Computer Organization and Architecture
Week 4

Clock Speed and Instructions Per Second

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.

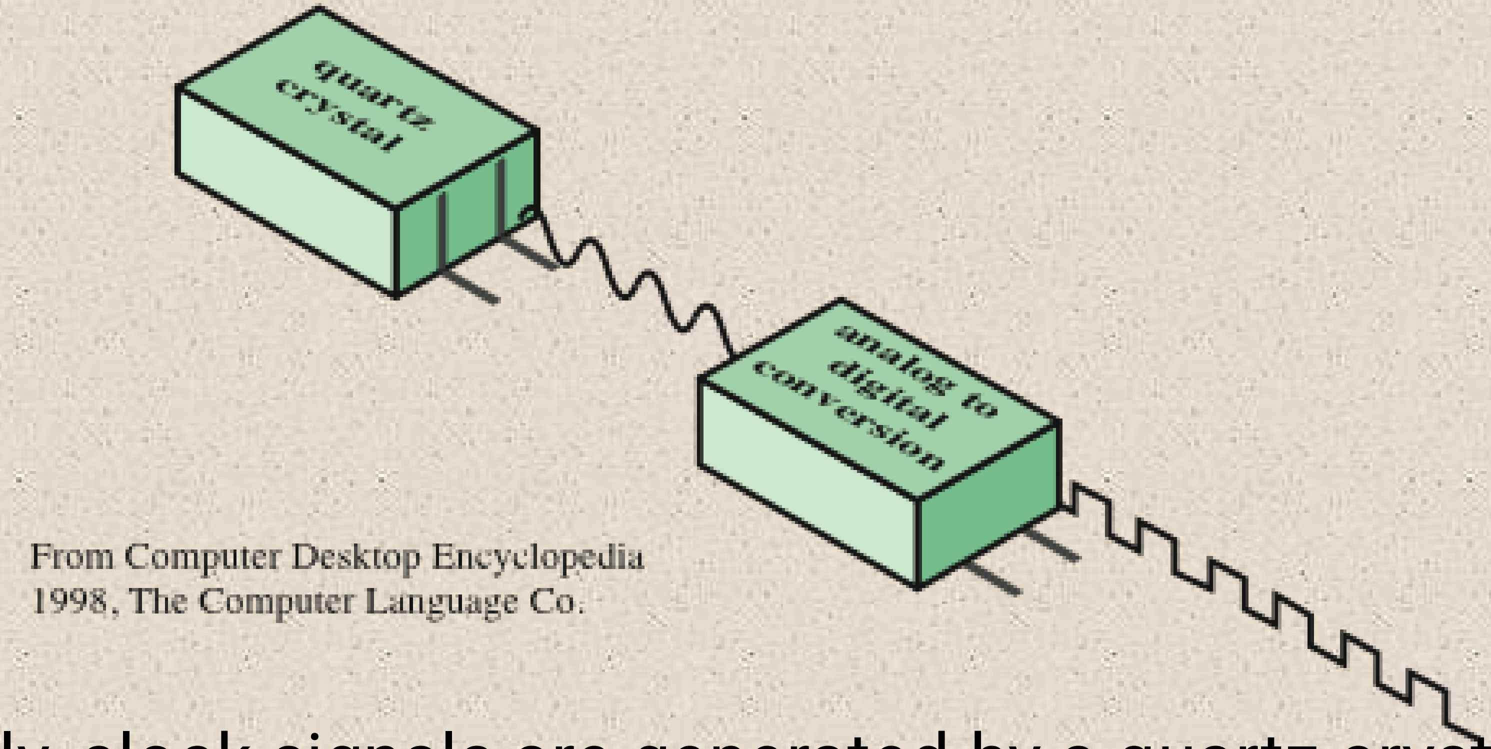
Typically, all operations begin with the pulse of the clock. Thus, at the most fundamental level, the speed of a processor is dictated by the pulse frequency produced by the clock, measured in cycles per second, or Hertz (Hz).

+

Common Prefixed Units / Frequency

Value	Symbol	Name	Amount
10^2	Hz	hertz	**cycles per second**
10^3 Hz	kHz	kilohertz	1,000
10^6 Hz	MHz	megahertz	1,000,000
10^9 Hz	GHz	gigahertz	1,000,000,000
10^{12} Hz	THz	Terahertz	1,000,000,000,000
10^{15} Hz	PHz	petahertz	1,000,000,000,000,000

System Clock

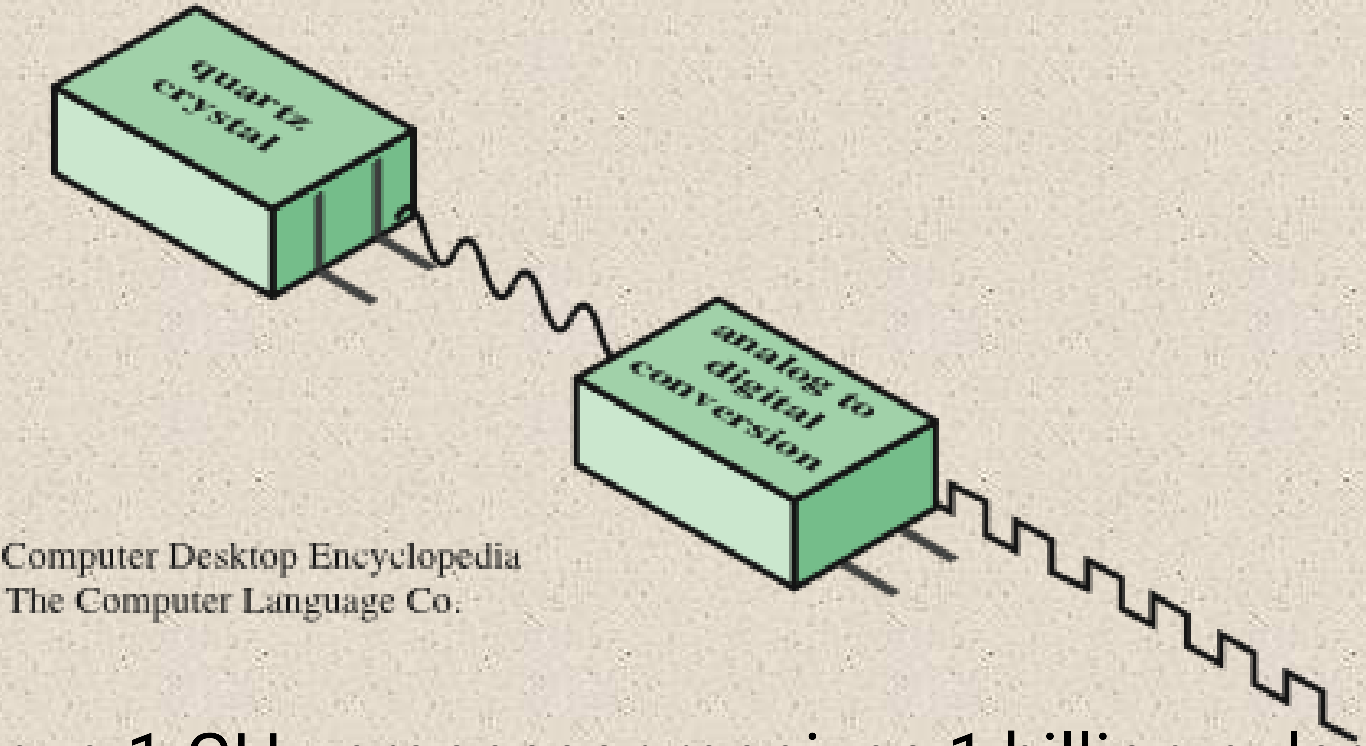


From Computer Desktop Encyclopedia
1998, The Computer Language Co.

Typically, clock signals are generated by a quartz crystal, which generates a constant signal 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 2.13 System Clock

+ System Clock



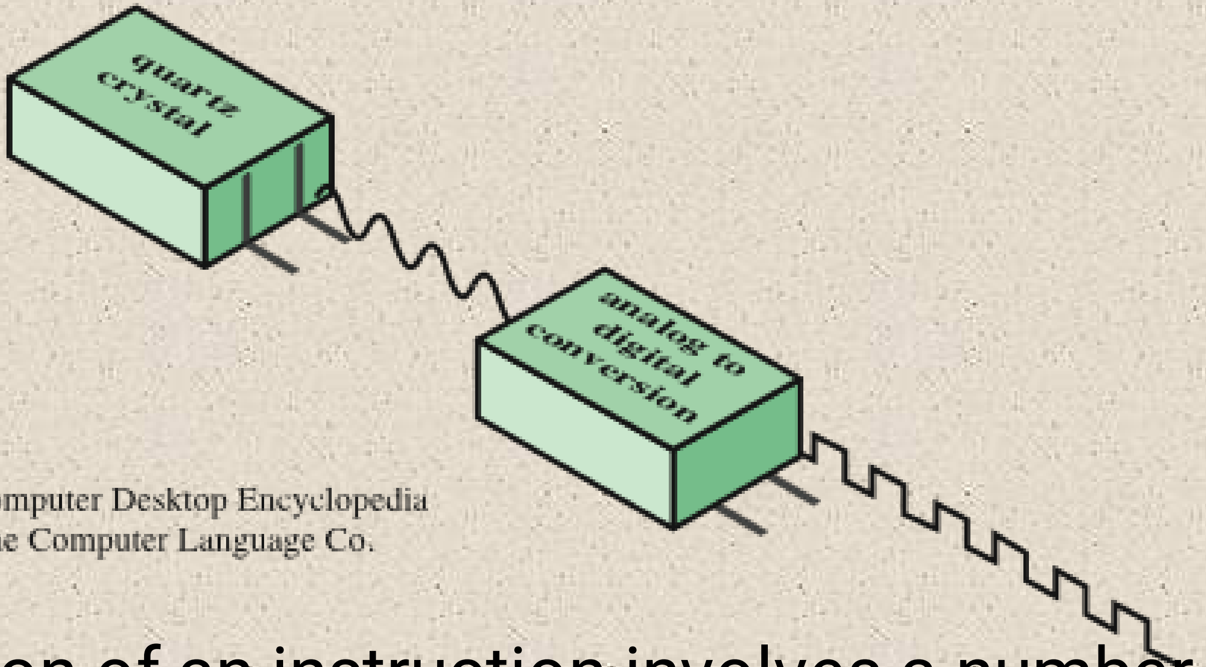
From Computer Desktop Encyclopedia
1998, The Computer Language Co.

Figure 2.13 System Clock

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 increment, or pulse, of the clock is referred to as a clock cycle, or a clock tick. The time between pulses is the **cycle time.**

System Clock



From Computer Desktop Encyclopedia
1998, The Computer Language Co.

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.

Figure 2.13 System Clock

Thus, most instructions on most processors require multiple clock cycles to complete. Some instructions may take only a few cycles, while others require dozens.

Terminologies

- CPI – Cycle Per Instruction
- I_c – Instruction Count (Total number of instruction to be executed to perform each instruction type– For CPI average)
- I_i – Instruction Type (Can be load, store, branch or so on)
- $\tau = {}^{1}/_{f}$ (Tau equals to 1 upon frequency(of the processor))
- $CPI = \sum_{i=1}^{n} (CPI_i \; * \; I_i)$
- $CPI = \frac{\sum_{i=1}^{n} (CPI_i * I_i)}{I_c}$ (For Average Cycles Per Instruction)

Cycles Per Instruction (CPI)

- Suppose there are 3 types of instructions e.g. Addition, Subtraction and Multiplication.
- CPI_i be the number of cycles required for instruction type i
- I_i be the number of executed instructions of type i for given program

+ Cycles Per Instruction (CPI)

- For Example
  - Addition (CPI_i =2 and I_i=4)
  - Subtraction (CPI_i =2 and I_i=3)
  - Multiplication (CPI_i =8 and I_i=5)

<u>According to formula</u>

$$CPI = \sum_{i=1}^{n} (CPI \; * \; I_i)$$

$CPI = (CPI_1 * I_1) + (CPI_2 * I_2) + (CPI_3 * I_3)$

$CPI = (2*4) + (2*3) + (8*5)$

$CPI = 8 + 6 + 40$

$CPI = 54$

Average Cycles Per Instruction (CPI)

- For Example
  - Addition (CPI_i =2 and I_i=4)
  - Subtraction (CPI_i =2 and I_i=3)
  - Multiplication (CPI_i =8 and I_i=5)
- $I_c = I_1+I_2+I_3 = 4+3+5 = 12$

According to formula

$$CPI = \frac{\sum_{i=1}^{n}(CPI * I_i)}{I_c}$$

$CPI = (CPI_1 * I_1)+ (CPI_2 * I_2)+ (CPI_3 * I_3)$

CPI = (2*4) + (2*3) + (8*5)

CPI = 8 + 6 + 40

CPI = 54/12

CPI = 4 Average CPI

τ (Tau Calculation)

- $\tau = {}^{1}/_{f}$

f = 1 GHz (For example)

f = 1 GHz = 10^9 Hz

$\tau = {}^{1}/_{10^9}$

$\tau = 10^{-9}$

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

$T = I_c * CPI * \tau$

$T = 12 * 54 * 10^{-9}$

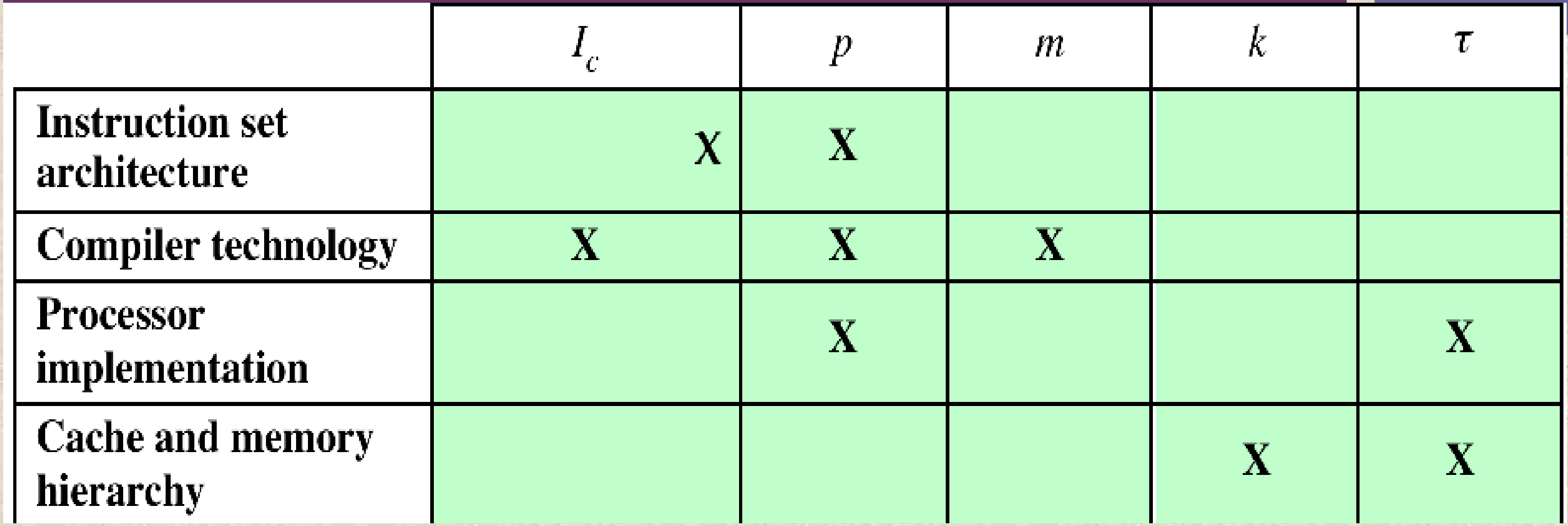
$T = 648 * 10^{-9}$

T= 648 Nano sec

<u>Measurement</u>

10^{-3}	Millie Sec
10^{-6}	Micro Sec
10^{-9}	Nano Sec
10^{-12}	Pico Sec

Five Performance Factors are influenced by Four 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

p =Number of processors
m =Number of memory references
k = ratio between memory cycle time and processor cycle time

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.***

MIPS (Millions of Instructions Per Second)

$$\text{MIPS} = \frac{I_C}{T * 10^6} = \frac{f}{CPI * 10^6}$$

MIPS (Millions of Instructions Per Second)

$$\text{MIPS} = \frac{I_c}{T * 10^6}$$

$$\text{MIPS} = \frac{12}{648 * 10^{-9} * 10^6}$$

$$\text{MIPS} = \frac{12}{648 * 10^{-3}}$$

$$\text{MIPS} = 0.0185185185185185 * 10^3$$

$$\text{MIPS} = 18.518$$

MIPS (Millions of Instructions Per Second)

$$\text{MIPS} = \frac{f}{CPI * 10^6}$$

$$\text{MIPS} = \frac{1 * 10^9}{54 * 10^6}$$

$$\text{MIPS} = \frac{1{,}000{,}000{,}000}{54{,}000{,}000}$$

$$\text{MIPS} = 0.0185185185185185 * 10^3$$

$$\text{MIPS} = 18.518$$

Another Example

Consider the execution of a program that results in the execution of 2 million instructions on a 400-MHz processor. The Instruction types are given below.

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

- CPI=(1*0.6)+(2*0.18)+(4*0.12)+(8*0.10)

CPI=0.6+0.36+0.48+0.8

CPI = 2.24

$$\text{MIPS rate} = \frac{f}{CPI * 10^6} = \frac{400 * 10^6}{2.24 * 10^6} = \frac{400000000}{2240000} = 178.57$$

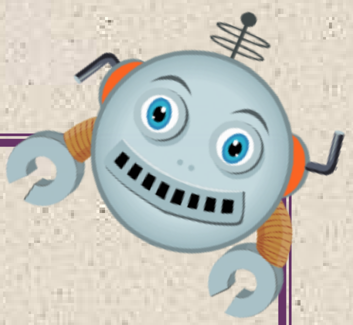
Exercise

Instruction Type	Instruction Count	Cycles per Instructions
Integer Arithmetic	45000	1
Data Transfer	32000	2
Floating Point	15000	2
Control Transfer	8000	2

A program is run on a 40 MHz Processor.

Determine the CPI and MIPS rate

Benchmarks



For example, consider this high-level language statement:

A = B + C /* assume all quantities in main memory */

With a traditional instruction set architecture, referred to as a complex instruction set computer (CISC), this instruction can be compiled into one processor instruction:

```
add mem(B), mem(C), mem (A)
```

On a typical RISC machine, the compilation would look something like this:

```
load mem(B), reg(1);
load mem(C), reg(2);
add reg(1), reg(2), reg(3);
store reg(3), mem (A)
```