Struttura gerarchica delle memorie

F. Pedersini

Dipartimento di Informatica
Università degli studi di Milano

Considerazioni introduttive

Tipologie di accesso a memoria:

- **Sequenziale** (e.g. Nastri)
- **Diretto** (posizionamento + attesa, e.g. dischi).
- **Random Access** (e.g. Cache e Memoria principale).
- **Associativa** (accesso al dato mediante chiave, non mediante indirizzo)

tipologie differenti ➔ velocità di accesso differenti

Dispositivi di memoria "sparsi" nell'elaboratore...

- Processore (**registri**)
- Memoria interna alla CPU (**cache interna**)
- Memoria esterna (**cache esterna + memoria principale**)
- Memoria di massa (**hard disk**)

... a "distanze" differenti dalla CPU:

distanze differenti ➔ velocità di accesso differenti

Considerazioni introduttive

Velocità e **costo** delle **differenti tecnologie** di memoria:

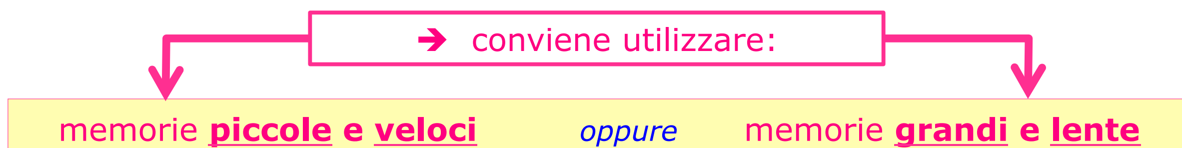
Tecnologia	tempo di accesso	costo di 1GB
S-RAM	0.5 ÷ 2.5 ns	2000 ÷ 5000 $
D-RAM	50 ÷ 70 ns	20 ÷ 75 $
dischi magnetici	1.000.000 ÷ 20.000.000 ns	0.1 ÷ 2 $

- Come organizzare la memoria di un elaboratore?
  - Quanta memoria? **(capacità)**
  - Quanto veloce? **(velocità)**

    Maggiore è la capacità, maggiore è il tempo di accesso.
  - Quanto deve/può costare? **(costo/bit)**

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

    Maggiore è la capacità, minore il costo per bit.



Struttura gerarchica della memoria

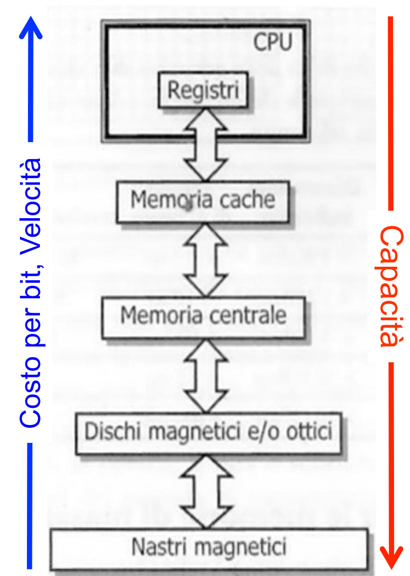
Struttura gerarchica:

Ciascun livello vede il livello inferiore

- a cui "attinge" quando necessario

- A livello superiore troviamo un sottoinsieme dei dati del livello inferiore.

Dispositivo di memoria	tecnologia	Capacità	Velocità di trasferimento (MB/s)
Registri	SRAM	< 1 kB	400.000 (4 byte in parallelo)
Cache Primaria (P4, 3Ghz)	SRAM	4÷12 kB	192.000 (32 byte in parallelo)
Cache Secondaria (P4, 3Ghz)	SRAM	256÷512 kB	96.000 (32 byte in parallelo)
Memoria centrale (DRAM)	DRAM	0,5 ÷ 4 GB	1600÷3000 (DDR – doppia lettura)
Dischi	magnetica	> 250 GB	800
Nastri	sequenz.	> 1 TB	10



Principi di località

Principi di località:

– *un programma riutilizza* ***spesso*** *dati e istruzioni* ***usati di recente***

– *...e utilizza* ***spesso*** *dati e istruzioni* ***vicini*** *a quelli usati* ***di recente***

Basandosi sul passato recente di un programma, è possibile predire con buona accuratezza quali dati e istruzioni esso userà nel prossimo futuro.

- **Principio di località temporale:** elementi ai quali si è fatto riferimento **di recente** saranno utilizzati ancora nel prossimo futuro.
- **Principio di località spaziale:** elementi i cui indirizzi **sono vicini**, tendono ad essere referenziati in tempi molto ravvicinati.

➔ si possono organizzare in memoria i programmi (e i dati) in modo da sfruttare al massimo il principio di località

Principio di funzionamento di una cache

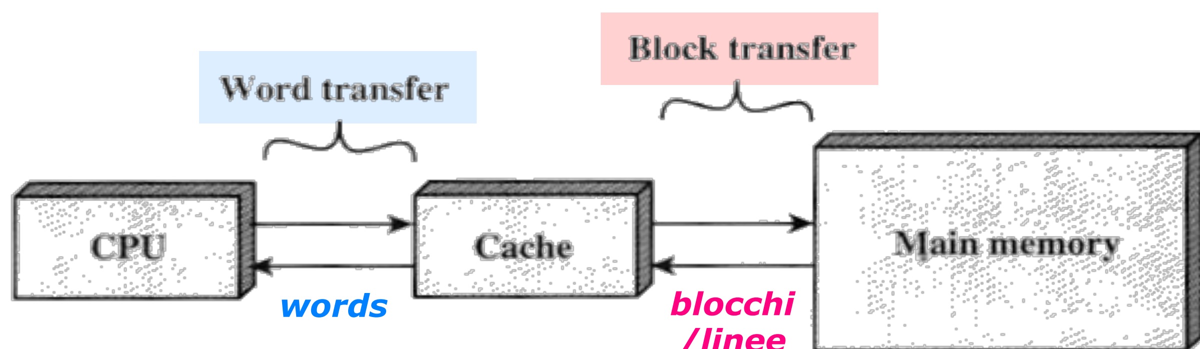
Struttura di una **memoria cache**

- La **cache** è interposta fra la CPU e la memoria principale
- Contiene **copia** di una parte della memoria principale, e cioè la parte che viene **correntemente utilizzata** dalla CPU.

- **Scopo: fornire alla CPU:**

  la velocità di trasferimento della memoria più veloce...

  ...e al contempo, **la capacità della memoria più grande**

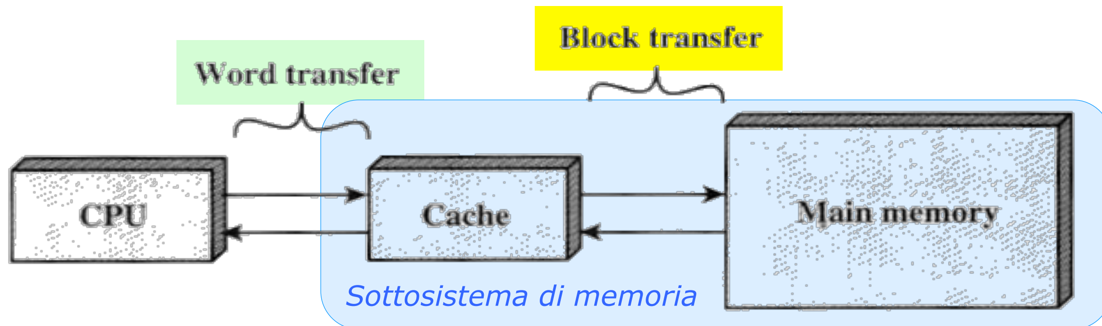


Sottosistema di memoria

Sottosistema di memoria: CACHE + MEMORIA PRINCIPALE

- circuiteria dedicata alla gestione del trasferimento dati tra CPU, cache e memoria principale

Funzione: porta nella cache primaria i dati richiesti mentre il binomio processore-memoria sta lavorando



HIT: la parola è già in cache

→ la leggo/scrivo direttamente in cache

MISS: la parola non si trova in cache

→ copio la parola (e quelle vicine) dalla memoria principale alla cache

Definizioni di funzionamento

Definizioni:

- **HIT:** successo nel tentativo di accesso ad un dato: il dato è presente al livello superiore della gerarchia.
- **MISS:** fallimento del tentativo di accesso al livello superiore della gerarchia → l'indirizzo deve essere cercato al livello inferiore
- **HIT_RATE:** percentuale dei tentativi di accesso ai livelli superiori della gerarchia che hanno avuto successo.

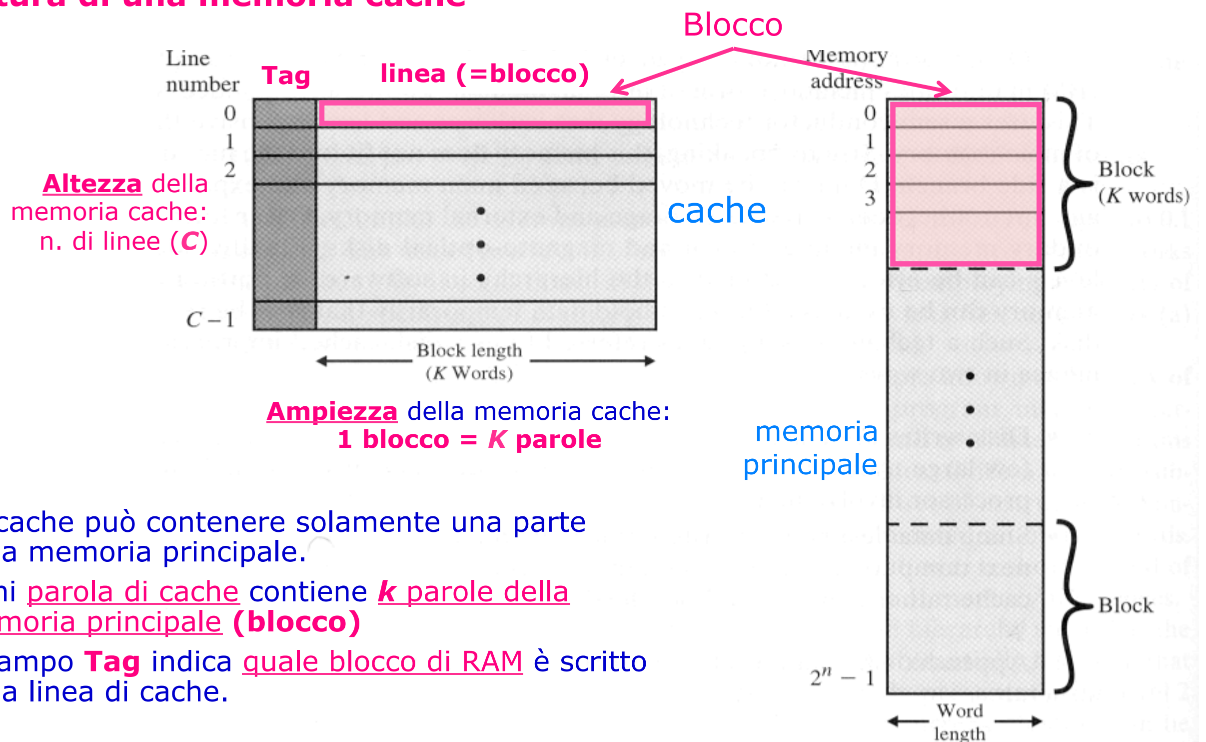
  HIT_RATE = Numero_successi / Num_totale_accessi_memoria
- **MISS_RATE:** percentuale dei tentativi di accesso ai livelli superiori della gerarchia che sono falliti

  MISS_RATE = Numero_fallimenti / Num_totale_accessi_memoria

HIT_RATE + MISS_RATE = 1 (100%)

Struttura e contenuto della memoria cache

Struttura di una memoria cache



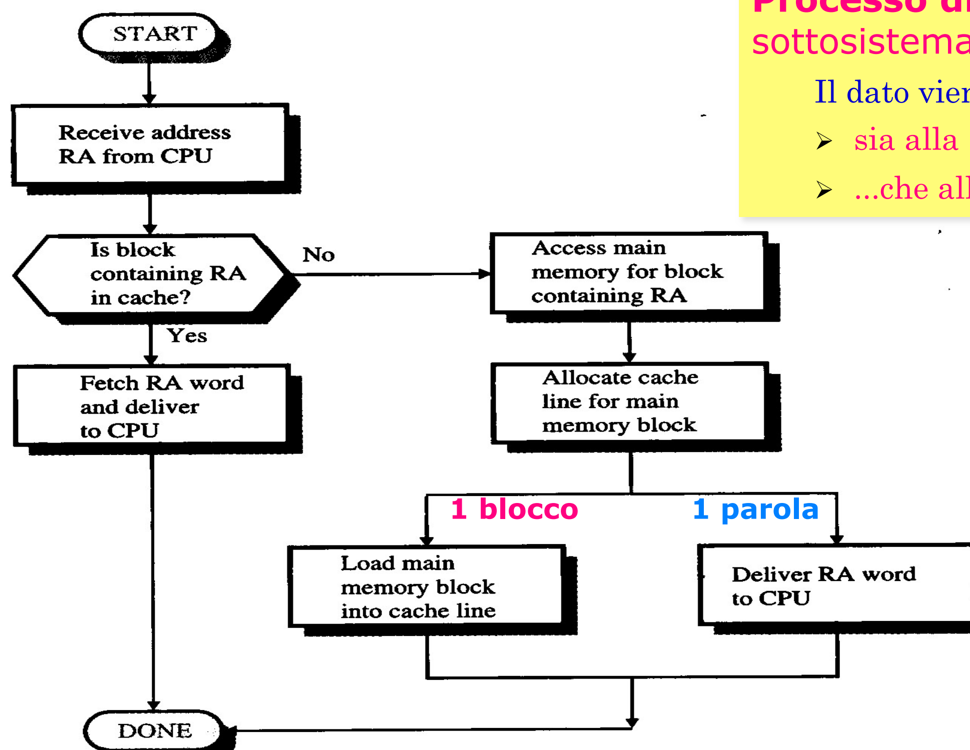
- La cache può contenere solamente una parte della memoria principale.
- Ogni parola di cache contiene *k* parole della memoria principale **(blocco)**
- Il campo **Tag** indica quale blocco di RAM è scritto nella linea di cache.

Funzionamento della memoria cache

Processo di lettura dal sottosistema di memoria

Il dato viene trasferito:

- sia alla **Cache (1 blocco)...**
- ...che alla **CPU (1 parola)**



Corrispondenza Cache – Memoria Principale

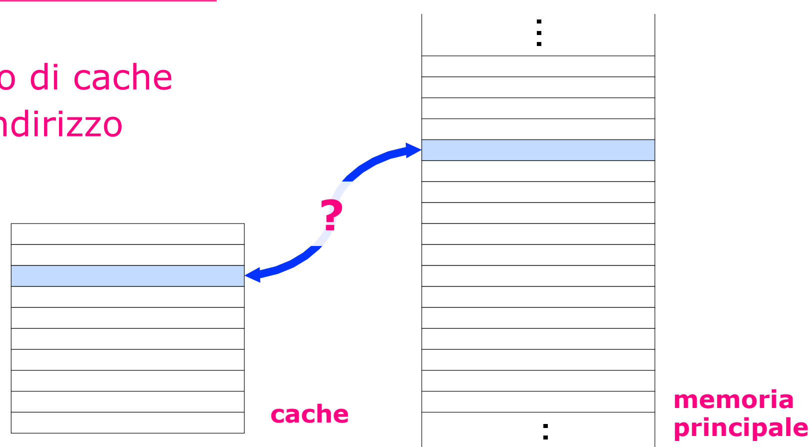
Una **linea** di cache corrisponde ad un **blocco** di memoria principale

- un blocco è costituito da **k parole**, ciascuna parola è costituita da **m byte** (ad esempio, MIPS: m=4)

Devo mettere in corrispondenza:

un **blocco di memoria principale** (*1* blocco: ***k*** words, ***n = k•m*** byte)
con una **linea di memoria cache**

- Come ottengo l'indirizzo di cache corrispondente ad un indirizzo di memoria principale?



Sommario

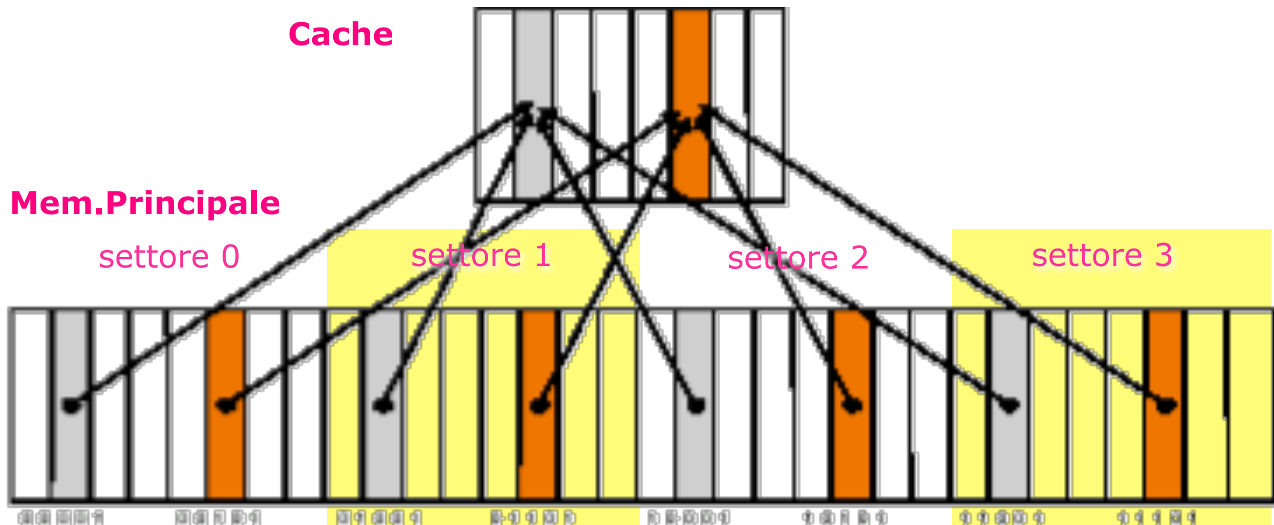
Architetture di memoria cache:

- cache a mappatura diretta
- cache parzialmente associative
- cache completamente associative

Mappatura diretta (direct map)

Cache a mappatura diretta: la memoria principale è divisa in SETTORI grandi quanto la cache intera

- Ad ogni indirizzo di **MP** corrisponde uno e un solo indirizzo di cache, ma diversi indirizzi di **MP** corrispondono allo stesso indirizzo cache



Esempio di parsing dell'indirizzo

Come calcolo l'indirizzo di Cache, a partire dall'indirizzo in Memoria Principale?

Es. dim. cache = dim. settori MP = ***1200 bytes***

Istruzione: `lw $t0, 1500($zero)` ➔ *indirizzo MP: 1500*

- **1500 div 1200 = 1** ➔ **settore 1** ← **divisione intera**
- **1500 mod 1200 = 300** ➔ **byte 300** ← **modulo (resto div. intera)**

Se il settore è diviso in ***6 linee****/blocchi da* ***200 byte*** *ciascuno:*

- **300 div 200 = 1** ➔ **linea/blocco 1** (il secondo)
- **300 mod 200 = 100** ➔ **byte 100**

Risultato: (dal settore 1), linea = 1, byte offset = 100

Parsing dell'indirizzo

PARSING
*ci serve un modo per fare il calcolo **velocemente***

Esempio: proviamo con dimensioni pari a **potenze di 10**

- Capacità memoria **principale:** **C = 100.000** (=10^5) **byte**
- Capacità memoria **cache** (e settori) **c = 1.000** (=10^3) **byte**

➔ **n. settori: C/c = 100**

Indirizzi a **5 cifre**, da: 00.000 a: 99.999

Indirizzo (decimale) di MP: 12345

- **12345 div 1000 = 12** ➔ **settore 12** ← divisione intera
- **123456 mod 1000 = 456** ➔ **byte 345** ← modulo (resto div. intera)

12345 ➔ 12 345

➔ Se le capacità sono **potenze di 10**, ***mod*()** e ***div*()** si ottengono come **sottoinsieme delle cifre** dell'indirizzo.

Parsing dell'indirizzo

*Esempio: cache ipotetica con dimensioni **potenze di 10:***

- **Memoria principale**: contiene **T=1000** (=10^3) **settori**
- **1 settore** (e la **cache**): contiene **N=100** (=10^2) **blocchi**
- **1 blocco (o linea)**: contiene **k=100** (=10^2) **parole**
- **1 parola**: contiene **m=10** (=10^1) **byte**

T, N, k, m sono tutti **potenze di 10.**

Altezza memoria: $10^3 * 10^2 * 10^2 * 10^1$ = **10^8 celle**
Indirizzi a **8 cifre**, da: 00.000.000 a: 99.999.999

Indirizzo (decimale) di MP: 12345678 ➔ 123 45 67 8

settore: **123**
blocco/linea: **45**
parola: **67**
byte: **8**

...è il byte **8** della parola **67**, nel blocco **45** del settore **123**

Parsing dell'indirizzo

*In un elaboratore, posso sfruttare tale proprietà con le **potenze di 2**:*

Struttura sottosistema di memoria cache:

- **Memoria principale:** contiene **T settori**
- **1 settore** (e la **cache**): contiene **N blocchi**
- **1 blocco/linea:** contiene **k parole**
- **1 parola:** contiene **m byte**

> Se **T, N, k, m** sono tutti **potenze di 2**, allora...
> **l'indirizzo di memoria**, in binario, corrisponde alla **composizione degli indirizzi di settore**, di **blocco**, di **parola** e di **byte.**

Es: **MP** composta da: **1024** settori, **64** blocchi di **16** parole di **4** byte

Capacità: 1024*64*16*4 = $2^{10}*2^{6}*2^{4}*2^{1}$ = $\mathbf{2^{22}}$ = 4 MB → **indirizzo: 22 bit**

ind. MP: **0000 0000 00 00 0000 0000 00**

settore — blocco — parola — byte

Parsing dell'indirizzo

Esempio:

Memoria Principale: 64 kBytes → indirizzi MP di **16 bit**

CACHE: **128 byte** suddivisi in: **8 linee,** ciascuna ontenente **4 parole** di **4 byte**

128 indirizzi – 2^3 linee, 2^2 parole/linea, 2^2 byte/parola ($2^3*2^2*2^2 = 2^7$)

Indirizzo **MP** di **16 bit:** **0000 0000 0000 00 (00)** → **0000 0000 0111 11 (11)**

1 Blocco (= **1 Linea**) contiene: **4** parole di **4** bytes → **16** bytes

- I **2 bit** meno significativi indicano la posizione del byte all'interno della parola: **BYTE OFFSET**
- I **2 bit** seguenti indicano la posizione della parola ricercata all'interno della linea della cache: **WORD OFFSET**
- I **3 bit** seguenti indicano la linea della cache interessata: **INDEX**
- I **bit seguenti** (16–2–2–3 = **9**) indicano il N. del settore della memoria principale messo in corrispondenza con la cache: **SETTORE**

Esempio: indirizzamento di una MP

0000 0000 0000 00(00)

↓

0000 0000 0111 11(11)

128 indirizzi diversi
(32 parole di 4 byte)

BLOCCO
4 words = 16 byte

Indirizzo				
0000 0000 0111 1100 – 31				Byte 127
0000 0000 0111 1000 – 30				
0000 0000 0111 0100 – 29				
0000 0000 0111 0000 – 28				
0000 0000 0110 1100 – 27				
0000 0000 0110 1000 – 26				
0000 0000 0110 0100 – 25				
0000 0000 0110 0000 – 24				
0000 0000 0101 1100 – 23				
0000 0000 0101 1000 – 22				
0000 0000 0101 0100 – 21				
0000 0000 0101 0000 – 20				
0000 0000 0100 1100 – 19				
0000 0000 0100 1000 – 18				
0000 0000 0100 0100 – 17				
0000 0000 0100 0000 – 16				
0000 0000 0011 1100 – 15				
0000 0000 0011 1000 – 14				
0000 0000 0011 0100 – 13				
0000 0000 0011 0000 – 12				
0000 0000 0010 1100 – 11				
0000 0000 0010 1000 – 10				
0000 0000 0010 0100 – 9				
0000 0000 0010 0000 – 8				
0000 0000 0001 1100 – 7	1 blocco			
0000 0000 0001 1000 – 6				
0000 0000 0001 0100 – 5				
0000 0000 0001 0000 – 4				
0000 0000 0000 1100 – w 3				
0000 0000 0000 1000 – w 2	1 word			
0000 0000 0000 0100 – w 1				
0000 0000 0000 0000 – w 0	Byte 0	Byte 1	Byte 2	Byte 3

Indirizzamento scartando i bit meno significativi

Indirizzo cache LINEA n.	Indirizzo decimale Memoria Principale	Indirizzo binario Memoria Principale
7 (111)	112-127, 240-255, 368-383, ...	00 0111 0000 – 00 0111 1111
6 (110)	96-111, 224-239, 352-367, …	00 0110 0000 – 00 0110 1111
5 (101)	80-95, 208-223, 336-351, ...	00 0101 0000 – 00 0101 1111
4 (100)	64-79, 192-207, 320-335, 448-461,...	00 0100 0000 – 00 0100 1111
3 (011)	48-63, 176-191, 304-319, ...	00 0011 0000 – 00 0011 1111
2 (010)	32-47, 160-175, 288-303, ...	00 0010 0000 – 00 0010 1111
1 (001)	16-31, 144-159, 272-287, ...	00 0001 0000 – 00 0001 1111
0 (000)	0-15, 128-143, 256-271, 384-399, ...	00 0000 0000 – 00 0000 1111

Capacità della cache: **8 linee * 4 words * 4 byte = 128 byte**

indirizzi di MP da: **x...x000 0000** a: **x...x111 1111**

Parsing dell'indirizzo

0000 0000 0000 00 (00) **0000 0000 0111 11 (11)**

128 indirizzi diversi (32 parole di 4 byte)

Prendiamo un indirizzo **M** di MP ed **operiamo per divisioni successive:**

M / [(bytes/parola) * (parole/linea) * n.linee] (Capacità cache) = N. settore di MP: **settore**

- Il resto **R1:** rappresenta l'offset (in byte) all'interno della cache.

R1/ [(bytes/parola) * (parole/linea)] (Dim. linea) = Numero linea di cache: **INDEX**

- Il resto, **R2**, rappresenta l'offset (in byte) all'interno della linea di cache.

R2/ [(bytes/parola)] (Dim. parola) = N. parola nella linea di cache: **WORD OFFSET**

- Il resto, **R3**, rappresenta l'offset (in byte) nella parola: **BYTE OFFSET**

Esempio di parsing dell'indirizzo

Cache – 8 linee di 4 parole (di 4 byte):

- Blocco / linea di cache ha dimensione: **n** = 4 * 4 byte = 16 byte
- Capacità della cache: **C** = 8 * 16 byte = 128 byte

128 indirizzi diversi (8 linee di 4 parole di 4 byte)

```
lw $t0, 195($zero)
```

- **195 div [4 * 4 * 8] = 1** → settore 1 (2°), resto: **R1 = 195 mod 128 = 67**
  - R1 rappresenta l'offset in byte all'interno della cache
- **67 div [4 * 4] = 4** → linea cache: 4 (5ª), resto: **R2 = 67 mod 16 = 3**
  - R2 rappresenta l'offset in byte all'interno della linea di cache
- **3 div 4 = 0** → parola 0 della linea (1°), resto: **R3 = 3 – 0 * 4 = 3**
  - R3 rappresenta l'offset in byte all'interno della parola

195_{10} = 11000011_2 = 0000 0000 1 100 00 11

Come scoprire se un dato è presente in cache?

Problema: come sapere se la cache contiene il mio dato?

- Dato un indirizzo **M** in memoria principale, la sua posizione in cache è definita da: **Index, Word Offset, Byte Offset**
- Nel meccanismo di mappatura diretta **si perde l'informazione del numero di settore di MP** di appartenenza:
- È necessario memorizzare il N. di settore: campo **TAG**
- Non è invece necessario memorizzare gli altri campi, perché ottenuti da **M.**

Campo "TAG":

- Associato a ciascuna linea di cache
- Contiene l'informazione: da quale settore di MP arriva il blocco?

TAG è costituito dai bit che costituiscono la parte più significativa dell'indirizzo.

- E' costituito da K bit: $K = M - \sup(\log_2 C)$
  - Nell'esempio precedente: $K = 16 - \sup(\log_2 128) = 9$ bit

Bit di validità

Può accadere che il dato in cache **non** sia significativo:

- ad esempio: cache ancora vuota → valore in cache non consistente con il valore in MP

Bit di validità:

- Indica la consistenza tra cache e MP
- Quando un blocco viene copiato da MP a cache, quindi è consistente:
  - **VALID → 1**
- Quando un blocco in cache non ha, o perde, la consistenza:
  - **VALID → 0**

→ Un blocco in cache può essere utilizzato (**HIT**) se:

- **Il campo TAG coincide con l'indirizzo**
- **VALID = 1**

...altrimenti deve essere recuperato da MP (**MISS**)

Struttura di una linea di cache

Quanti bit è lungo il campo **TAG**?

Quanti ne servono per indirizzare **tutti i settori**

- nell'esempio precedente: ***$512 = 2^9$ Settori → 9 bit***

Quanto è lunga una linea di cache?

Esempio precedente: **2^9**=512 settori, 8 blocchi di **4** parole da **4** byte:

→ avremo linee di cache di lunghezza: ***L = 9 + 1 + (4*4)*8 bit/byte = 138 bit***

Come leggere / scrivere su cache

Meccanismo di lettura/scrittura su cache:

1. Individuare la linea della cache dalla quale leggere / scrivere
2. Confrontare il campo **TAG** con il **settore** di MP in cui risiede il dato.
3. Controllare il bit di validità.
4. Se:

   (TAG = parte alta dell'indirizzo) ***AND*** *(Valid_bit = TRUE)*

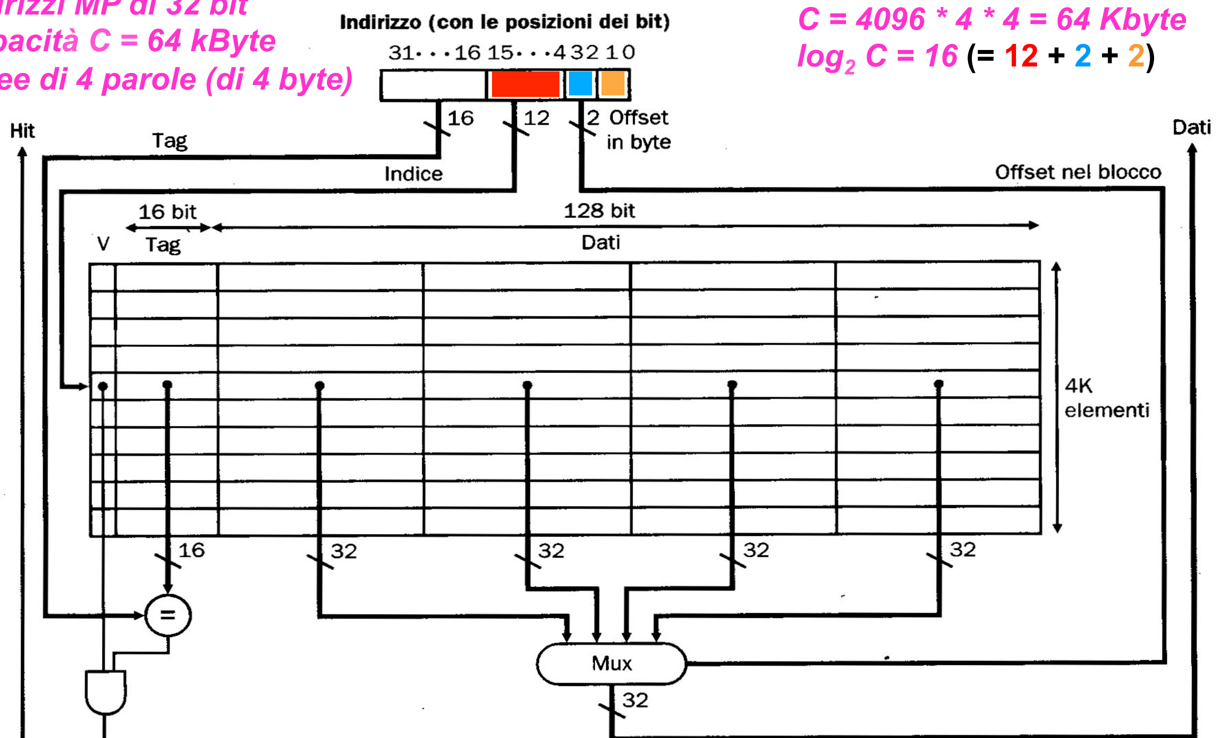
   Allora: ***HIT!***

   Else: ***MISS:*** leggere (scrivere) **l'intero blocco (linea)** di appartenenza della parola indirizzata.

Schema: cache a mappatura diretta

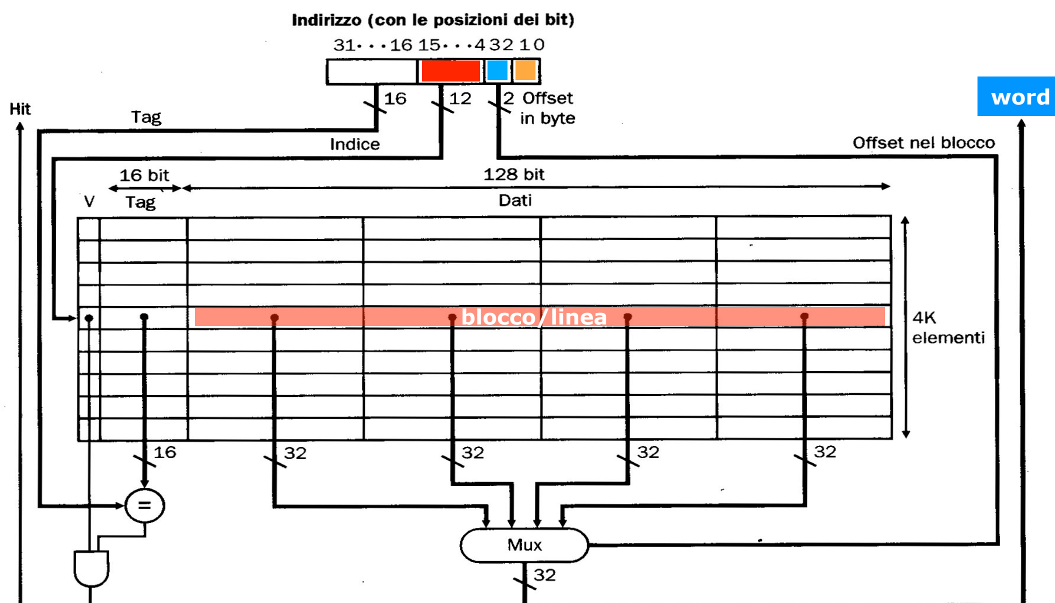
Indirizzi MP di 32 bit
Capacità C = 64 kByte
Linee di 4 parole (di 4 byte)

*C = 4096 * 4 * 4 = 64 Kbyte*
$\log_2 C = 16$ (= 12 + 2 + 2)



Schema: cache a mappatura diretta

Capacità cache: **64 kByte** (2^{16}) Bus indirizzi: **32 bit**
suddivisa in **4096** linee di **4** parole di **4** byte ($2^{16} = 2^{12} * 2^2 * 2^2$)



Esercizi

- *Sia data una cache a corrispondenza diretta contenente **64 kbyte** di dati e avente blocchi di **8 parole** da **32 bit**. Assumendo che gli indirizzi siano di 32 bit, qual è il numero totale di bit richiesto per ogni linea di cache (compresi il campo TAG e il bit di validità)?*

- *Supponendo che il MIPS abbia una cache di 512 byte, indicare cosa succede nei campi della cache quando vengono eseguite le seguenti istruzioni:*

```
lw $t1, 0x0000($t0)   $t0 = 1024 (= 1,024 byte)
lw $t1, 0x0000($t0)   $t0 = 0
lw $t1, 0x0202($t0)   $t0 = 1024 (= 1,024 byte)
lw $t1, 0x0001($t0)   $t0 = 0
lw $t1, 0x0201($t0)   $t0 = 1024 (= 1,024 byte)
```


Sommario

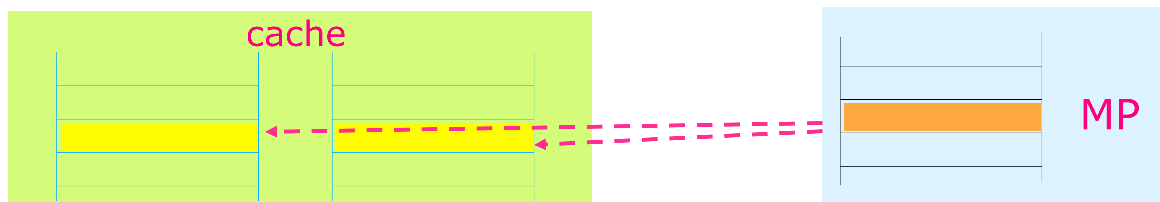
Architetture di memoria cache:

- cache a mappatura diretta
- cache parzialmente associative (N-associative)
- cache completamente associative

Limiti cache a mappatura diretta

- Cache a mappatura diretta:
  per ogni locazione di MP, esiste una e una sola locazione in cache.
  - Locazione determinata a priori: { ***Index* , *Word Offset* , *Byte Offset*** }.
  - SOLUZIONE SEMPLICE, MA INEFFICIENTE: potrei avere la cache semivuota, e ciononostante riscrivere linee già occupate

- Idea:
  prevedo per ogni linea, più alternative di collocazione
  - Linee multiple di cache, per ogni blocco di MP



Memorie n-associative

N-associative, o set-associative, o a N vie:

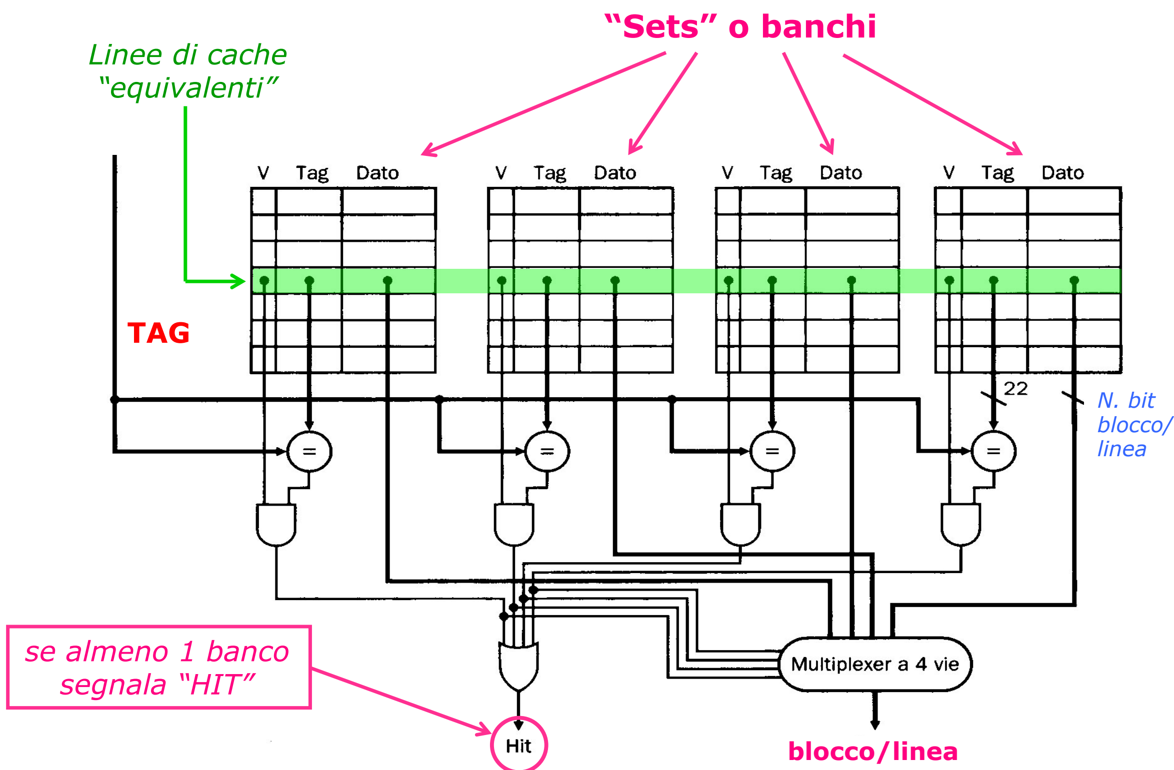
- La memoria cache è suddivisa in **N** insiemi, o banchi, ciascuno di **k** linee, posti in parallelo
  - ***Blocco / Linea di cache:***
    blocco di parole scambiate (lette/scritte) tra MP e cache.
  - ***Set / banco:*** strutturato come una cache a mappatura diretta
  - ***Capacità della cache:***

    Capacità [byte] = N.bytes/banco × N.banchi

- La corrispondenza tra MP e linea di un banco
  è a **mappatura diretta**

- La corrispondenza tra MP e banco è **associativa**
  - Per cercare un dato devo analizzare tutti i banchi della linea indirizzata.

Memorie N-associative



Accesso a cache N-associative

Come nelle cache a mappatura diretta:

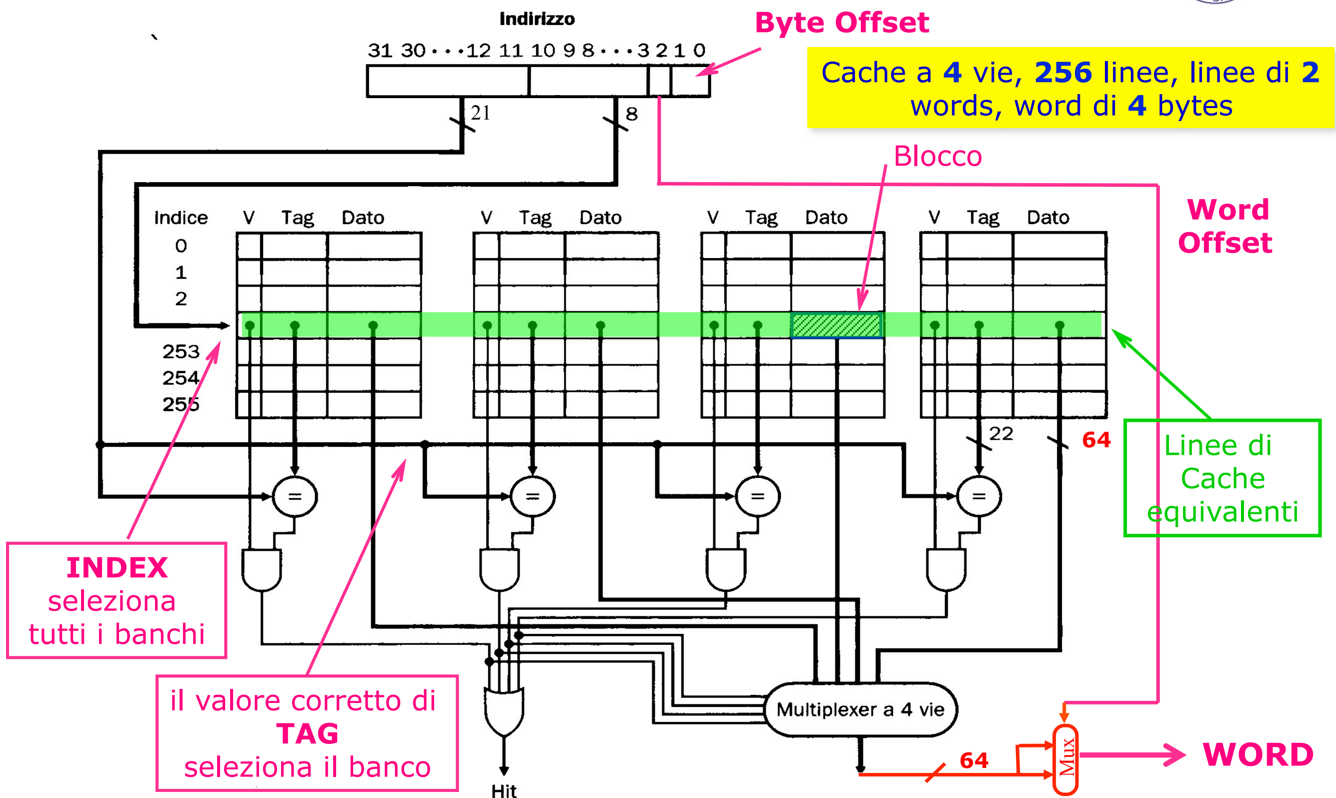
- **INDICE –** seleziona la **linea** su **tutti i banchi** (insieme) che potrebbe contenere il blocco desiderato.
  - La parola di MP memorizzata in cache si trova in una particolare linea di uno dei banchi → linea individuata dall' indice
  - L'indice è costituito da **k** bit, dove: $k = \log_2(\#linee)$ analogo al numero di linea nelle cache a mappatura diretta

- **TAG –** contiene il **settore** di provenienza, dalla MP
  - Cerca il tag di MP all'interno dei TAG associati alla linea individuata, in tutti i banchi.
  - Nelle memorie N–associative, ho N possibili linee da controllare

Ma, in più…

- I segnali "HIT" di ogni banco pilotano anche il **MUX**, che seleziona il banco di cache che effettivamente contiene il blocco desiderato.

Memorie set-associative



esempio (cont.) – blocchi di 2 parole

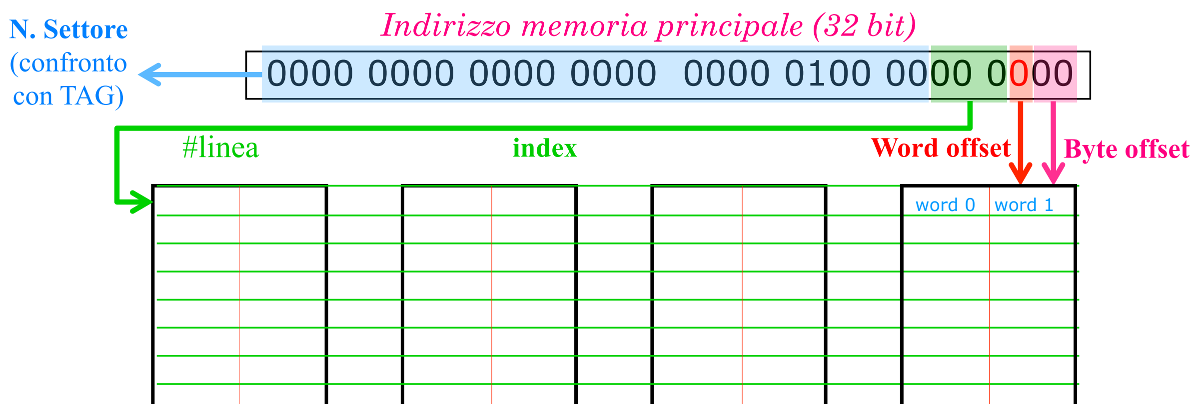
*Es: Indirizzi da **32** bit ➔ memoria principale: 2^{32} B = **4 GB***
*Cache a **4** vie, **8** linee, linee di **2** words, word di **4** bytes*

Capacità cache: C_{TOT} = 4*8*2*4 = $2^2 \times 2^3 \times 2^1 \times 2^2 = 2^8$ = **256 byte**

Capacità di un banco: C_B = 8*2*4 = $2^3 \times 2^1 \times 2^2$ = 26 = **64 byte**

- 4 banchi, ciascuno di 8 linee ➜ **index: 3** bit
- Linea di cache = 2 word, da 4 byte ➜ **byte offset: 2** bit, **word offset: 1** bit

Come viene elaborato l'indirizzo: `lw 0($s0) ? ($s0 = 1024)`



Sommario

Architetture di memoria cache:

- cache a mappatura diretta
- cache parzialmente associative (N-associative)
- cache completamente associative

Dalle cache ***direct map*** alle cache ***associative***

N*-associativa**: ***N canali (vie) di cache direct-map **in parallelo**

Set-associative ad una via (a corrispondenza diretta)

Blocco	Tag	Dato
0		
1		
2		
3		
4		
5		
6		
7		

Set-associative a due vie

Insieme	Tag	Dato	Tag	Dato
0				
1				
2				
3				

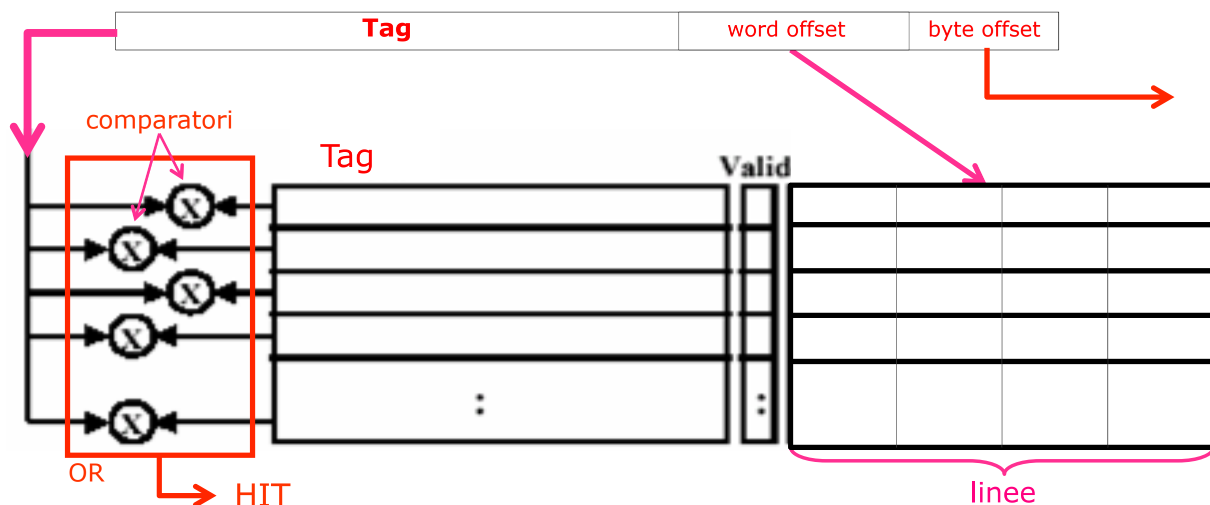
Set-associative a quattro vie

Insieme	Tag	Dato	Tag	Dato	Tag	Dato	Tag	Dato
0								
1								

Set-associative ad otto vie (completamente associativa)

Tag	Dato	Tag	Dato	Tag	Dato	Tag	Dato	Tag	Dato	Tag	Dato	Tag	Dato	Tag	Dato

Memorie associative



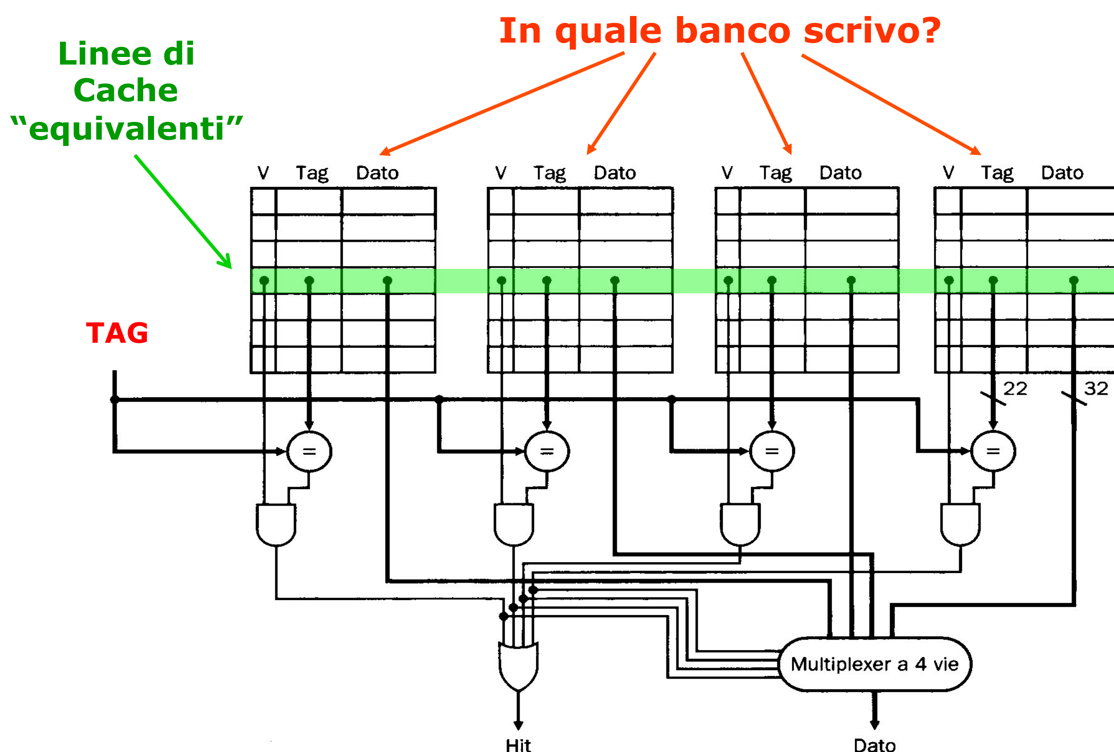
- Memoria completamente associativa.
  - Consente di caricare un blocco di RAM in una qualsiasi linea di cache.
  - Tramite comparatori individuo in quale blocco si trova il mio dato.
  - Il segnale di Hit si genera come AND (comparatore_output, Valido)
  - *Dove scrivo il blocco?*

Posizionamento di un blocco di MP in cache

Sintesi – **Posizionamento di un blocco in cache:**

- Corrispondenza diretta: in un'unica posizione.
  - Memoria ad **1 via**.
  - Controllo del tag del blocco → **1** comparazione
- N-associative: in **N** posizioni (N: grado di associatività).
  - Ho **N** insiemi (banchi)
  - Ciascun insieme è costituito da: **m = C/N** blocchi.
  - **N** comparazioni
- Completamente associative: in **C** posizioni (C blocchi).
  - Ciascun banco è costituito da **1 blocco**.
  - C insiemi o banchi.
  - **C** comparazioni: controllo di tutti i tag

Memorie N-associative



Strategie di sostituzione di un blocco

Strategie di sostituzione di un blocco in cache **associative**

Se devo scrivere un blocco di cache, quale banco scelgo?

Ho libertà di scelta: **N vie → N alternative**

Dove inserisco il blocco letto dalla memoria?

Vincolo: soluzione hardware, quindi algoritmo semplice!

- **a caso (random)**
  - Scelgo semplicemente a caso.
- **a rotazione ("*Round Robin*")**
  - Implementazione tramite buffer circolare.
- **LFU ("Least Frequently Used")**
  - Associa un contatore ad ogni blocco di cache.
- **LRU ("Least Recently Used")**
  - Viene associato ad ogni blocco un campo *"Time stamp"*

Prestazioni: *"random" non funziona molto peggio degli altri!*

Cache: *meccanismi di scrittura*

Strategie di riscrittura in MP (da cache a MP)

*ogni volta che scrivo un blocco nella cache, sovrascrivo il blocco precedente, che **potrebbe essere stato modificato***

→ **prima devo riscrivere il blocco precedente in MP**

2 strategie possibili:

1. Write-through: scrittura **contemporanea** in cache e in mem. principale

- ***Write_buffer*** per liberare la CPU (DEC 3100)
- Necessaria una gestione della sincronizzazione tra contenuto della Memoria Principale (che può essere letta anche da I/O e da altri processori) e Cache.
- Svantaggio: traffico intenso sul bus per trasferimenti di dati in memoria.

2. Write-back: scrittura **ritardata**: scrivo solo quando devo scaricare il blocco di cache e riportarlo in memoria principale

- Utilizzo un bit di flag: **DIRTY**, che viene settato quando modifico il contenuto del blocco (faccio una scrittura)
- Strategia vantaggiosa: minimizzo il traffico su bus

La cosa si complica un po' con cache n-associative…

Gestione dei fallimenti di una cache

Gestione dei casi di MISS

HIT: → il funzionamento della CPU non viene alterato.

MISS: → devo aspettare che il dato sia pronto in cache
→ **stallo!** : il programma non può continuare!!

Operazioni da eseguire in caso di MISS:

1. Ricaricare l'indirizzo dell'istruzione di accesso (PC→ PC–4)
2. Leggere il blocco di memoria dalla memoria principale.
3. Trasferire il blocco in cache (aggiornando i campi validità e tag)
4. Riavviare l'istruzione di accesso (fetch)

è un'operazione lunga! → MISS PENALTY

È quindi opportuno:

1. Rendere **minimo** il tasso di fallimento (**miss_rate**)
2. Rendere **massima la velocità** delle operazioni di gestione di MISS.

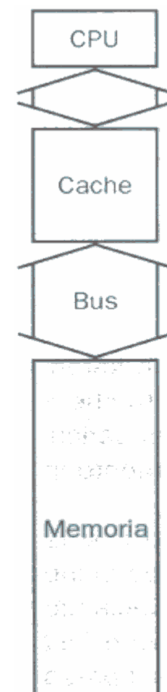
→ Massimizzare la velocità di trasferimento tra cache e memoria principale

Miglioramento prestazioni del sistema di memoria

Massimizzazione della velocità di scrittura in cache

la durata totale della gestione di MISS dipende dai tempi di accesso a memoria:

- **1 ciclo** di clock per inviare l'indirizzo.
- **15 cicli** per ciascuna attivazione della MP (lettura di parola).
- **1 ciclo** per trasferire una parola al livello superiore (cache).



ESEMPIO: *Blocco di cache: 4 parole; Bus dati ↔ RAM: 1 parola*

Miss_Penalty:

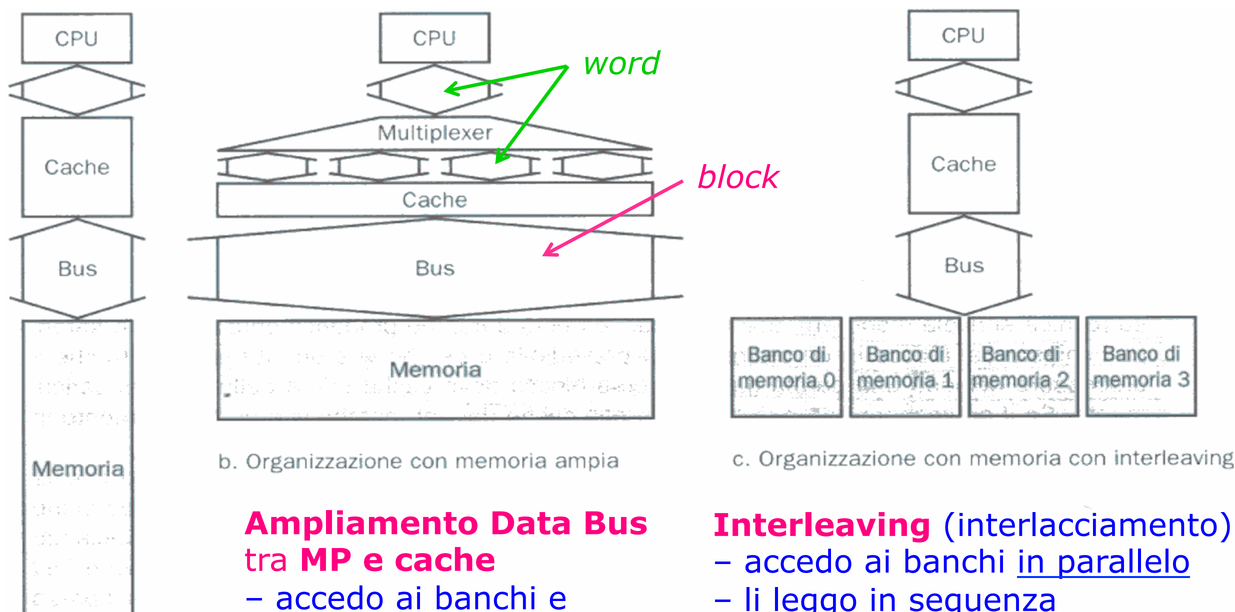
1 + 15 * 4 *(parole)* + 1 * 4 *(parole)* = ***65 cicli_clock***

Transfer Rate:

#byte/ciclo_clock = 4*(parole)* * 4*(byte/parola)* / 65*(cicli_clock)* =
≅ **0,25 *byte/ciclo_clock***

Obiettivo: ridurre la penalità di fallimento (**miss_penalty**)

Strutture per ridurre la *miss penalty*



b. Organizzazione con memoria ampia

c. Organizzazione con memoria con interleaving

Struttura tradizionale

Ampliamento Data Bus tra **MP e cache**
– accedo ai banchi e li leggo in parallelo

Interleaving (interlacciamento):
– accedo ai banchi in parallelo
– li leggo in sequenza

Valutazione riduzione di "miss_penalty"

- **Architettura standard**: penalità di miss è di **65** cicli_clock.
- **Maggiore ampiezza della memoria:**
  - Organizzazione della Memoria Principale per blocchi.
  - Bus più ampio (bus dati largo un blocco, **4 parole**).

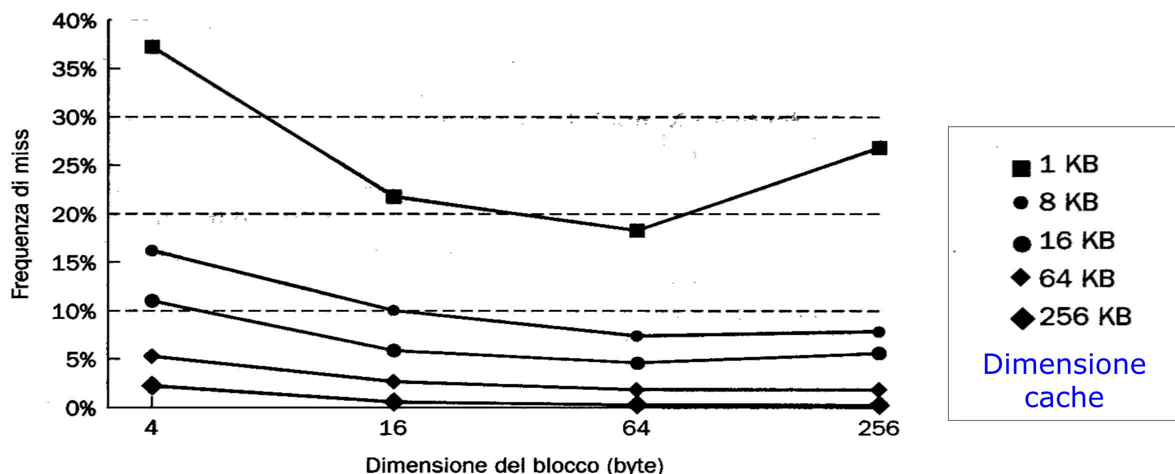
  Per blocchi di MP di 4 parole, blocchi di cache di 4 parole:
  - **Miss_penalty** = 1 + 15 * 1 + 1 * 1 = **17 cicli_clock**
  - ***Transfer_rate*** = 4(*parole*) * 4(*byte/parola*) / 17(*cicli_clock*)
    = **0,94 (*byte/ciclo_clock*)**
- **Interleaving:**
  - Organizzazione della Memoria Principale per banchi con accesso indipendente alla memoria (interleaving)
  - L'accesso alla parola da leggere in MP è fatto in parallelo su ogni banco
  - Bus standard ➔ trasferimento di 1 parola alla volta

  Per blocchi di MP di 1 parola, blocchi di cache di 4 parole:
  - **Miss_penalty** = 1 + 15 * 1 + 1*4 = **20 cicli_clock**
  - ***Transfer_rate*** = 4 (*parole*) * 4 (*byte/parola*) / 20 (*cicli_clock*) = **0,80 (*byte/ ciclo_clock*)**

Miss-rate e dimensione dei blocchi

Dimensionamento del blocco(linea) di cache:
- La dimensione della linea di cache è multipla della parola della macchina.
- La dimensione ottima della linea dipende dalla parola del processore.

- All'aumentare della dimensione della linea/blocco…
  - **Per la località spaziale ➔ diminuisce la frequenza di MISS.**
  - **Per le dimensioni del blocco rispetto alla capacità della cache ➔ aumenta la penalità di MISS (competizione per le poche linee di cache)**



Sistemi *multiprocessore*

Sistemi **multi-processore:**
N processori → N cache, ma **una sola MP**

Problema: Cache coherence
mantenimento della coerenza tra le informazioni contenute nelle N cache

Soluzioni:

- **Bus watching** with **write-through**
  - Tutti i processori adottano il meccanismo di write-through (scrittura in cache → scrittura contestuale in MP)
  - Il controller della cache monitora: il bus indirizzi + il segnale di controllo MemWrite della memoria principale
  - Invalida (*Valid bit* → 0) un blocco di cache se il suo corrispondente in memoria principale viene scritto.
- **Hardware trasparency**
  Circuito addizionale attivato ad ogni scrittura della Memoria Principale:
  - copia la parola aggiornata in tutte le cache che contengono quella parola.
- **Non-cacheable memory**
  - Viene definita un'area di memoria condivisa, che non viene copiata in cache.

Strategie moderne di caching

Cache a più livelli:

- Cache **primaria** nel processore.
- Cache **secondaria** esterna o nel chip, con bus dedicato per il trasferimento dati da/verso il processore.
  - Problemi: complessità nel circuito che deve assicurare la cache coherence.

Split-cache: separazione tra **Cache Dati** e **Cache Istruzioni**

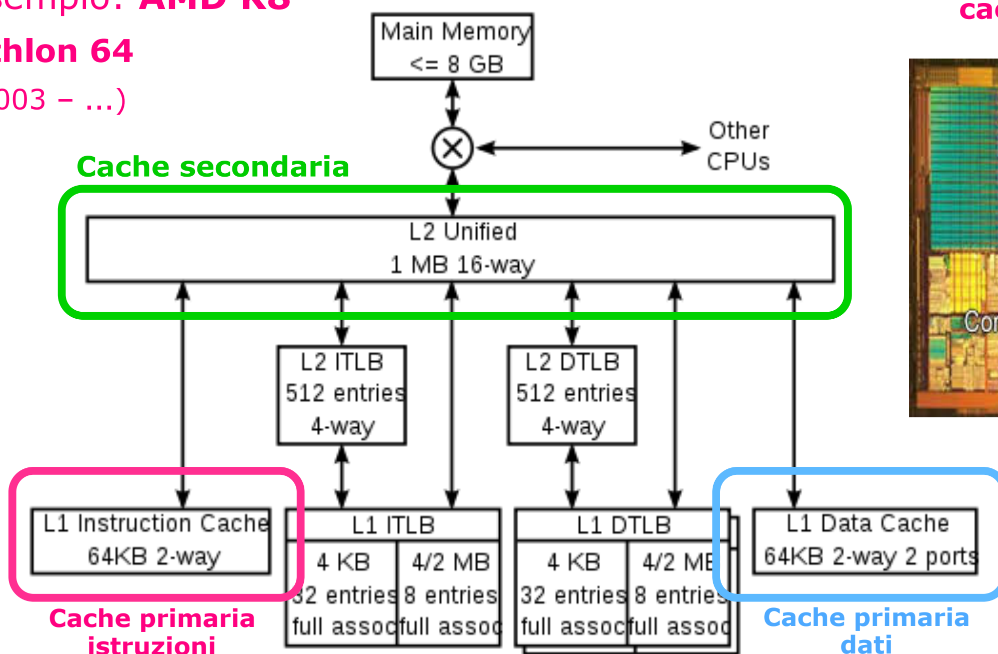
- Vantaggi
  - Principio di località diversamente ottimizzato per istruzioni e dati
  - Possibilità di analizzare le istruzioni in coda (contenute nella cache istruzioni) mentre si eseguono altre istruzioni (che lavorano su dati contenuti nella cache dati), senza dover competere per l'accesso alla cache.
  - 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 o istruzioni a seconda del frammento di codice correntemente in esecuzione.

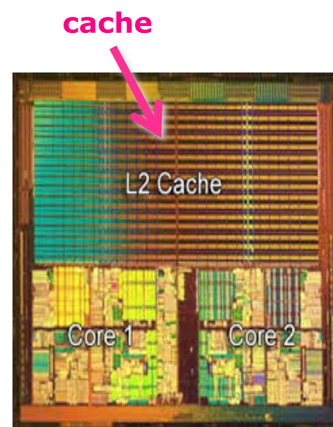
Cache moderne: esempio

Esempio: **AMD K8**

Athlon 64

(2003 – ...)





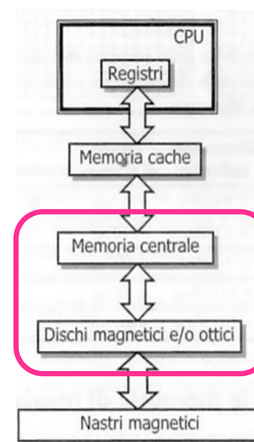
(immagini da Wikipedia)

Memoria Virtuale

Memoria virtuale: estensione (cache) della Memoria Principale (RAM) su memoria di massa (hard disk)

2 obiettivi:

- **Estensione** della **memoria fisica**. Maggiore quantità di memoria.
- Gestione del **multi-tasking**. trasparente, tramite la gestione della memoria virtuale.



concettualmente analoga alla cache.

- Blocco/linea di memoria → **Pagina**
- MISS → **Page Fault**

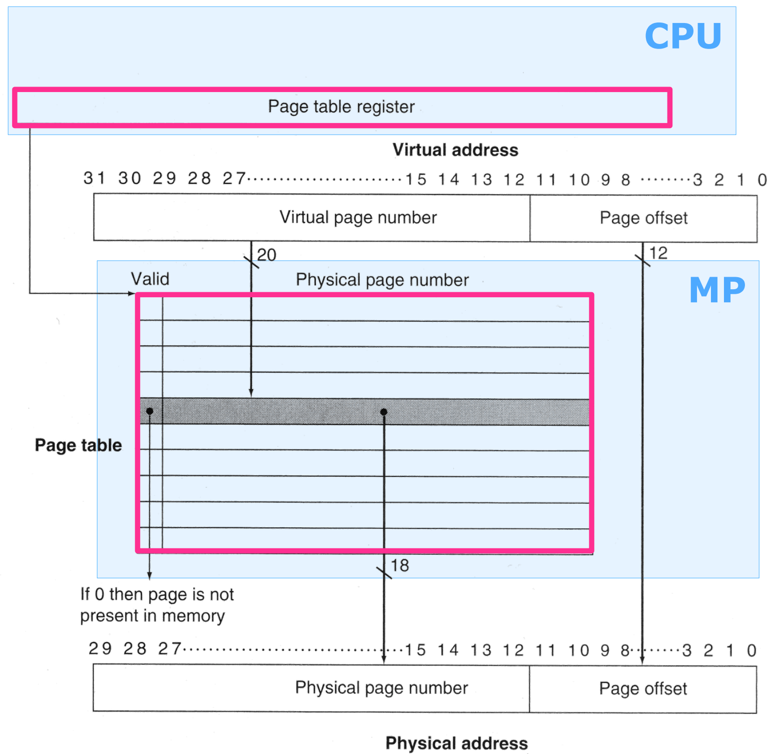
2 funzionalità:

- **ESTENSIONE**: spazio in memoria maggiore dello spazio fisico in MP
- **PROTEZIONE**: gestione di più processi (utenti), ciascuno con il proprio spazio privato di memoria, separato da tutti gli altri

Memoria Virtuale

Memoria virtuale: struttura

- Memoria suddivisa in **pagine**
- Ogni programma ha un proprio spazio di indirizzamento virtuale, gli indirizzi di CPU sono virtuali → indirizzo **virtuale**
- ...che viene mappato, quando si accede realmente alla memoria, in → indirizzo **fisico**
- Il mapping **virtuale → fisica** viene memorizzato nella **page table**
- La **page table** si trova in memoria principale, all'indirizzo specificato nel registro: **Page Table Register** che si trova in CPU



Memoria Virtuale

Esempio

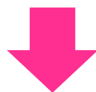
Memoria **virtuale**: **4 GB** (2^{32})

Memoria **fisica**: **1 GB** (2^{30})

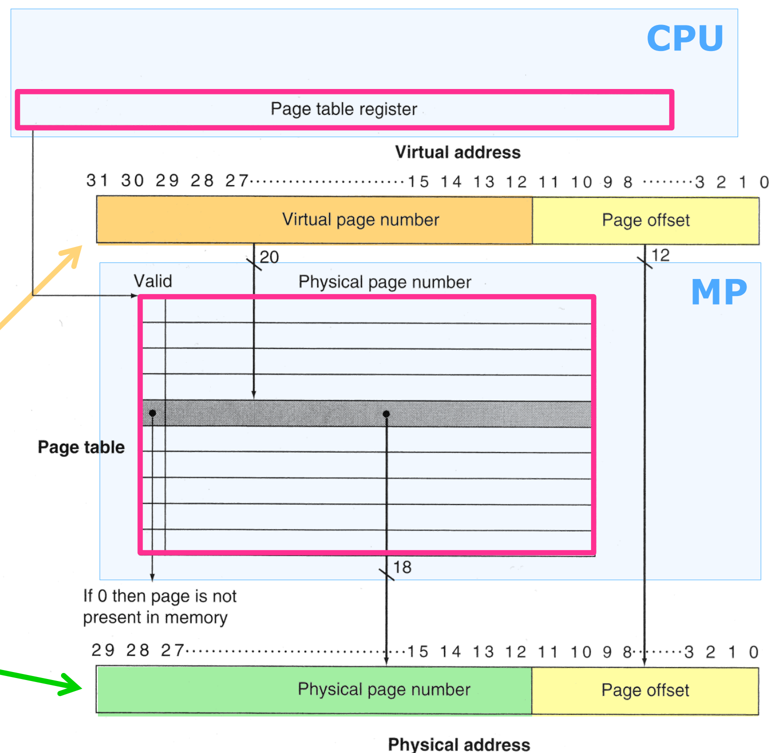
Pagine da: **4 kB** (2^{12})

→ memoria fisica suddivisa in:
2^{18} pagine da **2^{12} byte**
2^{18} pagine da **2^{10} word**

Indirizzo **virtuale (32 bit):**
– **N. pag. virtuale (20 bit)**
– Offset (12 bit)



Indirizzo **fisico(30 bit):**
– **N. pag. fisica (18 bit)**
– Offset (12 bit)



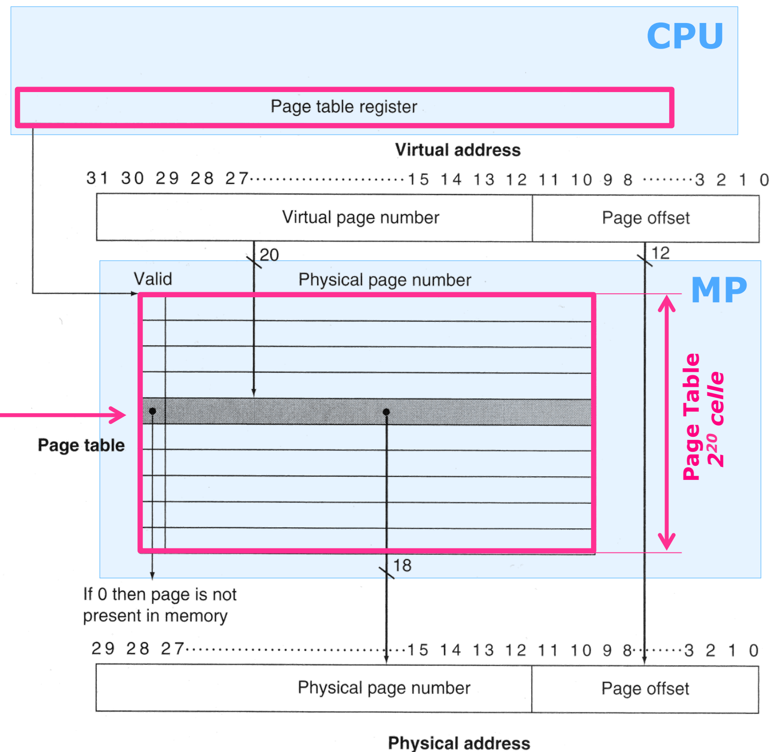
Virtual page number = indirizzo in page table

- 20 bit ➔ **1 Mcelle da 18 bit**

Indirizzo in Page Table:

- **$PTR + VPNumber**

A = $PTR + VPNumber



Gestione memoria virtuale:

Ogni **linea** della **Page Table** contiene un **VALID** bit:

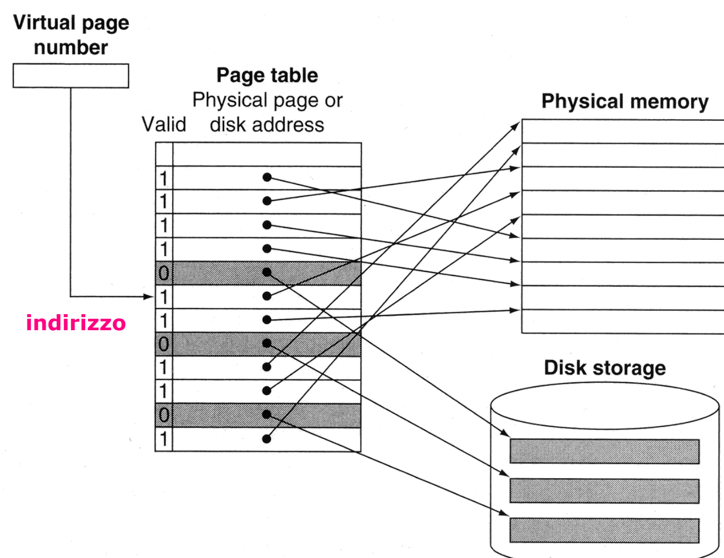
VALID=1 ➔ pagina presente in RAM
Page table contiene l'indirizzo fisico

VALID=0 ➔ pag. non presente in RAM
Page table contiene l'indirizzo in memoria di massa

➔ devo fare un **page swap**

Page swap è molto pesante!

➔ opportune strategie sofisticate di riscrittura (**LRU**)



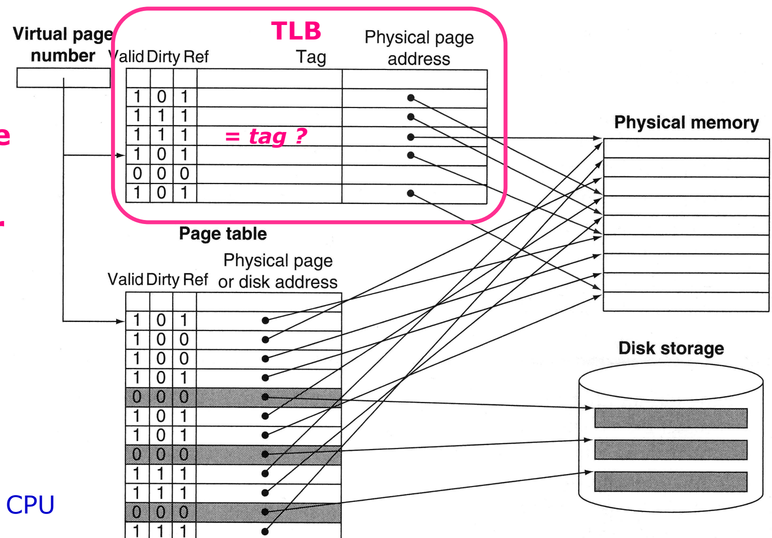
Memoria Virtuale

Memoria virtuale – struttura ottimizzata:
Translation Look-aside Buffer (TLB)

Con la **page table** in memoria principale, ogni accesso a memoria è costituito almeno da due accessi:
- 1 accesso per leggere la Page Table
- 1 accesso alla cella richiesta dalla CPU

Soluzione: **cache** per la **page table**

TLB:
Translation Look-aside Buffer



Meccanismo di accesso:

1. **Accesso a TLB:**
   - **HIT**: ottengo indirizzo fisico
   - **MISS**: leggo la Page Table e aggiorno la TLB
2. **Accesso alla cella** richiesta dalla CPU in RAM / HardDisk

Memoria Virtuale: modo protetto

Memoria virtuale: implementazione della protezione

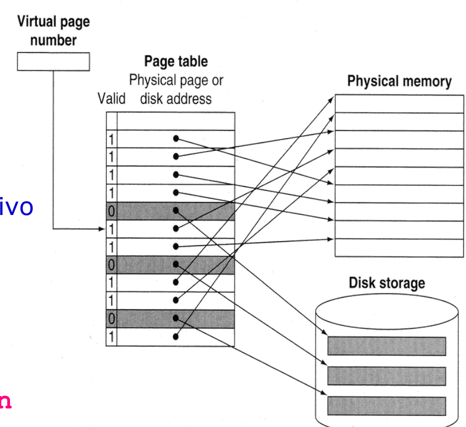
Ogni processo ha la propria memoria virtuale (privata)
➔ possiede una **propria Page Table** in memoria...
➔ ...puntata dal proprio valore, posto nel **Page Table Register**

Cambio di processo (context switch):
cambio valore nel PTR ➔ cambio **Page Table** ➔ cambio spazio memoria virtuale

Problema: se un utente scrive nel PTR, può accedere a dati/programmi di altri!

Soluzione:
➔ **accessi a PTR e Page Tables VIETATI all'utente**
mi serve una doppia modalità di funzionamento:
- Modo **UTENTE** (**User mode**), per tutti i processi degli utenti
- Modo **SUPERVISORE** (**Kernel mode**), **riservato** al sist. operativo

Alcune istruzioni (scrittura P.T.R.) e l'accesso ad alcune zone di memoria (page tables) permesse **soltanto in KERNEL mode** (quindi solo al S.O.)



User mode ➔ **Kernel** mode: `syscall`, che genera un'eccezione

Kernel mode ➔ **User** mode: `ERET: return from exception`