Introduction to Computer Science
Lecture 2: Data Manipulation

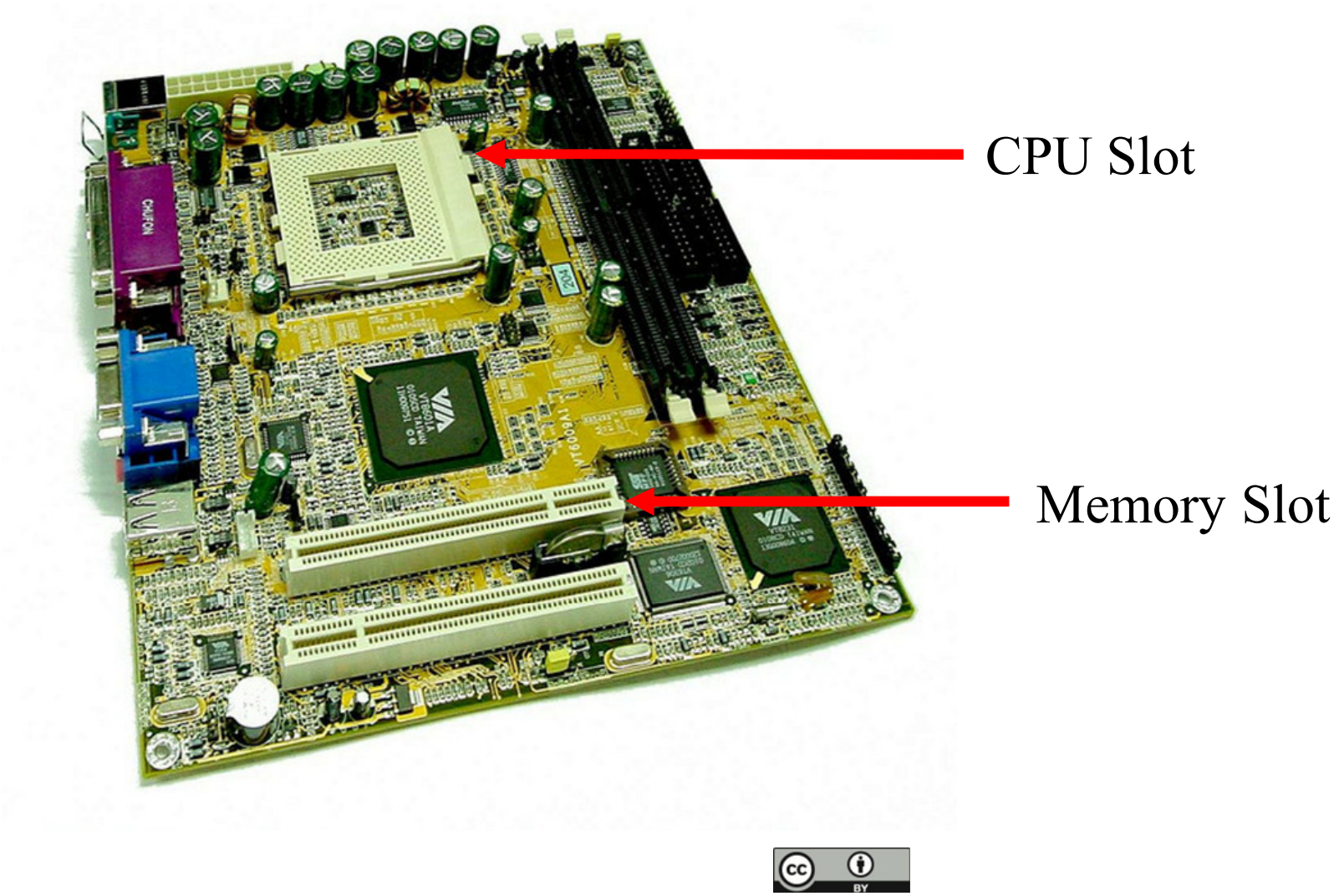
Tian-Li Yu

Taiwan Evolutionary Intelligence Laboratory (TEIL)
Department of Electrical Engineering
National Taiwan University

tianliyu@cc.ee.ntu.edu.tw

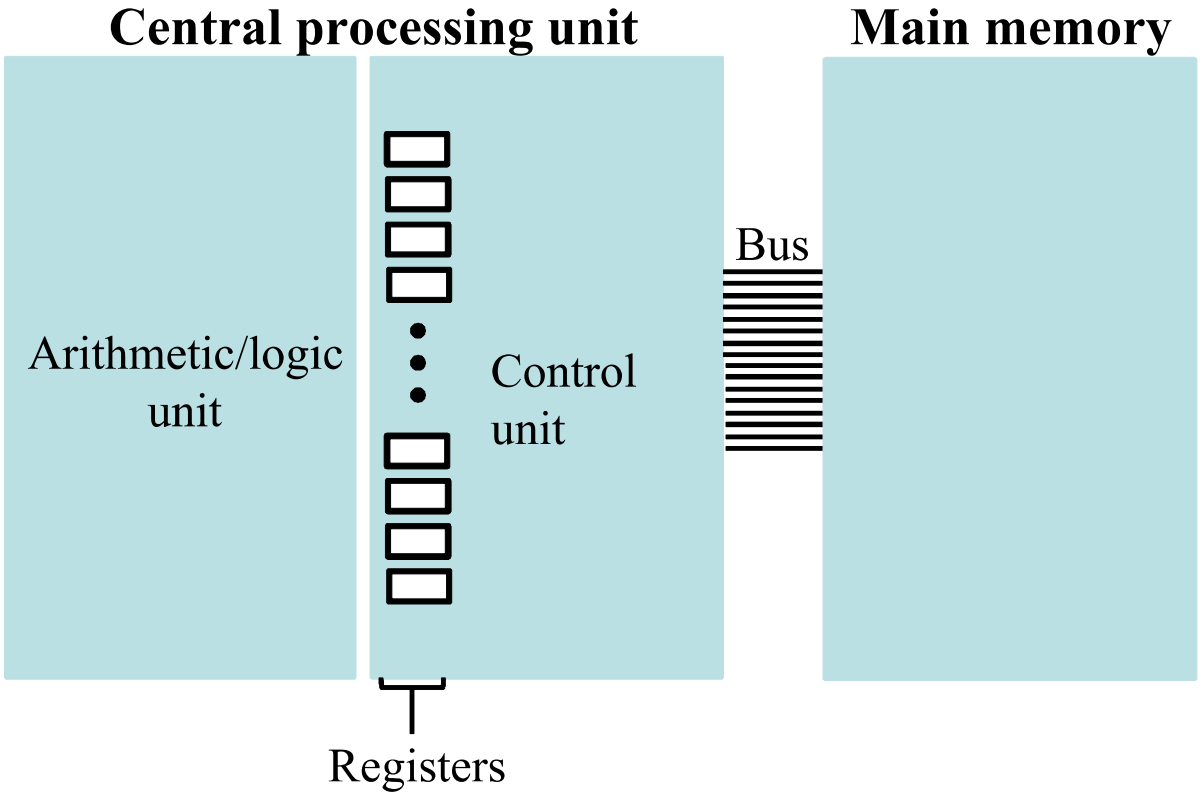
Slides made by Tian-Li Yu, Chu-Yu Hsu, and Jay-Wie Wu

Motherboard



Computer Architecture

- CPU
  (central processing unit)
- Registers
- Memory
- Bus
- Motherboard



Adding Two Values Stored in Memory

1. Get one of the values to be added from memory and place it in a register. register 暫存器
2. Get the other value to be added from memory and place it in another register.
3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.
4. Store the result in memory.
5. Stop.

Machine Instructions

- Data transfer
  - LOAD, STORE, I/O
- Arithmetic/logic
  - AND, OR, ADD, SUB, etc.
  - SHIFT, ROTATE
- Control
  - JUMP, HALT

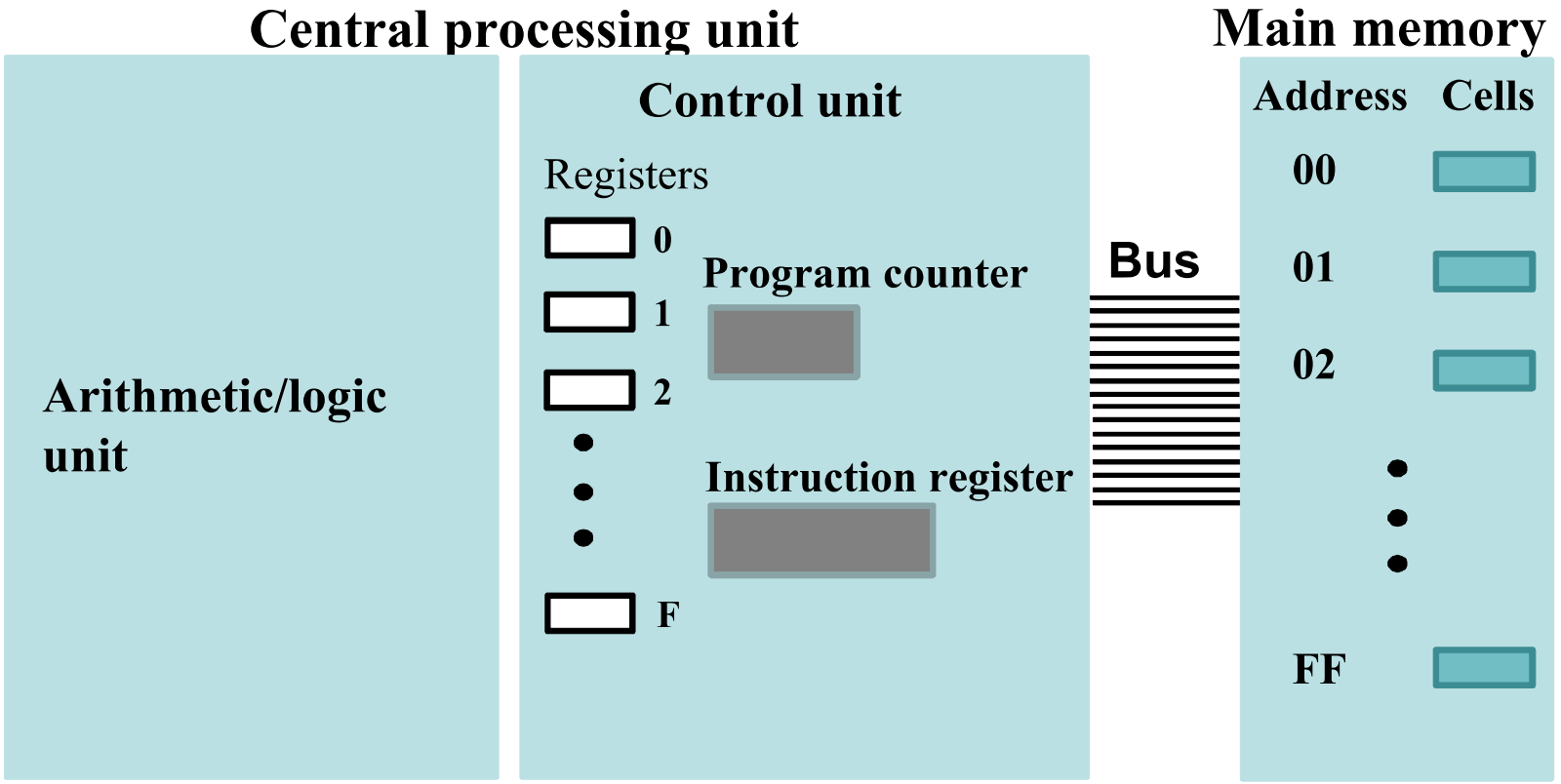
指令長度相同、簡單、便宜 e.g. 8051

- RISC (reduced instruction set computing) (PRC, SPARC) vs. CISC (complex instruction set computing) (x86, x86-64)

指令複雜、長度不一
多個cycle

Intel 都是CISC
較複雜

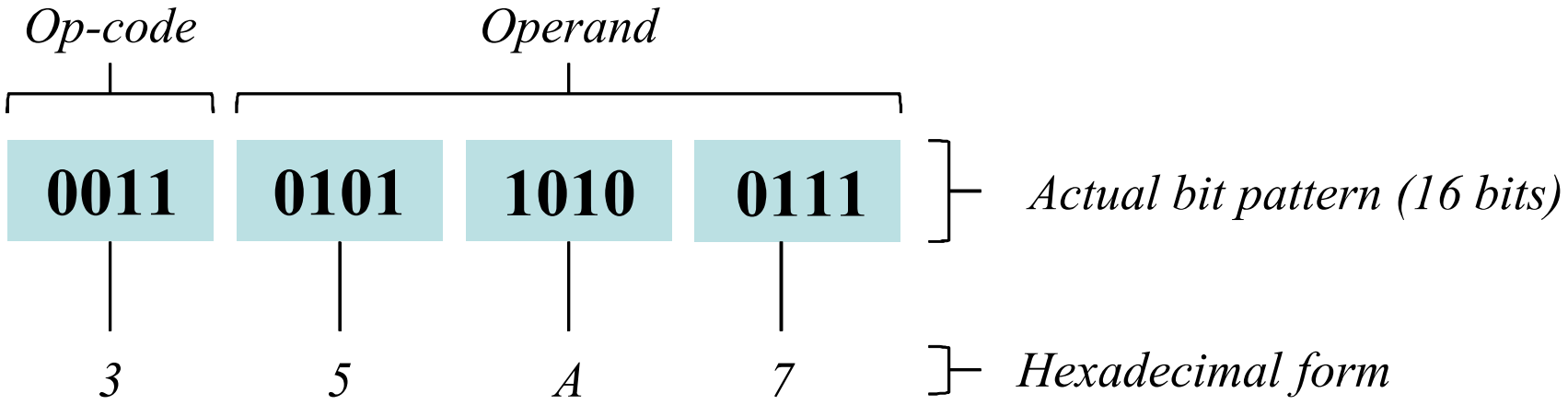
Architecture of a Simple Machine



PC:紀錄程式執行到的位置（Pointer）
IR:紀錄要抓什麼、一次抓兩個byte

CPU有兩種：
General Purpose Registers
Special Purpose Registers

Example of Machine Instructions



Store (3) the content of register No. 5 to the memory cell addressed A7

Memory reference: $2^8 = 256$ cells (bytes)

指令集instruction site
此圖為16bits 有8bits(A7)是可定值
e.g. 32bit 64bit (可定值addressable) 的電腦
如果只有32bit 2^32 bytes = 2^2 * 2^30 bytes = 4 GB
插8GB記憶體
只能用4GB

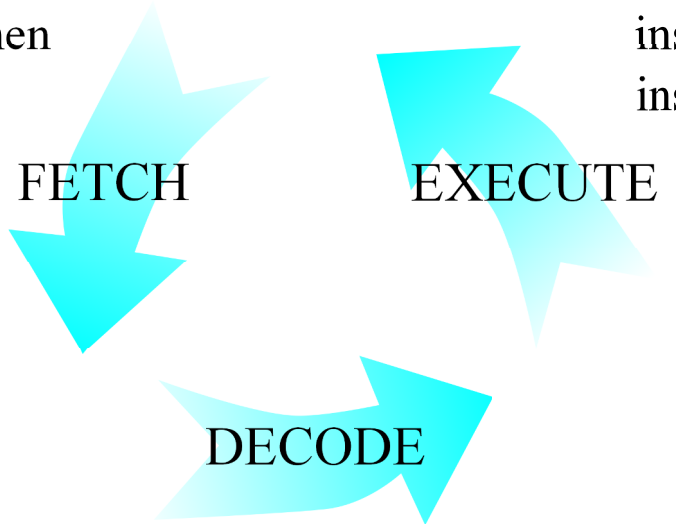
Adding Two Values (Revisited)

Instructions	Translation	Possible Assembly	Possible C
156C	Load register 5 with the bit pattern found in the memory cell at address 6C.	LOAD 5, 6C	
166D	Load register 6 with the bit pattern found in the memory cell at address 6D.	LOAD 6, 6D	
5056	Add the contents of register 5 and 6 as two's complement representation and leave the result in register 0.	ADD 0, 5, 6	c = a + b;
306E	Store the contents of register 0 in the memory cell at address 6E.	STORE 0, 6E	
C000	Halt.	HALT	

Program Execution

- Instruction register (IR), program counter (PC)
- Machine cycle
  - clock
  - benchmarking

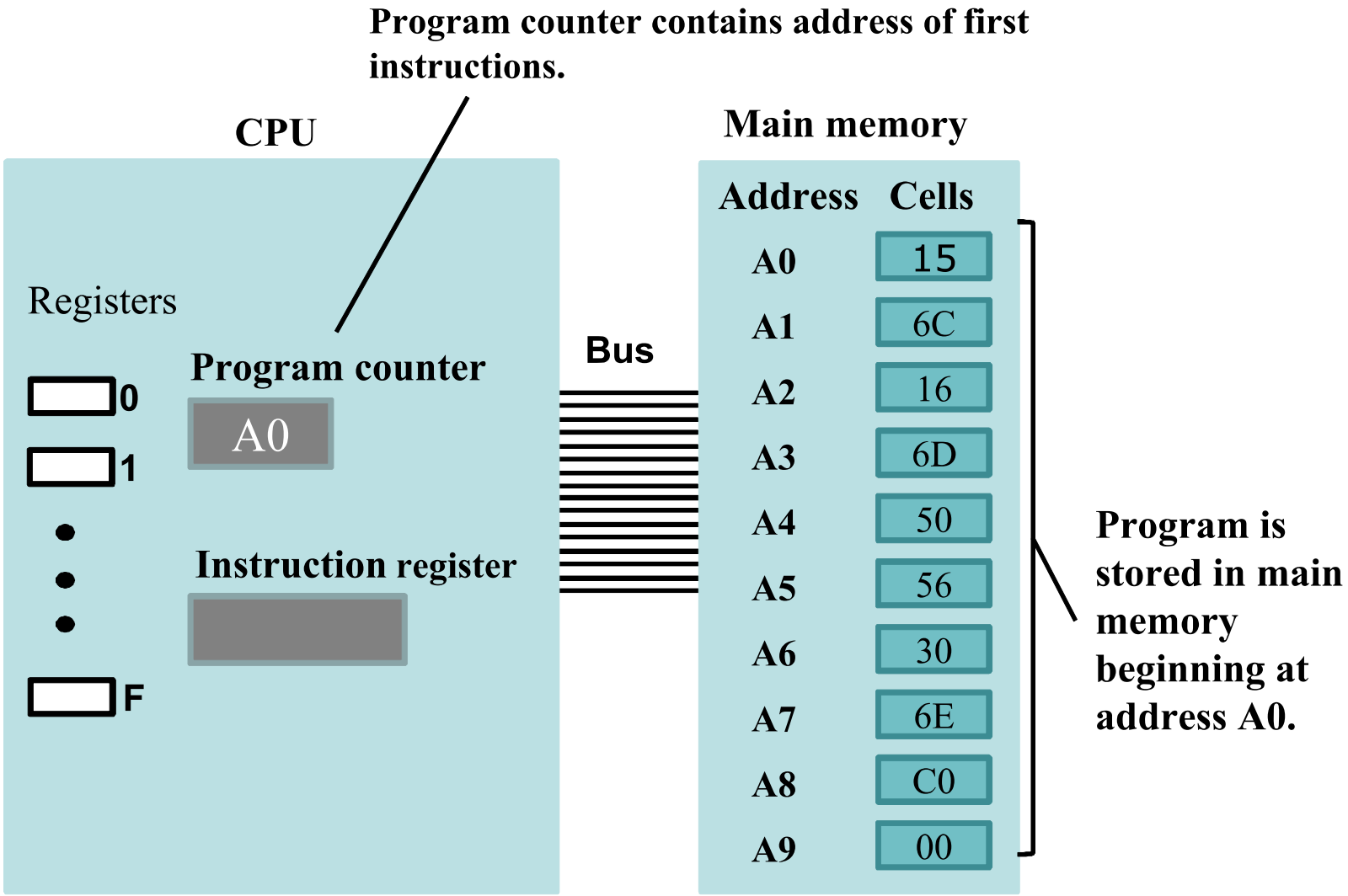
1. Retrieve the next instruction from memory (as indicated by the program counter) and then increment the program counter.



3. Perform the action required by the instruction in the instruction register.

2. Decode the bit pattern in the instruction register.

Fetch


Program counter contains address of first instructions.
CPU
Main memory
Registers
Program counter
A0
Bus
0
1
F
Instruction register
Address
Cells
A0
15
A1
6C
A2
16
A3
6D
A4
50
A5
56
A6
30
A7
6E
A8
C0
A9
00
Program is stored in main memory beginning at address A0.

Arithmetic and Logic Unit (ALU)

- Arithmetic operations
- Logic/bit operations
  - Masking

AND

```
  01010101
& 00001111
----------
  00000101
```

Setting the first 4 bits to 0.

OR

```
  01010101
| 00001111
----------
  01011111
```

Setting the latter 4 bits to 1.

XOR

```
  01010101
^ 00001111
----------
  01011010
```

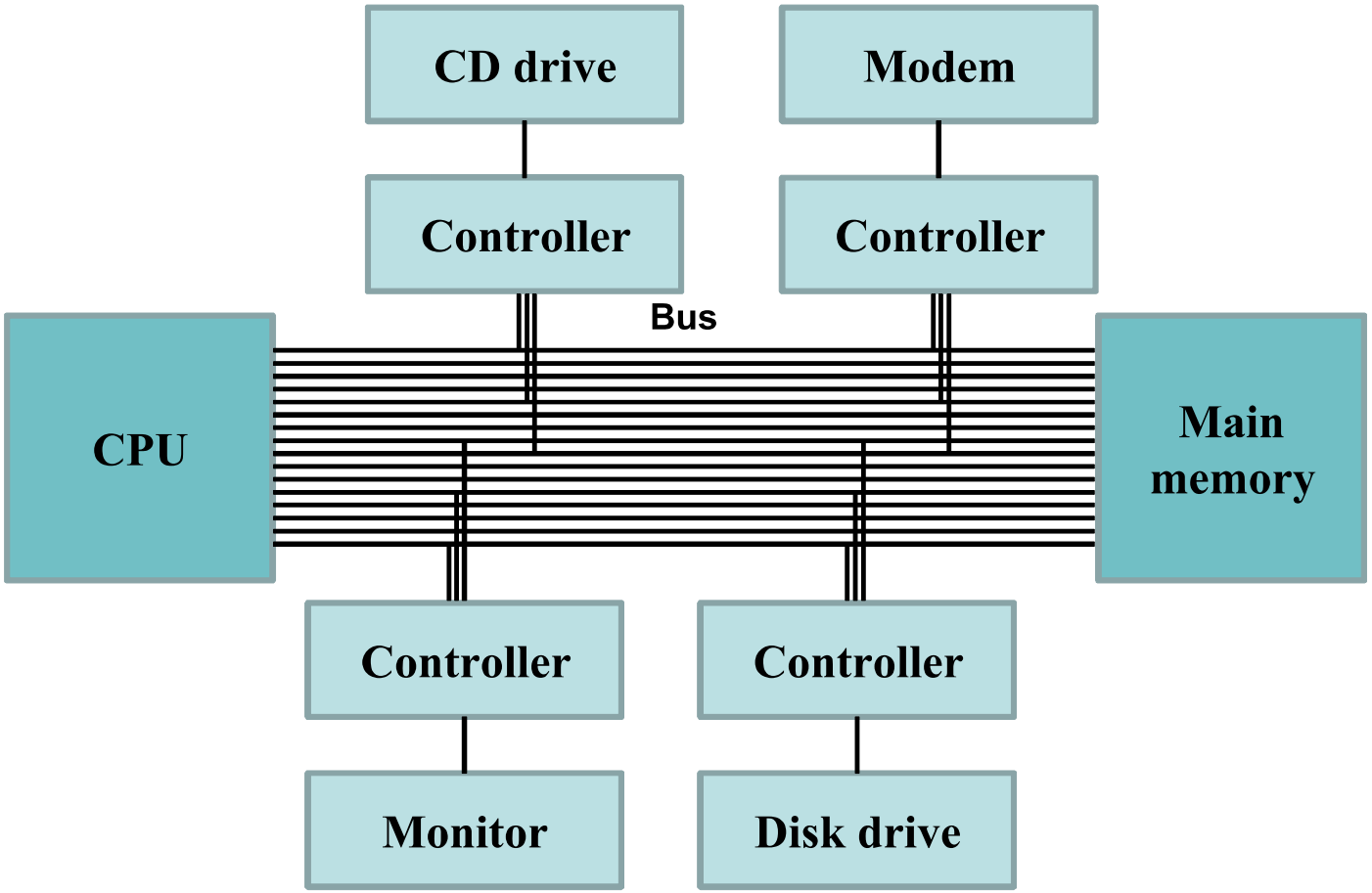
Inverting the latter 4 bits.

Shift/Rotation

- Logic shift
  10110000 $\rightarrow$ 01011000 (right)
  $\rightarrow$ 01100000 (left)
- Arithmetic shift
  10110000 $\rightarrow$ 11011000 (right)
  $\rightarrow$ 11100000 (left)
- Rotation
  10110000 $\rightarrow$ 01011000 (right)
  $\rightarrow$ 01100001 (left)

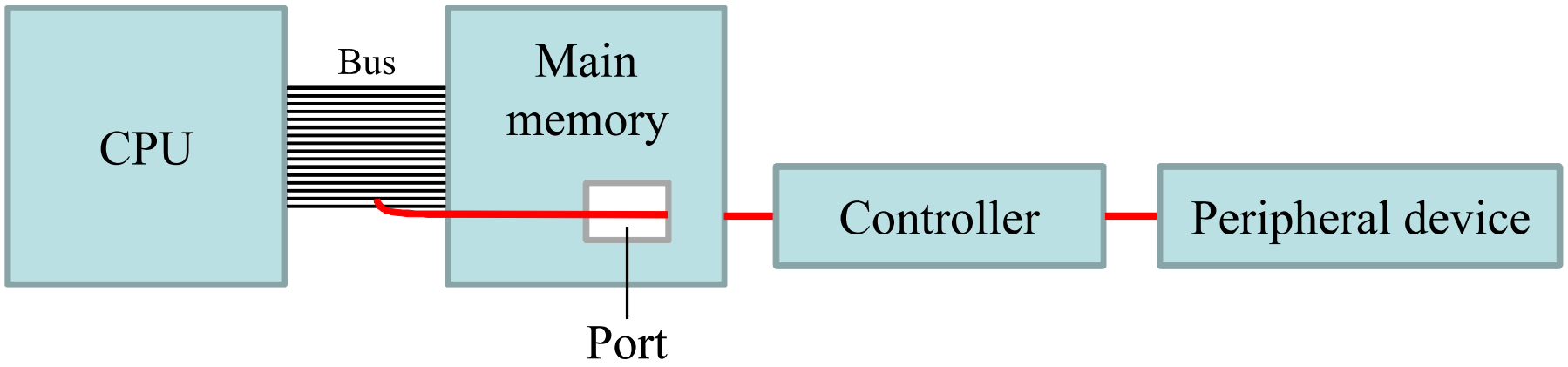
Controller

- Specialized
- General: USB, FireWire



Memory-mapped I/O

- I/O as LOAD, STORE

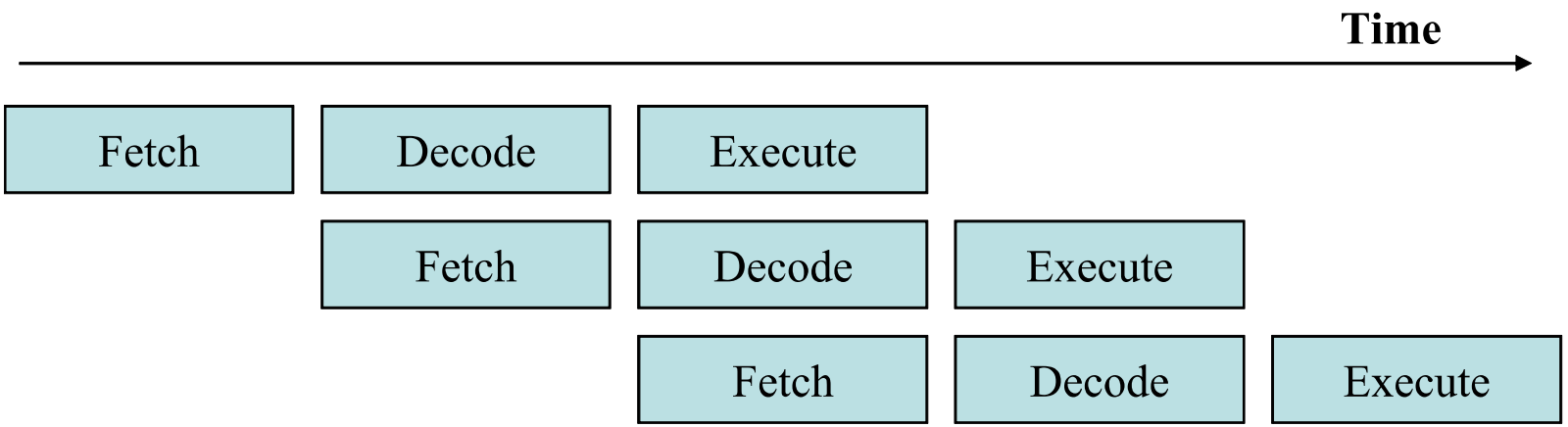


Communication with Other Devices

- DMA: direct memory access
  - Once authorized, controllers can access data directly from main memory without notifying CPU.
- Hand shaking
  - 2-way communication
  - Coordinating activities
- Parallel/Serial
- Transfer rate: **bit** per second (bps, Kbps, Mbps, etc)

Pipelining

- Throughput increased
  - Total amount of work accomplished in a given amount of time.
- Example: pre-fetching
  - Issue: conditional jump



Parallel/distributed Computing

- Parallel
  - Multiprocessor
  - SISD, MISD (*e.g.*, pipelining), SIMD (*e.g.*, MMX, SSE, vectorization), and MIMD.
- Distributed
  - Linking several computers via network
  - Separate processors, separate memory
- Issues:
  - Data dependency
  - Load balancing
  - Synchronization
  - Reliability

To Parallelize XOR Not to Parallelize

How to parallelize?

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

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

```
declare A[0]~A[99]
input A[0], A[1], A[2]

for (i = 3; i<100; i++)
    A[i] = A[i-2] + A[i-3];
```

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
}
```

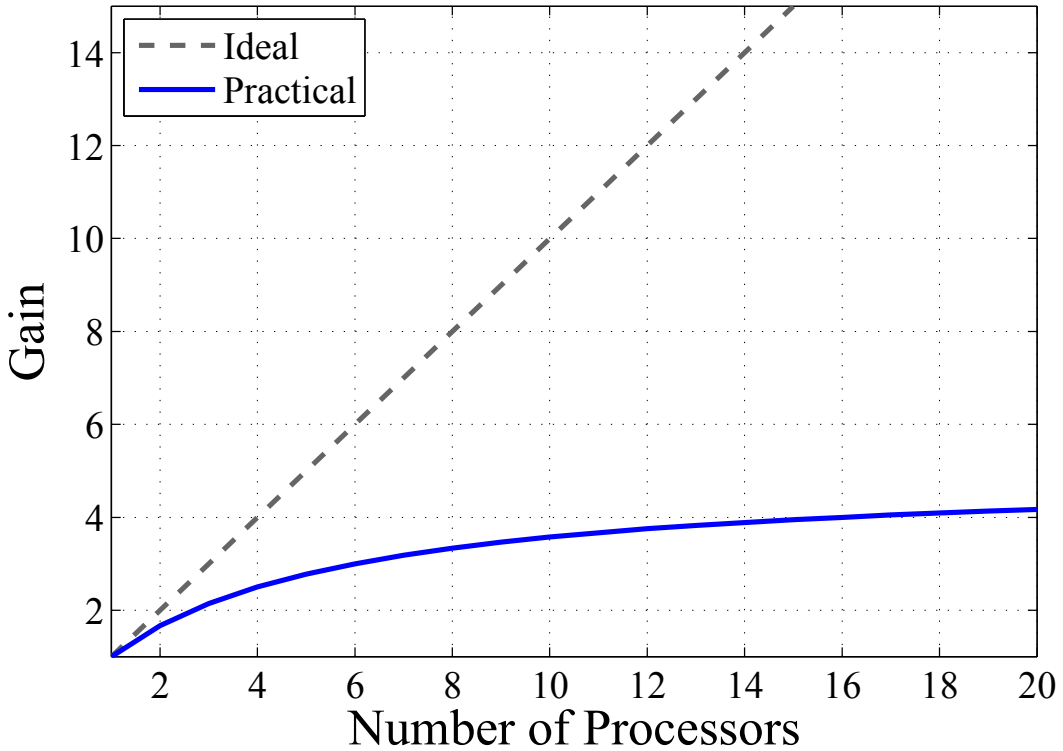
```
for (i = 3; i<98; i+=2) {
    A[i] = A[i-2] + A[i-3];     ← CPU 0
    A[i+1] = A[i-1] + A[i-2];   ← CPU 1
}
A[99] = A[97] + A[96];
```

Speedup & Scaling

- Speedup (Amdahl's law)

P: proportion that can be parallelized.

$$Gain = \frac{1}{\frac{P}{M} + (1 - P)}$$



- $P = 0.8$.
- Maximum gain is $\frac{1}{1-0.8} = 5$.