.2 \times I\end{aligned}$$

$$\begin{aligned}\text{CPU Time}_A &= 250 \times 2.0 \times I \\ &= 500I \text{ ps} \\ \text{CPU Time}_B &= 500 \times 1.2 \times I \\ &= 600I \text{ ps}\end{aligned}$$

$$\begin{aligned}n &= \frac{600I}{500I} \\ &= 1.2\end{aligned}$$

Computer A is 1.2 times or 20% faster than B for this program

This now gives us the CPU time equation in terms of instruction count, clock cycles per instruction and clock period as

$$\text{CPU Time} = \text{Instruction count} \times CPI \times \text{Clock cycle time}$$

Or

$$\text{CPU Time} = \frac{\text{Instruction count} \times CPI}{\text{Clock Rate}}$$

Example 3.4.2

Question 2

A compiler designer is trying to decide between two code sequences for a particular computer. The hardware designers have supplied the following facts:

	CPI for each instruction class		
	A	*B*	*C*
CPI	1	2	3

For a particular high-level language statement, the compiler writer is considering two code sequences that require the following instruction counts: Instruction counts for each instruction class

	Instruction counts for each instruction class		
Code sequence	*A*	*B*	*C*
1	2	1	2
2	4	1	1

Which code sequence executes the most instructions? Which will be faster? What is the CPI for each sequence?

Solution: Sequence 1 executes $2 + 1 + 2 = 5$ instructions while Sequence 2 executes $4 + 1 + 1 = 6$ instructions.

$$\text{CPU Clock Cycles} = \sum_{i=1}^{n} (CPI_i \times C_i)$$

$$\begin{aligned} \text{CPU Clock Cycles}_1 &= (2 \times 1) + (1 \times 2) + (2 \times 3) \\ &= 10 \text{ ps} \\ \text{CPU Clock Cycles}_2 &= (4 \times 1) + (1 \times 2) + (1 \times 3) \\ &= 9 \text{ ps} \end{aligned}$$

$$\begin{aligned} CPI &= \frac{\text{CPU Clock Cycles}}{\text{Instruction Count}} \\ CPI_1 &= \frac{10}{5} \\ &= 2 \\ CPI_2 &= \frac{9}{6} \\ &= 1.5 \end{aligned}$$

Chapter 4

Layers of Abstraction

The layers of abstraction in a computer system is as follows:

4.1 Hardware / Digital Logic Layer

This layer is the physical level where computers are made up of transistors, logic gates and other electronic components. It is where the orchestration of electrons occurs to perform computations. At this level, we have the following components:

- Transistors
- Logic Gates
- Flip-Flops
- Multiplexers
- Demultiplexers

An explicit example of an abstraction at this layer is an integrated circuit with millions of transistors that implements an Arithmetic Logic Unit (ALU).

4.2 Microarchitecture Layer

This layer is where the organization and design of the processor is defined. It describes how hardware components like, ALU, registers, caches, and pipelines interact to implement the Instruction Set. An example the choice of a single cycle, multi-cycle or pipelined implementation of the processor.

4.3 Instruction Set Architecture (ISA) Layer

This layer is an abstract model that defines the interface between the hardware and software, describing the set of instructions that the processor can execute. Some examples include the MIPS32, x86, x86_64, ARM, and RISC-V instruction sets. The ISA defines the instruction formats, addressing modes, and the behaviour of each instruction.

4.4 Operating System Layer

This layer provides higher level abstractions for hardware resources and offers an interface through which applications can interact with the hardware. Some examples include GNU/Linux, Windows, and macOS. The OS manages resources such as memory, processes, and I/O devices, providing services like process scheduling, memory management, and file systems. The kernel is the core component of the OS that interacts directly with the hardware and manages system resources.

4.5 High-Level Language / Application Layer

This layer is where high-level programming languages (HLLs) and applications reside. HLLs provide abstractions that allow programmers to write code without needing to understand the underlying hardware details. An example of this is a web browser written in C++.