ARCHITETTURA DEGLI ELABORATORI

A.A. 2020-2021

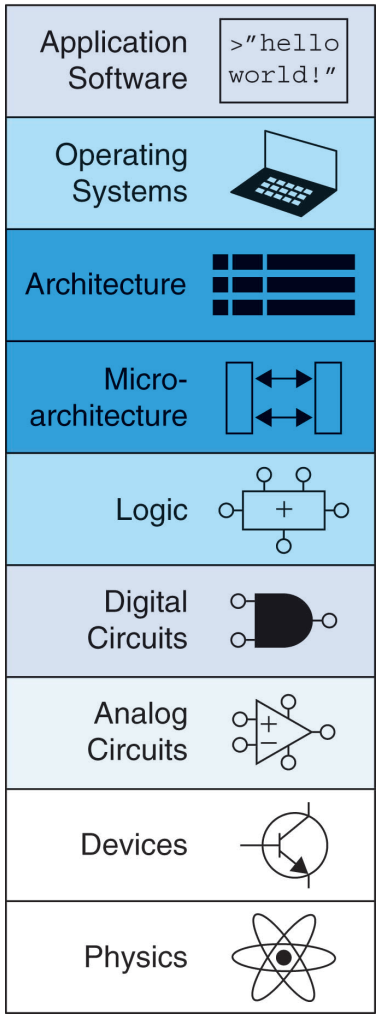
Università di Napoli Federico II
Corso di Laurea in Informatica

Docenti
Proff. Luigi Sauro gruppo 1 (A-G)
Silvia Rossi gruppo 2 (H-Z)

MEMORIE

Chapter 8 :: Topics

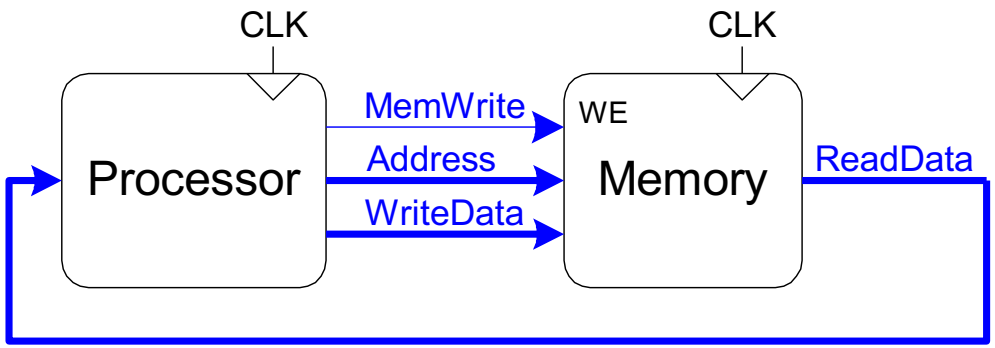
- Introduction
- Memory System Performance Analysis
- Caches
- Virtual Memory
- Memory-Mapped I/O
- Summary



Memorie vs CPU

Nell'architettura Von Neuman il canale di comunicazione tra la CPU e la memoria è il punto critico (collo di bottiglia) del sistema.

La tecnologia consente di realizzare CPU sempre più veloci e memorie sempre più grandi ma la velocità di accesso delle memorie non cresce così rapidamente come la velocità della CPU.



Memorie vs CPU

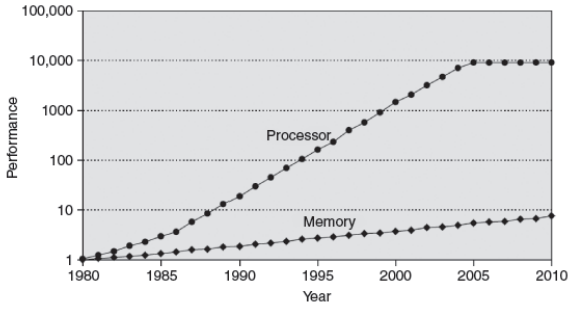
Storicamente le **CPU** sono sempre state più veloci delle **memorie**.

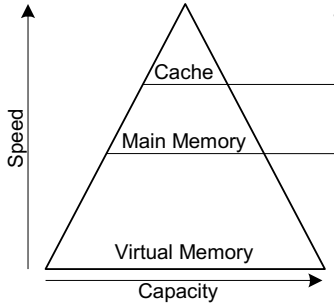
Al giorno d'oggi siamo in grado di produrre delle memorie veloci quanto una CPU moderna, il reale problema è dato da:

- queste memorie hanno un costo elevatissimo.
- le memorie dovrebbero essere piazzate in gran parte sugli stessi chip delle CPU, il che non è possibile.

Per ovviare a questo problema, gli ingegneri ricorrono ad uno schema chiamato "**a gerarchia di memoria**" in cui si combinano:

- una quantità molto **piccola** di memoria estremamente **veloce** (la cache)
- una quantità molto **grande** di memoria **lenta**.





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

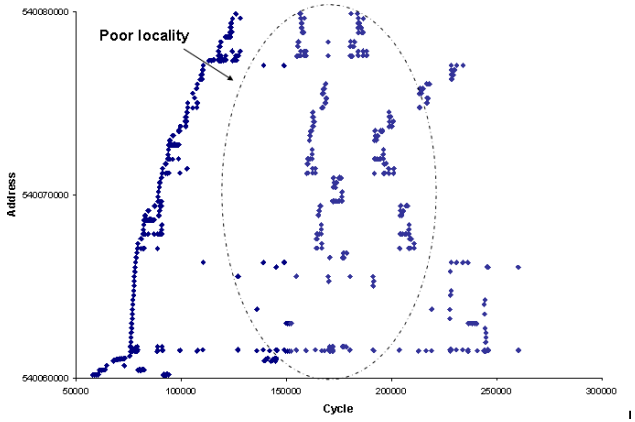
Inquadramento del Problema

La maggior parte del tempo di esecuzione di una CPU è, di solito, impegnato da procedure **in cui vengono eseguite ripetutamente le stesse istruzioni**.

In realtà, non è importante conoscere lo schema dettagliato della sequenza di istruzioni, il punto chiave è che molte istruzioni in aree ben localizzate del programma vengono eseguite ripetutamente in un determinato periodo, e si accede al resto del programma relativamente di rado.

Questa proprietà viene chiamata ***località dei riferimenti***. Si manifesta in due modi: ***località temporale*** e ***località spaziale***.

- La ***località temporale*** rappresenta la probabilità che un'istruzione eseguita di recente venga eseguita nuovamente, entro breve tempo.
  - -> Mantenere i dati usati di recente nei livelli più alti della gerarchia di memoria
- La ***località spaziale*** rappresenta invece la probabilità che istruzioni vicine ad un'istruzione eseguita di recente (dove la vicinanza è espressa in termini di indirizzi delle istruzioni) siano anch'esse eseguite nel prossimo futuro.
  - -> quando si accede ai dati, portare i dati vicini nei livelli più alti di gerarchia



Cache

- Highest level in memory hierarchy
- Fast (typically ~ 1 cycle access time)
- Ideally supplies most data to processor
- Usually holds most recently accessed data

Utilità di avere una cache memory

La velocità con cui la memoria risponde alle richieste di istruzioni e dati della CPU ha un peso determinante sulle prestazioni di un sistema.

Se tra la memoria principale e la CPU si potesse **interporre una memoria molto veloce, contenente le parti di programma e di dati che, volta per volta, interessano l'elaborazione,** il tempo totale di esecuzione verrebbe ridotto in modo significativo.

Questa è esattamente la funzione della "***cache memory***" che in inglese significa letteralmente "***schermo della memoria***".

Si tratta quindi di un elemento che inserito tra CPU e memoria principale impedisce alla prima di "***vedere***" i tempi di risposta reali della memoria.

La struttura della CPU, infatti, non viene influenzata dalla presenza o meno di una ***cache memory***. L'interfaccia verso la memoria applica il "***protocollo***" di trasferimento ***dalla*** memoria o ***verso*** la memoria ignorando la presenza di una struttura intermedia.

Agli albori di questa tecnica, infatti, la cache memory era solo un accessorio a pagamento.

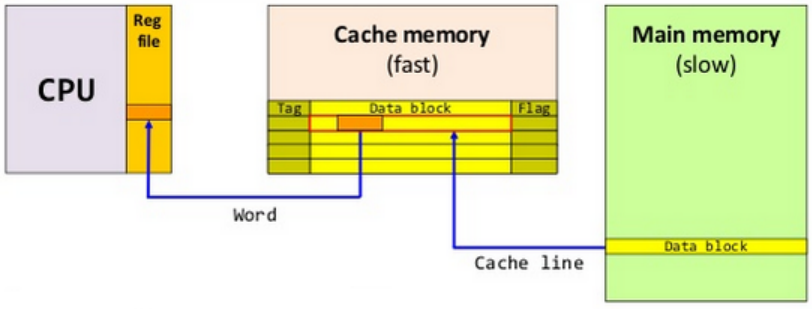
Una traduzione possibile in italiano del concetto è "***memoria tampone***".

Principi di funzionamento

Concettualmente, le operazioni svolte da una memoria cache sono molto semplici. I circuiti di controllo della memoria cache sono progettati per avvantaggiarsi della proprietà della località dei riferimenti.

- L'aspetto temporale di questa proprietà suggerisce di portare un elemento (istruzioni o dati) nella cache quando viene richiesto per la prima volta, in modo tale che rimanga a disposizione nel caso di una nuova richiesta.

- L'aspetto spaziale suggerisce che, invece di portare dalla memoria principale alla cache un elemento alla volta, è conveniente portare un insieme di elementi che risiedono in indirizzi adiacenti.



Si farà riferimento al termine ***blocco*** per indicare un insieme di indirizzi contigui di una qualche dimensione.

Un altro termine utilizzato di frequente per indicare un blocco della cache è ***linea di cache***.

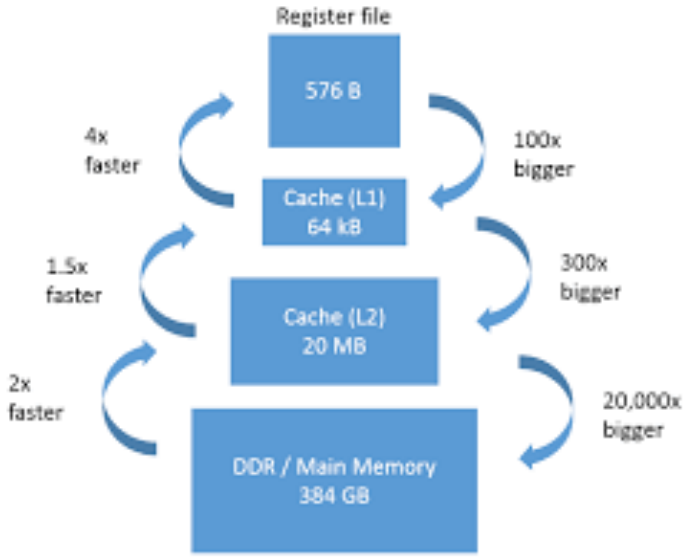
Dimensione della cache

Quando si riceve una richiesta di lettura fatta dalla CPU, il contenuto di un blocco di parole di memoria contenenti la locazione specificata viene trasferito nella cache, una o più parole alla volta.

In seguito, ogniqualvolta il programma fa riferimento a una di queste locazioni del blocco, i valori desiderati vengono letti direttamente dalla cache.

Affinché la cache svolga efficacemente il suo compito deve avere tempi di accesso molto più brevi di quelli della memoria principale e ciò come vedremo impone che sia piccola.

Se è piccola non potrà che contenere una frazione ridotta delle istruzioni ed i dati della memoria principale.



Tecniche di gestione

La corrispondenza tra i blocchi della memoria principale e quelli della cache è specificata dalla funzione di ***posizionamento*** (***mapping***).

Quando la cache è piena e si fa riferimento a una parola di memoria (istruzione o dato) che non è presente nella cache, l'hardware di controllo della cache deve decidere quale blocco della cache debba essere rimosso per far spazio al nuovo blocco, che contiene la parola a cui si fa riferimento.

L'insieme di regole in base alle quali viene fatta questa scelta costituisce ***l'algoritmo di sostituzione***.

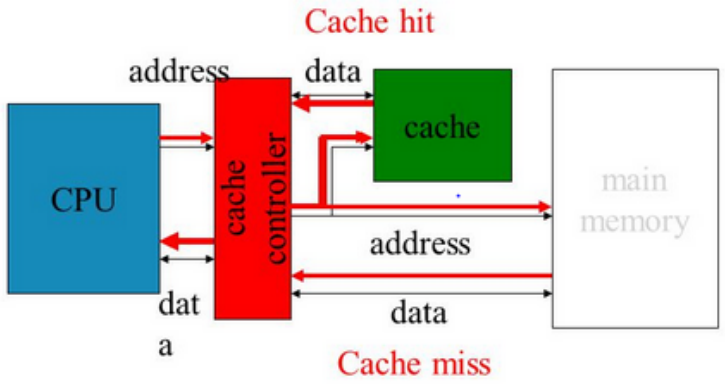
Tecniche di gestione

La struttura dei sistemi è, di solito, organizzata in modo che la presenza di una cache non influenzi il funzionamento della CPU.

Quest'ultima, infatti, nell'esecuzione del programma **effettua le richieste di lettura e scrittura utilizzando gli indirizzi delle locazioni nella memoria principale**.

E' la logica di controllo della cache che si fa carico di determinare se la parola richiesta è presente o meno nella cache.

Se è presente, viene effettuata l'operazione di lettura o scrittura della locazione di memoria appropriata. In questo caso si dice che **l'accesso in lettura o in scrittura ha avuto successo** (***read hit*** o ***write hit***).



Possibili gestioni di un “cache hit”

Se l’accesso alla cache ha avuto successo bisogna distinguere due tipi di situazioni:

Operazione di lettura, la memoria principale non viene coinvolta

Operazione di scrittura, il sistema può procedere in due modi:

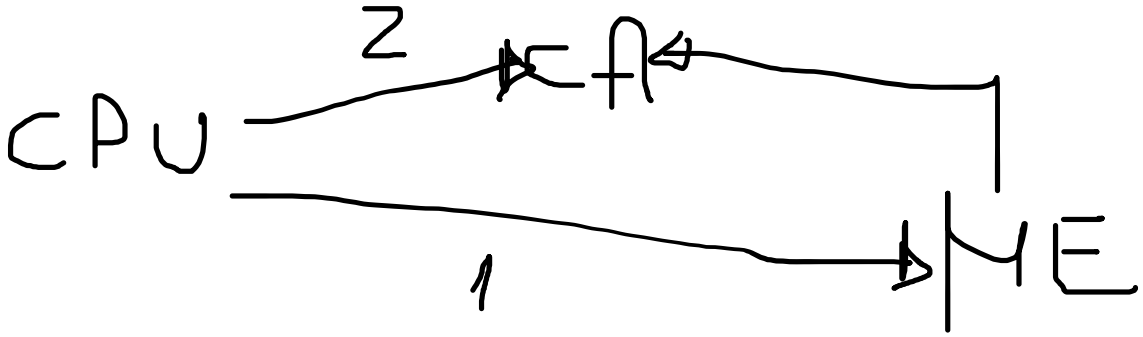
- Nel primo (***write-through***), la locazione della cache e quella della memoria principale **vengono entrambe aggiornate**.
  - Il ***write-through*** è più semplice, ma implica inutili operazioni di scrittura nella memoria principale, se una parola viene aggiornata più volte durante il periodo in cui risiede nella cache.
- Nel secondo (***write-back***) si aggiorna **soltanto la locazione della memoria cache**, segnandola come aggiornata con un **bit di modifica** o ***dirty***. La locazione della memoria principale viene aggiornata in seguito, quando il blocco contenente la parola marcata deve essere rimosso dalla memoria cache per far posto a un nuovo blocco.
  - Anche il protocollo ***write-back*** può causare inutili scritture nella memoria principale, visto che, quando si procede con la scrittura nella memoria principale di un blocco, **tutte le parole del blocco vengono scritte**, anche se solo una delle parole del blocco della cache è stata modificata.

Gestione di un “read miss”

Quando la parola indirizzata durante un'operazione di lettura non è presente nella cache si dice che **l'accesso in lettura è fallito** (***read miss***).
Il blocco di parole contenente la parola richiesta deve essere copiato dalla memoria principale nella memoria cache.

Due sono le possibilità:
Dopo aver caricato l'intero blocco nella memoria cache, la parola richiesta viene inviata alla CPU.
In alternativa, è possibile inviare immediatamente la parola alla CPU, non appena la si legge dalla memoria principale.

Quest'ultimo approccio, chiamato ***load-through***, o anche ***early restart***, riduce in qualche modo il tempo di attesa della CPU, a discapito di una maggiore complessità del circuito di controllo della cache.



Gestione di un “write miss”

Durante un'operazione di scrittura, se la parola indirizzata non è nella cache si dice che **l'accesso in scrittura è fallito** (***write miss***).

Anche in questo caso ci sono le due alternative,
se si utilizza un protocollo ***write-through***, le informazioni vengono scritte direttamente nella memoria principale,
nel caso invece di un protocollo ***write-back***, il blocco, contenente la parola indirizzata, viene prima caricato nella cache, poi viene soprascritto con le nuove informazioni.

Prestazioni

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

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

Average memory access time (AMAT): average time for processor to access data

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

Prestazioni

- Un programma ha 2.000 loads e stores
- 1.250 di questi dati sono presenti nella cache
- Il resto è fornito da altri livelli nella gerachia di memoria
- **Quale è l'hit e il miss rate per la cache?**

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

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

Prestazioni

- Assumi che un processore ha 2 livelli di memoria: cache e main memory
- t_{cache} = 1 cycle, t_{MM} = 100 cycles
- **Qual'è l'AMAT dati gli hit e miss rare precedenti?**

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

Terminologia Cache

- **Capacity (*C*):**
  - Numero di byte nella cache
- **Block size (*b*):**
  - Grandezza di un blocco, ovvero numero di bytes che sono trascritti nella cache per volta
- **Number of blocks (*B = C/b*):**
  - Numero di blocchi nella cache
- **Degree of associativity (*N*):**
  - Numero di blocchi in un set
- **Number of sets (*S = B/N*):**
  - Ogni indirizzo di memoria è mappato in esattamente un set della cache

Tipologie di memorie cache

- La memoria Cache è organizzata in *S* set
- Ogni indirizzo di memoria mappa in esattamente un unico set
- Diverse categorie di memoria cache dipendono dal numero di blocchi che un set contiene:
  - **Direct mapped:** 1 blocco per ogni set
  - ***N*-way set associative:** *N* blocchi per set
  - **Fully associative:** tutti i blocchi di una cache sono in un unico set

Tipologie di memorie cache

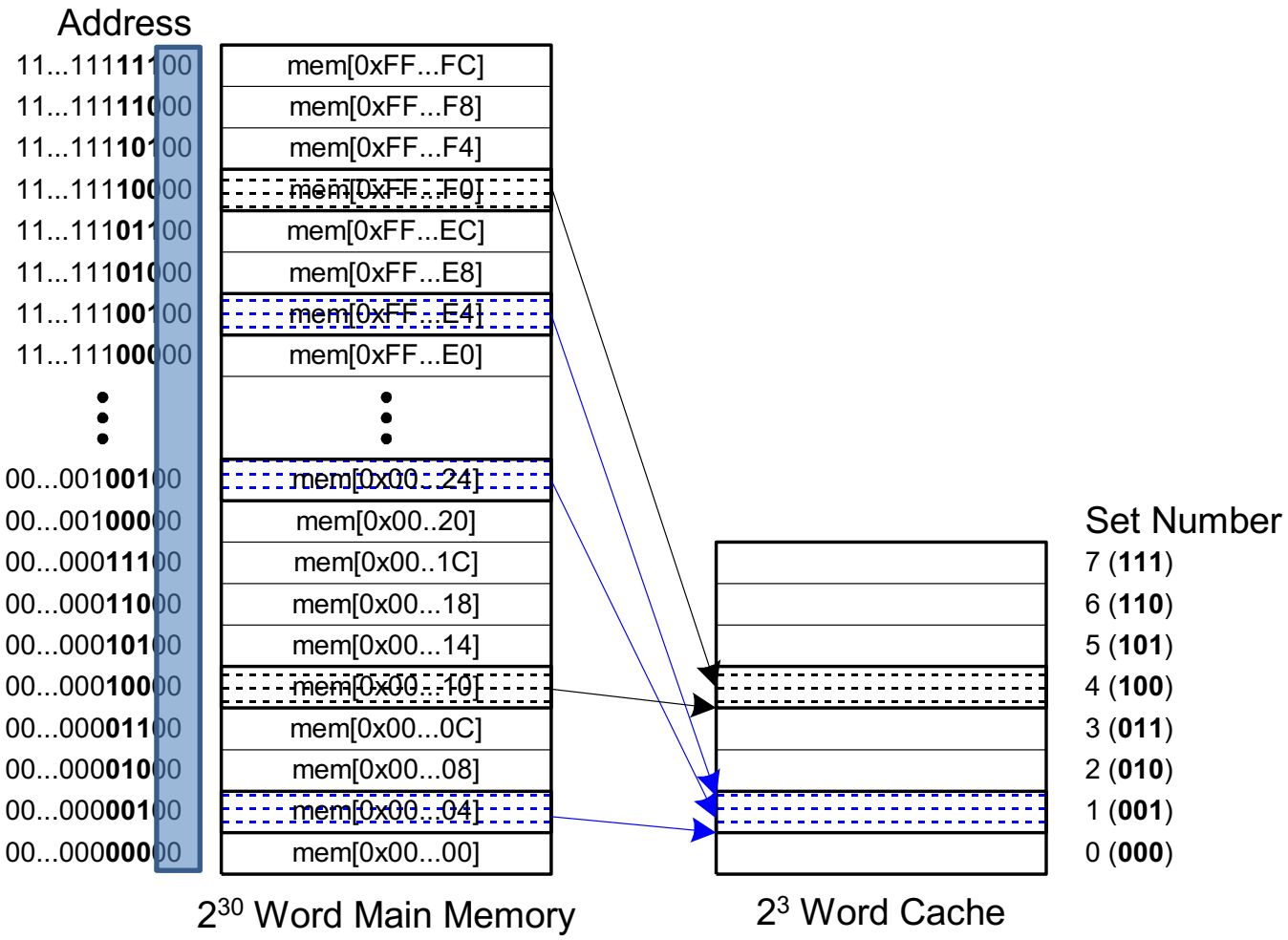
- Esaminiamo ogni tipologia con una cache con:
  - Capacity (C = 8 words)
  - Block size (b = 1 word)
  - Quindi, il numero di blocchi è (B = 8)

  Ovviamente una memoria del genere è ridicolmente piccola, serve solo per scopi didattici

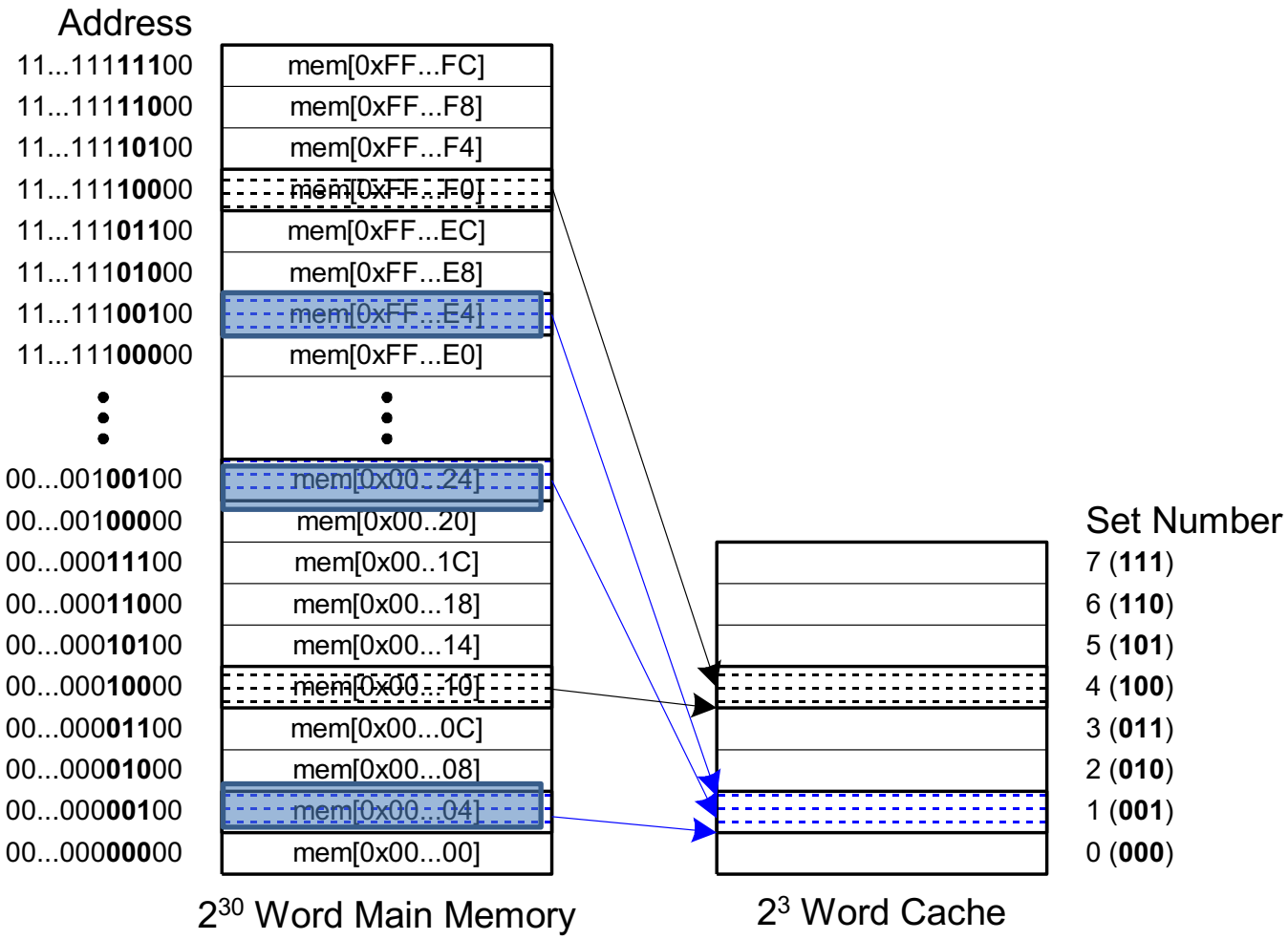
- Capacity: C
- Block size: b
- Number of blocks in cache: $B = C/b$
- Number of blocks in a set: N
- Number of sets: $S = B/N$

Organization	Number of Ways (N)	Number of Sets ($S = B/N$)
Direct Mapped	1	B
N-Way Set Associative	$1 < N < B$	B / N
Fully Associative	B	1

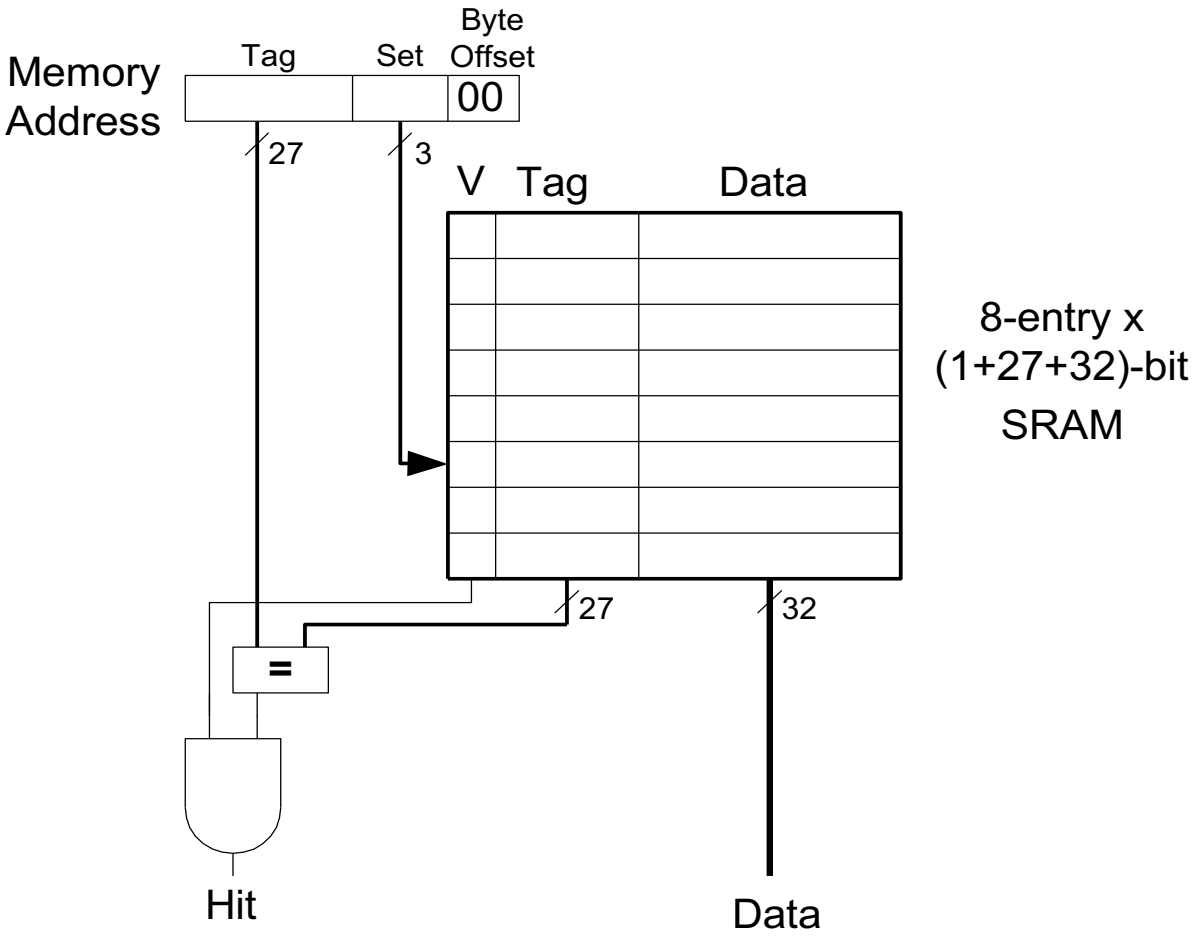
Direct Mapped Cache



Direct Mapped Cache



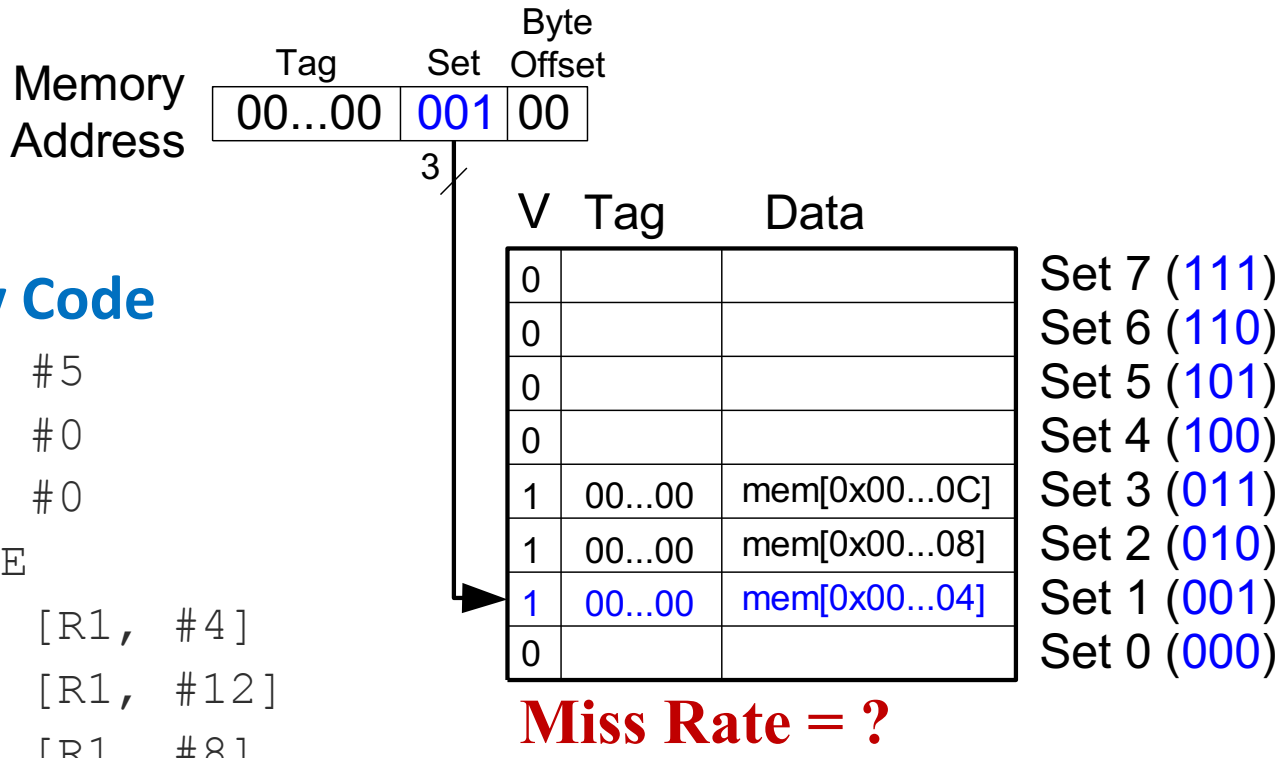
Direct Mapped Cache Hardware



Direct Mapped Cache Performance

ARM Assembly Code

```
        MOV R0, #5
        MOV R1, #0
LOOP    CMP R0, #0
        BEQ DONE
        LDR R2, [R1, #4]
        LDR R3, [R1, #12]
        LDR R4, [R1, #8]
        SUB R0, R0, #1
        B   LOOP
DONE
```

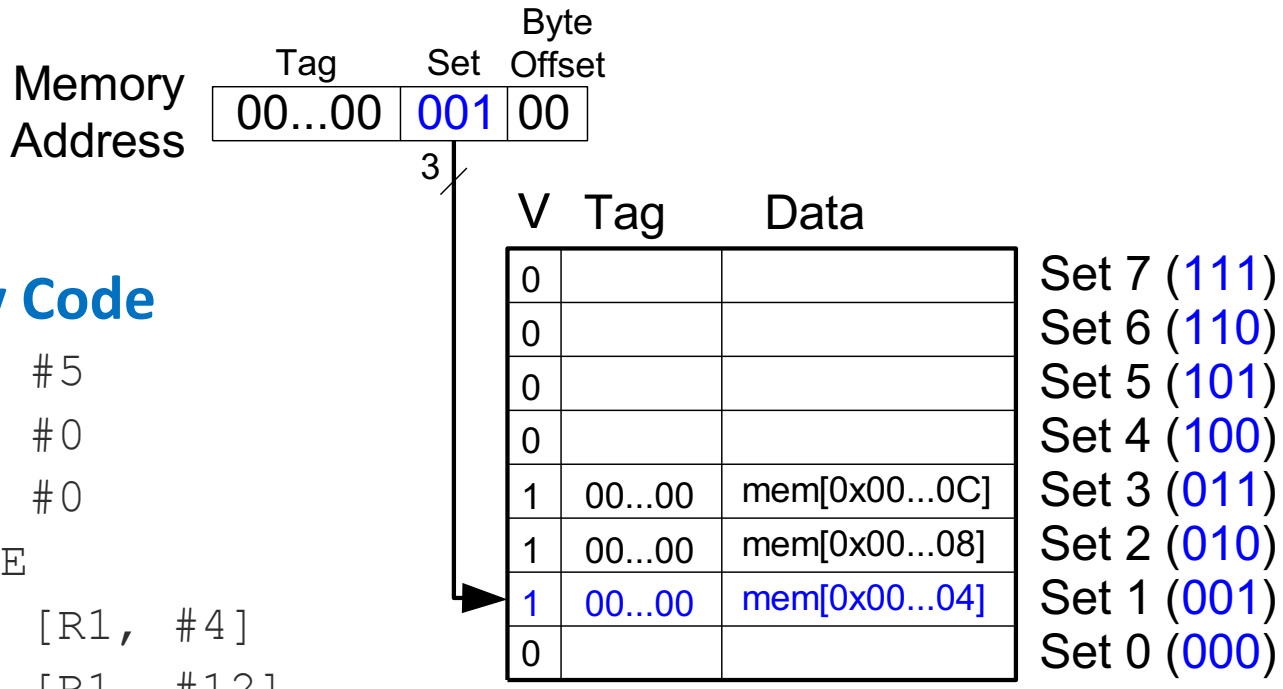


Miss Rate = ?

Direct Mapped Cache Performance

ARM Assembly Code

```
        MOV R0, #5
        MOV R1, #0
LOOP    CMP R0, #0
        BEQ DONE
        LDR R2, [R1, #4]
        LDR R3, [R1, #12]
        LDR R4, [R1, #8]
        SUB R0, R0, #1
        B   LOOP
DONE
```

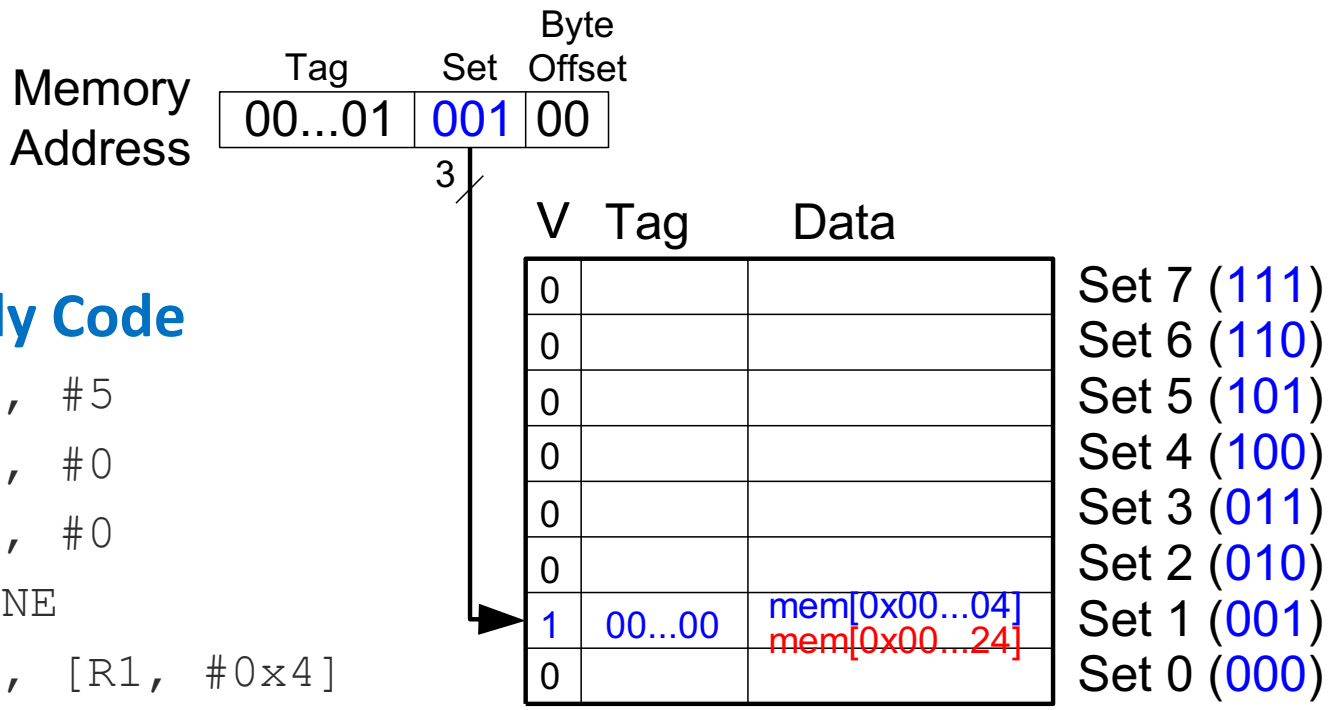


Miss Rate = 3/15
= 20%

Temporal Locality

Compulsory Misses

Direct Mapped Cache: Conflict

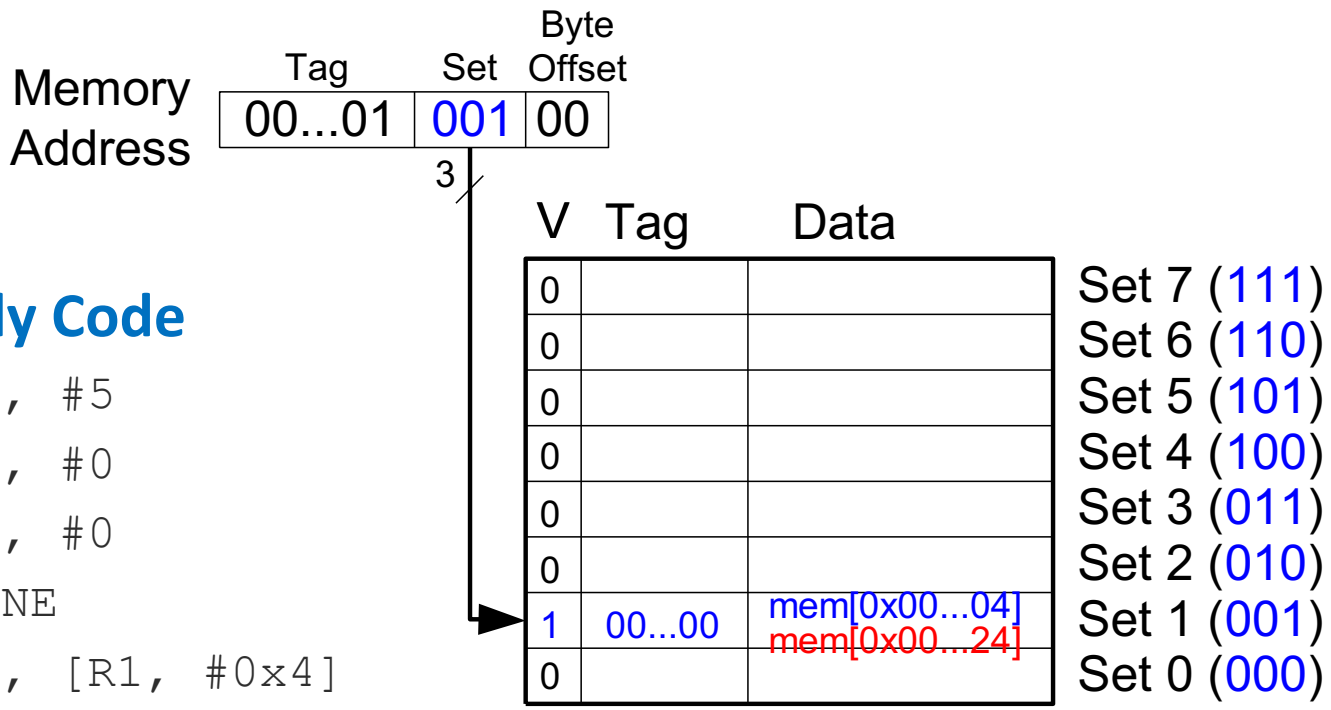


ARM Assembly Code

```
        MOV R0, #5
        MOV R1, #0
LOOP    CMP R0, #0
        BEQ DONE
        LDR R2, [R1, #0x4]
        LDR R3, [R1, #0x24]
        SUB R0, R0, #1
        B   LOOP
DONE
```

Miss Rate = ?

Direct Mapped Cache: Conflict



ARM Assembly Code

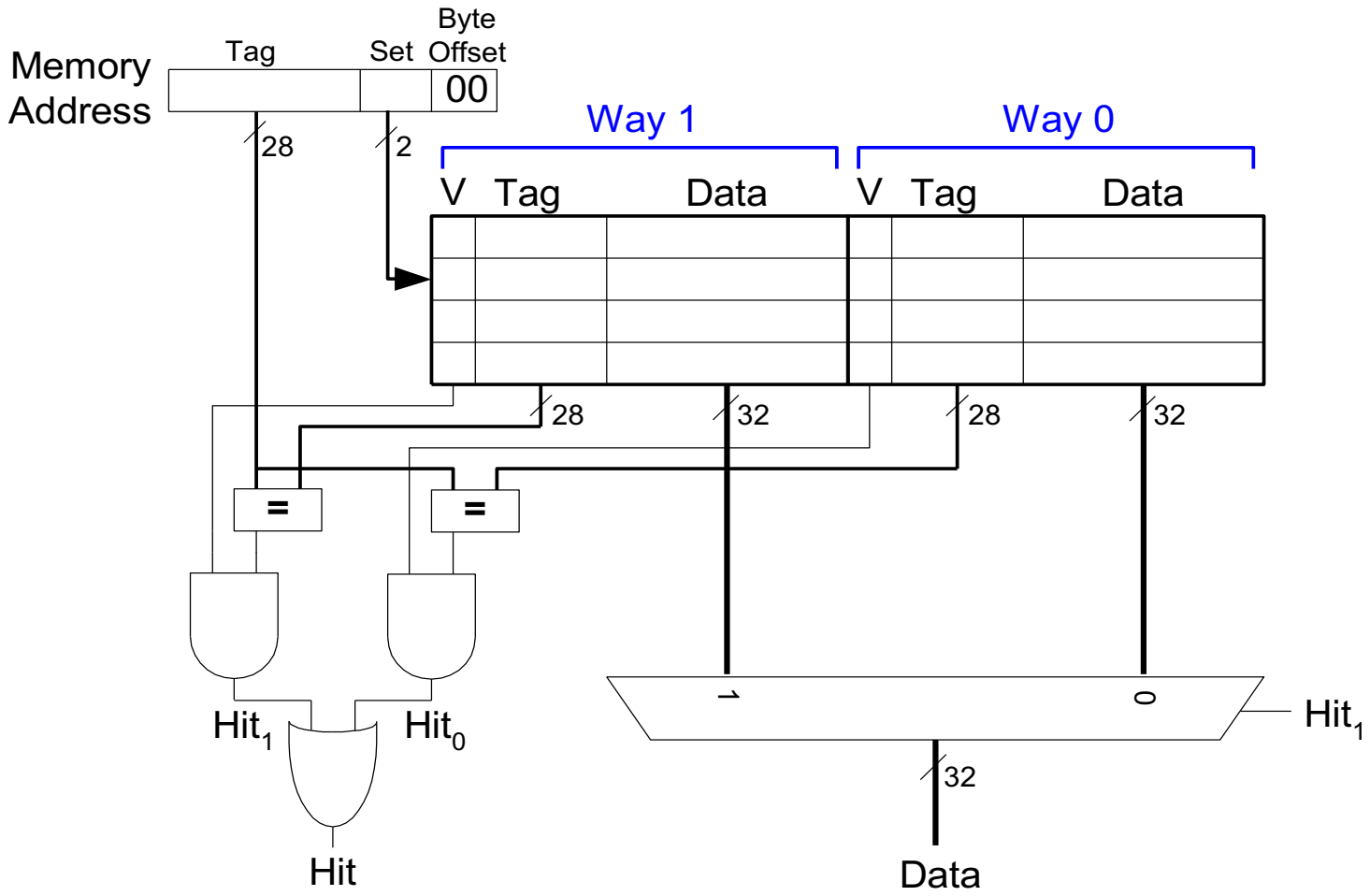
```
        MOV R0, #5
        MOV R1, #0
LOOP    CMP R0, #0
        BEQ DONE
        LDR R2, [R1, #0x4]
        LDR R3, [R1, #0x24]
        SUB R0, R0, #1
        B   LOOP
DONE
```

Miss Rate = 10/10

= 100%

Conflict Misses

N-Way Set Associative Cache

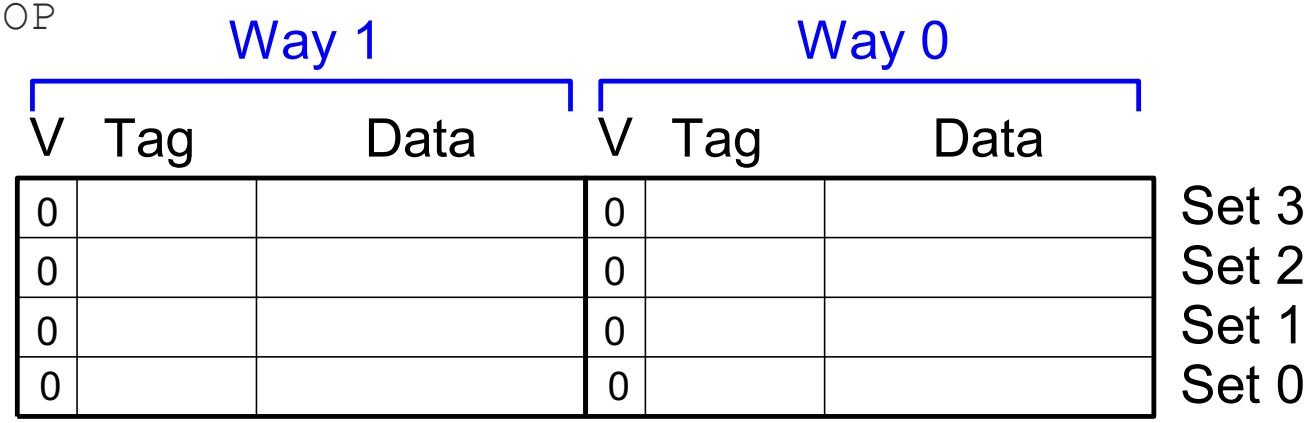


N-Way Set Associative Performance

ARM Assembly Code

```
        MOV R0, #5
        MOV R1, #0
LOOP    CMP R0, 0
        BEQ DONE
        LDR R2, [R1, #0x4]
        LDR R3, [R1, #0x24]
        SUB R0, R0, #1
        B   LOOP
DONE
```

Miss Rate = ?



N-Way Set Associative Performance

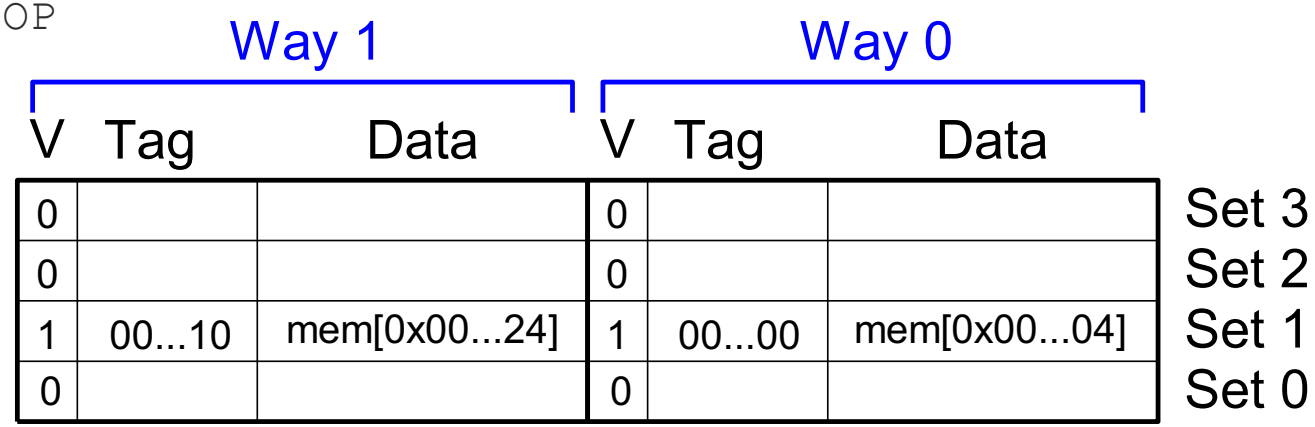
ARM Assembly Code

```
        MOV R0, #5
        MOV R1, #0
LOOP    CMP R0, 0
        BEQ DONE
        LDR R2, [R1, #0x4]
        LDR R3, [R1, #0x24]
        SUB R0, R0, #1
        B   LOOP
DONE
```

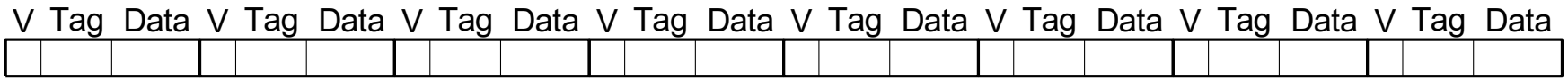
Miss Rate = 2/10

= 20%

Associativity reduces conflict misses



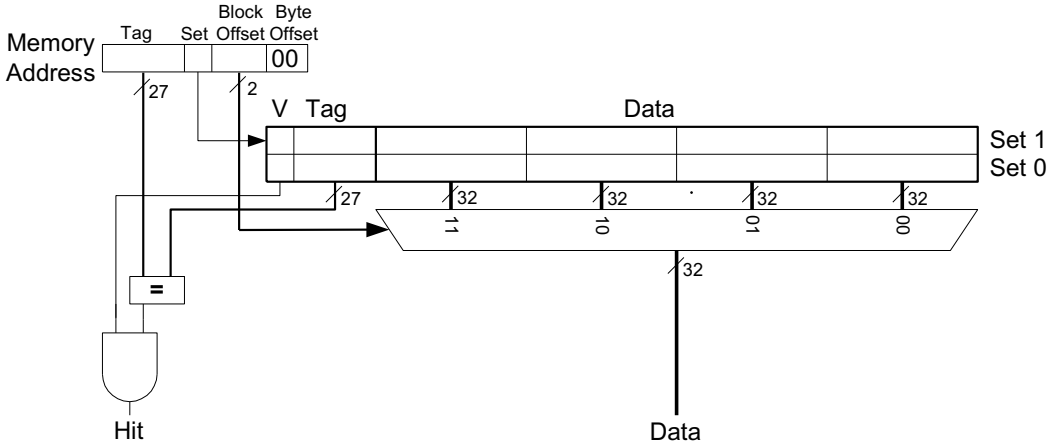
Fully Associative Cache



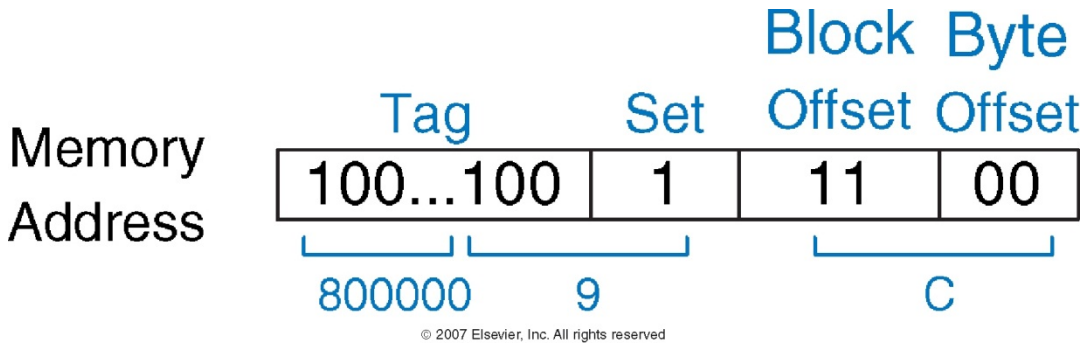
- Consente la massima flessibilità nella scelta della locazione nella cache in cui posizionare un blocco di memoria
- Lo spazio della cache può essere utilizzato in modo più efficiente e questo comporta una riduzione dei miss di conflitto
- Il costo della ricerca di un blocco in cache è notevole, mentre nel caso di indirizzamento diretto è praticamente nullo
  - Una operazione di questo tipo si chiama ricerca associativa o per contenuto, in quanto corrisponde all'operazione di ricerca di un item in un elenco
- E' più costosa da costruire

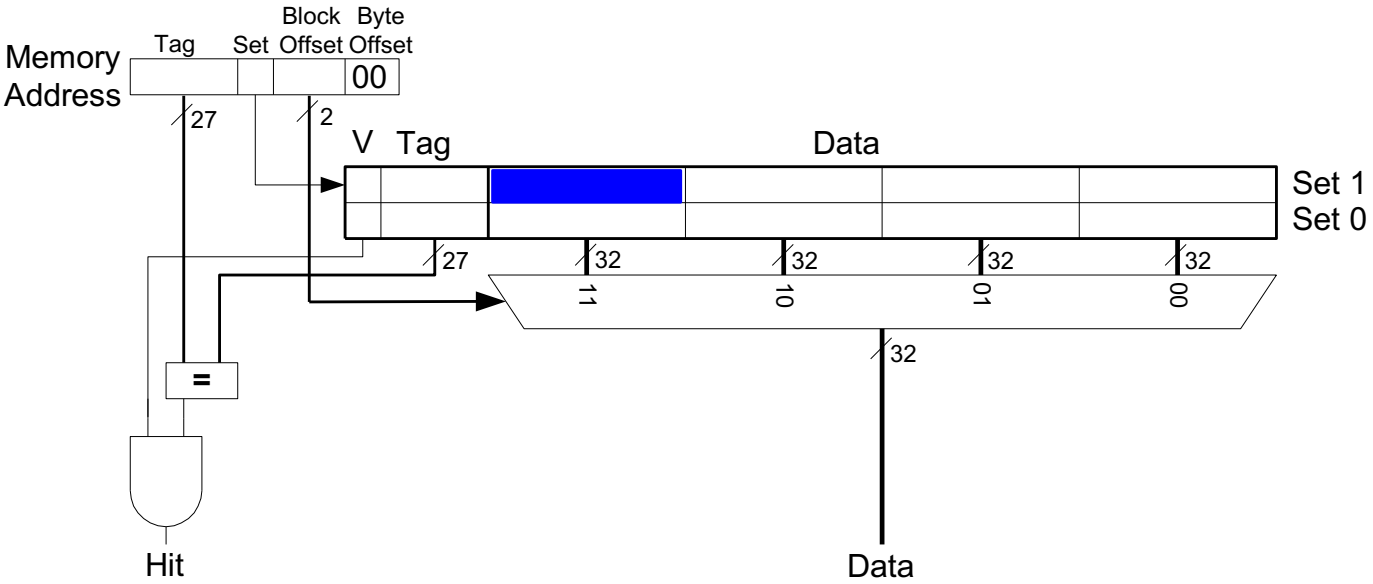
Come sfruttare la località spaziale?

- Incrementare il block size:
  - Block size, ***b* = 4 words**
  - *C* = 8 words
  - Direct mapped (1 block per set)
  - Number of blocks, ***B* = 2** (*C*/*b* = 8/4 = 2)



Cache with Larger Block Size





Direct Mapped Cache Performance

ARM assembly code

```
        MOV R0, #5
        MOV R1, #0
LOOP    CMP R0, 0
        BEQ DONE
        LDR R2, [R1, #4]
        LDR R3, [R1, #12]
        LDR R4, [R1, #8]
        SUB R0, R0, #1
        B   LOOP
DONE
```

Miss Rate = ?

Direct Mapped Cache Performance

ARM assembly code

```
        MOV R0, #5
        MOV R1, #0
LOOP    CMP R0, 0
        BEQ DONE
        LDR R2, [R1, #4]
        LDR R3, [R1, #12]
        LDR R4, [R1, #8]
        SUB R0, R0, #1
        B   LOOP
DONE
```

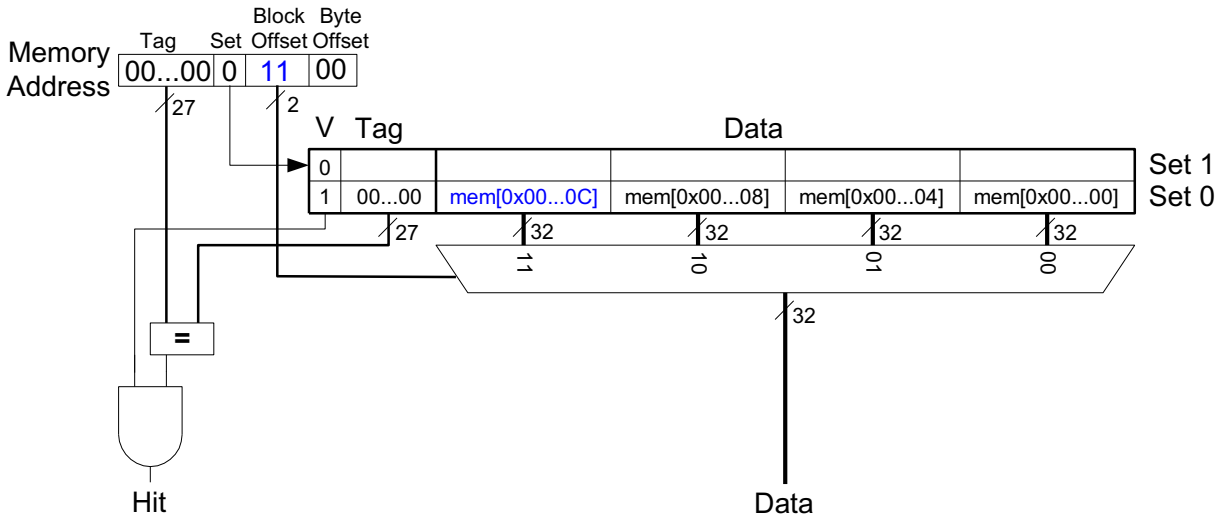
Miss Rate = 1/15

= 6.67%

Larger blocks

reduce compulsory misses

through spatial locality



Cache Organization Recap

- Capacity: C
- Block size: b
- Number of blocks in cache: $B = C/b$
- Number of blocks in a set: N
- Number of sets: $S = B/N$

Organization	Number of Ways (N)	Number of Sets ($S = B/N$)
Direct Mapped	1	B
N-Way Set Associative	$1 < N < B$	B / N
Fully Associative	B	1

Il bit di validità

- Un ulteriore bit di controllo, chiamato ***bit di validità***, è necessario per ogni blocco.
- **Questo bit indica se il blocco contiene o meno dati validi**.
- Non deve però essere confuso con il ***bit di modifica***, menzionato in precedenza, il bit di modifica, che indica se il blocco è stato modificato o meno durante il periodo in cui è stato nella cache, serve soltanto nei sistemi che non fanno uso del metodo ***write-through***.

I bit di validità dei blocchi che si trovano nella cache vengono tutti posti a 0 nell'istante iniziale di accensione del sistema e, in seguito, ogni volta che nella memoria principale vengono caricati programmi e dati nuovi dal disco.

3. Si riportino i valori dei flag V C N Z risultanti dalla somma delle parole a 32 bit #800348CC e #8CFFFFFA:

   V C N Z: 1 | 0 0

4. Il seguente codice assembly rappresenta la funzione fattoriale, ma ci sono due errori. Scrivere nell'apposito riquadro il codice corretto. [Si ricordi che SP sta per stack pointer, LR per link register e PC per program counter.]

```
FACTORIAL
  PUSH R0, LR
  CMP R0, #1
  BEQ ELSE
  MOV R0, #1
  ADD SP, SP, #8
  MOV PC, LR
ELSE
  SUB R0, R0, #1
  B FACTORIAL
  POP R1, LR
  MUL R0, R1, R0
  MOV PC, LR
```

4. Si consideri il seguente programma assembly:

MOVE R0, #2
MOVE R1, #0xBEF02C40
MOVE R2, #0x1
LOOP
LDR R3, [R1], #4
ADD R2, R2, R3
SUBS R0, R0, #1
BPL LOOP

Address	Data
BEF02C4C	**00000001**
BEF02C48	**00000005**
BEF02C44	**0000001A**
BEF02C40	**00000002**

Indicare esadecimale il valore di R2 al termine dell'esecuzione considerando la configurazione in memoria riportata.

R2: