Architettura degli Elaboratori

La cache – introduzione

SAPIENZA
UNIVERSITÀ DI ROMA

Alessandro Checco

checco@di.uniroma1.it

Special thanks and credits:

Andrea Sterbini, Iacopo Masi,
Claudio di Ciccio

[S&PdC]
5.1, 5.3 – 5.4

Esercizio per casa

SAPIENZA
UNIVERSITÀ DI ROMA

Esercizio per casa

Si consideri l'architettura MIPS con pipeline [...]. Il programma a fianco effettua le operazioni di somma o sottrazione indicate nella stringa fornita in input.

Si supponga che *tutte le istruzioni impiegate fanno parte del set supportato dalla CPU in figura*, ossia non si fa uso di alcuna pseudoistruzione.

Si indichino (ignorando hazard che possano concernere la syscall):

1) [...]

2) [...]

3) [...]

4) quanti **cicli di clock** sarebbero necessari per eseguire il programma **senza forwarding**, spiegando il calcolo effettuato;

5) [...]

```
1   .globl main
2
3   .data
4   W: .asciiz "+6-5+4+2-8"
5
6   .text
7   main:
8         or   $v0, $zero, $zero # Init: risultato
9         and  $t0, $t0, $zero   # Init: $t0
10        and  $t1, $t0, $t0     # Init: $t1
11        la   $a0, W            # Indirizzo stringa
12  Cycle: lb  $t1, 0($a0)       # Segno in $t1
13        beq  $t1, $zero, Exit  # Fine stringa? → esci
14        lb   $t0, 1($a0)       # Simbolo num. in $t0
15        subi $t0, $t0, '0'     # Val. assoluto num.
16        bne  $t1, '-', Add     # Non '-'? → somma
17        sub  $t0, $zero, $t0   # Else → v. opposto
18  Add:  add  $v0, $v0, $t0     # Addizione in $v0
19        addi $a0, $a0, 2       # Avanza di 2 char.
20        j    Cycle             # Torna a inizio ciclo
21  Exit: move $a0, $v0          # Risultato in $a0
22        li   $v0, 1            # Imposta stampa num.
23        syscall                # Stampa
24        li   $v0, 10           # Imposta terminazione
25        syscall                # Termina
```

Esecuzione senza forwarding (ingresso e prima iterazione)

	W/o forwarding							+6																	
7	main:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
8	or $v0, $zero, $zero	F	D	X	M	W																				
9	and **$t0**, $t0, $zero		F	D	X	M	**W**																			
10	and $t1, **$t0**, **$t0**			>	>	F	**D**	X	M	W																
11	la **$a0**, W						F	D	X	M	**W**															
12	Cycle: lb **$t1**, 0(**$a0**)							>	>	F	**D**	X	M	**W**												
13	beg **$t1**, $zero, Exit										>	>	F	**D**	X	M	W									
14	lb **$t0**, 1($a0)													F	D	X	M	**W**								
15	sub $t0, **$t0**, '0'														>	>	F	**D**	X	M	W					
16	bne $t1, '-', Add																	F	**D**	X	M	W				
17	sub **$t0**, $zero, $t0																									
18	Add: add $v0, $v0, **$t0**																		>	**F**	D	X	M	W		
19	addi **$a0**, $a0, 2																				F	D	X	M	**W**	
20	j Cycle																					F	D	X	M	W
21	Exit: move $a0, $v0																									
22	li $v0, 1																									
23	syscall																									
24	li $v0, 10																									
25	syscall																									

	CC	I	DH	CH	×	=
	Caricamento pipeline	4				4
	Pre-ciclo	4				7
	In ciclo (+)	8	5	1	3	42

Esecuzione senza forwarding (seconda iterazione)

	W/o forwarding		-5																+4			
7	main:		22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
8		or $v0, $zero, $zero																					
9		and **$t0**, $t0, $zero																					
10		and $t1, **$t0**, **$t0**																					
11		la **$a0**, W																					
12	Cycle:	lb **$t1**, 0(**$a0**)	>	F	**D**	X	M	**W**												>	F	**D**	X
13		beq $t1, $zero, Exit			>	>	F	**D**	X	M	W											>	>
14		lb **$t0**, 1($a0)						F	D	X	M	**W**											
15		sub $t0, **$t0**, '0'							>	>	F	**D**	X	M	W								
16		bne $t1, '-', Add										F	**D**	X	M	W							
17		sub **$t0**, $zero, $t0											>	**F**	D	X	M	**W**					
18	Add:	add $v0, $v0, **$t0**	M	W											>	>	F	**D**	X	M	W		
19		addi **$a0**, $a0, 2	X	M	**W**													F	D	X	M	**W**	
20		j Cycle	D	X	M	W													F	D	X	M	W
21	Exit:	move $a0, $v0																					
22		li $v0, 1																					
23		syscall																					
24		li $v0, 10																					
25		syscall																					

	CC	I	DH	CH	×	=
	Caricamento pipeline	4				4
	Pre-ciclo	4	3			7
	In ciclo (+)	8	5	1	3	42
	In ciclo (-)	9	8	0	2	34

Split one stall with the next step because two stalls are necessary at instruction 12 only while entering the cycle. For every other iteration, one stall is sufficient

Esecuzione senza forwarding (uscita dal ciclo e termine)

	W/o forwarding	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98
7	main:																										
8	or $v0, $zero, $zero																										
9	and **$t0**, $t0, $zero																										
10	and $t1, **$t0**, **$t0**																										
11	la **$a0**, W																										
12	Cycle: lb **$t1**, 0(**$a0**)												>	F	**D**	X	M	**W**									
13	beg **$t1**, $zero, Exit	X	M	W											>	>	F	**D**	X	M	W						
14	lb **$t0**, 1($a0)	D	X	M	**W**																						
15	sub $t0, **$t0**, '0'	>	>	F	**D**	X	M	W																			
16	bne $t1, '-', Add				F	**D**	X	M	W																		
17	sub **$t0**, $zero, $t0					>	**F**	D	X	M	**W**																
18	Add: add $v0, $v0, **$t0**							>	>	F	**D**	X	M	W													
19	addi **$a0**, $a0, 2										F	D	X	M	**W**												
20	j Cycle											F	D	X	M	W											
21	Exit: move $a0, $v0																	>	**F**	D	X	M	W				
22	li $v0, 1																			F	D	X	M	W			
23	syscall																				F	D	X	M	W		
24	li $v0, 10																					F	D	X	M	W	
25	syscall																						F	D	X	M	W

	CC	I	DH	CH	×	=
	Caricamento pipeline	4				4
	Pre-ciclo	4	3			7
	In ciclo (+)	8	5	1	3	42
	In ciclo (-)	9	8	0	2	34
	Uscita ciclo	2	3	0		5
	Post-ciclo	5		1		6
	TOTALE					98

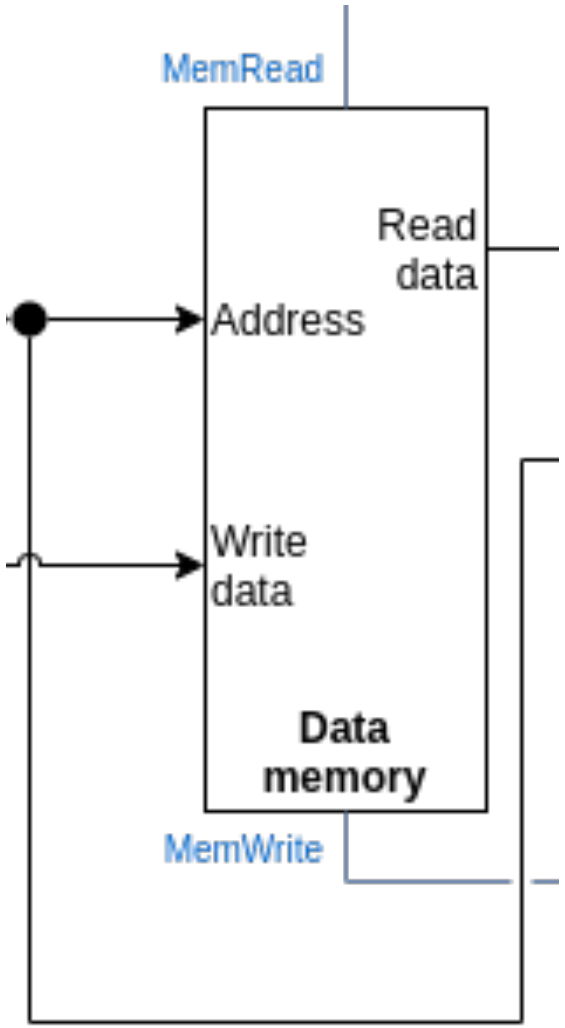
Split one stall with the next step because two stalls are necessary at instruction 12 only while entering the cycle. For every other iteration, one stall is sufficient

Caching della memoria principale

SAPIENZA
UNIVERSITÀ DI ROMA

Argomento

Cosa accade in fase di accesso a memoria?



Argomento

Cosa accade in fase di accesso a memoria?

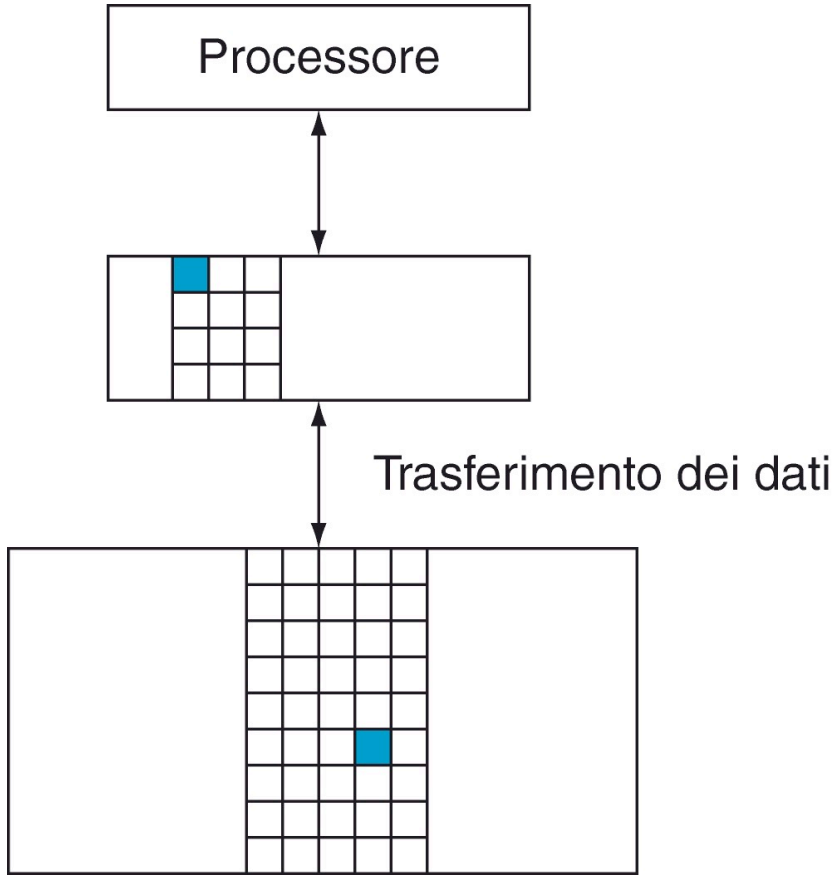
Problema:

la RAM è **molto più lenta** del processore (circa 10-100 volte più lenta)

Una memoria più veloce è **molto più costosa**

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

- **Idea 1:**
  - mettiamo da parte in una **piccola** memoria **veloce** (**cache**) solo i **dati più usati**
- **Idea 2:**
  - quando memorizziamo un dato **mettiamo da parte anche i dati vicini**



Argomento

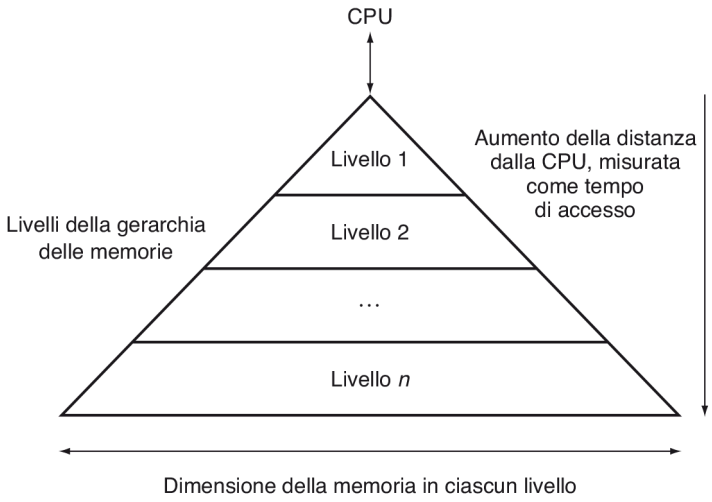
Problema:

la RAM è **molto più lenta** del processore (circa 10-100 volte più lenta)

Una memoria più veloce è **molto più costosa**

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

- **Idea 1:**
  - mettiamo da parte in una **piccola** memoria **veloce** (**cache**) solo i **dati più usati**
- **Idea 2:**
  - quando memorizziamo un dato **mettiamo da parte anche i dati vicini**



Schema e terminologia

La memoria è divisa in **blocchi** di dimensioni uguali

La cache contiene un numero **N** di **linee**:

Quando la CPU richiede un indirizzo:

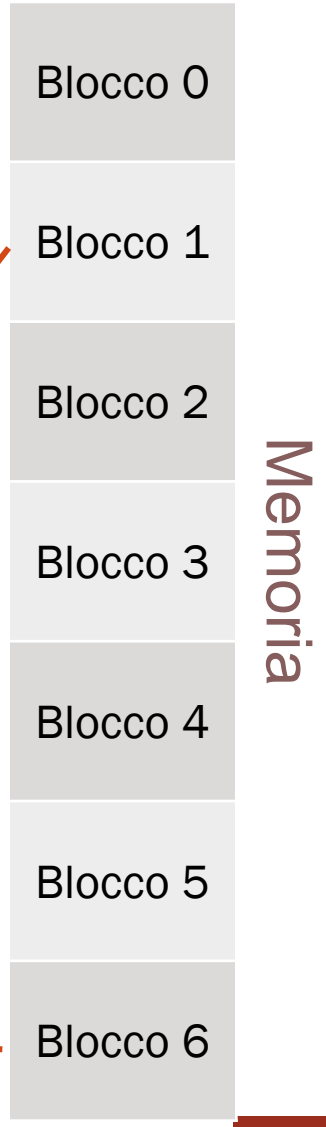
- Il blocco corrispondente è presente ⟶ il dato è caricato dalla cache (**HIT**)
- Altrimenti ⟶ dato è preso dalla RAM e il blocco è copiato in cache (**MISS**)

Le linee contengono:

- 1 bit di **validità**
  - indica se la linea contiene dati validi
- Il campo **tag**
  - distingue quale blocco della memoria è nella linea
- Il **blocco** memorizzato nella linea

Cache

# linea	V	Tag	Blocco
0	0		
1	1	0	
2	1	1	
3	0		



Performance

Supponiamo che un programma preveda **1 000 000** di accessi in memoria
e che il tempo di accesso sia di **100 ns**

Il tempo totale che il programma 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%**

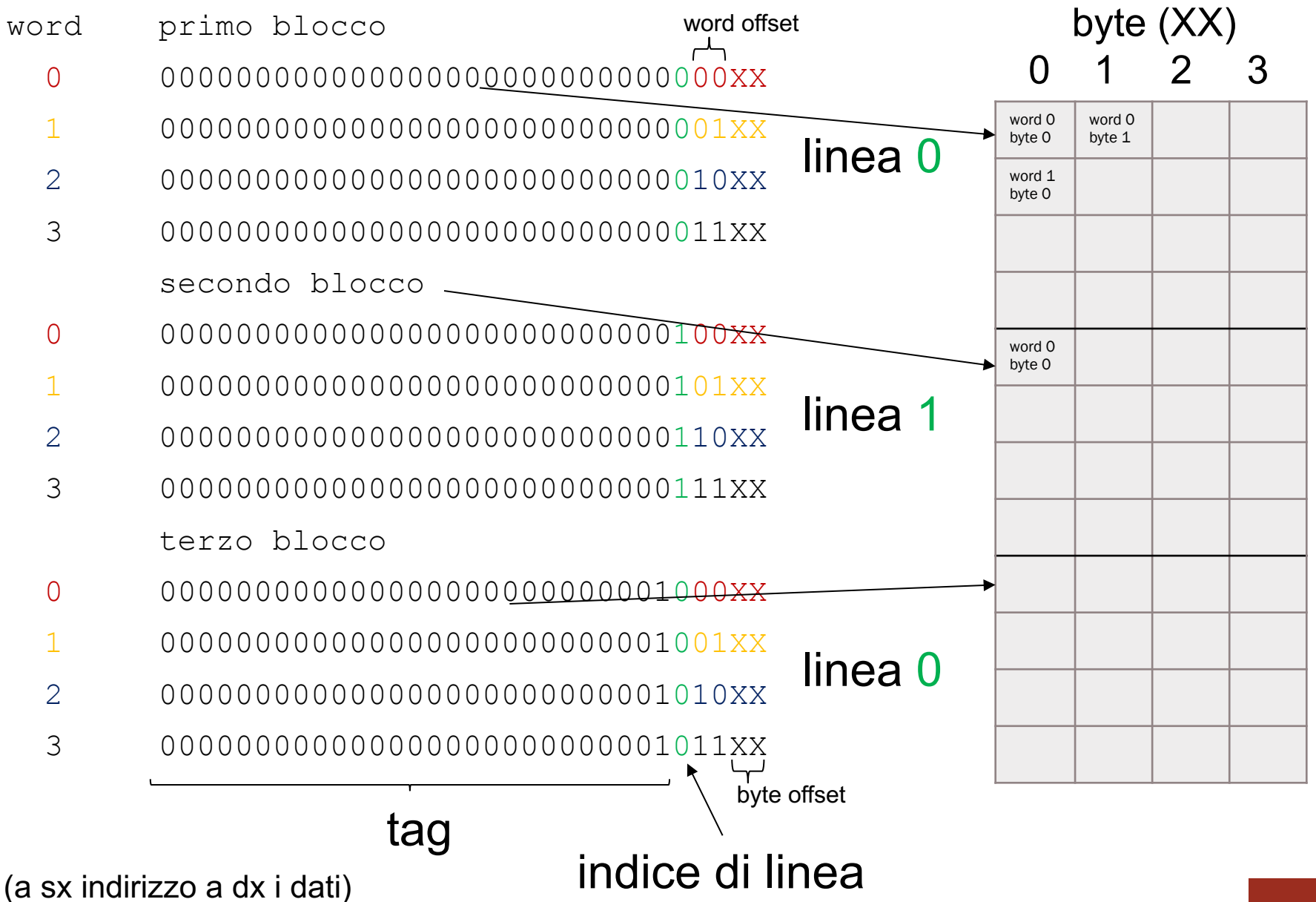
- il 90% di 1 000 000 accessi (**HIT**) impiegheranno 1 ns × 1 000 000 × 90% = **0.9 ms**
- il 10% rimanente (**MISS**) impiegheranno 100 ns × 1 000 000 × 10% = **10 ms**

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 - Blocchi di 4 word, cache di 2 linee



(a sx indirizzo a dx i dati)

Direct mapping

Numero di blocco																								Offset			
Tag																				Indice di linea			Word		Byte	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	1	1	0	1

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

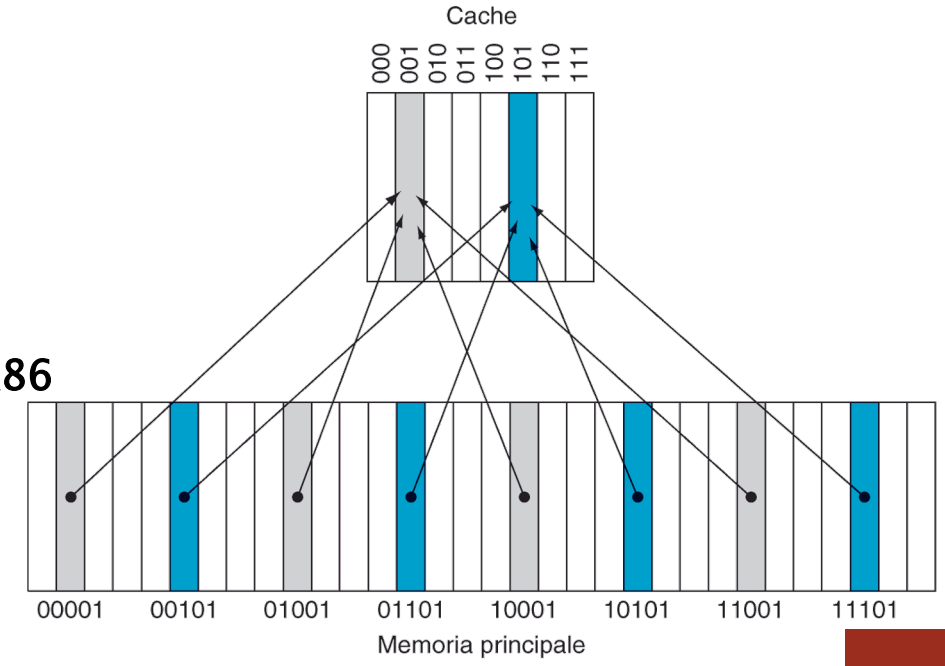
Esempio:

Direct mapped cache con:

- Blocco di 4 word = 16 byte
- 8 linee

Indirizzo: **2157 = 0x0000 086D**

- **Numero di blocco** = 2157 / 16 = **134 = 0x86**
- **Offset di blocco** = 2157 % 16 = **13 = 0xD**
- **Indice di linea** = 134 % 8 = **6 = 0x6**
- **Tag** = 134 / 8 = **16 = 0x10**



Esempio di sequenza di accessi

Direct mapped cache

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

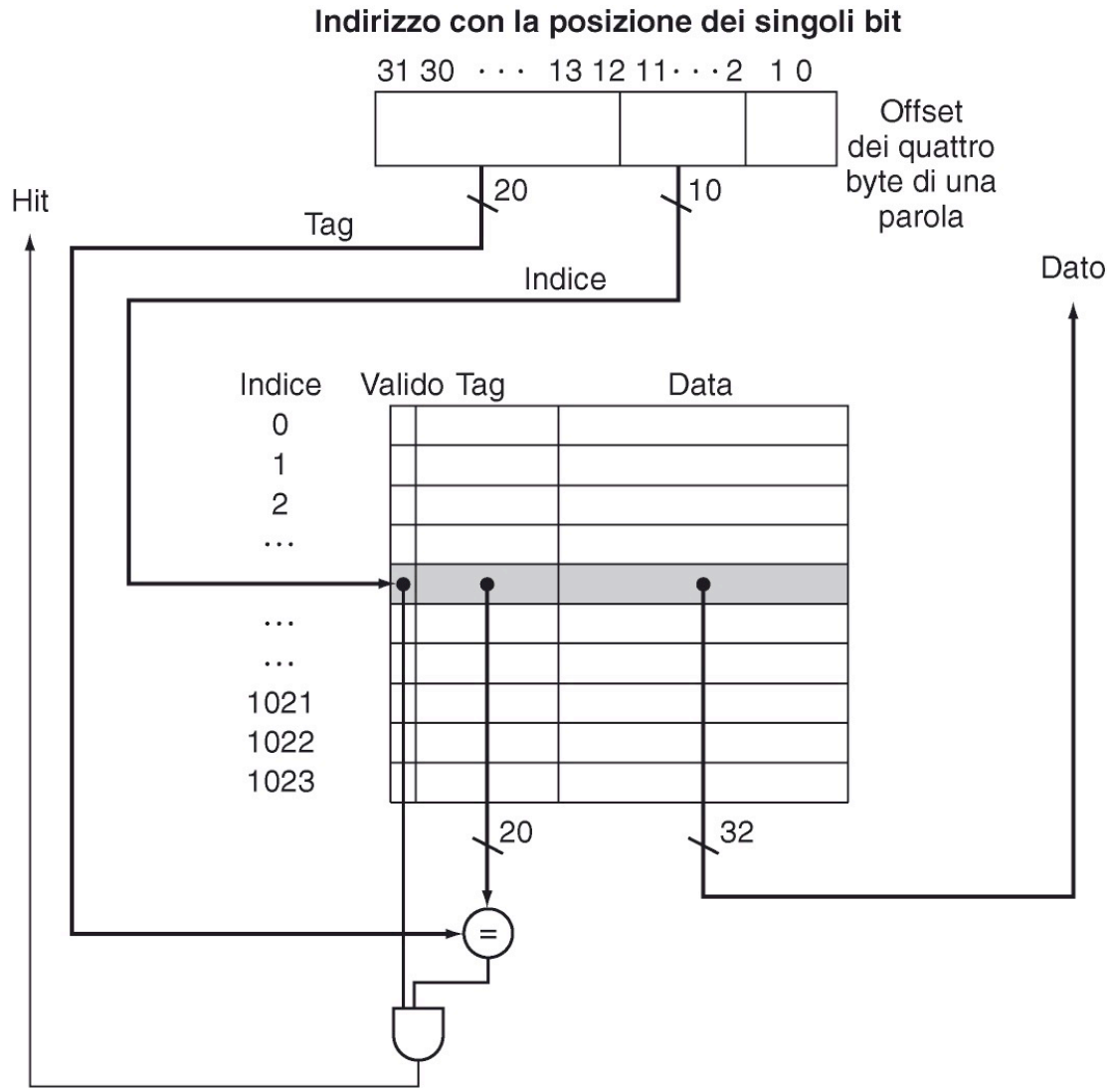
10000 → 10 000

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} **HIT**	Memoria (10110_{due})
111	N		

Come determinare HIT/MISS

Cache con:

- blocchi da 1 word
- 1024 linee



Come determinare HIT/MISS

V	Tag	Indice di linea	Offset di blocco	Offset di parola

Cache con:

- blocchi da 16 word
- 256 linee

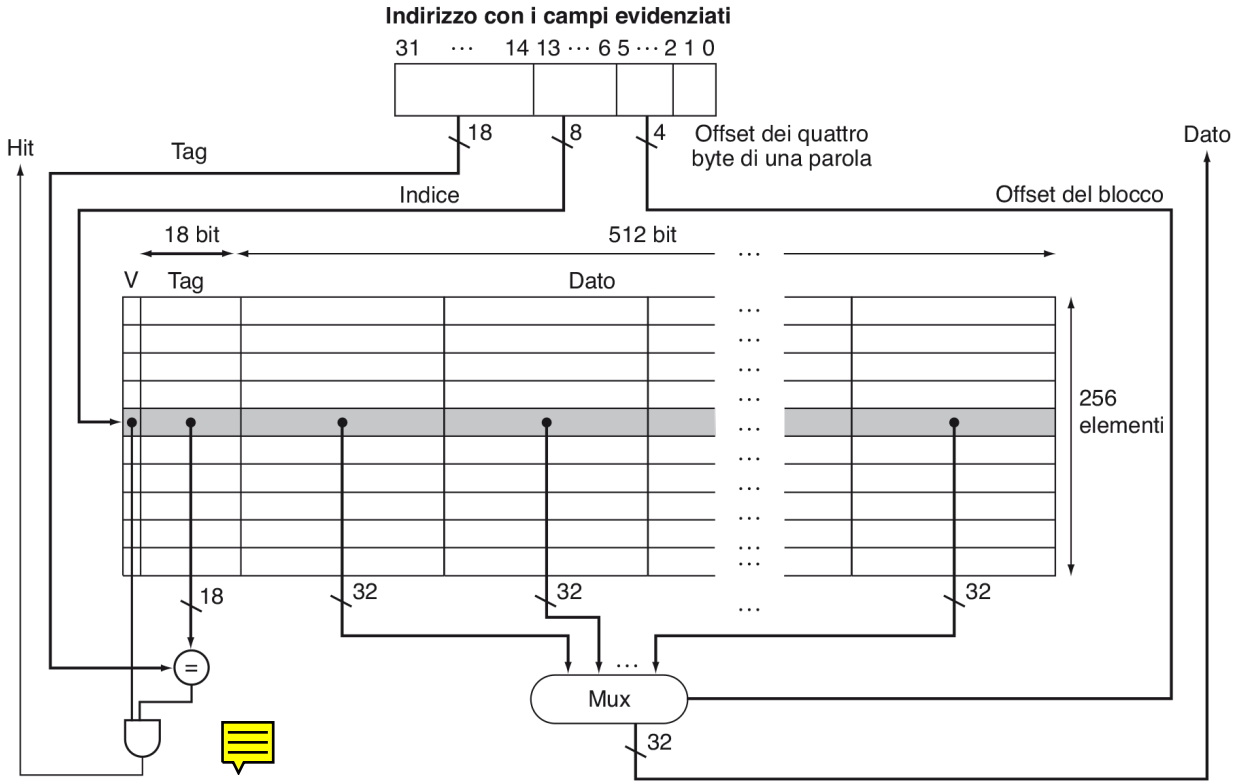


Immagine: David A. Patterson, John L. Hennessy, STRUTTURA E PROGETTO DEI CALCOLATORI, Zanichelli editore S.p.A. Copyright © 2015

Dimensioni della cache

V	Tag	Indice di linea	Offset di blocco	Offset di parola

Dati:

- Cache (direct mapping) con 2^n linee
- Blocchi di dimensione 2^m word da 4 byte ciascuna, cioè 2^{m+2} byte (2^{m+5} bit)
- 1 bit di validità

Dimensione del campo Tag: $32 - n - m - 2$

Dimensione della cache in bit: $2^n \times [2^m \times 32 + (32 - n - m - 2) + 1]$

Dimensioni della cache

V	Tag	Indice di linea	Offset di blocco	Offset di parola

Dati:

- Cache (direct mapping) con 2^n linee
- Blocchi di dimensione 2^m word da 4 byte ciascuna, cioè 2^{m+2} byte (2^{m+5} bit)
- 1 bit di validità

Dimensione del campo Tag: $32 - n - m - 2$

Dimensione della cache in bit: $2^n \times [2^m \times 32 + (32 - n - m - 2) + 1]$

Esempio:

Cache con:

- blocchi da 16 word $\rightarrow m = 4$
- 256 linee $\rightarrow n = 8$

Dimensione del campo Tag: $32 - 8 - 4 - 2 = 18$ bit

Dimensione della cache in bit: $256 \times [16 \times 32 + 18 + 1] = 2^8 \times 531 = 2^{10} \times 531/4$

$135{,}936$ kbit $= 132{,}75$ Kibit