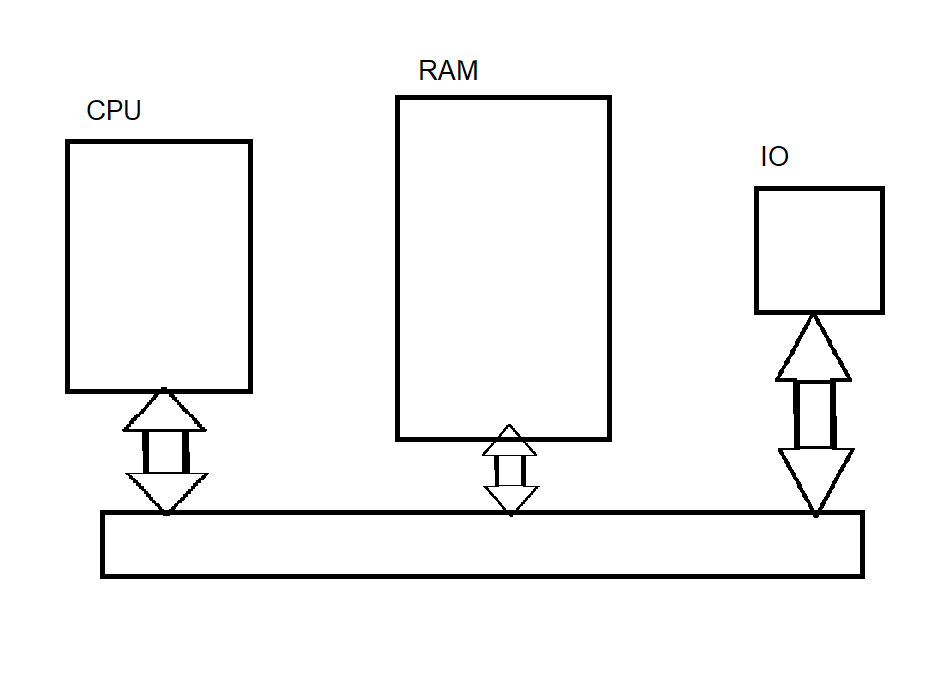
Abbiamo visto come tecnica di sincronizzazione il busy waiting. Ma abbiamo anche detto che questo è un metodo un po’ troppo costoso perché spreca molti cicli di clock del processore. Per evitare questo inconveniente viene introdotto un meccanismo che viene detto delle “interruzioni” (Interrupt).

Introduciamo però prima il bus, esso non era presente nella macchina di Von Neumann ma è presente in tutte le CPU moderne. I fili di indirizzamento, dei dati e di controllo fanno tutti parte un corposo gruppo di fili chiamato “BUS”. Sul BUS ci sono poi dei fili aggiuntivi: i fili di richiesta di interruzione.



Ovviamente la RAM non genera richieste di interruzione, ma i dispositivi di input output sì. L’idea è che se ci fossero più dispositivi di input output potrebbero tutti condividere il filo di Interrupt Request, il filo sarebbe quindi gestito da una funzione di tipo OR: se anche solo un dispositivo richiede l’interrupt l’informazione arriva alla CPU (questo tipo di connessione è detta OR cablato).

L’interrupt si divide in tre fasi: La fase di Segnalazione, la fase di Risposta e la fase di Gestione. La Gestione può essere strutturata in due modi diversi: Si può avere una gestione Immediata o una gestione Differita.

Il Meccanismo di segnalazione è implementato a livello hardware (con l’aggiunta del filo per l’Interrupt Request e di una componente hardware del processore che causa un comportamento diverso quando riceve un valore true).

La risposta arriva dal processore, che interrompe l’esecuzione del programma in esecuzione, dopo aver salvato la situazione corrente (per poterlo riprendere, ad esempio salvando i registri nello Stack), e chiama un programma chiamato “Gestore delle Interruzioni”.

Come si può realizzare tutto ciò? Beh, il nostro Processore per eseguire un istruzione segue i passaggi di fetch, decode e execute in maniera ciclica (e progressivamente prendendo istruzioni successive). Il meccanismo delle interruzioni è implementato in modo da interrompere il programma alla fine della fase di execute (si evita che le istruzioni rimangano eseguite a metà). Per ottenere ciò basta aggiungere una quarta fase: La fase di Verifica della presenza di Richieste di Interruzione (Verifica Int. Req.). Questo ovviamente causerà un po' di ritardo nel processore, perché servirà un ciclo di clock in più per fare anche la verifica della richiesta di interruzioni.

Per fare tale Verifica la Cpu si limita a controllare se sul filo corrispondente sia presente il valore 0 (Nessuna interruzione) o 1 (Richiesta di interruzione).

Quando avviene l’interruzione viene effettuato il PUSH del PC (nello Stack). Fatto questo si può modificare il contenuto del PC; si inserisce quindi un indirizzo particolare (chessò la cella 28) che corrisponde alla prima istruzione del gestore delle interruzioni:  
RAM[SP] <- PC

PC <- IND. INT. HANDLER

A questo punto la CPU prosegue con la fase di fetch, che a questo punto non raccoglierà più un’istruzione del programma che stava venendo eseguito prima, ma bensì la prima del gestore delle interruzioni. A livello hardware non è quindi necessario fare altro: l’interruzione viene gestita come se fosse una chiamata di funzione (al termine della gestione dell’interruzione il flusso del programma riprenderà normalmente). A differenza di una funzione, però, la sequenza di istruzioni del gestore delle interruzioni non è programmata dallo sviluppatore, ma è predefinita.  
Dopo l’esecuzione della prima istruzione dell’ INT handler avviene nuovamente la verifica della richiesta di interruzione, ma a questo punto la richiesta di interruzione c’è ancora (perché è stata eseguita solo la prima istruzione dell’int.handler) e questo è un grave problema perché sovrascrive il valore del PC salvato nello Stack. Poiché questo non va bene bisogna fare una piccola modifica a questo sistema di funzionamento, aggiungendo un bit all’interno del registro di stato, chiamato Bit di Maschera dell’Interruzione. L’idea è che le interruzioni arrivate dall’esterno vengono “filtrate”, ad esempio attraverso una funzione AND della quale si guarda l’uscita. Inizialmente il valore del bit di stato (di Maschera) assume il valore 1 all’uscita della funzione AND si ha lo stesso valore presente sul filo di Richiesta delle interruzioni. Se si cambia il valore del bit di Maschera, però, si porta a 0 il valore di uscita della funzione AND; ciò permette al processore di “tapparsi le orecchie”. Questo meccanismo evita che il processore senta nuovamente la richiesta di un’interruzione. Quindi, l’ordine reale delle istruzioni quando arriva una richiesta di interruzione è:

RAM[SP] <- PC

PC <- IND. INT. HANDLER

MASK <- 0

Così facendo la seconda/terza ecc. verifica di richiesta di interruzione non ritornano vere mentre si sta già gestendo un’interruzioni (-> il gestore di interruzioni non può e non deve essere interrotto da una richiesta di interruzione, perché è già lui che la sta risolvendo).

A questo punto posso concentrarmi sulla scrittura del programma INT Handler, che deve fare l’equivalente di quel che faccio io quando vado a rispondere al citofono. Questa è la fase di gestione.

Ricapitolando:

La segnalazione è strutturata a livello hardware, La Risposta è hard-coded a livello Software è la Gestione è anch’essa gestita a livello software.

All’interno di uno dei registri dei dispositivi di I/O (mappati in memoria) ce ne sarà uno che presenta un bit positivo che segnali che è stato lui a chiamare l’interruzione e, una volta trovato, si può procedere a gestirla.

Vediamo come esempio di I/O una tastiera e una Stampante. Il dispositivo di Input potrebbe essere programmato per inviare un’interruzione ogni volta che l’utente preme un tasto. La richiesta di interruzione fa sì che, attraverso una risposta sufficientemente rapida, il valore nel registro di input possa essere salvato velocemente, evitando perdite di dati in caso di premuta di più tasti in rapida successione.

Il problema visto con la stampante (cioè la necessità di attendere finché non è finita la stampa del carattere) può essere risolto avviando/continuando l’esecuzione di un altro programma che non richieda l’uso della stampante e programmando il dispositivo in modo che mandi un’interruzione quando finisce la stampa (interrompendo il programma che non c’entrava con la stampante e riprendendo l’esecuzione della scrittura di dati nel registro di output).

Supponiamo che per qualche motivo arrivino due richieste di interruzione contemporaneamente (un tasto è stato premuto e la stampante è pronta per ricevere un nuovo dato). Il gestore delle interruzioni potrebbe andare a vedere prima la tastiera e poi la stampante e poi viceversa: in ogni caso troverebbe il valore 1 e quindi saprebbe che quel dispositivo ha inviato una richiesta di interruzione. L’ Int Handler si ritrova quindi a dover scegliere cosa fare/ a cosa dare priorità. Il criterio che viene seguito è l’analizzare quali sono i tempi di risoluzione delle richieste dei vari dispositivi (in particolare si tiene conto di cosa può succedere se non si riesce a fare in tempo a gestire la lettura da tastiera oppure la stampa di un carattere). Nel caso non si riesca a gestire in tempo l’interruzione da tastiera si ha una perdita di dati (se l’utente sta digitando velocemente) poiché si salva solo l’ultimo carattere premuto dall’utente. Nel caso della stampante, se lascio passare tanto tempo dopo la richiesta di interruzione da parte della stampante (che corrisponde alla notifica di disponibilità di stampa) la stampante si limita a restare lì ferma, senza causare però gravi problemi (solo attese potenzialmente fastidiose).

Quindi nell’esempio proposto, tra i due dispositivi è più importante gestire più velocemente l’input da tastiera. Si può quindi introdurre il concetto di priorità. Nell’esempio proposto il sistema di input da tastiera ha una priorità più alta rispetto alla stampante, tuttavia non è neanche così alto (la perdita di uno o due caratteri non è Gravissimo). Per confronto, il sistema di frenata di un automobile ha probabilmente uno dei livelli di priorità più elevati in assoluto.

Potremmo quindi far passare più fili di richiesta di interruzione, ciascuno dei quali identifica un particolare livello di priorità dell’interruzione (e quindi possiamo assegnare livelli di priorità ai dispositivi collegandoli a fili diversi). Si complica un pochino il sistema di Segnalazione, poiché bisogna prevedere di avere diversi bit di Mascheramento (di sistema), uno per ogni livello di priorità e far terminare i fili e i bit in un and ciascuno, per poi far convogliare gli output degli AND in un OR. In questo modo, quando si fa partire il gestore delle interruzioni viene azzerato il bit (o i bit) di sistema corrispondente alle richieste di interruzione a priorità uguale o minore a quella presa in esame. Il gestore delle interruzioni eseguirà quindi un codice apposito per la gestione di situazioni a quel livello di priorità.

Se quindi arriva una richiesta ad alta priorità vengono azzerati tutti o quasi i bit di maschera per le interruzioni, quindi non potrà essere interrotto. Quando avrà finito verranno ripristinati i valori 1 nei bit di Maschera che erano stati azzerati.

Se arriva una richiesta a priorità bassa verranno azzerati pochi (o solo uno) bit di maschera per le interruzioni, quindi il gestore delle interruzioni, in questo caso, potrà essere interrotto ma solo da interruzioni a priorità più alta di quella corrente.

Il Termine Vettore di Interruzioni (Interrupt Vector) sono le coppie di valori che vanno inseriti nel PC e nel bit di maschera quando viene richiesta un’interruzione.

Il come si ripristina l’esecuzione del programma dipende dal tipo di gestione dell’interruzione.

Per esempio se effettuiamo una gestione immediata si va a leggere direttamente il registro del dispositivo di I/O e ne si copia il valore in una cella dello Stack, a questo punto poiché abbiamo il valore salvato in memoria possiamo riprendere l’esecuzione del programma precedentemente in esecuzione attraverso l’istruzione RETI (acronimo simbolico, Return From Interrupt) ripristina il valore originario del Program Counter e dei Bit di Maschera del registro di Stato (attraverso funzioni POP). Così si sono ripristinati i valori dei bit di maschera e il programma può riprendere l’esecuzione da dove era stato interrotto (perché si è ripristinato il PC).

Un esempio di gestione differita è se la Stampante è libera ma al momento non ci sono altri caratteri da stampare; l’informazione (che la stampante è libera) viene salvata in memoria e si effettua l’istruzione RETI.

La differenza tra gestione immediata e differita è che in quella immediata si è spesa una certa quantità di tempo per gestire l’interruzione e si è fatto con gli interrupt di priorità inferiore o uguale disabilitati, invece in quella differita è che il sistema si limita a prendere Rapidamente nota di uno stato e riguarda un’operazione che si può fare ma che non è al momento necessario fare.

La gestione immediata è per quando l’interruzione è urgente, mentre quella differita è o per quando non è urgente o per quando gestirla prenderebbe al momento troppo tempo perché ne valga la pena.

La differenza tra funzioni e gestione delle interruzioni è la velocità: nelle funzioni non ho vincoli sul tempo che essa ci può mettere per venir eseguite, mentre nel caso di gestione delle interruzioni può essere critico il tempo che ci metto a mandare in esecuzione l’Interrupt Handler (e si cerca di rimanere per il minor tempo possibile con gli interrupt -mask- disabilitati).

Osservazioni: Le applicazioni sono a un livello di priorità più basso anche rispetto alle interruzioni di livello più basso. Inoltre, mentre sta venendo gestita una interruzione a priorità più alta, non possono venir gestite altre interruzioni a priorità alta, quindi è il tipo di gestione che deve prendere il meno tempo possibile (è infatti consigliabile la gestione immediata di questo tipo di interruzione SOLO se ci vogliono pochi cicli di Clock). In genere il sistema (ma il metodo può variare da sistema a sistema) operativo si occupa della gestione delle interruzioni a priorità più alto tramite gestione differita (cioè l’appunto in memoria della situazione) e l’avvio di un’applicazione che gestisca in maniera, appunto, differita, la situazione.

Supponiamo di avere a disposizione un sistema sul quale sono stati definiti Quattro livelli di priorità, che quindi ha 4 fili di interruzione nel BUS e ha 4 bit di Maschera nel registro di stato. Ci saranno 4 funzioni AND e 1 funzione OR a 4 ingressi. Ciò ci costa quindi un paio di complicazioni hardware ma che a seconda dei casi potrebbe valerne la pena.

A livello di organizzazione, potrebbe essere conveniente stabilire una convenzione. Si stabilisce una volta per tutte che alcune celle della memoria non possono essere usate per operazioni qualsiasi, ma devono essere utilizzate per scopi particolari. Prendiamo quindi che dalla cella 18 vada inserito il vettore delle interruzioni: abbiamo un certo numero di bit in cui vada inserito l’indirizzo dell’ Int. Handler corrispondente al livello 0 di priorità e un altro numero (4) in cui siano presenti i valori iniziali delle maschere. Ci sono poi altre 3 celle (\*2 in termini di indici nello Stack) in cui, oltre alle maschere ci sono presenti gli indirizzi degli Int. Handler di livello superiore.

Questa convenzione permette al processore di far sì che quando riceve un interruzione di un preciso livello esso cambi il valore del PC in quello dell’indirizzo corrispondente nel vettore delle interruzioni e imposta i bit di maschera a quelli riportati. In genere è chi programma il sistema operativo a stabilire i valori da inserire nei bit di maschera (potrebbe per esempio decidere di lasciare attivi alcuni bit di maschera che normalmente non lo sarebbero, per qualche motivo a me sconosciuto). Quindi si dà la possibilità di organizzare a livello software delle cose che altrimenti difficilmente ci si immaginerebbe di poter fare.

Infatti, se non vengono inizializzati correttamente i valori nel vettore delle interruzioni ciò causa gravi problemi al processore, ecco perché si permette al sistema operativo (e quindi a chi lo programma) di manipolare quei valori. Questo spiega inoltre perché è necessaria una fase di bootstrap iniziale: perché inizialmente il processore funziona in maniera incompleta e attende per operare a pieno regime che il sistema operativo inizi correttamente i valori ( cosa che può richiedere un po’ di tempo, dato che il sistema operativo non si avvia istantaneamente).