Introduction to Computer Science
Lecture 2: Data Manipulation 資料的操作

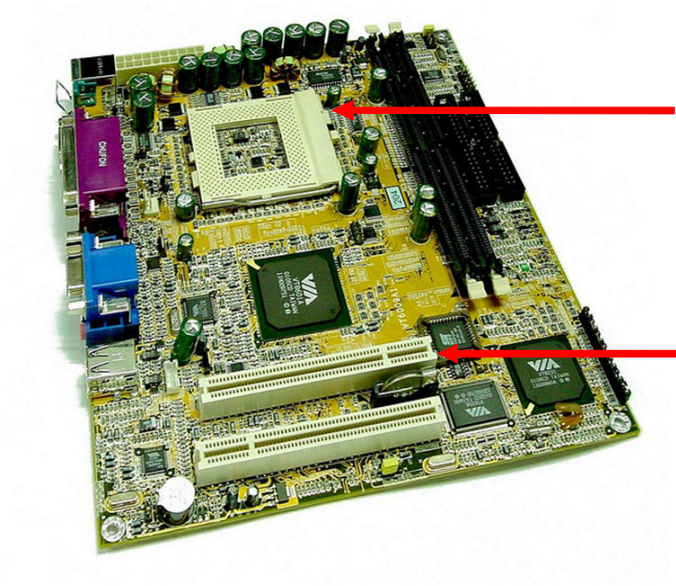
Tian-Li Yu

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

tianliyu@cc.ee.ntu.edu.tw

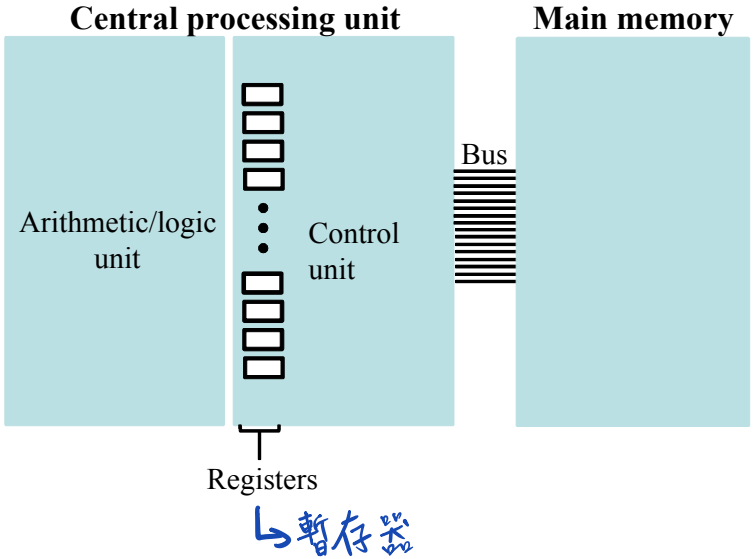
Slides made by Tian-Li Yu, Jie-Wei Wu, and Chu-Yu Hsu

Motherboard



Computer Architecture

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



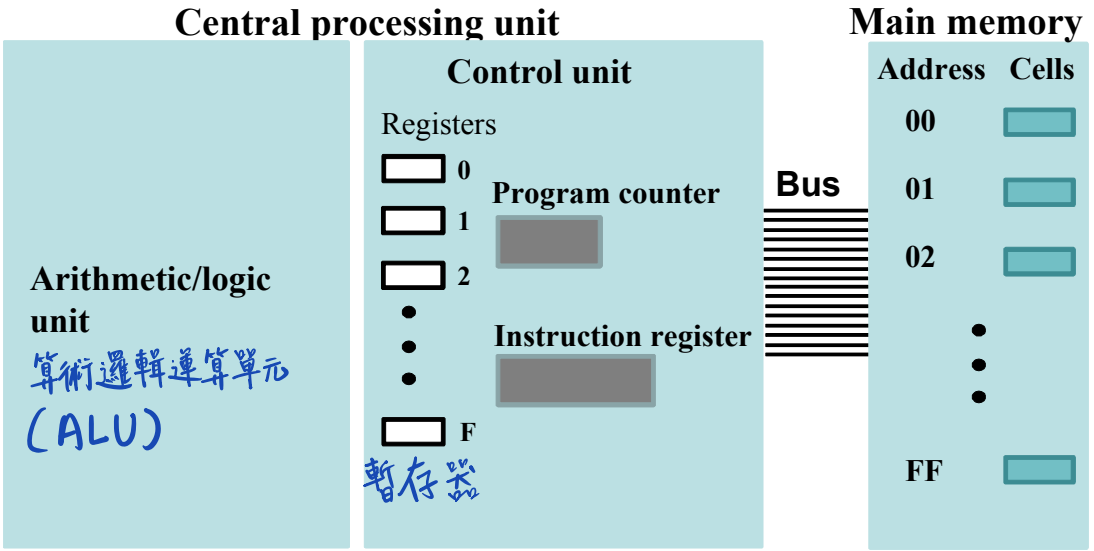
Adding Values Stored in Memory

1. Get one of the values to be added from memory and place it in a register.
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.

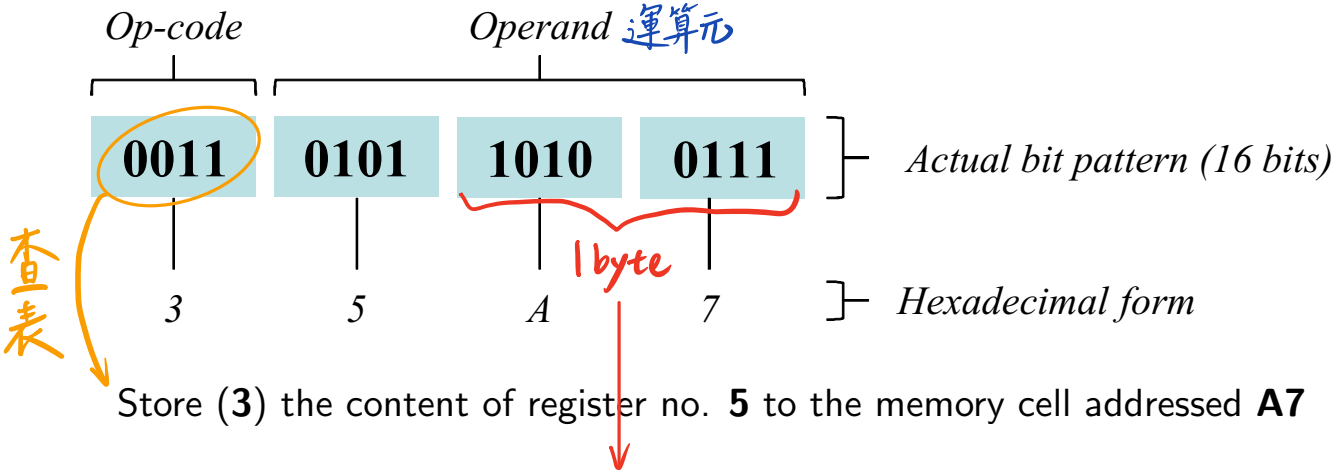
Machine Instructions 機械指令

- Data transfer 資料傳輸
  - LOAD, STORE, I/O
    (RAM→CPU) (CPU→RAM)
- Arithmetic/logic
  - AND, OR, ADD, SUB, etc.
  - SHIFT, ROTATE
- Control
  - JUMP, HALT
- RISC (Reduced Instruction Set Computing) (PRC, SPARC)
  vs. CISC (Complex instruction set computing) (x86, x86-64)
  ↳目前主流

Architecture of a Simple Machine



Example of a Machine Instructions



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

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

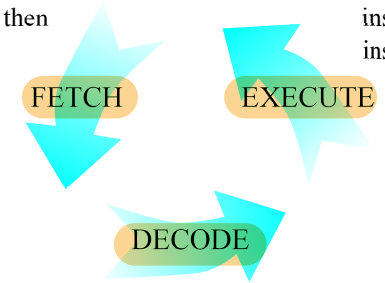
Adding Two Values Revisited

Encoded 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 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 終止	

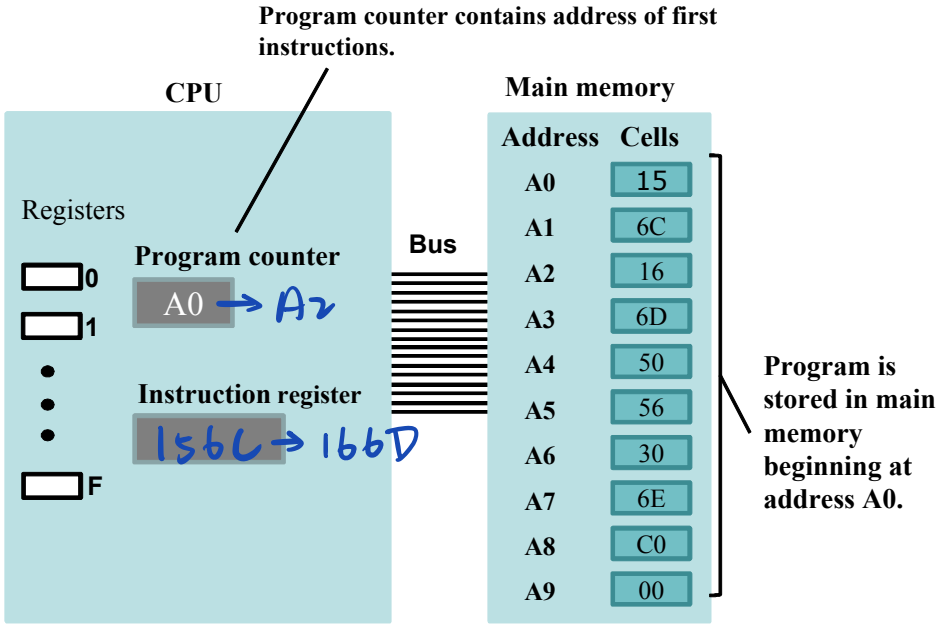
高階語言(ex: C/C++)在做的事

Program Execution

- Instruction register, program counter
- Machine cycle
  - clock 時脈 → 每秒執行 Machine cycle 的次數
  - benchmarking



Fetch


Program counter contains address of first instructions.
CPU
Main memory
Registers
Program counter
A0
A2
Bus
Address
Cells
A0
15
A1
6C
A2
16
A3
6D
A4
50
A5
56
A6
30
A7
6E
A8
C0
A9
00
0
1
F
Instruction register
156C
166D
Program is stored in main memory beginning at address A0.

Logic/Bit Operations

- Masking

AND

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

Setting the first 4 bits to 0.

目的：把一些bit設成0.

OR

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

Setting the latter 4 bits to 1.

目的：把一些bit設成1.

XOR

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

Inverting the latter 4 bits.

Shift/Rotation

- Logic shift

  新補的數都是0.

  10100000 → 01010000 (right)

  → 01000000 (left)

- Arithmetic shift

  和原本第一位數字相同(保持正負)

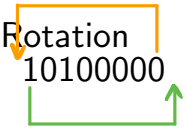
  10100000 → 11010000 (right)

  → 11000000 (left)

- Rotation

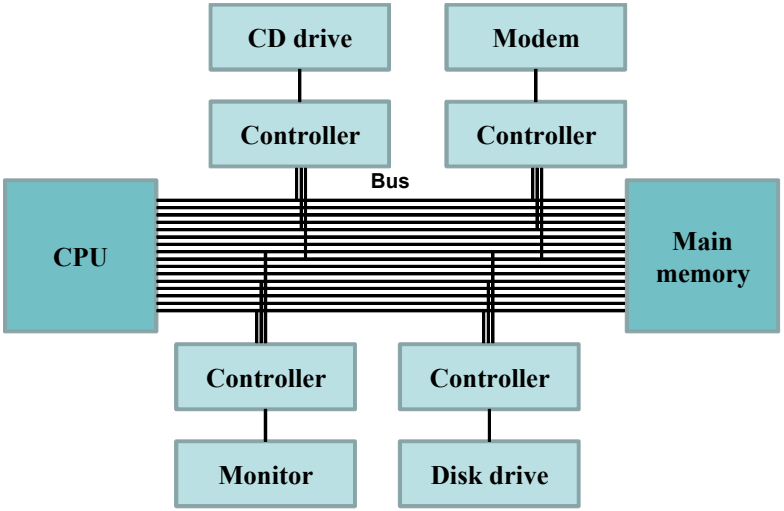
  10100000 → 01010000 (right)

  → 01000001 (left)



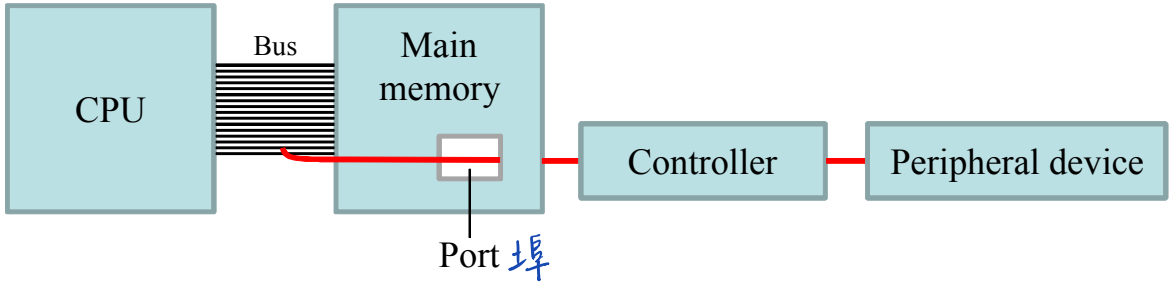
Controller

- Specialized
- General: USB, FireWire



Memory-mapped I/O

- I/O as LOAD, STORE

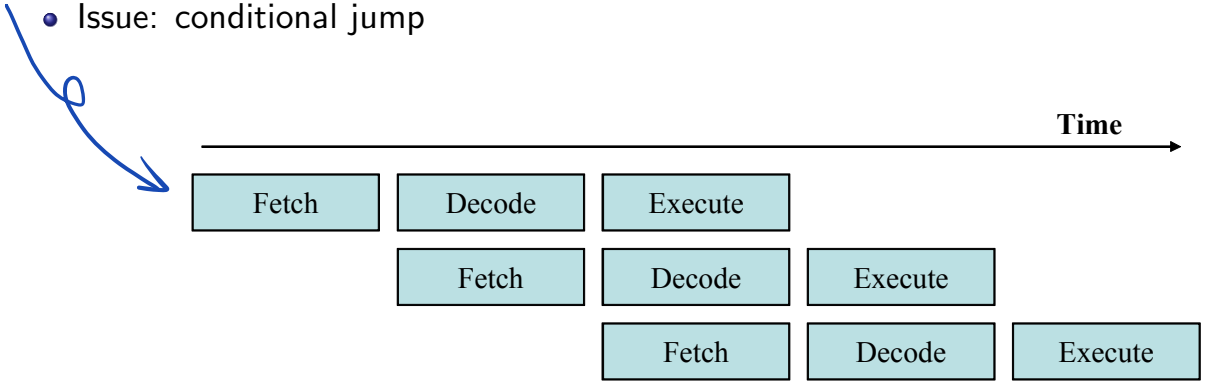


Communication with Other Devices

- 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 平行傳輸 / 序列傳輸
- 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



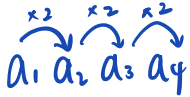
Parallel/distributed Computing

- Parallel 平行處理
  - Multiprocessor
  - MIMD, SISD, SIMD, MISD
    - ex: pipelining
- Distributed 分散處理
  - Linking several computers via network
  - Separate processors, separate memory
- Issues: 問題
  - Data dependency
  - Load balancing
  - Synchronization
  - Reliability 信度

S/M: single/multiple
I: instruction
D: data

To Parallelize XOR Not to Parallelize

How to parallelize?



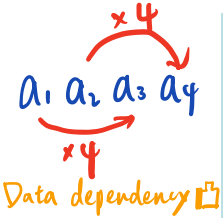

```
declare A[0]~A[99]
input A[0]

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



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

← CPU 0
← CPU 1



2 CPUs

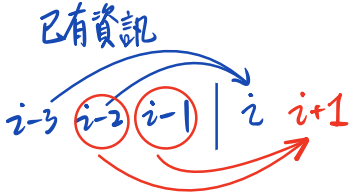
```
declare A[0]~A[99]
input A[0], A[1], A[2]

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



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

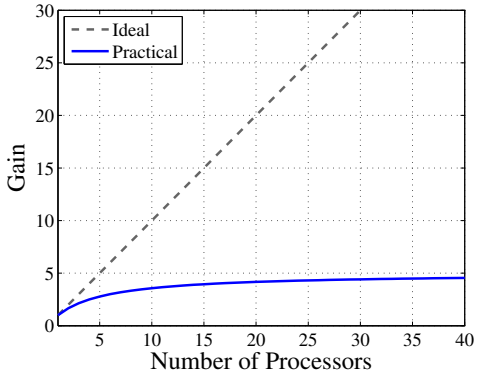
← CPU 0
← CPU 1



Speedup & Scaling

- Speedup (Amdahl's law) → 核心數增加後可增加的速度

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



P : *parallelizable* (proportion that can be parallelized)

S : *serial only*

ex: $P = 0.8$

Max Gain = $\frac{1}{1-0.8} = 5$

License

Page	File	Licensing	Source/ author
2			flickr, Author: viagallery.com, Source: http://www.flickr.com/photos/viagallery/2036598755/, Date: 2012/02/19, This file is made available under the Creative Commons Attribution 2.0 Generic