Se consideră o expresie postfixată formată din *n* simboluri, fiecare din ele reprezentând o constantă sau un operator. Pentru evaluarea acestei expresii prin utilizarea unei stive, se poate utiliza algoritmul următor.

1. Se inițializează o variabilă *k* cu 1.
2. Se atribuie unei variabile *s* valoarea simbolului cu numărul de ordine *k* din expresia postfixată (de exemplu, dacă *k* = 1, *s* ia valoarea primului simbol al expresiei).
3. Dacă *s* este o constantă, se memorează în stivă şi se trece la pasul 5.
4. Dacă *s* este un operator, se extrag cele două elemente din vârful stivei, şi se efectuează asupra lor operația indicată de *s*, astfel încât elementul din vârful stivei reprezintă operandul din dreapta operatorului. Rezultatul se memorează în stivă.
5. Dacă *k* = *n* (s-a ajuns la ultimul simbol al expresiei), operația se termină, rezultatul fiind în vârful stivei. În caz contrar, se incrementează *k* şi se continuă de la pasul 2.

Exemplul 8.2

Expresia infixată: `(8 + 2 * 5) / (1 + 3 * 2 - 4)`
Expresia postfixată: `8 2 5 * + 1 3 2 * + 4 - /`

Se prezintă în Figura 8.9 stiva pentru fiecare pas al operației.

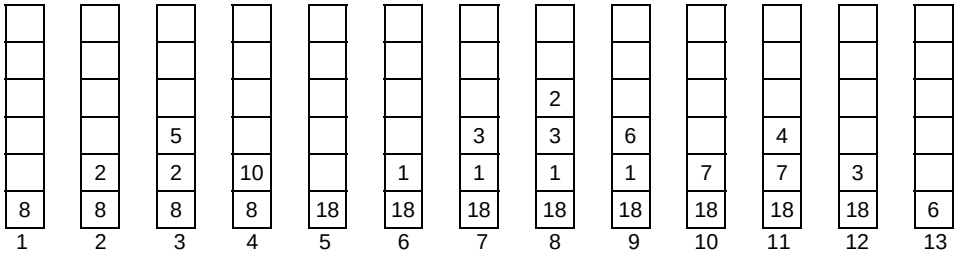

Figura 8.9. Conținutul stivei la evaluarea unei expresii postfixate.

8.5. Memoria cache

8.5.1. Principiul memoriei cache

Viteza UCP este superioară vitezei memoriilor, astfel că după inițierea unui ciclu de acces la memorie, UCP trebuie să rămână inactivă un timp, aşteptând răspunsul acesteia.

Memoriile rapide sunt realizabile din punct de vedere tehnologic, dar costul lor este ridicat. Sunt cunoscute însă tehnici pentru combinarea unei memorii rapide de dimensiuni mici cu o memorie mai lentă de dimensiuni mai mari, pentru a se obține aproximativ viteza memoriei rapide şi capacitatea mare a memoriei lente, la un preț moderat. Memoria rapidă de dimensiune mică se numeşte *memorie cache* (din limba franceză: *cacher* - a ascunde).

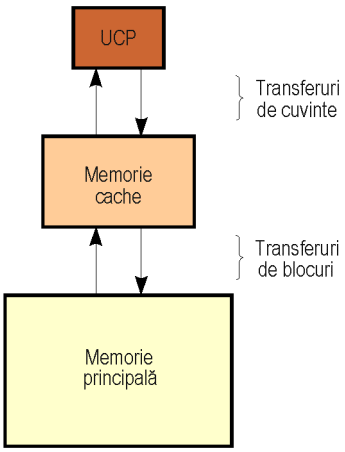

Figura 8.10. Principiul memoriei *cache*.

Principiul memoriei *cache* este ilustrat în Figura 8.10. Există o memorie principală de dimensiuni relativ mari, dar mai lentă, şi o memorie *cache* mai redusă, dar mai rapidă. Memoria *cache* conține o copie a unor părți din memoria principală. Atunci când UCP încearcă citirea unui cuvânt din memorie, se testează dacă respectivul cuvânt se află în memoria *cache*. În caz afirmativ, cuvântul este furnizat unității centrale. În caz contrar, se încarcă în memoria *cache* un bloc al memoriei principale, constând dintr-un număr fix de cuvinte, iar apoi cuvântul este returnat unității centrale.

Se cunoaşte că programele nu fac acces la memorie în mod complet aleator. Dacă se face o referire la o anumită adresă, este probabil că următoarea referire la memorie va fi în vecinătatea acestei adrese. În spațiul adreselor de memorie, câteva regiuni au o probabilitate ridicată de a fi accesate, câteva au o probabilitate moderată, iar celelalte au o probabilitate foarte mică de a fi accesate în viitorul apropiat.

O regiune care are o probabilitate înaltă este cea corespunzătoare contorului de program actual, deoarece este probabil să se execute următoarea instrucțiune din secvența de instrucțiuni. Alte regiuni care au o probabilitate mare de a fi accesate sunt cele care conțin datele active, procedurile şi punctul de întoarcere dintr-o procedură. Dacă programul este scris într-un limbaj structurat pe blocuri, ca de exemplu *Pascal*, zona de stivă pentru variabile locale şi parametri este o altă zonă cu probabilitate ridicată de acces.

Observația că referințele la memorie efectuate într-un interval scurt de timp utilizează o mică porțiune a memoriei reprezintă *principiul localității*, şi formează baza

sistemelor de memorie *cache*. Atunci când este adresat un cuvânt, acesta este transferat din memoria lentă în memoria *cache*, astfel încât la următoarea utilizare va putea fi accesat în mod rapid.

În Figura 8.11 se prezintă structura unui sistem de memorie format din memoria principală şi o memorie *cache*.

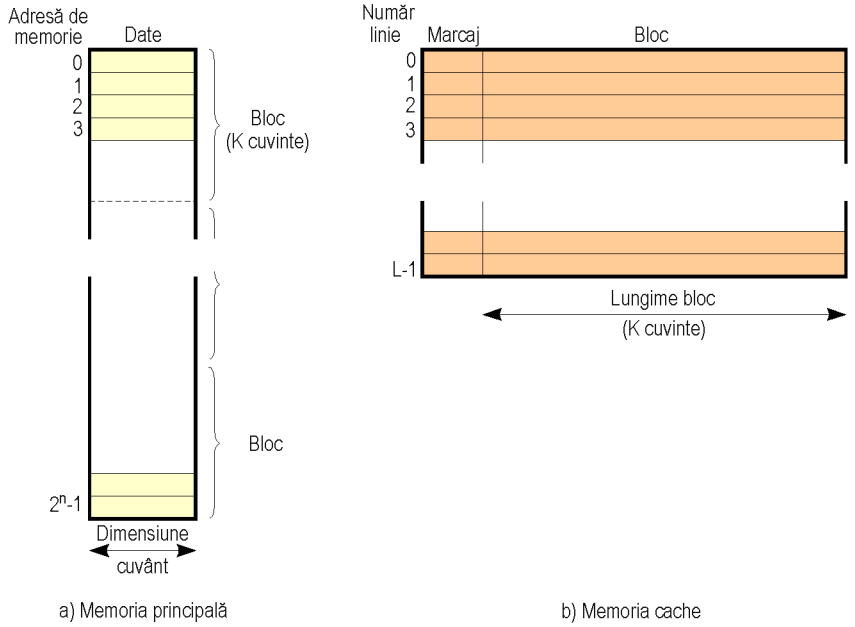

Figura 8.11. Structura unui sistem de memorie.

Memoria principală constă din 2^n cuvinte adresabile, fiecare cuvânt având o adresă unică de n biţi. Se consideră că această memorie este formată dintr-un număr de blocuri de lungime fixă de K cuvinte fiecare. Există deci $2^n/K$ blocuri. Memoria *cache* constă din blocuri de câte K cuvinte fiecare, un asemenea bloc fiind numit *linie*. Există L linii în memoria *cache*, numărul de linii fiind mult mai mic decât numărul blocurilor din memoria principală ($L << K$).

În orice moment, o anumită parte a blocurilor de memorie se află în liniile memoriei *cache*. Dacă se citeşte un cuvânt al unui bloc din memoria principală, blocul respectiv este transferat într-una din liniile memoriei *cache*. Deoarece există mai multe blocuri decât linii, o anumită linie nu poate fi dedicată în mod unic şi permanent unui anumit bloc. De aceea, fiecare linie conţine un *marcaj* care identifică blocul pe care îl conţine linia respectivă. Marcajul este de obicei o parte a adresei din memoria principală.

În Figura 8.12 se ilustrează operaţia de citire. UCP generează adresa unui cuvânt care trebuie citit (A). Dacă acest cuvânt se află în memoria *cache*, este transferat

unității centrale. În caz contrar, blocul care conține acest cuvânt este încărcat într-o linie a memoriei *cache*, iar cuvântul este transferat unității centrale.

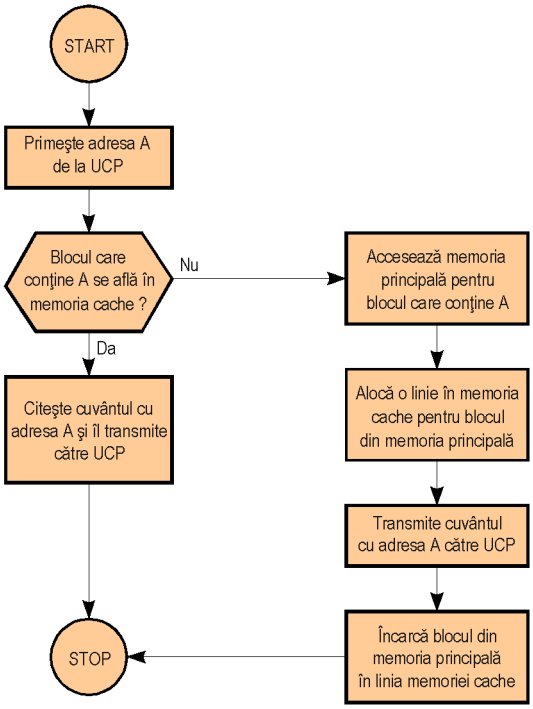

Figura 8.12. Operația de citire a memoriei *cache*.

Considerăm parametrii principali ai memoriei *cache*. *Costul mediu pe bit* C_S al sistemului de memorie format din memoria principală şi memoria *cache* este dat de:

$$C_S = \frac{C_C D_C + C_M D_M}{D_C + D_M} \tag{8.2}$$

unde:

C_C = costul mediu pe bit al memoriei *cache*;
C_M = costul mediu pe bit al memoriei principale;
D_C = dimensiunea memoriei *cache*;
D_M = dimensiunea memoriei principale.

Este de dorit ca pentru memoria *cache* costul C_C să fie aproximativ egal cu C_M. Deoarece $C_C >> C_M$, aceasta necesită ca $D_C << D_M$.

Timpul de acces al sistemului de memorie nu depinde numai de viteza memoriei *cache* şi a memoriei principale, ci şi de probabilitatea ca un anumit cuvânt adresat de

UCP să fie găsit în memoria *cache*. Această probabilitate este numită *rată de succes*. Avem:

$$T_S = P_S\, T_C + (1 - P_S)\, T_M \tag{8.3}$$

unde:

T_S = timpul mediu de acces al sistemului de memorie;
T_C = timpul de acces al memoriei *cache*;
T_M = timpul de acces al memoriei principale;
P_S = rata de succes.

Este de dorit ca $T_S \approx T_C$. Deoarece $T_C << T_M$, este necesară o rată de succes apropiată de 1.

Studiile au arătat că o dimensiune relativ redusă a memoriei *cache*, de 64 KB sau 128 KB, este în general adecvată, obținându-se o rată de succes de peste 0,75, indiferent de dimensiunea memoriei principale.

8.5.2. Caracteristici ale memoriei cache

Deşi există un număr mare de implementări pentru memoria *cache*, există un număr de caracteristici utilizate pentru clasificarea şi diferențierea dintre sistemele de memorie *cache*.

8.5.2.1. Dimensiunea

Această dimensiune trebuie să fie suficient de mică, astfel încât costul mediu pe bit al întregului sistem de memorie să fie apropiat de cel al memoriei principale, şi suficient de mare, astfel încât timpul mediu de acces al întregului sistem de memorie să fie apropiat de cel al memoriei *cache*. O dimensiune relativ redusă, între 64 KB şi 512 KB, satisface de obicei aceste cerințe.

8.5.2.2. Funcția de mapare

O caracteristică de bază a memoriei *cache* este *funcția de mapare* (de translatare), care atribuie locații din memoria *cache* blocurilor din memoria principală. Se pot utiliza trei tehnici: *directă*, *asociativă* şi *asociativă pe seturi*. Aceste alternative se vor examina printr-un exemplu. Se consideră o memorie *cache* de 1024 (1 K) octeți. Datele se transferă între memoria principală şi memoria *cache* în blocuri de câte 8 octeți. Aceasta înseamnă că memoria *cache* are 128 de linii de câte 8 octeți fiecare. Memoria principală constă din 64 K octeți, fiecare octet fiind direct adresabil printr-o adresă de 16 biți. Se poate considera că memoria principală constă din 8 K blocuri de câte 8 octeți fiecare.

Deoarece există un număr mai mic de linii ale memoriei *cache* față de numărul blocurilor memoriei principale, este necesar un algoritm pentru plasarea blocurilor memoriei principale în liniile memoriei *cache*. În plus, este necesar un mijloc de a determi-

na care bloc al memoriei principale ocupă la un moment dat o linie a memoriei *cache*. În cazul tehnicii celei mai simple, numită *mapare directă*, fiecare bloc al memoriei principale poate ocupa o singură linie posibilă a memoriei *cache* (Figura 8.13). Maparea este:

$$C = A \bmod L$$

unde:

C = numărul liniei din memoria *cache*;
A = adresa din memoria principală;
L = numărul de linii din memoria *cache*.

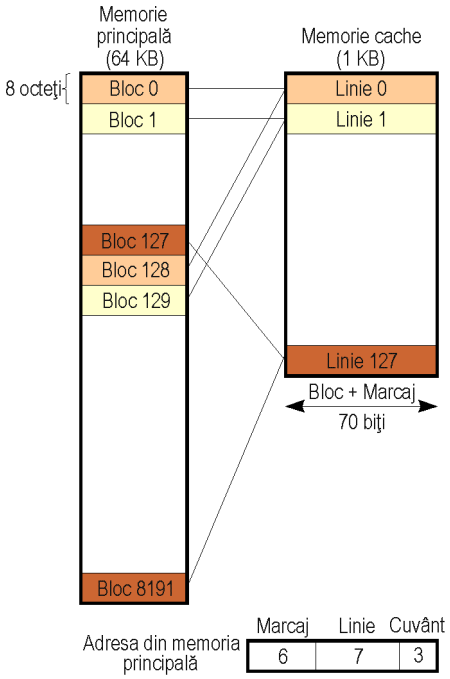

Figura 8.13. Memorie *cache* cu mapare directă.

Pentru exemplul considerat, $L = 128$ şi $C = A \bmod 128$. Funcția de mapare poate fi implementată simplu utilizând adresa de 16 biți. Cei 3 biți mai puțin semnificativi identifică un cuvânt (octet) unic în cadrul unui bloc al memoriei principale. Restul de 13 biți specifică unul din cele $2^{13} = 8$ K blocuri ale memoriei principale. Câmpul de 7 biți, numit *linie*, indică numărul blocului, modulo 128. Astfel, blocurile 0, 128, 256, …, 8064 vor fi amplasate în linia 0; blocurile 1, 129, …, 8065 vor fi amplasate în linia 1, şi aşa mai departe, până la blocurile 127, 255, …, 8191, care vor fi amplasate în linia 127. Câmpul de 6 biți, numit *marcaj*, are rolul de a identifica în mod unic blocul din linie. Astfel, blocurile 0, 128, 256, …, 8064 au numerele de marcaje 0, 1, 2, …, respectiv 63.

Considerând din nou Figura 8.12, o operație de citire are loc astfel. Memoriei *cache* i se prezintă o adresă de 16 biți. Numărul liniei de 7 biți este utilizat ca un index în memoria *cache* pentru accesul la o anumită linie. Dacă marcajul de 6 biți este egal cu marcajul liniei respective, atunci numărul de 3 biți al cuvântului este utilizat pentru a selecta unul din cei 8 octeți ai liniei respective. În caz contrar, marcajul şi numărul liniei (în total 13 biți) se utilizează pentru a încărca un bloc din memoria principală.

Maparea directă este o tehnică care se poate implementa simplu. Principalul dezavantaj al acesteia este că există o locație fixă în memoria *cache* pentru oricare bloc dat. Deci, dacă un program face referire în mod repetat la cuvinte din două blocuri diferite care se mapează în aceeaşi linie, blocurile vor fi interschimbate în mod continuu în memoria *cache*, şi rata de succes va fi redusă.

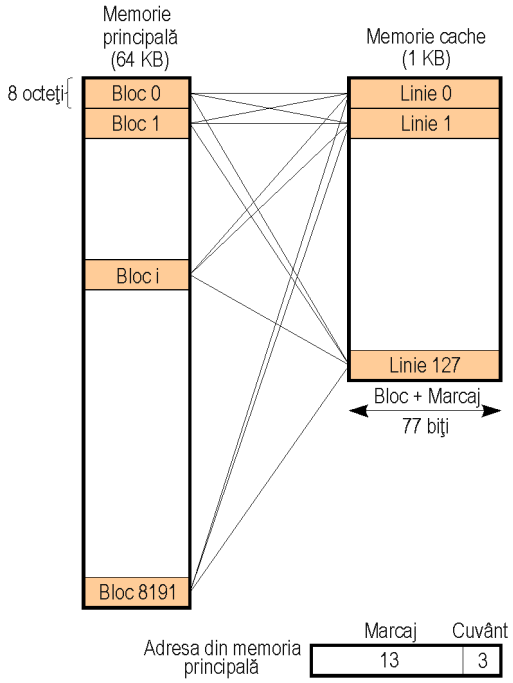

Figura 8.14. Memorie *cache* cu mapare asociativă.

O metodă care elimină dezavantajul mapării directe este *maparea asociativă* (Figura 8.14). În acest caz, adresa memoriei principale constă dintr-un marcaj de 13 biți şi un număr al cuvântului de 3 biți. Un bloc din memoria principală se poate încărca în oricare linie, iar marcajul său de 13 biți se memorează împreună cu blocul. Pentru a determina dacă un bloc se află în memoria *cache*, sunt necesare circuite pentru a compara simultan marcajul său cu marcajele fiecărei linii.

În cazul mapării asociative, există o flexibilitate în privința înlocuirii unui bloc atunci când un nou bloc este încărcat în memoria *cache*. Algoritmii de înlocuire a blocurilor din memoria *cache*, prezentați în continuare, au rolul de a creşte eficiența. Princi-

palul dezavantaj al acestei metode îl reprezintă circuitele complexe necesare pentru a examina marcajele tuturor liniilor din memoria *cache*.

Maparea asociativă pe seturi reprezintă un compromis care reține avantajele mapării directe şi a celei asociative. În acest caz, memoria *cache* este împărțită în I seturi, şi fiecare din acestea conține J linii. Avem:

$$L = I \times J$$
$$K = A \bmod I$$

unde K = numărul setului. Cu această metodă, blocul conținând adresa A poate fi mapat în oricare din liniile setului I. Exemplul prezentat (Figura 8.15) este o mapare asociativă pe seturi cu două linii pe set. De observat că în cazul extrem în care $I = L$, $J = 1$, maparea asociativă pe seturi se reduce la maparea directă, iar atunci când $I = 1$, $J = L$, se obține maparea asociativă.

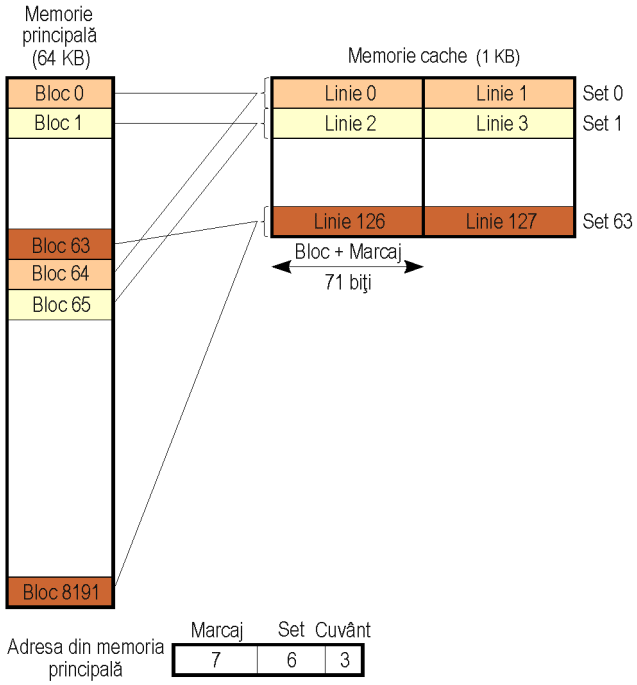

Figura 8.15. Memorie *cache* cu mapare asociativă pe seturi.

Utilizarea a două linii pe set este cea mai utilizată organizare asociativă pe seturi, care îmbunătățeşte semnificativ rata de succes față de maparea directă.

8.5.2.3. Metoda de înlocuire a blocurilor

Atunci când este încărcat un nou bloc în memoria *cache*, unul din blocurile existente în această memorie trebuie înlocuit. Este necesar un algoritm care să selecteze blocul care va fi înlocuit. Se prezintă principalii algoritmi utilizați.

- LRU (*Least Recently Used*): Se înlocuieşte blocul care a fost cel mai puțin recent utilizat. Deoarece se presupune că blocurile utilizate cel mai recent au probabilitatea cea mai mare de a fi accesate, această metodă dă cele mai bune rezultate.
- FIFO (*First-In, First-Out*): Se înlocuieşte blocul care a fost încărcat primul în memoria cache (cel mai puțin recent).
- LFU (*Least Frequently Used*): Se înlocuieşte blocul care a fost utilizat cel mai puțin.

Există şi o tehnică bazată pe înlocuirea aleatoare a unui bloc, fără a se ține cont de utilizarea acestuia.

8.5.2.4. Tehnica de scriere

Înaintea înlocuirii unui bloc, trebuie să se testeze dacă acest bloc a fost modificat în memoria *cache*, dar nu a fost modificat şi în memoria principală. Dacă blocul nu a fost modificat în memoria *cache*, poate fi înlocuit, dar dacă a fost modificat, trebuie să se actualizeze memoria principală cu noul conținut al blocului.

Sunt posibile mai multe tehnici de actualizare a memoriei *cache*. Acestea trebuie să țină cont de două elemente. În primul rând, la memoria principală pot avea acces mai multe componente ale sistemului, de exemplu un modul de intrare/ieşire (I/E) utilizat pentru conectarea unui periferic. Dacă un cuvânt a fost modificat în memoria *cache*, cuvântul corespunzător al memoriei principale nu mai este valid. În al doilea rând, dacă un modul de I/E a modificat memoria principală, cuvântul din memoria *cache* nu mai este valid.

Tehnica cea mai simplă se numeşte *"write-through"*. În acest caz, toate operațiile de scriere se efectuează atât în memoria principală, cât şi în memoria *cache*, asigurându-se faptul că memoria principală este validă întotdeauna. Dezavantajul acesteia este că există un număr ridicat de operații de scriere care nu sunt necesare.

O altă tehnică, numită *"write-back"*, minimizează numărul operațiilor de scriere. in cazul acestei tehnici, actualizările se efectuează numai în memoria *cache*. La o operație de actualizare, este setat un bit asociat cu blocul din memoria *cache*. La înlocuirea unui bloc, acesta este scris în memoria principală numai dacă bitul asociat acestuia este setat. Problema care apare în acest caz este că anumite porțiuni din memoria principală sunt invalide, deci accesul la memorie de către un modul de I/E poate fi realizat numai prin memoria *cache*.