Esempio (con simulatore) di realizzazione di Input/Output con interruzioni

Indice

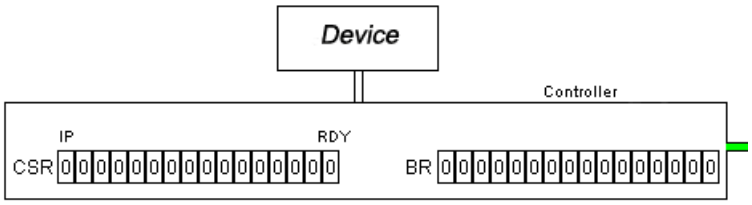
Introduzione
La gestione delle interruzioni
Rilevamento interruzioni e chiamata gestore
Il gestore delle interruzioni
Il controller della tastiera
Il driver della tastiera
Il controller della stampante
Il driver della stampante

[⇑] Introduzione

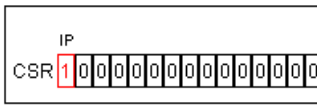
Questa è una delle tante possibilità di avere I/O gestito da interruzioni nel livello ISA didattico del testo, IJVM. Ovviamente le cose non sono esattamente come nella realtà per ragioni di semplicità.
Alla macchina del nostro livello ISA supponiamo di poter connettere fino a 8 "devices"(dispositivi, periferiche).

In questo esempio abbiamo una tastiera e una stampante

Supponiamo di utilizzare una politica di Memory Mapped I/O, cioè i registri relativi a dispositivi di I/O si possono leggere e scrivere con normali operazioni di lettura e scrittura in memoria (v. 3.7.2 del testo). Nel nostro esempio supponiamo che l'elettronica di ogni device disponga di due registri di 16 bit (un Buffer Register (BR) ed un Control Status Register (CSR)). Questi, siccome abbiamo memory mapped I/O, pur appartenendo fisicamente al device, sono visti dalla macchina come due parole di memoria e come tali possono essere utilizzate dalla CPU.



Quando un device necessita di "attenzione" da parte della CPU (vuoi per segnalare semplicemente un evento, vuoi perché desidera che la CPU svolga per conto del device alcune operazioni) questo pone ad uno il bit 15 del suo CSR (detto IP, Interrupt Pending), fa richiesta cioè di un'interruzione. L'hardware copierà il bit IP in OPC

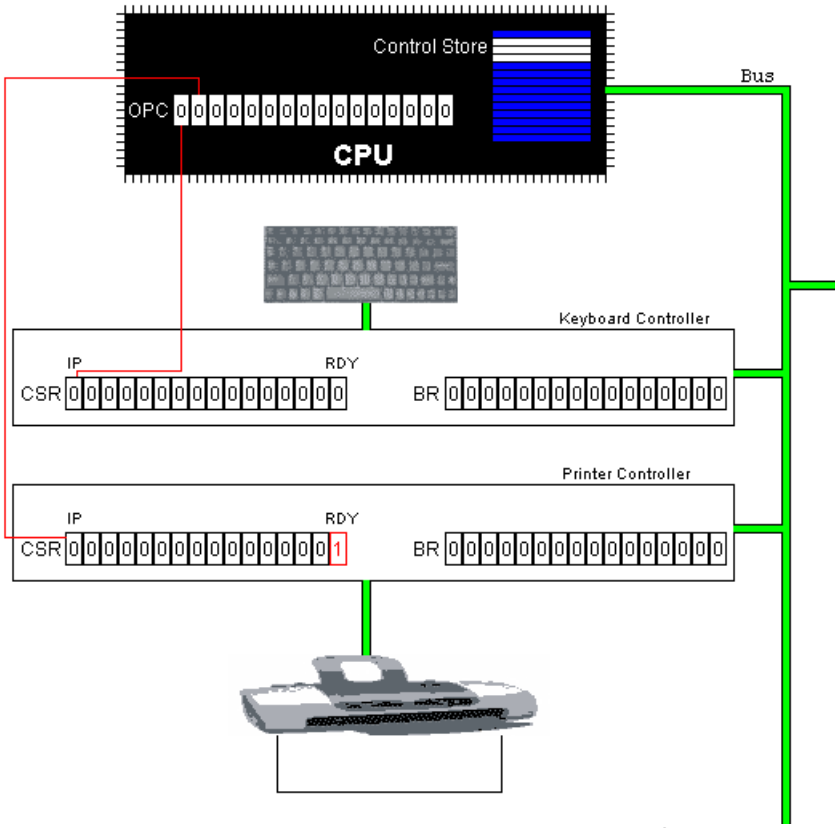


È compito dell'interprete del linguaggio macchina accorgersi della presenza di richieste da parte dei vari devices. Se il linguaggio macchina (IJVM o un altro) è realizzato via software sopra Mic-1, sarà il codice del microinterprete Mic-1 a realizzare le operazioni da fare in presenza di segnali di interruzione. Vediamo come avviene la gestione delle interruzioni.

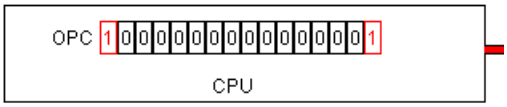
[⇑] La gestione delle interruzioni

La gestione delle interruzioni sarà demandata ad una procedura software (nel nostro esempio nel linguaggio che è realizzato sopra Mic-1). La presenza di richieste di interruzioni può essere "sentita" dalla nostra CPU poiché supponiamo che l'hardware della macchina faccia sì che al termine di un ciclo di clock il bit 15 del CSR del device i (0<=i<=7) venga copiato nel bit (15-i) del registro OPC (in particolare, nel nostro esempio, il bit 15 del CSR della tastiera viene copiato nel bit 15 di OPC, il bit 15

del CSR della stampante viene copiato nel bit 14 di OPC). OPC è uno dei registri della macchina Mic-1 che in questo caso supponiamo non abbia altro utilizzo.



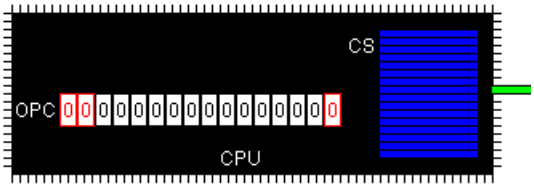
Per far sì che la macchina possa lavorare ad interruzioni disabilitate (cioè non sempre le richieste di interruzione debbano esser "sentite" dalla CPU), utilizzeremo il bit 0 del registro OPC per indicare ciò, e praticamente OPC[0]=0 abilitiamo le interruzioni e OPC[0]=1 le disabilitiamo.



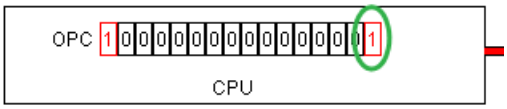
È compito del microinterprete accorgersi della presenza di richieste di interruzione.

[⇑] Rilevamento interruzioni e chiamata del gestore

L'interprete di IJVM è realizzato in Mic-1 (per questo viene chiamato, anche nel testo, microinterprete). Le microistruzioni che lo compomgono sono contenute nel Control Store (CS).



Il microinterprete dovrà, all'inizio di ogni ciclo di fetch-execute, controllare per prima cosa che le interruzioni siano abilitate(OPC[0]=0).



In caso positivo, e se è presente almeno un'interruzione, vengono eseguite le seguenti operazioni (N.B. per semplicità nella rappresentazione grafica, OPC si suppone lungo 16 bit):

- vengono salvati sullo stack i registri che contengono le informazioni sullo stato della computazione corrente, uno dei quali è sicuramente PC (Program Counter)

- viene salvato sullo Stack il contenuto del registro OPC. Questo permetterà al gestore delle interruzioni di poter leggere i valori dei bit contenuti in OPC al momento del rilevamento della presenza di interruzioni. (Questo salvataggio di OPC è indispensabile nel caso il gestore sia realizzato in un linguaggio che, come IJVM, non rende possibile l'accesso diretto ai registri di Mic-1. Infatti non ci sarebbe altrimenti modo di controllare il contenuto di OPC. Ovviamente se realizzassimo il gestore delle interruzioni in microcodice Mic-1 anziché come programma IJVM, allora le cose cambierebbero)
- viene caricato in PC il contenuto della locazione 255 (decimale). Si suppone che in memoria la locazione 255 contenga l'indirizzo del gestore delle interruzioni, il programma in linguaggio macchina che appunto gestisce le interruzioni.
- vengono disabilitate le interruzioni
- si salta al microcodice che cura il fetch della prossima istruzione.

Come già detto, il linguaggio macchina considerato può benissimo essere diverso da IJVM, l'importante, nel nostro esempio, è che sia realizzato per interpretazione sopra Mic-1.
Da notare che, al contrario dell'esempio del testo (sezione 5.6.5), più vicino alla realtà, non si parla di interrupt vector (numero intero messo sul bus dal controllore del dispositivo quando questo si accorge che la CPU ha rilevato la richiesta di interrupt), poiché abbiamo supposto di avere linee multiple di interrupt, attraverso le quali è possibile rilevare l'identità di chi ha richiesto un interrupt controllando il contenuto di OPC.
Vediamo come andrebbe modificato il microcodice della Figura 4.17 del testo per far sì che ci si accorga della presenza di richieste di interruzioni e, nel caso, si eseguano le operazioni descritte in precedenza.
Per semplicità supponiamo di avere un linguaggio Mic-1 esteso nel quale sia possibile, per esempio, fare dei salti condizionati le cui condizioni sono bit di OPC o espressioni booleane che li utilizzano (con la notazione OPC[X] indichiamo il bit X di OPC). La modifica consiste in pratica nell'aggiungere le seguenti microistruzioni prima di quella con etichetta Main1:

```
int1 if opc[0] goto main1;/ controllo se le interruzioni sono abilitate. Estendiamo il mic-1 con il costrutto
if (OPC[x]) goto main1 per indicare che si deve saltare alla etichetta main1 se il bit presente in OPC[x] è
pari a uno, cioè in questo caso se le interruzioni sono disabilitate.

int2 if (not(opc[15]) and not(opc[14]) ... and not(opc[8])) goto main1; / controlla se ci sono interruzioni;
se tutti i devices non fanno richiesta di interruzione si deve saltare alla etichetta main1

int3 SP=MAR=SP+1;

int4 MDR=PC; wr; /salva il PC del programma corrente sullo stack (*).

int5 SP=MAR=SP+1;

int6 MDR=OPC; wr; /salva OPC sullo Stack

int7 MAR=OxFF; rd; /iniziamo la lettura del contenuto della locazione 255. La locazione 255 contiene
l'indirizzo del gestore delle interruzioni

int8 OPC[0]=1; /disabilito gli interrupt, ci penserà il gestore degli interrupt a riabilitare;

int9 PC=MDR; fetch; /inizio l'estrazione della prima istruzione del gestore delle interruzioni;

int10 /attendo il completamento del fetch precedente

main1 PC=PC+1; fetch; goto(MBR); / tutti i goto seguenti nel codice del microinterprete che prima, nel caso
senza interruzioni, andavano a main1, ora devono andare a int1 /

:

resto del microinterprete

:
```

(*) Le informazioni da salvare sullo stack per poter ripristinare lo stato della computazione corrente sono molte di più. Possiamo supporre che tali informazioni vengano salvate dal gestore delle interruzioni, anche se in realtà il gestore, se è scritto in IJVM, non ha accesso diretto ai registri di Mic-1.

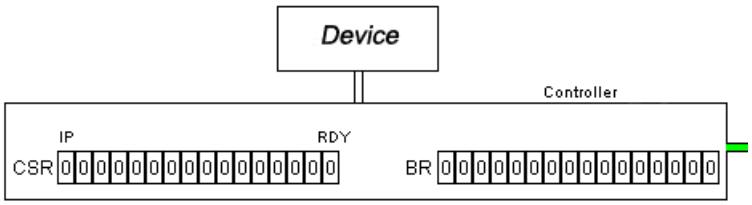
Se l'interprete quindi, ad interruzioni abilitate, trova 1 nei bit 0-7 di OPC, chiama il gestore delle interruzioni. Vediamo nel seguito cosa succede nel gestore.

[⇑] Il gestore delle interruzioni

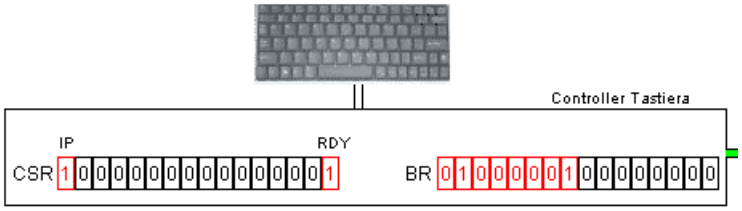
Descriviamo ora il comportamento del Gestore delle interruzioni.
Supponiamo che il gestore delle interruzioni sia un programmino in linguaggio macchina, IJVM o quello che c'è (potevamo anche pensare di averlo realizzato con un microprogramma Mic-1, nel qual caso andava modificato qualcosa nelle microistruzioni sopra).
Il gestore deve mandare in esecuzione i driver (altri programmini in linguaggio macchina) dei devices che hanno fatto richiesta di interruzione. L'ordine di esecuzione dei driver non è arbitrario, ma seguirà una certa politica (si potrebbe anche pensare che i devices hanno una certa priorità fra di loro e che queste priorità siano descritte in una particolare sezione di memoria che viene caricata, per esempio, quando il sistema viene inizializzato).`Nel nostro caso la stampante ha priorità maggiore`
Una volta che tutte le richieste di interruzione siano state servite, il gestore riabiliterà le interruzioni e passerà il controllo al programma che è stato interrotto. Per permettere di far ciò, supponiamo che il set di istruzioni del nostro linguaggio macchina contenga l'istruzione RETI (RETurn from Interrupt). L'esecuzione di tale istruzione, ripristinerà lo stato della computazione interrotta (per esempio ripristinando il valore del PC che era stato inserito sullo Stack) e riabiliterà le interruzioni. Ovviamente se volessimo programmare il sistema in modo che anche il gestore e/o i driver siano interrompibili, come descritto dalla fig.5.46 del testo (5.44 nella IV edizione), dovremmo modificare il microcodice in modo che non disabiliti le interruzioni. Anche l'istruzione RETI andrebbe modificata in modo da non abilitare le interruzioni. L'abilitazione e la disabilitazione potrebbero essere effettuate da due apposite istruzioni: EINT (Enable INTerrupt), DINT (Disable INTerrupt).
Vedremo ora nel seguito il comportamento dei controller e dei driver di tastiera e stampante.

[⇑] Il controller della tastiera

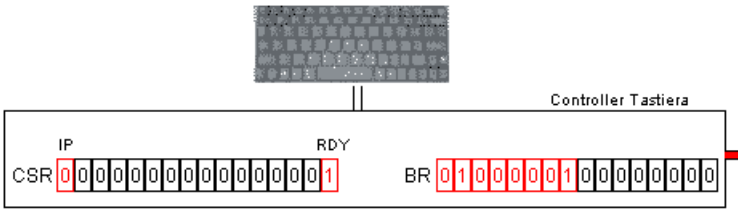
Un controller non è altro che l'hardware che praticamente interfaccia il device vero e proprio con il resto del sistema (possiamo pensare al chip di I/O come parte di esso); gestisce infatti l'accesso al bus (vedi sezione 3.7.1 del testo). Ogni controller nel nostro esempio avrà un registro CSR ed un registro BR.



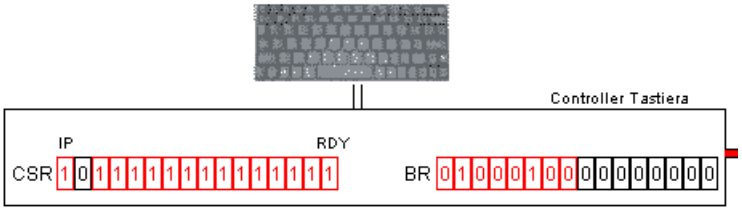
Quando un tasto viene premuto sulla tastiera, il byte contenente il codice ASCII del carattere corrispondente al tasto premuto viene inserito nei bit 15-8 del BR della tastiera. `La figura che segue rappresenta lo stato dei registri dopo la pressione del tasto "A" - carattere 65 della tabella ASCII - 01000001`



Vengono nello stesso momento posti ad 1 i bit 15 (detto IP) e 0 (detto RDY) del CSR.
Il bit IP indica che viene fatta richiesta di un'interruzione dalla tastiera, mentre RDY indica la presenza di un carattere nel BR. (I bit IP e RDY non hanno sempre lo stesso valore, poiché il driver, appena inizia ad operare, pone IP a 0 per indicare che l'interruzione richiesta è stata accolta ed è in fase di elaborazione).



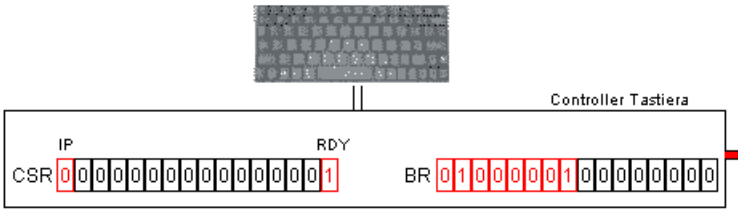
Ovviamente è possibile che in precedenza un tasto sia stato premuto e che ancora la gestione dell'interruzione relativa non sia terminata. È possibile cioè che il controller, quando si preme un tasto, trovi ad uno il bit RDY. In questo caso possiamo supporre che quel che viene fatto sia semplicemente di inserire nei bit 13-1 del CSR un codice di errore



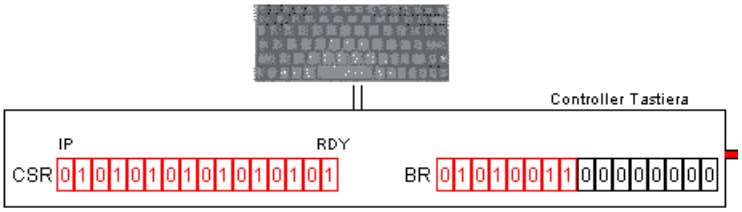
allo scopo di segnalare al driver una condizione di anomalia. Quando il microinterprete accoglierà la richiesta di interruzione chiamerà il driver della tastiera.

[⇑] Il driver della tastiera

I driver sono routine in linguaggio macchina, di livello ISA cioè, come IJVM.
Il driver della tastiera, una volta chiamato, per prima cosa azzererà IP per segnalare al device che l'interruzione è stata accolta e che le operazioni relative stanno avendo luogo



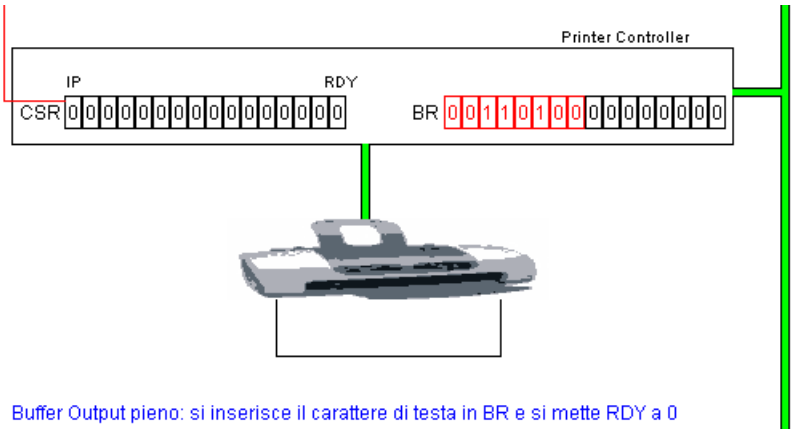
(volendo, si potrebbe anche supporre che l'hardware sia fatto in modo tale che una lettura del BR causi automaticamente l'azzeramento di IP). Poi preleverà il carattere che si trova nel BR e lo inserirà in un buffer circolare che contiene appunto i caratteri letti dal driver e che possono essere utilizzati da qualsiasi programma attraverso una procedura GETCH (tale procedura non farà altro che prelevare un carattere dal buffer circolare del driver della tastiera, magari ponendolo sullo Stack nel caso il linguaggio macchina sia IJVM). Il driver dunque, per memorizzare i caratteri letti deve anche gestire il buffer circolare. Il driver della tastiera potrebbe trovare il buffer pieno, nel qual caso il carattere viene perso ed inserisce nel CSR un codice d'errore.



Dopo aver fatto quanto specificato sopra, il driver controllerà se nei bit 13-1 del CSR sia presente un codice di errore. A questo punto il driver può porre a 0 il bit RDY del CSR (indicando così che il carattere è stato prelevato), abilitare nuovamente le interruzioni e ritornare dall'interruzione.
Ricordiamo che, trattandosi di memory-mapped I/O, le operazioni sui registri vengono effettuate come normali letture/scritture in memoria.

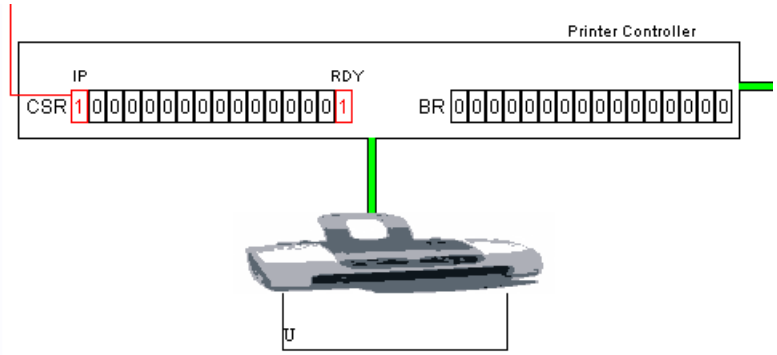
[⇑] Il controller della stampante

Supponiamo di voler utilizzare il driver della stampante per "stampare" i caratteri presenti nel buffer circolare che contiene i caratteri di output. Quando il buffer di output è pieno viene inviato il carattere di testa del buffer al controller della stampante e viene messo RDY a 0. Nella figura che segue viene inviato il carattere '4'.



Buffer Output pieno: si inserisce il carattere di testa in BR e si mette RDY a 0

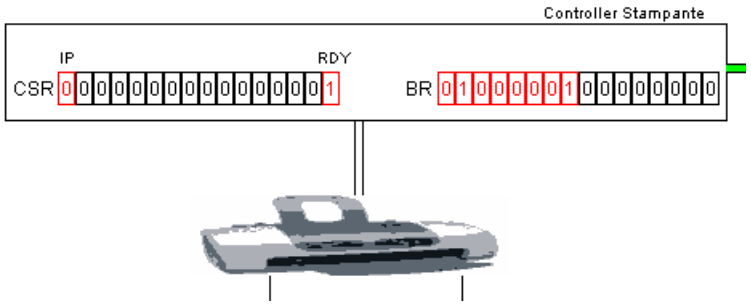
Trovando RDY a zero la stampante inizia a stampare; intanto chi ha messo il carattere in BR (il programma oppure il driver) continua la sua esecuzione.



Quando la stampante termina la stampa del carattere, mette IP e RDY a 1 per chiedere interruzione ed ottenere un altro carattere dal driver stampante(se ce ne sono nell'output buffer).

[⇑] Il driver della stampante

Il driver della stampante, una volta chiamato:
per prima cosa azzera IP per segnalare che l'interruzione è stata accolta e che le operazioni relative stanno avendo luogo;



preleva il carattere in testa all'output buffer e lo mette in BR (nella figura sopra ha prelevato il carattere 'A', codice ASCII 65);
mette RDY a 0 per avviare la stampa;
il driver ha così finito, e si restituisce il controllo al gestore delle interruzioni.
Se, una volta chiamato, il driver trova l'output buffer vuoto si limita a mettere IP a zero e restituisce il controllo al gestore.