



# La gerarchia delle memorie

Prof. Alberto Borghese  
Dipartimento di Informatica  
[alberto.borghese@unimi.it](mailto:alberto.borghese@unimi.it)

Università degli Studi di Milano

Riferimento Patterson: Sezioni 5.1, 5.3



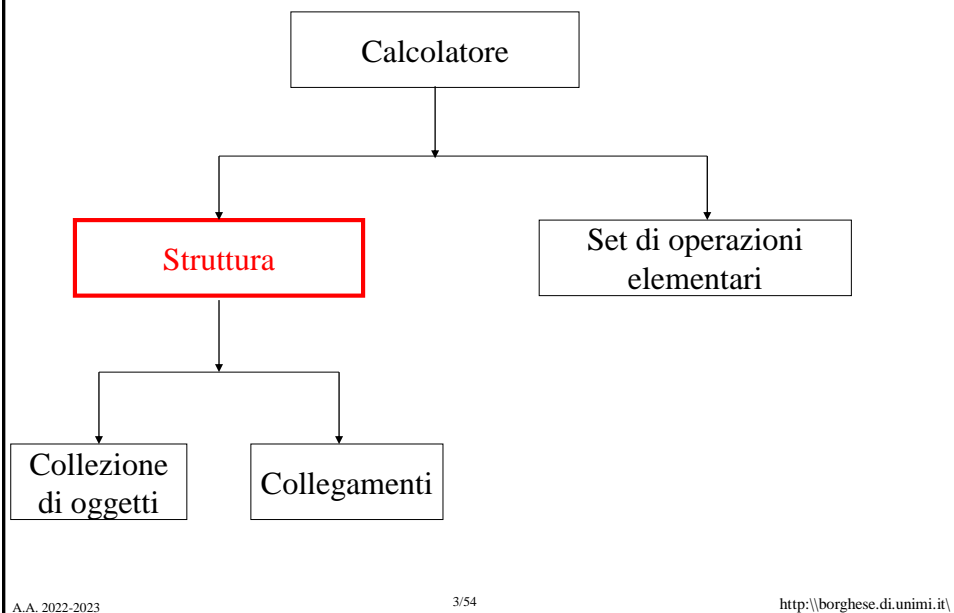
## Sommario

Struttura di un sistema di memoria

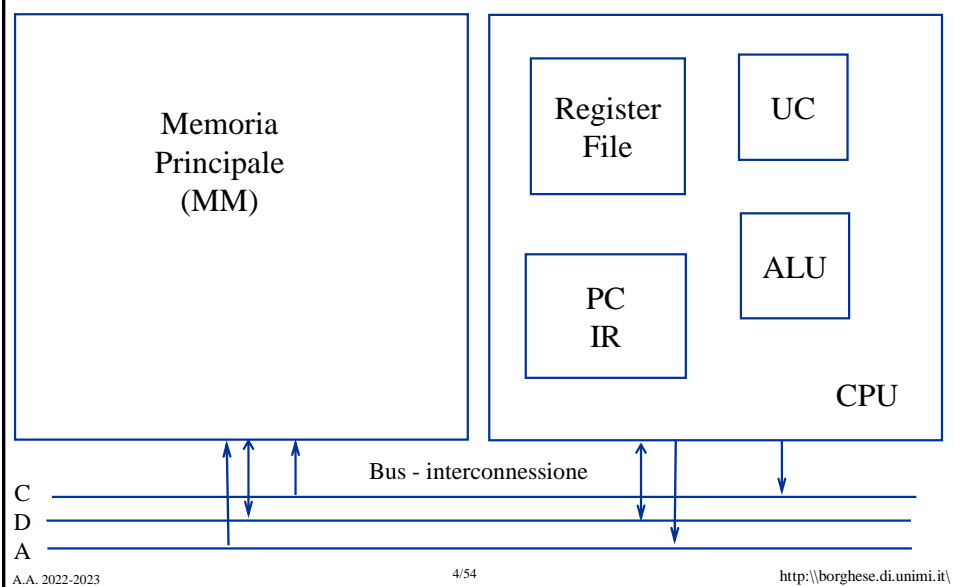
Cache a mappatura diretta



## Descrizione di un elaboratore



## Gli attori principali di un'architettura





# Memorie



- Memoria procedurale (algoritmo) => FSM
- **Memoria episodica => memoria dati**

## Memoria episodica:

- Accesso per indirizzo: indirizzo -> dato
- Accesso per chiave (memorie associative): chiave -> dato

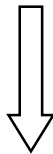


# Principio di progettazione di una memoria



Quanta memoria?  
Quanto deve essere veloce?  
Quanto deve costare?

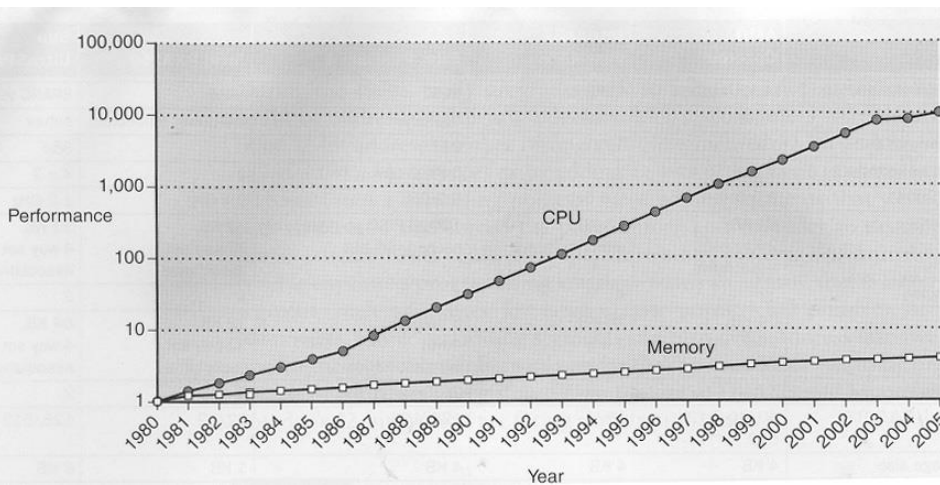
Maggiore è la velocità di accesso, maggiore il costo per bit.  
Maggiore è la capacità, minore il costo per bit.  
Maggiore è la capacità, maggiore è il tempo di accesso.



Memorie piccole e veloci.  
Memorie grandi e lente.



## Prestazioni processore vs memoria principale



Prestazioni misurate in tempo per completare un'operazione sulla memoria  
Utilizzo diretto della memoria principale da parte della CPU provocherebbe tantissimi stalli



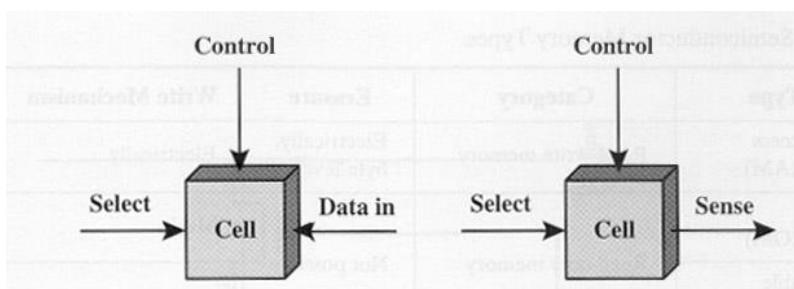
## Cella di memoria



La memoria è suddivisa in celle, ciascuna delle quali assume un valore binario stabile.

Si può scrivere il valore 0/1 in una cella.

Si può leggere il valore di ciascuna cella.



Quale struttura di memoria abbiamo già incontrato?

**Control** (abilitazione; scrittura)

**Select** (selezione celle di memoria in lettura / scrittura)

**Data** (Data in & Data out - Sense).

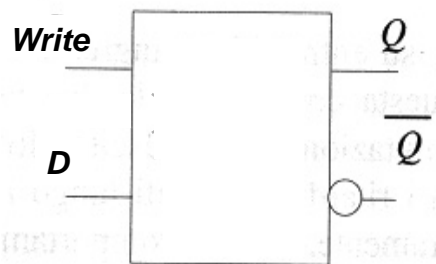
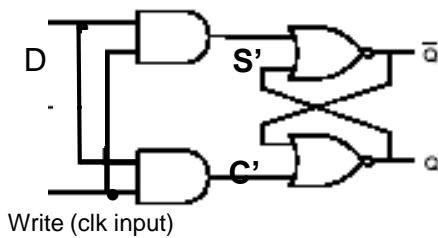


## Latch D

E' trasparente quando Write = 1

Se Write = 1  $Q_{t+1} = D$

Se Write = 0  $Q_{t+1} = Q_t$



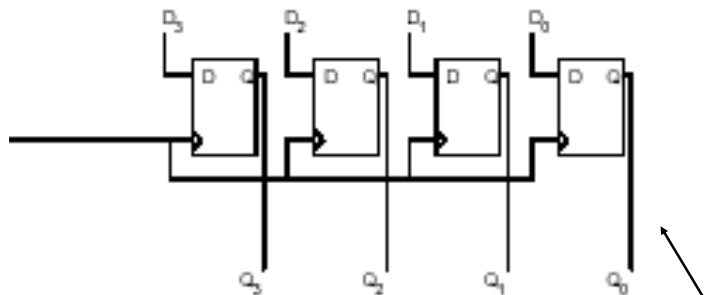
Letture - sempre disponibile in uscita

Scrittura - segnale esplicito («apertura porta di accesso al latch»)



## Registri

Insieme di bistabili che vengono scritti e letti in parallelo



Un registro a 4 bit.

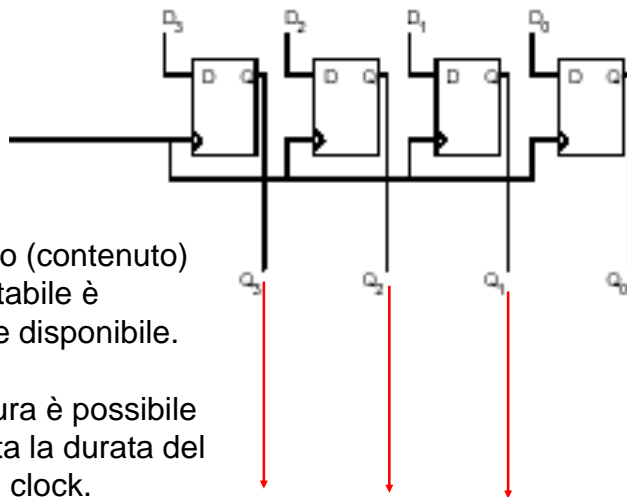
Memorizza 4 bit.

Latch di tipo D

NB Non è un registro a scorrimento (shift register!)



## Lettura di un registro



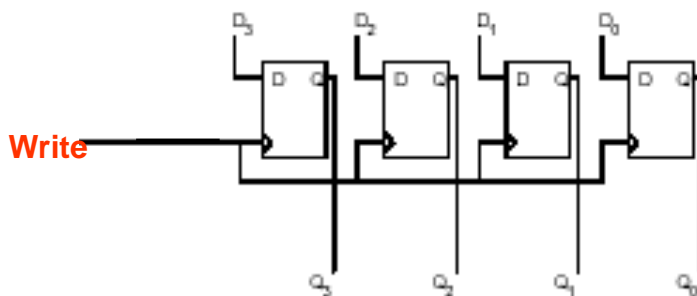
Lo stato (contenuto) del bistabile è sempre disponibile.

La lettura è possibile per tutta la durata del ciclo di clock.



## Scrittura di un registro

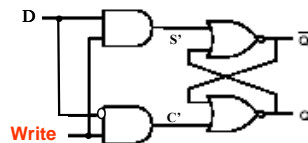
Ad ogni colpo di clock lo stato del registro assume il valore dell'ingresso dati.



Cosa occorre modificare perchè il registro venga scritto quando serve? Introdurre una sorta di "apertura del cancello (chiusura circuito)".

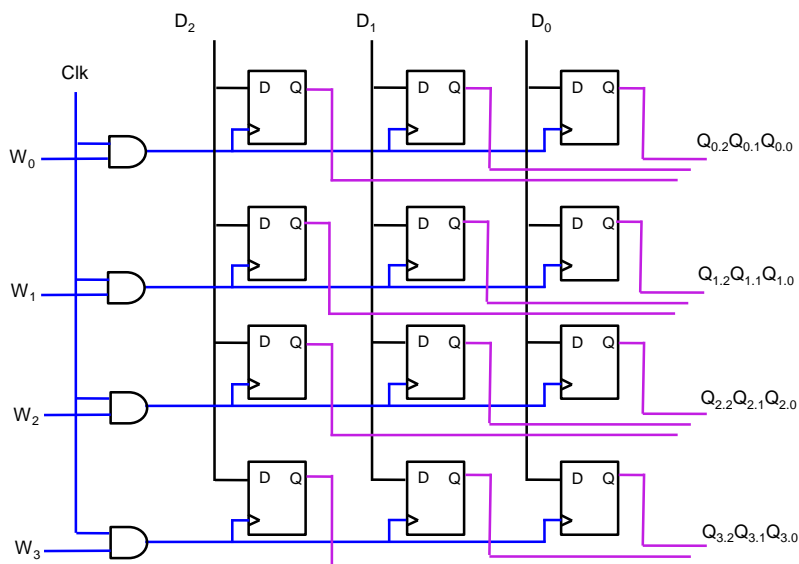
Può essere sincronizzata o meno con il clock.

Il segnale di Write apre il passaggio al contenuto di D attraverso il latch (latch trasparente). Quando il segnale di Write è a zero, lo stato non varia.





# Un banco di 4 registri x 3 bit



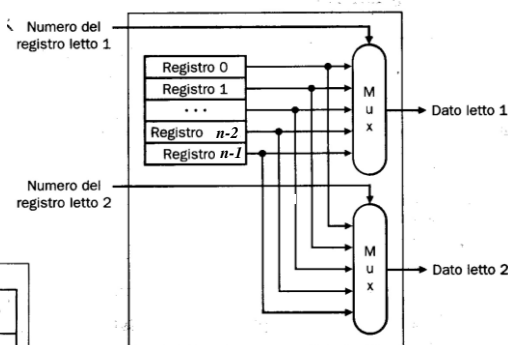
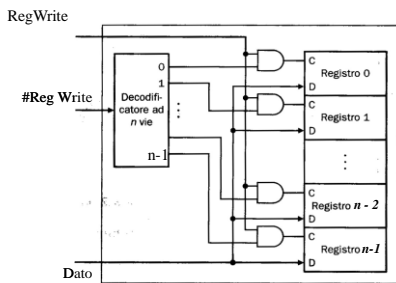
# Register file



Il tempo di lettura dipende dal cammino critico dei Mux.

Il tempo di scrittura dipende dal cammino critico del Decoder.

Numero\_registro = selettore.



Selezione - #registro

Letture - sempre disponibile in uscita (dopo tempo di commutazione del MUX)

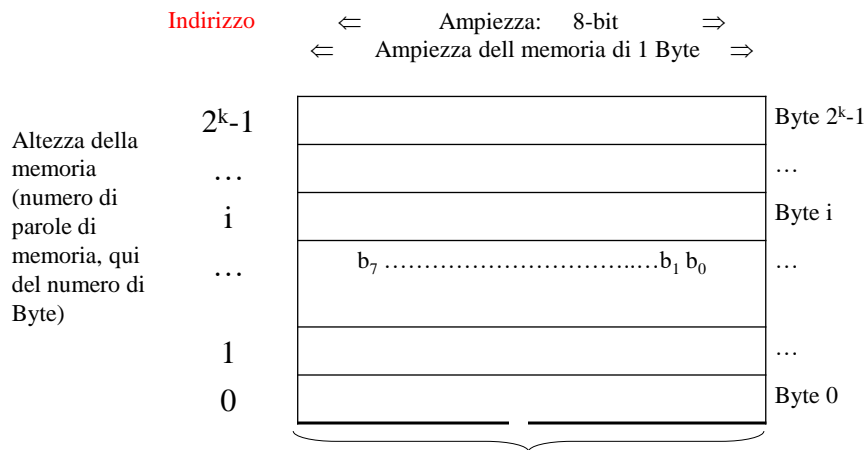
Scrittura - segnale esplicito (in AND con il clock in caso di cella sincrona).



## La memoria principale



- La memoria è vista come un unico grande **vettore uni-dimensionale** di ampiezza 1 Byte.
- Un **indirizzo di memoria** costituisce un **indice** all'interno dell'array e il numero del Byte.



## Memoria Principale



- Le memorie in cui ogni locazione può essere raggiunta in un breve e prefissato intervallo di tempo misurato a partire dall'istante in cui si specifica l'indirizzo desiderato, vengono chiamate **memorie ad accesso casuale** (*Random Access Memory – RAM*)
- Nelle RAM il *tempo di accesso alla memoria* (tempo necessario per accedere ad una parola di memoria) è *fisso e indipendente* dalla posizione della parola alla quale si vuole accedere.
- Il contenuto delle locazioni di memoria può rappresentare sia istruzioni che dati, sui quali l'architettura sta lavorando.
- La memoria può essere vista come un vettore.
- Ogni **Byte** di memoria è associata ad un **indirizzo** composto da  $n$ -bit ( $n$  *ampiezza parola*).
- I  $2^n$  indirizzi costituiscono lo *spazio di indirizzamento* del calcolatore. Ad esempio un indirizzo composto da  $32$ -bit genera uno spazio di indirizzamento di  $2^{32}$  o  $4Gbyte$ .





# Misura della capacità di una memoria



**Ampiezza della memoria.** Minimo numero di bit consecutivi che possono essere indirizzati (Memoria Principale 1 Byte; Register file: 1 word; cache: K word).

**Altezza della memoria.** Numero di elementi della memoria (Register file: 32).

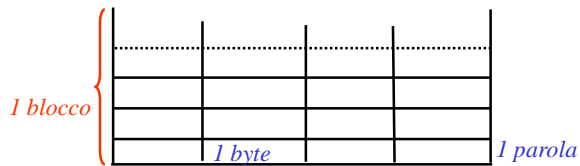
**Parola.** E' l'unità naturale in cui i dati vengono organizzati (e.g. 32/64 bit in MIPS).

**Unità indirizzabile.** E' il minimo numero di *unità contigue* indirizzabili. Nella CPU è la parola, nella memoria principale il Byte.

Bit utili di indirizzamento della memoria è:  $N = \log_2$  Capacità

**Unità di trasferimento:**

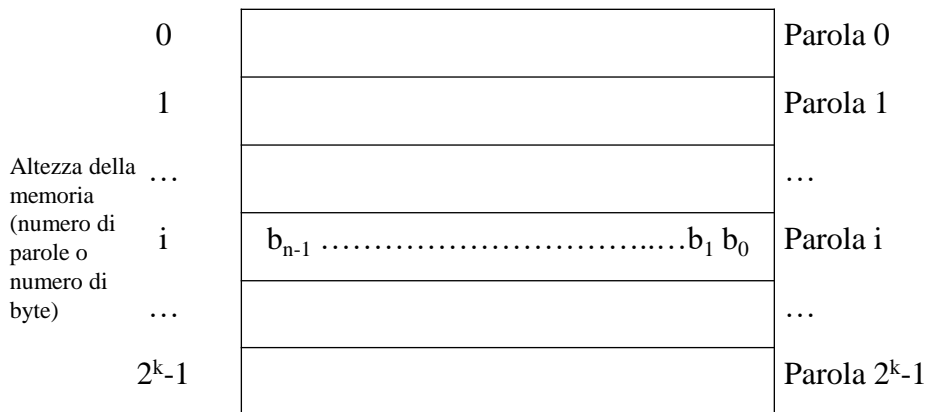
- Blocco (parole contigue)
- Parola (byte contigui)
- Byte



# Rappresentazione della memoria



Indirizzo  $\leftarrow$  n-bit  $\rightarrow$  Parola



Ampiezza della memoria (dimensione della parola)

Capacità della memoria  $C = \#\_parole \times dim\_parola$

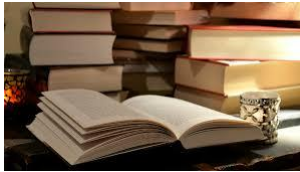


## Ricerca informazioni



Memoria principale

Quanti libri?  
Come?



Memoria di  
lavoro



Troppi!

A.A. 2022-2023

19/54

<http://\borghese.di.unimi.it/>



## Principi di località



I programmi riutilizzano dati e istruzioni che hanno usato di recente.

**Regola pratica:** un programma spende circa il **90%** del suo tempo di esecuzione per solo il **10%** del suo codice.

Basandosi sul passato recente del programma, è possibile predire con ragionevole accuratezza quali dati e istruzioni userà nel prossimo futuro.

**Località temporale:** elementi ai quali si è fatto riferimento di recente saranno utilizzati ancora nel prossimo futuro.

**Località spaziale:** elementi i cui indirizzi sono vicini, tendono ad essere referenziati in tempi molto ravvicinati.

Si possono organizzare programmi e dati in modo da sfruttare al massimo il principio di località (e.g. scrittura di blocchi di dati nei dischi, salti locali...).

**Per la memoria:** se viene richiesto un dato, e' alta la probabilità che vengano richiesti anche i dati vicini (i libri vicini).

A.A. 2022-2023

20/54

<http://\borghese.di.unimi.it/>



## Gerarchia di memorie

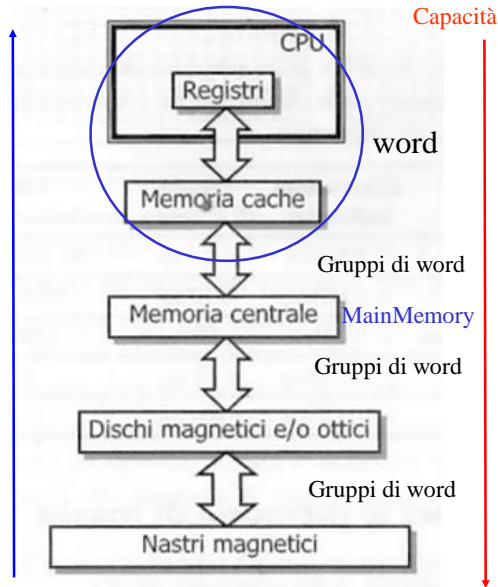


Livelli multipli di memorie con diverse dimensioni e velocità.

*Nel livello superiore troviamo un sottoinsieme dei dati del livello inferiore.*

*Ciascun livello vede il livello inferiore e viceversa.*

*Cache (memoria nascosta)*

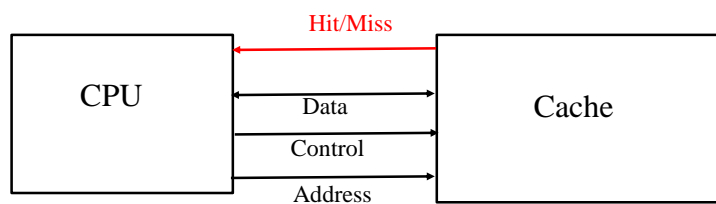


## Tassonomia del funzionamento



**HIT** Successo nel tentativo di accesso ad un dato: è presente al livello superiore della gerarchia.

**MISS** Fallimento del tentativo di accesso al livello superiore della gerarchia => il dato o l'indirizzo devono essere cercati al livello inferiore.



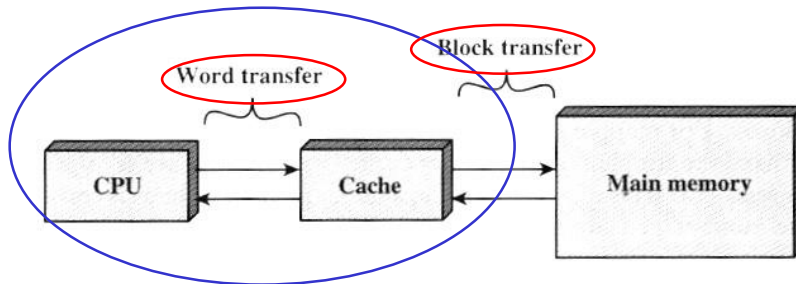


## Principio di funzionamento di una cache



**Scopo:** fornire alla CPU una velocità di trasferimento pari a quella della memoria più veloce con una capacità pari a quella della memoria più grande.

Una cache “disaccoppia” i dati utilizzati dal processore da quelli letti/scritti nella Memoria Principale.



Word transfer (dato o istruzione). In MIPS = 1 parola (facilmente più parole adiacenti, cf. multiple-issue).

Block transfer (più parole consecutive in MM, cf. organizzazione DRAM)

**La cache contiene una copia di parte del contenuto della memoria principale.**

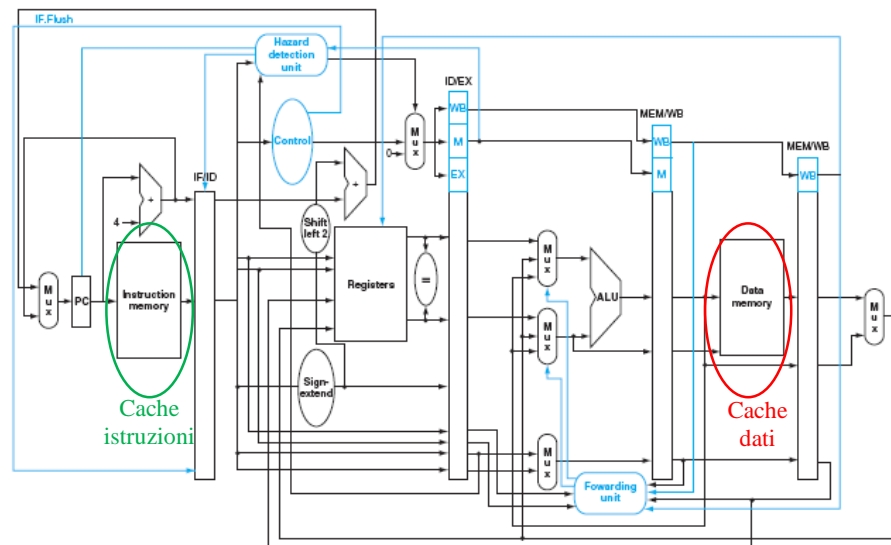
A.A. 2022-2023

23/54

<http://borghese.di.unimi.it/>



## MIPS con pipeline



A.A. 2022-2023

24/54

<http://borghese.di.unimi.it/>



# Split cache



Split-cache: Cache L-1 dati e Cache L-1 istruzioni.

**Vantaggi.** Possibilità di analizzare le istruzioni in coda (contenute nella cache istruzioni) mentre si eseguono altre istruzioni (che lavorano su dati contenuti nella cache dati), senza dovere competere per l'accesso alla cache. Efficiente per le architetture superscalari.

**Svantaggi.** Minore hit rate, perchè non si sfrutta al meglio la memoria cache. Si potrebbe riempire un'unica cache maggiormente con dati od istruzioni a seconda del frammento di codice correntemente in esecuzione.

Il register spilling e le miss sono inevitabili →  
Come fare vedere al processore una memoria sufficientemente veloce?



# Cache di un ARM Cortex-A8 e Intel Core i7



Caratteristica	Cortex-A8 ARM	Core i7 Intel
Organizzazione cache L1	Cache separate per dati e istruzioni	Cache separate per dati e istruzioni
Dimensione cache L1	32KiB per istruzioni/dati	32KiB per istruzioni/dati per ogni core
Grado associatività cache L1	Set-associativa a 4 vie (I), a 4 vie (D)	Set-associativa a 4 vie (I), a 8 vie (D)
Modalità sostituzione L1	Random	LRU approssimato
Dimensione blocco L1	64 byte	64 byte
Politica scrittura L1	Write-back, Write-allocate (?)	Write-back, Write-allocate
Tempo hit L1 (utilizzo load)	1 ciclo di clock	4 cicli di clock (pipeline)
Organizzazione cache L2	Un'unica cache per istruzioni e dati	Un'unica cache per istruzioni e dati
Dimensione cache L2	Da 128 KiB a 1 MiB	256 KiB (0,25 MiB)
Grado associatività cache L2	Set-associativa a 8 vie	Set-associativa a 8 vie
Modalità sostituzione L2	Random (?)	LRU approssimato
Dimensione blocco L2	64 byte	64 byte
Politica scrittura L2	Write-back, Write-allocate (?)	Write-back, Write-allocate
Tempo hit L2	11 cicli di clock	10 cicli di clock
Organizzazione cache L3	-	Un'unica cache per istruzioni e dati
Dimensione cache L3	-	8 MiB, condivisa dai core
Grado associatività cache L3	-	Set-associativa a 16 vie
Modalità sostituzione L3	-	LRU approssimato
Dimensione blocco L3	-	64 byte
Politica scrittura L3	-	Write-back, Write-allocate
Tempo hit L3	-	35 cicli di clock



# Sommario

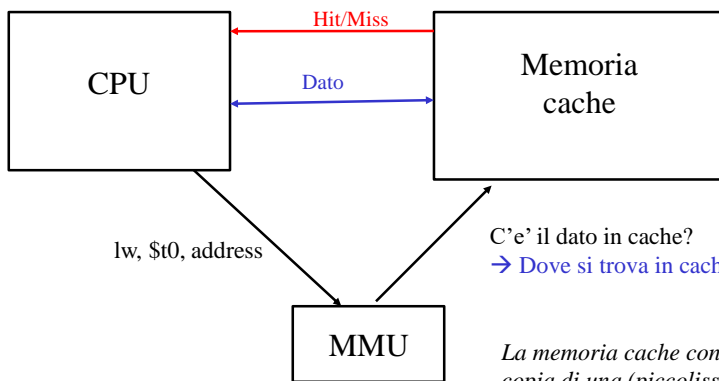


Struttura di un sistema di memoria memoria

Cache a mappatura diretta



# Le domande alla memoria cache



C'e' il dato in cache?  
→ Dove si trova in cache?

*La memoria cache contiene una copia di una (piccolissima) parte dei dati della memoria principale.*



## Divisione in modulo



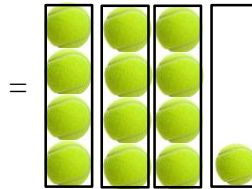
$$13 : 4 = 3 \text{ con resto } 1$$

Significato fisico:



Riempio **completamente** 3 scatole da 4 palline (il quoziente: i moduli completati).

L'ultima scatola contiene 1 sola pallina (il resto della divisione).



## Divisione in modulo per una potenza della base



$$31.879 : 100 = 318 \text{ con resto } 79$$

E' una divisione intera. Quando si divide per una potenza della base si estraggono le cifre corrispondenti.

$$\begin{array}{c} 31.879 : 100 \\ \uparrow \uparrow \uparrow \\ 10^2 \end{array}$$

E' equivalente a un'operazione di shift a destra di 2 posizioni. Separo quoziente e resto.

L'operazione di shift può essere vista come l'estrazione di un gruppo di cifre (cf. shift nella CPU per beq e jump):



# Divisione in modulo binaria



$$13 : 4 = 3 \text{ con resto } 1$$

$$\begin{array}{r} 11 \\ \uparrow \uparrow \\ 1101 : 100 = 11 \text{ con resto } 1 \end{array}$$

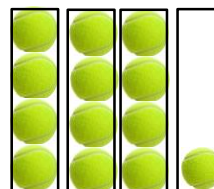
2<sup>2</sup> Shift a dx di 2 posizioni  
Separo quoziente e resto

Significato "fisico":

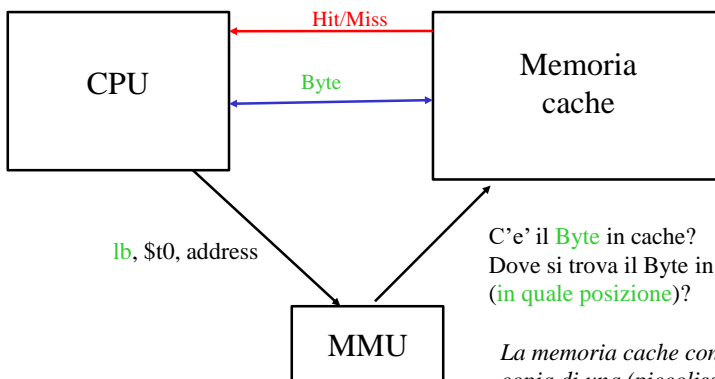


Riempio **completamente** 3 scatole da 4 palline (il quoziente: i moduli completati).

L'ultima scatola contiene 1 sola pallina (il resto della divisione).



# Le domande alla memoria cache



C'e' il **Byte** in cache?  
Dove si trova il Byte in cache  
(in quale **posizione**)?

*La memoria cache contiene una copia di una (piccolissima) parte dei dati della memoria principale.*



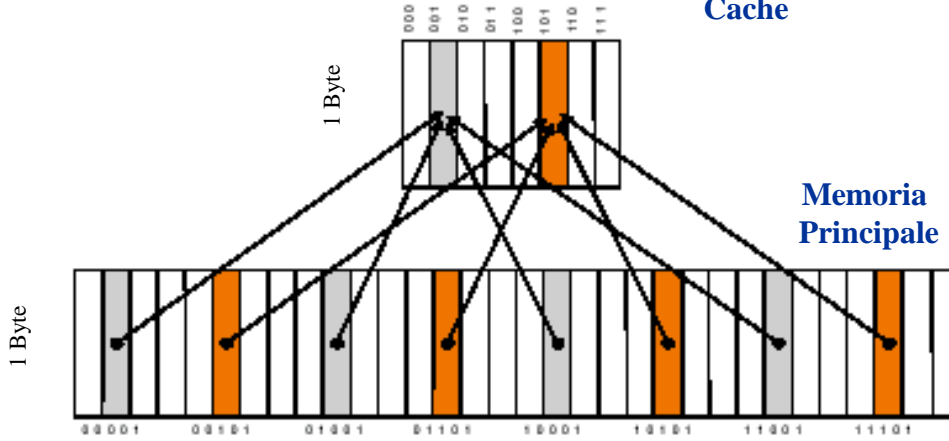


## Corrispondenza diretta (direct mapped)



Cache di 8 Byte (indirizzi su 3 bit)

Memoria  
Cache



Memoria  
Principale

MM di 32 Byte (indirizzi su 5 bit)



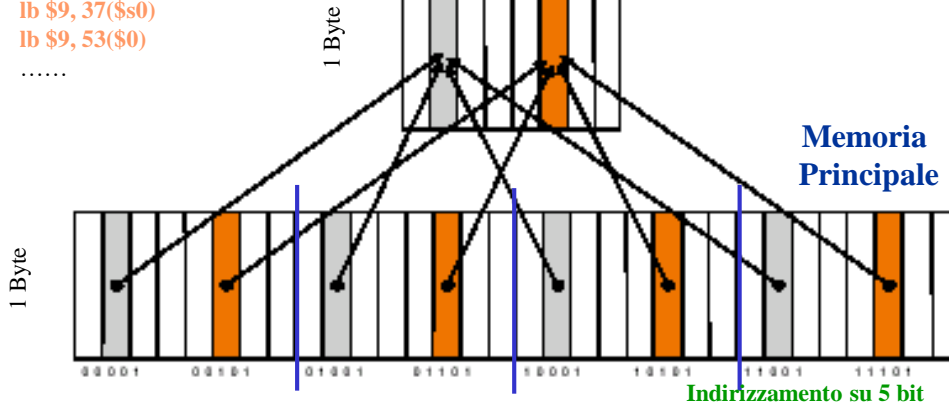
## Corrispondenza diretta (direct mapped)



Ad ogni indirizzo di Memoria Principale corrisponde un indirizzo di cache.

- lb \$9, 69(\$0)
- lb \$9, 5(\$s0)
- lb \$9, 37(\$s0)
- lb \$9, 53(\$0)
- .....

Cache di 8 Byte  
Indirizzamento su 3 bit



Memoria  
Principale

Indirizzamento su 5 bit

Indirizzi diversi di Memoria Principale corrispondono allo stesso indirizzo di cache (la corrispondenza non è biunivoca).

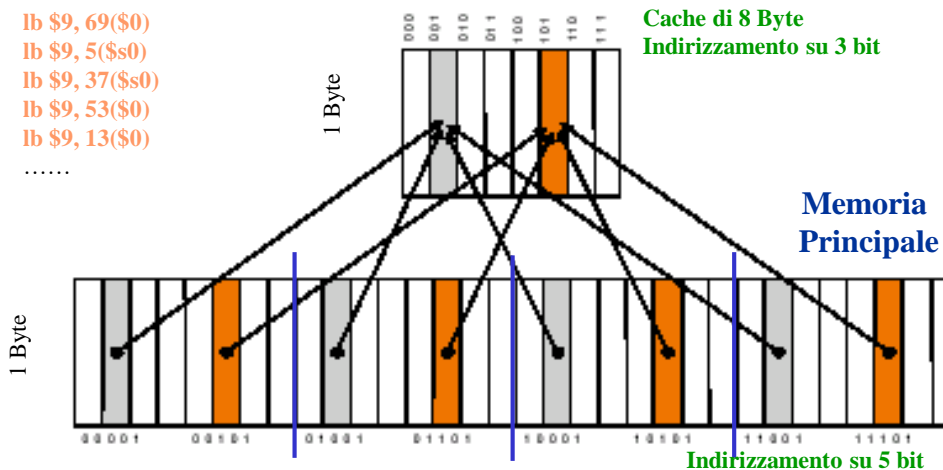


## Il problema



Ad ogni indirizzo di Memoria Principale corrisponde un indirizzo di cache.  
A ogni indirizzo di cache corrispondono diversi (tanti) indirizzi di MM

lb \$9, 69(\$0)  
lb \$9, 5(\$s0)  
lb \$9, 37(\$s0)  
lb \$9, 53(\$0)  
lb \$9, 13(\$0)  
.....



Devo trovare una funzione (mappatura) che mette in corrispondenza 1 indirizzo della memoria principale con 1 indirizzo di cache (molti a uno)



## Divisione in modulo



lb \$9, 13(\$0)

$$13 : 8 = 1 \text{ con resto } 5$$

$$9 : 4 = 2 \text{ con resto } 1$$

Significato "fisico"  
(riempio le scatole progressivamente)


Riempio **completamente** 1 scatola da 8 palline-bit (il quoziente: i moduli completati).

La seconda scatola contiene solo 5 palline-bit (il resto della divisione).





Dove cerco una certa pallina (e.g. la #13)?  
- Numero scatola  
- Numero della pallina nella scatola





## Mappatura diretta





Ib \$9, 13(\$0)

MM

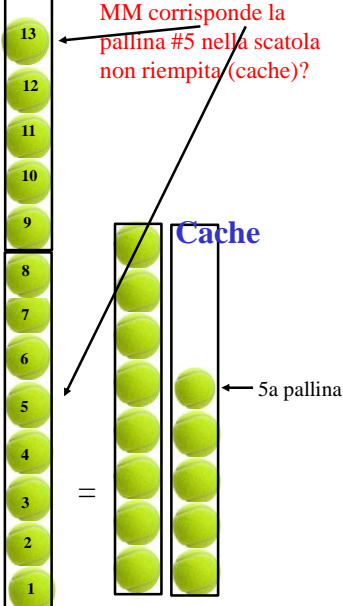
$13 : 8 = 1$  con resto 5

$9 : 4 = 2$  con resto 1

Significato "fisico"  
(riempio le scatole progressivamente)

Riempio **completamente** 1 scatola da 8 palline-bit (il quoziente: i moduli completati).

La seconda scatola contiene solo 5 palline-bit (il resto della divisione).



A quale posizione in MM corrisponde la pallina #5 nella scatola non riempita (cache)?

Cache


← 5a pallina

=


<http://borghese.di.unimi.it/>

A.A. 2022-2023

37/54



## Criterio di mappatura



La MMU "ragiona" come se ci fossero tante cache.

A ogni insieme di dati consecutivi in MM, pari alla capacità della cache, è assegnato un numero progressivo. Questo indirizzo progressivo rappresenta il **macro-blocco di memoria MM**. L'indirizzo viene trasformato in **numero di macro-blocco** (sulla pallina viene scritto il numero della scatola di appartenenza).

I dati all'interno del macro-blocco di RAM vengono mappati in modo ordinato in cache: le parole di memoria sono ordinate per indirizzo crescente. Indirizzi adiacenti all'interno del macro-blocco di RAM sono mappati su indirizzi adiacenti in cache (le palline vengono inserite nella scatola in ordine sequenziale). Il numero d'ordine all'interno del contenitore rappresenta il **numero all'interno della scatola**.

Dall'indirizzo di MM, si può quindi ricavare la posizione del dato all'interno della cache (dal numero della pallina si può ricavare la sua posizione all'interno della scatola).

A.A. 2022-2023

38/54

<http://borghese.di.unimi.it/>



## Corrispondenza diretta

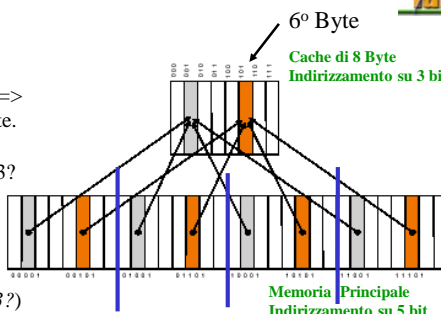


lb \$9, 53(\$0)

Supponiamo di avere una cache con **8 linee di un byte** =>  
**Capacità\_Cache = 8 Byte**: 8 celle di memoria da 1 Byte.

A quale macro-blocco di MM corrisponde l'indirizzo 53?  
(quante scatole da 8 palline riusciamo a riempire completamente?)

A quale Byte all'interno della cache corrisponde?  
(quale pallina nella scatola identificata è la pallina #53?)



### Divisione in modulo!

$53 / 8 = 6$  occorre riempire 6 scatole da 8 palline (saltare 6 blocchi di MM da 8 byte) => la pallina #53 è contenuta nella 7a scatola.

$R = 5$  Il resto rappresenta la posizione della pallina nella scatola, cioè la **posizione del dato all'interno della memoria cache**. La pallina sarà la 5a pallina all'interno della 7a scatola.

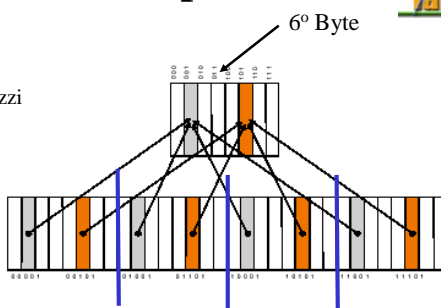


## Corrispondenza diretta in pratica



lb \$9, 53(\$0)

L'indirizzo 53 corrisponde al 54o Byte perchè gli indirizzi partono da zero.



### Divisione in modulo!

$53 / 8 = 6$  occorre riempire 6 scatole da 8 palline (saltare 6 blocchi di MM da 8 byte) => la pallina #53 è contenuta nella 7a scatola. NB La 7a scatola è il **macro-blocco di MM numero 6**, perché si inizia a contare dal macro-blocco #0 (scatola #0).

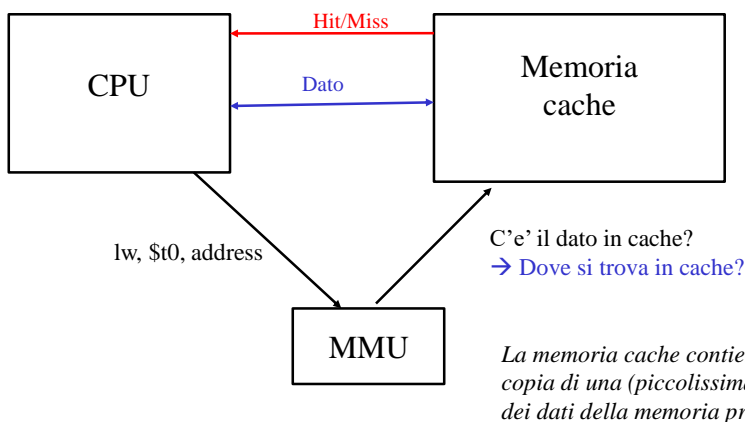
$R = 5$  Il resto rappresenta la posizione della pallina nella scatola, cioè la **posizione del dato all'interno della memoria cache**. Riempio interamente 5 byte e quindi la pallina sarà la 6a. NB La 6a pallina sarà la pallina numero 5 partendo a contare da zero.

Abbiamo identificato la possibile posizione in cache del dato in posizione 53 in RAM.  
NB Questa posizione in cache è la stessa posizione anche dei dati: ... 37, 45, 53, 61, 69...

**Ogni cella di cache può contenere un dato che proviene da un macro-blocco diverso.**



## Le domande alla MMU sulla memoria cache



Abbiamo identificato un algoritmo per determinare la posizione del dato in cache:

- NumeroMacroBlocco\_MM = Indirizzo\_MM / Capacità\_cache
- Posizione\_Cache = resto



## Cache con linee ampie più di un Byte



- Diversa ampiezza tra MM e cache
  - Gli indirizzi di cache sono riferiti a gruppi di byte (e.g. 4 parole)
  - Gli indirizzi di MM sono riferiti al byte.

Organizzazione **a matrice** della cache.

Devo determinare la posizione della pallina 9:



- Numero di linea
- Numero di colonna

Numero di linea = 2 =>  $9 / \text{num\_palline\_linea} = 9 / 4 = 2$

(riempio due linee, la pallina si trova nella terza linea)

Numero di Colonna = 1 =>  $R = 1$

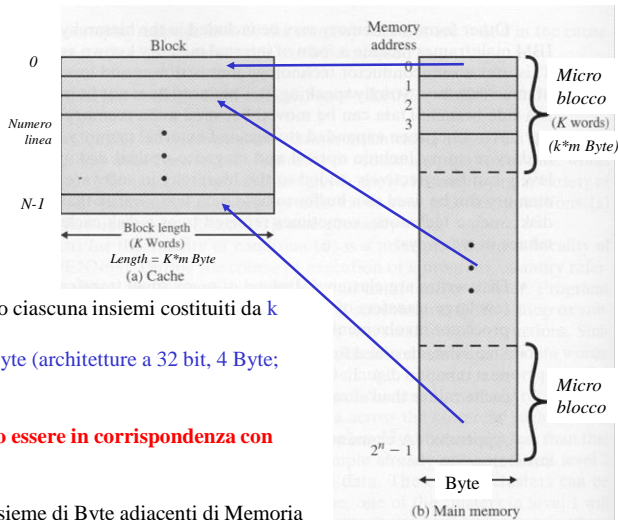


## Determinazione della legge di corrispondenza generale (linee di k parole)



4 misure di capacità:

- Cache
- Linea di cache.
- Parola.
- Byte.



Le  $N$  linee di una cache contengono ciascuna insieme costituiti da  $k$  parole.

Ciascuna parola è costituita da  $m$  byte (architetture a 32 bit, 4 Byte; architetture a 64 bit, 8 Byte)

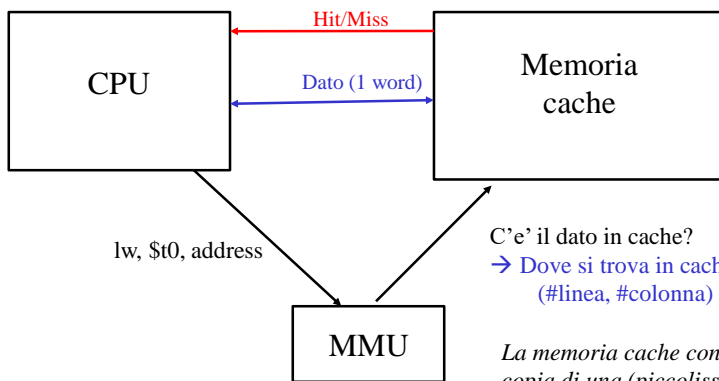
**Linee diverse della cache possono essere in corrispondenza con micro-blocchi di MM diversi.**

Posso mettere in corrispondenza un insieme di Byte adiacenti di Memoria Principale (micro-blocco) di  $k * m$  byte con una intera linea di cache (i dati sulla linea di cache (micro-blocchi) hanno indirizzi adiacenti in MM).

<http://borghese.di.unimi.it/>



## Le domande alla MMU sulla memoria cache



La memoria cache contiene una copia di una (piccolissima) parte dei dati della memoria principale.

Abbiamo identificato un algoritmo per determinare la posizione del dato in cache:

a) NumeroMacroBlocco\_MM = Indirizzo\_MM / Capacità\_cache

b) Posizione\_Cache = ?? = {#linea, #colonna}



## Corrispondenza diretta tra MM e cache



Cache con linee di ampiezza pari a 4 parole ed altezza di 8 linee con parole di 32 bit.

4 misure:

- **Capacità cache** = 8 linee \* 4 word/linea \* 4 Byte/word = 128 Byte
- **Linee di cache** = 4 word \* 4 Byte/word = 16 Byte
- **Parola** = 4 Byte
- **Elemento di base** = Byte.

Alcune richieste di dati:

lw \$t0, 76(\$zero)

$76 / 128 = 0$  (R=76): mappiamo il 1° blocco di MM sulla cache (macro-blocco 0)

$R = 76 / 16 = 4$  (R=12): il dato è contenuto nella **5ª linea** della cache (riempio interamente 4 linee con l'indirizzo 76).

$R = 12 / 4 = 3$  (R=0): il dato appartiene alla **4ª parola** nella 5ª linea della cache.

Il byte viene letto (trasferito nella CPU), assieme ai byte di indirizzo 73, 74, 75, nel registro \$t0.

lw \$t0, 204(\$zero)

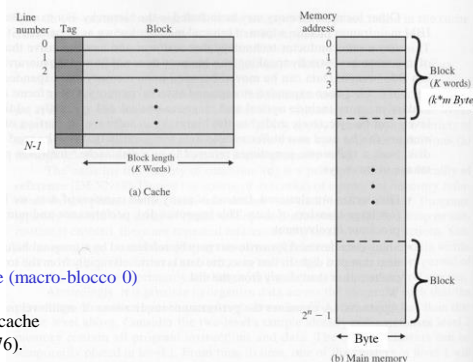
$204 / 128 = 1$  (R=76): mappiamo il 2° macro-blocco di MM sulla cache.

$R = 76$ . Il resto rappresenta i Byte («le palline») che dobbiamo ancora inserire nella cache («nella scatola»)

$R = 76 / 16 = 4$  il dato è contenuto nella **5ª linea** della cache (riempio interamente 4 linee con l'indirizzo).

$R = 12 / 4 = 3$ : il dato appartiene alla **4ª parola** nella 5ª linea della cache.

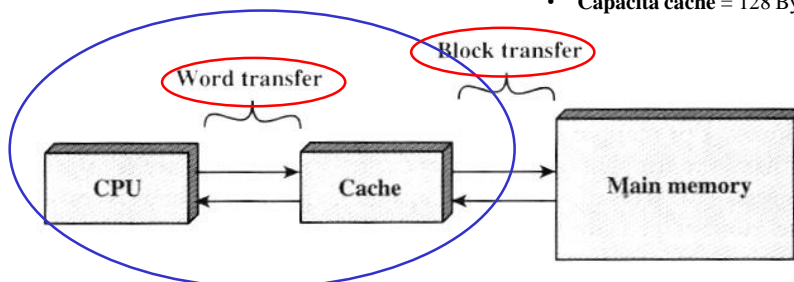
Il dato viene letto (trasferito nella CPU), assieme ai byte di indirizzo 205, 206, 207 in MM, nel registro \$t0.



## Principio di funzionamento di una cache



- **Linee di cache** = 4 word (16 Byte)
- **Capacità cache** = 128 Byte (32 word)



lw \$t0, 204(\$zero) → 5ª linea e 4ª parola della cache

Il byte viene letto (trasferito nella CPU), assieme ai byte di indirizzo 205, 206, 207 in MM, nel registro \$t0 (word transfer).

Se il dato non viene trovato in cache, dalla MM occorre trasferire i byte di indirizzo 192, 193, 194, 195 (word0), 196, 197, 198, 199 (word1), 200, 201, 202, 203 (word2), 204, 205, 206, 207 (word3). **Viene trasferito un (micro)blocco intero.**



# Implementazione efficace della mappatura



lw \$t0, 204(\$zero)

$204_{10} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1100\ 1100_2$

Campi potenza di 2

Address (in Byte) / Capacità\_cache (in Byte)

$204 / 128 = 1$ : mappiamo il 2° macro-blocco di MM sulla cache.

$204 / 128 = 204 / 2^7 = \gg 7$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1100\ 1100_2 \gg 7 = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1$

#Macro-Blocco di MM  $\uparrow$  Resto (7 bit)  
 $2^7 = 128$

R = 76. Il resto rappresenta i Byte («le palline») che dobbiamo ancora inserire nella cache («nella scatola») che è organizzata a matrice.

$R = 1001100_2 = 76_{10}$

R (in Byte) / Capacità\_linea (Byte)

$76 / 16 = 4$  il dato è contenuto nella 5ª linea della cache (riempio interamente 4 linee con l'indirizzo).

$204 / 16 = 76 / 2^4 = \gg 4$

$100\ 1100_2 \gg 4 = 100$

#linea Resto (4 bit)

R (in Byte) / Capacità della Word (#Byte/word)

$R = 12 / 4 = 3$ : il dato appartiene alla 4ª parola (riempio interamente 3 parole) nella 5ª linea della cache.

$12 / 4 = 12 / 2^2 = \gg 2$

$1100_2 \gg 2 = 11_2$

A. #word Resto (2 bit)

it\



# HW efficace della mappatura



lw \$t0, 204(\$zero)

$204_{10} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1100\ 1100_2$

$204 / 128 = 1$ ;  $2^7 = 128$  – Capacità della cache  
Numero di macro-blocco di MM  $\uparrow$

$R = 76 / 16$ ;  $2^4 = 16$  – Byte per linea  
Linea della cache (micro-blocco di MM)  $\uparrow$

$R = 12 / 4$ ;  $2^2 = 4$  – Byte per word  
Word nella linea (word nel micro-blocco)  $\uparrow$

R = 0

Cosa rappresenta l'ultimo resto?

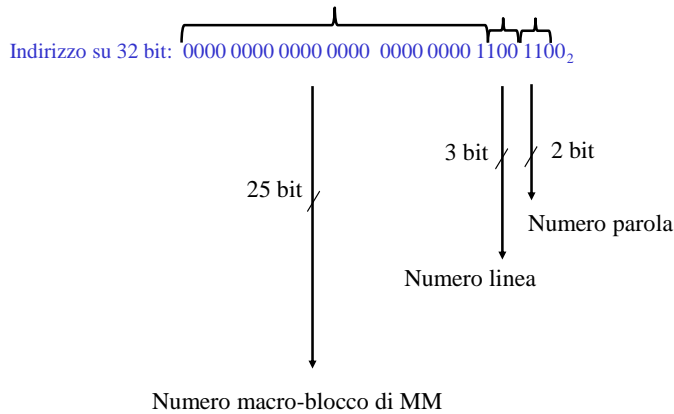
Quanti macro-blocchi di MM avremo delle dimensioni della cache?

Quanti micro-blocchi di MM avremo delle dimensioni della cache?





## I bus interni alla cache



Struttura di bus congruente al formato della cache



## Indirizzamento scartando i bit più significativi

Indirizzo cache (# linea)	Indirizzo decimale MM	Indirizzo binario MM
111	112-127, 240-255, 368-383,...	00 0111 0000 - 00 0111 1111
110	96, 224, 352	00 0110 0000 - 00 0110 1111
101	80-95, 208, 336...	00 0101 0000 - 00 0101 1111
100	64-79, 192-207, 320-335, 448, 461,...	00 0100 0000 - 00 0100 1111
011	48-63, 176, 304...	00 0011 0000 - 00 0011 1111
010	32-47, 160, 288...	00 0010 0000 - 00 0010 1111
001	16-31, 44, 272...	00 0001 0000 - 00 0001 1111
000	0-15, 128, 256, 384,...	00 0000 0000 - 00 0000 1111

NB: La capacità della cache è di:  $8 * 16 \text{ byte} = 128 \text{ byte} = \text{x}000\ 0000 - \text{x}111\ 1111$



## Indirizzamento scartando i bit più significativi (2° blocco di MM)



Indirizzo cache (# linea)	Indirizzo decimale MM	Indirizzo binario MM
111	112-127, 240-255, 368-383,...	00 1111 0000 - 00 1111 1111
110	96, 224, 352	00 1110 0000 - 00 1110 1111
101	80-95, 208, 336...	00 1101 0000 - 00 1101 1111
100	64-79, 192-207, 320-335, 448, 461,...	00 1100 0000 - 00 1100 1111
011	48-63, 176, 304...	00 1011 0000 - 00 1011 1111
010	32-47, 160, 288...	00 1010 0000 - 00 1010 1111
001	16-31, 44, 272...	00 1001 0000 - 00 1001 1111
000	0-15, 128, 256, 384,...	00 1000 0000 - 00 1000 1111

NB: La capacità della cache è di:  $8 * 16 \text{ byte} = 128 \text{ byte} = x1000\ 0000 - x1111\ 1111$



## Indirizzamento scartando i bit più significativi (3° blocco di MM)



Indirizzo cache (# linea)	Indirizzo decimale MM	Indirizzo binario MM
111	112-127, 240-255, 368-383,...	01 0111 0000 - 01 0111 1111
110	96, 224, 352	01 0110 0000 - 01 0110 1111
101	80-95, 208, 336...	01 0101 0000 - 01 0101 1111
100	64-79, 192-207, 320-335, 448, 461,...	01 0100 0000 - 01 0100 1111
011	48-63, 176, 304...	01 0011 0000 - 01 0011 1111
010	32-47, 160, 288...	01 0010 0000 - 01 0010 1111
001	16-31, 44, 272...	01 0001 0000 - 01 0001 1111
000	0-15, 128, 256, 384,...	01 0000 0000 - 01 0000 1111

NB: La capacità della cache è di:  $8 * 16 \text{ byte} = 128 \text{ byte} = x1\ 0000\ 0000 - x1\ 0111\ 1111$

