Memorie cache

Gerarchia di memoria

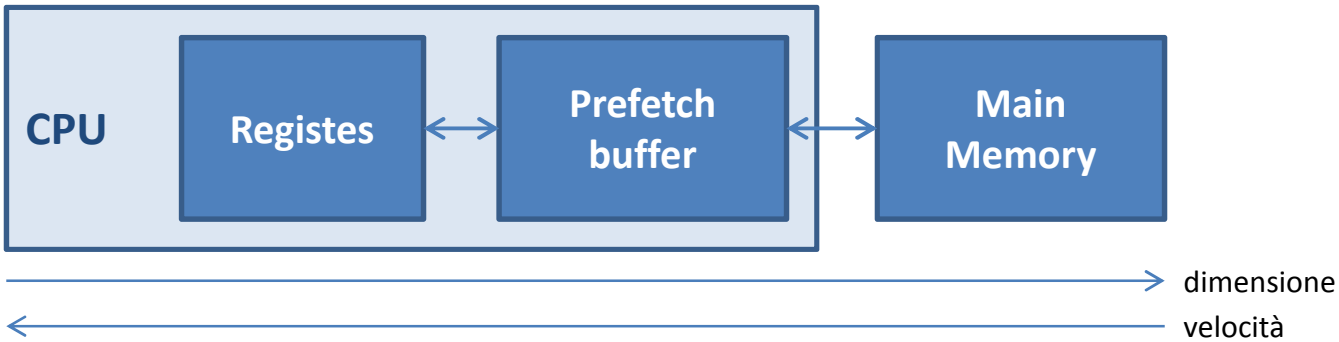
Principi

Mapping

Politiche

Gerarchia di memoria

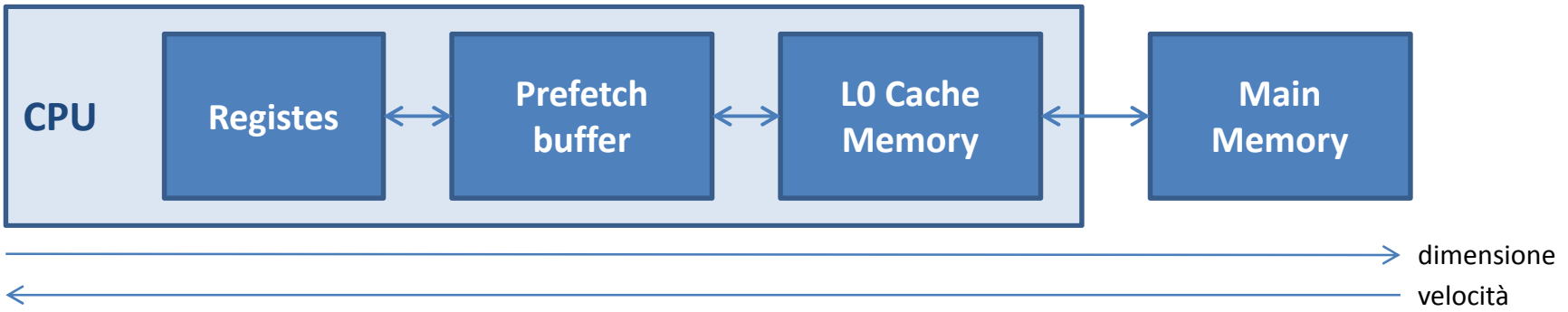
- **Fino a questo punto abbiamo considerato sistemi così strutturati**



- **Velocità di accesso**
  - Diminuisce con l'aumentare della dimensione della memoria
- **L'accesso alla memoria principale è il fattore limitante**
  - Alcuni cicli di clock del processore
  - Richiesto per ogni istruzione durante il ciclo di fetch
  - Limitazione dell'efficacia del pipelining
- **Sarebbe utile**
  - Avevre una memoria "temporanea" veloce, quindi piccola

Gerarchia di memoria

- **Aggiungendo una piccola memoria temporanea si ha la nuova architettura**



- **Tale memoria prende il nome di "Cache di livello zero" o "L0 Cache"**
  - Come intuibile, possono essere introdotte cache di livello 1, 2 ecc.
  - Tali memorie saranno
    - Progressivamente più "lontane" dalla CPU e più "vicine" alla memoria centrale
    - Progressivamente più grandi
    - Progressivamente più lente
- **Tutte le memorie cache si basano su**
  - Principi comuni
  - Architetture simili

Memoria cache

- **D'ora in poi considereremo una architettura dotata solo della cache L0**
  - I concetti che esporremo sono tuttavia generali
- **La memoria cache L0**
  - E' molto piccola, di solito qualche kilobyte
  - E' molto veloce, generalmente richiede un solo ciclo di clock
  - Non può contenere tutti i dati e tutte le istruzioni di un programma
  - Ne consegue che il suo contenuto deve essere periodicamente rimpiazzato
    - Sostituire il contenuto della cache richiede l'accesso alla memoria centrale
    - Lo stesso indirizzo di memoria della cache contiene dati diversi in istanti di tempo diversi
- **Ciò ha senso solo se**
  - La maggior parte degli accessi alla cache trovano i dati richiesti
  - Solo raramente è necessario sostituire il contenuto della cache
  - L'interfaccia di memoria della CPU genera indirizzi sempre allo stesso modo
    - Sia che il sistema disponga di una cache o più, sia che non ne disponga affatto
  - Il programmatore ed il compilatore non devono essere influenzati dalla presenza della memoria cache

Principi di località

- **L'analisi sperimentale delle sequenza di indirizzi (dati e istruzioni) generati dall'esecuzione di un gran numero di programmi ha evidenziato due proprietà note come "principi di località"**

- **Principio di località spaziale**
  - Se al tempo T il programma accede ad un indirizzo A, allora è molto probabile che al tempo T+1 il prgramma richieda un accesso ad un indirizzo vicino ad A

- **Principio di località temporale**
  - Se al tempo T il programma accede ad un indirizzo A, allora è molto probabile che esso richieda un nuovo accesso all'indirizzo A nell'immediato futuro di T

- **Ciò significa che un programma tende a lavorare su insiemi di dati che cambiano molto più lentamente rispetto al ciclo istruzione**
  - Si può immaginare una "finestra" che scorre lentamente sulla memoria

Problemi

- **Grazie a questi principi l'itroduzione delle memorie cache porta notevoli benefici in termini di tempo di esecuzione**

- **Tuttavia ci sono alcuni problemi da risolvere**
  - Qual'è la dimensione ideale di una cache?
  - Come scegliere la porzione di programma o di dati da caricare in cache?
  - Come correlare gli indirizzi del programma o dei dati in memoria centrale con quelli nella memoria cache?
  - Come e quando aggiornare i dati nella memoria centrale a seguito di una modifica della loro copia in cache?
  - Quando è necessario caricare una nuova porzione di programma o di dati come scegliere la porzione di memoria cache da liberare?

- **Vedremo la soluzione di tutti questi problemi partendo dalla struttura hardware delle memorie cache**

Interazione con il processore

- **Il processore genera una richiesta di accesso alla memoria**
  - Mediante l'interfaccia di lettura/scrittura o quella di fetch
- **Il comportamento della cache è differente a seconda che si tratti di una**
  - Lettura
  - Scrittura

- **Si possono avere due casi**
  - Cache hit: il dato richiesto si trova in cache
  - Cache miss: il dato richiesto non si trova in cache

- **Il comportamento della cache è diverso a seconda dei casi**
  - Lettura / hit
  - Lettura / miss
  - Scrittura / hit
  - Scrittura / miss

Interazione con il processore

- **Lettura / Hit**
  - La cache ha decodificato l'indirizzo e ha localizzato il dato
  - Il dato viene trasferito dalla cache alla CPU
  - Il ciclo di lettura termina
- **Lettura / Miss**
  - Un moduleo della cache carica dalla memoria centrale una "linea", cioè un blocco di dati
    - Viene caricato un intero blocco poiché per i principi di località è probabile che a breve si accederà allo stesso dato o a dati vicini
  - Dopo avere caricato il blocco, la cache trasferisce il dato alla CPU
- **Scrittura / Hit**
  - Analogo alla lettura, ma il trasferimento è nel verso opposto
    - Dopo una scrittura in cache si perde la "coerenza" tra cache e memoria centrale
    - La soluzione adottata è detta "politica di scrittura"
- **Scrittura / Miss**
  - Anche in questo caso si hanno diverse possibilità
    - La soluzione adottata è detta "politica di scrittura"

Politiche di scrittura

- **Quando un dato presente in cache viene aggiornato**
  - La memoria cache deve essere aggiornata
  - La memoria centrale deve essere aggiornata
- **Write-through**
  - La memoria centrale e la cache sono scritte simultaneamente
    - Facile da implementare
    - Il processore deve rispettare le tempistiche della memoria centrale lenta
    - Possono verificarsi scritture superflue
- **Write-back**
  - La memoria centrale viene aggiornata solo quando in cache è necessario sotituire un blocco che è stato modificato
    - È necessario un bit aggiuntivo, detto dirty bit, che viene scritto ogniqualvolta si aggiorna il vontenuto di un blocco in cache
    - Riduce significativamente il numero di accessi alla memoria centrale lenta
- **Se il dato non è presente in cache**
  - Si carica la pagina che contiene il dato
  - A questo punto la situazione è analoga a quanto appena visto

Politiche di sostituzione

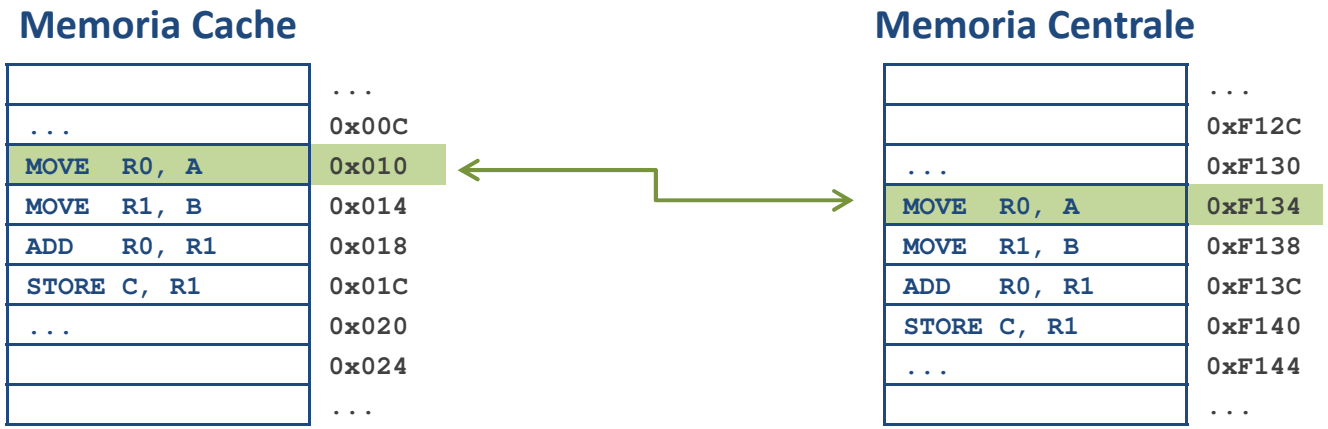
- **A seguito di cache miss**
  - E' necessario caricare una nuova pagina
  - Se nella cache c'è "spazio", non si hanno problemi
  - In caso contrario si deve scegliere una "politica di sostituzione"

- **La politica di sostituzione determina quale blocco sosituire**
  - Le politiche principali sono le tre seguenti

- **Politica casuale**
  - Semplice e mediamente efficiente
- **Politica LRU: Least-Recently Used**
  - Sostituire il blocco usato meno di recente
- **Politica FIFO: First-In-First-Out**
  - Inserisce i blocchi in una coda al momento della copia in cache
  - Legge dalla coda il blocco da sostituire

Mapping

- **Il problema cruciale consiste nel definire un meccanismo per associare gli indirizzi della memoria centrale a quelli della cache e vice versa**



- **Si hanno tre soluzioni al problema del mapping**
  - Mapping diretto
  - Mapping completamente associativo
  - Mapping set-associativo

Mapping diretto

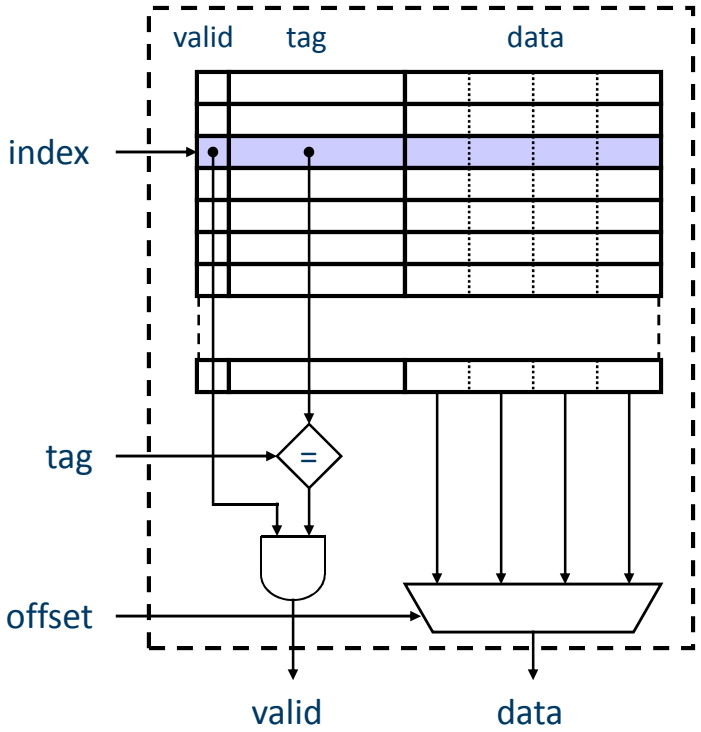
- **L'indirizzo in memoria centrale viene visto come costituito da tre campi:**

- **Index**
  - Indirizzo della linea di cache
  - Il numero di bit di questo campo dipende dalle dimensioni della cache
- **Tag**
  - La parte alta dell'indirizzo
  - Viene confrontato con il tag salvato in cache all'indirizzo di index
  - Se i tag sono uguali si ha un cache hit
- **Offset**
  - Indica una particolare word, halfword o byte in una linea di cache

- **Inoltre è necessario disporre di un valid bit**
  - Indica se i dati presenti in una linea di cache sono validi

Mapping diretto

- **Architettura hardware**



Mapping completamente associativo

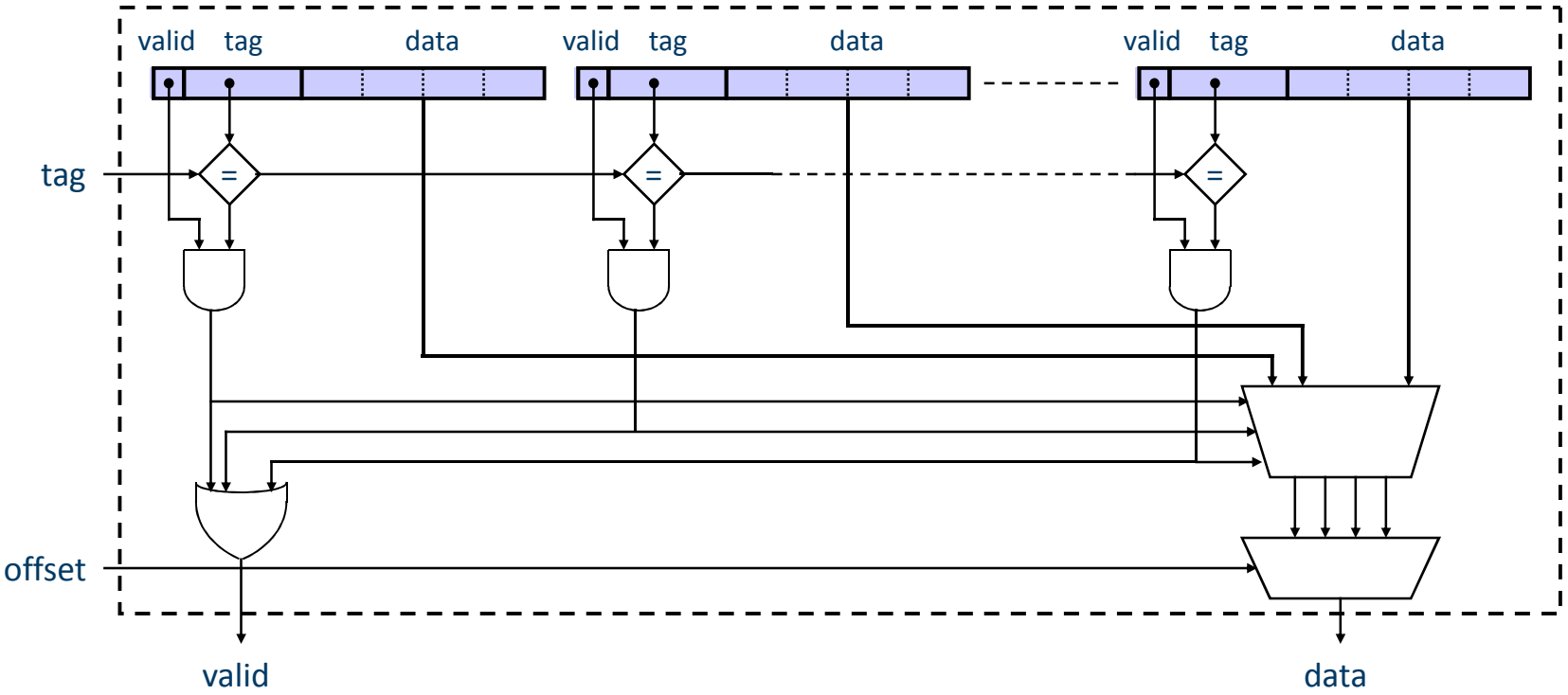
- **L'indirizzo in memoria centrale viene visto come costituito da due campi:**

- **Tag**
  - La parte alta dell'indirizzo
  - Viene confrontato con tutti i tag salvati in cache simultaneamente
  - Se i tag sono uguali si ha un cache hit
- **Offset**
  - Indica una particolare word, halfword o byte in una linea di cache

- **Inoltre è necessario disporre di un valid bit**
  - Indica se i dati presenti in una linea di cache sono validi

Mapping completamente associativo

- **Architettura hardware**

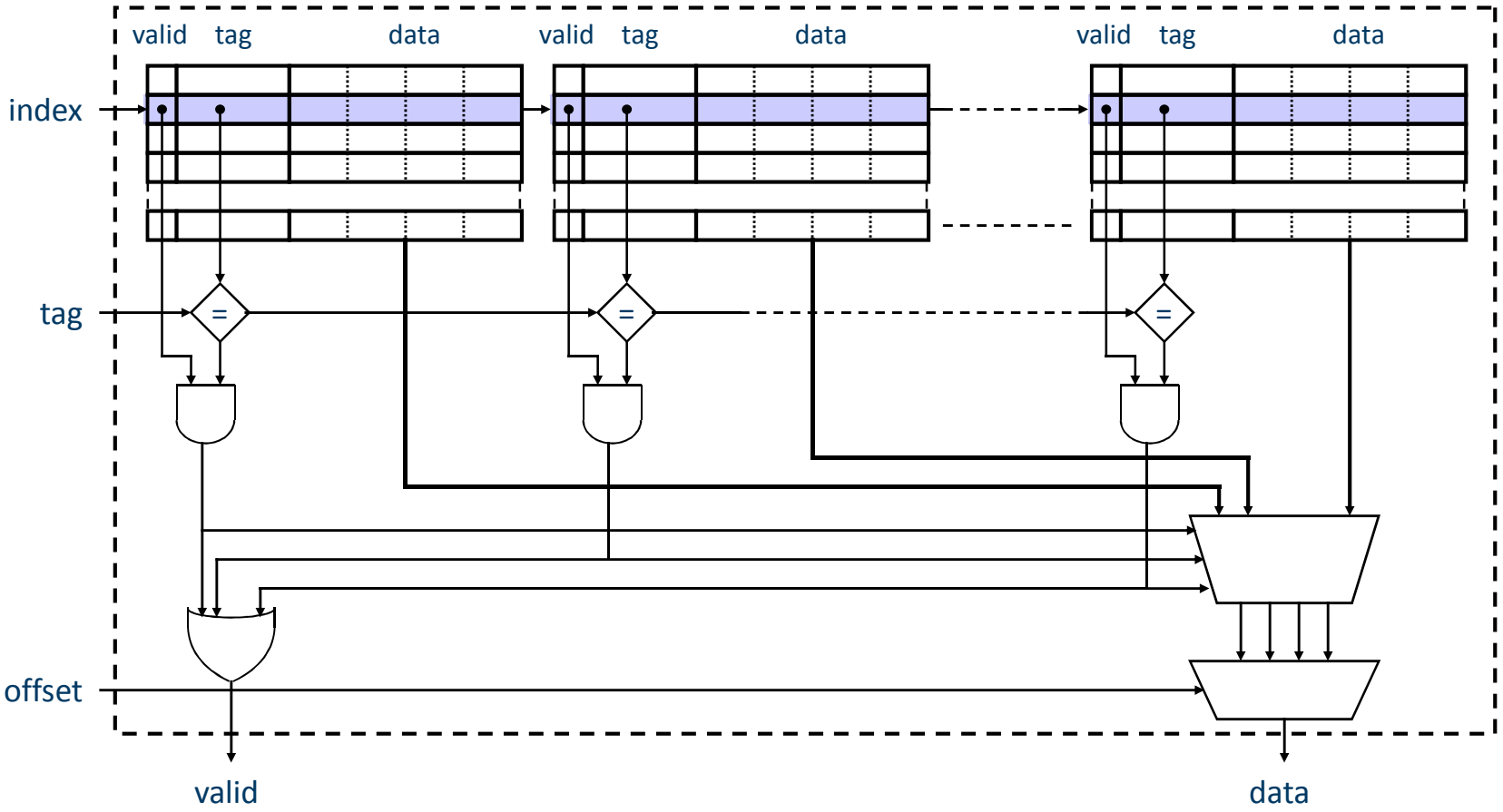


Mapping set-associativo

- **È un compromesso**
  - Tra il mapping diretto e quello completamente associativo
- **L'indirizzo della memoria centrale è scomposto in tre campi**
  - Index, Offset e Tag
  - Come nel caso di mapping diretto
- **Ma…**
  - Ad ogni indirizzo in cache corrispondono più indirizzi di memoria
    - 2, 4, ..., 32 set
  - I tag dei diversi set corrispondenti allo stesso indice vengono confrontati con il tag richiesto simultaneamente
    - Come nel caso di mapping completamente associativo
- **Di fatto si tratta di una struttura in cui più cache basate su mapping diretto sono organizzate in parallelo**

Mapping set-associativo

- Architettura hardware



Esempio 1: Dimensionamento

- **Si consideri un sistema dotato di**
  - Memoria RAM della dimensione di 16 Mbyte
  - Memoria cache
    - Mapping diretto
    - Dimensione memoria 32 Kbyte
    - Dimensione linea 32 byte
- **Ne consegue**
  - Dimensione indirizzo: $\ln_2 16M = \ln_2 2^4 \times 2^{20} = \ln_2 2^{24} = 24$ bit
  - Numero di linee di cache: $32 \text{ Kbyte} / 32 \text{ byte} = 2^5 \times 2^{10} / 2^5 = 2^{10}$
  - Dimensione del campo offset: $\ln_2 32 = 5$ bit
  - Dimensione del campo index: $\ln_2 2^{10} = 10$ bit
  - Dimensione del campo tag: $24 - 10 - 5 = 9$ bit
- **La srtuttura dell'indirizzo è pertanto**

- **La dimensione totale (in bit) della memoria necessaria per realizzare la cache è**
  - $2^{10} \times (1 \text{ valid bit} + 9 \text{ tag bit} + (32 \times 8) \text{ data bit}) = 266 \text{ Kbit} \cong 33.25 \text{ Kbyte}$

Esempio 2: Speedup

- Si calcoli lo speedup di un programma, descritto dalle seguenti statistiche, la cui esecuzione comporta un hit-rate del 90%

Istruzione	# Istruzioni eseguite	Tempo di esecuzione senza cache (cicli)	Tempo di esecuzione con cache (cicli)
Load	10 %	5	1
Store	5 %	5	1
Salti	15 %	3	3
Altre istruzioni	70 %	1	1

- Indicando con N il numero totale di istruzioni
- Tempo di esecuzione senza cache
  - $T_{NOCACHE} = [(0.10 \times 5)_{LD} + (0.05 \times 5)_{ST} + (0.15 \times 3)_{BR} + (0.7 \times 1)_{OTHR}] \times N = 1.9 \times N$
- Tempo di esecuzione con cache
  - $T_{CACHE} = [(0.10 \times 5 \times 0.1 + 0.10 \times 5)_{LD} + (0.05 \times 5 \times 0.1 + 0.05 \times 1 \times 0.9)_{ST} + (0.15 \times 3)_{BR} + (0.7 \times 1)_{OTHR}] \times N = 1.36 \times N$
- Speedup
  - $S = (T_{NOCACHE} - / T_{CACHE}) / T_{NOCACHE} = (1.9 \times N – 1.36 \times N) / (1.9 \times N) = 0.284 = 28.4 \%$