Architettura degli elaboratori lezione 16

📚 Appunti di Davide Scarlata 2024/2025

👩‍🏫 **Prof:** Claudio Schifanella
✉️ **Mail:** claudio.schifanella@unito.it
📌 **Corso:** C
🔗 **Moodle Unito**
🗓️ **Data:** 20/05/2025

Mappatura diretta : vantaggi e svantaggi

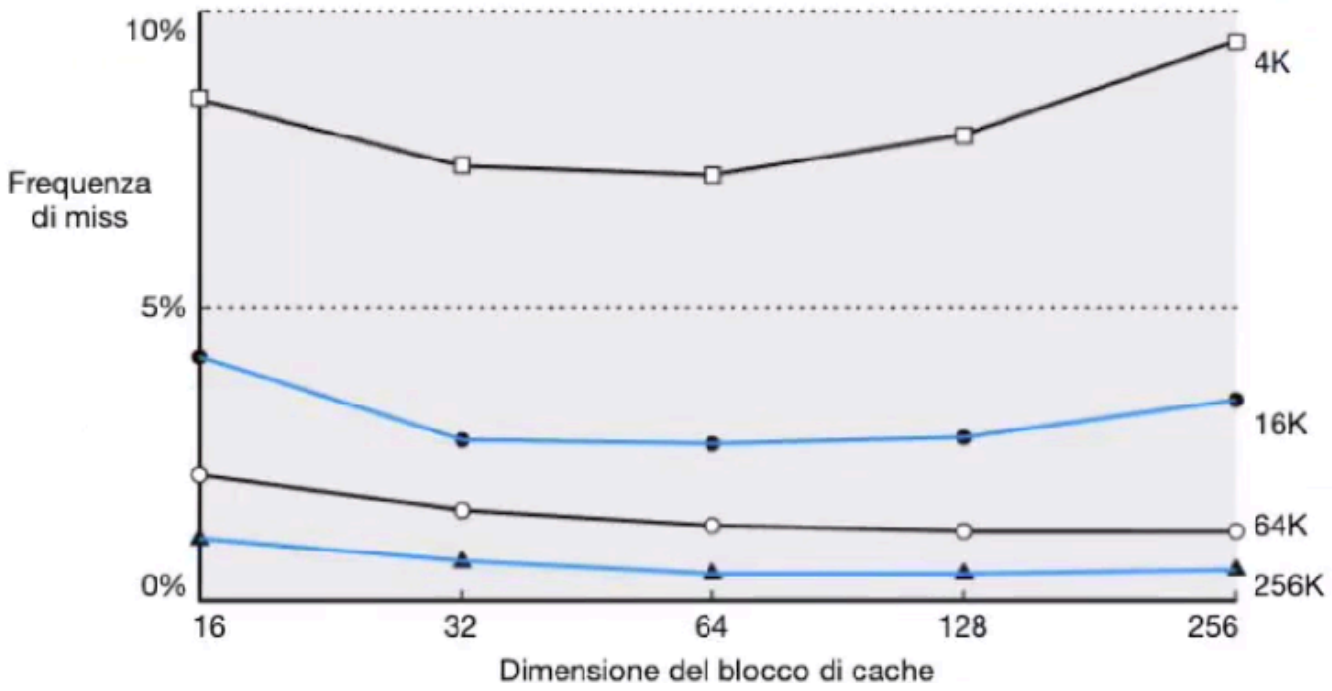
Vantaggi :

- Efficienza, è molto semplice passare dall'indirizzo del blocco dalla memoria all'indirizzo nella cache (basta considerare gli ultimi n bit e aggiungere il tag)

Svantaggi :

- Capita spesso che dei blocchi salvati sulla cache debbano essere sovrascritti. Questo è influito anche dalla grandezza dei blocchi.

Grafico per mostrare come aumentare la dimensione del blocco non è sempre la soluzione ottima :



Completamente associativa

Un altro metodo di mappatura dalla memoria cache. Questo metodo permette di scrivere nella cache in un punto qualsiasi.

C'é un contro, la ricerca va effettuata su tutti gli elementi della cache. Questo influisce sia la ricerca che la scrittura di un blocco. Quando vogliamo scrivere dobbiamo cercare un blocco sulla cache libero oppure dobbiamo trovare un blocco da sovrascrivere. Ci sono varie criteri che si possono seguire per sovrascrivere un blocco, ovvero :

- Il primo blocco scritto
- L'ultimo blocco scritto
- Il blocco meno letto fra quelli salvati
- Il blocco nella cache con l'accesso più vecchio (LRU - Last Recently Used)

Come possiamo scegliere quale usare? Molto semplicemente basta fare dei test di benchmark per capire quello più performante.

Abbiamo ancora il problema che con questa implementazione dobbiamo controllare cella per cella, come si può risolvere? Possiamo risolvere questo problema parallelizzando la ricerca con dei **comparatori** (i quali fanno aumentare i costi, i consumi, il calore).

Cache set-associativa

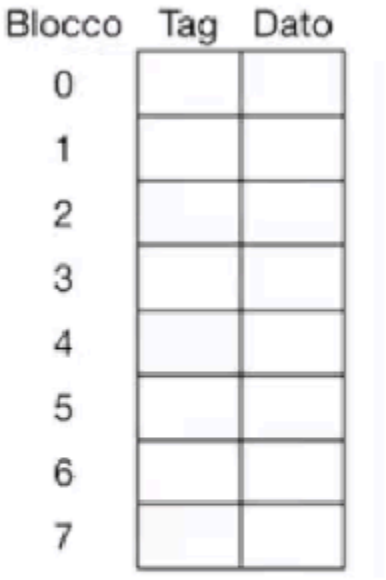
Quale è meglio tra il metodo "completamente associativa" e "mappatura diretta"? Attualmente, quasi mai, nessuna delle due, viene usata la "Cache set-associativa". Questo metodo è una via di mezzo tra le prime due.

Funzionamento

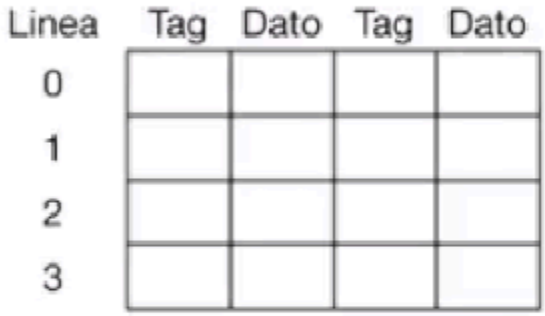
Ci sono delle linee che contengono più blocchi (detto numero di vie). Individuata la linea, il blocco viene ricercato all'interno degli slot della linea. Di seguito uno schema di molteplici costruzioni di questo modello :

Cache set-associativa a una via

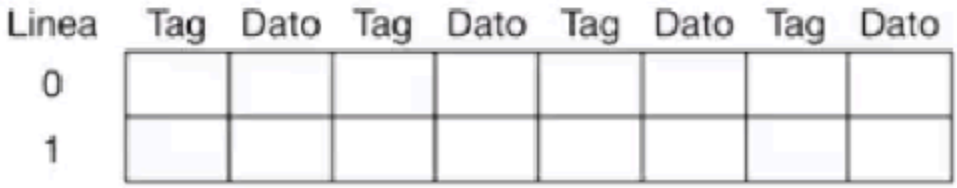
(a mappatura diretta)



Cache set-associativa a due vie



Cache set-associativa a quattro vie



La linea che dobbiamo scegliere è scelta tramite la mappatura diretta. Trovata la via corretta, si ricerca il dato all'interno della via usando l'algoritmo completamente associativo (ricerca esaustiva).

Un esempio delle prestazioni :

| Associatività | Frequenza di miss |

| :----------: | :--------------: |

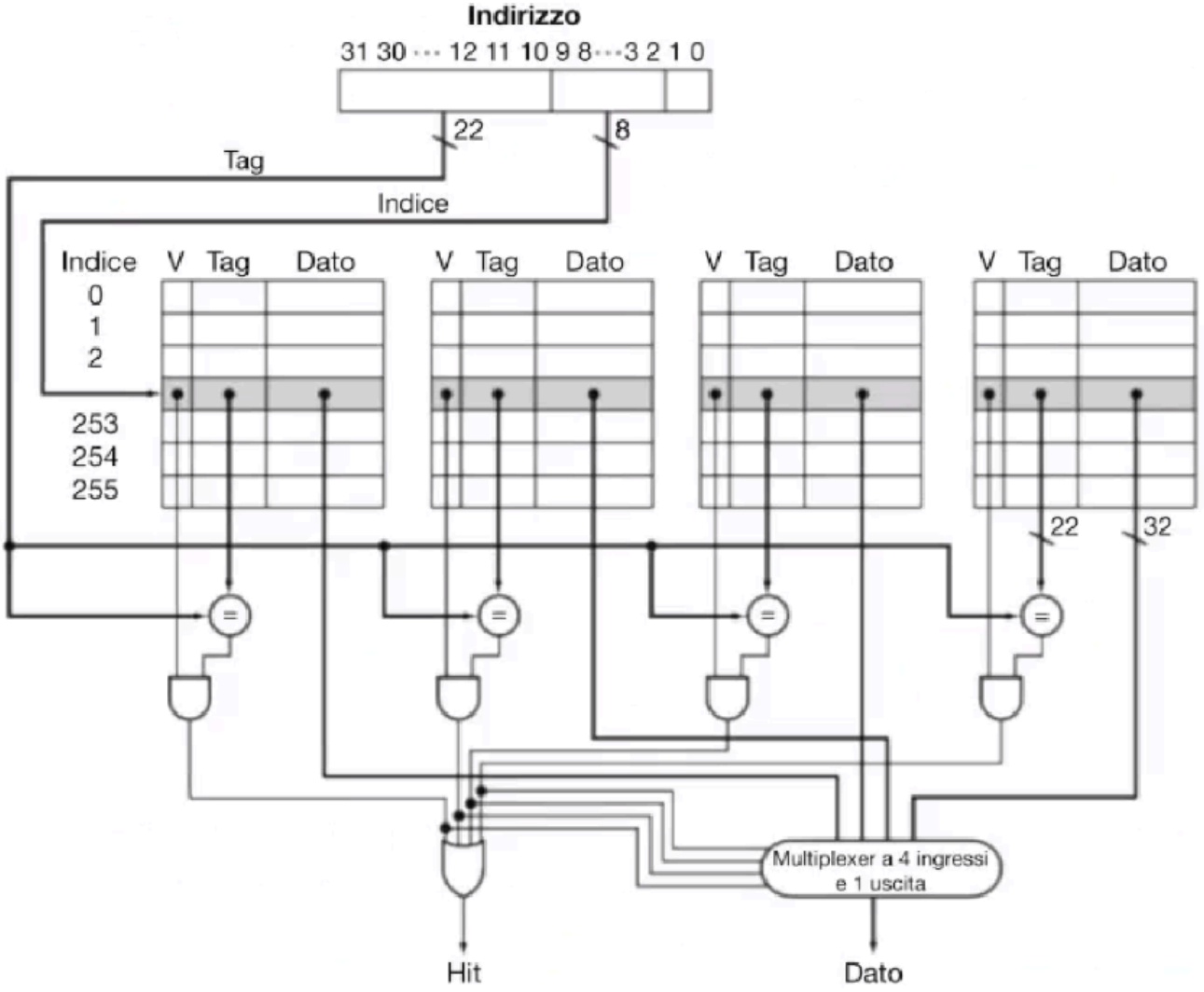
| 1 | 10.3% |

| 2 | 8.6% |

| 3 | 8.3% |

| 4 | 8.1% |

L'implementazione della cache sopra è la seguente :



Gestione delle miss

La gestione delle miss e hit necessita di una modifica all'unità di controllo del processore (approfondimento disponibile sul libro di testo 5.9, non richiesto).

Gestione della scrittura

Fino ad adesso abbiamo sempre parlato della lettura della RAM/cache e di come ottimizzarla, ora ci poniamo il problema della scrittura. La scrittura però porta dei nuovi problemi. Quando scriviamo (e il dato è in cache), dove scriviamo? in cache e poi riportiamo il dato nella RAM? non lo riportiamo? Che strategie possiamo usare?

Write-through

Ad ogni scrittura in cache, si modifica il valore anche in memoria. Se il dato da scrivere non è presente in cache, viene prima caricato, poi modificato in cache e in memoria.

Le prestazioni non sono molto buone (stiamo usando un buffer di scrittura).

Write-back

Il dato viene scritto solo in cache. La scrittura al livello inferiore avviene solo quando il blocco nella cache deve essere rimpiazzato. È indubbiamo più veloce, perché modificare la cache è molto più veloce e poi modifichiamo una sola volta la ram, però è anche più complesso da implementare.

Input e Output

La CPU comunica con i device di I/O tramite i controllori, che hanno il compito di trasformare i comandi della CPU in segnali elettrici per le periferiche e i segnali delle periferiche in dati per la CPU.

Ogni controllore ha al suo interno dei registri identificati da un indirizzo, che può :

- Appartenere allo stesso insieme di indirizzi di memoria (memory mapped I/O, dove l'OS decide l'indirizzo associato ad un controllore)
- Essere un indirizzo dedicato (isolated I/O).

La comunicazione con i controllori è simile agli accessi in memoria.

Registri dei controllori

Cosa possono contenere? Ecco :

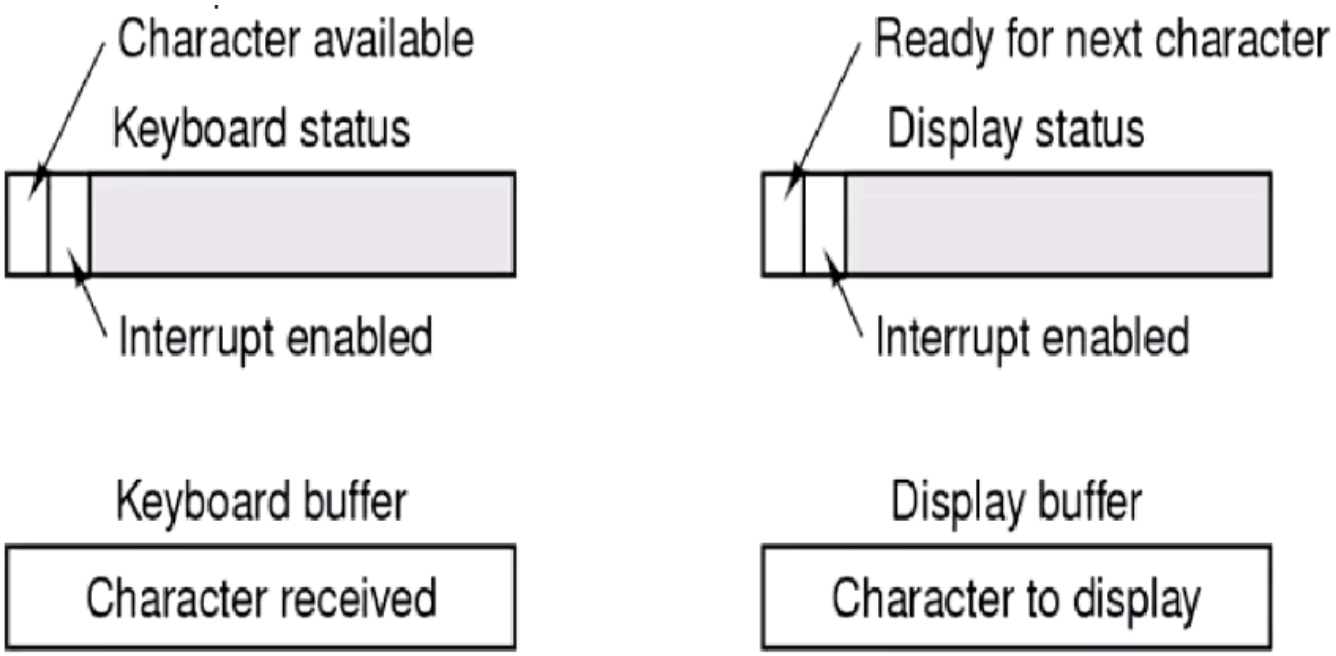
- dati (di ingresso e uscita). Es, per una stampante possono essere i dati da stampare.
- comandi (dalla CPU alla periferica). Es, i comandi di stampa.
- informazioni sullo stato della periferica (dalla periferica alla CPU). Es, informazioni sulla stato della stampante o sull'avanzamento dei lavori.

Modalità di input/output

I/O programmato con Busy waiting

La CPU controlla lo stato di avanzamento del I/O e lo stato della periferica ispezionando in un ciclo il bit del registro di stato del controllore READY fino a che questo non segnala di essere pronto per un nuovo comando di I/O.

Supponiamo di avere una tastiera. La CPU dovrà usare il tasto premuto sulla tastiera. Ogni tasto è codificato tramite un codice che sarà contenuto all'interno di un registro del controllore della tastiera e quando viene schiacciato deve arrivare alla CPU.



Come fa quindi la tastiera a sapere che è stato premuto un tasto? Un modo è proprio usando la tecnica di busy waiting, cioè la CPU chiede alla tastiera "è stato premuto un tasto?" e la tastiera risponde di conseguenza. Come avviene la richiesta? Tramite un bit di controllo, che nel caso di sopra è segnato in "keyboard status".

Questo controllo la CPU lo fa in modo ciclico. Viene detto "polling" (verifica ciclica appunto dei dispositivi di I/O tramite i bit di controllo).

L'esempio della tastiera vale per la lettura. Per la scrittura invece possiamo fare l'esempio di voler stampare qualcosa a video. Con lo stesso modo, la CPU chiede al display qualcosa tipo "sei pronto a stampare la prossima informazione" e poi il display risponde. Se dice sì, ottimo, altrimenti la CPU (come prima) torna dopo del tempo e gli chiede di nuovo.

Questo metodo fa sprecare delle risorse alla CPU, non è ottimale (ma può essere comunque usato in alcuni casi in cui questo schema risulta comodo).

Interrupt

Il dispositivo quando ha finito genera un segnale che avverte la CPU (tramite una linea di bus direttamente collegata alla CPU) di aver completato il proprio lavoro. La CPU può abilitare l'interrupt a 1 un opportuno bit.

Per poter sapere chi ha generato l'interrupt abbiamo bisogno di un "vettore di interrupt", il quale ci fa sapere chi ha generato un interrupt.

Funzionamento

L'interrupt interrompe il programma in atto e trasferisce il controllo ad un gestore di interrupt che eseguirà le azioni appropriate. Una volta terminata la gestione dell'interrupt, il controllo torna al programma interrotto.

Trasparenza

Gli interrupt sono asincroni rispetto al programma e devono essere gestiti in modo trasparente, ovvero, lo stato dell'esecuzione dopo la gestione dell'interrupt deve tornare esattamente come era prima dell'interrupt stesso. Quindi, lo stato dei vari registri deve tornare esattamente come prima (o salvando in memoria tutti i registri in memoria e poi ripristinarli oppure usando altri registri, detti "shadow registers" che servono per salvare una copia dei registri "primari" per poi ripristinarli).

DMA (direct memory access)

I/O programmato, ma svolto da un'apposito componente con accesso diretto al bus. Il DMA è impostato direttamente dal software o il sistema operativo inizializzando opportuni registri. La CPU e il DMA si contendo l'uso del bus (e questa è l'unica cosa di cui si deve fare attenzione durante l'uso di questo metodo).

Trap (o eccezioni/interrupt sincroni)

Sono chiamate a procedure automatiche che possono essere causate dal verificarsi di qualche condizione eccezionale, oppure da istruzioni apposite per richiedere servizi di base del sistema operativo. Queste trap sono generate dal programma.

Example

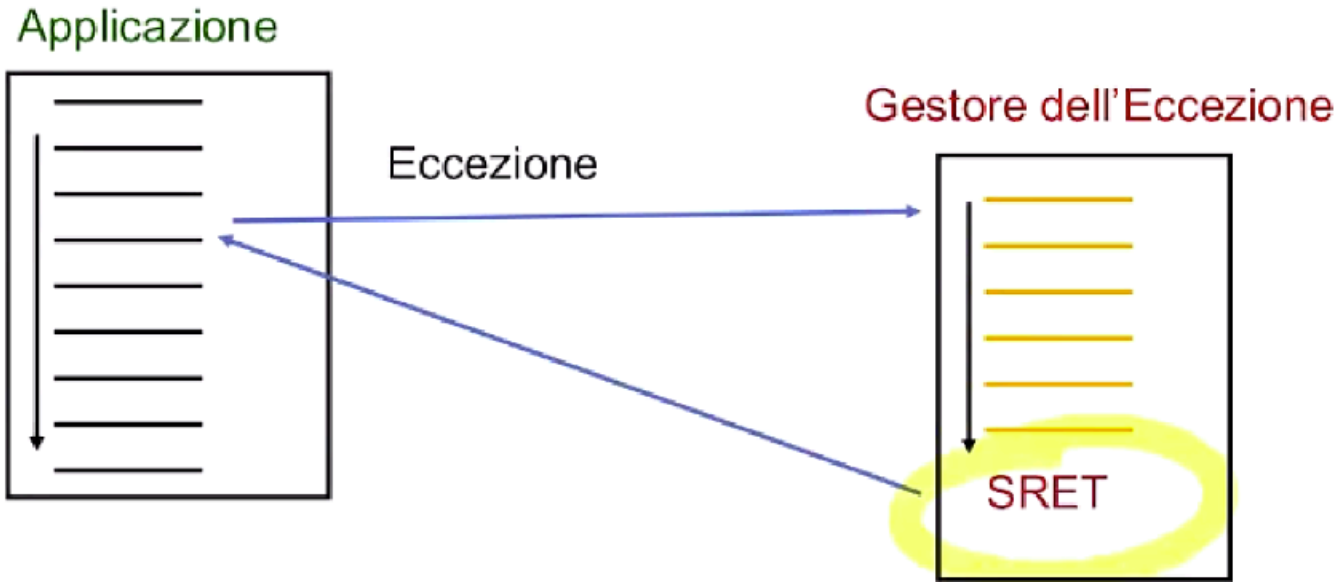
> Alcuni esempi che POSSONO generare una trap (o eccezioni/interrupt sincroni) :
>
> - Tentativo di accesso ad un area di memoria non consentita
> - Divisione per 0
> - Un overflow

Note

> Se si usa il termine "interrupt sincrono" (anziché gli altri due), si intendo che l'interrupt va gestista subito, in modo sincrono al programma.

Gestione della trap

Il sistema gestisce l'eccezione eseguendo il "gestore dell'eccezione".



SRET = Supervisor return. È un'istruzione speciale a conclusione del gestore.

Tipi di eccezioni

Troviamo due tipi di eccezioni :

- utente (user mode), le applicazioni vengono eseguite in modalità user, senza poter fare accesso alle risorse privilegiate hardware del calcolatore
- supervisor (kernel mode), durante l'eccezione, il calcolatore esegue codice del sistema operativo

Andiamo in modalità utente quando, ad esempio, chiudiamo un programma. Invece, in modalità supervisor quando c'é un errore (es. overflow), una "Environment Call" (o "trap" o "system call") o ancora "Environment break" (debug).

Come si comporta il calcolatore

- Il programma in esecuzione deve essere sospeso e poi riattivato.

- Il calcolatore salva il punto del codice (PC) in cui si è verificate l'eccezione (1).

- Il controllo passa quindi al "gestore dell'eccezione" (Exception Handler)

- il gestore dell'eccezione deve rimediare alla situazione

- A seconda del tipo di eccezione, il gestore esegue azioni diverse

- Il calcolatore identifica e salva la causa dell'eccezione (2)

- **RISC-V : registri speciali SEPC e SCAUSE**

  - (1) SEPC, l'indirizzo del codice in cui si verifica l'eccezione

  - (2) SCAUSE, la cause dell'eccezione

⚠ **Warning**

> NOTE :
>
> - Il gestore dell'eccezione deve evitare di modificare lo stato dell'applicazione (i registri...)
> - Il gestore dell'eccezione deve essere eseguito in modalità protetta (kernel mode)

Implementazione gestione delle eccezioni

In RISC-V troviamo 2 modi di gestione :

- Salto diretto all'indirizzo della routine di gestione

  - Viene generata un'eccezione

  - Esiste un registro che contiene l'indirizzo del gestore dell'eccezioni

  - Nella routine di gestione c'é un codice che va a trovare la causa dell'eccezione (che viene salvata in SCAUSE)

  - Il sistema operativo decide dove in memoria deve essere il gestore delle eccezioni.

  - L'indirizzo che continente l'indirizzo di base si chiama "SVTEC"

- Vettore di interruzione

  - SVTEC contiene l'indirizzo di base

  - In memoria abbiamo un vettore delle interruzione, cioè in memoria c'é un array che contiene gli indirizzi di tutte le routine di gestione di tutte le eccezioni.

  - Ogni causa ha un codice, si moltiplica per 4 (perché dipende dalla dimensione della causa e degli indirizzi) e lo si somma all'indirizzo di base.

  - Abbiamo l'indirizzo della routine di gestione della causa del nostro errore che punta alla memoria dove ci sono effettivamente le istruzioni.