



University of Babylon, College of science for women
Dept. of Computer science

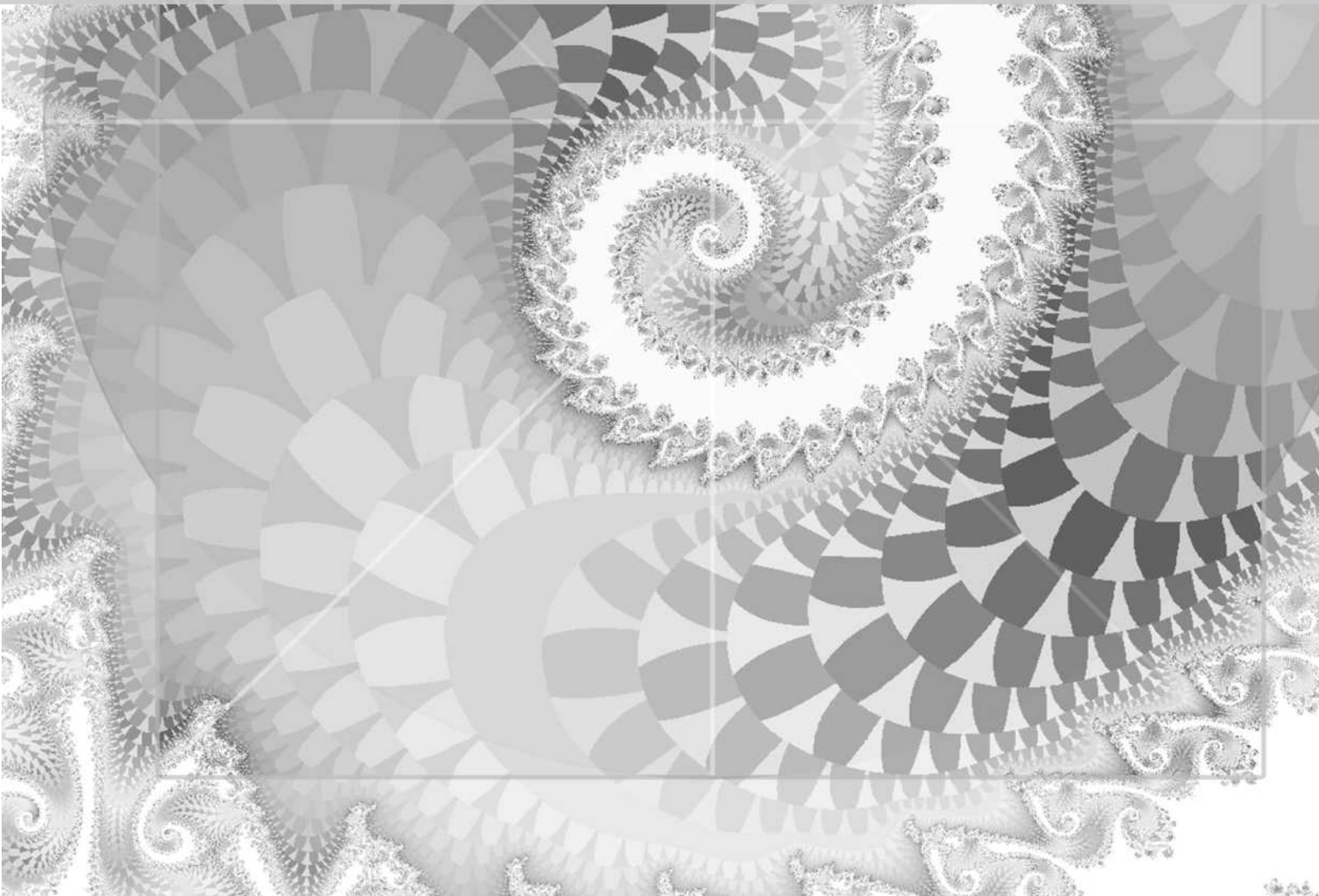
Computer Architecture

Second year

Dr. Salah Al-Obaidi

Lecture #8: Performance Issues

Spring 2024



Contents

Contents	i
10 Performance Issues	86
10.1 Microprocessor Speed	86
10.2 Basic Measures of Computer Performance	88

10. Performance Issues

This lecture addresses the issue of computer system performance.

10.1 Microprocessor Speed

In microprocessors, the addition of new circuits, and the speed boost that comes from reducing the distances between them, has improved performance four- or fivefold every three years or so since Intel launched its x86 family in 1978.

But the raw speed of the microprocessor will not achieve its potential unless it is fed a constant stream of work to do in the form of computer instructions. Accordingly, while the chipmakers have been busy learning how to fabricate chips of greater and greater density, the processor designers must come up with ever more elaborate techniques for feeding the processor. Among the techniques built into contemporary processors are the following:

- **Pipelining:** The execution of an instruction involves multiple stages of operation, including fetching the instruction, decoding the opcode, fetching operands, performing a calculation, and so on. Pipelining enables a processor to work simultaneously on multiple instructions by performing a different phase for each of the multiple instructions at the same time. The processor overlaps operations by moving data or instructions into a conceptual pipe with all stages of the pipe processing simultaneously. For example, while one instruction is being executed, the computer is decoding the next instruction. This is the same principle as seen in an assembly line.

- **Branch prediction:** The processor looks ahead in the instruction code fetched from memory and predicts which branches, or groups of instructions, are likely to be processed next. If the processor guesses right most of the time, it can prefetch the correct instructions and buffer them so that the processor is kept busy. The more sophisticated examples of this strategy predict not just the next branch but multiple branches ahead. Thus, branch prediction potentially increases the amount of work available for the processor to execute.
- **Superscalar execution:** This is the ability to issue more than one instruction in every processor clock cycle. In effect, multiple parallel pipelines are used.
- **Data flow analysis:** The processor analyzes which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions. In fact, instructions are scheduled to be executed when ready, independent of the original program order. This prevents unnecessary delay.
- **Speculative execution:** Using branch prediction and data flow analysis, some processors speculatively execute instructions ahead of their actual appearance in the program execution, holding the results in temporary locations. This enables the processor to keep its execution engines as busy as possible by executing instructions that are likely to be needed.

These and other sophisticated techniques are made necessary by the sheer power of the processor. Collectively they make it possible to execute many instructions per processor cycle, rather than to take many cycles per instruction.

Improvements in Chip Organization and Architecture

As designers wrestle with the challenge of balancing processor performance with that of main memory and other computer components, the need to increase processor speed remains. There are three approaches to achieving increased processor speed:

- **Increase the hardware speed of the processor.** This increase is fundamentally due to shrinking the size of the logic gates on the processor chip, so that more gates

can be packed together more tightly and to increasing the clock rate. With gates closer together, the propagation time for signals is significantly reduced, enabling a speeding up of the processor. An increase in clock rate means that individual operations are executed more rapidly.

- Increase the size and speed of caches that are interposed between the processor and main memory. In particular, by dedicating a portion of the processor chip itself to the cache, cache access times drop significantly.
- Make changes to the processor organization and architecture that increase the effective speed of instruction execution. Typically, this involves using parallelism in one form or another.

Traditionally, the dominant factor in performance gains has been in increases in clock speed due and logic density.

10.2 Basic Measures of Computer Performance

In evaluating processor hardware and setting requirements for new systems, **performance is one of the key parameters to consider, along with cost, size, security, reliability, and, in some cases, power consumption.** It is difficult to make meaningful performance comparisons among different processors, even among processors in the same family. Raw speed is far less important than how a processor performs when executing a given application.

The application performance depends on the following:

1. the raw speed of the processor.
2. the instruction set.
3. choice of implementation language.
4. efficiency of the compiler.
5. skill of the programming done to implement the application.

In this section, we look at some traditional measures of processor speed.

Clock Speed

Operations performed by a processor, such as fetching an instruction, decoding the instruction, performing an arithmetic operation, and so on, are governed by a system clock. Typically, all operations begin with the pulse of the clock. Thus, at the most fundamental level, the speed of a processor is dictated by the pulse frequency produced by the clock, measured in cycles per second, or **Hertz (Hz)**.

Typically, clock signals are generated by a quartz crystal, which generates a constant sine wave while power is applied. This wave is converted into a digital voltage pulse stream that is provided in a constant flow to the processor circuitry (Figure 10.1). For example, a 1-GHz processor receives 1 billion pulses per second. The rate of pulses is known as the **clock rate**, or **clock speed**. One pulse of the clock is referred to as a **clock cycle**, or a **clock tick**. The time between pulses is the **cycle time**.

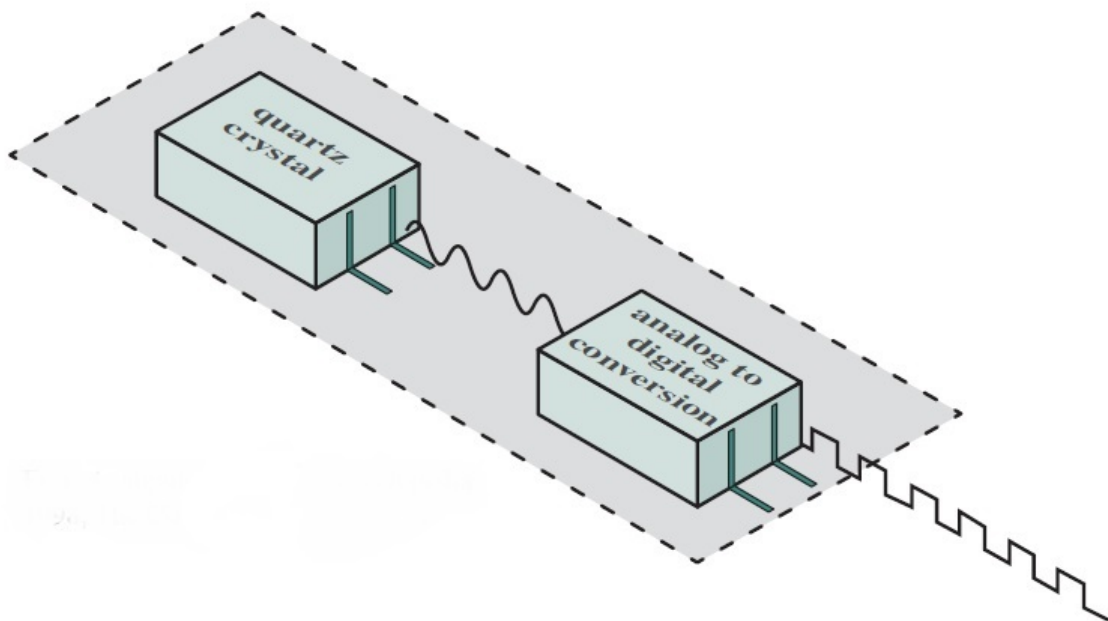


Figure 10.1: System Clock.

The clock rate is not arbitrary, but must be appropriate for the physical layout of the processor. Actions in the processor require signals to be sent from one processor element to

another. When a signal is placed on a line inside the processor, it takes some finite amount of time for the voltage levels to settle down so that an accurate value (**logical 1 or 0**) is available. Furthermore, depending on the physical layout of the processor circuits, some signals may change more rapidly than others. Thus, operations must be synchronized and paced so that the proper electrical signal (voltage) values are available for each operation.

The execution of an instruction involves a number of discrete steps, such as fetching the instruction from memory, decoding the various portions of the instruction, loading and storing data, and performing arithmetic and logical operations. Thus, most instructions on most processors require multiple clock cycles to complete. Some instructions may take only a few cycles, while others require dozens. In addition, when pipelining is used, multiple instructions are being executed simultaneously. Thus, a straight comparison of clock speeds on different processors does not tell the whole story about performance.

Instruction Execution Rate

A processor is driven by a clock with a constant frequency f or, equivalently, a constant cycle time τ , where $\tau = 1/f$. Define the instruction count, I_c , for a program as the number of machine instructions executed for that program until it runs to completion or for some defined time interval. Note that this is the number of instruction executions, not the number of instructions in the object code of the program. An important parameter is the average cycles per instruction (**CPI**) for a program. If all instructions required the same number of clock cycles, then CPI would be a constant value for a processor. However, on any given processor, the number of clock cycles required varies for different types of instructions, such as load, store, branch, and so on. Let CPI_i be the number of cycles required for instruction type i , and I_i be the number of executed instructions of type i for a given program. Then we can calculate an overall **CPI** as follows:

$$CPI = \frac{\sum_{i=1}^n CPI_i \times I_i}{I_c} \quad (10.1)$$

The processor time T needed to execute a given program can be expressed as

$$T = I_c \times CPI \times \tau \quad (10.2)$$

We can refine this formulation by recognizing that during the execution of an instruction, part of the work is done by the processor, and part of the time a word is being transferred to or from memory. In this latter case, the time to transfer depends on the memory cycle time, which may be greater than the processor cycle time. We can rewrite the preceding equation as

$$T = I_c \times [p + (m \times k)] \times \tau \tag{10.3}$$

where p is the number of processor cycles needed to decode and execute the instruction, m is the number of memory references needed, and k is the ratio between memory cycle time and processor cycle time. The five performance factors in the preceding equation (I_c, p, m, k, t) are influenced by four system attributes:

1. the design of the instruction set
2. compiler technology (how effective the compiler is in producing an efficient machine language program from a high-level language program)
3. processor implementation;
4. cache and memory hierarchy

Table 10.1 is a matrix in which one dimension shows the five performance factors and the other dimension shows the four system attributes. An **X** in a cell indicates a system attribute that affects a performance factor.

Table 10.1: Performance Factors and System Attributes

	I_c	p	m	k	τ
Instruction set architecture	X	X			
Compiler technology	X	X	X		
Processor implementation		X			X
Cache and memory hierarchy				X	X

A common measure of performance for a processor is the rate at which instructions are executed, expressed as millions of instructions per second (**MIPS**), referred to as

the **MIPS rate**. We can express the **MIPS** rate in terms of the clock rate and **CPI** as follows:

$$MIPSrate = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6} \quad (10.4)$$

Another common performance measure deals only with floating-point instructions. These are common in many scientific and game applications. Floating-point performance is expressed as millions of floating-point operations per second (**MFLOPS**), defined as follows:

$$MFLOPSrate = \frac{\text{Number of executed floating point operations in a program}}{\text{Execution time} \times 10^6} \quad (10.5)$$

EXAMPLE 2.2 Consider the execution of a program that results in the execution of 2 million instructions on a 400-MHz processor. The program consists of four major types of instructions. The instruction mix and the *CPI* for each instruction type are given below, based on the result of a program trace experiment:

Instruction Type	<i>CPI</i>	Instruction Mix (%)
Arithmetic and logic	1	60
Load/store with cache hit	2	18
Branch	4	12
Memory reference with cache miss	8	10

The average *CPI* when the program is executed on a uniprocessor with the above trace results is $CPI = 0.6 + (2 \times 0.18) + (4 \times 0.12) + (8 \times 0.1) = 2.24$. The corresponding MIPS rate is $(400 \times 10^6)/(2.24 \times 10^6) \approx 178$.