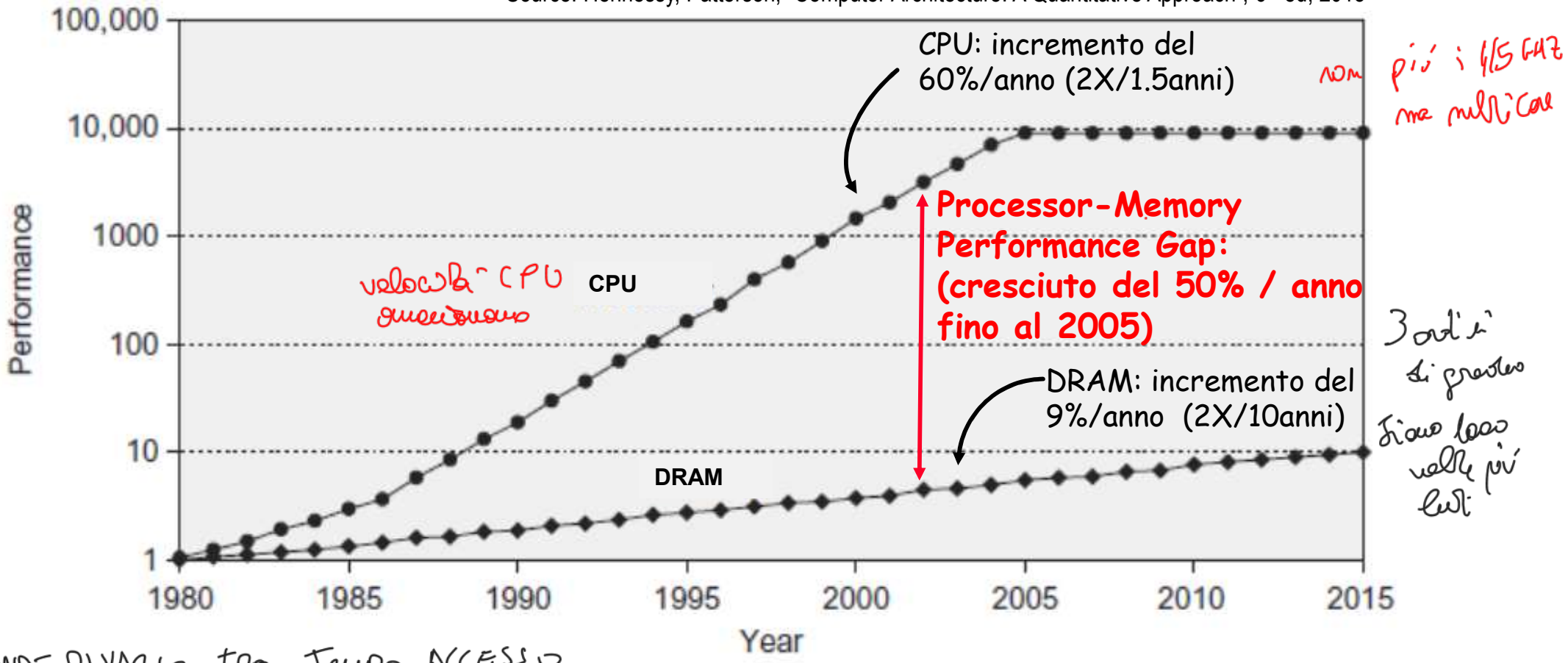
Lezione 18
Introduzione alla memoria cache

cache, funzione e tra per migliorare prestaz. macchina

Patterson: 5.3

- **Gap Prestazioni Processore-Memoria**

Source: Hennessy, Patterson, "Computer Architecture: A Quantitative Approach", 6th ed, 2018



non più i GHz ma multicore

velocità CPU aumentano

3 ordini di grandezza

fino ieri valle più lenti

GRANDE DIVARIO tra Tempo ACCESSO alla MEMORIA e Tempo con cui CPU vorrebbe OPERARE... sempre maggiore nel tempo

→ Necessità di nuove architetture per le memorie

velocità memoria non è aumentata molto, siamo nella metà più lenti

Gap Processore-Memoria: un esempio

Valutiamo il gap Processore-Memoria in termini di istruzioni per eseguite dal processore durante 1 accesso in memoria

Intel Core i7-9900k: 26 ns/0.2 ns = 130 clk x 4istr/clk → **520 istr.**

AMD Ryzen 2700X: 26ns/0.23ns = 113 clk x 4istr/clk → **452 istr.**

IBM Power8: 26ns/0.3ns = 86.7 clk x 8ist/clk → **693 istr.**

Tempo di accesso alla memoria (best-case DDR4)

Periodo di clock del processore (5GHz,4.3GHz,3.3GHz risp.)

Istruzioni eseguite in un ciclo di clock

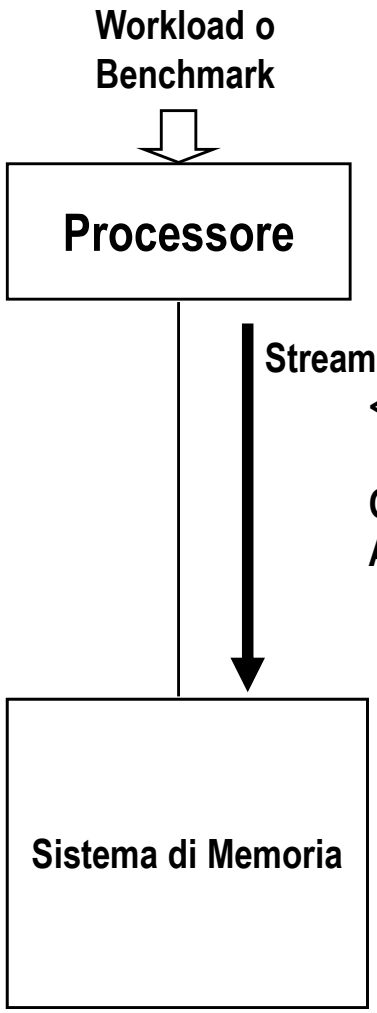
700 istruz. che possono eseguire nello stesso tempo in cui faccio una sola load

Istruzioni eseguite dal processore nel tempo in cui avviene 1 accesso in memoria

nel tempo in cui faccio un accesso alla memoria, potrei fare 500 istr.

⇒ voglio ridurre gli accessi alla memoria se non posso?

"L'Arte della Progettazione del Sistema di Memoria"



Possiamo ottimizzare il Sistema di Memoria in modo tale da minimizzare il tempo medio di accesso per i tipici programmi che utilizziamo (workload)?

Principio di Località dei Programmi

Proprietà fondamentale

Se osservo gli indirizzi accessivi

- **Un riferimento in memoria tende ad essere ripetuto dopo poco tempo**

  → **LOCALITA' TEMPORALE**

  Se osservo gli indirizzi che la CPU genera....

  - **Esempio:**
    - 0008 l'indirizzo 8 conterrà una variabile del pogram,
    - 1000
    - 0008
    - 2000 IN UN BREVE PERIODO HO ACCESSI CONTINUI ALLA STESSA LOCAZ. DI MEMORIA
    - 0008

- **Un riferimento in memoria tende ad essere a locazioni vicine a quelle usate recentemente**

  → **LOCALITA' SPAZIALE** es. Fetch del codice, le istr. successive

  - **Esempio:** = accesso ad indirizzi che sono vicini

    accesso ad istr. 4, 8 =

    - 0004
    - 0008
    - 000C
    - 0010
    - 0014

Se ho queste località, posso allora derivare una architettura diversa del sist. di MEMORIA

⇒ Se ho queste località, ho ripetizione di indirizzi, meno di mettere questi indirizzi e contenuto in una memoria più vicina al processore, CACHE

Implicazioni del Principio di Località

1) Posso utilizzare piccole memorie veloci per mantenere i riferimenti in memoria più utilizzati dal processore

(indirizzi e contenuto)

→ **MEMORIA CACHE** Piccola memoria che tiene i dati più recenti che sto utilizzando in memoria

↳ così posso fare accesso in maniera più rapida (uso SRAM)

2) Se i riferimenti utilizzati non possono essere soddisfatti nella memoria veloce

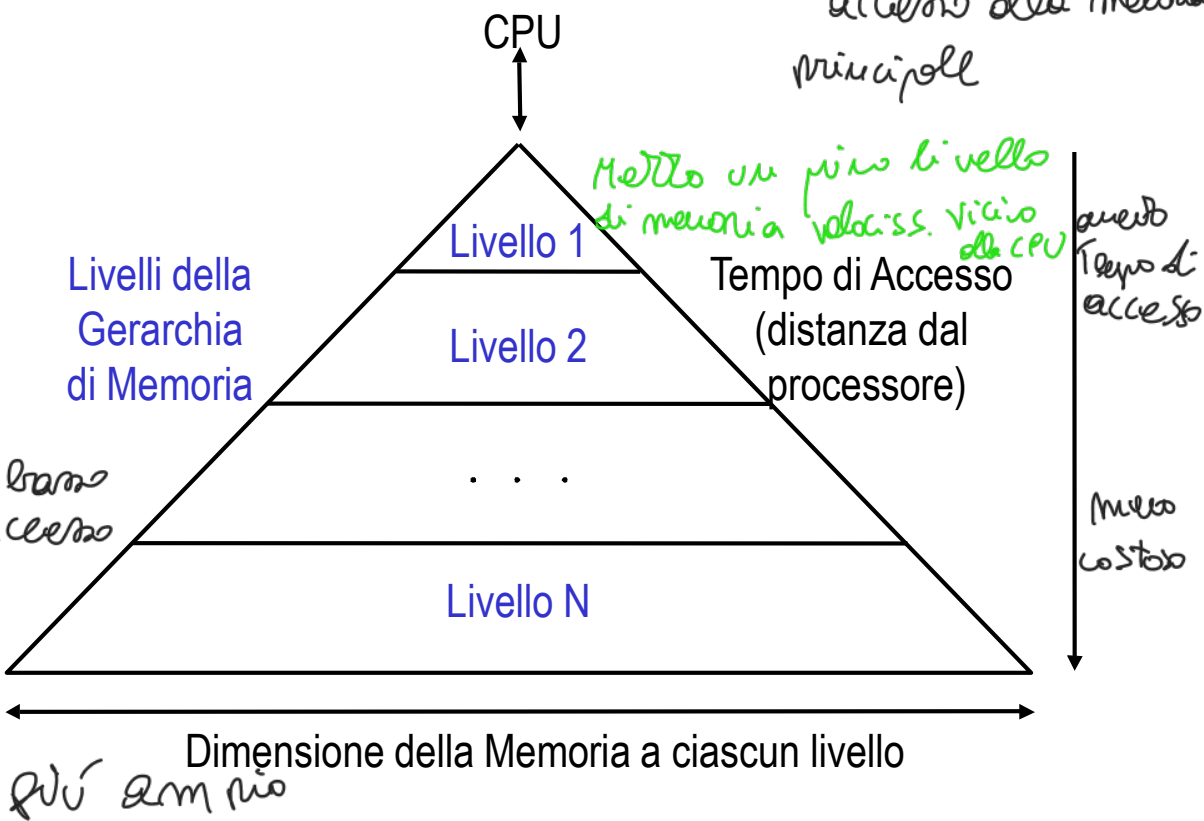
→ creo un secondo livello di memoria un po' più grande più ampio e un po' meno veloce

} Se non trovo dato nella cache, faccio accesso alla memoria principale

3) Posso reiterare questo ragionamento per N livelli di memoria

→ **GERARCHIA DI MEMORIA** avrò una

=> andando verso il basso aumento i tempi di accesso e le dimensioni



Gerarchia di Memoria

- **Presenta al processore una quantità di memoria pari a tutta quella disponibile all'ultimo livello (realizzata con la tecnologia più economica)**
- **Fornisce un tempo di accesso che si avvicina a quello del primo livello (in cui posso usare la tecnologia più veloce possibile). Es. con 2 soli livelli:**
  - **Primo Livello: Cache, tecnologia SRAM**
  - **Secondo Livello: Memoria Principale, tecnologia DRAM**
- **Matematicamente:**
  - **h= probabilità di trovare un dato nel primo livello - Maggiore LOCALITA' comporta h→1**
  - **m= probabilità di trovare un dato nel livello successivo (ovviamente h+m=1)**
  - **t_{hit}= tempo di accesso al dato dal primo livello**
  - **t_{miss}= tempo di accesso al dato nel livello successivo**

→**Average Memory Access Time (AMAT):**

AMAT = $h \times t_{hit} + m \times t_{miss}$

Se scomponiamo MissTime = HitTime + PenaltyTime

$= h \times t_{hit} + m \times (t_{hit} + t_{penalty}) = t_{hit} + m \times t_{penalty}$

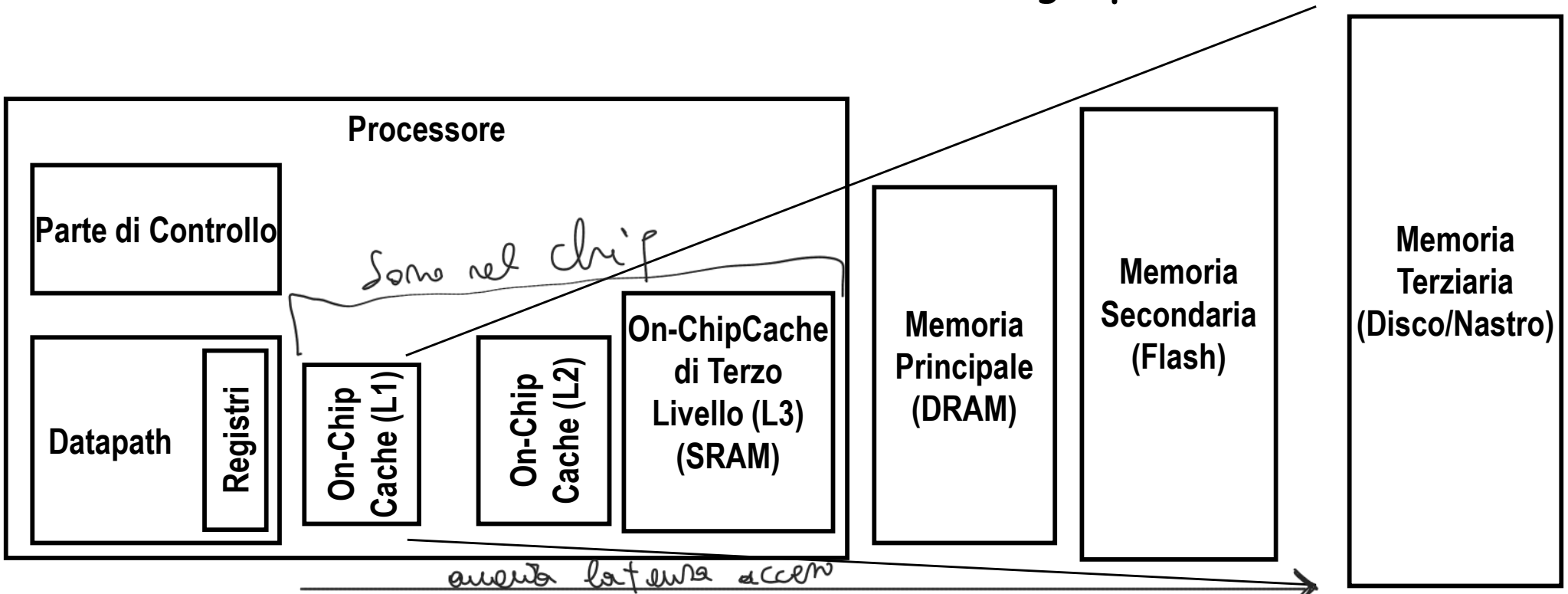
Terminologia

blocco: insieme di byte a locazioni contigue ed indirizzi multipli della dimensione del blocco
hit: il riferimento può essere soddisfatto nel livello di memoria immediatamente successivo
miss: il riferimento richiede un accesso a livelli di memoria oltre il successivo (più lenti)

↳ probabilità di miss

Organizzazione gerarchica della memoria

- Sfruttando il principio di località è possibile:
  - Presentare all'utente tutta la memoria presente al livello della tecnologia meno costosa (es. disco o nastro)
  - Fornire la velocità di accesso offerta dalla tecnologia più veloce



Velocità (ns):	0.3-1	3-10	10-20	100	10,000,000 (10 ms)	10,000,000,000 (10 s)
Capacità (bytes):	1-64K	64K-256K	0.5M-32M	16G-128G	1-4T	4-16T
Costo (EURO/GB):	40000	40000	40000	10-20	0.5	0.0001

(Anno 2020)

Memoria Cache

La cache ragiona a blocchi, cerca di catturare il working set, i blocchi di Memoria più usati dal Processore

- Scopo della cache è di memorizzare (per un rapido accesso) copie di blocchi di memoria principale
  - Funzionamento: ogni volta che faccio accesso ad una locazione di memoria, lascio in cache una copia del corrispondente blocco
  - Conseguenza: per il principio di località la cache tende a memorizzare i blocchi più usati dal processore (working set)
- Normalmente si ha:
  - B = dimensione del Blocco di cache in byte = 2^b
  - C = dimensione della Cache in byte = 2^c
  - M = dimensione della Memoria in byte = 2^m

Saranno sempre in potenze del due

$M >> C$ per ragioni di costo

ovvero

$N_M >> N_C$

N_M = Numero di blocchi della Memoria = M/B (con cui è composta)

N_C = Numero di blocchi della Cache = C/B

Inoltre, ad un certo istante per lo working set:

$W_C \subset W_M$

W_C = insieme dei blocchi del mio programma presenti in cache

W_M = insieme dei blocchi (totali) del mio programma in memoria

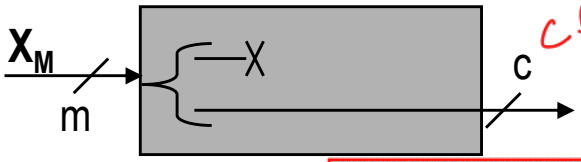
Primo Tipo di Cache

Cache ad ACCESSO DIRETTO

1) Parto dall'indirizzo specificato dal processore al blocco, X_M, prendo 3 bit meno significativi X_S per indirizzare la cache

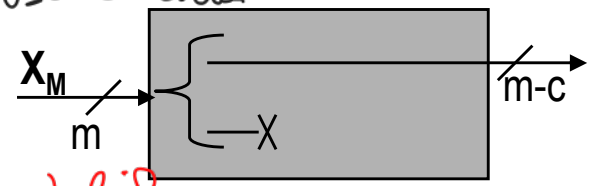
- X_M = INDIRIZZO al BLOCCO
- X_S = BIT con cui INDIRIZZO la CACHE, sottoinsieme

Prendere c bit per indirizzare la cache



Equivale a fare $X_S = X_M \bmod N_C$

Per capire quale dei 4 blocchi è stato memorizzato in cache, uso etichetta



devo memorizzare gli altri 2 BIT

prendo gli 2 bit

Equivale a fare $X_T = X_M \text{ div } N_C$

Prendo i bit

divisione rispetto al # blocchi di cache

X_S andrà a specificare lo slot di cache in cui andrà a copiare dato,

X_T servirà per confrontare se la CPU ha specificato uno dei quei 4 Termini possibili

Memoria 32 Blocchi

00000
00001
00010
00011
00100
00101
00110
00111
01000
01001
01101
10001
10101
11001
11101
11111

Esempio:
$N_C=8$, $N_M=32$
(c=3, m=5)

$\rightarrow X_S = F_S(X_M)$
$= X_M \bmod N_C$
$= X_M \bmod 8$

Cache 8 Blocchi

000 in questo slot di cache finiscono tutti i blocchi che finiscono con 000
001
010
011
100
101
110
111

in questa locazione di cache può finire uno X_T=00,01,10,11

X_{Tag} bit di questi blocchi.

resto dell'indirizzo

X_T mi servono per ritrovare il blocco

Schema logico di una cache ad accesso diretto

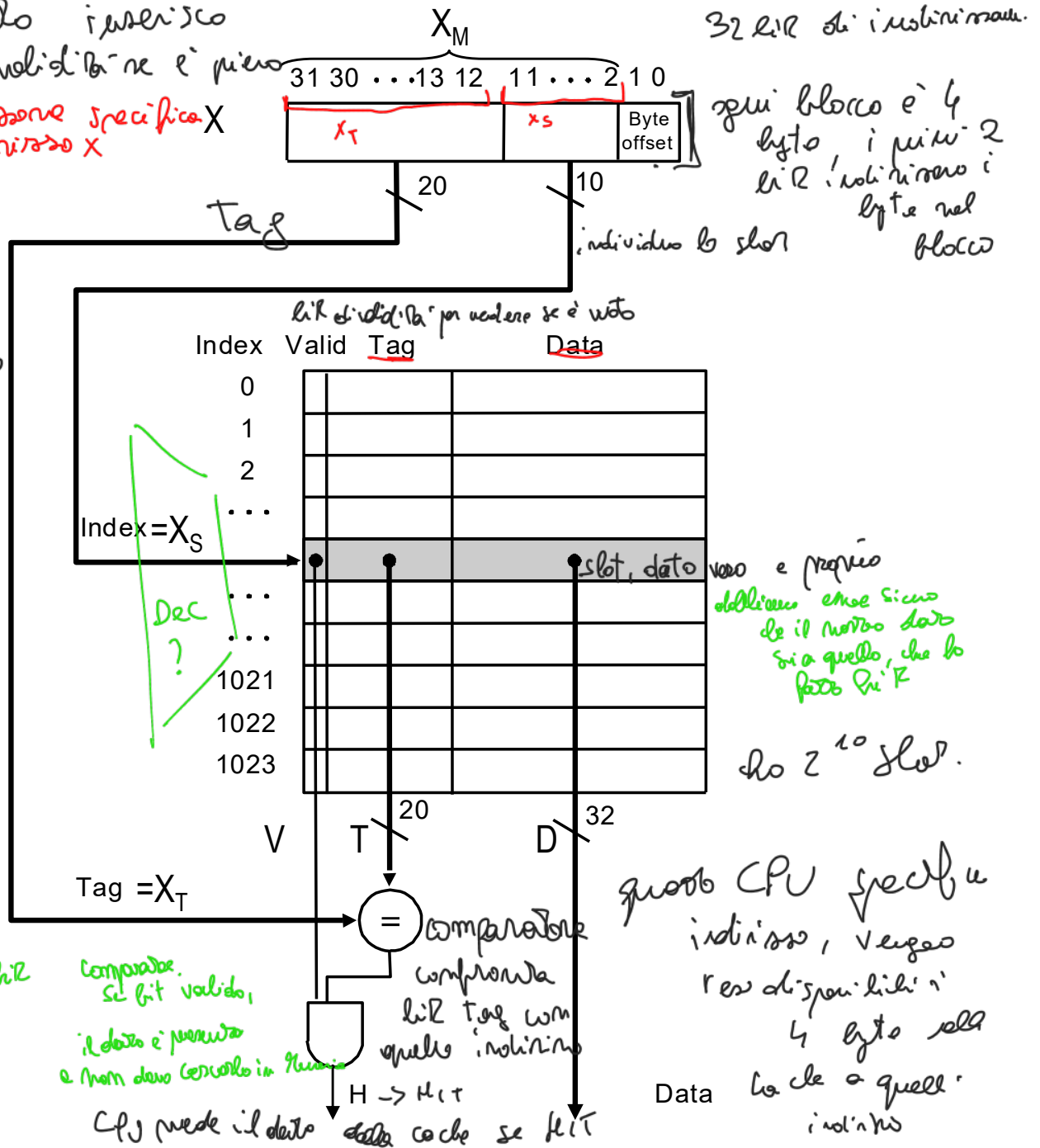
- M = 4 Gbyte
- C = 4 Kbyte
- B = 4 byte

X indirizzo al byte

$X_M = X / B = X / 2^2$

$X_S = X_M \bmod N_C = X_M \bmod C/B = X_M \bmod 2^{10}$

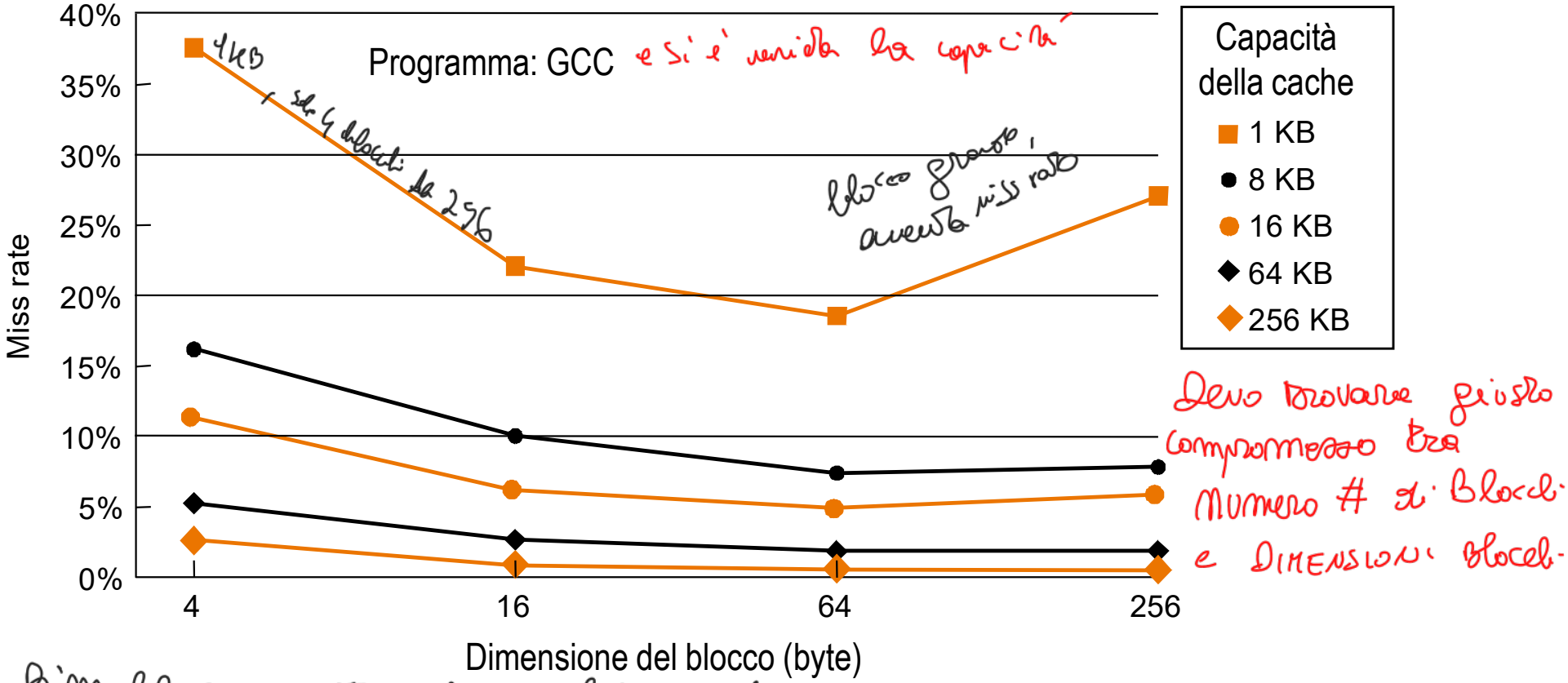
$X_T = X_M / N_C = X_M / (C/B) = X_M / 2^{10}$



Dipendenza del Miss Rate dalla dimensione del blocco

- **Aumentando la dimensione del blocco il miss rate tende a diminuire**

Meglio avere blocchi grandi o blocchi piccoli per migliorare # hit?



Se aumento dim blocco, catturo la località spaziale, diminuisce miss rate, ma devo anche aumentare il num di blocchi, altrimenti avrei solo 4 blocchi da 256 Byte con cache di 1KB

Program	Block size	Miss rate
gcc	4	5.4%
	16	1.9%
spice	4	1.2%
	16	0.4%

Conviene avere blocchi più grandi

Cache a blocchi più grandi: sfrutto la località spaziale

- M = 4 Gbyte
- C = 64 Kbyte
- B = 16 byte ogni riga deve contiene 4 word da 4 byte, 128 bit

X indirizzo al byte

$X_M = X / B$
$= X / 2^4$

X_S: 12 bit per individuare il set

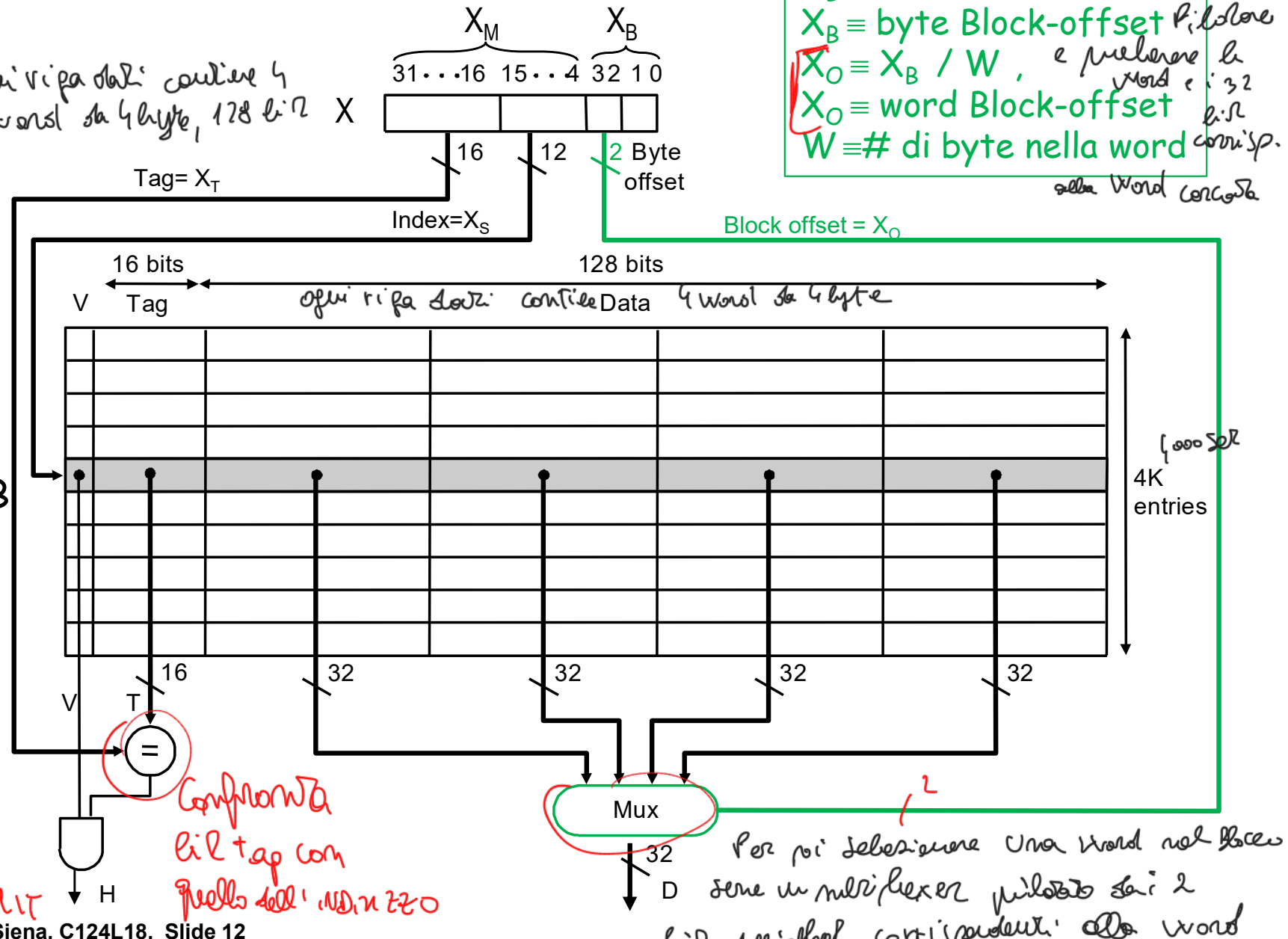
$X_S = X_M \bmod N_C$
$= X_M \bmod C/B$
$= X_M \bmod 2^{12}$

16 bit per i tag

$X_T = X_M / N_C$
$= X_M / (C/B)$
$= X_M / 2^{12}$

$X_B \equiv X \bmod B$
$X_B \equiv$ byte Block-offset
$X_O \equiv X_B / W$,
$X_O \equiv$ word Block-offset
$W \equiv$ # di byte nella word

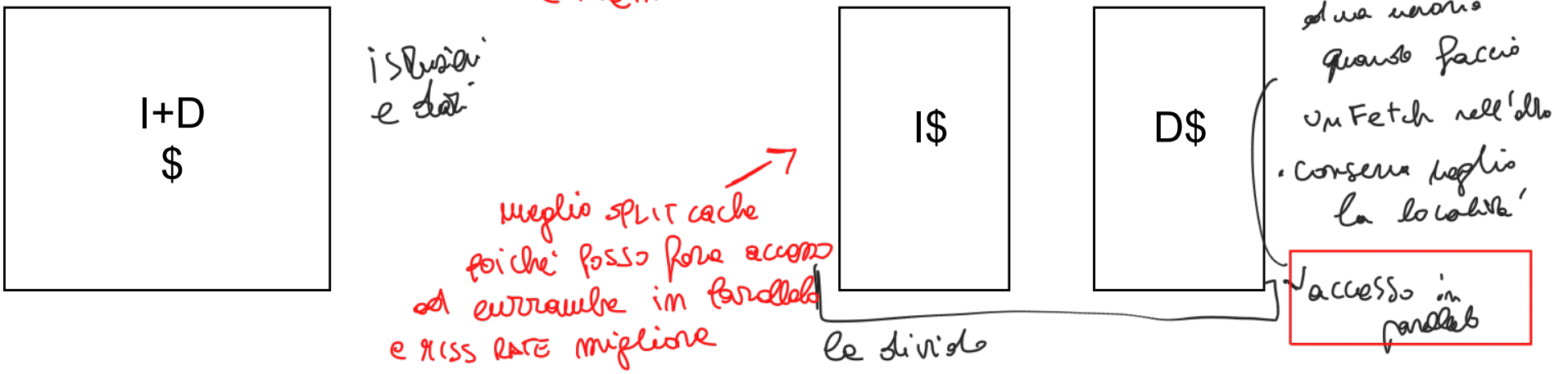
per selezionare word nel blocco per prelevare la word e i 32 bit corrisp. alla word cercata



Confronta il tag con quello dell'indirizzo

Per poi selezionare una word nel blocco serve un multiplexer pilotato dai 2 bit dell'offset corrispondenti alla word cercata

Split Cache



- La split cache consente di sfruttare maggiormente la località spaziale soprattutto nell'accesso ai dati

Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

QUANTIFICHIAMO PRESTAZIONI della CACHE

Numero medio di riferimenti per istruzione

ad ogni istruz. la cache viene interrogata

- **OGNI ISTRUZIONE FA ACCESSO ALLA MEMORIA**
  es. ADD R1, R2, R3 **genera 1 riferimento (alla memoria istruzioni)**
  LW R1, 0(R2) **genera 2 riferimenti (1 alla memoria dati + 1 alla mem. istruzioni)**

- **Indichiamo quindi con**
  R_{INST} = Average References per Instruction

$$\overline{R}_{INST} \stackrel{\Delta}{=} \frac{N_{REF}}{N_{CPU}}$$

rif. complessivi / # istruz. dinamic. eseguite

Numero Medio di riferim. per istruz.

- **$N_{REF,i}$=numero di riferimenti generati dalla istruzione i-esima**
  - $N_{REF,i1}$ =1 (es. istruz. ADD)
  - $N_{REF,i2}$ =2 (es. istruz. LW)

ogni istruzione genera RIFERIM. alla cache, cioè INDIRIZZI

$$\overline{R_{INST}} = \frac{N_{REF}}{N_{CPU}} = \frac{1}{N_{CPU}} \sum_{i=1}^{N_{CPU}} N_{REF,i}$$

- **Nell'esempio precedente:**
  - N_{REF} = numero totale di riferimenti = 3
  - N_{CPU} = 2

→$/R_{INST}$=1.5 in media faremo 1,5 riferimenti alla cache

Nota: la cache vede al suo ingresso RIFERIMENTI, non istruzioni !

Equazione delle prestazioni MODIFICATA

- **Equazione delle prestazioni:**

$$T_{CPU} = C_{CPU} \cdot T_C = N_{CPU} \cdot \overline{CPI} \cdot T_C$$

- **/CPI stavolta è un "CPI-efficace" = "CPI ideale" +**

  + numero cicli medi di stallo (penalty) per istruzione

$$T_{CPU} = N_{CPU} \cdot \left(\overline{CPI}_{ideal} + \frac{N_{REF}}{N_{CPU}} \cdot \overbrace{m}^{\bar{R}_{INST}} \cdot C_{pen} \right) \cdot T_C$$

$$= T_{CPU,ideal} + N_{REF}^{cache} \cdot m \cdot C_{pen} \cdot T_C$$

→ **Ho due modi per aumentare le prestazioni:**

- **Diminuire il miss rate**
- **Diminuire i cicli di penalty**

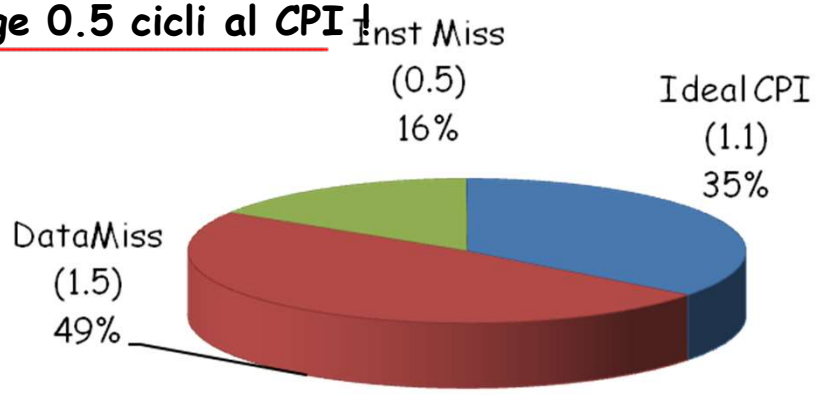
Esempio: impatto della cache sulle prestazioni

- Supponiamo di avere un processore a 200 MHz (5 ns per ciclo)
  - CPI_{ideal} = 1.1
  - 50% arit/log, 30% ld/st, 20% control
- Supponiamo che il miss-rate per le istruzioni sia l'1%
- Supponiamo che il 10% delle operazioni in memoria subisca miss con una penalty pari a a 50 cicli (di clock)
- Da (**) si ha per il CPI: $CPI = CPI_{ideal} + (m_I + N_{CPU,L/S}/N_{CPU} \times m_D) \times C_{pen}$

$$= 1.1 \text{ (cycles)} + 0.01 \text{ (miss/inst)} \times 50 \text{ (cycles/miss)}$$
$$+ 0.30 \text{ (dmop/inst)} \times 0.10 \text{ (miss/dmop)} \times 50 \text{ (cycles/miss)}$$
$$= 1.1 \text{ cycles} + 0.5 \text{ cycles} + 1.5 \text{ cycle} = 3.1$$

→Per il 65% (2.0/3.1) del tempo il processore attende la mem.!

→Un 1% di miss-rate per le istruzioni aggiunge 0.5 cicli al CPI!



dmop=data memory operation

Con cache ad accesso diretto, solo un blocco su 4 viene accettato (per es.) come posso migliorare?
↳ più indirizzi sono legati allo stesso slot

Cache Associative

ASSOCIO ALLO STESSO SLOT PIÙ BLOCCHI

Patterson: 5.4

- Sullo stesso indirizzo di cache insistono più blocchi.
  → posso fare in modo di «catturarne» più di uno

  Associo allo stesso slot

Il numero di blocchi associati ad un dato indirizzo di cache è fisso per una data cache ed è chiamato:

  dalla cache più blocc

A = "grado di Associatività" della cache o "numero di vie" della cache

A = #BLOCCHI ASSOCIATI a quello SLOT ← numero quadrini

- Detti inoltre
  - B = dimensione del Blocco di cache in byte
  - C = Capacità della cache in byte

  potenze di 2

dove:

$A \in \{1, 2, \ldots, A_{MAX}\}$ A=2, 2 blocchi in quello slot

$B = 2^b$ con $b \in \{0, 1, 2, \ldots, b_{MAX}\}$

$C = 2^c$ con $c \in \{0, 1, 2, \ldots, c_{MAX}\}$

multipli di 2

SOTTOSISTEMI MEMORIA dove CPU RAGIONANO a BLOCCHI

capacità della cache

- Sussistono le seguenti relazioni

  $N_C = C / B = 2^{c-b}$ → Numero di blocchi della Cache

  $N_M = M / B = 2^{m-b}$ → Numero di blocchi della Memoria (essendo M la dimensione della memoria in byte e $m = \log_2(M)$)

dim. memoria

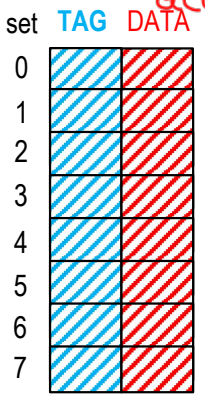
CPU può fare load e store

Cache ragiona a Blocchi

blocchi cache a 32 o 64 byte | Blocco del disco, es. 4 KB

Cache Set Associative

Cache ad 1 via (direct mapped o direct access)



accesso diretto

8 blocchi, cioè 8 set

2 blocchi possibili del set #0

Cache Set-Associativa a 2 vie (A=2)

a parità di memoria si riducono i set ma ho più flessibilità nel mapping da memoria a cache

Cache Set-Associativa a 4 vie (A=4)

4 blocchi, 2 soli set

ad ogni set sono associati 4 slot

- **Data una cache ad 1 via, con 8 blocchi, posso organizzare gli stessi blocchi in modo diverso**
- **Raggruppo i blocchi in insiemi o "set": il numero di blocchi in uno stesso set è detto A="numero di vie" o associatività della cache**

#set = prima era C/B

#set da indirizzare dall'indirizzo dato dalla CPU

$$N_S = C / (B \cdot A)$$

C/B/A

diviso per blocchi e per associatività

→ Numero di Set della cache

Avendo A, ad un set corrispondono A blocchi, #set = C/B/A

→ L'indirizzo del set sarà

bit meno significativi di X_M

$$X_S = X_M \bmod N_S$$

Prelevo set di interesse

$X_S = X_M \,\%\, \#set$

...e il tag

↳ indirizzo del blocco

bit più significativi di X_M

$$X_T = X_M \operatorname{div} N_S$$

diviso per N_S per il Tag

per trovare Tag

Tante vie aumento, meno set altro

Nota: come caso particolare queste relazioni valgono anche per le cache ad accesso diretto (A=1→N_S=N_C)

Cache Full Associative

Cache Set-Associativa a 8 vie (A=8) → Completamente Associativa (o "Full Associative")

$N_S = 1 \rightarrow X_T = X_M$ e $X_S = 0$ (cioè non occorre indirizzare il set)

- **Come individuo il blocco all'interno dello stesso set?**
  - Il problema si ha in generale in tutte le cache associative
  - Devo confrontare in parallelo il tag X_T coi tag memorizzati T_k
  - La funzione di hit sarà più complessa
    - $\forall k=1 \ldots A, X_T == T_k \;\&\&\; V_k == 1 \;?\; 1 : 0$

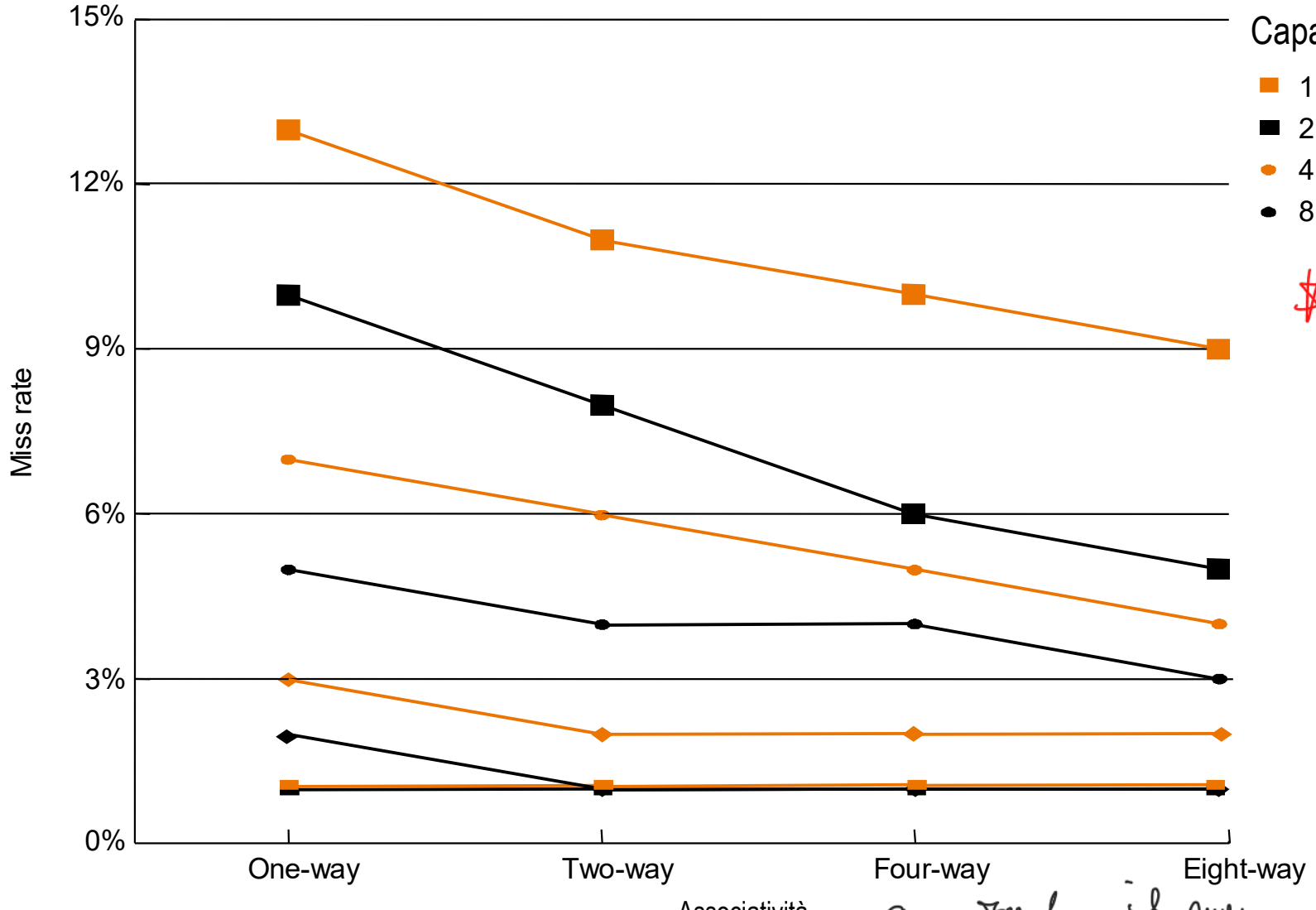
Devo controllare il Tag $\forall A$

Vale la pena fare cache associative? SI!

Prestazioni: Direct-access vs. Set-associative cache

- E' utile ?
  Sì: l'associatività tipicamente riduce il miss rate

Aumentando Cache diminuisce MISS RATE



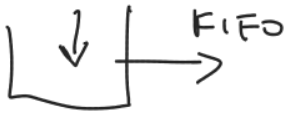
vie aumenta complicazione nella ricerca e COSTO

aumentando il num di vie riduco il miss rate

Politica di Rimpiazzamento (bits U)

- Come si procede quando un set è pieno?
  1) Scelgo il blocco da rimpiazzare secondo una certa politica
  2) Carico il nuovo blocco nella locazione liberata
  3) Leggo o scrivo in quel blocco

- Le politiche di rimpiazzamento possono essere di svariato tipo; noi consideriamo le seguenti tre:
  - **random**: scelgo un blocco a caso
    - Veloce, economica, prestazioni scarse...
  - **LRU** (Least Recently Used): scelgo il blocco usato meno recentemente
    - Ottime prestazioni (sfrutta la località temporale), costosa da implementare (con due vie è facile tenere traccia, ma per più vie l'hardware è più complesso)
    - Spesso viene usato una versione semplificata (es. Intel usa pseudo-LRU) che ha prestazioni un poco inferiori
  - **FIFO**: scelgo il blocco caricato meno recentemente
    - Non molto usata
    - Ha prestazioni intermedie fra random e LRU

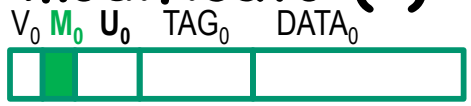


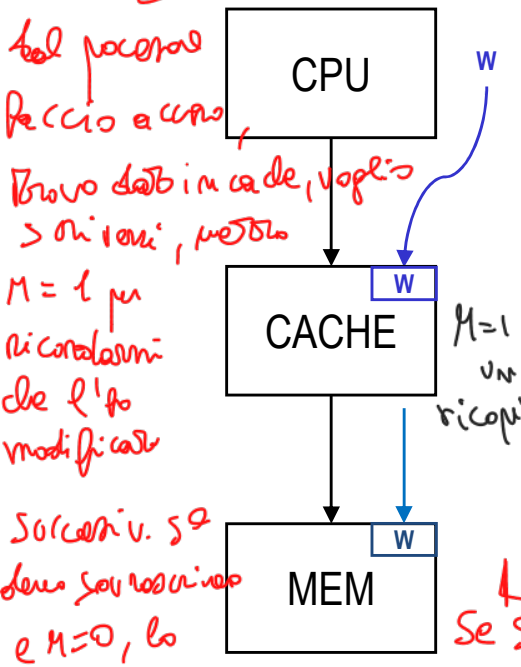
- Per gestire la politica di rimpiazzamento si aggiungono ulteriori bits (bits U) a fianco di ogni set

es. Per LRU occorrono $\lceil \log_2(A) \rceil$ bits U

Politica di gestione delle scritture su hit (bit M)

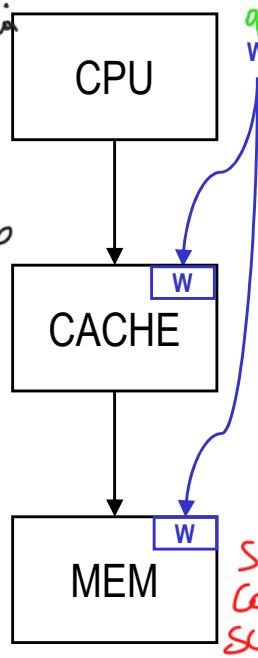
- **Un'ottimizzazione della cache consiste nell'evitare di ricopiare in memoria un blocco rimpiazzato, se NON è stato modificato (*)**
- **Ho due tecniche:**
  - **Write-Back (WB)**
  - **Write-Through (WT)**



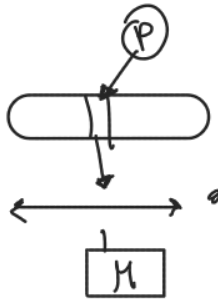


(WB1) hit
→ scrivo SOLO in cache e metto a 1 il bit M di quel blocco (inizialmente M=0)

(WB2) In caso di rimpiazzamento, il blocco va ricopiato in memoria SOLO se M=1



(WT1) hit
→ scrittura in cache e anche in memoria (in caso di rimpiazzamento non occorre ricopiare il blocco in memoria)



- **Confronto Write-Back (WB) / Write-Through (WT):**
  - **WB: occorre un bit M per ogni blocco (in WT il bit M non occorre)**
  - **+ WB: riduco il traffico sul bus di memoria (write), rispetto a WT**
  - **WB: si introduce un po' di traffico di aggiornamento della memoria**

(*)Nota: questa politica può essere usata anche in cache a 1 via

Schema architetturale di una cache 4-way set associative con politica di rimpiazzamento LRU/FIFO e write-back

- M = 4 Gbyte
- A = 4
- B = 64 byte
- C = 32 Kbyte

X indirizzo al byte

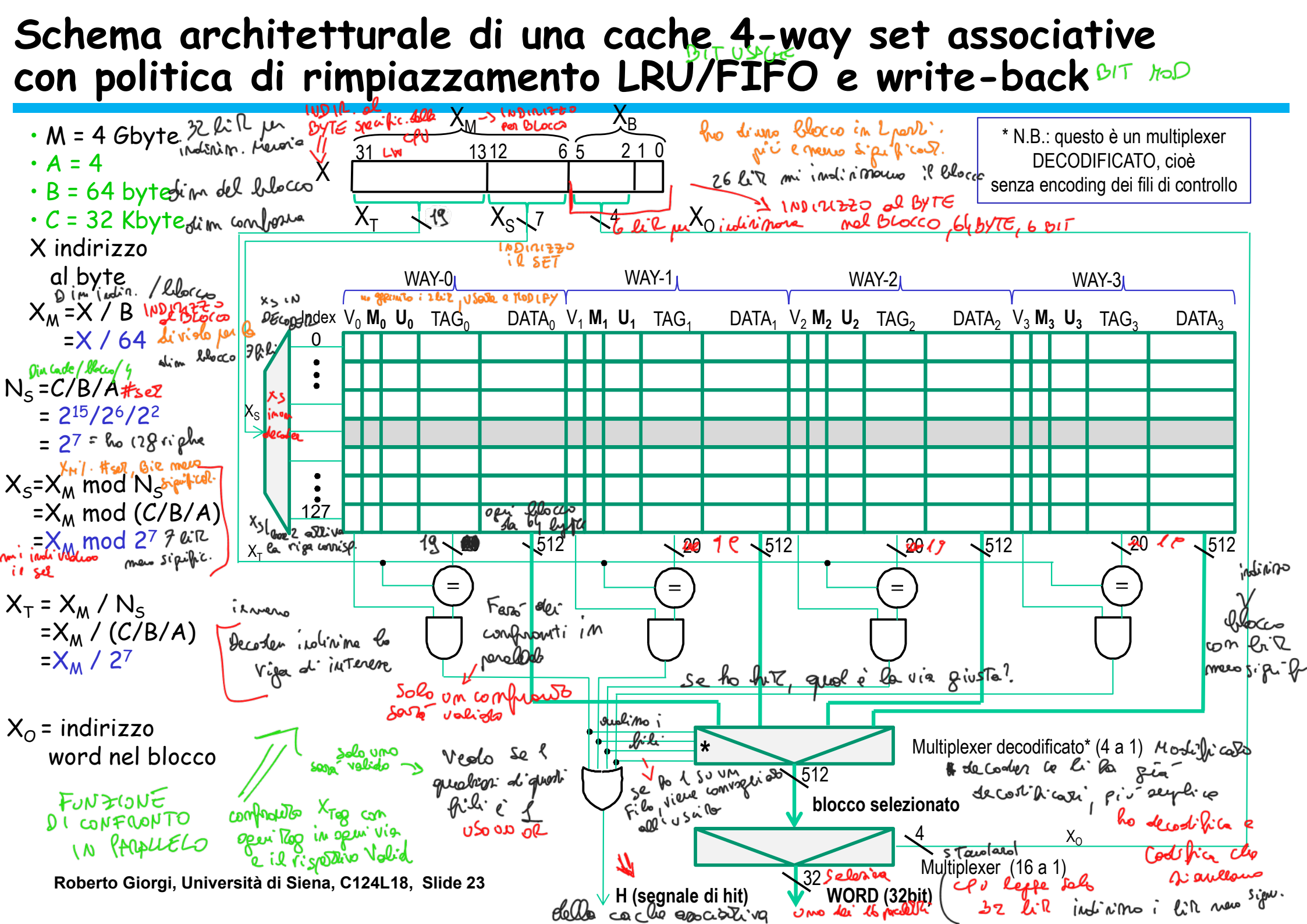
$X_M = X / B$
$= X / 64$

$N_S = C/B/A$
$= 2^{15}/2^6/2^2$
$= 2^7$ = ho 128 righe

$X_S = X_M \bmod N_S$
$= X_M \bmod (C/B/A)$
$= X_M \bmod 2^7$ 7 bit meno significativi

$X_T = X_M / N_S$
$= X_M / (C/B/A)$
$= X_M / 2^7$

X_O = indirizzo word nel blocco



Multiplexer decodificato* (4 a 1)

* N.B.: questo è un multiplexer DECODIFICATO, cioè senza encoding dei fili di controllo

FUNZIONE DI CONFRONTO IN PARALLELO

16 blocchi da 32 bit, ne esce 1 da 32 bit

↘ seleziona uno dei 16 raccolti da 16 bit

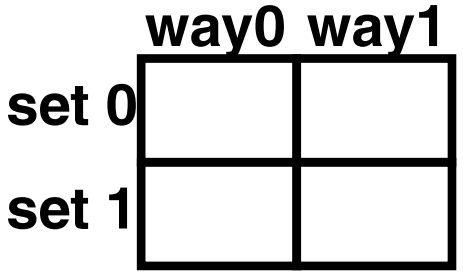
Esercizio con politica di rimpiazzamento LRU (1)

- Consideriamo una cache set-associative a due vie, con una capacità totale di 4 word e blocchi da 1 word:

A=2
B=4 (una word × 4 blocchi)
C=16 (16 byte)

$\Rightarrow N_S=2$

caratterizzano cache



lavoro con la parte destra dell'indirizzo

- Supponiamo che i riferimenti X_M (al blocco) generati dal processore siano:

R	0
W	6
R	0
R	1
W	4
R	0
R	2
R	3
W	5
W	4

lettura scrittura

es. cerca dell'indirizzo 0... trascuro R, W, Data a 6...

leg. indirizzi

Come vengono gestiti?

- Quanti hit e quanti miss saranno generati se la politica di rimpiazzamento è LRU?

come vengono gestiti i blocchi?

Esercizio con politica di rimpiazzamento LRU (2)

- Indirizzi 0, 6, 0, 1, 4, 0, ...

  X_M %2 modulo /2 divisione per 2 (N_S=2)

Primo accesso

0: (X_S=0,X_T=0) → miss, carico nel set 0 (way 0)

la cache è vuota

scelgo arbitrariamente way 0

1 via usata

6: (X_S=0,X_T=3) → miss, carico nel set 0 (way 1)

non ho tag 3

posso caricarlo, ma devo obbligatoriamente usare un solo way 1 nel set 0

0: (X_S=0,X_T=0) → hit

ho dato in cache

LS ha usato way 0, LRU = way 1

1: (X_S=1,X_T=0) → miss, carico nel set 1 (way 0)

apro su set #1

4: (X_S=0,X_T=2) → miss, carico nel set 0 (way 1, rimpiazzo il 3)

devo inserire il tag 2

scarto LRU=way 1

0: (X_S=0,X_T=0) → hit

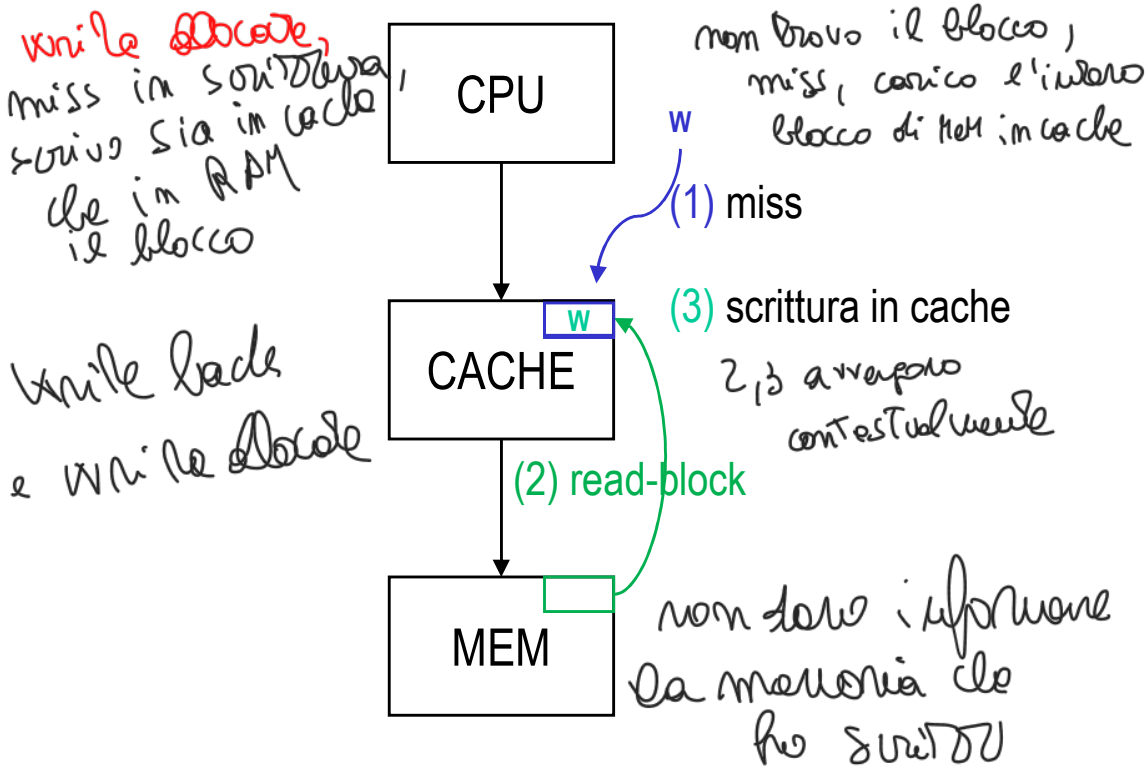
	way0	way1	
set 0	0	lru	(salvo tag 0)
set 1			
set 0	lru 0	3	
set 1			
set 0	0	lru 3	
set 1			
set 0	0	lru 3	
set 1	0	lru	copiato in Mem
set 0	lru 0	2	→3
set 1	0	lru	
set 0	0	lru 2	
set 1	0	lru	

Politica di gestione delle scritture su miss: WA/WNA

- **Cosa faccio se durante una scrittura si verifica miss?**
- **Esistono due tecniche:**

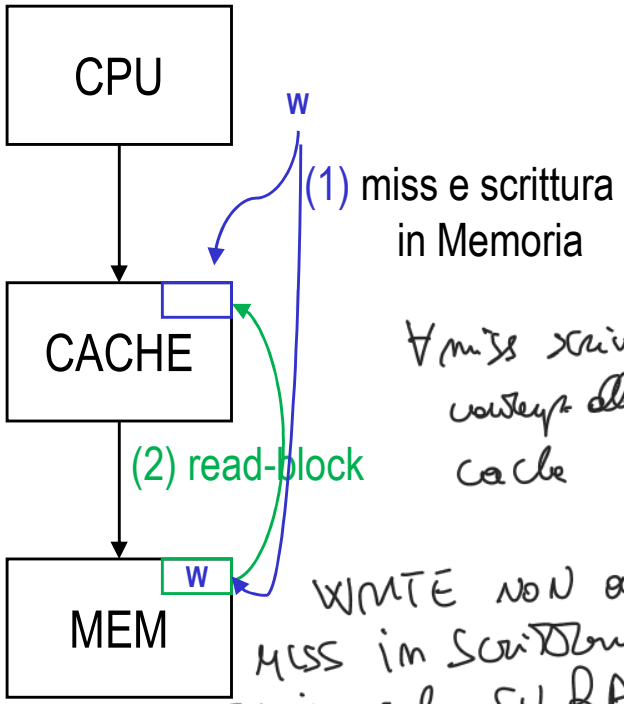
Allocate: porto in cache un blocco prima di scriverlo

- **Write Allocate (WA)**



- **Write Not-Allocate (WNA)**

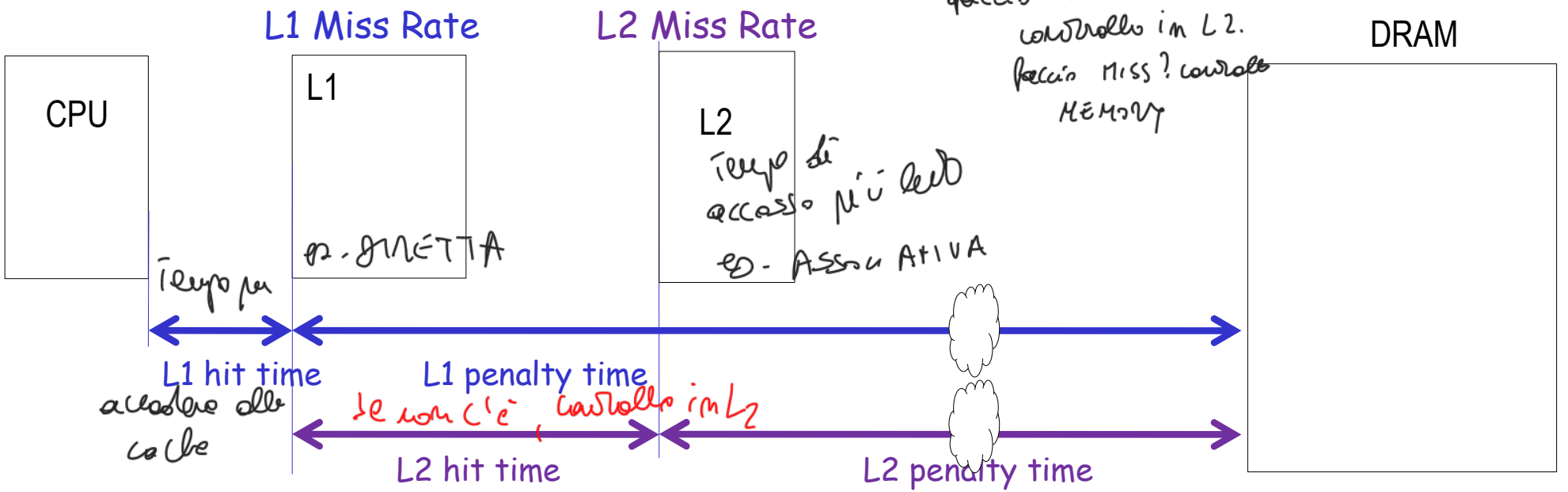
Tutto accesso miss WNA, logica prevede di indirizzo la scrittura alla cache e alla memoria



WA, NOTA: Le operazioni 2 e 3 vengono compiute come una singola azione
(si scrive nel blocco durante il suo caricamento)

WNA, NOTA: L'operazione 2, per scelta progettuale, può non essere implementata (ovvero NON si carica il blocco in cache): il beneficio effettivo di prelevare il blocco o meno può dipendere fortemente dal tipo di applicazione; noi assumeremo che venga compiuta.

Cache a 2 livelli



AMAT = L1 Hit Time + L1 Miss Rate * L1 Penalty Time

L1 Penalty Time = L2 Hit Time + L2 Miss Rate * L2 Penalty Time

⇩

AMAT = L1 Hit Time + L1 Miss Rate * (L2 Hit Time + L2 Miss Rate * L2 Penalty Time)

Valori tipici

- L1
  - Capacità: decine di KB
  - Hit-time: uno (o due) cicli di clock
  - Miss-rate tipici: 1-5%
- L2:
  - Capacità: centinaia di KB
  - Hit-time: qualche ciclo di clock
  - Miss-rate tipici: 10-20%
- I miss di L2 sono una frazione dei miss di L1
  - Perché allora il miss rate di L2 è così alto?

Cache a più livelli: esempio (confronto con 1 solo livello)

- Supponiamo di avere L1 e L2 con i seguenti dati:
  - L1 Hit-Time = 1 cc
  - L1 Miss-rate = 5%
  - L2 Hit-Time = 5 cc
  - L2 Miss-rate = 15% (% dei miss di L1 che fa miss in L2)
  - L2 Penalty-Time = 100 cicli

- Con 2 LIVELLI:
  - L1 Penalty Time = L2 Hit Time + L2 Miss Rate * L2 Penalty Time = 5 + 0.15 x 100 = 20 cc
  - AMAT = L1 Hit-Time + L1 Miss-Rate * L1 Penalty-Time = 1 + 0.05 x 20 = 2 cc

20 cicli di Penalty Time

media mente

- Con 1 solo LIVELLO:
  - L1 Hit-Time = 1 cc
  - L1 Miss-rate = 5%
  - L1 Penalty-Time = 100 cc
  - AMAT = 1 + 0.05 x 100 = 6 cc

no secondo livello

avere più livelli comporta vantaggio

→ La cache L2 migliora il tempo medio di accesso di un fattore 3

cc= clock cycles

Formule, non pretendo che le ricordiamo

Contributi separati delle istruzioni generiche e L/S

$$\mathrm{T_{CPU}} = N_{\mathrm{CPU}} \cdot \left(\overline{CPI}_{ideal} + \left(\frac{N_{\mathrm{REF,I}}}{N_{\mathrm{CPU}}} \cdot m_I + \frac{N_{\mathrm{REF,D}}}{N_{\mathrm{CPU}}} \cdot m_D \right) \cdot C_{pen} \right) \cdot \mathrm{T_C}$$

$$m_I = N_{MISS\_ISTRUZIONI} \,/\, N_{REF,I}$$
$$m_D = N_{MISS\_DATI} \,/\, N_{REF,D}$$

$$(**) = N_{\mathrm{CPU}} \cdot \left(\overline{CPI}_{ideal} + \left(m_I + \frac{N_{\mathrm{CPU,L/S}}}{N_{\mathrm{CPU}}} \cdot m_D \right) \cdot C_{pen} \right) \cdot \mathrm{T_C}$$

$$N_{\mathrm{REF,I}} = N_{\mathrm{CPU}}$$
$$N_{\mathrm{REF,D}} = N_{\mathrm{CPU,L/S}}$$

$$= \mathrm{T_{CPU,ideal}} + \left(N_{\mathrm{CPU}} \cdot m_I + N_{\mathrm{CPU,L/S}} \cdot m_D \right) \cdot C_{pen} \cdot \mathrm{T_C}$$

m= miss combinato dati+istruzioni

$$= \mathrm{T_{CPU,ideal}} + N_{\mathrm{REF}} \cdot \left(\frac{N_{\mathrm{CPU}} \cdot m_I + N_{\mathrm{CPU,L/S}} \cdot m_D}{\underbrace{N_{\mathrm{REF}}}_{= m}} \right) \cdot C_{pen} \cdot \mathrm{T_C}$$

$$m = \frac{N_{\mathrm{CPU}} \cdot m_I + N_{\mathrm{CPU,L/S}} \cdot m_D}{N_{\mathrm{REF}}}$$

$$= \mathrm{T_{CPU,ideal}} + N_{REF} \cdot m \cdot C_{pen} \cdot \mathrm{T_C} = \mathrm{T_{CPU,ideal}} + N_{CPU} \cdot m \cdot \overline{R}_{INST} \cdot C_{pen} \cdot \mathrm{T_C}$$

Nota: per una split cache gli accessi avvengono in parallelo. Quindi: $m_{I,SPLIT} \neq m_I$, $m_{D,SPLIT} \neq m_D$, $m_{SPLIT} \neq m$

Fine parte teorica