4 UNITATILE FUNCTIONALE ALE UNUI CALCULATOR

4.1 Modelul functional al calculatorului

Un sistem de calcul poate fi studiat din diferite puncte de vedere, rezultand astfel o ierarhie de niveluri (fig.4.1.1):

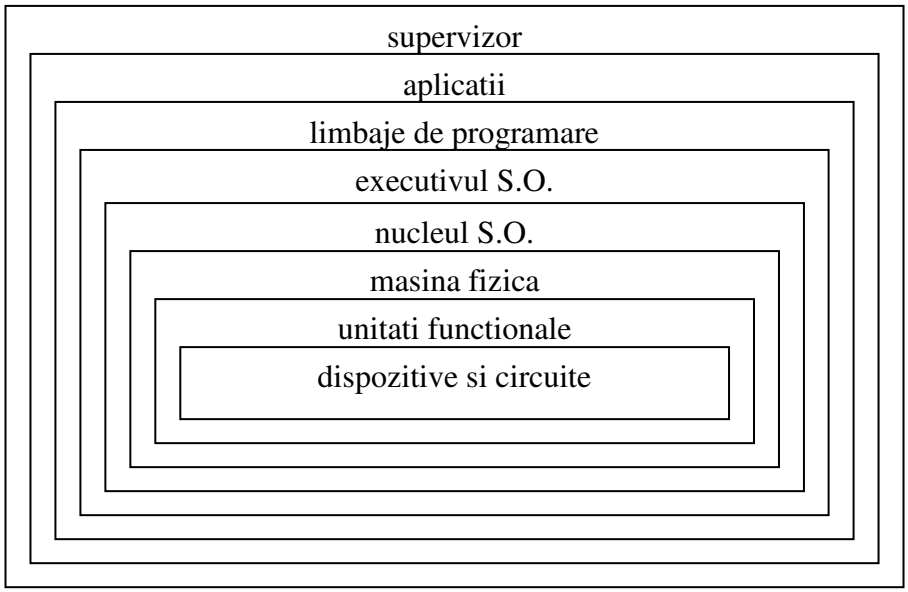


Fig.4.1.1 Sistem de calcul.

-nivelul dispozitivelor si circuitelor electronice: reprezinta nivelul cel mai de jos al sistemului de calcul, la care sistemul este reprezentat prin scheme electronice, masti de circuite integrate, diagrame de semnale, ecuatii analitice de functionare;

-nivelul unitatilor functionale: sistemul de calcul este reprezentat prin intermediul schemelor logice, descrieri de unitati functionale, specificari ale unitatilor de comanda;

-nivelul masinii fizice reprezinta nivelul arhitecturii calculatorului real. Este primul nivel pentru care se poate vorbi de programare;

-nivelul nucleului sistemului de operare contine o serie de functii speciale legate de alocarea memoriei, gestionarea procesorului, realizarea operatiilor de intrare / iesire la nivel fizic, tratarea cazurilor de exceptie (erori, intreruperi, etc.);

-nivelul executivului sistemului de operare introduce un set nou de functii dintre care realizarea operatiilor de intrare / iesire la nivel logic;

-nivelul limbajelor de programare: incepand de la acest nivel utilizatorul are la dispozitie un sistem de calcul pe care il poate utiliza prin intermediul diferitelor limbaje de programare;

-nivelul aplicatiilor este cel mai important nivel deoarece asigura rezolvarea de diferite probleme practice cum sunt: gestiune economica, proiectare inginereasca, simulare, astronomie, geofizica, etc.;

-nivelul supervizorului formeaza interfata directa dintre om si calculator, asigurand dialogul prin intermediul unui limbaj de comanda, reprezentand comenzile sistemului de operare (exemplu: sistemul de operare Unix), sau in mod grafic interactiv (exemplu: sistemul de operare Windows).

In cadrul acestui capitol se va face un studiu sumar al calculatorului la nivelul unitatilor functionale. Vor fi discutate unitatea aritmetica – logica, memoria, unitatea de comanda si subsistemul de intrare / iesire.

4.2 Unitatea aritmetica - logica

UAL executa toate operatiile aritmetice si logice din calculator, iar rezultatele sunt depuse in memorie sau trimise la unitatea de iesire pentru a fi furnizate in exterior. O unitate aritmetica – logica poate fi reprezentata prin simbolul (fig.4.2.1):

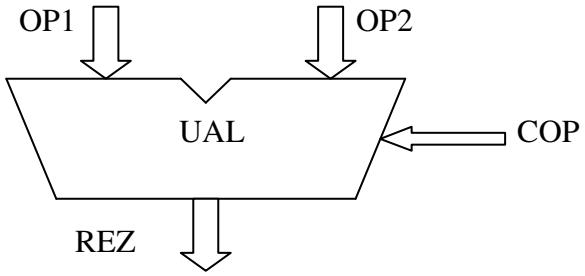


Fig.4.2.1 Unitate aritmetica – logica.

unde,

OP1, OP2 sunt cei doi operanzi reprezentati fiecare pe *n* biti;

REZ este rezultatul operatiei, de asemenea reprezentat pe *n* biti;

COP este codul de selectie a operatiei (codul operatiei), reprezentat pe *m* biti, deci se pot codifica in total 2^m operatii diferite.

UAL poate fi conectat intr-un sistem de calcul cu celelalte unitati functionale in diferite moduri:

Magistrala unica. Sistemul utilizeaza o magistrala unica, accesul la UAL realizandu-se prin intermediul a doua registre temporare, T1 si T2, inaccesibile programatorului (fig.4.2.2). Cei doi operanzi se incarca succesiv de pe magistrala in registrele temporare, iar rezultatul produs de UAL este plasat pe magistrala pentru a fi transferat la destinatie intr-un registru accesibil prin program, locatie de memorie sau port de iesire.

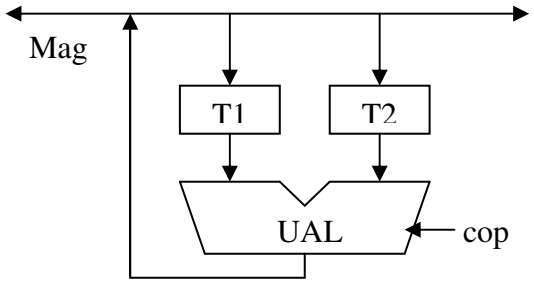


Fig.4.2.2 Conectarea UAL prin registre temporare.

Este cea mai simpla solutie, dar si cea mai putin eficienta. Astfel, in cazul cel mai defavorabil, cand cei doi operanzi si rezultatul sunt locatii de memorie, o operatie completa la nivelul UAL necesita trei cicluri succesive de memorie, doua citiri si o scriere.

Registru accumulator. Solutia utilizeaza un registru accumulator (Acc) in care se incarca de pe magistrala unul dintre operanzi (fig.4.2.3). Al doilea operand este furnizat de la sursa (registru utilizator, locatie de memorie sau port de I/E) direct prin intermediul magistralei. Rezultatul obtinut este incarcat in accumulator si poate reprezenta un operand pentru operatia urmatoare. Rezultatul final se va transfera din accumulator la destinatie, prin intermediul magistralei.

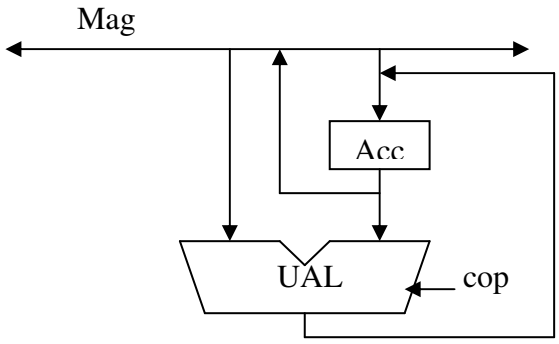


Fig.4.2.3 Conectarea UAL prin registru accumulator.

Aceasta solutie ofera o crestere a vitezei de executie a instructiunilor aritmetice – logice.

Interconectare cu trei magistrale. Aceasta solutie (fig.4.2.4) utilizeaza trei magistrale pentru furnizarea operanzilor (Mag A si Mag B) si pentru preluarea rezultatului (Mag C).

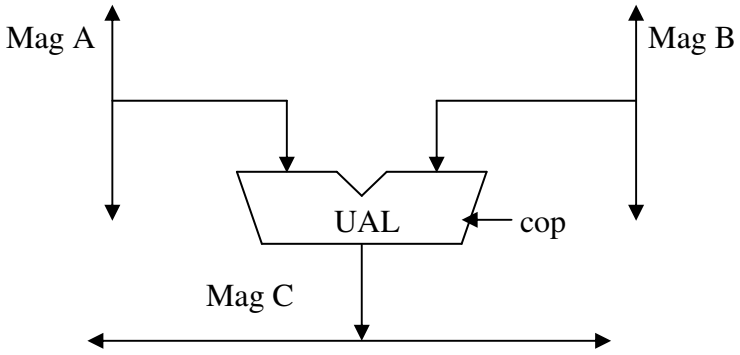


Fig.4.2.4 Conectarea UAL prin trei magistrale.

Chiar daca solutia are un cost ridicat, ofera in schimb viteza mare in prelucrarea operanzilor. Astfel, in cazul cel mai defavorabil, cand cei doi operanzi si rezultatul sunt locatii de memorie, dar in module independente conectate fiecare la cate o magistrala, o operatie completa la nivelul UAL necesita, in medie, un singur ciclu de memorie.

In continuare se vor studia cateva exemple de unitati aritmetice – logice.

Sumator-scazator pentru numere in cod complementar

Aceasta unitate aritmetica permite executia operatiilor de adunare si scadere pentru operanzi reprezentati in virgula fixa, cod complementar. Structura acestei unitatii este reprezentata in urmatoarea schema bloc (fig.4.2.5):

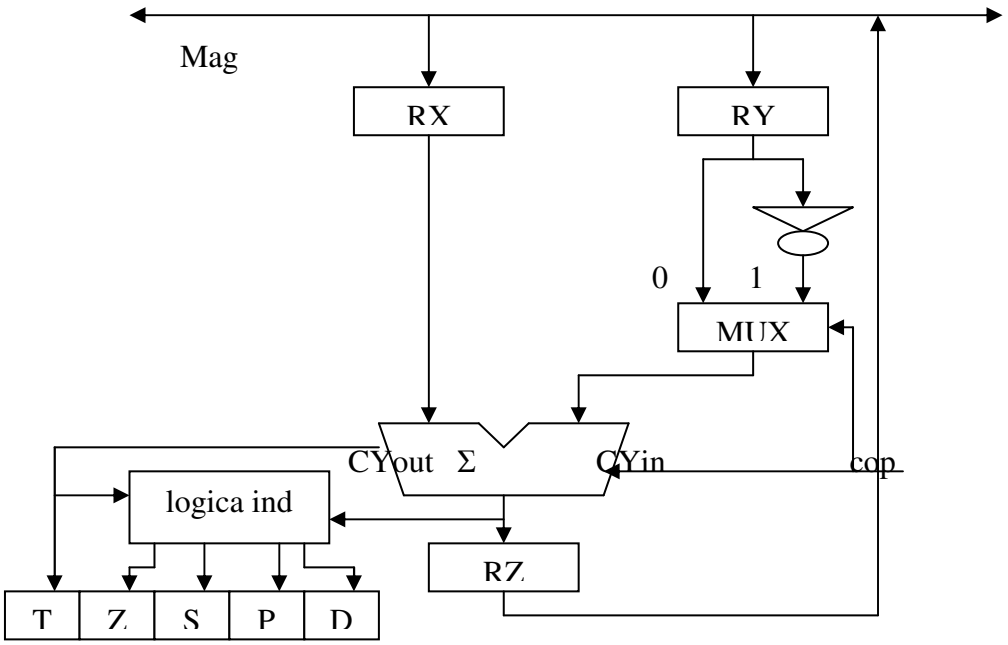


Fig.4.2.5 Sumator / scazator pentru cod complementar.

Resursele unitatii sumator – scazator sunt:

RX , RY sunt doua registre temporare (de tipul T1 si T2 din solutia generala precedenta), in care se incarca succesiv de pe magistrala cei doi operanzi, X si Y, pe *n* biti fiecare.

RZ este un registru care preia rezultatul si il plaseaza apoi pe magistrala pentru a fi transferat la destinatie.

Σ este un sumator, efectuand intotdeauna operatia de insumare.

MUX este un bloc de *n* multiplexoare 2:1 comandat de codul de selectie a operatiei *cop*, un singur bit (*cop* = 0 adunare, *cop* = 1 scadere). Pe intrarile 0 ale multiplexorului se conecteaza al doilea operand in forma directa (Y), iar pe intrarile 1, al doilea operand in forma complementata (Y#). Acelasi semnal de selectie *cop* este conectat si pe intrarea de transport a sumatorului *CYin*, care se aduna in rangul c.m.p.s. la o operatie efectuata. Astfel,

cop = 0 => Z = X + Y + 0 = X + Y (adunare)

cop = 1 => Z = X + Y# + 1 = X –Y (scadere).

Logica ind este o logica combinationala care pozitioneaza cativa bistabili, numiti indicatorii de conditie. Acesti indicatori se pozitioneaza in 1 sau 0 dupa fiecare operatie, reflectand caracteristici ale rezultatului. Astfel:

T este indicatorul de transport si se pozitioneaza in *T* = 1 daca exista transport de la rangul c.m.s al rezultatului si in *T* = 0 daca nu exista

transport (se observa in schema ca acest bistabil este pozitionat direct de catre iesirea de transport a sumatorului, *CYout*);

Z este indicatorul pentru rezultat zero si se pozitioneaza in *Z* = 1 pentru rezultat nul si *Z* = 0 pentru rezultat diferit de zero;

S este indicatorul de semn, fiind pozitionat de catre bitul c.m.s. al rezultatului, care este bitul de semn. Astfel, *S* = 1 pentru rezultat negativ si *S* = 0 pentru rezultat pozitiv;

P este indicatorul de paritate. Se poate utiliza paritatea para sau impara. In cazul utilizarii paritatii impare, acest indicator se pozitioneaza *P* = 1 daca suma modulo 2 peste toti bitii rezultatului este 0 si *P* = 0 in caz contrar;

D este indicatorul de depasire. Asa cum s-a studiat in capitolul 2, un transport la adunare in cazul numerelor reprezentate in cod complementar nu inseamna rezultat eronat, deci era necesar un alt indicator care sa semnaleze cazurile de depasire. Acest indicator se pozitioneaza *D* = 1 daca exista depasire (rezultat incorect) si *D* = 0 daca nu exista depasire (rezultat corect).

Unitate de inmultire in cod direct

Aceasta unitate aritmetica executa operatia de inmultire a doua numere reprezentate in virgula fixa cod direct. Schema bloc a dispozitivului este prezentata in figura urmatoare (fig.4.2.6):

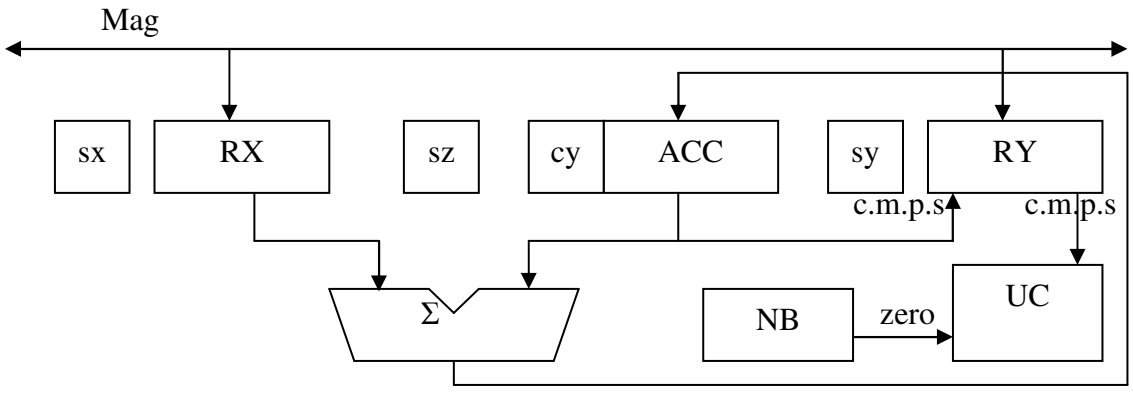


Fig.4.2.6 Unitate de inmultire pentru cod direct.

Unitatea de inmultire contine urmatoarele resurse:

RX registru pentru modulul deinmultitului;

sx bistabil pentru semnul deinmultitului;

RY registru pentru modulul inmultitorului;

sy bistabil pentru semnul inmultitorului;

sz bistabil pentru semnul rezultatului;

Σ sumator;

ACC registru accumulator pentru pastrarea rezultatelor partiale, iar in final va contine bitii c.m.s. ai modulului produsului;

cy bistabil pentru memorarea transportului furnizat de sumator;

NB numarator de biti pentru numararea pasilor operatiei de inmultire (inmultirea se executa intr-un numar de pasi egal cu numarul de biti folositi la reprezentarea modulului fiecarui operand);

UC unitate de comanda.

La inceputul operatie se incarca de pe magistrala, succesiv, cei doi operanzi in sx,RX si sy,RY, iar in ACC se incarca 0. Registrele cy,ACC si RY sunt registre cu deplasare, informatia poate fi deplasata o pozitie spre dreapta, cu trecerea bitului c.m.p.s. din ACC in RY (registrele sunt concatenate). La fiecare pas, unitatea de comanda testeaza bitul curent al inmultitorului, care este adus prin deplasari in pozitia c.m.p.s. din registrul RY. Daca acest bit este 1 se aduna prin intermediul sumatorului deinmultitul RX cu rezultatul partial ACC, rezultatul fiind memorat in ACC, iar apoi se executa o deplasare cu o pozitie spre dreapta pentru cy,ACC,RY. Daca bitul curent al inmultitorului este 0, nu se mai face operatia de adunare, dar se executa deplasarea. Numaratorul de biti este initializat la inceputul operatiei cu valoarea *n*-1 (numarul de biti pentru reprezentarea modulului fiecarui operand) si este decrementat la fiecare pas. Cand numaratorul ajunge in zero, anunta unitatea de comanda prin activarea unui semnal (*zero*) incheierea operatiei de inmultire. Rezultatul este obtinut pe lungime dubla in ACC,RY. In continuare, este prezentata o descriere precisa in pseudocod pentru functionarea unitatii de inmultire.

sx,RX ← X *//deinmultitul pe n biti*
sy,RY ← Y *// inmultitorul pe n biti*
cyy,ACC ← 0
NB ← n–1 *//numar de pasi*
sz ← *sx* ⊕ *sy* *//semnul rezultatului*
repeta
  daca $RY_0 = 1$ **atunci**
    cyy,ACC ← RX + ACC
  cyy,ACC,RY ← *cyy*,ACC,RY*2^{-1} *//deplasare dreapta o pozitie*
  NB ← NB – 1
cat timp NB ≠ 0
Z = *sz*,ACC,RY *//rezultatul pe 2n-1 biti, lungime dubla*

Unitate logica

In principiu o unitate logica genereaza rezultatele tuturor operatiilor logice posibile, iar rezultatul final este selectat prin multiplexare pe baza codului de operatie. In acest paragraf este descrisa o unitate logica foarte simpla care poate executa patru operatii logice cu operanzi pe 4 biti fiecare. Deci sunt necesari doi biti pentru reprezentarea codului operatiei *cop*. Aceste operatii sunt prezentate in tabela urmatoare:

cop	Operatia
00	X SAU Y
01	X SI Y
10	NU X
11	X • 2

Ultima linie din tabela semnifica deplasare stanga o pozitie. Schema de principiu a acestei unitati logice este prezentata in figura urmatoare (fig.4.2.7):

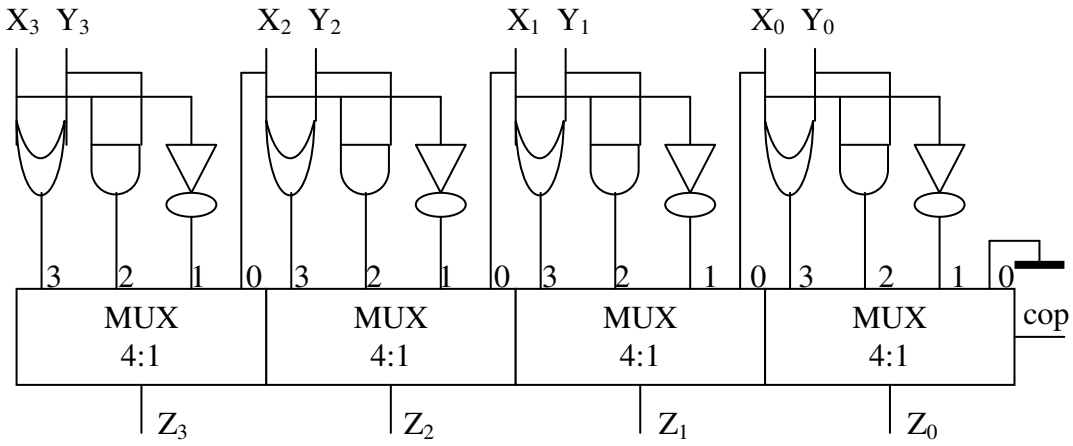


Fig.4.2.7 Unitate logica pe 4 biti.

Evident ca pentru operatiile logice SAU, SI, NU au fost utilizate porti logice de tipurile corespunzatoare. Operatia de deplasare s-a realizat prin decalarea conexiunilor, astfel in pozitia 3 s-a conectat bitul X_2, in pozitia 2 bitul X_1, ..., iar in pozitia 0 s-a conectat 0 logic (masa electrica), caci prin deplasare spre stanga se introduce un bit 0 prin dreapta numarului.

4.3 Memoria

Organizarea memoriei principale

Memoria principala a unui sistem de calcul poate fi formata din mai multe module, care pot fi si de tipuri diferite. In functie de modul de plasare a adreselor in aceste module se disting doua organizari de baza:

-cu intretesere de ordin superior: adrese succesive sunt plasate in acelasi modul de memorie. Activarea unui modul se face prin decodificarea bitilor superiori de adresa. Este organizarea specifica majoritatii sistemelor de calcul, in special a calculatoarelor secventiale.

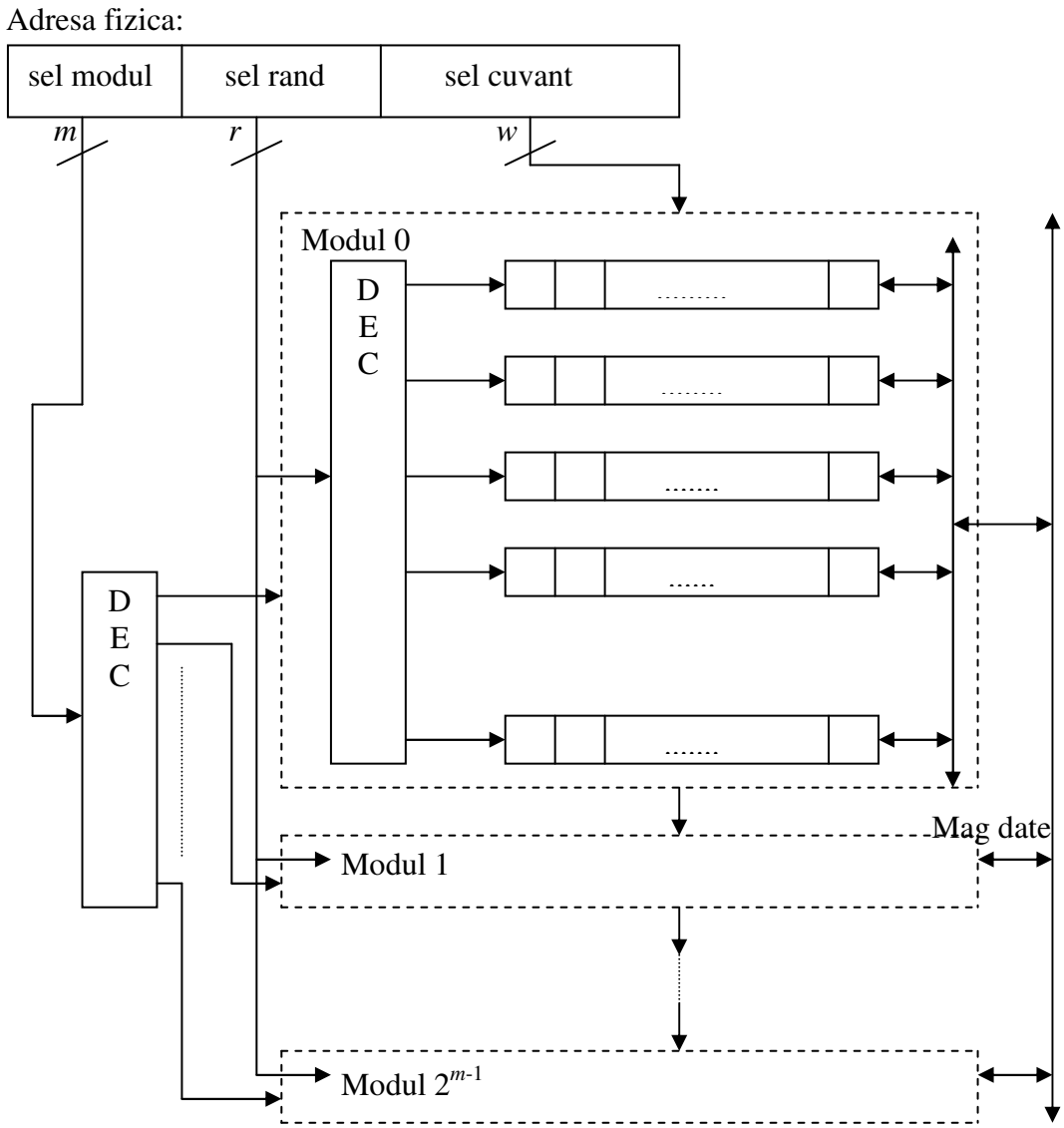


Fig.4.3.1 Organizarea unei memorii principale.

-*cu intretesere de ordin inferior*: adrese succesive sunt plasate in module consecutive. Pentru activarea unui modul se decodifica bitii inferiori de adresa. Aceasta este organizarea proprie unor sisteme speciale de calcul, cum sunt calculatoarele vectoriale. Prezinta avantajul ca se pot memora componentele unui vector la adrese succesive in module diferite, astfel componentele pot fi accesate in paralel, pentru a fi prelucrate.

In continuare este prezentata organizarea unei memorii principale pentru un sistem de calcul secvential, care utilizeaza intreteserea de ordin superior (fig.4.3.1). Cuvantul de adresa de *n* biti contine trei campuri:

-*m* biti c.m.s selecteaza un modul din cele maxim 2^m module;

-*r* biti mijlocii selecteaza in cadrul modulului activ un rand de circuite de memorie, din cele maxim 2^r randuri;

-*w* biti c.m.p.s. selecteaza cuvantul din cadrul randului de circuite de memorie.

Gestionarea memoriei principale

Pentru gestionarea memoriei principale se utilizeaza mecanisme de memorie virtuala, in principal memoria paginata si memoria segmentata. Prin intermediul acestor mecanisme, un programator ”vede” un spatiu de memorie avut la dispozitie (spatiu virtual) mult mai mare decat spatiul real (spatiu fizic) alocat de catre sistemul de operare. In memoria fizica se gaseste incarcata la un moment dat numai o parte a spatiului real al programului, iar pe masura ce sunt necesare si alte blocuri din program sau date, acestea sunt incarcate in memoria fizica, inlocuind blocuri care nu mai sunt necesare.

Memoria paginata. Mecanismul de memorie paginata imparte spatiul virtual al unui program in blocuri de lungime fixa, numite pagini (fig.4.3.2). Spatiul fizic de memorie la dispozitia programului este impartit in blocuri de aceeasi lungime numite cadre de pagina. In mod normal, spatiul fizic este (mult) mai mic decat spatiul virtual. La un moment dat numai anumite pagini din memoria virtuala se gasesc incarcate in cadrele de bloc din memoria fizica. In timpul executiei programului, cand programul lanseaza o adresa virtuala pentru o pagina care nu se gaseste in memoria fizica, aceasta pagina este adusa de pe disc (pe disc se gasesc toate paginile programului) si incarcata in memoria fizica, inlocuid o alta pagina. Translatarea unei adrese virtuale generate la executia programului in adresa fizica se face prin intermediul unei tabele de pagini, aflata in

memorie, care contine informatii despre toate paginile programului. In figura urmatoare este prezentat modul de translatare a unei adrese virtuale in adresa fizica:

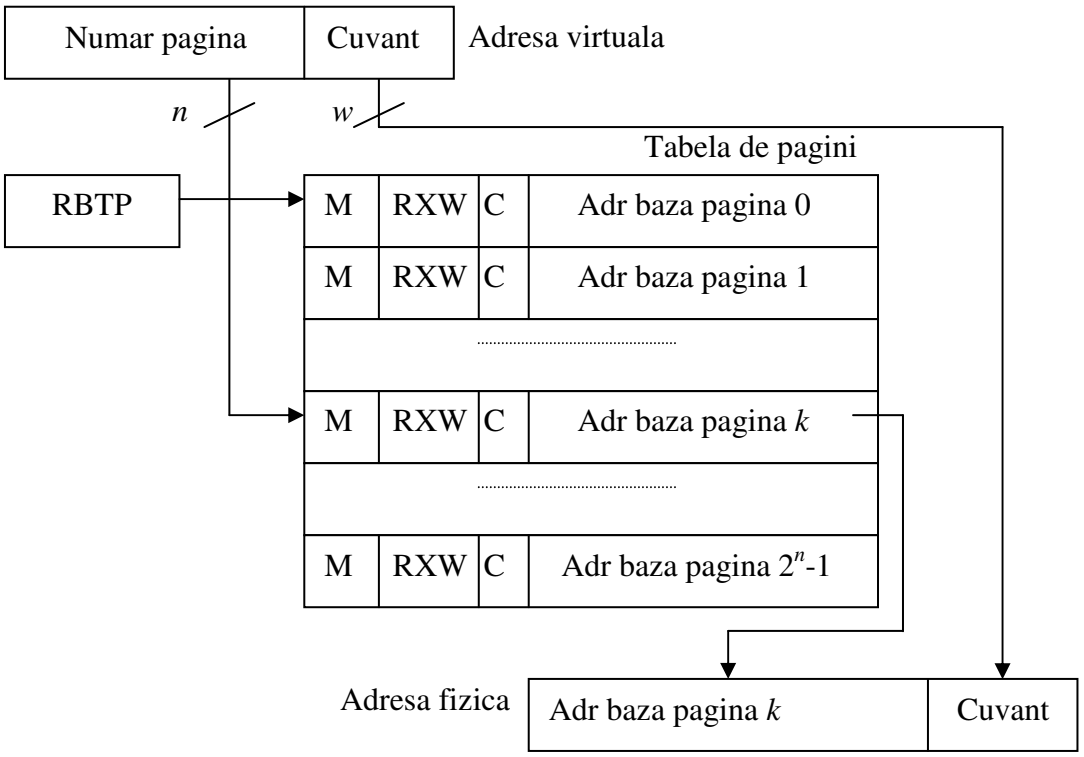


Fig.4.3.2 Memoria paginata.

La inceputul executiei programului in RBTP (registrul de baza al tabelei de pagini) se incarca de catre sistemul de operare adresa de inceput a tabelei de pagini a programului respectiv. In aceasta tabela, pentru fiecare pagina a programului se gasesc o serie de informatii de control si adresa de baza (de inceput a paginii in memoria fizica, de fapt bitii c.m.s. de adresa). Informatiile de control sunt:

-M (Memory) este un bit care indica faptul ca pagina este sau nu incarcata in memorie. Astfel, M=1 pagina se gaseste in memorie, M=0, pagina nu este in memoria fizica, aceasta trebuie incarcata de pe disc.

-RXW (Read, Execute, Write) specifica drepturile de acces ale programului in pagina respectiva, pentru citire date, executie cod de program, scriere date.

-C (Change) este un bit care specifica daca pagina a fost modificata sau nu (a avut loc cel putin o operatie de scriere in pagina respectiva). Astfel, C=0 pagina nu a fost modificata si C=1 pagina a fost modificata, deci este necesara actualizarea copiei de pe disc la eliminarea acesteia din memorie.

Adresa virtuala generata de program contine doua campuri: numarul paginii (*n* biti) si deplasamentul in cadrul paginii pentru accesul la cuvantul cerut (*w* biti). Deci, programul are cel mult 2^n pagini, iar dimensiunea unei pagini este de 2^w cuvinte. Numarul de pagina din adresa virtuala reprezinta un deplasament in cadrul tabelei de pagini, fata de inceputul tabelei indicata de continutul RBTP. Astfel, se acceseaza inregistrarea corespunzatoare paginii respective si daca pagina este in memoria fizica, dupa verificarea drepturilor de acces, se citeste adresa de inceput a paginii care se concateneaza cu deplasamentul in cadrul paginii indicand cuvantul, obtinandu-se adresa fizica. Rezulta ca:

Adresa fizica = Adresa de baza a paginii , Deplasament cuvant

Deoarece se realizeaza doar concatenarea celor doua componente ale adresei, mecanismul de memorie paginata este foarte rapid.

In momentul in care programul solicita o pagina care nu este incarcata in memoria fizica (evenimet lipsa de pagina), aceasta este adusa de pe disc si inlocuieste o pagina deja incarcata. Alegerea paginii din memoria fizica care va fi eliminata pentru aducerea paginii noi se face conform unui algoritm precis. Printre cei mai utilizati algoritmi de eliminare de pagina sunt:

Algoritmul LRU (Least Recently Used) elimina pagina care a fost accesata cel mai putin recent (cel mai demult).

Algoritmul MIN (algoritmul optimal al lui Belady) elimina pagina care va fi referita in viitor cel mai tarziu dintre paginile aflate la un moment dat in memoria fizica.

Algoritmul LFU (Least Frequently Used) elimina pagina care a fost utilizata cel mai putin frecvent (cel mai rar). Pentru implementarea acestui algoritm este necesar sa se utilizeze cate un contor de referinte pentru fiecare pagina aflata in memoria fizica, contor incrementat pentru fiecare acces la pagina respectiva.

Algoritmul FIFO (First In First Out) elimina pagina care a fost incarcata prima in memoria fizica. Este necesara in acest caz o coada care sa memoreze ordinea in care au fost incarcate paginile in memorie.

Algoritmul Clock elimina un dezavantaj important al algoritmului FIFO. In cadrul algoritmului FIFO o pagina intens utilizata de program poate fi eliminata din memoria fizica pe motiv ca a fost incarcata prima. Algoritmul Clock tine in plus si o evidenta a utilizarii paginilor, evitand eliminarea unei pagini intens utilizate.

Algoritmul RAND (Random) alege aleator o pagina pentru eliminare.

In timp ce algoritmul MIN este anticipativ, deci neimplementabil, ceilalti algoritmi sunt neanticipativi, deci implementabili. Totusi algoritmul MIN poate fi util pentru evaluarea performantelor celorlalti algoritmi de eliminare de pagina. In continuare, se prezinta performantele acestor algoritmi (fig.4.3.3) in cadrul unei reprezentari grafice a numarului de evenimente lipsa de pagina in functie de dimensiunea spatiului fizic aflat la dispozitia programului.

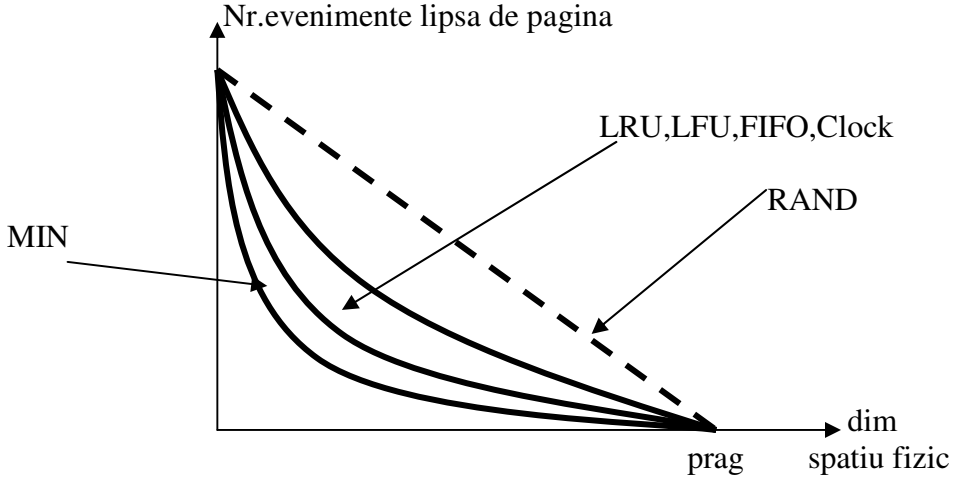


Fig.4.3.3 Performantele algoritmilor de inlocuire de pagina.

Cele mai bune performante le ofera algoritmul MIN, cele mai slabe algoritmul RAND, iar ceilalti algoritmi se situeaza in zona mijlocie a performantelor. Se observa ca pentru o anumita valoare de prag a dimensiunii spatiului fizic de memorie nu se mai genereaza evenimente lipsa de pagina, aceasta deoarece toate paginile programului se gasesc incarcate in memoria fizica.

Memoria segmentata. Mecanismul de memorie segmentata imparte spatiul virtual al unui program in blocuri de lungime variabila, numite segmente (fig.4.3.4). Spatiul fizic de memorie la dispozitia programului este utilizat pentru incarcarea unor segmente. De asemenea, spatiul fizic este (mult) mai mic decat spatiul virtual. La un moment dat numai anumite segmente din memoria virtuala se gasesc incarcate in memoria fizica. In timpul executiei programului, cand programul lanseaza o adresa virtuala pentru un segment care nu se gaseste in memoria fizica, acest segment este adus de pe disc (pe disc se gasesc toate segmentele programului) si incarcat in memoria fizica, inlocuid un alt segment. Translatarea unei adrese virtuale generate la executia programului in adresa fizica se face prin intermediul unei tabele de segmente, aflata in memorie, care contine

informatii despre toate segmentele programului. In figura urmatoare este prezentat modul de translatare a unei adrese virtuale in adresa fizica:

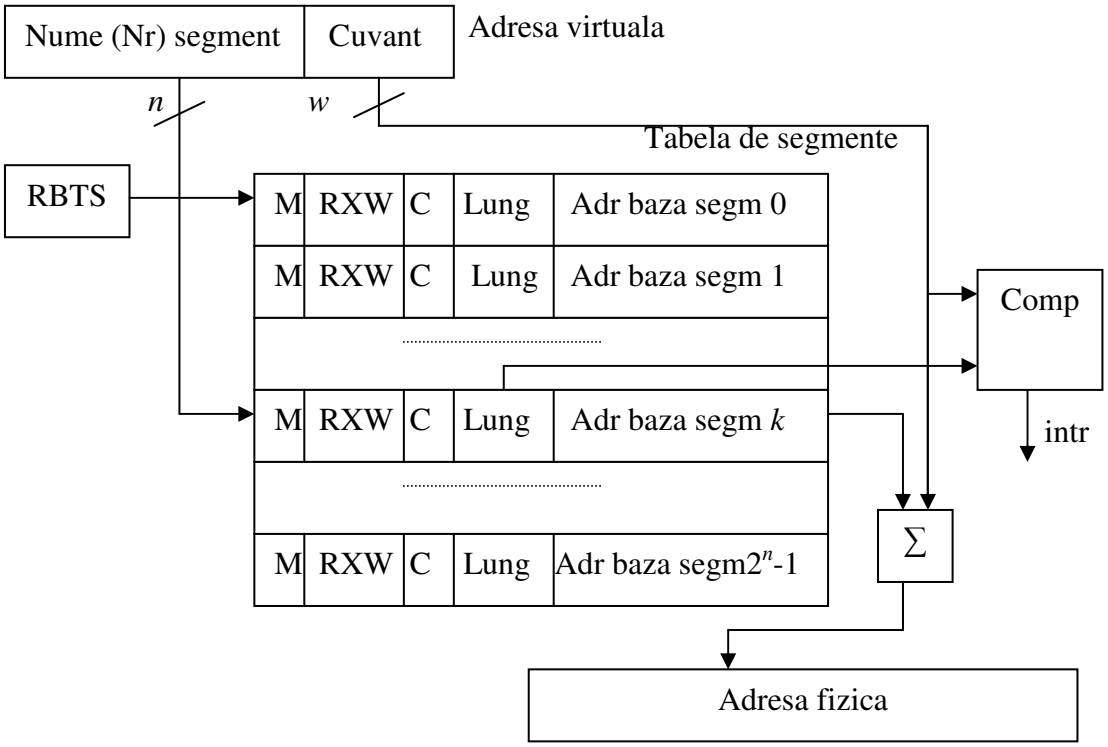


Fig.4.3.4 Memoria segmentata.

La inceputul executiei programului in RBTS (registrul de baza al tabelei de segmente) se incarca de catre sistemul de operare adresa de inceput a tabelei de segmente corespunzatoare programului respectiv. In aceasta tabela, pentru fiecare segment al programului se gasesc o serie de informatii de control si adresa de baza (de inceput a segmentului in memoria fizica). Informatiile de control sunt foarte asemanatoare cu cele de la memoria paginata:

-M (Memory) este un bit care indica faptul ca segmentul este sau nu incarcat in memorie. Astfel, M=1 segmentul se gaseste in memorie, M=0, segmentul nu este in memoria fizica, acesta trebuie incarcat de pe disc.

-RXW (Read, Execute, Write) specifica drepturile de acces ale programului in segmentul respectiv, pentru citire date, executie cod de program, scriere date.

-C (Change) este un bit care specifica daca segmentul a fost modificat sau nu (a avut loc cel putin o operatie de scriere in segmentul respectiv). Astfel, C=0 segmentul nu a fost modificat si C=1 segmentul a fost modificat, deci este necesara actualizarea copiei de pe disc la eliminarea acestuia din memorie.

Adresa virtuala generata de program contine doua campuri: numele (numarul) segmentului si deplasamentul in cadrul segmentului pentru accesul la cuvantul cerut.. Numarul de segment din adresa virtuala reprezinta un deplasament in cadrul tabelei de segmente, fata de inceputul tabelei indicata de continutul RBTS. Astfel, se acceseaza inregistrarea corespunzatoare segmentului respectiv si daca segmentul este in memoria fizica, dupa verificarea drepturilor de acces, se citeste adresa de inceput a segmentului care se aduna cu deplasamentul in cadrul segmentului indicand cuvantul, obtinandu-se adresa fizica. Rezulta ca:

Adresa fizica = Adresa de baza a segm + Deplasament cuvant

In plus fata de memoria paginata, in cadrul inregistrarii corespunzatoare fiecarui segment din tabela de segmente se gaseste memorata lungimea segmentului respectiv. La un acces se compara lungimea segmentului citit din tabela cu deplasamentul cuvantului din cadrul adresei virtuale. In cazul in care deplasamentul este mai mare decat lungimea, se impiedica accesul in afara segmentului generandu-se o intrerupere. Aceasta are rol de protectie intre programele utilizator si de protectie a sistemului de operare insusi.

Memoria cache

Memoria cache sau memoria intermediara este un nivel al ierarhiei memoriei unui sistem de calcul, plasat intre registrele procesorului si memoria principala, avand viteza foarte mare de lucru si continand blocuri de cod si date necesare la momentul respectiv. Pe masura ce executia programului necesita alte informatii, acestea sunt aduse din memoria principala si incarcate in memoria cache pentru a putea fi accesate mai rapid de catre procesor. Organizarea unui sistem de calcul avand memroie cache este prezentata in figura urmatoare:

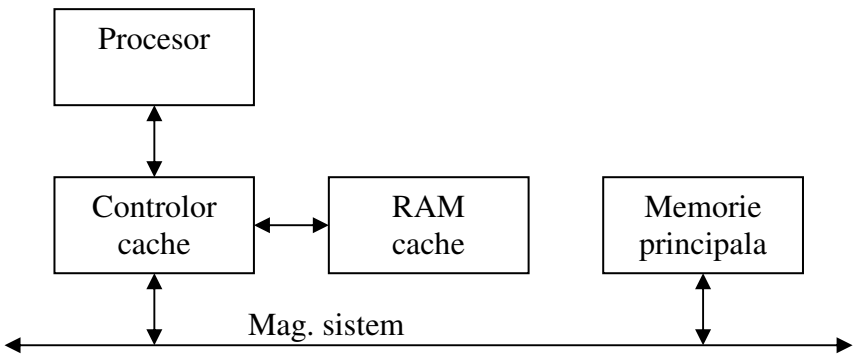


Fig.4.3.5 Organizarea unui sistem cu memorie cache.

Memoria cache contine doua componente principale: controlorul de cache si memoria RAM cache. Controlorul preia cererile de acces la memorie ale procesorului si le dirijeaza catre RAM-ul cache daca informatia ceruta se gaseste aici, sau, in caz contrar, cererile sunt transferate catre memoria principala pentru aducerea informatiei solicitate. Ram-ul cache contine informatia utila, cod si date, care sunt necesare la momentul respectiv pentru executia programului.

In cadrul controlorului cache se gaseste un modul important, reprezentat de directorul cache-ului. Acesta este realizat in principiu dintr-o memorie asociativa (adresabila dupa continut), ceea ce creste costul sistemului. In director se afla informatii de identificare a blocurilor de date si cod care se gasesc curent incarcate in RAM-ul cache.

Din punct de vedere al memoriei cache, memoria principala este impartita in blocuri de dimensiune fixa si relativ mica. Memoria cache este de asemenea impartita in zone de aceeasi dimensiune, numite cadre de bloc. La un moment dat anumite blocuri din memoria principala pot fi incarcate in cadrele de bloc din memoria cache. In momentul in care un bloc cerut de procesor nu se gaseste in memoria cache, acesta va fi adus din memoria principala si va inlocui un bloc aflat deja in memoria cache.

Exista o serie de solutii de incarcare a blocurilor din memoria principala in cadrele de bloc din memoria cache. Principalele solutii vor fi discutate in continuare. Se va presupune ca memoria principala are 256 Mocteti, memoria cache are 1 Moctet, iar dimensiunea unui bloc este de 16 octeti. Rezulta ca:

-memoria principala are

$$M = 256 \bullet 2^{20} / 2^4 = 2^{28} / 2^4 = 2^{24} \text{ blocuri}$$

-memoria cache are

$$C = 1 \bullet 2^{20} / 2^4 = 2^{20} / 2^4 = 2^{16} \text{ cadre de bloc}$$

Maparea directa. In cadrul acestei solutii (fig.4.3.6) blocul numarul *i* din memoria principala se poate incarca numai in cadrul de bloc numarul *i* modulo C (pentru exemplul considerat, in cadrul de bloc numarul *i* modulo 65536). Solutia directa de mapare si structura cuvantului de adresa fizica sunt ilustrate in figura urmatoare:

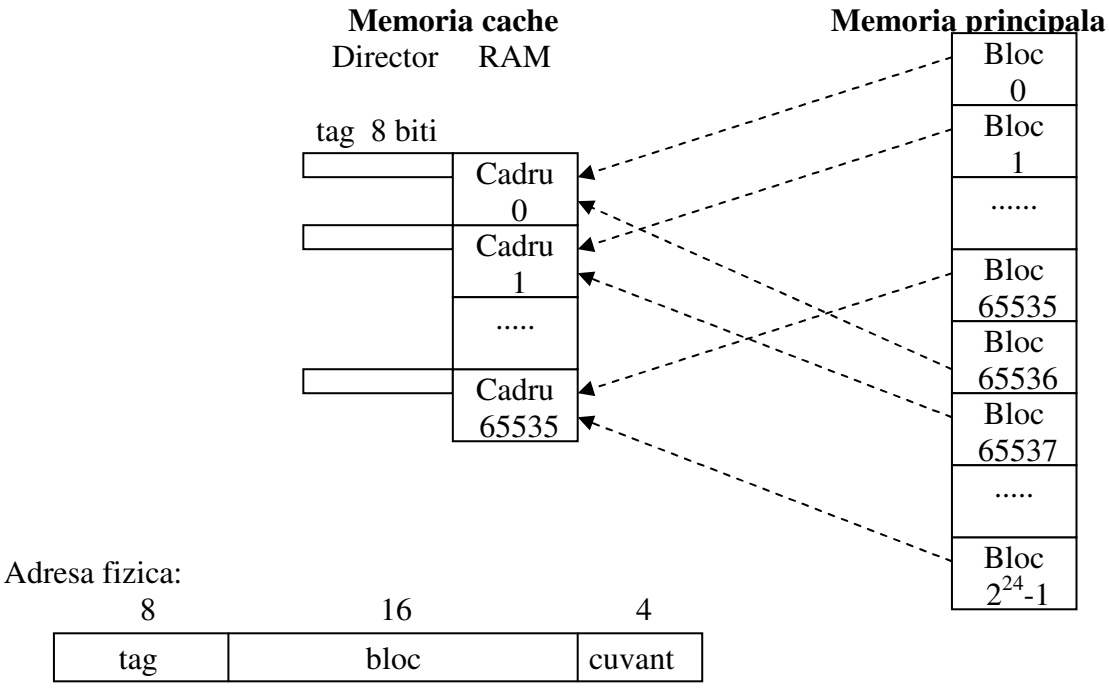


Fig.4.3.6 Maparea directa.

Pentru o cerere a procesorului cei 16 biti mijlocii din adresa fizica (bloc) sunt utilizati pentru citirea din memoria cache. Se citesc in paralel blocul respectiv si tagul asociat (identificatorul blocului), comparandu-se tagul citit cu tagul din adresa. La coincidenta, din blocul citit se livreaza procesorului cuvantul selectat pe baza ultimilor biti c.m.p.s. semnificativi de adresa (cuvant). Daca tagurile difera, se incarca in memoria cache blocul cerut din memoria principala.

Avantajul solutiei este costul relativ scazut, caci directorul memoriei cache se poate implementa cu memorie RAM obisnuita, fara sa fie necesara memorie asociativa scumpa. Dezavantajul este reprezentat prin performantele mai scazute ale acestei solutii: astfel, in cazul in care doua blocuri intens cerute de procesor (un bloc de cod si un bloc de date prelucrate de acel cod) se mapeaza in acelasi cadru de memorie cache, performantele sistemului scad dramatic, chiar sub performantele unui sistem fara memorie cache.

Maparea complet asociativa. In cadrul acestei solutii un bloc din memoria principala se poate incarca in oricare cadru de bloc din memoria cache (fig.4.3.7). Solutia complet asociativa de mapare si structura cuvantului de adresa fizica sunt ilustrate in figura urmatoare:

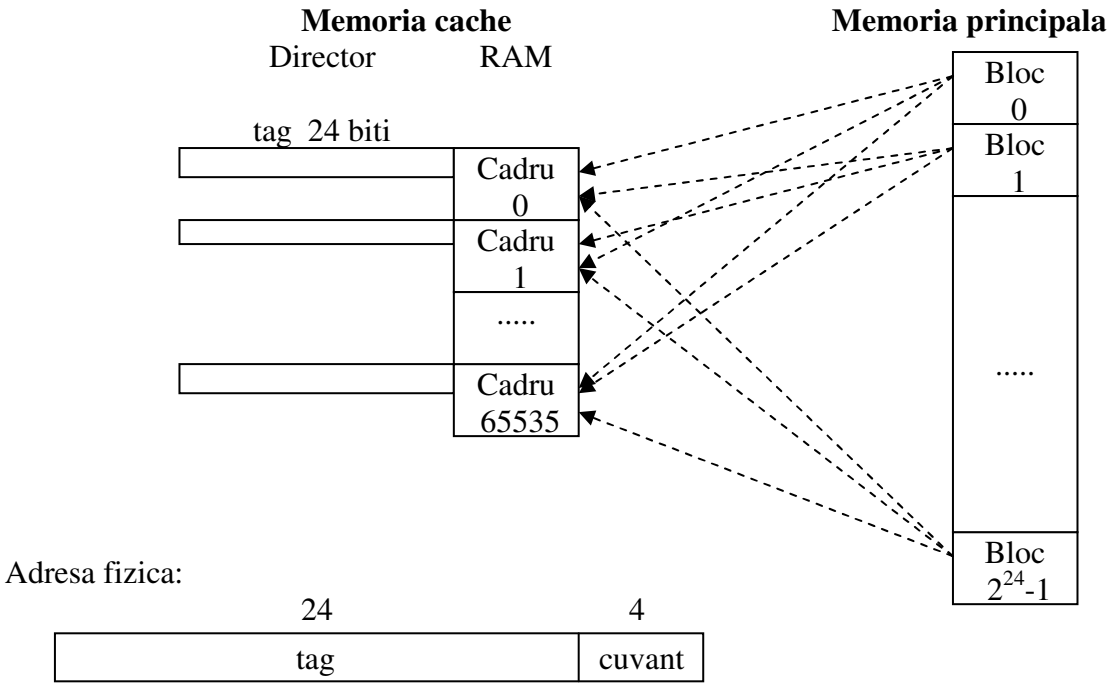


Fig.4.3.7 Maparea complet asociativa.

In aceasta solutie directorul este implementat cu memorie asociativa. Pentru o cerere a procesorului se face o cautare asociativa in memoria directorului pe baza bitilor c.m.s. din adresa (tag). Daca blocul cautat se gaseste in memoria cache, se face un al doilea acces la RAM-ul cache si se citeste blocul respectiv, livrandu-se procesorului cuvantul selectat pe baza ultimilor biti c.m.p.s. de adresa (cuvant). Solutia prezinta flexibilitate maxima, dar si costul implementarii este mare, datorita memoriei asociative mari reprezentand directorul (in cadrul exemplului considerat, memoria directorului contine 64 kcuvinte de 24 de biti).

Maparea set asociativa. In aceasta solutie (fig.4.3.8) se considera ca memoria cache este impartita in seturi de dimensiune E (de exemplu E = 2 cadre de bloc / set). Deci, memoria cache contine S = C / E (in exemplul considerat, S = C / E = 2^{16} / 2^{1} = 2^{15} = 32768) seturi. Blocul *i* din memoria principala poate fi incarcat in oricare cadru de bloc apartinand setului i modulo S din memoria cache. Solutia set asociativa de mapare si structura cuvantului de adresa fizica sunt ilustrate in figura urmatoare:

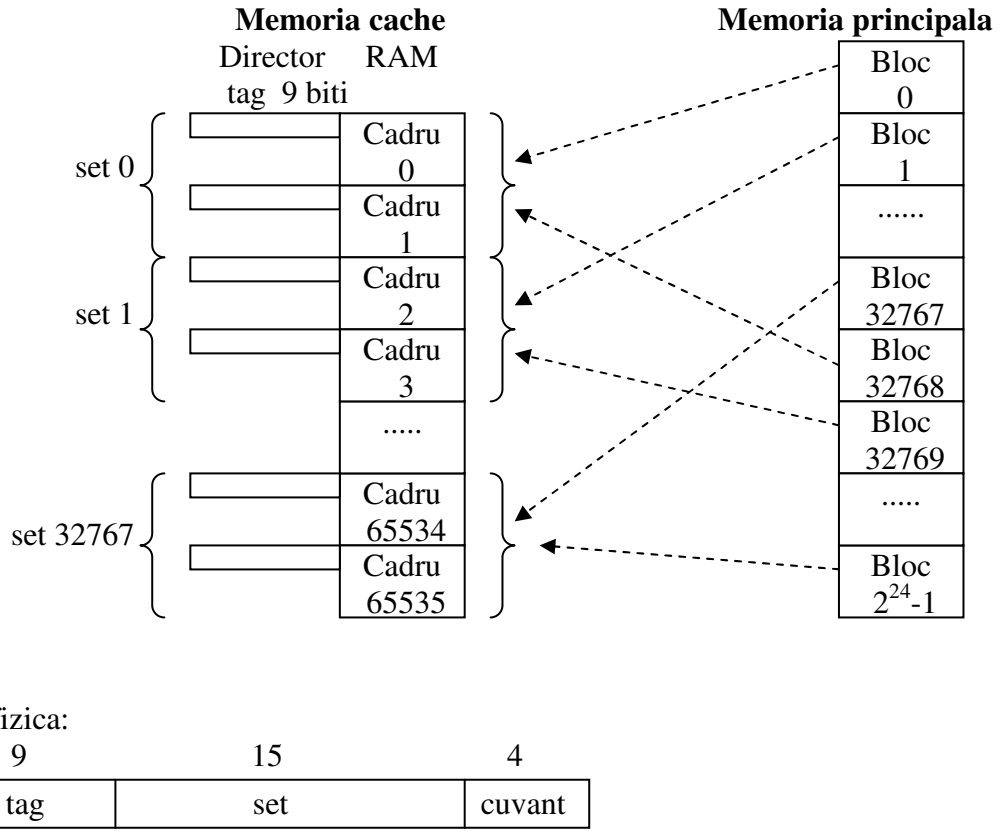
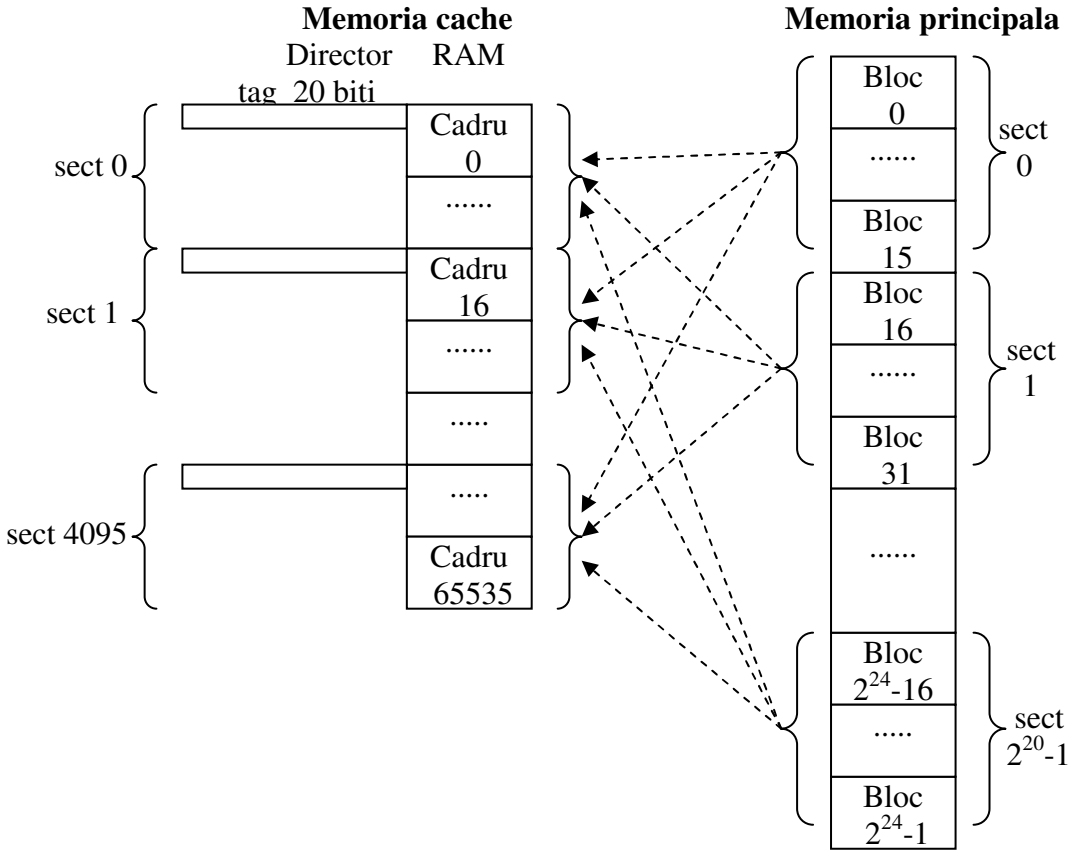


Fig.4.3.8 Maparea set asociativa.

Avantajul acestei solutii este ca memoria asociativa mare a directorului de la solutia complet asociativa a fost impartita in mai multe memorii mai mici. In cadrul exemplului considerat memoria asociativa de 64 kcuvinte de 24 de biti a fost impartita in 32768 memorii de 2 cuvinte fiecare de 9 biti. Se obtine astfel o scadere importanta a costului implementarii, dar si o micsorare a flexibilitatii. Maparea set asociativa este cea mai utilizata in practica. Aceasta reprezinta un compromis intre maparea directa si cea complet asociativa. Intr-adevar, daca se considera cazurile extreme in care E = 1, se obtine S = C (maparea directa), respectiv E = C, rezulta S = 1 (maparea complet asociativa).

Maparea sector asociativa. In aceasta solutie (fig.4.3.9) se considera ca atat memoria cache cat si memoria principala sunt impartite in sectoare de dimensiune E (de exemplu E = 16 blocuri / sector). Deci, memoria cache contine S = C / E (in exemplul considerat, S = C / E = 2^{16} / 2^4 = 2^{12} = 4096) sectoare, iar memoria principala contine P = M / E (in exemplul considerat, P = M / E = 2^{24} / 2^4 = 2^{20}) sectoare. Sectorul *i* din memoria principala poate fi incarcat in oricare cadru de sector din memoria cache, dar maparea blocurilor in cadrul unui sector este fixa. Solutia sector

asociativa de mapare si structura cuvantului de adresa fizica sunt ilustrate in figura urmatoare:



Adresa fizica:

20	4	4
tag(sector)	bloc	cuvant

Fig.4.3.9 Maparea sector asociativa.

Avantajul solutiei este miscsorarea dimensiunilor memoriei asociative, astfel pentru exemplul considerat memoria asociativa necesara este de 4096 de cuvinte de 20 biti. In cazul in care la o cerere a procesorului blocul nu se gaseste in memoria cache, este adus un intreg sector din memoria principala si incarcat in memoria cache. Transferul consuma timp apreciabil. O varianta a solutiei realizeaza transferul numai a unui singur bloc din sector, in pozitia corespunzatoare din cadrul de sector al memoriei cache, iar celelalte blocuri aflate in cadrul de sector, care apartin altor sectoare, sunt dezactivate. In aceasta solutie se utilizeaza pentru fiecare cadrul de bloc din memoria cache un bit de validare, care sa specifice daca informatia respectiva este valida sau nu.