.CH 1 基本概論

A. **Hardware 5 大單元(John von Neumann)** – input、output、memory、ALU、control (von neumann)

B. **Machine Instruction 完成週期** – Instruction Fetch、Decode、Execution、Memory/Fetch Operands、Write Result/Write Back

C. Pipeline - 第一條指令總 clock cycle 數 + (n-1)*max(指令週期中各週期所須之 clock 數)

D. **Bus 匯流排** – Data Bus、Address Bus、Control Bus→ Von neumann bottleneck(馮紐曼瓶頸)

E. **Store Hierachy 儲存層次架構** – Reigister、cache、memory(volatile)；Flash Memory、SSD、Hard Disk、CD-ROM、Blue-Ray、Tape(Non-volatile)

F. **Register 暫存器** – IR、PC、MAR(address)、MDR(data)/MBR(buffer)、Accmulator、Base&Limit register

G. **Instruction Cycle and Register 指令週期與暫存器**

H. **Cache** – p * (cache time + 記憶體存取 time) + (1 – p) * (cache time + (n + 1)記憶體存取 time)[N 為幾層]、write through、write back

I. **Caching 概念** – 先存慢再存快

J. **Memory** – Bootstrap Loader、RAM : DRAM(電容、電晶體)、SRAM(Flip-Flop)；ROM : PROM、EPROM、EEPROM、Flash ROM

K. **CPU 介紹** – Control Unit + ALU + Register + Cache

L. **CPU 指令集架構** – RISC(精簡，complex instruction set computer)、CISC(複雜，reduced instruction set computer)

M. **CPU 衡量速度** – CPI(Cycle Per Instruction)、MIPS(Million Instructions Per Second)、MFLOPS、Clock Rate

N. **計量單位 ： 儲存** – KB、MB、GB、TB、PB($2^{50}, 10^{15}$)、EB、ZB、YB

O. **計量單位 ： 時間** – ms、us、ns、ps

P. **定址模式(Addressing Mode)** – 直接(Immediate)、間接(Direct)→放記憶體位置；絕對(Relative)、相對(Indirect)→放數值、Offset

Q. **電腦世代劃分** – 真空管、電晶體、IC(積體電路)、VLSI、AI

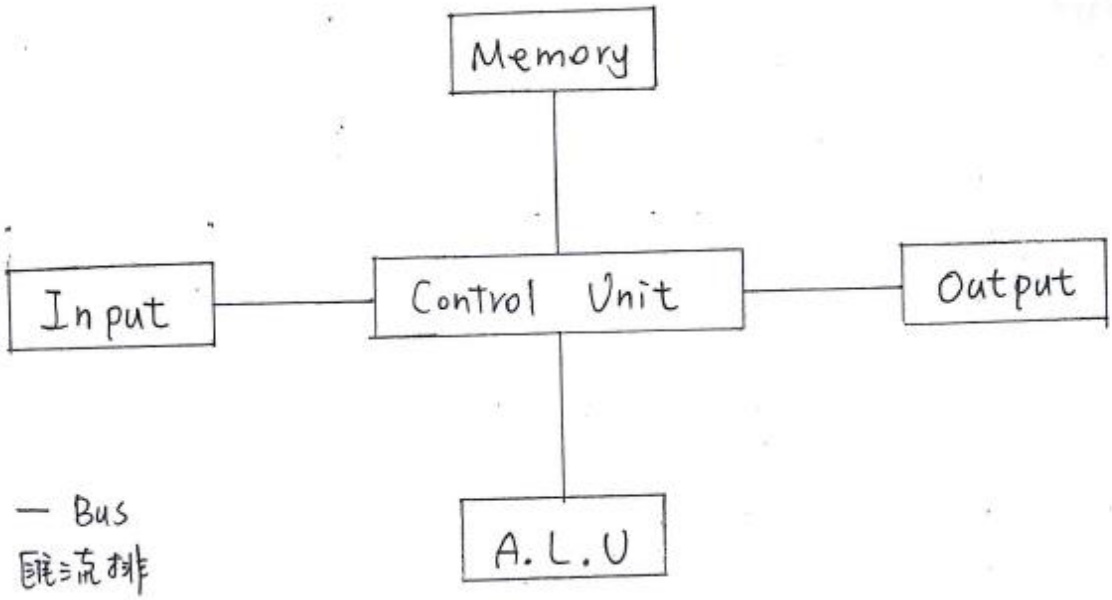
R. **摩爾定律** – 每 18 個月，電晶體增加倍

S. **費林分類法(Flynn's Taxonomy)** – SISD(個人電腦)、MISD(無)、SIMD(超級電腦、顯卡)、MIMD(分散式計算)

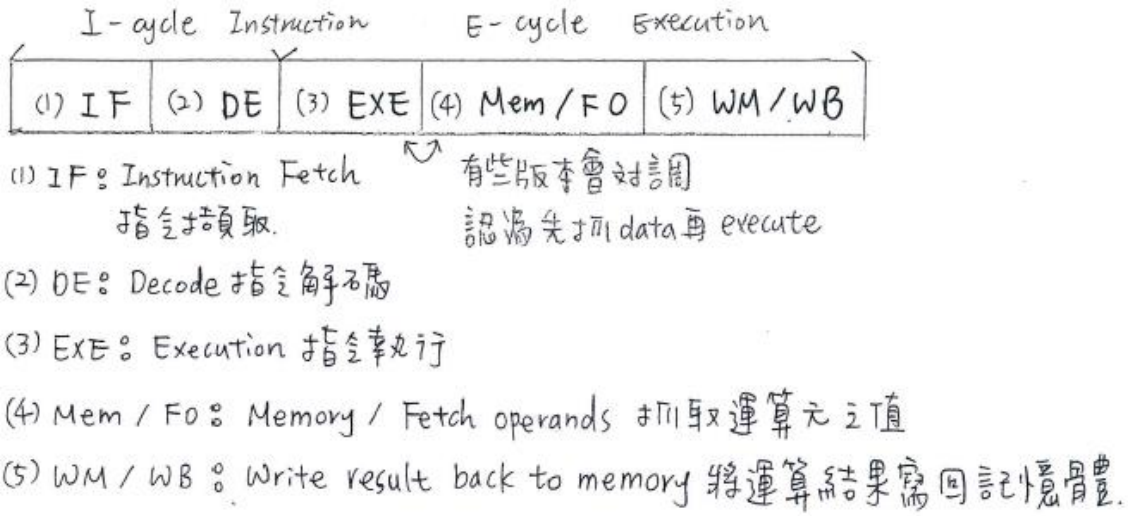
T. **系統架構** – System Software(OS、Text Editor、Linking Loader、Compiler)、Appliaction Software

---

1. Hardware 5 大單元(John von Neumann) – 內儲程式概念(Stored – program Concept)
   - Consequential execution(依序執行)
   - 架構圖
     (1) **Input Unit**： 接收 users 所輸入之資料、程式、指令之硬體 → 鍵盤、滑鼠、掃描器、Disk、touch screen
     (2) **output Unit**： 負責輸出運算所需之結果、程式等之設備 →螢幕、印表機、touch screen、VR 頭盔
     (3) **Memory Unit**： 通常泛指 RAM、ROM，也可包含 Cache、Registers
     (4) **A.L.U(Arithmetic & Logic Operation Unit)**： 算術及邏輯運算單元、指令被執行/運算之單元
     (5) Control Unit(C.U)： 控制單元，負責指揮、協調、控制各單元之運算，以完成機械指令，對指令 Decode



2. Machine Instruction 完成週期



- **Instruction fetch**：依據 PC (Program Counter)之值至 memory， 抓取對應指令，PC 存下一條指令位置
- **Decode**：Control Unit 解譯指令之 op-code；若 op-code 有 n bits，則 cpu 最多提供 2^n 個指令；反之 n 個指令，op-code 為 log(n)→200 條指令，log(200) = 7.xxx 無條件進位 = 8
- **Execution :** 通知 ALU 做對應運算
- **Fetch Operand (optional mem access)**： 從 register 或 memory 抓取 operand
- **Write result to Memory (optional mem access)**： 寫回暫存器或沒有要寫或更改 PC 值或寫回 Memory

- 其他版本 ：IF -> DE -> EXE -> WB or IF -> DE -> EXE

3. Pipeline

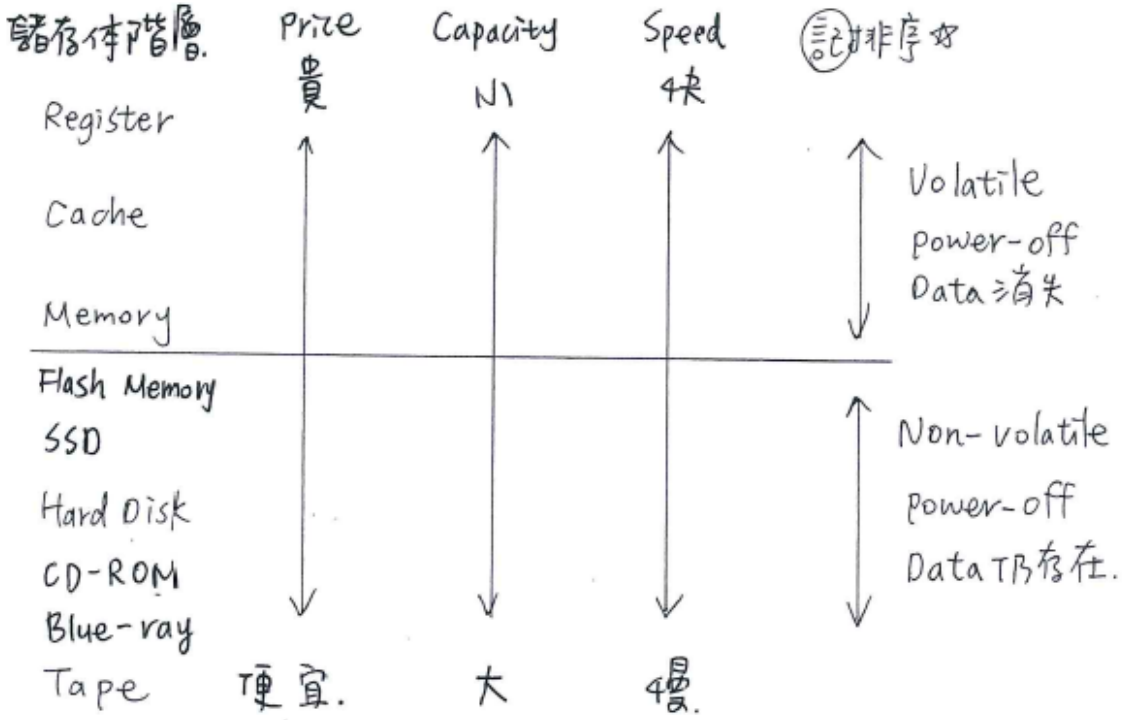
- 將不同指令的不同周期重疊執行
- N 個指令 ： 第一條指令總 clock cycle 數 + (n-1)*max(指令週期中各週期所須之 clock 數)
- Ex. 3 條指令(IF -> ID -> EX -> MEM ->WB)→5 +(3-1)*1 = 7 最長指令 cycle 為 1

Instr No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

4. Bus 匯流排

- 連接硬體間的排線
- 主要分為三種
  (1) **Data Bus(資料匯流排)**：傳輸指令或資料，雙向；資料匯流排線有幾位元，就可以決定電腦是幾位元的電腦，例如 64 位元電腦指的是電腦的資料匯流排寬度為 64 位元
  (2) **Address Bus(位址匯流排)**：傳輸 memory address，只送往 memory，單向；位址匯流排線路的數目取決於記憶體位址空間，如果記憶體有2^N個 Memory space(0~$2^N - 1$)(Byte 為最小單位)，位址匯流排必須有 N 條線路
  (3) **Control Bus(控制匯流排)**：傳送從 control uint 發出的 control signal，用來傳送控制信號的線路，負責傳送 CPU 執行指令時所發出的訊號，由於信號僅由 CPU 發出，所以是單向輸出的排線
- 例 ： 一記憶體大小為 256MB，共 16 條資料線，位址線幾條?
  →16bits = 2bytes，一個記憶體位置最小單位為 2byte，256MB/2 = 128MB = 2^{27}，27 條
- **Von neumann bottleneck(馮紐曼瓶頸)**： 系統速度受限於 bus 傳輸效能

5. Store Hierarchy 儲存層次架構



6. 暫存器(Register)

- 於 cpu 內，用於暫存指令、運算元、中間值、結果值
- 常見 Register

(1) IR (Instruction Register) 指令暫存器
Control Unit 從 Memory 抓取之指令，會暫置於此 register 中。

(2) Programming Counter (P.C.) 程式計數器
存放 address of the next instruction。

(3) MAR (Memory Address Register)
存放 memory 存入，取出之位址。

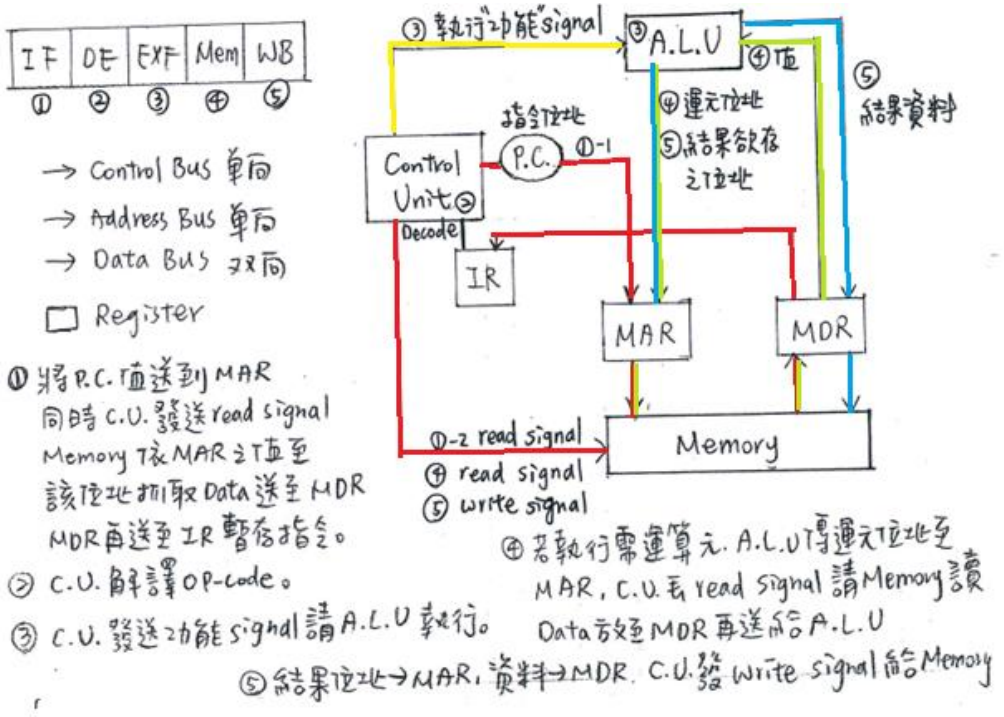
(4) MDR / MBR (Memory Data / Buffer Register)
欲寫入 memory 之 Data 或從 memory 讀取之 Data 均先暫存於此。

(5) Accumulator 累加器 或 General purpose registers
暫存運算元之值、中間值或結果值之用途。通常很多個。
eig. R0 ~ R15, Intel AX registers
例. 高階語言 組合語言 Accumulator
A=A+B → LOAD R1, A // A值載入R1暫存器中
LOAD R2, B // B值載入R2暫存器中
ADD R1, R2 // R1與R2相加, 結果放回R1
STORE A, R1 // 將R1值傳回變數A中

(6) Base and Limit register
OS用此套register記錄程式在memory中之起始位址與大小。

(7) Base register 放置基底相對位址之用途。
Index register 放置索引相對位址之用途。

7. 指令週期與暫存器



8. Cache / Cache Memory 快取記憶體

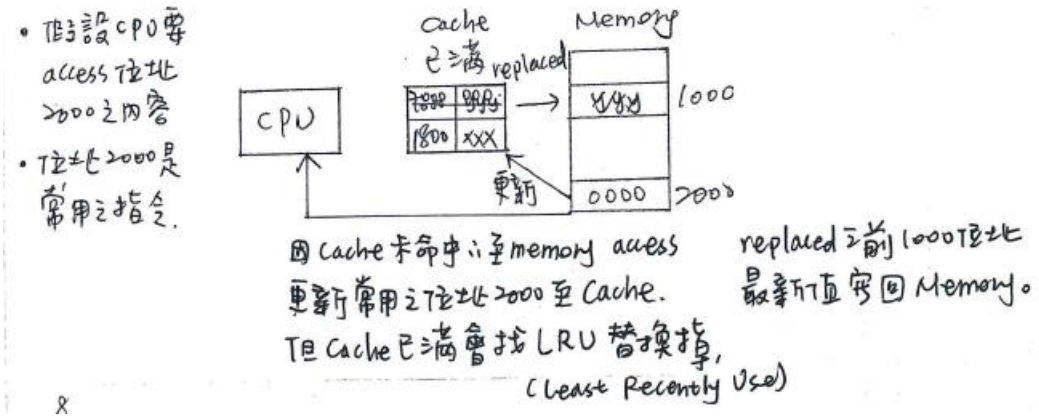
- 目的：改善 memory 存取速度慢，cpu 需花時間等待
- 作法：將經常被存取之資料置入 cache 中，先到 Cache 找，有就拿，沒有就到 memory 拿且可能須更新 cache 內容
- 有效時間計算：沒 hit 則須記憶體存取時間加上原本 cache 時間

$$P * \text{Cache Time} + \text{沒命中}(1-P) * (\text{Cache Time} + \text{Memory Access Time})$$

P = Cache 之 Hit ratio 命中率　　* CPU 抓到指令/資料百分比過程
= 有效的 memory access time

假設：Cache Time：10 ns
Memory access time：400 ns
Cache Hit ratio：80 %
則 effective memory access time = ? ns
0.8 * 10 ns + 0.2 * (10 ns + 400 ns) = 90 ns ※

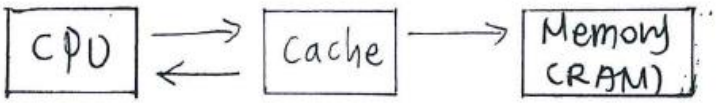
- Cache 內容何時寫回 memory：寫回 cache 但未寫回 memory 資料不一致
  (1) **Write through(即刻寫回)**：寫回 cache 時，也寫回 memory
    A. 優：保證一致
    B. 缺：耗時，若短時間內對某記憶體位址更新頻繁(for loop)，此作法多餘，喪失 cache 用意；memory 寫入頻繁時，喪失 cache 優勢
  (2) **Write back(稍後寫回)(居多)**：當 cache 內容要被 replace 之前，或 I/O 之前才寫回



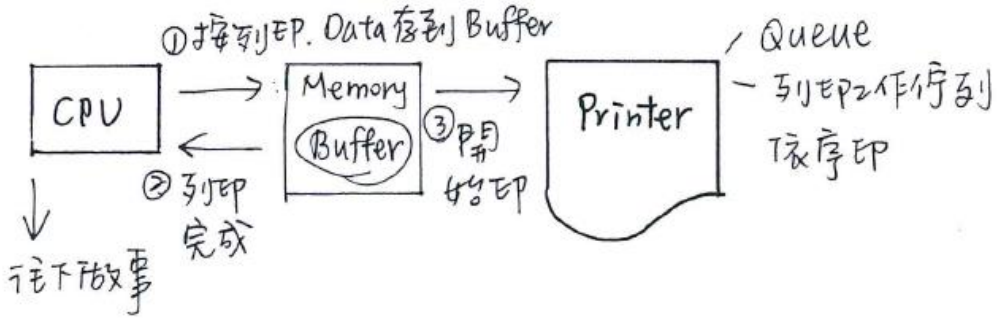
9. Caching

- 一種機制或觀念：在存取速度慢之前，先到速度快的存取，若沒有命中則到速度慢的存取
- 例子

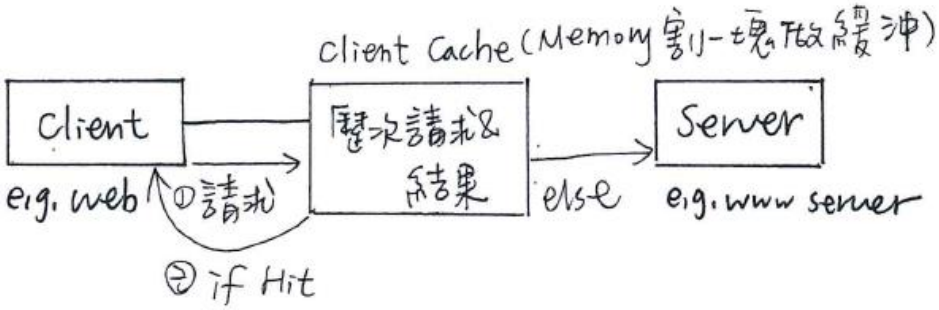
例1：Cache



例2：Buffering



例3：Client Cache



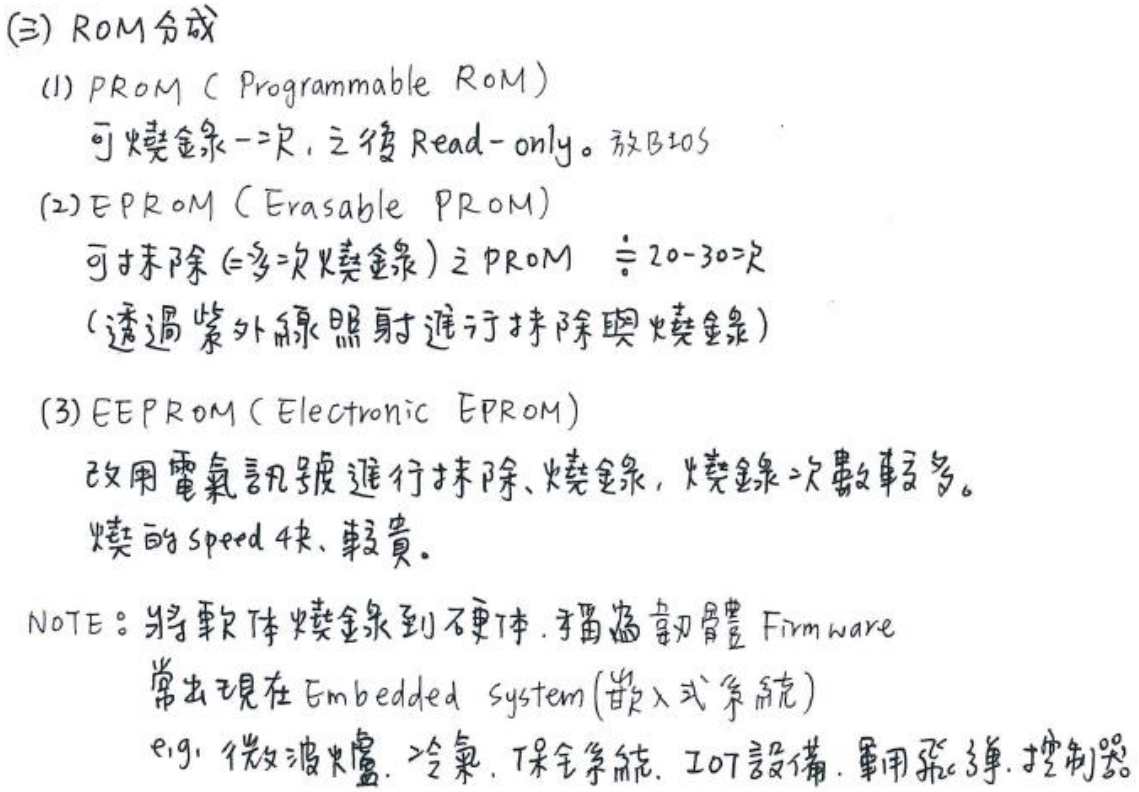
10.Memory

RAM	ROM
Random Access Memory	Read Only Memory
可隨意存取(write)取(read)Data	唯讀 Memory，不能寫
Volatile： 沒電就不能存資料	Non- Volatile： 沒電還能存資料
容量較大	容量較小
存取速度快	存取速度慢
分成 DRAM、SRAM 用以暫存輸入資料、Program output input result 之用	分成 PROM、EPROM、EEPROM、Flash ROM，通常存放 BIOS、Bootstrap Loader

BIOS : Basic Input/Output System → 完成硬體檢測和資源分配，將硬碟 MBR 中的 Boot Loader 讀到系統的 RAM 中，然後將控制權交給 OS Boot Loader

Bootstrap Loader： 主要執行任務就是將核心映像從硬碟上讀到 RAM 中，然後跳轉到核心的入口點去執行，也即開始啟動作業系統

DRAM	SRAM
Dynamic RAM	Static RAM
電容、電晶體	Flip-Flop(正反器)
由於電容會放電，每隔一段時間必須重複充電(更新)，因此比較耗時	Non- Volatile： 沒電還能存資料
容量較大	容量較小
電容會放電，每隔一段時間必須重複充電(更新)，因此比較耗時	正反器不會放電，不須重複充電(更新)，因此比較省時
較便宜	較貴
存取速度較慢	存取速度較快
DDR4	Cache Memory



[補充]

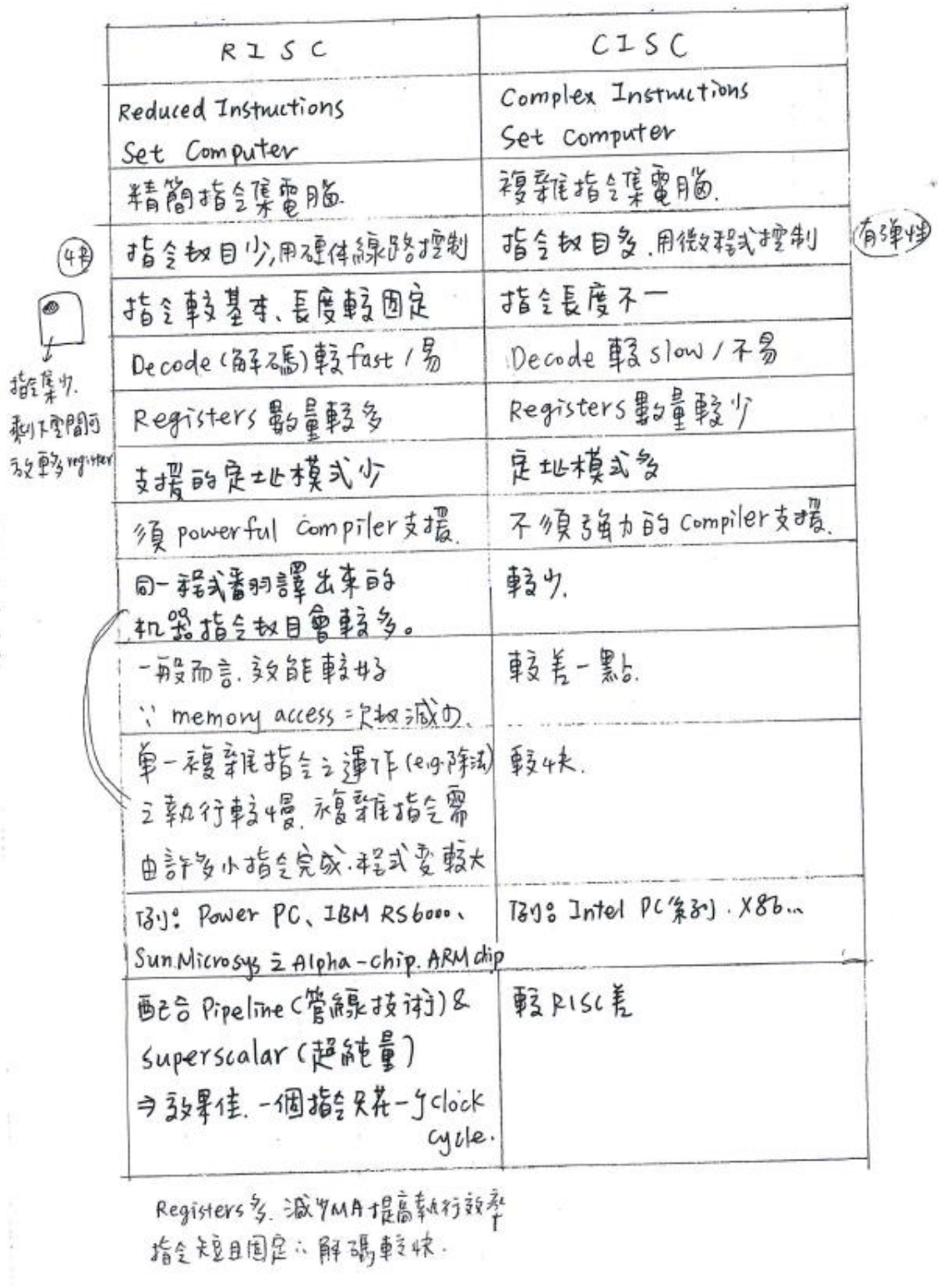
Flash ROM(Flash Read-Only Memory, 快閃型唯讀記憶體)：主要用以替代 EEPROM，作為系統程式儲存及記錄。著重在指令的快速讀取及對系統的開機管理，多用於 PC Card 記憶卡、主機板和 Smart Card；三大主流架構 →NOR、NAND、EE-NOR

11.CPU 介紹

- Central Processing Unit

- 理論上組成 : control unit + ALU + memory
- 實際上組成 : control unit + ALU + registers + 內建 cache
- 並未包含 memory(ram、rom)
- Cache memory 可分多層，level 1 最快以此類推

12.CPU 設計之兩大策略 : RISC()、CISC



RISC	CISC
Reduced Instructions Set Computer	Complex Instructions Set Computer
精簡指令集電腦.	複雜指令集電腦.
指令數目少，用硬體線路控制	指令數目多，用微程式控制
指令較基本、長度較固定	指令長度不一
Decode(解碼)較 fast / 易	Decode 較 slow / 不易
Registers 數量較多	Registers 數量較少
支援的定址模式少	定址模式多
須 powerful Compiler 支援.	不須強力的 Compiler 支援.
同一程式翻譯出來的機器指令數目會較多。	較少.
一般而言，效能較好 ∵ memory access 次數減少.	較差一點.
單一複雜指令之運作(e.g.陣列)之執行較慢. 複雜指令需由許多小指令完成，程式變較大	較快.
例：Power PC、IBM RS6000、Sun Microsys 之 Alpha-chip. ARM chip	例：Intel PC 系列. X86...
配合 Pipeline(管線技術)& Superscalar(超純量) ⇒ 效果佳. 一個指令只花一個 clock cycle.	較 RISC 差

(4字)

指定數少. 剩下空間可放更多 register

(有彈性)

Registers 多. 減少MA提高執行效率

指令短且固定，解碼較快.

13.CPU 速度衡量

- **CPI (Cycle per instruction) :** 每條指令之平均時間週期
- **MIPS(Million instruction per second)**：每秒可執行多少百萬(10^6)條指令 (million instructions per second)

  (1) Cpu 速度 5 MIPS，則一條指令花幾秒 => $\frac{1}{5*10^6}$秒 (一秒 5*10^6 條指令)

  公式：$\frac{1}{MIPS速度*10^6}$

  (2) 平均一條花 $\frac{5}{10^8}$ 秒，則 CPU 幾 n MIPS => $\frac{5}{10^8} = \frac{1}{2*10^7} = \frac{1}{20*10^6}$

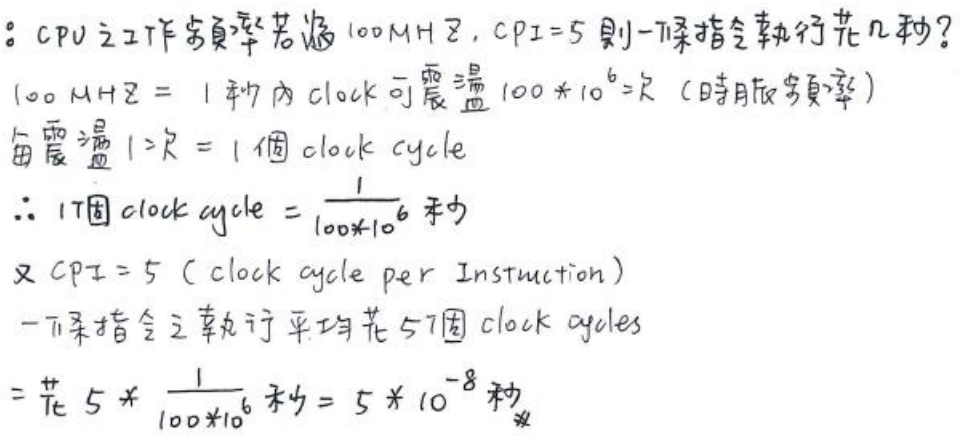
  => MIPS 速度為 20
- MFLOPS (Floating-point operations per second) 每秒浮點運算次數
- **Clock 與 CPI (指令平均周期數) (Cycle per instruction)**

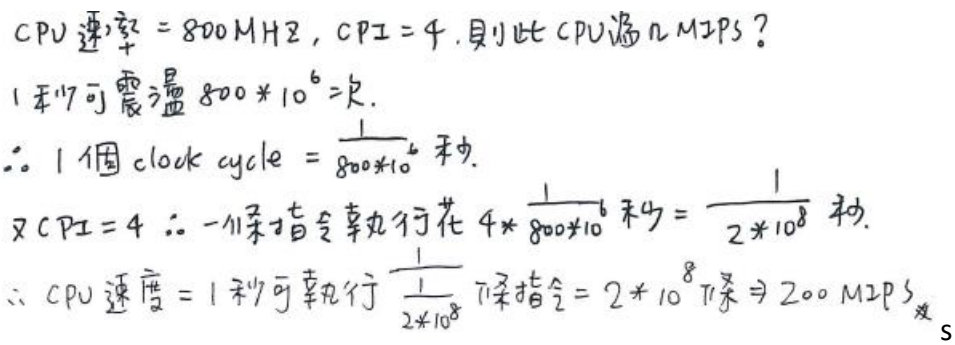
  ⇨ Clock cycle： 震盪一次的時間 = $\frac{1}{frequency}$, ex. 100MHz => $\frac{1}{100*10^6} = \frac{1}{10^8}$

  ⇨ 時脈速度(頻率)(clock rate) : CPU 每秒執行的 cycle 數 , ex. 100MHz => 每秒 100*10^6 個 clock cycle

  ⇨ **CPU Time**： 執行一個 process 全部所需的 Clock Cycle 所需的時間

  = CPU Clock Cycle * Clock Cycle Time = $\frac{Clock\ Cycle\ Time}{Clock\ Rate}$





14.計量單位 – 儲存

位元組的次方單位

十進位字首 (SI)			二進位字首 (IEC 60027-2)		
名字	縮寫	次方	名字	縮寫	次方
千位元組	KB	10^3	Kibibyte	KiB	2^{10}
百萬位元組	MB	10^6	Mebibyte	MiB	2^{20}
吉位元組	GB	10^9	Gibibyte	GiB	2^{30}
兆位元組	TB	10^{12}	Tebibyte	TiB	2^{40}
拍位元組	PB	10^{15}	Pebibyte	PiB	2^{50}
艾位元組	EB	10^{18}	Exbibyte	EiB	2^{60}
皆位元組	ZB	10^{21}	Zebibyte	ZiB	2^{70}
佑位元組	YB	10^{24}	Yobibyte	YiB	2^{80}

- memory, disk 之容量常用二進位表示，1KiB = 1,024Byte，kilo binary byte
- **位元 (Bit)**：一個位元 (bit) 可以包含 「0」、「1」 這兩個數值，是計算機的最基本資料單位
- **位元組 (Byte)**:又稱字節，為 8 = 2^3 個位元，為大部分計算機架構 (architecture) 中的定址單位 (Byte addressing)，記憶體機構的最小尋址單位為 1 個位元組，無法單獨存取 1 bit 的資訊或者任意小於 1 位元組的資訊
- **字組 (word)**：又稱字元組，設計處理器 (CPU) 時，處理資料的自然單位，由一或多個位元組 (byte) 所組成
  - ⇨ 假設記憶體大小為 16 bytes = 2^4 bytes，一個 word 為 1 byte，則位置空間為 0000~1111(0~15)，因此,MAR 需要 4bits 來表示，MAR 位址線有 4 條
  - ⇨ 32 條 address bus(MAR 32 bits) -> 2^32 * 記憶體一個位置(2^3 bits)(1

byte) = 4GByetes

15.計量單位 – 時間

符號	m	μ	n	p
名字	毫 milli	微 micro	奈 nano	皮 pico
精確值	2^{-10}	2^{-20}	2^{-30}	2^{-40}
近似值	10^{-3}	10^{-6}	10^{-9}	10^{-12}

*常用在時間、速率、速度、傳輸率上。

1 ms = 10^{-3} second = 10^{3} μs = 10^{6} ns

1 μs = 10^{-6} second = 10^{-3} ms = 10^{3} ns

1 ns = 10^{-9} second = 10^{-3} μs = 10^{-6} ms

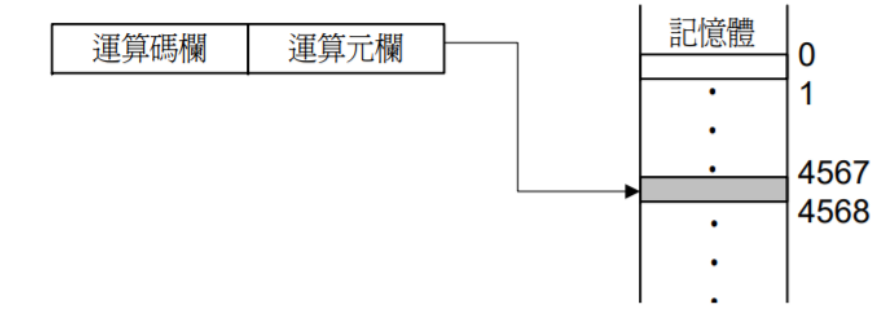
- Cpu 速度 = 10 MIPS，一條指令多少 ns ?

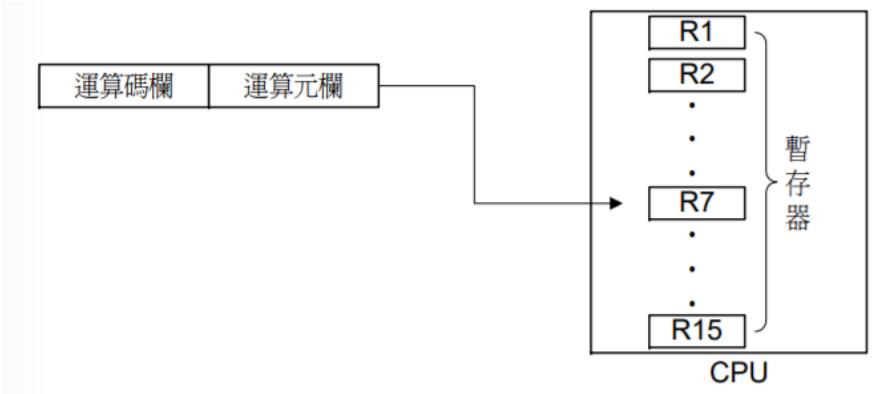
  ⇨ $\frac{1}{10*10\text{^}6}$ 秒 = 10^{-7}秒 = 0.1 us = 100 ns

16.定址模式 (Addressing Mode)

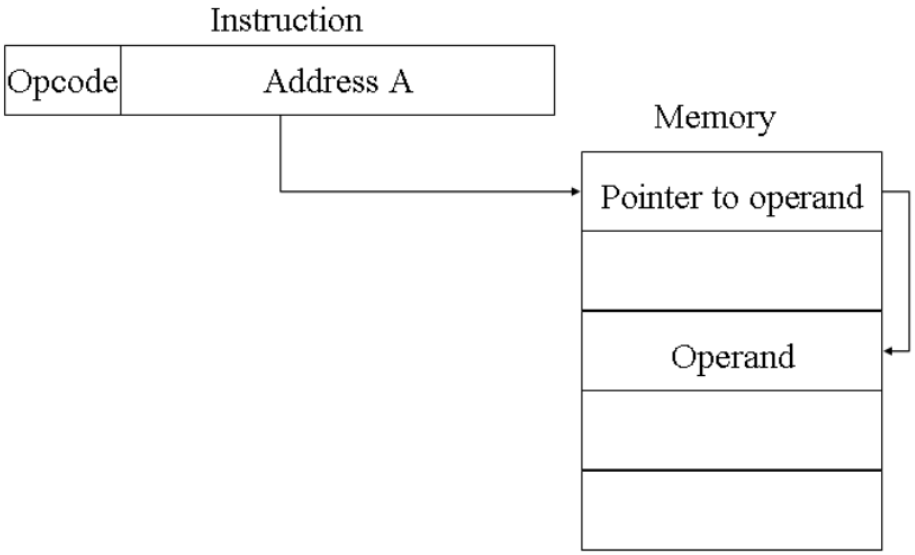
- **直接定址(direct address)**
  - ⇨ 指令的運算元欄內的值表示資料存放於記憶體的實際位址 (有效位址，Effective Address)，故需要做一次的記憶體讀取，以取得所需之資料
  - ⇨ 優 : operand 值域放在記憶體，較不受限制
  - ⇨ 缺 : 與立即定址相比，需存取 memory 一次，速度較慢；程式執行起始位址改變，運算元欄需 relocation (linking loader 負責修正)

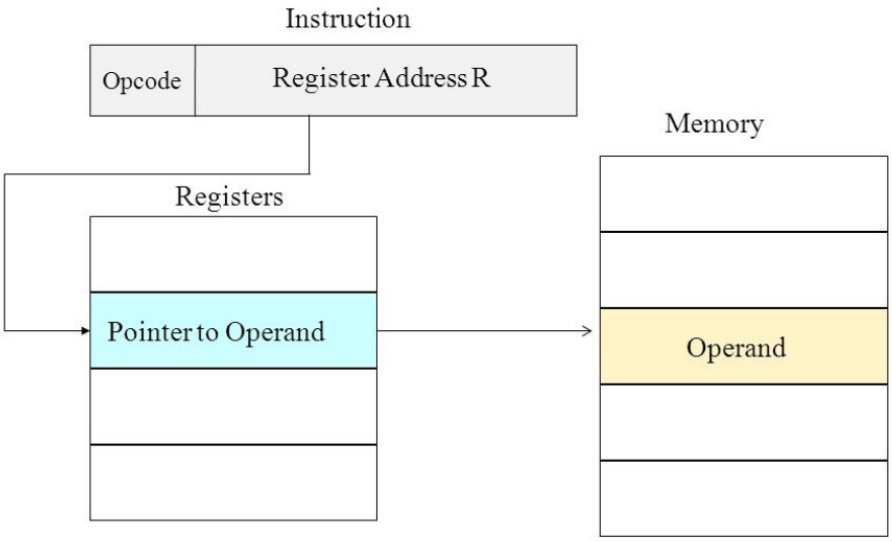
例：MOV AX，[4567]



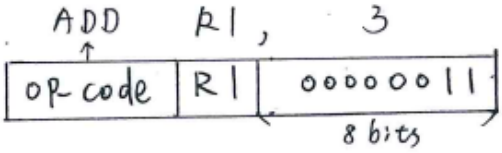


- **間接定址模式(indirect address)**
  - ⇨ 指令的運算元欄內的值為存放 operand 所在地址之值的位址
  - ⇨ 優：適合跳躍擷取
  - ⇨ 缺：需兩次存取得 operand 值，程式執行起始位址改變，運算元欄需 reloaction(linking loader 負責修正)





- **立即(絕對)定址(Immediate；absolute operand)**
  - ⇨ Operand 欄位為欲運算之值



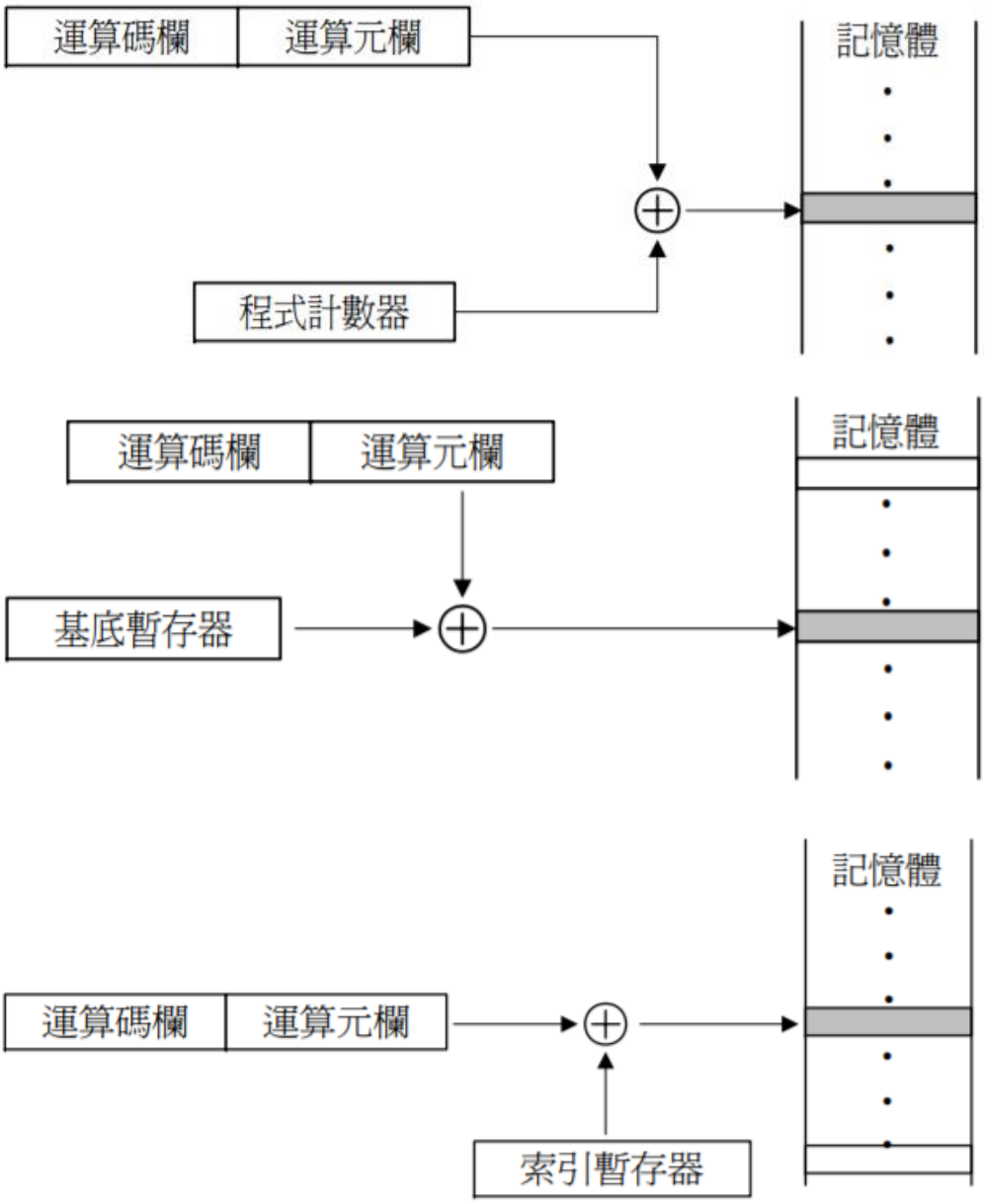
  - ⇨ 優：速度最快，不須額外 memory access，不須 relocation
  - ⇨ 缺：值域受限

- **相對定址模式(relative address)**
  - ⇨ Operand 放 Operand 的偏移量(offset)，不論從何執行，偏移量都一樣
  - ⇨ 優：不須做 relocation
  - ⇨ 缺：抓 operand 值須先做加法，速度稍慢，offset 不可太大，萬一超過會採直接定址

常用→ (1) P.C. 相對定址 ⇒ offset = operand 位址 − P.C. 值.

(2) 基底 (Base) 相對定址 ⇒ offset = operand 位址 − Base register 值.

(3) 索引 (Index) 相對定址 ⇒ offset = operand 位址 − Index register 值

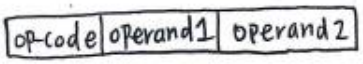


- 比較

	立即定址	直接定址	相對定址	間接定址
抓取 operand 之速度	快			→ 慢
Memory Access 次數	0	1	1 (先加法)	2
重定位修正	No	Yes	No	Yes

- 範例

EX：機器指令格式如下：



- operand 1及2皆採 Direct addressing mode.
- 若CPU有250條指令，Memory大小 = 1MB.
  求此指令長度佔？bytes.

Ans: CPU有250條指令，∴ op-code佔 8 bits ($\because 2^7=128$ 不夠，$2^8=256>250$)

∵ operand 1及2採直接定址，存放 memory address.

由於 Memory Size = 1MB = 2^{20} bytes. ∴ address 佔 20 bits.

∴ 指令長度 = 8 + 20 + 20 = 48 bits = 48/8 = 6 bytes.

17.電腦世代劃分

- 第一代：主要電子元件為真空管
- 第二代：〃 〃 電晶體
- 第三代：〃 〃 積體電路 (I.C. = Integrated Circuit)
- 第四代：〃 超大型積體電路 (VLSI = Very Large Scale IC)
- 第五代：以人工智慧AI為主要規劃

} 以電子元件作區分

18.摩爾定律

□ 摩爾定律 Moore's Law

Def：I.C.(半導體)每18個月可置入的元件數目加倍 (double)
或功能速度加倍.
或價格減半

Note：Turing 圖靈：電腦之父及AI之父

Turing Test：用以判斷电脑是否具有人的能力之測試

19.費林分類法（Flynn's Taxonomy）

- S : single ； M : mutiple ； I : instruction stream ； D : data stream

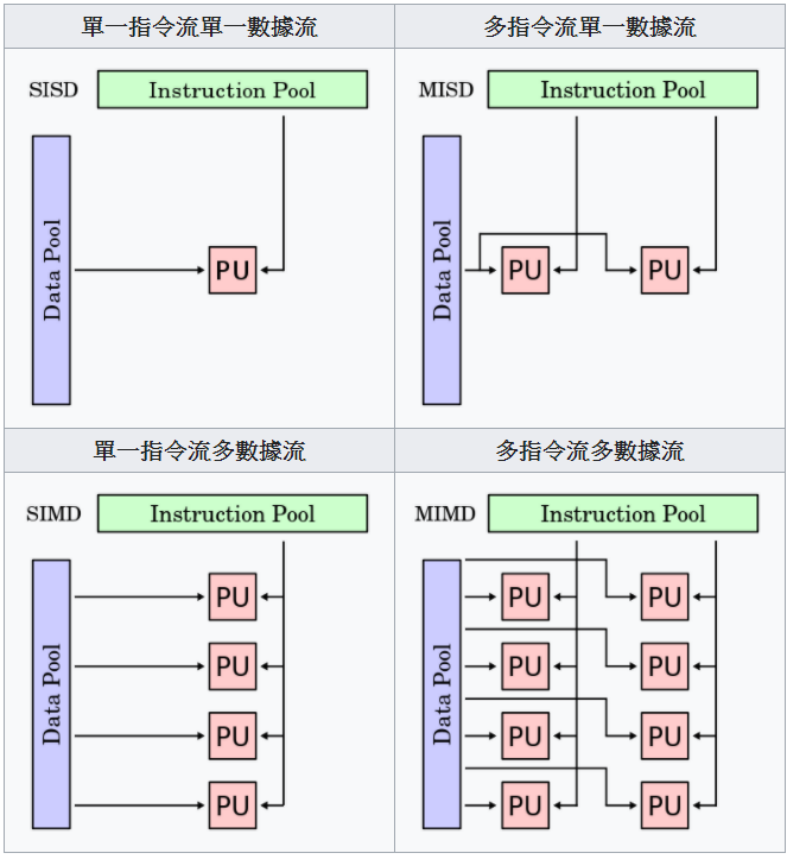
[1] SISD：單一指令處理單一資料來源，例：個人電腦PC，Von-Neumann - Computer

[2] MISD：不存在 e.g. 對A和B同時做+-*/ 工作站電腦.

[3] SIMD：單一指令處理多重資料流，例：陣列(向量)處理器 Array (vector) processor
e.g. Cray-I/II/III (Super Computer) 超級電腦

[4] MIMD：多個指令處理多重資料，例：Distributed system 分散式系統
Multiprocessor System 多顆CPUs系統 ⇒ 平行執行多個不同工作.

下圖有四種類型，"PU" 是指程序單元（processing unit）：



20.Software system 架構

- **System software** :
  (1) Operating System： 電腦資源分配，提供 user 易用介面及應用程式一直行之環境，確保安全 →Windows、MS-DOS、Mac OS、Linux
  (2) System Software： 協助 programmer 開發之工具
    A. Text Editor : assembler(組譯器)→組語，Compiler(編譯器)→C++、C，Interpreter(直譯器)→python、Java
    B. Linking Loader： 重定位修正、library 參考、外部符號參考修正

C. Debugger

- **Application Software**(解決特定問題之應用軟體)
  (1) 自行開發
  (2) Package 套裝軟體