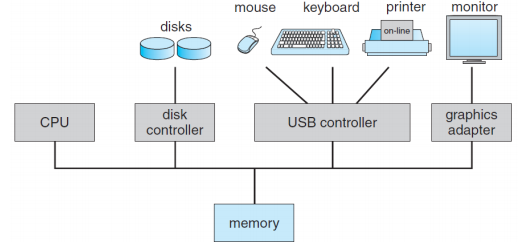
概述

現代電腦會使用bus連結元件與memory、CPU (目的是CPU、devices同時完成memory cycle)



Device controller

其中每個device都有其device controller，負責控制

Device controller有自己的

local storage buffer暫存資料

register 做控制(status reg)、搬運資料(Data reg)用途 (register就需要C PU更動)

Device controller主要任務是交換 local storage buffer 與 IOdevice的資料

Device controller 如今可多功能化(抽象化)，如USB插槽

Device driver

OS對每一個Device controller都有一個device driver，並且會用uniform interface與它們溝通，提供OS資源

\*為甚麼要使用Dc與Dd?

可以與CPU運算平行化、不衝突的使用memory，完成memory cycle

接下來將講解 interrupt機制、儲存結構 與 IO結構

1. Interrupt

讀取IO過程

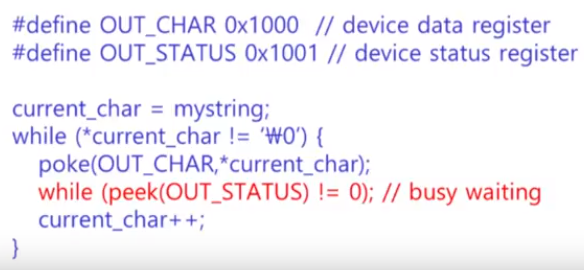
**OS->device driver -> device controller**

device driver使用 device controller的register，要求輸入IO (如:讀一個字元)

**device controller <-> IO**

device controller的local buffer就會與IO交換資料

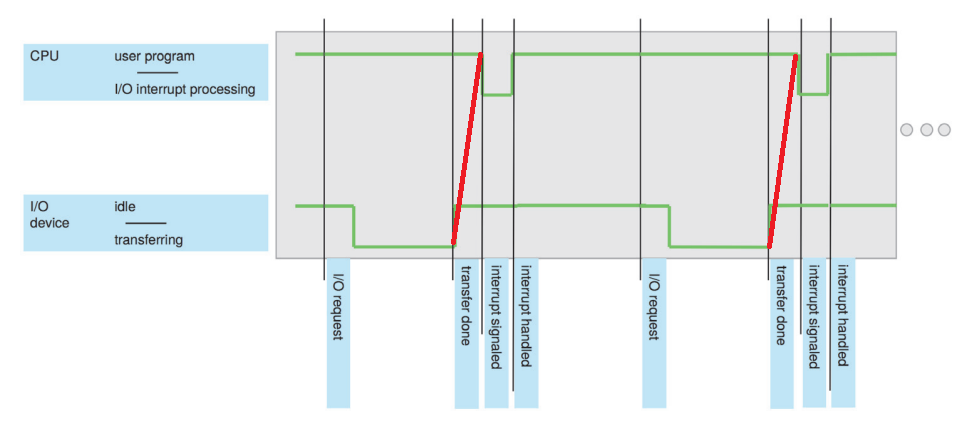
問題: 如何 device controller 通知OS device driver 工作完成?

如果沒有interrupt? 

CPU就需要等到IO完成才能運作

Sol: 使用interrupt，Interrupt是OS與hardware溝通的利器

概念上



\*紅線即為interrupt signal，會將CPU從其他工作切換成IO工作

\*signal發出會經一段時間才送達

\*CPU會發給IO端 Ioreuqest

\*CPU會有interrupt handler(像event handler)，接收interrupt

\***hardware、software發生錯誤也會引起interrupt**

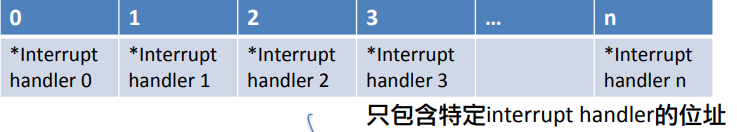
\*hardware interrupt=>signal ,

software interrupt=>trap (error 或 system call(IO一定要透過system call發出 trap，否則不安全))

實作上

1.會使用interrupt vector，為一個int – function address 的map

也就是一個寫死的array



\*Interrupt如此設計是追求速度

\*CPU端會有interrupt request line，每執行完一道指令就檢查有沒有interrupt

\*vector應越小越好，且越重要擺前面(在記憶體中也是在最前面)

=>interrupt chain

Interrupt執行步驟

1.(IO結束)device controller發生interrupt

2.CPU接收interrupt number (catch)

3.查interrupt vector (dispatch 跳到handler的記憶位置)

4.執行interrupt handler(routine) (clear )

5.resume (紀錄變化前CPU的值，讓interrupt結束後能回到原狀態)

現代的追求

1. 某些時候需要延遲interrupt (重要工作上)
2. 需要有效率的分配interrupt給適合的handler
3. 需為interrupt分級，才能讓OS進行最佳排序

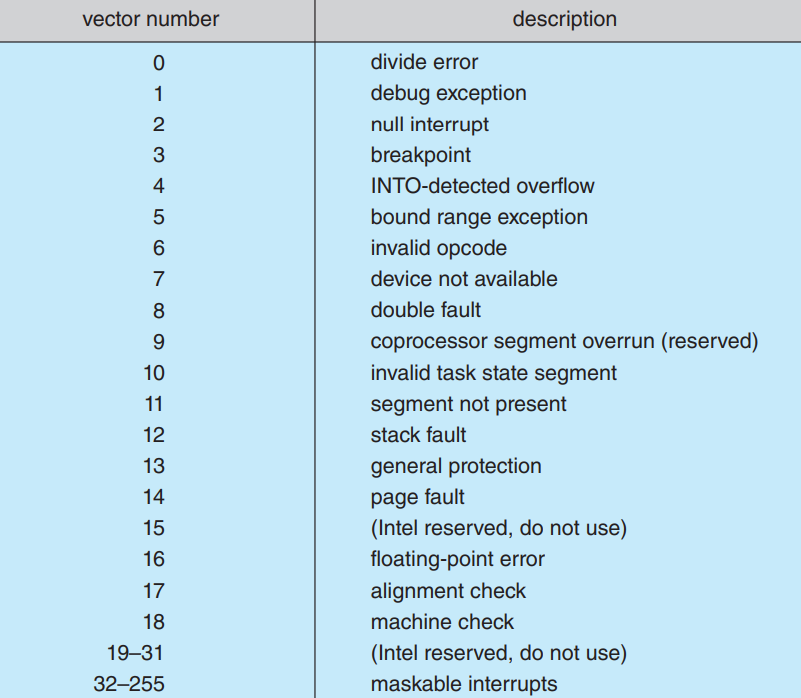
因此maskable與non-maskable因應而生(處理1、3)

Cpu會有兩條interrupt request handler，處理重要(non-maskable 不可延後)與不重要(maskable(可延後處理))的interrupt

Interrupt chain (處理2)

會先用table對到linked-list(先分類)，再逐一尋找適合handler

當然non-maskable一定會在list前面

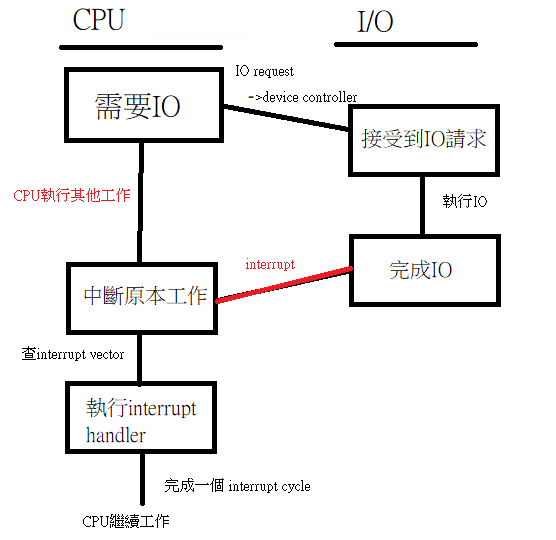


interrupt整理

\*Interrupt是為了追求”非同步處理”(效率需求)

\*interrupt可以想像為event與event handler

\*整個流程(IO interrupt)

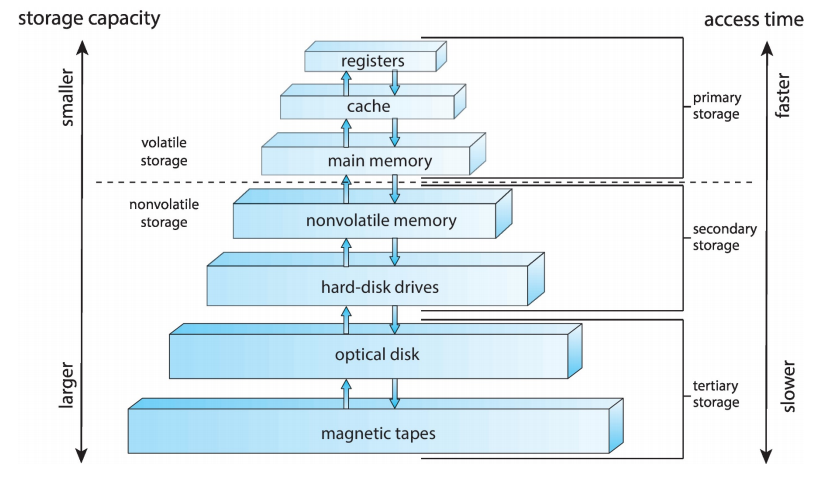


\*maskable & non-maskable: 不重要與重要(系統錯誤)的interrupt

\*interrupt vector 與 interrupt chain: 數字與interrupt handler的map

，主要概念是non-maskable一定要放先被找到的地方

1. 儲存結構



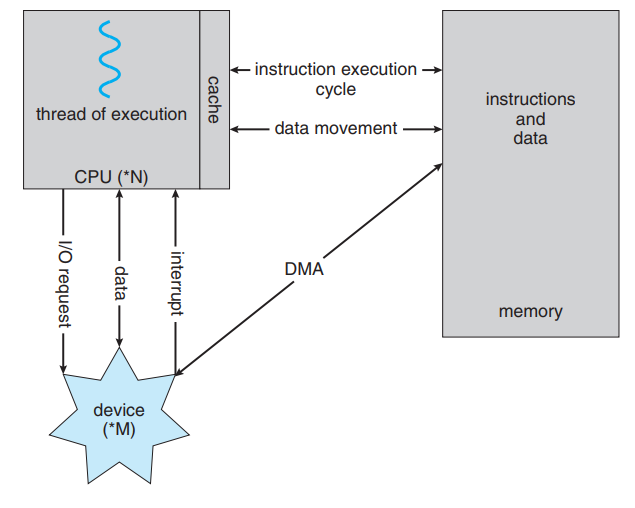
重點: \*儲存速度與容量成反比

\*main memory又稱ramdon access memory(RAM)，所以重要的開機程式要放在ROM(read-only memory)

\*volatile: (蒸發)關機後會消失

\*CPU會從memory讀instruction 放到instruction register執行 (recall:計組)

3.IO結構



DMA(direct memory access):

讓一大塊資料由CPU設定好device controller參數後(pointer、counter…)，由DMA傳與記憶體直接交換資料，CPU只會在完成IO後被完成通知中斷一次 (比每一byte被通知一次有效率的多)

結論:

\*錯誤、IO都會引起interrupt

\*Interrupt是用 非同步處理 省下IO等待時間

\*Interrupt vector的設計 是縮短interrupt的處理時間

\*Maskable與non-maskable的設計則是區分嚴重性，以分配interrupt時間達到CPU時間最佳利用

\*DMA是減少IO造成的interrupt

補充:

\*為了追求效率，interrupt是不允許被打亂的

(如:有時滑鼠會卡住)

\*storage追求:1.速度2.價格3.volatility

\*CPU是直接跟RAM溝通

\*disk會因為位置不同，讀的時間不同 (圓盤、機械式)

(所以SSD會快，因為電子化、類memeory)

\*caching: 從低階儲存裝置 複製 到高階 (不是單純搬移)

(改上層會有不一致性，但只要保證使用者看到一致就好，甚至如FB網頁例子，不追求一致性也可以)