INTERRUZIONI
+
CONCETTO DI "*DAISY CHAIN*"

Questo modello di interazione tra ***processore e dispositivi*** non sfrutta il "test" continuo del valore del Flip flop di status dei vari devices connessi sul bus I/O;

In questo modello di interazione sarà il dispositivo che solleverà una "***richiesta di interruzione***" quando sarà pronto ad interagire con il processore: il dispositivo può aver finito di processare i dati che il processore gli aveva precedentemente mandato (interfaccia di output) oppure può aver finito di preparare i dati che il processore deve andare a leggere (interfaccia di input).

In che modo questo accade?

Discorsi da BAR.

IL DISPOSITIVO ALZA UNA BANDIERINA - IL FLUSSO D'ESECUZIONE VIENE INTERROTTO TEMPORANEAMENTE - SI ASSEGNA IL CONTROLLO AL "GESTORE DELL'INTERRUZIONE" O DRIVER CHE PERMETTE DI GESTIRE L'INTERAZIONE CON IL DISPOSITIVO. AL TERMINE DELL'ESECUZIONE DEL GESTORE, IL PROCESSORE RIPRENDE LA NORMALE ESECUZIONE DEL PROGRAMMA.

Quando sorseggi il tuo caffè in un tipico *bar italiano* anche se pronunci un discorso molto preciso ed elegante comunque qualcuno deve trovare delle criticità.

- Prima criticità: ci possono essere più dispositivi pronti allo stesso istante di tempo.

Il processore può eseguire un solo driver per volta e per identificare quale driver attivare deve identificare quali dispositivi hanno sollevato la richiesta di interruzione.

- Soluzione: assegnare una priorità alle richieste di interruzione. I dispositivi sono inseriti in una catena: *DAISY CHAIN*. I dispositivi vicini hanno priorità maggiore.

  - Seconda criticità: Identificare i dispositivi che hanno sollevato richieste di interruzione.

- Soluzione: Realizzare un protocollo basato su firmware per l'identificazione dei dispositivi che hanno sollevato

Basta chiacchiere, usciamo dal bar. 

Nella pratica per poter gestire un interrupt è necessario assicurare due concetti fondamentali:

1. La CPU deve essere disponibile per essere interrotta il che implica dire che gli interrupt devono essere abilitati.

Il registro FLAGS - in questa slide la sua conoscenza sarà scontata - possiede alcuni bit di "controllo" e il loro scopo è quello di modificare specifiche modalità di esecuzione del processore.
In modo particolare il flag IF (Flag di interrupt) determina se gli interrupt sono abilitati o meno.
(NOTA: *Il valore di questo flag di controllo può essere modificato utilizzando le istruzioni **CLI** e **STI***).
All'atto dell'avvio del sistema il flag IF = 0 pertanto, se si desidera utilizzare il supporto degli interrupt **il software deve abilitarlo** esplicitamente.
Se il valore di IF = 1 allora il processore risponde alle richieste di interruzione.

2. L'interruzione del programma che attualmente è in esecuzione non deve influire sulla sua correttezza quando il controllo ritornerà all'istruzione successiva del frammento in cui il programma si era interrotto.

In modo particolare al programma che correntemente è in esecuzione viene effettuato un salvataggio dello stato attuale - CONTESTO DI ESECUZIONE (INTERRUPT FRAME).

Si procede verso l'identificazione del programma di servizio relativo alla periferica che ha generato la richiesta di interruzione (driver).

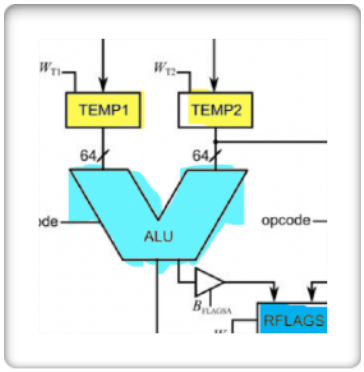
Si esegue il programma di servizio (driver).

Si riprende ad eseguire il programma che precedentemente era stato interrotto.

Per garantire la correttezza d'esecuzione di un frammento di codice è necessario che la CPU esegua in modo "non interrompibile": **ATOMICO**!

Immaginiamo che il processore stia eseguendo questo microprogramma:
"addl %eax, %ebx".

Se il processore interrompesse immediatamente l'esecuzione del microprogramma per attivare il gestore delle interruzioni potrebbero verificarsi alcune complicazioni.
Le micro-operazioni associate a questo microprogramma richiedono il caricamento del valore dei due operandi nei registri TEMP1 e TEMP2 (reg. invisibili al programmatore) per poter effettuare la somma.

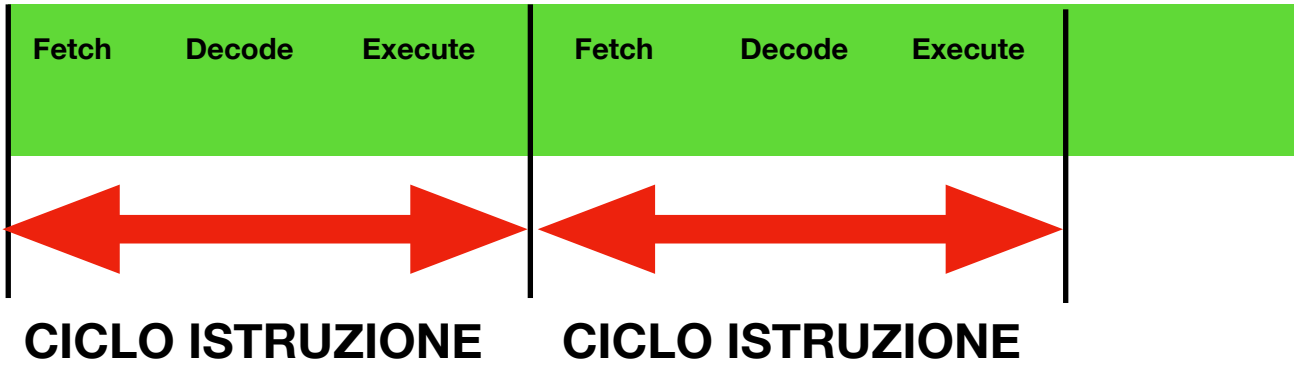


Allo stesso tempo però il programma di servizio (driver) "potrebbe" contenere al suo interno un'istruzione di questo tipo:
"sub %rcx, %rdx".

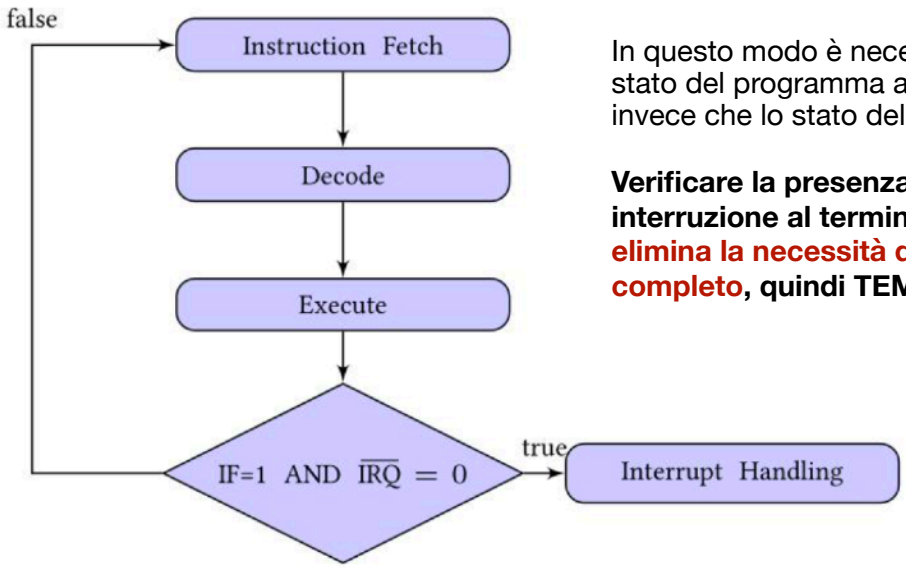
Questa istruzione utilizza gli stessi registri invisibili dell'unità di processamento TEMP1 e TEMP2 per effettuare la sottrazione.
Quindi supponendo che il programma di servizio parta nel mentre è in esecuzione una micro-operazione del microprogramma principale che sta usando i registri invisibili, al termine dell'esecuzione del driver il microcodice del programma principale troverebbe modificati i valori di TEMP1 e TEMP2, portando ad un possibile risultato errato.

SIAMO IN PRESENZA DI UNO SCENARIO INACCETTABILE.

L'unità di controllo del processore una volta che ha completato l'esecuzione di un microprogramma di un programma principale ossia un ciclo macchina (intervallo necessario ad eseguire una sola fase tra fetch, decode ed execute) ritorna al microprogramma che implementa la fase di fetch successiva.



Il ciclo istruzione l'intervallo di tempo necessario ad eseguire una istruzione nella sua interezza.
E a seconda del tipo di istruzione possono essere necessari **più cicli macchina**.
Se il supporto per le interruzioni è abilitato (IF = 1) il controllo delle presenza di una richiesta di interruzione **viene effettuato solo alla fine di ogni ciclo istruzione**.



In questo modo è necessario salvare solo lo stato del programma attualmente in esecuzione invece che lo stato del microprogramma.

Verificare la presenza delle richieste di interruzione al termine del ciclo istruzione elimina la necessità di salvare il contesto completo, quindi TEMP1 e TEMP2.

Se IRQ" = 0 significa che c'è almeno un device che ha sollevato una richiesta di interruzione, e quindi IRQ = 1.

CAMBIO DI CONTESTO

- Il cambio di contesto è l'insieme delle attività eseguite dalla CPU per interrompere il flusso d'esecuzione corrente ed attivare il driver.
- Il contesto di esecuzione viene salvato su *stack* (**interrupt frame**).

Se IRQ = 1 c'è un dispositivo che chiede attenzione al processore.
Il dispositivo dovrà essere identificato.

Cosa fa la CPU?
All'inizio IF = 1.
Prima di eseguire il salvataggio di contesto per eseguire il DRIVER, imposterà IF = 0, scriverà l'interruppe frame sullo stack, e solo a questo punto partirà l'esecuzione del driver. Durante il cambio di contesto IF viene azzerato dal firmware.

BISOGNA *MASCHERARE* TUTTE LE RICHIESTE DI INTERRUZIONE.

Ora abbiamo un driver in esecuzione - che dovrà sperabilmente terminare prima o poi.

Ora all'interno del driver BISOGNA cancellare il più presto possibile la richiesta di interruzione scrivendo IRQ" = 1.
Questo passaggio deve essere effettuato sul driver del dispositivo.
In sostanza bisogna abbassare la bandierina del device con cui si è stabilità l'interazione.
Quindi si resetta il Flip flop INT_REQ con IRQ = 0, IRQ" = 1.
Se questo non viene fatto il processore sarà portato a pensare che sia arrivata un'altra richiesta di interruzione dello stesso dispositivo e conseguentemente scriverà un'altro interrupt frame per la stessa richiesta di interruzione.
Detto in altre parole per evitare che il processore attivi lo stesso driver dopo aver eseguito un'istruzione di IRET.
Il flip flop INT_REQ è scrivibile dal processore utilizzando un'istruzione OUT: una scrittura diretta sul flip flop ne resetta il valore in modo da poter cancellare la causa dell'interruzione. Questo lo deve fare il software.

Dopo che nel driver è stata cancellata la richiesta/causa di interruzione si può rendere il driver interrompibile con un'operazione di STI che abilita la ricezione di interruzioni.

Così tutto ciò che rimane è solo un "normale" processamento dati che si può eseguire con IF = 1 cosicché nella bottom del driver l'esecuzione del codice potrà ancora essere interrotta per servire un'altra richiesta di interruzione.

Riguardo al contesto da salvare sullo stack (interrupt frame) - cosa verrà effettivamente salvato?

1. **Registri di uso generale.**
2. **Registro dei flag.**
3. **RIP.**

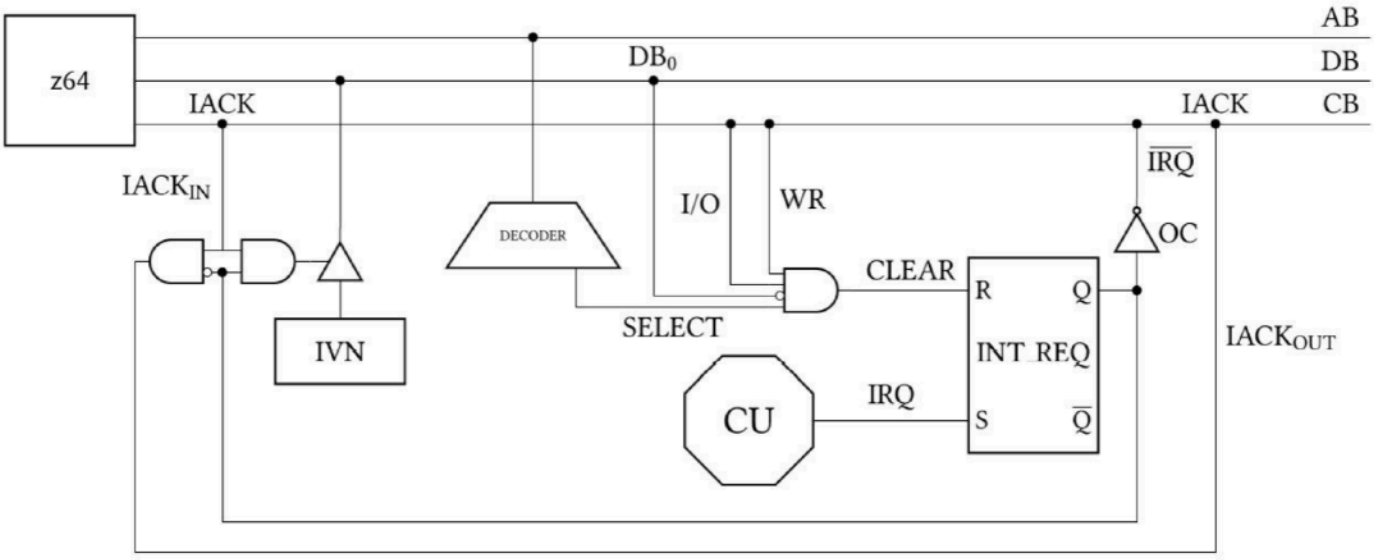
Torniamo ai devices.

Abbiamo sin ora specificato che il dispositivo deve interrompere il processore e per fare questo deve alzare una bandierina.

Il discorso da bar della bandierina è bello fin tanto che siamo su un tavolo a bere e a farci due risate.
Quì siamo al corso di Calcolatori Elettronici e non siamo a bivaccare.

Viene utilizzato il flip flop INT_REQ che bufferizza una richiesta di interruzione. Q = IRQ è una sorta di "bandierina".
Se IRQ = 1, IRQ" = 0 allora il dispositivo (almeno uno) ha sollevato la richiesta di interruzione e deve essere eseguito il suo driver.
Bisogna solo capire qual'è il dispositivo che l'ha sollevata.
IRQ viene impostato dalla CU del dispositivo quando il device richiede l'attenzione - pertanto la CPU riceve la richiesta di interruzione ed è pronta ad interagire con un device.

IL PROCESSORE COMUNICA SUL CONTROL BUS CHE È PRONTO AD ESEGUIRE UNA RICHIESTA DI INTERRUZIONE E PERTANTO ABILITA IL SEGNALE DI "*IACK*" CHE ARRIVA AL CIRCUITO COMBINATORIO DEL DISPOSITIVO.



Questo segnale di IACK non si propaga lungo tutto il control bus ma arriva solo al primo dispositivo che incontra e, semmai dovesse servire, sarà il dispositivo che lo propagherà al dispositivo successivo.
Ora se la bandierina è alzata (Q = IRQ = 1, quindi il device ha sollevato una richiesta di interruzione) e IACK = 1, l'AND tra IACKin e IRQ vale 1 (and di desta) e pertanto si apre un buffer three-state che scrive sul databus il contenuto del registro IVN, in questo modo la periferica si è identificata e il processore capisce che è questo il dispositivo che deve servire.
Viceversa se il dispositivo non ha mandato nessuna richiesta di interruzione (Q = IRQ = 0) l'and di destra vale 0 e il buffer three-state non viene pilotato ma l'and di sinistra vale 1 il segnale di IACK passa al dispositivo successivo nella Daisy chain.

In questo modo potrà essere eseguito il driver all'interno della Interrupt Vector Table (IVT) secondo lo schema delle interruzioni vettorizzate.

ACCESSO : DEVICEIVN x 8

Attenzione: Per consentire una rapida attivazione del driver il processore omette il salvataggio dei registri di uso General Purpose sullo stack, lasciando al codice del driver il compito di salvare sullo stack (e di conseguenza ripristinare prima dell'esecuzione di IRET) tutti i registri General Purpose che verranno effettivamente utilizzati.