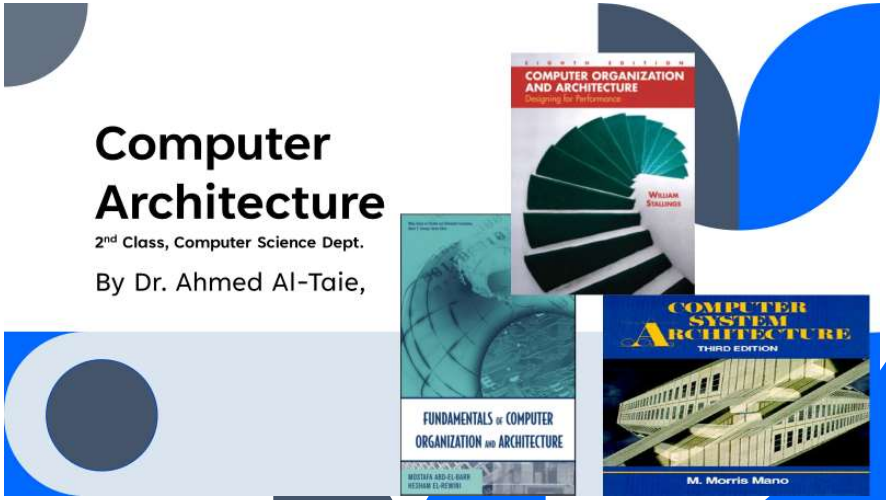
Computer Architecture

2nd Class, Computer Science Dept.

By Dr. Ahmed Al-Taie,

Chapter 1
Introduction to Computer Systems
Outline

- Historical Background
- Architectural Development and Styles
- Technological Development
- Performance Measures



PERFORMANCE MEASURES

- Assess the performance of a computer is **very important** issue.
- In order to **assess the performance** of a computer, we need to **develop performance measures**.
- There are various **facets to the performance** of a computer.
- For example, a **user of a computer** measures its performance based on the **time taken to execute a given job (program).**
- On the other hand, a **laboratory engineer** measures the performance of his system by the **total amount of work done in a given time**.
- While the user considers the **program execution time** a measure for performance, the laboratory engineer considers the **throughput** a more important measure for performance.
- A **metric for assessing** the **performance of a computer helps comparing alternative designs**.

PERFORMANCE MEASURES

- **Performance analysis** should **help answering questions** such as**: how fast can a program be executed using a given computer?**
- In order to answer such a question, we need to determine the **time taken by a computer to execute a given job**.
- We define the **clock cycle time** as the **time between two consecutive rising (trailing) edges of a periodic clock signal** (**Fig. 1.1**).
- **Clock cycles** allow **counting unit computations**, because the **storage of computation results** is synchronized with rising (trailing) **clock edges**. The **time required** to **execute a job by a computer** is often **expressed** in **terms of clock cycles**.

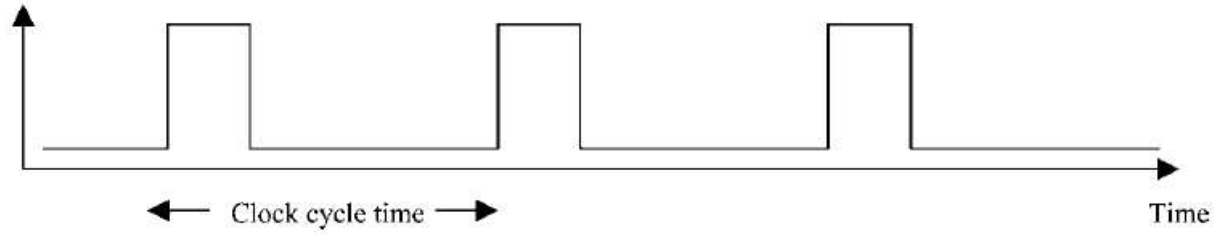


Figure 1.1 Clock signal

PERFORMANCE MEASURES

- We denote the **number of CPU clock cycles** for **executing a job** to be the **cycle count** (**CC**), the **cycle time** by **CT**, and the **clock frequency** by $f = 1/CT$.
- The **time taken** by the **CPU** to **execute a job** can be **expressed** as

$$CPU\ time = CC \times CT = CC/f$$

- It may be easier to count the number of instructions executed in a given program as compared to counting the number of CPU clock cycles needed for executing that program
- Therefore, the **average number of clock cycles per instruction** (**CPI**) has been used as an **alternate performance measure**. The following equation shows how to compute the CPI.

$$CPI = \frac{CPU\ clock\ cycles\ for\ the\ program}{Instruction\ count}$$

$$CPU\ time = Instruction\ count \times CPI \times Clock\ cycle\ time = \frac{Instruction\ count \times CPI}{Clock\ rate}$$

PERFORMANCE MEASURES

- It is **known** that the instruction set of a given machine **consists of** a **number of instruction categories**: **ALU** (**simple assignment** and **arithmetic** and **logic** instructions), **load, store, branch**, and **so on**.
- In the case that the **CPI** for each **instruction category** is **known**, the overall **CPI** can be computed as

$$CPI = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count}$$

- where Ii is the number of times an instruction of type i is executed in the program and CPIi is the average number of clock cycles needed to execute such instruction.

PERFORMANCE MEASURES

- **Example:** Consider computing the overall **CPI** for a machine **A** for which the following **performance measures** were recorded when executing a set of benchmark programs. Assume that the **clock rate** of the **CPU** is **200 MHz**.

Instruction category	Percentage of occurrence	No. of cycles per instruction
ALU	38	1
Load & store	15	3
Branch	42	4
Others	5	5

- Assuming the execution of 100 instructions, the overall CPI can be computed as

$$CPI_a = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count} = \frac{38 \times 1 + 15 \times 3 + 42 \times 4 + 5 \times 5}{100} = 2.76$$

PERFORMANCE MEASURES

- It should be noted that the **CPI** reflects the **organization** and the **instruction set architecture** of the processor while **the instruction count reflects** the **instruction set architecture** and **compiler** technology used.
- This shows the degree of interdependence between the two performance parameters.
- Therefore, it is imperative that both the **CPI** and the **instruction count** are considered in assessing the merits of a given computer or equivalently in comparing the performance of two machines.
- A different performance measure that has been given a **lot of attention in recent years** is **MIPS** (million instructions-per-second (the rate of instruction execution per unit time)), which is defined as

$$MIPS = \frac{Instruction\ count}{Execution\ time \times 10^6} = \frac{Clock\ rate}{CPI \times 10^6}$$

PERFORMANCE MEASURES

- **Example:** Suppose that the same set of benchmark programs considered above were executed on another machine, call it machine **B**, for which the following measures were recorded.

Instruction category	Percentage of occurrence	No. of cycles per instruction
ALU	35	1
Load & store	30	2
Branch	15	3
Others	20	5

- What is the MIPS rating for the machine considered in the previous example (machine A) and machine B assuming a clock rate of 200 MHz?

$$CPI_a = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count} = \frac{38 \times 1 + 15 \times 3 + 42 \times 4 + 5 \times 5}{100} = 2.76$$

$$MIPS_a = \frac{Clock\ rate}{CPI_a \times 10^6} = \frac{200 \times 10^6}{2.76 \times 10^6} = 70.24$$

PERFORMANCE MEASURES

$$CPI_b = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count} = \frac{35 \times 1 + 30 \times 2 + 20 \times 5 + 15 \times 3}{100} = 2.4$$

$$MIPS_b = \frac{Clock\ rate}{CPI_a \times 10^6} = \frac{200 \times 10^6}{2.4 \times 10^6} = 83.67$$

- **Thus MIPSb > MIPSa.**

It is interesting to note here that although MIPS has been used as a performance measure for machines, one has to be careful in using it to compare machines having different instruction sets. This is because MIPS does not track execution time. Consider, for example, the following measurement made on two different machines running a given set of benchmark programs.

PERFORMANCE MEASURES

$$CPI_a = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count} = \frac{(8 \times 1 + 4 \times 3 + 4 \times 3 + 2 \times 4) \times 10^6}{(8 + 4 + 4 + 2) \times 10^6} \cong 2.2$$

$$MIPS_a = \frac{Clock\ rate}{CPI_a \times 10^6} = \frac{200 \times 10^6}{2.2 \times 10^6} \cong 90.9$$

$$CPU_a = \frac{Instruction\ count \times CPI_a}{Clock\ rate} = \frac{18 \times 10^6 \times 2.2}{200 \times 10^6} = 0.198\ \text{s}$$

$$CPI_b = \frac{\sum_{i=1}^{n} CPI_i \times I_i}{Instruction\ count} = \frac{(10 \times 1 + 8 \times 2 + 4 \times 4 + 2 \times 4) \times 10^6}{(10 + 8 + 4 + 2) \times 10^6} = 2.1$$

$$MIPS_b = \frac{Clock\ rate}{CPI_a \times 10^6} = \frac{200 \times 10^6}{2.1 \times 10^6} = 95.2$$

$$CPU_b = \frac{Instruction\ count \times CPI_a}{Clock\ rate} = \frac{20 \times 10^6 \times 2.1}{200 \times 10^6} = 0.21\ \text{s}$$

$MIPS_b > MIPS_a$ and $CPU_b > CPU_a$

Instruction category	No. of instructions (in millions)	No. of cycles per instruction
Machine (A)		
ALU	8	1
Load & store	4	3
Branch	2	4
Others	4	3
Machine (B)		
ALU	10	1
Load & store	8	2
Branch	2	4
Others	4	3

PERFORMANCE MEASURES

- The example shows that although machine B has a higher MIPS compared to machine A, it requires longer CPU time to execute the same set of benchmark programs.
- Million floating-point instructions per second, MFLOP (rate of floating-point instruction execution per unit time) has also been used as a measure for machines' performance. It is defined as

$$MFLOPS = \frac{Number\ of\ floating\text{-}point\ operations\ in\ a\ program}{Execution\ time \times 10^6}$$

- While MIPS measures the rate of average instructions, MFLOPS is only defined for the subset of floating-point instructions.
- An argument against MFLOPS is the fact that the set of floating-point operations may not be consistent across machines and therefore the actual floating-point operations will vary from machine to machine.
- Yet another argument is the fact that the performance of a machine for a given program as measured by MFLOPS cannot be generalized to provide a single performance metric for that machine.

PERFORMANCE MEASURES

- The performance of a machine regarding one particular program might not be interesting to a broad audience.
- The use of arithmetic and geometric means are the most popular ways to summarize performance regarding larger sets of programs (e.g., benchmark suites). These are defined below.

$$Arithmetic\ mean = \frac{1}{n}\sum_{i=1}^{n} Execution\ time_i$$

$$Geometric\ mean = \sqrt[n]{\prod_{i=1}^{n} Execution\ time_i}$$

- where execution time i is the execution time for the ith program and n is the total number of programs in the set of benchmarks.

The following table shows an example for computing these metrics.

PERFORMANCE MEASURES

Item	CPU time on computer A (s)	CPU time on computer B (s)
Program 1	50	10
Program 2	500	100
Program 3	5000	1000
Arithmetic mean	1835	370
Geometric mean	500	100

- We conclude our coverage in this section with a discussion on what is known as the **Amdahl's law** for **speedup (SUo**) due to enhancement. In this case, we consider speedup as a measure of how a machine performs after some enhancement relative to its original performance. The following relationship formulates **Amdahl's law**.

PERFORMANCE MEASURES

$$SU_o = \frac{Performance\ after\ enhancement}{Performance\ before\ enhancement}$$

$$Speedup = \frac{Execution\ time\ before\ enhancement}{Execution\ time\ after\ enhancement}$$

- Consider, for example, a possible enhancement to a machine that will reduce the execution time for some benchmarks from 25 s to 15 s. We say that the speedup resulting from such reduction is SUo =25/15 = 1:67.

PERFORMANCE MEASURES

- In its given form, **Amdahl's law** accounts for cases whereby improvement can be applied to the instruction execution time.
- However, sometimes it may be possible to achieve performance enhancement for only a fraction of time, Δ. In this case a new formula has to be developed in order to relate the speedup, SU_Δ due to an enhancement for a fraction of time Δ to the speedup due to an overall enhancement, SU_O.
- This relationship can be expressed as

$$SU_o = \frac{1}{(1-\Delta) + (\Delta / SU_\Delta)}$$

PERFORMANCE MEASURES

- should be noted that when Δ = 1, that is, when enhancement is possible at all times, then $SU_o = SU_\Delta$, as expected.
- Consider, for example, a machine for which a speedup of 30 is possible after applying an enhancement. If under certain conditions the enhancement was only possible for 30% of the time, what is the speedup due to this partial application of the enhancement?

$$SU_o = \frac{1}{(1-\Delta) + (\Delta/SU_\Delta)} = \frac{1}{(1-0.3) + \frac{0.3}{30}} = \frac{1}{0.7 + 0.01} = 1.4$$

PERFORMANCE MEASURES

- It is interesting to note that the above formula can be generalized as shown below to account for the case whereby a number of different independent enhancements can be applied separately and for different fractions of the time, Δ_1, Δ_2, . . . , Δ_n, thus leading respectively to the speedup enhancements SU Δ_1 , SU Δ_2 , . . . , SU Δ_n .

$$SU_o = \frac{1}{[1 - (\Delta_1 + \Delta_2 + \cdots + \Delta_n)] + \frac{(\Delta_1 + \Delta_2 + \cdots + \Delta_n)}{(SU_{\Delta_1} + SU_{\Delta_2} + \cdots + SU_{\Delta_n})}}$$

Thank you