微算機原理與應用

基本觀念(一)

授課教師：雲林科技大學 張慶龍 老師

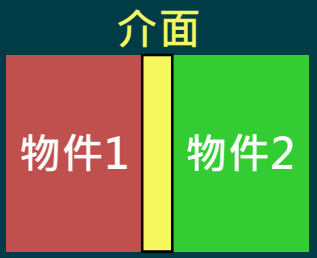
單元學習目標與大綱

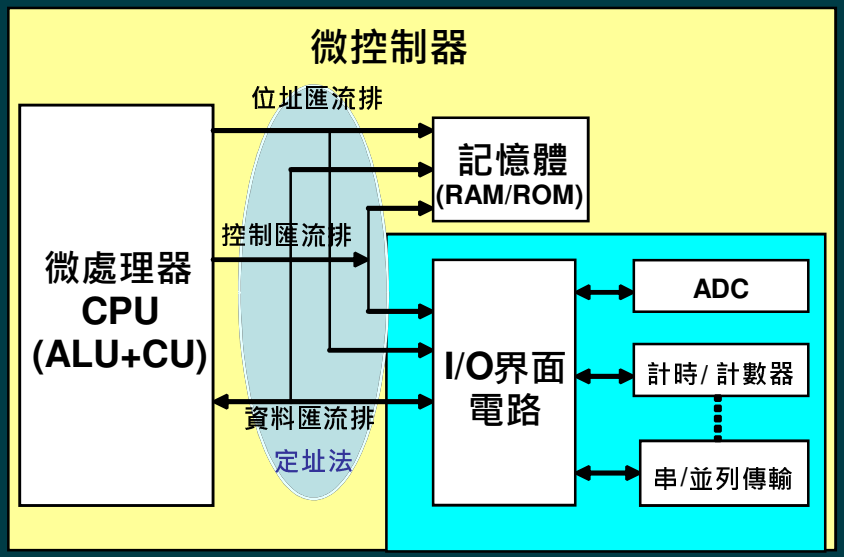
- I/O與周邊
- I/O之資料搬移
- 中斷運作原理

Part 1
I/O與周邊

I/O介面

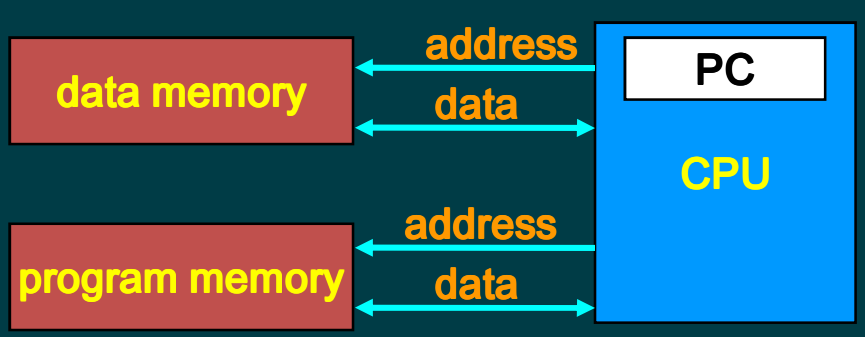
- 什麼是介面
- 什麼是周邊裝置(Peripheral device)
  - 以CPU的角度來定義
- 什麼是I/O介面
  - 以CPU的運作來定義
  - 除了記憶體外之周邊裝置皆稱為I/O裝置
- CPU與周邊之溝通方法
  - 匯流排系統
    - 位址匯流排
    - 資料匯流排
    - 控制匯流排





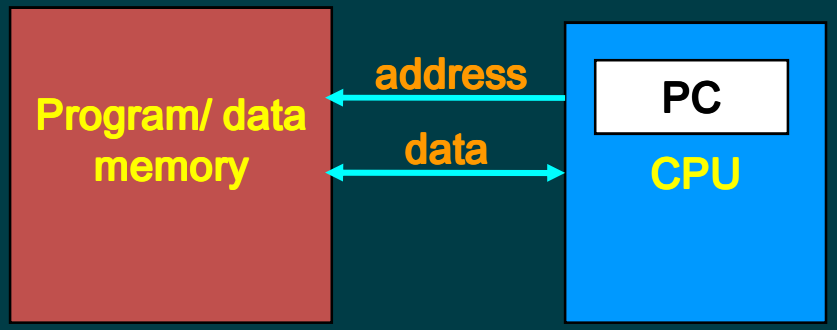
CPU與記憶體之關係

- 哈佛架構(Harvard architecture)
- 汎紐曼架構(von Neumann architecture)



Harvard architecture

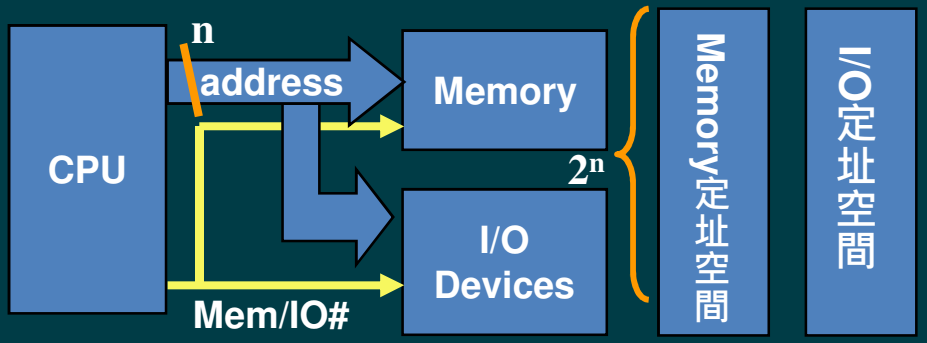
HT66Fxx屬於此種架構



von Neumann architecture

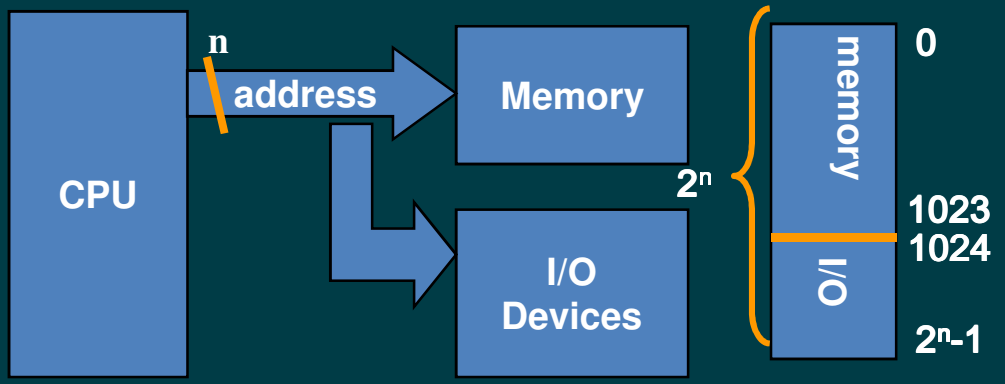
I/O定址法

- I/O mapped I/O
  - I/O與記憶體有各別獨立之定址空間
  - 對I/O操作與對記憶體操作的指令是不同的
  - 利用硬體訊號Mem/IO#來區分是要對I/O操作或記憶體操作
  - MOV AX, [10]
  - IN AX, [10]

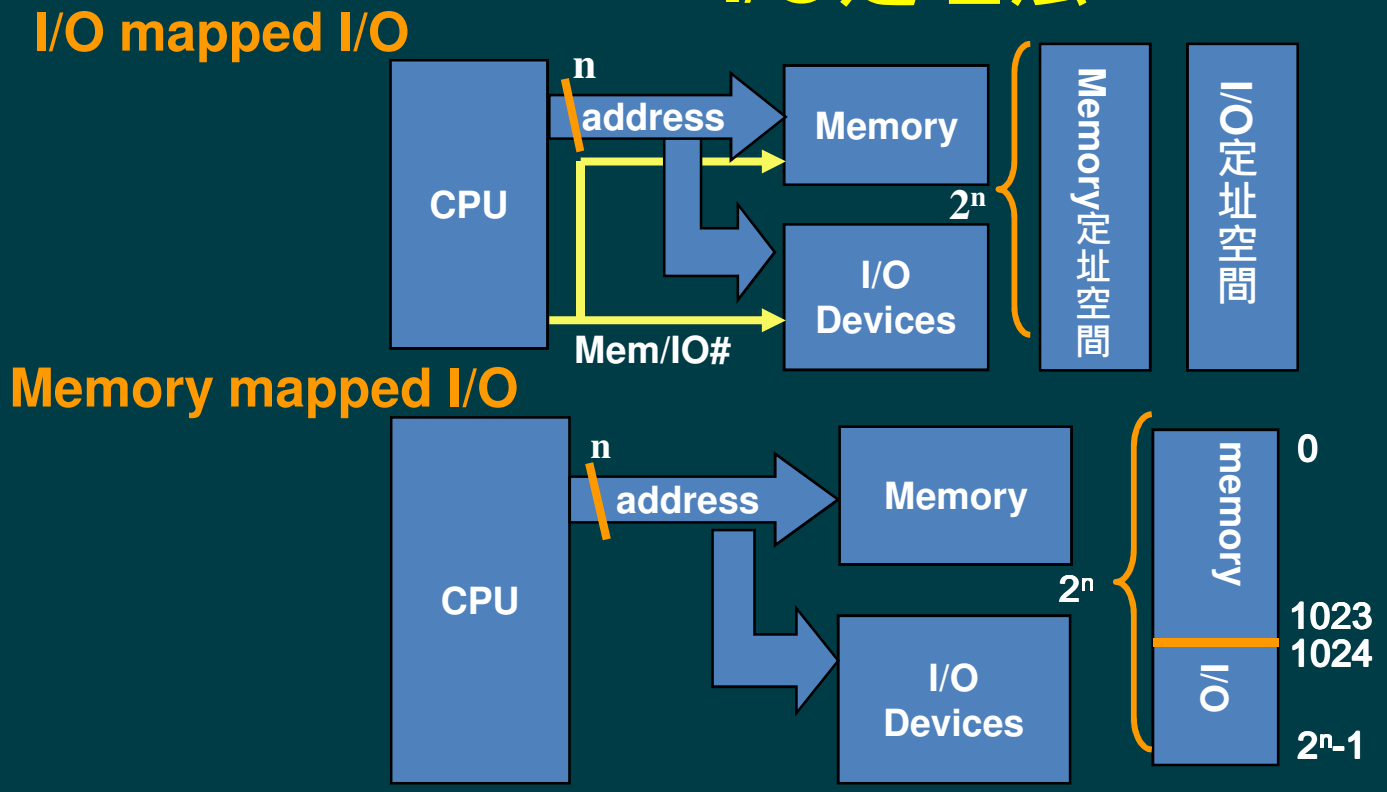


I/O定址法

- Memory mapped I/O
  - 使用相同指令來對I/O與記憶體存取
  - 由CPU產生的位址範圍決定是對I/O操作或對記憶體操作
  - MOV AX, [10]
  - MOV AX, [1024]



I/O定址法


I/O mapped I/O
CPU
n
address
Memory
I/O
Devices
Mem/IO#
2n
Memory定址空間
I/O定址空間
Memory mapped I/O
CPU
n
address
Memory
I/O
Devices
2n
memory
I/O
0
1023
1024
2n-1

CPU與I/O之溝通

- Polling

```
polling()
{
   while (is device A ready?);
        讀取device A資料;
}
```

- Interrupt

Device A
中斷訊號

```
main()
{
    while(1)
      {
      做主程式正常做的事情;
      }
}
```

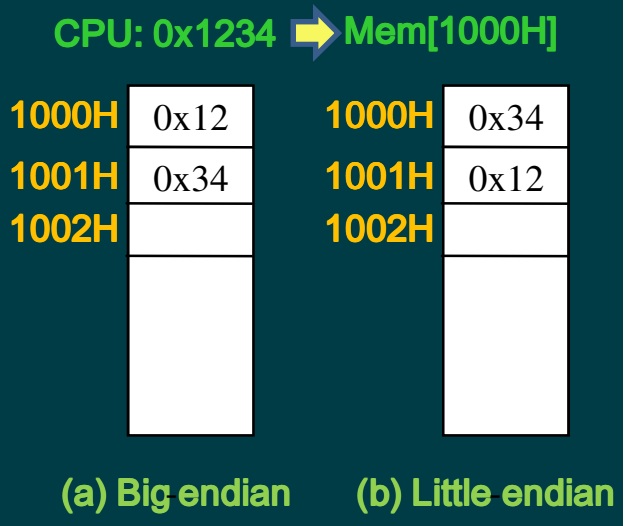
```
Device_A_ISR()
{
      讀取device A資料;
}
```

CPU指令集與效能

- The Instruction Set of Microprocessor
  - RISC: Reduced Instruction Set Computer
  - CISC: Complex Instruction Set Computer
- Performance Measurement
  - MIPS: Million of Instruction Per Second
  - EX: 500 MHz工作頻率的CPU, 若其執行一個指令平均需2個Clocks, 則此CPU的MIPS值為何?

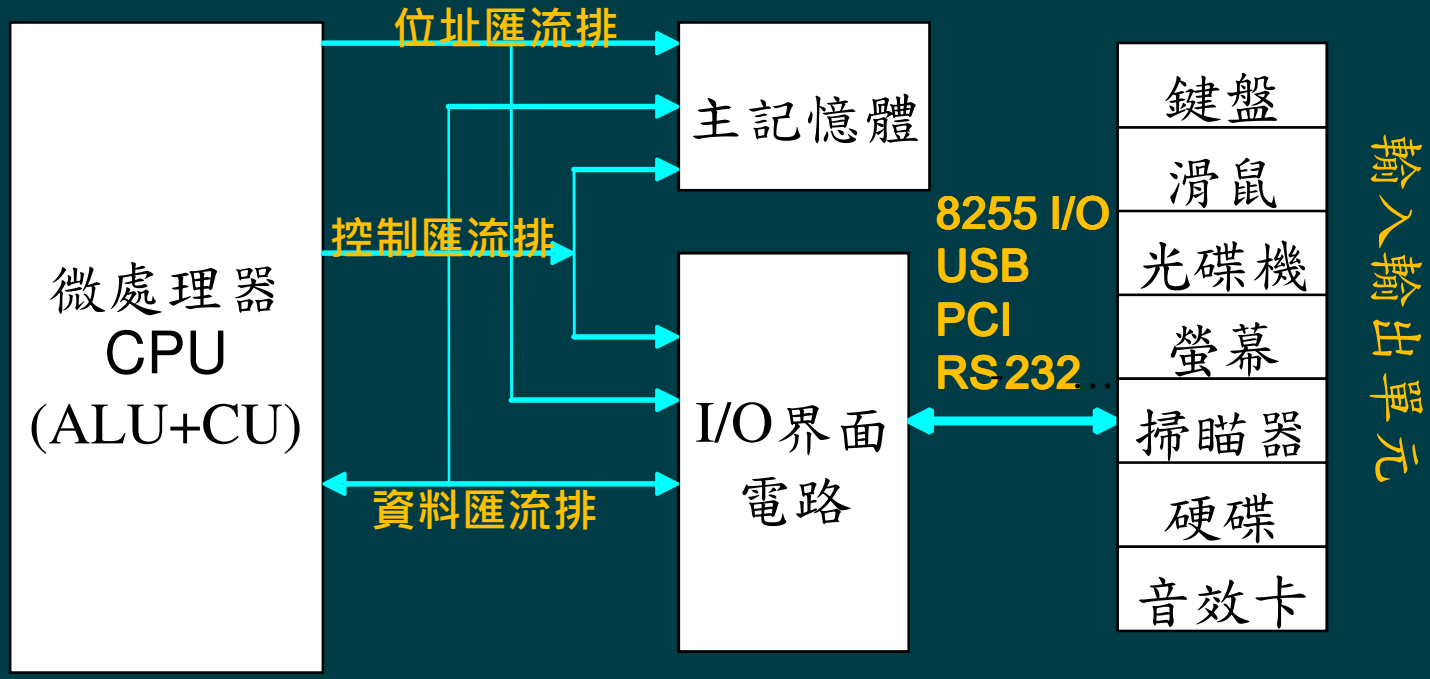
CPU將資料寫入記憶體

- Big-Endian CPU：CPU寫一筆資料到某一記憶體位址時，是將高位元組資料(High-byte)寫到記憶體之低位址(high byte first)。
- Little-Endian CPU：CPU寫一筆資料到某一記憶體位址時，是將低位元組資料(Low-byte)寫到記憶體之低位址(low byte first)



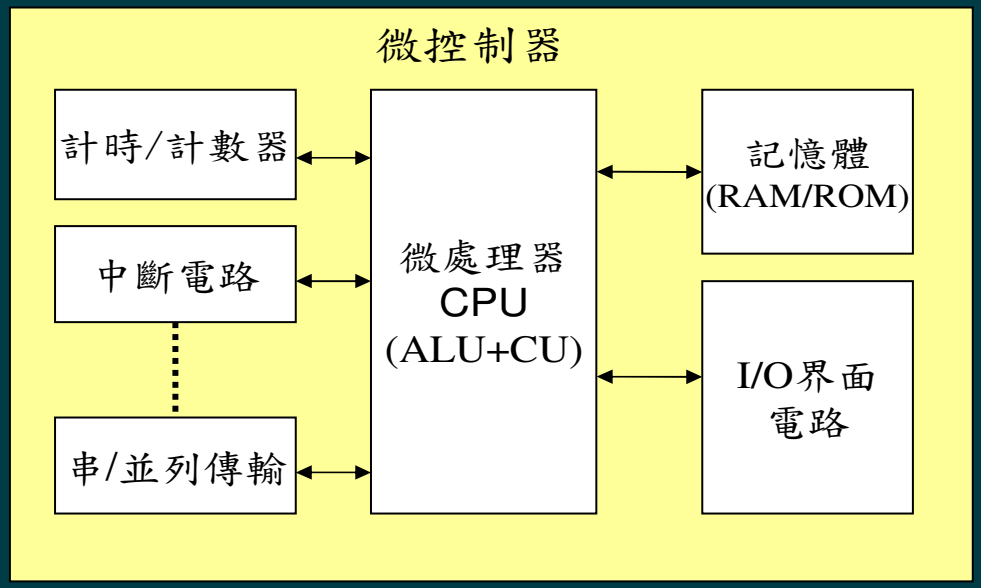
計算機系統

- 5-unit
  - ALU, Memory, Input, Output, Control



Microcontroller or Single Chip

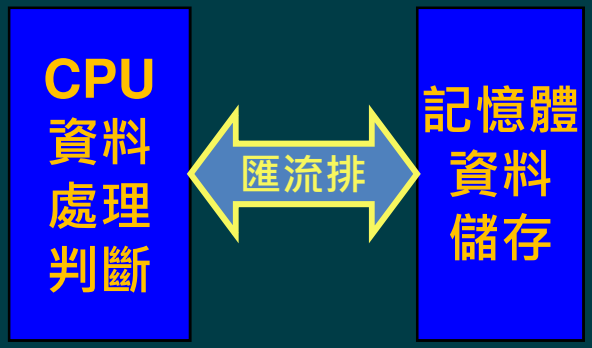
- 微處理器(Microprocessor)與微控制器(Microcontroller)最主要的差異:
  - 微控制器整合了ROM or Flash Memory(儲存指令)/RAM(儲存資料)
  - 微控制器整合了週邊裝置➔又稱為單晶片(single chip)



Part 2
I/O資料搬移

I/O資料搬移

- 現有的計算機架構
  - CPU負責處理資料與判斷
  - 所處理的資料是放在記憶體
- CPU處理資料前
  - 需將資料由記憶體搬至CPU
  - I/O周邊的資料需先搬至記憶體，CPU才可做後續處理
- CPU處理完資料
  - 需將資料寫回記憶體
- CPU與記憶體間的資料搬移是仰賴匯流排(Bus)

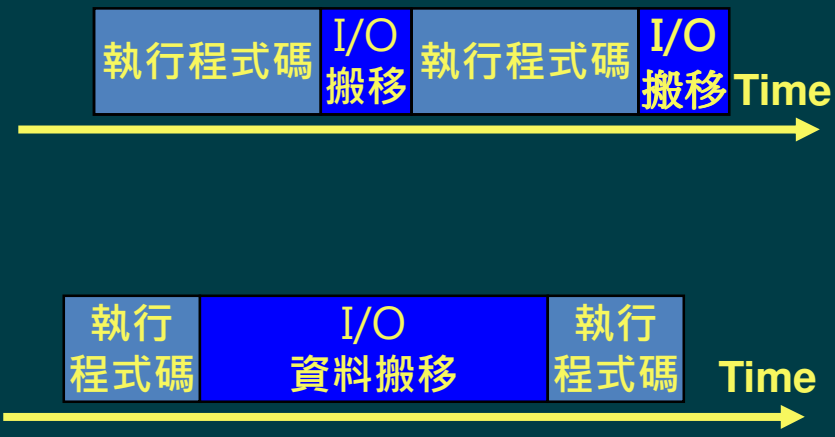


Task之功作分類

- 一個嵌入式系統是由很多Tasks(或稱Processes)所組成
  - 這些Tasks互相分工合作，完成系統功能
- 每個Task依其工作內容，可分為兩部分
  - 資料處理部分(佔用CPU時間)➔此部分不會使用到匯流排
  - 資料搬移部分(佔用I/O匯流排)➔此部分不會使用到CPU

Task之工作行為

- Task依其資料處理與資料搬移比例可分為
  - CPU bound
    - 資料處理佔了大部分時間
    - 即I/O大部分在idle
  - I/O bound
    - 資料搬移佔了大部分時間
    - CPU大部分在idle

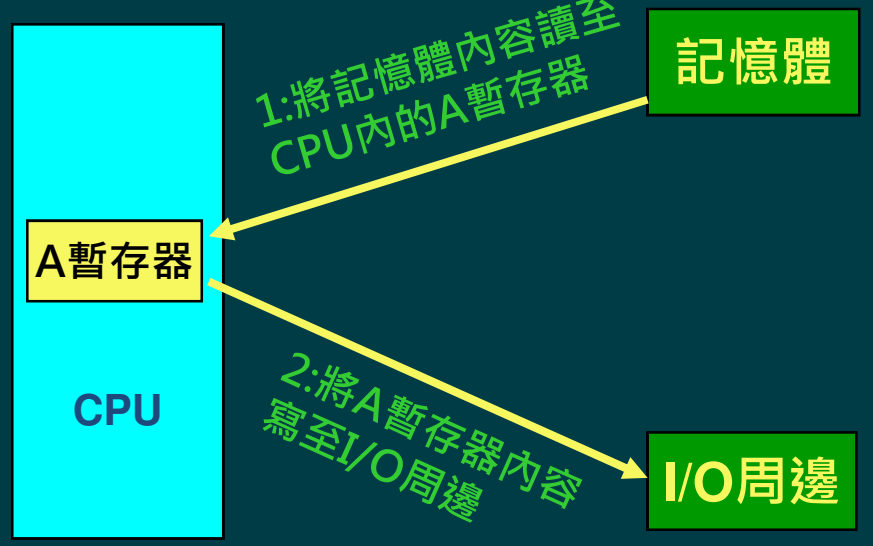


資料搬移方法

- **CPU-based 資料搬移**
  - 仰賴CPU執行資料搬移指令
  - 在資料搬移期間，皆會佔用CPU時間
- **DMA-based資料搬移**
  - 有一額外硬體DMAC (Direct Memory Access Controller)負責做資料搬移工作
  - CPU只要做DMAC相關設定，其餘資料搬移工作皆有DMAC負責
  - 資料搬移過程，不佔用CPU時間

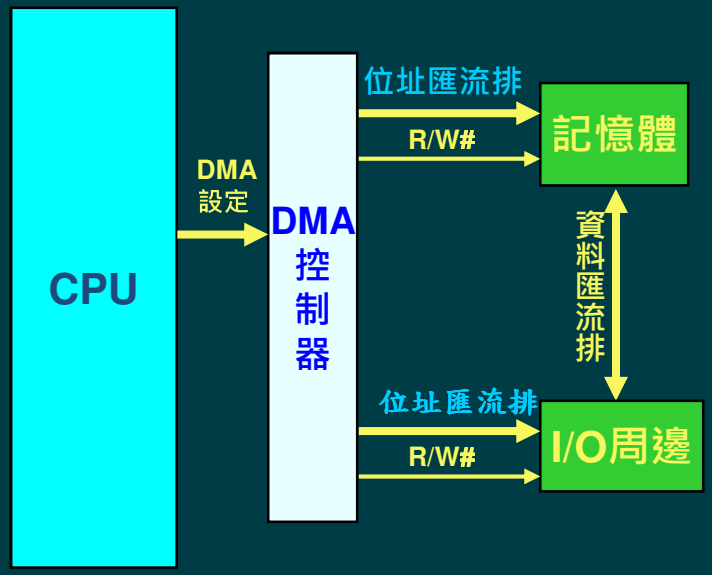
CPU-based資料搬移

- 利用CPU將資料由記憶體搬至I/O
  - MOV A, [Mem-address]
  - MOV [IO-address], A
  - 適合少量之資料搬移



DMA資料搬移

- 由DMAC將資料由記憶體搬至I/O周邊
  - CPU設定DMA控制器
    - 記憶體起始位址
    - 資料長度
    - 搬移方向(Read/Write)
    - I/O位址
    - 啟動搬移
  - DMA控制器依設定開始搬資料
    - 產生記憶體位址
    - 產生記憶體之Read/Write訊號
    - 產生I/O位址
    - 產生I/O之Read/Write訊號
  - 適合大量之資料搬移

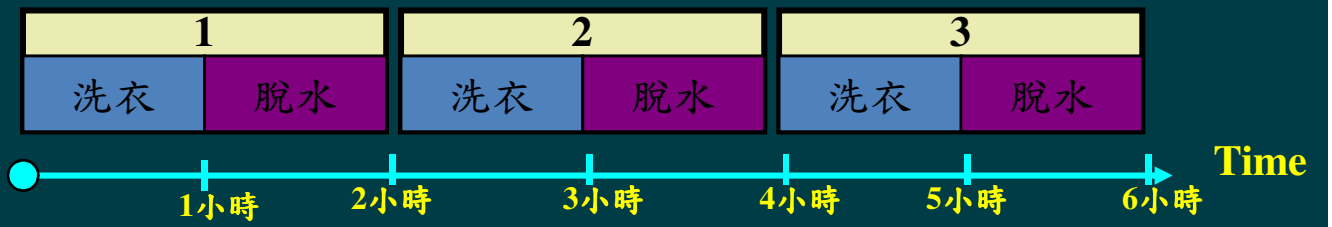


Part 3
中斷 Interrupt

平行處理與管理(Pipeline)

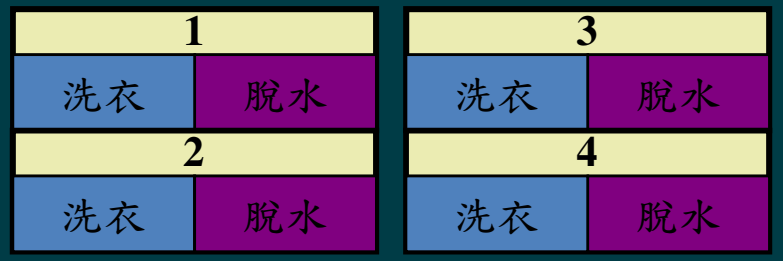
- 洗衣服之工作可分成2部分:洗衣、脫水

1台
(洗衣+脫水)機



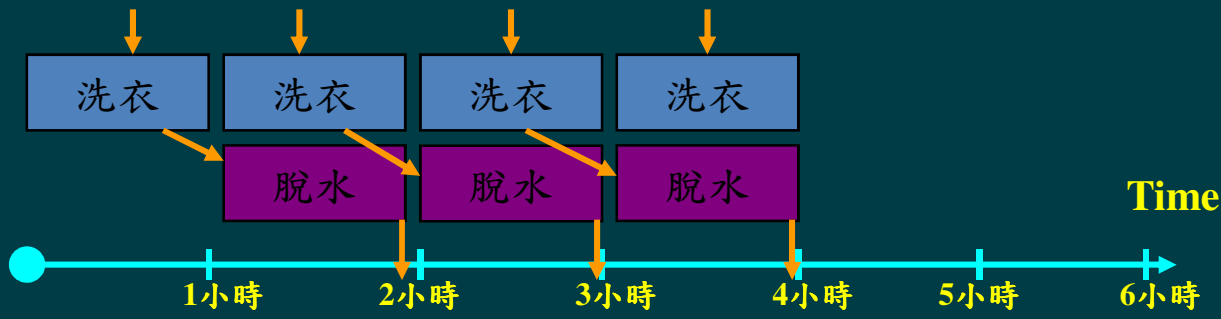
- 平行處理

2台
(洗衣+脫水)機



- 管線處理 Pipeline

1台洗衣機 +1台脫水機



CPU如何得知周邊裝置之狀態

- 輪詢(Polling)
  - CPU主動去問周邊，是不是有需要我幫忙處理的?
- 中斷(Interrupt)
  - 周邊裝置透過中斷訊號，主動通知CPU
  - CPU收到中斷訊號後
    - 1. CPU完成目前指令
    - 2. CPU將PC push to Stack(Return address)
    - 3. CPU將此中斷對應的中斷向量(Interrupt Vector)寫至PC(跳去ISR執行)
      - 中斷向量即是服務此中斷之中斷服務副程式(Interrupt service routine, ISR)位址

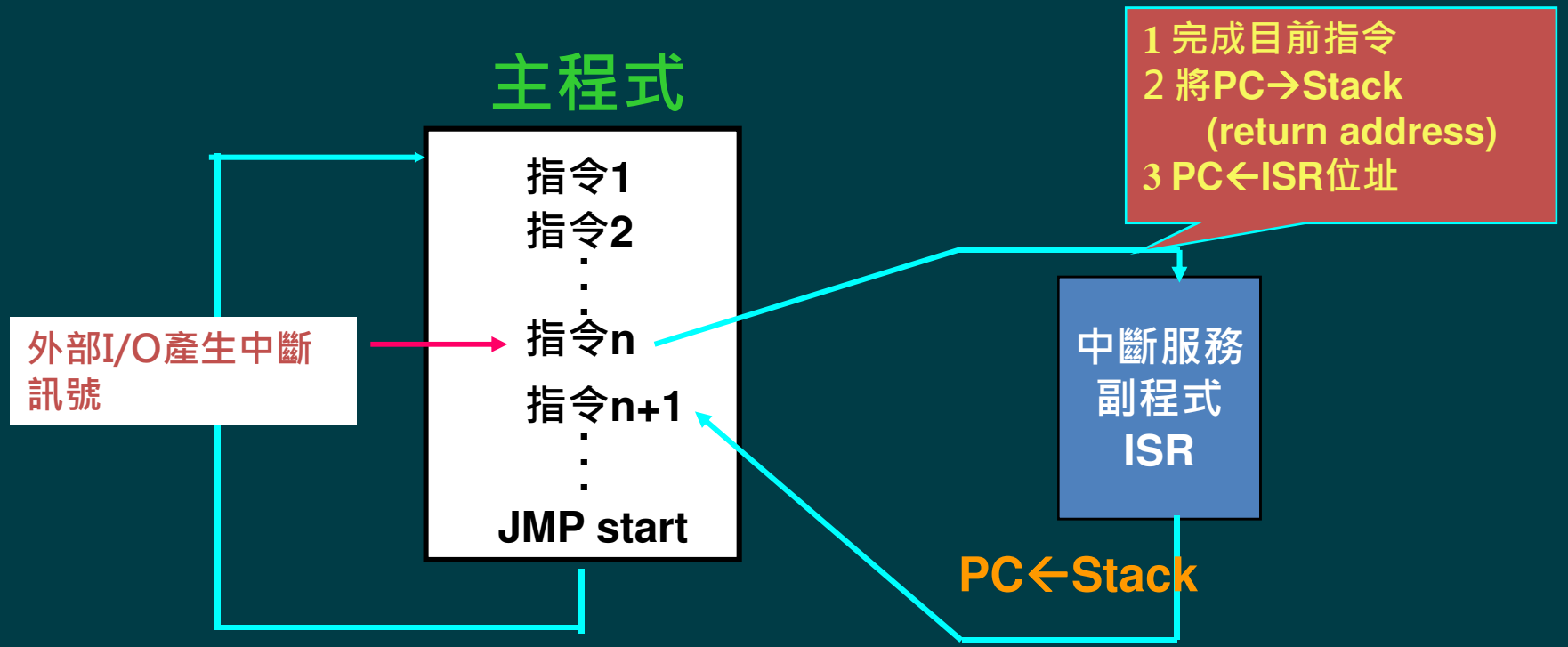
有那些周邊裝置之情況

- 滑鼠移動
- 鍵盤有按鍵輸入
- 印表機沒紙張了
- 感測訊號輸入
- …

中斷的種類

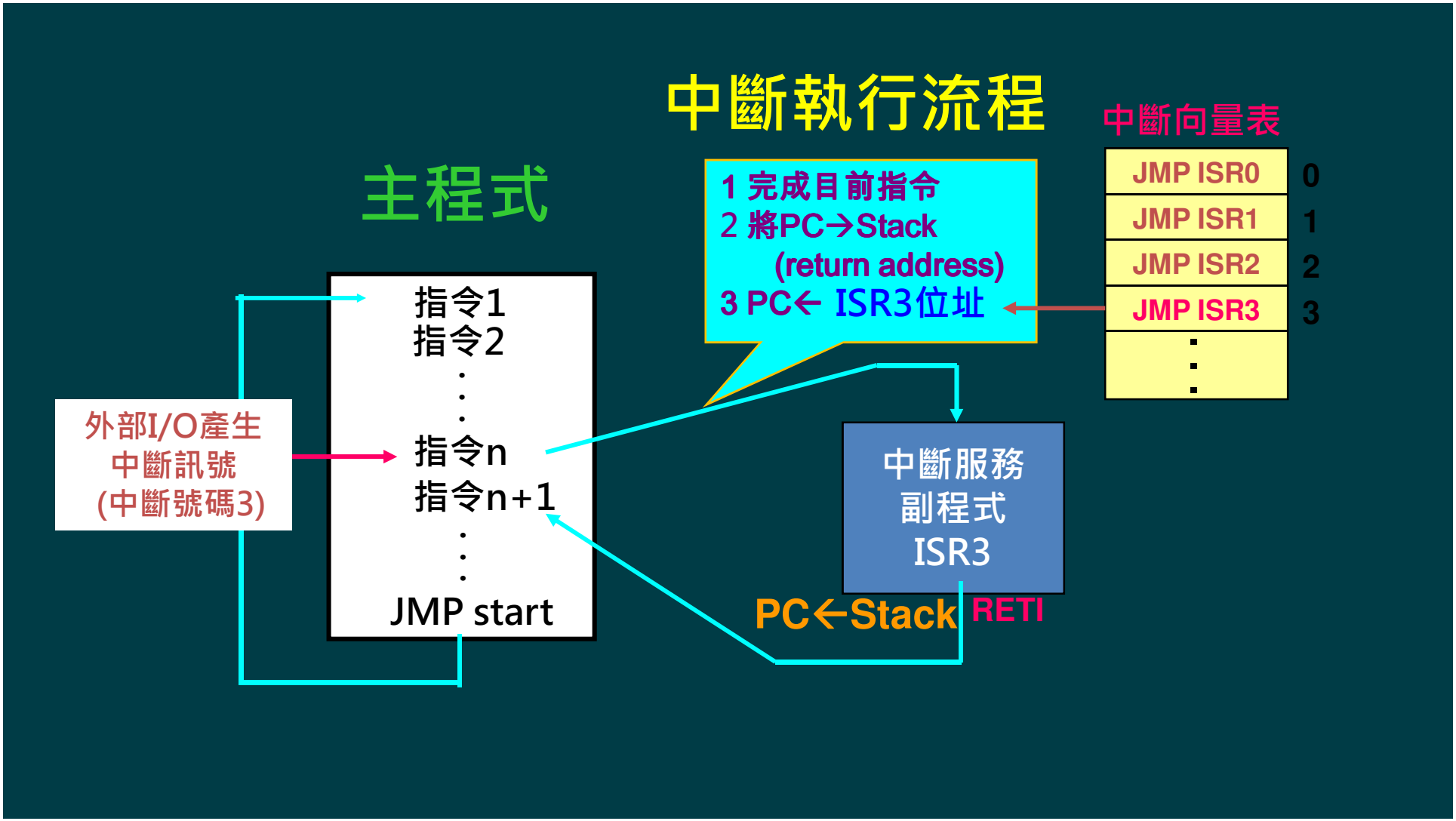
- 外部中斷
  - 外部周邊裝置之事件(event)產生的中斷
  - 如: 網路卡收到封包之事件
- 內部中斷
  - 內部元件事件或例外狀況(Exception)產生的中斷
  - 如: 執行到一個錯誤指令
- 軟體中斷
  - 利用軟體指令產生的中斷

具中斷功能之程式執行流程



中斷向量表(Interrupt Vector Table)

- CPU將每個中斷訊號加以編號
- 當有中斷發生時
  - 1. PC → Stack (儲存return address)
  - 2. CPU根據中斷編號至中斷向量表取得中斷服務副程式(ISR)位址，並設定給PC (program counter) (即跳至ISR執行)


中斷執行流程
主程式
中斷向量表
1 完成目前指令
2 將PC→Stack
(return address)
3 PC← ISR3位址
JMP ISR0
JMP ISR1
JMP ISR2
JMP ISR3
0
1
2
3
指令1
指令2
指令n
指令n+1
JMP start
外部I/O產生
中斷訊號
(中斷號碼3)
中斷服務
副程式
ISR3
PC←Stack
RETI

ISR程式

- 中斷訊號是隨時會發生
- 主程式用到的暫存器與變數，若ISR也會用到，則必需要加以保護

Ex:

中斷訊號 →

主程式

```
MOV  A, 10
MOV  PA, A
```

A=10

PA=50

ISR

```
...
MOV  A, 50
...
RETI
```

A=50

- 如何保護
  - ISR程式一開始要將共用的暫存器與變數儲存起來
  - ISR程式在執行RETI前，要將所改變的暫存器與變數恢復原狀

ISR

```
儲存 Acc
儲存 Status
儲存 相關變數
指令1
.
.
指令n
回存 Acc
回存 Status
回存 相關變數
RETI
```