SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Architettura degli Elaboratori
17 – la Cache

Prof. Andrea Sterbini – sterbini@di.uniroma1.it

Argomenti

- **Argomenti della lezione**
  - Introduzione alla cache
  - Esercizio sulla cache

Argomenti

- **Argomenti della lezione**
  - Introduzione alla cache
  - Esercizio sulla cache

- **Problema:** la RAM è **MOLTO PIU' LENTA** del processore (circa 10-100 volte più lenta)
- **NON soluzione:** usare una memoria più veloce (ma **MOLTO PIU' COSTOSA**)

Argomenti

- **Argomenti della lezione**
  - Introduzione alla cache
  - Esercizio sulla cache

- **Problema:** la RAM è **MOLTO PIU' LENTA** del processore (circa 10-100 volte più lenta)
- **NON soluzione:** usare una memoria più veloce (ma **MOLTO PIU' COSTOSA**)

- **Principio di località temporale**
- «un programma tende ad accedere allo stesso elemento in momenti vicini tra loro»

Argomenti

- **Argomenti della lezione**
  - Introduzione alla cache
  - Esercizio sulla cache

- **Problema:** la RAM è **MOLTO PIU' LENTA** del processore (circa 10-100 volte più lenta)
- **NON soluzione:** usare una memoria più veloce (ma **MOLTO PIU' COSTOSA**)

- **Principio di località temporale**
- «un programma tende ad accedere allo stesso elemento in momenti vicini tra loro»
- **Principio di località spaziale**
- «un programma tende ad accedere successivamente elementi di memoria vicini tra loro»

Argomenti

- **Argomenti della lezione**
  - Introduzione alla cache
  - Esercizio sulla cache

- **Problema:** la RAM è **MOLTO PIU' LENTA** del processore (circa 10-100 volte più lenta)
- **NON soluzione:** usare una memoria più veloce (ma **MOLTO PIU' COSTOSA**)

- **Principio di località temporale**
- «un programma tende ad accedere allo stesso elemento in momenti vicini tra loro»
- **Principio di località spaziale**
- «un programma tende ad accedere successivamente elementi di memoria vicini tra loro»

- **IDEA 1:** memorizziamo in una **piccola memoria veloce** (CACHE) **solo i dati più usati**

Argomenti

- **Argomenti della lezione**
  - Introduzione alla cache
  - Esercizio sulla cache

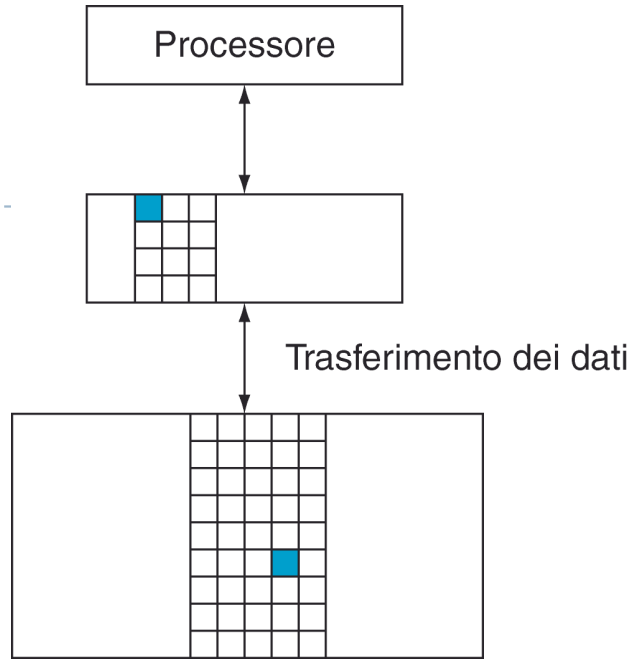
- **Problema:** la RAM è **MOLTO PIU' LENTA** del processore (circa 10-100 volte più lenta)
- **NON soluzione:** usare una memoria più veloce (ma **MOLTO PIU' COSTOSA**)

- **Principio di località temporale**
- «un programma tende ad accedere allo stesso elemento in momenti vicini tra loro»
- **Principio di località spaziale**
- «un programma tende ad accedere successivamente elementi di memoria vicini tra loro»

- **IDEA 1:** memorizziamo in una **piccola memoria veloce** (CACHE) **solo i dati più usati**
- **IDEA 2:** quando memorizziamo un dato **memorizziamo anche i dati vicini**

Argomenti

- **Argomenti della lezione**
  - Introduzione alla cache
  - Esercizio sulla cache



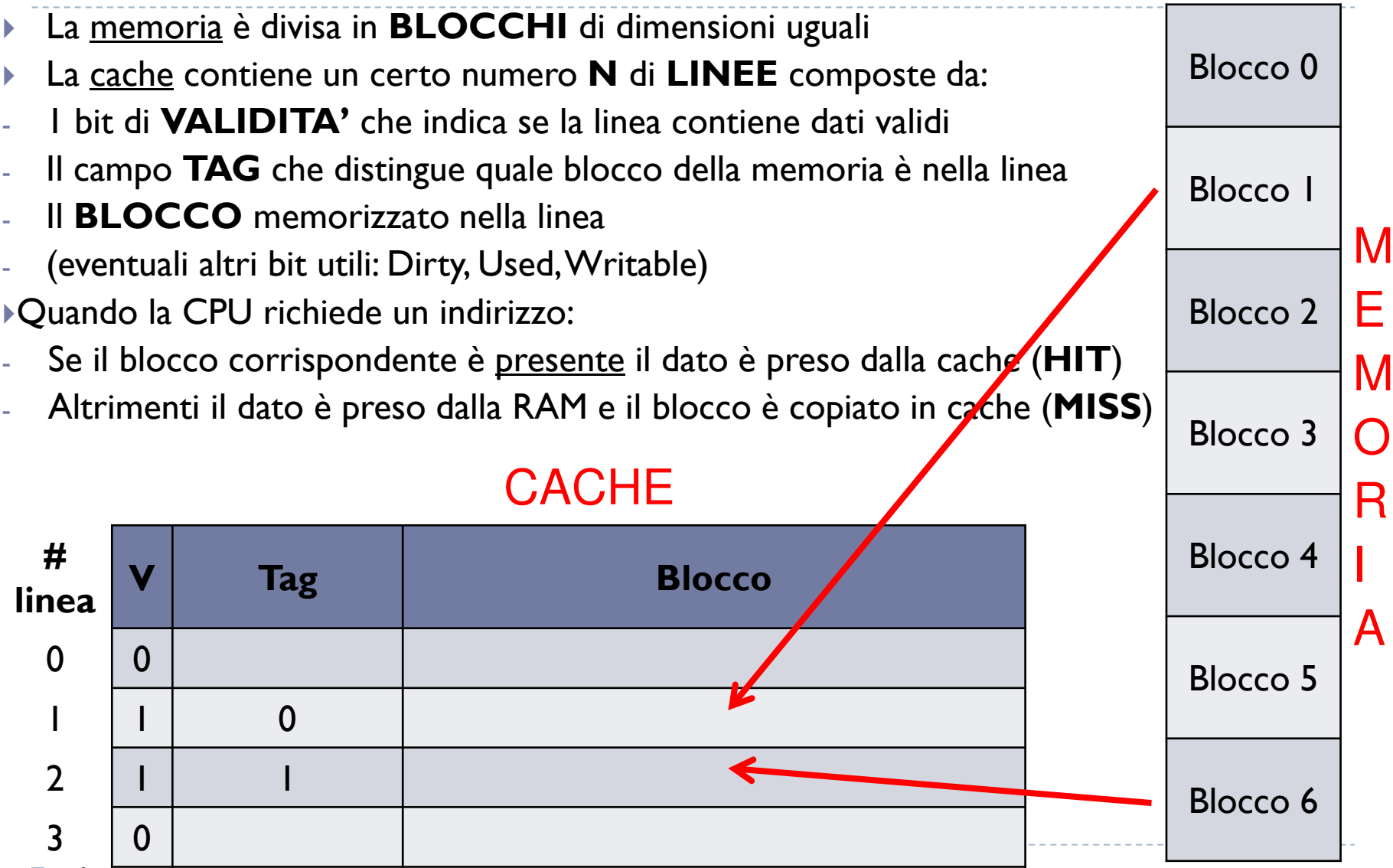
- **Problema:** la RAM è **MOLTO PIU' LENTA** del processore (circa 10-100 volte più lenta)
- **NON soluzione:** usare una memoria più veloce (ma **MOLTO PIU' COSTOSA**)

- **Principio di località temporale**
- «un programma tende ad accedere allo stesso elemento in momenti vicini tra loro»
- **Principio di località spaziale**
- «un programma tende ad accedere successivamente elementi di memoria vicini tra loro»

- **IDEA 1:** memorizziamo in una **piccola memoria veloce** (CACHE) **solo i dati più usati**
- **IDEA 2:** quando memorizziamo un dato **memorizziamo anche i dati vicini**

Schema e terminologia

- La memoria è divisa in **BLOCCHI** di dimensioni uguali
- La cache contiene un certo numero **N** di **LINEE** composte da:
  - 1 bit di **VALIDITA'** che indica se la linea contiene dati validi
  - Il campo **TAG** che distingue quale blocco della memoria è nella linea
  - Il **BLOCCO** memorizzato nella linea
  - (eventuali altri bit utili: Dirty, Used, Writable)
- Quando la CPU richiede un indirizzo:
  - Se il blocco corrispondente è presente il dato è preso dalla cache (**HIT**)
  - Altrimenti il dato è preso dalla RAM e il blocco è copiato in cache (**MISS**)



Performance

- Supponiamo di avere un programma che fa **1.000.000** di accessi in memoria
- e che il tempo di accesso sia di **100 ns**

- Il tempo totale che impiegherà per l'accesso è 1.000.000 * 100ns = **100 ms**

- Se si usa una **cache** con tempo di accesso **1 ns** e la percentuale di miss è il **10%**

Performance

- Supponiamo di avere un programma che fa **1.000.000** di accessi in memoria
- e che il tempo di accesso sia di **100 ns**

- Il tempo totale che impiegherà per l'accesso è 1.000.000 * 100ns = **100 ms**

- Se si usa una **cache** con tempo di accesso **1 ns** e la percentuale di miss è il **10%**
- il 90% di 1.000.000 accessi (**HIT**) impiegano 1ns*1.000.000 * 90% = **0.9 ms**
- il 10% rimanente (**MISS**) impiegheranno 100ns*1.000.000 * 10% = **10 ms**

Performance

- Supponiamo di avere un programma che fa **1.000.000** di accessi in memoria
- e che il tempo di accesso sia di **100 ns**

- Il tempo totale che impiegherà per l'accesso è 1.000.000 * 100ns = **100 ms**

- Se si usa una **cache** con tempo di accesso **1 ns** e la percentuale di miss è il **10%**
  - il 90% di 1.000.000 accessi (**HIT**) impiegano 1ns*1.000.000 * 90% = **0.9 ms**
  - il 10% rimanente (**MISS**) impiegheranno 100ns*1.000.000 * 10% = **10 ms**

- In totale il tempo di accesso medio sarà 10ms + 0.9ms/1.000.000 = **10.9 ns**
- Con un aumento di velocità di 100ns / 10.9ns = **9 volte circa**

Performance

- Supponiamo di avere un programma che fa **1.000.000** di accessi in memoria
- e che il tempo di accesso sia di **100 ns**

- Il tempo totale che impiegherà per l'accesso è 1.000.000 * 100ns = **100 ms**

- Se si usa una **cache** con tempo di accesso **1 ns** e la percentuale di miss è il **10%**
  - il 90% di 1.000.000 accessi (**HIT**) impiegano 1ns*1.000.000 * 90% = **0.9 ms**
  - il 10% rimanente (**MISS**) impiegheranno 100ns*1.000.000 * 10% = **10 ms**

- In totale il tempo di accesso medio sarà 10ms + 0.9ms/1.000.000 = **10.9 ns**
- Con un aumento di velocità di 100ns / 10.9ns = **9 volte circa**
- **Esempio: il processore FastMath**

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

Direct mapping

- Il blocco va nella linea data dal numero di blocco modulo il numero di linee disponibili
- (ogni N blocchi lo schema si ripete)

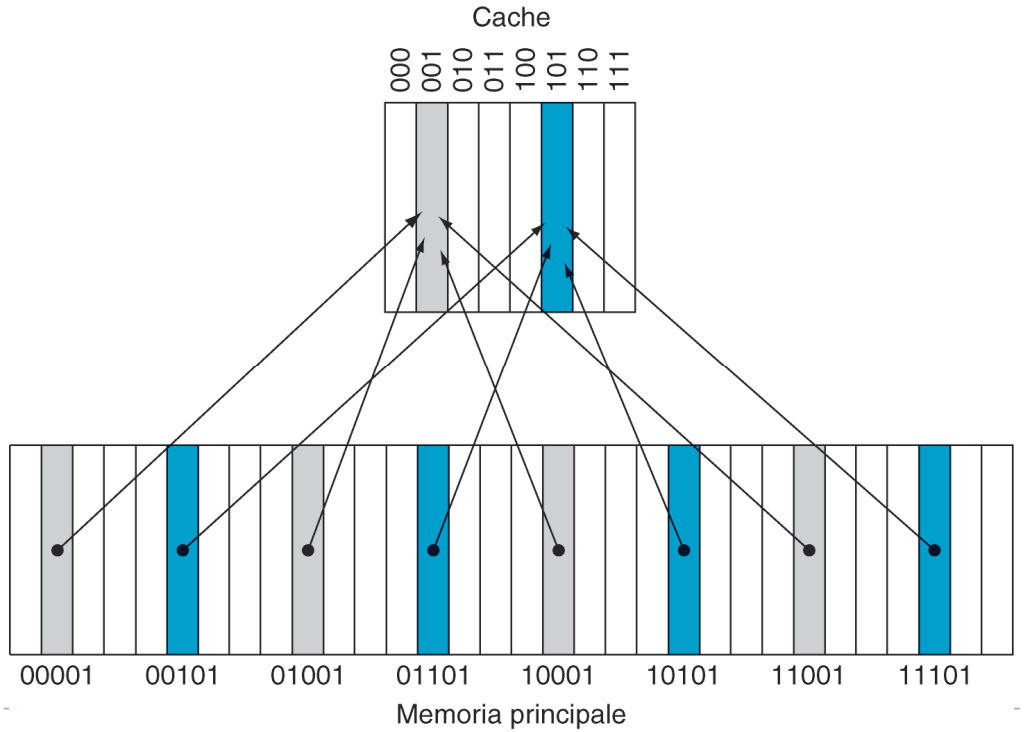
- **Esempio:**
- Cache direct mapped con:
- Blocco di 4 word = 16 byte
- 8 linee
- Indirizzo: **2157**

  # di blocco = 2157 / 16 = **134**

  Offset = 2157 % 16 = **13**

  # di linea = 134 % 8 = **6**

  Tag = 134 / 8 = **16**



Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**
- Sequenza di accessi:
- #blocco → tag indice

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110

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

MISS

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010

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

MISS

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000

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

MISS

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000
- 00011 → 00 011

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000
- 00011 → 00 011

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000
- 00011 → 00 011

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

MISS

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000
- 00011 → 00 011

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000
- 00011 → 00 011
- 10010 → 10 010

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000
- 00011 → 00 011
- 10010 → 10 010

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000
- 00011 → 00 011
- 10010 → 10 010

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000
- 00011 → 00 011
- 10010 → 10 010

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

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

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000
- 00011 → 00 011
- 10010 → 10 010
- 10110 → 10 110

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000
- 00011 → 00 011
- 10010 → 10 010
- 10110 → 10 110

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000
- 00011 → 00 011
- 10010 → 10 010
- 10110 → 10 110

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

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

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

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000
- 00011 → 00 011
- 10010 → 10 010
- 10110 → 10 110
- 10000 → 10 000

Esempio di sequenza di accessi

- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000
- 00011 → 00 011
- 10010 → 10 010
- 10110 → 10 110
- 10000 → 10 000

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

Esempio di sequenza di accessi

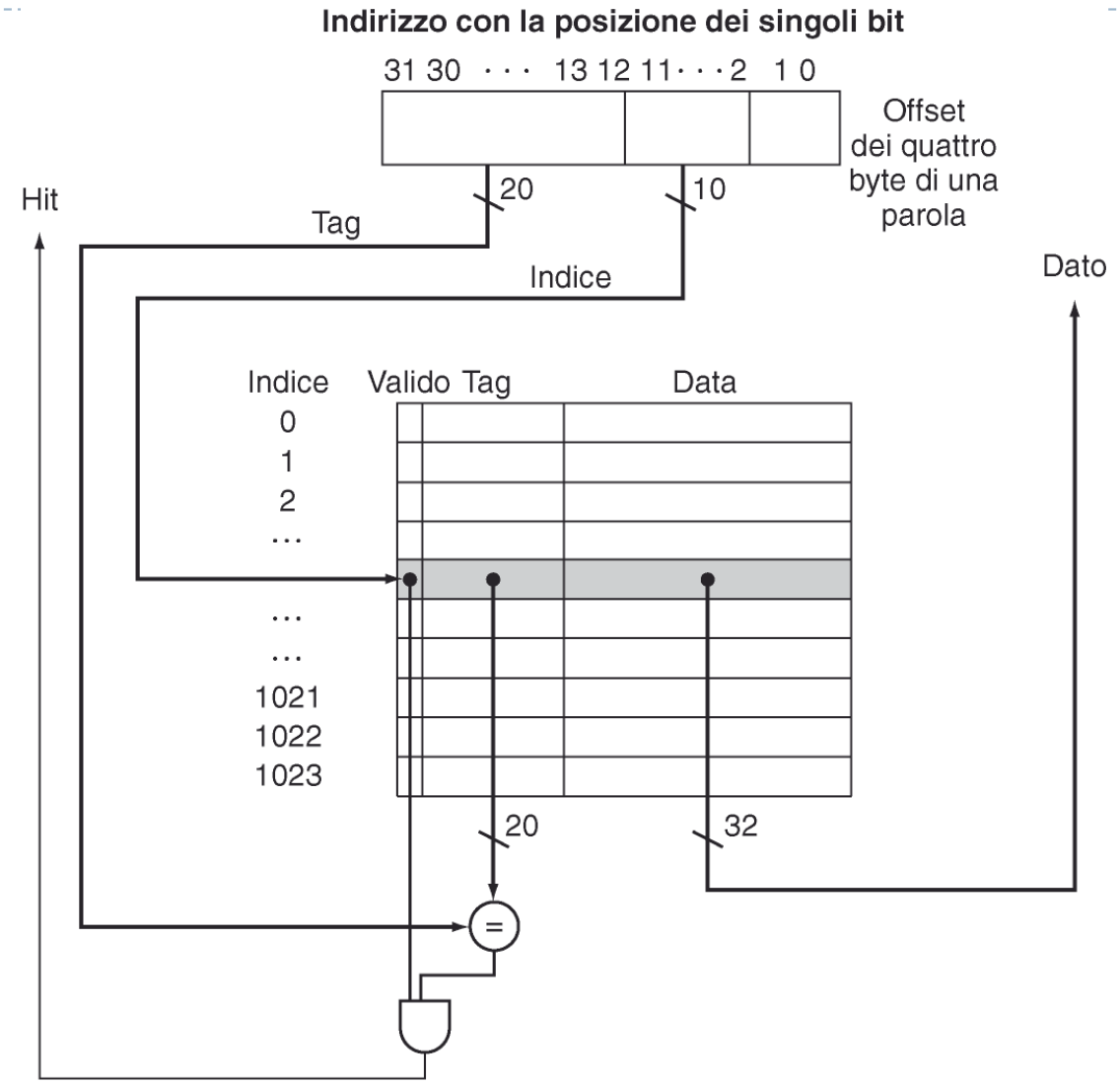
- Cache **direct mapped**
- Con **8 linee**

- Sequenza di accessi:
- #blocco → tag indice
- 10110 → 10 110
- 11010 → 11 010
- 10000 → 10 000
- 00011 → 00 011
- 10010 → 10 010
- 10110 → 10 110
- 10000 → 10 000

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

Come determinare HIT/MISS

- Cache con:
- blocchi da 1 word
- 1024 linee



Altri tipi di mapping

- Come migliorare la percentuale di HIT?
- **direct mapping:** ogni blocco viene messo in una linea fissata (che dipende da #blocco)

  PRO: hardware più semplice e meno costoso

  è facile determinare in quale linea cercare il dato
  #linea = #blocco % N

  CON: blocchi diversi mappano nella stessa linea, se gli accessi a questi blocchi si alternano si ottengono molti MISS (**trashing**)

Altri tipi di mapping

- Come migliorare la percentuale di HIT?
- **direct mapping:** ogni blocco viene messo in una linea fissata (che dipende da #blocco)

  PRO: hardware più semplice e meno costoso

  è facile determinare in quale linea cercare il dato
  #linea = #blocco % N

  CON: blocchi diversi mappano nella stessa linea, se gli accessi a questi blocchi si alternano si ottengono molti MISS (**trashing**)

- **mapping fully-associative:** un blocco può essere messo in una linea qualsiasi

  PRO: massima flessibilità

  CON: hardware più complesso e più costoso

Altri tipi di mapping

- Come migliorare la percentuale di HIT?
- **direct mapping:** ogni blocco viene messo in una linea fissata (che dipende da #blocco)

  PRO: hardware più semplice e meno costoso

  è facile determinare in quale linea cercare il dato
  #linea = #blocco % N

  CON: blocchi diversi mappano nella stessa linea, se gli accessi a questi blocchi si alternano si ottengono molti MISS (**trashing**)

- **mapping fully-associative:** un blocco può essere messo in una linea qualsiasi

  PRO: massima flessibilità

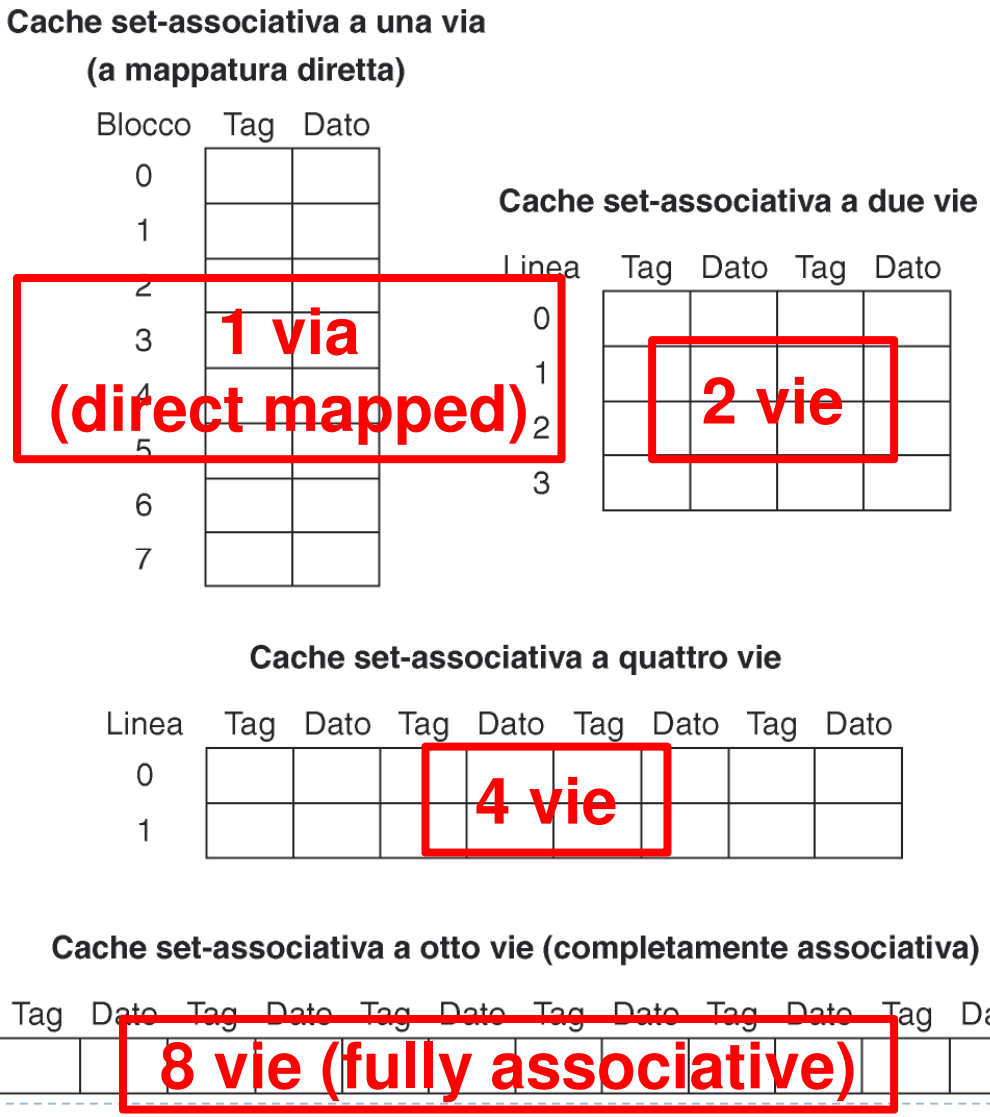
  CON: hardware più complesso e più costoso

- **mapping set-associativo:** una via di mezzo, le linee sono suddivise in **S** gruppi (set) formati ciascuno da **W** elementi (vie) e ogni blocco è disposto in una qualsiasi delle linee del set fissato (che dipende da #blocco)

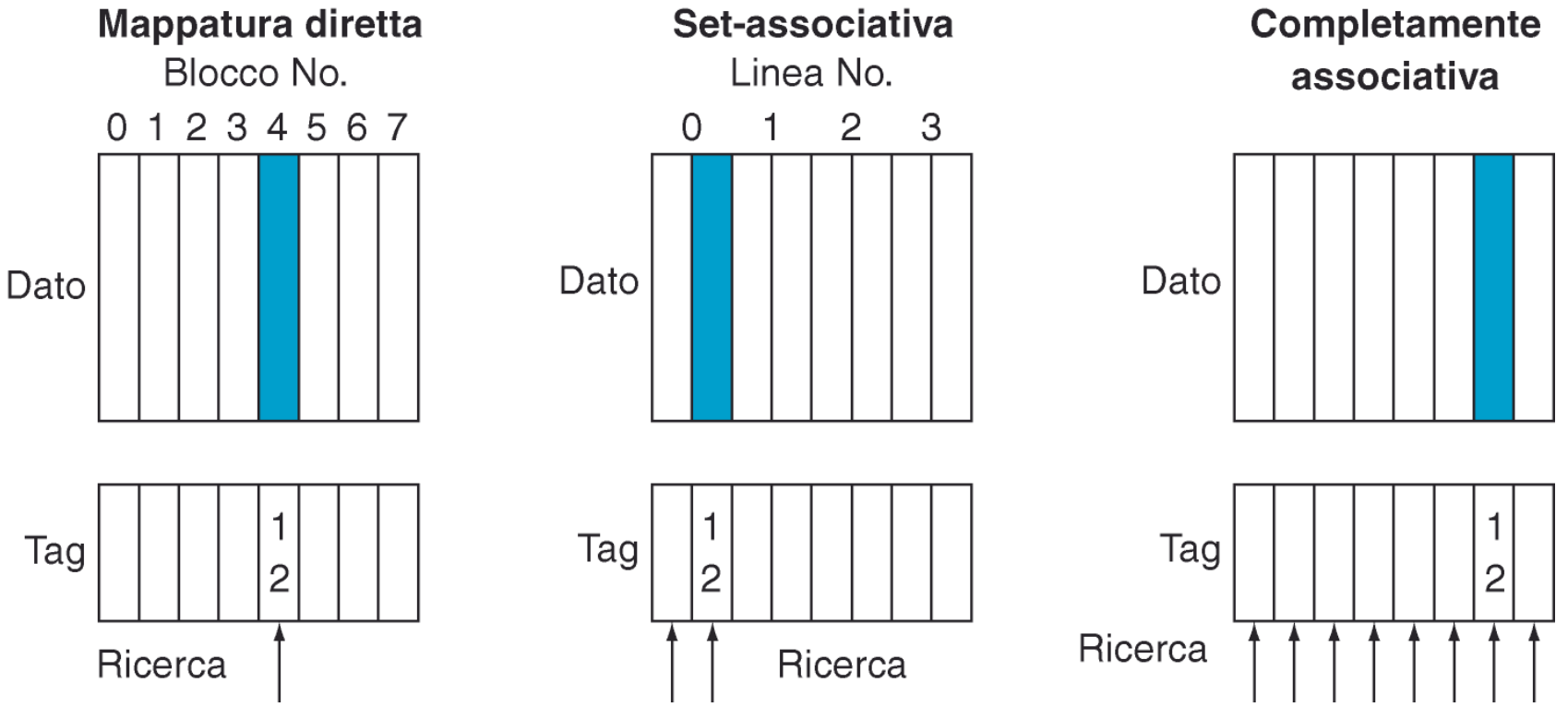
  #set = #blocco % S

Associatività crescente

Cache con **8 linee** organizzate con diversi gradi di associatività



Livelli di associatività

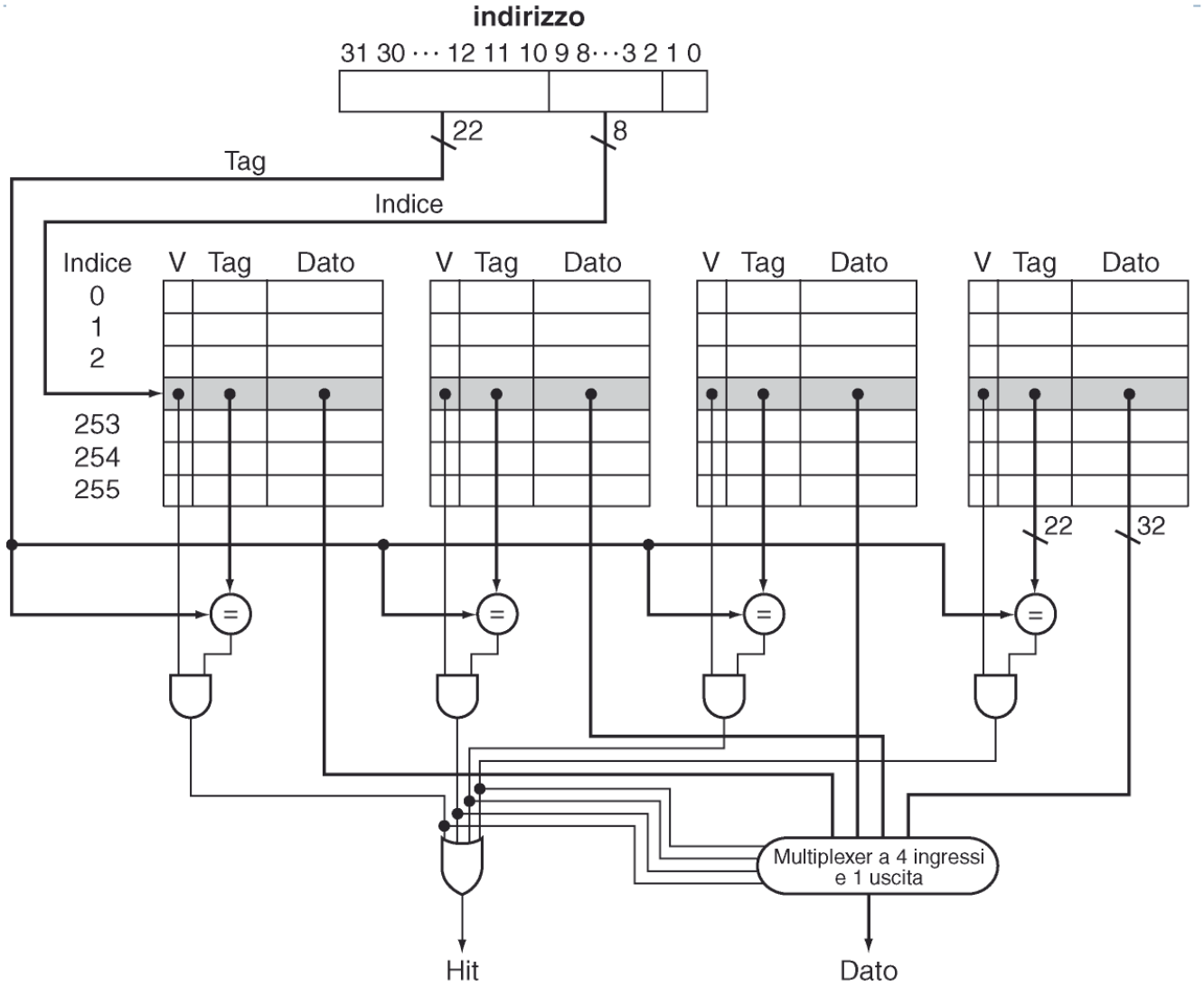


Associatività	Frequenza di miss
1	10,3%
2	8,6%
4	8,3%
8	8,1%

Cache set-associativa a 4 vie

E' necessario un comparatore per ciascuna via

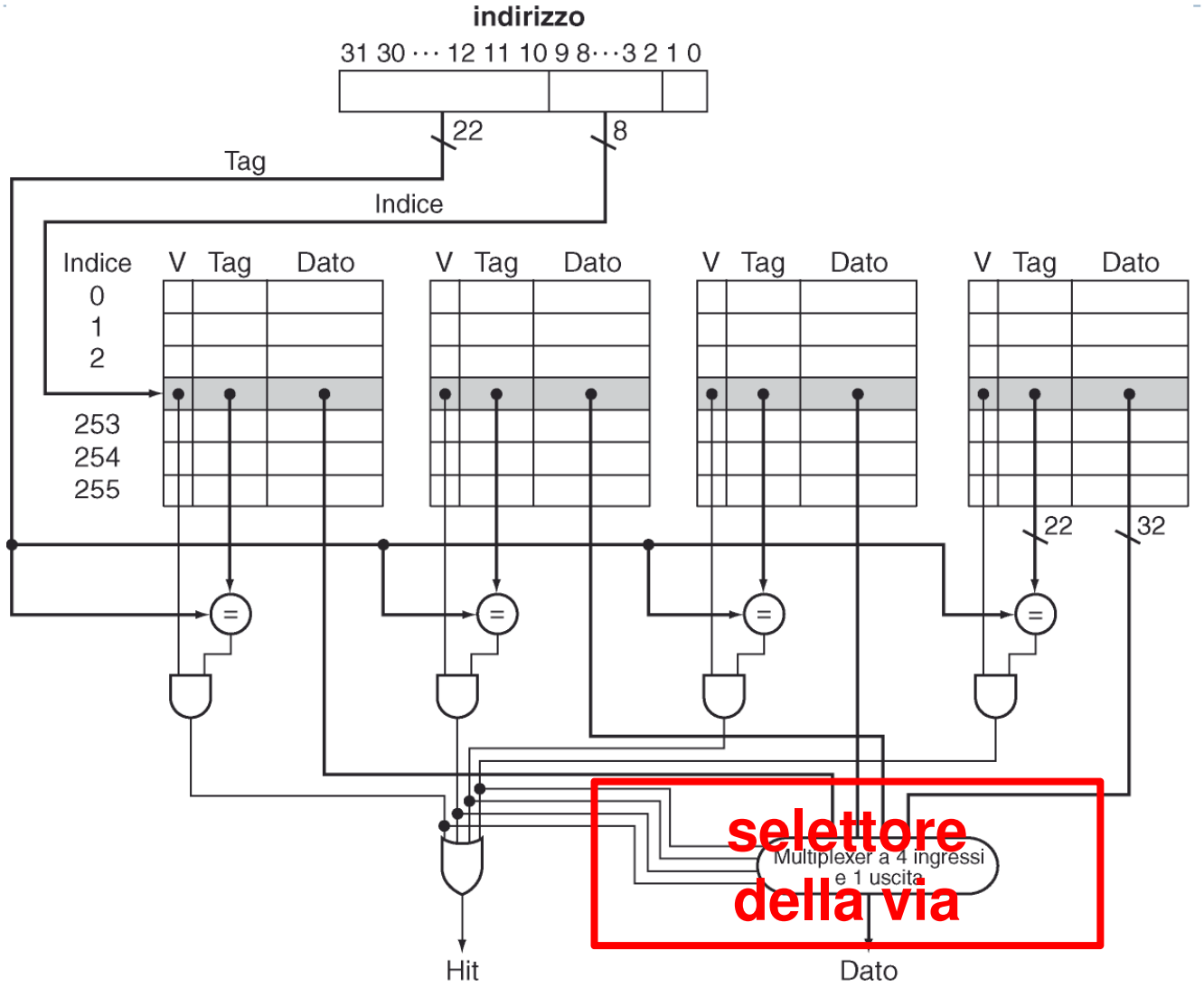
fully-associative: tanti comparatori quante sono le linee



Cache set-associativa a 4 vie

E' necessario un comparatore per ciascuna via

fully-associative: tanti comparatori quante sono le linee



Quanto è grande una cache?

- Per ogni **via** abbiamo una tabella
- ogni tabella contiene **S** linee (tante quanto gli **insiemi**)
  - ogni linea contiene:
    - il bit di **validità** (ed eventualmente i bit **Used** e **Dirty**)
    - il **tag**: 32 bit – bit per offset – bit per indice di set
    - il **blocco**: 8 * numero di byte
- I bit necessari per l'**offset** sono log2(dimensione blocco) e per l'**indice** log2(# insiemi)

- **Esempio:** cache a **4 vie**, con **8 insiemi** e **blocchi da 4 word**
  - dimensione blocco = 4 word * 4 = 16 byte = 16*8 bit = **128 bit**
  - numero di bit per **offset** = log2(16) = **4 bit**
  - numero di bit per **indice** = log2(8) = **3 bit**
  - numero di bit per **tag** = 32 – 3 – 4 = **25 bit**
- **Dim .totale in bit:**
  - 4 vie * 8 insiemi * (1 validità + 128 blocco + 25 tag) = 32*154 = **4928 bit**

Come rimpiazzare un blocco?

- **Come scegliere** quale blocco sostituire quando un set della cache è pieno?
- **Politica di rimpiazzo:**

Come rimpiazzare un blocco?

- **Come scegliere** quale blocco sostituire quando un set della cache è pieno?

- **Politica di rimpiazzo:**

- **LRU**: (Least Recently Used) sostituire il blocco **«più vecchio»**
  - Approssimazione: si associa un bit (**Used**) a ciascuna linea e lo si azzera ogni tot nanosecondi
  - Ad ogni accesso si mette Used = 1

    → le linee che hanno **Used=1** sono **«più recenti»**
    e quelle con **Used=0** sono **«più vecchie»**

Come rimpiazzare un blocco?

- **Come scegliere** quale blocco sostituire quando un set della cache è pieno?

- **Politica di rimpiazzo:**

- **LRU**: (Least Recently Used) sostituire il blocco **«più vecchio»**
  - Approssimazione: si associa un bit (**Used**) a ciascuna linea e lo si azzera ogni tot nanosecondi
  - Ad ogni accesso si mette Used = 1

    → le linee che hanno **Used=1** sono **«più recenti»**
    e quelle con **Used=0** sono **«più vecchie»**
- **LFU**: (Least Frequently Used) sostituire il blocco **«meno usato»**
  - Si può associare un contatore a ciascuna linea ed aggiornarlo ad ogni accesso

Come rimpiazzare un blocco?

- **Come scegliere** quale blocco sostituire quando un set della cache è pieno?

- **Politica di rimpiazzo:**

- **LRU**: (Least Recently Used) sostituire il blocco **«più vecchio»**
  - Approssimazione: si associa un bit (**Used**) a ciascuna linea e lo si azzera ogni tot nanosecondi
  - Ad ogni accesso si mette Used = 1

    → le linee che hanno **Used=1** sono **«più recenti»**
    e quelle con **Used=0** sono **«più vecchie»**
- **LFU**: (Least Frequently Used) sostituire il blocco **«meno usato»**
  - Si può associare un contatore a ciascuna linea ed aggiornarlo ad ogni accesso

- **RANDOM**: sostituire un blocco **«a caso»** (per esempio in sequenza)

Tipi di MISS

- Un accesso alla cache può dare MISS per tre motivi: (in una cache con **W** ways e **S** sets)
- **Cold start**: è la prima volta che quel blocco viene richiesto, va caricato per forza
- **Conflict:** il blocco è stato sovrascritto per via del grado di associatività della cache, e sarebbe una HIT se la cache fosse fully-associative
  - se il blocco è stato richiesto meno W*S volte prima allora in una fully-associative → HIT
- **Capacity:** il blocco è stato sostituito ma non sarebbe stato possibile avere una HIT nemmeno se la cache fosse stata fully-associative
  - il blocco è stato richiesto più W*S volte prima allora in una fully-associative → MISS

Tipi di MISS

- Un accesso alla cache può dare MISS per tre motivi: (in una cache con **W** ways e **S** sets)
  - **Cold start**: è la prima volta che quel blocco viene richiesto, va caricato per forza
  - **Conflict:** il blocco è stato sovrascritto per via del grado di associatività della cache, e sarebbe una HIT se la cache fosse fully-associative
    - se il blocco è stato richiesto meno W*S volte prima allora in una fully-associative → HIT
  - **Capacity:** il blocco è stato sostituito ma non sarebbe stato possibile avere una HIT nemmeno se la cache fosse stata fully-associative
    - il blocco è stato richiesto più W*S volte prima allora in una fully-associative → MISS

- Quali parametri della cache hanno influenza sui tre tipi di MISS?
  - **Cold:** la **dimensione del blocco** (blocchi più grandi fonderanno il caricamento di più blocchi piccoli, diminuendo il numero di cold miss)
  - **Conflict:** il **grado di associatività** (una cache con più vie ha meno conflitti)
  - **Capacity:** la **dimensione della cache** (un maggior numero di linee permette di rispondere a un maggior numero di richieste)

Quando aggiornare la memoria?

- Quando un dato viene modificato in cache la memoria sottostante deve essere aggiornata

- **Politiche di scrittura**

- **Write Through:** **ad ogni modifica** viene aggiornato il blocco in memoria
  - **Pro:** in presenza di cache multiple la coerenza dei dati viene mantenuta
  - **Contro:** per la località degli accessi se avvengono più scritture nello stesso blocco si perde molto tempo (mitigabile con un buffer di scrittura)
- **Write Back:** il blocco viene aggiornato **solo quando viene sostituito**
  - **Pro:** la scrittura del blocco in memoria avviene raramente (è un blocco «vecchio») quindi la cache è molto più veloce
  - **Contro:** il contenuto della cache non è più coerente con quello della RAM (complicando i sistemi multiprocessore e multicache)

- **Ottimizzazione:** con il bit **Dirty** (modificato) si può evitare di scrivere i blocchi non modificati

Esercizio

- Per una cache con:
- blocchi da **8 word**
- **4 vie**
- **2 insiemi** per via
- strategia di rimpiazzo **LRU**

1) Calcolate la **dimensione in bit** della cache
2) Calcolate nella sequenza di accessi qui sotto **quali sono HIT e quali sono MISS**

address	130	310	1800	150	519	342	1820	127	529	2000	325	516	1794	317
#blocco														
indice linea														
tag														
Hit/Miss														