

COMPUTER SYSTEM AND ARCHITECTURE REVIEW

Jo, Heeseung

Representing Information

Information = Bits + Context

- Computers manipulate representations of things
- Things are represented as binary digits
- What can you represent with N bits?
 - 2^N things
 - Numbers, characters, pixels, positions, source code, executable files, machine instructions, ...
 - Depends on what operations you do on them

	01110011	01100101	01101101	01101001	01110011	01100101	01101101	01101001
(char)	's'	'e'	'm'	'i'	's'	'e'	'm'	'i'
(int)	1768777075				1768777075			
(double)	7.03168990329170808178... x 10 ¹⁹⁹							

What You Learned

How programs are translated into the machine language

- And how the hardware executes them

The hardware/software interface

What determines program performance

- And how it can be improved

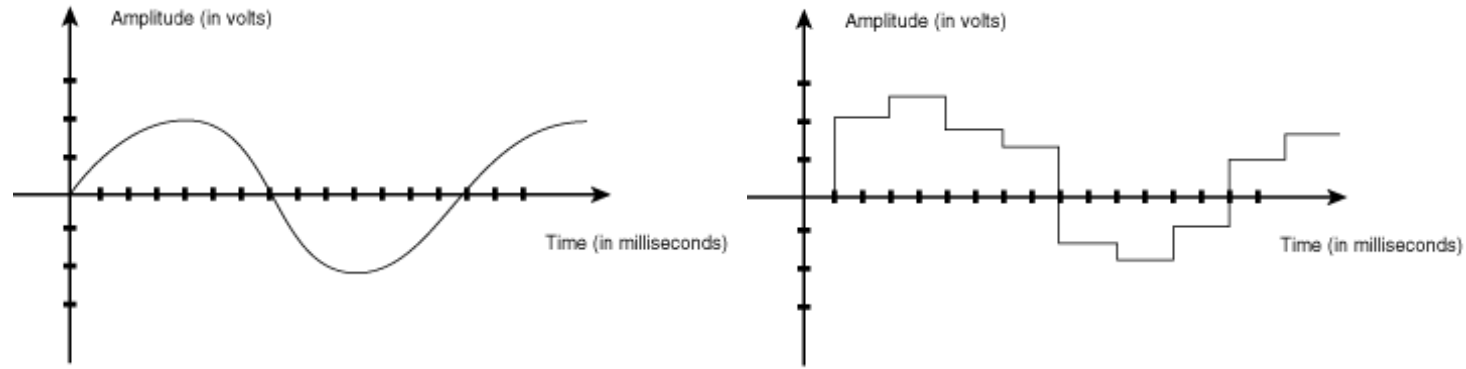
How hardware designers improve performance

What is parallel processing

Introduction

The advent of the digital age

- Analog vs. digital?



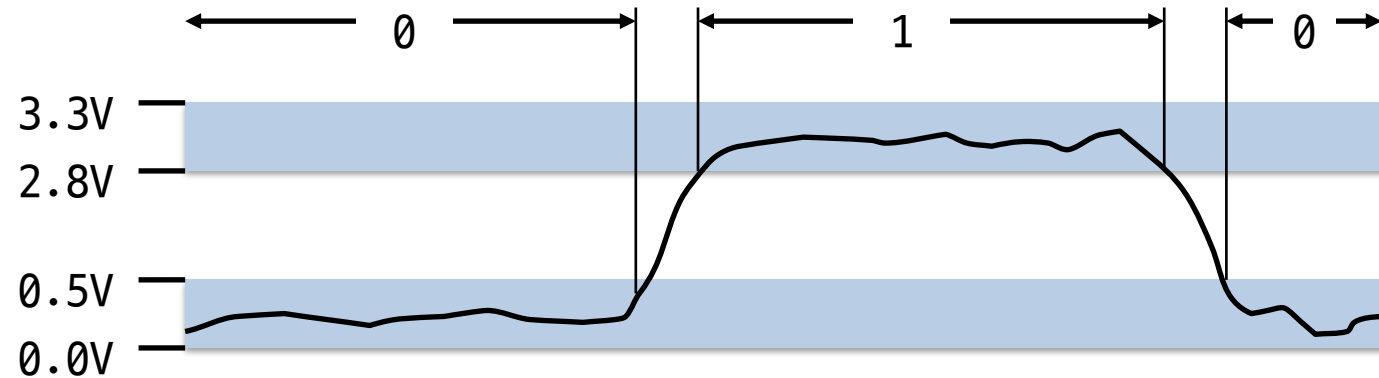
- Compact disc (CD)
 - 44.1 KHz, 16-bit, 2-channel
- MP3
 - A digital audio encoding with lossy data compression

Binary Representations

Why not base 10 representation?

- Easy to store with bistable elements
- Straightforward implementation of arithmetic functions
- Reliably transmitted on noisy and inaccurate wires

Electronic implementation



Encoding Byte Values

Byte = 8 bits

- Binary: 00000000_2 to 11111111_2
- Octal: 000_8 to 377_8
 - An integer constant that begins with 0 is an octal number in C
- Decimal: 0_{10} to 255_{10}
 - First digit must not be 0 in C
- Hexadecimal: 00_{16} to FF_{16}
 - Base 16 number representation
 - Use characters '0' to '9' and 'A' to 'F'
 - Write $FA1D37B_{16}$ in C as `0xFA1D37B` or `0xfa1d37b`

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Boolean Algebra (1)

Developed by George Boole in 1849

- Algebraic representation of logic
 - Encode "True" as 1 and "False" as 0

And

- $A \& B = 1$ when both $A=1$ and $B=1$

$\&$	0	1
0	0	0
1	0	1

Not

- $\sim A = 1$ when $A=0$

$\sim$	
0	1
1	0

Or

- $A \mid B = 1$ when either $A=1$ or $B=1$

$\mid$	0	1
0	0	1
1	1	1

Exclusive-Or (Xor)

- $A \wedge B = 1$ when either $A=1$ or $B=1$, but not both

$\wedge$	0	1
0	0	1
1	1	0

Boolean Algebra (2)

0	0	1	1
0	1	0	1

X

Y

0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Constant 0

X & Y ; AND

$\sim (X \rightarrow Y)$

X

$\sim (Y \rightarrow X)$

Y

$X \wedge Y$; XOR

$X \mid Y$; OR

$\sim (X \mid Y)$; NOR

$\sim (X \wedge Y)$; X-NOR

$\sim Y$

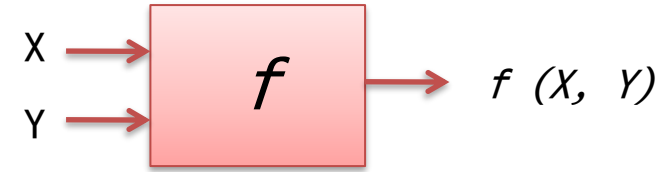
$Y \rightarrow X$

$\sim X$

$X \rightarrow Y$; Implication

$\sim (X \& Y)$; NAND

Constant 1

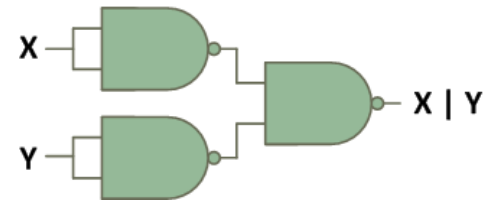
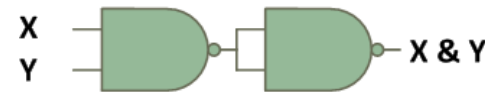


Basic operations: AND(&), OR(|), NOT(~)

$$X \wedge Y = (X \& \sim Y) \mid (\sim X \& Y)$$

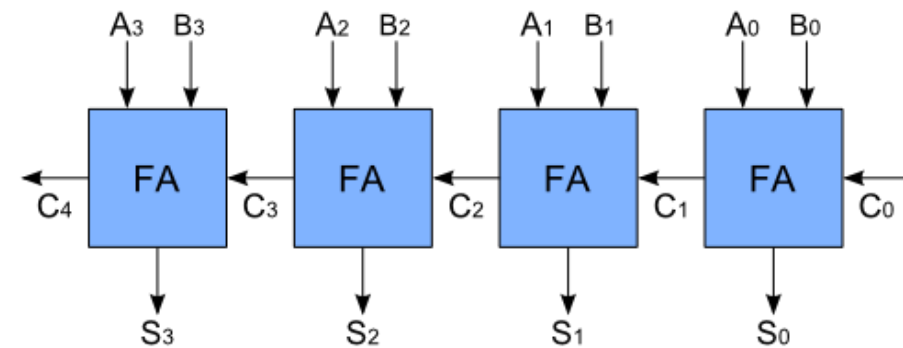
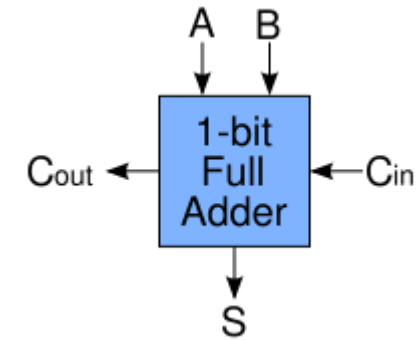
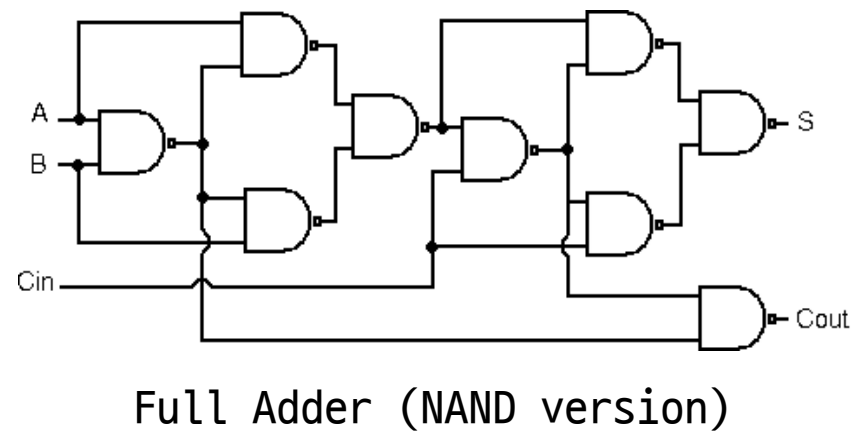
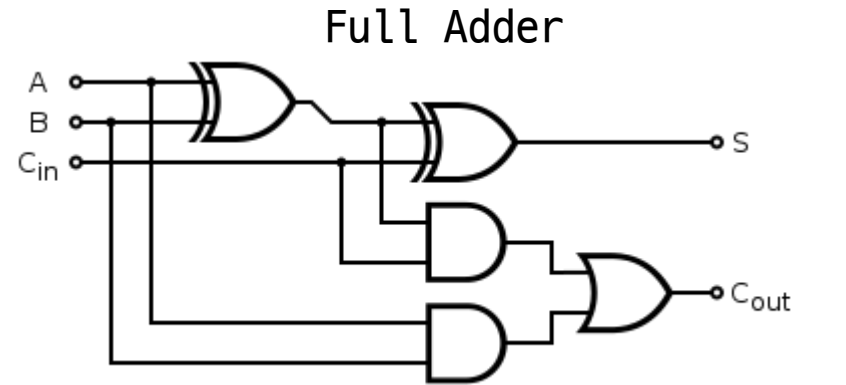
$$X \rightarrow Y = \sim X \mid Y$$

A complete set: NAND = $\sim (X \& Y)$



Combinational Logic

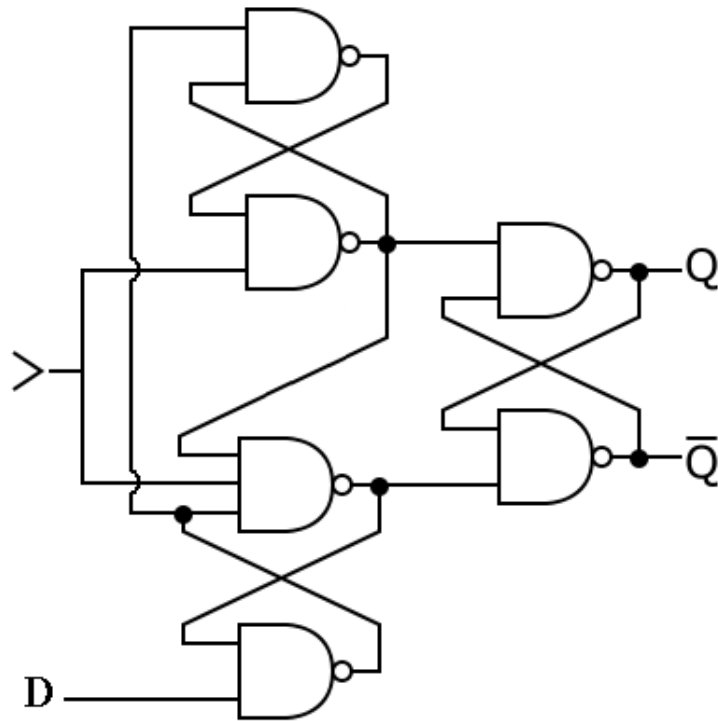
Adder



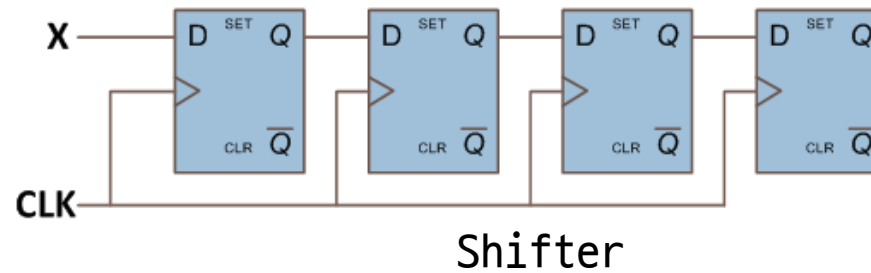
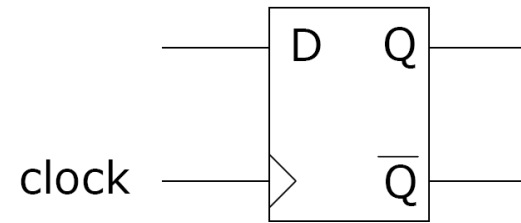
4-bit Ripple Carry Adder

Sequential Logic

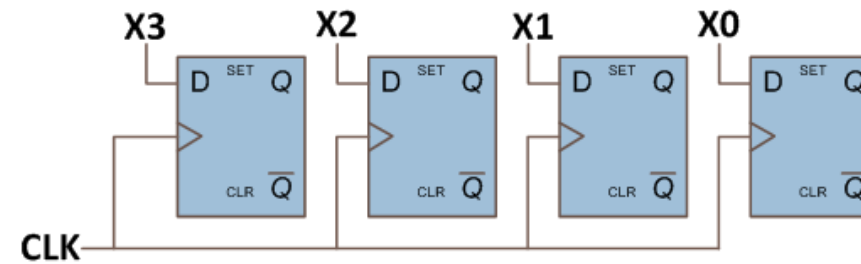
Flip-flops



Edge triggered D flip-flop



Shifter

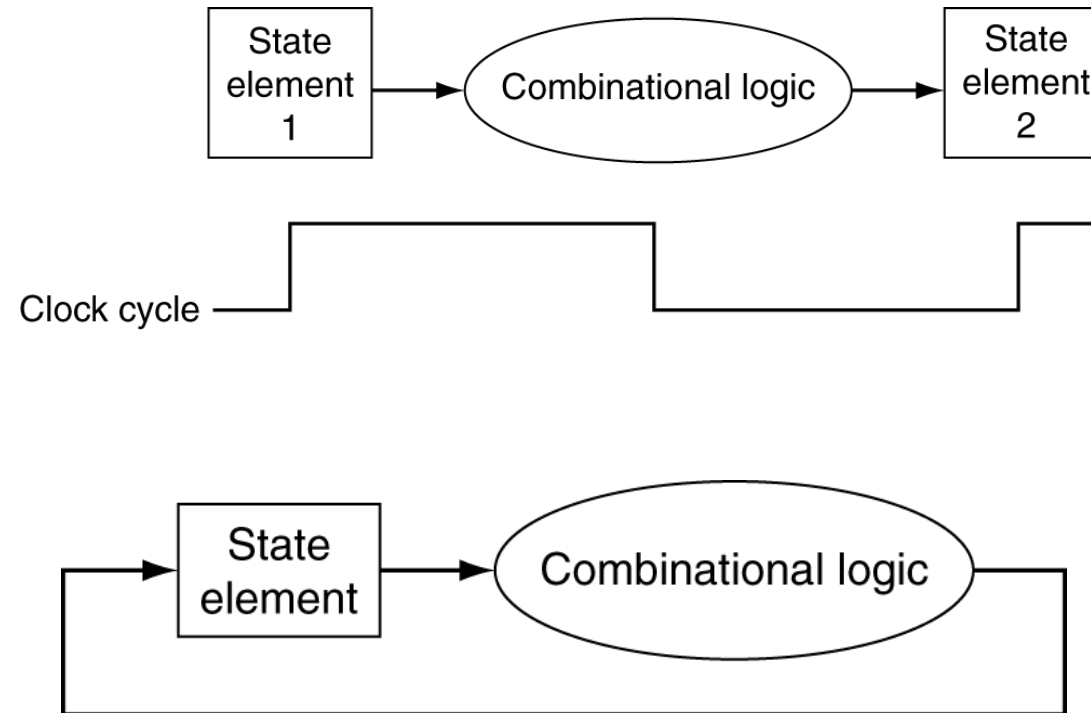


4-bit register

Clocking Methodology

Combinational logic transforms data during clock cycles

- Between clock edges
- Input from state elements, output to state element
- Longest delay determines clock period

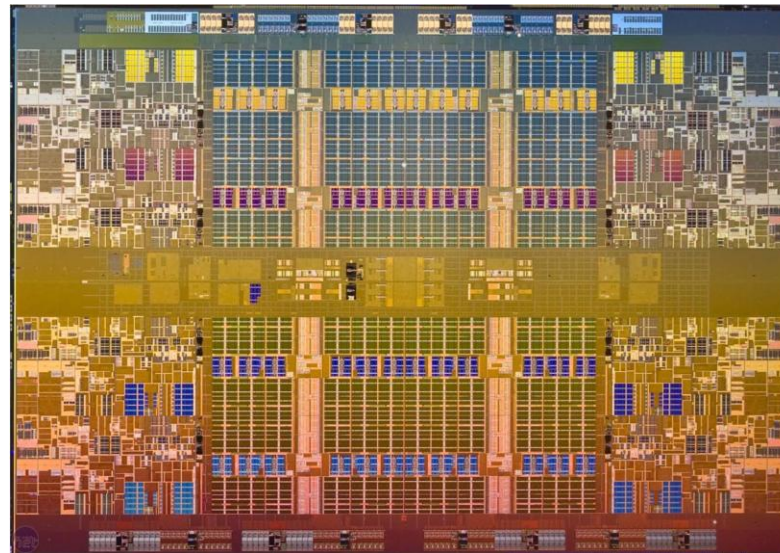


Digital Systems

Summary

- Boolean algebra is a mathematical foundation for modern digital systems
- Boolean algebra provides an effective means of describing circuits built with switches
 - Claude Shannon in the late 1930's
- You can build any digital systems with NAND gates
- A NAND gate can be easily built with CMOS transistors
- The transistor is the basic building block for digital systems

Intel Xeon 7560 (8-core): 2.3B transistors



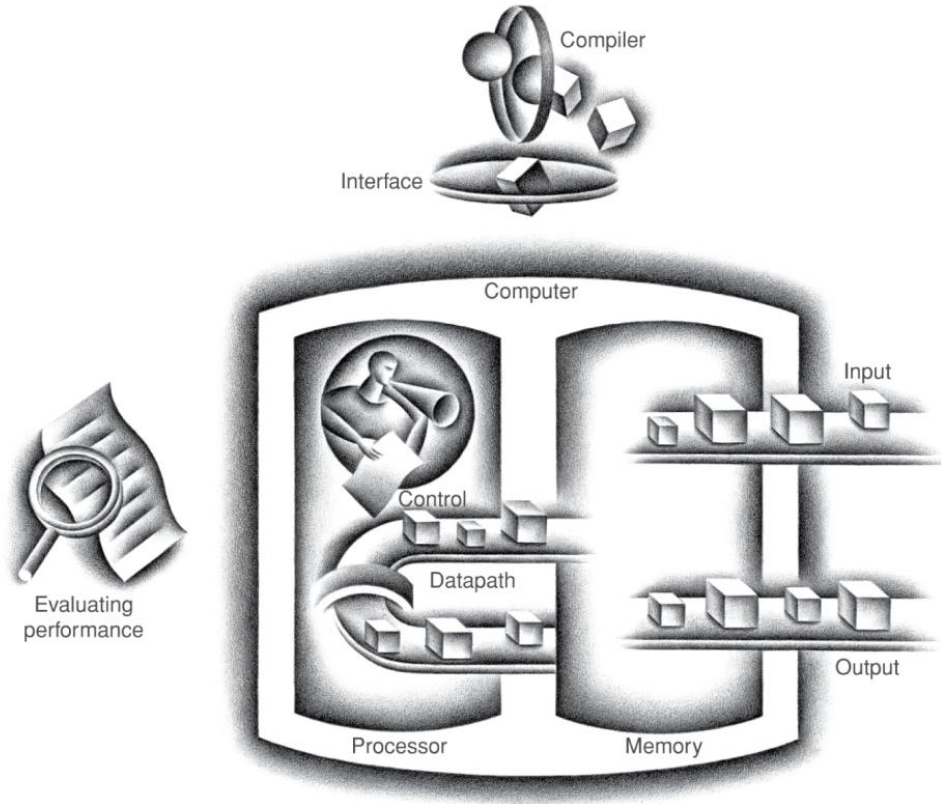
Components of a Computer

Same components for all kinds of computer

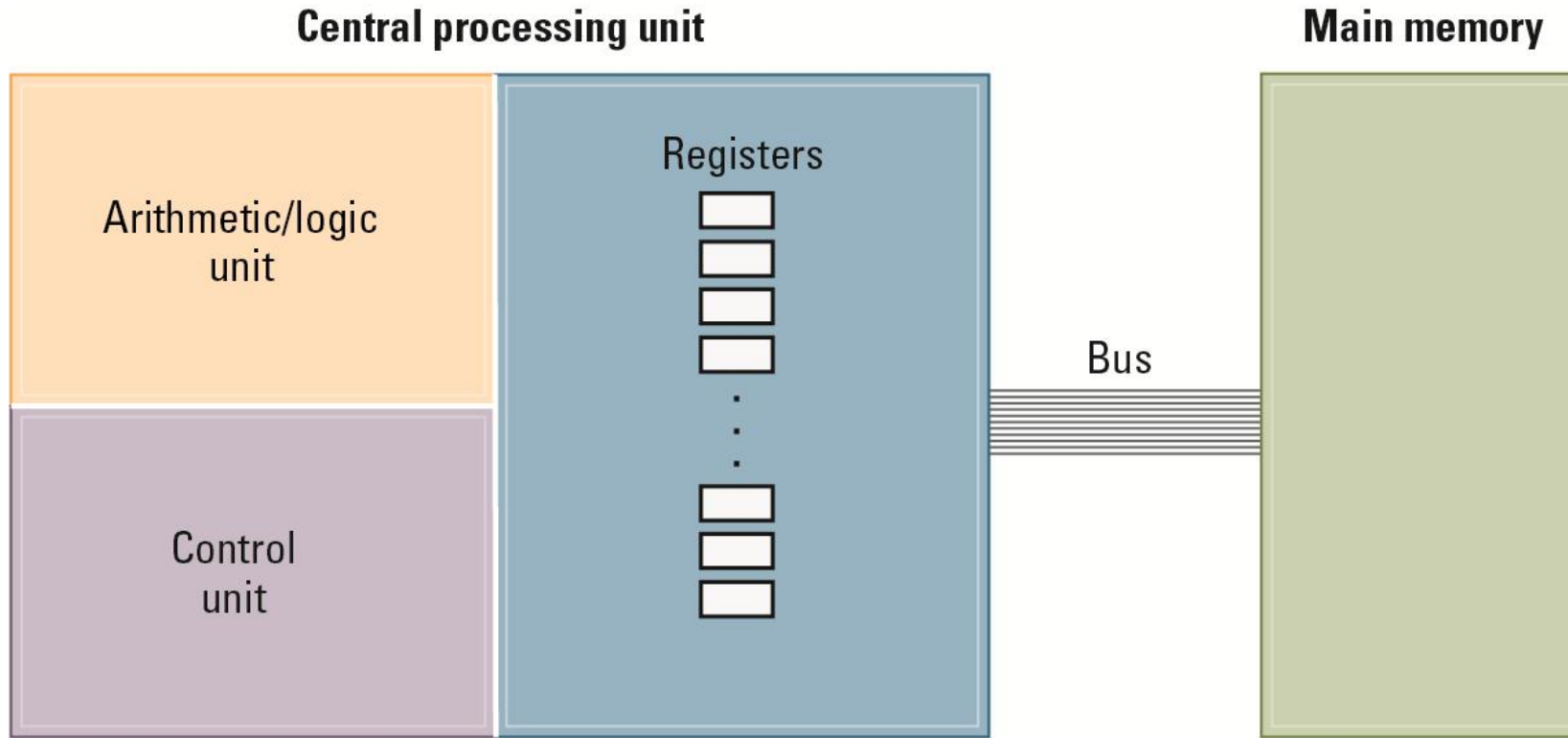
- Desktop, server, embedded

Input/output includes

- User-interface devices
 - Display, keyboard, mouse
- Storage devices
 - Hard disk, CD/DVD, flash
- Network adapters
 - For communicating with other computers



컴퓨터의 구성요소



중앙처리장치의 주요 구성요소

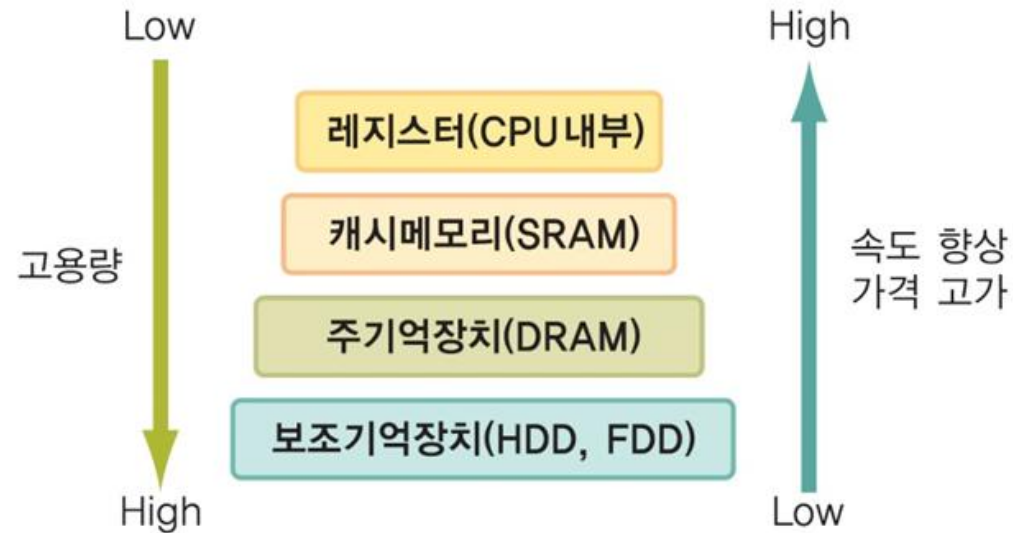
레지스터 장치(register unit)

- 레지스터(Register)
 - CPU내에서 정보를 임시로 저장하기 위해 사용
 - 여러 개의 레지스터를 가짐 (register file)
 - 성능에 매우 중요한 요소이므로 가격과 성능을 고려하여 결정
- 레지스터 종류
 - 범용 레지스터 : CPU에서 처리되는 데이터를 위한 임시 저장공간
 - 연산장치 회로의 입력들을 저장하고 있거나, 연산장치에서 계산된 결과를 저장하기 위한 공간으로 이용
 - 용도 지정 레지스터 : 임의 사용 불가능
 - 명령 레지스터, 프로그램 카운터 등

컴퓨터 내의 기억장치의 계층

기억장치 계층의 필요

- 기억장치의 속도와 용량, 가격과 그 쓰임새를 고려
- 기억장치의 속도가 빠르면 가격이 비쌈
- 동일한 비용으로 속도를 유지하려면 용량은 작아져야 함



컴퓨터 내의 기억장치의 계층

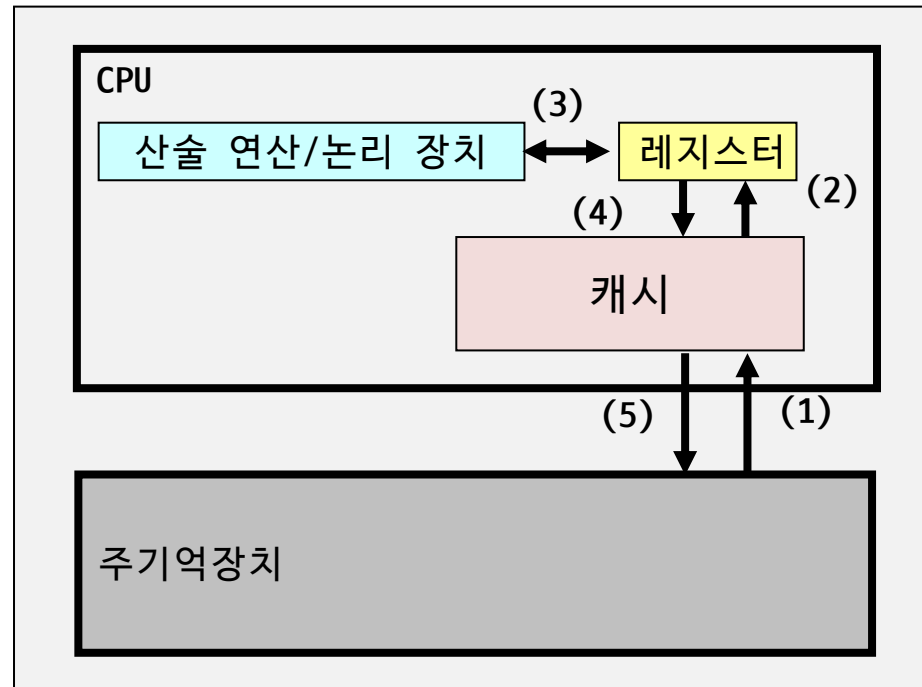
다양한 기억장치의 이용

- 레지스터 : 곧바로 사용될 데이터 보관
- 캐시 메모리
 - CPU 내부에 위치한 고속 메모리
 - 현재 자주 사용하는 주기억장치 부분의 복사본 유지
 - 캐시 메모리에 일어나는 변경들은 모아서 적절한 시점에 한꺼번에 주기억장치로 전달
 - 레지스터보다 느리지만 메인메모리보다 빠른 속도
 - 왜 쓰나? -> 주기억장치가 느리기 때문
- 주기억장치 : 가까운 시간 안에 사용될 데이터 보관
- 보조기억장치 : 당장 필요하지는 않은 데이터 보관

데이터 연산 처리 과정

주기억장치에 저장된 데이터에 대한 연산을 수행하려면..

- 제어장치가 데이터를 주기억장치로부터 범용 레지스터로 전송
- 어느 레지스터가 데이터를 갖고 있는지 연산장치에 알려주고,
- 연산장치 안의 적절한 회로를 작동시키고,
- 연산장치에게 결과를 받을 레지스터가 어느 것인지 알려줌



Understanding Performance

Algorithm

- Determines number of operations executed

Programming language, compiler, architecture

- Determine number of machine instructions executed per operation

Processor and memory system

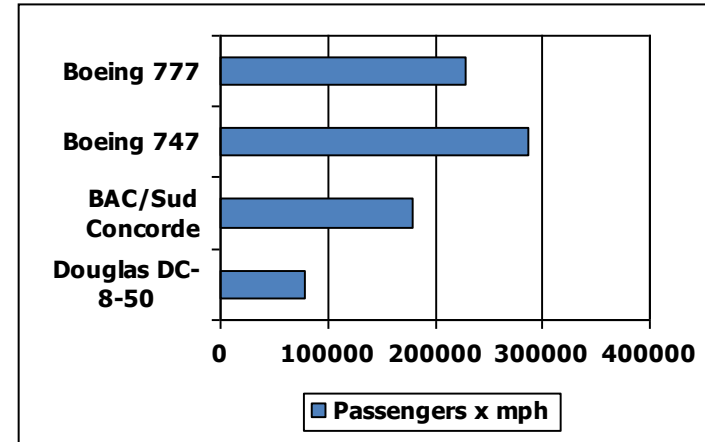
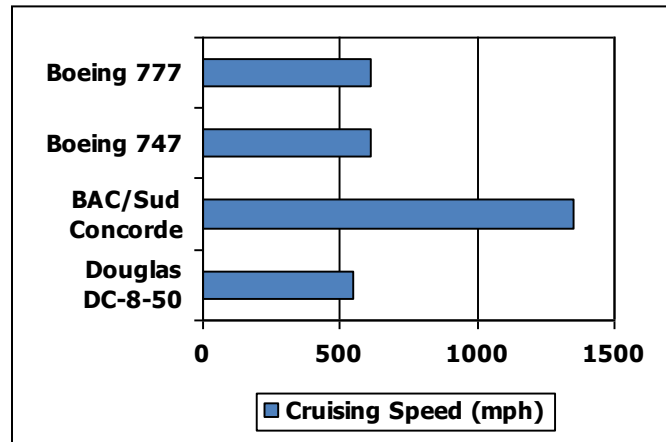
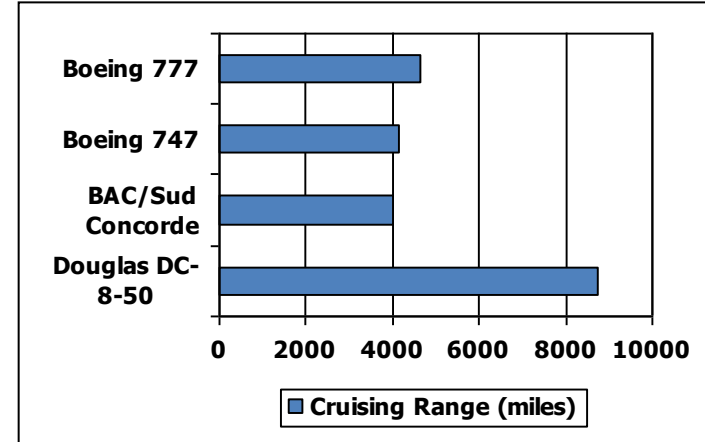
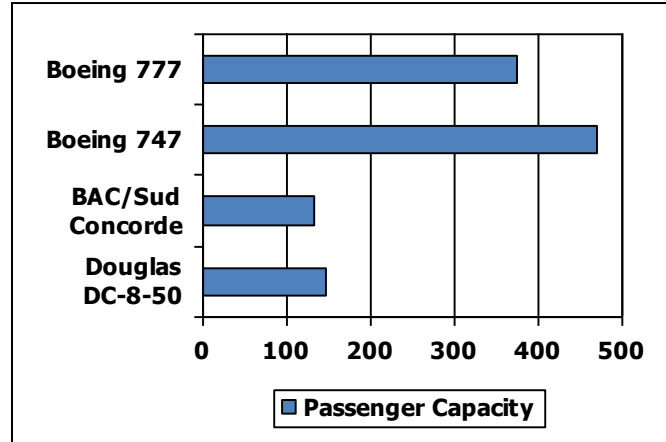
- Determine how fast instructions are executed

I/O system (including OS)

- Determines how fast I/O operations are executed

Defining Performance

Which airplane has the best performance?



Response Time and Throughput

Response time

- How long it takes to do a task

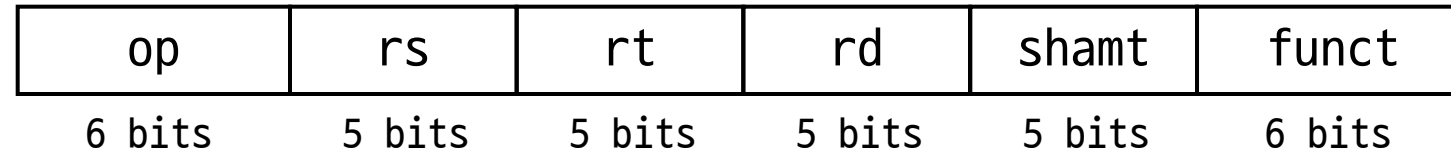
Throughput

- 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?

MIPS R-format Instructions



Instruction fields

- op: operation code (opcode)
- rs: first source register number
- rt: second source register number
- rd: destination register number
- shamt: shift amount (00000 for now)
- funct: function code (extends opcode)

R-format Example

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

add \$t0, \$s1, \$s2

special	\$s1	\$s2	\$t0	0	add
---------	------	------	------	---	-----

0	17	18	8	0	32
---	----	----	---	---	----

000000	10001	10010	01000	00000	100000
--------	-------	-------	-------	-------	--------

$$00000010001100100100000000100000_2 = 02324020_{16}$$

프로그램 내장(stored program) 개념

초기의 컴퓨터 : hard-wired programming

- CPU의 회로를 변경하도록 설계됨
- 스위치 조작으로 프로그래밍
- 데이터는 주기억장치에 저장, 프로그램 CPU의 일부로 생각

프로그램 내장 개념

- 프로그램도 비트 패턴으로 인코딩되어 주기억장치에 저장
- CPU는 주기억장치에서 명령들을 읽어와서 실행
- 또한 실행될 프로그램을 주기억장치 안에서 쉽게 변경

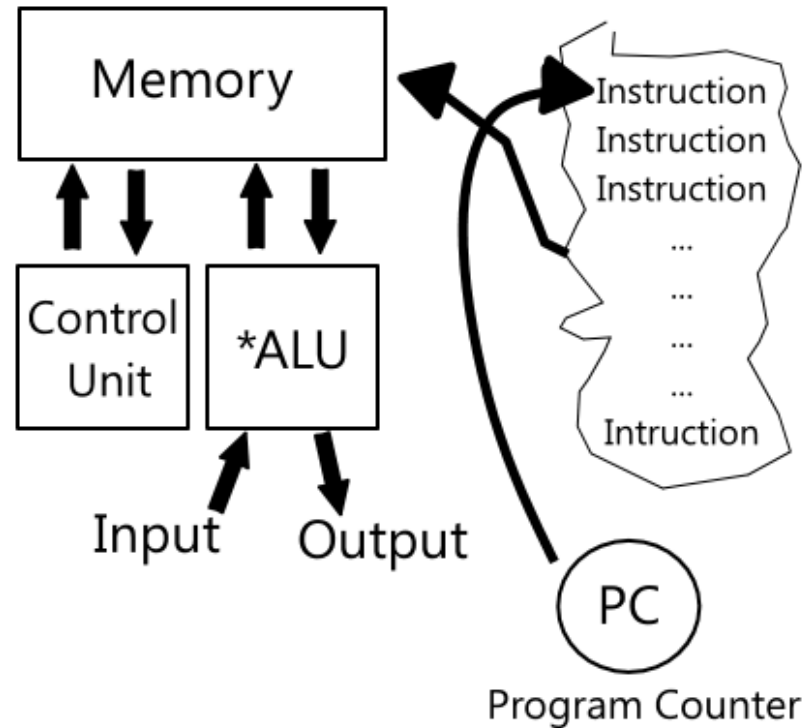
프로그램 내장형 컴퓨터(폰 노이만 기계)

- 컴퓨터 프로그램을 데이터와 동일하게 주기억장치 안에 저장
- 제어장치가 메모리에서 프로그램을 가져와서 명령을 해석하고 실행하도록 설계
- 제어장치 회로를 변경하는 대신, 컴퓨터의 메모리 내용을 변경하는 것만으로도 컴퓨터가 수행할 프로그램 변경가능

프로그램의 실행

CPU가 프로그램을 실행시키려면?

- 프로그램들이 메모리에 위치
 - 프로그램이 실행되려면 먼저 프로그램이 실행 가능한 상태로 준비되어 있어야 함
- 컴퓨터는 필요한 대로 명령들을 메모리에서 CPU로 복사
- 일단 CPU로 옮겨진 명령은 해석되고 실행됨



*ALU(Arithmetic Logic Unit)

Levels of Program Code

High-level language

- Level of abstraction closer to problem domain
- Provides for productivity and portability

Assembly language

- Textual representation of instructions

Hardware representation

- Binary digits (bits)
- Encoded instructions and data

High-level
language
program
(in C)

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

Compiler

Assembly
language
program
(for MIPS)

```
swap:
    muli $2, $5, 4
    add  $2, $4, $2
    lw   $15, 0($2)
    lw   $16, 4($2)
    sw   $16, 0($2)
    sw   $15, 4($2)
    jr   $31
```

Assembler

Binary machine
language
program
(for MIPS)

```
000000001010000100000000000011000
000000000000110000001100000100001
100011000110001000000000000000000
100011001111001000000000000000100
101011001111001000000000000000000
101011000110001000000000000000100
00000011111000000000000000001000
```

Translation and Startup

