Input / Output

Mon, 16 May

I **dispositivi** di **I/O** sono dispostivi aggiuntivi rispetto al "cuore" del computer.

Questi dispositivi sono caratterizzati da:

- Velocità trasmerimento (**bandwidth**)
- **Latenza**
- **Sincronizzazione**
- Modalità di **interazione**

BUS

Tali dispositivi sono collegati a CPU e RAM tramite dei **BUS**.

Implementazione

Alcune implementazioni prevedono BUS *aggiuntivi* per i dispositivi di I/O, *MIPS/SPIM* utilizza il BUS *comune* di *sistema* anche per la comunicazione con le *periferiche*.

Il **BUS di sistema** è suddiviso in:

- BUS **dati**

  È un BUS che comunica i dati tra la memoria o le periferiche e il processore
- BUS **indirizzi**

  È un BUS che trasferisce gli indirizzi richiesti dal processore verso la memoria
- BUS di **controllo**

  Trasferisce i segnali di controllo in uscita dal processore

Software

A livello software, ci sono varie implementazioni, e in *MIPS/SPIM* si utilizzano le stesse istruzioni (`LW` , `SW` ...) usate per la RAM anche con le *periferiche*.

Ogni periferica ha un **indirizzo risevato** che la distingue da le altre, in questo modo è possibile determinare il *target* delle *operazioni* richieste.

È possibile scegliere *arbitrariamente* di *limitare* gli indirizzi relativi alla memoria per dedicarne alcuni alle *periferiche* (*esempio: MSB a 0 per la RAM e a 1 per I/O*), altrimenti si possono implementare circuiti di *codifica e decodifica* per minimizzare lo spreco di *risorse*.

Interfaccie

Tra i dispositivi e l'unità di controllo ci sono delle **interfaccie** che fanno "da tramite" per rendere possibile la comunicazione tra i due.

Registri

Un'interfaccia contiene dei *registri*:

- Registro **dati** (*buffer*)

- Registro di **stato**.

Registro Dati

`0xffff0004` `[ | 8 ]`

Gli 8 bit meno significativi contengono i dati ricevuti (*Recieved byte*).

Registro di Stato

`0xffff0000` `[ |1|0]`

I due bit meno significativi svolgono funzioni particolari:

- `[1]` Interrupt

  Bit che può essere letto o scritto dal software.
- `[2]` Ready

  È un bit di sola lettura. Indica lo *stato* della periferica, una volta attivato (*esempio: tasto premuto sulla tastiera*) rimane attivo finchè il *processore* non "interviene".

Prestazioni

Per valutare l'efficienza della gestione delle operazioni legate alle periferiche si utilizzano due parametri:

- Banda passante (**bandwidth**)

  Indica la quantità di dati che si può trasferire per unità di tempo, è una *misura di flusso*
- Latenza (**latency**)

  Indica il tempo che intercorre tra l'istante in cui la periferica è pronta per il trasferimento dati e l'istante in cui tale trasferimento si completa

Conversazione CPU - Periferiche

I/O Gestito da programma

Per "*conversare*" con le *periferiche*, il *processore* segue questo *procedimento*:

1. La CPU interroga lo stato della periferica
2. La periferica restituisce il suo stato

   Bit di *Ready* del *Registro* di Stato
3. Se la periferica è pronta per la trasmissione, la CPU richiede il trasferimento di dati

   In caso contrario, si ripete dal passo 1.
4. Avviene il trasferimento

Valutazione

Con questo metodo, viene impegnato molto tempo della CPU (che si occupa unicamente di tutte le operazioni di richiesta trasferimento finchè la periferiche non risulta pronta), per cui si hanno svantaggi legati alla **bandwith**.

D'altro canto, si ha una rapida risposta all'attivazione del *bit* di *Ready* della periferica, minimizzando la **latenza**.

I/O Gestito dagli interrupt

Fri, 20 May

Precedentemente al trasferimento dati avviene una "*predisposizioni al trasferimento*" che imposta la direzione di trasferimento e la dimensione. Avvenuta la predisposizione, invece di entrare nel ciclo di *busy wait*, la CPU si occpua di altro.

In modo **asincrono**, la periferica comunica di essere pronta per il trasferimento, e l'esecuzione viene *interrotta* e passata al **gestore delle eccezioni** (predisposto in precedenza), *eventualmente disabilitando le eccezioni* (in una situazione realistica in cui il tempo di esecuzione del *gestore* è molto basso e quindi è improbabile una perdita dati dovuta ad un'interruzione che avviene durante quel tempo).

Gestire la possibilità di eccezioni tramite le maschere di interrupt è detto "gestione software" delle priorità. Esiste anche una "Gestione hardware" in cui le priorità sono cablate nella macchina.

L'istruzione di `ERET` conclude il *gestore delle eccezioni* riabilitando gli interrupt, e consente il ritorno ad un eventuale **eccezione pendente** (nel caso in cui si fosse innestate eccezioni durante l'esecuzione del gestore).

In *MIPS*, i registri `$Kx` sono utilizzati liberamente dal gestore delle eccezioni.

Vettorizzazione

Il gestore delle eccezioni necessita di un meccanismo di identificazione delle perifetiche per capire quale periferica ha lanciato l'interrupt. Tale meccanismo veien chiamato **vettorizzazione**.

Si introduce a livello hardware un *set di fili* aggiuntivo al *BUS* di controllo, in modo tale che un interrupt "porta con se" anche un *valore identificatore*. Sommando questo *id* ad una **base del vettore delle interruzioni** ("offset", salvato in memoria) si ottiene una posizione del vettore che indica l'indirizzo di inizio del gestore della periferica indicata dall'identificatore.

Valutazione

Rispetto all'approccio di gestione da programma, ogni ciclo base si eseguono molte più operazioni per effettuare il trasferimento del dato dalla periferica, ma dal'altro canto si evita di entrare in un ciclo di *busy wait*.

I/O Direct Memory Access [DMA]

L'idea alla base della tecnica DMA è di *liberare completamente* la CPU dal trasferimento dati, lasciandole solo il processo di **predisposizione al trasferimento**, in modo da ottere **prestazioni elevate** sia in termini di **latenza** che di **bandwith**, aggiungendo *complessità* a livello *hardware*.

Secondo questo approccio, la periferica diventa **BUS Master**, ovvero avere la possibilità di leggere e scrivere sulla memoria. Si crea così la possibilità di *conflitto* per l'accesso al BUS.

Esiste una *ciruciteria terza* che si chiama **BUS Arbitrer** che garantisce l'accesso agli attori che che lo chiedono, uno alla volta.

L'accesso diretto in memoria è gestito da un modulo denominato **DMA Controller**.

Si aggiungono a `Status` e `Cause` due registri periferica, "`Dove`" e "`Quanti`" che indicano rispettivamente il target della periferica e la dimensione dei dati. Tali registri vengono aggiornati *elettronicamente* ad ogni passo elementare del trasferimento.

Dopo che la CPU completa la *predisposizione al trasferimento*, si fa "partire" il trasferimento, cambiando un bit nel registro di `Status` della priferica che indica l'inizio dell'attesa per il trasferimento. Dopo aver terminato la predisposizione, la CPU è libera, ma anche ignara del progresso del trasferimento.

Una volta finito il trasferimento (cosa di cui ci si accorge sempre a *livello elettronico*), viene generata una **interrupt request** per dire al livello logico che il trasferimento si è concluso.

Valutazioni

L'approccio DMA presenta effettivamente dei vantaggi sia a livelllo di *latenza* che di *bandwith* rispetto sia alla *gestione da programma* che alla *gestione tramite interruzioni*, aggiungendo *complessità* a livello **hardware**.