PERFORMANCE OF A COMPUTER

RESPONSE TIME AND THROUGHPUT

- **Response Time** : Also called execution time
- The time between the start and completion of a task
- **Throughput or Bandwidth**: Another measure of performance , it is the number of tasks completed per unit time.

Do the following changes to a computer system increase throughput, decrease response time, or both?

1. Replacing the processor in a computer with a faster version
2. Adding additional processors to a system that uses multiple processors for separate tasks—for example, searching the web

- Decreasing response time almost always improves throughput.
- Hence, in case1, both response time and throughput are improved.
- In case 2, no one task gets work done faster, so only throughput increases.
- If, however, the demand for processing in the second case was almost as large as the throughput, the system might force requests to queue up. In this case, increasing the throughput could also improve response time, since it would reduce the waiting time in the queue.

Performance

- To maximize performance, we want to minimize response time or execution time for some task.

$$\text{Performance}_X = \frac{1}{\text{Execution time}_X}$$

- This means that for two computers X and Y, if the performance of X is greater than the performance of Y, then , the execution time on Y is longer than that on X.
- Ie; X is faster than Y.

Performance

$$\text{Performance}_X > \text{Performance}_Y$$

$$\frac{1}{\text{Execution time}_X} > \frac{1}{\text{Execution time}_Y}$$

$$\text{Execution time}_Y > \text{Execution time}_X$$

- “X is n times faster than Y”—or equivalently “X is n times as fast as Y”—to mean

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

Measuring Performance

- Program execution time is measured in seconds per program.
- The most straightforward definition of time is called wall clock time, response time, or elapsed time.
- These terms mean the total time to complete a task, including disk accesses, memory accesses, input/output (I/O) activities, operating system overhead—everything.
- **CPU execution time or simply CPU time**, is the time the CPU spends computing for this task and does not include time spent waiting for I/O or running other programs.

CPU TIME

- CPU time can be further divided into
    1. User CPU time: The CPU time spent in a program itself.
    2. System CPU time: The CPU time spent in the operating system performing tasks on behalf of the program.

CPU Performance and Its Factors

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

Alternatively, because clock rate and clock cycle time are inverses,

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

The hardware designer can improve performance by reducing the number of clock cycles required for a program or the length of the clock cycle.

Instruction Performance

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

- The term clock cycles per instruction, which is the average number of clock cycles each instruction takes to execute, is often abbreviated as CPI.
- 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.

The Classic CPU Performance Equation

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

or, since the clock rate is the inverse of clock cycle time:

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

Comparing Code Segments

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:

Code sequence	Instruction counts for each instruction class		
	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?

Sequence 1 executes 2 + 1 + 2 = 5 instructions. Sequence 2 executes 4 + 1 + 1 = 6 instructions. Therefore, sequence 1 executes fewer instructions.

We can use the equation for CPU clock cycles based on instruction count and CPI to find the total number of clock cycles for each sequence:

$$\text{CPU clock cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{C}_i)$$

This yields

$$\text{CPU clock cycles}_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10 \text{ cycles}$$

$$\text{CPU clock cycles}_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9 \text{ cycles}$$

So code sequence 2 is faster, even though it executes one extra instruction. Since code sequence 2 takes fewer overall clock cycles but has more instructions, it must have a lower CPI. The CPI values can be computed by

$$\text{CPI} = \frac{\text{CPU clock cycles}}{\text{Instruction count}}$$

$$\text{CPI}_1 = \frac{\text{CPU clock cycles}_1}{\text{Instruction count}_1} = \frac{10}{5} = 2.0$$

$$\text{CPI}_2 = \frac{\text{CPU clock cycles}_2}{\text{Instruction count}_2} = \frac{9}{6} = 1.5$$

A given application written in Java runs 15 seconds on a desktop processor. A new Java compiler is released that requires only 0.6 as many instructions as the old compiler. Unfortunately, it increases the CPI by 1.1. How fast can we expect the application to run using this new compiler? Pick the right answer from the three choices below:

a. $\frac{15 \times 0.6}{1.1} = 8.2 \text{ sec}$

b. $15 \times 0.6 \times 1.1 = 9.9 \text{ sec}$

c. $\frac{15 \times 1.1}{0.6} = 27.5 \text{ sec}$