

CSE261: Computer Architecture

7. Performance

Seongil Wi

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

3

- 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 x 10⁻²⁰

- *Very large numbers*

→
Floating decimal point

3.15576 x 10¹⁹

Can be represented with a limited number of bits!

Solution: Floating-point Number Representation

Recap: IEEE 754 Floating-point Standard

5

- Developed in response to *divergence of representations*

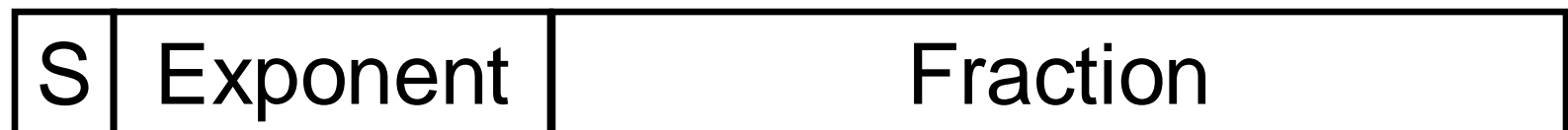
Divergence of representations

Normalized representation

IEEE 754 representation

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

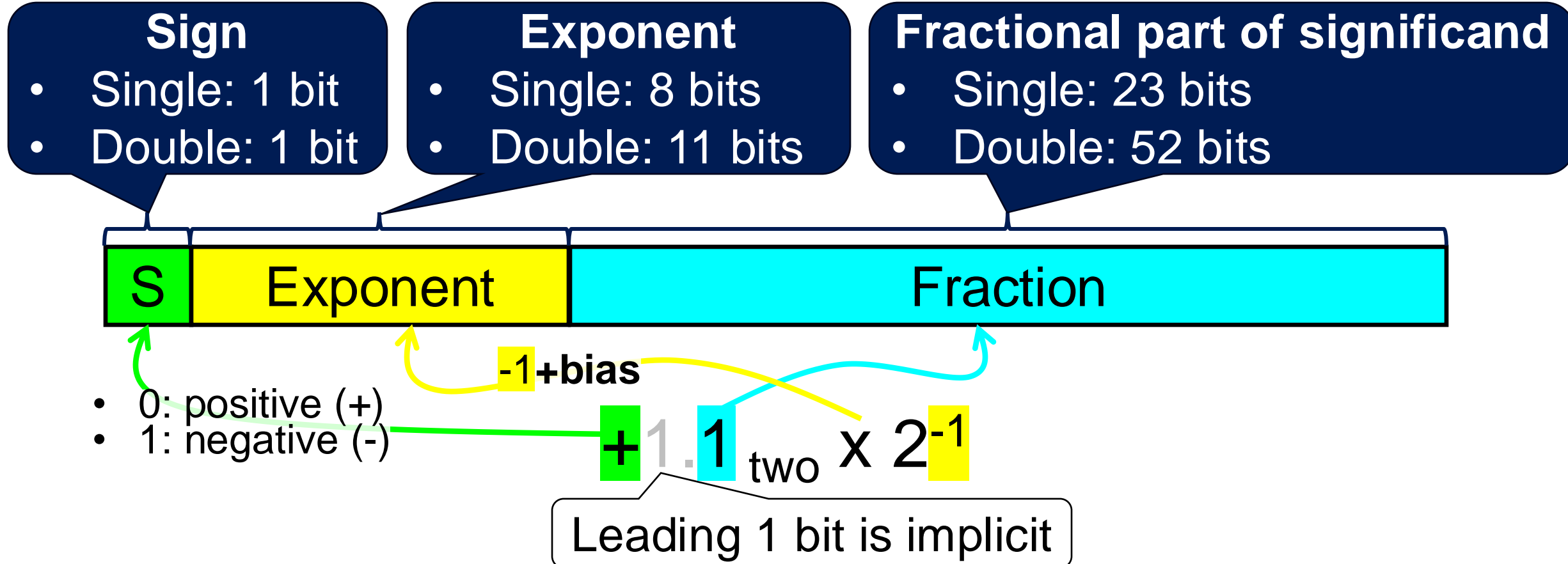
$$1.1_{\text{two}} \times 2^{-1}$$



Recap: IEEE 754 Floating-point Standard

6

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



Recap: Special Cases



- Exponent = 00...0, Fraction = 00...0
→ Not 1.0×2^{-127} but **0**
- Exponent = 00...0, Fraction \neq 00...0
→ Not $(1 + \text{fraction}) \times 2^{-127}$ but **(0 + fraction) $\times 2^{-126}$**
→ Denormalized real numbers (to represent very small numbers)
- Exponent = 11...1, Fraction = 00...0
→ **\pm infinity**
- Exponent = 11...1, Fraction \neq 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

4. Round

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

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

Recap: Floating-point Multiplication

- Now consider a 4-digit binary example

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

1. Add exponents

– Unbiased: $-1 + -2 = -3$

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

For biased exponents,
subtract bias from sum

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 \times $-$ sign \Rightarrow $-$ 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)

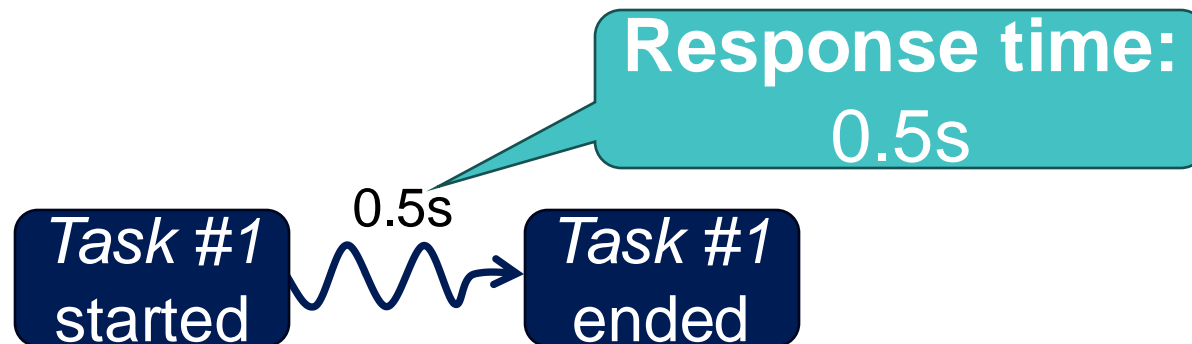
We Focus on the Time

- Most important thing: time, time, and time

Performance Metrics



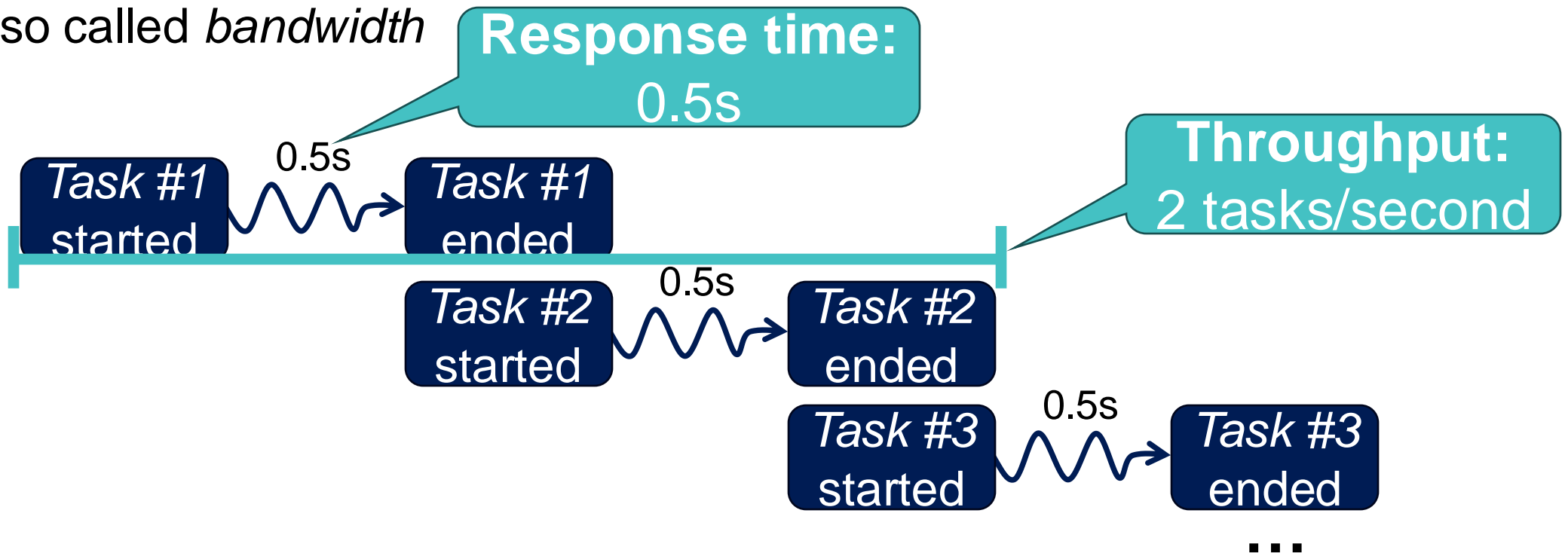
- **Response time:** the time between the start and completion of a task
 - Also called *execution time*, *latency*



Performance Metrics

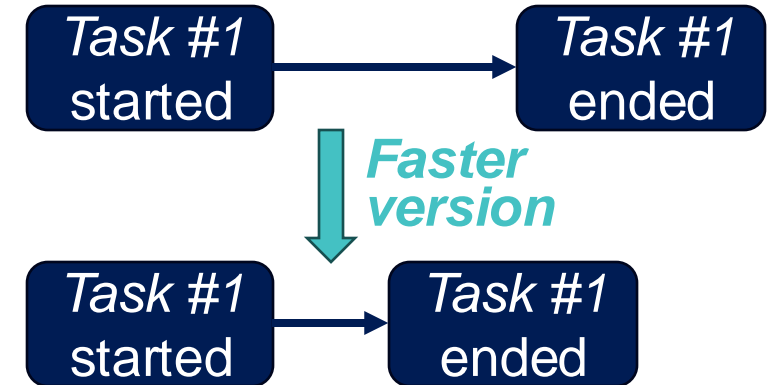


- **Response time:** the time between the start and completion of a task
 - Also called *execution time*, *latency*
- **Throughput:** total work done per unit time
 - E.g., # of *tasks/transactions/...* per hour
 - Also called *bandwidth*

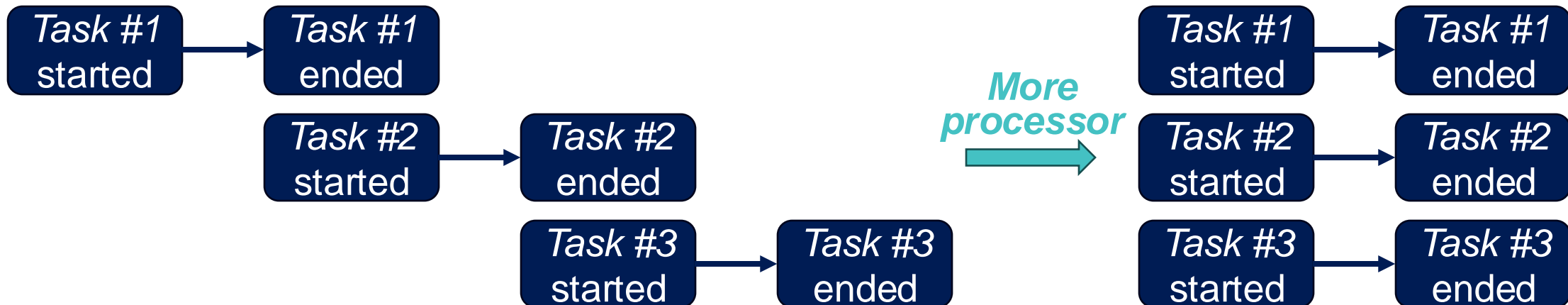


Throughput and Response Time

- How are response time and throughput 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*

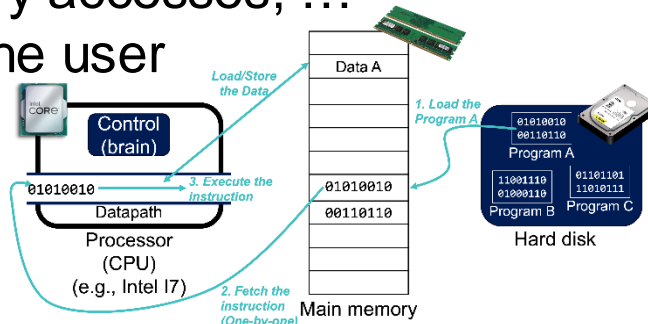
Execution Time

Execution Time (Response Time)

Elapsed Time

The total time to complete a task

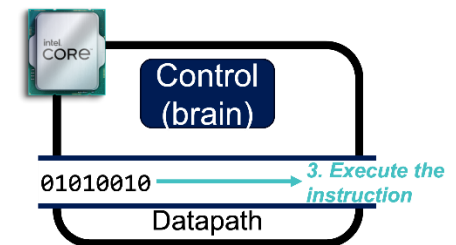
- Counts everything, including disk accesses, memory accesses, ...
- Experienced by the user



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
```



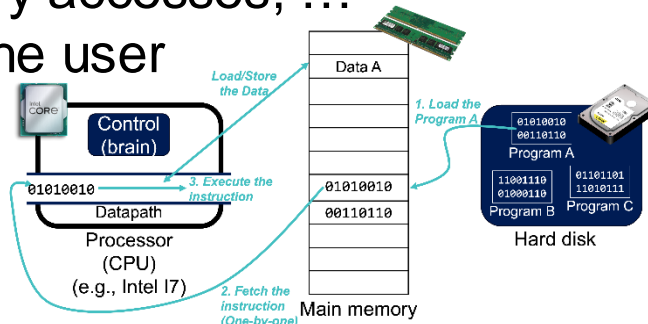
Execution Time

Execution Time (Response Time)

Elapsed Time

The total time to complete a task

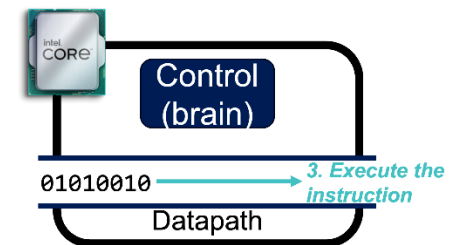
- Counts everything, including disk accesses, memory accesses, ...
- Experienced by the user



CPU Time

The actual time the CPU spends

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



Our Focus

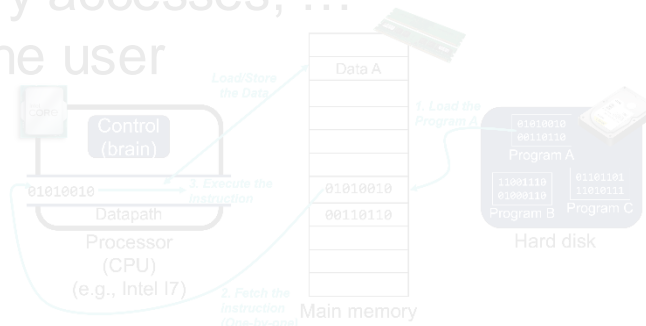
Execution Time (Response Time) *

We'll focus on
CPU time for now!

CPU Time 

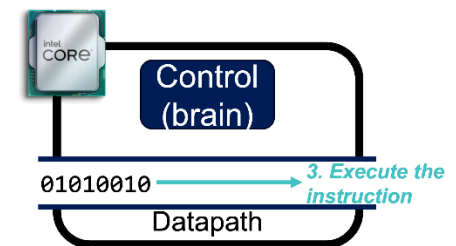
The total time to complete a task

- 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

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

- **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 ?

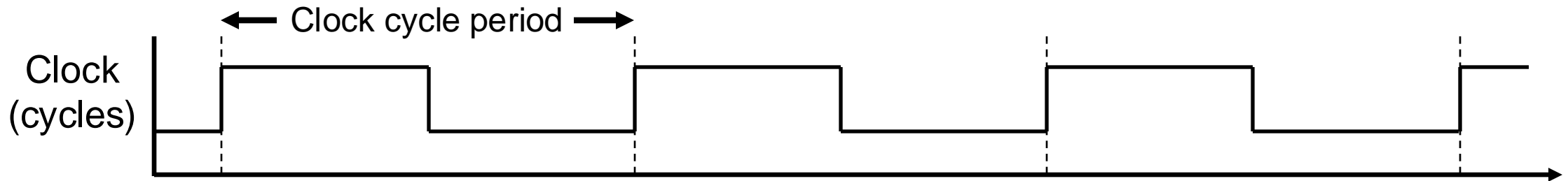
$$\frac{\text{Execution Time}_B}{\text{Execution Time}_A} = \frac{15s}{10s} = 1.5$$

Recap: CPU Clocking



23

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



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

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

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
<i>base unit</i>		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

26



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

3.4GHz



Performance ↑, but
Power ↑, Stability ↓

5 GHz



Basic clock rate

Maximum clock rate

[INTEL] 코어 i7-13700K 랩터레이스 (3.4GHz/33MB) 정품박스

인텔(소켓1700) / 8+12코어 / 16+12쓰레드 / 기본 클럭: 3.4GHz / 최대 클럭: 5.6GHz / L3 캐시: 33MB / PBP : 125W / PCIe5.0 , 4.0 / 메모리 규격: DDR5 / 픽: 탑재 / 인텔 UHD 770 / 기술 지원: 하이퍼스레딩 / 쿨러: 미포함

FYI: CPU Overclocking

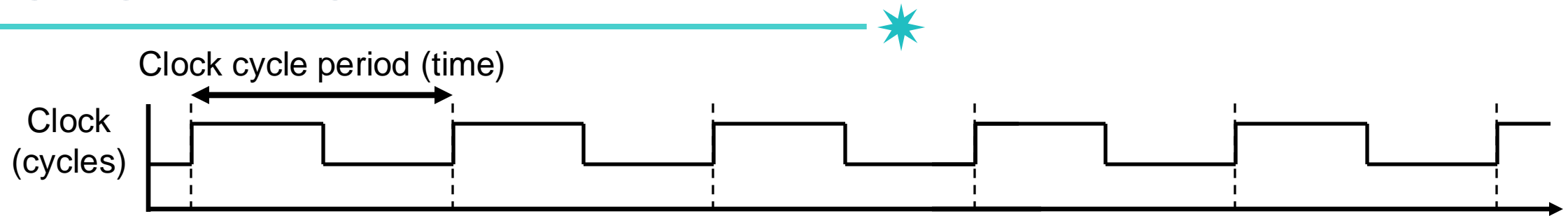
27



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

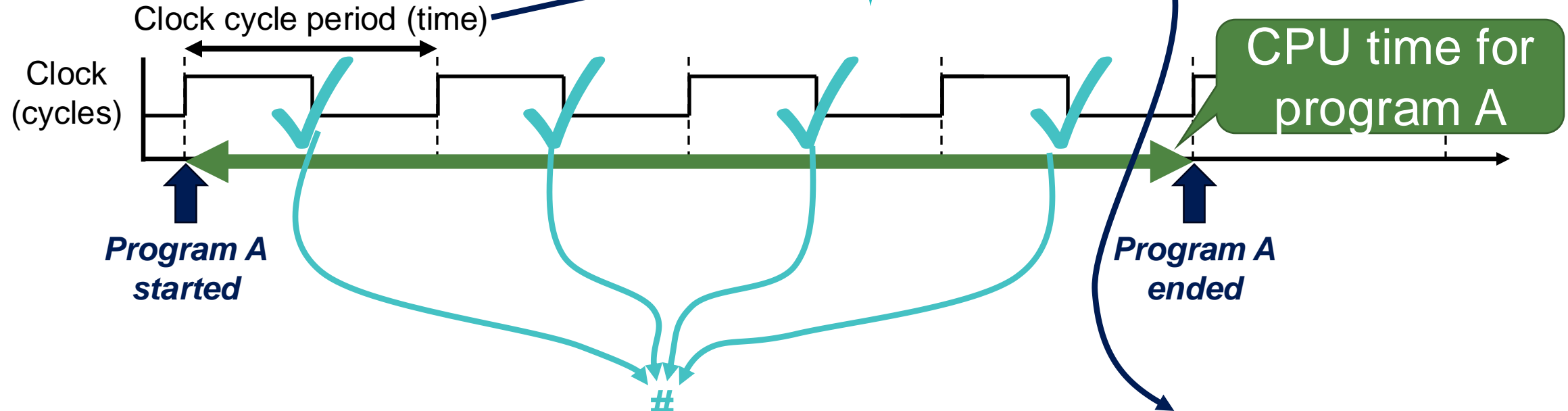
CPU Time

28



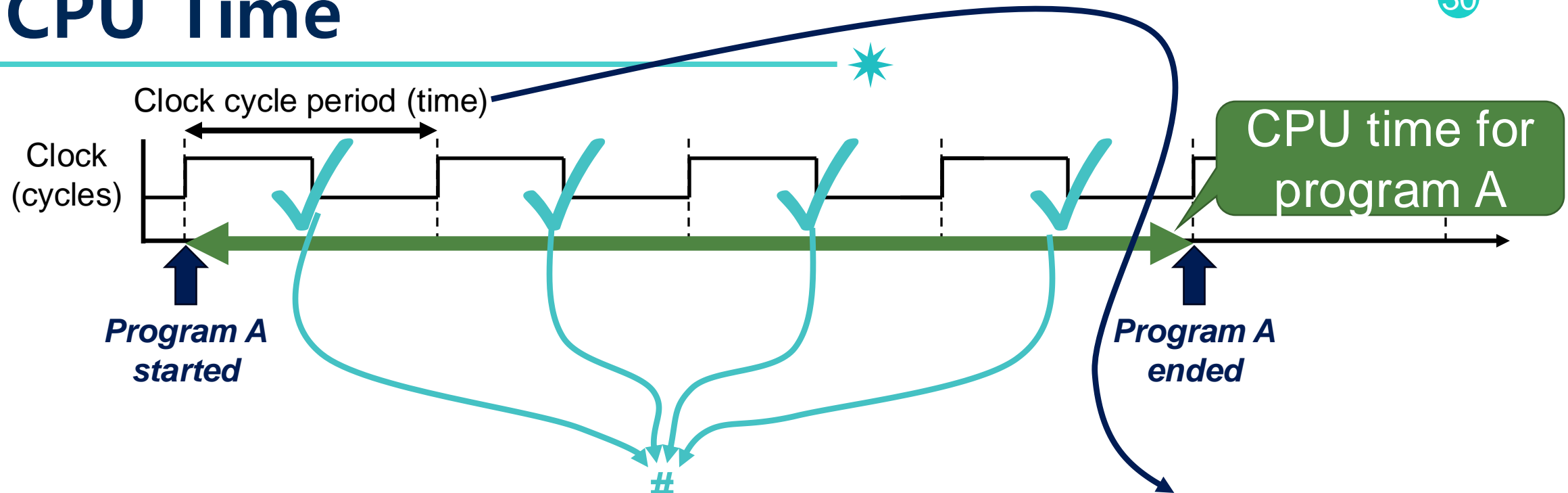
$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

CPU Time



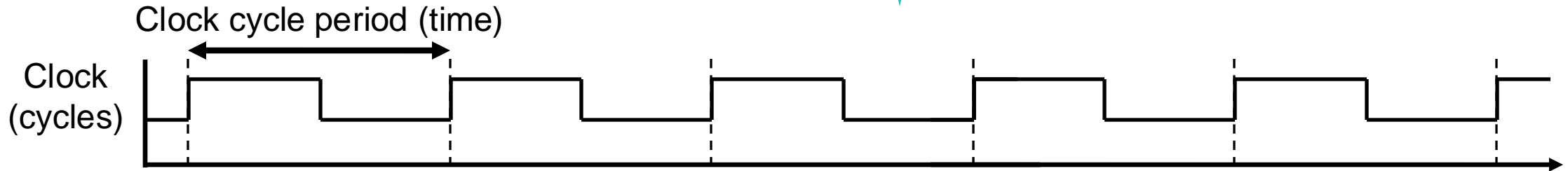
$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

CPU Time



$$\begin{aligned} \text{CPU Time} &= \text{CPU Clock Cycles} \times \text{Clock Cycle Time} \\ &= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}} \end{aligned}$$

Performance Improvement



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

$$\begin{aligned}\text{CPU Time} &= \text{CPU Clock Cycles} \times \text{Clock Cycle Time} \\ &= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}\end{aligned}$$

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)

$$\begin{aligned} \text{CPU Time} &= \boxed{\text{CPU Clock Cycles}} \times \text{Clock Cycle Time} \\ &= \boxed{\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}}} \times \frac{\text{Seconds}}{\text{Clock cycle}} \end{aligned}$$

Instruction Count

of Instructions per Program (Instruction Count)

35

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

```
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) *

$$\begin{aligned} \text{CPU Time} &= \boxed{\text{CPU Clock Cycles}} \times \text{Clock Cycle Time} \\ &= \boxed{\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}}} \times \frac{\text{Seconds}}{\text{Clock cycle}} \\ &\quad \text{Instruction Count} \quad \text{Average CPI} \end{aligned}$$

Clock Cycles per Instruction (CPI)

Different instructions
have different CPI

Average CPI

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

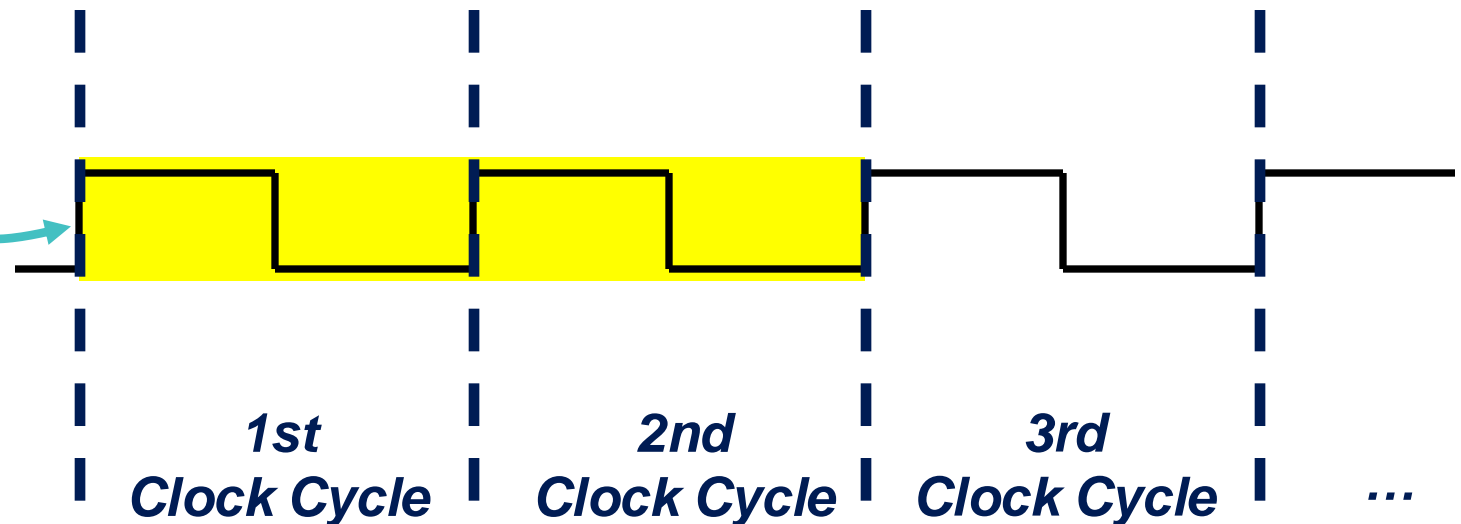
CPI = 2

gap:

multi \$2, \$5, 4

add \$2, \$4, \$2

...



CPU Time

38

$$\begin{aligned} \text{CPU Time} &= \text{CPU Clock Cycles} \times \text{Clock Cycle Time} \\ &= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}} \\ &\quad \text{Instruction Count} \quad \text{Average CPI} \\ &= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}} \end{aligned}$$

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

$$\begin{aligned}\text{CPU Time}_A &= \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\ &= 1 \times 2.0 \times 250\text{ps} = 1 \times 500\text{ps}\end{aligned}$$

A is faster...

$$\begin{aligned}\text{CPU Time}_B &= \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B \\ &= 1 \times 1.2 \times 500\text{ps} = 1 \times 600\text{ps}\end{aligned}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{1 \times 600\text{ps}}{1 \times 500\text{ps}} = 1.2$$

...by this much

CPI in More Detail



How is the average CPI calculated?

Different instructions have different CPI

Average CPI

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

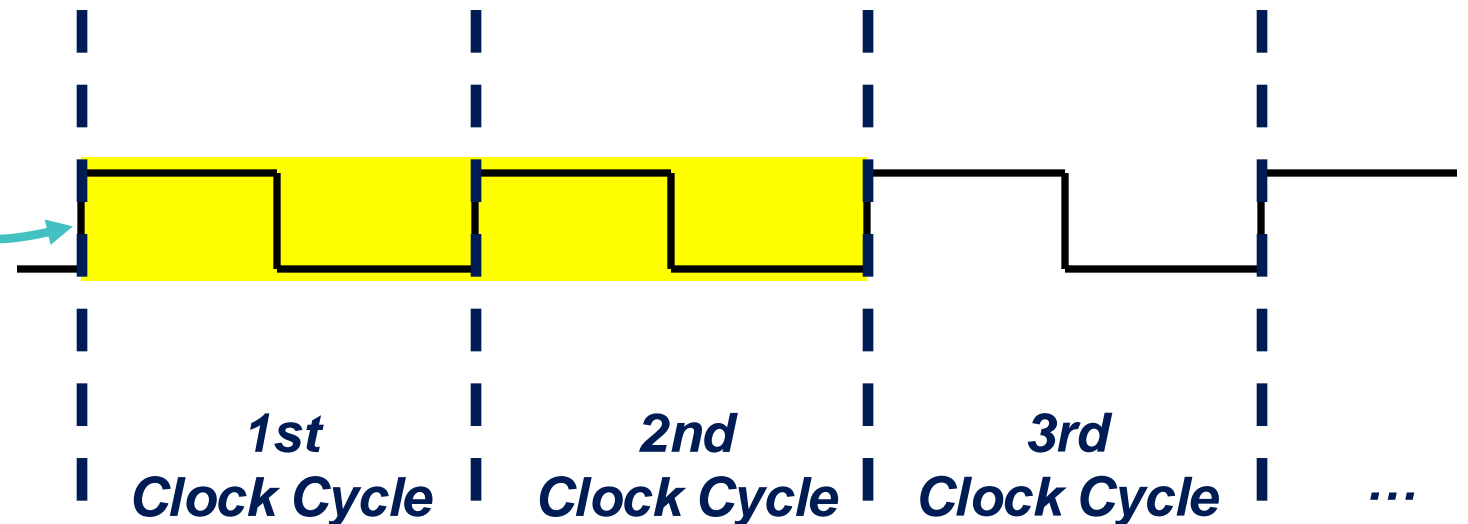
CPI = 2

map:

multi \$2, \$5, 4

add \$2, \$4, \$2

...



CPI in More Detail

of instruction
classes

41

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

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

CPI = 2

CPI = 1

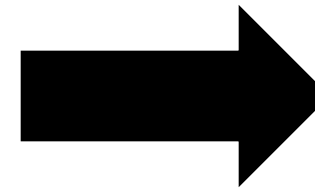
CPI = 2

Program A:

multi \$2, \$5, 4

add \$2, \$4, \$2

multi \$3, \$4, 6



$$\text{Clock Cycles} = (2 \times 2) + (1 \times 1) = 5$$

CPI in More Detail

of instruction classes

42

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

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

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

CPI = 2

CPI = 1

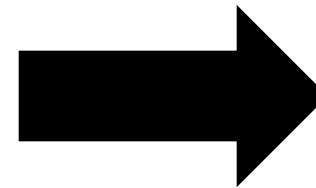
CPI = 2

Program A:

multi \$2, \$5, 4

add \$2, \$4, \$2

multi \$3, \$4, 6



$$\text{Average CPI} = (2 \times 2/3) + (1 \times 1/3) = 5/3$$

Performance Summary

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

Diagram illustrating the components of CPU Time:

- Instruction Count** (under Instructions/Program)
- Average CPI** (under Clock cycles/Instruction)
- Clock Cycle Period** (under Seconds/Clock cycle)

- 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?