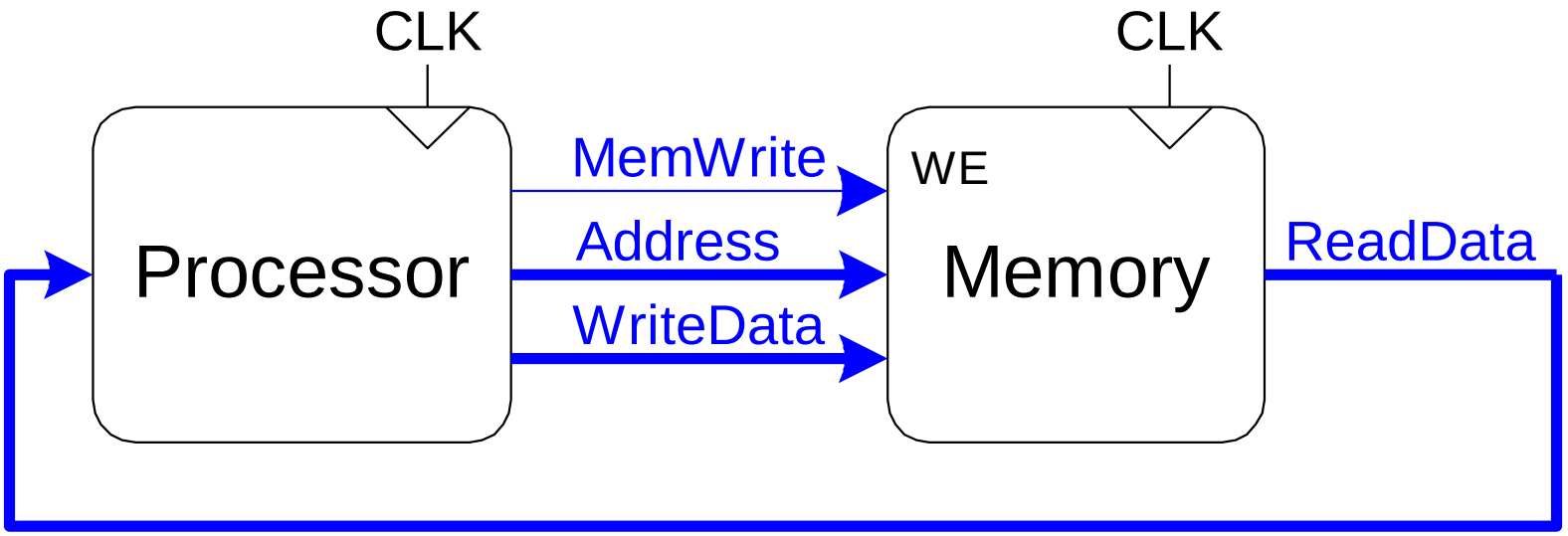
Gestione della memoria

Michele Favalli

Introduzione

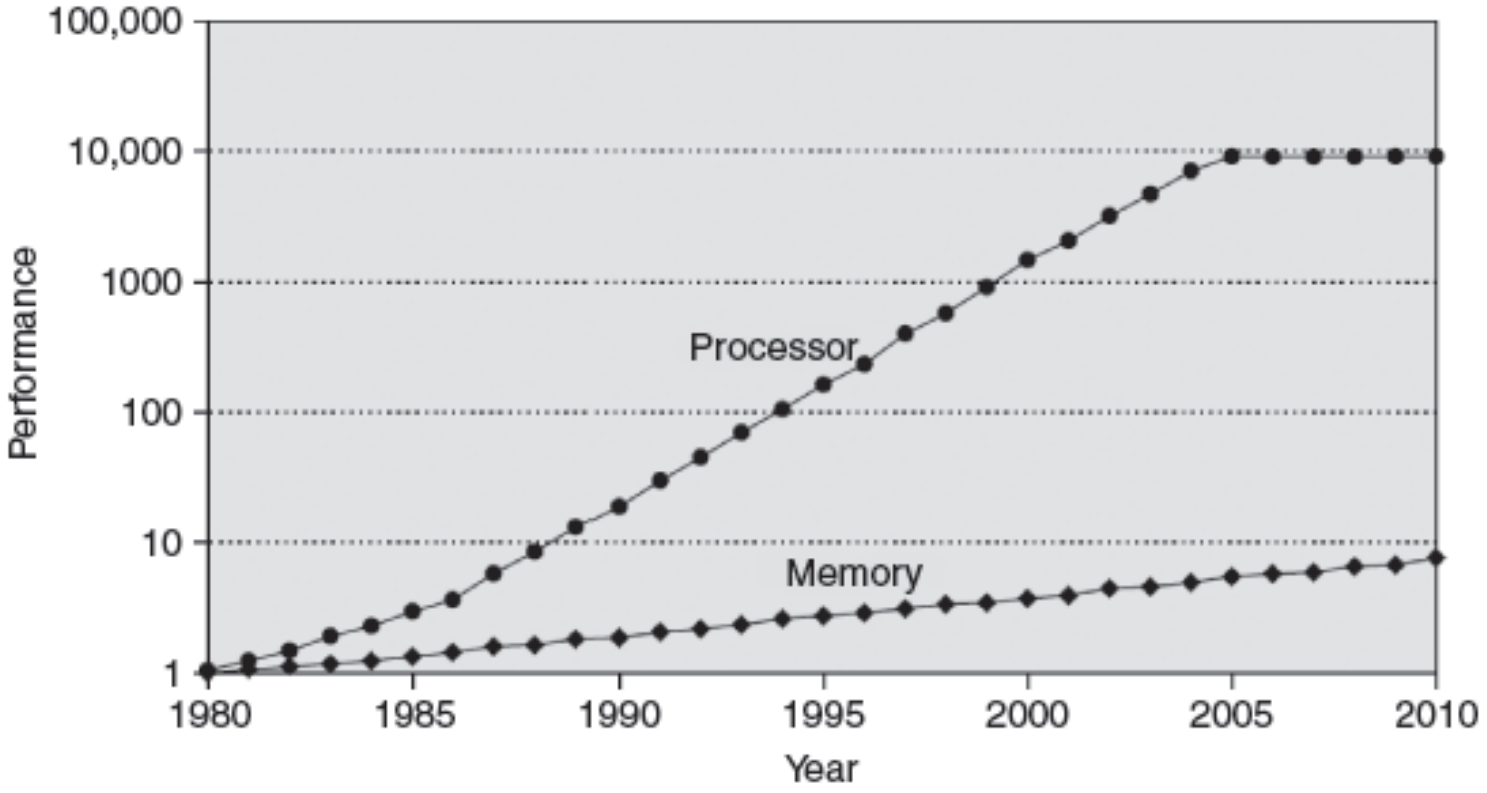
- Le prestazioni di computer dipendono da:
  - Prestazioni del processore
  - Prestazioni della memoria

Memory Interface



Gap fra meoria e processore

Studiando l'architettura MIPS si è assunto che la memoria fosse accessibile in un ciclo di clock, ma non è così da qualche decennio



Obbiettivi nell'utilizzo della memoria

- Rendere la memoria veloce come il processore
- Utilizzo di una gerarchia di memorie
- Memoria ideale
  - Veloce
  - Poco costosa
  - Grande

Se ne possono ottenere soltanto due!

Gerarchia di memoria

- Memorizzando su una **memoria piccola, veloce e «vicino» al processore** i dati/istruzioni a cui sto accedendo, oltre a quelli «vicini»

- *Bassa Latenza*
- *Alto throughput*
- *Piccola dimensione*
- *Alto costo-per-bit*

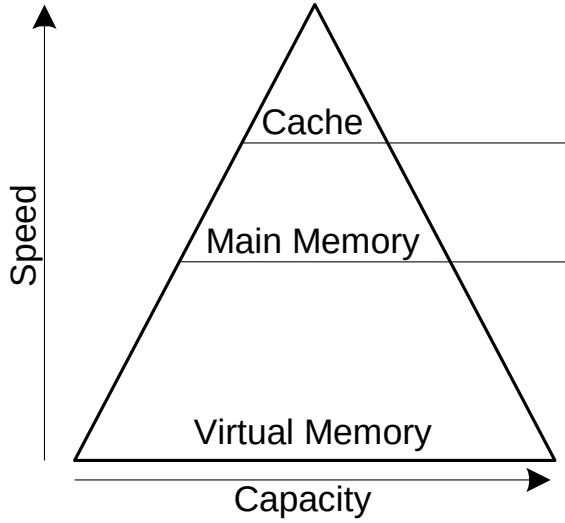
Implementando il sistema di memoria come una «gerarchia», il microprocessore ha l'illusione di avere una memoria grande come l'ultimo livello, ma veloce come al primo livello

Come usare la gerarchia di memoria

- Se cerco qualcosa a un livello L e lo trovo ho una **hit**
- Se cerco qualcosa a un livello L e non lo trovo ho una **miss**
- In questo caso lo vado a cercare al livello L+1 e se ho una hit lo porto al livello L
- In realtà si portano più cose al livello L, vedremo il motivo

Gerarchia di memoria, qualche dato

Vediamo un po’ di dati (chiaramente evolvono)



Technology	Price / GB	Access Time (ns)	Bandwidth (GB/s)
SRAM	$10,000	1	25+
DRAM	$10	10 - 50	10
SSD	$1	100,000	0.5
HDD	$0.1	10,000,000	0.1

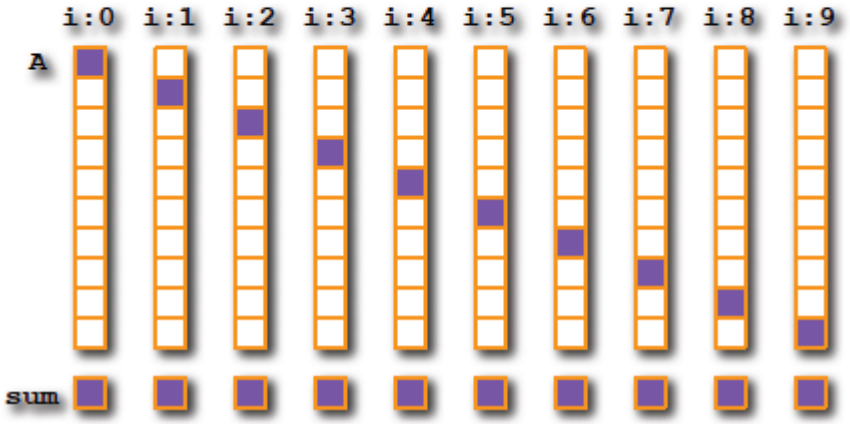
Località

Il software esibisce due proprietà che ci aiutano

- **Località temporale:**
  - Se un dato/istruzione è stato usato di recente, è probabile che verrà riutilizzato presto
  - **Utilizzo:** gli oggetti acceduti di recente vengono tenuti nei livelli più alti della gerarchia di memoria
- **Località spaziale:**
  - Se un dato/istruzione è stato usato di recente, è probabile che dati/istruzioni vicini (come indirizzo) siano usati presto
  - **Utilizzo:** quando si accede a un oggetto, si portano anche i dati vicini nei livelli più alti di memoria

Esempio: Dati

```
sum = 0;
for (i = 0; i < 10; i++)
  sum += A[i];
```



Come sfruttare questo fatto?

- **Località spaziale**
  - **Accesso agli elementi A[i] in successione**
- **Località temporale**
  - **Ad ogni iterazione viene utilizzata la variabile «somma»**

Esempio: Istruzioni

```
sum = 0;
for (i = 0; i < 10; i++)
  sum += A[i];
```

```
    movl $0, -12(%ebp)
    movl $0, -16(%ebp)
    jmp L2
L3:
    movl -16(%ebp), %eax
    movl -56(%ebp,%eax,4), %edx
    leal -12(%ebp), %eax
    addl %edx, (%eax)
    leal -16(%ebp), %eax
    incl (%eax)
L2:
    cmpl $9, -16(%ebp)
    jle L3
```

- **Località spaziale**
  - **Accesso alle istruzioni in sequenza**
- **Località temporale**
  - **Ripetizione iterativa delle istruzioni del loop**

Prestazioni della memoria

Hit Rate = # hits / # memory accesses
= 1 – Miss Rate

Miss Rate = # misses / # memory accesses
= 1 – Hit Rate

- **Average memory access time (AMAT):** tempo medio usato dal processore per accedere ai dati in un sistema con 3 livelli di memoria (cache, memoria pincipale (MM) e memoria virtuale (VM))

AMAT = $t_{\text{cache}} + MR_{\text{cache}}[t_{MM} + MR_{MM}(t_{VM})]$

Esempio di valutazionedi hit e miss rate

- Un programma ha 2000 load e store
- 1250 di questi dai sono trovati nella cache
- I rimanenti sono presi da livelli più bassi di memoria
- **Quali valori hanno l'hit e il miss rate per la cache?**

Hit Rate = 1250/2000 **= 0.625**

Miss Rate = 750/2000 = **0.375** = 1 – Hit Rate

Esempio di valutazione delle prestazioni di una memoria

- Supponiamo che ci siano due soli livelli di memoria: cache e memoria principale
- $t_{\text{cache}} = 1$ ciclo, $t_{MM} = 100$ cicli
- **Che valore ha AMAT del programma considerate nell'esempio?**

$$\begin{aligned} \textbf{AMAT} &= t_{\text{cache}} + MR_{\text{cache}}(t_{MM}) \\ &= [1 + 0.375(100)] \text{ cycles} \\ &= \textbf{38.5 cycles} \end{aligned}$$

Gene Amdahl

- **Legge di Amdahl:** lo sforzo speso per migliorare le prestazioni di un sottosistema è sprecato se tale sottosistema non condiziona una grande percentuale delle prestazioni complessive

Cache

- Livello più alto in memoria (a parte I registri che però sono gestiti diversamente)
- Veloce (tipicamente 1 un tempo di accesso pari a un period di clock)
- Nel caso ideale dovrebbe contenere tutti I dati (o le istruzioni) cui il processore deve accedere
- Tipicamente tiene i dati o le istruzioni utilizzati più di recente

Progettazione della cache

- Ragioniamo su una cache dati per semplicità
- Quali dati tenere nella cache?
- Come trovarli?
- Come sostituirli?
- Consideriamo le load (le store usano principi simili)

Quali dati tenere nella cache?

- Idealmente tutti i dati dovrebbero essere inseriti anticipatamente nella cache
- Non è possibile predire il futuro
- Si utilizza la storia del sistema basandosi sul principio di località
  - **Località temporale:** ogni nuovo dato cui si accede viene copiato nella cache => ha un elevate probabilità di essere richiesto nuovamente
  - **Località spaziale:** copia anche i dati adiacenti nella cache => è probabile che si accedera' anche a questi

Definizioni per una cache

- **Capacità (*C*):**
  - numero di byte della cache
- **Block size (*b*):**
  - numero bytes o word trasferiti da/per la cache per volta
- **Numero di blocchi (B):**
  - $B = C/b$
- **Grado di associatività (*N*):**
  - numero di blocchi in un set
- **Numero di set (*S*):**
  - $S=B/N$
  - **ogni indirizzo di memoria si mappa in un set della cache**
  - quindi dati ad indirizzi diversi finiscono nello stesso set della cache => la cache, oltre al dato, deve contenere un informazione sull'indirizzo

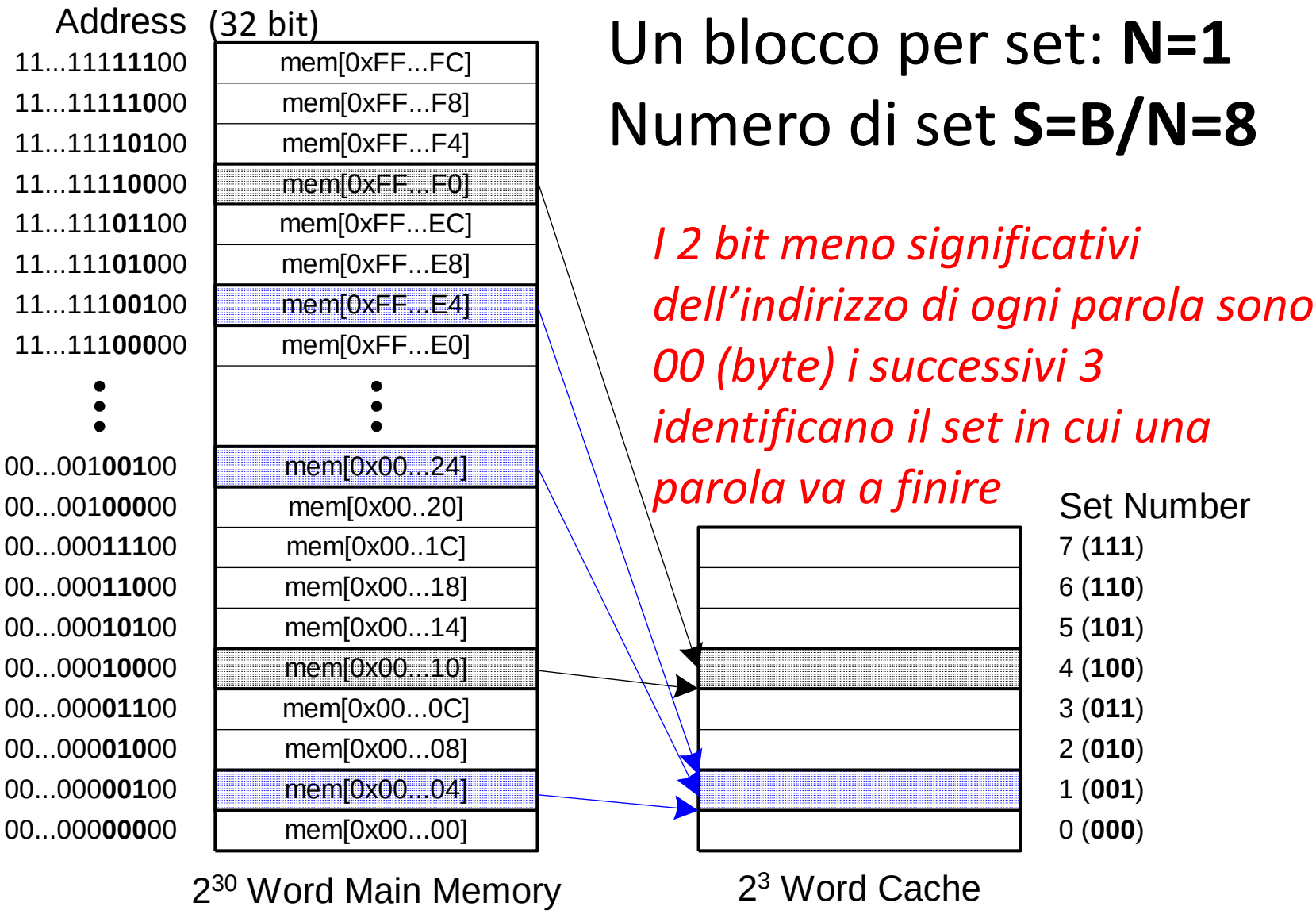
Come trovare i dati?

- La cache è organizzata in *S* set
- Ogni indirizzo di memoria si mappa esattamente su un set
- Ci sono diverse categorie di cache dipendentemente dal numero di blocchi in un set:
  - **Direct mapped:** 1 blocco per set
  - ***N*-way set associative:** *N* blocchi per set
  - **Fully associative:** tutti i blocchi della cache in un set

Esempio di cache

- ***C* = 8** word (capacità)
- ***b* = 1** word o 4 byte (block size)
- ***B* = C/b = 8** (# of blocchi)

Direct Mapped Cache

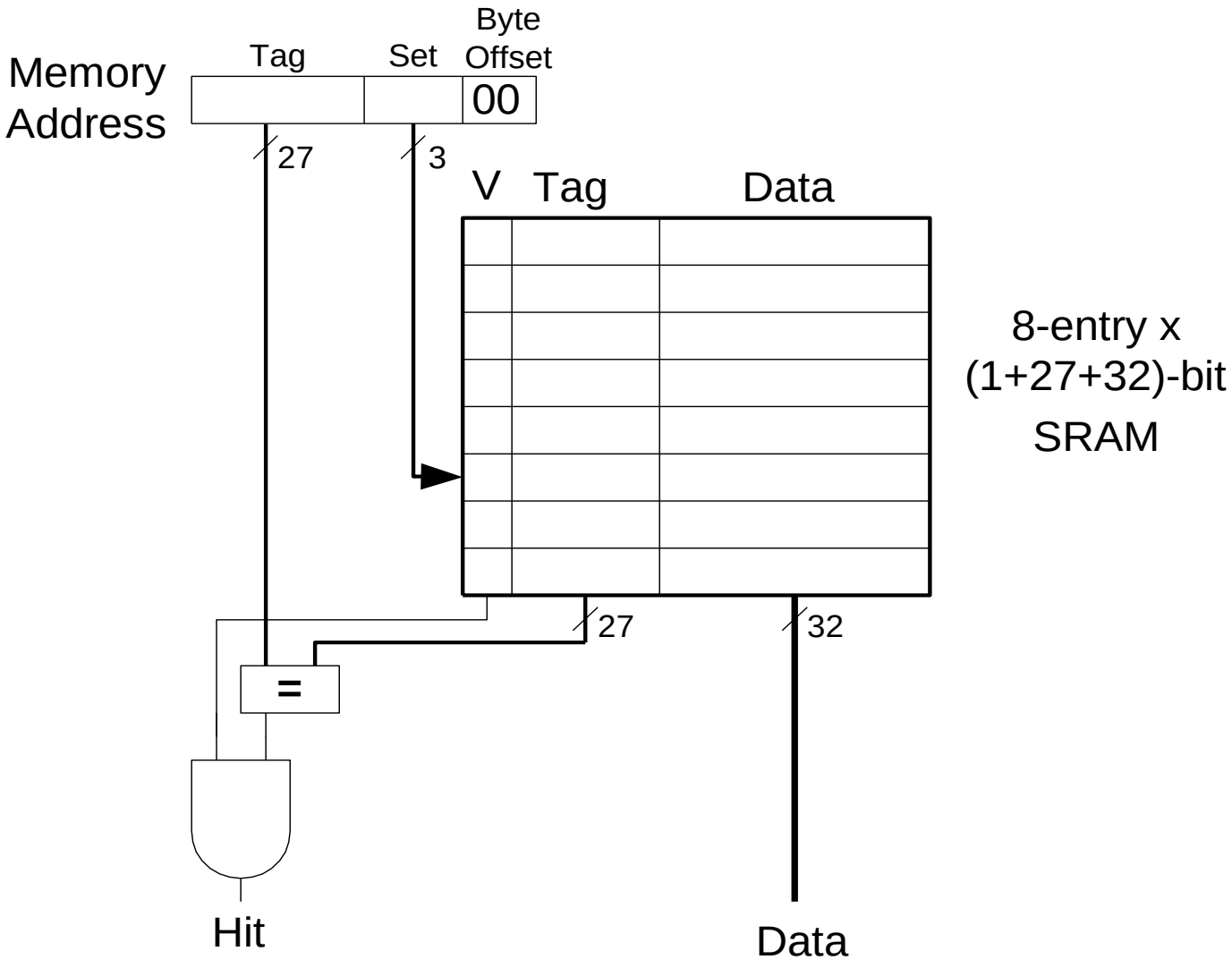


Un blocco per set: **N=1**

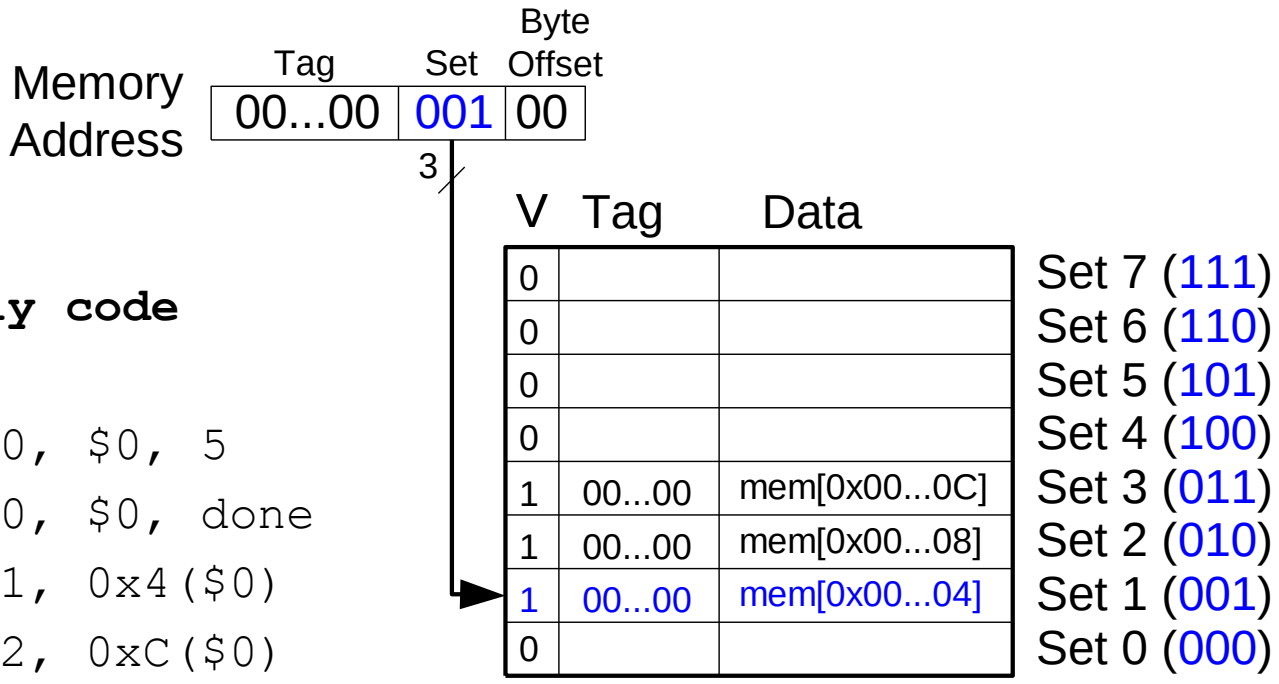
Numero di set **S=B/N=8**

I 2 bit meno significativi dell'indirizzo di ogni parola sono 00 (byte) i successivi 3 identificano il set in cui una parola va a finire

Hardware per gestire una cache Direct Mapped



Prestazioni della cache Direct Mapped

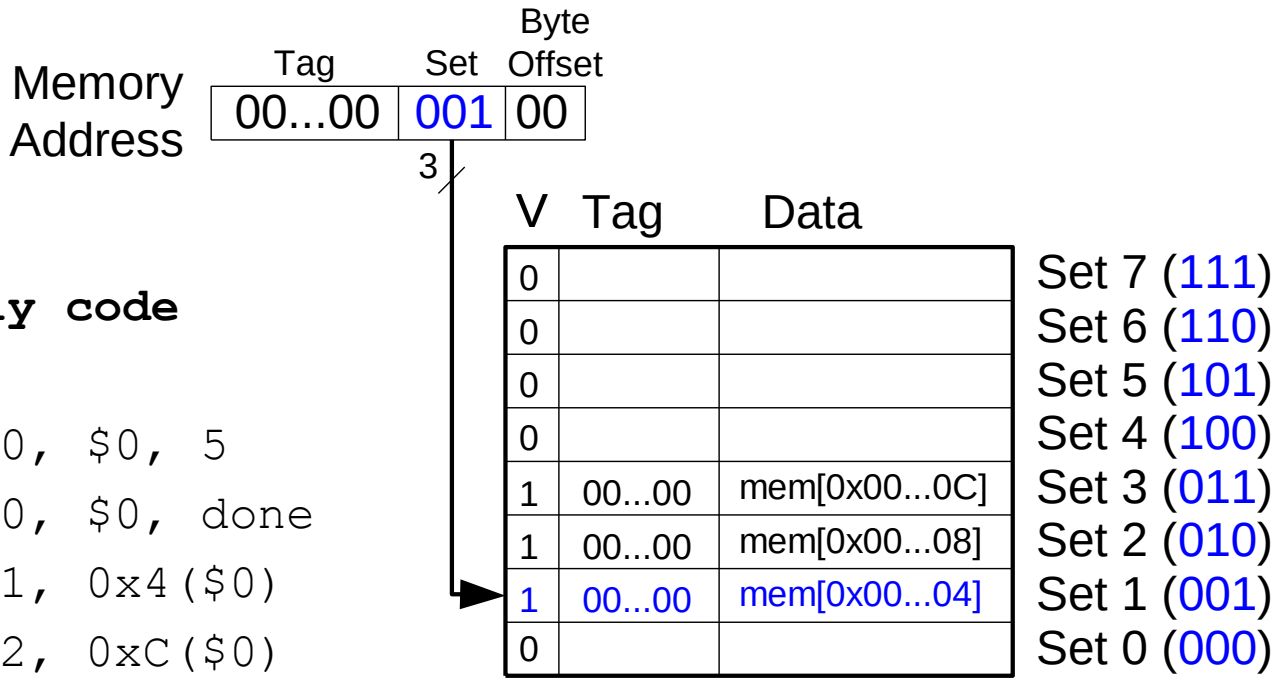



```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0xC($0)
        lw   $t3, 0x8($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = ?

Direct Mapped Cache Performance




```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0xC($0)
        lw   $t3, 0x8($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = 3/15

= 20%

Località temporale

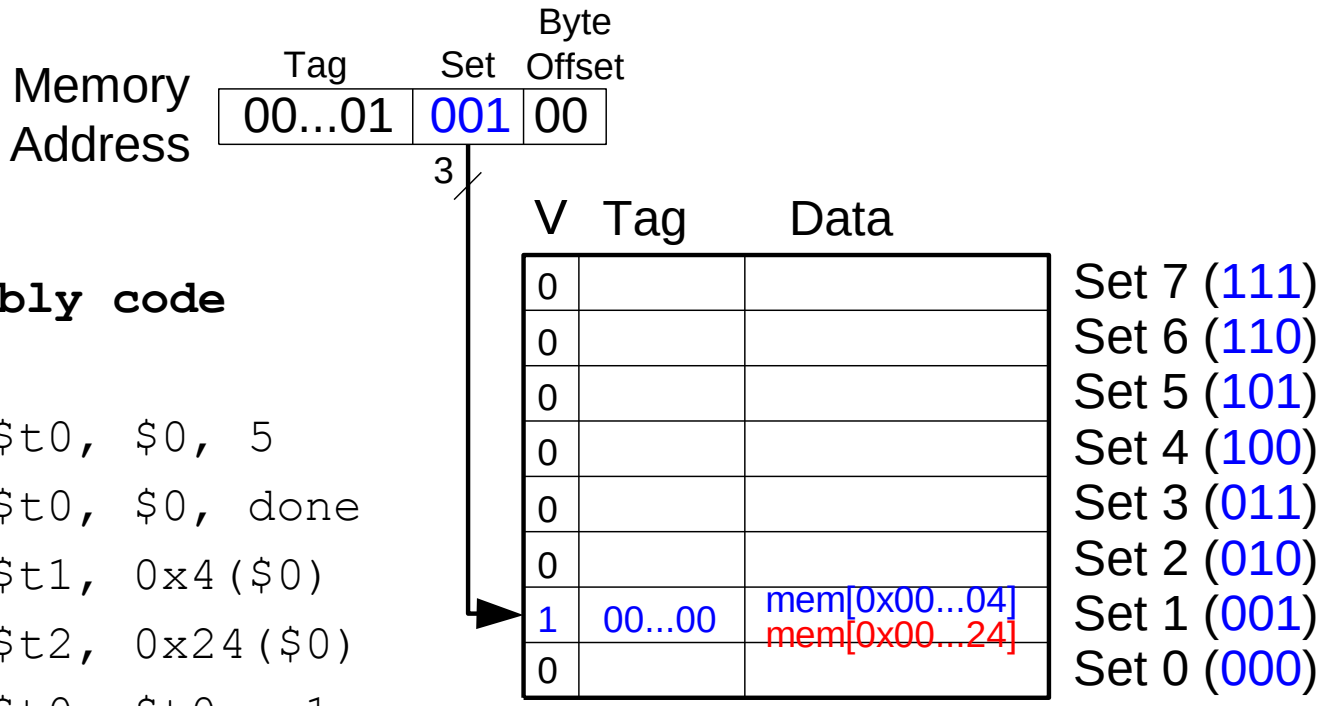
Compulsory Miss

(dovute al caricamento iniziale dei dati)

Direct Mapped Cache: conflitti

```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```

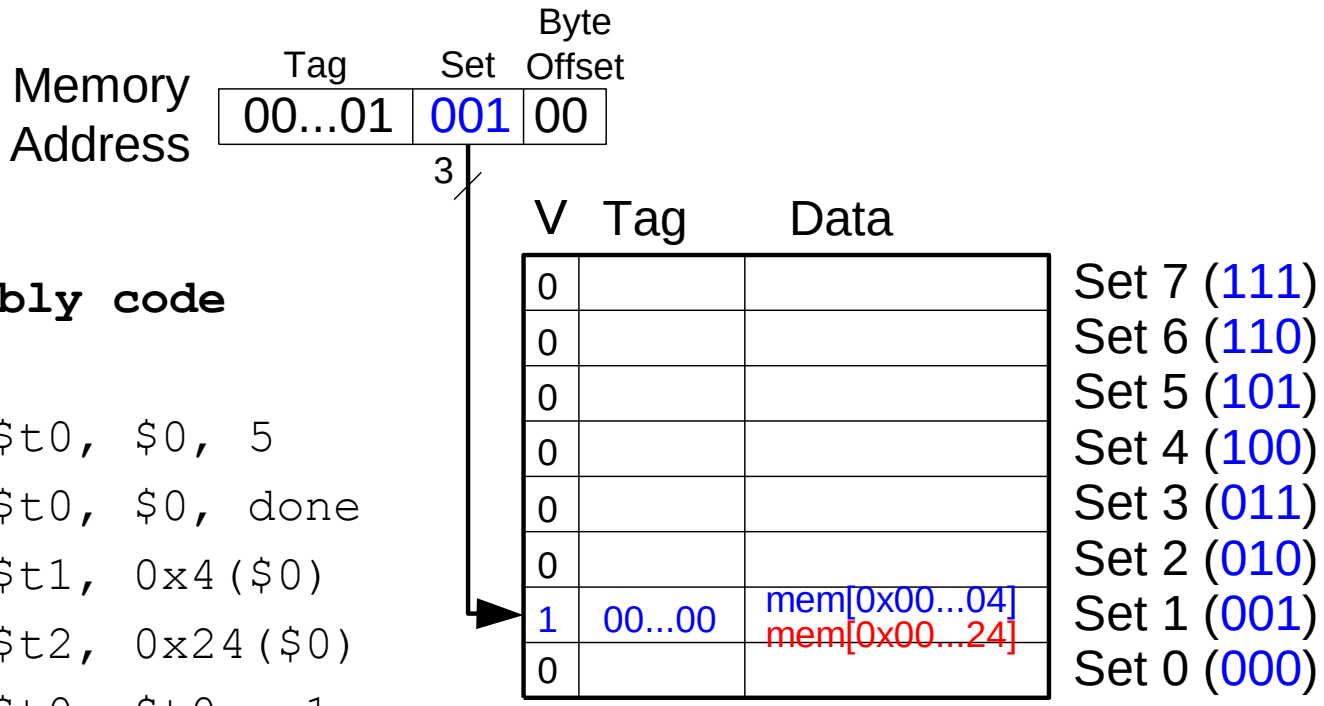


Miss Rate = ?

Direct Mapped Cache: Conflict

```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```



Miss Rate = 10/10

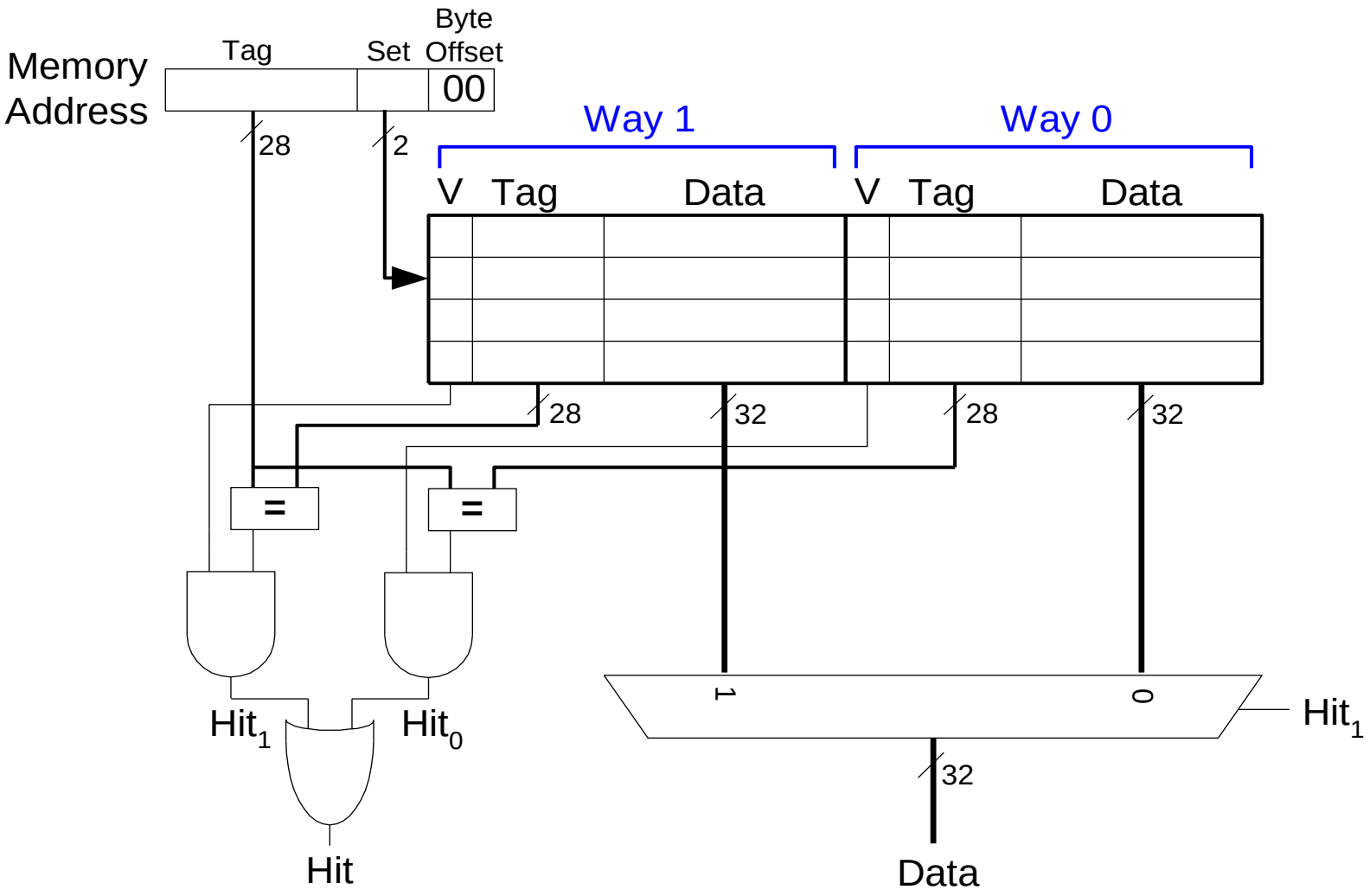
= 100%

Miss dovute a Conflitti

Come mitigare il problema?

- Si considerano dei set con più blocchi al loro interno
- In questo modo ciascun set può contenere più di un dato
- Cache di tipo N-way associative

N-Way Set Associative Cache

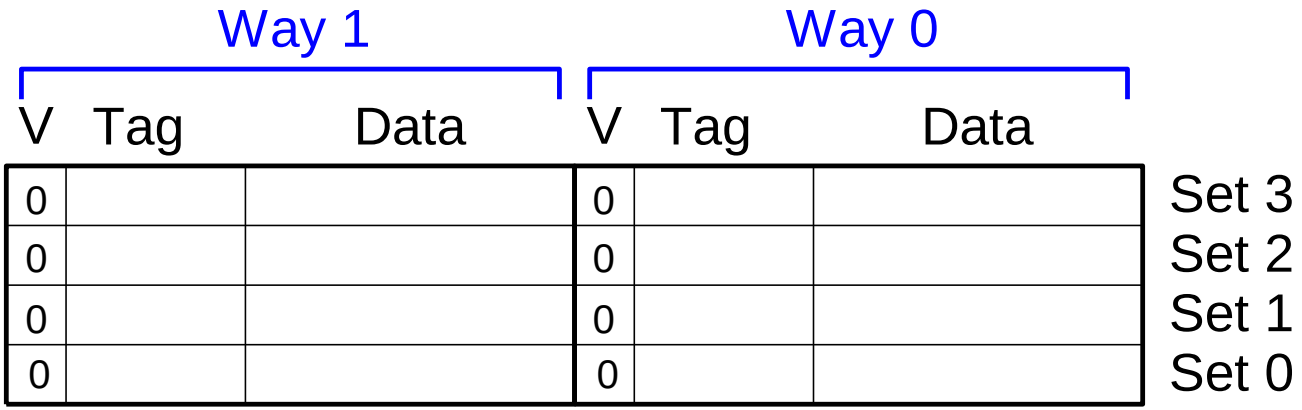


Prestazioni della cache set associativa (*N*-Way)

```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = ?



Prestazioni della cache set associativa (*N*-Way)

```
# MIPS assembly code

        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = 2/10

= 20%

L'accociatività riduce i conflitti dovuti a miss

	Way 1			Way 0		
	V	Tag	Data	V	Tag	Data
Set 3	0			0		
Set 2	0			0		
Set 1	1	00...10	mem[0x00...24]	1	00...00	mem[0x00...04]
Set 0	0			0		