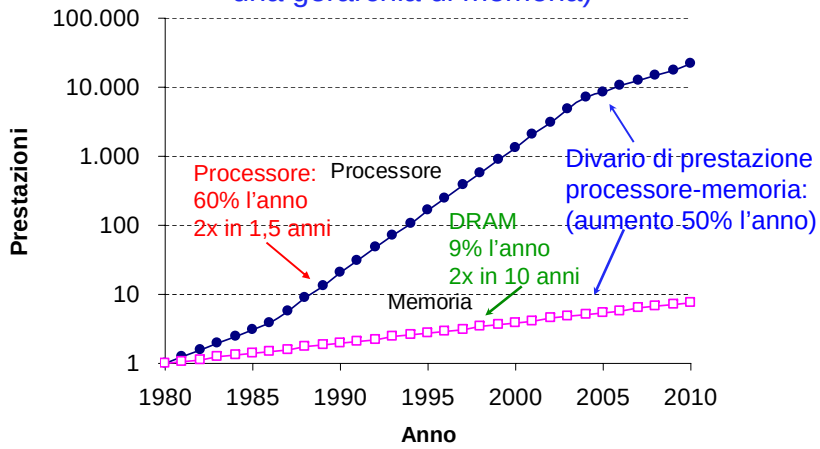
La gerarchia di memorie (1)

Architetture Avanzate dei Calcolatori

Valeria Cardellini

Divario delle prestazioni processore-memoria

Soluzione: memorie cache più piccole e veloci tra processore e DRAM (creazione di una gerarchia di memoria)

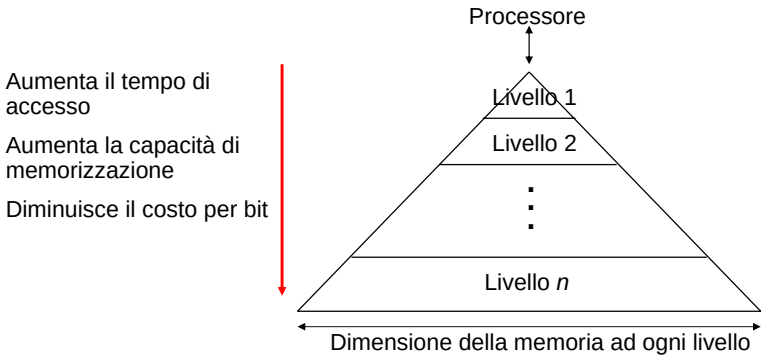


Obiettivo

- Illusione di avere a disposizione una memoria che sia:
  - grande
  - veloce (ritardo della memoria simile a quello del processore)
  - economica
- Osservazioni:
  - Le memorie di grandi dimensioni sono lente
  - Le memorie veloci hanno dimensioni piccole
  - Le memorie veloci costano (molto) più di quelle lente
  - Non esiste una memoria che soddisfi simultaneamente tutti i requisiti!
- Come creare una memoria che sia grande, economica e veloce (per la maggior parte del tempo)?
  - Gerarchia
  - Parallelismo

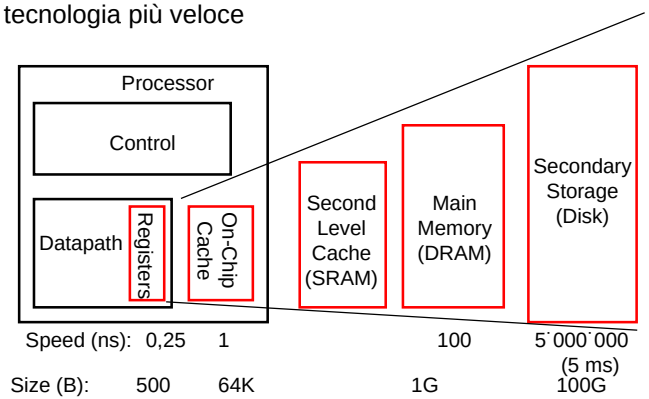
La soluzione: gerarchia di memorie

- Non un livello di memoria…
- Ma una **gerarchia** di memorie
  - Ognuna caratterizzata da differenti *tecnologie*, *costi*, *dimensioni*, e *tempi di accesso*



La soluzione: gerarchia di memorie (2)

- Obiettivi della gerarchia di memorie:
  - Fornire una quantità di memoria pari a quella disponibile nella tecnologia più economica
  - Fornire una velocità di accesso pari a quella garantita dalla tecnologia più veloce



Esempio: Apple iMac G5

Gestito dal compilatore — Gestito dall'hardware — Gestito da SO, hardware, applicazioni

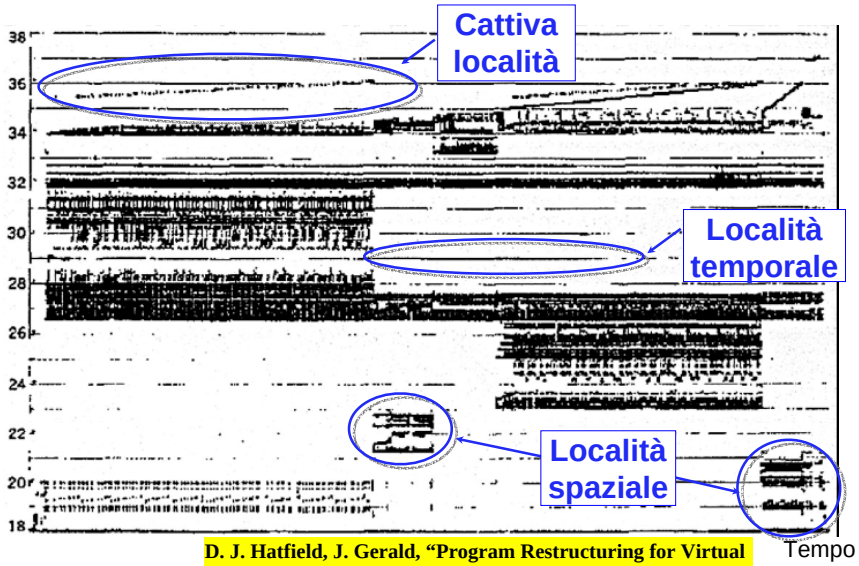
	Reg	L1 Inst	L1 Data	L2	DRAM	Disk
Size	1K	64K	32K	512K	256M	80G
Latency Cycles, Time	1, 0.6 ns	3, 1.9 ns	3, 1.9 ns	11, 6.9 ns	88, 55 ns	10^7, 12 ms

iMac G5 1.6 GHz

Principio di località

- Alla base della gerarchia di memoria vi è il principio di ***località***
- Esistono due tipi differenti di località
- Località ***temporale*** (nel tempo):
  - Se un elemento di memoria (dato o istruzione) è stato acceduto, tenderà ad essere acceduto nuovamente entro breve tempo
  - *Caso tipico*: le istruzioni ed i dati entro un ciclo saranno acceduti ripetutamente
- Località ***spaziale*** (nello spazio):
  - Se un elemento di memoria (dato o istruzione) è stato acceduto, gli elementi i cui indirizzi sono vicini tenderanno ad essere acceduti entro breve tempo
  - *Casi tipici:* gli accessi agli elementi di un array presentano un'elevata *località spaziale*; nell'esecuzione di un programma è altamente probabile che la prossima istruzione sia contigua a quella in esecuzione

Principio di località (2)



D. J. Hatfield, J. Gerald, "Program Restructuring for Virtual Memory", IBM Systems Journal 10(3): 168-192, 1971

Principio di località (3)

- La località è fortemente dipendente dall'applicazione
  - Alta (sia temporale che spaziale) per cicli interni di breve lunghezza che operano su dati organizzati in vettori
  - Bassa nel caso di ripetute chiamate a procedure
- In alcune applicazioni i dati hanno località di un solo tipo
  - Es.: dati di tipo streaming in elaborazione video (non hanno località temporale)
  - Es.: coefficienti usati in elaborazioni di segnali o immagini (si usano ripetutamente gli stessi coefficienti, non c'è località spaziale)

Livelli nella gerarchia di memorie

- Basandosi sul principio di località, la memoria di un calcolatore è realizzata come una gerarchia di memorie
- **Registri**
  - La memoria più veloce, intrinsecamente parte del processore
  - Gestiti dal compilatore (che alloca le variabili ai registri, gestisce i trasferimenti allo spazio di memoria)
- **Cache di primo livello**
  - Sullo stesso chip del processore (L1 cache), tecnologia SRAM
  - I trasferimenti dalle memorie di livello inferiore sono completamente gestiti dall'hardware
  - Di norma, la cache è trasparente al programmatore e al compilatore (vi sono delle eccezioni che vedremo più avanti!)
  - Può essere *unificata* (un'unica cache sia per dati che per istruzioni) oppure possono esserci cache *separate* per istruzioni e dati (**I-cache** e **D-cache**)

Livelli nella gerarchia di memorie (2)

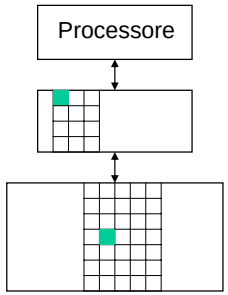
- **Cache di secondo (e terzo) livello**
  - Quando esiste, può essere sia sullo stesso chip del processore (solo L2 cache), sia su un chip separato; tecnologia SRAM
  - Il numero dei livelli di cache e delle loro dimensioni dipendono da vincoli di prestazioni e costo
  - Come per la cache di primo livello, i trasferimenti dalla memoria di livello inferiore sono gestiti dall'hardware e la cache è trasparente al programmatore e al compilatore
- **Memoria RAM**
  - Di solito in tecnologia DRAM (SDRAM)
  - I trasferimenti dalle memorie di livello inferiore sono gestiti dal sistema operativo (memoria virtuale) e dal programmatore

Livelli nella gerarchia di memorie (3)

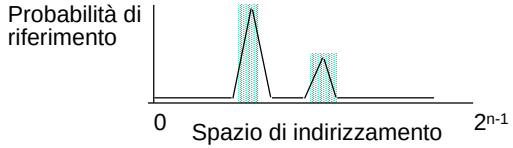
- Livelli di memoria *inclusivi*
  - Un livello superiore della gerarchia (più vicino al processore) contiene un sottoinsieme di informazioni dei livelli inferiori
  - Tutte le informazioni sono memorizzate nel livello più basso
  - Solo il livello massimo di cache (L1 cache) è acceduto direttamente dal processore
- *Migrazione* delle informazioni fra livelli della gerarchia
  - Le informazioni vengono di volta in volta copiate solo tra livelli adiacenti

Migrazione delle informazioni

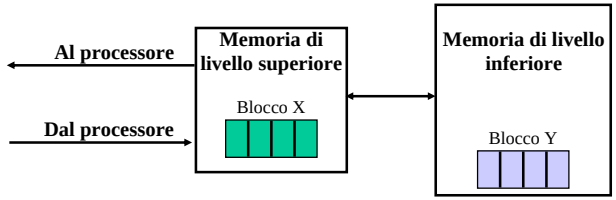
- ***Blocco***: la minima unità di informazione che può essere trasferita tra due livelli adiacenti della gerarchia
  - La dimensione del blocco influenza direttamente la larghezza (banda) del bus
- ***Hit*** (successo): l'informazione richiesta è presente nel livello acceduto
- ***Miss*** (fallimento): l'informazione richiesta non è presente nel livello acceduto
  - Deve essere acceduto il livello inferiore della gerarchia per recuperare il blocco contenente l'informazione richiesta



Come sfruttare il principio di località



- Per sfruttare la località temporale:
  tenere i blocchi acceduti più frequentemente vicino al processore
- Per sfruttare la località spaziale:
  spostare blocchi contigui tra livelli della gerarchia



Strategia di utilizzo della cache

- Cache strutturata in ***linee***
  - Ogni linea contiene un blocco (più parole: da 4 a 64 byte)
- La prima volta che il processore richiede un dato in memoria si ha un ***cache miss***
  - Il blocco contenente il dato viene trasferito dal livello inferiore di memoria e viene copiato anche nella cache
- Le volte successive, quando il processore richiede l'accesso alla memoria
  - Se il dato è presente in un blocco contenuto nella cache, la richiesta ha successo ed il dato viene passato direttamente al processore
    - Si verifica un ***cache hit***
  - Altrimenti la richiesta fallisce ed il blocco contenente il dato viene caricato anche nella cache e passato al processore
    - Si verifica un ***cache miss***
- Obiettivo: aumentare quanto più possibile il tasso di cache hit

Alcune definizioni

- ***Hit rate*** (frequenza dei successi): frazione degli accessi in memoria risolti nel livello superiore della gerarchia di memoria
  Hit rate = numero di hit / numero di accessi in memoria
- ***Miss rate*** (frequenza dei fallimenti): 1 - hit rate
- ***Hit time*** (tempo di successo): tempo di accesso alla cache in caso di successo
- ***Miss penalty*** (penalità di fallimento): tempo per trasferire il blocco dal livello inferiore della gerarchia
- ***Miss time***: tempo per ottenere l'elemento in caso di miss
  miss time = miss penalty + hit time
- Tempo medio di accesso alla memoria (**AMAT**)

$$AMAT = c + (1-h) \cdot m$$

c: hit time h: hit rate
1-h: miss rate m: miss penalty

Le decisioni per la gerarchia di memorie

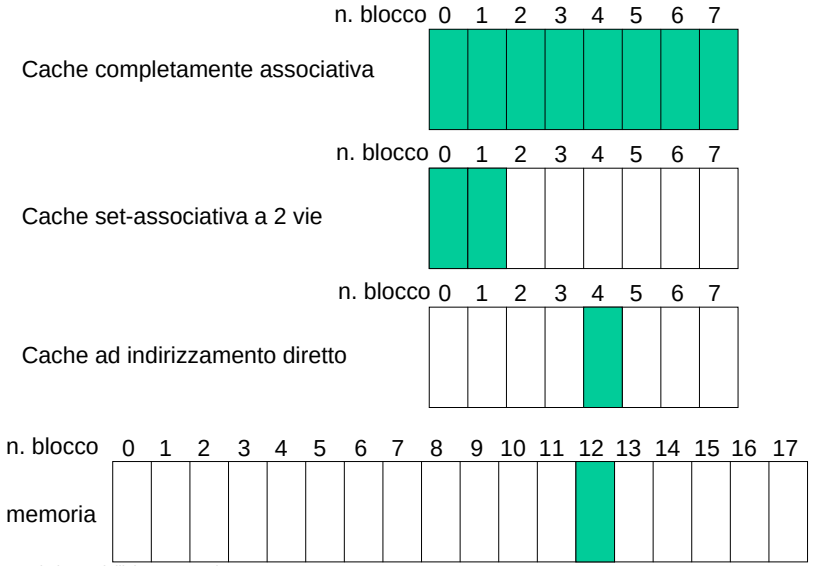
Quattro decisioni da prendere:

1. Dove si può portare un blocco nel livello gerarchico più alto (***posizionamento del blocco*** o block placement)
2. Come si trova un blocco nel livello gerarchico più alto (***identificazione del blocco*** o block identification)
   - Le prime due decisioni sono collegate e rappresentano le tecniche di indirizzamento di un blocco
3. Quale blocco nel livello gerarchico più alto si deve sostituire in caso di miss (***algoritmo di sostituzione*** o block replacement)
4. Come si gestiscono le scritture (***strategia di aggiornamento*** o write strategy)

Posizionamento del blocco

- Tre categorie di organizzazione della cache in base alla restrizioni sul posizionamento del blocco in cache
- In una sola posizione della cache:
  - cache ad ***indirizzamento diretto*** (a mappatura diretta o direct mapped cache)
- In un sottoinsieme di posizioni della cache:
  - cache ***set-associativa a N vie*** (set-associative cache)
- In una qualunque posizione della cache:
  - cache ***completamente associativa*** (fully-associative cache)

Posizionamento del blocco (2)



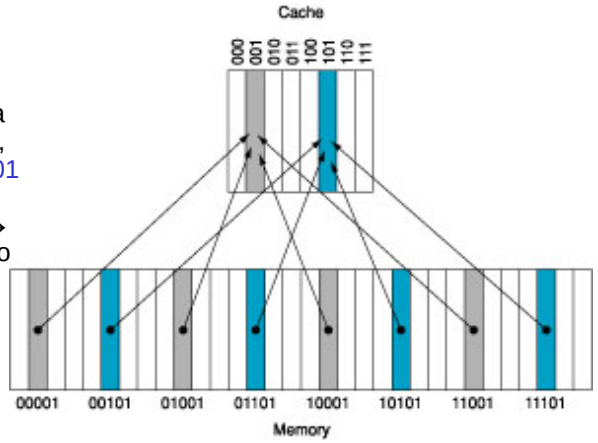
Cache ad indirizzamento diretto

- Ogni blocco nello spazio degli indirizzi trova il suo corrispondente in uno e un solo blocco in cache
  - N_B: numero di blocchi in cache
  - B_{AC}: indirizzo del blocco in cache
  - B_{AM}: indirizzo del blocco in memoria
  - $B_{AC} = B_{AM}$ modulo N_B
- L'indirizzo del blocco in cache (detto *indice* della cache) si ottiene usando i $\log_2(N_B)$ bit meno significativi dell'indirizzo del blocco in memoria
  - La definizione si modifica opportunamente se il blocco contiene più parole (vediamo come tra breve)
- Tutti i blocchi della memoria che hanno i $\log_2(N_B)$ bit meno significativi dell'indirizzo uguali vengono mappati sullo stesso blocco di cache

Cache ad indirizzamento diretto (2)

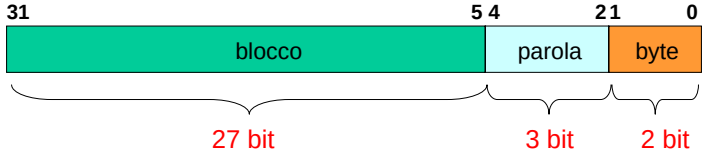
- Esempio di cache ad indirizzamento diretto con 8 blocchi

I blocchi di memoria con indirizzo 00001, 01001, 10001, 11001 hanno gli ultimi $\log_2 8 = 3$ bit uguali → mappati nello stesso blocco in cache



Esempio: organizzazione della memoria

- Indirizzi a 32 bit
- Parole di 4 byte
- Linee di cache di 32 byte (8 parole)
- Struttura dell'indirizzo
  - I 27 bit più significativi dell'indirizzo rappresentano il numero di blocco
  - I successivi 3 bit rappresentano il numero della parola all'interno del blocco
  - Gli ultimi 2 bit rappresentano il numero del byte all'interno della parola

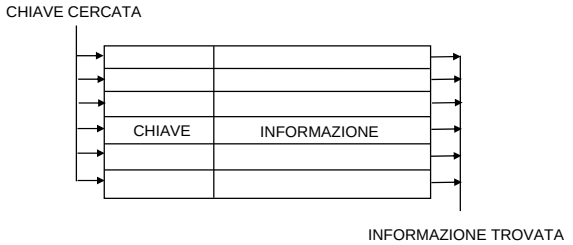


Ricerca di un blocco in cache

- Una cache contiene un sottoinsieme di blocchi di memoria di indirizzo non contiguo
- Quando il processore cerca una parola, non sa in quale posizione essa si possa trovare nella cache (se effettivamente c'è)
- Non c'è modo di risalire dall'indirizzo di un blocco di memoria alla sua posizione in cache
- Non è possibile utilizzare il normale meccanismo di indirizzamento della memoria
  - Si fornisce un indirizzo
  - Viene letto il dato che si trova all'indirizzo specificato
- Soluzione: si usa una **memoria associativa**

Memoria associativa

- Ciascun elemento è costituito da due parti: la *chiave* e l'*informazione*
- L'accesso ad un elemento viene effettuato non in base all'indirizzo ma a parte del suo contenuto (chiave)
- L'accesso *associativo* avviene in un unico ciclo
- Nel caso di una cache:
  - La chiave è il numero del blocco
  - L'informazione sono i byte del blocco

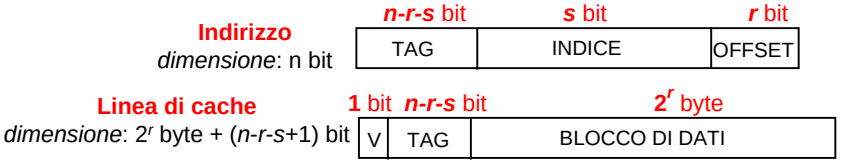


Contenuto di una linea di cache

- In una cache ad indirizzamento diretto ogni linea di cache include:
  - Il **bit di validità**: indica se i dati nella linea di cache sono validi
    - All'avvio, tutte le linee sono non valide (*compulsory miss*)
  - Il **tag** (etichetta): consente di individuare in modo univoco il blocco in memoria che è stato mappato nella linea di cache
  - Il blocco di dati vero e proprio, formato da più parole

Struttura dell'indirizzo e della linea di cache

- Spazio di memoria di 2^n byte, diviso in blocchi da 2^r byte
- Cache ad indirizzamento diretto di capacità pari a 2^s linee
- Si associa ad ogni blocco la linea di cache indicata dagli *s* bit meno significativi del suo indirizzo
  - Se il blocco è in cache deve essere in quella linea, e lì bisogna cercarlo
- Il tag permette di distinguere tra tutti i blocchi che *condividono* la stessa linea
- Il tag è dato dagli *n-r-s* bit più significativi dell'indirizzo
- Il tag è contenuto nella linea di cache



Esempio

- Stato iniziale

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

- Miss di $(10110)_2$
  indice = 110
  tag = 10

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory(10110_{two})
111	N		

Esempio (2)

- Miss di $(11010)_2$
  indice = 010
  tag = 11

Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

- Miss di $(10000)_2$
  indice = 000
  tag = 10

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Esempio (3)

- Miss di $(00011)_2$
  indice = 011
  tag = 00

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

- Miss di $(10010)_2$
  indice = 010
  tag = 10

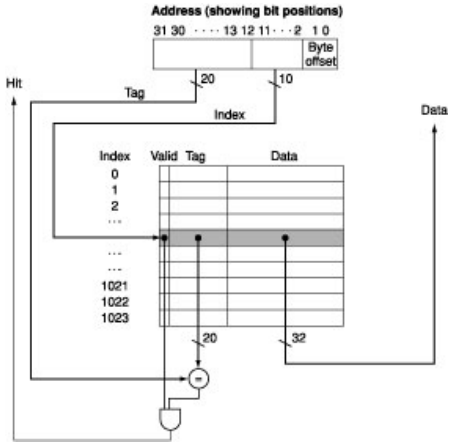
Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Un esempio più realistico

- Indirizzi a 32 bit
- Blocco di dati da 32 bit (4 byte)
- Cache con 1K di linee
- Quindi: n=32, r=2, s=10
- La struttura dell'indirizzo è:
  - I 20 bit più significativi dell'indirizzo rappresentano il tag
  - I successivi 10 bit rappresentano il numero del blocco in cache (l'indice della cache)
  - Gli ultimi 2 bit rappresentano il numero del byte all'interno del blocco (l'offset)

Accesso in cache

- Consideriamo l'esempio precedente
- Si confronta il tag dell'indirizzo con il tag della linea di cache individuata tramite l'indice
- Si controlla il bit di validità
- Viene segnalato l'hit al processore
- Viene trasferito il dato

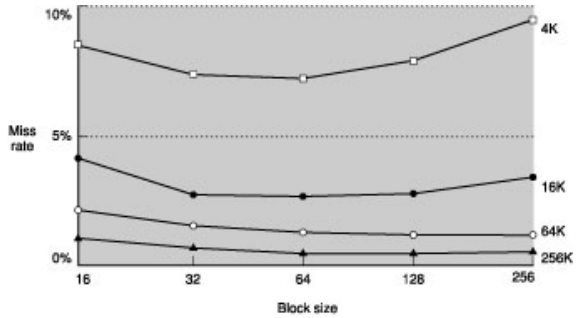


Dimensione cache ad indirizzamento diretto

- Quanti bit in totale sono necessari per una cache ad indirizzamento diretto?
  - Ciascuna linea di cache ha una dimensione pari a 2^r byte + (n-r-s+1) bit
  - Nella cache ci sono 2^s linee
  - Quindi occorrono $2^s(2^{r+3}+n\text{-}r\text{-}s+1)$ bit
  - Overhead = $2^s \cdot (n\text{-}r\text{-}s+1) / 2^s \cdot 2^{r+3} = (n\text{-}r\text{-}s+1)/2^{r+3}$
- Esempio
  - Indirizzi a 32 bit
  - Cache con 16 KB di dati e blocchi da 4 parole
  - Quindi:
    - n = 32
    - $s = \log_2(16\text{KB}/16\text{B}) = \log_2(2^{10}) = 10$
    - $r = \log_2(4*4) = \log_2(16) = 4$
  - Quindi occorrono $2^{10}(2^7+32\text{-}10\text{-}4+1)$ bit = 147 Kbit = 18,375 KB
  - Overhead = 2,375KB/16KB ~ 15%

Scelta della dimensione del blocco

- In generale, una dimensione ampia del blocco permette di sfruttare la località spaziale, ma…
  - Blocchi più grandi comportano un miss penalty maggiore
    - E' necessario più tempo per trasferire il blocco
- Se la dimensione del blocco è troppo grande rispetto alla dimensione della cache, aumenta il miss rate



Gestione di cache hit e cache miss

- In caso di hit: continua
  - Accesso al dato dalla memoria dati = cache dati
  - Accesso all'istruzione dalla memoria istruzioni = cache istruzioni
- In caso di miss:
  - Stallo del processore (come nel pipelining) in attesa di ricevere l'elemento dalla memoria
  - Invio dell'indirizzo al controller della memoria
  - Reperimento dell'elemento dalla cache
  - Caricamento dell'elemento in cache
  - Ripresa dell'esecuzione

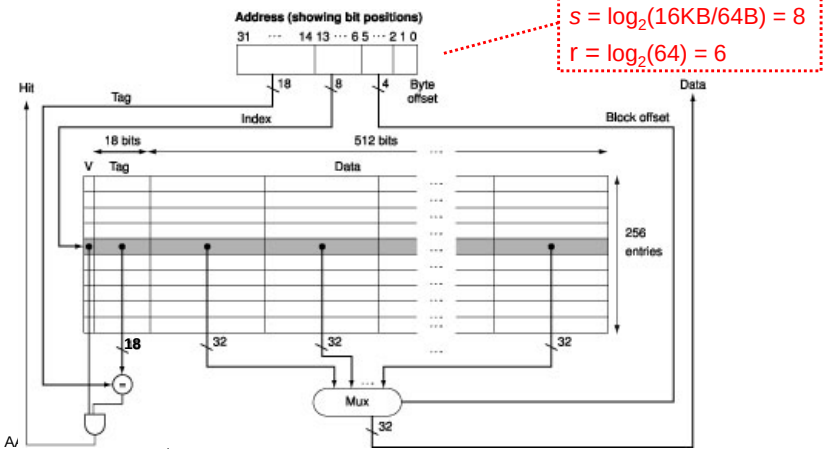
Esempio: il processore Intrisity FastMATH

- E' un processore embedded basato sull'architettura MIPS ed una semplice implementazione di cache
  - Cache istruzioni e cache dati separate, da 16 KB ciascuna e con blocchi di 16 parole (parole da 32 bit)

$n = 32$

$s = \log_2(16KB/64B) = 8$

$r = \log_2(64) = 6$



Sostituzione nelle cache ad indirizzamento diretto

- Banale: se il blocco di memoria è mappato in una linea di cache già occupata, si elimina il contenuto precedente della linea e si rimpiazza con il nuovo blocco
  - I miss sono dovuti a conflitti sull'indice di cache (*conflict miss*)
- La sostituzione non tiene conto della località temporale!
  - Il blocco sostituito avrebbe potuto essere stato usato molto di recente
  - Facile il fenomeno di ***thrashing***
- Vantaggi della cache ad indirizzamento diretto
  - Implementazione facile
  - Richiede poca area
  - E' veloce
- Svantaggi
  - Non molto efficiente per quanto riguarda la politica di sostituzione

Cache completamente associativa

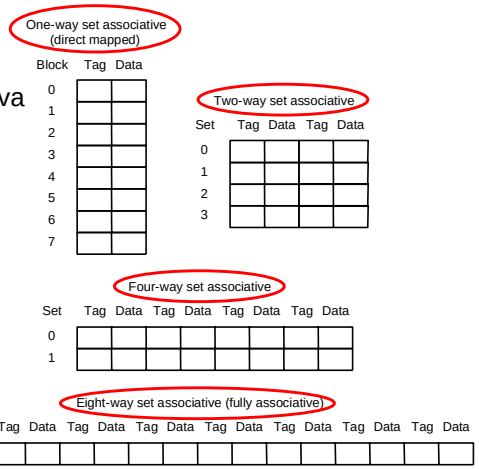
- E' altro estremo per il posizionamento del blocco in cache: nessuna restrizione sul posizionamento
- Ogni blocco di memoria può essere mappato in una qualsiasi linea di cache
  - Non ci sono conflict miss, ma i miss sono generati soltanto dalla capacità insufficiente della cache (*capacity miss*)
- Il contenuto di un blocco in cache è identificato mediante l'*indirizzo completo* di memoria
  - Il tag è costituito dall'indirizzo completo della parola
  - L'accesso è indipendente dall'indice di cache
- La ricerca viene effettuata mediante confronto in parallelo dell'indirizzo cercato con *tutti i tag*
- Problemi
  - Hardware molto complesso
  - Praticamente realizzabile solo con un piccolo numero di blocchi

Cache set-associativa a N vie

- Compromesso tra soluzione ad indirizzamento diretto e completamente associativa
- La cache è organizzata come insieme di set, ognuno dei quali contiene N blocchi (N: *grado di associatività*)
- Anche la memoria è vista come organizzata in set
  - Ogni set della memoria viene correlato ad uno e un solo set della cache con una filosofia ad indirizzamento diretto
- Ogni indirizzo di memoria corrisponde ad un unico set della cache (individuato tramite l'indice) e può essere ospitato in un blocco qualunque appartenente a quel set
  - Stabilito il set, per determinare se un certo indirizzo è presente in un blocco del set è necessario confrontare in parallelo tutti i tag dei blocchi nel set
- Si attenua il problema della collisione di più blocchi sulla stessa linea di cache

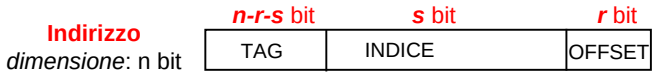
Confronto tra organizzazioni

- Una cache da 8 blocchi organizzata come
  - Ad indirizzamento diretto
  - Set-associativa a 2 vie
  - Set-associativa a 4 vie
  - Completamente associativa



Cache set-associativa a N vie (2)

- L'indirizzo di memoria ha la stessa struttura dell'indirizzo per la cache ad indirizzamento diretto
  - L'indice identifica il set



- A parità di dimensioni della cache, aumentando l'associatività di un fattore 2
  - Raddoppia il numero di blocchi in un set e si dimezza il numero di set
  - L'indice è più corto di un bit, il tag aumenta di un bit
  - Il numero dei comparatori raddoppia (i confronti sono in parallelo)
- Cache set-associativa a N vie: N comparatori

Esempio di cache set-associativa

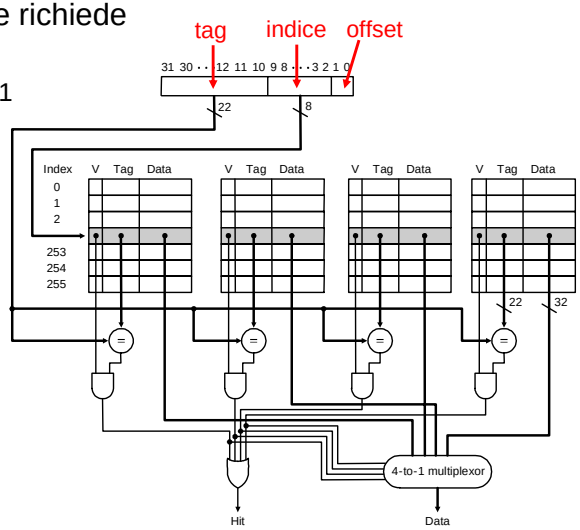
- Indirizzi di memoria a 32 bit
- Cache set-associativa a 4 vie da 4KB
- Blocco di dimensione pari a 1 parola (4 byte)
- Quindi:
  - *n* = *32*
  - Numero di blocchi = dim. cache/dim. blocco = 4KB/4B = 1K
  - Numero di set = dim. cache/(dim.blocco x N) = 4KB/(4B x 4) = = 256 = $2^8 \rightarrow s = \log_2(2^8) = 8$
  - $r = \log_2(4) = 2$

- Quindi la struttura dell'indirizzo è:
  - I 22 bit (*n-r-s*) più significativi dell'indirizzo rappresentano il tag
  - I successivi 8 bit (*s*) rappresentano il numero del set
  - Gli ultimi 2 bit (*r*) rappresentano il numero del byte all'interno del blocco

Cache set-associativa a 4 vie

- L'implementazione richiede
  - 4 comparatori
  - 1 multiplexer 4-to-1

- Tramite l'indice viene selezionato uno dei 256 set
- I 4 tag nel set sono confrontati in parallelo
- Il blocco viene selezionato sulla base del risultato dei confronti



Dimensione del tag e associatività

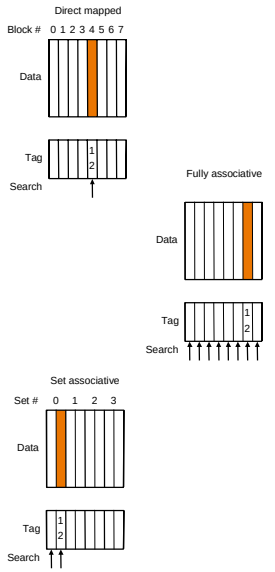
- Aumentando il grado di associatività
  - Aumenta il numero dei comparatori ed il numero di bit per il tag
- Esempio
  - Cache con 4K blocchi, blocco di 4 parole, indirizzo a 32 bit
  - $r = \log_2(4*4)=4 \rightarrow$ n-r = (32-4) = 28 bit per tag e indice
  - Cache ad indirizzamento diretto
    - $s = \log_2(4K) = 12$
    - Bit di tag totali = (28-12)*4K = 64K
  - Cache set-associativa a 2 vie
    - $s = \log_2(4K/2) = 11$
    - Bit di tag totali = (28-11)*2*2K = 68K
  - Cache set-associativa a 4 vie
    - $s = \log_2(4K/4) = 10$
    - Bit di tag totali = (28-10)*4*1K = 72K
  - Cache completamente associativa
    - s = 0
    - Bit di tag totali = 28*4K*1 = 112K

Identificazione del blocco e associatività

- Cache a mappatura diretta
  - Calcolo posizione del blocco in cache
  - Verifica del tag
  - Verifica del bit di validità

- Cache completamente associativa
  - Verifica dei tag di tutti blocchi in cache
  - Verifica del bit di validità

- Cache set-associativa a N vie
  - Identificazione dell'insieme in cache
  - Verifica di N tag dei blocchi nel set
  - Verifica del bit di validità



Incremento dell'associatività

- Principale vantaggio
  - Diminuzione del miss rate
- Principali svantaggi
  - Maggior costo implementativo
  - Incremento dell'hit time

- La scelta tra cache ad indirizzamento diretto, set-associativa e completamente associativa dipende dal costo dell'associatività rispetto alla riduzione del miss rate

Sostituzione nelle cache completamente associative e set-associative

- Quale blocco sostituire in caso di miss (capacity miss)?
  - In caso di cache completamente associativa: ogni blocco è un potenziale candidato per la sostituzione
  - In caso di cache set-associativa a N vie: bisogna scegliere tra gli N blocchi del set
- Politica di sostituzione **Random**
  - Scelta casuale
- Politica di sostituzione Least Recently Used (**LRU**)
  - Sfruttando la località temporale, il blocco sostituito è quello che non si utilizza da più tempo
  - Ad ogni blocco si associa un contatore all'indietro, che viene portato al valore massimo in caso di accesso e decrementato di 1 ogni volta che si accede ad un altro blocco
- Politica di sostituzione First In First Out (**FIFO**)
  - Si approssima la strategia LRU selezionando il blocco più vecchio anziché quello non usato da più tempo

Problema della strategia di scrittura

- Le scritture sono molto meno frequenti delle letture
- Le prestazioni sono migliori per le letture
  - La lettura può iniziare non appena è disponibile l'indirizzo del blocco, prima che sia completata la verifica del tag
  - La scrittura deve aspettare la verifica del tag
- In conseguenza di un'operazione di scrittura effettuata su un blocco presente in cache, i contenuti di quest'ultima saranno diversi da quelli della memoria di livello inferiore
  - Occorre definire una strategia per la gestione delle scritture
  - Strategia write-through
  - Strategia write-back

Strategia write-through

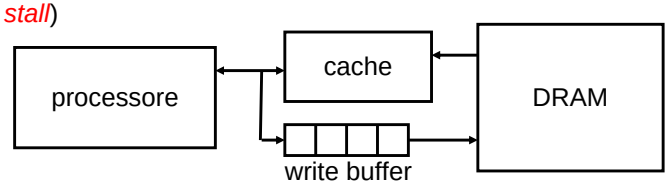
- Scrittura *immediata*: il dato viene scritto simultaneamente sia nel blocco della cache sia nel blocco contenuto nella memoria di livello inferiore
- Vantaggi
  - E' la soluzione più semplice da implementare
  - Si mantiene la **coerenza** delle informazioni nella gerarchia di memorie
- Svantaggi
  - Le operazioni di scrittura vengono effettuate alla velocità della memoria di livello inferiore $\rightarrow$ diminuiscono le prestazioni
  - Aumenta il traffico sul bus di sistema
- Alternative
  - Strategia **write-back**
  - Utilizzo di un **write buffer**

Strategia write-back

- Scrittura *differita*: i dati sono scritti solo nel blocco presente in cache; il blocco modificato viene trascritto nella memoria di livello inferiore solo quando viene sostituito
  - Subito dopo la scrittura, cache e memoria di livello inferiore sono *inconsistenti* (mancanza di coerenza)
  - Il blocco in cache può essere in due stati (*dirty bit*):
    - clean: non modificato
    - dirty: modificato
- Vantaggi
  - Le scritture avvengono alla velocità della cache
  - Scritture successive sullo stesso blocco alterano solo la cache e richiedono una sola scrittura nel livello inferiore di memoria
- Svantaggi
  - Ogni sostituzione del blocco (ad es. dovuto a read miss) può provocare un trasferimento in memoria

Strategia write-through con write buffer

- Buffer per la scrittura (*write buffer*) interposto tra la cache e la memoria di livello inferiore
  - Il processore scrive i dati in cache **e** nel write buffer
  - Il controller della memoria scrive il contenuto del write buffer in memoria: la scrittura avviene in modo asincrono e indipendente
- Il write buffer è gestito con disciplina FIFO
  - Numero tipico di elementi del buffer: 4
  - Efficiente se la frequenza di scrittura << 1/write cycle della DRAM
  - Altrimenti, il buffer può andare in saturazione ed il processore deve aspettare che le scritture giungano a completamento (*write stall*)



Write miss

- Le scritture possono indurre **write miss**: tentativi di scrivere in un blocco non presente in cache
- Soluzioni possibili:
  - **Write allocate** (anche detta *fetch-on-write*): il blocco viene caricato in cache e si effettua la scrittura (con una delle strategie viste prima)
  - **No-write allocate** (anche detta *write-around*): il blocco viene scritto direttamente nella memoria di livello inferiore, senza essere trasferito in cache
- In generale:
  - Le cache write-back tendono ad usare l'opzione write allocate
    - In base al principio di località, si spera che scritture successive coinvolgano lo stesso blocco
  - Le cache write-through tendono ad usare l'opzione no-write allocate
    - Le scritture devono comunque andare alla memoria di livello inferiore

Prestazioni delle cache

- Il tempo di CPU può essere suddiviso in due componenti

  CPU time = (CPU execution clock cycles + memory-stall clock cycles) × clock cycle time
- Gli stalli in attesa della memoria sono dovuti ad operazioni di lettura o scrittura

  memory-stall clock cycles = read-stall cycles + write-stall cycles
- Gli stalli per operazioni di lettura sono dati da:

  read-stall cycles = reads/program × read miss rate × read miss penalty
- Usando la strategia write-through con write buffer, gli stalli per operazioni di scrittura sono dati da:

  write-stall cycles = (writes/program × write miss rate × write miss penalty) + write buffer stalls

Prestazioni delle cache (2)

- Nella maggior parte delle organizzazioni di cache che adottano la strategia write-through, il miss penalty per scritture è uguale a quello per letture

  memory-stall clock cycles = memory access/program × miss rate × miss penalty

  O anche:

  memory-stall clock cycles = instructions/program × misses/instruction × miss penalty
- Impatto delle prestazioni della cache sulle prestazioni complessive del calcolatore (ricordando la legge di Amdahl…)
  - Se riduciamo il CPI (o aumentiamo la frequenza del clock) senza modificare il sistema di memoria: gli stalli di memoria occupano una frazione crescente del tempo di esecuzione

L1 cache

- Come si sceglie la cache di primo livello?
- La scelta è tra:
  - cache unificata
  - cache dati (D-cache) e istruzioni (I-cache) separate
- Cache separate possono essere ottimizzate individualmente
  - La I-cache ha un miss rate più basso della D-cache
  - La I-cache è di tipo read-mostly
    - Località spaziale molto buona (tranne che nel caso di chiamate a procedura molto frequenti)
  - La località della D-cache è fortemente dipendente dall'applicazione

Cause dei cache miss

- **Compulsory miss**
  - Detti anche miss al primo riferimento: al primo accesso il blocco non è presente in cache
  - Non dipendono dalle dimensioni e dall'organizzazione della cache
- **Capacity miss**
  - Durante l'esecuzione del programma, alcuni blocchi devono essere sostituiti
  - Diminuiscono all'aumentare delle dimensioni della cache
- **Conflict miss**
  - Può accadere di ricaricare più tardi nello stesso set i blocchi che si sono sostituiti
  - Diminuiscono all'aumentare delle dimensioni della cache e del grado di associatività
  - "Regola del 2:1": il miss rate di una cache ad indirizzamento diretto di N byte è circa pari a quello di una cache set-associativa a 2 vie di N/2 byte