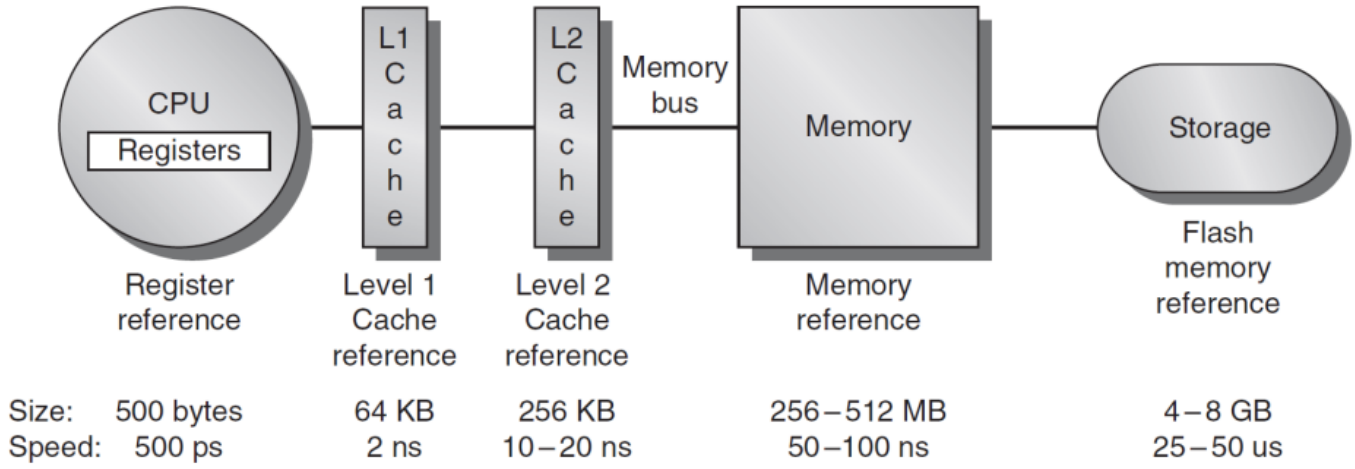


The size and time units change by a factor of 10^9.

La frequenza con **500 ps** è di $2GHz$, con **2 ns** di **20MHz** e così via. Dunque è importante la velocità di accesso alla memoria.

Principio di base e di località

La presenza **cache** migliora le performance grazie a due principi di località:

- **Località spaziale:** quando si accedono dei dati contigui tra di loro(ad esempio un array)
- **Località temporale:** quando si accede la stessa cella di memoria in tempi diversi(ad esempio un contatore in un ciclo)
  L'idea è quindi di portare un intero blocco di dati dentro la cache, così per un certo istante Δt, il programma troverà tutto quello che gli serve dentro la cache.

Performance

Dati:

- **h:** cache hit ratio
- **C:** cache access time
- **M:** memory access time quando i dati non sono nella cache
  allora il tempo d'accesso medio alla memoria è dato da:
  $t_{ave} = h * C + (1 - h) * M$
  In media h si aggira intorno a 0.9(90% di cache hit)

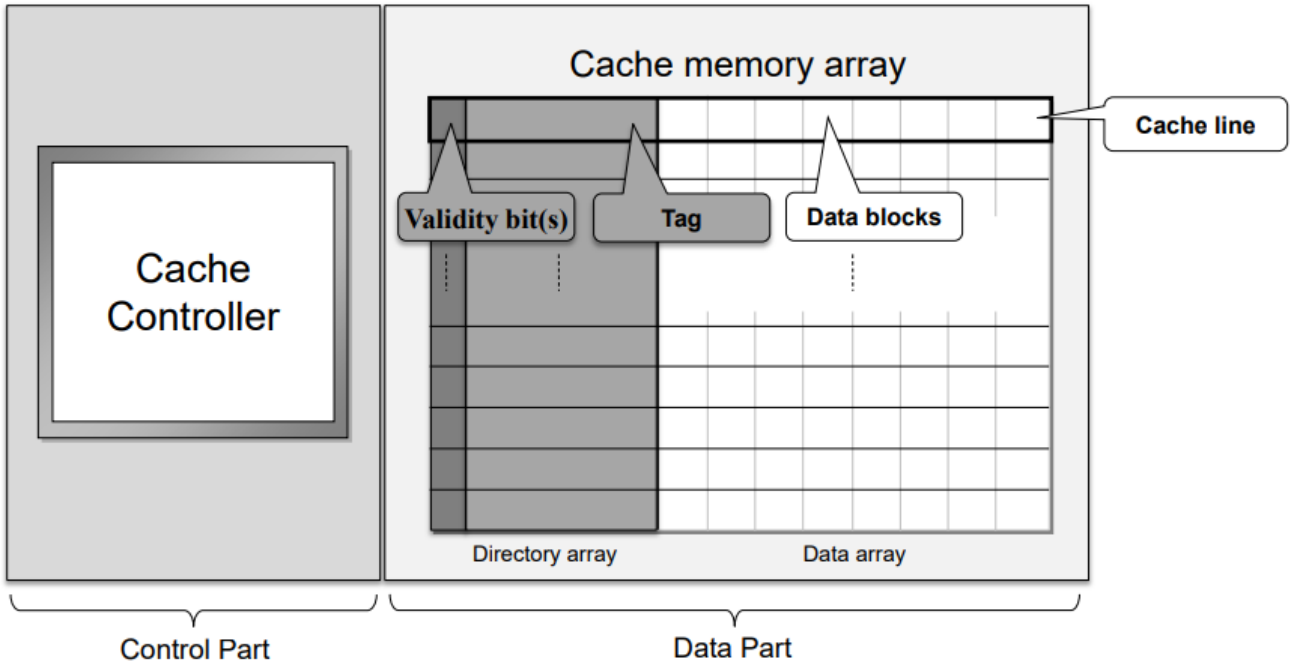
Organizzazione della cache

E' composta da due parti:

- **Control part:** un **cache controller** gestisce la matrice di dati della cache.
  Contiene anche la logica per intercettare l'indirizzo prodotto dal processore,controllare dentro la cache che ci siano i dati richiesti o caricarli dalla RAM.
- **Data part:** i veri e propri dati della cache, gestititi dal controller.

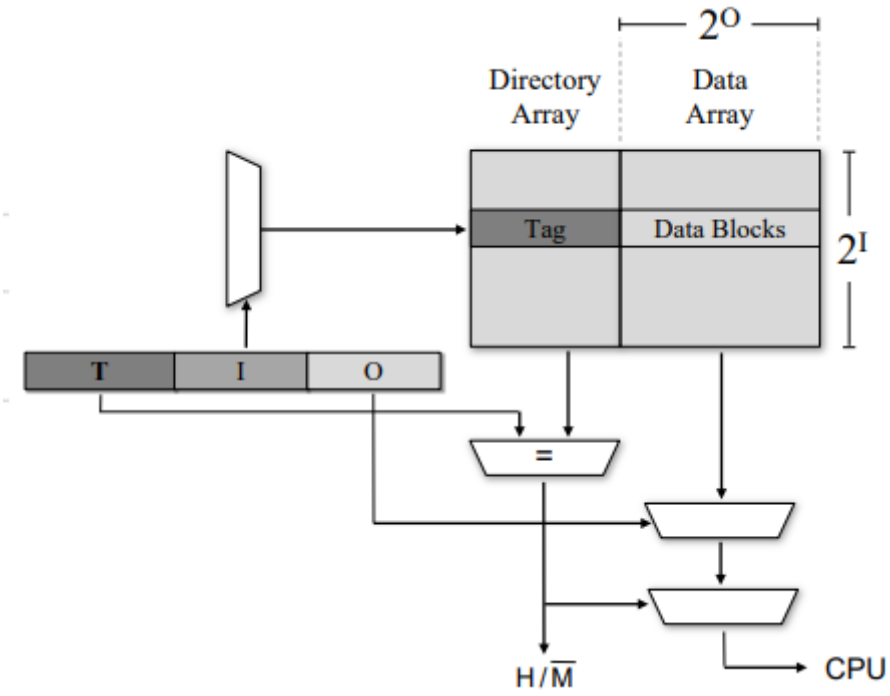
Ogni riga della matrice di cache(cache memory array),chiamata **cache line**, è composta da:

- **tag:** indica la zona della RAM da dove è stato pescato il dato
- **validity bit:** indica se il dato è valido o bisogna aggiornarlo con i valori della RAM(o cache di livello successivo).
- **data/memory block:** blocco di dati effettivi, con più **parole(word)**



Logica di controllo

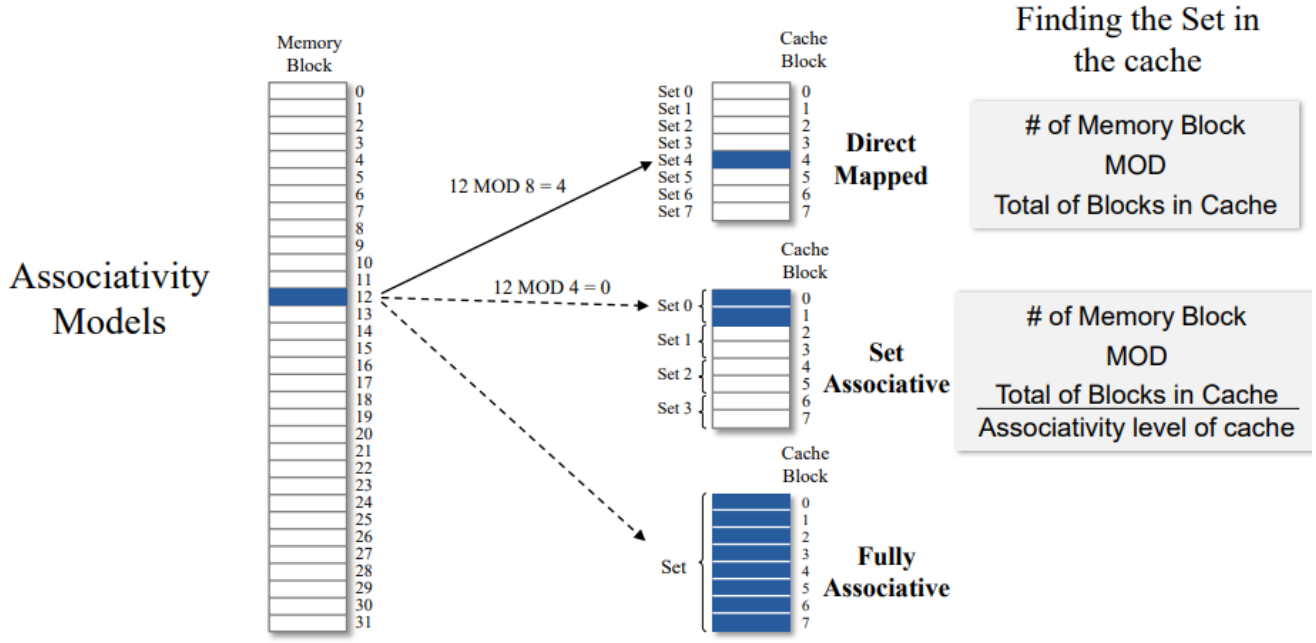
L'effective address prodotto dalla CPU viene visto dalla cache come una tripletta
Tag+Index+Offset.
L'**index** viene usato per trovare il tag nella cache.
Il **tag** della cache viene confrontato con "quello" della CPU.
Se i **tag** corrispondono, e il **validity bit** della cache line è impostato a 1, i dati vengono passati alla CPU.



Talvolta si usa un'architettura Harvard, ovvero una cache dedicata ai dati e una cache dedicata alle istruzioni(I-Cache e D-Cache).
Il campo offset serve per pescare la word precisa, visto che nel data block sono presenti più word

Mapping

Il mapping definisce in quale **cache line** i dati sono contenuti.



Direct Mapping

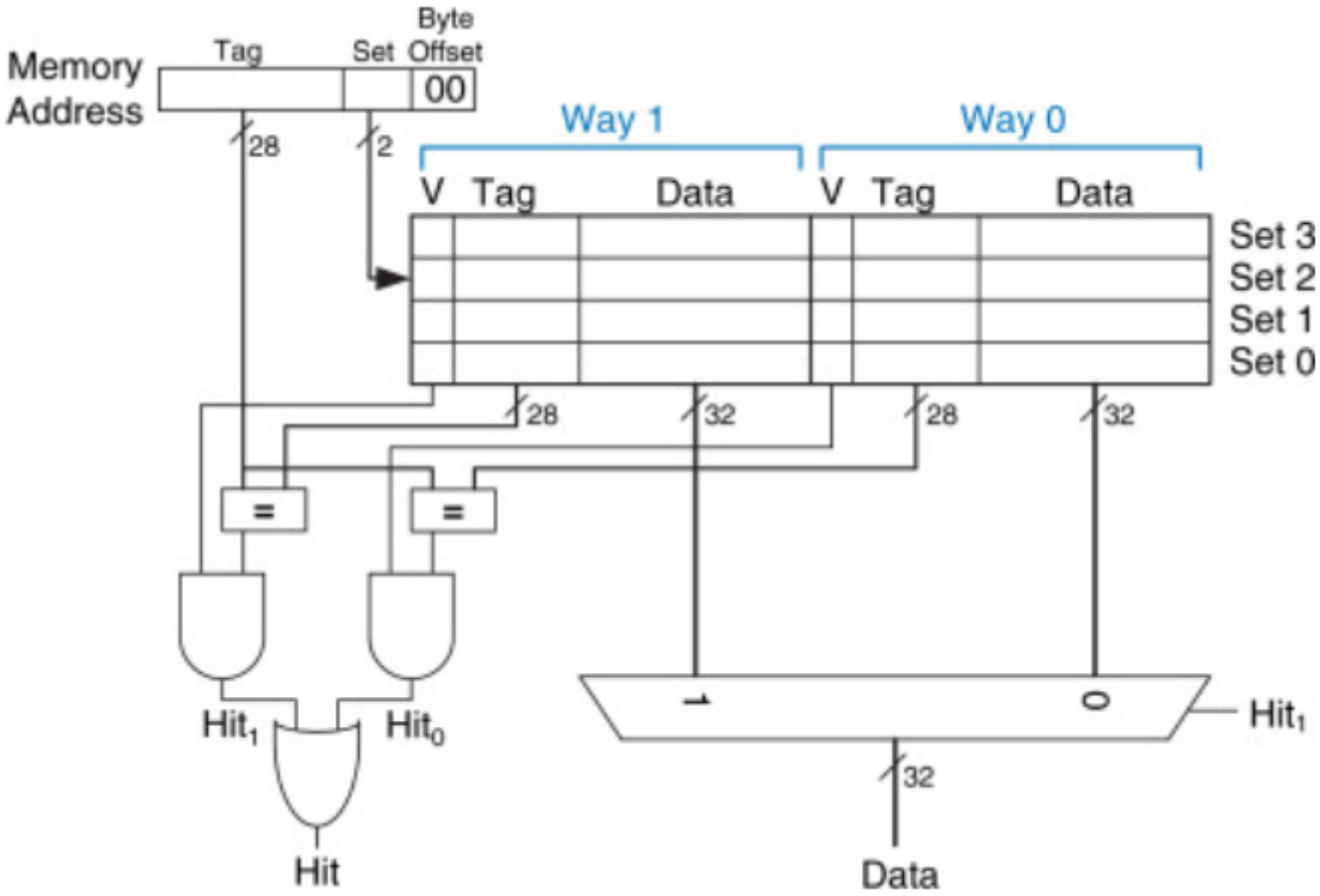
Ogni blocco di memoria viene mappato ad un blocco di cache **modulo** il numero di blocchi presenti nella cache.
Viene usato un indice per selezionare la **cache line**.
Se in quella **cache line**, il suo **tag** corrisponde a quello dell'effective addresso computato dalla CPU e il validity bit è a 1, allora si otterrà un **cache hit**.
Esempio: voglio il blocco 12 ma la cache ha solo 8 blocchi $\rightarrow 12\%8 = 4$

- **Vantaggi:** molto semplice da realizzare a livello hardware
- **Svantaggi:** sovrapposizione di blocchi di memoria sulla stessa cache line

Set Associative

Ogni **cache line** viene associato con n blocchi della memoria. Viene mappato a **numero blocchi di ram modulo (numero blocchi cache/associatività)**.
Esempio: con **n=2** di associatività, i blocchi di memoria 12 e 16, che danno entrambi resto 0, punteranno allo stesso set associativo che conterrà entrambi i dati

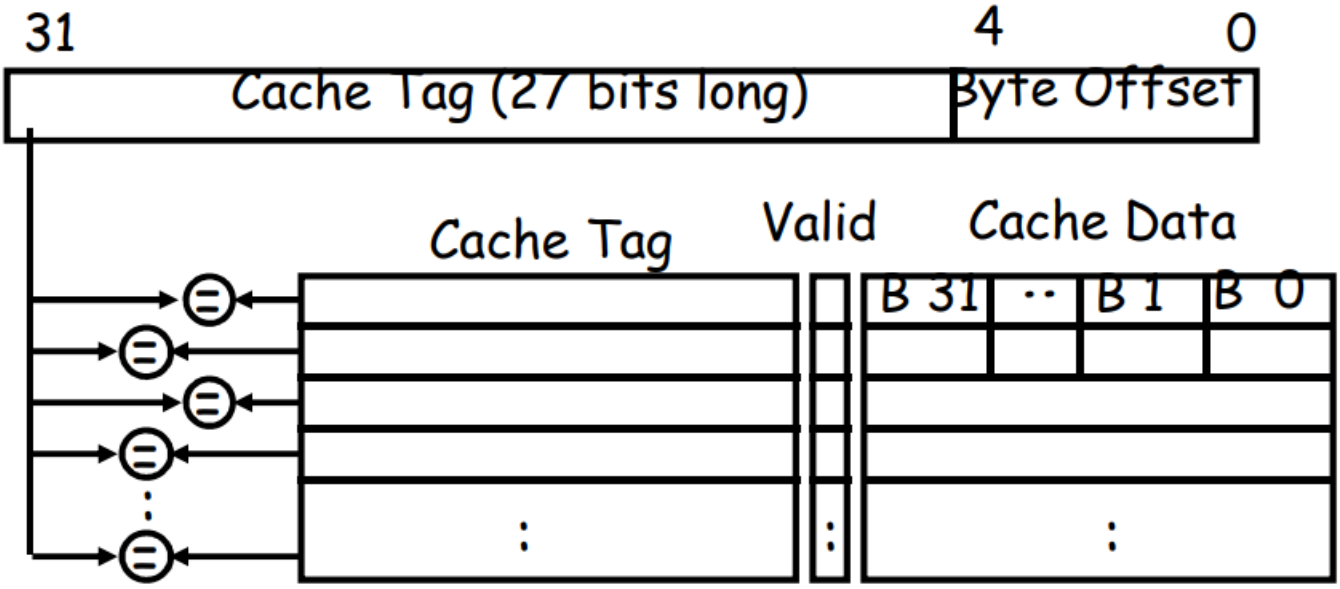
In 0 sarà contenuto il blocco 12, in 1 il blocco 16



Circuiteria per la set associative cache. Una cache line contiene più blocchi di memoria.
Il tag del memory address va confrontato con tutti i tag presenti nella cache line

Fully associative

Ogni blocco della memoria principale viene salvato in qualsiasi blocco di cache.
Dunque, nella cache sarà salvato soltanto il TAG e l'offset



Il Cache tag viene confrontato in parallelo

- **Vantaggi:** massima flessibilità
- **Svantaggi:** complessità dell'hardware per la ricerca parallela

Algoritmi di rimpiazzamento dei blocchi

Definisce quale **cache line** dovrebbe essere usato per salvare un blocco di memoria.
Diverse policy:

- **LRU:** si sceglie quello meno usato di recente. Policy più usata
- **LFU:** si sceglie quello meno usato di recente. Policy migliore ma più costosa
- **FIFO:** first-in first-out
- **Random:** semplice ed efficace.

Aggiornamento della memoria

Se scrivo sulla cache, devo aggiornare anche la memoria principale.
Due soluzioni principali:

- **write-back**
- **write-through**

Write back

Viene aggiornato il valore in memoria solo nel caso in cui serva. Molto efficiente nel caso di **single cpu.**
Si usa un **dirty bit** che indica che quel blocco di cache è stato aggiornato.
Quando il blocco di cache viene svuotato, viene aggiornata anche la memoria principale

Write through

Ogni volta viene aggiornata la memoria principale.
Può portare a problemi di accesso alla memoria in un sistema con memoria condivisa(sistemi multi processore).

Esempio di cache size

TODO