Data Manipulation

Introduction to Computer

Yu-Ting Wu

(with some slides borrowed from Prof. Tian-Li Yu)

Outline

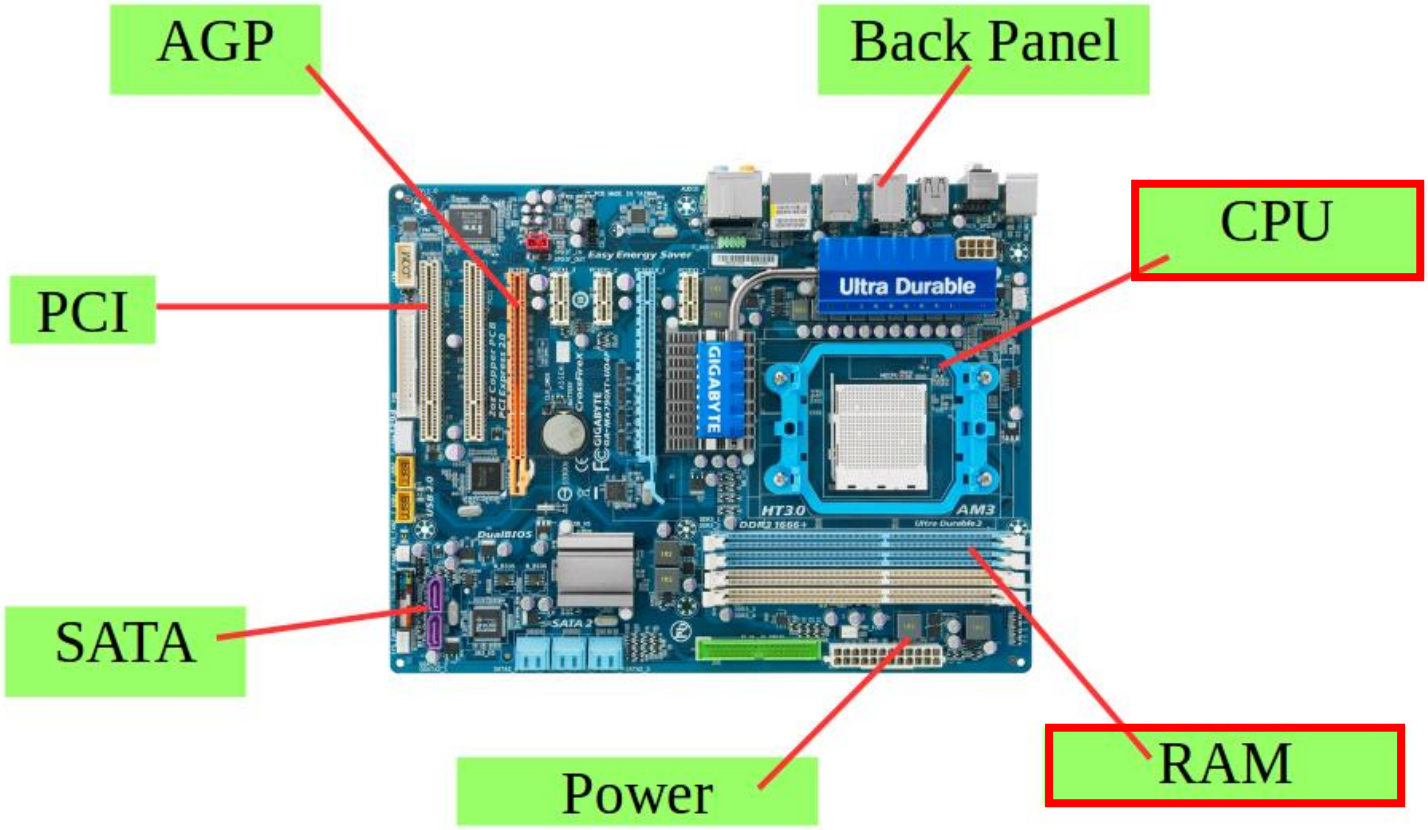
- Computer architecture
- Machine language
- Program execution
- Arithmetic and logic
- Communicating with other devices
- Other architectures

Outline

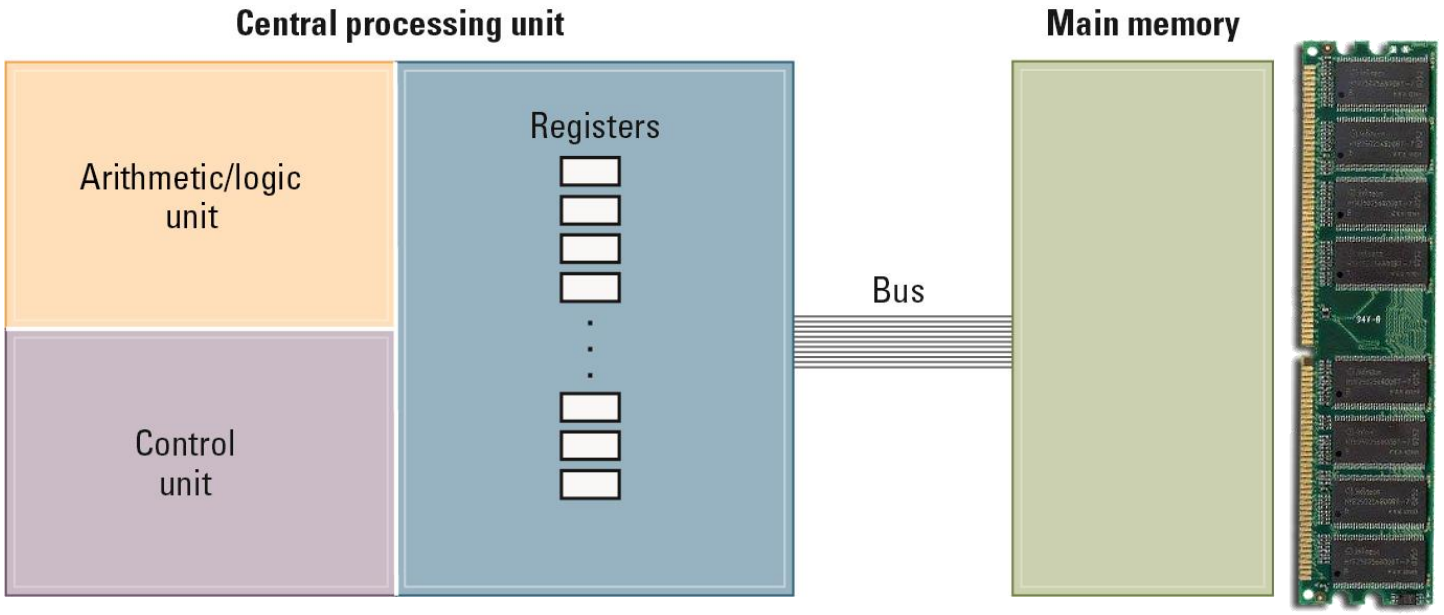
- Computer architecture
- Machine language
- Program execution
- Arithmetic and logic
- Communicating with other devices
- Other architectures

Computer Architecture

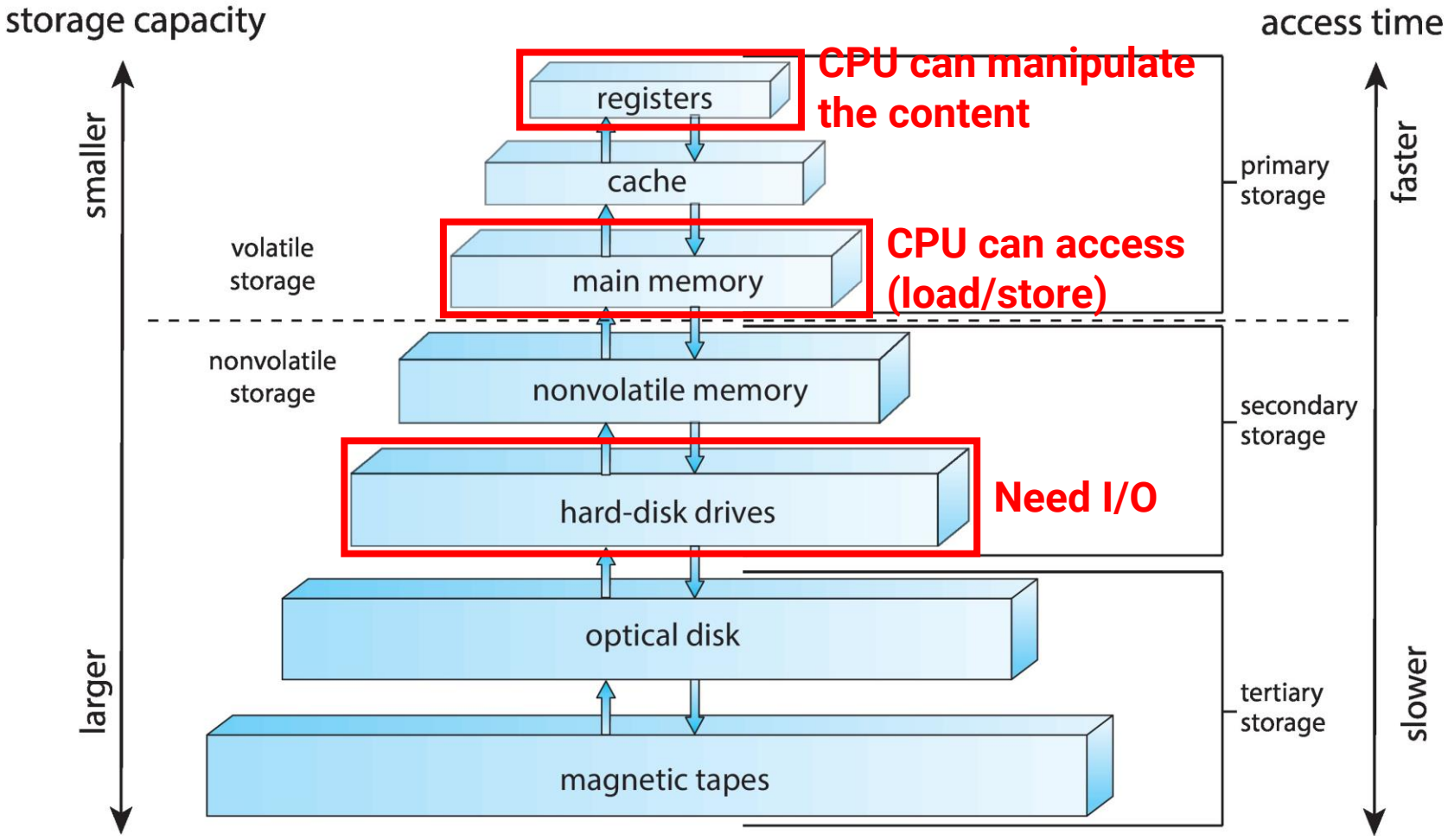
- Motherboard



Computer Architecture (cont.)



Recap: Storage Structure



Recap: Storage Structure (cont.)

Level	1	2	3	4	5
Name	registers	cache	main memory	solid-state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25-0.5	0.5-25	80-250	25,000-50,000	5,000,000
Bandwidth (MB/sec)	20,000-100,000	5,000-10,000	1,000-5,000	500	20-150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Example of Adding Two Values

- Procedure
  - Get one of the values to be added from the memory and place it in a register *R1* — **LOAD**
  - Get the other value to be added from the memory and place it in another register *R2* — **LOAD**

  BUS
  - Activate the addition circuitry with the registers *R1* and *R2* as inputs and another register designated to hold the result — **ADD**
  - Store the result in memory — **STORE**

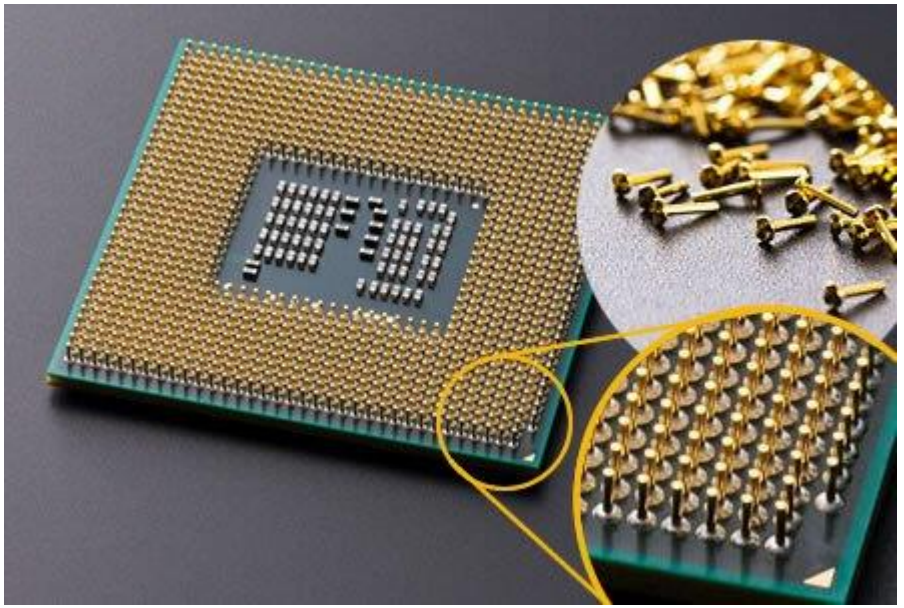
  BUS
  - Stop

Outline

- Computer architecture
- Machine language
- Program execution
- Arithmetic and logic
- Communicating with other devices
- Other architectures

Machine Instructions

- An instruction encoded as a bit pattern recognizable by the CPU
  - **Data transfer**
    - LOAD, STORE, I/O
  - **Arithmetic / Logic**
    - ADD, SUB, etc.
    - AND, OR, SHIFT, etc.
  - **Control**
    - JUMP, HALT



Machine Language Philosophies

- The set of all instructions recognized by a machine

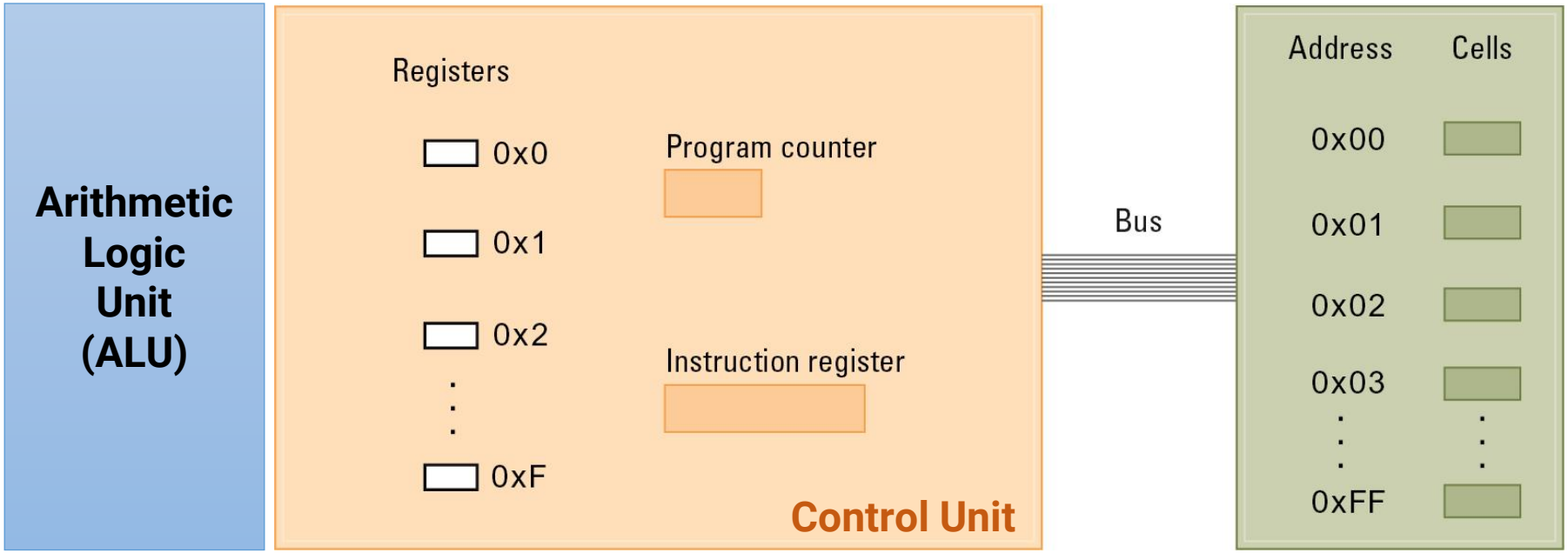
- **Reduced Instruction Set Computing (RISC)**
  - Few, simple, efficient, and fast instructions
  - Examples: PowerPC, SPARC

- **Complex Instruction Set Computing (CISC)**
  - Many, convenient, and powerful instructions
  - Examples: Intel x86 and x86-64

Outline

- Computer architecture
- Machine language
- Program execution
- Arithmetic and logic
- Communicating with other devices
- Other architectures

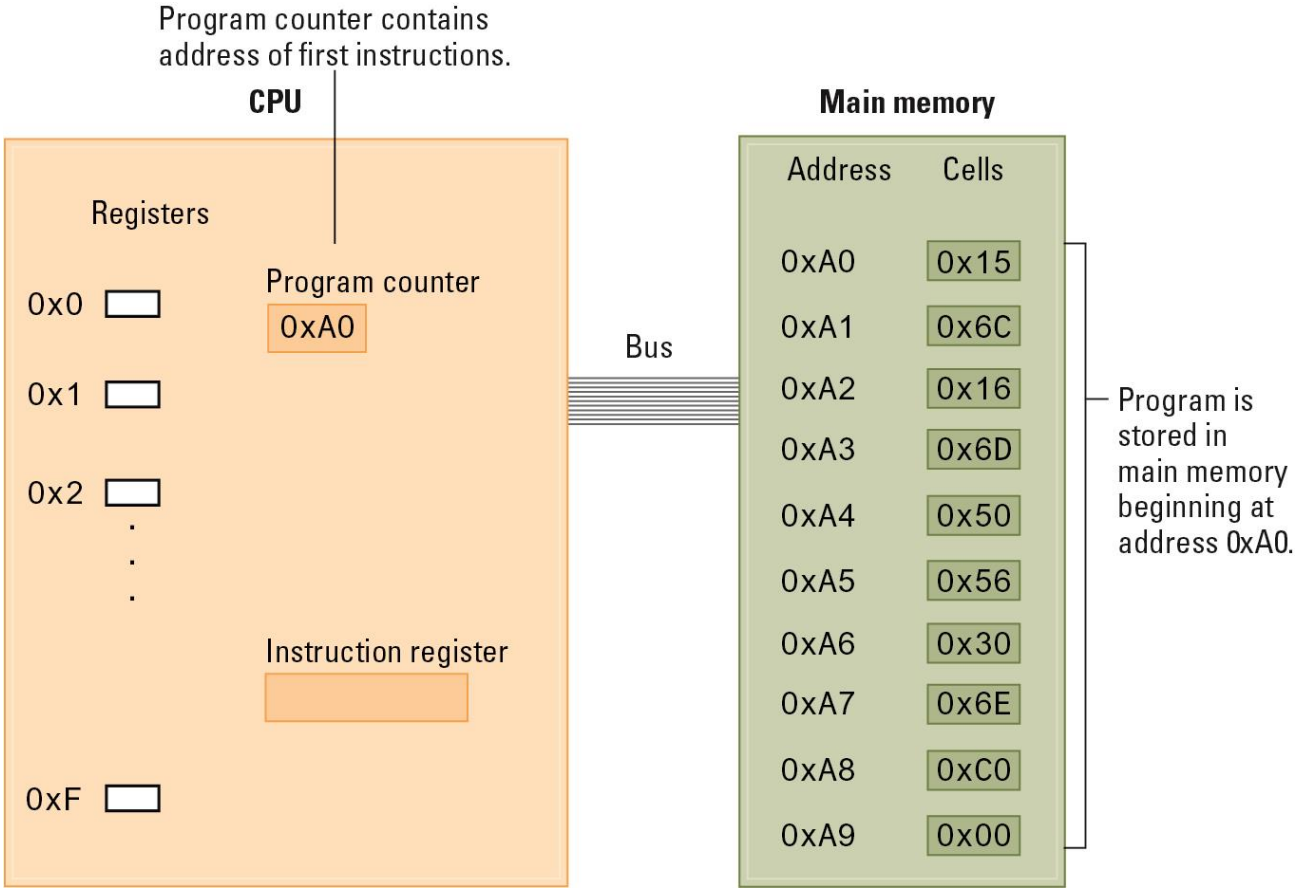
Program Execution



- Special registers
  - **Program counter:** hold the address of the next instruction
  - **Instruction register:** hold the content of the current instruction

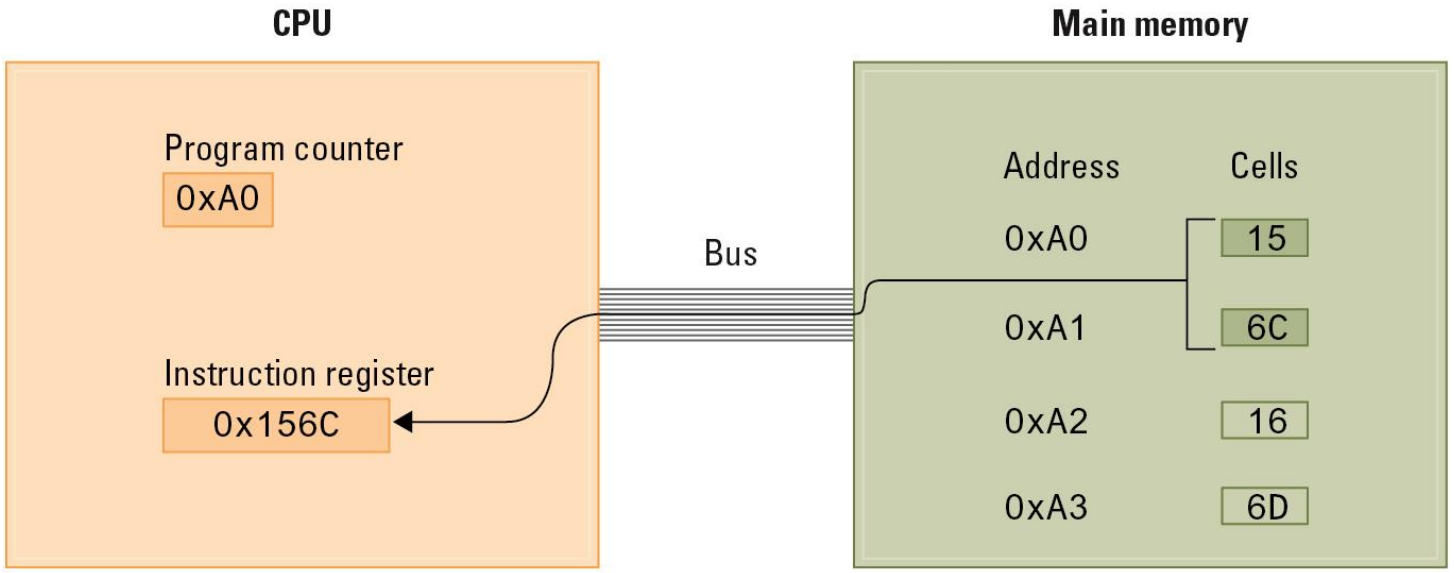
Program Execution (cont.)

- How Program Counter and Instruction Register works



Program Execution (cont.)

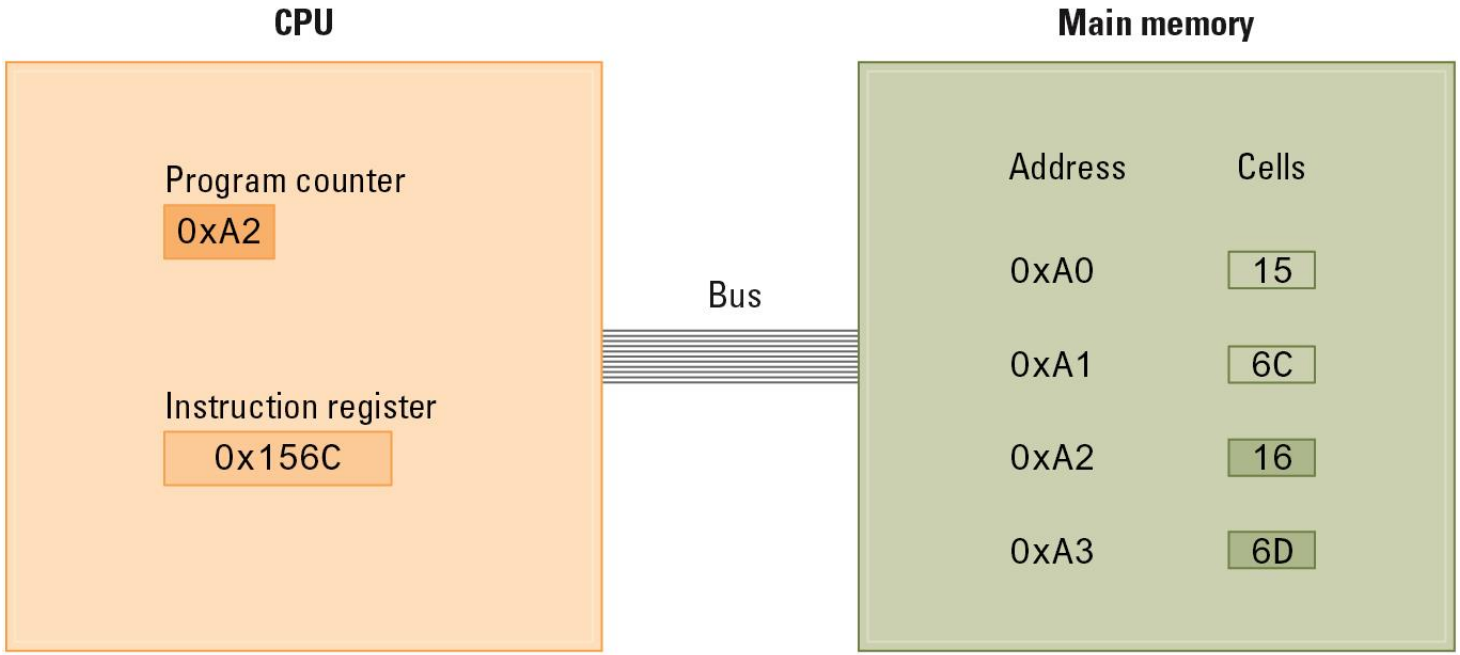
- How Program Counter and Instruction Register works
  - At the beginning of the fetch step, the instruction starting at addresses **0xA0** is retrieved from the memory and placed in the **Instruction Register**



Assume each instruction is two-byte long

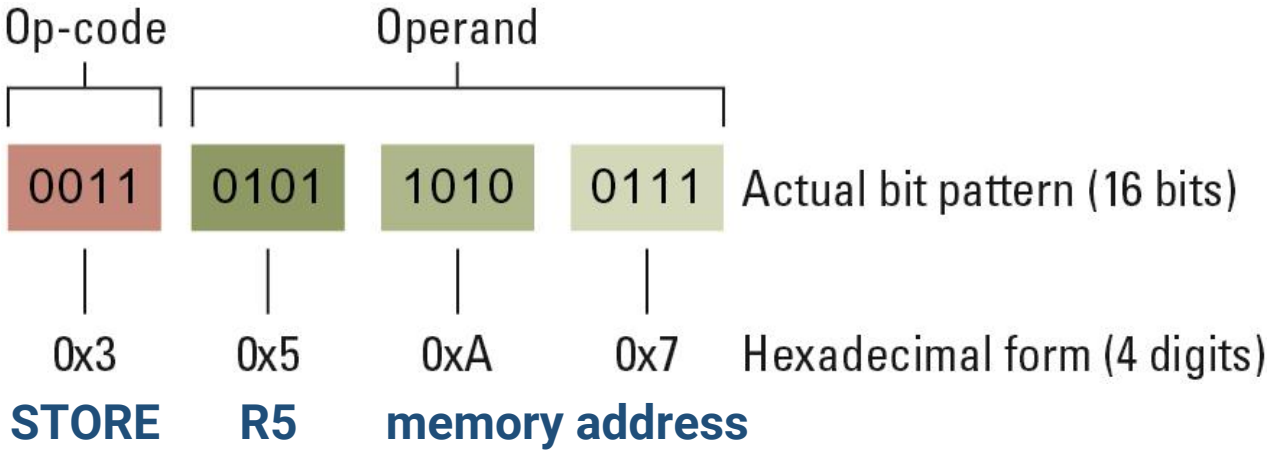
Program Execution (cont.)

- How Program Counter and Instruction Register works
  - Then the **Program Counter is incremented** so that it points to the next instruction



Example of Machine Instructions

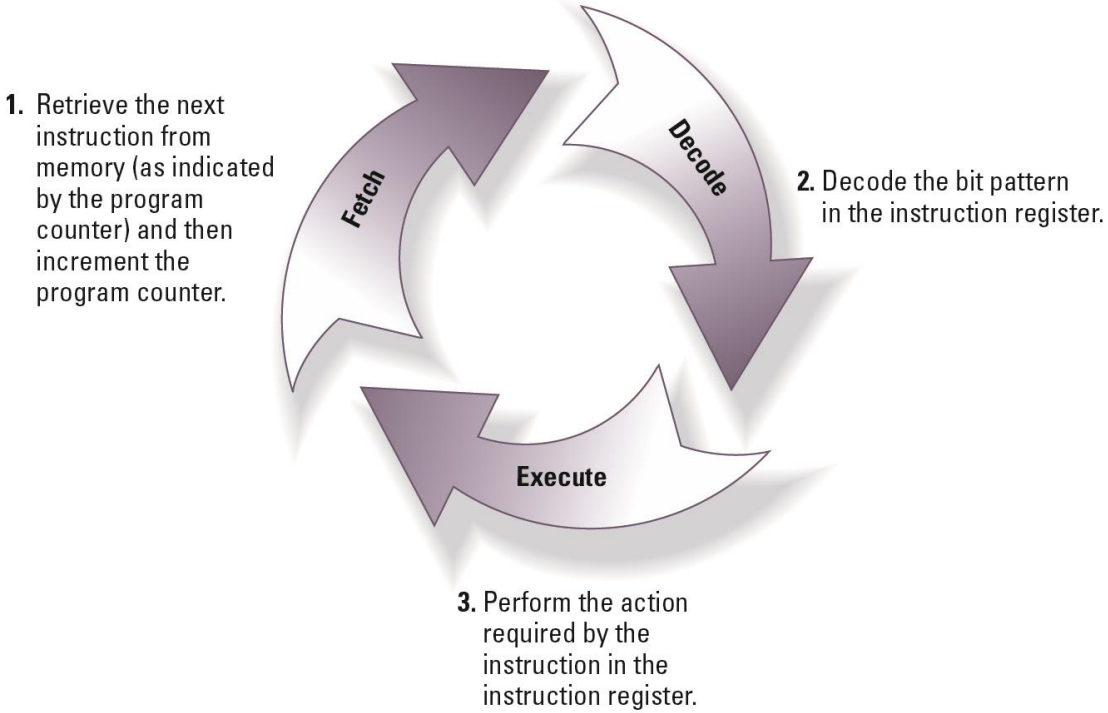
- **Op-code:** specifies which operation to execute
- **Operand:** gives more detailed information about the operation
  - Interpretation of operand varies depending on op-code



Largest memory reference in this example: 2^8 = 256 cells (bytes)

Program Execution Overview

- **Machine cycle (repeat these 3 steps)**

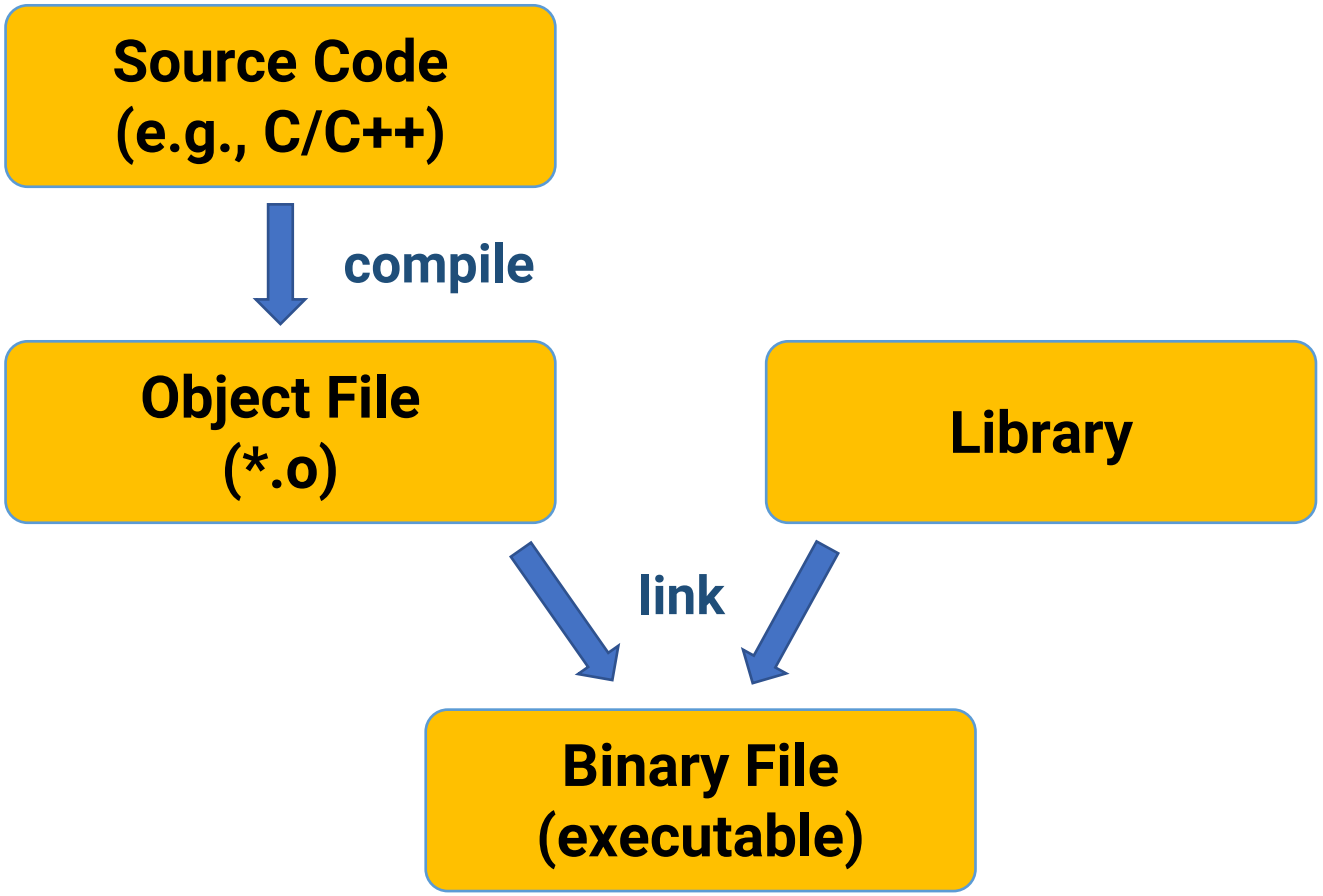


- **Clock: how many machine cycles can be done in 1 sec.**
  - **E.g., 3.0 GHz (3 x 10^9 cycles in 1 sec.)**

Revisit the Example of Adding Two Values

Encoded instructions	Translation	Possible Assembly	Possible C
0x156C	Load register 0x5 with the bit pattern found in the memory cell at address 0x6C.	LOAD 5, 6C	
0x166D	Load register 0x6 with the bit pattern found in the memory cell at address 0x6D.	LOAD 6, 6D	
0x5056	Add the contents of register 0x5 and 0x6 as though they were two's complement representation and leave the result in register 0x0.	ADD 0, 5, 6	c = a + b;
0x306E	Store the contents of register 0x0 in the memory cell at address 0x6E.	STORE 0, 6E	
0xC000	Halt.	HALT	

Compiling and Linking



Outline

- Computer architecture
- Machine language
- Program execution
- Arithmetic and logic
- Communicating with other devices
- Other architectures

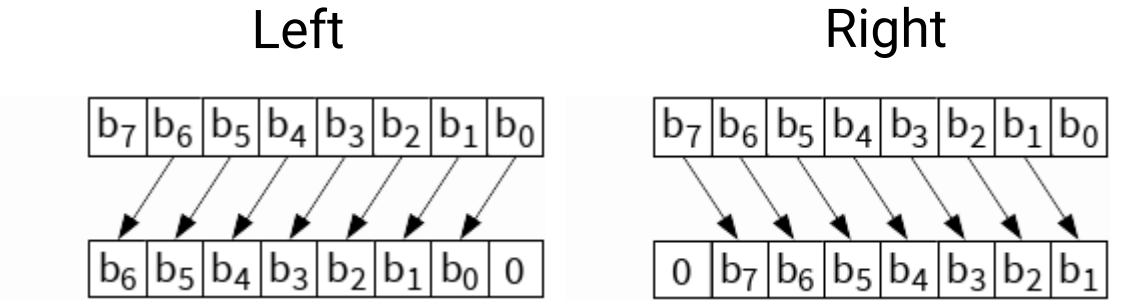
Arithmetic / Logic Unit (ALU)

- **Arithmetic operations**
- **Logic operations**
  - Masking

AND	OR	XOR
01010101	01010101	01010101
00001111	00001111	00001111
00000101	01011111	01011010
Setting the first 4 bits to 0	Setting the latter 4 bits to 1	Inverting the latter 4 bits

Arithmetic / Logic Unit (ALU) (cont.)

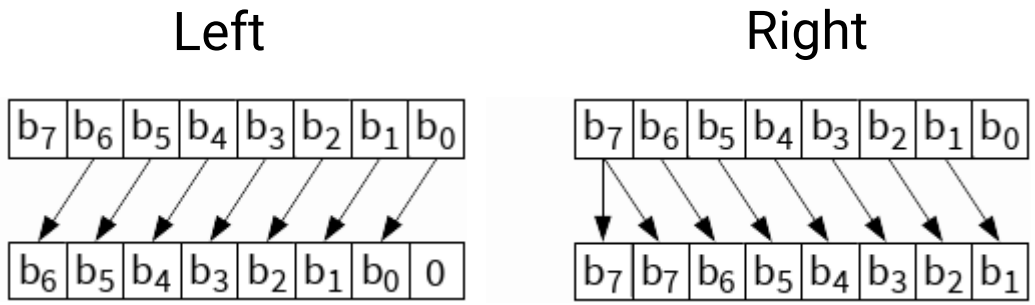
- **Arithmetic operations**
- **Logic operations**
  - Logic shift



	00101011	43		00101011	43
➡	01010110	86	➡	00010101	21
	11110101	-11		11110101	-11
➡	11101010	-22	➡	01111010	122

Arithmetic / Logic Unit (ALU) (cont.)

- **Arithmetic operations**
- **Logic operations**
  - Arithmetic shift

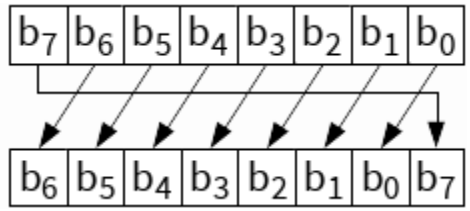


	00101011	43		00101011	43
→	01010110	86	→	00010101	21
	11110101	-11		11110101	-11
→	11101010	-22	→	11111010	-6

Arithmetic / Logic Unit (ALU) (cont.)

- **Arithmetic operations**
- **Logic operations**
  - Rotation (circular shift)

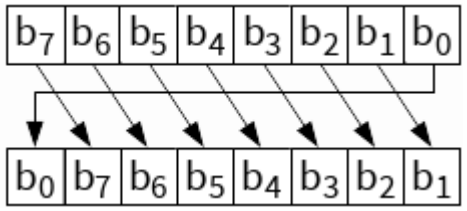
Left



10100000

01010111

Right



10100000

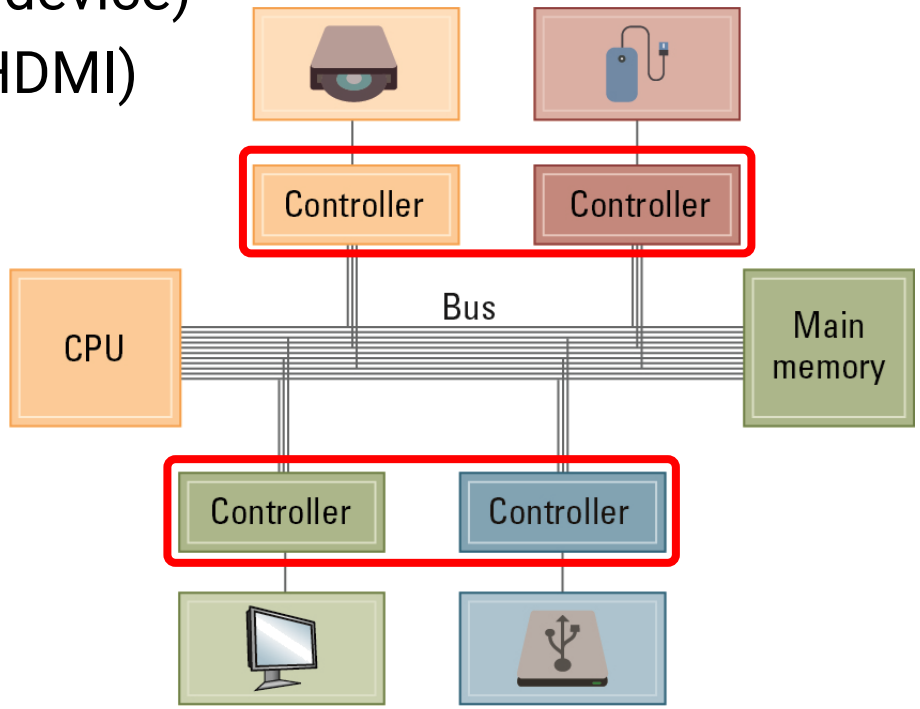
01010000

Outline

- Computer architecture
- Machine language
- Program execution
- Arithmetic and logic
- Communicating with other devices
- Other architectures

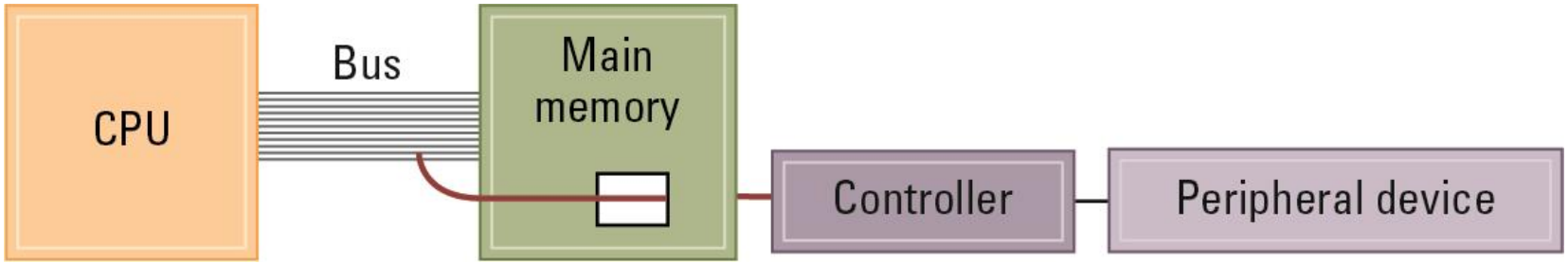
Communicating with Other Devices

- **Controller**
  - Handle communication between the computer and other devices
  - Specialized (by types of device)
  - General purpose (USB, HDMI)



Communicating with Other Devices (cont.)

- **Port**
  - The point at which a device connects to a computer
- **Memory-mapped I/O**
  - Devices appear to the CPU as though they were memory locations
  - I/O as LOAD, STORE



Communicating with Other Devices (cont.)

- **Direct memory access (DMA)**
  - Once authorized, controllers can access data directly from the main memory without notifying the CPU
  - Main memory access by a controller over the bus
- **Handshaking**
  - 2-way communication
  - The process of coordinating the transfer of data between the computer and the peripheral device
- **Communication media**
  - **Parallel:** several signals transferred at the same time, each on a separate “line” (computer’s internal bus)
  - **Serial:** signals are transferred one after the other over a single “line” (U**S**B, FireWire)

Data Communication Rates

- **Measurement unit**
  - bps: **bits** per second
  - Kbps: Kilo-bps (1,000 bps)
  - Mbps: Mega-bps (1,000,000 bps)
  - Gbps: Giga-bps (1,000,000,000 bps)

- **Bandwidth: maximum available rate**

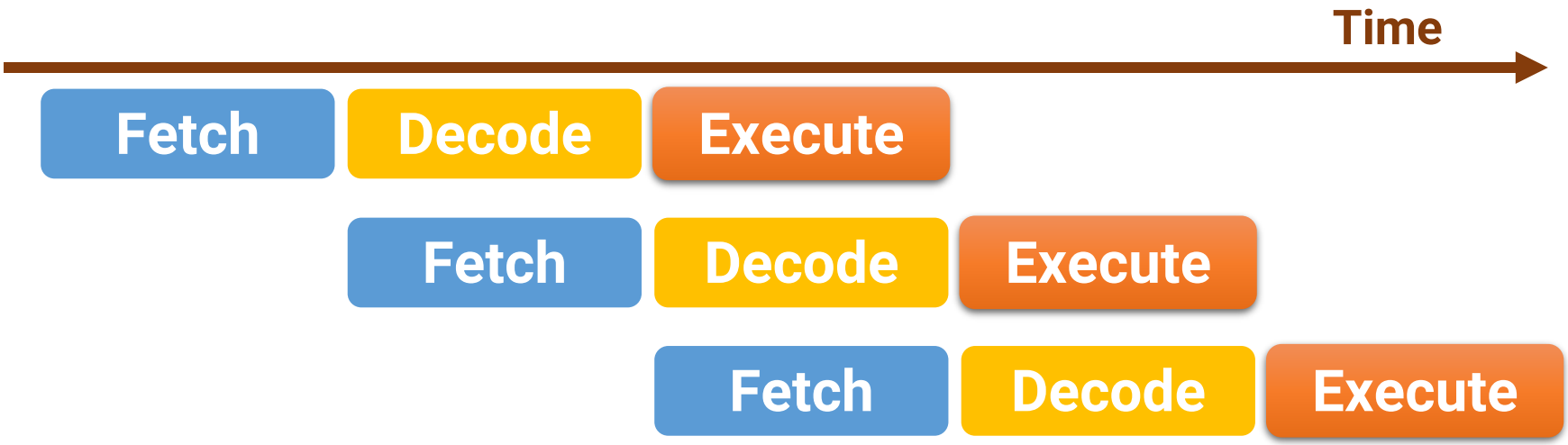
Outline

- Computer architecture
- Machine language
- Program execution
- Arithmetic and logic
- Communicating with other devices
- Other architectures

Pipelining

Pipelining (cont.)

- Goal: increase throughput
- Pre-fetching
  - Issue: conditional jump



Parallel / Distributed Computing

- **Parallel processing**
  - Use multiple processors simultaneously
  - **SISD:** Single Instruction, Single Data
    - No parallel processing
  - **SIMD:** Single Instruction, Multiple Data
    - Same program, different data
  - **MIMD:** Multiple Instruction, Multiple Data
    - Different programs, different data
- **Distributed computing**
  - Linking several computers via a network
  - Separate processors, separate memory

Parallel / Distributed Computing (cont.)

- **Issues of parallel / distributed computing**
  - Data dependency
  - Load balancing
  - Synchronization
  - Reliability

Parallel / Distributed Computing (cont.)

- Example of parallel programming (from Prof. Yu's slides)

```
declare A[0]~A[99]
input A[0]

for (i = 1; i<100; i++)
    A[i] = A[i-1] * 2;
```

2 CPUs

```
A[1]=A[0] * 2;
for (i = 2; i<100; i+=2) {
   A[i] = A[i-2] * 4;      ← CPU 0
   A[i+1] = A[i-1] * 4;    ← CPU 1
}
```

Parallel / Distributed Computing (cont.)

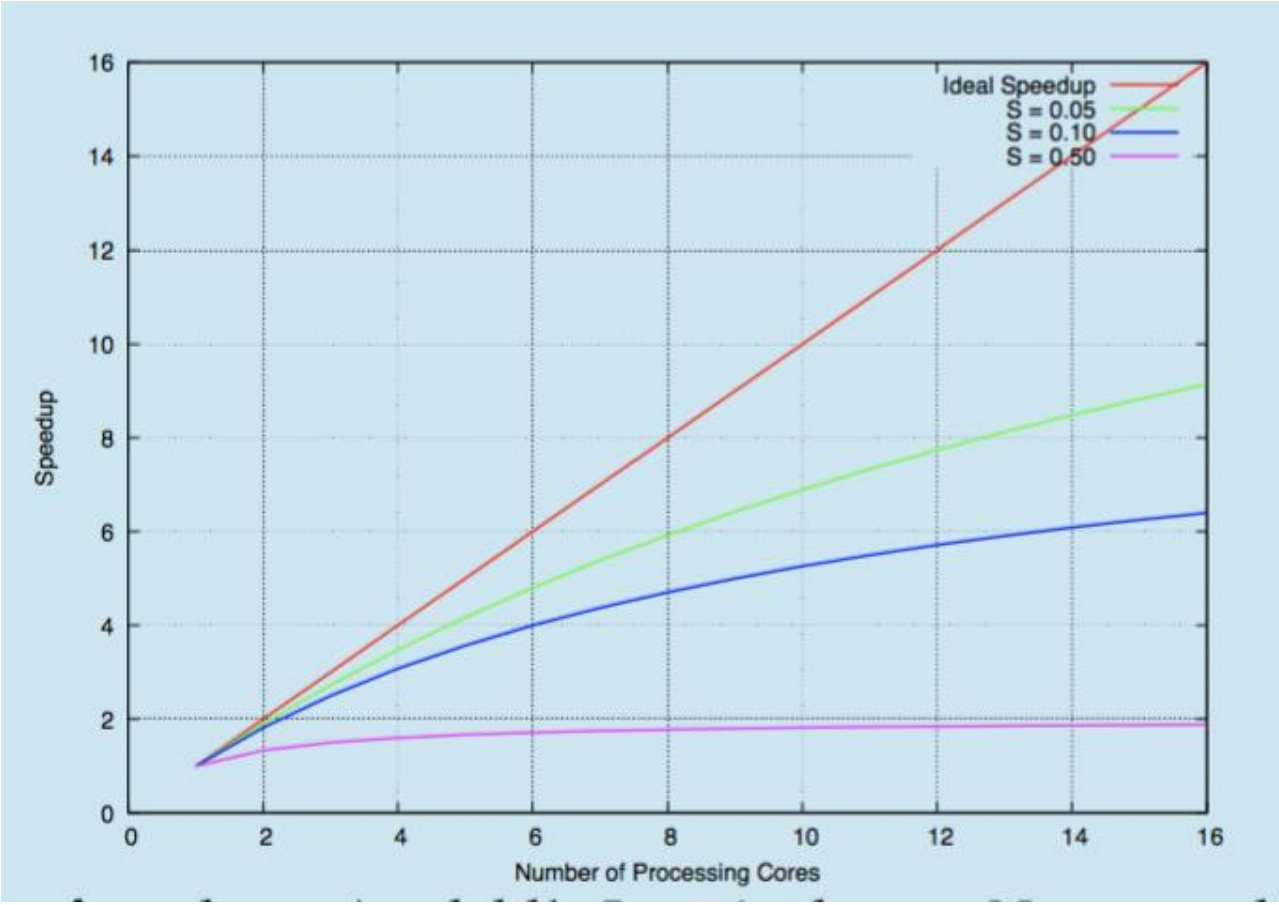
- Speedup of parallel computing (**Amdahl's law**)
  - Assume ***S*** is the serial portion and the system has ***N*** processing cores

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

  - Example:
    - 75% parallel / 25% serial, moving from 1 to 2 cores results in a speedup of 1.6 times
  - As *N* approaches infinity, speedup approaches *1/S*

Parallel / Distributed Computing (cont.)

- Speedup of parallel computing (Amdahl's law)



Any Questions?