Interrupt

(cosa sono, a cosa servono e come si usano)

Con interrupt, in informatica, si intende un segnale generalmente asincrono che indica il "bisogno di attenzione" (ad esempio da parte di una periferica). In realtà il termine interrupt si utilizza anche per un evento sincrono (coincidente con l'esecuzione di una particolare istruzione LM) che consente anch'esso **l'interruzione del processo in corso sulla CPU** qualora si verifichino determinate condizioni.

Nel primo caso (segnale elettrico) si parla di **interrupt HW** e/o interrupt **asincrono**. Si parla di segnale asincrono perché questo, generato da un dispositivo esterno alla CPU, non è detto che avvenga contemporaneamente al segnale di clock che scandisce l'avanzamento della CPU.

Nel secondo caso (Istruzione LM di richiesta Interrupt) si parla anche di **interrupt SW** e/o interrupt **sincrono**, ma molto spesso si trovano pure i termini SVC (**Super Visor Call**) o Eccezione. In pratica in tal caso si tratta di una richiesta al sistema operativo da parte di un processo in esecuzione.

In entrambi i casi un Interrupt provoca il seguente effetto: Dopo una verifica se l'INT deve essere "ascoltato", se così è, si interrompe il processo in esecuzione (salvando il suo contesto, in modo che possa essere poi in seguito ripreso dall'istruzione successiva all'interruzione senza errori. Per far questo è indispensabile al livello più basso l'utilizzo dello Stack che consente il salvataggio come minimo del valore del Program Counter del processore ed il suo successivo recupero con politica LIFO, il che consentendo l'annidamento di subroutine permette a volte anche di annidare le chiamate di interrupt. Si avvia quindi l'esecuzione di una routine di Interrupt (che deve essere specifica per l'interrupt, per far questo vi sono molte tecniche HW/SW che approfondiremo poi) la quale svolge quanto deve e quindi al suo termine avviene il ripristino della situazione precedente l'interrupt, riprendendo l'esecuzione del codice precedentemente in corso.

Perché gli Interrupt HW sono Indispensabili? Pensiamo alla gestione di un qualsiasi dispositivo di I/O, ad esempio una tastiera quando in un programma ad alto livello prevediamo una istruzione di Input come la ReadLN() Pascal. Normalmente il programma eseguibile si blocca in attesa dell'input e/o della pressione del tasto invio da parte nostra e nel frattempo il processore cosa fa? Sostanzialmente attende, svolgendo dei cicli di istruzioni di "attesa" come delle NOP. Questo può andar bene se il programma che stiamo eseguendo fosse l'unico attualmente in esecuzione sul nostro computer (è uno spreco ma il computer non avrebbe altri compiti da svolgere), ma se invece si trattasse di un computer multiutente come un vecchio Mainframe con più processi (programmi in esecuzione) di vari utenti? Si tratterebbe di un terribile spreco di tempo CPU. Lo stesso vale con un moderno PC, normalmente monoutente ma comunque multitasking: vi piacerebbe se ogni volta che pensiamo quale tasto digitare il PC non fosse in grado di fare altro? Nessun Download in Background, nessuna musica dalle casse, nulla, la CPU è impegnata a fare delle NOP!

La situazione "sprecona" si chiama **attesa attiva** in gergo tecnico mentre quella ottimizzata si chiama attesa passiva e consiste nel far svolgere altri compiti alla CPU intanto che arriva un segnale di INT che consente di procedere. Poi in merito vedremo l'esempio del "risotto" con o senza campanello,

Inoltre c'è il problema della segnalazione di un evento: come far sapere alla CPU che abbiamo premuto il tasto invio oppure che la masterizzazione di un CD-WORM è terminata? Appunto tramite un segnale di interrupt HW inviato da quel dispositivo di I/O!

Per gli interrupt SW occorre invece vedere degli esempi nell'ambito dei SO, che esamineremo poi! Per ora possiamo dire che il meccanismo di gestione è simile (interruzione del codice in corso per avviare una subroutine di gestione), ma si dicono sincroni perché avviati da una istruzione LM.

Come funzionano gli Interrupt ad esempio in una vecchia CPU come lo Z80? (si noti che gli stessi concetti con le dovute variazioni valgono anche per CPU più moderne)

Lo Z80 dispone di 2 possibili segnali di Interrupt HW, quindi di due PIN che giungono al processore.

Il primo, che si chiama **NMI** (Not Maskerable Interrupt) è un segnale della massima priorità e per questo si dice "non mascherabile" e serve per **segnalare delle situazioni eccezionali e urgenti**, come ad esempio un calo di tensione che deve interrompere le attività in corso ad esempio per avviare l'utilizzo di un alimentatore alternativo (come un UPS) e procedere al salvataggio su memoria permanente della situazione parziale delle attività, in modo poi che se si spegne il computer si possa riavviare in seguito (dopo la riparazione del guasto) senza perdere dati. Questo tipo di interrupt viene sempre ascoltato e nello Z80 il suo arrivo corrisponde alla seguente sequenza di azioni:

1) Al termine dell'istruzione LM corrente (quindi pochi cicli di clock) si copia il valore nel flag IFF1 nel flag IFF2 (questi flag servono per decidere se gli altri INT, quelli mascherabili, dovranno essere ascoltati) e si pone IFF1 ad un valore che disabilita ogni altro INT.
2) Viene salvato sull'area di memoria utilizzata come stack (tramite adeguato utilizzo del registro SP o Stack Pointer che quindi viene modificato decrementandolo di 2 unità) il valore attuale del registro PC o Program Counter.
3) Viene **caricato sul PC il valore 0066h** che è la posizione della RAM riservata a contenere la subroutine di gestione del NMI, in pratica si fa una JP 0066h.
4) Svolta la gestione del NMI (dentro la ruotine ci si dovrà occupare di capire il motivo e gestirlo) la subroutine termina con una istruzione **RETN** che utilizzando SP e Stack ripristina la situazione precedente del contatore di programma o registro PC e provvede anche a copiare il valore sul flag IFF2 su IFF1 (ritornando al precedente stato di abilitazione o disabilitazione delle INT mascherabili).

Il secondo, che si chiama **INT** (Interrupt) è il segnale comunemente usato ed è "mascherabile" attraverso l'uso delle istruzioni LM **EI** (Enable Interrupt) e **DI** (Disable Interrupt) che agiscono sui flag IFF1 ed IFF2. Al reset la CPU Z80 ha gli INT disabilitati (IFF1=0=IFF2) mentre dopo una istruzione EI IFF1=1=IFF2 e quindi divengono abilitati. Quindi se le INT sono abilitate (IFF1=1) ed arriva un segnale di INT avviene la seguente sequenza di azioni:

1) Al termine dell'istruzione LM corrente (quindi pochi cicli di clock) si disabilitano gli interrupt ponendo a Zero sia IFF1 che IFF2.
2) Viene salvato sull'area di memoria utilizzata come stack (tramite adeguato utilizzo del registro SP) il valore attuale del registro PC ovvero il Program Counter.
3) A questo punto vi sono **3 possibilità alternative** che dipendono dallo stato dello Z80 come modalità di gestione Interrupt: al reset la condizione di default è lo stesso stato che si ottiene con l'istruzione **IM0** (Interrupt Mode 0), ma sono più interessanti (perché più generiche ed applicabili anche ad altre CPU) le altre 2 modalità, ovvero il modo 1 (ottenibile con l'istruzione **IM1**) ed il modo 2 (**IM2**) che viene anche detto **modalità vettorizzata** (il modo zero è una sorta di modalità vettorizzata limitata, specifica dello Z80 e non ce ne occupiamo, mentre il modo 2 è abbastanza complesso e lo esamineremo in seguito).
5) Supponendo che lo stato sia **IM1, viene caricato sul PC il valore 0038h** che è la posizione della RAM riservata a contenere la generica subroutine di gestione delle INT in modo 1, in pratica si è fatta una CALL 0038h (considerata la precedente gestione dello Stack).
6) La **subroutine di gestione** (NB **modo 1**) **dovrà occuparsi di capire quale motivazione ha provocato il segnale di INT** (che potrebbe avere decine di origini differenti) e quindi **svolge un riconoscimento SW dell'interruzione**, nel senso che tramite istruzioni LM si andranno a consultare tutti i dispositivi di I/O che potrebbero aver generato la INT sino a che non si determina cosa deve essere fatto per gestirla.
7) Svolta la gestione della INT la subroutine termina con una istruzione **RETI** che utilizzando SP e Stack ripristina la situazione precedente del contatore di programma o registro PC ma, avendo in precedenza azzerato entrambi i Flag IFF1 ed IFF2, **lascia le INT disabilitate**, per cui è necessario anteporre alla RETI una EI se si desidera che le prossime INT mascherabili siano abilitate.

Terminologia: Segnalo che il termine ***Polling*** (controllo via SW) viene associato a volte all'attesa attiva, ma anche alla gestione SW del riconoscimento di una Interruzione dentro la sua ISR (Interrupt Service Routine).

Esaminiamo ora il **Modo 2 o Vettorizzato** per le INT: Questa modalità è decisamente più complessa e sofisticata! Quando la CPU riceve il segnale di INT quando si è nello stato indotto da EI, poi avviene una serie di operazioni "automatiche" che fanno giungere alla CPU un BYTE tramite il DataBus (questo Byte viene generato da dispositivi di I/O predisposti a farlo, come PIO (Parallel Input Output), SIO (Serial Input Output), CTC (Conter Timer Circuit) ed altri, che sono dei CHIP che sono stati progettati a "corredo" dello Z80 per gestire le problematiche di Input/Output). Questo byte, considerato Low e che deve avere il bit LSB di valore 0, viene giustapposto al un byte, considerato High, che è presente dentro la CPU nel registro I (che deve quindi essere stato precedentemente caricato con un apposito valore), formando una word di 16 bit, utilizzabile come un indirizzo di memoria. Poi vengono caricati sulla CPU dalla memoria centrale due Byte selezionati da una zona di RAM di 256 Byte (detta **Vettore Interruzioni** o Interrupt Vector) individuata dai valori presenti sul registro I e dal Byte ricevuto dall'I/O. Questi due Byte costituiscono l'indirizzo di inizio della subroutine di gestione interruzione che viene avviata con una "call" che ha le stesse modalità viste prima per le INT modo1 (in pratica non si trova all'indirizzo 0038h ma in un indirizzo di memoria fra 128 possibili determinato dalle impostazioni del Vettore Interruzioni e da tutta la procedura vista prima). Questa modalità "vettorizzata" consente di individuare (con un apposito supporto HW) "automaticamente" la corretta subroutine di gestione interruzione, quindi **il riconoscimento di quale situazione gestire con la INT avviene a livello HW** (e non SW in "polling" come nel modo1) **risultando molto più veloce**. Ovviamente la necessità di un supporto HW ha un costo, oltre a dover essere già predisposta.

Altri processori (ad esempio Intel x86) prevedono delle modalità che sfruttano un supporto HW esterno per raggiungere risultati simili, in particolare ad esempio un chip esterno di supporto specializzato per la gestione delle interruzioni che riceve N differenti segnali di INT e comunica un codice alla CPU (se N=8 si può ovviamente codificare su 3 bit) per indicare quale subroutine di gestione avviare, con effetti analoghi alla tecnica vettorizzata dello Z80.

Nel caso specifico dello Z80 vi sono problematiche connesse alla priorità dei vari dispositivi di I/O (fra loro asincroni) nel caso di più richieste contemporanee, in tal caso la gestione delle priorità (arbitraggio - predisposto a livello HW) viene affidata ad una politica detta "Daisy Chain" (ma per i Nostri obiettivi didattici è inutile qui approfondire oltre).

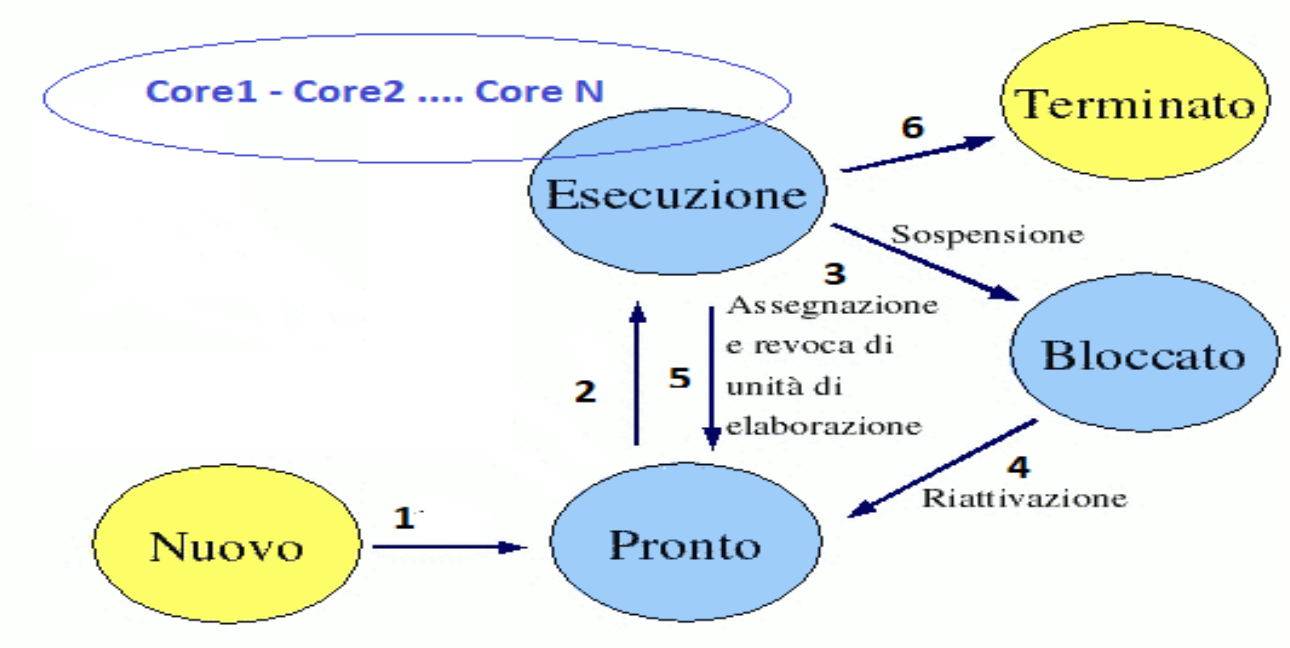
Interruzioni SW (sincrone) denominate a volte anche **SVC, Eccezioni o TRAP.**

Queste interruzioni si considerano "sincrone" in quanto attivate da una apposita istruzione LM, ad esempio **nell'architettura Intel x86** gli **interrupt software** vengono determinati dall'esecuzione di istruzioni **INT N** ed in tal caso viene attivata la corrispondente ISR (Interrupt Service Routine) il cui indirizzo è all'N-esimo posto della tabella delle interruzioni (Interrupt Vector Table). Nello Z80 invece non sono previste specifiche istruzioni di "call" di INT SW nel set LM. In altri processori hanno nomi tipo **SVC** (nel senso che si effettua una chiamata al SuperVisore, praticamente il codice del SO) o **TRAP** (il nome TRAP ovvero "trappola" potrebbe richiamare l'idea di una buca in cui si cade, ad esempio a causa di un malfunzionamento come un errore di DivisionByZero, dentro la quale si svolgono delle operazioni di gestione del problema tramite la ISR, tornando poi in superficie al codice precedente) ed il loro utilizzo è consentito in base a dei subset LM disponibili o meno a seconda di varie modalità di funzionamento del processore (modalità protetta, supervisore, utente, etc).

Passando al loro utilizzo nell'ambito dei SO (Sistemi Operativi) sono appunto molto utili per gestire eccezioni (quindi situazioni fuori standard come ad esempio il già citato errore di DivisionByZero) e per le transizioni di stati dei processi svolte dal Kernel o Nucleo del SO nel livello di gestione del processore. Vediamo di seguito alcuni esempi di INT HW e INT SW legate a questo.

Partiamo da un diagramma degli stati di avanzamento dei processi in un SO Multitasking che indica anche sommariamente come lo stato di esecuzione potrebbe contenere più processi contemporaneamente se si dispone di un microprocessore Multicore o un sistema di elaborazione multiprocessore, ovviamente per semplicità supponiamo che la CPU sia unica ed abbia 1 solo core.

NB I seguenti termini sono intercambiabili: Esecuzione=Running; Pronto=Ready, Bloccato=Wait.



Esaminiamo quale tipologia di INT provoca le varie transizioni (numerate a fianco delle frecce):

1) INT SW, il codice del kernel del SO gestisce l'avvio del codice di un programma in Processo attivo, ma innescato presumibilmente da un INT HW (ad esempio quando si fa click sull'icona di un programma da avviare).
2) INT SW innescato da un altro INT SW (esempio transazioni 3 o 6) oppure un INT HW (transazione 5), il codice del kernel del SO gestisce l'arrivo nello stato di Running del codice di un Processo attivo, ripristinando sulla CPU tutte le informazioni prelevate dal suo PCB (Process Control Block).
3) Tipico INT SW, il processo necessita di una risorsa e quindi richiede al codice del kernel del SO di gestire il suo passaggio in attesa passiva nello stato di Bloccato in attesa che la risorsa sia disponibile (ad esempio una banale pressione del tasto invio, oppure la conclusione di una operazione in memoria secondaria, etc), chiaramente con il salvataggio del suo PCB ed innescando una transazione di tipo 2 dopo che la politica di gestione del SO avrà selezionato quale processo fra quelli nello stato di pronto porre nello stato di running.
4) Generalmente si tratta di un INT HW (ma potrebbe anche essere un INT SW se la risorsa era gestita dal SW di SO), ad esempio un dispositivo di Input ha preparato i dati che il processo attendeva per poter procedere nella sua esecuzione. La gestione da parte del codice del kernel provvede ad aggiornare la posizione del processo dallo stato di WAIT a quello di Pronto.
5) INT HW: l'orologio di sistema segnala che è **terminato il time-slice** a disposizione del processo attualmente in esecuzione quindi tale **processo deve pre-rilasciare il processore** per consentire ad altri di occuparlo mantenendo per l'utente l'apparente sensazione di contemporaneità di esecuzione fra tutti i processi con cui interagisce. Ovviamente il codice del kernel del SO provvede a gestire il salvataggio del suo PCB e innescare una transazione di tipo 2 per rioccupare il processore.
6) INT SW: il codice in esecuzione arriva al termine e quindi una Call al Supervisore segnala il suo passaggio nello stato di terminato

In realtà possono esserci altre transazioni, ad esempio fra gli stati di pronto oppure di bloccato ed uno stato non indicato nel diagramma soprastante che potremmo chiamare "SOSPESO", ad esempio quando il numero di processi attivi è eccessivo ed un processo viene temporaneamente congelato liberando spazio su RAM (Swap). Oppure all'avvio un Job non prioritario potrebbe essere subito sospeso.