# Memoria cache
# - continuare -

Până acum am folosit doar schema de amplasare a blocurilor din memoria cache denumită corespondenţă directă.

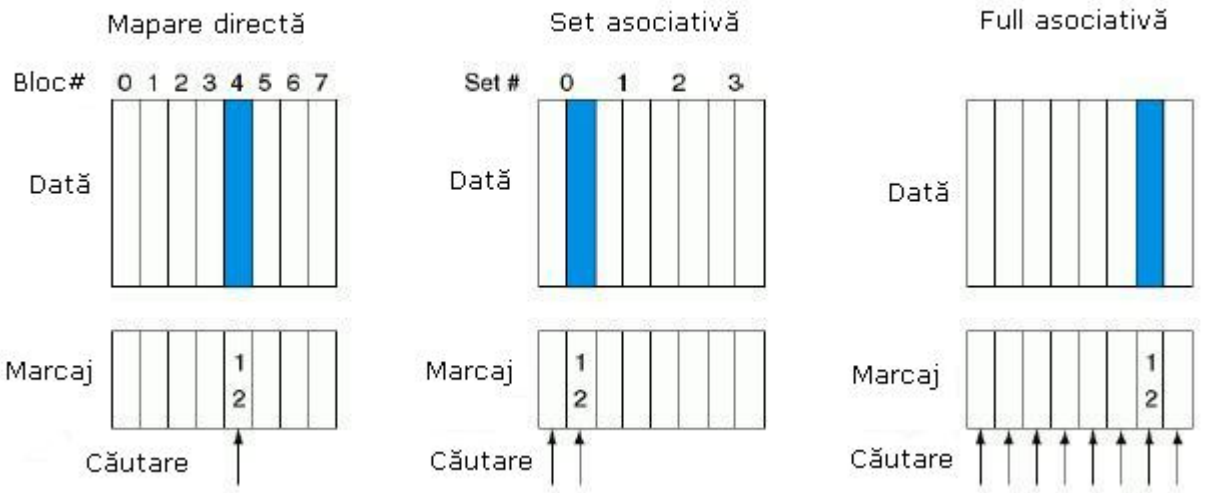
Schema în care un bloc de date poate fi amplasat în orice locaţie din memoria cache se numeşte schemă cu asociativitate totală.

Regăsirea blocului de date presupune examinarea tuturor locaţiilor de memorie cache => paralelizarea căutării

O altă schemă care este între cele două se numeşte schema cu asociativitate parţială.

În aceste tipuri de scheme memoria cache are un număr fix de locaţii în care se poate amplasa fiecare bloc de date. Deci vom avea un număr de seturi fiecare format din n blocuri de date.

Pentru regăsirea blocului este necesar să parcurgem toate blocurile unui set.




Într-o memorie cache cu asociativitate parţială, setul conţinând un anumit bloc de memorie este dat de relaţia:

(numărul blocului) modulo (numărul de seturi din memoria cache)

**OBSERVAŢIE** : Creşterea gradului de asociativitate reduce rata de eşec, dar creşte timpul de HIT.

**EXEMPLU**

Avem 3 memorii cache fiecare având 4 blocuri de câte 1 cuvânt. Cele trei memorii cache sunt cu asociativitate totală, asociativitate cu 2 căi şi corespondenţă directă.

Să se găsească numărul de eşecuri pentru fiecare dintre cele 3 scheme de amplasare având următoarea secvenţă de adrese de bloc: 0,8, 0, 6, 8.

**SOLUŢIE**

Cazul 1 – memoria cache cu corespondenţă directă

Detectăm blocul din memoria cache corespunzător adreselor date:

| Adresa blocului | Blocul memoriei cache |
|---|---|
| 0 | 0 modulo 4 = 0 |
| 6 | 6 modulo 4 = 2 |
| 8 | 8 modulo 4 = 0 |

Conţinutul memoriei cache după fiecare referinţă

| Adresa blocului de memorie accesat | HIT sau MISS | Blocul 0 | Blocul 1 | Blocul 2 | Blocul 3 |
|---|---|---|---|---|---|
| 0 | Miss | Mem(0) | | | |
| 8 | Miss | Mem(8) | | | |
| 0 | Miss | Mem(0) | | | |
| 6 | Miss | Mem(0) | | Mem(6) | |
| 8 | Miss | Mem(8) | | Mem(6) | |

Cazul 2. Memoria cache cu asociativitate parţială cu 2 căi conţine 2 seturi (indicii fiind 0 şi 1), fiecare având 4 elemente. Vom determina setul corespunzător fiecărei adrese a blocurilor

| Adresa blocului | Blocul memoriei cache |
|---|---|
| 0 | 0 modulo 2 = 0 |
| 6 | 6 modulo 2 = 0 |
| 8 | 8 modulo 2 = 0 |

Pentru înlocuire vom folosi LRU

| Adresa blocului de memorie accesat | HIT sau MISS | Set 0 | Set 1 | Set 2 | Set 3 |
|---|---|---|---|---|---|
| 0 | Miss | Mem(0) | | | |
| 8 | Miss | Mem(0) | Mem(8) | | |
| 0 | HIT | Mem(0) | Mem(8) | | |
| 6 | Miss | Mem(0) | Mem(6) | | |
| 8 | Miss | Mem(8) | Mem(6) | | |

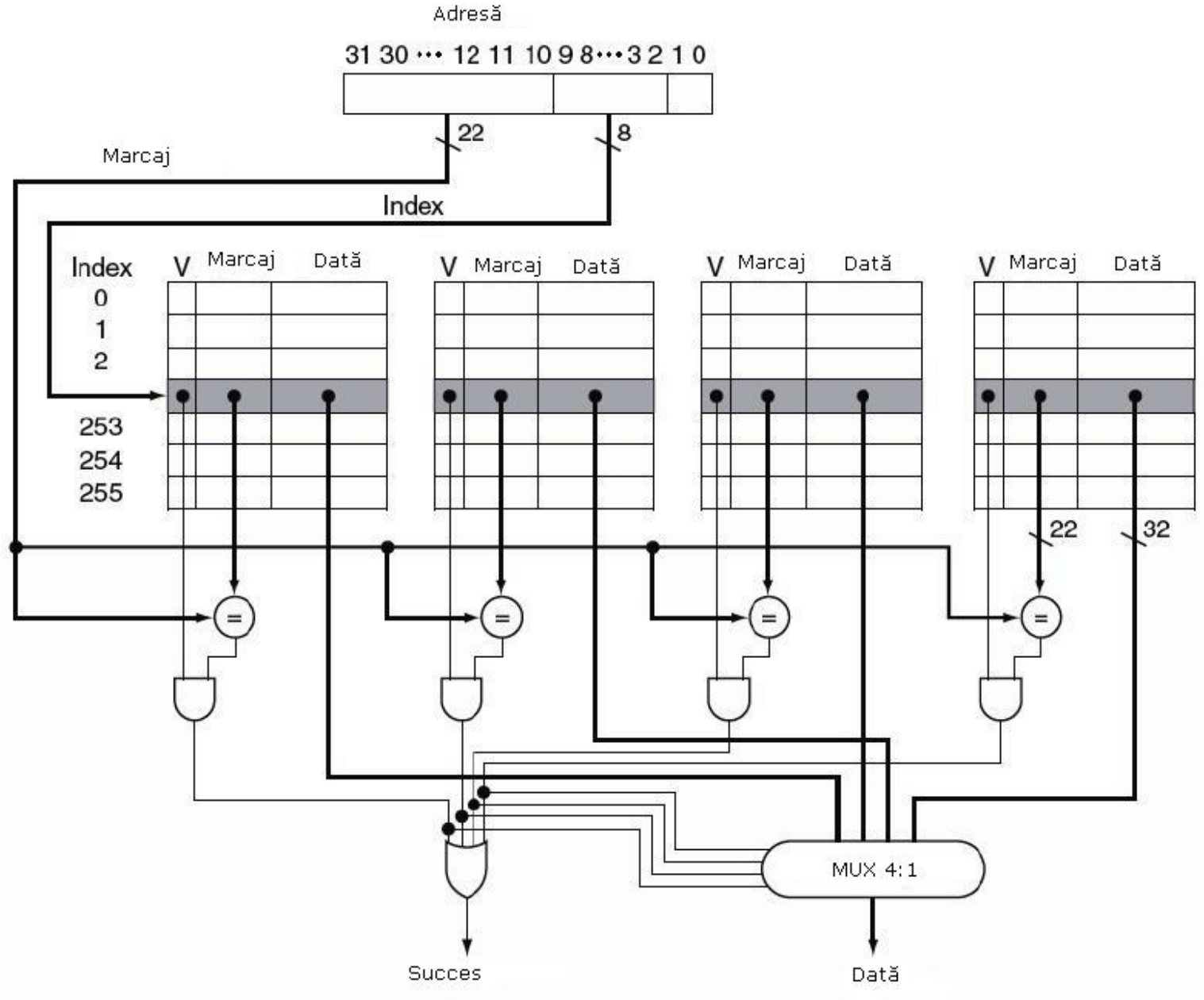
Avem doar 4 eşecuri, deci soluţia aceasta este mai bună decât precedenta

Memoria cache cu asociativitate totală

| Adresa blocului de memorie accesat | HIT sau MISS | Set 0 | Set 1 | Set 2 | Set 3 |
|---|---|---|---|---|---|
| 0 | Miss | Mem(0) | | | |
| 8 | Miss | Mem(0) | Mem(8) | | |
| 0 | HIT | Mem(0) | Mem(8) | | |
| 6 | Miss | Mem(0) | Mem(8) | Mem(6) | |
| 8 | HIT | Mem(0) | Mem(8) | Mem(6) | |

Aceasta este varianta optimă – avem doar 3 eşecuri

# Localizarea unui bloc în memoria cache cu asociativitate parţială



Păstrăm dimensiunea memoriei cache constantă şi încercăm să mărim asociativitatea => numărul de blocuri/set va creşte => va creşte numărul de comparaţii efectuate în paralel.

Creşterea asociativităţii cu un factor de doi va dubla numărul blocurilor din set şi va înjumătăţii numărul de seturi => descreşterea dimensiunii indexului cu 1 bit şi o creştere a dimensiunii marcajului cu 1 bit.

***Exemplu:*** Presupunem o memorie cache cu blocuri de 4Kb şi adrese de 32 de biţi. Să se găsească numărul total de seturi şi de biţi de marcaj pentru memoria cache cu corespondenţă directă, cu asociativitate parţială cu 2 şi 4 căi şi cu asociativitate totală.

a). Nr. de seturi = nr. de blocuri => log2(4Kb) = 12 biţi de index => (32-12)4K=80Kb numărul total al biţilor din marcaj

b). Cu 2 căi: 2K seturi şi numărul total al biţilor de marcaj este (32-11)*2*2K = 84Kb
Cu 4 căi: 1K seturi şi numărul total al biţilor de marcaj este (32-10)*4*1K = 88Kb

c). 1 set cu 4K blocuri iar marcajul are 32 de biţi => 32*4K*1 = 128 biţi pentru marcaj

## MEMORIA VIRTUALĂ

Memoria principală poate acţiona ca o memorie cache pentru nivelul de stocare secundar – uzual implementat cu discuri magnetice.

De ce avem nevoie de o memorie virtuală ?

1. Permite folosirea în comun, eficientă şi sigură a memoriei de către mai multe programe

2. Înlăturarea problemelor de programare cauzate de o memorie principală mică

Memoria principală trebuie să conţină doar porţiunile active ale programelor, deci va fi necesar un mecanism de protecţie a programelor între ele – trebuie să ne asigurăm că un program va scrie şi va citi doar din memoria principala atribuită lui.

Memoria virtuală implementează translatarea spaţiului de adrese al programului în adrese fizice. Acestă translatare asigură unicitatea spaţiului de adrese al unui program faţă de alte programe.

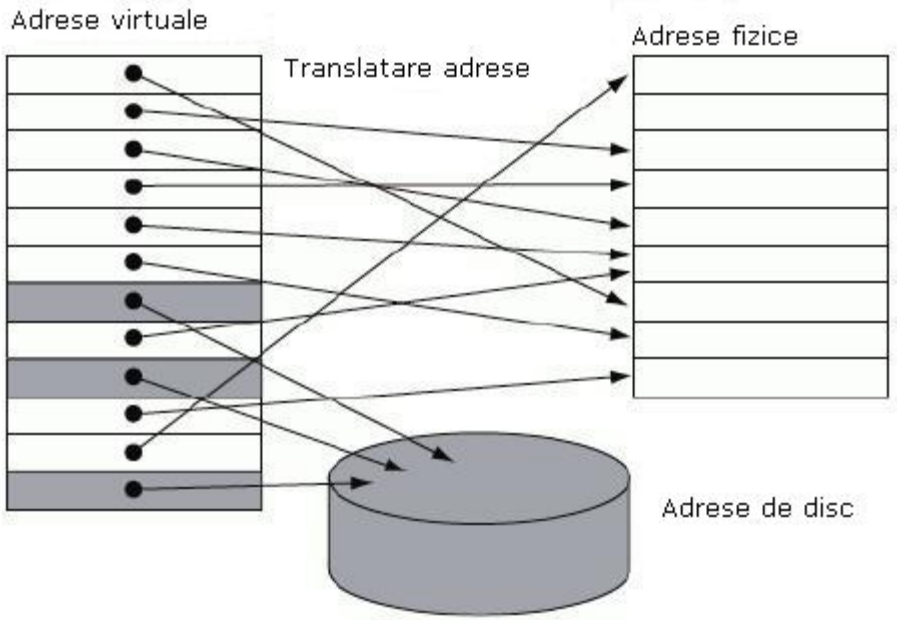
Până la apariţia acestui concept, depăşirea dimensiunii de memorie implica intervenţia programatorului.

Se împărţea programul în componente şi se trecerea la determinarea excluderilor mutuale.
Suprapunerile erau încărcate sau scoase din memorie în timpul execuţiei programului.
Apelurile dintre procedurile aflate în diferite module determinau suprapunerea unui modul cu altul.

Un bloc de memorie virtuală este denumit ***pagină***, iar un eşec la accesarea memoriei virtuale se numeşte ***page fault***.

Pentru memoria virtuală vom avea ***adrese virtuale*** ce sunt translatate în ***adrese fizice***.

***Exemplu***: Adresa virtuală este numele unei cărţi iar adresa fizică reprezintă locaţia cărţii în bibliotecă.




Procesorul generează adrese virtuale în timp ce memoria este accesată folosind adrese fizice

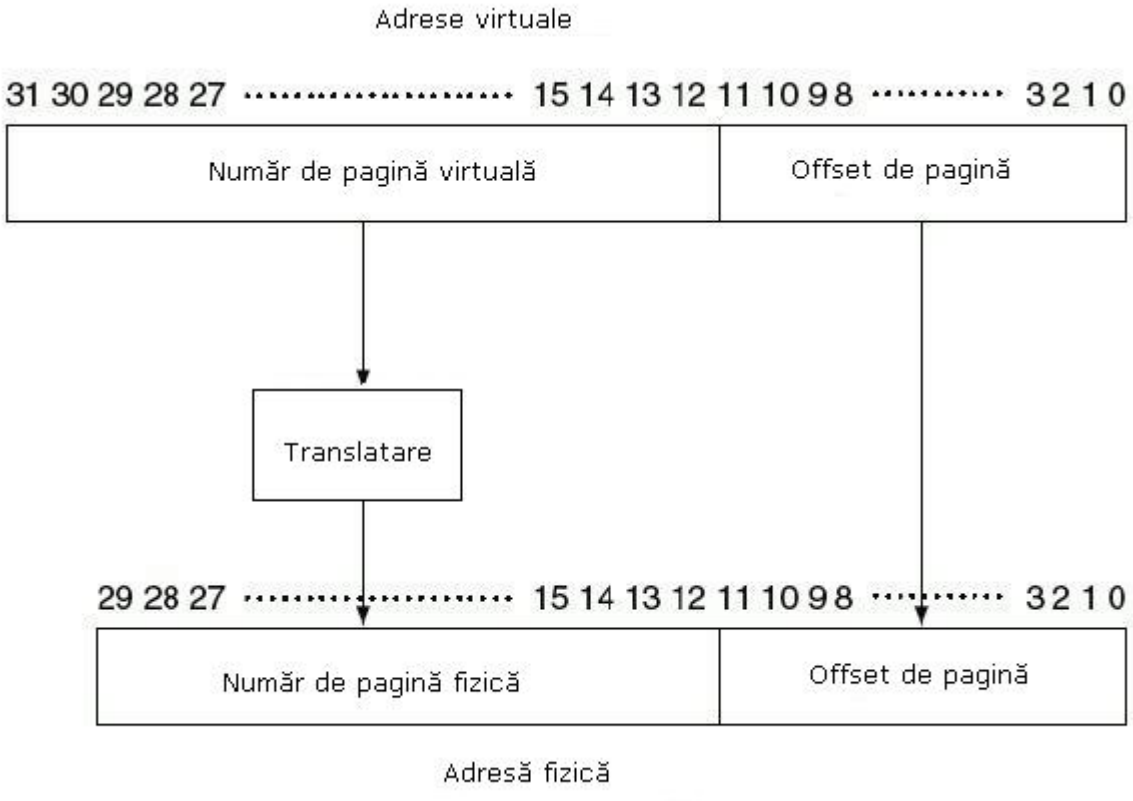
Ambele memorii sunt compuse din pagini (virtuale/fizice) între care există o corespondenţă de 1:1

Există posibilitatea ca o pagină virtuală să fie prezentă numai pe disc => imposibilitatea de a avea ca şi corespondenţă o pagină fizică.

O pagină fizică  poate fi folosită în comun – două adrese virtuale fac referire la aceeaşi adresă fizică.

Memoria virtuală oferă mecanismul de ***realocare*** – se calculează corespondenţa dintre adresele virtuale folosite de program şi diferitele adrese fizice , înainte ca adresele fizice să fie folosite de program.

Realocarea se face pe bază de blocuri de dimensiune fixă.



***Proiectarea sistemelor de memorie virtuală***

Dimensiunea paginilor trebuie să fie mare pentru a amortiza timpul de acces ridicat – 32KB sau 64KB spre exemplu

Amplasarea complet asociativă a paginilor – se reduce complet frecvenţa de page fault.

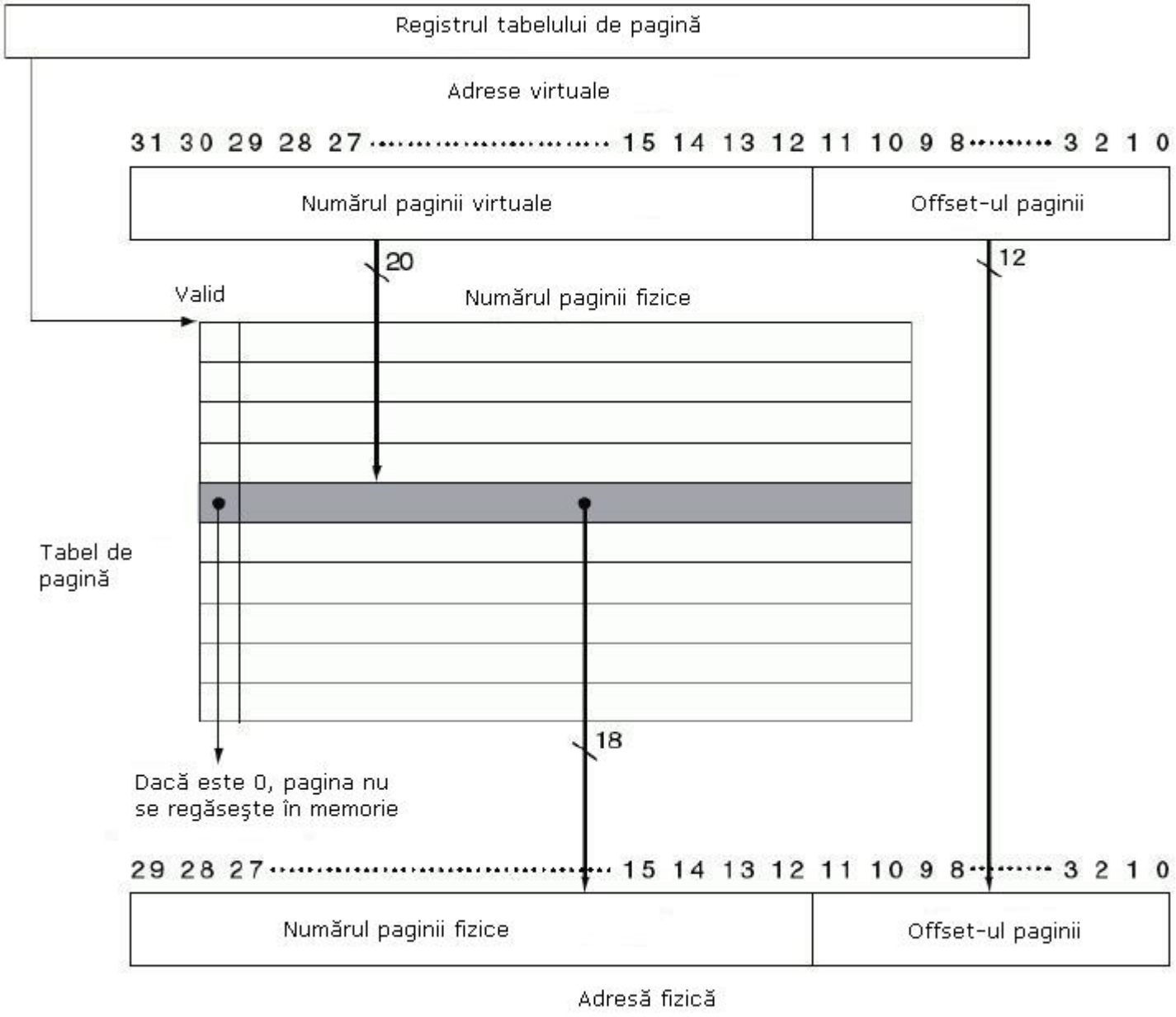
Page fault-urile pot fi tratate prin intermediul software-ului.

În memoria virtuală, paginile sunt localizate prin folosirea unui tabel ce indexează memoria; acestă structură se numeşte ***page table – tabel de pagină***.

Fiecare program are tabelul său de pagini, ce conţine corespondenţa dintre spaţiul său de adrese virtuale şi adresele memoriei principale.

Adresa unui tabel este memorată într-un registru ce indică adresa de început a tabelului – ***page table register***.

!!!! Presupunem că tabelul se găseşte într-o regiune fixă şi continuă din memorie


Registrul tabelului de pagină
Adrese virtuale
31 30 29 28 27 ..... 15 14 13 12 11 10 9 8 ..... 3 2 1 0
Numărul paginii virtuale
Offset-ul paginii
20
12
Valid
Numărul paginii fizice
Tabel de pagină
18
Dacă este 0, pagina nu se regăseşte în memorie
29 28 27 ..... 15 14 13 12 11 10 9 8 ..... 3 2 1 0
Numărul paginii fizice
Offset-ul paginii
Adresă fizică

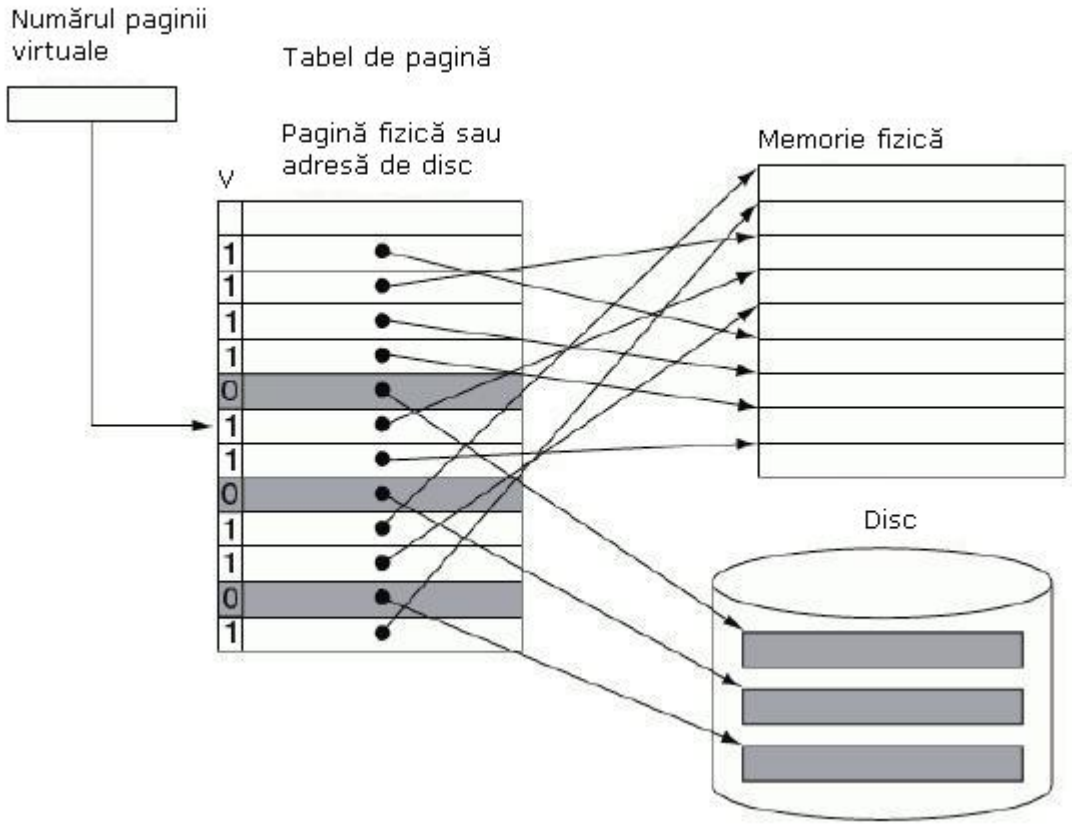
## *Page faults*

În cazul în care bitul de validare este 0 (apare page fault) controlul va fi preluat de SO prin intermediul unui mecanism de tratare a excepţiilor

SO-ul caută pagina în următorul nivel de memorie din ierarhie, adresa virtuală nu poate indica unde se găseşte pagina cerută.

SO-ul creează un spaţiu pe disc pentru toate paginile unui proces odată cu crearea acestuia, împreună cu o structură de date pentru înregistrarea locului unde este memorată fiecare pagină virtuală pe disc.

SO-ul creează de asemenea, o structură de date ce ţine evidenţa proceselor şi adreselor virtuale folosite de către fiecare pagină fizică.




Tabelele cu adresele paginilor fizice şi ale paginilor de pe disc vor fi memorate în două structuri de date separate.

Cantitatea de memoria folosită pentru memorarea tabelor de pagini este mare.

***Tehnici pentru reducerea volumului de memorie ocupat de tabelele de pagini şi minimizarea memoriei principale alocate acestora***

1. Utilizarea unui registru de limitare ce constrânge dimensiunea tabelului de pagini pentru un proces dat.

2. Majoritatea limbajelor necesită 2 porţiuni expandabile – una ce conţine stiva şi cealaltă ce conţine zona de acumulare (HEAP).

Dezavantaj – nu funcţionează bine atunci când spaţiul de adrese este folosit într-un mod discontinuu.

3. Aplicarea unei funcţii de căutare – HASHING – pentru adresa virtuală, astfel încât structura de date ce conţine tabelul de pagini să aibă o dimensiune egală doar cu numărul de pagini fizice existente în memoria principală.

4. Paginarea tabelor de pagini.

5. Utilizarea mai multor niveluri de tabele de pagini.