

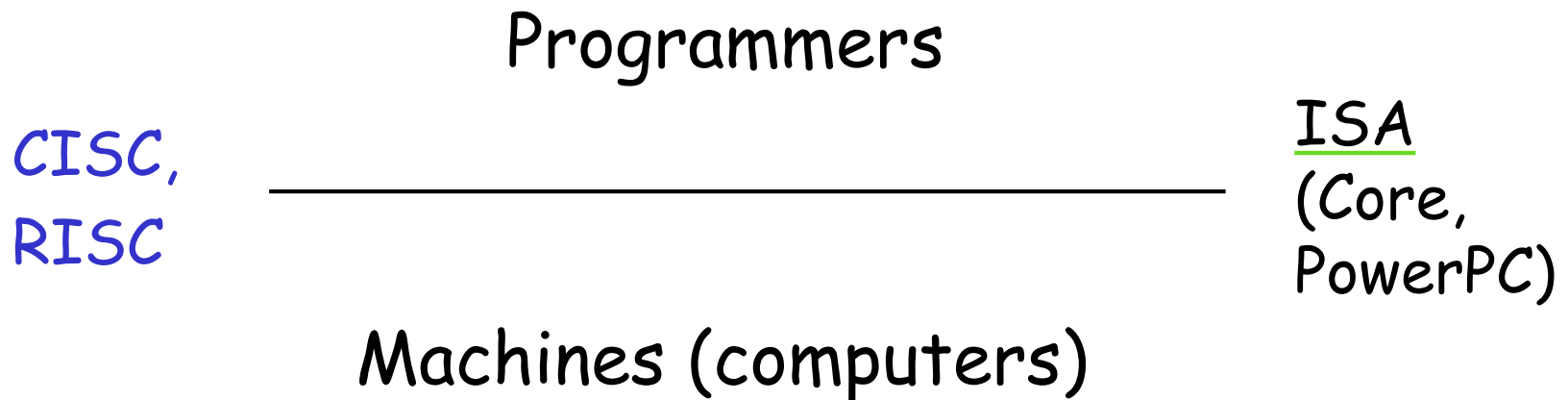
# Class Topics (클래스 홈페이지 참조)

- ❑ Part 1: Fundamental concepts and principles
- ❑ Part 2: 빠른 컴퓨터를 위한 ISA design
  - Topic 1 Computer performance and ISA design (Ch. 1)
    - ❑ 1-1 Performance evaluation & performance models
    - ❑ 1-2 RISC versus CISC, power limit
  - Topic 2 RISC (MIPS) instruction set (Chapter 2)
  - Topic 3 Computer arithmetic and ALU (Chapter 3)
- ❑ Part 3: ISA 의 효율적인 구현 (pipelining, cache memory)

# Most Important Interface in CS

(반복)

- ❑ Interface vs. implementation
- ❑ Hardware-software interface



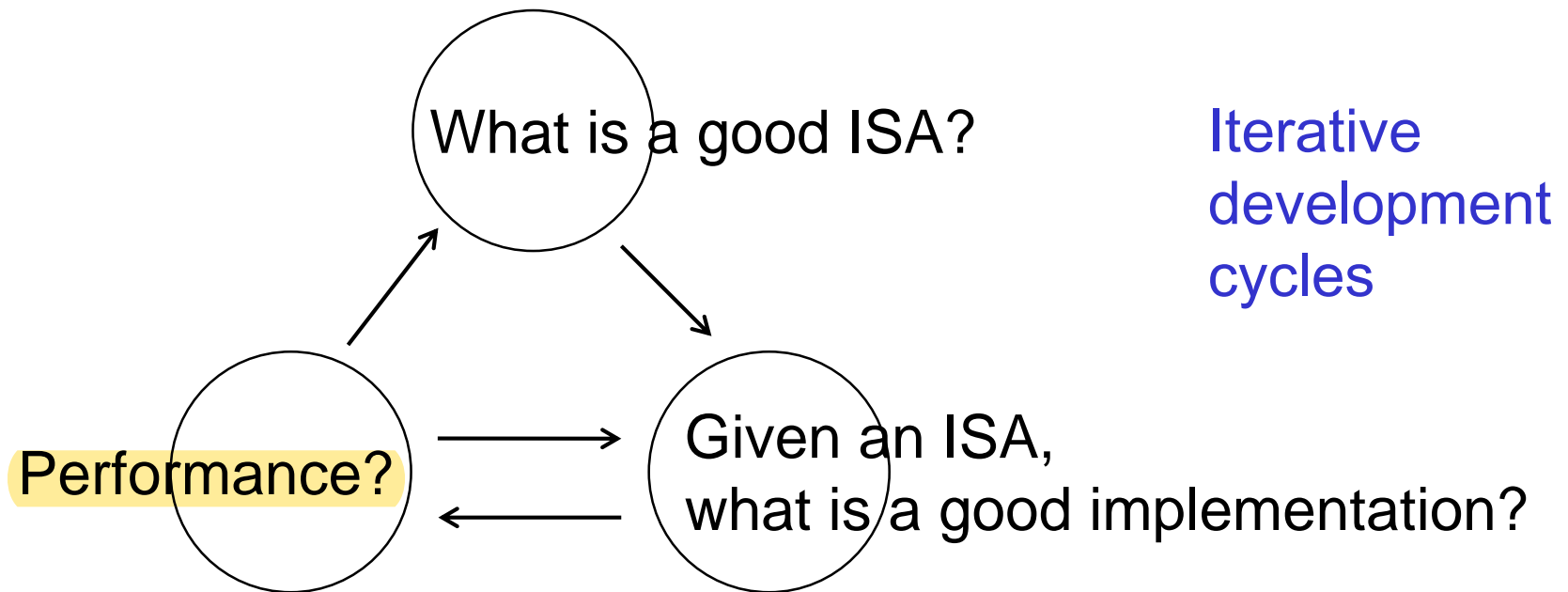
# Product Development Paradigm

- ❑ Marketing, requirements analysis and planning
  - New (or existing) product; can we sell it?
- ❑ Design and implementation
  - External interface
    - How the user will use the product
  - Internal implementation
- ❑ Testing and release
- ❑ Continual enhancement

Iterative  
development  
cycles

# Big Picture

- ❑ Issue 1: Fundamental concepts and principles
- ❑ Issue 2: ISA design (external I/F design)
- ❑ Issue 3: (High-level) implementation of ISA



# Big Picture (see Textbook or Class Homepage)

- ❑ Issue 1: computers, CSE, computer architecture? (4 주)
  - Fundamental concepts and principles
- ❑ Issue 2: ISA (HW-SW interface) design (5 주)
  - Ch. 1: computer performance
  - Ch. 2: language of computer; ISA
  - Ch. 3: data representation and ALU
- ❑ Issue 3: implementation of ISA (internal design) (5주)
  - Ch. 4: processor (data path, control, pipelining)
  - Ch. 5: memory system (cache memory)
- ❑ Short introduction to parallel processors

---

# Computer Performance

## Part 1

### Performance Benchmarking, CPU Performance Equation

#### References:

1. Computer Organization and Design & Computer Architecture, Hennessy and Patterson (slides are adapted from those by the authors)

# Computer Performance Evaluation

---

## □ Why important?

- Intelligent purchase (see through marketing hype)
- Design: to answer what-if questions (this class)
  - How does the machine's instruction set affect performance?
    - † LD R1, R31(+1) vs. LD R1, R31, 0x10000001
      - Many such questions

# Paradigms

- ❑ Primitive-composition-abstraction
- ❑ Programming
  - Procedural, OO, functional, logic
- ❑ Problem solving by programming
  - Ideas, solutions, software skills
- ❑ Iterative product development
  - Marketing, requirements analysis, development (interface and implementation), test, maintenance
- ❑ Quantitative engineering paradigm
  - Cost-performance trade-off



# Key Questions (Quantitative Eng. Paradigm)

---

## ❑ Engineering

- Cost/performance (speed, reliability, ...) trade-off
- Measurement, quantitative (numbers)

## ❑ Three key questions

- 1) What is a good computer? (what to measure)
  - Performance measures
- 2) How do we perform measurements? (how to measure)
  - Evaluation methods
- 3) What determines performance? (how to improve)
  - Performance models, domain knowledge

# **Key Questions (Quantitative Eng. Paradigm)**

---

1) What is a good chalk? (what to measure)

- 부러지지 않음, 균일하게 써 짐
- 가루가 적음, 가루가 유해하지 않음
- 잘 지워짐, 손에 묻지 않음
- ...

2) How do we perform measurements?

- Evaluation methods
  - Give us a number for each performance measure

3) How can we improve?

- Performance models - domain knowledge (?)

# Power of Single Number

---

<u>Airplane</u>	<u>Passengers</u>	<u>Range (mi)</u>	<u>Speed (mph)</u>
Boeing 737-100	101	630	598
Boeing 747	470	4150	610
BAC/Sud Concorde	132	4000	1350
Douglas DC-8-50	146	8720	544

- ☐ How much faster is the Concorde compared to the 747?
- ☐ How much bigger is the 747 than the Douglas DC-8?
- ☐ Which has the highest performance?
  - Power of single number
- ☐ Engineering perspective vs. business perspective

---

# Quantitative Engineering

## Computer Performance

- 1) What do we measure?  
(Performance measures)

# I. Computer Performance: TIME, TIME, TIME

---

## ☐ Response time

- How long it takes to do a task

## ☐ Throughput

- Total work done per unit time (e.g., transactions per hour)

## ☐ We'll focus on response time for now...

- Why?

## ☐ How are response time and throughput affected by

- Replacing the processor with a faster version?
- Adding more processors?

- Need parallel programming to improve response time

# Multiprocessor Systems

## ❑ Parallel programming

- Solve a single large problem (e.g., weather forecasting)
  - Use multiple processors to reduce **response time**
- Supercomputers (scientific computing)

## ❑ Natural parallelism

- Data centers (business computing)
  - Each visitor's work is independent
- Assign independent tasks to different processors  
**(throughput)**

# Execution Time (Designer Perspective)

---

❑ UNIX utility: “time a.out↵”

90.7s 12.9s 2:39 65%

- Elapsed time: CPU time plus I/O time (159s)
- User and system CPU time (103.6s)
  - Applications rely on OS services

❑ Separation of concerns

- Processor designer's focus: user CPU time (90.7s)
- OS designer's perspective (12.9s)
- I/O system designer's perspective (56.4s)

❑ What is the system designer's perspective?

# Measuring Execution Time (부연설명)

---

- ❑ Elapsed time (user perspective)
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- ❑ CPU time (this class)
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
- ❑ Processor designer's focus: user CPU time (CPU time if OS is fixed)



# Relative Performance

---

- ❑ For some program running on machine X,

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

- ❑ "X is n times faster than Y"

$$\text{Performance}_X / \text{Performance}_Y = n$$

- ❑ Problem:

- Machine A runs a program in 20 seconds
- Machine B runs the same program in 40 seconds

# Understanding Performance (참고)

---

- ❑ Algorithm (or software architecture)
  - Determines number of operations executed
- ❑ Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation
- ❑ Processor and memory system
  - Determine how fast instructions are executed
- ❑ I/O system (including OS)
  - Determines how fast I/O operations are executed


---

# Quantitative Engineering Computer Performance

2) How do we measure?  
(Evaluation methods)

## II. Choosing programs to evaluate performance

---

- ❑ End user's performance measure: response time
- ❑ How can we measure response time?
- ❑ What about individual users?
- ❑ What about computer designers?
  - Benchmarks 
    - A collection of programs
    - “representative” set of “real” programs
  - Benchmarking

# Many Benchmarks

---

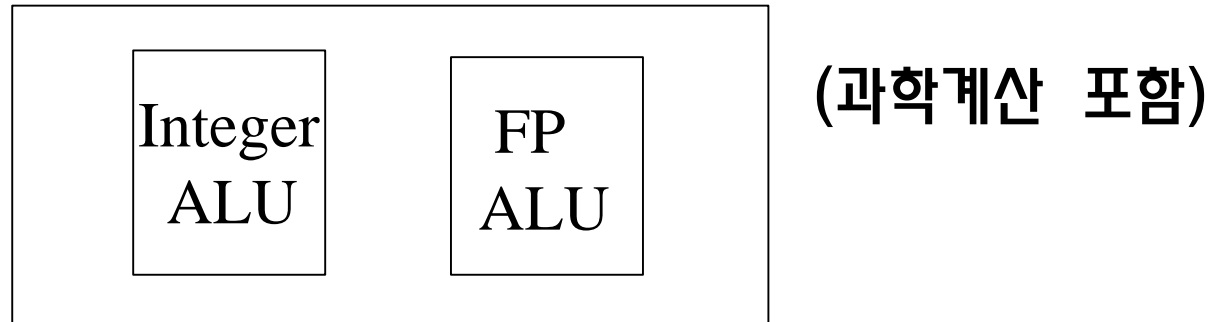
- ❑ SPEC (System Performance Evaluation Corporation)
  - Develop benchmarks, review and publish results
  - CPU benchmark
    - Computation-intensive workloads
    - CPU 2017 (89, 92, 95, 2000, 2006)
  - High performance computing
    - Computation-intensive parallel applications
  - Web servers
  - Power consumption

# SPEC CPU Benchmark 2006

<u>Integer benchmarks</u>		<u>FP Benchmarks</u>	
Name	Description	Name	Description
perl	Interpreted string processing	wrf	Weather
bzip2	Block-sorting compression	sphinx3	Speech recognition
gcc	GNU C Compiler	bwaves	Fluid dynamics
mcf	Combinatorial optimization	gamess	Quantum chemistry
go	Go game (AI)	milc	Quantum chromodynamics
hmmer	Search gene sequence	povray	Image ray-tracing
sjeng	Chess game (AI)	gromacs	Molecular dynamics
libquantum	Quantum computer simulation	cactusADM	General relativity
h264avc	Video compression	calculix	Structural mechanics
omnetpp	Discrete event simulation	GemsFDTD	Electromagnetics
astar	Games/path finding	dealII	Finite element analysis
xalancbmk	XML parsing	soplex	Linear programming
		5 more apps.	

# Integer and FP ALUs (반복)

- ❑ Processors for general-purpose computers



- ❑ What is an integer application?
  - What is a floating-point application?

# SPEC CPU Benchmark 2006

## □ Reference machine and SPECint number

Name	Description	Exec time	<u>Ref time</u>	SPECratio
perl	Interpreted string processing	637	9,777	15.3
bzip2	Block-sorting compression	817	9,650	11.8
gcc	GNU C Compiler	24	8,050	11.1
mcf	Combinatorial optimization	1,345	9,120	6.8
go	Go game (AI)	721	10,490	14.6
hmmer	Search gene sequence	890	9,330	10.5
sjeng	Chess game (AI)	37	12,100	14.5
libquantum	Quantum computer simulation	1,047	20,720	19.8
h264avc	Video compression	993	22,130	22.3
omnetpp	Discrete event simulation	690	6,250	9.1
astar	Games/path finding	773	7,020	9.1
xalancbmk	XML parsing	1,143	6,900	6.0
Geometric mean				<u>11.7</u>



# Benchmarking

---

- ❑ Results reproducible by a third party
  - Disclose configurations and measurement conditions
- ❑ Performance/\$
  - Disclose cost
- ❑ Single number
  - e.g., SPECint2006, SPECfp2006
  - How do you get the number?
    - Relative performance over reference machine
- ❑ Hard to cheat

# Benchmark Games

---

*An embarrassed Intel Corp. acknowledged Friday that a bug in a software program known as a compiler had led the company to overstate the speed of its microprocessor chips on an industry benchmark by 10 percent. However, industry analysts said the coding error...was a sad commentary on a common industry practice of “cheating” on standardized performance tests...The error was pointed out to Intel two days ago by a competitor, Motorola ...came in a test known as SPECint92...Intel acknowledged that it had “optimized” its compiler to improve its test scores. The company had also said that it did not like the practice but felt to compelled to make the optimizations because its competitors were doing the same thing...At the heart of Intel’s problem is the practice of “tuning” compiler programs to recognize certain computing problems in the test and then substituting special handwritten pieces of code...*

*Saturday, January 6, 1996 New York Times*

# Benchmark Games

---

❑ *“There are lies, damn lies, and benchmarking.”*

❑ *“There are lies, damn lies, and statistics.”*

# SPEC Power Benchmark (참고)

---

- ❑ Power consumption of server at different workload levels
  - Performance: ssj\_ops/sec
  - Power: Watts (Joules/sec)

$$\text{Overall ssj\_opsper Watt} = \left( \sum_{i=0}^{10} \text{ssj\_ops}_i \right) / \left( \sum_{i=0}^{10} \text{power}_i \right)$$

- ❑ Who are interested in power consumption?
  - Mobile devices
  - Large-scale data centers
- ❑ Power-aware software design

# SPECpower\_ssj2008 for X4 (참고)

Target Load %	Performance (ssj_ops/sec)	Average Power (Watts)
100%	231,867	295
90%	211,282	286
80%	185,803	275
70%	163,427	265
60%	140,160	256
50%	118,324	246
40%	920,35	233
30%	70,500	222
20%	47,126	206
10%	23,066	180
0%	0	141
Overall sum	1,283,590	2,605
$\Sigma \text{ssj\_ops} / \Sigma \text{power}$		493

# Many Benchmarks

---

- ❑ Business servers and throughput
- ❑ TPC (Transaction Processing Performance Council)
  - TPC-C (order-entry transactions against database)
    - Enter/deliver order, record payment, check status of order, monitor level of stock
  - TPC-E (brokerage firm)
    - Trades, account inquiries
  - TPC-H (decision support systems; data mining)
    - Answer critical business questions

# PC Benchmarks

---

Name	Description
Business Winstone	Run a script that simulates a user switching among and running different applications
CC Winstone	Simulate multiple applications focused on content creation, such as Photoshop and audio-editing programs
Winbench	Run multiple scripts that test the performance of CPU, video system and disk

- ☐ What do you use instead?
  - Clock cycle, memory size, disk capacity, display
- ☐ Who will use the PC benchmark?

# Other Benchmarks

---

- ❑ EEMBC (embedded benchmark suites)
  - Autonomous driving
  - Mobile imaging
  - Internet of Things
  - Mobile devices, ...
  
- ❑ Be ready to develop your own benchmarks
  
- ❑ Many more benchmarks (e.g., parallel, deep learning, ...)



# Related Questions

---

- ❑ Know about computer performance benchmarking
- ❑ Why not hear about benchmarking of other products?
  - e.g., Consumer Reports
    - Power of consumers
- ❑ What about company/university benchmarking?
  - Consulting company

---

# Quantitative Engineering

## Computer Performance

3) How can we improve?

(Performance models)  
(Domain knowledge)

# Execution Time (Designer Perspective) (반복)

---

## ❑ Separation of concerns

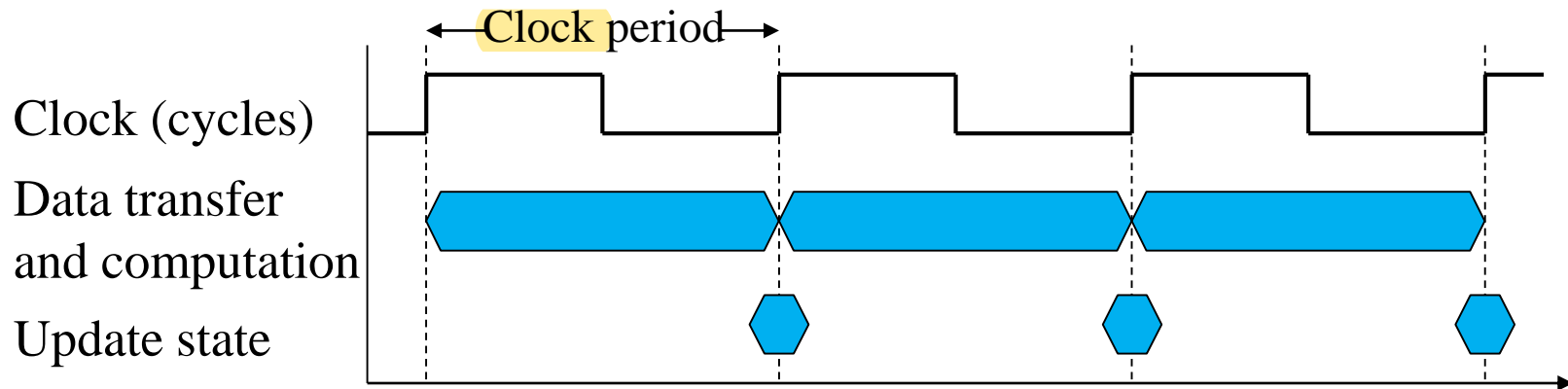
- Processor designer's focus: user CPU time
- OS designer's perspective: system CPU time
- I/O system designer's perspective: I/O time

## ❑ Processor designer's focus: CPU time

- User and system CPU time if OS is fixed

# CPU Clocking

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



- Clock period: duration of a clock cycle
  - e.g.,  $1 \text{ ns} = 1.0 \times 10^{-9} \text{ s}$
- Clock frequency (rate): cycles per second
  - e.g.,  $1.0 \text{ GHz} = 1.0 \times 10^9 \text{ Hz}$

# 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 improved by
  - Reducing number of clock cycles
  - Increasing clock rate (e.g., 1 GHz → 2GHz)
- ❑ Hardware designer must often trade off clock rate against cycle count

# CPU Time Example (read textbook)

---

- ❑ Computer A: 2GHz clock, 10s CPU time
- ❑ Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes  $1.2 \times$  clock cycles
- ❑ How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\begin{aligned}\text{Clock Cycles}_A &= \text{CPU Time}_A \times \text{Clock Rate}_A \\ &= 10\text{s} \times 2\text{GHz} = 20 \times 10^9\end{aligned}$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

# Performance Model

---

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

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

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

- ❑ Call it “CPU performance equation” (많은 모델 가능, 핵심은 효율성)
  - Minimize the product of three factors: IC, CPI, cct

# What determines performance? (미리보기)

□ CPU time = IC \* CPI \* clock cycle time

- |   |                              |    |            |
|---|------------------------------|----|------------|
|   | • ISA:                       | IC | (CPI, cct) |
|   | (Interface design)           |    |            |
| ↑ | • High-level implementation: | -  | CPI (cct)  |
|   | (e.g., pipeline, cache)      |    |            |
|   | • Low-level implementation:  | -  | - cct      |

□ 상위 레벨 설계는 하위 레벨에 영향 미침


□ Implementation 개선해도 IC 는 불변



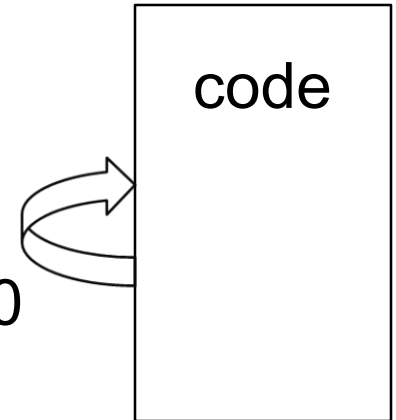
# Three Factors: IC, CPI, cct

---

## ❑ IC (instruction count)

- Number of machine instructions executed
- Dynamic (runtime) count:  실제 실행되는 instruction의 수
  - Dynamic vs. static
  - Static IC: size of executable file

loop  
n=10,000



- 같은 프로그램이라도 input data 다르면 변함
  - Input 고정하거나 여러 input 에 대한 평균 취하면 됨

# Three Factors: IC, CPI, cct

---

## ❑ IC (instruction count)

- Dynamic number of machine instructions executed

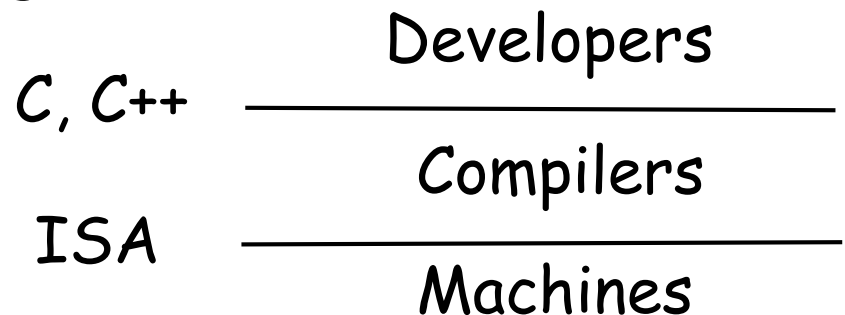
## ❑ 최상위 개념 IC는 ISA (I/F) 설계에서 결정됨

- ISA 확정되면, compiler 설계 가능

- CPU simulator 로 benchmark 돌리면 IC 얻음

- 이에 따라 implementation (CPI 및 cct )이 영향 받음

† e.g., RISC or CISC



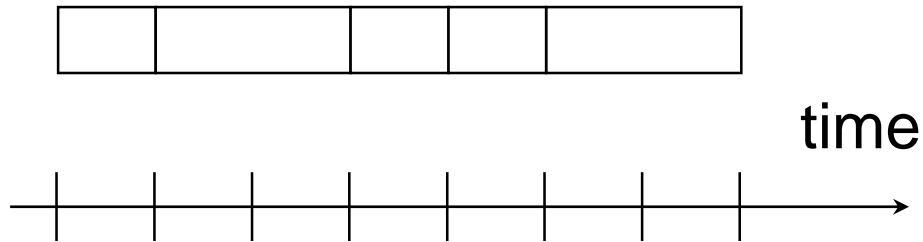
# Three Factors: IC, CPI, cct

---

- ❑ CPI (cycle per instruction): 차상위 개념, **high-level impl.**
  - No. of clock cycles necessary to execute an instruction
  - Instruction 마다 다름 (next slides 참조)
    - 최종적인 CPI 는 빈도에 따른 weighted average
      - † Dynamic (실제 실행되는 instruction 들만 계산)
    - 프로그램에 따라 또 같은 프로그램에서도 input data 에 따라 instruction mix 가 달라지므로 CPI 도 변함
      - † Benchmark 및 여러 input 에 대한 평균 취하면 됨
  - 설계된 ISA 구현시 (**high-level impl.**) CPI가 결정됨

# Different numbers of cycles for different instructions

---



- ❑ Multiplication takes more time than addition
  - Floating point operations take longer than integer ones
  - Accessing memory takes more time than accessing registers
  
- ❑ *Important point: changing the clock cycle time often changes the number of cycles required for various instructions*

# CPI in More Detail

- ❑ If different instruction classes take different numbers of cycles

Instructions	CPI	IC
Class I	1	10
Class II	2	60
Class III	3	30

Average CPI

$$\begin{aligned} &= 1 \cdot 0.1 + 2 \cdot 0.6 + 3 \cdot 0.3 \\ &= 2.2 \end{aligned}$$

- Weighted average CPI

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

Relative frequency

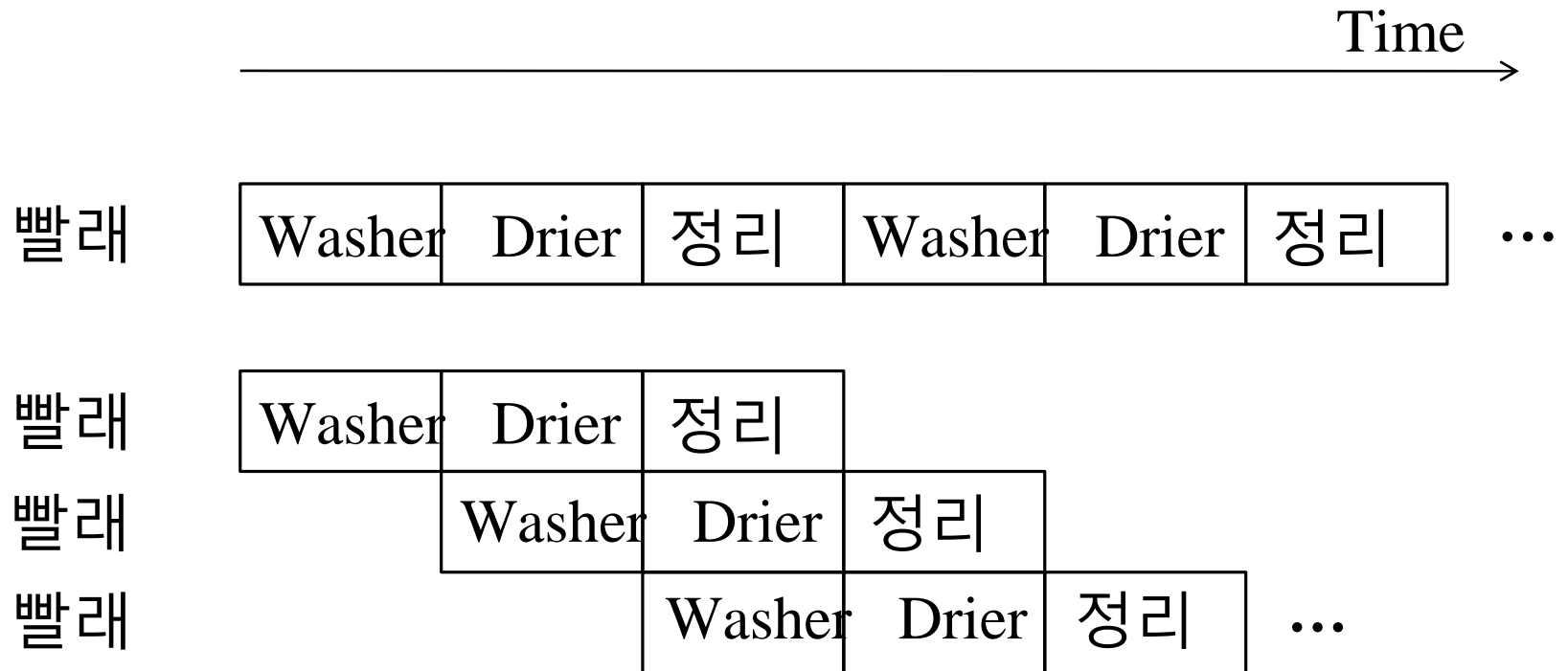
# CPU Implementation

---

- ❑ Key speedup techniques in high-level implementation
  - Pipelining
  - Cache memory
- ❑ Key techniques for improving CPI (and clock cycle time)
  - Heavily used in RISC (since 1980s)

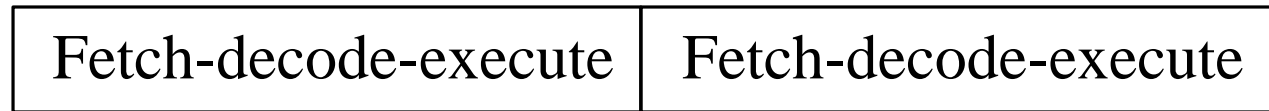
# Pipelining - General Speedup Technique (반복)

- 3-stage pipeline (e.g., washer-dryer example)
  - Speedup?

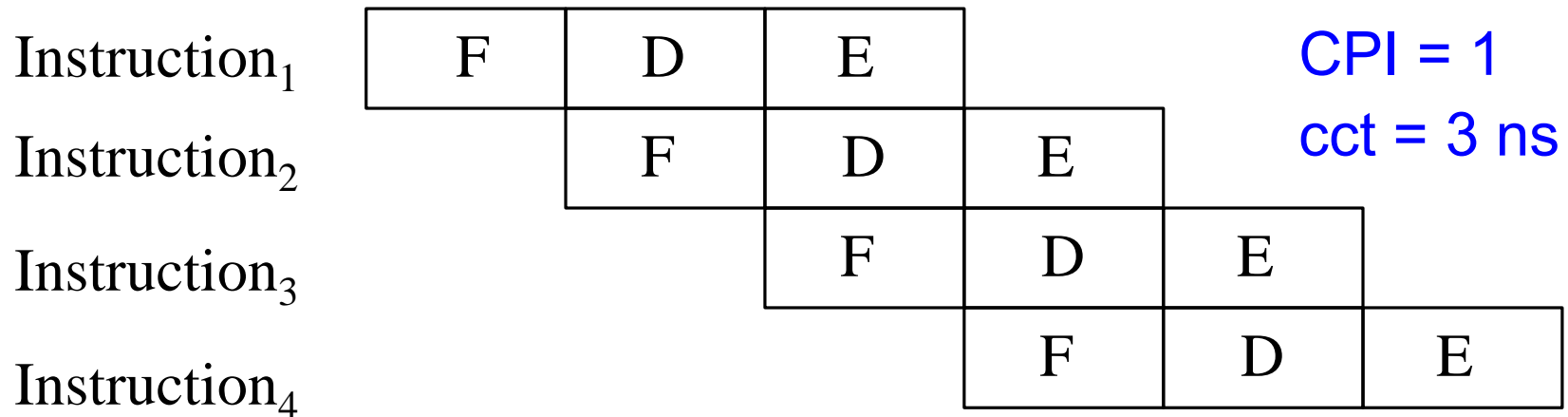
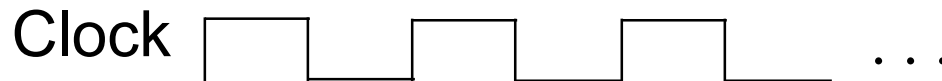
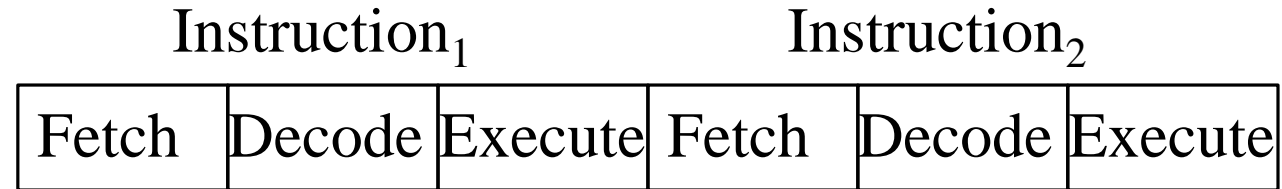


# Pipelining: 3-Stage Pipeline (IC: constant)

CPI = 1  
cct = 9 ns



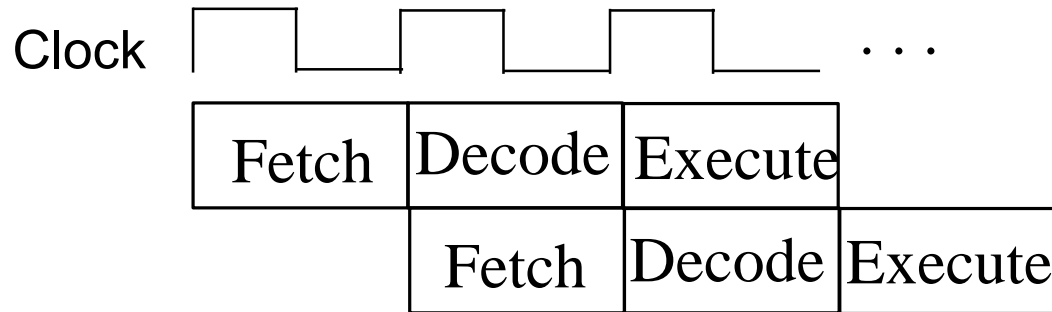
CPI = 3  
cct = 3 ns



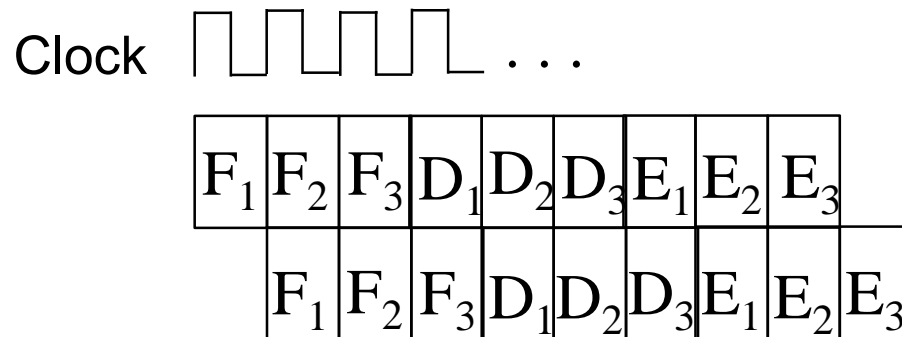


# Advanced Pipelining (IC: constant)

- ❑ Powerful processors - ideal speedup for 9-stage pipeline?



CPI = 1  
cct = 3 ns



CPI = 1  
cct = 1 ns

CPI ≈ 0.25  
cct = 1 ns

- ❑ What if we build 4 pipelines per processor?
- ❑ Role of cache memory? Why die size keep increasing?

# Three Factors: IC, CPI, cct

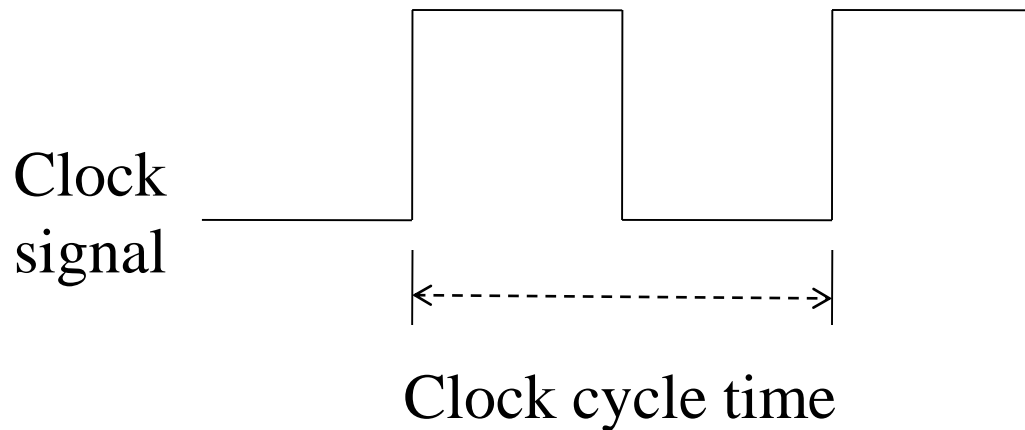
---

- ❑ CPI (cycle per instruction): 차상위 개념, high-level impl.
  - 설계된 ISA 구현시 (high-level impl.) CPI가 결정됨
    - ISA 설계시 이미 CPI (및 cct) 고려됨
  - CPI 설계 (high-level implementation)는 cct 에 파급 효과

# Three Factors: IC, CPI, cct

---

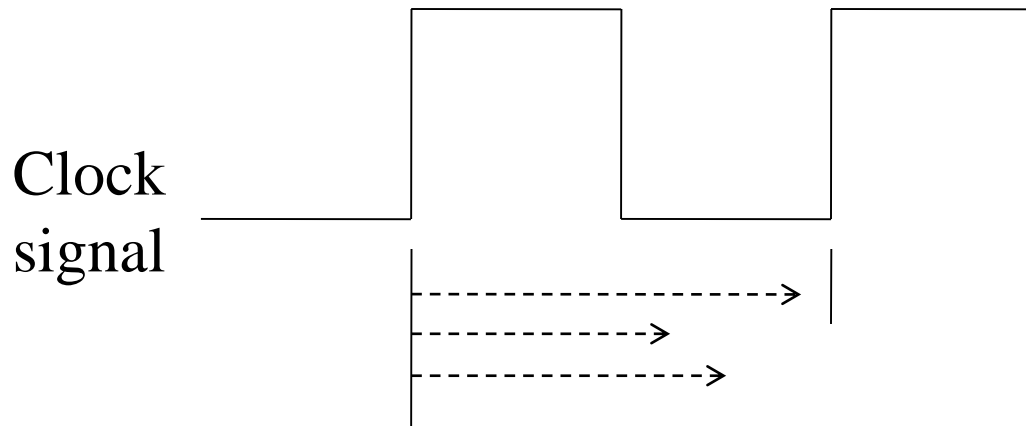
- ❑ cct (clock cycle time): 하위 개념, low-level implementation
  - Clock 한 주기의 길이
  - ISA 및 high-level implementation 이 결정 되면, 한 클럭 내에서 할 일 들이 결정됨
    - cct 를 줄일수록 좋음



# Three Factors: IC, CPI, cct

---

- ❑ cct (clock cycle time): 하위 개념, low-level implementation
  - 회로 설계자는 cct 를 줄이려 노력 (다른 요인도 고려)
    - Critical (slowest) path 를 찾아 이 부분을 개선함



- 이 부분은 컴퓨터 설계라기 보다 회로 설계 분야임
  - VLSI circuits design (or VLSI CAD or chip design)

# What determines performance? (다시보기)

---

- CPU performance equation (or performance model)

$$\text{CPU time} = \text{IC} * \text{CPI} * \text{clock cycle time}$$

- ISA: IC (CPI, cct)  
(Interface)
- High-level implementation: - CPI (cct)  
(e.g., pipeline, cache)
- Low-level circuit design: - - cct

- 상위 레벨 설계는 하위 레벨에 영향 미침

- Implementation 개선해도 IC 는 불변

---

# Quantitative Engineering Computer Performance

종합

# Today's Design Style: RISC

---

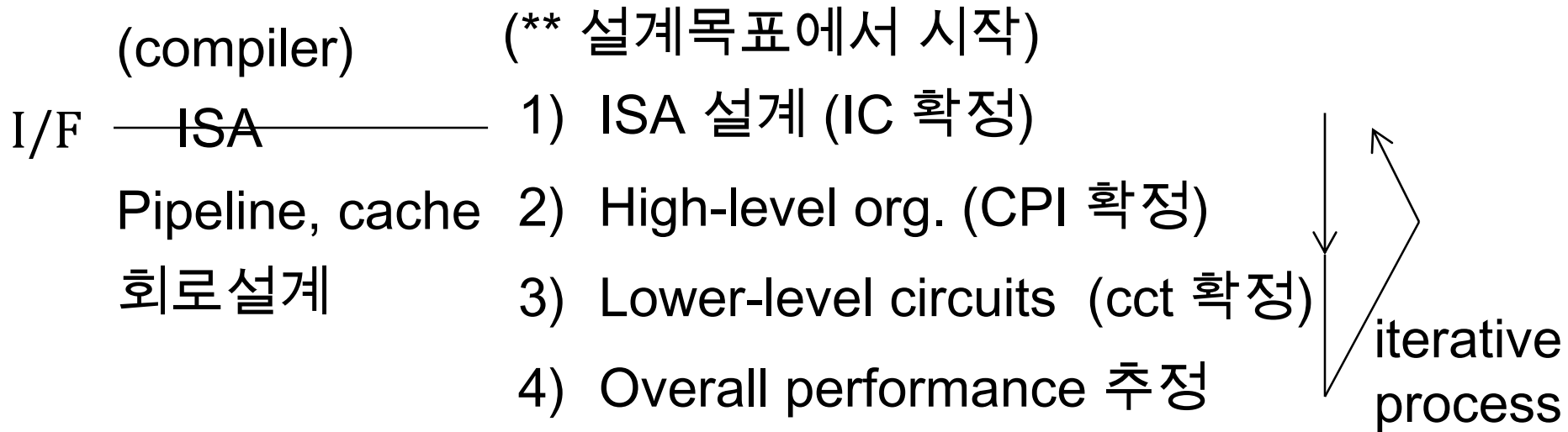
## ❑ RISC ISA

- Instruction 하나가 하는 일이 작음
  - IC 증가
- ISA 단순하여 pipelining 에 유리
  - CPI 감소 (Pipelining 쓰면 개념적으로  $CPI = 1$ )
  - Pipeline stage 수를 늘이면 cct 추가적으로 감소

## ❑ Title of text: computer organization and design

- Meaning of organization
- Meaning of design

# ISA Design and Implementation: Big Picture



□ ISA candidate 결정되면: build compiler and instruction-level simulator → IC

□ High-level organization: pipeline simulator → CPI

□ Low-level 회로설계 및 시뮬레이션 → cct

• 마지막으로 반도체 공장에서 CPU 제조

❖ 성능 요구조건 만족할 때까지 iterative cycles



# Computer Design

---

❑ 컴퓨터 설계라 하면 그 가능한 의미는?

- ALU
- Processor (ISA)
- Computers (processor, memory, I/O)
- Computer systems (machine + OS + compiler)
- Embedded systems (machine + kernel + application)
- Warehouse-scale computers (networking, power, cooling)
- Supercomputers

# Example (compare execution time)

- ❑ Two implementations for same ISA
  - Computer A: Cycle Time = 250ps, CPI = 2.0
  - Computer B: Cycle Time = 500ps, CPI = 1.2
- ❑ Which is faster, and by how much?

$$\text{CPUTime}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$

$$= 1 \times 2.0 \times 250\text{ps} = 1 \times 500\text{ps}$$

A is faster...

$$\text{CPUTime}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$

$$= 1 \times 1.2 \times 500\text{ps} = 1 \times 600\text{ps}$$

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

by this much

# Example (compare execution time)

---

- ❑ Alternative compiled code sequences using instructions in classes A, B, C (same CPU but two different compilers)

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

- Sequence 1: IC = 5
  - Clock Cycles  
 $= 2 \times 1 + 1 \times 2 + 2 \times 3$   
 $= 10$
- Avg. CPI =  $10/5 = 2.0$
- Sequence 2: IC = 6
  - Clock Cycles  
 $= 4 \times 1 + 1 \times 2 + 1 \times 3$   
 $= 9$
- Avg. CPI =  $9/6 = 1.5$

# What determines performance? (참고)

---

## CPU Performance Equation


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

### □ At higher-levels

- Algorithm: affects IC, possibly CPI
- Programming language and compiler: affects IC, CPI
  - “affect CPI” means “may use low-CPI instructions”
- Instruction set architecture: affects IC, CPI,  $T_c$

# Story of CPU Companies

---

- ❑ What is a good ISA? 
- ❑ What is a good implementation?

—————→ P3 (ISA 설계 및 구현)

→ → → ..... Clock cycle time (회로 설계 개선)

————→ ..... CPI (high-level impl. 개선)

—————→ P4 (new ISA 설계 및 구현)

→ → → .....

————→ .....

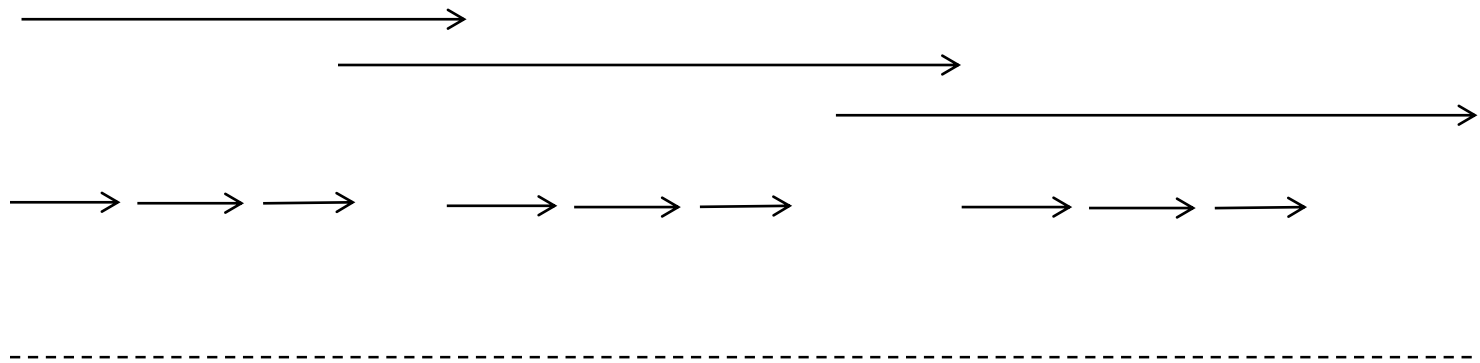
—————→ P5

- ❑ P3, P4, P5 (why new names? why new ISAs?)

# Story of Software Companies

---

- ❑ Major release (e.g., every few years)
  - Minor release (e.g., every year)
  - Interim versions (e.g., bug fixes as necessary)



- ❑ Version management and bug tracking

# Homework #5 (see Class Homepage)

1) Write a report summarizing the materials discussed in Topic 1-1

\*\* 문장으로 써도 좋고 파워포인트 형태의 개조식 정리도 좋음

2) Solve chapter 1 exercises: 2, 5, 7, 9, 11, 14

□ Due: see Blackboard

- Submit electronically to Blackboard

# Class Topics (클래스 홈페이지 참조)

- ❑ Part 1: Fundamental concepts and principles
- ❑ Part 2: 빠른 컴퓨터를 위한 ISA design
  - Topic 1 Computer performance and ISA design (Ch. 1)
    - ❑ 1-1 Performance evaluation & performance models
    - ❑ 1-2 RISC versus CISC, power limit
  - Topic 2 RISC (MIPS) instruction set (Chapter 2)
  - Topic 3 Computer arithmetic and ALU (Chapter 3)
- ❑ Part 3: ISA 의 효율적인 구현 (pipelining, cache memory)