Computer Architecture Review

Jo, Heeseung

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

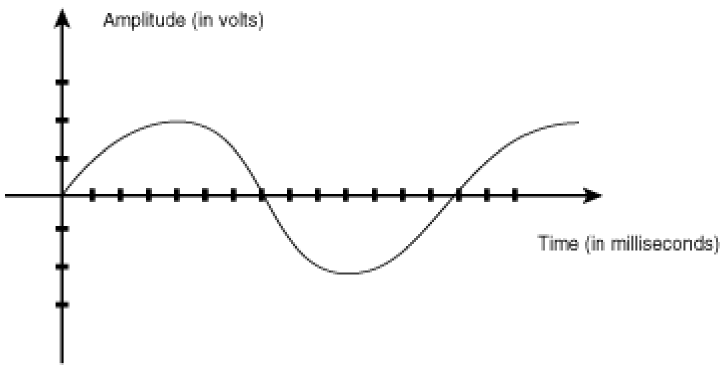
What is parallel processing

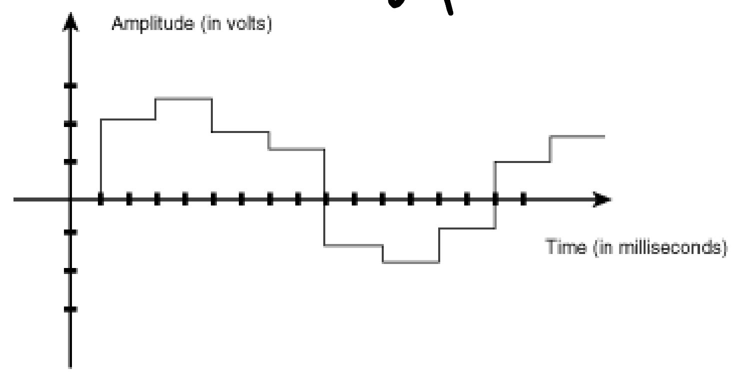
Introduction

연속적이냐 → analog
아니냐 → digital

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

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

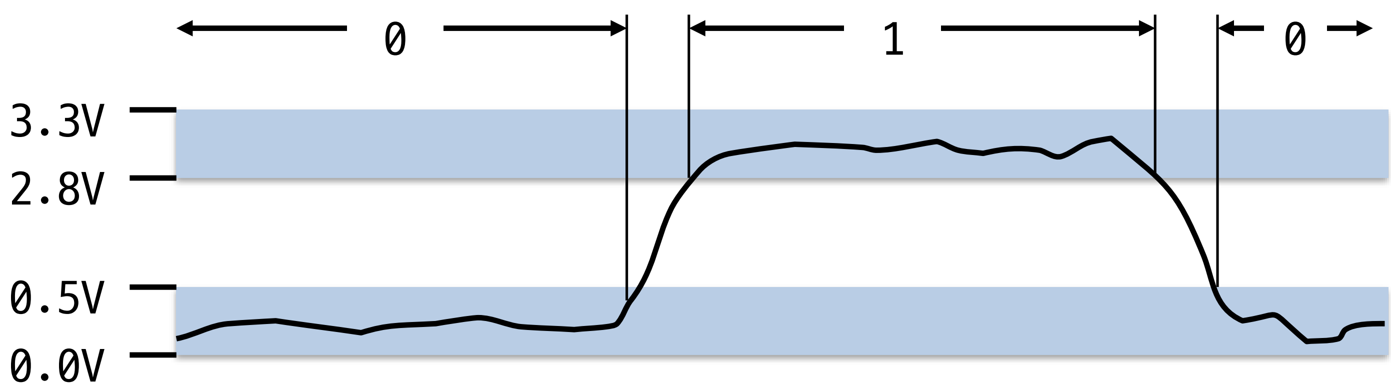
	01110011	01100101	01101101	01101001	01110011	01100101	01101101	01101001
(char)	's'	'e'	'm'	'i'	's'	'e'	'm'	'i'
(int)	1768777075				1768777075			
(double)	$7.0316899032917080817 8\ldots \times 10^{199}$							

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

ex) int k = ~~0~~2;
↳ 8진법으로 준거다.
(in C)

Byte = 8 bits

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

Or

- A¦B = 1 when either A=1 or B=1

|	0	1
0	0	1
1	1	1

Not

- ~A = 1 when A=0

~	
0	1
1	0

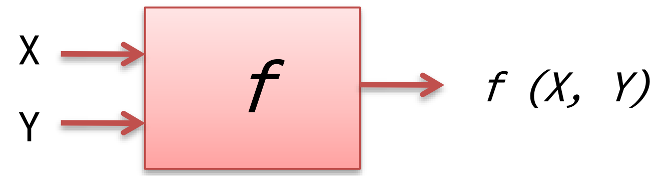
Exclusive-Or (Xor)

- A^B = 1 when either A=1 or B=1, but not both

^	0	1
0	0	1
1	1	0

Boolean Algebra (2)

0	0	1	1	X	
0	1	0	1	Y	
0	0	0	0	Constant 0	
0	0	0	1	X & Y	; AND
0	0	1	0	~ (X → Y)	
0	0	1	1	X	
0	1	0	0	~ (Y → X)	
0	1	0	1	Y	
0	1	1	0	X ^ Y	; XOR
0	1	1	1	X ¦ Y	; OR
1	0	0	0	~ (X ¦ Y)	; NOR
1	0	0	1	~ (X ^ Y)	; X-NOR
1	0	1	0	~ Y	
1	0	1	1	Y → X	
1	1	0	0	~ X	
1	1	0	1	X → Y	; Implication
1	1	1	0	~ (X & Y)	; NAND
1	1	1	1	Constant 1	

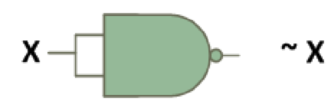


Basic operations: AND(&), OR(¦), NOT(~)

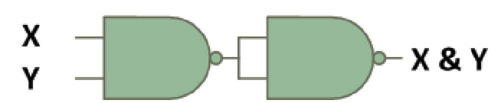
X ^ Y = (X & ~Y) ¦ (~X & Y)

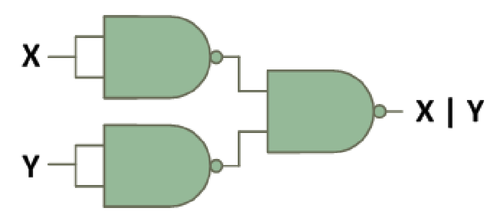
X → Y = ~X ¦ Y

A complete set: NAND = ~ (X & Y)



4 트랜지스터에 이용

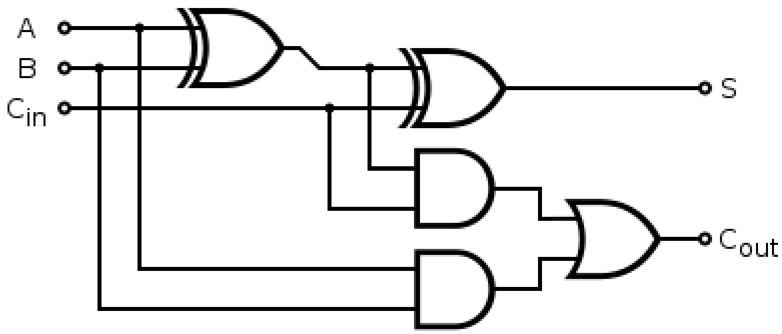


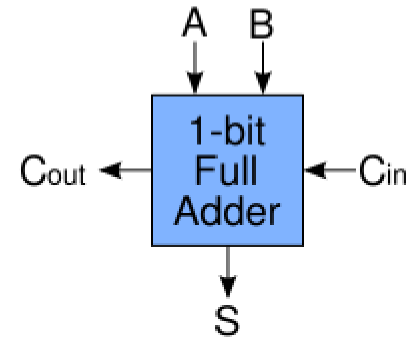


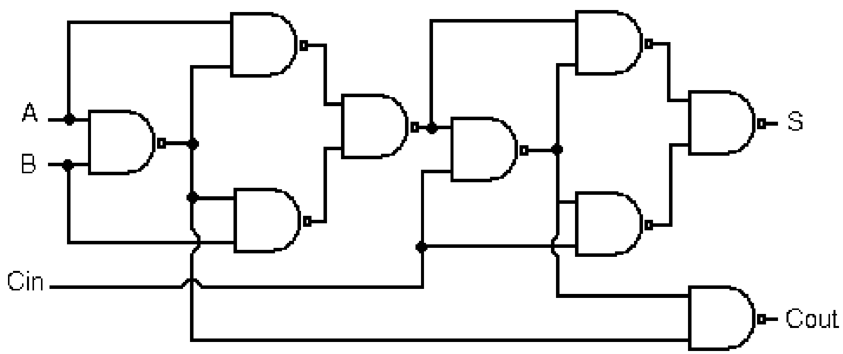
Combinational Logic

Adder

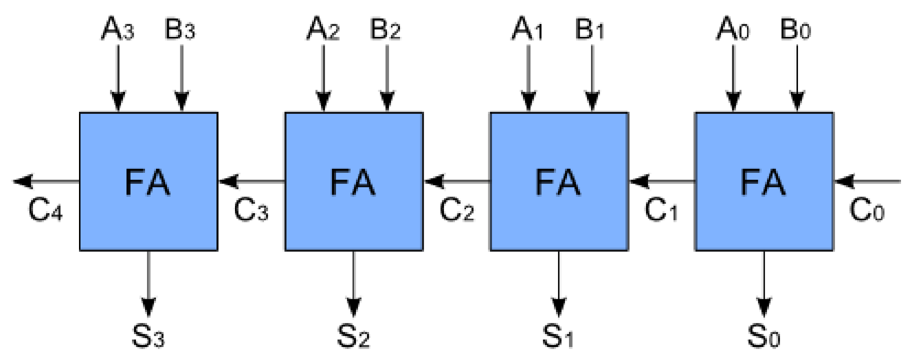
Full Adder







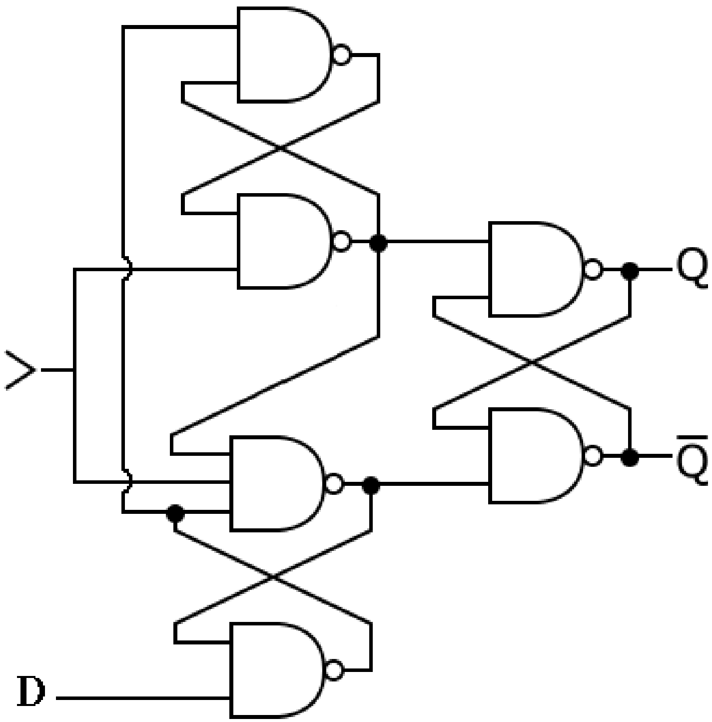
Full Adder (NAND version)



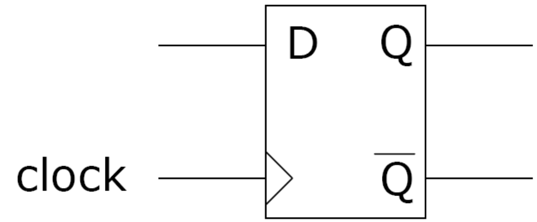
4-bit Ripple Carry Adder

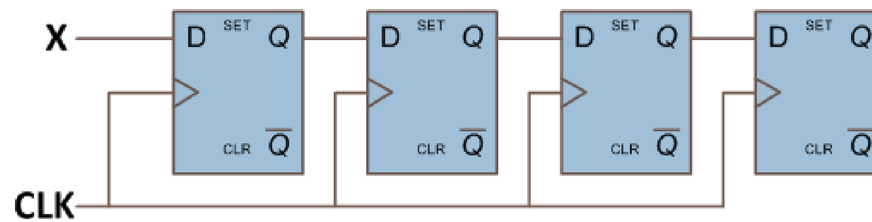
Sequential Logic

Flip-flops

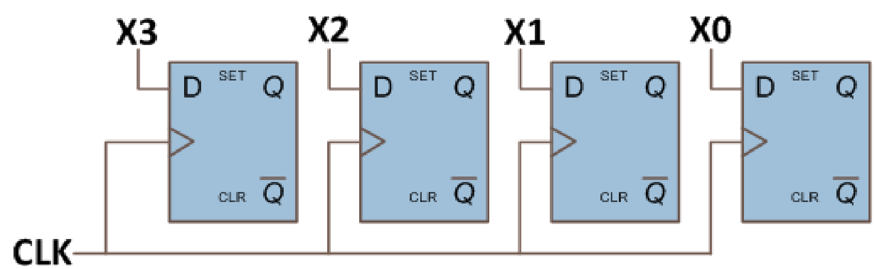


Edge triggered D flip-flop





Shifter



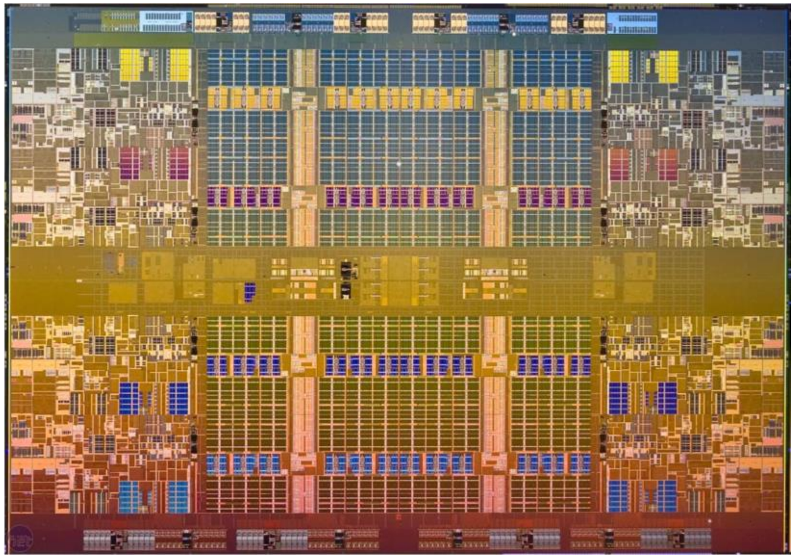
4-bit register

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



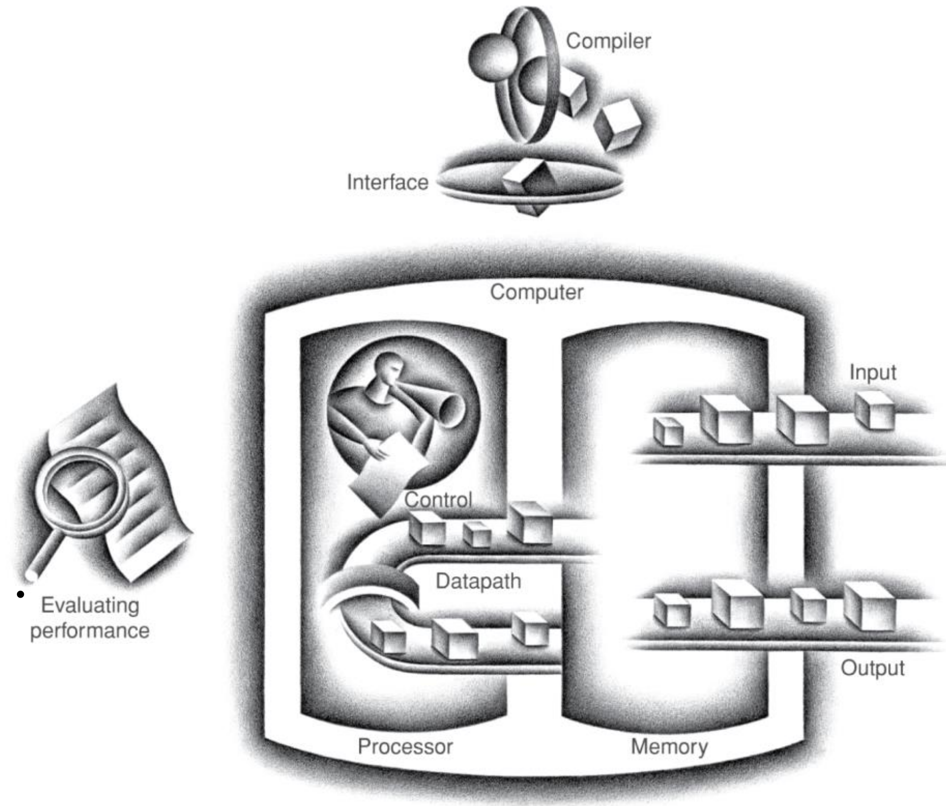
Components of a Computer

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



Understanding Performance

Algorithm

- Determines number of operations executed

Programming language, compiler, architecture

- Determine number of machine instructions executed per operation

Processor and memory system

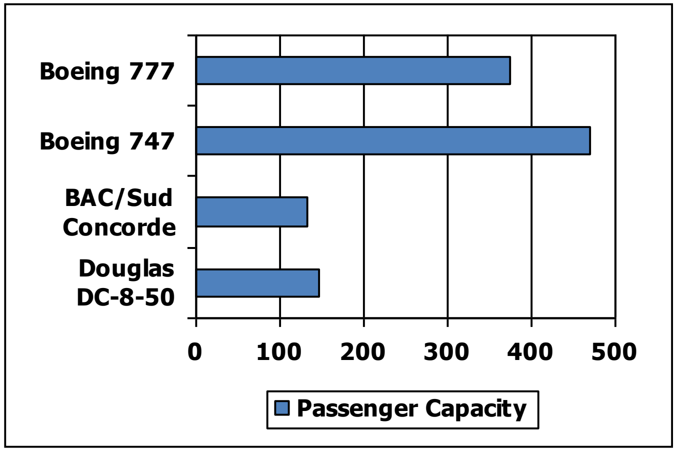
- Determine how fast instructions are executed

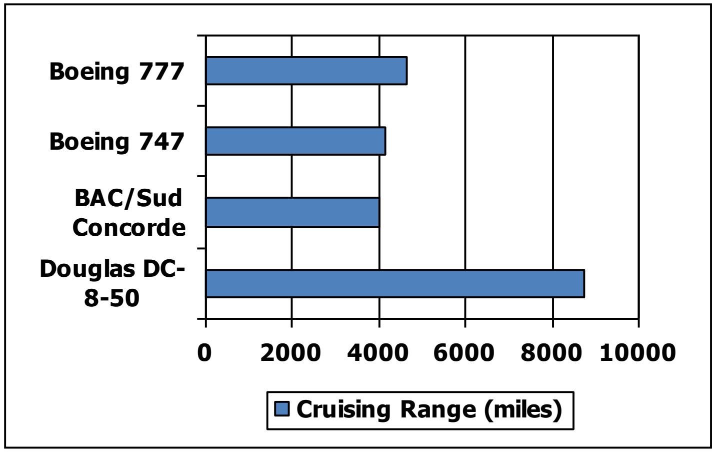
I/O system (including OS)

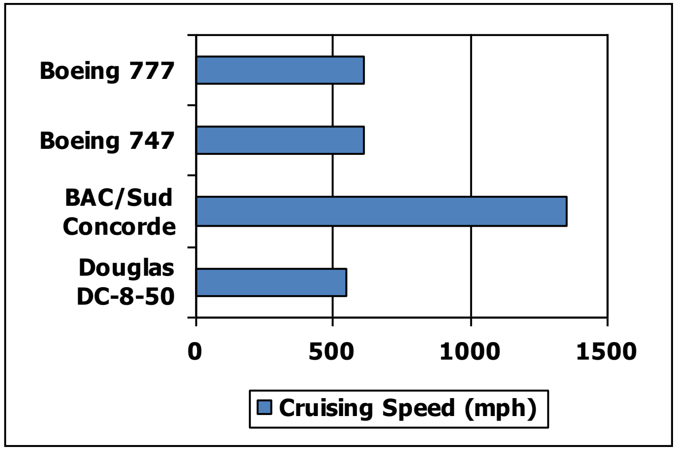
- Determines how fast I/O operations are executed

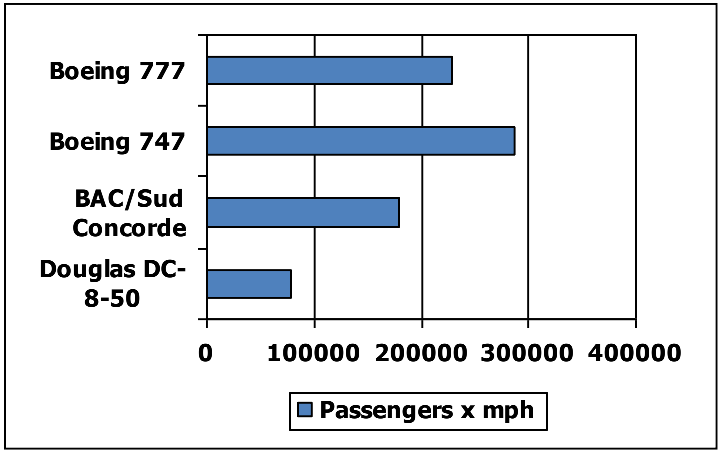
Defining Performance

Which airplane has the best performance?









Response Time and Throughput

$RT \not\propto TP$

↳ 반비례

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?

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

Levels of Program Code

High-level language

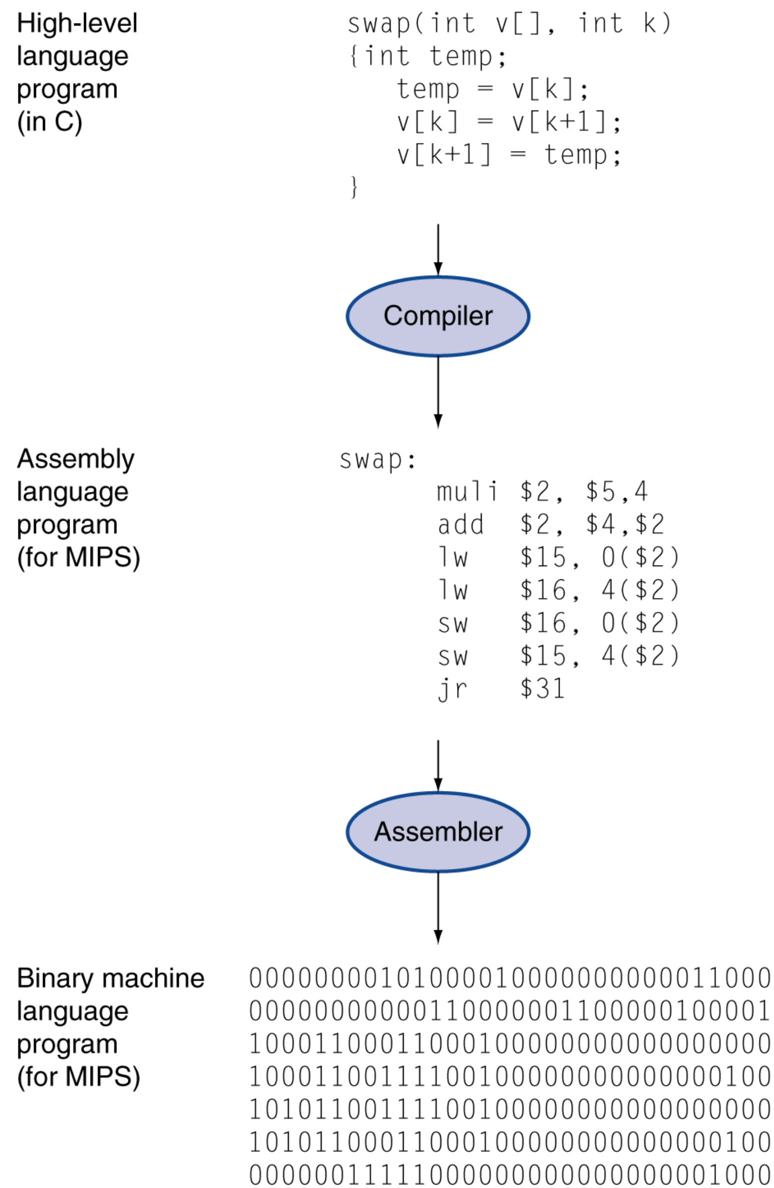
- Level of abstraction closer to problem domain
- Provides for productivity and portability

Assembly language

- Textual representation of instructions

Hardware representation

- Binary digits (bits)
- Encoded instructions and data



MIPS R-format Instructions

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

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

$00000010001100100100000000100000_2 = 02324020_{16}$

Translation and Startup