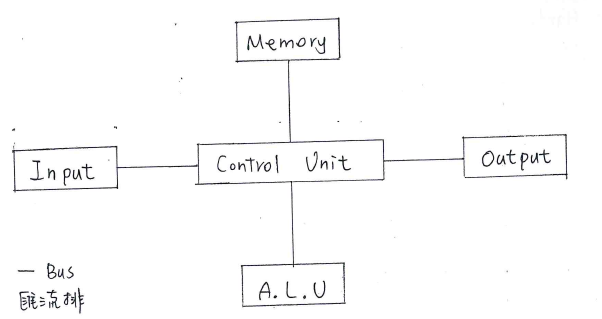
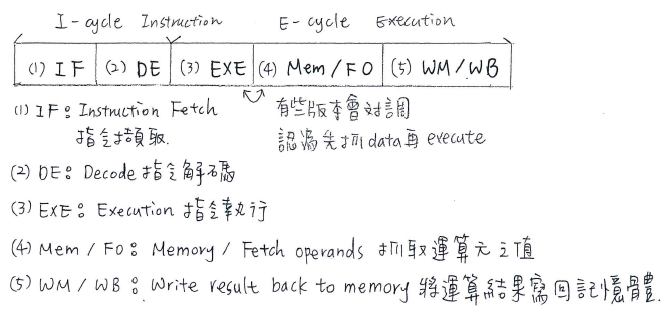
**.CH 1 基本概論**

1. **Hardware 5大單元(John von Neumann)** – input、output、memory、ALU、control (von neumann)
2. **Machine Instruction完成週期** – Instruction Fetch、Decode、Execution、Memory/Fetch Operands、Write Result/Write Back
3. 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)
4. **Bus匯流排** – Data Bus、Address Bus、Control Bus🡪 Von neumann bottleneck(馮紐曼瓶頸)
5. **Store Hierachy 儲存層次架構** – Reigister、cache、memory(volatile)；Flash Memory、SSD、Hard Disk、CD-ROM、Blue-Ray、Tape(Non-volatile)
6. **Register 暫存器** – IR、PC、MAR(address)、MDR(data)/MBR(buffer)、Accmulator、Base&Limit register
7. **Instruction Cycle and Register 指令週期與暫存器**
8. **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
9. **Caching概念** –先存慢再存快
10. **Memory** – Bootstrap Loader、RAM : DRAM(電容、電晶體)、SRAM(Flip-Flop)；ROM : PROM、EPROM、EEPROM、Flash ROM
11. **CPU介紹** – Control Unit + ALU + Register + Cache
12. **CPU指令集架構** – RISC(精簡，complex instruction set computer)、CISC(複雜，reduced instruction set computer )
13. **CPU衡量速度** – CPI(Cycle Per Instruction)、MIPS(Million Instructions Per Second)、MFLOPS、Clock Rate
14. **計量單位 : 儲存** – KB、MB、GB、TB、PB()、EB、ZB、YB
15. **計量單位 : 時間** – ms、us、ns、ps
16. **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
17. **電腦世代劃分** – 真空管、電晶體、IC(積體電路)、VLSI、AI
18. **摩爾定律** – 每18個月，電晶體增加倍
19. **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)
20. **系統架構** – System Software(OS、Text Editor、Linking Loader、Compiler)、Appliaction Software
21. 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



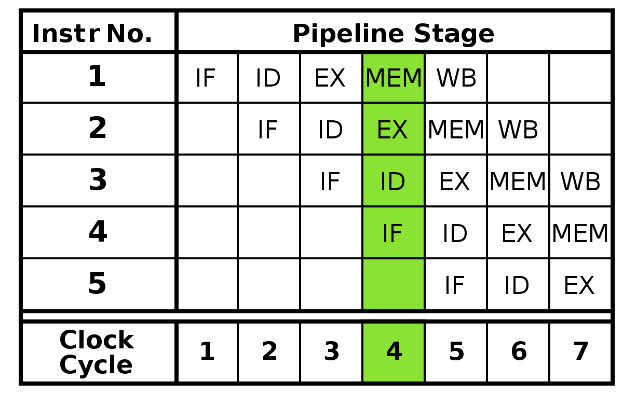
1. **Machine Instruction完成週期**



* **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

1. **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



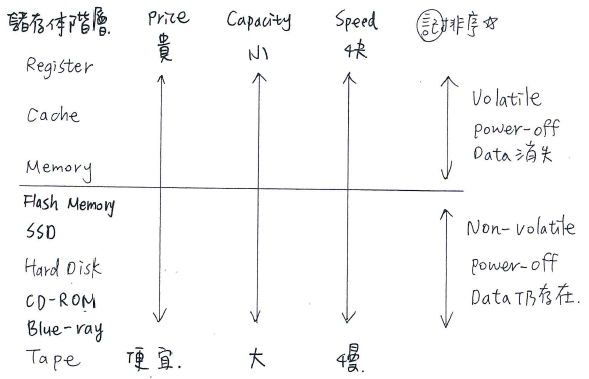
1. **Bus 匯流排**

* 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，如果記憶體有個Memory space(0~)(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 = ，27條

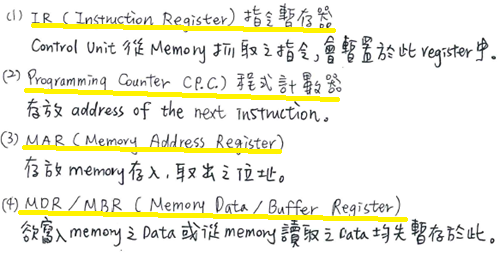
* **Von neumann bottleneck(馮紐曼瓶頸)** : 系統速度受限於bus傳輸效能

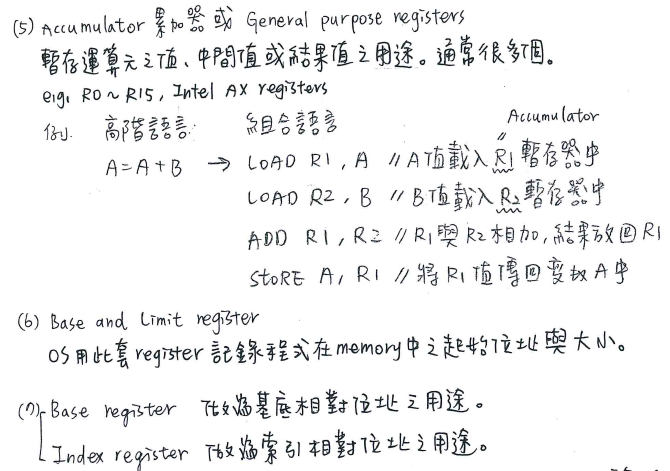
1. **Store Hierarchy 儲存層次架構**



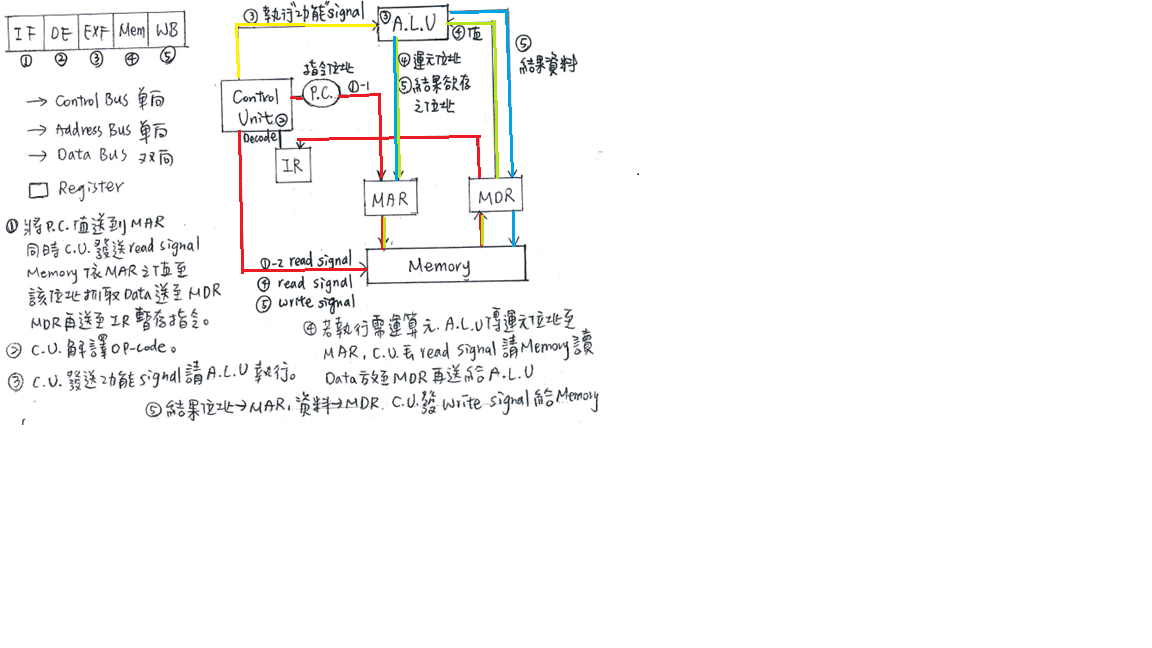
1. **暫存器(Register)**

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



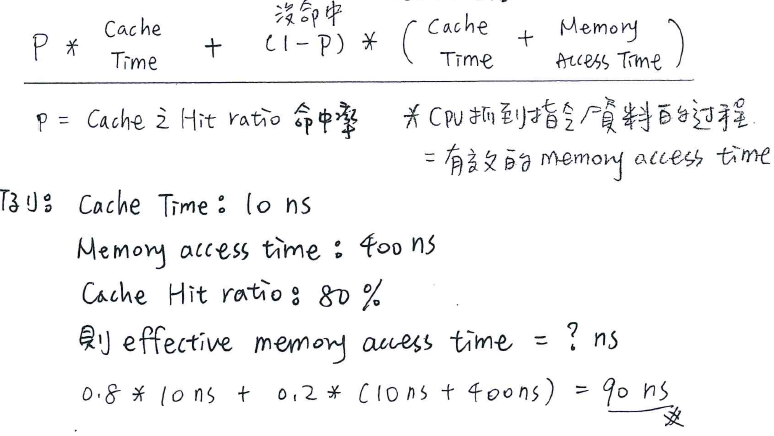


1. **指令週期與暫存器**



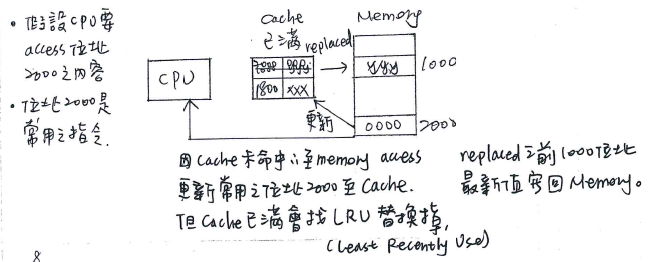
1. **Cache / Cache Memory 快取記憶體**

* 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



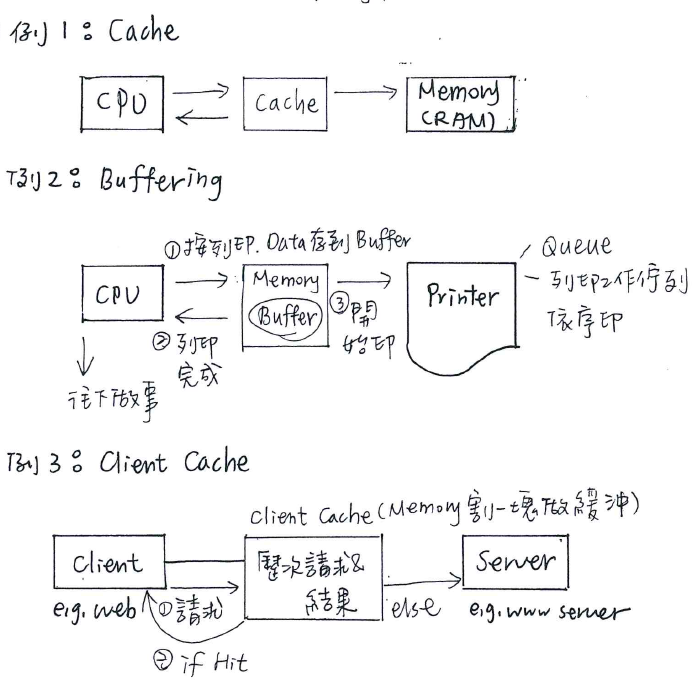
* Cache內容何時寫回memory : 寫回cache但未寫回memory 資料不一致

1. **Write through(即刻寫回)** : 寫回cache時，也寫回memory
2. 優 : 保證一致
3. 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
4. **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



1. **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



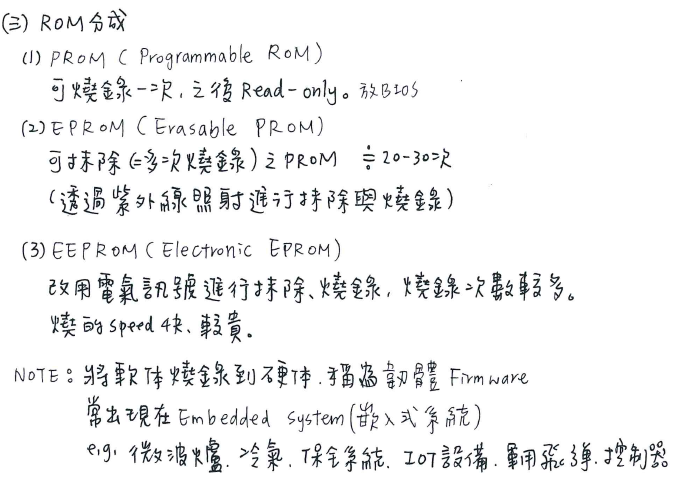
1. **Memory**

|  |  |
| --- | --- |
| 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 |



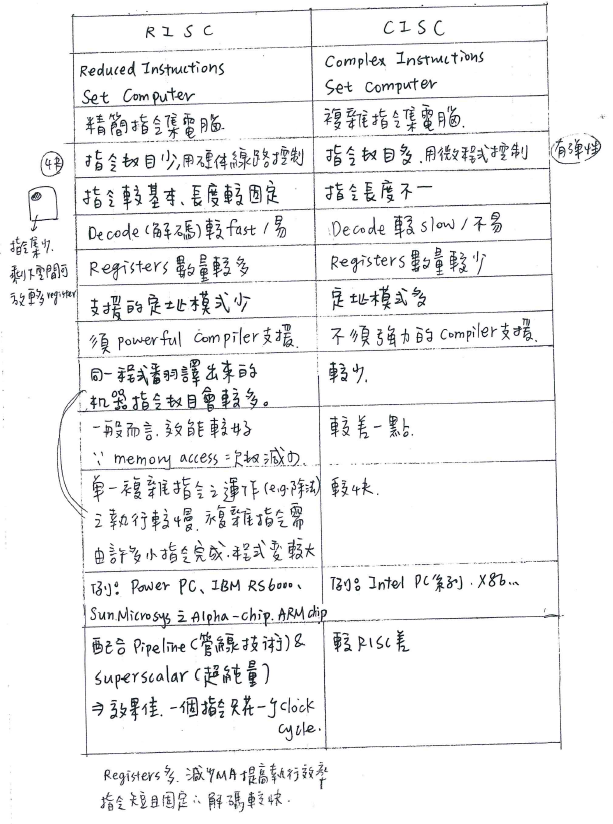
[補充]

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

1. **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

1. **CPU設計之兩大策略 : RISC()、CISC**



1. **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)

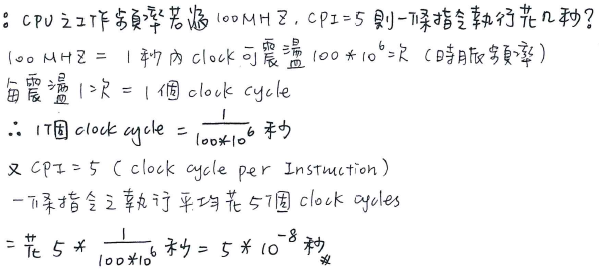
1. Cpu速度5 MIPS，則一條指令花幾秒 => 秒 (一秒 5\*10^6條指令)

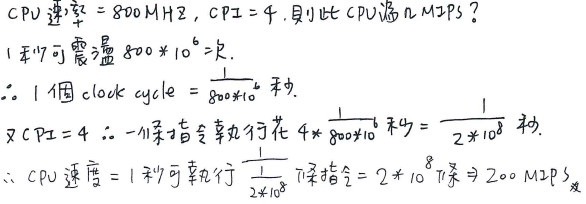
公式 :

1. 平均一條花 秒，則CPU幾n MIPS => = = => 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 = , ex. 100MHz => =
* 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 =

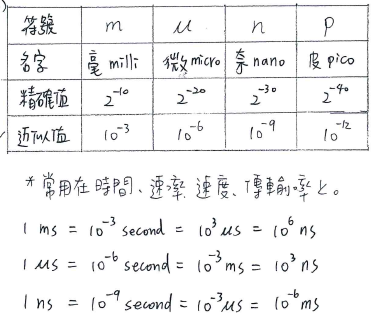


s

1. **計量單位 – 儲存**

* 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

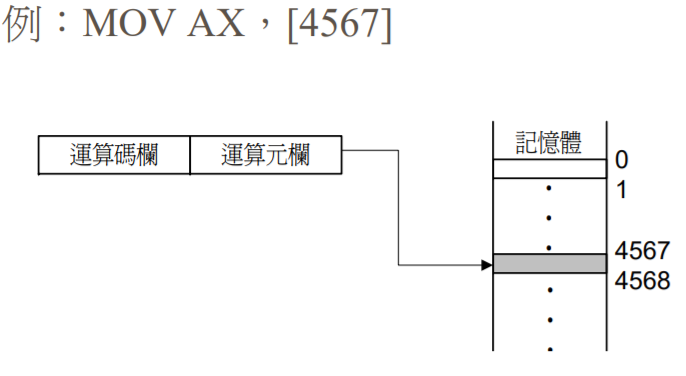
1. **計量單位 – 時間**

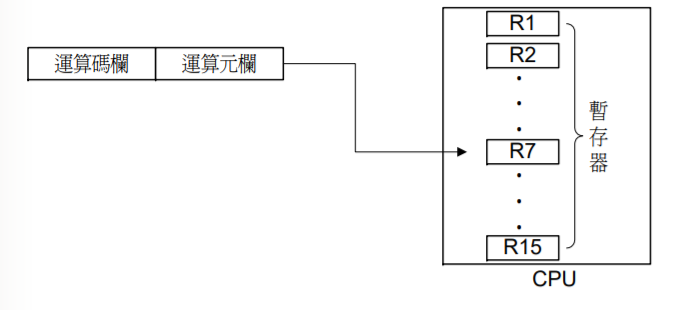


* Cpu 速度 = 10 MIPS，一條指令多少ns ?
* 秒 = 秒 = 0.1 us = 100 ns

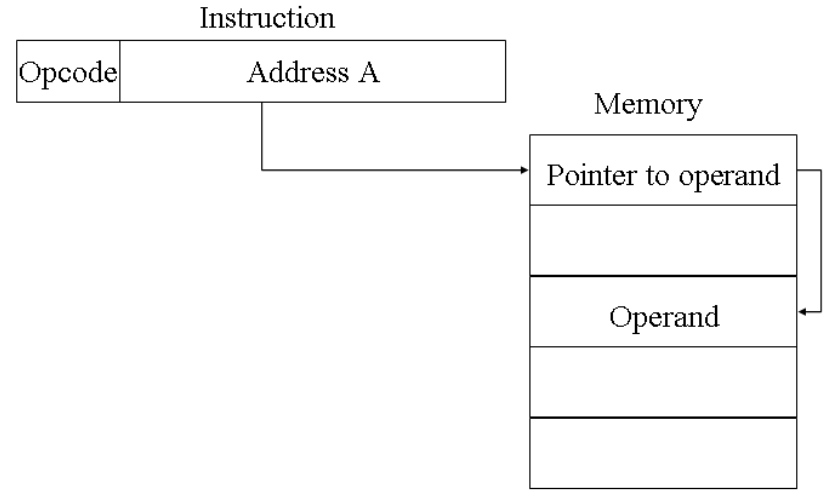
1. **定址模式 (Addressing Mode)**

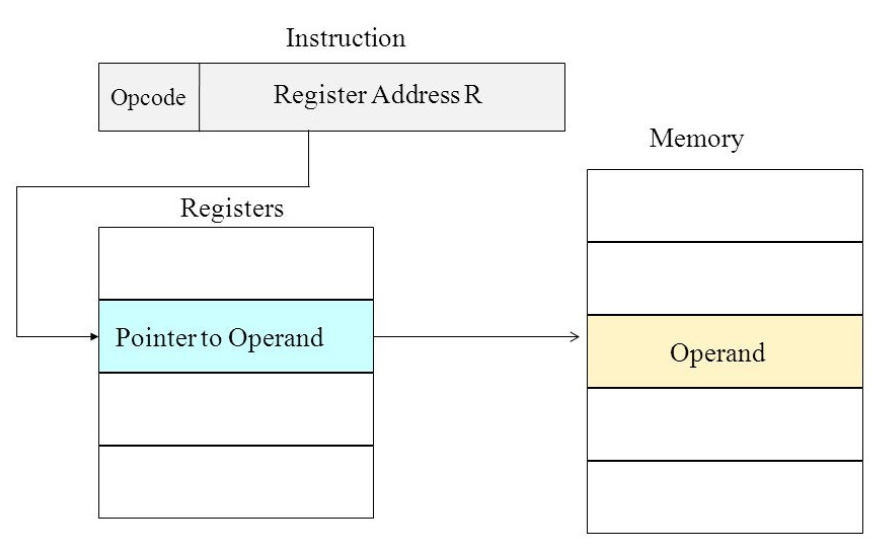
* **直接定址(direct address)**
* 指令的運算元欄內的值表示資料存放於記憶體的實際位址 (有效位址，Effective Address)，故需要做一次的記憶體讀取，以取得所需之資料
* 優 : operand值域放在記憶體，較不受限制
* 缺 : 與立即定址相比，需存取memory一次，速度較慢；程式執行起始位址改變，運算元欄需relocation (linking loader負責修正)



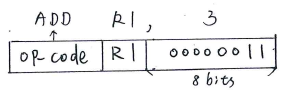


* **間接定址模式(indirect address)**
* 指令的運算元欄內的值為存放operand所在地址之值的位址
* 優 : 適合跳躍擷取
* 缺 : 需兩次存取得operand值，程式執行起始位址改變，運算元欄需reloaction(linking loader負責修正)

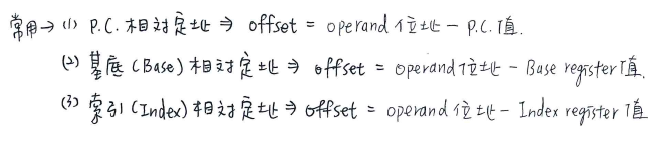


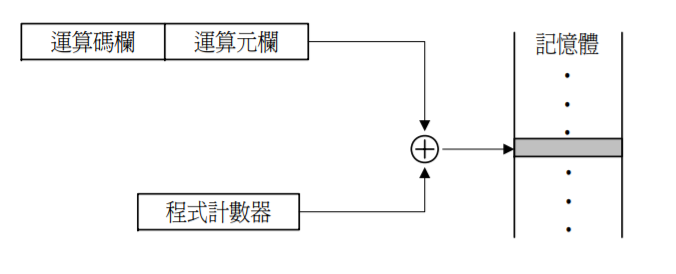


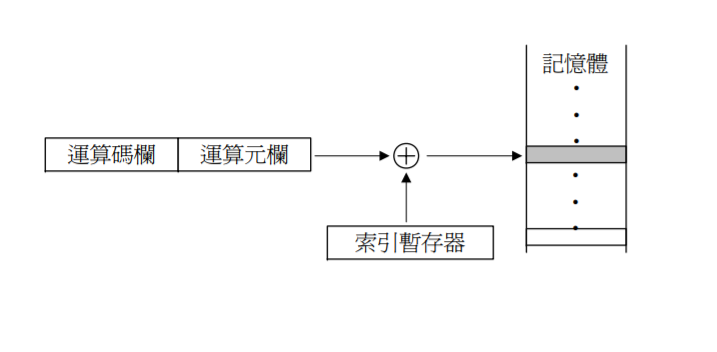
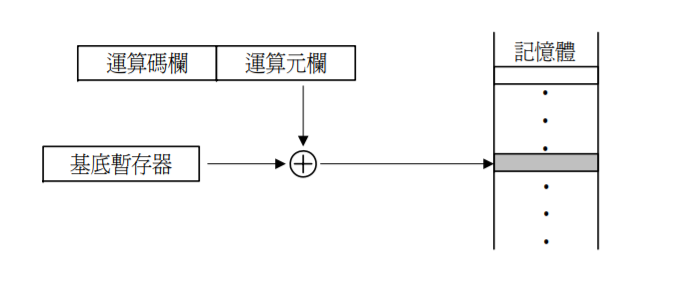
* **立即(絕對)定址(Immediate；absolute operand)**
* Operand欄位為欲運算之值



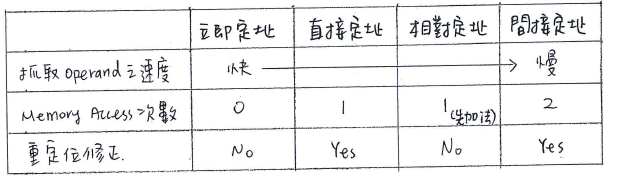
* 優 : 速度最快，不須額外memory access，不須relocation
* 缺 : 值域受限
* **相對定址模式(relative address)**
* Operand放Operand的偏移量(offset)，不論從何執行，偏移量都一樣
* 優 : 不須做relocation
* 缺 : 抓operand值須先做加法，速度稍慢，offset不可太大，萬一 超過會採直接定址



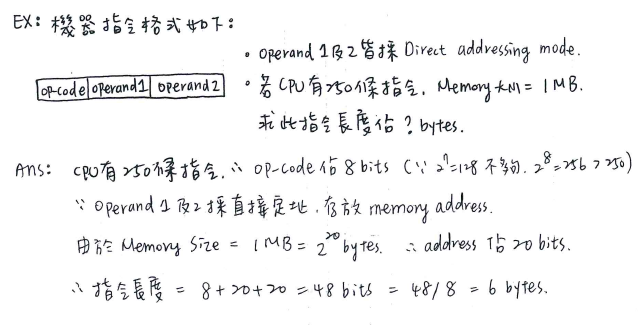




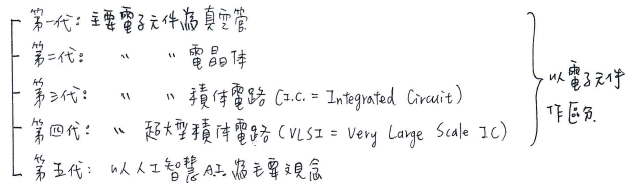
* 比較



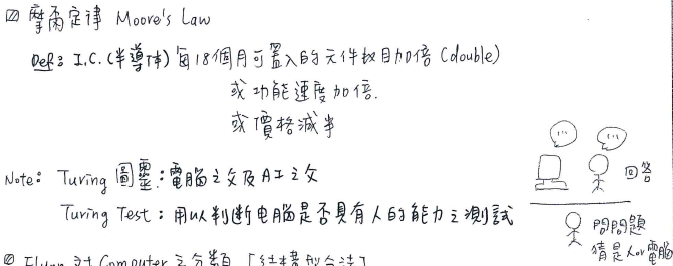
* 範例



1. **電腦世代劃分**

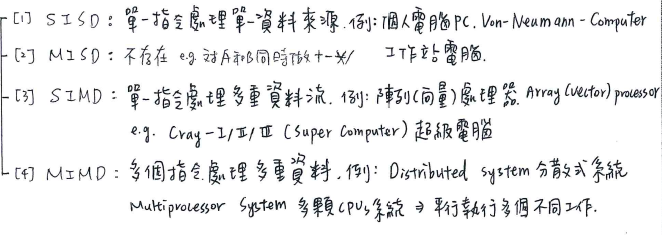


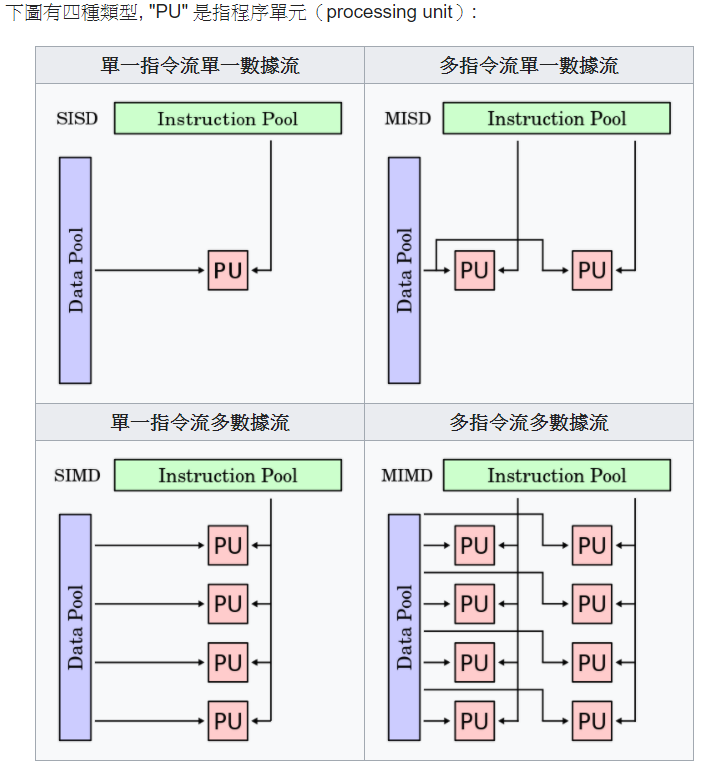
1. **摩爾定律**



1. **費林分類法（Flynn's Taxonomy）**

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

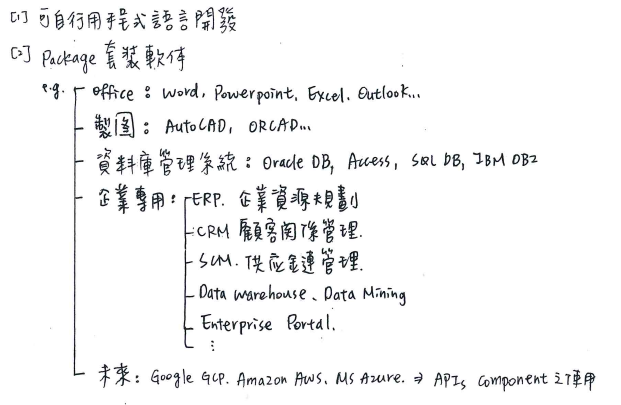




1. **Software system 架構**

* **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

1. Text Editor : assembler(組譯器)🡪組語，Compiler(編譯器)🡪C++、C，Interpreter(直譯器)🡪python、Java
2. Linking Loader : 重定位修正、library參考、外部符號參考修正
3. 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