

---

# Computer Performance

## Part 1

### Performance Benchmarking, CPU Performance Equation

Slides by Hennessy and Patterson (modified)

# Performance

---

- Why important?
  - Intelligent purchase (see through marketing hype)
  - Design
    - To answer what-if questions
      - How does the machine's instruction set affect performance?
        - LD R1, R31(+1) vs. LD R1, R31, 0x10000001
          - Many such details
- † Perspective of company sales executive

# Key Questions (Quantitative Paradigm)

---

## † Engineering

- Cost/performance (reliability, power consum., ...) trade-off
- Measurement, quantitative
- What is a good computer?
  - Performance measures
- How do we perform measurements?
  - Evaluation method
- What determines performance?
  - Factors, possible optimal combinations
- † Domain knowledge

# **Which of these airplanes has the best performance?**

<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?

# 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., tasks/transactions/... per hour
- 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
    - † Parallel computers: advanced issue
- We'll focus on response time for now...

# Execution Time

---

- UNIX utility: “time a.out↵”  
90.7s 12.9s 2:39 65%
  - Elapsed time: CPU time plus I/O time  
⇒ Separation of concerns
  - User and system CPU time  
⇒ Applications rely on OS services  
⇒ OS API not uniform
- Processor designer’s focus: user CPU time
  - OS designer’s perspective
  - I/O system designer’s perspective
  - System designer’s perspective
- What is system engineering?

# Measuring Execution Time

---

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance
- ❖ Our 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 25 seconds



# Understanding Performance (skip)

---

- 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

## II. Choosing programs to evaluate performance

---

- System manager: notion of workload
  - 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 Cooperation)
  - Develop benchmarks, publish results
  - Open Systems Group
    - Desktops, workstations, servers
    - CPU2006 (89, 92, 95, 2000)
    - Java server, power consumption, file server
  - High Performance Group
    - Large applications in scientific computing
    - Standard parallel programming APIs (OpenMP, MPI)
  - Graphics and Workstation Performance Group
  - Research Group

# SPEC CPU Benchmark 2000

Integer benchmarks		FP benchmarks	
Name	Description	Name	Type
gzip	Compression	wupwise	Quantum chromodynamics
vpr	FPGA circuit placement and routing	swim	Shallow water model
gcc	The Gnu C compiler	mgrid	Multigrid solver in 3-D potential field
mcf	Combinatorial optimization	applu	Parabolic/elliptic partial differential equation
crafty	Chess program	mesa	Three-dimensional graphics library
parser	Word processing program	galgel	Computational fluid dynamics
eon	Computer visualization	art	Image recognition using neural networks
perlbnk	perl application	equake	Seismic wave propagation simulation
gap	Group theory, interpreter	facerec	Image recognition of faces
vortex	Object-oriented database	ammp	Computational chemistry
bzip2	Compression	lucas	Primality testing
twolf	Place and rote simulator	fma3d	Crash simulation using finite-element method
		sixtrack	High-energy nuclear physics accelerator design
		apsi	Meteorology: pollutant distribution

**FIGURE 4.5 The SPEC CPU2000 benchmarks.** The 12 integer benchmarks in the left half of the table are written in C and C++, while the floating-point benchmarks in the right half are written in Fortran (77 or 90) and C. For more information on SPEC and on the SPEC benchmarks, see [www.spec.org](http://www.spec.org). The SPEC CPU benchmarks use wall clock time as the metric, but because there is little I/O, they measure CPU performance.

# SPEC CPU Benchmark 2006

## □ SPEC CPUint 2006 for Opteron X4 2356

Name	Description	Exec time	Ref time	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				11.7

# Benchmarking (Basic Principles)

- ❑ Reproducible
  - Disclose configurations and measurement conditions
- ❑ Performance/\$
  - Disclose cost
- ❑ Single number
  - e.g., SPECint2006, SPECfp2006
  - How do you get the number?
    - Reference machine
- ❑ Hard to cheat

# Benchmarking

Hardware		Software	
Model number	Precision WorkStation 410	O/S and version	Windows NT 4.0
CPU	700 MHz, Pentium III	Compilers and version	Intel C/C++ Compiler 4.5
Number of CPUs	1	Other software	See below
Primary cache	16KBI+16KBD on chip	File system type	NTFS
Secondary cache	256KB(I+D) on chip	System state	Default
Other cache	None		
Memory	256 MB ECC PC100 SDRAM		
Disk subsystem	SCSI		
Other hardware	None		

## SPEC CINT2000 base tuning parameters/notes/summary of changes:

+FDO: PASS1=-Qprof\_gen PASS2=-Qprof\_use

Base tuning: -QxK -Qipo\_wp shlW32M.lib +FDO

shlW32M.lib is the SmartHeap library V5.0 from MicroQuill [www.microquill.com](http://www.microquill.com)

Portability flags:

176.gcc: -Dalloca=\_alloca /F100000000 -Op

186.crafy: -DNT\_i386

253.perlbmk: -DSPEC\_CPU2000\_NTOS -DPERLDLL /MT

254.gap: -DSYS\_HAS\_CALLOC\_PROTO -DSYS\_HAS\_MALLOC\_PROTO

**Figure 1.14** The machine, software, and baseline tuning parameters for the CINT2000 base report on a Dell Precision WorkStation 410. These data are for the base CINT2000 report. The data are available online at [www.spec.org/osg/cpu2000/results/cpu2000.html](http://www.spec.org/osg/cpu2000/results/cpu2000.html).

# 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*



# SPEC Power Benchmark

---

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

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

# SPECpower\_ssj2008 for X4

---

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

---

- ❑ 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
    - Benchmark for decision support systems
    - Answer critical business questions
  - TPC-Energy

# PC Benchmarks

Benchmark name	Benchmark description
Business Winstone	Runs a script consisting of Netscape Navigator and several office suite products (Microsoft, Corel, WordPerfect). The script simulates a user switching among and running different applications.
CC Winstone	Simulates multiple applications focused on content creation, such as Photoshop, Premiere, Navigator, and various audio-editing programs.
Winbench	Runs a variety of scripts that test CPU performance, video system performance, and disk performance using kernels focused on each subsystem.

**Figure 1.11** A sample of some of the many PC benchmarks. The first two are scripts

❑ What do you use instead?

- Clock cycle, memory size, disk capacity, display

# EEMBC Benchmarks

Benchmark type	Number of kernels	Example benchmarks
Automotive/industrial	16	6 microbenchmarks (arithmetic operations, pointer chasing, memory performance, matrix arithmetic, table lookup, bit manipulation), 5 automobile control benchmarks, and 5 filter or FFT benchmarks
Consumer	5	5 multimedia benchmarks (JPEG compress/decompress, filtering, and RGB conversions)
Networking	3	Shortest-path calculation, IP routing, and packet flow operations
Office automation	4	Graphics and text benchmarks (Bézier curve calculation, dithering, image rotation, text processing)
Telecommunications	6	Filtering and DSP benchmarks (autocorrelation, FFT, decoder, encoder)

**Figure 1.13** The EEMBC benchmark suite, consisting of 34 kernels in five different classes. See [www.eembc.org](http://www.eembc.org) for more information on the benchmarks and for scores.

❑ Be ready to develop your own benchmark

# Interesting Question

---

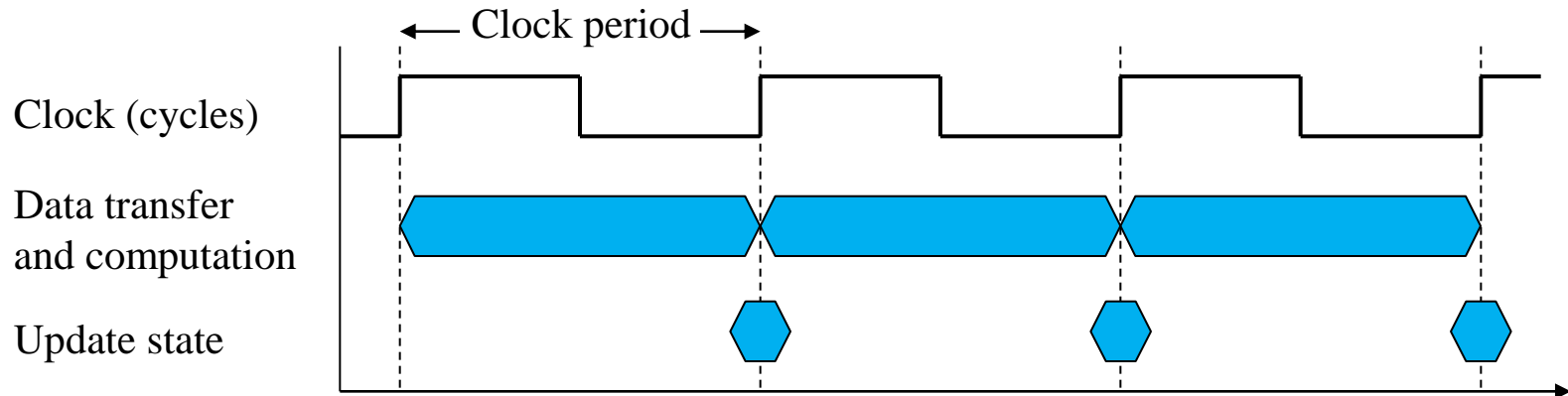
- Know about computer performance benchmarking
- Why not hear about benchmarking of other products?
  - E.g., Consumer Reports
    - † Power of consumers

---

## **III. How to improve (Performance Model)**

# CPU Clocking

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



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



# 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
  - 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?

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

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

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

# Instruction Count and CPI

---

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

$\text{CPUTime} = \text{Instruction Count} \times \text{CPI} \times \text{ClockCycleTime}$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{ClockRate}}$$

- Call it “CPU performance equation”
  - Minimize the product of three factors
    - † IC, CPI, cct

# Three Factors: IC, CPI, cct

---

- IC (instruction count)
  - Number of machine instructions executed
  - Dynamic (runtime) count: 실제 실행되는 instruction 의 수
    - † Static IC: size of executable file
  - 같은 프로그램이라도 input data 다르면 변함
    - † Input 고정하거나 여러 input 에 대한 평균 취하면 됨
- 설계자는 결과적인 IC 추정하며 ISA 결정
  - 이에 따른 CPI 및 cct 도 추정
- RISC vs. CISC

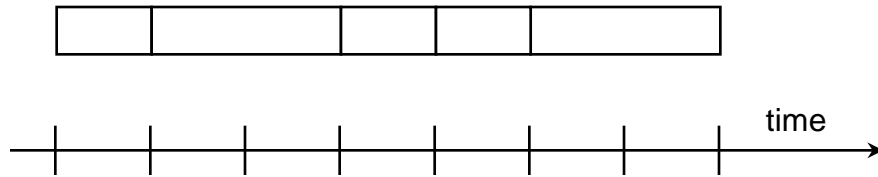
# Three Factors: IC, CPI, cct

---

- CPI (cycle per instruction)
  - No. of clock cycles necessary to execute an instruction
    - Dynamic (실제 실행되는 instruction 들만으로 계산됨)
  - Instruction 마다 다름 (next slides 참조)
    - 일반연산, multiplication, floating point 연산 등
      - 최종적인 CPI 는 빈도에 따른 weighted average
    - 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
  - 설계자는 ISA 를 결정할 때, 최종 CPI (및 cct) 목표를 세우고, 이를 위해 각 instruction 군에 대한 CPI 목표를 정하여 **high-level organization** 설계 진행

# 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

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

## ■ 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

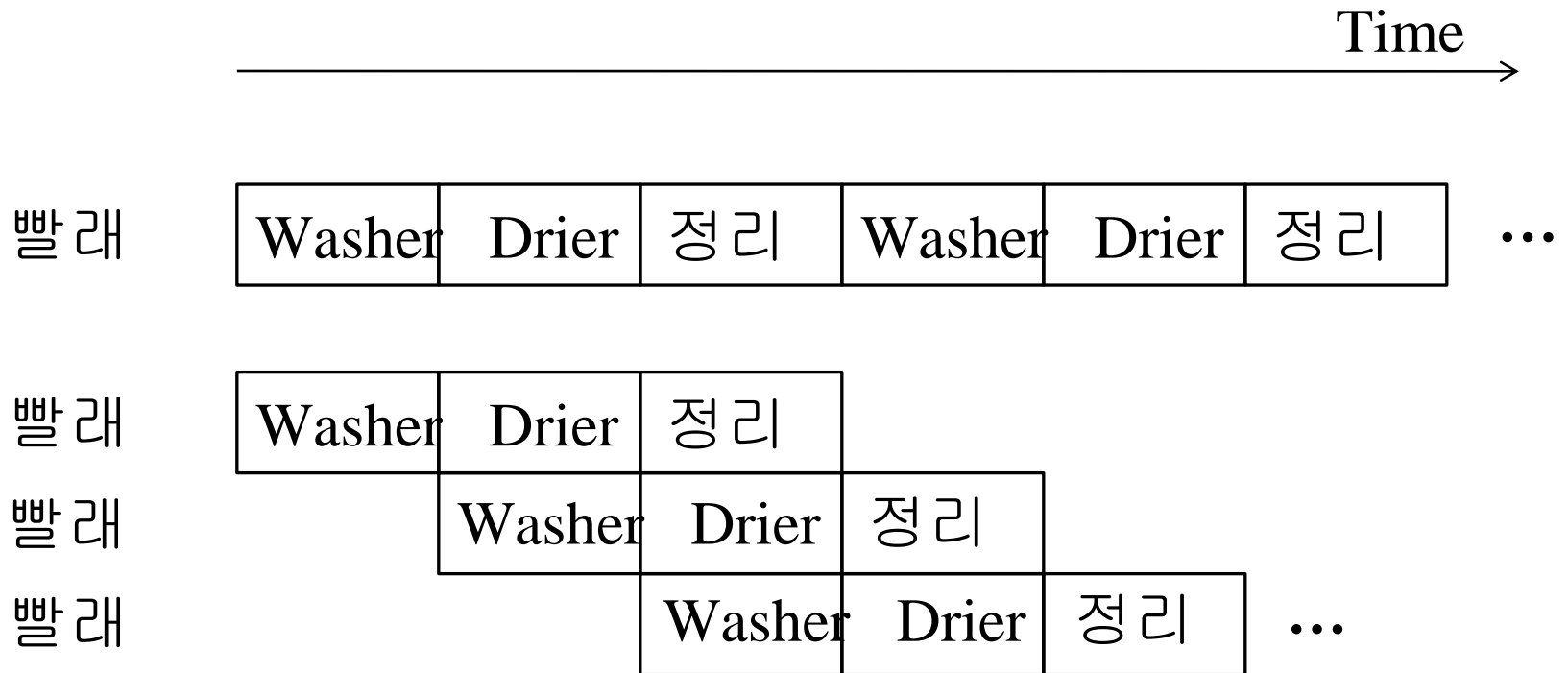
---

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



# Pipelining - General Speedup Technique

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

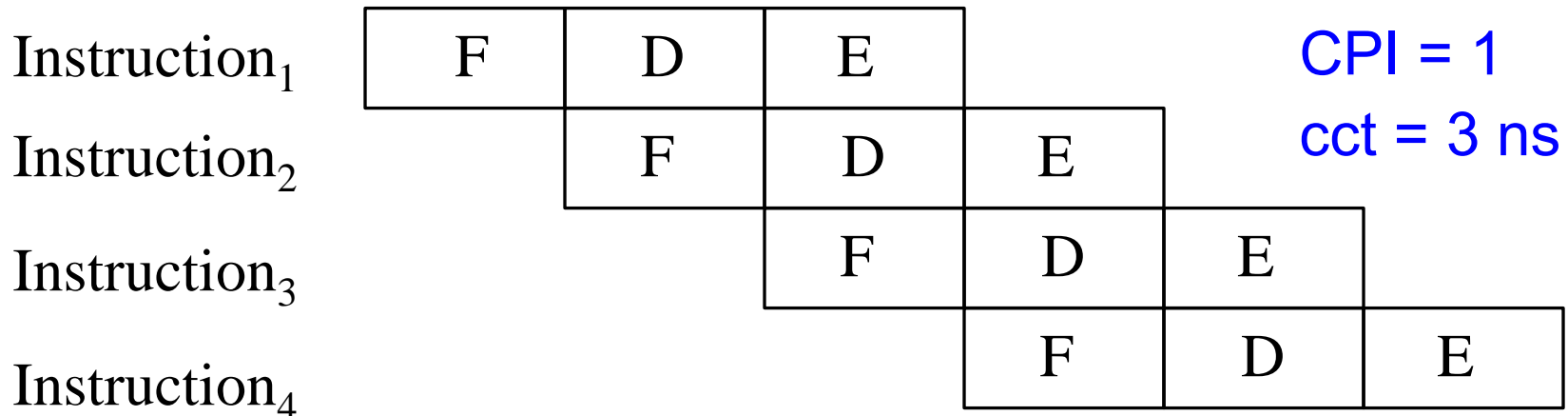
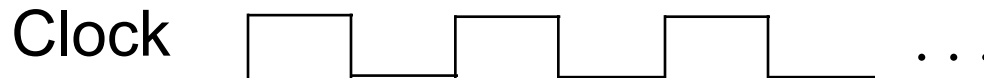
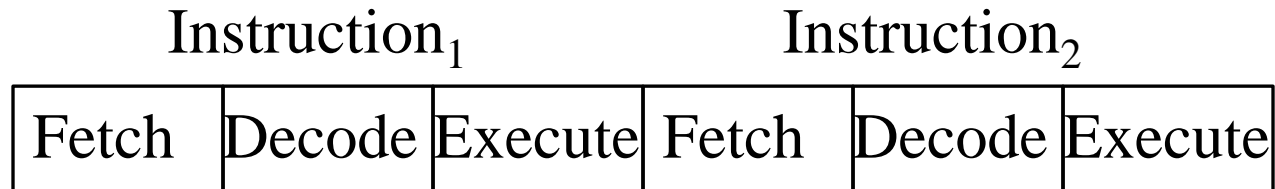


# Pipelining

- 3-stage pipeline for fetch-decode-execute

CPI = 3

cct = 3 ns



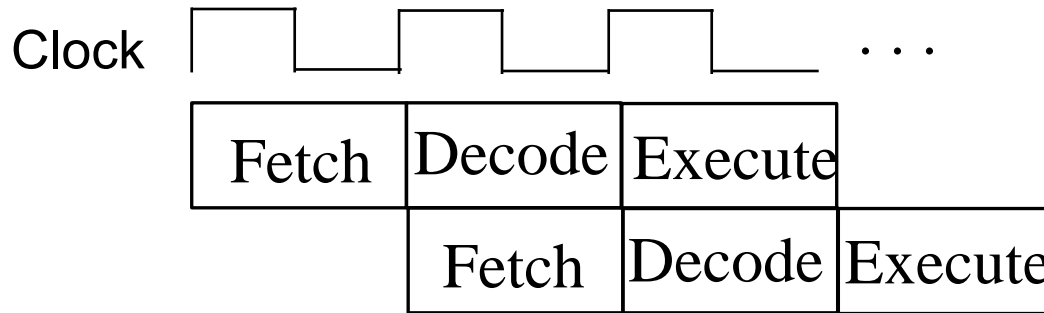
CPI = 1

cct = 3 ns

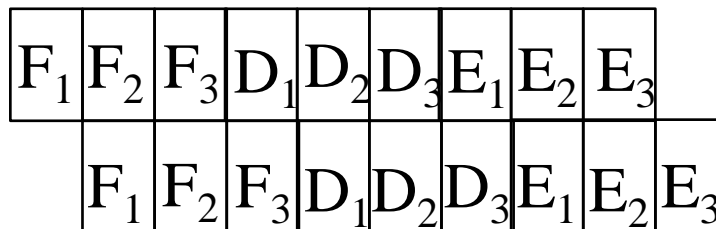
- What is the role of cache memory?
- Why did the die size keep increasing?

# Advanced Pipelining

- Powerful server processors
  - Ideal speedup for 9-stage pipeline?



CPI = 1  
cct = 3 ns



CPI = 1  
cct = 1 ns

- What if we build 4 pipelines per processor? CPI ≈ 0.25
- Why did the die size keep increasing? cct = 1 ns

# Three Factors: IC, CPI, cct

---

- IC (instruction count)
- CPI (cycle per instruction)
- cct (clock cycle time)
  - 클럭 한 주기의 길이
  - ISA 및 high-level organization 이 결정 되면, 한 클럭 내에서 할 일 들이 결정됨
  - 회로 설계자는 이 일을 가장 빨리 수행할 수 있도록 설계하여 cct 를 줄임
    - Critical (slowest) path 를 찾아 이 부분을 개선함
  - 이 부분은 컴퓨터 설계라기 보다 회로 설계 색채가 강함
    - VLSI circuits design (or VLSI CAD or chip design)

# What determines performance?

---

† CPU performance equation

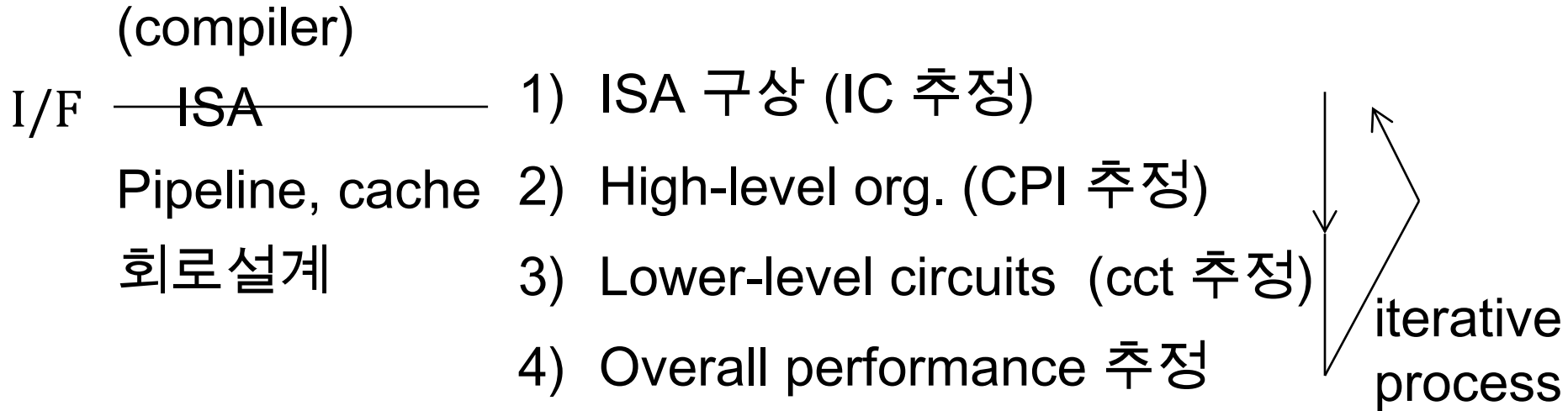
$$\begin{aligned}\text{CPU time} &= \text{\#instructions to execute} * \text{CPI} * \text{clock cycle time} \\ &= \text{IC} * \text{CPI} * \text{clock cycle time}\end{aligned}$$

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

† 상위 레벨 설계는 하위 레벨에 영향 미침

† Smart compiler의 영향은?

# ISA Design and Implementation: Big Picture



- ISA candidate 결정되면: build compiler and instruction-level simulator → IC
- High-level org. candidate: pipeline simulator → CPI
- 만족스러우면 (불만족이면 ISA and/or high-level org. 수정)
  - CPU 구현 (low-level 회로설계 시뮬레이션까지) → cct
  - 마지막으로 반도체 공장에서 CPU 제조

# ISA Design and Implementation: Big Picture

---

- Observations
  - ISA design = processor design 의 모든 것
    - Processor 는 컴퓨터라는 기계의 핵심
  - 이것을 다루는 본 교과목의 구성
    - Key concepts in Computer Science and Engineering
    - Performance and ISA design, real ISA
    - High-level organizations (pipelining and cache)
      - † 전자공학(VLSI 회로 설계, 반도체공학)과의 연관성
- † 컴퓨터 설계라 하면 그 가능한 의미는?
  - CPU, computer system (CPU + memory + I/O)
  - Computer system + OS + compiler
  - Warehouse-scale computer (networking, power, cooling)

# Three Factors: IC, CPI, cct (반복 정리)

---

- IC

- ISA 결정되고, compiler 만들면 확정 (즉, CPU 가 제공하는 서비스가 확정되면)
  - 이후 CPU (instruction-level) simulator 만들어 IC 최종 정량화
    - CPU (즉 hardware)가 아직 구현되지 않았더라도 가능
  - ISA 영향 예: RISC는 instruction 간단하여 IC 증가함
  - Compiler 영향 예: smart compiler 는 IC 줄이는 기법 적용
- † 물론 ISA 확정 전에 이것의 hardware 구현 (즉, high-level organization 및 low-level 회로설계에 대해 구체적으로 구상함 - 그래야 CPI, cct 가 추정되고 성능이 예측되므로)



# Three Factors: IC, CPI, cct (반복 정리)

---

- CPI, cct
  - CPU hardware design 시 최종 확정됨 (그러나 앞의 iterative design process에서 보듯이, ISA 설계시 이미 목표치가 거의 확정됨)
    - CPI: high-level organization (앞의 pipelining 자료 참조)
    - cct: 회로설계 및 high-level organization
  - ISA 는 CPI, cct 에 영향 미침
    - 예: RISC 는 이들을 줄임

# Three Factors: IC, CPI, cct (반복 정리)

---

- CPI
  - CPU 에 대한 high level organization (어떤 기능 모듈들을 어떻게 연동 하여 동작할 것인가; 주로 pipelining 관련) 확정되면 CPI 결정됨
    - 예: pipelined organization, cache
  - Low-level 회로 설계 없어도 pipeline 시뮬레이션을 통해 CPI 정량화
  - CPI 는 상위 설계에서 결정된 ISA 의 영향 받음
    - 예: RISC 는 instruction 단순하여 CPI 낮아짐
- + 1980년대 후반 출현한 RISC 는 CPI 줄이기 위해 pipeline 과 cache 집중 사용 (중간고사 이후 공부 주제)

# Three Factors: IC, CPI, cct (반복 정리)

---

† High-level org. 확정되면 세부적인 회로 설계 시작함

- Clock cycle time
  - Low-level 회로 설계 끝나야 확정됨 (물론 ISA 설계시 목표치는 거의 확정됨)
    - Clock 주기를 짧게 하기 위하여 critical path (slowest path) 찾아 회로 설계 개선함
    - Computer design 이라기 보다 회로 설계의 개념
  - 상위 설계에서 결정된 ISA, high-level org. 영향 받음
    - 예: RISC 는 instruction 단순하여 cct 작아짐
    - 예: pipeline organization (stage 수)의 직접적인 영향 받음; stage 수 늘어나면 cct 작아짐

# 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?

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

A is faster...

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

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

by this much

# CPI Example

---

- Alternative compiled code sequences using instructions in classes A, B, C

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?

---

## 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 및 구현)

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

————→ ..... CPI (high-level org. 개선)

—————→ P4 (New ISA 및 구현)

→ → → .....

————→ .....

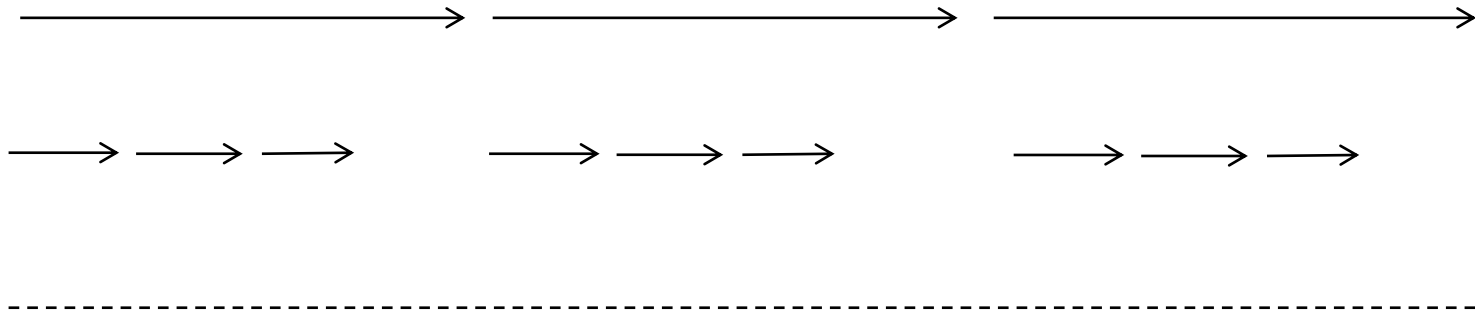
—————→ P5

- † Difference: P3, P4, P5 (why new names? Why change?)
- † Intel vs. AMD
- † Other CPUs (MIPS, Sparc, PowerPC) much different?

# Story of Software Companies

---

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



- † Version management and bug tracking
  - Are you aware of GNU tools?