CSE261: Computer Architecture

7. Performance

Seongil Wi

UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY
2 0 0 7

Department of Computer Science and Engineering

Notification: Midterm Exam

- Oct. 24 (Thursday)
- Class Time (1h 15m), Closed book

- T/F problems + Computation problems + Descriptive problems
- Scope: everything learned from September 3 to October 17
  - ***Understanding is important!***
  - The MIPS reference card will be provided. Do not memorize the content about it.

- If you are taking Linear Algebra (MTH20401), please send me an email (Those who have already sent an email are excluded)

Q&A Session for Your Midterm Exam

- Today, after the class
  - 45 minutes lecture
    - It is okay to leave the room after the lecture is end
  - 30 minutes Q&A session

Recap: Floating-point Number

We need a way to represent ...

- *Infinite decimal (e.g., 3.1415926535…)* → *Approximate value* → 3.1415
- *Very small numbers* → *Floating decimal point* → 0.001×10^{-20}
- *Very large numbers* → *Floating decimal point* → 3.15576×10^{19}

Can be represented with a limited number of bits!

Solution: Floating-point Number Representation

Recap: IEEE 754 Floating-point Standard

- Developed in response to *divergence of representations*

Divergence of representations

↓

Normalized representation

↓

IEEE 756 representation

$$0.11_{two} = 1.1_{two} \times 2^{-1} = 11_{two} \times 2^{-2}$$

$$1.1_{two} \times 2^{-1}$$

S	Exponent	Fraction

Recap: IEEE 754 Floating-point Standard

- Two representations
  - **Single precision (32-bit)**: type `float` in C
  - **Double precision (64-bit)**: type `double` in C

Sign
- Single: 1 bit
- Double: 1 bit

Exponent
- Single: 8 bits
- Double: 11 bits

Fractional part of significand
- Single: 23 bits
- Double: 52 bits

S	Exponent	Fraction

- 0: positive (+)
- 1: negative (-)

-1+bias

$$+1.1_{two} \times 2^{-1}$$

Leading 1 bit is implicit

Recap: Special Cases

- Exponent = 00…0, Fraction = 00…0
  → Not 1.0 x 2^{-127} but **0**

- Exponent = 00…0, Fraction ≠ 00…0
  → Not (1 + fraction) x 2^{-127} but **(0 + fraction) x 2^{-126}**
  → Denormalized real numbers (to represent very small numbers)

- Exponent = 11…1, Fraction = 00…0
  → **±infinity**

- Exponent = 11…1, Fraction ≠ 00…0
  → **Not-a-Number (*NaN*)**
  → Indicates illegal or undefined result

How to represent the result of invalid operations (e.g., 0/0)?

Recap: Floating-point Addition

- Now consider a 4-digit binary example

  $1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2}$ (0.5 + –0.4375)

1. Align binary points

- Shift number with *smaller exponent*
- $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$

2. Add significands

- $0.001_2 \times 2^{-1}$

3. Normalize result & check for over/underflow

- $1.000_2 \times 2^{-4}$, with no over/underflow

Check $-126 \leq -4 \leq +127$
in case of a single precision

4. Round

- $1.000_2 \times 2^{-4} = 0.0625$

Recap: Floating-point Multiplication

- Now consider a 4-digit binary example

  $1.000_2 \times 2^{-1} \times -1.110_2 \times 2^{-2}$ (0.5 × –0.4375)

For biased exponents, subtract bias from sum

1. Add exponents

- Unbiased: –1 + –2 = –3
- Biased: (–1 + 127) + (–2 + 127) – 127 = –3 + 127

2. Multiply significands

- $1.000_2 \times 1.110_2 = 1.110_2 \Rightarrow 1.110_2 \times 2^{-3}$

3. Normalize result & check for over/underflow

- $1.110_2 \times 2^{-3}$ (no change) with no over/underflow

4. Round

- $1.110_2 \times 2^{-3}$ (no change)

5. Determine sign: + sign × – sign ⇒ – sign

- $-1.110_2 \times 2^{-3} = -0.21875$

Today’s Topic

- I originally intended to cover logic design
- But I will first address the performance aspect

- Please delete the slide file of the logic design basics
- Your midterm exam scope includes the material covered up to today

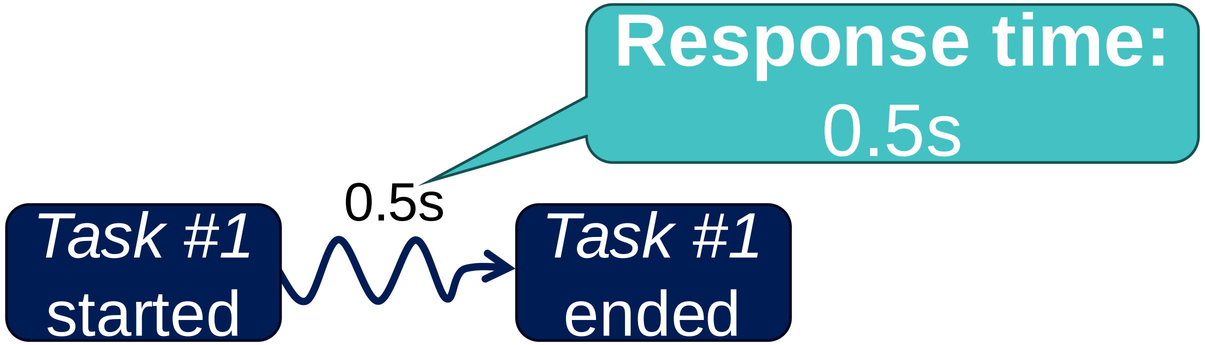
Performance
(A review including more detailed information)

We Focus on the Time

- Most important thing: time, time, and time

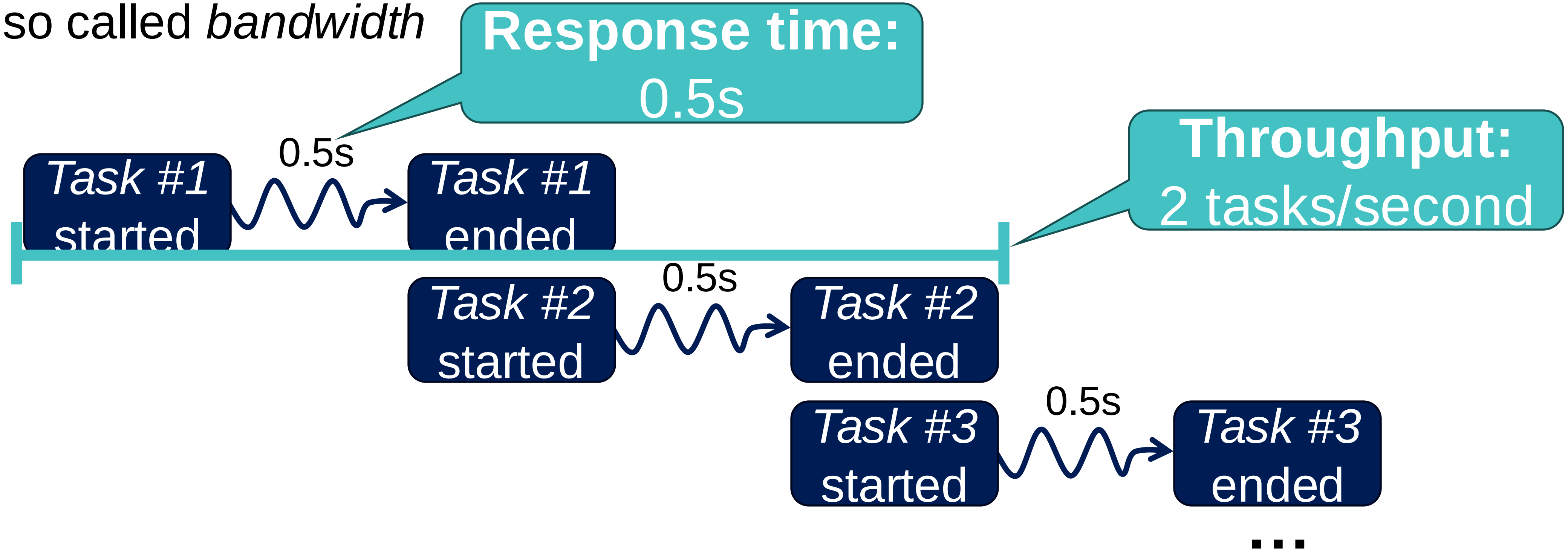
Performance Metrics

- **Response time**: the <u>time between the start and completion</u> of a task
  - Also called *execution time*, *latency*



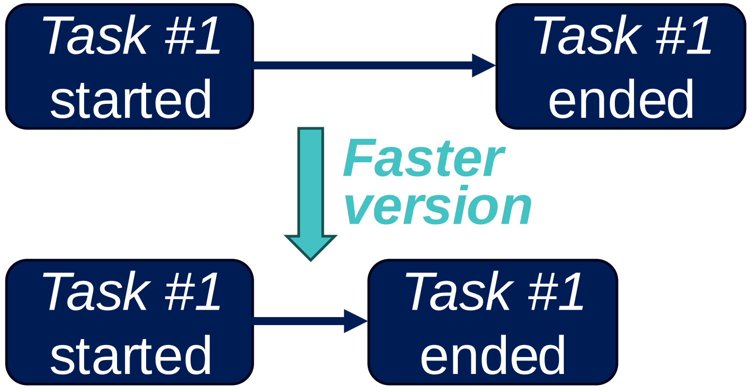
Performance Metrics

- **Response time**: the <u>time between the start and completion</u> of a task
  - Also called *execution time*, *latency*
- **Throughput:** total work done <u>per unit time</u>
  - E.g., *# of tasks/transactions/… per hour*
  - Also called *bandwidth*

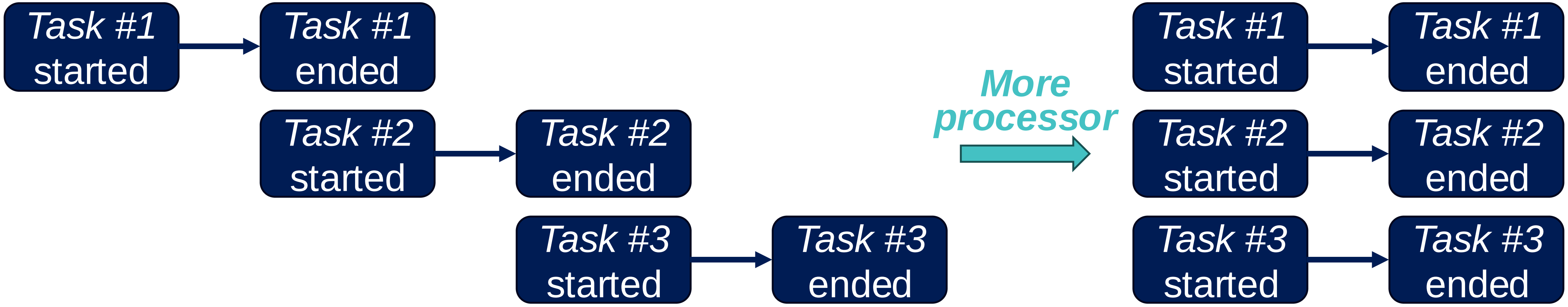


Throughput and Response Time

- How are <u>response time</u> and <u>throughput</u> affected by
  - Replacing the processor with a faster version?
    → Response time ↓, Throughput ↓



  - Adding more processors?
    → Throughput ↓ (No one task gets work done faster)



Performance Metrics

- **Response time**: the time between the start and completion of a task
  - Also called *execution time*, *latency*
- **Throughput:** total work done per unit time
  - E.g., *# of tasks/transactions/… per hour*
  - Also called *bandwidth*

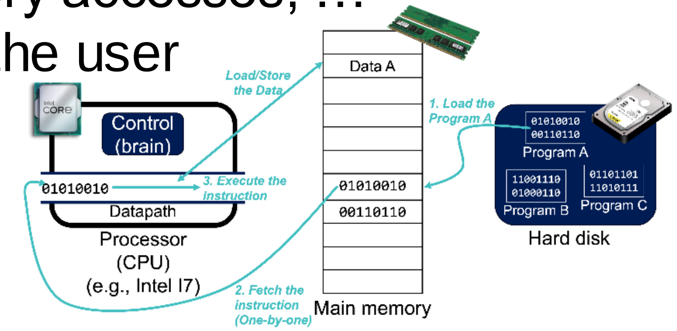
Execution Time

Execution Time (Response Time)

Elapsed Time

The total time to complete a task

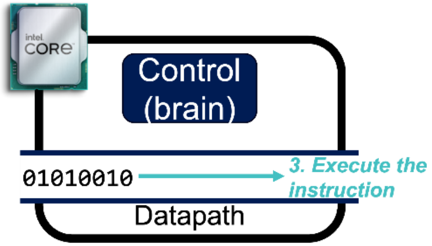
- Counts everything, including disk accesses, memory accesses, …
- Experienced by the user



CPU Time

The actual time the CPU spends

- Doesn't include time spent waiting for I/O or running other programs



time Command in Linux

Execution Time (Response Time)

Elapsed Time

The total time to complete a task

CPU Time

The actual time the CPU spends

```
$ time a.out
real 341m58.124s
user 464m9.282s
sys 13m10.743s
```

time Command in Linux

Execution Time (Response Time)

Elapsed Time

The total time to complete a task

CPU Time

The actual time the CPU spends

```
$ time a.out
real 341m58.124s
user 464m9.282s
sys  13m10.743s
```

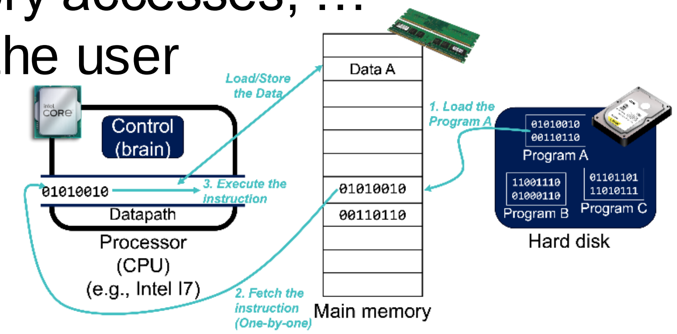
Execution Time

Execution Time (Response Time)

Elapsed Time

The total time to complete a task

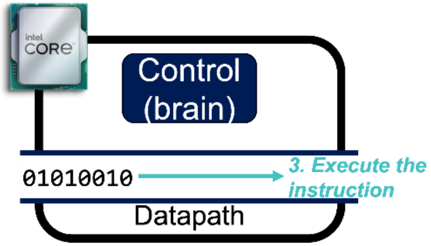
- Counts everything, including disk accesses, memory accesses, …
- Experienced by the user



CPU Time

The actual time the CPU spends

- Doesn’t include time spent waiting for I/O or running other programs



Our Focus

Execution Time (Response Time)

We'll focus on CPU time for now!

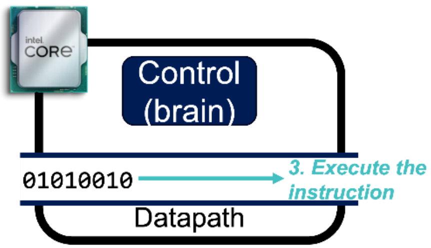
CPU Time

The total time to complete a task

- Counts everything, including disk accesses, memory accesses, …
- Experienced by the user

The actual time the CPU spends

- Doesn't include time spent waiting for I/O or running other programs



Performance and Execution Time

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

- **Relative performance**: “X is *N* times faster than Y”

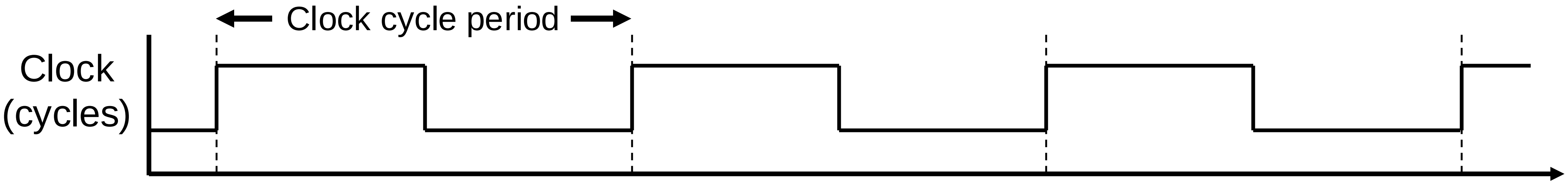
$$N = \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution Time}Y}{\text{Execution Time}X}$$

- Exercise: time taken to run a program
  - 10s on A
  - 15s on B
  - Q. A is *N* times faster than B. What is *N*?

$$\boxed{\frac{\text{Execution Time}B}{\text{Execution Time}A} = \frac{15\text{s}}{10\text{s}} = 1.5}$$

Recap: CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



- **Clock cycle (period):** duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- **Clock rate (frequency):** # of cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

$$\text{Frequency (Hz)} = \frac{1}{\text{Clock Cycle Period}}$$

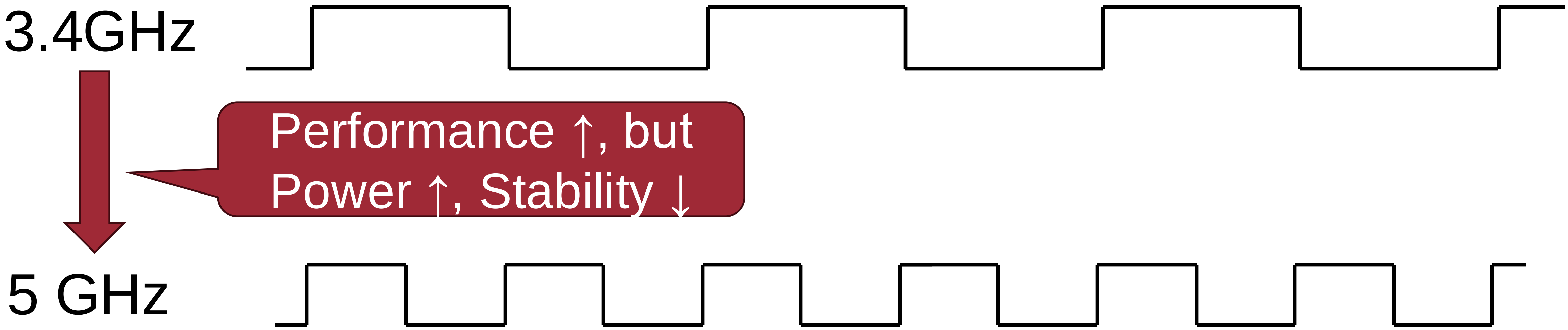
Background: Metric Prefixes

peta	P	10^{15}		1 000 000 000 000 000
tera	T	10^{12}		1 000 000 000 000
giga	G	10^{9}		1 000 000 000
mega	M	10^{6}		1 000 000
kilo	k	10^{3}		1 000
hecto	h	10^{2}		100
deka	da	10^{1}		10
base unit		10^{0}		1
deci	d	10^{-1}	1/10	0.1
centi	c	10^{-2}	1/100	0.01
milli	m	10^{-3}	1/1 000	0.001
micro	μ	10^{-6}	1/1 000 000	0.000 001
nano	n	10^{-9}	1/1 000 000 000	0.000 000 001
Ångström	Å	10^{-10}	1/10 000 000 000	0.000 000 000 1
pico	p	10^{-12}	1/1 000 000 000 000	0.000 000 000 001

This information will be provided in your midterm exam!

FYI: CPU Overclocking

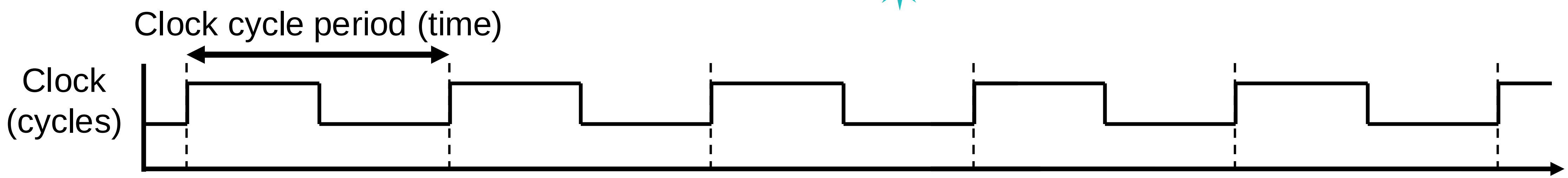
- The practice of **increasing the clock rate** of a computer to exceed that certified by the manufacturer



FYI: CPU Overclocking

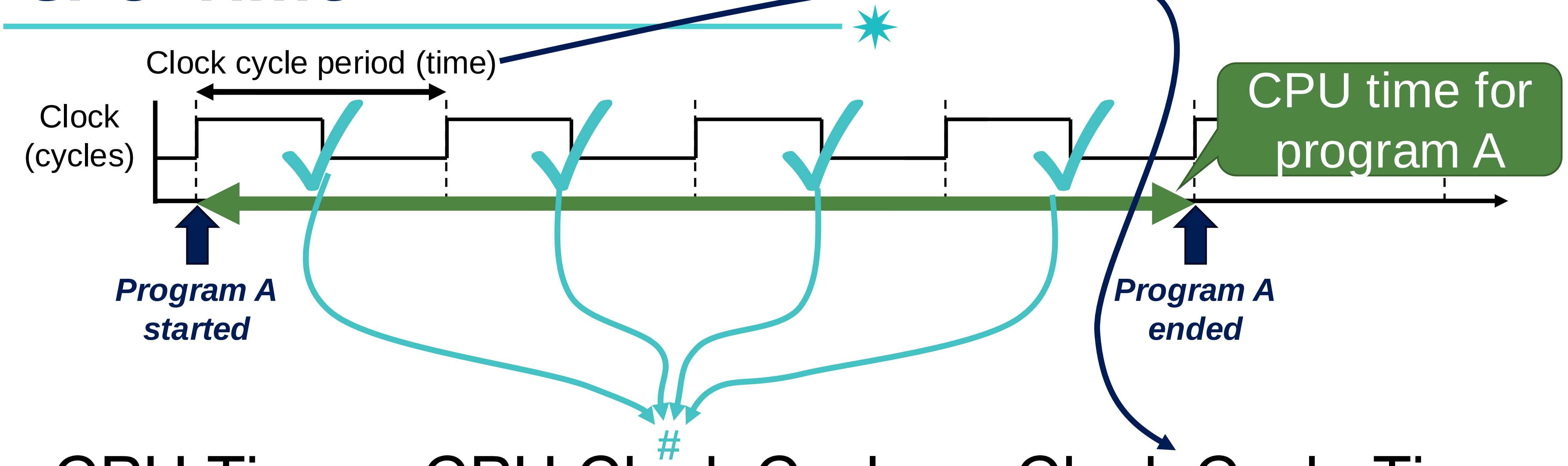
Image from https://track2training.com/2021/09/10/cpu-overclocking/

CPU Time



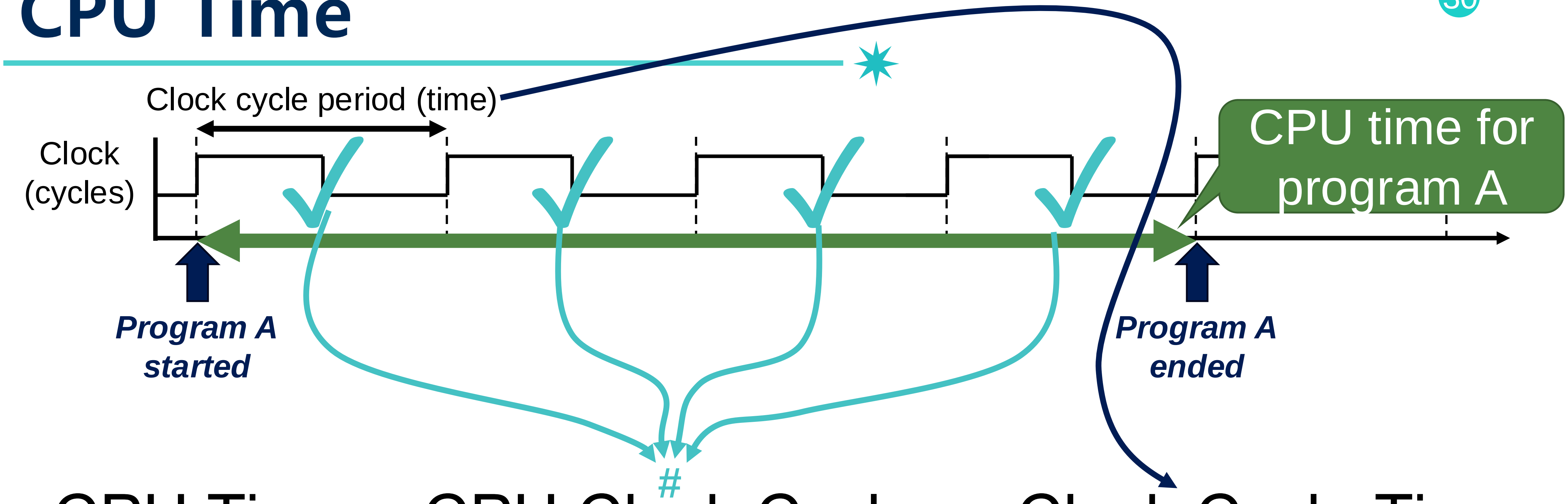
CPU Time = CPU Clock Cycles x Clock Cycle Time

CPU Time



CPU Time = CPU Clock Cycles x Clock Cycle Time

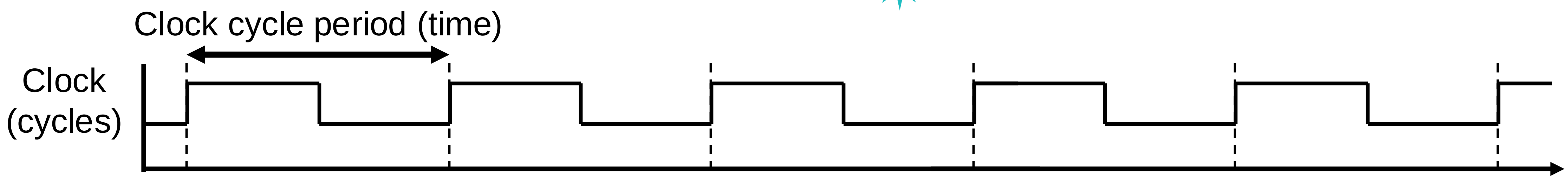
CPU Time



$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

Performance Improvement



Performance improved by
- Reducing number of clock cycles
- Increasing clock rate

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

Trade off

Reducing clock cycles requires **more operations per cycle**, which lowers the clock rate

CPU Time

CPU Time = CPU Clock Cycles x Clock Cycle Time

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

of Instructions per Program (Instruction Count)

$$\text{CPU Time} = \boxed{\text{CPU Clock Cycles}} \times \text{Clock Cycle Time}$$

$$= \boxed{\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Instruction Count

of Instructions per Program (Instruction Count)

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

```
swap:
  multi $2, $5, 4
  add $2, $4, $2
  …
```

# of instructions per program

Affected by:

- *Compiler*
- *Algorithm*
- *Programming language*
- *ISA*

Clock Cycles per Instruction (CPI)

$$\text{CPU Time} = \boxed{\text{CPU Clock Cycles}} \times \text{Clock Cycle Time}$$

$$\| $$

$$= \boxed{\underbrace{\frac{\text{Instructions}}{\text{Program}}}_{\text{Instruction Count}} \times \underbrace{\frac{\text{Clock cycles}}{\text{Instruction}}}_{\text{Average CPI}}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

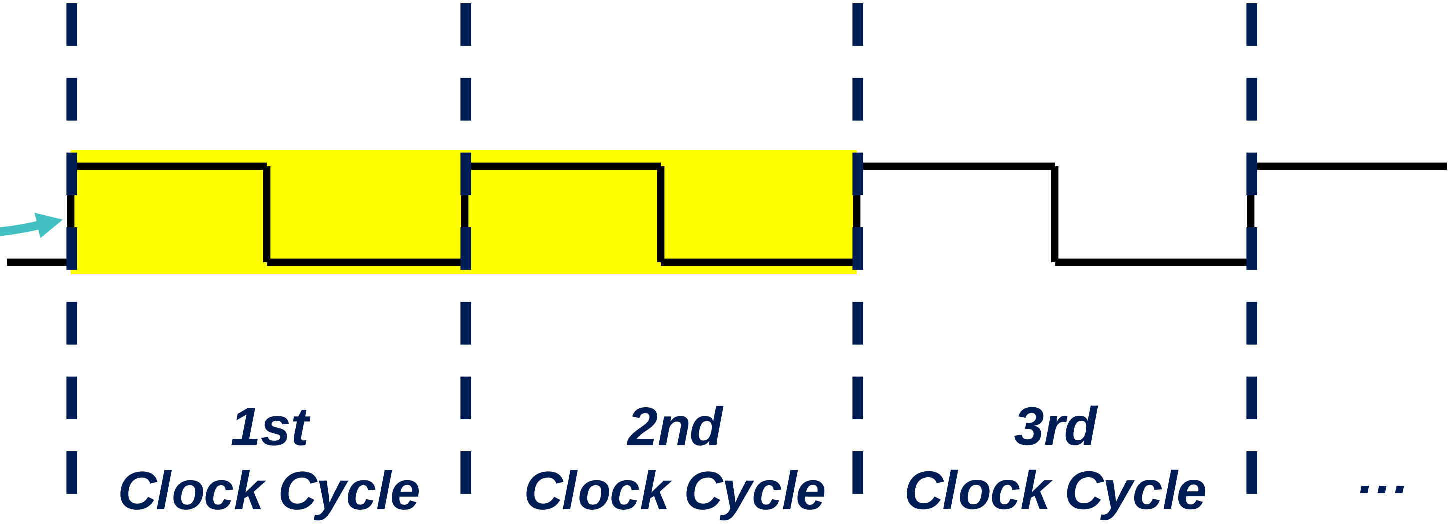
Clock Cycles per Instruction (CPI)

Different instructions have different CPI → Average CPI

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

CPI = 2

```
wap:
multi $2, $5, 4
add $2, $4, $2
…
```



CPU Time

$$\text{CPU Time} = \boxed{\text{CPU Clock Cycles}} \times \boxed{\text{Clock Cycle Time}}$$

$$= \boxed{\underbrace{\frac{\text{Instructions}}{\text{Program}}}_{\text{Instruction Count}} \times \underbrace{\frac{\text{Clock cycles}}{\text{Instruction}}}_{\text{Average CPI}}} \times \boxed{\frac{\text{Seconds}}{\text{Clock cycle}}}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

CPI Example

- **Computer A**: Cycle Time = 250ps, CPI = 2.0
- **Computer B**: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

A and B consists of the same instructions

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$
$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$
$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2$$

…by this much

CPI in More Detail

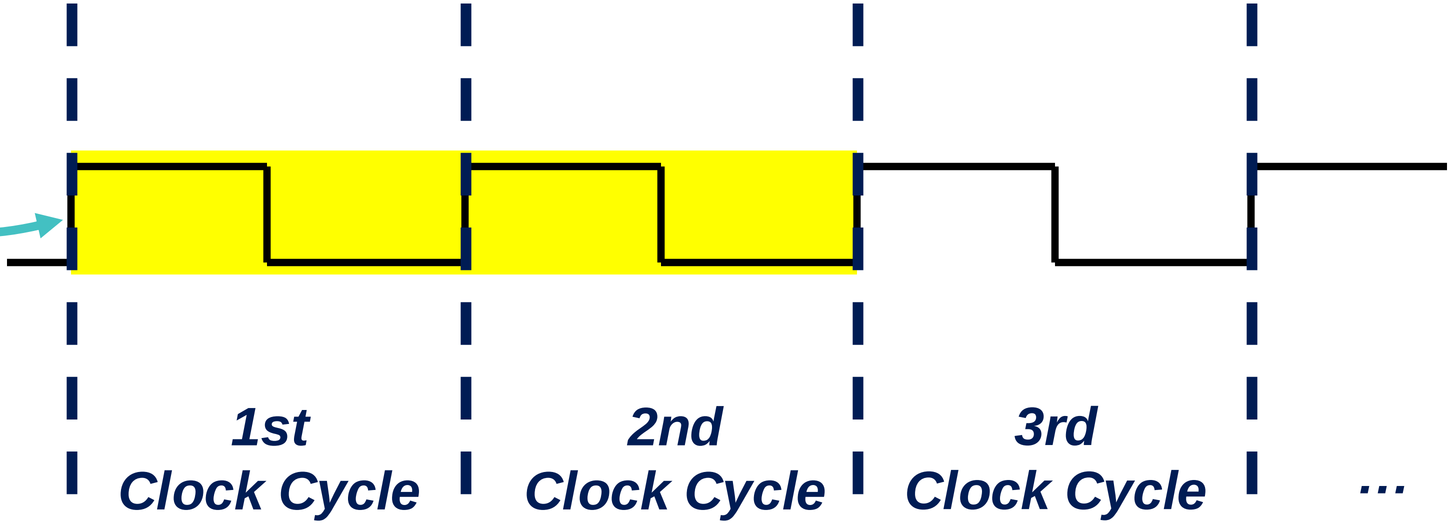
How is the average CPI calculated?

Different instructions have different CPI → Average CPI

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

CPI = 2

```
wap:
multi $2, $5, 4
add $2, $4, $2
…
```



CPI in More Detail

of instruction classes

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

CPI = 2

CPI = 1

CPI = 2

```
Program A:
  multi $2, $5, 4
  add $2, $4, $2
  multi $3, $4, 6
```

Clock Cycles =
(2 x 2) + (1 x 1) =
5

CPI in More Detail

# of instruction classes

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

$$\text{(Weighted average) CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}} \right)$$

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

CPI = 2

CPI = 1

CPI = 2

```
Program A:
  multi $2, $5, 4
  add $2, $4, $2
  multi $3, $4, 6
```

Average CPI =
(2 x 2/3) + (1 x 1/3) =
5/3

Performance Summary

$$\text{CPU Time} = \underbrace{\frac{\text{Instructions}}{\text{Program}}}_{\text{Instruction Count}} \times \underbrace{\frac{\text{Clock cycles}}{\text{Instruction}}}_{\text{Average CPI}} \times \underbrace{\frac{\text{Seconds}}{\text{Clock cycle}}}_{\text{Clock Cycle Period}}$$

- Performance depends on
  - **Algorithm**: affects IC, CPI
  - **Programming language**: affects IC, CPI
  - **Compiler**: affects IC, CPI
  - **Instruction set architecture (ISA):** affects IC, CPI, Clock Cycle Period

Question?