CALCOLATORI
La gerarchia di memoria

Marco Roveri
marco.roveri@unitn.it

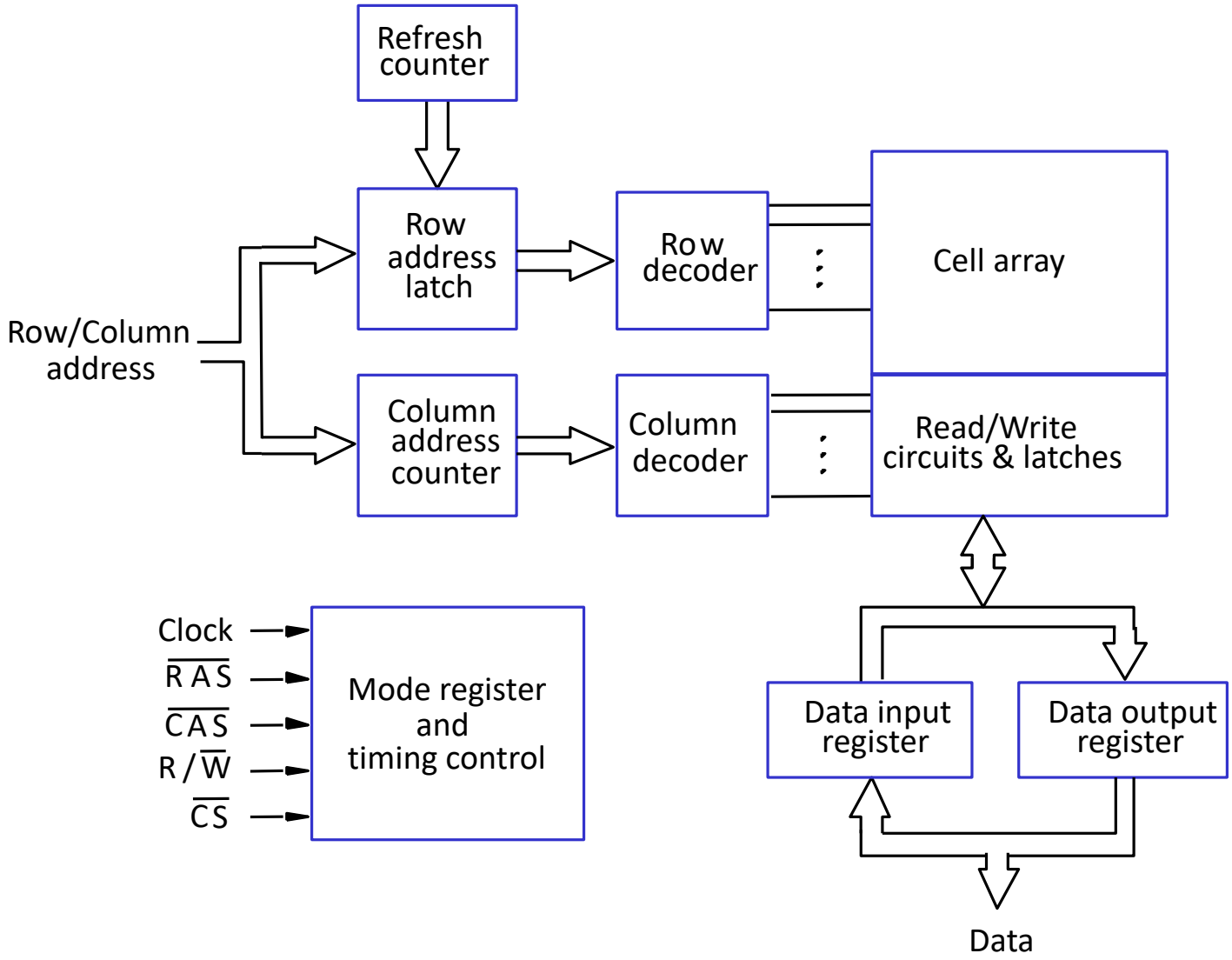
Lezione basata su materiale preparato dai Prof. Luigi Palopoli e Giovanni Iacca

UNIVERSITÀ DEGLI STUDI DI TRENTO
Dipartimento di Ingegneria
e Scienza dell'Informazione

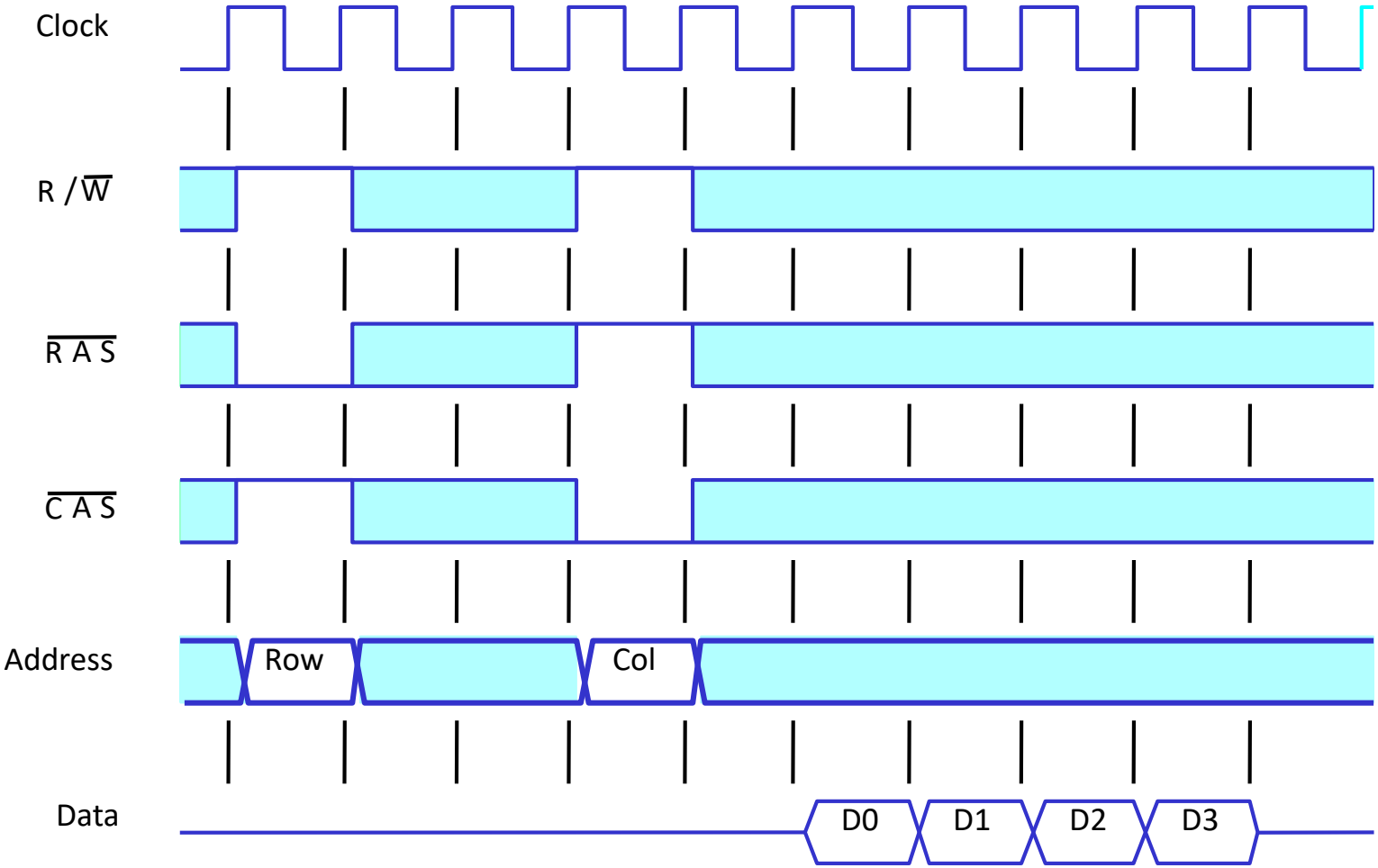
DRAM sincrone (SDRAM)

- Le DRAM viste prima sono dette “asincrone” perché non esiste una precisa temporizzazione di accesso, ma la dinamica viene governata dai segnali RAS e CAS
- Il processore deve tenere conto di questa potenziale “asincronicità”
  - in caso di rinfresco in corso può essere fastidiosa
- Aggiungendo dei buffer (latch) di memorizzazione degli ingressi e delle uscite si può ottenere un funzionamento sincrono, disaccoppiando lettura e scrittura dal rinfresco, e si può ottenere automaticamente un accesso FPM pilotato dal clock

Organizzazione base di una SDRAM



SDRAM: esempio di accesso in FPM



Velocità e prestazione

- **Latenza:** tempo di accesso ad una singola parola
  - è la misura "principe" delle prestazioni di una memoria
  - dà un'indicazione di quanto il processore dovrebbe poter aspettare un dato dalla memoria nel caso peggiore
- **Velocità o "banda":** velocità di trasferimento massima in FPM
  - molto importante per le operazioni in FPM che sono legate all'uso di memorie cache interne ai processori
  - è anche importante per le operazioni in DMA, posto che il dispositivo periferico sia veloce

Double-Data-Rate SDRAM (DDR-SDRAM)

- DRAM sincrona che consente il trasferimento dei dati in FPM sia sul fronte positivo che sul fronte negativo del clock
- Latenza uguale a una SDRAM normale
- Banda doppia
- Sono ottenute organizzando la memoria in due banchi separati
  - **uno contiene le posizioni pari: si accede sul fronte positivo**
  - **l'altro quelle dispari: si accede sul fronte negativo**
- Locazioni contigue sono in banchi separati e quindi si può fare l'accesso in modo interlacciato

Gerarchia di memoria

- Abbiamo visto i vari blocchi di memoria con diverse caratteristiche di velocità e capacità
- Ma ritorniamo a come ottenere velocità e capacità insieme
- Il problema è noto da molto tempo….

Idealmente si desidererebbe una memoria indefinitamente grande, in cui ogni particolare parola risulti immediatamente disponibile […]

Burks, Goldstine, Von Neumann, 1946

Gestione "piatta"

- Immaginiamo di essere un impiegato che lavora al comune
- Per effettuare il mio lavoro ho bisogno di avere accesso ad un archivio dove sono presenti le varie pratiche
- Ogni volta che mi serve una pratica vado a prenderla, ci opero su, e poi la rimetto a posto

Gestione "piatta"

- Per accedere alla pratica scrivo su un bigliettino lo scaffale dove la pratica può essere trovata, lo affido a un attendente, e aspetto che me la porti
- Osservazioni:
  - la mia capacità di memorizzazione è molto grande
  - la gran parte del mio tempo (direi il 90%) la spreco aspettando che l'attendente vada a prendere le pratiche
  - Sicuramente non è una gestione efficiente del mio tempo
- Posso essere più veloce?

Gestione "veloce"

- In alternativa posso tenere le pratiche sul mio tavolo e operare solo su quelle
- Osservazioni
  - Sicuramente non perdo tempo (non ho da aspettare attendenti che vadano in su e in giù)
  - Tuttavia il numero massimo di pratiche che possono gestite è molto basso
- Posso operare su più dati?

Capre e cavoli

- Per riuscire ad avere al tempo stesso velocità e ampiezza dell'archivio, posso fare due osservazioni:

1. Il numero di pratiche su cui posso concretamente lavorare in ogni giornata è limitato
2. Se uso una pratica, quasi sicuramente dovrò ritornare su di essa in tempi brevi… tanto vale tenersela sul tavolo

Un approccio gerarchico

- L'idea è che ho un certo numero di posizioni sulla mia scrivania
  - Man mano che mi serve una pratica la mando a prendere
  - Se ho la scrivania piena faccio portare a posto quelle pratiche che non mi servono più per fare spazio
- In questo modo posso contare su un'ampia capacità di memorizzazione, *ma* la maggior parte delle volte accedo ai dati molto velocemente

Torniamo ai calcolatori

- Fuor di metafora, all'interno di un calcolatore possiamo dare al processore (e ai programmi) l'illusione di avere uno spazio di memoria molto grande ma con grande velocità
- Questo è possibile grazie a due princìpi
  - Principio di località spaziale
  - Principio di località temporale

Località temporale

- Quando si fa uso di una locazione, la si riutilizzerà presto con elevata probabilità
- Esempio.

```
Ciclo:  slli x10, x22, 3
        add x10, x10, x25
        ld  x9, 0(x10)
        bne x9, x24, Esci
        addi x22, x22, 1
        beq x0, x0, Ciclo
Esci: ….
```

Queste istruzioni vengono ricaricate ogni volta che si esegue il ciclo

Località spaziale

- Quando si fa riferimento a una locazione, nei passi successivi si farà riferimento a locazioni vicine

```
Ciclo:      slli x10, x22, 3
            add x10, x10, x25
            ld  x9, 0(x10)
            bne x9, x24, Esci
            addi x22, x22, 1
            beq x0, x0, Ciclo
Esci:       ….
```

ISTRUZIONI: la modalità di esecuzione normale è il prelievo di istruzioni successive

DATI: Quando si scorre un array si va per word successive

Qualche dato

Facciamo riferimento a qualche cifra (relativa al 2012)

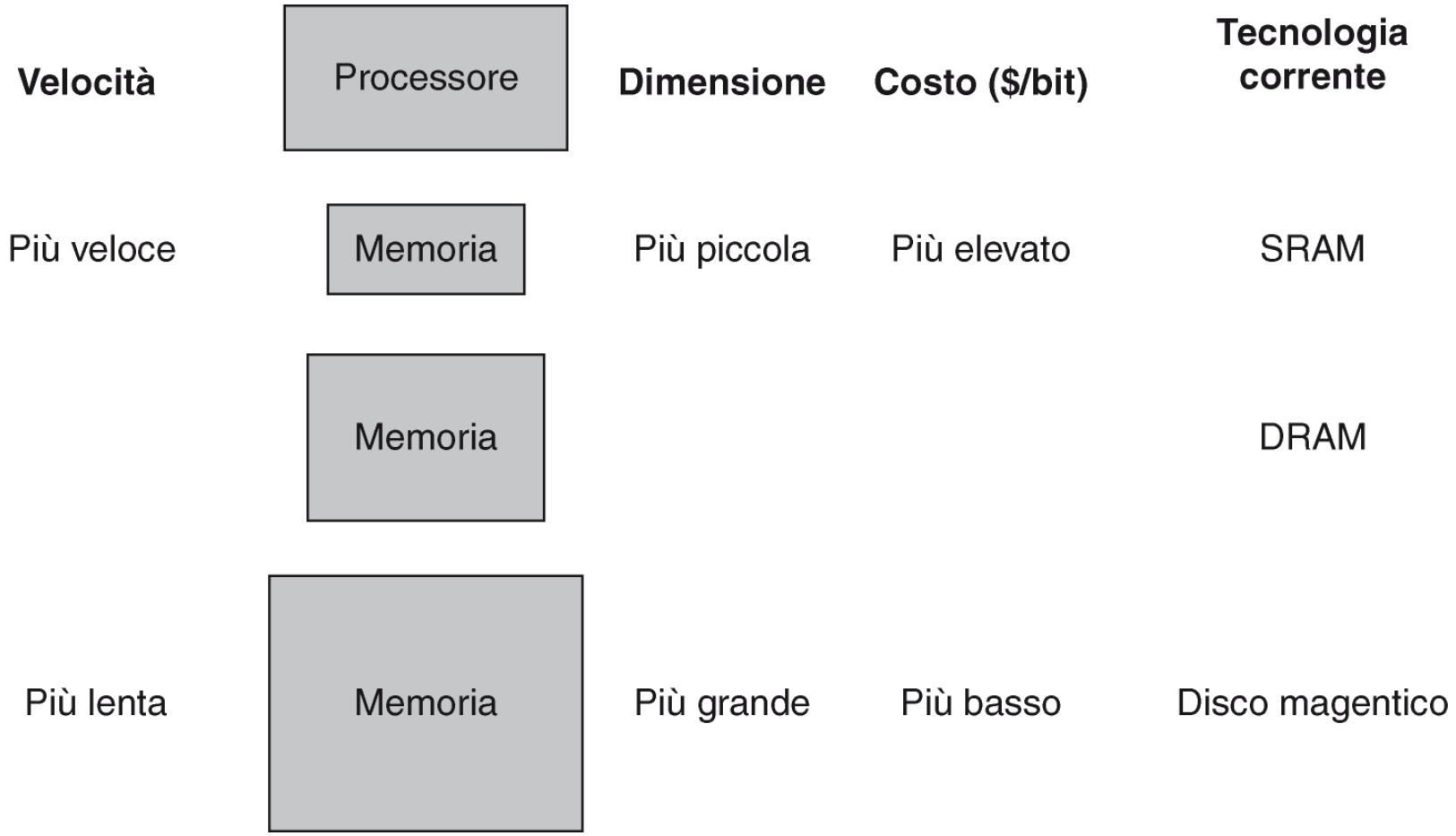
Tecnologia di Memoria	Tempo di accesso tipico	$ per GB (2010)	$ per GB (2012)
SRAM	0.5-2.5 ns	$2000 - $5000	$500 - $1000
DRAM	50 – 70 ns	$20 - $75	$10 - $20
Memoria flash	70 – 150 ns	$4 - $12	$0.75 - $1
Dischi magnetici	5 000 000 – 20 000 000 ns	$0.2 - $2	$0.05 - $0.1

Queste cifre suggeriscono l'idea della gerarchia di memoria

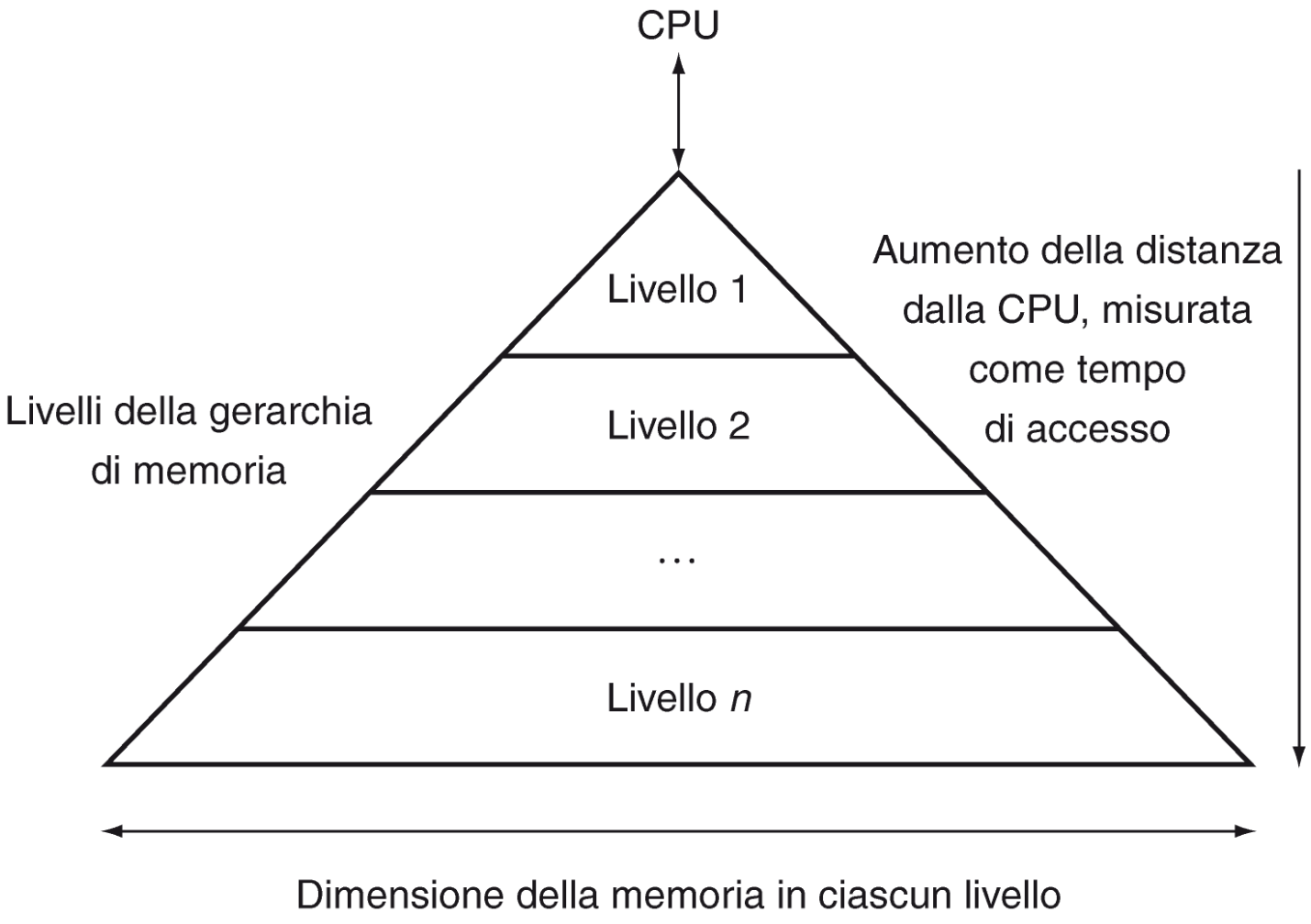
- Memorie piccole e veloci vicino al processore
- Memorie grandi e lente lontane dal processore

Gerarchia di memoria

- Struttura base

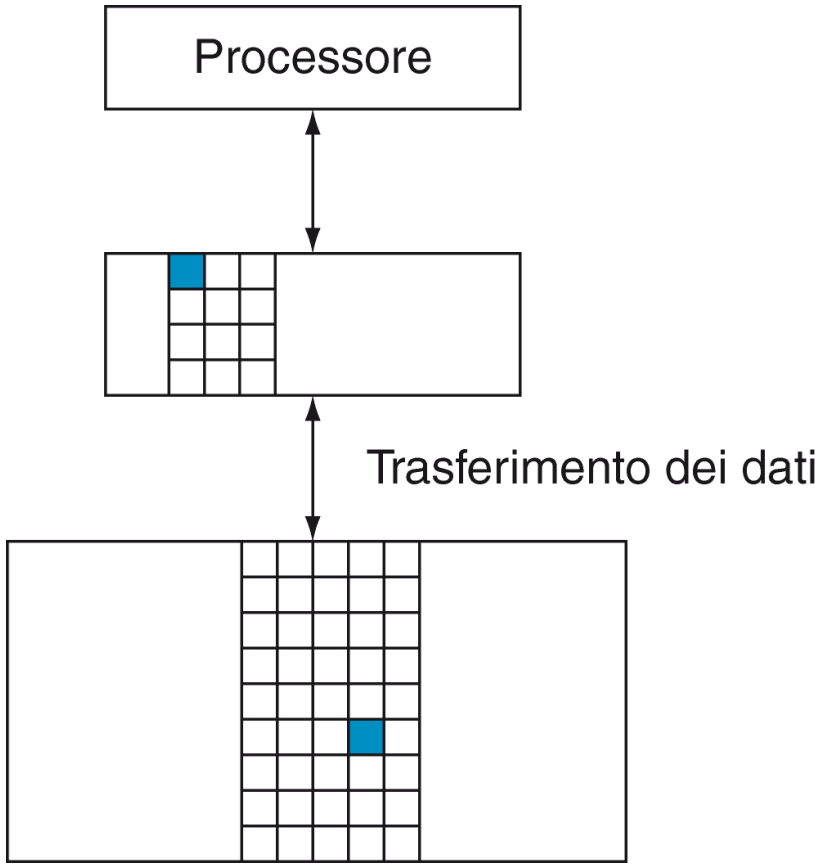


Struttura della gerarchia



Esempio

- Due livelli



Terminologia

- **Blocco:** unità minima di informazione che può essere presente o assente in ciascun livello
  - Un faldone nell'esempio di un archivio
- **Hit rate**: Frequenza di successo = frazione degli accessi in cui trovo il dato nel livello superiore
  - Quante volte trovo il faldone che mi serve nella scrivania
- **Miss Rate:** 1.0 – Hit rate: frazione degli accessi in cui non trovo il dato nel livello superiore
  - Quante volte devo andare a cercare un faldone in archivio
- **Tempo di hit:** Tempo che occorre per accedere al dato quando lo trovo nel livello superiore
  - Quanto mi ci vuole a leggere un documento nel faldone sulla scrivania
- **Penalità di miss:** quanto tempo mi ci vuole per accedere al dato se non lo trovo nel livello superiore
  - Tempo per spostare il faldone dall'archivio alla scrivania + tempo di accesso al documento nel faldone (dopo che arriva sulla scrivania)

Considerazioni

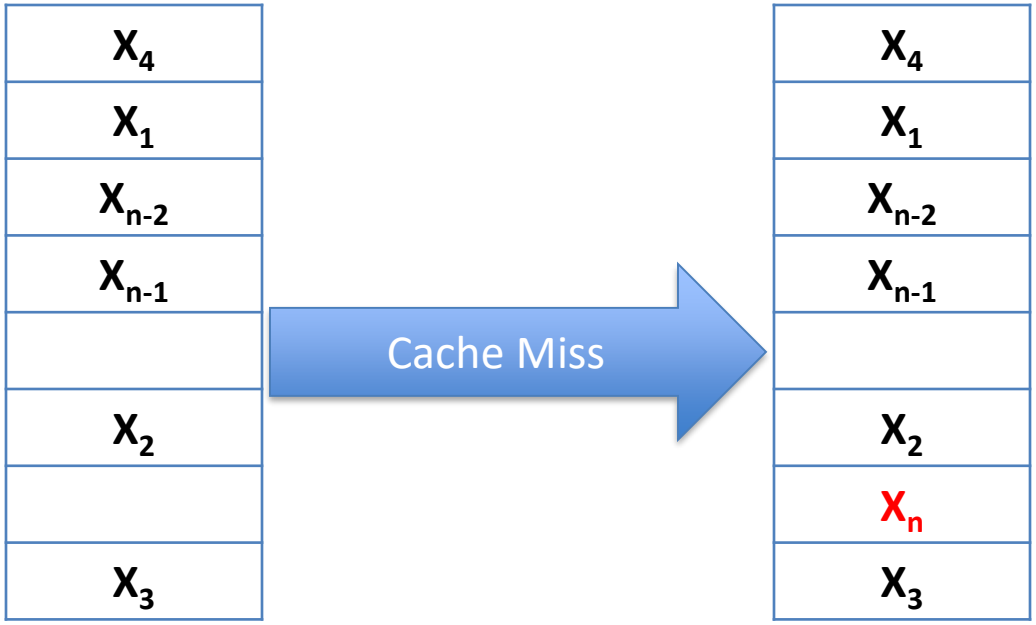
- La penalità di miss è molto maggiore del tempo di hit (e anche del trasferimento in memoria di un singolo dato)
  - Da cui il vantaggio
- Quindi è importante ridurre frequenza di miss
  - In questo ci aiuta il principio di località che il programmatore deve sfruttare al meglio

Cache

- **Cache**: posto sicuro [nascosto] dove riporre le cose
- Nascosto perché il programmatore non vede la cache direttamente (non vi accede)
  - L'uso della cache è interamente trasparente
- L'uso della cache fu sperimentato per la prima volta negli anni '70 e da allora è stato largamente adottato in tutti i calcolatori

Un esempio semplice

- Partiamo da un semplice esempio in cui i blocchi di cache siano costituiti da una sola word
- Supponiamo che a un certo punto il processore richieda la parola X_n che non è in cache



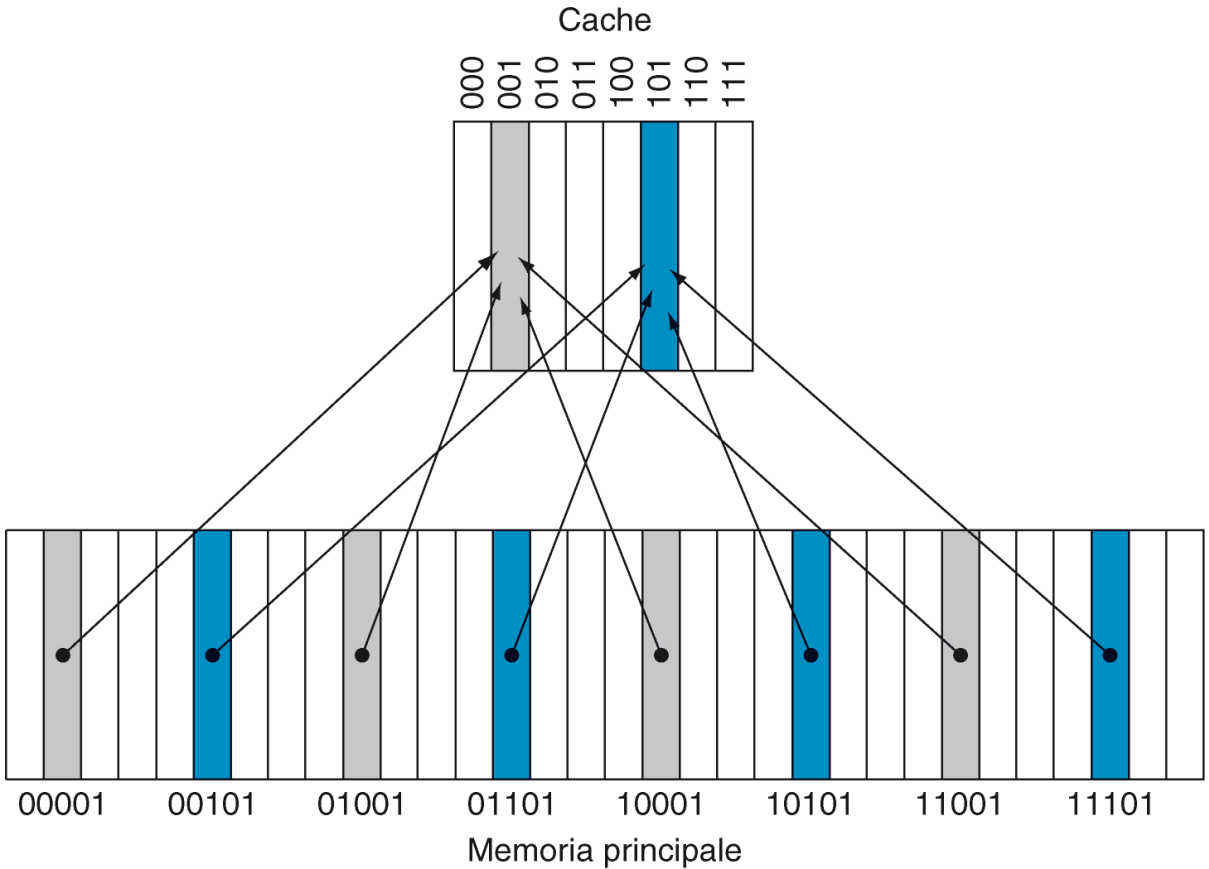
Domande

- Come facciamo a capire se un dato richiesto è nella cache?
- Dove andiamo a cercare per sapere se c'è?

- *Cache a mappatura diretta:* a ogni indirizzo della memoria corrisponde una precisa locazione della cache
- Possibilità: indirizzo locazione dove un indirizzo è mappato = (indirizzo blocco) *modulo* (numero di blocchi in cache)
  - Se la dimensione (=numero di blocchi) della cache è potenza di due, è sufficiente prendere i bit meno significativi dell'indirizzo in numero pari al logaritmo in base due della dimensione della cache

Esempio

- Se la nostra cache dispone di 8 parole devo prendere i tre bit meno significativi



Problema

- Siccome molte parole posso essere mappate sullo stesso blocco di cache, come facciamo a capire se, in un dato momento, vi si trova l'indirizzo che serve a noi?

- ***Si ricorre a un campo, detto tag, che contiene un'informazione sufficiente a risalire al blocco correntemente mappato in memoria***
- *Ad esempio possiamo utilizzare i bit più significativi di una parola, che non sono usati per la mappatura sulla cache, per trovare la locazione di memoria corrispondente all'indirizzo mappato da quel blocco di cache.*

Validità

- Usiamo i bit più significativi (due nell'esempio fatto) per capire se nel blocco di cache memorizziamo l'indirizzo richiesto
- Inoltre abbiamo un bit di validità che ci dice se quello che memorizziamo in un blocco di cache in un certo momento sia o meno valido

Esempio

Indice	V	Tag	Dati
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. Lo stato iniziale della cache dopo l'accensione del calcolatore

Accesso a 10110: Miss

Indice	V	Tag	Dati
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	S	10_{due}	Memoria (10110_{due})
111	N		

b. Dopo avere gestito una miss all'indirizzo (10110_{due})

Accesso a 11010: Miss

Accesso a 11010: hit

Indice	V	Tag	Dati
000	N		
001	N		
010	S	11_{due}	Memoria (11010_{due})
011	N		
100	N		
101	N		
110	S	10_{due}	Memoria (10110_{due})
111	N		

c. Dopo avere gestito una miss all'indirizzo 11010_{due}

Indice	V	Tag	Dati
000	S	10_{due}	Memoria (10000_{due})
001	N		
010	S	11_{due}	Memoria (11010_{due})
011	N		
100	N		
101	N		
110	S	10_{due}	Memoria (10110_{due})
111	N		

d. Dopo avere gestito una miss all'indirizzo 10000_{due}

Indice	V	Tag	Dati
000	S	10_{due}	Memoria (10000_{due})
001	N		
010	S	11_{due}	Memoria (11010_{due})
011	S	00_{due}	Memoria (00011_{due})
100	N		
101	N		
110	S	10_{due}	Memoria (10110_{due})
111	N		

e. Dopo avere gestito una miss all'indirizzo 00011_{due}

Accesso a 10010: miss

Indice	V	Tag	Dati
000	S	10_{due}	Memoria (10000_{due})
001	N		
010	S	10_{due}	Memoria (10010_{due})
011	S	00_{due}	Memoria (00011_{due})
100	N		
101	N		
110	S	10_{due}	Memoria (10110_{due})
111	N		

f. Dopo avere gestito una miss all'indirizzo 10010_{due}

Esempio RISC-V a 64 bit

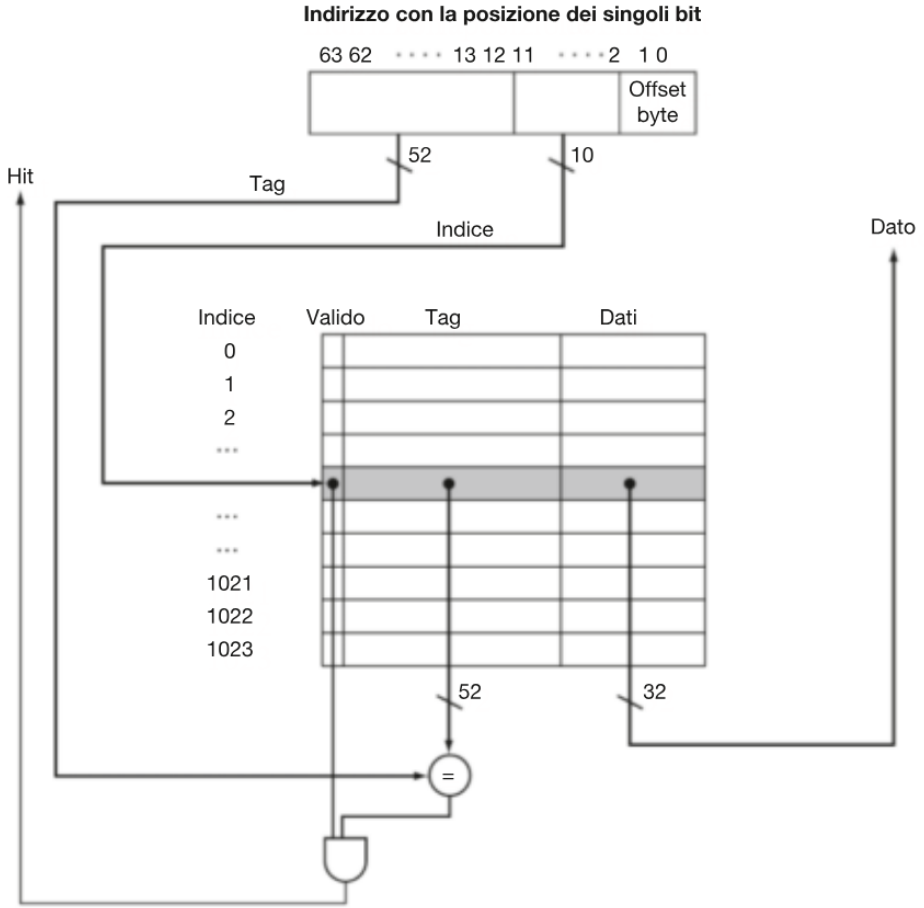
- Esempio
  - Indirizzo su 64 bit
  - Cache a mappatura diretta
  - Dimensioni della cache: 2^n blocchi, di cui n bit usati per l'indice
  - Dimensione del blocco di cache*:* 2^m parole, ossia 2^{m+2} byte (m bit usati per individuare una parola nel blocco), due bit per individuare un byte in una parola

In questo caso la dimensione del tag è data da:

$$64-(n+m+2)$$

Schema di risoluzione

- Un indirizzo viene risolto in cache con il seguente schema (n=10, m=0, dimensione tag = 52):



Se il campo tag è uguale ai 52 bit più significative dell'indirizzo e se il bit di validità è 1, allora si ha una hit e si da il dato al processore, altrimenti scatta una miss.

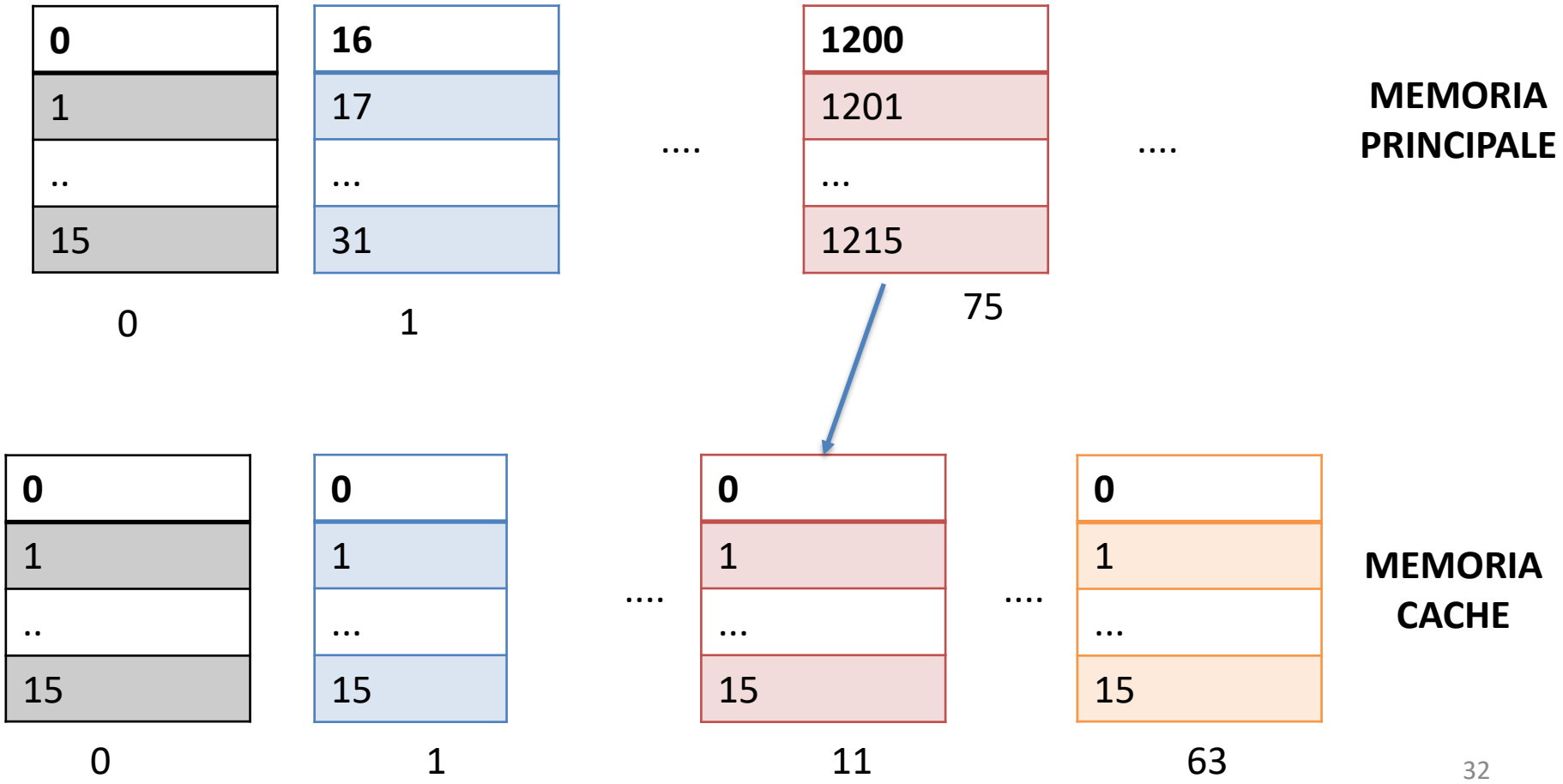
Esempio

- Si consideri una cache con 64 blocchi di 16 byte ciascuno. A quale numero di blocco corrisponde l'indirizzo 1200 espresso in byte?
- Blocco identificato da:
  (indirizzo blocco) *modulo* (numero blocchi in cache)
- Dove:

$$\text{Indirizzo Blocco} = \frac{\text{Indirizzo del Dato in byte}}{\text{Byte per blocco}}$$

Esempio

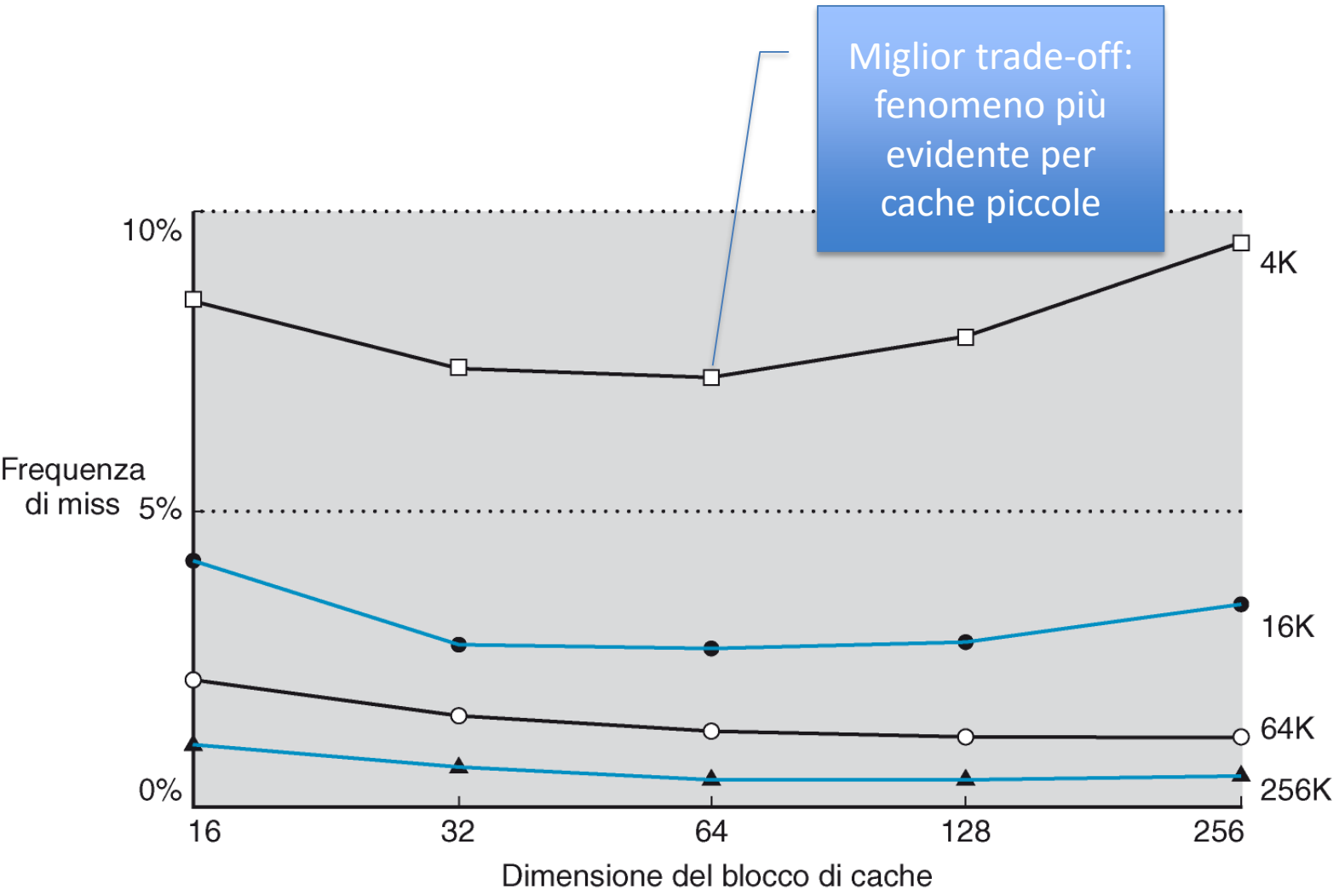
- Quindi l'indirizzo del blocco è 1200/16=75
- Blocco contente il dato è: 75 modulo 64 = 11



Trade-off

- Blocchi di cache molto grandi esaltano la località spaziale e da questo punto di vista diminuiscono le probabilità di miss
- Tuttavia, a parità di dimensioni della cache avere pochi blocchi diminuisce l'efficacia nello sfruttamento della località temporale
- Quindi abbiamo un trade-off
- Inoltre avere dei *miss* con blocchi grandi porta a un costo di gestione alto (bisogna spostare molti byte)

Frequenza delle miss



Gestione delle miss

- La presenza di una cache non modifica molto il funzionamento del processore (con pipeline) fino a che abbiamo delle hit
  - Il processore non si «accorge» neanche della presenza della cache
  - In caso di miss, bisogna generare uno stallo nella pipeline e gestire il trasferimento da memoria principale alla cache (ad opera della circuiteria di controllo)

Gestione delle miss

- Ad esempio, per una miss sulla memoria istruzioni, bisognerà:
  1. Inviare il valore PC – 4 alla memoria (PC viene incrementato all'inizio, quindi la miss è su PC-4)
  2. Comandare alla memoria di eseguire una lettura e attenderne il completamento
  3. Scrivere il blocco che proviene dalla memoria della cache aggiornando il tag
  4. Far ripartire l'istruzione dal fetch, che stavolta troverà l'istruzione in cache

Scritture

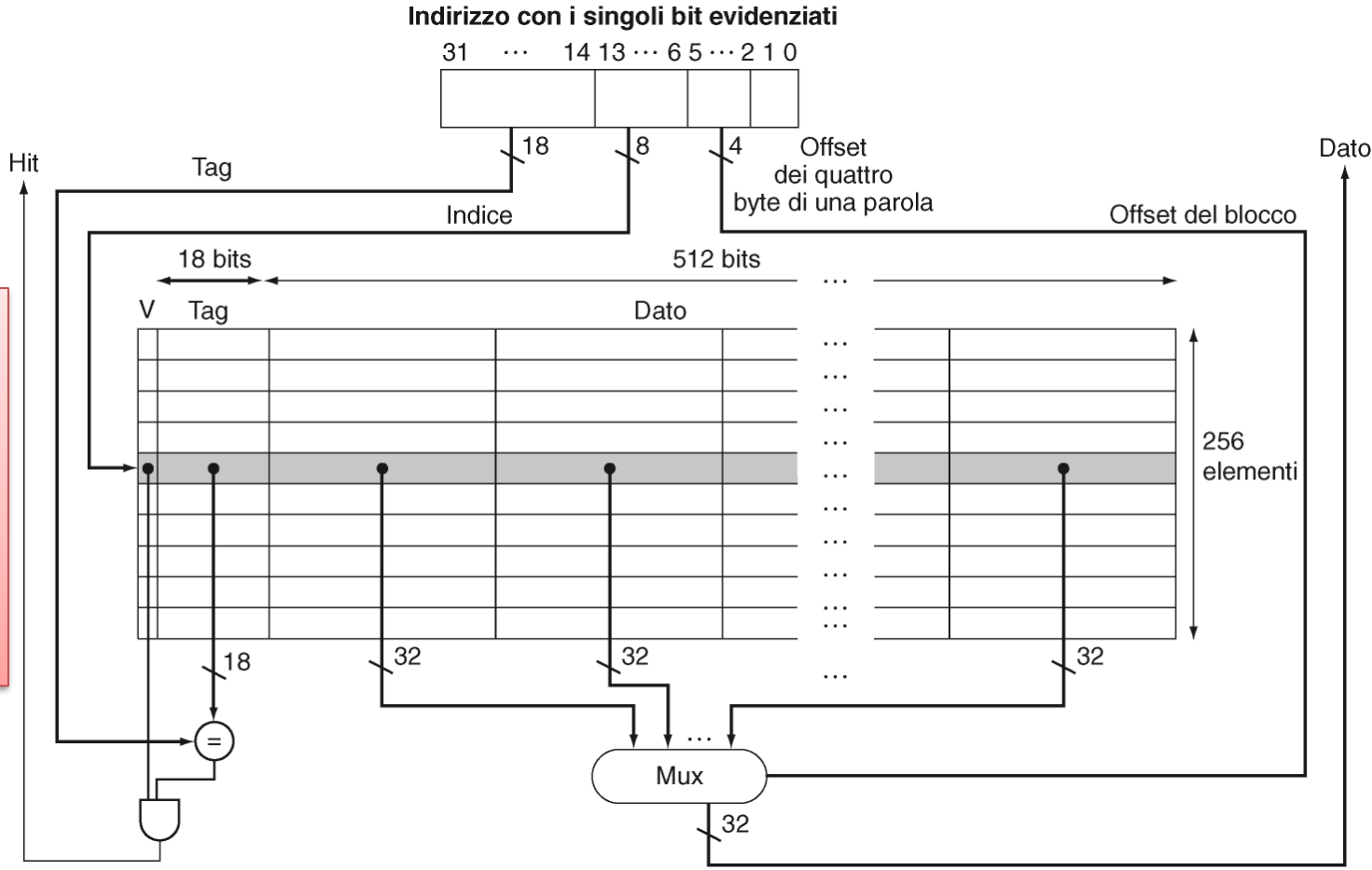
- Gli accessi in lettura alla memoria dati avvengono con la stessa logica
- Gli accessi in scrittura sono un po' più delicati perché possono generare problemi di consistenza
- Una politica è la cosiddetta **write-through**
  - Ogni scrittura viene direttamente effettuata in memoria principale (sia che si abbia una hit che una miss)
  - In questo modo non ho problemi di consistenza, ma le scritture sono molto costose
  - Posso impiegare un buffer di scrittura (una coda in cui metto tutte le scritture che sono in attesa di essere completate)

Scritture

- Un’altra possibile politica è la **write-back**
  - Se il blocco è in cache le scritture avvengono localmente in cache e l’update viene fatto solo quando il blocco viene rimpiazzato (o quando una locazione nel blocco viene acceduta da un altro processore su architetture multi-core)
  - Questo schema è conveniente quando il processore genera molte scritture e la memoria non ce la fa a «stargli dietro»

Esempio
FastMath Intrinsity
(basato su architettura MIPS)

- Cache di 16K
- 16 parole per blocco
- Possibilità di operare in write-through o in write-back



Esempio
FastMath Intrinsity
(basato su architettura MIPS)

Tipiche performance misurate su benchmark SPEC CPU2000

Frequenza di miss per le istruzioni	Frequenza di miss per i dati	Frequenza di miss totale
0,4%	11,4%	3,2%

Cache associative

- Le cache a mappatura diretta sono piuttosto semplici da realizzare
- Tuttavia hanno un problema: se ho spesso bisogno di locazioni di memoria che si mappano sullo stesso blocco, ho cache miss in continuazione
- All'estremo opposto ho una cache completamente associativa
  - Posso mappare qualsiasi blocco in qualsiasi blocco di cache

Cache completamente associativa

- Il problema per le cache completamente associative è che devo cercare ovunque il dato (il tag è tutto l'indirizzo del blocco)
- Per effettuare la ricerca in maniera efficiente, devo farla su tutti i blocchi in parallelo
- Per questo motivo ho bisogno di *n* comparatori (uno per ogni blocco di cache) che operino in parallelo
- Il costo HW è così alto che si può fare solo per piccole cache

Cache set-associativa

- Le cache set-associative sono un via di mezzo tra le due che abbiamo visto
- In sostanza ogni blocco di memoria può essere mappato su una linea di *n* blocchi diversi di cache (*n* «vie»)
- Quindi combiniamo due idee
  - Associamo ciascun blocco di memoria a una certa linea (e quindi a uno degli *n* blocchi di quella linea su cui possiamo mappare il blocco di memoria)
  - All'interno della linea, effettuiamo una **ricerca parallela** come se avessimo una cache completamente associativa

Mappatura del blocco

- In una cache a mappatura diretta un blocco di memoria viene mappato in un blocco di cache dato da:

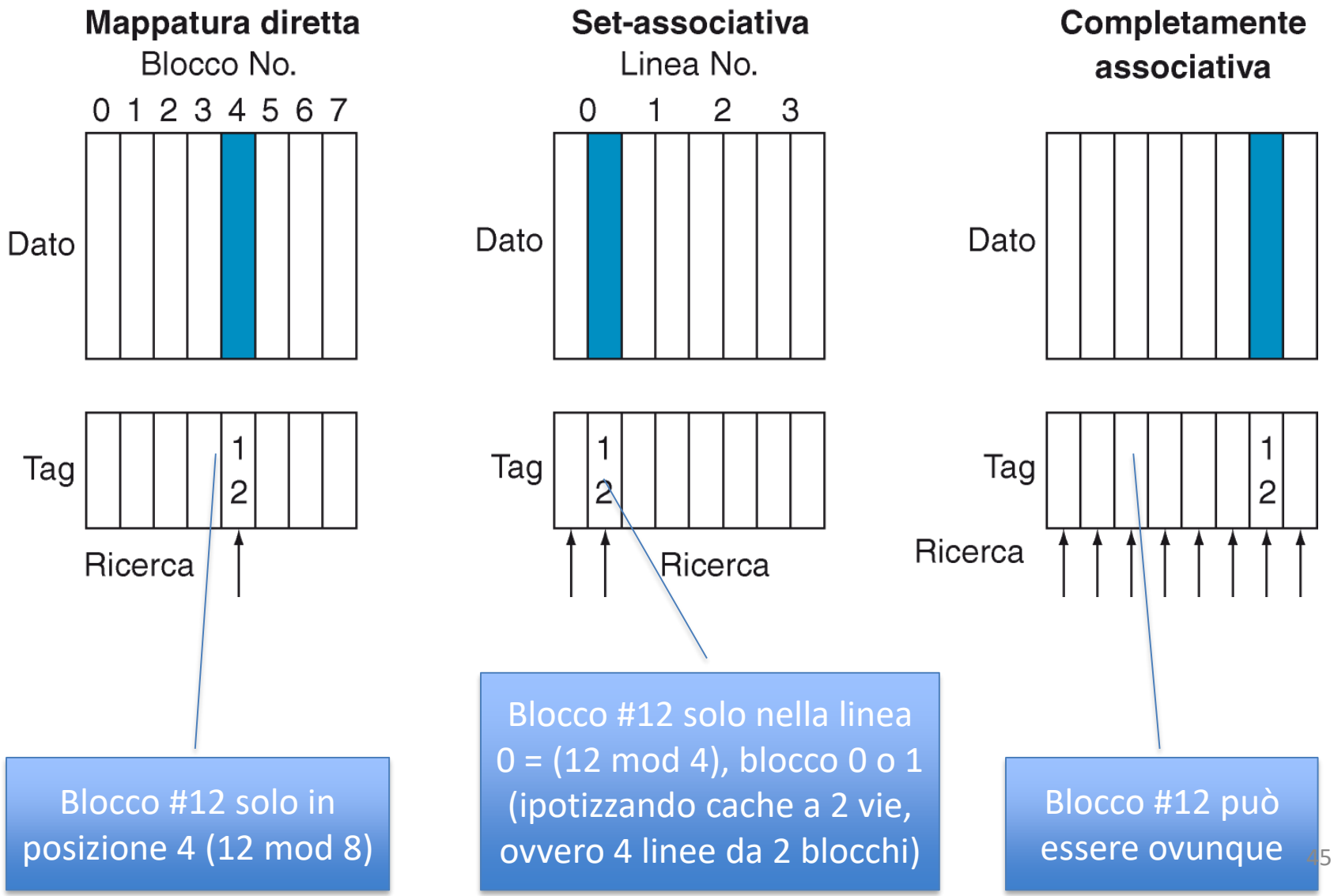
  (indirizzo blocco) *modulo* (numero **blocchi** della cache)

- In una cache set-associativa un blocco di memoria viene mappato nella linea data da:

  (indirizzo blocco) *modulo* (numero **linee** della cache)

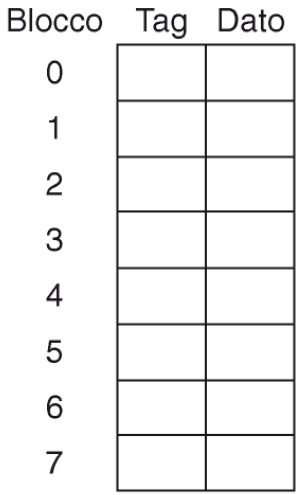
- Quindi, per trovare il blocco all'interno della linea dobbiamo confrontare (in parallelo) il tag del blocco con tutti i tag dei blocchi di quella linea

Posizione del blocco

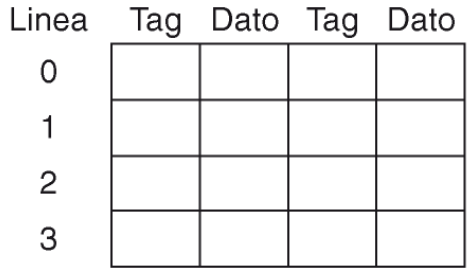


Varie configurazioni

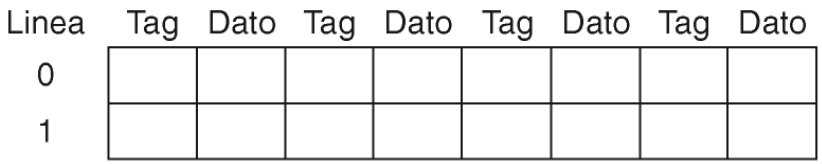
Cache set-associativa a una via
(a mappatura diretta)



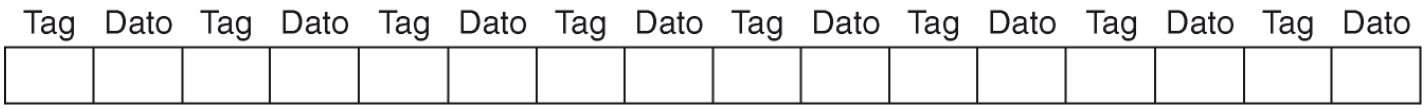
Cache set-associativa a due vie



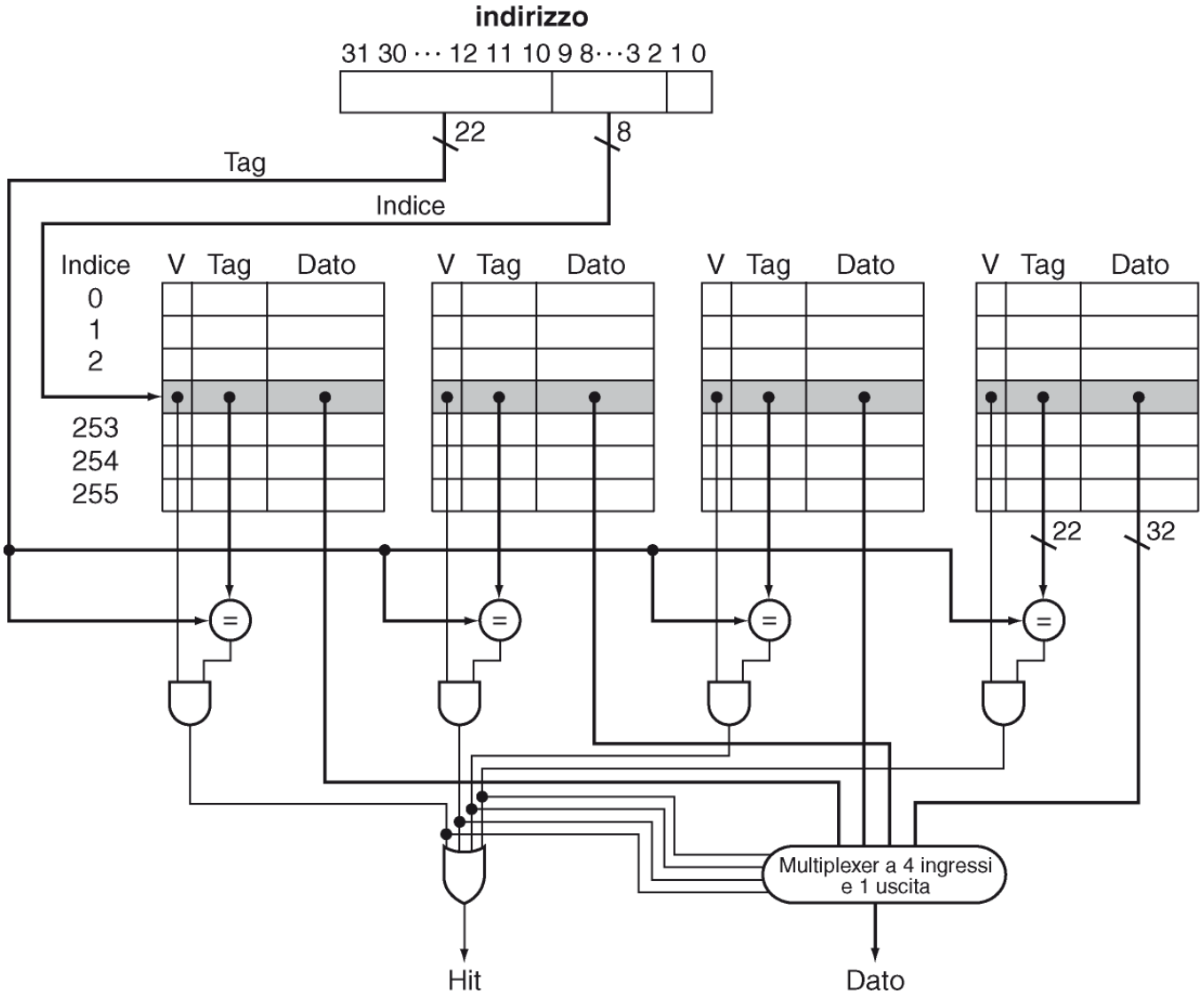
Cache set-associativa a quattro vie



Cache set-associativa a otto vie (completamente associativa)



Schema per cache a 4 vie



Vantaggi dell'associatività

Associatività	Frequenza di miss
1	10,3%
2	8,6%
4	8,3%
8	8,1%

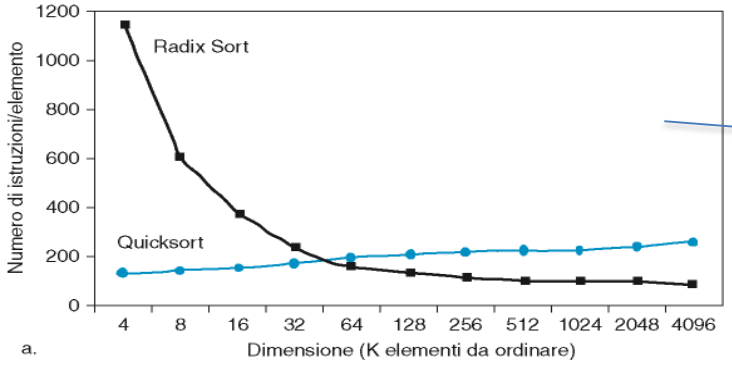
Aumentando l'associatività abbiamo vantaggi (frequenza di miss) e svantaggi (complessità). La scelta viene fatto tenendo presente questo trade-off

Un problema in più

- Nelle cache a mappatura diretta quando ho una cache miss sicuramente so chi sostituire (l'unico blocco in cui posso mapparmi)
- Nelle cache associative ho più scelte: se la linea è piena chi sostituisco?
- Varie politiche
  - FIFO
  - Least Recently Used

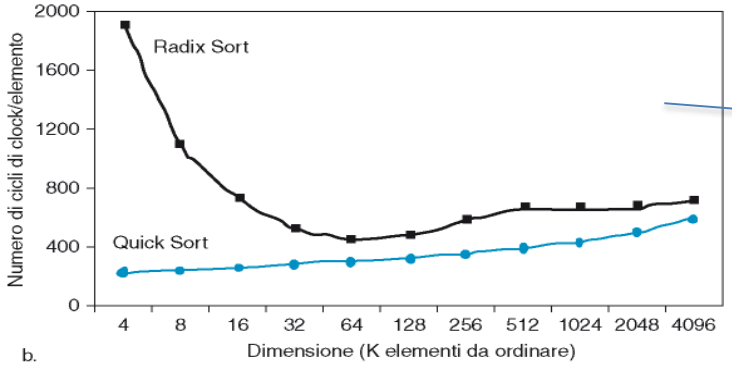
Richiede una serie di bit in più per contare l'ultimo accesso

Ok, ma a che ci serve questa roba?



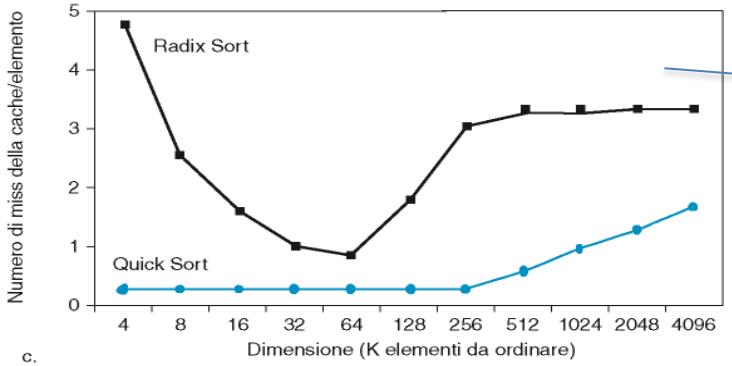
a.

Numero di istruzioni per elemento da ordinare



b.

Tempo di esecuzione



c.

Cache miss

CALCOLATORI
I/O

Marco Roveri
marco.roveri@unitn.it

Lezione basata su materiale preparato dai Prof. Luigi Palopoli e Giovanni Iacca

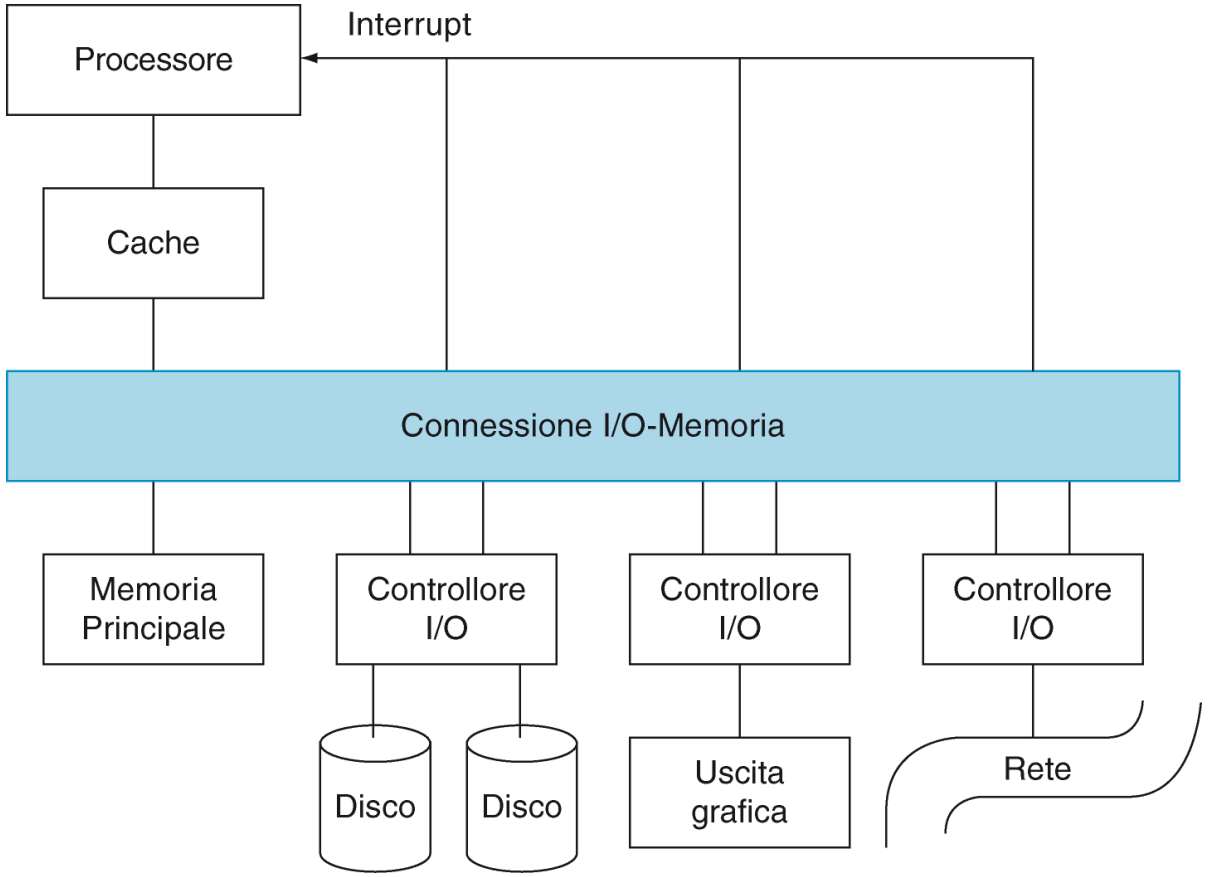
UNIVERSITÀ DEGLI STUDI DI TRENTO
Dipartimento di Ingegneria
e Scienza dell'Informazione

La necessità di comunicare

- Un calcolatore è completamente inutile senza la possibilità di caricare/salvare dati e di comunicare con l'esterno
- I dispositivi di I/O devono essere
  - espandibili
  - eterogenei
- I dispositivi di I/O sono molto vari e la tipologia di prestazione è diversa
  - In alcuni casi interessa il tempo di accesso (la latenza) e il tempo di risposta
    - ✓ Es. dispositivi interattivi come tastiere o mouse
  - In altri casi siamo interessati al throughput
    - ✓ Caso di dischi o interfacce di rete

Un semplice schema

- I dispositivi sono collegati al processore da un dispositivo di comunicazione chiamato *bus*



Classificazione

- I dispositivi di I/O sono di vario tipo e possono essere classificati in vari modi
  - Comportamento: che operazioni posso effettuare con il dispositivo (R/W)
  - Partner: può essere un uomo o una macchina
  - Velocità di trasferimento

Esempi

Dispositivo	Comportamento	Partner	Frequenza dati (Mbit/s)
Tastiera	Input (ingresso)	Uomo	0,0001
Mouse	Input (ingresso)	Uomo	0,0038
Input vocale	Input (ingresso)	Uomo	0,2640
Input audio	Input (ingresso)	Macchina	3,0000
Scanner	Input (ingresso)	Uomo	3,2000
Output vocale	Output (uscita)	Uomo	0,2640
Output audio	Output (uscita)	Uomo	8,0000
Stampante laser	Output (uscita)	Uomo	3,2000
Display grafico	Output (uscita)	Uomo	800,0000-8000,0000
Modem via cavo	Input o output	Macchina	0,1280-6,0000
Rete/LAN	Input o output	Macchina	100,000-10000,0000
Rete/LAN wireless	Input o output	Macchina	11,0000-54,0000
Disco ottico	Memoria	Macchina	80,0000-220,0000
Nastro magnetico	Memoria	Macchina	5,0000-120,0000
Memoria flash	Memoria	Macchina	32,0000-200,0000
Disco magnetico	Memoria	Macchina	800,0000-3000,0000

Prestazioni

- A seconda del tipo di applicazione, posso essere interessato a diverse prestazioni
  - Ad esempio per un sistema di streaming mi interessa il throughput
  - Per un sistema bancario, mi può servire massimizzare il numero di file di piccole dimensioni su cui opero contemporaneamente

Connessione tra processori e periferiche

- Le connessioni avvengono tramite delle strutture di comunicazione chiamate *bus*
- Esistono due tipi di bus
  - Bus processore/memoria:
    - ✓ specializzati, corti e veloci
  - Bus I/O
    - ✓ possono essere lunghi e permettono il collegamento con periferiche eterogenei
    - ✓ tipicamente, non sono collegati alla memoria in maniera diretta ma richiedono un bus processore/memoria o un bus di sistema
- Nelle prime architetture avevamo un unico grosso bus parallelo che collegava tutto
- Per problemi di clock e frequenze ora si usano architetture di comunicazione più complesse fatte di più bus paralleli condivisi e di bus seriali punto/punto

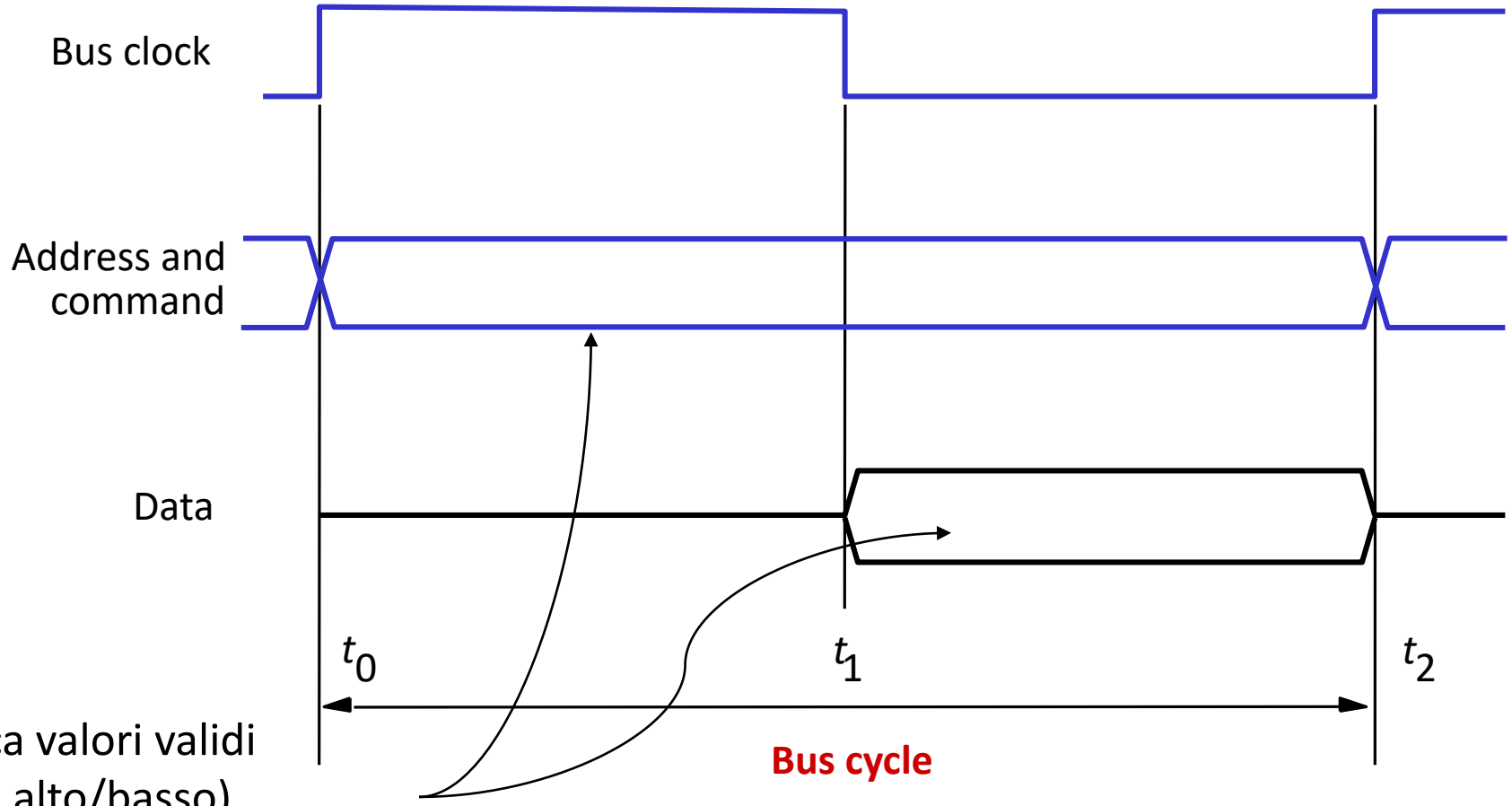
Terminologia

- Transazione di I/O
  - Invio indirizzo e spedizione o ricezione dei dati
- Input
  - Trasferimento di dati da una periferica verso la memoria dove il processore può leggerla
- Output
  - Trasferimento dalla memoria al dispositivo

Bus sincrono

- Tra le linee di controllo deve avere clock
- Le comunicazioni avvengono con un protocollo collegato al ciclo di clock
- Esempio: dato richiesto al clock n viene messo al clock n+5 sul bus

Bus sincrono: funzionamento base



Indica valori validi
(1/0, alto/basso)
su una struttura parallela

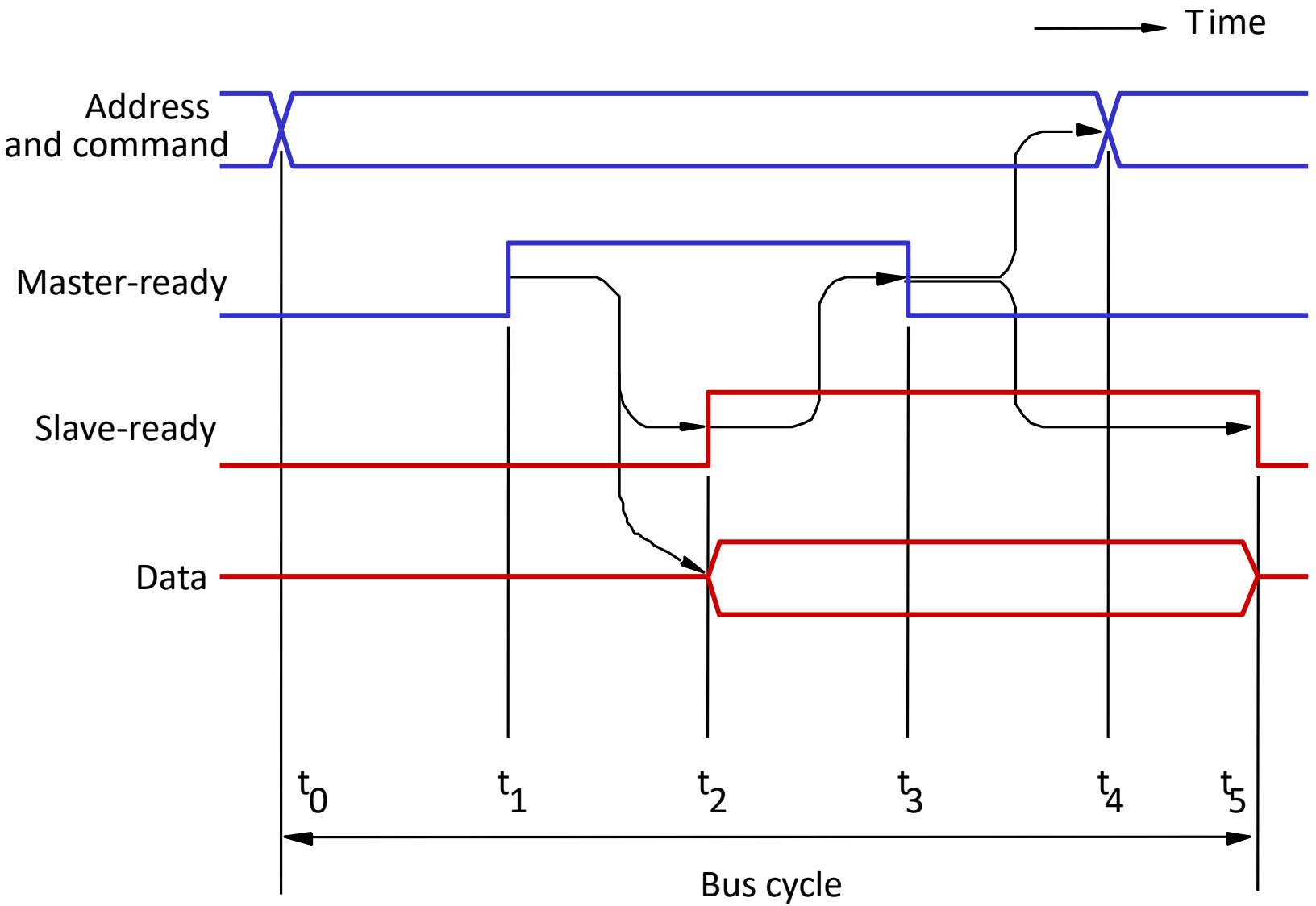
Bus sincrono

- Pro
  - Molto semplice da implementare (piccola macchina a stati finti)
  - Molto veloce (pochi segnali di controllo)
- Contro
  - Poca robustezza al *drift* del clock
  - Tutte le periferiche devono andare alla velocità del clock

Bus asincrono

- Per ovviare agli inconvenienti discussi si tende a usare interconnessioni asincrone
- In sostanza non abbiamo più un clock e tutte le transazioni sono governate da una serie di segnali di *handshake*
- Questo richiede l'introduzione di apposite linee di controllo per segnalare inizio e fine di transazioni, ma permette di collegare periferiche a velocità diversa

Ciclo di un bus asincrono



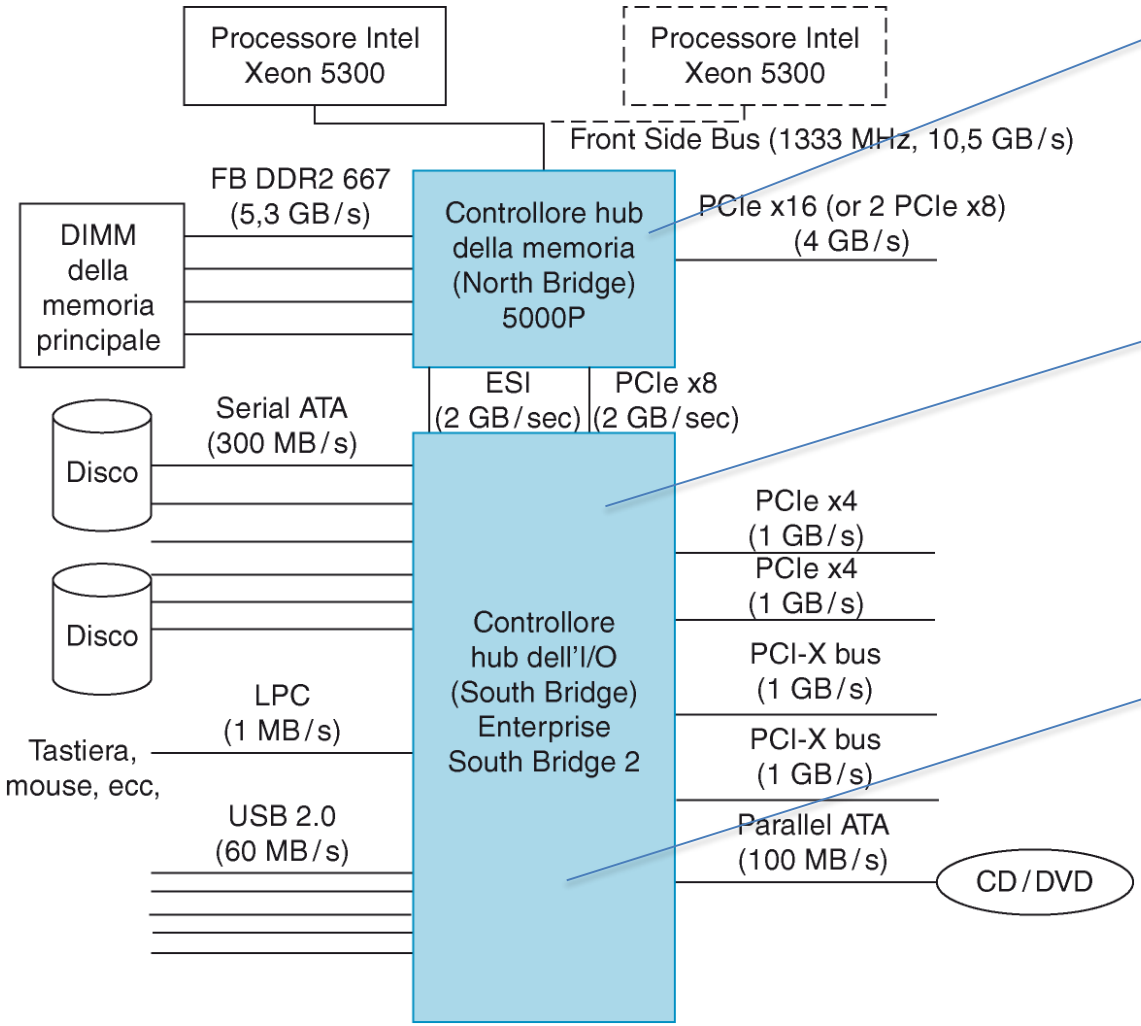
Bus asincrono

- Pro
  - Consente di essere robusto rispetto a ritardi
  - Consente di comunicare con periferiche di tipo diverso
- Contro
  - Lento nelle interazioni (diversi segnali di controllo devono circolare per riuscire a comunicare)
  - Circuiteria di gestione del protocollo complessa
- Spesso si usano tecnologie ibride (in cui c'è un segnale di clock) ma prevalentemente asincrone

Tecnologie (asincrone) attuali

Caratteristica	Firewire (1394)	USB 2.0	PCI Express	Serial ATA	Serial Attached SCSI
Utilizzo previsto	Esterno	Esterno	Interno	Interno	Esterno
Numero dispositivi per canale	63	127	1	1	4
Larghezza base dei dati (numero di segnali)	4	2	2 per linea	4	4
Larghezza di banda di picco teorica	50 MB/s (Firewire 400) o 100 MB/s (Firewire 800)	0,2 MB/s (low speed), 1,5 MB/s (full speed), o 60 MB/s (high speed)	250 MB/s per linea (1x); le schede PCIe sono disponibili in versione 1x, 2x, 4x, 8x, 16x o 32x	300 MB/s	300 MB/s
Collegamento a caldo	Sì	Sì	Dipende dalle dimensioni	Sì	Sì
Lunghezza massima del bus (piste in rame)	4,5 metri	5 metri	0,5 metri	1 metro	8 metri
Nome dello standard	IEEE 1394, 1394b	Forum degli implementatori USB	SIG PCI	SATA-IO	Comitato T10

Esempio x86



HUB per il controllo memoria

HUB per connessione ad altri bus

Trend più recente: incorporare il tutto all'interno del processore

Prospettiva del programmatore

- Rimane da capire
  - Come trasformare una richiesta di I/O in un comando per la periferica
  - Come trasferire i dati
  - Qual è il ruolo del sistema operativo?
- Riguardo al SO occorre osservare
  - Programmi che condividono il processore condividono anche il sistema di I/O
  - I trasferimenti dati vengono spesso effettuati usando interrupt, che hanno un impatto sulle funzionalità del SO.
    - ✓ Quindi devono essere eseguiti in una particolare modalità del processore (supervisor) cui solo il codice del *kernel* può accedere
  - Il controllo di operazioni I/O spesso si interseca con problematiche di concorrenza

Funzionalità richieste al SO

- Garantire che un dato utente abbia accesso ai dispositivi di I/O cui ha diritto (permessi) di accedere
- Fornire comandi di alto livello per gestire le operazioni di basso livello
- Gestire le interruzioni generate dai dispositivi di I/O (in maniera simile a quanto avviene con le eccezioni generate nei programmi)
- Ripartire l'accesso a ciascun dispositivo in maniera equa tra i vari programmi che lo richiedono

Requisiti

- Per implementare le funzionalità appena discusse occorre
  - Rendere possibile al SO di inviare comandi alle periferiche
  - Rendere possibile ai dispositivi notificare la corretta esecuzione di un'operazione
  - Consentire trasferimenti diretti di dati tra dispositivo e memoria

Come impartire i comandi ai dispositivi

- Questo si fa fornendo sulle relative linee di bus alcune «parole» di controllo
- Può essere fatto in due modi:
  - Scrivendo/leggendo in particolari locazioni di memoria (*memory mapped I/O*)
  - Tramite alcune istruzioni speciali (dedicate all'I/O)

Esempio

- Scrivendo una particolare parola in una locazione di memoria associata al dispositivo
  - Il sistema di memoria ignora la scrittura
  - Il controllore di I/O intercetta l'indirizzo particolare e trasmette il dato al dispositivo sotto forma di comando
- Queste particolari locazioni di memoria sono inaccessibili ai programmi utente ma solo al sistema operativo (quindi occorre una chiamata di sistema che faccia commutare il processore in modalità supervisore)
- Il dispositivo stesso può usare queste locazioni per trasmettere dati o pre-segnalare il suo stato
- Ad esempio posso chiedere la stampa di un carattere a terminale, e a stampa finita un particolare bit di un registro di stato mappato in memoria verrà commutato

Come trasmettere/ricevere i dati

- La modalità più semplice per trasferire i dati è la cosiddetta attesa attiva (polling)
- In sostanza si manda una comando di lettura/scrittura alla periferica e poi si fa un ciclo di attesa testando il bit di stato per vedere quando il dato è pronto

Esempio

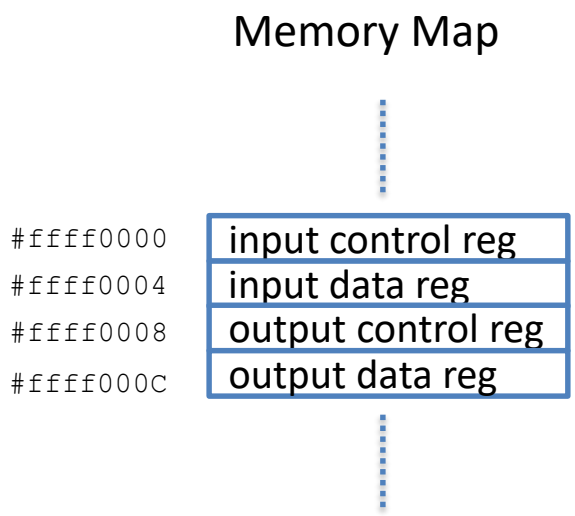
- Input: lettura dalla tastiera in x7

```
            lui  x5, 0xffff #ffff0000
Waitloop:   lw   x6, 0(x5) #control
            andi x6, x6,0x0001
            beq  x6, x0, Waitloop
            lw   x7, 4(x5) #data
```

- Output: stampa del dato da x7

```
            lui  x5, 0xffff #ffff0000
Waitloop:   lw   x6, 8(x5) #control
            andi x6,x6,0x0001
            beq  x6,x0, Waitloop
            sw   x7, 12(x5) #data
```

- Questo ciclo di attesa è chiamato *polling*



Costo del polling

- Consideriamo un processore a 500Mhz e supponiamo che occorrano 400 cicli di clock per un'operazione di polling. Qual è il costo percentuale?
  - Esempio 1: Mouse. Per non perdere movimenti da parte dell'utente occorre acquisire il dato 30 volte al secondo.
  - Esempio 2: Hard disk. I dati vengono trasferiti in blocchi di 16 byte a 8MB/s senza la possibilità di perdite.

Esempio 1 (Mouse)

- Cicli di clock al secondo spesi per il polling

  = 30 * 400 = 12000 clocks/sec

  - % Processor for polling:

    $12*10^3/500*10^6 = 0.002\%$

    $\Rightarrow$ Fare polling sul mouse ruba un utilizzo di processore trascurabile

Questo overhead viene pagato sempre, sia che ci sia il trasferimento, sia che non ci sia

Esempio 2 (Hard disk)

- Numero di volte/sec che occorre fare cicli di attesa per non perdere dati:

  = 8 MB/s /16B = 500K polls/sec

- Spesa in cicli di clock/sec

  = 500K * 400 = 200,000,000 clocks/sec

- % processore

  $200*10^6/500*10^6 = 40\%$

  $\Rightarrow$ Inaccettabile perché pagata sempre

CALCOLATORI
I/O

Marco Roveri
marco.roveri@unitn.it

Lezione basata su materiale preparato dai Prof. Luigi Palopoli e Giovanni Iacca

UNIVERSITÀ DEGLI STUDI DI TRENTO
Dipartimento di Ingegneria
e Scienza dell'Informazione

Costo del polling

- Consideriamo un processore a 500Mhz e supponiamo che occorrano 400 cicli di clock per un'operazione di polling. Qual è il costo percentuale?
  - Esempio 1: Mouse. Per non perdere movimenti da parte dell'utente occorre acquisire il dato 30 volte al secondo.
  - Esempio 2: Hard disk. I dati vengono trasferiti in blocchi di 16 byte a 8MB/s senza la possibilità di perdite.

Esempio 1 (Mouse)

- Cicli di clock al secondo spesi per il polling

  = 30 * 400 = 12000 clocks/sec

  - % Processor for polling:

    $12*10^3/500*10^6 = 0.002\%$

    $\Rightarrow$ Fare polling sul mouse ruba un utilizzo di processore trascurabile

Questo overhead viene pagato sempre, sia che ci sia il trasferimento, sia che non ci sia

Esempio 2 (Hard disk)

- Numero di volte/sec che occorre fare cicli di attesa per non perdere dati:

  = 8 MB/s /16B = 500K polls/sec

- Spesa in cicli di clock/sec

  = 500K * 400 = 200,000,000 clocks/sec

- % processore

  $200*10^6/500*10^6 = 40\%$

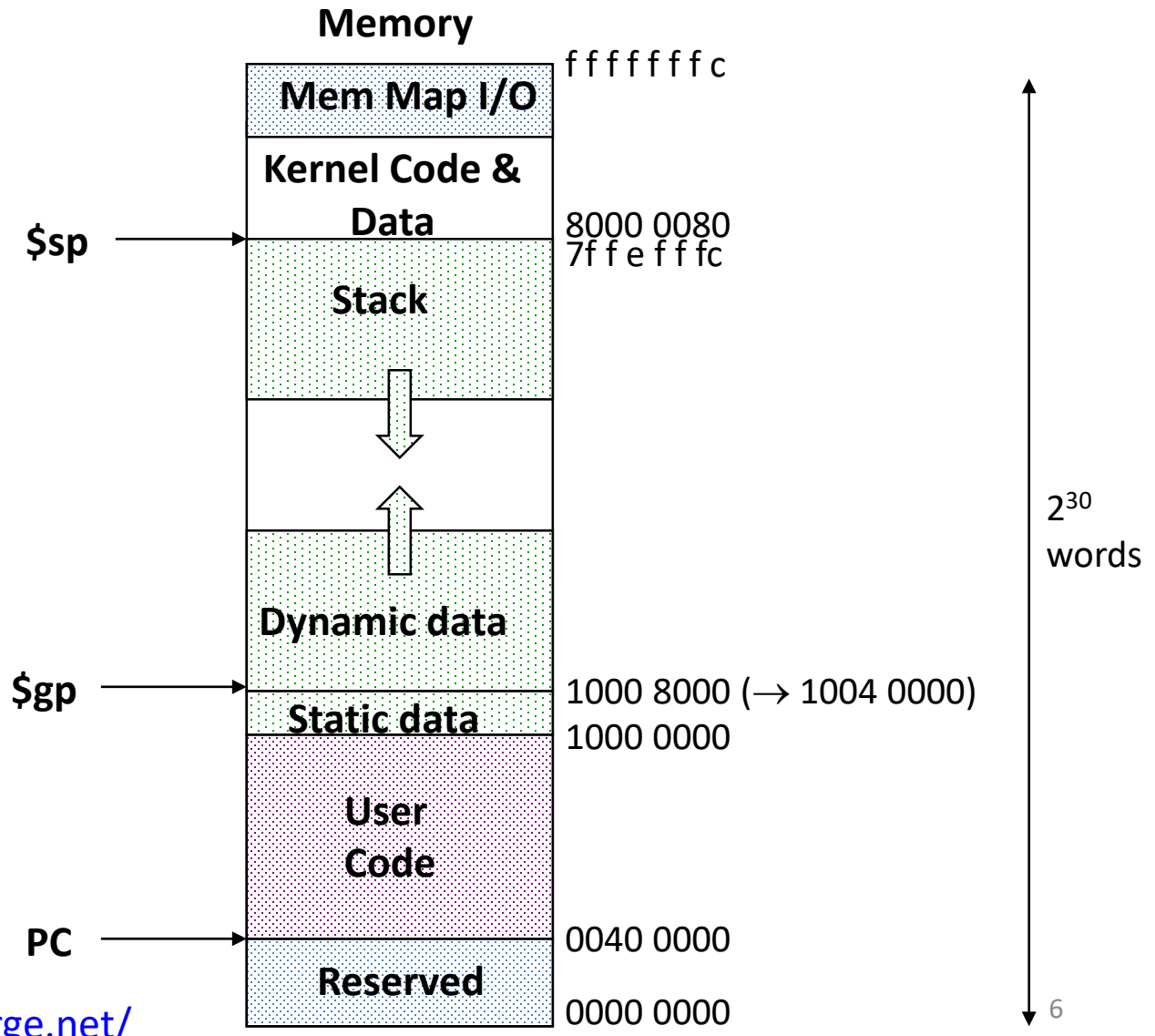
  $\Rightarrow$ Inaccettabile perché pagata sempre

Considerazioni sul polling

- L'attesa attiva fa perdere tempo al processore che dedica cicli macchina a letture inutili
- Il polling può essere usato quando le operazioni di I/O avvengono con velocità di trasferimento predeterminata (es. applicazioni di controllo) e comunque il processore ha poco altro da fare
- Sicuramente se i dati vengono trasferiti con elevati *bitrate* il ciclo di attesa attiva dura poco
- In altri casi lo spreco è inaccettabile e per questo motivo è stato inventato l'I/O a interruzione di programma (interrupt driven I/O)

Organizzazione memoria SPIM (MIPS)

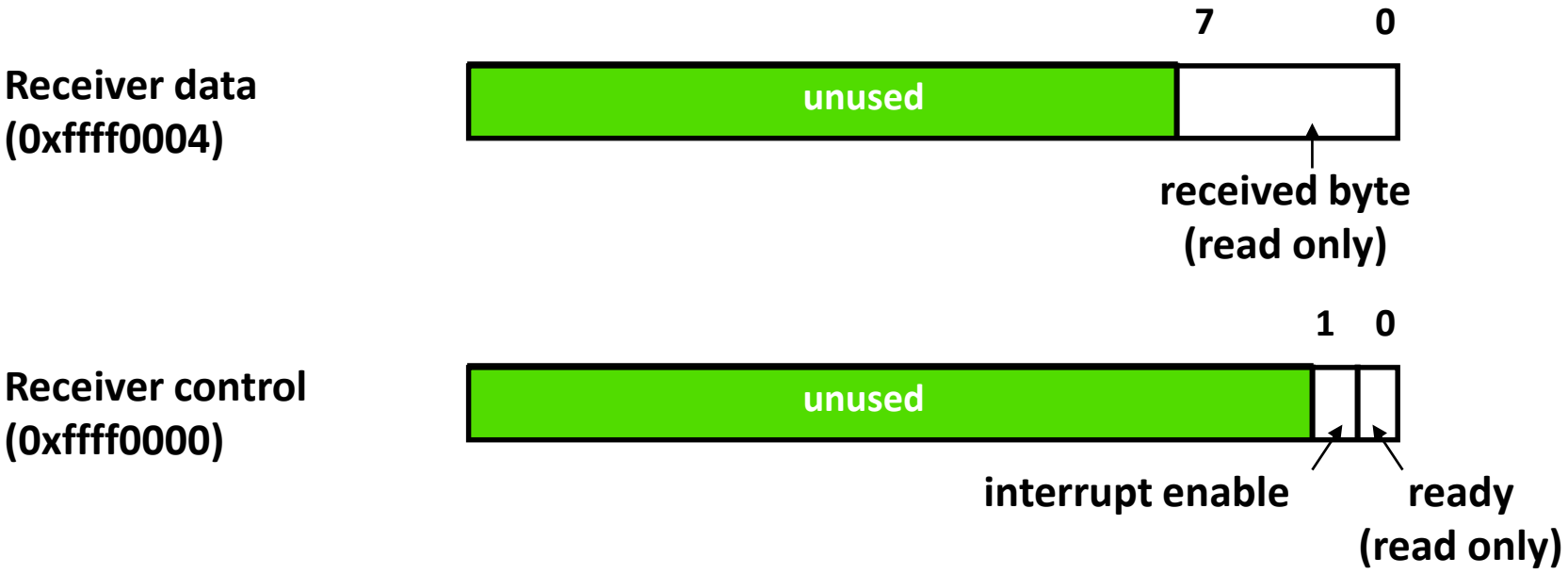
- Partiamo da una tipica organizzazione di memoria (e.g quella dell'emulatore SPIM)



http://spimsimulator.sourceforge.net/

Controllo del terminale in SPIM (INPUT)

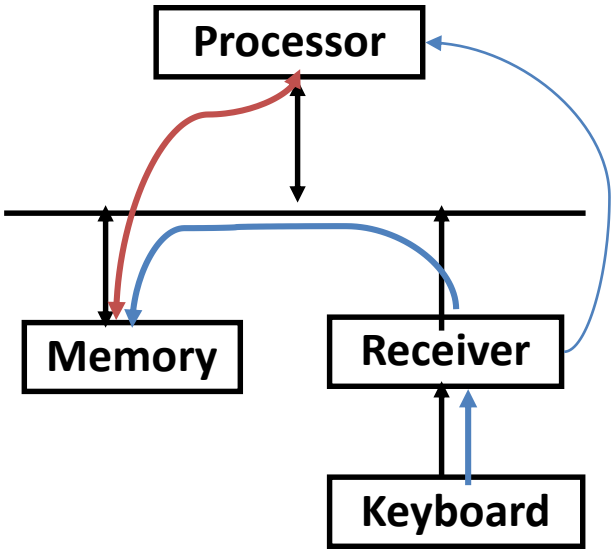
- Osserviamo da vicino le locazioni per il controllo del terminale (INPUT)

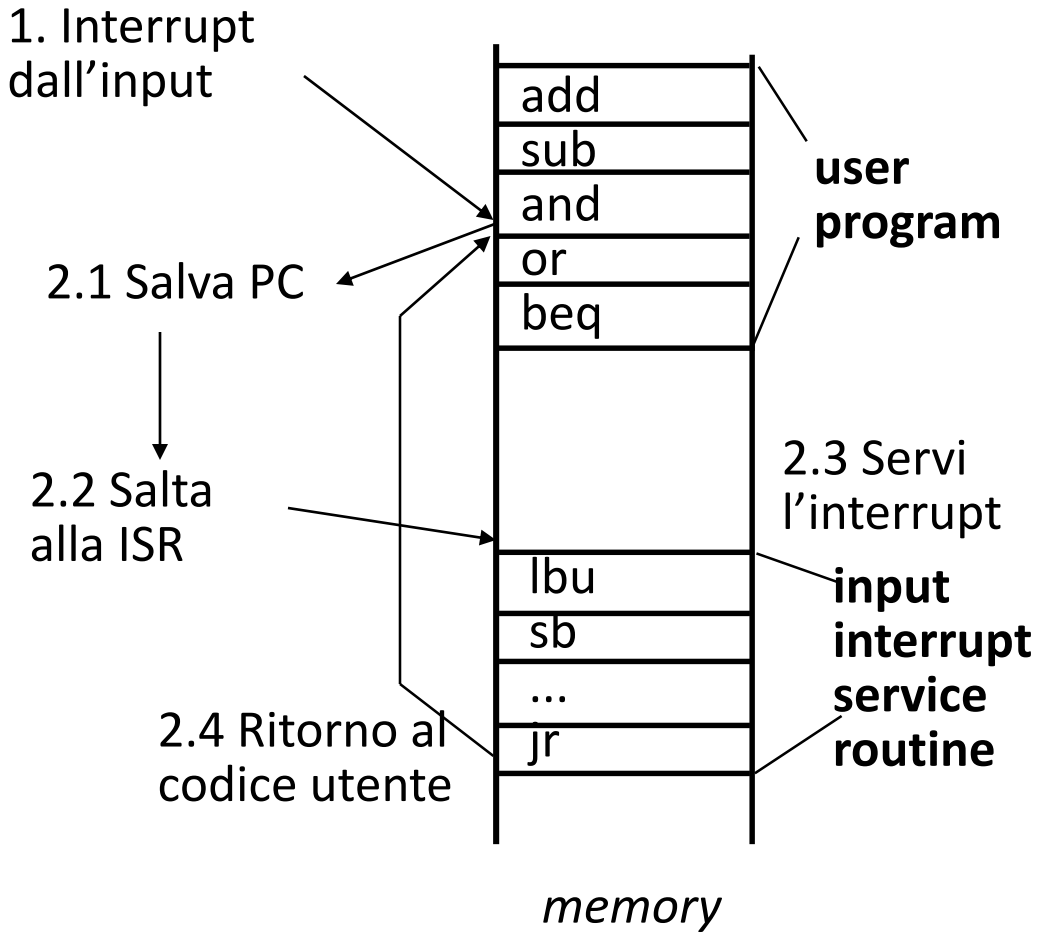


Interruzioni di programma

- Un'interruzione I/O è un segnale usato per segnalare al processore che la periferica è pronta ad eseguire il trasferimento richiesto
  - Le interruzioni possono avere diverso grado di urgenza (è possibile definire priorità).
  - Occorre un modo per segnalare al processore quale periferica richiede l'interruzione.
- Le interruzioni I/O sono sempre asincrone rispetto all'esecuzione delle istruzioni
  - Non esistono particolari istruzioni assembly per eseguire le interruzioni. Un'interruzione può arrivare mentre una qualsiasi istruzione viene eseguita, e dà comunque modo di terminare l'esecuzione dell'istruzione.
    - ✓ Il programmatore può spesso differire l'esecuzione dell'interruzione a un momento più conveniente (es. sezioni non interrompibili nel codice del kernel).
- Vantaggio
  - Non occorre interrompere l'esecuzione del programma se non quando il dato può essere effettivamente riferito in memoria.
- Svantaggio – occorre un hardware speciale per:
  - Permettere ai dispositivi di I/O di generare un'interruzione.
  - Rilevare l'interruzione, salvare lo stato del processore per eseguire una particolare routine di servizio (Interrupt Service Routine, ISR) e poi riprendere dal punto dove si era interrotto.

Input a interruzione di programma

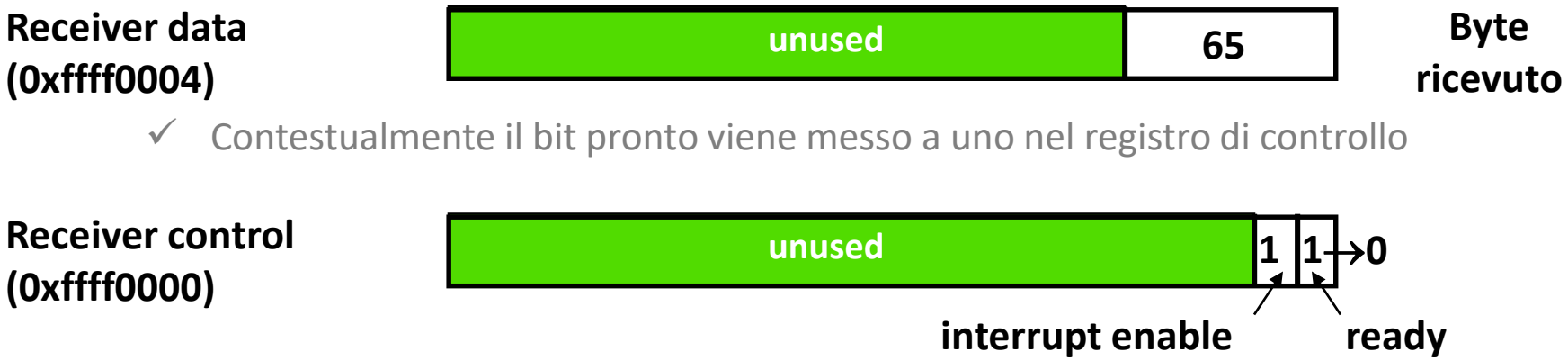




Nel mezzo c'è una commutazione in modalità «Supervisore» (solo il SO può gestire l'interruzione)

Esempio controllo terminale SPIM

1. La periferica indica con un'interruzione che ha un nuovo carattere dalla tastiera nell'opportuno registro di ricezione



   - ✓ Contestualmente il bit pronto viene messo a uno nel registro di controllo

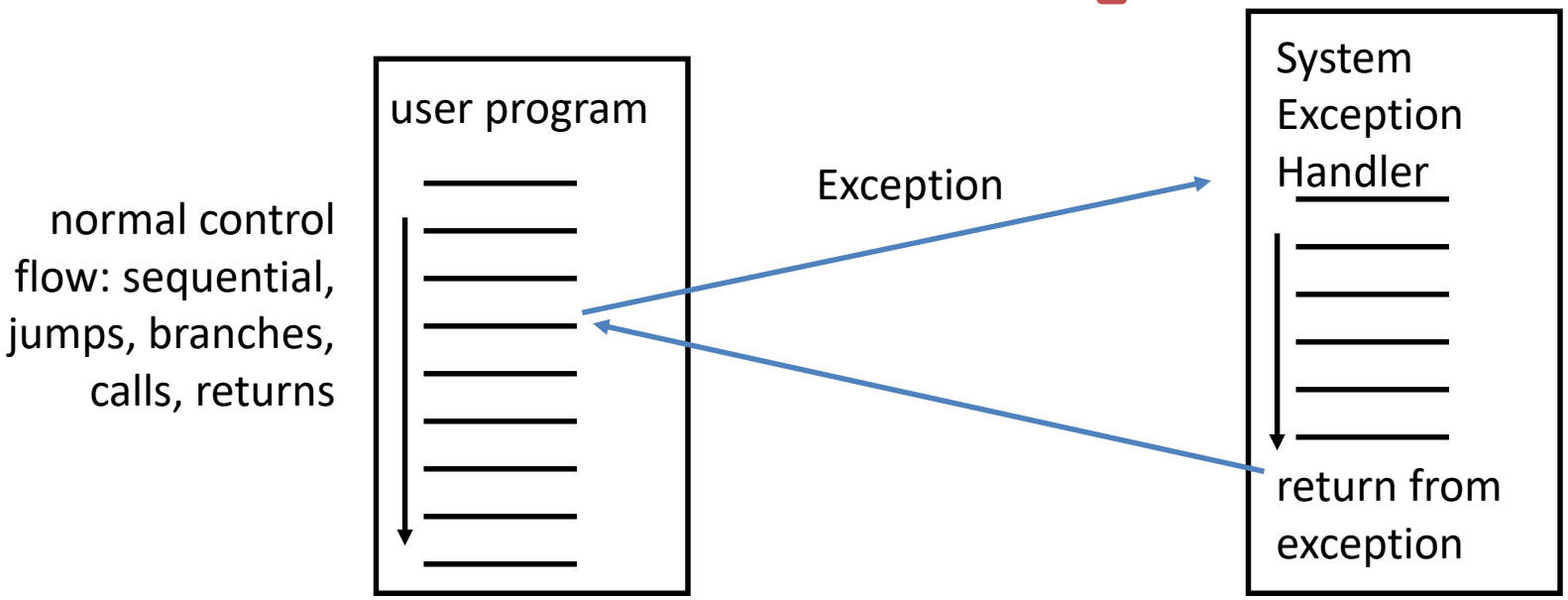
2. Il processo utente viene interrotto trasferendo il controllo a una ISR che copia il dato in memoria utente
   - ✓ All'atto della lettura il bit ready viene ri-azzerato
   - ✓ Notare che prima di effettuare il trasferimento il bit interrupt enable era stato posto a 1 per abilitare le interruzioni

Che ci si guadagna?

- Ritorniamo all'esempio di prima dell'hard disk e supponiamo che un interrupt costi 500 cicli di clock (plausibile che costi di più del polling)
- Se le interruzioni vengono generate alla frequenza di polling
  - Disk Interrupts/sec = 8 MB/s /16B
    = 500K interrupts/sec
  - Disk Polling Clocks/sec = 500K * 500
    = 250,000,000 clocks/sec
  - % Processor: $250*10^6/500*10^6$= 50%
  - Sembrerebbe che non ci sia guadagno…anzi
- Tuttavia se l'hard disk è attivo solo per il 5% del tempo, gli interrupt generati saranno il 5% e la spesa di processore sarà:
  5% * 50%=2.5%

L'overhead si paga solo quando vengono effettivamente generate richieste

Eccezioni in generale



- Eccezione = trasferimento del controllo del programma non programmato
  - Il sistema effettua delle azioni per gestire le eccezioni.
    - ✓ Ad esempio deve sapere dove registrare il punto di interruzione e dove salvare lo stato, e poi (a eccezione finita) come riprendere dal punto immediatamente successivo al punto di interruzione

Tre tipi di eccezioni

- Interrupts
  - Causate da eventi esterni (I/O)
  - Asincrone
  - Possono essere gestite nello spazio tra due istruzioni
  - Semplicemente sospendono il programma e riprendono dal punto in cui era stato interrotto
- Traps (Eccezioni)
  - Causate da eventi interni al programma
    - ✓ Condizioni eccezionali (e.g., arithmetic overflow, undefined instr.)
    - ✓ Errori (e.g., hardware malfunction, memory parity error, segmentation fault)
    - ✓ Fault (e.g., non-resident page – page fault)
  - Sincrone all'esecuzione del programma
  - Gestite da un trap handler
  - E' possibile riprovare ad eseguire l'istruzione che ha causato l'eccezione o abortire il programma
- Environment call/break
  - La Environment call (istruzione ecall) è causata da una esplicita richiesta di un servizio di sistema (e.g. Stampa di un carattere, un intero, …) attraverso esplicita ecall.
  - La Environment break (istruzione ebreak) è causata da una esplicita chiamata a ebreak per motivi diagnostici o di debug (e.g. la break nel GDB)

Gestione delle eccezioni
Exception Handler

- Esistono vari metodi per gestire le eccezioni
  - Vettore di interruzione
  - Salto diretto all'indirizzo della routine di gestione

- In entrambi gli approcci lo stato della macchina deve essere preservato
  - E.g. salvando i registry nello stack o in apposite registri

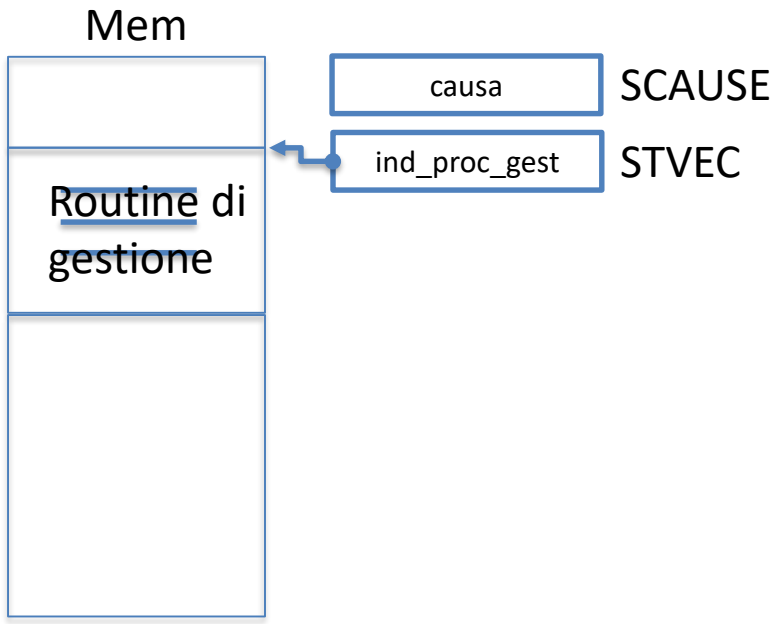
Salto diretto ad indirizzo della routine di gestione

- Salto diretto ad un indirizzo specifico:

  PC <- ind_proc_gest

  - Nel RISC-V
    - ✓ Indirizzo di gestione è contenuto nel registro speciale STVEC
    - ✓ La causa dell'eccezione è memorizzata in registro speciale SCAUSE

- Vantaggi:
  - Non è necessario fare un accesso in memoria per prelevare l'indirizzo della routine di gestione
- Svantaggi
  - Nella routine di gestione occorre analizzare la causa dell'eccezione

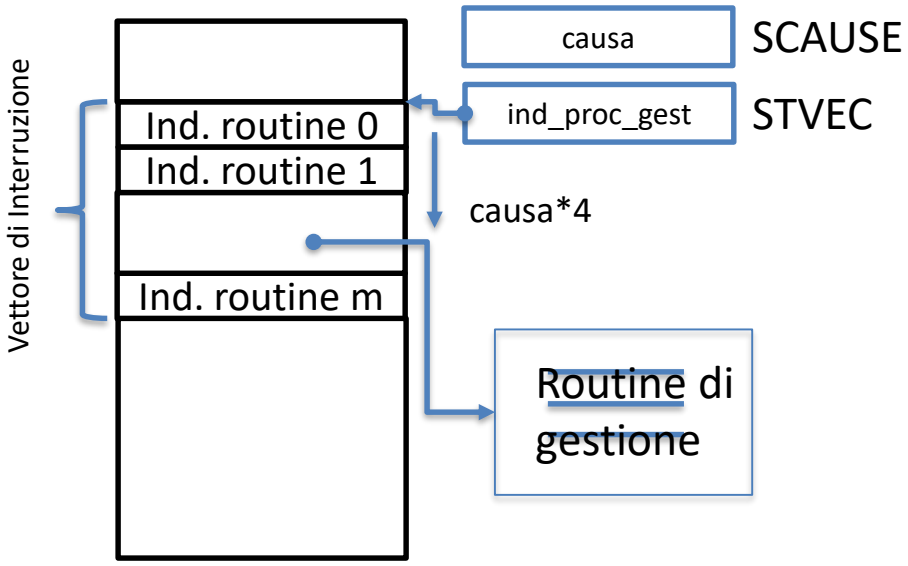


Vettore di Interruzione

- Memorizzo in una tabella gli indirizzi delle Routine di Gestione per ogni possible causa:

  PC <- Mem[base + causa*4]

- Nel RISC-V
  - Indirizzo base delle routine di gestione delle eccezioni contenuto in registro speciale STVEC
  - La causa dell'eccezione è memorizzata in registro speciale SCAUSE
- Vantaggi
  - La causa dell'eccezione è nota ed utilizzata per identificare la routine relativa di gestione
- Svantaggi
  - E' necessario accedere alla memoria per prelevare l'indirizzo della routine di gestione



Salvataggio dello stato della macchina al verificarsi di una eccezione

- Vari approcci
  - Salvataggio sullo stack
  - Salvataggio in registry ausiliari (sia visibili che non)
  - Salvataggio in registri speciali

- Nel RISC-V
  - Salvataggio sullo stack e su registry speciali
    - ✓SEPC, SCAUSE, SSTATUS, STVAL (o BadVaddr), …

Supporto alla gestione delle eccezioni nel RISC-V

- Il RISC-V ha vari registri a 64 bit
  - SEPC - contiene indirizzo dell'istruzione che ha generato l'eccezione
  - SSTATUS - contiene i bit di abilitazione globale degli interrupt
  - SCAUSE - I bit 63 e [3-0] codificano le possibili sorgenti di eccezione
    - ✓ illegal instruction = {0, 2}
    - ✓ Breakpoint = {0, 3}
    - ✓ Time interrupt = {1, 5}
    - ✓ External interrupt = {1,9}
    - ✓ …
  - STVAL – Supervisor Trap Value – contiene l'indirizzo al quale si è verificato un riferimento errato alla memoria
  - SIE – Supervisor Interrupt Enable – specifica abilitazioni più fini degli interrupt in attesa
  - SIP – Supervisor Pending Interrupt – monitoraggio degli interrupt in attesa
  - STVEC – Supervisor Trap Vector – Indirizzo base della lista dei vettori di interrupt
  - SSCRATCH – Registro per salvataggi temporanei

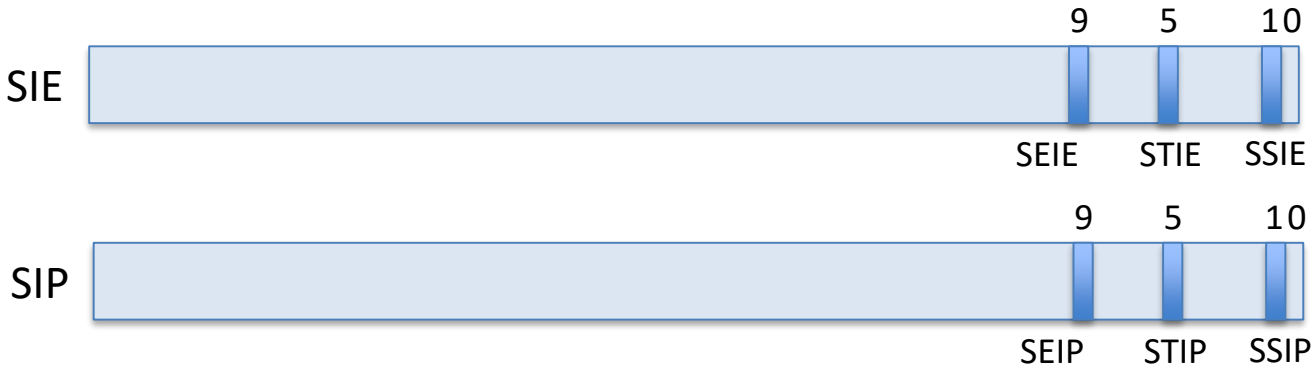
- Questi registri si trovano in un banco interno al processore chiamato Control Status Register (CSR)

- Al verificarsi di una eccezione la parte di controllo della CPU modifica questi registri
  - Note:
    - ✓ In fase di fetch il PC è aggiornato a PC+4. L'istruzione colpevole da memorizzare in SEPC è quindi il PC prima dell'incremento
    - ✓ In PC viene (sovra-)scritto direttamente il nuovo valore (PC <- STVEC)

Status Register

- Il livello di privilegio può essere S- per Supervisor o U- per User
  - Questi sono rispettivamente codificati con i valori 1 e 0
  - Chi si trova ad un certo livello non può sapere se esistono livelli di privilegio superiori (tale informazione non è accessibile)
- SIE (S- Interrupt Enable) abilita globalmente gli interrupt a quel dato livello – per non «entrare in loop» viene subito disabilitato (0)
- SPIE (S- Previous Interrupt Enable) indica lo stato precedente del bit SIE al momento in cui si verifica l'eccezione
- SPP (S- Previous Privilege mode) indica il livello di privilegio precedente al momento in cui si verifica l'eccezione

Controllo fine degli interrupt

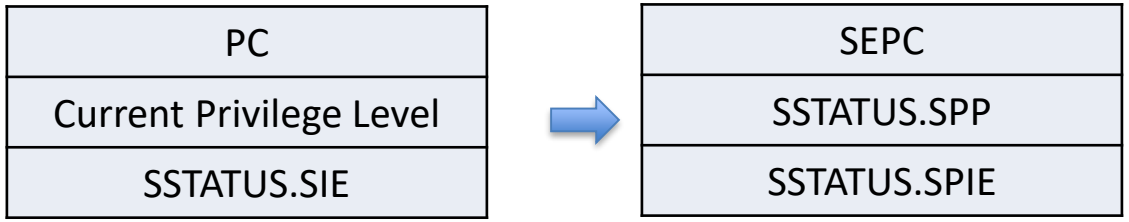


- SxIE sono bit di abilitazione più fine degli interrupt
- SxIP sono bit di indicazione di interrupt pendenti (in attesa)

- x si riferisce alla possibile sorgente dell'eccezione
  - X = E - interrupt Esterno
  - x = T - interrupt dal Timer
  - X = S - interrupt Software (ovvero eccezione)

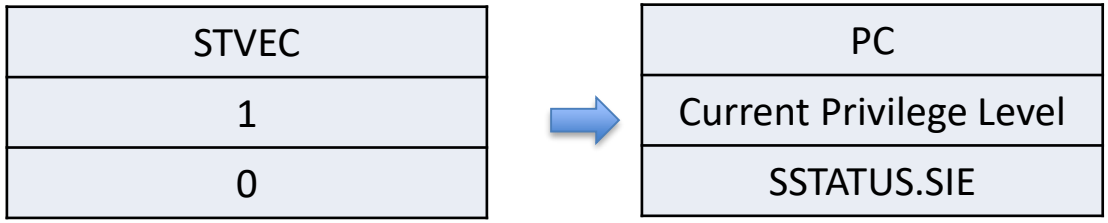
Ingresso ed Uscita in modalità Supervisor

- Al verificarsi di una eccezione



Salvataggio dello Stato della Macchine: valori correnti in rispettivi registri valori precedenti

E poi…

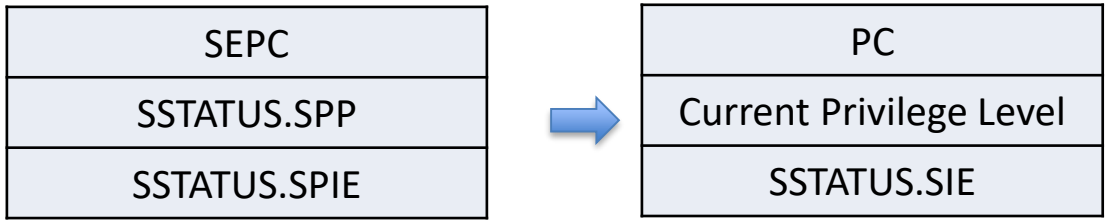


Salto alla Routine di Gestione

Modalità supervisor

Interrupt disabilitati

- All'esecuzione della SRET[1]



Ripristino dello Stato della Macchine: valori precedenti in rispettivi registri valori correnti

[1]Tipicamente si usa SRET per ritornare da una routine di gestione interrupt o eccezione

SCAUSE Register

- Int (1 bit) se vale 1, la sorgente è un interrupt, se vale 0 la sorgente è una eccezione

- Code (4 bits) codifica la ragione dell'eccezione
  - 0 - Instruction address misaligned
  - 2 - Illegal instruction
  - 3 - Breakpoint
  - 4 - Load address misaligned
  - 5 - Load address fault
  - 6 - Store address misaligned
  - 7 - Store address fault
  - 8 - Environment call from U-mode
  - 9 - Environment call from S-mode
  - C - Instruction page fault
  - D - Load page fault
  - …

Lettura/Scrittura di SEPC, SCAUSE, STVEC, STVAL, …

- Per modificare i contenuti dei registri speciali CSR:
  - CSRRW/CSRRWI CSR Read+Write (/Immediate)
  - CSRRS/CSRRSI CSR Read+Set (/Immediate)
  - CSRRC/CSRRCI CSR Read+Clear (/Immediate)

- Esempi

```
CSRRW t0,stvec,t1 # carica in t0 stvec e lo aggiorna con t1
CSRRSI t0,sie,32 # t0 -> sie e mette a 1 il bit-5 di sie (int. timer)
CSSRSI t0,sie,512 # NON possibile (l'imm. deve stare su 5 bit)
CSSRC x0,sie,t1 # (t1=29+25=544) azzera i bit 9 e 5 di sie
                # (disabilita interrupt esterni e interrupt timer)
CSRRS t0,sstatus,x0 # carica in t0 il contenuto di sstatus
                    # (caso particolare in quanto rs1=x0)
CSSRW x0,sscratch,a0 # salva a0 nel reg. speciale sscratch
```

Supporto del RISC-V per la gestione delle eccezioni

- I soli tipi di eccezioni che possono essere generate nell'implementazione della CPU analizzata sono:
  - Esecuzione di una istruzione non valida
  - Malfunzionamenti hardware
- In caso di eccezione il processore deve:
  - Salvare l'indirizzo dell'istruzione che la ha generate nel *registro causa delle eccezioni* (*SEPC - supervision exception cause* register)
  - Salvare la causa che lo ha generato nel *registro causa di supervision dell'eccezione* (*SCAUSE – supervisor cause exception* register)
  - Trasferire il controllo ad un indirizzo specifico del sistema operativo per gestire l'eccezione
  - Terminata la gestione ripristinare lo stato del processore e ritornare all'esecuzione precedente (se possibile).
- Le eccezioni sono trattate nel RISC-V come se fossero degli hazard sul controllo

Supporto del RISC-V per la gestione delle eccezioni (cont.)

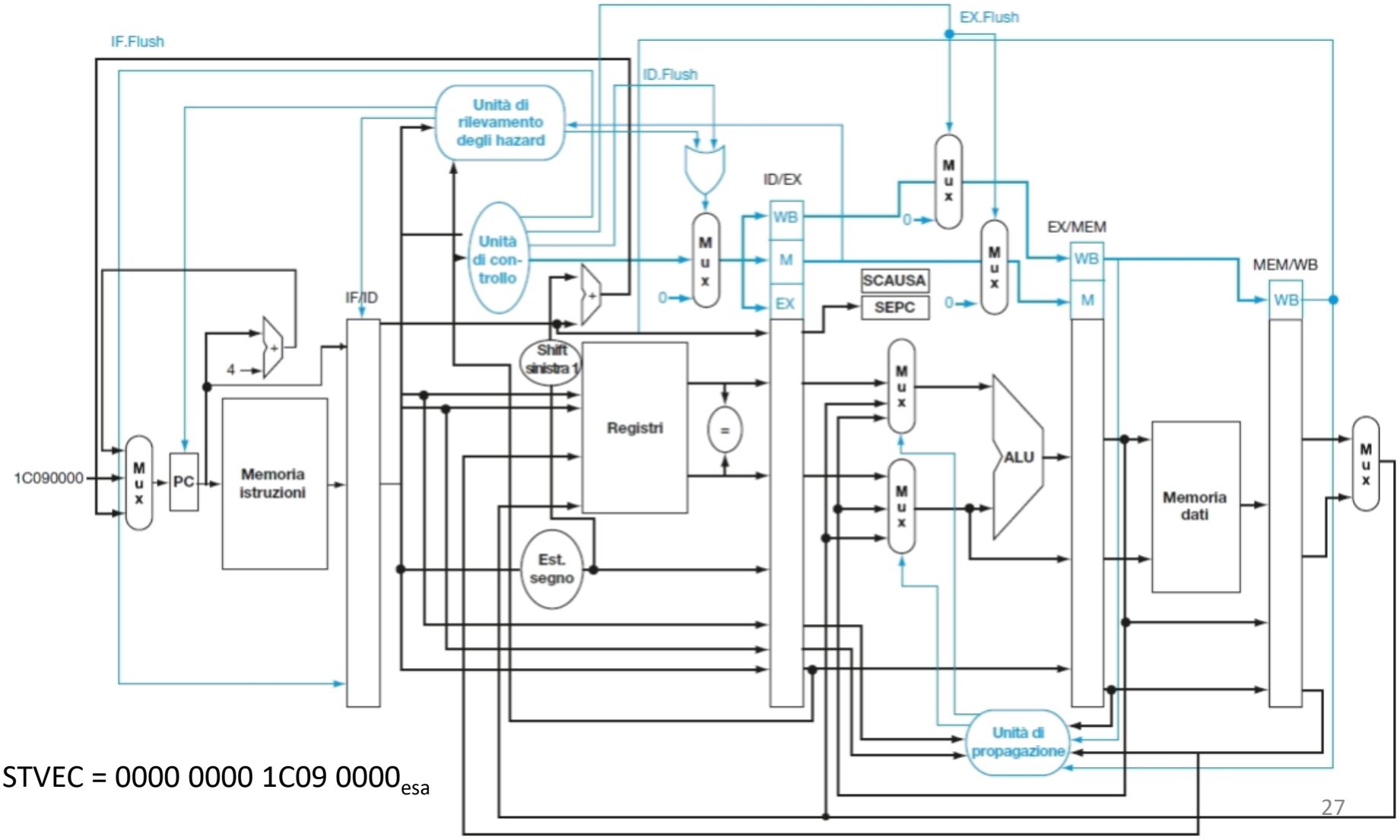
- La gestione di una eccezione in IF si usa stesso meccanismo di gestione errore di predizione per i salti convertendo operazione in una nop
- Introduciamo un nuovo segnale di controllo ID.Flush messo in OR con il segnale di stallo che proviene da unità rilevamento hazard in modo da eliminare l'istruzione nello stadio ID e realizzare così uno stallo
- Introduciamo un nuovo segnale EX.Flush che pilota un nuovo multiplexer per mettere a 0 i segnali di controllo di questo stadio
- Assumendo che l'indirizzo della prima istruzione del codice di gestione eccezioni sia $0000\ 0000\ 1C09\ 0000_{esa}$, per saltare al codice di gestione dell'eccezione occorre aggiungere una linea che porta il valore $0000\ 0000\ 1C09\ 0000_{esa}$ al multiplexer che seleziona il nuovo valore del PC.
- Salvare indirizzo istruzione che ha causato eccezione in registro SEPC

Supporto del RISC-V per la gestione delle eccezioni (cont.)

- Problema:
  - Se non si interrompe istruzione prima della fine della sua esecuzione non sarà più possibile vedere valore originale del registro x1 (verrà sporcato perchè potrebbe essere indirizzo di destinazione dell'istruzione che ha generato eccezione)
    - ✓ Se supponiamo che eccezione venga riconosciuta nello stadio EX si può utilizzare EX.Flush per prevenire che l'istruzione scriva il registro destinazione nello stadio WB

- Nota:
  - Molte eccezioni richiedono di completare normalmente l'esecuzione dell'istruzione che ha causato l'eccezione.
    - ✓ Il modo più semplice per ottenere ciò consiste nell'eliminare l'istruzione e farla eventualmente ripartire dall'inizio dopo che l'eccezione è stata gestita

Supporto del RISC-V per la gestione delle eccezioni (cont.)



STVEC = 0000 0000 1C09 0000_{esa}