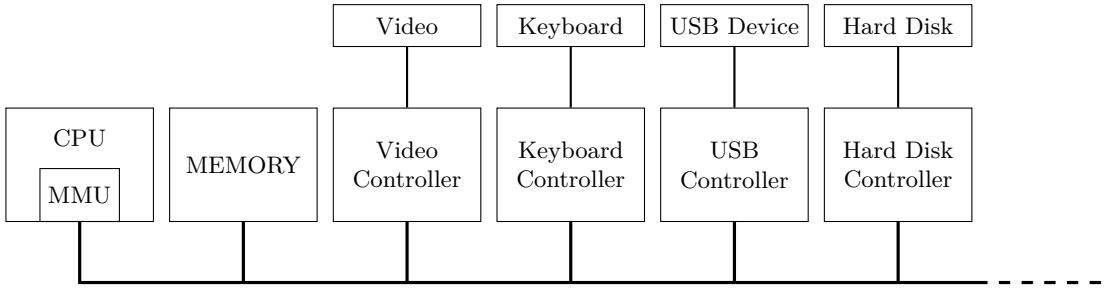
Azzolini Riccardo 2019-09-25

CPU e interrupt

1 SO e hardware

Il SO interagisce direttamente con l'hardware.

Semplificando, l'hardware si può rappresentare come segue:



2 CPU

La CPU preleva le istruzioni dalla memoria (*fetch*), le decodifica (*decode*) e le esegue (*execute*). Una CPU "tradizionale" esegue queste tre operazioni in modo ciclico:

1. fetch;
2. decode;
3. execute.

La CPU è dotata di **registri**

- **generali** (*general purpose registers*), che costituiscono il primo livello della gerarchia di memoria: essi contengono i dati e i risultati temporanei su cui la CPU lavora;
- **di controllo** (*control registers*), che controllano lo stato, e quindi il funzionamento, della CPU stessa:
  - **Program Counter** (**PC**): l'indirizzo dell'istruzione da prelevare;

- **Stack Pointer** (**SP**): l'indirizzo della cima (top) dello stack in cui sono contenuti i frame (o record di attivazione) delle procedure;
- **Program Status Word** (**PSW**): un insieme di uno o più registri che è suddiviso in gruppi di bit con significati particolari:
  * **Privileged Mode** (**PM**): bit di modalità user/kernel;
  * **Condition Code** (**CC**): codici di condizione impostati dall'ALU a seguito di particolari operazioni (confronti, aritmetica, ecc.);
  * **Interrupt Mask** (**IM**) e **Interrupt Code** (**IC**): usati per gestire gli interrupt;
  * **Memory Protection Information** (**MPI**): informazioni sulla porzione di memoria accessibile (come ad esempio i registri LBR e UBR).

Nota: Alcuni autori considerano i registri PC e SP parte della PSW.

3 Interrupt

Definizione: Gli **interrupt** (**interruzioni**) costituiscono un meccanismo per *segnalare alla CPU un evento o una condizione* avvenuti nel sistema che devono essere trattati dal SO.

Gli obiettivi degli interrupt sono due:

- *interrompere il normale ciclo di esecuzione* della CPU, che di norma continuerebbe a eseguire le istruzioni del programma corrente;
- *richiedere l'intervento del SO*, cioè avviare l'esecuzione di codice appartenente al SO.

Osservazione: Interrompere il programma non vuol dire forzarne la terminazione: solitamente, una volta finito l'intervento del SO, l'esecuzione del programma riprende.

4 Interrupt request e categorie di interrupt

Un interrupt avviene quando la CPU riceve un segnale hardware (**interrupt request**) su un'apposita linea del bus di sistema.

A seconda dell'origine di questo segnale, gli interrupt si suddividono in:

hardware interrupt: il segnale è inviato dal clock, oppure dal controller di un dispositivo;

program interrupt: il segnale è provocato dal programma in esecuzione.

5 Hardware interrupt

Gli hardware interrupt si suddividono ulteriormente in:

timer interrupt: usati dal clock per notificare il tick (cioè il passaggio di un determinato quanto di tempo);

I/O interrupt: usati dai dispositivi di I/O per notificare eventi che devono essere trattati dal SO (come ad esempio la terminazione di una lettura/scrittura su disco).

Questi sono *eventi asincroni* rispetto al programma in esecuzione, che infatti non ha modo di prevedere se e quando si verificano.

Le operazioni che il SO deve eseguire per gestire questi interrupt variano in base al tipo di evento. Ad esempio, deve:

- controllare se il tick ha causato l'esaurimento del time slice del programma in esecuzione, e in tal caso eseguire un context switch;
- rendere nuovamente schedulabile un programma quando termina l'operazione di I/O che esso stava aspettando.

6 Fasi dell'hardware interrupt

Il meccanismo degli hardware interrupt è composto (semplificando) da 5 fasi:

1. Mentre sta eseguendo l'istruzione i di un programma P, la CPU riceve un'interrupt request.
2. Una volta terminata l'esecuzione dell'istruzione i, la CPU sospende l'esecuzione di P (senza quindi prelevare l'istruzione $i+1$) e salta invece alla procedura di gestione dell'interrupt, detta **interrupt handler**, che è parte del codice del SO.
3. L'interrupt handler gestisce l'interrupt.
4. L'interrupt handler restituisce il controllo a P, oppure a un altro programma P' interrotto in precedenza.
5. Il programma schedulato al punto 4 riprende la propria esecuzione dal punto in cui era stato interrotto. In particolare, se è stato schedulato P, viene prelevata l'istruzione $i + 1$.

Osservazione: Le fasi 2 e 4 non sono affatto banali, perché il valore dei registri (e in particolare del PC) va salvato nella fase 2 e ripristinato nella fase 4, ma ciò non può essere effettuato mediante delle normali istruzioni: esse non sono presenti nel programma interrotto (dato che questo non può prevedere l'interruzione), e non possono neanche essere scritte nell'interrupt handler perché, quando inizia l'esecuzione di quest'ultimo, almeno il valore del PC è già stato modificato.

7 Interrupt a cascata

Può capitare che durante l'esecuzione di un'interrupt handler arrivi un'altra interrupt request. In generale, questa può essere gestita:

1. si sospende l'esecuzione dell'handler corrente;
2. viene eseguito l'handler corrispondente alla nuova interruzione;[1]
3. si riprende l'esecuzione dell'handler sospeso.

Bisogna però evitare un numero eccessivo di interrupt a cascata, perché lo spazio in memoria dedicato al salvataggio dello stato dei programmi interrotti è limitato. A tale scopo, gli interrupt vengono organizzati in **classi di priorità**. Durante la gestione di un'interrupt request, vengono *temporaneamente ignorate* le richieste con priorità *minore o uguale*: esse rimangono **pendenti**, e verranno trattate in seguito. In questo modo, il numero massimo di interrupt a cascata corrisponde al numero di classi di priorità.

Un esempio di gerarchia degli interrupt hardware, elencati dalla priorità massima alla minima, è:

1. machine error;
2. clock interrupt;
3. disk interrupt;
4. fast device interrupt;
5. slow device interrupt.

Per specificare quali classi di interrupt sono abilitate (**enabled**) e quali no (**masked off**), si usano i bit di **Interrupt Mask** (**IM**) contenuti nella PSW. Ci sono due possibili modi per codificare queste informazioni nei bit dell'IM:

- un bit enabled/masked off per ogni classe di interrupt;
- un valore m che abilita tutti e soli gli interrupt di priorità $\geq m$.

[1] *Osservazione*: Quando viene sospeso un programma utente e si salta a un interrupt handler, è necessario passare dalla modalità user alla modalità kernel (e viceversa quando l'esecuzione dell'handler termina). Invece, quando si salta da un handler a un altro per la gestione degli interrupt a cascata, il processore rimane semplicemente in modalità kernel.

In modalità utente, tutti gli interrupt sono abilitati, e il programma non può modificare la PSW, altrimenti potrebbe impedire completamente al SO di interromperlo. L'IM viene invece impostata in modo opportuno quando il controllo passa a un interrupt handler.