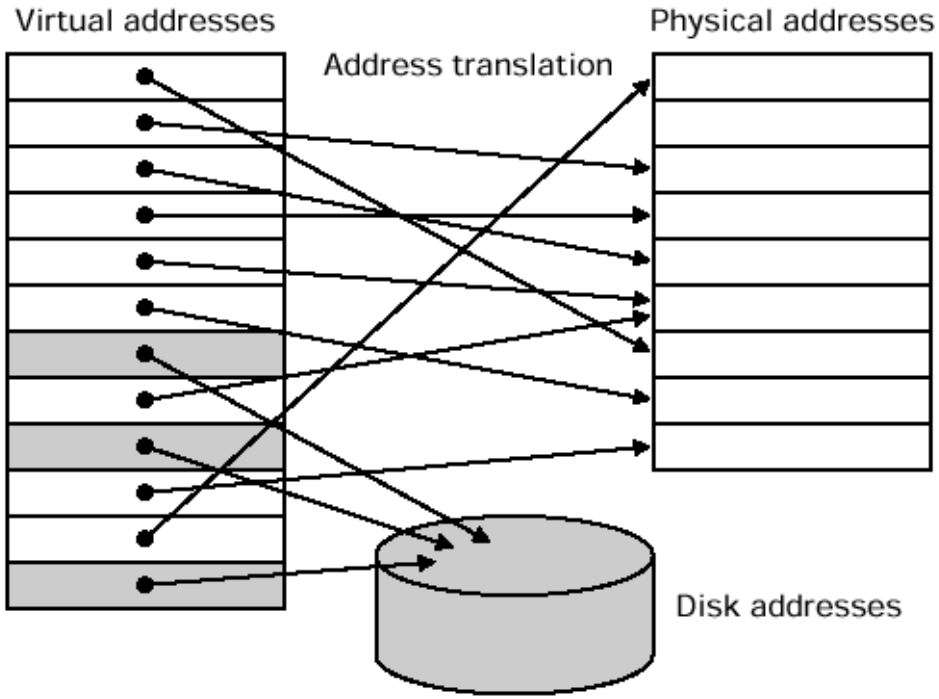
Gerarchie di memoria (memoria virtuale)

Salvatore Orlando

Memoria Virtuale

- **Uso della memoria principale come una cache della memoria secondaria (disco)**



- **I programmi sono *compilati* rispetto ad uno spazio di indirizzamento virtuale, diverso da quello fisico**
- **Il processore, in cooperazione con il SO, effettua la traduzione**
  - **indirizzo virtuale → indirizzo fisico**

Vantaggi della memoria virtuale

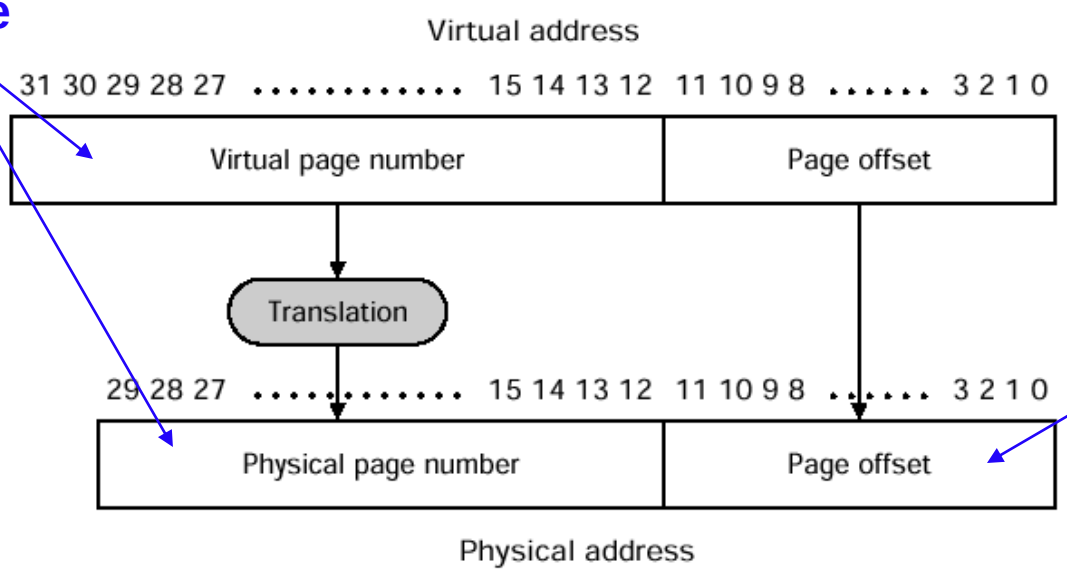
- **Illusione di avere più memoria fisica di quella disponibile**
  - **solo le parti attive dei programmi e dei dati, indirizzati rispetto ad uno *spazio virtuale*, sono presenti in memoria fisica**
  - **è possibile che più programmi, con codici e dati di dimensioni maggiori della memoria fisica, possano essere in esecuzione**

- **Traduzione dinamica degli indirizzi**
  - **i programmi, compilati rispetto a uno spazio virtuale, sono caricati in memoria fisica *on demand***
  - **tutti i riferimenti alla memoria sono *virtuali* (fetch istruzioni, load/store), e sono tradotti dinamicamente nei corrispondenti indirizzi fisici**

- **Il meccanismo di traduzione garantisce la protezione**
  - **c'è la garanzia che gli spazi di indirizzamento virtuali di programmi diversi siano effettivamente mappati su indirizzi fisici distinti**

Memoria virtuale paginata

Numero di pagine virtuali possono essere maggiori di quelle fisiche

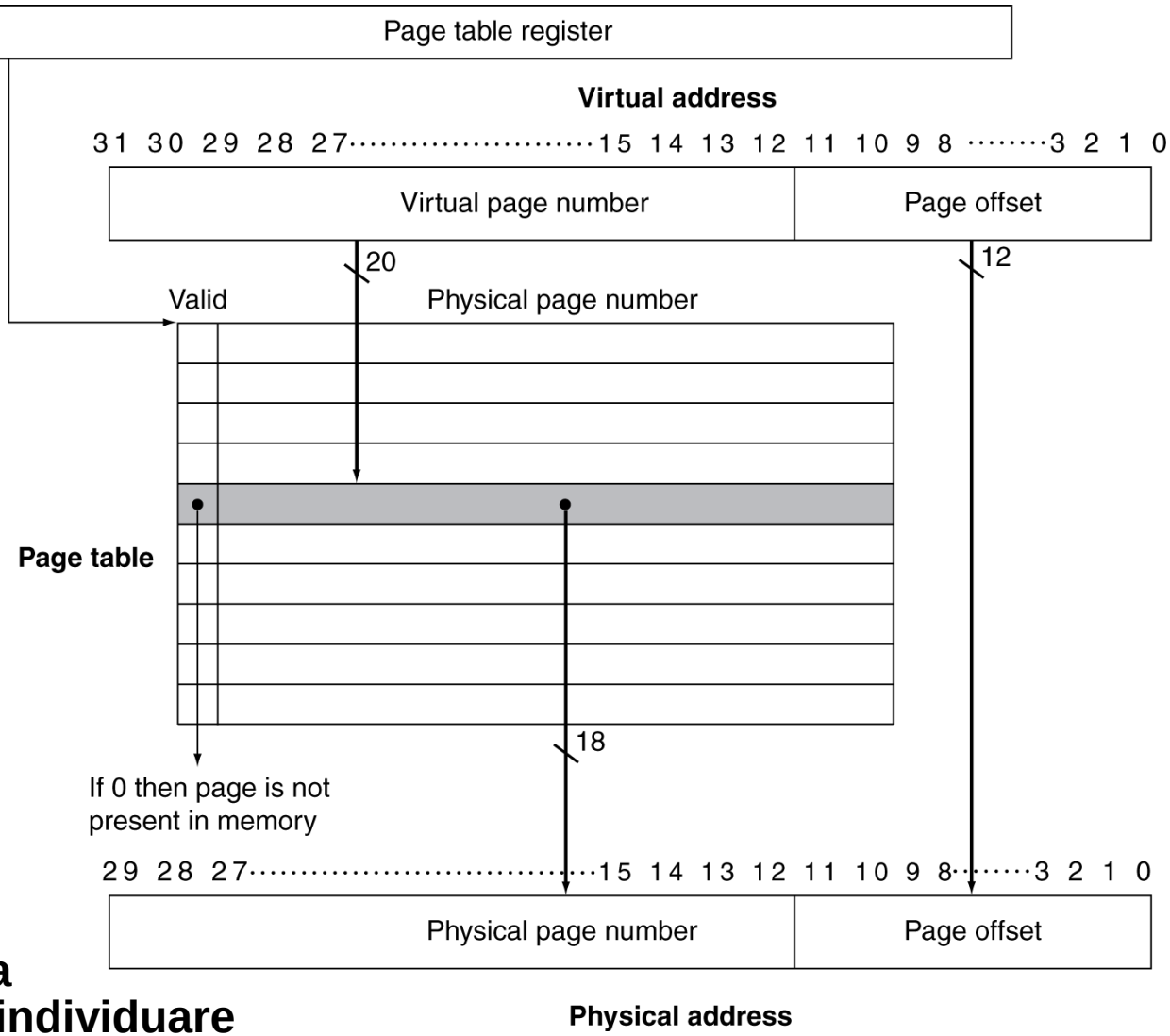


Dimensione dell'*Offset* dipende dal *page size*:
$\log_2$(Page size)

- **Oltre la paginazione, storicamente la memoria virtuale è stata anche implementata tramite *segmentazione***
  - **blocco di dimensione variabile**
  - **Registri Relocation e Limit**
  - **enfasi su protezione e condivisione**
- **Svantaggio: esplicita suddivisione dell'indirizzo virtuale in *segment number* + *segment offset***

Page table, traduzione indirizzi, e associatività

- **Page Table (PT) mantiene la corrispondenza tra pagine virtuale e fisica**
- **La PT di un programma in esecuzione (processo) sta in memoria:**
  - **la PT è memorizzata ad un certo indirizzo fisico, determinato dal *page table register***
- **Ogni pagina virtuale può corrispondere a qualsiasi pagina fisica (completa associatività)**
  - **Trovare la pagina fisica corrispodente è come individuare la via in una cache completamente associativa**



Pagine: sono i blocchi della memoria virtuale

- ***Page fault*** **≡** ***miss*****: la pagina non è in memoria, e deve essere letta dal disco**

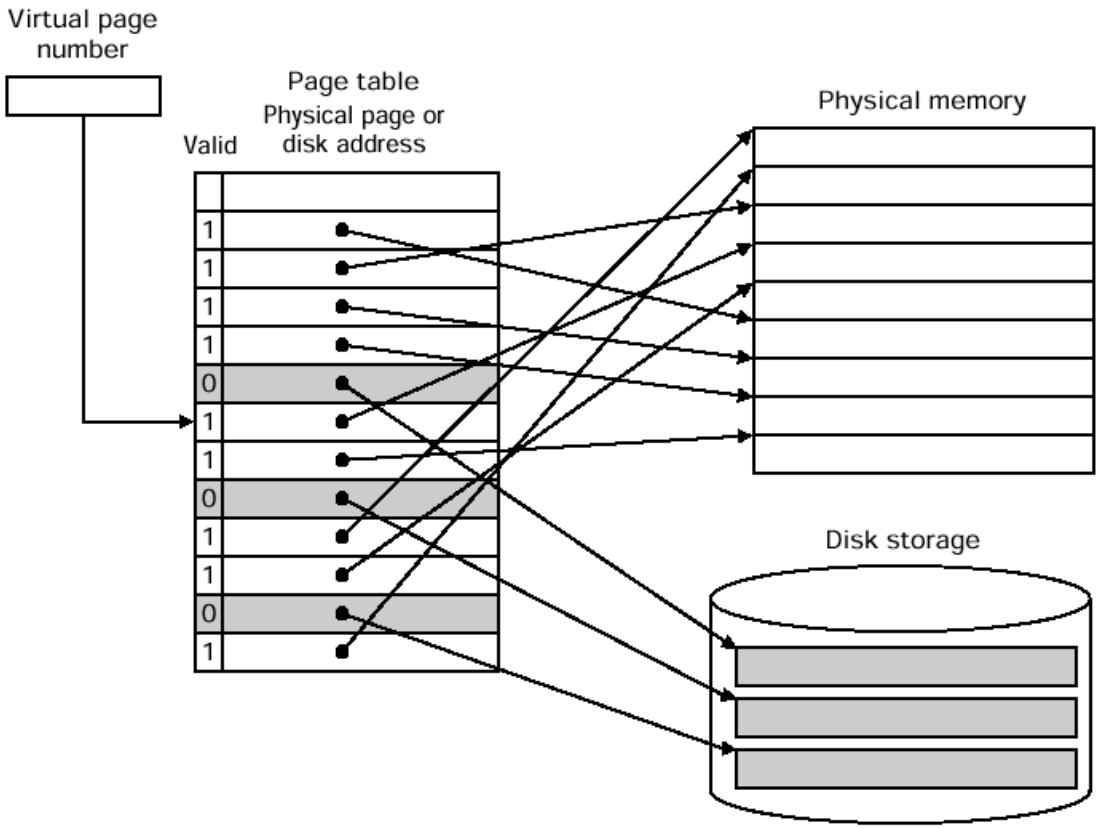
- ***Miss penalty*** **grande (msec, milioni di cicli di clock)**
  - **è utile che i blocchi (pagine) siano grandi (es.: 4KB)**
  - **le letture da disco hanno un costo iniziale alto, dovuto a movimenti meccanici dei dispositivi**

- **Ridurre i *page fault* è quindi molto importante**
  - **mapping dei blocchi (pagine) completamente associativo**
  - **politica LRU, per evitare di eliminare dalla memoria pagine da riusare subito dopo, a causa della località degli accessi**

- ***Miss (page fault)*** **gestiti a software tramite l'intervento del SO**
  - **algoritmi di mapping e rimpiazzamento più sofisticati**

- **Solo politica *write-back* (perché scrivere sul disco è costoso)**

Memoria paginata



- **Al loading del processo, viene creato su disco l'immagine delle varie pagine del programma e dei dati**
- **Page table (o *struttura corrispondente*) usata anche per registrare gli indirizzi su disco delle pagine**
  - **indirizzi su disco utilizzati dal SO per gestire il page fault, e il rimpiazzo delle pagine**

Approfondimenti

- **Spesso, oltre al valid bit, sono aggiunti altri bit associati alla pagine**
  - **dirty bit**
    - **serve a sapere se una pagina è stata modificata.**
    - **Grazie a questo bit è possibile sapere se la pagina deve essere ricopiata sul livello di memoria inferiore (disco).**
    - **Il bit è necessario in quanto usiamo una politica write-back**
  - **reference bit**
    - **serve a sapere se, in un certo lasso di tempo, una certa pagina è stata riferita.**
    - **bit azzerato periodicamente, settato ogni volta che una pagina è riferita.**
    - **Il reference bit è usato per implementare una politica di rimpiazzo delle pagine di tipo LRU (Least Recently Used)**

Approfondimenti

- **La *page table*, per indirizzi virtuali grandi, diventa enorme**
  - **supponiamo ad esempio di avere un ind. virtuale di 32 b, e pagine di 4 KB. Allora il *numero di pagina virtuale* è di 20 b. La *page table* ha quindi 2^{20} entry. Se ogni entry fosse di 4 B, la dimensione totale sarebbe:**

    $$2^{22}\ \text{B} = 4\ \text{MB}$$

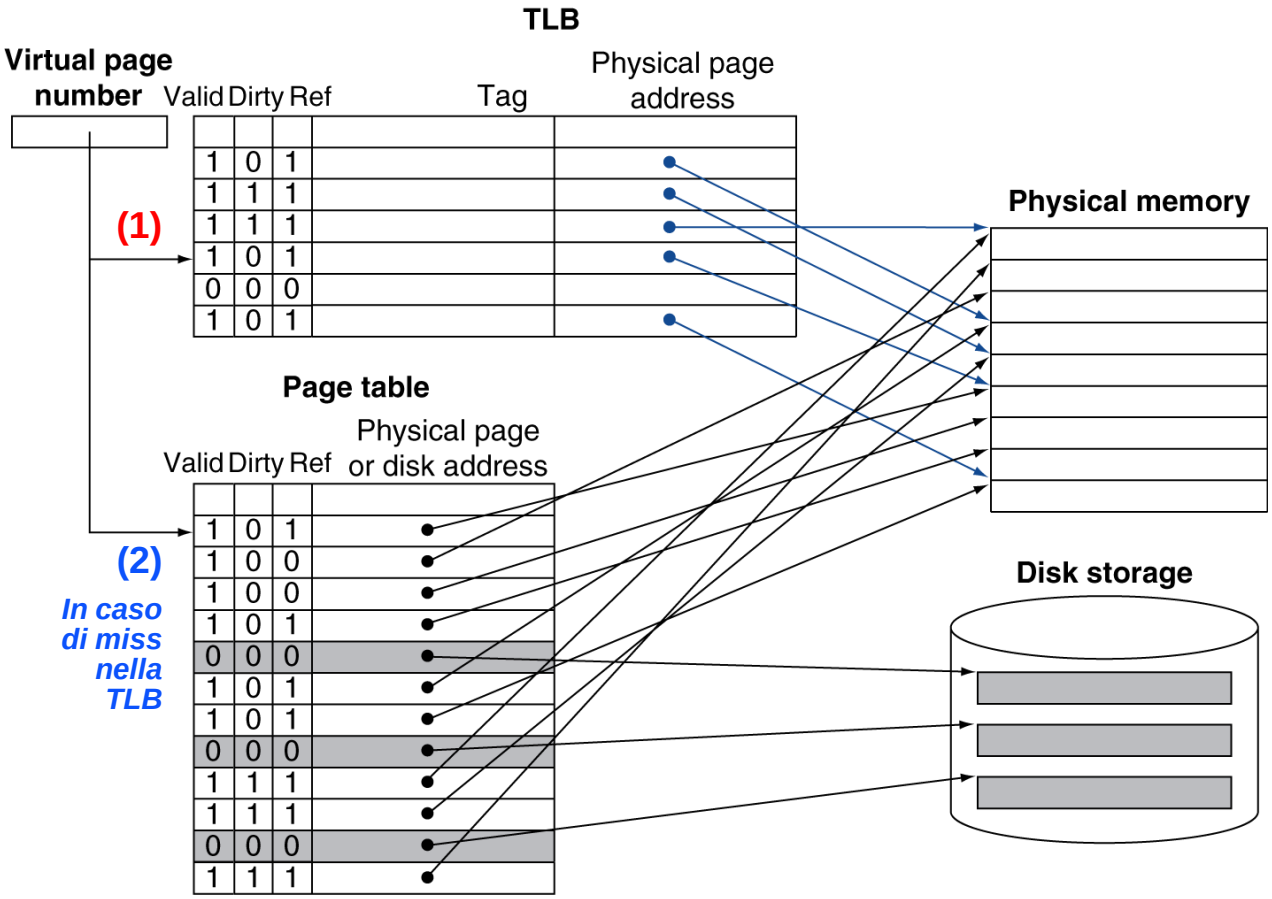
  - **se ci fossero molti programmi in esecuzione, una gran quantità di memoria sarebbe necessaria per memorizzare *soltanto* le varie *page table***
- **Esistono diversi metodi per ridurre la memoria usata per memorizzare la PT**
  - **i metodi fanno affidamento sull'osservazione che i programmi (piccoli) usano solo una piccola parte della page table a loro assegnata (es. meno di 2^{20} pagine), e che c'è anche una certa località nell'uso delle page table**
  - **es.: page table paginate, page table a due livelli, ecc.…**

TLB: traduzione veloce degli indirizzi

- **TLB = Translation Lookaside Table**
  - **Dispositivo hw veloce che traduce degli indirizzi virtuali**
- **La TLB è in pratica una cache della page table, e quindi conterrà solo alcune entry della page table (le ultime riferite)**
  - **a causa della località, i riferimenti ripetuti alla stessa pagina sono molto frequenti**
  - **il primo riferimento alla pagina (page hit) avrà bisogno di leggere la page table. Le informazioni per la traduzione verranno memorizzate nella TLB**
  - **i riferimenti successivi alla stessa pagina potranno essere risolti velocemente in hardware, usando solo la TLB**

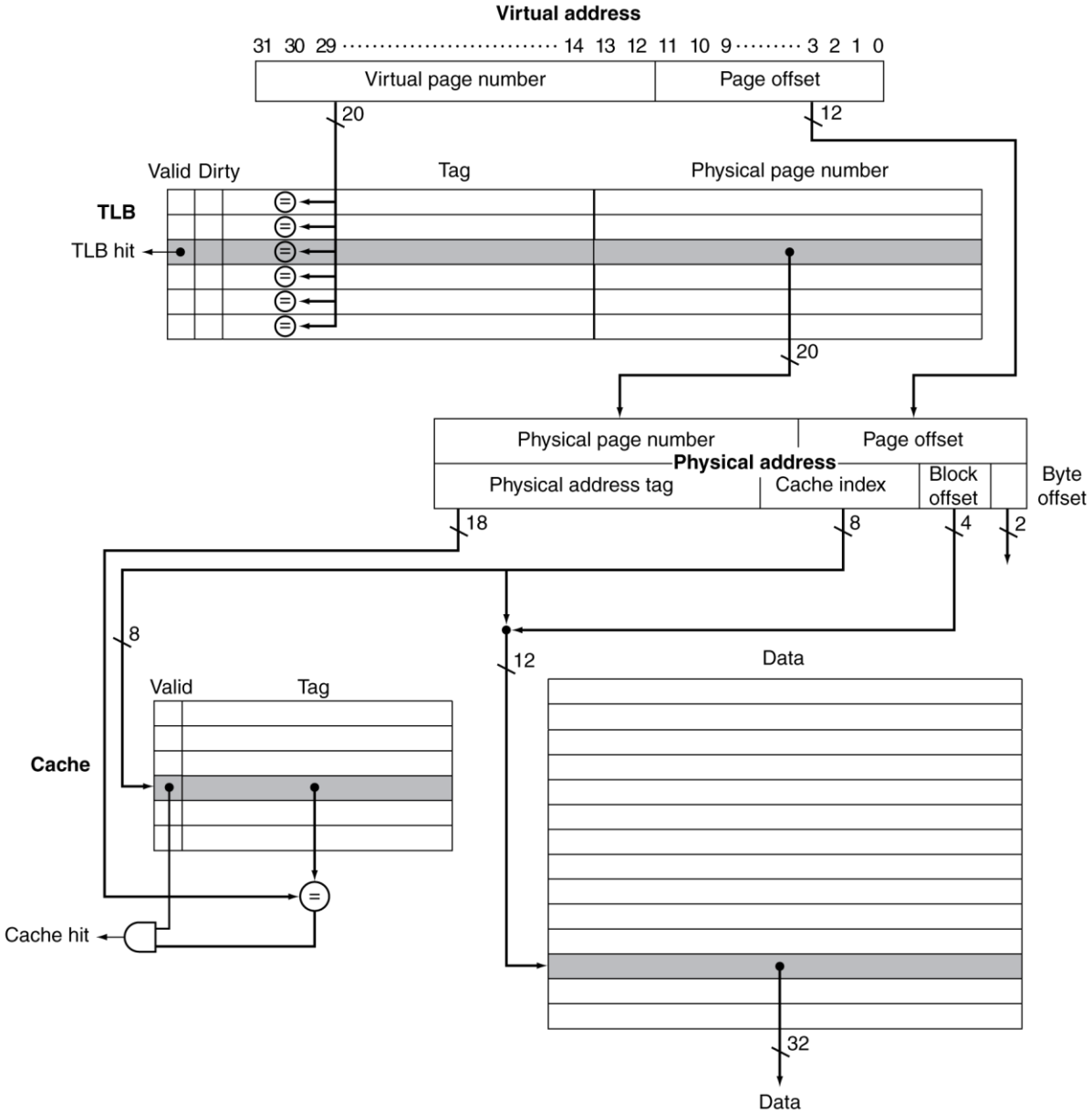
TLB

- **Esempio di TLB *completamente associativa***
  - **in questo caso il TAG della TLB è proprio il numero di pagina virtuale da tradurre**
  - **per ritrovare il numero di pagina fisica, bisogna confrontare il numero di pagina virtuale con tutti i TAG della TLB**



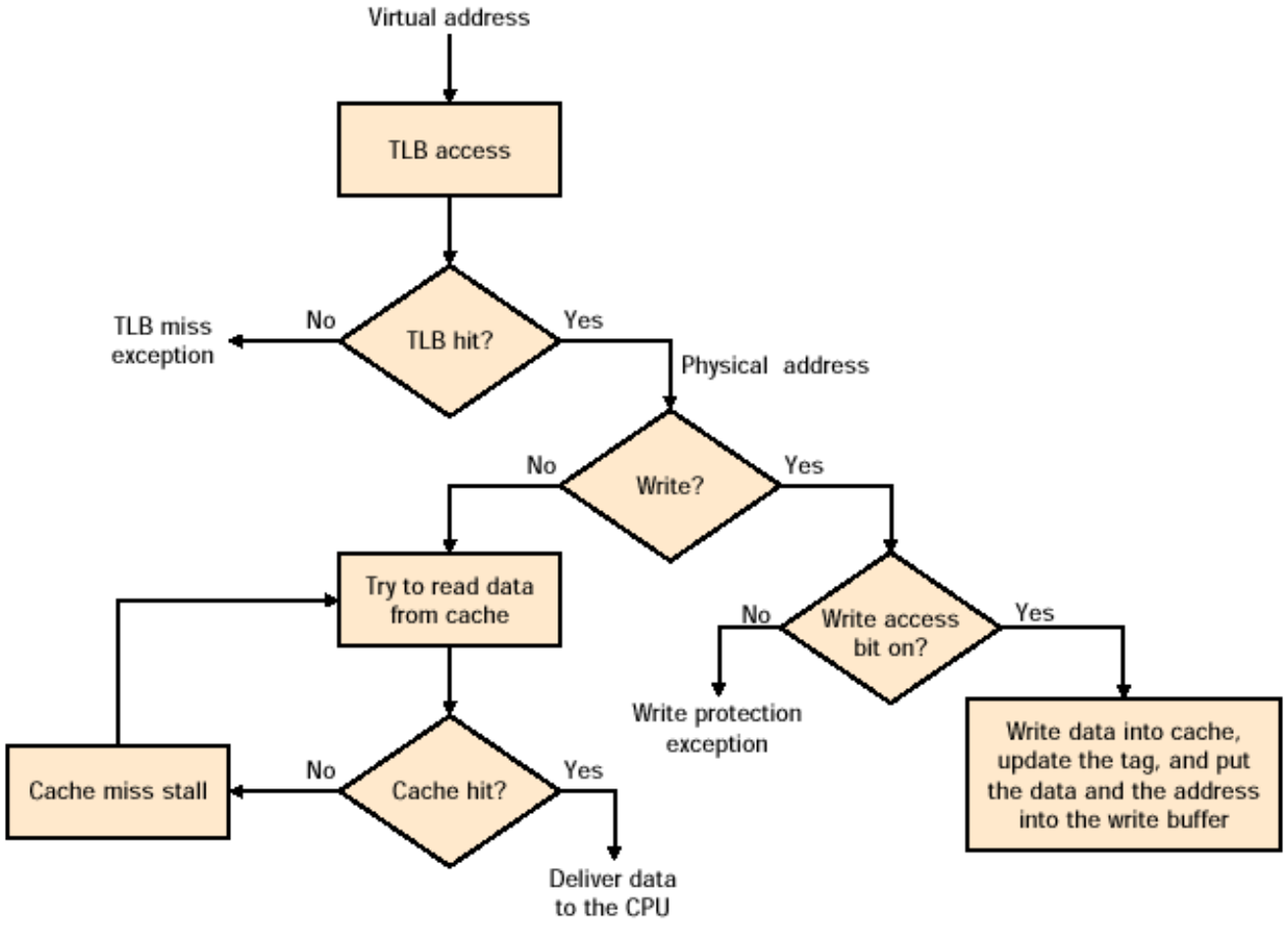
- **La TLB contiene solo entry che risultano anche Valid nella page table**
  - **Se una pagina viene eliminata dalla memoria fisica, l'entry della TLB viene invalidata**
- **La TLB, come la cache, può essere implementata con vari livelli di associatività**

TLB e cache (FastMATH MIPS)



- **TLB completamente associativa**
- **Pagine da 4 KB**
- **Cache diretta, acceduta con l'indirizzo fisico**
- **Dimensione del blocco della cache: 2^6 = 64 B**
- **I dati della cache (blocchi) sono memorizzati separatamente da Valid/Tag**

TLB e cache (FastMATH MIPS)



- **Gestione letture/scritture**
  - **TLB hit e miss**
  - **cache hit e miss**
- **Nota i cicli di stallo in caso di cache miss**
- **Nota la cache con politica write-through**
- **Nota le eccezioni**
  - **TLB miss**
  - **write protection (usato come trucco per aggiornare i bit di dirty sulla page table)**

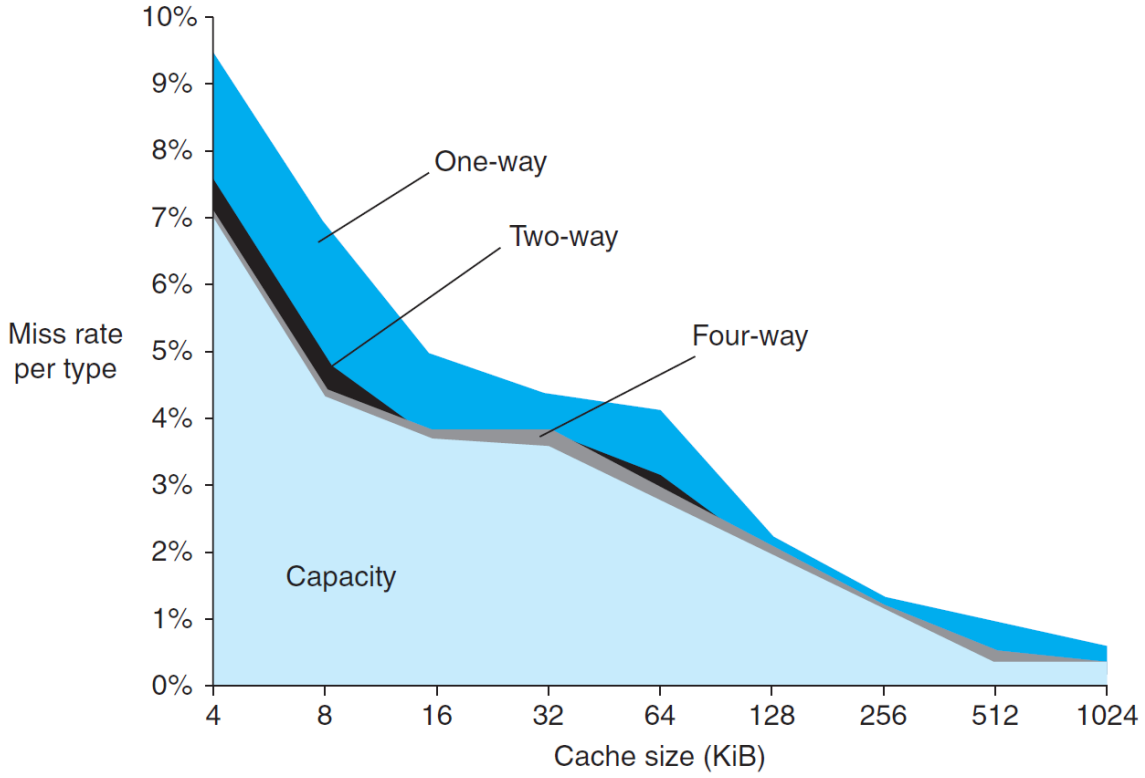
Modello di classificazione dei *miss*

- **Nelle varie gerarchie di memoria, i miss si possono verificare per cause diverse**
  - **modello delle tre C per classificare i miss**
- **Ci riferiremo al livello cache, anche se il modello si applica anche agli altri livelli della gerarchia**

- **Tipi di miss**
  - **Miss Certi (Compulsory)**
    - **miss di partenza a freddo, che si verifica quando il blocco deve essere portato nella cache per la prima volta**
  - **Miss per Capacità**
    - **la cache non è in grado di contenere tutti i blocchi necessari all'esecuzione del programma. Si verifica quando un blocco vien espulso e subito dopo riammesso nella cache.**
  - **Miss per Conflitti/Collisioni**
    - **due blocchi sono in conflitto per una certa posizione**
    - **può verificarsi anche se la cache NON è piena**
    - **questo tipo di miss non si verifica se abbiamo una cache completamente associativa**

Modello di classificazione dei *miss*

- **SPEC CPU2000 integer/floating-point benchmarks**
- **Compulsory non sono visibili (solo lo 0.006%)**
- **Il miss rate di *capacità*: dipende dal cache size, ed è presente anche in cache completamente associative**
- **Miss classificati come *conflitti* se non si verificano nella cache completamante associativa**



Sistema Operativo (SO) e gestione della memoria

- **Per quanto riguarda la memoria virtuale, il SO viene invocato per gestire due tipi di eccezioni**
  - **TLB miss (anche se la TLB miss può essere gestita in hardware, in quanto il penalty è minore: accesso alla Page Table in memoria RAM)**
  - **Page fault**

Ma cos'è un'eccezione?

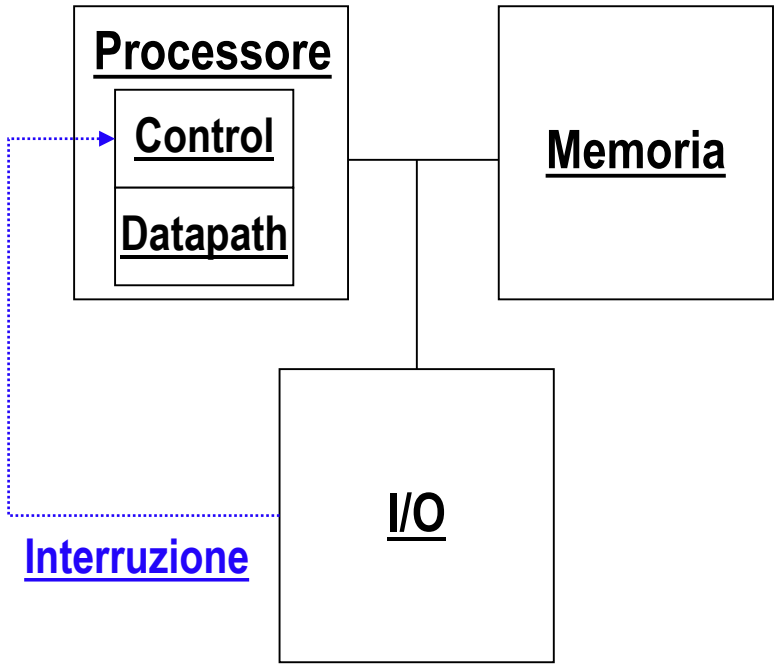
Eccezioni / Interruzioni

Eccezioni e interruzioni

- **Interruzioni ed eccezioni sono eventi inattesi da gestire**

- **Eccezione**
  - **evento *sincrono*, generato all'interno del processore, e provocato da problemi nell'esecuzione di un'istruzione**
    - **es.: *overflow, istruzione non valida, page fault, TLB-miss***

- **Interruzione**
  - **evento *asincrono*, che giunge dall'esterno del processore**
  - **segnale che giunge da una periferica (unità di I/O) utilizzato per comunicare alla CPU il verificarsi di certi eventi**
    - **es.: la *terminazione di un'operazione di I/O* la cui esecuzione era stata richiesta in precedenza dalla CPU**



Gestione di eccezioni e interruzioni

- **Il controllo di ogni processore deve essere predisposto per gestire il verificarsi di eventi inattesi**
- **Tutti i processori, quando si verifica un evento di eccezione/interruzione, la gestiscono secondo lo schema seguente:**
  - **interruzione dell'esecuzione del programma corrente**
  - **salvataggio di parte dello stato di esecuzione corrente (almeno PC)**
  - **salto ad una routine del codice che costituisce il Sistema Operativo (SO)**
    - **il SO è stato caricato in memoria al momento del boot del sistema**
    - **il salvataggio dello stato del programma interrotto serve al SO per poter riprenderne *eventualmente* l'esecuzione, successivamente e se necessario**

Gestione di eccezioni e interruzioni

- **Problema: l'handler (il gestore) del SO deve essere in grado di capire quale evento si è verificato.**

- **Soluzione 1:**
  - **interruzioni vettorizzate: esistono handler diversi per eccezioni/interruzioni differenti. Il controllo della CPU deve scegliere l'handler corretto, saltando all'indirizzo corretto.**
    A questo scopo, viene predisposto un vettore di indirizzi, uno per ogni tipo di eccezioni/interruzioni, da indirizzare tramite il codice numerico dell'eccezione/interruzione

Gestione di eccezioni e interruzioni

- **Soluzione 2 (MIPS):**
  - **il controllo della CPU, prima di saltare all'handler predisposto dal SO (ad un indirizzo fisso). L'handler accederà ad un registro interno per determinare la causa dell'eccezione/interruzione**
  - **L'handler unico esegue una sequenza di instruzioni con scopo di controllare e capire la causa dell'eccezione/interruzione e saltare ad eseguire il codice di gestione dell'eccezione/interruzione**

- **Nel MIPS viene questa soluzione, usando un registro, denominato *Cause*, per memorizzare il motivo dell'eccezione/interruzione**
- **Il PC corrente viene invece salvato nel registro *EPC (Exception PC)***

Gestione eccezioni nel MIPS

- **Il *Controllo* (ma anche il *Datapath* corrispondente) deve essere progettato per**
  - **individuare l'evento inatteso**
  - **interrompere l'istruzione corrente**
  - **salvare il PC corrente (nel registro interno *EPC* = Exception PC)**
  - **salvare la causa dell'interruzione nel registro *Cause***
    - **consideriamo solo le eccezioni di *overflow* e *istruzione non valida***

      0 = *istruzione non valida* **1=*overflow***
  - **saltare ad una routine del SO (exception/interrupt handler) ad un indirizzo fisso: 0xC0000000**

Gestione eccezioni nel MIPS

- **Il MIPS non salva nessun altro registro oltre PC**
  - **è compito dell'handler salvare altre porzioni dello stato corrente del programma (es. tutti i registri generali), se necessario**
    - **approccio RISC => *semplice e minimale***
  - **esistono CPU dove il "salvataggio esteso" dello stato viene sempre effettuato prima di saltare all'interrupt handler**
    - **salvataggio garantito dal microcodice => *complesso***
    - **approccio CISC**

Rilevamento eccezione

- ***Overflow***
  - **rilevato sulla base del segnale che arriva al controllo dall'ALU**
- ***Istruzione non valida***
  - **rilevato sulla base del campo op dell'istruzione**

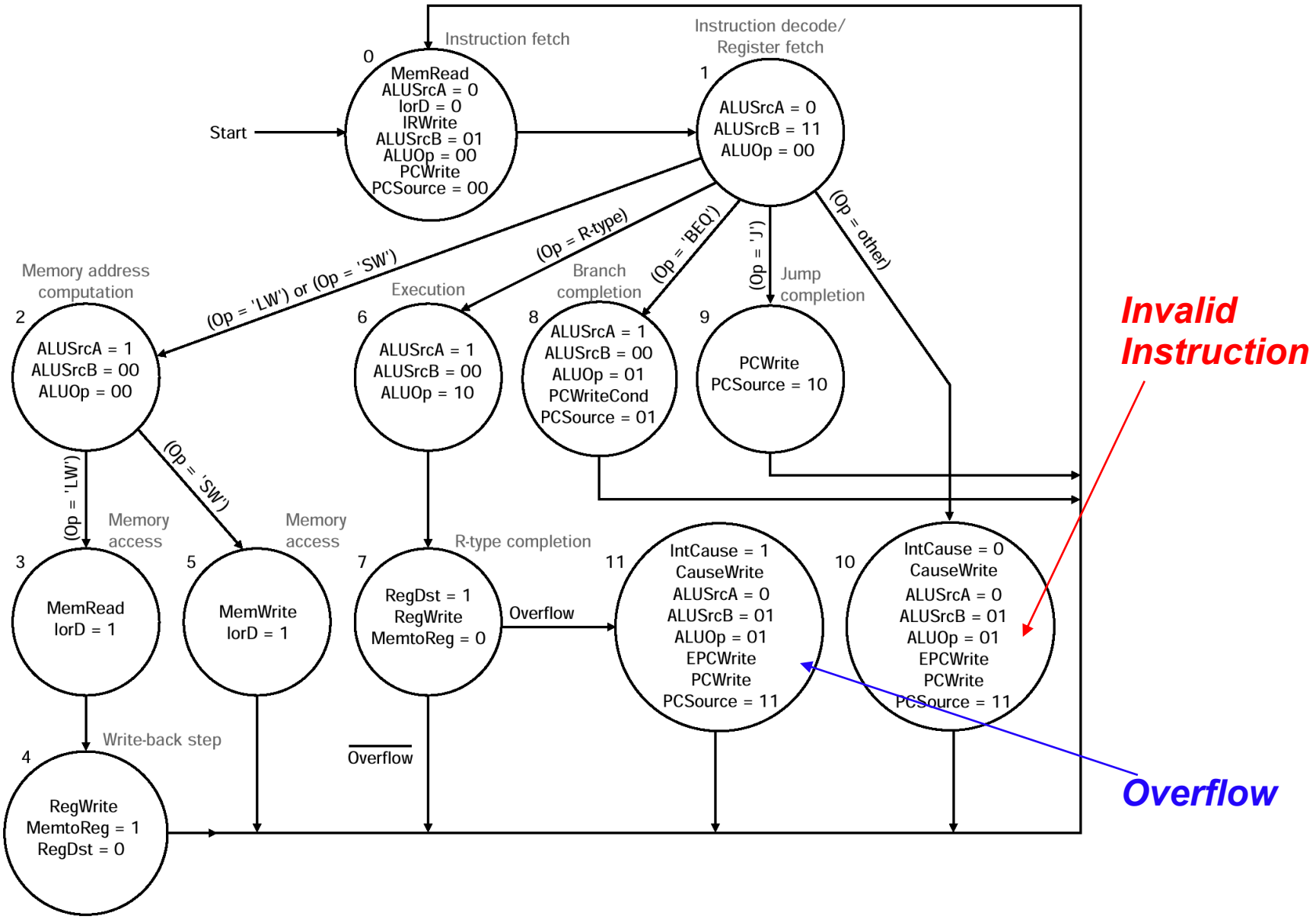
Rilevamento eccezione nel progetto multiciclo

- **Dobbiamo aggiungere 2 nuovi stati al nostro automa**
  - **a partire dallo stato *Instruction Decode*, dobbiamo poter transire nel nuovo stato *Invalid Instruction***
    - **ma solo se viene riconosciuto un *Op-code* non valido**
  - **a partire dallo stato *R-type Completion*, dobbiamo poter transire nello stato *Overflow***
    - **solo se giunge un segnale di overflow dal Datapath (ALU) al 3º ciclo**
    - **questa transizione di stato si potrebbe anticipare allo stato Execution ?**
      - **Problema**: *next state* da calcolare in base ad un segnale calcolato dal Datapath durante lo stesso ciclo di clock => avremmo bisogno di un ciclo di clock più lungo

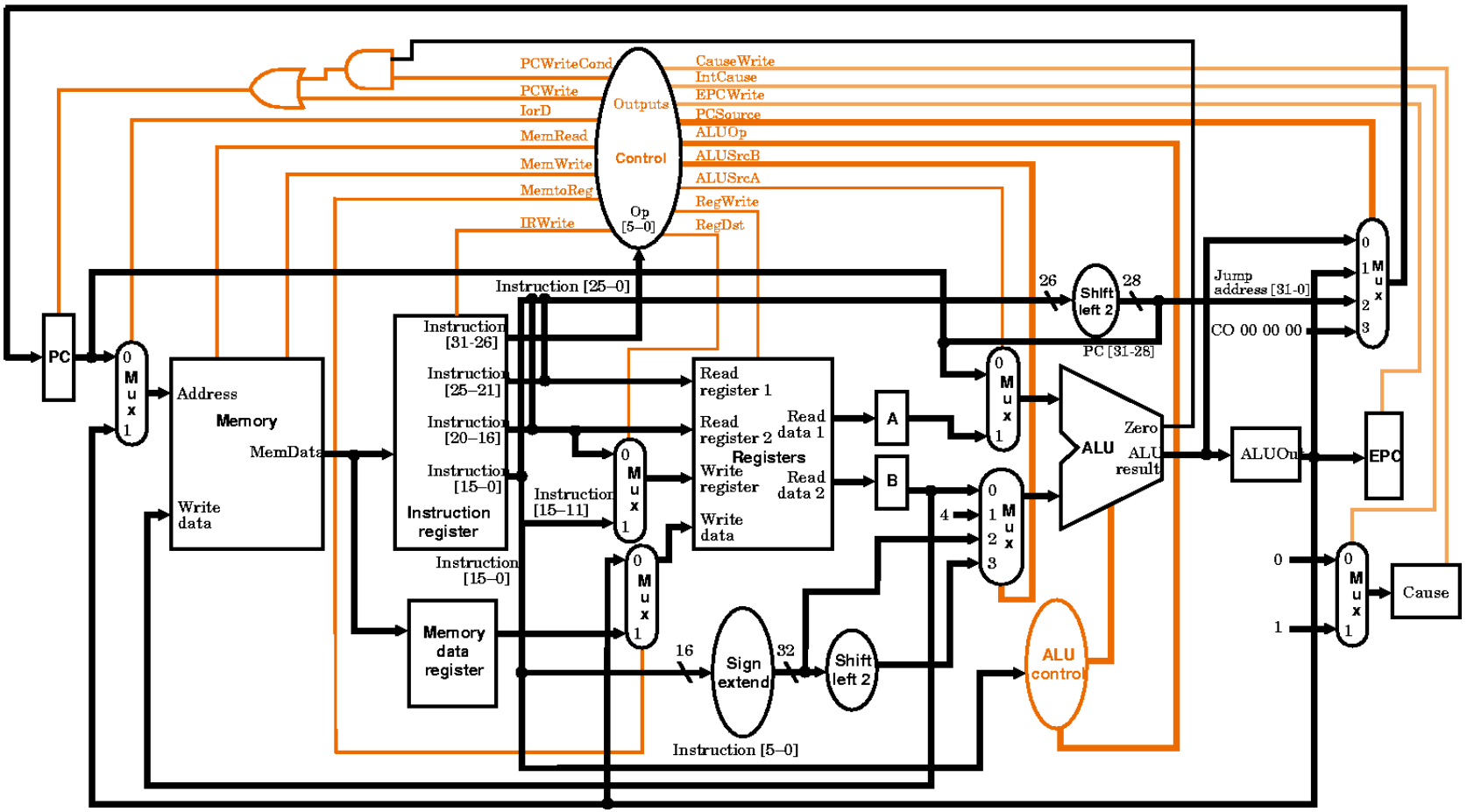
Rilevamento eccezione

- **I nuovi stati che gestiscono le eccezioni dovranno occuparsi**
  - **di salvare in EPC il PC corrente (ovvero PC - 4)**
  - **di salvare 0/1 in Cause**
  - **di memorizzare 0xC0000000 in PC**
- **Il prossimo stato sarà il *Fetch* *(lettura della prima istruzione dell'handler)***

Nuovo controllo MULTICICLO



Nuovo Datapath MULTICICLO



- **Nuovo ingresso per PC (ind. Interrupt handler): 0xC0000000**
- **EPC viene caricato con il risultato dell'ALU, che calcola PC- 4**
- **Cause viene caricato con 0, oppure con 1.**

Pipeline: interruzioni/eccezioni

- Il verificarsi di un'eccezione è legata all'esecuzione di una certa istruzione
  - le istruzioni precedenti devono essere completate
  - l'istruzione che ha provocato l'eccezione e quelle successivamente entrate nella pipeline devono essere eliminate dalla pipeline (trasformate in nop)
  - deve essere fetched la prima routine dell'*exception handler*
- Le interruzioni sono asincrone, ovvero non sono legate ad una particolare istruzione
  - siamo più liberi nello scegliere quale istruzione interrompere per trattare l'interruzione

Sistema Operativo, gestione della memoria e protezione

Sistema Operativo (SO) e gestione della memoria

- **Per quanto riguarda la memoria virtuale, il SO viene invocato per gestire due tipi di eccezioni**
  - **TLB miss**
  - **Page fault**

- **In risposta ad un'eccezione/interruzione**

  - **il processore salta alla *routine di gestione del SO***

  - **effettua anche un passaggio di modalità di esecuzione**
    user mode* → *kernel (supervisor) mode

SO e gestione della memoria

- **Alcune operazioni possono essere solo effettuate dal SO che esegue in *kernel mode* ⇒ *PROTEZIONE ESECUZIONE PROGRAMMI.***

- **Un programma in *user mode:***
  - **Non può modificare il PT register**
  - **Non può modificare le entry della TLB**
  - **Non può settare direttamente il bit che fissa l'execution mode**
  - **Esistono istruzioni speciali, eseguibili SOLO in *kernel mode*, per effettuare le operazioni di cui sopra**

SO e gestione della memoria

- **Un programma utente in esecuzione *user mode* può passare volontariamente in *kernel mode* SOLO invocando una `syscall`**
  - **L'istruzione di `syscall` serve ad invocare il SO (ad effettuare una chiamata di sistema) per funzioni come *gestire il file system*, *gestire i processi in esecuzione*, *gestire le operazioni di networking*, ecc.**
  - **Le routine corrispondenti alle varie `syscall` sono prefissate, e fanno parte del SO (l'utente non può crearsi da solo una sua `syscall` e invocarla)**

SO e gestione della memoria

- solo TLB miss
  - la pagina è presente in memoria
  - l'eccezione può essere risolta tramite la Page Table allocata in RAM (penalty simile al cache miss)
  - l'istruzione che ha provocato l'eccezione deve essere rieseguita

La semplice TLB miss può essere gestita in hardware, in quanto il penalty è piccolo

SO e gestione della memoria

- **TLB miss che si trasforma in page fault**
  - **la pagina non è presente in memoria, cioè l'ingresso corrispondente della PT è INVALID**
  - **la pagina deve essere portata in memoria dal disco**
    - **operazione di I/O dell'ordine di ms**
    - **è impensabile che la CPU rimanga in stallo, attendendo che il page fault venga risolto**
  - ***context switch***
    - **salvataggio dello stato (contesto) del programma (processo) in esecuzione**
      - fanno ad esempio parte dello stato i registri generali, e quelli speciali come il registro della page table
      - processo che ha provocato il fault diventa bloccato
    - **ripristino dello stato di un altro processo pronto per essere eseguito**
    - **restart del nuovo processo**
  - **completamento page fault**
    - **processo bloccato diventa pronto, ed eventualmente riprende l'esecuzione**

SO e gestione della memoria

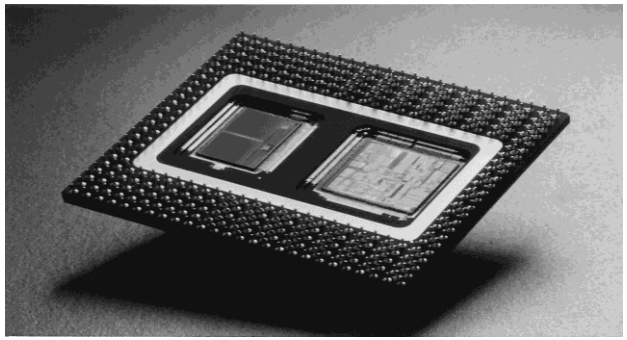
- **Page fault e rimpiazzamento di una pagina**
  - **se la memoria fisica è (completamente o quasi) piena, bisogna risolvere il fault rimpiazzando una pagina**
    - **es. usando una politica LRU**
  - **la pagina deve anche essere scritta in memoria secondaria se *dirty* (write-back)**
  - **poiché in questo caso modifichiamo una entry della page table, se questa entry è cached nella TLB, bisogna anche ripulire la TLB**

- **Protezione**
  - **il meccanismo della memoria virtuale impedisce a ciascun processo di accedere porzioni di memoria fisica allocata a processi diversi**
  - **TLB e PT non possono essere modificate da un processo in esecuzione in modalità utente**
    - **possono essere modificate solo se il processore è in *stato kernel***
    - **ovvero solo dal SO**

Casi di studio

- **Gerachie di memoria: Intel Pentium Pro e IBM PowerPC 604**

Characteristic	Intel Pentium Pro	PowerPC 604
Virtual address	32 bits	52 bits
Physical address	32 bits	32 bits
Page size	4 KB, 4 MB	4 KB, selectable, and 256 MB
TLB organization	A TLB for instructions and a TLB for data Both four-way set associative Pseudo-LRU replacement Instruction TLB: 32 entries Data TLB: 64 entries TLB misses handled in hardware	A TLB for instructions and a TLB for data Both two-way set associative LRU replacement Instruction TLB: 128 entries Data TLB: 128 entries TLB misses handled in hardware



Characteristic	Intel Pentium Pro	PowerPC 604
Cache organization	Split instruction and data caches	Split instruction and data caches
Cache size	8 KB each for instructions/data	16 KB each for instructions/data
Cache associativity	Four-way set associative	Four-way set associative
Replacement	Approximated LRU replacement	LRU replacement
Block size	32 bytes	32 bytes
Write policy	Write-back	Write-back or write-through