

Memoria cache
- continuare -

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

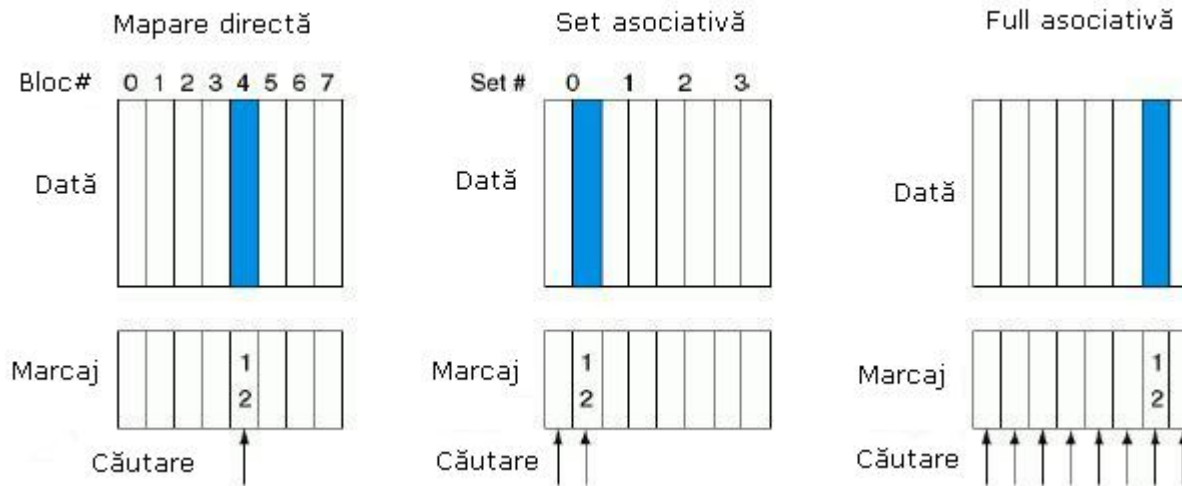
Schema în care un bloc de date poate fi amplasat în orice locație din memoria cache se numește **schemă cu asociativitate totală**.

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

O altă schemă care este între cele două se numește **schema cu asociativitate parțială**.

În aceste tipuri de scheme memoria cache are un număr fix de locații 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 parțială, setul conținând un anumit bloc de memorie este dat de relația:

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

OBSERVAȚIE : Creșterea gradului de asociativitate reduce rata de eșec, dar crește 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 și corespondență directă.

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

SOLUȚIE

Cazul 1 – memoria cache cu corespondență directă

Detectăm blocul din memoria cache corespunzător adreselor date:

Adresa blocului	Blocul memoriei cache
0	$0 \text{ modulo } 4 = 0$
6	$6 \text{ modulo } 4 = 2$
8	$8 \text{ modulo } 4 = 0$

Conținutul memoriei cache după fiecare referință

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 parțială cu 2 căi conține 2 seturi (indicii fiind 0 și 1), fiecare având 4 elemente. Vom determina setul corespunzător fiecărei adrese a blocurilor

Adresa blocului	Blocul memoriei cache
0	$0 \text{ modulo } 2 = 0$
6	$6 \text{ modulo } 2 = 0$
8	$8 \text{ modulo } 2 = 0$

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)		

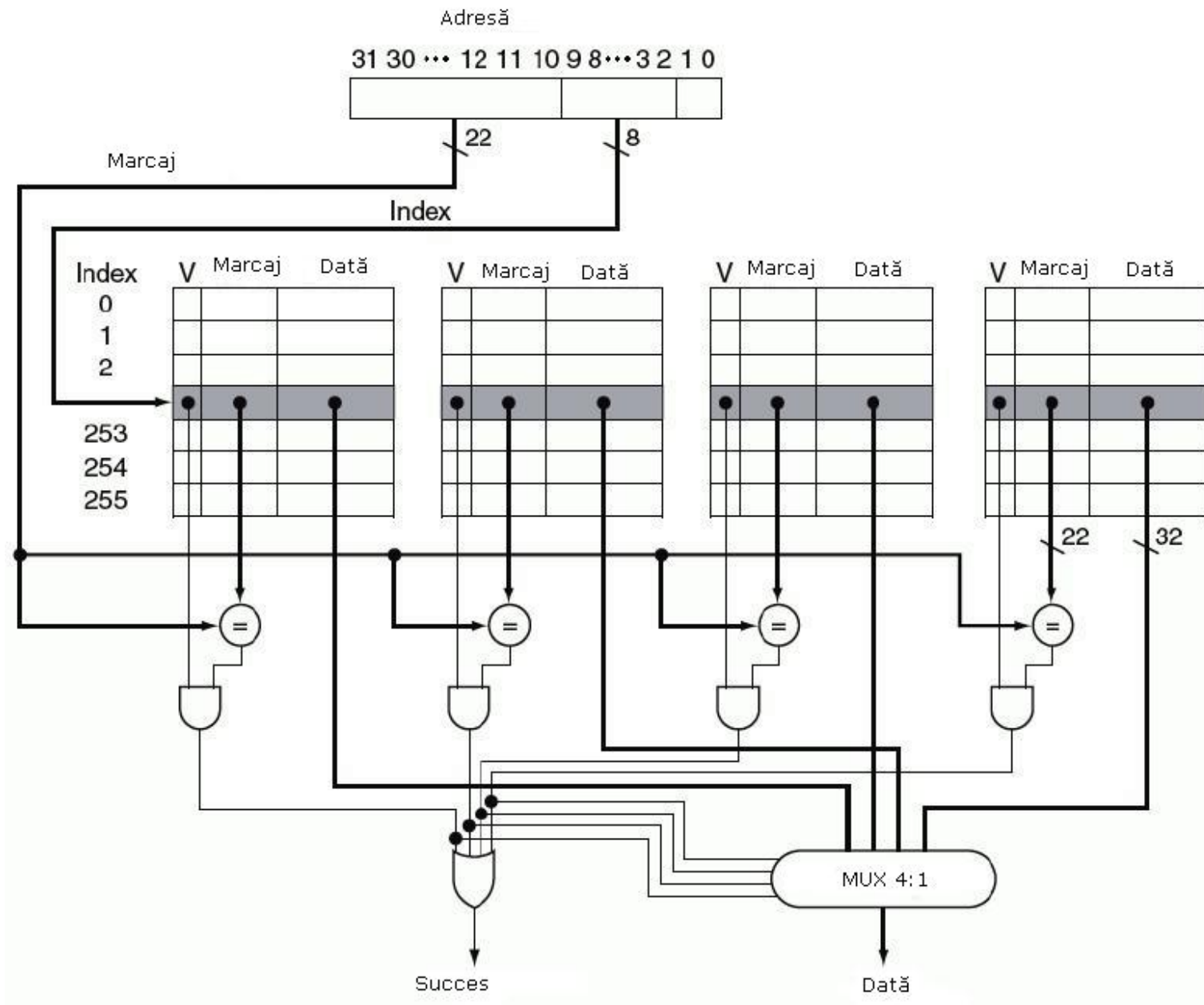
Avem doar 4 eșecuri, deci soluția aceasta este mai bună decât precedentă

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 eșecuri

Localizarea unui bloc în memoria cache cu asociativitate parțială



Păstrăm dimensiunea memoriei cache constantă și încercăm să mărim asociativitatea
=> numărul de blocuri/set va crește => va crește numărul de comparații efectuate în paralel.

Creșterea asociativității cu un factor de doi va dubla numărul blocurilor din set și va înjumătăți numărul de seturi => descreșterea dimensiunii indexului cu 1 bit și o creștere a dimensiunii marcajului cu 1 bit.

Exemplu: Presupunem o memorie cache cu blocuri de 4Kb și adrese de 32 de biți. Să se găsească numărul total de seturi și de biți de marcaj pentru memoria cache cu corespondență directă, cu asociativitate parțială cu 2 și 4 căi și cu asociativitate totală.

a). Nr. de seturi = nr. de blocuri => $\log_2(4Kb) = 12$ biți de index => $(32-12)4K=80Kb$
numărul total al biților din marcaj

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

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

MEMORIA VIRTUALĂ

Memoria principală poate acționa 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 și sigură a memoriei de către mai multe programe
2. Înlăturarea problemelor de programare cauzate de o memorie principală mică

Memoria principală trebuie să conțină doar porțiunile active ale programelor, deci va fi necesar un mecanism de protecție a programelor între ele – trebuie să ne asigurăm că un program va scrie și va citi doar din memoria principală atribuită lui.

Memoria virtuală implementează translatarea spațiului de adrese al programului în adrese fizice. Această translatare asigură unicitatea spațiului de adrese al unui program față de alte programe.

Până la apariția acestui concept, depășirea dimensiunii de memorie implica intervenția programatorului.

Se împărțea programul în componente și se trecerea la determinarea excluderilor mutuale.

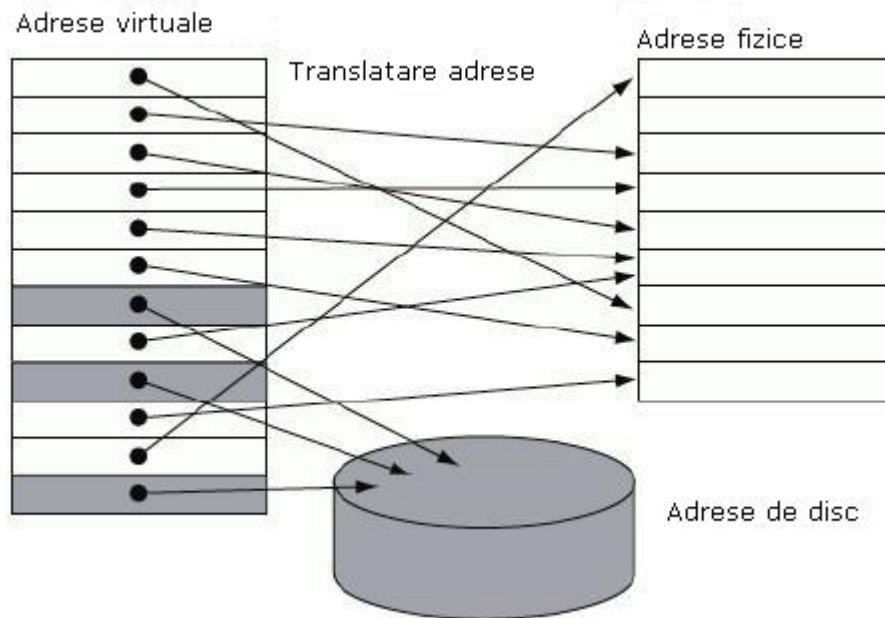
Suprapunerile erau încărcate sau scoase din memorie în timpul execuției programului.

Apelurile dintre procedurile aflate în diferite module determinau suprapunerea unui modul cu altul.

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

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

Exemplu: Adresa virtuală este numele unei cărți iar adresa fizică reprezintă locația cărții în bibliotecă.



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

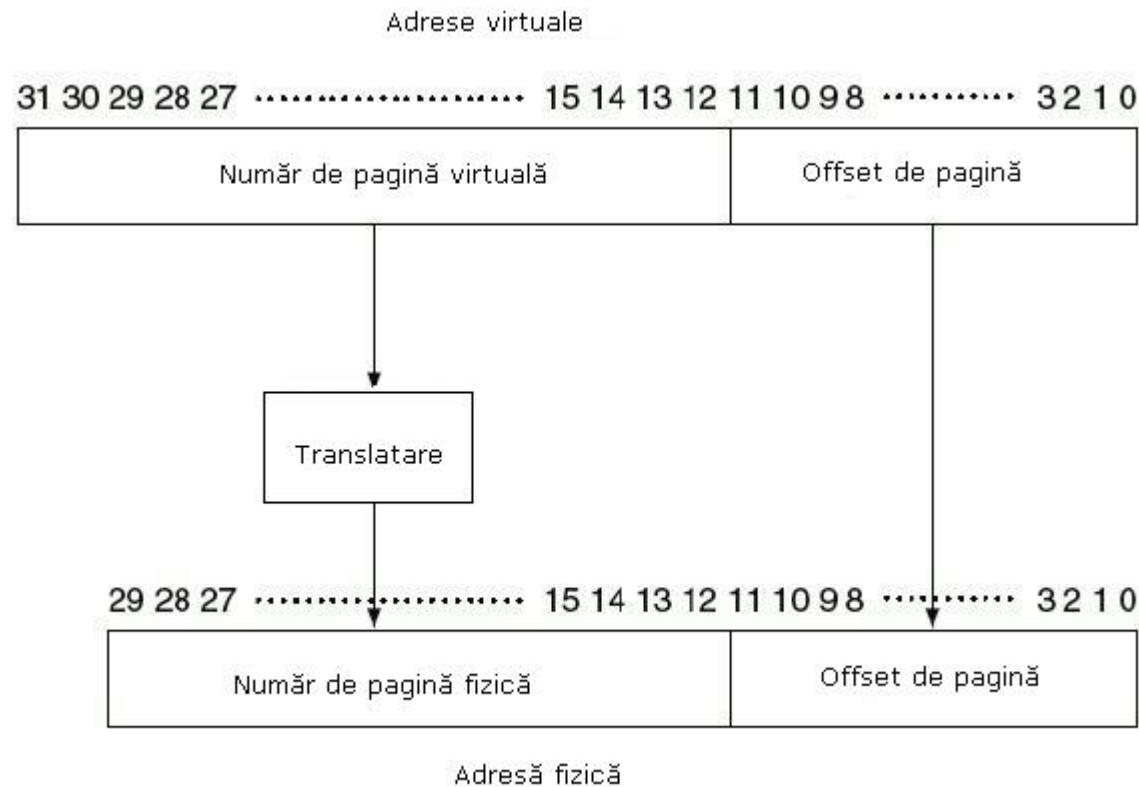
Ambele memorii sunt compuse din pagini (virtuale/fizice) între care există o corespondență de 1:1

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

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

Memoria virtuală oferă mecanismul de **realocare** – se calculează corespondența dintre adresele virtuale folosite de program și 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 frecvența de page fault.

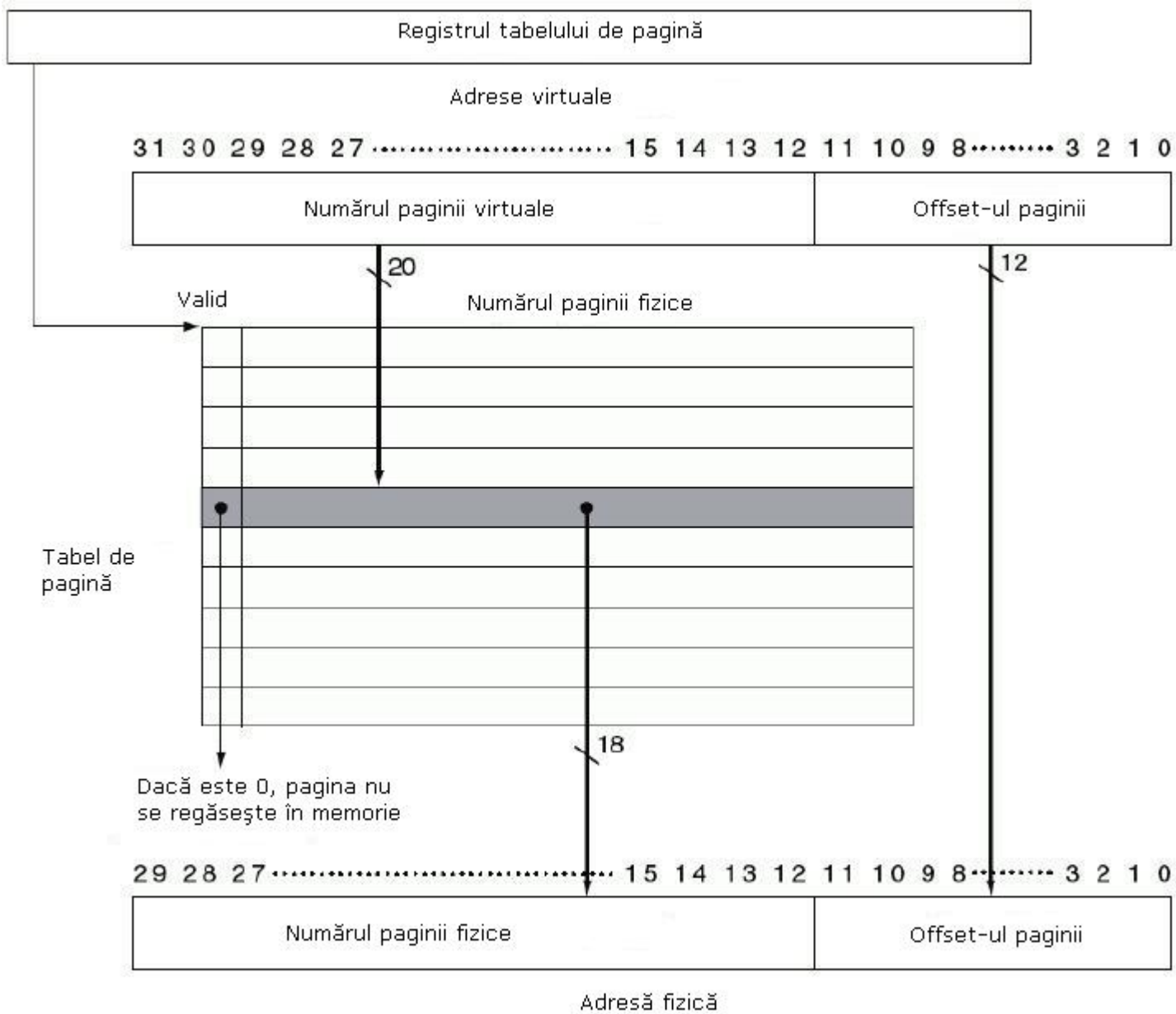
Page fault-urile pot fi tratate prin intermediul software-ului.

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

Fiecare program are tabelul său de pagini, ce conține corespondența dintre spațiul său de adrese virtuale și 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 găsește într-o regiune fixă și continuă din memorie



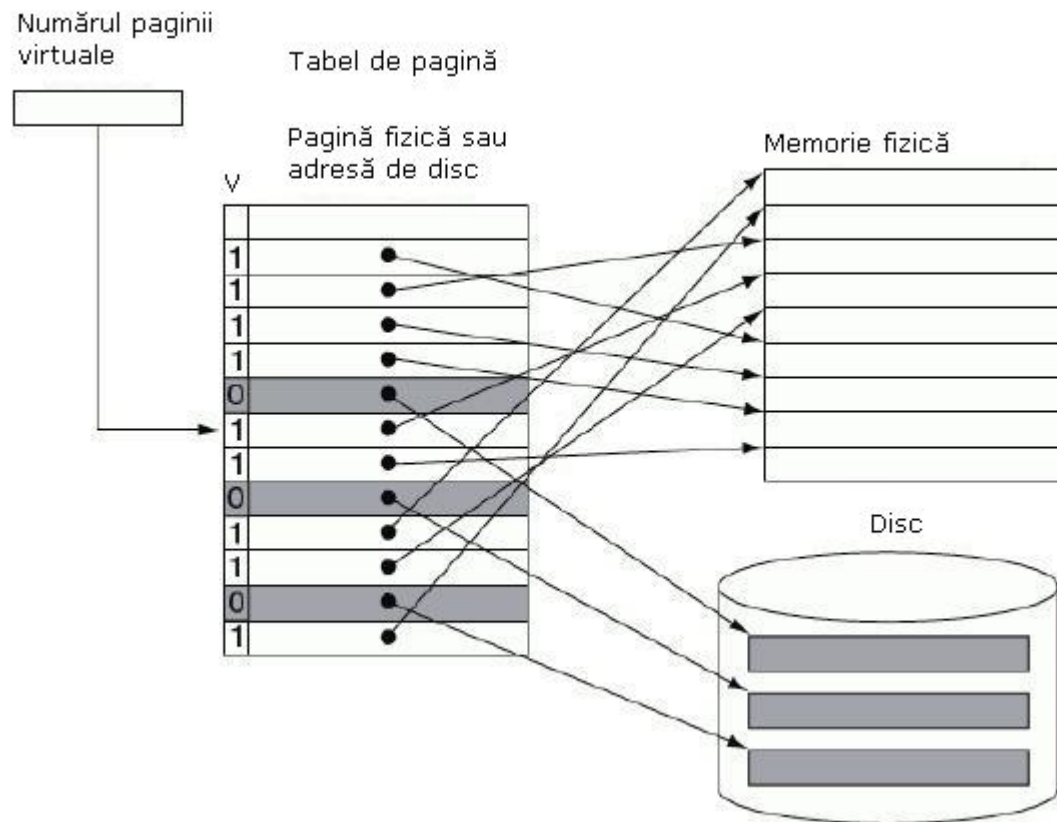
Page faults

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 excepțiilor

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

SO-ul creează un spațiu 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 ține evidența proceselor și adreselor virtuale folosite de către fiecare pagină fizică.



Tabelele cu adresele paginilor fizice și 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 și 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 porțiuni expandabile – una ce conține stiva și cealaltă ce conține zona de acumulare (HEAP).

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

3. Aplicarea unei funcții de căutare – HASHING – pentru adresa virtuală, astfel încât structura de date ce conține 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.