MEMORIA CACHE

Memoria cache a apărut din necesitatea de a elimina diferenţa de viteză dintre procesorul rapid şi memoria sistem, mult mai lentă. Este o memorie statică (SRAM) de dimensiuni mici dar foarte rapidă, folosită pentru a aduce datele mai aproape de unităţile de execuţie din processor. Ca urmare a peformanţelor de funcţionare a memoriei cache, actualmente doar un procent de aproximativ 10% din accesele la memorie ale procesorului se fac din memoria sistem, iar restul de 90% se fac din memoria cache.

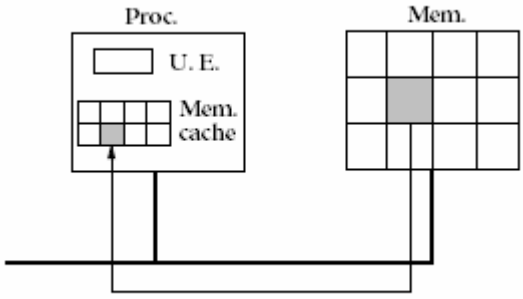

Figura 1. Memoria cache

Memoria cache se bazează pe principiul localităţii. Avem două tipuri de localitate:

a) localitate temporală – locaţiile de memorie accesate la un moment dat tind să fie accesate din nou în viitor;
b) localitate spaţială – datele aflate în vecinătatea unor date accesate tind sa fie şi ele accesate.

Exemple de memorii cache:

- memoria TLB (*Translation Look-aside Buffer*) este un cache special pentru translatarea între adresele fizice şi adresele virtuale din calculator. Datorită folosirii memoriei TLB, calculul adreselor pentru memoria virtuală nu se face la fiecare acces al memoriei, ci se ia din tabela TLB.

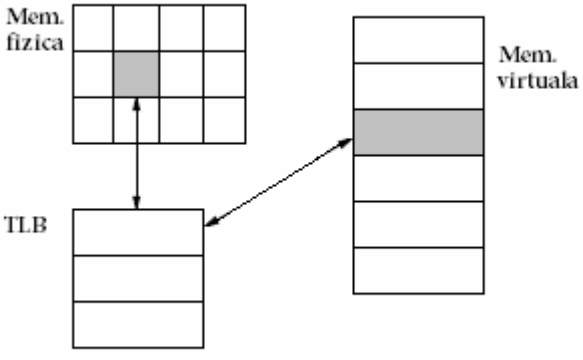

Figura 2. Memoria TLB

- memoria cache pentru instrucţiuni - este folosită pentru încărcarea în avans a instrucţiunilor în procesor ;
- memoria cache de date - folosită pentru încărcarea în avans în datelor necesare din memoria sistem.

Există 3 moduri de mapare a datelor din memoria fizică în memoria cache:

1) mapare directă
2) mapare cu asociere completă
3) mapare cu asociere pe blocuri (set asociativă)

1) Memoria cache cu mapare directă

În acest caz există o mapare directă, secvenţială între adresele din memoria principală şi locaţiile memoriei cache. Aici memoria fizică e împărţită în pagini, fiecare pagină fiind de dimensiunea memoriei cache. Atât paginile din memoria sistem cât şi memoria cache sunt împărţite în blocuri de aceeaşi dimensiune.

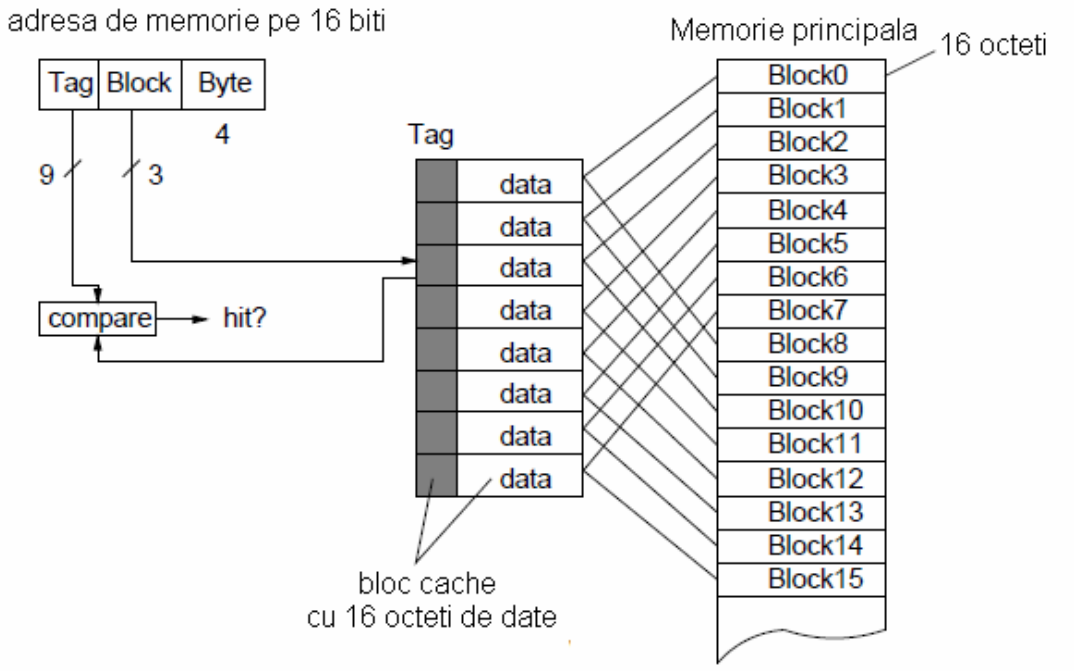

Figura 3. Memoria cache cu mapare directă

În exemplul de mai sus, o pagină conţine 8 blocuri; astfel, blocul 0 din memoria cache corespunde cu blocurile 0, 8, 16, etc. din memoria sistem.

Pentru a şti exact cu ce bloc din memorie corespunde un anumit bloc din memoria cache, se foloseşte un tag (indicator) la fiecare intrare din memoria cache. Tag-ul conţine cei mai semnificativi biţi ai adresei blocului din memoria principală, reprezentând de fapt adresa de pagină a blocului.

Dacă procesorul vrea să acceseze un bloc de date din memorie situat la o anumită adresă, mai întâi trebuie să verifice dacă blocul căutat se află sau nu în memoria cache. Pentru aceasta, se va proceda astfel:

a) se selectează o intrare din memoria cache pe baza informaţiei de bloc din adresa căutată ;

b) se compară câmpul tag din adresa cerută de procesor cu cel din intrarea cache;
c) dacă cele două sunt egale, avem asa-numitul cache-hit, adică blocul căutat din memoria principală se regăseşte în memoria cache;
d) dacă cele două tag-uri sunt diferite, atunci blocul căutat nu există în memoria cache şi el va fi încărcat din memoria principală.

Avantajul modului de mapare directă:
Memoriile cu mapare directă sunt memorii simple şi de mare viteză. Ele se pot utiliza ca memorii cache de nivel 2, putând avea dimensiuni mai mari.

2) **Memorii cache cu asociere completă**

Aici, un bloc din memoria sistem poate fi plasat oriunde în memoria cache.

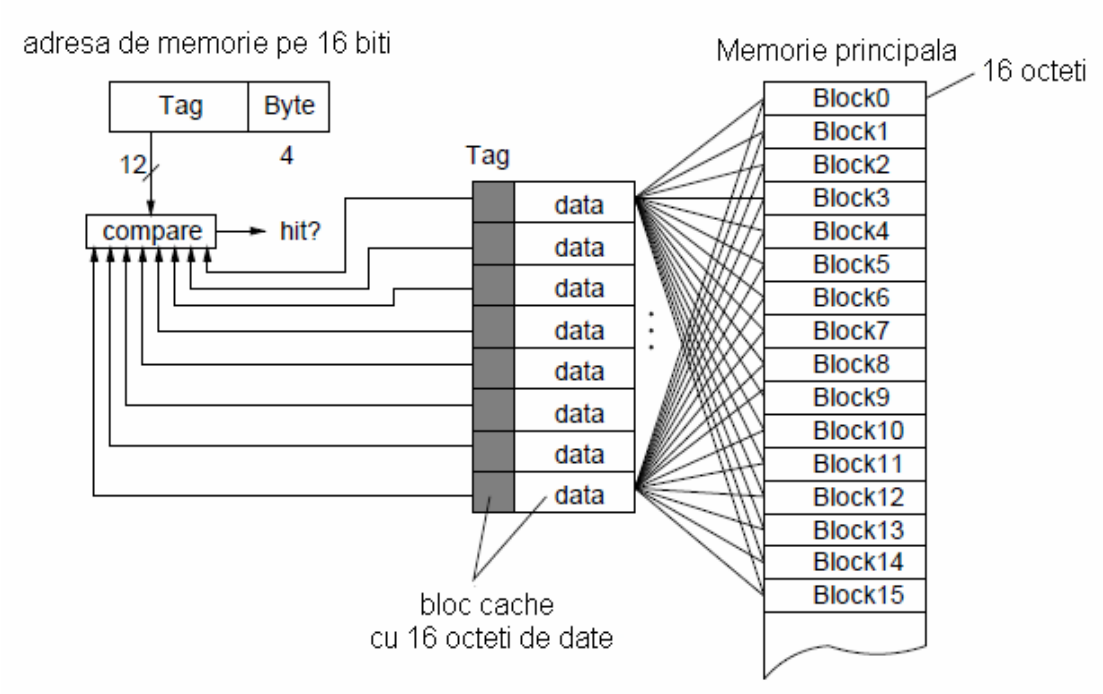

Figura 4. Memoria cache cu asociere directă

Pentru a detecta dacă un bloc din memoria sistem există în memoria cache, se face o comparare în paralel a tag-urilor din toate blocurile memoriei cache. Dacă se identifică un tag, înseamnă că s-a găsit blocul căutat.

Avantajul acestui tip de memorie: foloseşte un mod de mapare mai flexibil.

Dezavantaj: este mai costisitor de implementat datorită comparaţiilor care se fac în paralel.

3) **Memorii cache set asociative**

Acestea folosesc o combinaţie între maparea directă şi maparea cu asociere completă. Aici, memoria cache e împărţită în mai multe seturi de intrări, existând o mapare directă între blocurile din memoria principală şi seturile din memoria cache. În interiorul seturilor se face o mapare cu asociere completă.

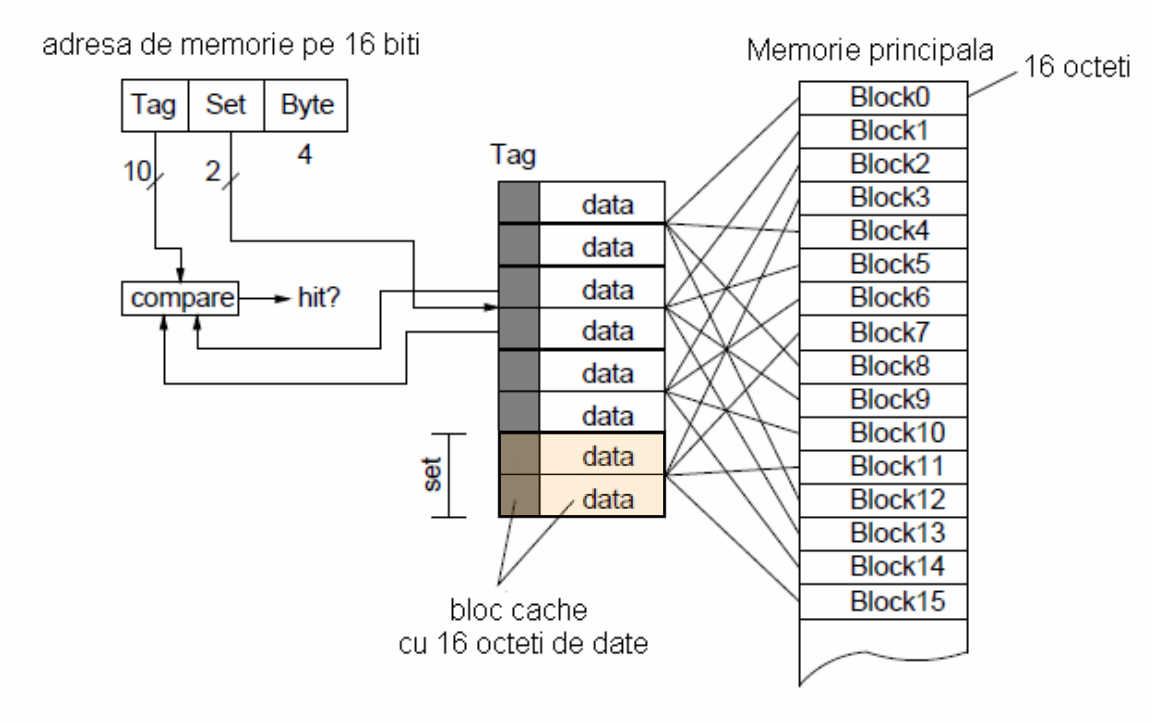

Figura 5. Memoria cache set-asociativă

Pentru a vedea dacă blocul dorit se găseste în memoria cache, se face o comparare în paralel a tag-urilor doar pentru intrările dintr-un singur set.

Avantajul memoriilor set-asociative: se reduce numărul de comparaţii efectuate în paralel; sunt mai puţin costisitoare decât memoriile complet asociative; se folosesc ca memorii cache de nivelul 1, având performanţe ridicate.

Observaţie: În sistemele actuale există o ierarhie de memorii cache organizată pe mai multe nivele; în funcţie de poziţia memoriei cache faţă de procesor, putem avea: memorie cache de nivel 1 (care se găseşte în interiorul capsulei procesorului), memorie cache de nivel 2, de nivel 3, etc.

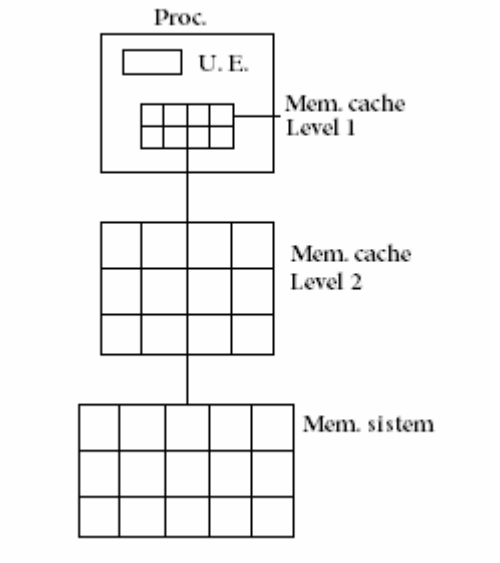

Figura 6. Ierarhia de memorii cache într-un sistem de calcul

Strategii utilizate pentru memoriile cache:

În legătură cu anumite evenimente ce pot sa apară la accesarea memoriei cache, se stabilesc următoarele strategii:

S1) Strategia *cache-miss*. Această strategie se referă la memorii cache complet asociative. Astfel dacă e nevoie de un bloc de memorie care nu se regăseşte în cache (*cache-miss*), trebuie să se stabilească care anume dintre blocurile existente în memoria cache va fi înlocuit de noul bloc care va fi adus din memoria sistem.
Strategia de înlocuire poate fi:

- aleatorie
- *first in, first out* - se înlocuieşte primul bloc introdus în cache
- se înlocuieşte cel mai puţin recent utilizat. Aici se înlocuieşte blocul care nu a fost accesat de cel mai mult timp (cel mai vechi).

S2) Strategii de scriere a memoriilor cache. Să presupunem că procesorul efectuează o operaţie STORE (scriere în memorie). Există 2 posibilităţi:

a) valoarea se scrie în memoria cache **şi** în memoria sistem. Aceste memorii se numesc *write-through*; valoarea respectivă se scrie prin cache în memoria sistemului.

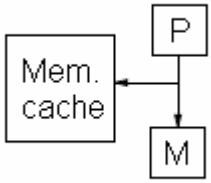

Figura 7. Memoria *write-through*

b) scrierea se face doar în memoria cache. Scrierea în memoria sistem se face ulterior doar când este necesar acest lucru. Aceste memorii se numesc *write-back*.

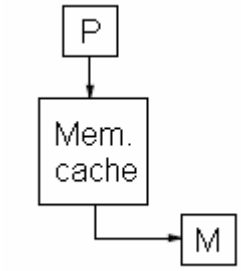

Figura 8. Memoria *write-back*

Avantajele memoriei *write-through*:

- memoria principală e întotdeauna consistentă cu memoria cache.

Dezavantajele memoriei *write-through*:

- această memorie este mai înceată pentru că se aşteaptă scrierea în memoria sistem. La o operaţie STORE se aşteaptă scrierea în memoria sistem, care este mai lentă.
- se măreşte traficul între magistrală şi memoria principală datorită faptului că fiecare acţiune STORE a procesorului se propagă în memoria principală.

Acest lucru nu se petrece la memoria *write-back*. În schimb, la acest tip de memorie este necesar un bit suplimentar pentru fiecare bloc din memoria cache, care să specifice dacă blocul este consistent cu memoria principală. Acest bit se numeşte bit *dirty*.

Dacă bitul *dirty* e 0, înseamnă că blocul din memoria cache e consistent (identic) cu blocul din memoria sistem. Dacă bitul *dirty* e 1, înseamnă că blocul nu e consistent (nu e actualizat în memoria sistem). Când un astfel de bloc *dirty* trebuie să fie eliminat din memoria cache, el va fi scris mai întâi în memoria principală.

Dezavantaj al memoriei *write-back*:

La schimbarea contextului program de către un sistem de operare multi-tasking se foloseste întotdeauna un context diferit de memorie, de aici rezultă că un mare număr de blocuri din memoria cache vor trebui să fie înlocuite şi astfel timpul de comutare a contextului program creşte foarte mult.

S3) Strategii *write-miss*. Evenimentul *write-miss* apare când există un *cache-miss* la o instrucţiune STORE, adică blocul dorit nu se află în memoria cache atunci când procesorul vrea să scrie în acesta.

Memoriile *write-back* folosesc 2 tipuri de strategii:

a) Strategia "alocă la scriere" (*allocate on write*). Aici memoria cache alocă un bloc cache, după care realizează acţiunea de a scrie în acel bloc.
b) Strategia "încarcă la scriere" (*fetch on write*). Aici memoria cache mai întâi citeşte blocul din memoria principală, apoi scrie data în blocul citit.

Memoriile *write-through* folosesc strategia "nu se alocă la scriere" (*no allocate on write*). Aici, dacă există un *cache-miss* la scriere, atunci nu se mai alocă un bloc în memoria cache, ci data trimisă de procesor va fi scrisă direct în memoria principală a sistemului.