## CUPRINS


1. INTRODUCERE ..... **1-1**

1.1 PARALELISMUL - O NOUĂ PARADIGMĂ ÎN ARHITECTURA CALCULATOARELOR ..... 1-2
*1.1.1 Sisteme cu prelucrare în Bandă de Asamblare* ..... *1-4*
*1.1.2 Procesoare paralele* ..... *1-4*
*1.1.3 Topologii de tip reţea* ..... *1-4*
1.2 CLASIFICAREA SISTEMELOR CU PRELUCRARE PARALELĂ (SPP) ..... 1-5
*1.2.1 Schema propusă de M. Flynn* ..... *1-5*
*1.2.2 Schema de clasificare a lui Feng* ..... *1-8*
*1.2.3 Schema de clasificare a lui I. Shore* ..... *1-10*
*1.2.4 Schema de clasificare a lui Handler* ..... *1-12*
1.3 REPREZENTAREA STRUCTURALĂ A SISTEMELOR CU PRELUCRARE PARALELĂ ..... 1-16
*1.3.1 Generalităţi* ..... *1-16*
*1.3.2 Unităţile funcţionale şi regulile pentru reprezentarea organizării lor* ..... *1-17*
*1.3.3 Comanda unităţilor* ..... *1-26*
1.4 REPREZENTAREA PMS A STRUCTURII CALCULATOARELOR ..... 1-28
*1.4.1 Primitivele PMS* ..... *1-29*
*1.4.2 Exemple* ..... *1-32*
1.5 CLASIFICAREA STRUCTURALĂ A CALCULATOARELOR ..... 1-47
*1.5.1 Sisteme cu paralelism funcţional şi de tip bandă de asamblare* ..... *1-49*
*1.5.2 Clasificarea masivelor de procesoare* ..... *1-49*
*1.5.3 Clasificarea sistemelor MIMD* ..... *1-50*

# **1.** INTRODUCERE

## OBIECTIVE

Primul capitol începe cu o scurtă introducere în universul prelucrării paralele văzut prin prisma unor aplicaţii mari consumatoare de putere de calcul. În continuare sunt abordate:

- Prezentarea celor mai răspândite clase de sisteme cu prelucrare paralelă (SPP);
- Clasificarea SPP după diferite criterii funcţionale;
- Modalităţi de descriere şi reprezentare a SPP.

## 1.1 Paralelismul - o nouă paradigmă în arhitectura calculatoarelor

Ştiinţa, cercetarea, ingineria sau arta au nevoie de sisteme de calcul cu performanţe care să permită rezolvarea problemelor specifice într-o manieră cât mai apropiată de realitate. De exemplu, pentru a studia cromodinamica cuantică - o teorie fundamentală a particulelor elementare, este necesară recalcularea repetată a peste 100 milioane de valori reprezentând distribuţia câmpului într-o regiune cu patru dimensiuni, spaţiu - timp în jurul protonului. În domeniul dinamicii fluidelor, pentru a modela adecvat structurile fine ale curgerii turbulente (caracterizate prin numere Reynolds foarte mari) este nevoie de putere de calcul foarte mare. Sunt multe aplicaţii cu constrângeri de timp real, ex: recunoaşterea şi înţelegerea în timp real a vorbirii, sisteme grafice pentru animaţie, sisteme de navigaţie etc. Pentru astfel de aplicaţii trebuie asigurate performanţele de viteză necesare, cu orice preţ. Trebuie remarcat că cererea de calculatoare din ce în ce mai performante este crescândă, în timp ce performanţele supercalculatoarelor convenţionale par să atingă limitele fizice fundamentale determinate de viteza luminii.

Chiar dacă dimensiunea unui calculator monoprocesor este foarte mică iar componentele sale transmit semnale cu viteza luminii, tot nu se pot obţine mai mult de câteva miliarde de operaţii pe secundă, ceea ce este insuficient pentru aplicaţii din domeniul fizicii, dinamicii fluidelor sau prelucrării în timp real a imaginilor şi a limbajului natural.

Pentru a obţine o creştere apreciabilă a capacităţii de prelucrare a unui sistem de calcul, trebuie depăşită gâtuirea secvenţială a calculatoarelor de tip Von Neumann. Soluţia constă în arhitecturi în care mai multe procesoare lucrează la aceeaşi problemă, în acelaşi timp. Este nevoie, deci, de prelucrare concurentă, sau mai exact, prelucrare paralelă. În aplicaţii de inteligenţă artificială (AI), prelucrarea paralelă este practic indispensabilă. Până în prezent, succesele AI în reprezentarea şi prelucrarea unor baze de cunoştinţe de mare capacitate sau în rezolvarea unor probleme de recunoaştere de forme sunt modeste. Creierul uman poate realiza aceste sarcini utilizând un mare număr de neuroni în paralel, fiecare fiind destul de lent. Aceasta sugerează necesitatea unor arhitecturi paralele de sisteme de calcul.

Desigur, se poate emite ideea că, teoretic, arhitecturile convenţionale sunt la fel de puternice ca şi cele paralele, în sensul că orice problemă poate fi rezolvată pe oricare din ele. Dar arhitectura sistemului utilizat influenţează în mod semnificativ modul în care se elaborează soluţia la problemă, respectiv programul care implementează soluţia. Multe aplicaţii sunt inerent paralele, ca şi fenomenele din realitate. În realitate, fenomenele apar în acelaşi loc în spaţiu, dar secvenţiate în timp, sau în acelaşi timp, dar în locuri diferite în spaţiu. Trebuie să reflectăm puţin asupra contrastului dintre natura concurentă a lumii şi natura secvenţială a calculatorului digital, convenţional. şi cum scopul principal al utilizării calculatorului este de a modela lumea, este evidentă nepotrivirea. Pentru a modela lumea, programatorii de calculatoare secvenţiale trebuie să găsească soluţii pentru a mima evenimentele concurente printr-o secvenţă de instrucţiuni. Pentru a modela în mod adecvat lumea reală, este de preferat utilizarea mai multor procesoare care operează simultan asupra aceluiaşi program. Acestea pot fi structurate conform clasei de aplicaţii pentru care au fost gândite. Posibilitatea de a alege arhitectura adecvată problemei face ca programarea acestora să fie uşoară şi naturală.

Desigur, trebuie remarcat şi faptul că, în special pentru cei care au crescut într-un mediu de programare serială, programarea eficientă a unor sisteme paralele poate să pară mai dificilă. Pentru a evita descurajarea se pot face referiri, de exemplu la lucrarea lui Tony Hoare "Communicating Sequential Processes" în Communications of the ACM, 1978 în care ideea

pricipală este că prelucrarea paralelă poate fi implementată de un număr de maşini von Neumann pur secvenţiale care comunică între ele printr-o reţea de canale cu autosincronizare. Această idee a stat la baza proiectării de către firma INMOS a familiei de transputere şi a limbajului paralel Occam. În ceea ce priveşte performanţele absolute ale procesoarelor, trebuie să constatăm evoluţia spectaculoasă din ultimii ani.

Dacă reprezentăm principalele caracteristici care determină viteza unui procesor într-un sistem de coordonate, aşa cum se arată în figura următoare, se poate utiliza ca metrică pentru măsurarea performanţelor de viteză, volumul paralelipipedului obţinut.

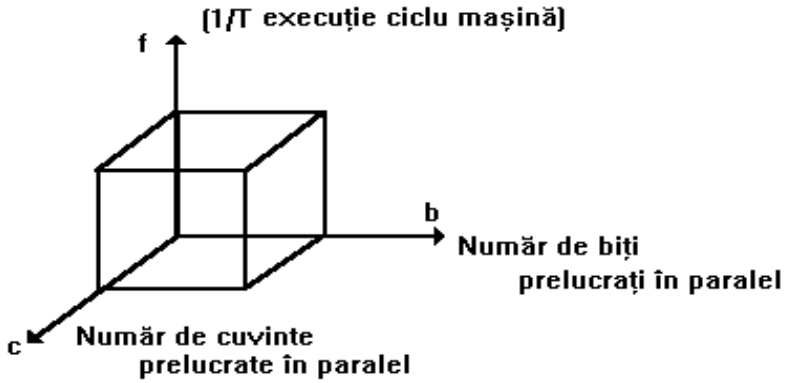



Figura 1.1 Volumul de date prelucrate

Sarcina proiectantului este de a "trage" de aceşti parametri, atât pe verticală (axa f), cât şi pe orizontală (axele b şi c).

Axa f. Reducerea timpului de execuţie a unui ciclu maşină se poate realiza în două moduri:

- Creşterea frecvenţei generatorului de tact până la apropierea de limitele fizice, în cadrul unei tehnologii date. În momentul scrierii acestui text există deja calculatoare personale care utilizează procesoare cu frecvenţa de 330 MHz. Este evident că această pistă duce la limitarea fizică dată de viteza de deplasare a electronilor.
- Reducerea numărului de tacte pe ciclu instrucţiune. Acest lucru este posibil prin optimizarea circulaţiei datelor prin unitatea de execuţie a procesorului. Astfel I8080 are 5 tacte/ciclu maşină, I8086 are 4 tacte/ciclu maşină, I80286 are 3 tacte/ciclu maşină, I80386 are 2 tacte/ciclu maşină iar I860 poate executa 3 instrucţiuni într-un tact. Aceasta s-a obţinut printr-o paralelizare a structurii interne după modelul "benzii de asamblare" (vezi capitolul II).

Axa b. Creşterea numărului de biţi prelucraţi în paralel conduce de asemenea la creşterea vitezei de prelucrare a unui procesor. Astfel, pentru a ne referi la acelaşi exemplu, lungimea cuvântului prelucrat la microprocesoarele Intel a evoluat astfel: 4 biţi la I4004, 8 biţi la I8080, 16 biţi la I8086 şi I80286, 32 biţi la I80386 şi I80486 şi 64 biţi la I860. Formatul intern al operanzilor prelucraţi de I8087 este de 80 biţi. Există procesoare asociative care prelucrează supercuvinte de lungime foarte mare. Cel puţin în acest moment, 64 biţi par să constituie o limită impusă de cerinţele aplicaţiilor.

Axa c. Numărul de cuvinte prelucrate în paralel exprimă gradul de multiplicitate a resurselor de prelucrare - (procesoare, unităţi de execuţie) şi este teoretic nelimitat. Există sisteme cu prelucrare paralelă cu zeci de mii de procesoare. Astfel, supercalculatorul

Connection Machine-2, este realizat din 65536 procesoare de 32 biţi şi poate atinge performanţa de vârf de 30Gflops (30 Giga FLoating point OPerations per Second).

In continuare se vor prezenta foarte pe scurt cîteva clase de sisteme de prelucrare paralelă.

### 1.1.1 Sisteme cu prelucrare în Bandă de Asamblare

O modalitate de efectuare în paralel a unor operaţii este dată de utilizarea principiului benzii de asamblare (BA -"pipeline") în arhitectura sistemelor de prelucrare a datelor. BA este aplicabilă dacă taskul de prelucrat poate fi descompus în subtaskuri astfel că ieşirile unor subtaskuri sunt intrări pentru altele. Să considerăm un compilator simplu care funcţionează în BA. Analiza lexicală, analiza sintactică, analiza semantică, optimizarea şi generarea codului sunt realizate de procese distincte. Programul sursă este analizat lexical şi elementele lexicale recunoscute sunt trimise ca intrări analizorului sintactic care construieşte intrările pentru analizorul semantic care la rândul lui produce un arbore pentru generatorul de cod. Generatorul de cod face o optimizare înainte de a produce codul obiect de ieşire, ca produs final al compilatorului. Trebuie notat că procesele care rezultă prin decompoziţie în vederea lucrului în BA sunt eterogene.

### 1.1.2 Procesoare paralele

Performanţele unui sistem cu procesoare paralele depind atât de arhitectura de bază a sistemului, cât şi de modul în care procesoarele sunt coordonate pentru a lucra în paralel la aceeaşi problemă. Într-o primă aproximaţie, un sistem cu procesoare paralele poate funcţiona astfel că toate procesoarele execută acceaşi instrucţiune, prelucrând date diferite sau fiecare procesor execută un flux propriu de instrucţiuni prelucrând de asemenea, date diferite de celelalte procesoare.

Un aspect important relativ la arhitecturile cu procesoare multiple este dat de relaţia procesoarelor cu memoria.

Astfel, în sistemele cu *memorie comună* toate procesoarele au acces la memoria comună prin legături directe realizate prin magistrale comune sau prin diferite reţele de interconectare. Astfel, fiecare procesor prelucrează date care sunt accesibile şi altor procesoare pentru citire sau modificare. Trebuie astfel luate precauţii pentru a evita accesul la date a unor procesoare înainte ca acestea să capete valorile adecvate, prin implementarea unor mecanisme de sincronizare, excludere mutuală etc.

În contrast cu modelul cu memorie comună există sisteme cu *memorie distribuită,* sau *memorie locală*. Astfel, fiecărui procesor i se ataşează o anumită cantitate de memorie proprie odată cu stabilirea unor legături între perechile procesor - memorie. Fiecare procesor are acces direct numai la memoria proprie. Modificarea datelor care nu se găsesc în memoria locală, se realizează prin transmiterea unor mesaje prin canale de comunicaţie care leagă nodurile (procesor - memorie) din sistem. Aceste noduri pot fi aranjate într-o topologie de tip reţea arbitrară.

### 1.1.3 Topologii de tip reţea

O schemă simplă de interconectare a elementelor unui sistem distribuit, cu procesoare multiple este topologia în inel în care fiecare procesor este conectat cu alte două procesoare,

adiacente. O altă schemă simplă de interconectare este o reţea rectangulară sub formă de grilă. Fiecare procesor este conectat cu patru procesoare, cei mai apropiaţi vecini. O limitare a acestor topologii constă în faptul că numărul de procesoare implicate în transmiterea unui mesaj de la un procesor oarecare la altul poate fi foarte mare.

Pentru a realiza un sistem multiprocesor cu memoria distribuită cu performanţe ridicate, reţeaua de interconectare trebuie să aibă următoarele caracteristici:

- să asigure comunicaţii rapide între noduri;
- să asigure conectivitate simplă pentru fiecare nod;
- să asigure flexibilitate în realizarea unei topologii de interconectare care se potriveşte geometriei aplicaţiei.

O clasă distinctă de sisteme de tip reţea de procesoare încearcă să mimeze funcţionarea creierului. A reapărut astfel, domeniul calculatoarelor neurale suportate cu mult entuziasm şi de tehnologie. "Calculul neural" are ca obiect studierea reţelelor de noduri adaptabile care, printr-un proces de învăţare prin experienţă, memorează cunoştinţele acumulate prin exemple pentru a fi apoi utilizate. Desigur trebuie notată distincţia dintre modul algoritmic şi modul experimental de rezolvare a problemelor. Întrebarea este cum pot fi organizate nodurile adaptabile pentru a memora cunoştinţele acumulate prin experienţă. Calculatoarele neurale sunt în general arhitecturi care conţin o secţiune de calcul convenţional care execută taskurile algoritmice şi controlează restul, o reţea neurală instruibilă care execută taskurile cognitive, dependente de cunoaştere şi o reţea neurală de senzori, care nu învaţă, dar care prelucrează semnale video şi audio pentru a le furniza reţelei cognitive în formatul dorit.

## 1.2 Clasificarea Sistemelor cu Prelucrare Paralelă (SPP)

Marea diversitate structurală a sistemelor de calcul a impus găsirea unor scheme adecvate de clasificare. O schemă de clasificare trebuie să posede următoarele caracteristici:

- să permită clasificarea tuturor structurilor de calcul, atât existente, cât şi previzibile în viitor;
- să diferenţieze structurile fundamental diferite;
- să clasifice în mod univoc oricare calculator sau structură de calcul.

Între schemele de clasificare mai cunoscute, în literatura de specialitate, se întâlnesc:

- schema propusă de M. Flynn (1966), bazată pe multiplicitatea fluxurilor de instrucţiuni şi de date într-un sistem de calcul;
- schema lui Tse-yun Feng ( 1972 ), bazată pe prelucrările serială, respectiv - paralelă a cuvintelor ce reprezintă informaţia;
- schema lui I. Shore ( 1973 ), care pleacă de la organizarea sistemului din diferite unităţi funcţionale;
- schema lui W. Handler ( 1977 ), care ia în consideraţie gradul de paralelism şi operarea în bandă de asamblare la diferitele niveluri ale subsistemelor componente;
- schema propusă de R. Hockney şi C. Jesshope (1981) tratează sistemul ca o colecţie de unităţi funcţionale, care manipulează date, conectate prin magistrale, şi care operează sub controlul unităţii de instrucţiuni.

### 1.2.1 Schema propusă de M. Flynn

Procesul de calcul constă în execuţia unei secvenţe de instrucţiuni asupra unui set de date. Termenul de flux specifică o secvenţă de instrucţiuni sau date manipulate de un singur procesor.

Un flux de instrucţiuni este o secvenţă de instrucţiuni executată de maşina fizică iar un flux de date este o secvenţă de date incluzând datele iniţiale, rezultatele parţiale/temporare şi rezultatele finale.

Organizarea calculatorului este caracterizată prin multiplicitatea hardware-lui destinat manipulării fluxurilor de instrucţiuni şi date.

Pentru ilustrarea conceptelor sunt necesare numai trei tipuri de unităţi: unitatea de comandă (UCd), unitatea de execuţie (UE) şi unitatea de memorie (UM).

Instrucţiunile (FI) şi datele (FD) sunt citite din UM, organizată eventual sub forma de module distincte, partajate de unităţile de execuţie. Instrucţiunile sunt decodificate de UCd, care trimite un flux decodificat de (FC) comenzi spre UE.

Fluxul de date între UE şi UM este bidirecţional. Fiecare flux de instrucţiuni este controlat de o UCd independentă. Fluxurile multiple de date î_i au originea în modulele multiple ale subsistemului de memorie.

**1°. Flux Singular de Instrucţiuni - flux Singular de Date (SISD)**

Reprezintă organizarea secvenţială obişnuită a calculatoarelor convenţionale, calculatoarele de tip von Neumann, cum mai sunt denumite. Instrucţiunile sunt executate strict secvenţial, sau în regim de suprapuneri în etajele unităţii de execuţie organizate pe principiul benzii de asamblare. Cele mai multe sisteme SISD monoprocesor moderne folosesc banda de asamblare. Un sistem SISD poate avea mai multe unităţi de execuţie; toate unităţile de execuţie se află sub controlul unei singure unităţi de comandă (Figura 1.2).

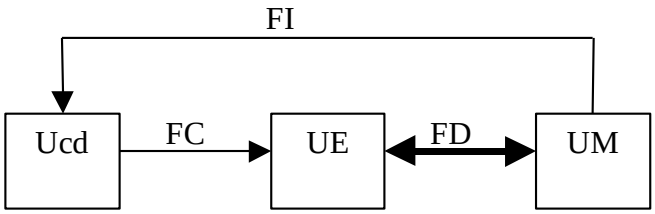



Figura 1.2 Schema bloc a unei structuri SISD

**2°. Flux Singular de Instrucţiuni - flux Multiplu de Date (SIMD)**

Această organizare corespunde procesoarelor vectoriale sau matriceale în care mai multe unităţi de execuţie operează sub controlul unei singure unităţi de comandă. Pe baza aceluiaşi flux de instrucţiuni, unităţile de execuţie manipulează fluxuri distincte de date în conjuncţie cu module distincte de memorie (Figura 1.3). Sistemele SIMD, după formatul datelor manipulate în unităţile de execuţie, se împart în sisteme cu prelucrare pe felii de cuvinte şi sisteme cu prelucrare pe felii de biţi.

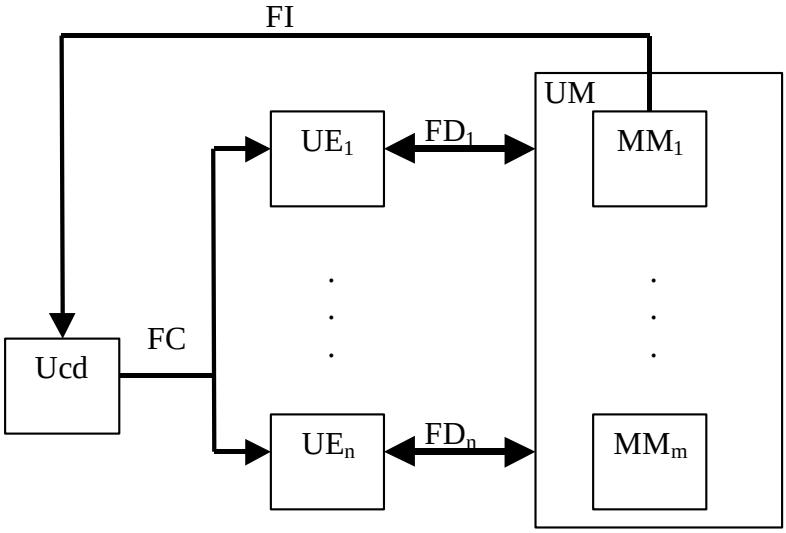



Figura 1.3. Schema bloc a unei structuri SIMD

**3°. Flux Multiplu de Instrucţiuni - flux Singular de Date (MISD)**

Unităţile de execuţie multiple primesc fluxuri diferite de instrucţiuni pentru a prelucra acelaşi flux de date, eventual aflate în stadii diferite de prelucrare. Astfel, într-o macrobandă de asamblare, rezultatul obţinut de la o UE este transferat la intrarea următoarei UE.

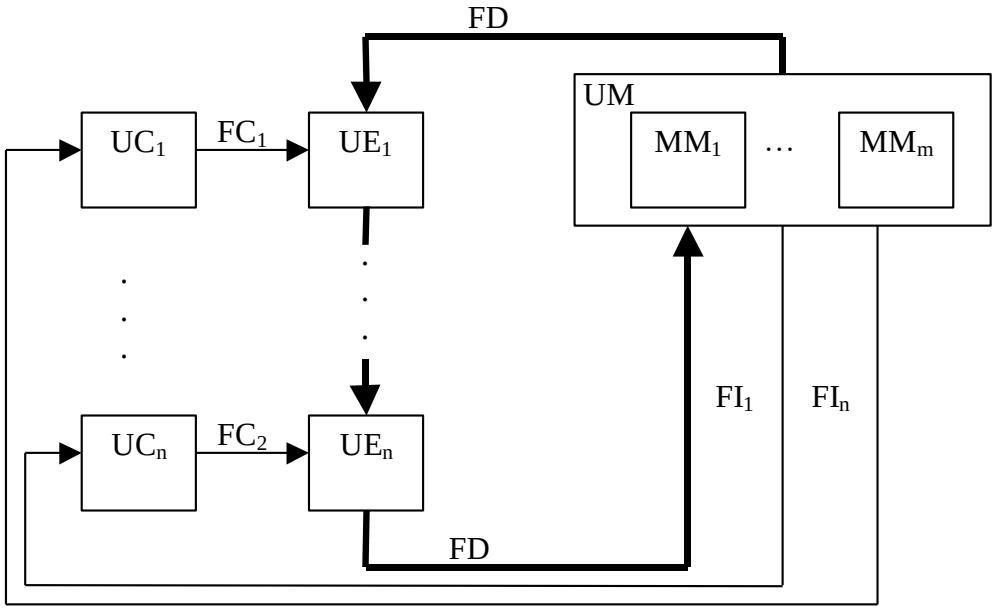



Figura 1.4. Schema bloc a unei structuri MISD

**4°. Flux Multiplu de Instrucţiuni - Flux Multiplu de Date (MIMD)**

Această organizare implică existenţa unei legături şi interacţiuni între unităţile de execuţie şi unităţile de comandă. Fluxul de date este derivat din acelaşi spaţiu partajat de memorie de către toate UE. Dacă memoria ar fi constituită din module distincte/independente, atunci s-ar obţine un set de n sisteme SISD independente.

Un sistem MIMD intrinsec este *strâns cuplat* dacă gradul de interacţiune între

procesoare este înalt, şi respectiv - *slab cuplat* dacă gradul de interacţiune este redus.

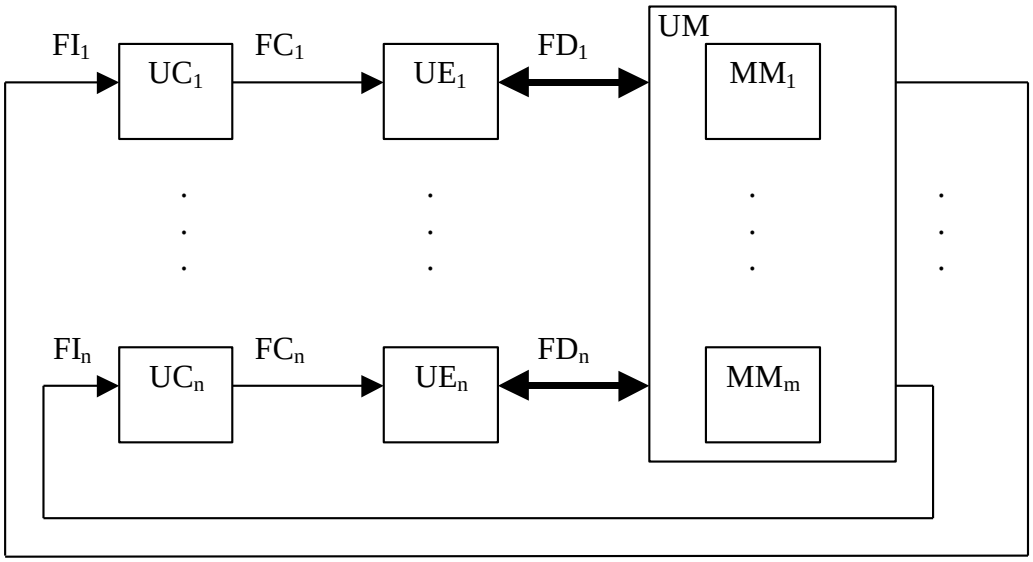



Figura 1.5. Schema bloc a unei structuri MIMD

Tabelul1.1 Exemple de calculatoare clasificate conform schemei lui M.Flynn.

| Clasa de calculatoare | Sisteme de calcul. |
|---|---|
| SISD (cu o singură unitate funcţională) | IBM 701, IBM 1620, IBM 7090, PDP VAX 11/ |
| SISD (cu mai multe unităţi funcţionale) | IBM 360/91, IBM 370/168, CDC 6600, CDC Star-100, TI-ASC, FPS-AP-120B, FPS-164, IBM 3838, Cray-1, CDC Cyber-205, Fujitsu VP-200, CDC-NASF Fujitsu-Facom. |
| SIMD (operare cu felii de cuvinte). | ILLIAC IV, PEPE, Burroughs- BSP. |
| SIMD (operare cu felii de biţi). | STARAN, MPP, ICL 2900-DAP. |
| MIMD (cuplate strâns). | IBM 370/168 MP, UNIVAC 1100/80 Tandem 1 IBM 3081/3084, Cm* |
| MIMD (cuplate slab). | Burroughs D-825, Cmmp, Cray-2 S-1, Cray-X M Denelcor. |

Schema de clasificare a lui Flynn utilizează un criteriu de clasificare foarte clar şi are avantajul că tratează unitar calculatoarele tip von Neumann şi sistemele cu prelucrare paralelă. Totuşi există unele limite legate de încadrarea în aceeaşi schemă a structurilor cu prelucrare paralelă de tip bandă de asamblare şi de faptul că nu există exemple practice de structuri MISD.

### 1.2.2 Schema de clasificare a lui Feng

În schema de clasificare a lui Tse-yun Feng se ia ca bază de departajare a diferitelor structuri gradul de paralelism exprimat prin numărul maxim de biţi prelucraţi de calculator în unitatea de timp. Gradul maxim de paralelism se notează cu P. Dacă $P_i$ reprezintă numărul de biţi prelucraţi în ciclul i al procesorului (sau în perioada i a ceasului), în condiţiile a s cicluri (i=1,2,...,s), rezultă un grad de paralelism mediu $P_m$ egal cu:

$$P_m = \frac{\sum_{i=1}^{s} P_i}{s} \qquad (1.1)$$

În general $P_m < P$. Rata de utilizare a calculatorului pe durata s cicluri va fi dată de expresia:

$$u = \frac{P_m}{P} = \frac{\sum_{i=1}^{s} P_i}{s.P} \qquad (1.2)$$

Diferitele tipuri de calculatoare se pot reprezenta în cadrul unui sistem de coordonate nxm, unde abscisa este dată de lungimea în biţi (n) a cuvântului prelucrat, iar ordonata de lungimea feliei de biţi (m ) prelucraţi. O tranşă de biţi reprezintă un şir de biţi, câte unul din fiecare cuvânt, la aceeaşi poziţie verticală de bit. De exemplu, TI - ASC are lungimea cuvântului n = 64 şi 4 benzi de asamblare pentru prelucrări aritmetice; fiecare bandă este constituită din 8 tronsoane. Astfel, sunt 8x4 = 32 biţi, pe fiecare felie de biţi, în 4 benzi de asamblare. Rezultă că sistemul TI - ASC se reprezintă, în figura 1.6, prin perechea: (nxm) = (64x32).

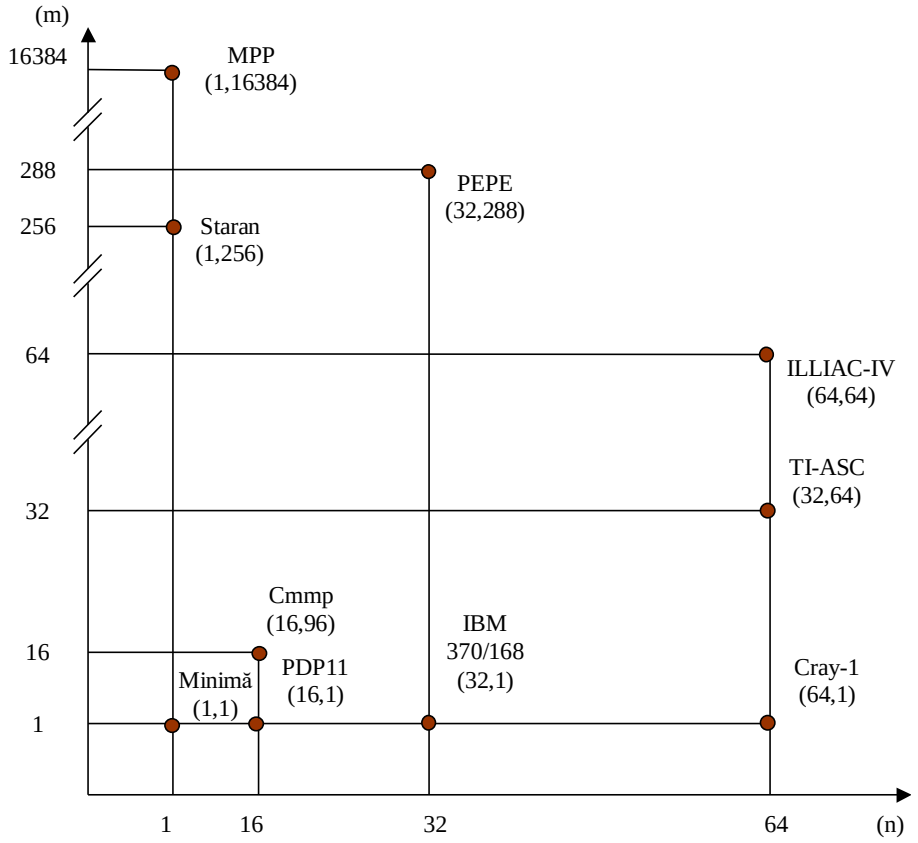



Figura 1.6. Reprezentarea clasificării lui Feng

Paralelismul maxim al unui sistem de calcul C este dat de expresia:

$$P(C) = n \times m \qquad (1.3)$$

ceea ce corespunde ariei dreptunghiului de laturi n, m.

Din cele arătate mai sus, rezultă existenţa a patru metode de prelucrare:

- Cuvânt-Serial şi Bit-Serial ( CSBS );
- Cuvânt-Paralel şi Bit-Serial ( CPBS );
- Cuvânt-Serial şi Bit-Paralel ( CSBP );
- Cuvânt-Paralel şi Bit-Paralel ( CPBS ).

Prelucrarea lentă, serială la nivel de bit, corespunde metodei CSBS (n=1,m=1). Aceasta a fost folosită în prima generaţie de calculatoare.

Prelucrarea pe grupuri de biţi este bazată pe metoda CPBS, care asigură tratarea simultană a unei felii de m biţi (n=1, m>1). Ca exemple se pot da sistemele: Staran (1,256), MPP (1,16384), ICL 2900 - DAP (1,406) etc.

Prelucrarea pe grupuri de cuvinte, conform metodei CSBP (n>1, m=1) este frecvent întâlnită în calculatoarele curente, dintre care se pot menţiona: IBM370/168 MP (64,1), CDC 6600 (60,1), Burroughs 7700 (48,1), VAX 11/780 (32,1) etc.

Prelucrarea complet paralelă, în baza metodei CPBP (n>1,m>1) permite tratarea simultană a n x m biţi de informaţie şi are ca rezultat o viteză mare de calcul. Dintre sistemele de calcul din această categorie se pot evidenţia: ILLIAC IV (64,64), TI-ASC (64,32), Cmmp (16,16), S-1 (36,16) sa.

### 1.2.3 Schema de clasificare a lui I. Shore

Această clasificare se bazează pe modul de organizare a sistemului, din unităţi funcţionale diferite: Unitatea de Comandă (UCd); Unitatea de Memorie (UM), constituită din Modulul de Memorie pentru Instrucţiuni (MMI) şi Modulul de Memorie pentru Date (MMD); Unitatea de Execuţie, care poate fi organizată pentru prelucrări pe grupuri de cuvinte (Unitate de Execuţie Orizontală - UEO) sau pentru prelucrări pe grupuri de biţi (Unitate de Execuţie Verticală - UEV). MMD poate fi adresată la nivel de Cuvânt, felie de cuvânt, (MMDC) sau la nivel de felie de Biţi (MMDB).

Au fost identificate şase tipuri de structuri, notate de la I, la VI.

**1°. Structura I** corespunde maşinii convenţionale von Neumann (Figura 1.7). O singură citire din MMD furnizează toţi biţii unui cuvânt, care sunt prelucraţi în paralel de către UEO. UE poate conţine unităţi funcţionale multiple, eventual bazate pe principiul Benzii de Asamblare (BA). În acest grup sunt incluse atât calculatoarele cu BA pentru scalari (CDC 7600), cât şi cele cu BA pentru vectori (Cray-1).

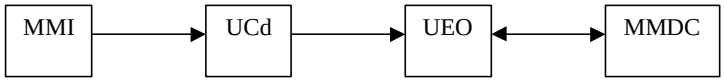



Figura 1.7. Structuri de tipul I

**2°. Structura II** este similară cu structura I, cu excepţia că o citire din MMD furnizează o felie de biţi, din toate cuvintele memoriei, în locul tuturor biţilor unui cuvânt. UE asigură operaţii seriale la nivel de bit. Dacă memoria este văzută ca un tablou bidimensional de biţi, cu un cuvânt memorat pe linie, structura II citeşte o felie verticală de biţi, în timp ce structura I citeşte o felie orizontală, un cuvânt (Figura 1.8). Ca exemple se pot da sistemele ICL 2900-DAP şi STARAN.

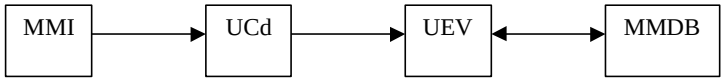


Figura 1.8. Structura de tip II

**3°. Structura III** reprezintă o combinaţie a structurilor I şi II. Ea este axată pe o memorie de date bidimensională, care poate fi adresată, fie la nivel de cuvânt, fie la nivel de grupuri de biţi şi pe două unităţi de execuţie de tip UEO şi UEV (Figura 1.9).

Acesta este calculatorul ortogonal descris de Shooman, în 1970. Seria de maşini OMEN-60, propusă de firma Sanders Associates, constituie o implementare a structurii III, după cum a fost definită de Higbie, în 1972.

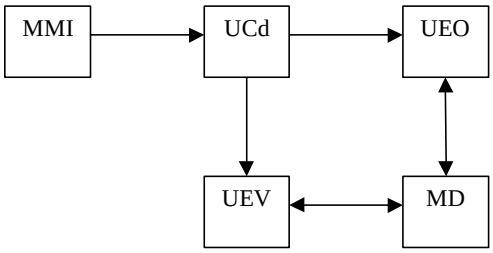


Figura 1.9. Structura de tip III

**4°. Structura IV** se obţine prin multiplicarea UE şi MMD din structura I. Unităţile de execuţie UE1,..,UEn operează sub controlul singurei UCd din sistem. Nu există comunicaţii directe între unităţile de execuţie (Figura 1.10). Comunicaţiile se realizează prin intermediul UCd. Absenţa comunicaţiilor directe între unităţile de execuţie limitează aplicabilitatea sistemului, dar permite extinderea lui prin adăugarea de noi UE. Ca exemplu se poate da sistemul PEPE.

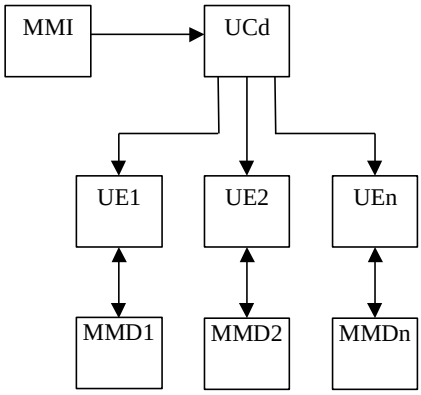


Figura 1.10 Structura de tip IV

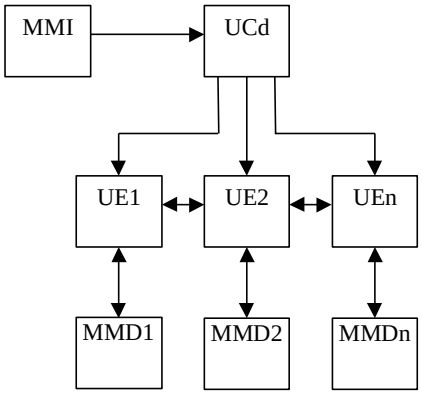


Figura 1.11 Structura de tip V

**5°. Structura V** porneşte de la structura IV în care s-au prevăzut facilităţi pentru comunicaţii de tip "vecinul cel mai apropiat" între unităţile de execuţie. Unităţile de execuţie sunt plasate în linie. Ele pot manipula date, atât din memoria proprie, cât şi din memoriile asociate unităţilor de execuţie vecine (Figura 1.11).

**6°. Structura VI**, spre deosebire de structurile I,...,V care folosesc MMD şi UE distincte, conectate prin magistrale, constituie o abordare deosebită prin distribuirea logicii unităţii de execuţie în cadrul memoriei (Figura 1.12). Ca exemple se pot da memoriile

asociative simple sau procesoarele asociative complexe.

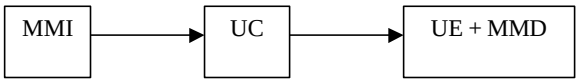



Figura 1.12. Structura de tip VI

Cu excepţia calculatorului vectorial cu BA, structurile II...V reprezintă cazuri particulare ale categoriei SIMD, din clasificarea lui Flynn. Structura I corespunde clasei SISD. Calculatorul cu BA nu este departajat satisfăcător de calculatoarele scalare fără BA sau fără paralelism intern.

### 1.2.4 Schema de clasificare a lui Handler

Această clasificare se bazează pe următoarele idei:

- pentru caracterizarea structurilor de calcul fundamentale se folosesc trei parametri semnificativi, simpli şi foarte sugestivi: număr UCd, număr UE şi lungimea cuvântului;
- clasificarea utilizează operaţiile +, * şi U, care permit formarea unor noi structuri, de complexitate sporită.

În contextul acestei clasificări se impune, pentru început, examinarea unor structuri paralele de tip multiprocesor, cu unităţi de execuţie multiple şi structuri de prelucrare bazate pe principiul benzii de asamblare.

Prezenţa mai multor procesoare (UCd + UE) într-un sistem de calcul permite realizarea unui sistem care poate executa simultan mai multe programe (Figura 1.13). De asemenea, existenţa unor unităţi de execuţie multiple, operând sub controlul aceleiaşi unităţi de comandă, oferă posibilitatea execuţiei aceluiaşi program pe seturi diferite de date ( Figura 1.13 ).

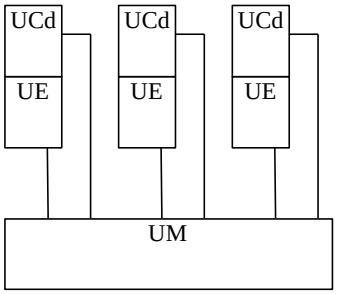



Figura 1.13

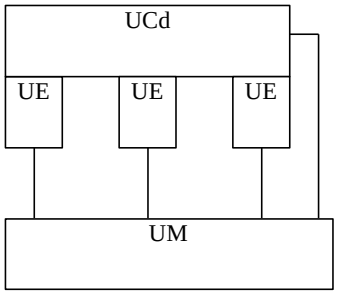



Figura 1.14.

O altă formă de paralelism se întâlneşte în structurile bazate pe principiul benzii de asamblare, care presupune existenţa mai multor procesoare sau unităţi de execuţie, conectate în cascadă şi operând în paralel. Astfel, în procesul de prelucrare, în faze diferite, se află un flux de date continuu. Pe parcursul întregului proces, asupra unei mulţimi de date, se efectuează succesiv n activităţi, unde n este numărul de staţii (tronsoane) din banda de asamblare.

Banda de asamblare poate fi organizată la nivelurile:

- programului (macrobanda de asamblare - Figura 1.15),
- instrucţiunilor (banda de asamblare pentru instrucţiuni - Figura 1.16)
- cuvintelor (banda de asamblare pentru operaţii aritmetice - Figura 1.17).

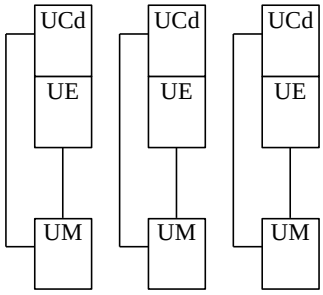



Figura 1.15 BA la nivel de program


Figura 1.16 BA la nivel de instrucţiuni

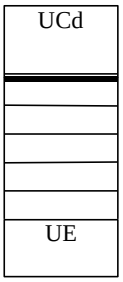



Figura 1.17. Ba la nivel de microoperaţii

Revenind la clasificarea structurilor, indiferent de cele trei tipuri de benzi de asamblare sau de prezenţa procesoarelor/unităţilor de execuţie multiple, fiecărei structuri i se poate ataşa un triplet:

$t = < k, d, w >$

unde:

- k este numărul unităţilor de comandă, care interpretează programul;
- d reprezintă numărul unităţilor de execuţie comandate de fiecare din cele k unităţi de comandă;
- w este lungimea cuvântului maşină sau numărul de ranguri prelucrate de fiecare din cele d unităţi de execuţie.

Cea mai simplă structură, Minimă, se caracterizează prin tripletul:

$t_{Minimă} = < 1, 1, 1 >$.

Structurile von Neumann au asociate triplete de forma:
$t = < 1, 1, 32 >$ ; $t = < 1, 1, 36 >$ ; $t = < 1, 1, 48 >$

Clasificarea de mai sus este consistentă în condiţiile în care structura nu conţine sisteme de comandă autonome, pentru I/E, care se pot programa independent.

În structurile tripletelor de bază pot intra diferite tipuri de benzi de asamblare. Banda de asamblare se caracterizează printr-un triplet extins:

$t = < k*k`, d*d', w*w' >$,

unde k, d, w au fost definite anterior,

- k' este numărul de UCd, care interpretează programul dat, în timp ce datele le parcurg succesiv (lungimea macrobenzii de asamblare);
- d' reprezintă numărul UE comandate de o singură UCd, care prelucrează un flux de date (lungimea benzii de asamblare pentru instrucţiuni );
- w' constituie numărul nivelurilor sau segmentelor benzii de asamblare aritmetice.

În continuare se vor da exemple de triplete von Neumann având în componenţa lor

benzi de asamblare:

- structura sistemului CDC 7600 - PC (Procesorul Central):

$t_{CDC\text{-}7600\text{-}PC}$ = < 1*1, 1*9, 60*1 > = < 1, *9, 60 >

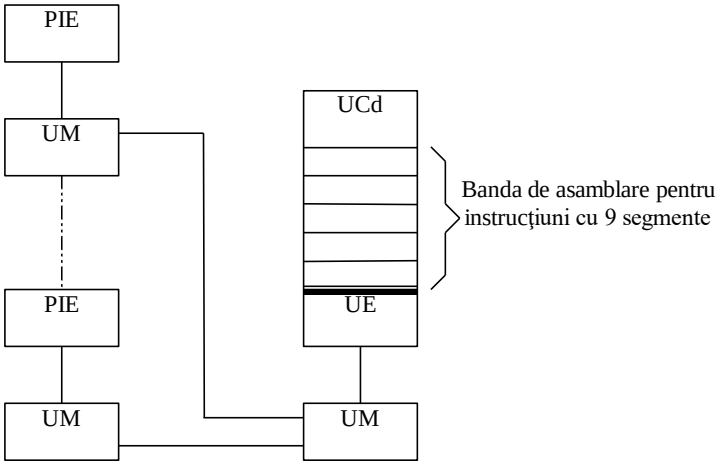



Figura 1.18. Structura lui CDC7600

Structura CDC 7600 (redenumit CYBER 70 model 76) este prevăzută cu un procesor central cu o unitate de comandă în bandă de asamblare cu 9 segmente şi cu o unitate de execuţie care prelucrează cuvinte de 60 de biţi. Calculatoarele periferice operează cu cuvinte de 12 biţi, care, asamblate în grupuri de câte 5, formează cuvinte de 60 de biţi compatibile cu formatul din unitatea de execuţie.

- structura CDC STAR ( Procesorul Central ):

$t_{CDC\text{-}STAR\text{-}100}$ = < 1,2,64*4 >

Configuraţia CDC-STAR-100 (Figura 1.19) constă din două unităţi de execuţie care operează cu cuvinte de 64 de biţi, în cadrul a două benzi de asamblare cu câte 4 segmente.

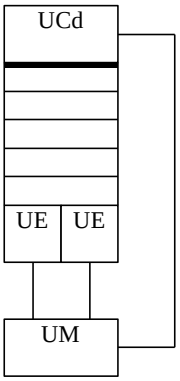



Figura 1.19 Structura lui CDC-STAR-100

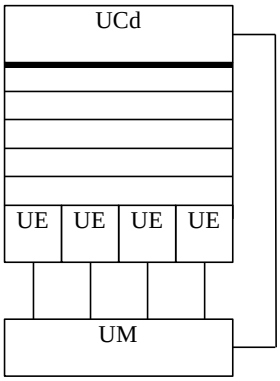



Figura 1.20.Structura lui TI-ASC-PC

- structura TI-ASC-PC ( Procesor Central ):

$t_{TI\text{-}ASC\text{-}PC}$ = < 1,4,64*8 >

Configuraţia TI-ASC-PC ( Figura 1.20 ) constă din patru unităţi de execuţie care

operează cu cuvinte de 64 de biţi, în cadrul a patru benzi de asamblare cu 8 segmente. Pentru a putea caracteriza noi structuri, cu complexitate sporită, trebuie introduse unele reguli de compoziţie. Tripletele de bază specificate mai sus permit descrierea sistemelor care au, fie mai multe unităţi de comandă identice (fiecare putând executa un program diferit), fie mai multe unităţi de execuţie împreună cu unităţile lor de comandă (fiecare putând prelucra un flux diferit de date).

Se introduc următorii operatori de compoziţie:

**+ (suma)**, pentru sistemele paralele care conţin două sau mai multe structuri diferite, specializate pe diferite prelucrări de date de natură concretă;

*** (produsul)**, pentru sistemele de tip bandă de asamblare (macrobandă) în care datele sunt prelucrate de toate structurile, fiecare structură fiind descrisă de către un triplet corespunzător;

$\vee$ **(reuniunea)**, pentru sistemele în care acelaşi hardware poate fi folosit în regimuri diferite de lucru.

Structurile neomogene obţinute pe baza operatorului "+" se întâlnesc mai rar, în timp ce structurile descrise cu operatorul "*" sunt mai frecvente.

Dacă pentru CDC 7600 se consideră şi cele 15 calculatoare periferice, se va obţine tripletul:

$$t_{CDC\text{-}7600} = < 15,1,12 > * < 1,*9,60 >.$$

Primul termen corespunde unui calculator de I/E, care comandă fluxul de lucrări şi de date. Al doilea termen defineşte calculatorul central.

Prin elasticitatea structurilor de calcul se înţelege multifuncţionalitatea, posibilitatea de folosire a aceluiaşi hardware în regimuri diferite de lucru. Elasticitatea calculatorului convenţional se consideră egală cu unitatea: un cuvânt maşină poate fi interpretat ca instrucţiune sau ca dată.

În sistemul ICL 2900-DAP informaţia poate fi interpretată în două moduri diferite:

$$t_{DAP} = < 1,128,32 > Ï < 1,4096,1 >.$$

Astfel, DAP poate fi configurat sub forma a 128 unităţi de execuţie, care operează cu cuvinte de 32 de biţi, sau sub forma a 4096 unităţi de execuţie care manipulează cuvinte de 1 bit.

În clasificarea efectuată nu s-a putut prinde caracteristica ICL DAP de a putea fi folosit ca bloc de memorie pentru sistemul 2900.

Sistemul ILLIAC IV a avut o configuraţie constând din două calculatoare PDP-10 şi o reţea de 64 procesoare, care putea fi folosită, fie pentru prelucrări pe 64 de biţi, fie pentru prelucrări pe 128 de biţi:

$$t_{PDP10\text{-}ILLIAC\text{-}IV} = < 2,1,16 > * [< 1,64,64 > Ï < 1,128,32 >]$$

În proiectul sistemului OMEN 60 au fost prevăzute două unităţi de execuţie, care se folosesc alternativ, şi o singură unitate de comandă, care interpretează un singur program. În funcţie de necesităţile programului se foloseşte regimul obişnuit sau cel asociativ:

$$t_{PDP11\text{-}OMEN60} = < 1,1,16 > + < 0,64,1 >.$$

Primul termen descrie sistemul PDP 11, care interpretează programul, în care pot exista instrucţiuni pentru reţeaua de 64 unităţi de execuţie pe un bit (cel de-al doilea termen), pentru care PDP 11 generează semnalele de comandă corespunzătoare. În acest sistem se găsesc două unităţi de execuţie: orizontală, în PDP 11, şi verticală, în OMEN 60, ambele fiind controlate de o unitate comună de comandă:

$t_{PDP11\text{-}OMEN60} = < 1,0,0 > * [< 0,1,16 > + < 0,64,1 >]$.

În final se prezintă tripletele unor maşini "istorice" de calcul

$t_{abac} = < 0,0,n >$ - unde n este numărul elementelor din registru;

$t_{calculator\ de\ birou} = < 0,1,n >$ - unde 1 arată existenţa unei unităţi de prelucrare;

$t_{von\ Neumann} = < 1,1,w >$ - unde w este lungimea în biţi a cuvântului maşină.

## 1.3 Reprezentarea structurală a sistemelor cu prelucrare paralelă

### 1.3.1 Generalităţi

Sistemele de calcul pot fi privite ca un ansamblu de unităţi funcţionale conectate printr-o schemă de interconectare dată.

Aceste sisteme prelucrează un mediu numit informaţie, care poate fi măsurat în biţi, ranguri zecimale, caractere etc. Sub acest aspect, componentele sistemului, pe lângă funcţiile specifice, se mai pot caracteriza, de exemplu, prin rata de transfer, capacitatea de stocare a informaţiei. Aceste caracteristici prezintă importanţă pentru studiul sistemelor de calcul cu ajutorul metodelor cercetării operaţionale, în numeroase aplicaţii (multiprogramare, multiprelucrare, timp divizat, reţele de calculatoare etc.) în care datorită concurenţei intervin probleme legate de congestii, cozi de a_teptare, zone tampon, debite de transfer de date etc.

Componentele primitive se caracterizează printr-o serie de atribute, iar atributele prin valori.

Reprezentarea structurală trebuie să evidenţieze modalităţile specifice de interconectare ale componentelor primitive. Astfel, un calculator C este reprezentat sub forma unei colecăii de unităţi funcţionale interconectate care prelucrează date sub controlul unei unităţi de instrucţiuni I.

Un calculator C, serial de tip von Neumann, va fi reprezentat astfel:

$$C = I [ E - M ]$$

în care componentele au următoarele semnificaţii:

- I unitate de instrucţiuni, singulară, care controlează unităţile plasate între parantezele drepte;
- E unitate aritmetică de execuţie, singulară, conectată printr-o legătură tip magistrală cu unitatea de memorie;
- M unitate de memorie, singulară, fără organizare pe blocuri;

- '-' conexiunea de tip magistrală (linie continuă între unităţile E şi M).

Asemenea notaţii [1,2] trebuie să aibă un caracter concis, să se poată tipări usor, să poată fi introduse într-un program pe calculator etc.

Cu ajutorul lor trebuie să se obţină reprezentări clare ale unor structuri relativ complexe (reţele/tablouri de procesoare cu dimensiuni variabile, unităţi de execuţie interconectate sub forma matricială etc.).

În continuare vor fi prezentate elementele limbajului de descriere:

- simbolurile pentru unităţile funcţionale ,
- simbolurile pentru conexiuni/legături,
- comentariile,
- comanda unităţilor etc.

### 1.3.2 Unităţile funcţionale şi regulile pentru reprezentarea organizării lor

**1°. Simbolurile folosite pentru reprezentarea unităţilor**

Simbolurile vor fi reprezentate printr-o singură literă şi sunt date în continuare în ordine alfabetică:

- **B** unitate de execuţie pentru întregi, virgulă fixă sau operanzi/vectori booleeni;
- **C** calculator alcătuit dintr-o combinaţie oarecare de unităţi funcţionale, inclusiv o unitate de instrucţiuni I;
- **Ch** canal de I/E care poate transfera date între o interfaţă de echipament de I/E şi memorie, independent de alte unităţi;
- **D** echipament de I/E a cărui natură este specificată într-un comentariu asociat;
- **E** unitate de execuţie care prelucrează date, prin efectuarea unor operaţii aritmetice, logice şi manipulări de biţi pe diverse fluxuri de date; adesea este împărţită în unităţi B şi F, fiind numită UAL (Unitate Aritmetică Logică);
- **F** unitate de execuţie în virgulă mobilă;
- **H** legătură de date de tip magistrală; transferă datele fără prelucrări (în unele cazuri au loc însă reordonări ale rangurilor, permutări);
- **I** unitate de instrucţiuni care decodifică şi interpretează un singur flux de instrucţiuni şi trimite comenzi spre unităţile de execuţie; I controlează secvenţierea instrucţiunilor cu ajutorul unui singur contor de instrucţiuni (mai poartă numele IPU = Instruction Processing Unit);
- **IE** interfaţă pentru un echipament de I/E care preia datele de la echipament şi le încarcă într-un registru local sau invers;
- **M** unitate de memorie unidimensională, în care sunt memorate datele şi instrucţiunile (ex.: registre, zonă de memorie tampon, memoria principală, disc etc. );
- **O** memorie bidimensională-ortogonală care poate fi accesată pe linii, sau pe coloane;
- **P** procesor (element de prelucrare), definit ca o colecţie de unităţi, incluzând cel puţin o unitate de tip E, fără a implica o unitate de tip I;
- **U** unitate nespecificată, simbol folosit când nu se poate utiliza nici una din notaţiile de mai sus; simbolul U va fi urmat de un comentariu; ca exemple se pot

da unităţile intermediare de control în cadrul unui sistem complex.

**2°. Operarea în bandă de asamblare**

Operarea în bandă de asamblare (BA) este indicată prin sufixul 'p' ("pipeline") asociat simbolului unităţii sau structurii cuprinse între acolade. Această notaţie indică suprapunerea în timp a operaţiilor executate de unităţile sau structurile respective.

Exemple:
Ip - unitate de instrucţiuni de tip BA;
Ep - unitate de execuţie de tip BA;
{ E ←,→ M }p - structură formată din unitate de execuţie şi memorie cu operaţii aritmetice şi de citire/scriere concurente.

**3°. Instrucţiuni pentru prelucrarea vectorilor**

În cazul în care setul de instrucţiuni al sistemului prevede operaţii cu vectori prin instrucţiuni specifice se va folosi sufixul 'v' în simbolurile I, E etc.; în cazul tablourilor de procesoare, când toate operaţiile se execută cu vectori simbolul este omis, deoarece este subânţeles.

Exemple:
Ipv - unitate de instrucţiuni de tip BA, cu instrucţiuni pentru prelucrarea vectorilor;
Iv - unitate de instrucţiuni, cu instrucţiuni pentru prelucrarea vectorilor .

**4°. Unităţi diferite**

Dacă există unităţi diferite de acelaşi tip, acestea pot fi "indexate" printr-un întreg care urmează simbolului ce defineşte tipul unităţii, de exemplu:
Ev1 - unitatea de execuţie cu prelucrare de vectori numărul 1;
Ev2 - unitatea de execuţie cu prelucrare de vectori numărul 2.

**5°. Substituţia**

În scopul obţinerii unei notaţii concise şi clare, ierarhizate, unităţi diferite cum sunt cele prezentate în definiţia de la 4) pot fi descrise separat prin ecuaţii despărţite de ';'.

Exemplu:
I[E1,E2]; E1= ; E2= ;

aici delimitatorul ',' indică faptul că E1 şi E2 pot opera simultan.

**6°. Unităţi multiple**

Prin folosirea unui întreg ca prefix se poate indica numărul de unităţi de acelaşi tip care operează simultan.

O unitate universală oricât de complexă, constituită din mai multe componente funcţionale care însă nu funcţionează simultan, este considerată a fi unică. De exemplu, în sistemul IBM 7090, există o singură unitate de execuţie E, multifuncţională, pentru adunare înmulţire, operaţii logice etc., care efectuează o singură operaţie la un moment dat; în sistemul CDC 6600 sunt 10 unităţi de execuţie (10E) independente pentru adunare, înmulţire, operaţii

logice etc., care pot funcţiona simultan.

**7°. Specificarea unităţilor identice**

Prin sublinierea unui simbol sau a unui grup de simboli delimitaţi de paranteze acolade, se specifică faptul că aceste unităţi sau grupuri de unităţi sunt identice.
Exemplu:

64P=64{E-M}.

reprezintă sistemul ILLIAC IV cu 64 elemente de prelucrare identice.

**8°. Gruparea unităţilor**

Grupările de unităţi pot fi definite prin plasarea separatorilor ',' şi '/' între unităţi şi includerea lor între acolade. Separatorul ',' specifică operarea concurentă (simultană) a unităţilor, în timp ce separatorul '/' indică operarea secvenţială.

Exemple:

{3Fp,2B} - specifică trei unităţi de execuţie în VM, de tip BA care operează concurent cu două unităţi de execuţie care manipulează întregi.

{E1/E2/E3/E4}- specifică 4 unităţi de execuţie care operează secvenţial (una la un moment dat).

În cadrul unei perechi de acolade se foloseşte numai un singur tip de separator, deşi o unitate multiplă (nF) poate executa operaţii simultane.

{2F1/B1} - specifică 2 unităţi VM, care pot opera simultan între ele, dar secvenţial în raport cu unitatea pentru întregi B1.

2F1={F(+),F(*)} - unităţi de adunare şi de înmulţire în VM, care pot opera simultan.

B1={B(+)/B(shift)} - unitate în virgulă fixă constituită dintr-un sumator şi un circuit de deplasare, care pot opera numai secvenţial.

**9°. Specificarea numărului de biţi prelucraţi în paralel**

Aceasta se realizează printr-un indice inferior asociat unităţii.
Exemple:

$I_{32}$ - unitate de instrucţiuni pe 32 de biţi;

$E_{16}$ - unitate de execuţie pe 16 biţi.

Pentru a descrie unităţile de memorie se observă că fiecare secţiune sau bloc de memorie, care operează independent este tratat ca o unitate separată.

Asteriscul '*' este folosit ca simbol pentru înmulţire.
Exemple:

$nM_{w*b}$ - specifică o memorie unidimensională împărţită în n blocuri; fiecare bloc conţine w cuvinte de câte b biţi, iar când este adresat furnizează b biţi în paralel.

$M_{1k*8}$ - specifică o unitate de memorie unidimensională cu 1024 de cuvinte a 8 biţi pe cuvânt.

$O_{w*b}$ - specifică o memorie ortogonală, bidimensională de w cuvinte a câte b biţi şi care poate

furniza fie cuvinte de b biţi pe linie, fie cuvinte de w biţi pe coloană.

**10°. Caracteristicile temporale ale unităţilor**

Acestea sunt indicate printr-un indice superior numeric. Unitatea de măsură pentru timp este ns. Dacă se folosesc alte unităţi, acestea vor fi cuprinse între paranteze normale ca şi comentarii.

Exemple:

$I^{13}$ - unitate de instrucţiuni cu un ceas cu perioada de 13 ns;

$E^{150}$ - unitate de execuţie cu un timp mediu de operare de 150 ns;

$M^{450}$ - unitate de memorie cu timpul de acces de 450 ns;

$M_{4*4}$ - reprezintă o unitate de memorie cu capacitatea de 1024 cuvinte a 4 biţi şi cu un timp de acces de 160 ns.

**11°. Conexiunile între unităţi prin magistrale de date**

Acestea sunt reprezentate cu ajutorul următorilor conectori:

——————— conexiune de tip nespecificat;

————————> conexiune simplex la dreapta;

<———————— conexiune simplex la stânga;

Conexiunea simplex poate transfera informaţii numai în direcţia indicată.

<——/——> conexiune semiduplex;

Conexiunea semiduplex asigură transferul de informaţii în ambele sensuri, dar nu simultan.

<————————> conexiune duplex;

Conexiunea duplex realizează transferul informaţiilor în ambele sensuri simultan.

Reprezentări cu aceeaşi semnificaţie se obţin prin utilizarea separatorilor "," şi "/":

<————————> abreviere pentru {<——,——>};

<——/——> abreviere pentru {<——/——>};

În reprezentarea curentă, conexiunea poate fi indicată prin linie continuă sau linie întreruptă, de lungime oarecare:

E<- - -M ; E<————————M ; E<- -/- ->M;

Se poate observa folosirea separatorului ";" între diverse expresii.

Lăţimea magistralei, ca număr de biţi de date + număr de biţi de adrese (opţional), se indică între acolade, în cadrul liniei care reprezintă magistrala:

———{16+16}——— magistrală care transferă date şi adrese de câte 16 biţi fiecare.

În cazul în care se doreşte specificarea perioadei de lucru a magistralei în ns., se va utiliza această valoare ca factor numeric precedat de simbolul " # ". Dacă magistrala de mai sus asigură transferul unui cuvânt la fiecare 150 ns., se va putea folosi următoarea notaţie:

———————{150#{16+16}}———————

O conexiune simplex la stânga, care transferă 8 biţi de date, se reprezintă astfel:

<————————8————————

Pentru a specifica multiplicitatea magistralelor înaintea acoladelor se plasează un

multiplicator numeric (factorul de multiplicitate ) urmat de un asterisc "*". Astfel:

<————{100#4*{64+16}}————>

reprezintă patru magistrale duplex care transferă câte 64 biţi de date şi 16 biţi de adresă la fiecare 100 ns.

Magistralele de date pot avea un caracter complex, comportându-se ca nişte comutatoare. Ele vor fi specificate printr-un identificator H (eventual cu un sufix numeric) şi utilizând substituţia se obţine expresia:

E-H2-M; H2={{-16->,<-8-}/<-24-}

Partea dreapta a definiţiei conţine numai specificarea magistralei de date. H2 reprezintă un comutator între unităţile E şi M care poate opera în trei regimuri: duplex (pe 8 biţi de date) sau simplex la stânga pe 24 de biţi de date sau la dreapta pe 16 biţi.

**12°. Conexiunile serie**

Conexiunile serie specifică un lană de unităţi, legate prin magistrale de date având aceeaşi semnificaţie ca şi legarea în serie a elementelor circuitelor electrice.

Exemplu:

E1-M1-M2 / conectate logic în serie

**13°. Conexiunile paralele**

Conexiunile paralele ale unităţilor au aceeaşi semnificaţie ca şi conectarea în paralel a elementelor circuitelor electrice, utilizează separatorii de concurenţă "," sau de secvenţialitate "/" între unităţi incluzând lista de unităţi între acolade. Pentru a obţine o flexibilitate mai mare este posibilă folosirea simbolului de neconectare "|".

În afara acoladelor, o unitate poate fi conectată după cum urmează:

**U** conexiune nespecificată în afara acoladelor;
**-U|** conexiune la stânga, fără conexiune la dreapta;
**U-** conexiune la dreapta, fără conexiune la stânga;
**-U-** conexiuni la stânga şi la dreapta.

Dacă nu apar ambiguităţi simbolul de neconectare poate fi omis.

Exemple:

{U1,U2,U3} - un grup de unităţi care operează concurent, conectat nespecificat la dreapta, fără conexiune la stânga;

-{U1/U2/U3} - un grup de unităţi care operează secvenţial, conectat nespecificat la stânga şi la dreapta;

-{-U1-,-U2-,-U3-} - un grup de trei unităţi conectate în paralel, care operează concurent, este conectat nespecificat la stânga şi la dreapta, Figura 1.21 a).

-{-U1-,|U2-,-U3|} - acelaşi grup ca şi mai sus cu menţiunea ca U2 nu este conectat la stânga, iar U3 nu este conectat la dreapta, Figura 1.21 b).

-{-U1-/-U2-/-U3-} - un grup de trei unităţi conectate în paralel care operează alternativ, conectat nespecificat la stânga şi la dreapta, Figura 1.21 c).

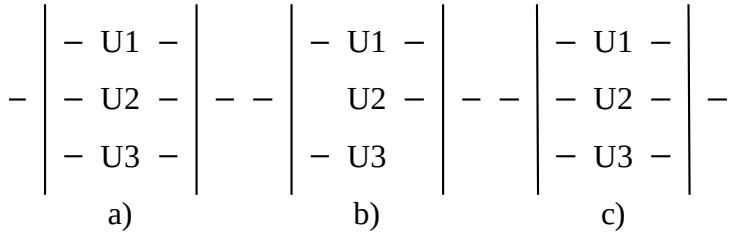



Figura 1.21 Unităţi conectate în paralel.

Se observă că diferenţa în timp între această schemă (Figura 1.21 c), şi cea cu lucru în paralel (Figura 1.21 a), nu poate fi exprimată în desen.

Pentru a ilustra posibilităţile de descriere a conexiunilor serie şi paralel cu eventualele structuri imbricate se dă următorul exemplu, din figura 1.22:

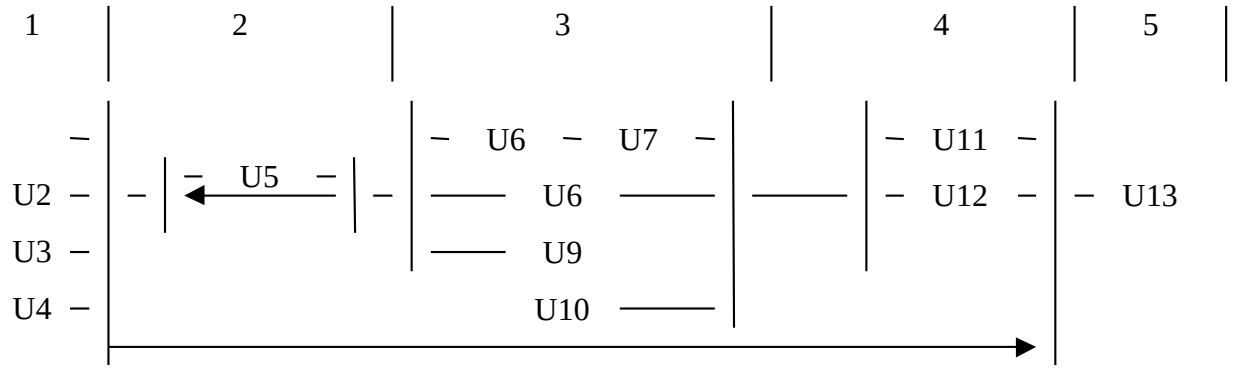



Figura 1.22 Interconectare complexă.

Trebuie notat că legăturile cu săgeată sunt unidirecţionale şi alternative.

La nivelul fiecărui tronson se deosebesc următoarele structuri:

1. {U1-,U2-,U3-,U4-}-
2. -{-U5-/<──}-
3. -{-U6-U7-,-U8-,-U9│,│U10-}-
4. -{-U11-,-U12-}-
5. -U13

2-3-4 /──────>

{U1-,U2-,U3-,U4-}-{{-U5-/<--}-{-U6-U7-,-U8-,-U9│, │U10-}-{-U11-,
-U1-}/-->}-U13

unde căile orientate sunt unidirecţionale şi alternative.

**14°. Puncte de conexiune**

Punctele de conexiune sunt specificate prin litere mici în interiorul sau la sfârţitul conectorilor:

E──a──M sau U<──/──>b

Ele sunt utilizate pentru a specifica conexiuni mai lungi sau structuri care nu pot fi descrise în maniera arătată mai sus.

De exemplu, structura constând din 3 calculatoare C1,C2,C3 conectate la 5 blocuri de

memorie M1,...,M5, va fi descrisă de schema din figura 1.23:

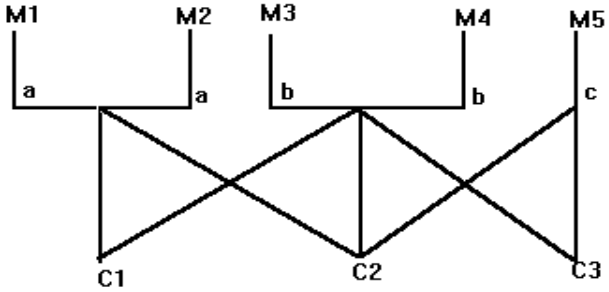



Figura 1.23 Puncte de conexiune

Algebric, aceasta se poate reprezenta astfel :

C = {C1-{a,b},C2-{a,b,c},C3-{b,c},{a-M1,a-M2,b-M3,b-M4,c-M5}}

unde punctele de conexiune a,b,c au fost specificate la fiecare bloc de memorie.

Pentru conexiunile multiple a fost plasată între acolade o listă de puncte de conexiune, ceea ce asigură facilităţi pentru a descrie o reţea arbitrară de unităţi interconectate.

**15°. Masive de procesoare cu structuri multidimensionale**

Aceste masive se pot descrie prin factorul de multiplicitate ca prefix. Structura factorului de multiplicitate indică organizarea dimensională a masivului.

Exemple:

128 * 64P specifică un ansamblu de 8192 procesoare identice, organizate într-o matrice 128 x 64.

$64^2$P indică un ansamblu de 64 x 64 procesoare identice.

Gradul de conectivitate al procesoarelor se indică sub forma unui indice superior la acolade "{}" sau paranteze drepte "[]". Forma indicelui este de tipul "c-nn", ceea ce înseamnă că un procesor poate schimba date cu cele mai apropiate (nearest neighbour) c procesoare. Dacă c este zero, conexiunea directă va lipsi.

288{3E-M}$^{0\text{-}nn}$; Sistemul PEPE cu 288 procesoare neconectate, fiecare având trei unităţi de execuţie şi o memorie, identice.

C[$64^2$P]$^{1\text{-}nn}$; Sistemul ICL-DAP cu 4096 procesoare, organizate într-o reţea de 64*64, şi cu conexiuni la cele mai apropiate procesoare, de pe primul nivel, Figura 1.24:

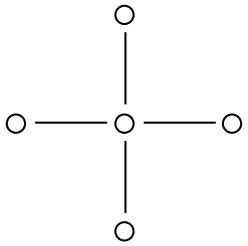


Figura 1.24 Interconectare pe primul nivel

$\{32^2\underline{P}\}^{2\text{-}nn}$; Sistemul CLIP pentru prelucrare de imagini, constituit dintr-o matrice de 32 x 32 procesoare identice, cu conexiuni la al doilea vecin mai apropiat, Figura 1.25.

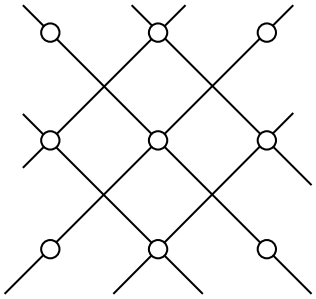


Figura 1.25 Interconectare pe al doilea nivel

Alte conexiuni de complexitate şi mai mare se pot specifica şi sub forma unor comentarii între paranteze.

**16°. Conexiunile încrucişate**

Conexiunile încrucişate ( de tip crossbar, sau altă reţea de comutare etc.) între unităţile multiple de execuţie şi unităţile de memorie sunt indicate prin simbolul "X". Timpul de transfer şi numărul de biţi transferaţi se pot indica într-o manieră asemănătoare celei folosite la magistrale.

Ip[16F X 17M]; sistemul BSP (Burroughs Scientific Processor), cu 16 unităţi de execuţie în VM identice interconectate cu 17 blocuri de memorie, de asemenea identice.

Pentru a facilita descrierea se pot folosi simboluri X multiple, care sunt semantic identice cu un singur simbol. În cazul în care se doreşte descrierea funcţiei de comutare/conectare, ca un comentariu între paranteze, la extremităţile comentariului se plasează simbolul X.

Să considerăm cazul conectării printr-o reţea de schimb cu permutare de date, plasată între n unităţi de execuţie şi m unităţi de memorie.

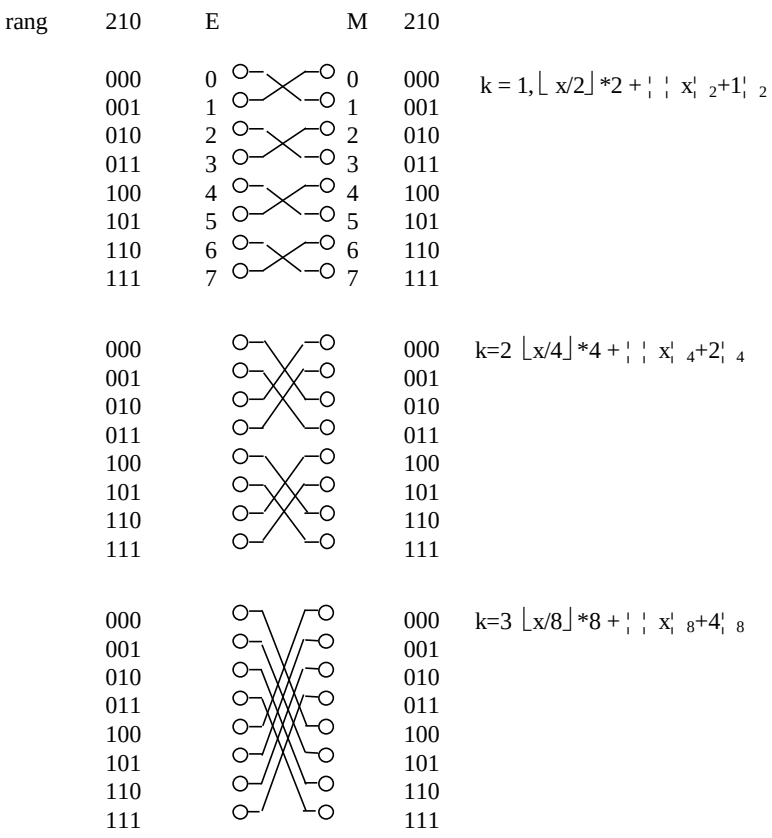
$$nE\ x\ (\lfloor x \rfloor / 2^K \mid 2^K + \mid \mid x \mid_2{}^K + 2^{K-1} \mid_2{}^K,\ k=1,2,...,\log_2(n))\ x\ mM$$

Aici $\mid f \mid$ este funcţia întreagă prin aproximarea inferioară a lui f, iar $\mid x \mid_2 K$ reprezintă faptul că argumentul este luat modulo $2^k$. Interconectările sunt arătate în figura 1.26.

Pentru a facilita claritatea se pot folosi şi alte notaţii:

nE x H2 x mM; H2 = H ( $\mid x \mid_2 (k+1)$ )

Se consideră cazul m=n=8




**17°. Comentariile**

Comentariile sunt incluse între paranteze rotunde şi conţin informaţii adiţionale despre simbolul imediat precedent.

De exemplu, o ierarhie de memorii poate fi descrisă astfel:

$M1^{10}$(bipolar)-$M2^{400}$(MOS)-$M3^{(1ms)}$(disc)

O unitate de înmulţire în VM de tip BA în tehnologia ECL:

Fp(*,ECL)

O unitate de instrucţiuni de tip BA cu 4 segmente :

Ip(4seg)

Simbolul nespecificat U permite definirea oricărei unităţi, de exemplu unitatea de comandă IBM 2803 se poate reprezenta astfel: U(2803).

În cazul conexiunilor specifice pentru transferul datelor între unităţi se folosesc următoarele notaţii:

E-(cablu coaxial 2 km)-M

E x (reţea de tip Banyan) x M

Pentru a specifica numărul de biţi manipulaşi se folosesc notaţiile : b(bit), B(byte-octet) cu prefixul uzual utilizat în SI şi cu convenţiile :

k=1024, $M=k^2$, $G=k^3$, $T=k^4$

### 1.3.3 Comanda unităţilor

În mod uzual un set de unităţi care operează sub controlul unui flux de instrucţiuni defineşte un calculator.

#### 18°. Calculatoare şi procesoare

Conform definiţiei de mai sus un calculator trebuie să conţină cel puţin o unitate I. În contextul unei structuri generale o unitate de tip I este programabilă şi prelucrează fluxul de instrucţiuni specificat de utilizator. Cel mai simplu calculator este descris astfel:

C = I [ E - M ]

Un procesor reprezintă un set de unităţi care pot prelucra date, dar nu pot prelucra fluxuri de instrucţiuni. El conţine unităţi E dar nu conţine nici o unitate I.

P = E - M

Această definiţie corespunde utilizării procesoarelor în cadrul unor masive constituite din unităţi E-M, aflate sub controlul unei unităţi externe, ca în cazul sistemului ICL-2900- DAP (unde ICL-2900 reprezintă calculatorul gazdă - unitatea de comandă). Unitatea de comandă prelucrează fluxul de instrucţiuni furnizat de utilizator. În realitate, multe unităţi de execuţie sunt controlate de unităţi de comandă, la nivelul lor, prin microprogram, microinstrucţiunile fiind executate de unitatea E. Pentru structura globală este relevant dacă o unitate sau sistemul sunt programabile de către utilizator, chiar dacă la nivelul unităţilor fiecare E este controlată de un flux de microinstrucţiuni. Acesta este cazul unităţilor aritmetice bazate pe principiul benzii de asamblare, în care, la nivelul dat, unitatea care prelucrează fluxul de microinstrucţiuni este de tipI.

#### 19°. Gradul de exercitare a comenzii

Gradul de exercitare a comenzii de către I sau C se indică cu ajutorul parantezelor drepte. Unităţile controlate sunt listate între paranteze separate prin virgulă sau linii oblice, dacă operează concurent sau secvenţial.

**Exemple:**

I1[4I2[64P]]; unde I1 este o unitate principală de instrucţiuni, care controlează patru cadrane, fiecare cu o unitate de instrucţiuni I2 şi câte 64 procesoare identice P, care operează simultan.

Ip[C1,C2,C3]; unde Ip este o unitate de comandă de tip bandă de asamblare care controlează trei calculatoare ce operează simultan.

C1[Ep1/Ep2]; C1=I[B-M]; unde C1 este un calculator de comandă care conţine o memorie şi efectuează operaţii booleene; el controlează două unităţi de execuţie de tip BA, care operează câte una la un moment dat.

#### 20°. Tipul de comandă

Tipul de comandă exercitată de o unitate I sau C poate fi indicat opţional printr-un indice inferior literal la sfârţitul parantezei de comandă.

**a(asincron)**: unităţile controlate dispun de mai multe generatoare de tact; ele nu sunt sincronizate iar comunicaţiile între unităţi trebuie să fie coordonate prin semafoare sau protocoale specifice.

**h(orizontal)**: o singură instrucţiune compusă controlează activitatea unui set de unităţi diferite, în cadrul fiecărei perioade de ceas (sistemul FPS AP-120B).

**l(pas cu blocare-lock step)**: un set de procesoare identice controlate asincron pentru a executa aceeaşi operaţie, în acelaşi interval de timp (sistemul ICL-DAP).

**r(semnalizează când este pregătit-issue when ready)**: instrucţiunile sunt emise spre unităţile de execuţie, în momentul când sunt pregătite atât unităţile, cât şi registrele solicitate (CRAY-1).

Exemple:
I[10F,10C]$_r$ : CDC6600 cu 10 unităti independente de execuţie în VM şi 10 calculatoare identice pentru I/E. Instructiunile sunt generate când unităţile sunt pregătite.
C[64P]$_l$ : ILLIAC IV cu 64 de procesoare identice controlate sincron.
I[4C]$_{(descrierea\ comenzii)}$: patru calculatoare controlate de o unitate I în maniera descrisă în paranteze.

Pentru exemplificarea notaţiilor şi a puterii de reprezentare a acestora, în continuare se consideră câteva exemple. Se observă că un calculator poate fi descris printr-o ecuaţie pe o linie de text. Notaţiile pot fi folosite pentru o descriere ierarhică, utilizând substituţia şi comentariile, începând de la arhitectura generală până la specificarea registrelor interne şi a conexiunilor acestora.

Exemple:
C(Z80+memorie)=C(Z80)-8-$M_{64K+8}$; C(Z80)=$I^{50}$[$B_8$-8-$M_{18*8}$]

C(Intel 8086)=$I_8$[$B_{16}$-16-$M_{13*8}$]-16-$M_{1M*16}$

C(EDSAC1)=I[$B_l$-M]; M=$M_{512*35}$

C(IBM 7090)=I[$F_{36}$-M]

C(CDCC 6600)=I[10E-32M-10P]$_r$; 10E={4$F_{60}$,6B}

C(CDC 7600)=I[9Ep-M-16P]$_r$; 9Ep={3Fp,6B}; M=32M1-8M2

C(IBM 360/195)=$Ip^{54}$[2C-32$M1^{160}$(1KB)-16$M2^{800}$];
2C={C1,C2}; C1=I[3Fp-M]; C2=I[B-M]

C(CRAY-1)=$Iv^{12}$[12$Ep^{12}$-16$M^{50}$]$_r$; 12Ep={3$Fp_{64}$,9B}

C(CYBER 205)=$Iv^{20}$[4F$p_{64}$-512$M_{K*32}$]

C(TIASC(2IPU,4pipe))=2Ipv[2Fp]-8M-8P

C(ILLIAC IV(4cadrane))=C1[4$C2^{80}$[64P]$^{-nn}$]; P=$F^0$-$M_{64}$

C(PEPE)=C1[3I[288{3E-M}]$^{-nn}$]; C1=C(CDC7600)

$C(BSP)=Ip[16\underline{F}'17\underline{M}]_l$

$C(64*64\ ICL\text{-}DAP)=C[64^2\underline{P}]^{-nn}$;  $P=B_1\text{-}M_{4K*1}$

$C(STARAN)=I[32[256\underline{B}_1'O_{256*256}]_l]\text{-}M$

$C(OMEN\text{-}64)=I_{16}[64\underline{B}_1\text{-}O_{64*16}\text{-}E_{16}]_l$

$C(HYPERCUB)=I[2^4\underline{C}]$;  $C=C(2*INTEL\ 8080)$

O reţea de calculatoare de tip stea cu un calculator central C şi cu 6 calculatoare C1,...,C6 satelit se poate reprezenta astfel:

C-{-C1,-C2,-C3,-C4,-C5,-C6}.

Un număr de 6 calculatoare conectate în inel pot fi reprezentate după cum urmează:

{-C1-C2-C3-C3-C5-C6-,-}  sau  2{-C-C-C-} .

## 1.4 Reprezentarea PMS a structurii calculatoarelor

O altă modalitate de a reprezenta structura sistemelor numerice de prelucrare a datelor este referită ca descriere PMS deoarece consideră că elementele de bază în cadrul structurii sunt Procesoarele, Memoriile şi Comutatoarele (Switch-urile).

Descrierea PMS are ca elemente principale:

- procesorul central ($P_c$)
- memoria (M)
- comutator (S)
- linie de legătură (L)
- procesor de intrare / ieşire ($P_{I/E}$)
- unitate de comandă (K)
- operator asupra datelor (D)
- interfaţă + terminal (T)

Sistemul de calcul examinat la acest nivel prelucrează informaţia care este caracterizată (măsurată) în biţi, ranguri zecimale, caractere alfanumerice etc.

Sistemele sunt caracterizate prin :

- debite de transfer;
- capacitate de memorare a informaţiei.

Aceste caracteristici sunt utilizate în studiul sistemelor de calcul, cu ajutorul metodelor cercetării operaţionale, în aplicaţii de multiprogramare, timp divizat, reţele de calculatoare, etc. Problemele apărute în aceste cazuri se referă la staţii de servire, legi de aproximare a ratei sosirilor, cozi de aşteptare, cereri de resurse în sistem, etc.

Primitivele cu care se operează la acest nivel sunt caracterizate prin atribute, iar atributele au asociate valori:

C ($a_1$:$v_1$ ; $a_2$:$v_2$ ; ..... ; $a_n$:$v_n$)

unde C - este primitiva (Pc, Mp, S, L, K, T etc)

$a_i$ sunt atributele primitivei;

$v_i$ sunt valorile asociate atributului.

### 1.4.1 Primitivele PMS

In continuare să prezentăm atributele şi valorile principalelor primitive utilizate în PMS.

**1°. Memoria M**

Memoria M are 2 componente : Mp = memorie principală;
Ma = memorie auxiliară.

Are rolul de a păstra informaţia şi de a o actualiza prin procesul de scriere/citire. Memoria trebuie să dispună de un sistem de adresare care face corespondenţa biunivocă adresă - locaţie de memorie.

Atributele memoriei sunt:

- funcţia: primară │ auxiliară;
- tehnologie: bipolar │ MOS │ statică │ dinamică;
- operaţii: citire │ citire / scriere (EPROM │ RAM);
- mod de acces: aleator │ secvenţial │ FIFO │ LIFO │ asociativ;
- lungime cuvânt: 8+1 │ 16+2 │ 32+4 (paritate la nivel de octet);
- capacitate : 64K │ 128K │ 256K │ 512K │ 1M │ 2M │ 4M │ 16M;
- ciclu de lucru : 40ns │ 80ns │ 100ns │ 150ns │ 200ns │ 300ns │ 400ns │ 450ns.

Exemplu : memoria sistemului PC/AT

M ( funcţie: primară;
tehnologie: MOS;
operaţii: citire, citire/scriere ;
mod de acces: aleator;
lungime cuvânt: 16 + 2 ≡ 2 x (8+1);
capacitate: 2Mb;
ciclu de lucru: 80 ns )

**2°. Procesorul P**

Procesorul P constituie o componentă care este capabilă să interpreteze un program (prin citirea, interpretarea şi execuţia instrucţiunilor - maşină) în vederea execuţiei unei secvenţe de operaţii. În procesul de citire, interpretare şi execuţie a instrucţiunilor - maşină, are loc generarea comenzilor către toate resursele sistemului precum şi citirea stării acestora.

Procesorul poate fi caracterizat prin atribute ca :

- funcţie: central │ de I/E │ specializat;

- număr instrucţiuni: n;
- implementare: convenţională │ microprogramată │ cu microprocesor;
- format instrucţiune: fix │ variabil;
- lungime instrucţiune: 8, 16, 32, 40, 48 biţi │ 1, 2, 3, 4, 5, 6 octeţi;
- lungime cuvânt de prelucrare: 8 │ 16 │ 32 biţi;
- ciclu instrucţiune: ciclu fix │ ciclu variabil (suma cicli maşină);
- tehnologie: SSI │ MSI │ LSI │ VLSI.

Exemplu :
$P_{AT}$ ( funcţie: central; nr. instrucţiuni: 128;
implementare: µp 80286;
format instr: variabil; lungime instr: 1, 2, 3 cuv;
lungime cuvânt de prelucrare: 8 │ 16;
ciclu instrucţiune: variabil, ciclu maşină: fix;
tehnologie: VLSI )

**3°. Comutator S**

Comutatorul S reprezintă o primitivă care asigură diverse conexiuni între componentele sistemului.

Poate avea o structură complexă, caz în care este controlat de o unitate de comandă proprie, comutarea făcându-se pe baza unor discipline de servire a cererilor de stabilire a legăturilor.

În cazul cel mai simplu comutatorul poate conecta o singură resursă la o componentă utilizată în comun şi în acest caz constă numai din buffere three-state comandate de altă resursă.

Principalele atribute ale comutatoarelor sunt:

- structura: neierarhică │ ierarhică;
- tip: simplex │ semiduplex │ duplex;
- legături realizate: număr intrări, număr ieşiri;
- concurenţa: 1 │ 2 ..... │ n (n legături simultane).

Exemplu: comutatorul de acces la magistrala de date la maşini CP/M
(Structura: neierarhică;
tip: semiduplex;
legături realizate: 8;
concurenţa: 1;
mod control acces: cereri independente;
standard: MULTIBUS modificat;
mod transfer: asincron de tip întrebare / răspuns cu interblocare completă).

**4°. Linie de legatură (L)**

Linia de legătură reprezintă o primitivă care asigură legătura între diverse componente ale sistemului în vederea transmisiei informaţiei. Primitiva L nu prelucrează informaţia, ci asigură numai deplasarea spaţială dintr-o zonă în alta.

Linia de legatură poate fi materializată printr-o magistrală sistem MAG (transmisie paralelă) sau printr-o legătură serială (de tip asincronă / sincronă).

Principalele atribute ale liniilor de legătură :

- funcţii: legătura paralelă │ legătura serială;
- lungime cuvânt: 8 │ 16 │ 32 │ │ 5, 6, 7, 8 biţi;
- modalitate transfer: asincron │ sincron sincron / asincron;
- mod dialog: fără răspuns │ întrebare-răspuns;
- mod control acces: înlănăuire serială │ interogare │ cereri independente;
- standard: MULTIBUS │ etc │ │ RS 232 │ CCITT V24;

Observaţii: Interfaţa serială poate fi privită din 2 puncte de vedere:

- fie ca linie de legătură;
- fie ca interfaţă + terminal.

Primul caz este considerat în momentul când avem structura tip reţea în care sistemele comunică între ele prin legături seriale.

Cazul al doilea este considerat pentru structurile independente în care interfaşa serială e utilizată pentru conectarea unui terminal.

**5°. Unitatea de comandă K**

Este o componentă care realizează controlul în diverse componente ale sistemului. Îndeplineşte funcţia de comandă a unor resurse asociate componentei respective. Nu trebuie confundată cu procesorul central. Această primitivă nu poate interpreta un program. În general, toate componentele unui sistem necesită o unitate de comandă mai simplă sau mai complexă.

Principalele atribute sunt :

- functii: comandă resursa i;
- implementare: convenţionala │ microprogramată;
- număr stări: n.

Exemplu: K pentru acces la magistrală
K(functii: automat acces magistrală;
implementare: convenţională;
număr stări: 4).

**6°. Procesor de intrări / ieşiri ($P_{I/E}$)**

Este o primitivă care poate fi incorporată în cadrul procesorului P, în cazul în care acesta are funcţii prestabilite de intrare - ieşire sau poate fi specificat ca primitivă independentă.

În cazul în care este privit ca primitivă independentă putem considera ca atribute generale:

- funcţie: acces direct la memorie │ canal I/E │ procesor; specializat de I / E;
- tip implementare: convenţional │ microprogramat │ microprocesor│ componente speciale;
- rata de transfer: Kocteţi / secundă;
- mod transfer: furt de ciclu │ rafală;
- lungime cuvânt: 8 │ 16 biţi.

**7.° Operator asupra datelor (D)**

Reprezintă o primitivă care produce unităţi de informaţie cu noi semnificaţii. În esenţă se efectuează operaţii aritmetice, logice şi de prelucrare primară (compactare, expandare, asociere) asupra datelor. În această categorie includem unităţile de prelucrare în virgulă mobilă, unităţile de prelucrare zecimale etc. ce sunt în general module opţionale şi de sine stătătoare (independente de procesorul central).

Ca atribute principale ale acestor primitive considerăm:

- funcţii: prelucrare în virgulă mobilă │ prelucrare zecimală │ prelucrare liste │ prelucrare vectori │ prelucrare matrice;
- tip implementare: convenţională │ microprogramată │ coprocesor matematic;
- structuri date: scalari │ vectori │ matrice │ liste;
- operaţii: + │ - │ * │ / │ căutare atom │ inserare atom │ extragere atom │ compactare;

Exemplu:
D(funcţie: virgulă mobilă;
tip implementare: microprogramată;
structură date: scalari;
operaţii: + │ - │ * │ / │ )

**8°. Interfaţă + terminal (T)**

Reprezintă o primitivă care asigură o conversie din punct de vedere fizic a datelor (adaptare electrică) precum şi sincronizarea între terminal şi Procesorul central sau Procesorul de I / E. În caz general interfaşa are asociată o unitate de comandă şi un comutator de legătură.

Ca atribute putem considera:

- funcţie: cuplare terminal i;
- tip cuplare: paralelă │ serială;
- caracteristici terminal: modalitate efectuare operaţii de citire / scriere;
- viteza de transfer: linii/minut │ caractere/sec │ cartele/ minut.

Exemplu: terminal imprimantă
T(funcţii: cuplor LPT;
tip cuplare: paralelă;
caracteristici terminal: 120 caractere/linie, 400 linii/ minut).
Standard: Centronix

Pentru reprezentări mai concise, adesea se va specifica doar funcţia îndeplinită de primitiva respectivă .

### 1.4.2 Exemple

**1°. Reprezentarea unui calculator simplu**

Mp ── Pc ── T ── Utilizator

Mp = memorie principală;

Pc = procesor central;
T = terminal;
—— = legături de date.

Prin detalierea componentelor procesorului central obţinem (Figura 1.27):

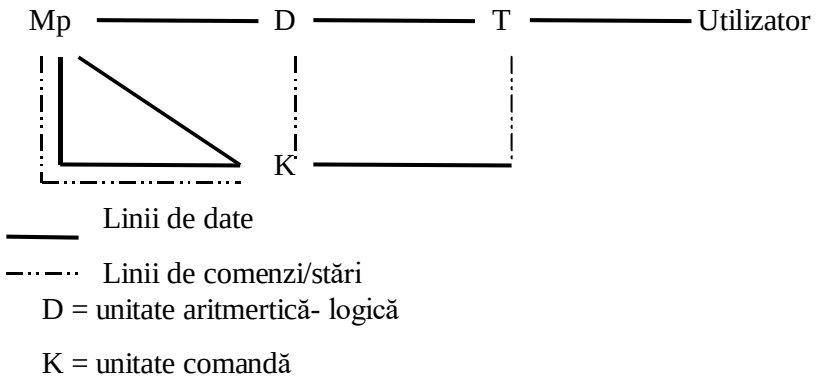



Figura 1.27.

Prin detalierea componentelor Mp, T, obţinem (Figura 1.28):

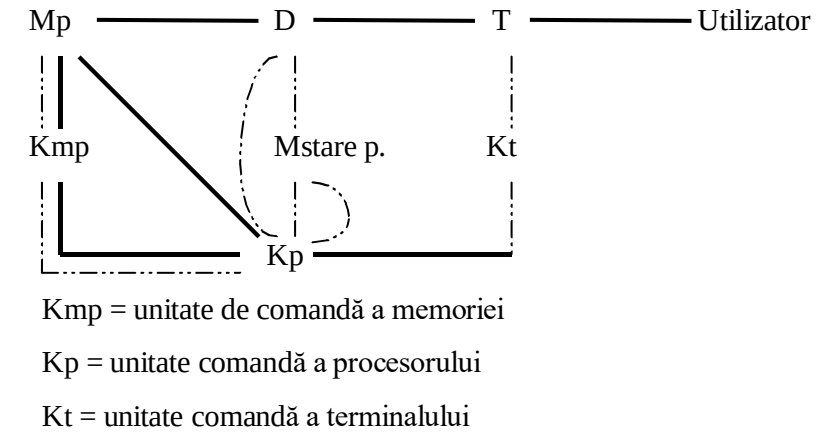



Figura 1.28.

Între componentele unei maşini de bază există transfer de date şi transfer de control.

**2°. Reprezentarea unor forme de transfer**

Să exemplificăm diversele forme de transfer date/control între componentele unui sistem numeric simplificat (Figura 1.29.):

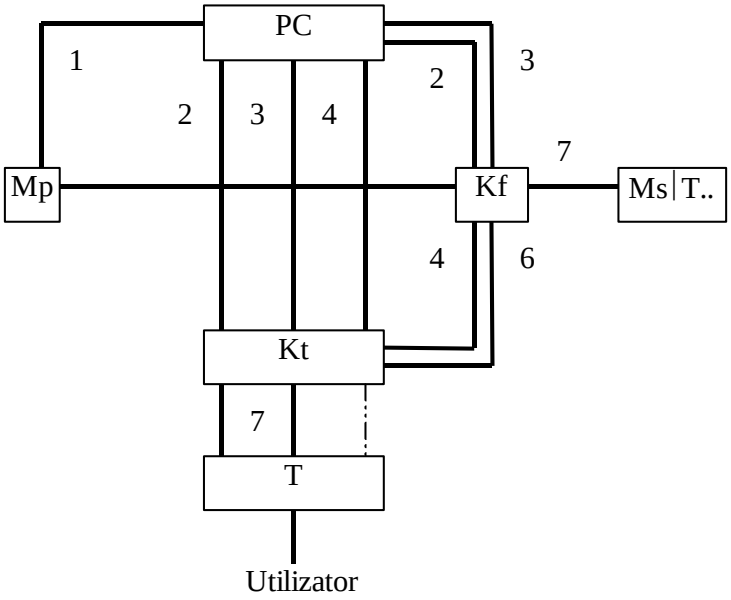



Figura 1.29.

1). Pc - Mp = comunicaţie de date:
- citirea, interpretarea şi execuţia instrucţiunilor maşină;
- manipularea datelor;

2). Pc - Kt = comunicaţie de control:
Pc - Kf
- procesorul central initializează şi comandă unităţile de comandă a interfeţelor;
- Kf, Kt execută acţiunile specificate de Pc

3). K - Pc = comunicaţie de stare:
- unitatea de comandă a interfeţei furnizează starea echipamentului de I/E.

4). Kt - Pc = comunicaţii de date:
- transferul datelor (prin transfer programat);
- Kt reprezintă o interfaţă simplă cu un debit redus de transfer;
- Pc se ocupă efectiv de transferul datelor.

5). Kf - Mp = comunicaţie de date:
- transferul datelor (fără intervenţia Pc) prin acces direct la memorie;
- Kf are resurse hardware pentru a realiza transferul de date independent de Pc.

6). Kf - Kt = comunicaţii de control
- unitatea de control a accesului direct la memorie Kf poate comanda Kt în vederea efectuării unui transfer T - Ms, T - TT.

7). K - Ms | T = comunicaţie de control, stare, date

- unitatea de comandă a interfeţei are:
- o legătură de control cu echipamentul periferic în vederea transmiterii acţiunilor pe care acesta trebuie să le execute;
- o legătură de stare prin care actualizează permanent starea EP;
- o legătură de date prin care furnizează sau preia datele de la EP.

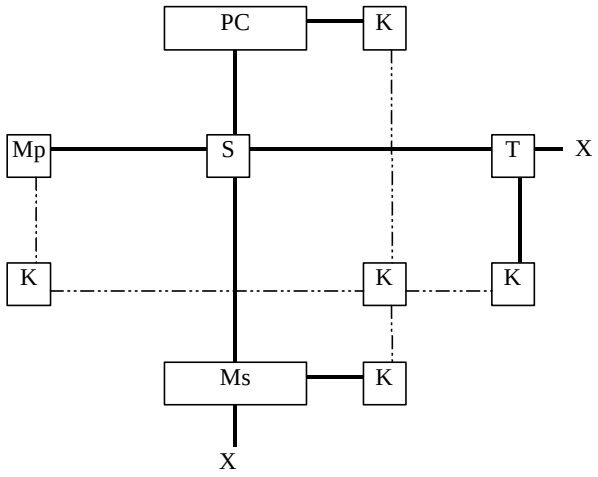
8). Pc - Pc = comunicaţii de control

- existenţa mai multor unităţi centrale de prelucrare, implică transfer de control între ele în vederea cooperării lor .

### 3°. Reprezentarea unor sisteme numerice

În continuare vom arăta descrierea PMS a unor modele de sisteme numerice.

1). Descrierea PMS a modelului general de calculator în care comanda este distribuită (Figura 1.30):




X = mediu extern (utilizator)
Ms ¦ Ma = memorie secundară - auxiliară

Figura 1.30. Model general de calculator cu comandă distribuită

2). Descrierea PMS a modelului general al unui sistem cu mai multe procesoare, blocuri de memorie principală şi periferice (Figura 1.31):

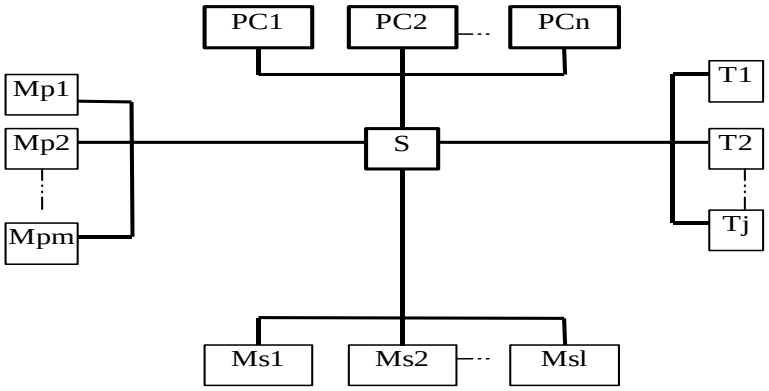
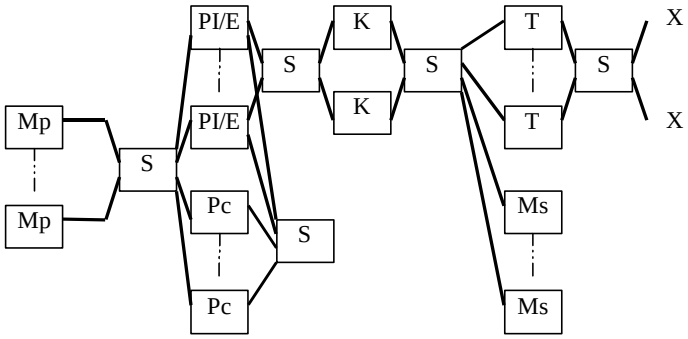



Figura 1.31.

Descrierea PMS a modelului general pentru sistem multiprocesor (Figura 1.32):




X = utilizator ¦ calculator ¦ reţea - proces fizic
PI/E = PI/E ¦ KI/E
K = K ¦ K - K ¦
T = T ¦ K - T ¦
Ms = Ms ¦ K - Ms

Figura 1.32.

Descrierea PMS a unui sistem monoprocesor în structura arborescentă (Figura 1.33.):

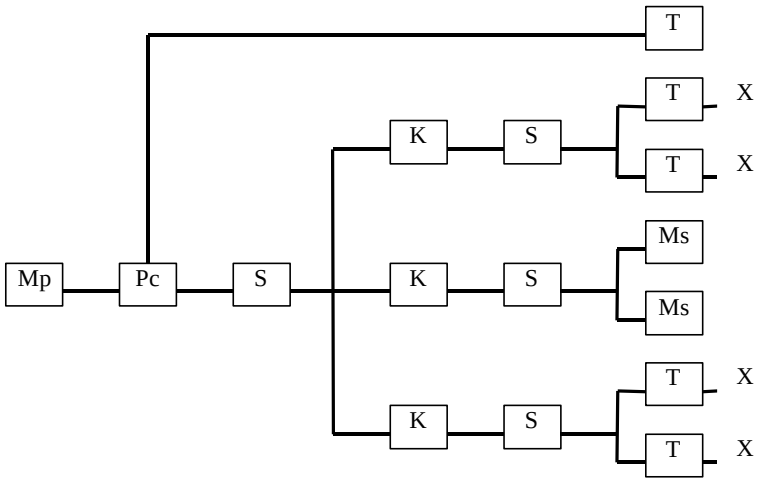



Figura 1.33

Descrierea PMS a unei structuri arborescente cu comutator de tip reţea (Figura 1.34.):

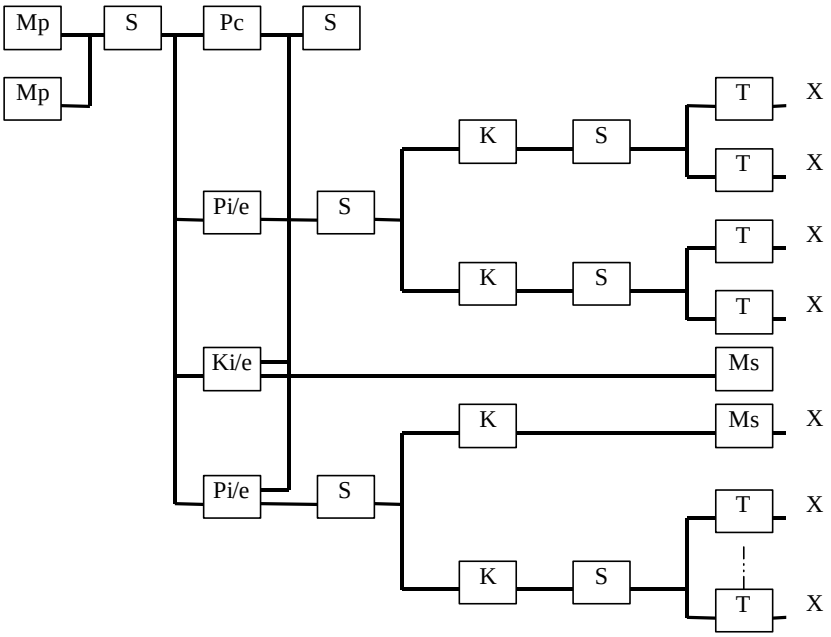



Figura 1.34.

Exemplu (Figura 1.35.)

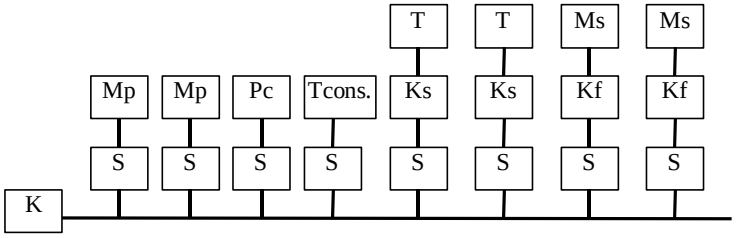



Figura 1.35. PDP 11

**Comutatoare**

Un obiectiv principal al proiectării unui sistem de calcul la nivel PMS îl constituie comutarea între resursele sistemului.

Comutatoarele se pot împărţi în două grupuri mari şi 10 categorii astfel:

**Grupul I - Comutatoare de tip ierarhic** - care comută o componentă $a_i$ la o altă componentă $b_j$.

**1°. comutator simplu** - asigură comunicaţia între o componentă de tip a şi o componentă de tip b (Figura 1.36.):

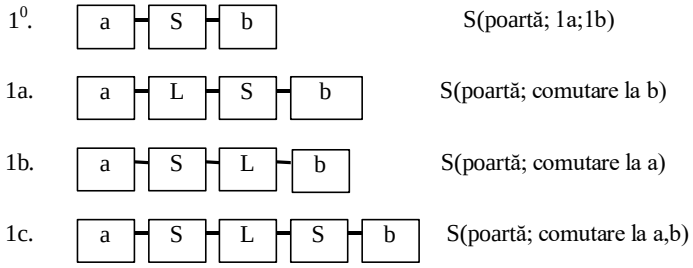



Figura 1.36. Comutator simplu

Acest comutator poate intra în structura altor comutatoare.

**2°. comutator duplex** - asigură comunicaţia între o componentă de tip a şi mai multe componente de tip b. Acest comutator este caracteristic structurilor de tip magistrală în care componenta de tip a este un procesor iar componentele de tip b sunt Mp, Ms | T (Figura 1.37):

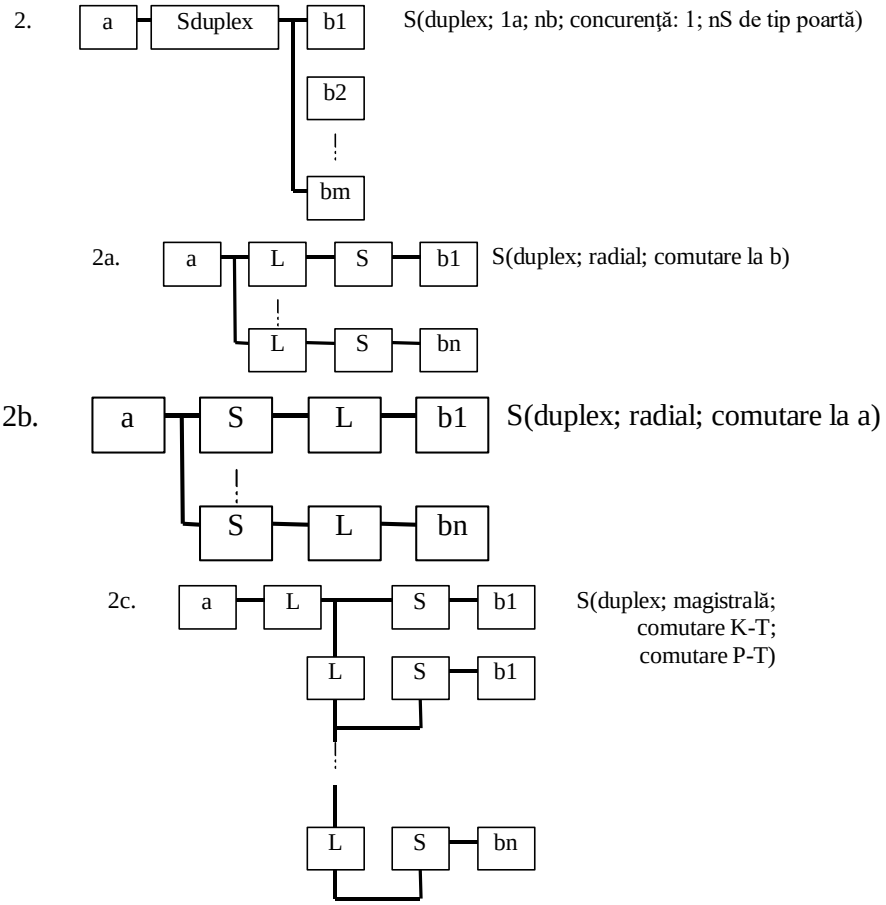
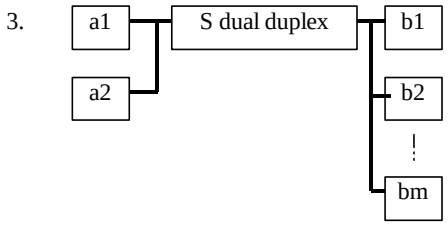
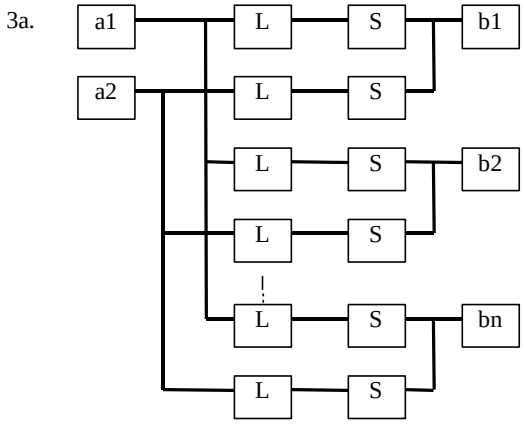


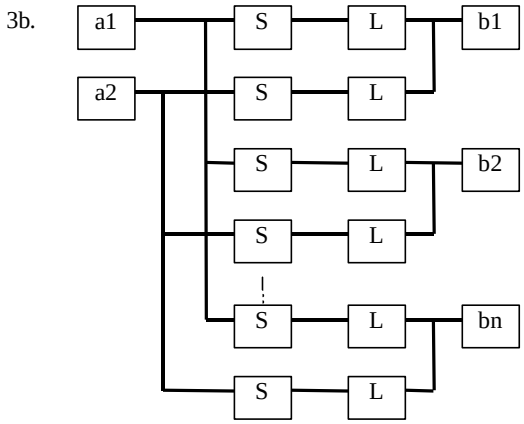

Figura 1.37. Comutator duplex

**3°. comutator dual duplex** - asigură comunicaţia între două componente de tip a şi mai multe componente de tip b. Se pot desfăşura două comunicaţii simultan (Figura 1. 38):

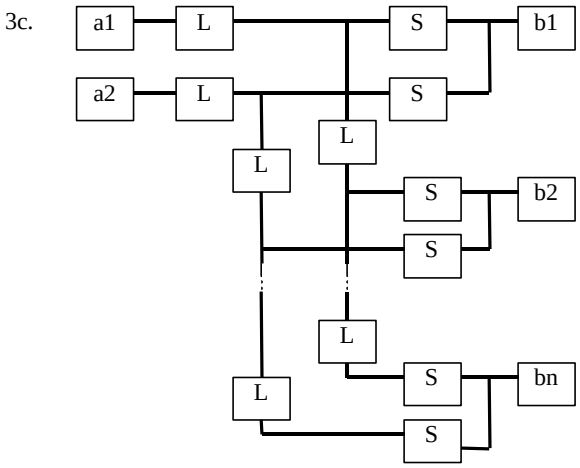





S(dual duplex; radial; comutare la b; duplex de tip 2a)



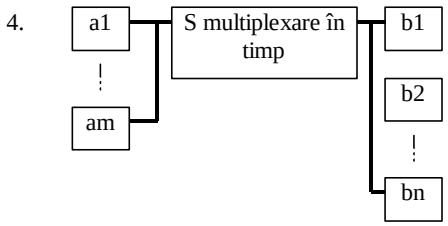

S(dual duplex; radial; comutare la a; duplex de tip 2b)



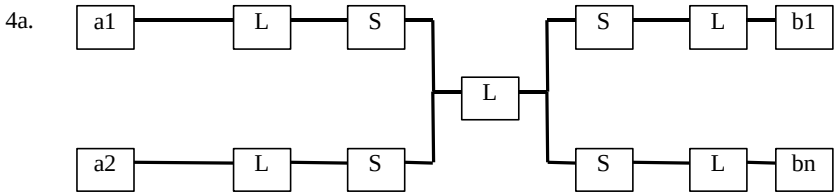

S(dual duplex; magistrală; comutatoare duplex tip 2c)

Figura 1.38. Comutator dual duplex

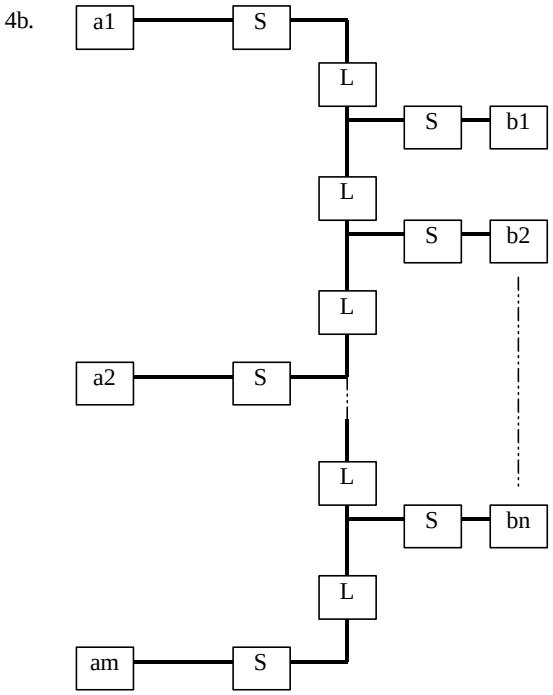
**4°. comutator multiplexat în timp (cross-point)** - asigură comunicaţia între orice componentă de tip a şi orice componentă de tip b - însă o singură legătură la un moment dat. Constituie o generalizare a comutatorului de tip duplex (Figura 1.39)




S(multiplexare în timp; ma; nb; concurenţă 1;
(m+n) S de tip poartă (cascadă de comutatoare duplex))




S(multiplexare în timp; radial; central; concurenţă 1)

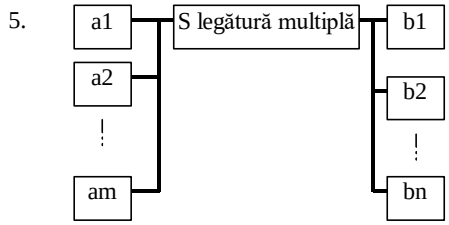



S(multiplexare în timp; magistrală)

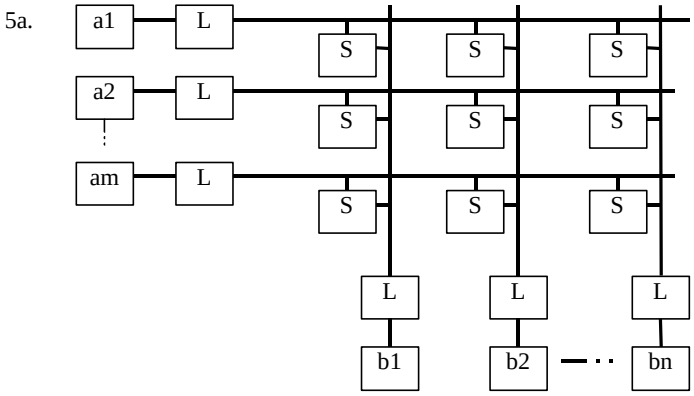
Figura 1.39. Comutator multiplexat în timp

**5°. comutatoare legături multiple** - care asigură comunicaţia între orice componentă de tip a şi orice componentă de tip b.

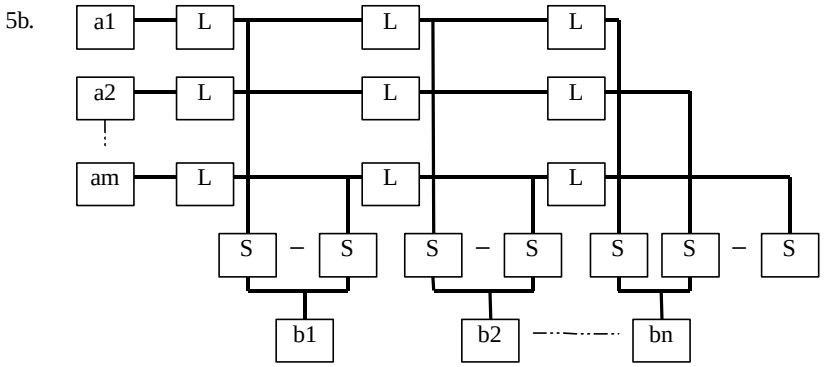
Constituie o generalizare a comutatorului de tip dual duplex. (Este utilizată în sistemele multiprocesor Cmmp şi Burroughs) (Figura 1.40):




S(legătură multiplă; ma; nb; concurenţă min(m,n);
(mxn) S de tip poartă)



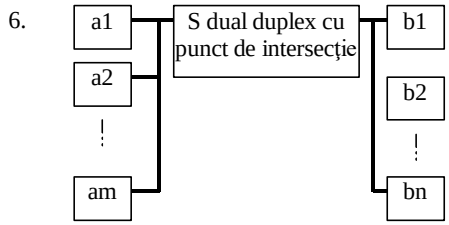

S(legătură multiplă; radial;)




S(legătură multiplă magistrale;)

Figura 1.40. Comutator legături multiple

**6°. comutatoare dual duplex cu legătură multiplă** - asigură comunicaţia între orice componentă de tip a şi orice componentă de tip b, cu o concurenţă de min(m,n) (Figura 1.41):




S(dual duplex cu legătură multiplă; ma; nb;
concurenţă min(m,n); 2 x m x n S de tip poartă)

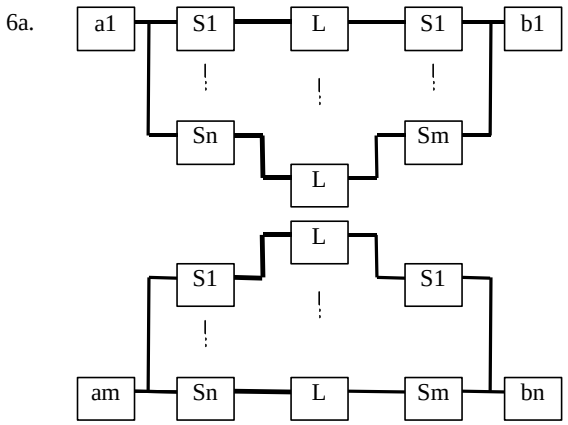



Figura 1.41. Comutator dual duplex cu legături multiple

**7°. comutare trunchi K** - îmbină concurenţa comutatoarelor multiplexate în timp şi costul redus al comutatoarelor de tip legătură multiplă (Figura 1.42).

Un exemplu foarte elocvent de comutatoare K este reţeaua telefonică.

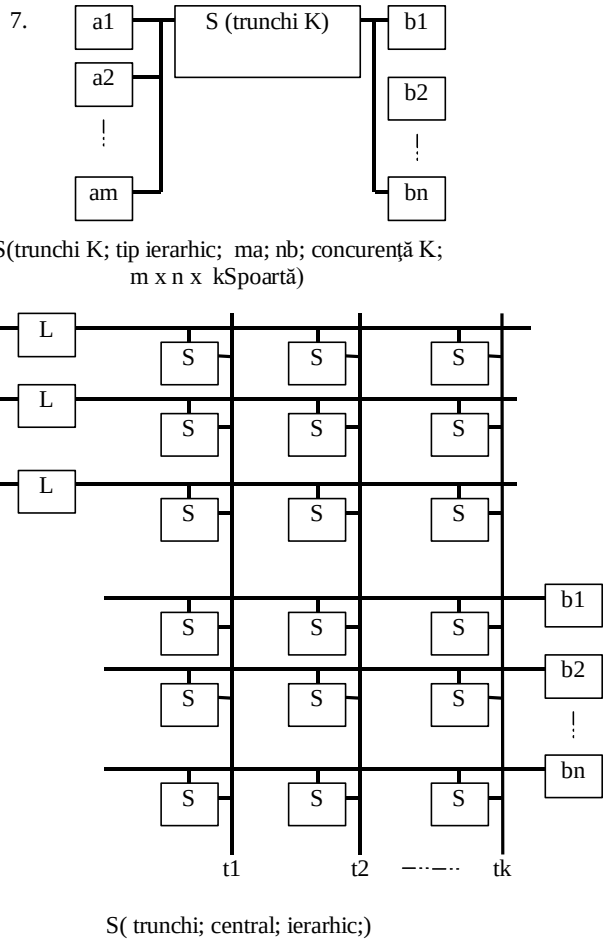
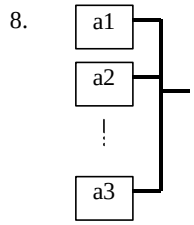


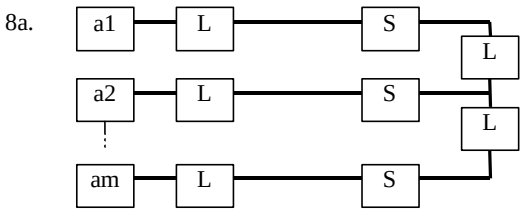

Figura 1.42. Comutare trunchi k

**Grupul II - comutatoare de tip neierarhic** - conectează componente similare.

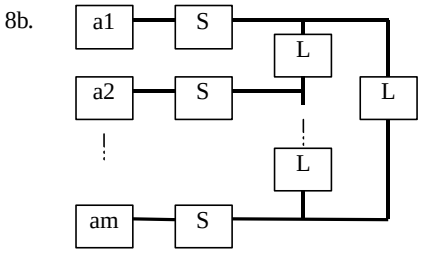
**8°. Comutatoare duplex** - corespunde comutatoarelor duplex asigurând conectarea componentelor $a_1$ .. $a_m$ prin intermediul a două căi de comunicaţie (Figura 1.43)




S(duplex; neierarhic; concurenţă 1)



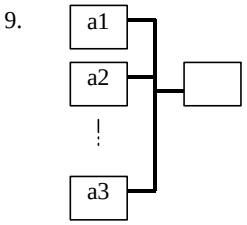

S(duplex; neierarhic; central;)




S(duplex; neierarhic; magistrală/înlănţuire)

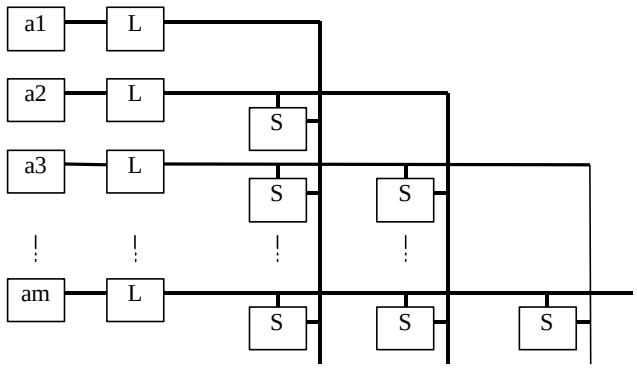
Figura 1.43. Comutatoare duplex

**9°. Comutatoare legătură multiplă - neierarhice** asigură legătura între componentele $a_1$ .. $a_m$ prin intermediul a m/2 căi (Figura 1.44.)



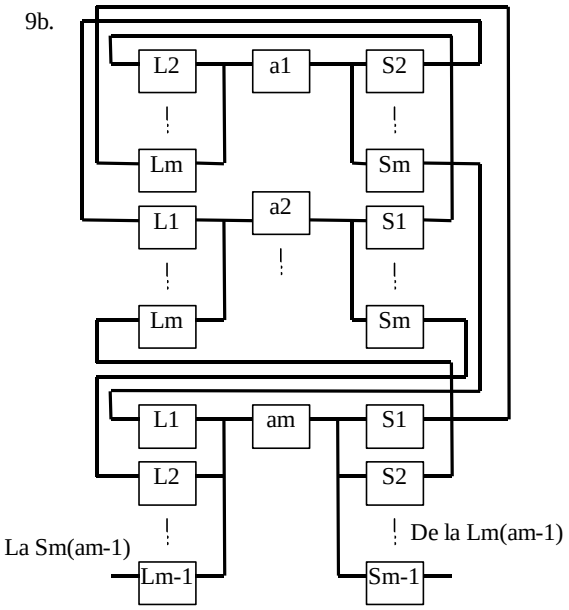

S(legătură multiplă; neierarhică;
ma; concurenţă m/2;
m x (m-1)/2 S poartă)

9a.




S( legătură multiplă; neierarhică; tip central)

9b.

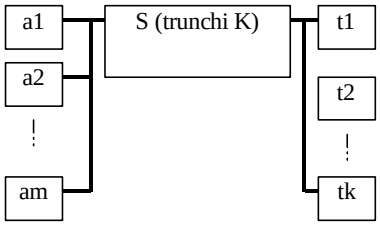



S( legătură multiplă; neierarhică; radial; m x (m-1) / 2legături )
toate nodurile au legături cu celelalte noduri.

Figura 1.44. Comutatoare legătură multiplă

**10°. Comutatoare trunchi K - neierarhice** (Figura 1.45)

10.




S(trunchi K; neierarhic; ma; concurență
;min(m/2,k) k x m S poartă)

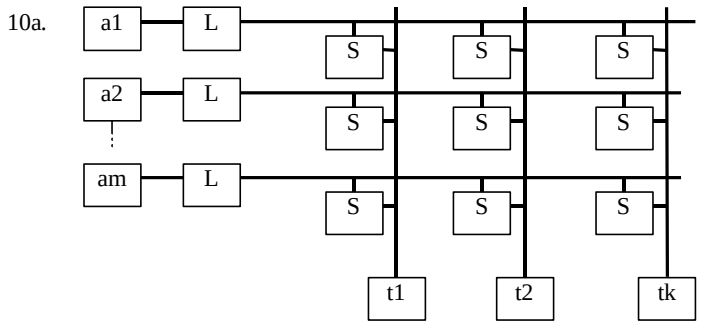



Figura 1.45. Comutatoare trunchi k neierarhice

**Exemple de comutatoare**

În cadrul structurii sistemelor numerice de prelucrare a datelor, o importanţă deosebită o au comutatoarele care asigură legătura Procesor - memorie $S_{PM}$ şi cele care asigură accesul Procesorului la magistrală $S_{PL}$.

**Comutatoare Procesor - memorie $S_{PM}$**

Aceste comutatoare prezintă importanţă în special în cadrul structurilor constând dintr-un microprocesor şi mai multe blocuri de memorie, sau cele care sunt formate din mai multe procesoare şi mai multe blocuri de memorie.

Adresele şi datele au căi separate (Figura 1.46)

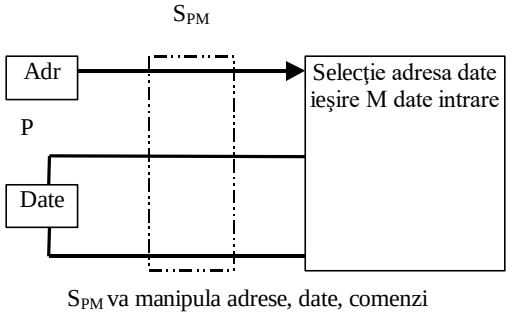



Figura 1.46. Comutator procesor -memorie având căi separate de date şi adrese

Adresele şi datele au căi separate, datele se transferă pe o magistrală bidirecţională (Figura 1.47):

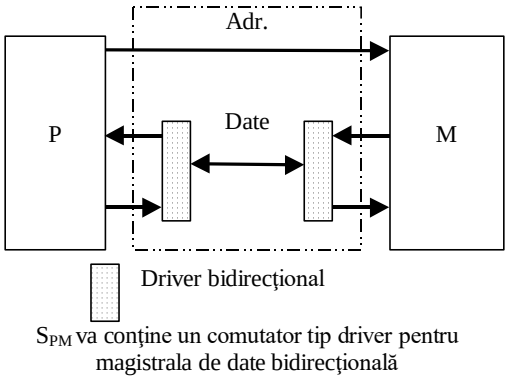



Figura 1.47. Utilizarea magistralei bidirecţionale

Adresele şi datele pot fi multiplexate pe aceeaşi magistrală (Figura 1.48):

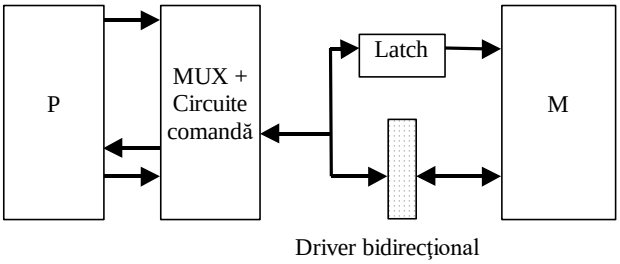



Figura 1.48. Multiplexarea adreselor şi datelor pe aceeaşi magistrală

Adresele şi datele pot fi multiplexate parţial pe aceeaşi magistrală (Figura 1.49):

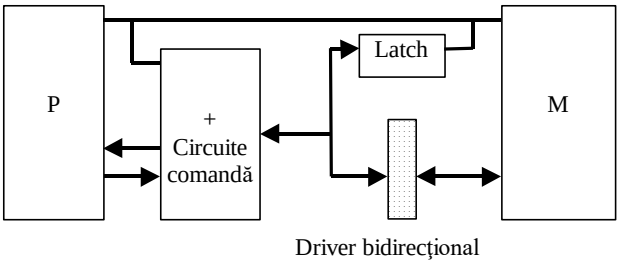



Figura 1.49. Adrese şi date multiplexate parţial

Adresele şi datele au căi separate, adresele sunt multiplexate pe magistrala de adrese (Figura 1.50.)

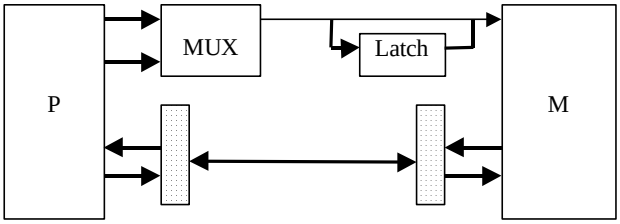



Figura 1.50. Căi separate pentru date şi adrese

Spaţiul de adresare extins pe calea de date (Figura 1.51):

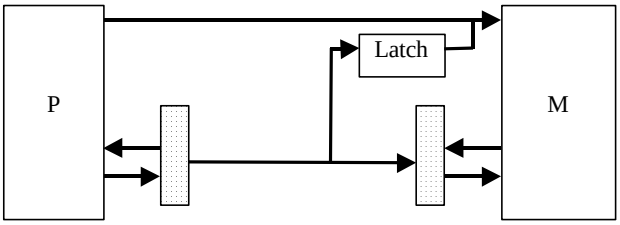



Figura 1.51. Spaţiu de adresare extins pe calea de date

**Comutatoare procesor - magistrală**

Comutatoarele procesor - magistrală se utilizează în sistemele multiprocesor sau multicalculator organizate pe o magistrală multiplexată în timp (Figura 1.52). Modulele de pe magistrală pot fi:

- de tip master -cele care pot solicita accesul la magistrală şi pot iniţia un transfer;
- de tip slave -cele care răspund comenzilor primite de la un master.

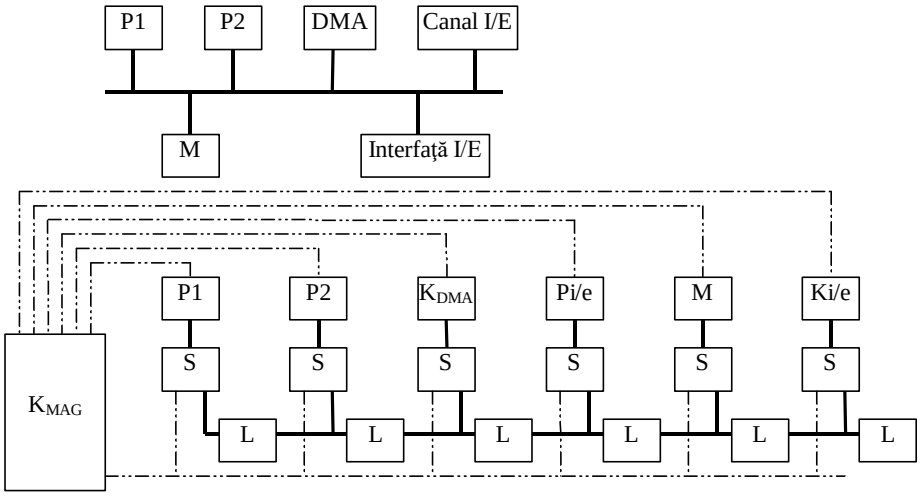


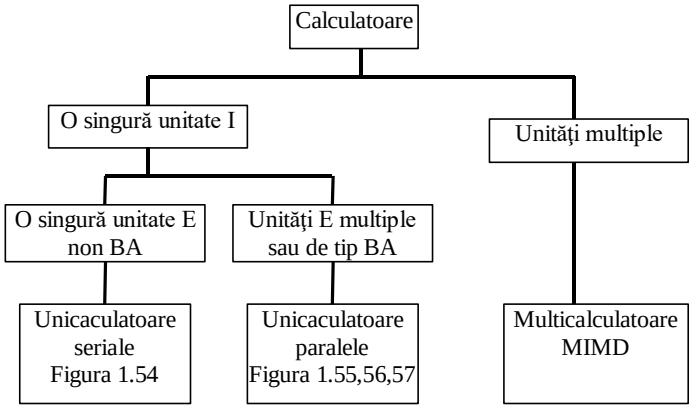

Figura 1.52 Comutatoare procesor magistrală

## 1.5 Clasificarea structurală a calculatoarelor

Utilizând notaţiile prezentate în paragrafele precedente se va defini o clasificare structurală unitară care include atât calculatoarele seriale cât şi cele paralele. În figura 1.53 se arată clasificarea generală a calculatoarelor. În continuare această schemă generală este detaliată pentru a se prezenta diferite clase de arhitecturi prin definiţia canonică şi caracteristicile acestora şi exemple din fiecare clasă.

Clasificarea structurală este realizată sub formă arborescentă, astfel că, un calculator care se găseşte pe o frunză are toate caracteristicile nivelelor pe care le traversează în calea de la rădăcina arborelui până la acea frunză.

In figura 1.53 se alege drept criteriu de clasificare, multiplicitatea unităţilor de comandă de tip I (citire, interpretare instrucţiuni). Se obţine astfel o clasificare în sisteme cu un singur flux de instrucţiuni (SI) şi cu flux de instrucţiuni multiplu (MIMD). Primele sunt apoi împărţite în clase după tipul unităţilor de execuţie E.




clasificare după:
- numărul de unităţi E (multiplicitate flux de date) sau de tip BA.
- numărul de unităţi I (multiplicitate flux de instrucţiuni);

Figura 1.53 Clasificarea generală a calculatoarelor

Următoarea clasificare se realizează după tipul operaţiilor aritmetice efectuate în unităţile de tip E.

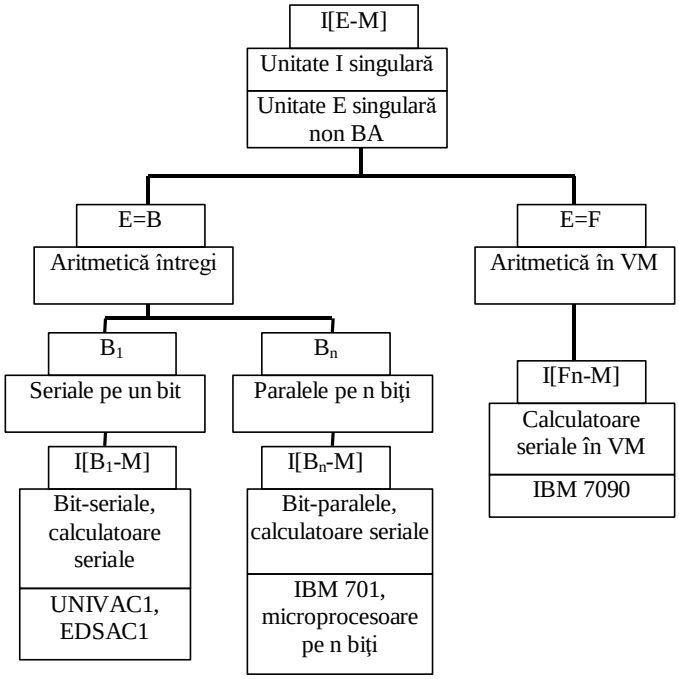



Figura 1.54 Clasificarea calculatoarelor seriale

Diferenţa de complexitate între o unitate aritmetică pe un bit şi o unitate aritmetică în VM este destul de mare pentru a justifica o împărţire calitativă - în clase diferite. Reţelele de procesoare în VM şi de procesoare pe un bit sunt diferite ca implementare şi proprietăţi. Clasificarea din figura 1.54 reflectă şi dezvoltarea istorică a microprocesoarelor.

### 1.5.1 Sisteme cu paralelism funcţional şi de tip bandă de asamblare

În figura 1.55 se arată o structură non BA de calculator scalar cu unitate de execuţie multiplă care obţine performanţele prin paralelism funcţional.

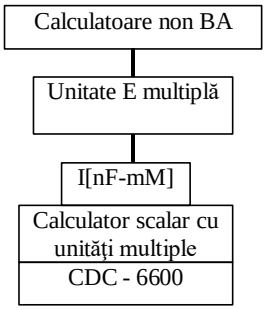



Figura 1.55 Monocalculator funcţional- paralel

În figura 1.56 structurile de tip BA sunt împărţite după cum au sau nu au instrucţiuni pentru calcule cu vectori. Aceasta diferenţiază calculatoarele scalare de cele vectoriale.

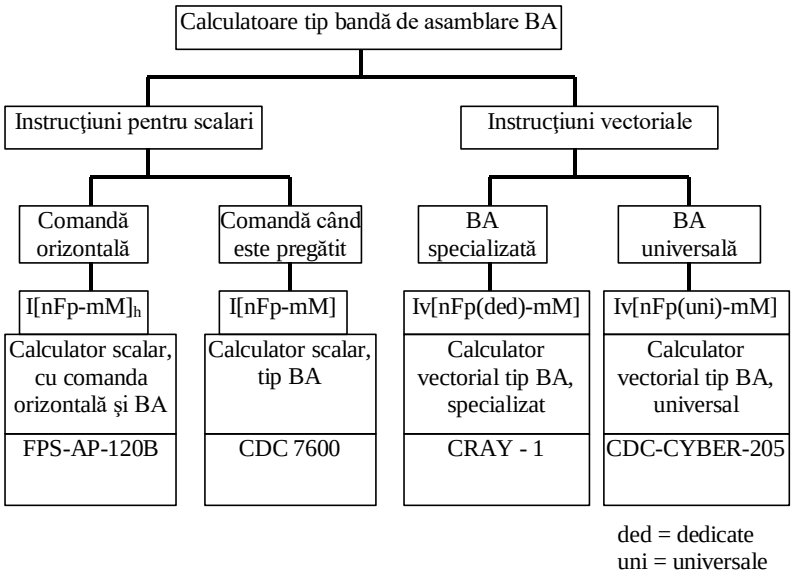



Figura 1.56 Monocalculator paralel în BA

### 1.5.2 Clasificarea masivelor de procesoare

Alternativa de obţinere a paralelismului prin multiplicarea procesoarelor şi controlul lor sincronizat (lock-stop) este prezentată în figura 1.57. Împărţirea este apoi făcută în clasa maşinilor cu prelucrare în VM şi cele cu număr mic de biţi (exemplu microprocesoare pe 8 biţi). Următoarea împărţire se face după conectivitatea procesoarelor. În sfârşit, sunt cuprinse în această clasificare procesoarele asociative şi cele ortogonale.

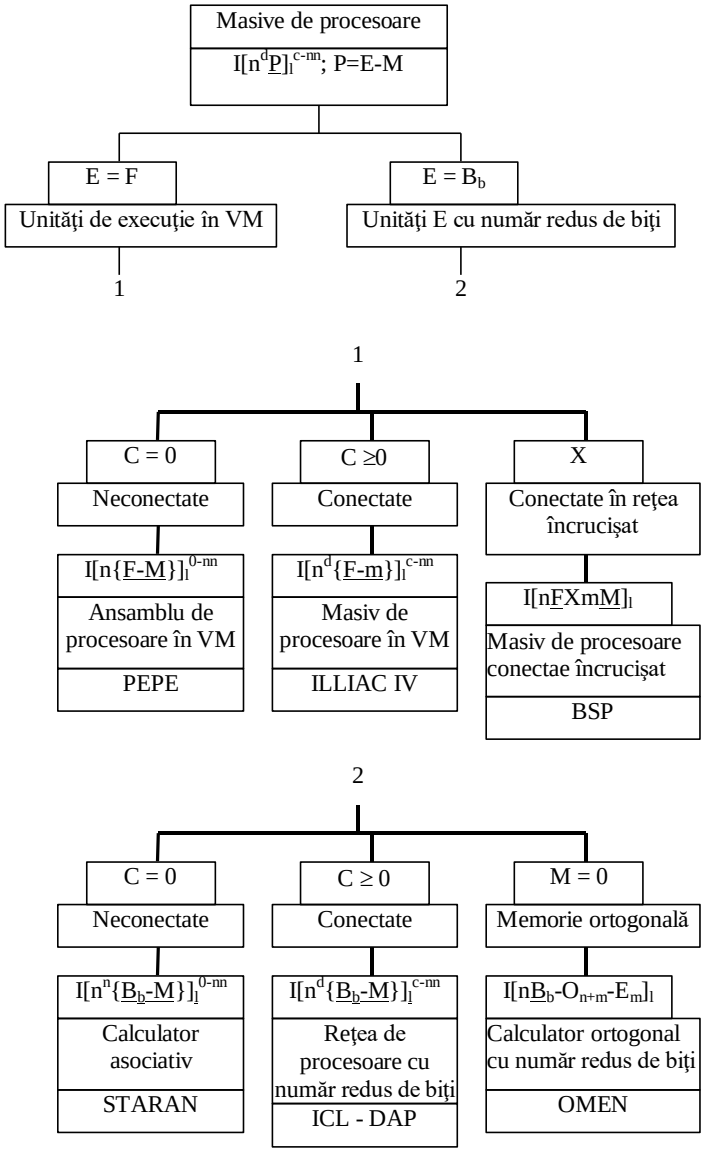



Figura 1.57 Clasificarea reţelelor de procesoare

### 1.5.3 Clasificarea sistemelor MIMD

Există o mare varietate de structuri de calculatoare care se încadrează în clasa MIMD. Sunt incluse în această clasă numai calculatoarele controlate de un flux multiplu de instrucţiuni convenţionale, deci controlate prin "flux de control" nu prin "flux de date". În figura 1.58 se arată o primă împărţire în structuri MIMD de tip BA, de tip comutat şi de tip reţea. În primul caz mai multe secvenţe de instrucţiuni utilizează o BA sofisticată, partajată în timp sau poate prevedea o unitate de execuţie pentru fiecare flux de instrucţiuni. A doua clasă presupune existenţa unui comutator separat pentru interconectare şi în sfârţit ultima clasă presupune interconectarea elementelor de prelucrare într-o reţea identificabilă şi extensibilă, care permite fiecărui element să comunice direct cu vecinii.

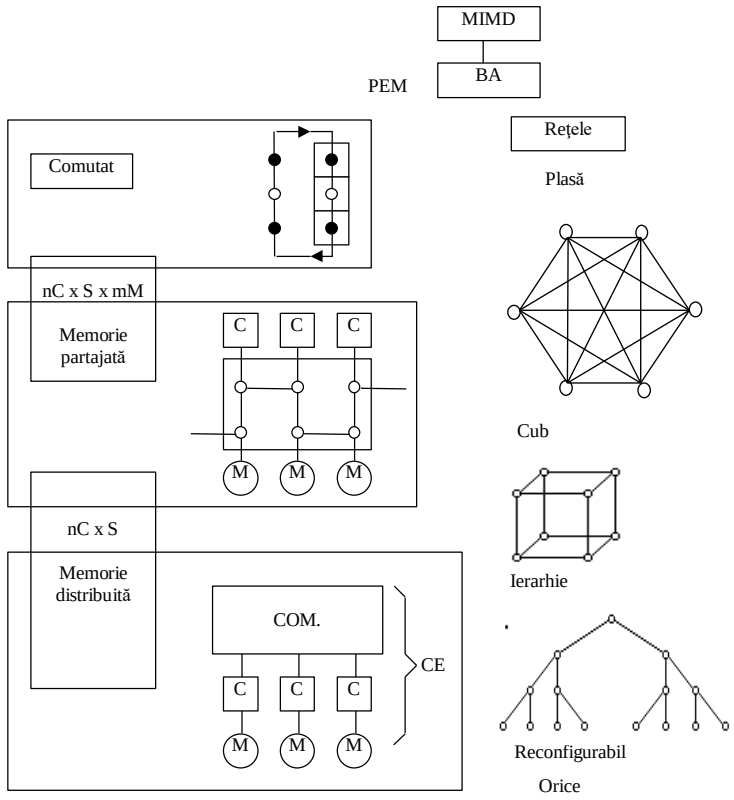



Figura 1.58.

În figura 1.59 se arată clasificarea în continuare a sistemelor de tip comutat după cum memoria este locală sau distribuită sau este partajată prin intermediul comutatorului.

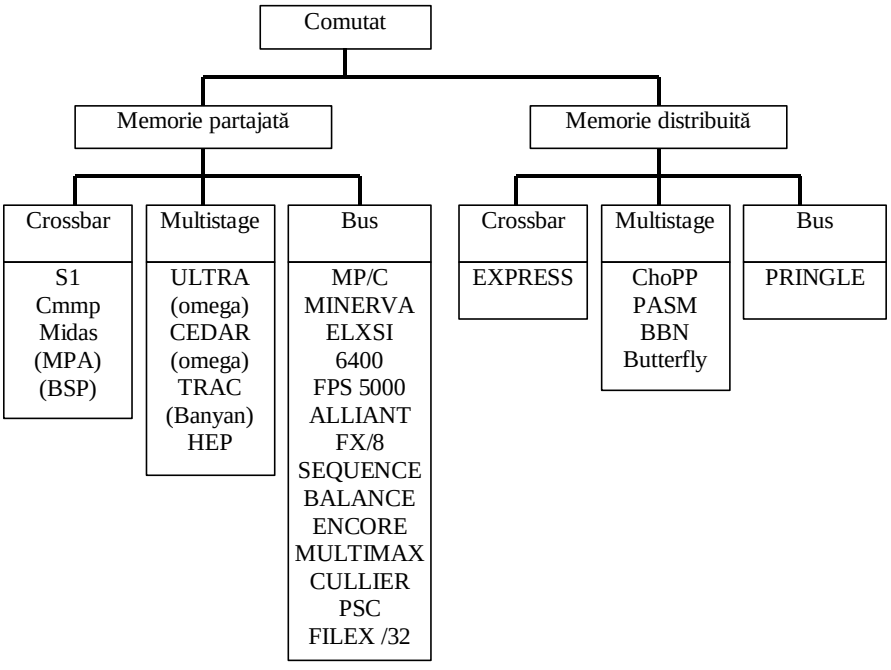



Figura 1.59

Tipul comutatorului utilizat constituie parametrul pentru următoarea subdiviziune.

În figura 1.60 se arată clasificarea sistemelor MIMD de tip reţea în funcţie de topologia reţelei formate prin interconectarea sistemelor componente.

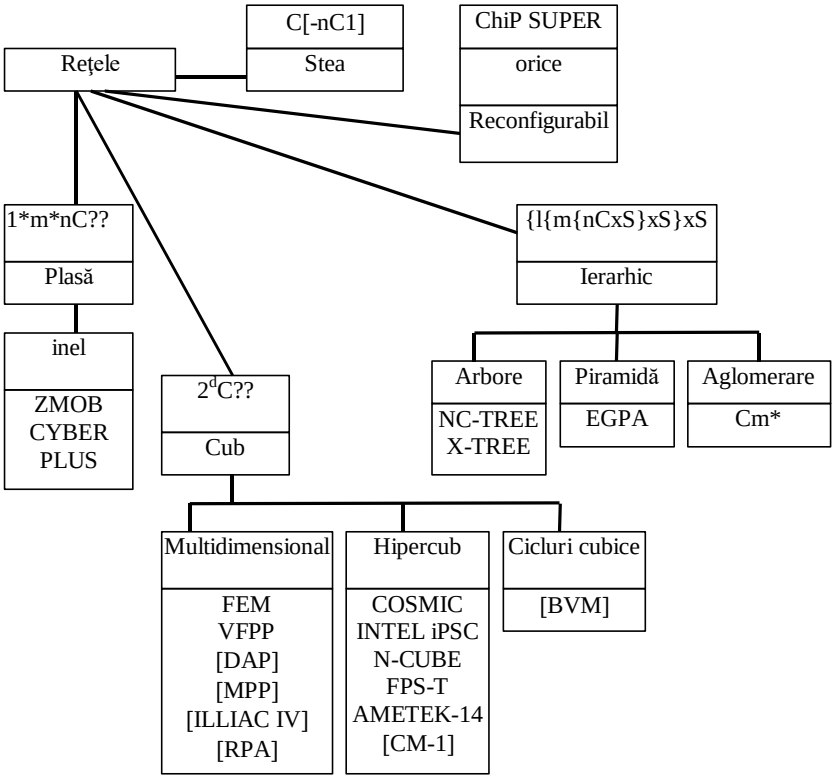



Figura 1.60 Clasificarea sistemelor paralele de tip MIMD

Cea mai simplă structură este în formă de stea în care mai multe calculatoare sunt conectate la un sistem gazdă. În această clasificare sunt incluse sisteme tip "plasă" uni şi multidimensionale, hipercuburi binare în care există doar două calculatoare pe fiecare dimensiune, reţele ierarhice sub formă de arbore, piramide sau conectate prin magistrale. În sfârţit, o grupă interesantă în această clasificare este formată din sisteme reconfigurabile sub controlul programului.

## CUPRINS


**3. SISTEME DE PRELUCRARE PARALELĂ CU MULTIPLICAREA RESURSELOR ..... 3-1**

3.1 REALIZAREA SISTEMELOR DE PRELUCRARE PARALELĂ CU MULTIPLICAREA RESURSELOR ..... 3-2
*3.1.1 Alternative pentru sistemele cu multiplicare spaţială ..... 3-2*
3.1.1.1 Sisteme cu reţea de comutare fixă ..... 3-2
3.1.1.2 Controlul prin flux de date ( data driven ). ..... 3-3
3.1.1.3 Controlul prin flux de cereri (demand driven). ..... 3-4
3.1.1.4 Control prin flux de comenzi ..... 3-5
3.2 PROCESOARE DE TIP REŢEA (ARRAY PROCESSORS) ..... 3-6
*3.2.1 Organizarea unor structuri SIMD de tip PA ..... 3-6*
3.3 SISTEME PARALELE DE TIP MULTIPROCESOR ..... 3-11
*3.3.1 Granularitatea paralelismului ..... 3-12*
*3.3.2 Caracteristici ale sistemului multiprocesor ..... 3-12*
*3.3.3 Multiprocesoarele strîns interconectate ..... 3-13*
*3.3.4 Multiprocesoare slab cuplate ..... 3-14*
3.4 ARHITECTURA ŞI STRUCTURA SISTEMULUI CM* ..... 3-15
*3.4.1 Structura lui Cm* ..... 3-15*
*3.4.2 Accesul la resursele locale ..... 3-17*
*3.4.3 Mecanismul de referinţe intermodule ..... 3-18*
3.4.3.1 Etapele de acces la memorie în interiorul unui cluster (Figura 3.20) ..... 3-18
3.4.3.2 Mecanismul de referinţă inter-clustere ..... 3-20
*3.4.4 Controlul mecanismului de comunicare ..... 3-22*
3.4.4.1 Asigurarea concurenţei în interiorul mecanismului de mapare ..... 3-22
3.4.4.2 Operaţiile care necesită excluderea mutuală (exemplu: translatarea unei adrese din adresă virtuală în adresă fizică) 3-22
*3.4.5 Structura lui $K_{MAP}$ ..... 3-22*
3.4.5.1 Comunicaţia $K_{bus}$ - $P_{map}$ ..... 3-23
3.4.5.2 Comunicaţia Link-$P_{map}$ ..... 3-24
3.4.5.3 Legătura $K_{bus}$-$MAP_{bus}$ ..... 3-24
3.4.5.4 Arhitectura lui $P_{map}$ - procesor de mapare a adresei ..... 3-25
3.4.5.5 Descrierea unui context în $K_{map}$ ..... 3-27
3.4.5.6 Structura Link şi a magistralei intercluster ..... 3-28
*3.4.6 Comunicaţii prin transmitere de mesaje ..... 3-28*
3.4.6.1 Protocolul magistralei intercluster ..... 3-28
3.4.6.2 Tranzacţia de mesaje interclustere (Figura 3.30) ..... 3-30
*3.4.7 Mecanismul de acces bazat pe capabilităţi ..... 3-30*
3.4.7.1 Modul de adresare la memorie a unui proces ..... 3-31
3.4.7.2 Generarea adresei virtuale ..... 3-32
3.4.7.3 Translatarea adresei procesor în adresă virtuală ..... 3-33
3.4.7.4 Translatarea adresei virtuale în adresă fizică ..... 3-34
3.5 PROGRAMAREA SISTEMELOR MULTIPROCESOR ..... 3-34
*3.5.1 Tipuri de sisteme de operare pentru sisteme multiprocesor ..... 3-35*
3.5.1.1 Sisteme de operare de tip master-slave ..... 3-35
3.5.1.2 Sisteme de operare distribuite ..... 3-35
3.5.1.3 Sistemul de operare cu supervizor flotant ..... 3-36
*3.5.2 Sisteme de operare pentru Cm* ..... 3-36*
3.5.2.1 Specificaţii de proiectare ..... 3-36
3.5.2.2 Sistemul de operare StarOS ..... 3-37
*3.5.3 Programarea sistemelor multiprocesor ..... 3-41*
*3.5.4 Aspecte privind programarea lui CM* ..... 3-43*
3.5.4.1 Operaţii cu capabilităţi şi costurile lor ..... 3-44
3.5.4.2 Facilităţi de dezvoltare soft în StarOS ..... 3-45
3.5.4.3 Aplicaţii pentru CM* ..... 3-48

# 3. Sisteme de prelucrare paralelă cu multiplicarea resurselor

**OBIECTIVE**

În acest capitol se prezintă o anumită clasă de sistem de prelucrare, denumită sistem cu multiplicarea resurselor. Având în vedere gama destul de largă de opţiuni de proiectare a SSP, există o mare varietate de modalităţi de prezentare a acestora. Programarea SSP de tip multiprocesor este un subiect vast care ar putea constitui obiectul unui curs separat.

În continuare se vor prezenta:

- Alternative de realizare a SSP cu multiplicarea resurselor;
- Structura şi Arhitectura sistemului multiprocesor Cm*, ca exemplu de SSP de tip multiprocesor;
- Programarea sistemului Cm*;
- Caracteristicile S.O. pentru sisteme multiprocesor;
- Alternative de realizare a S.O. pentru sisteme multiprocesor;
- Sistemul de operare StarOS pentru Cm*;
- Aspecte privind comunicaţia şi sincronizarea sistemelor multiprocesor.

## 3.1 Realizarea sistemelor de prelucrare paralelă cu multiplicarea resurselor

Există două modalităţi de introducere a paralelismului la nivel hardware:

1°. **Banda de asamblare ("pipeline");**

2°. **Multiplicarea spaţială a resurselor.**

De fapt *o bandă de asamblare este un sistem cu multiplicarea resurselor - sistem cu multiplicare în secvenţă*, în sensul că diferite componente hardware intră în lucru unul după altul. *Diferenţa principală dintre o bandă de asamblare şi un sistem cu multiplicare spaţială este că într-o BA componentele de calcul multiplicate şi care operează în paralel efectuează subfuncţii diferite. Aceste componente înseriate efectuează apoi funcţiile dorite*.

Gradul de paralelism al unei BA este determinat de posibilitatea descompunerii operaţiilor în suboperaţii. Rezultă clar că, exceptând situaţiile în care operaţiile sunt foarte complexe, apare o limitare a gradului de paralelism într-o BA.

Avantajul paralelismului prin BA este acela că nu pune probleme deosebite privind comunicaţia între componente.

Pe de altă parte, structurile în BA şi interconectarea componentelor reflectă fluxul natural al datelor pentru operaţiile executate în ansamblu de componentele BA.

Problema comunicaţiei între componente pentru sisteme cu multiplicare spaţială se poate rezolva printr-o *reţea de comutare a datelor*. Aceasta poate fi o *reţea cu o structură* fixă sau o *reţea programabilă*. Reţeaua cu structură fixă are dezavantajul că poate să fie ineficientă pentru diferite programe executate de sistem. Reţeaua programabilă are un cost foarte ridicat şi dificultăţi tehnologice de implementare, dar asigură o flexibilitate mult mai ridicată în configurarea sistemului corespunzător unei probleme.

Totuşi dezvoltarea sistemelor cu multiplicare spaţială a resurselor apare necesară datorită limitărilor proprii structurilor în BA.

### 3.1.1 Alternative pentru sistemele cu multiplicare spaţială

1. Un sistem cu multiplicare spaţială cu un număr mic de procesoare puternice are avantajul simplificării problemelor de comunicaţie (ex: se poate face printr-o memorie comună).

2. Pentru sisteme cu multiplicare spaţială cu un număr mare de procesoare caracterul secvenţial al comunicaţiilor prin memorie comună sau prin magistralele sistemului reprezintă o gâtuire a sistemului care-i limitează performanţele.

3. În ceea ce priveşte reţelele cu comutare ca infrastructură de comunicare în sistemele cu multiplicare spaţială sunt posibile doua tipuri de structuri.

#### 3.1.1.1 Sisteme cu reţea de comutare fixă

Sistemele cu reţea de comutare fixă sunt caracterizate prin faptul că odată cu creşterea numărului de procesoare creşte şi distanţa între acestea iar viteza de transfer scade ceea ce conduce la scăderea productivităţii sistemului.

La sistemele cu reţea de comutare programabilă aceste neajunsuri sunt limitate dar costul acestei reţele creşte cu pătratul numărului de componente interconectate.

Componentele principale ale sistemului sunt:

- $M_1, M_2 ... M_m$ = module de memorie
- $P_1, P_2 ... P_n$ = procesoare
- reţele de comunicare

Se poate observa paralelismul memoriilor ceea ce crează posibilitatea eliminării gâtuirii principale din sistemele clasice. Se observă că sistemul are o reţea de procesoare care pot comunica atât cu memoria cât şi între ele printr-o reţea de comunicare. Se observă că există o reţea de comunicare M-P (Figura 3.1.)

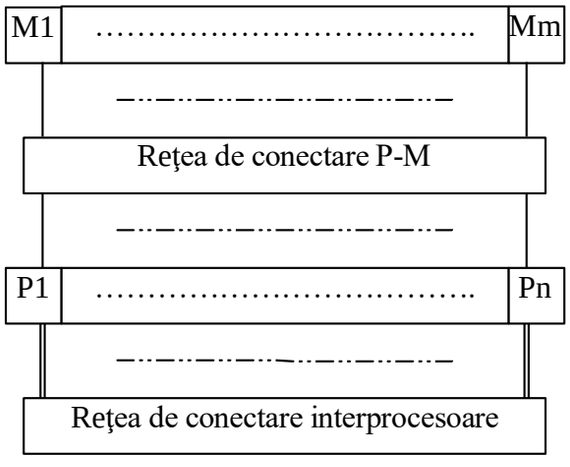



Figura 3.1

Considerând reţeaua de comunicare interprocesoare se observă că există două posibilităţi:

- putem avea un număr fix (în general mic) de conexiuni ce pot fi realizate între procesoare;
- în caz general putem avea asigurate n! căi de comunicare între procesoare.

În ceea ce priveşte reţeaua de interconectare memorie-procesoare există de asemenea două posibilităţi:

- reţeaua să asigure comunicarea identică. Fiecare procesor să comunice cu un singur modul de memorie, modulul propriu;
- reţeaua să asigure comunicarea fiecărui procesor cu oricare modul de memorie.

Se pune problema măsurării diferenţelor dintre diversele sisteme.

Această metrică ar trebui să cuprindă:

- numărul şi puterea procesoarelor. Pot fi de exemplu sisteme cu procesoare puţine dar foarte puternice;
- complexitatea reţelei de interconectare - depinde direct de punctul precedent dar şi de flexibilitatea ce se doreşte;
- distribuirea controlului la fiecare procesor. Această măsură ne arată unde se află sistemul nostru faţă de un sistem centralizat şi respectiv faţă de un sistem distribuit;
- metodele de control ale sistemului. Există cel puţin trei metode de control mai importante şi care generează tipuri de sisteme de tip MS:
  - control prin flux de comenzi (control flow);
  - control prin flux de date (data flow │ data driven);
  - control prin flux de cereri (reduction │ demand driven).

Metodele de control implementate constituie diferenţa principală între structurile cu prelucrare paralelă.

### 3.1.1.2 Controlul prin flux de date ( data driven ).

În cazul controlului prin flux de date succesiunea operaţiilor nu este controlată de valorile registrului contor program, ci de disponibilitatea operanzilor. Un program pentru un calculator "data

flow" este de fapt un grafic orientat care are în noduri operatori şi prin care circulă pachetele de date.

De exemplu expresia (A + B) * (C + D) poate fi reprezentată sub formă de arbore aşa cum se arată în figura 3.2. Fiecare operaţie este iniţiată de îndată ce operanzii sunt disponibili.

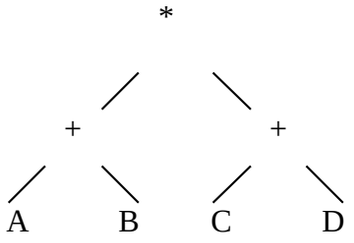



Figura 3.2. Arborele binar al unei operaţii aritmetice

O caracteristică a acestor structuri este faptul că *paralelismul este total asincron,* controlat de un mecanism "data driven".

O altă caracteristică constă în faptul că paralelismul, respectiv toate dependenţele de date, sunt cuprinse în graful programului, ceea ce înseamnă că nu este necesară declararea explicită a paralelismului el fiind încapsulat în graful programului ceea ce crează posibilitatea executării acestuia pe o structură multiprocesor.

Un calculator "data flow" trebuie să conţină:

- una sau mai multe unităţi de execuţie;
- unul sau mai multe module de memorie pentru date;
- o memorie (de obicei separată) care conţine o reprezentare a grafului programului;
- o unitate de excepţii care rezolvă diferite situaţii apărute accidental.

### 3.1.1.3 Controlul prin flux de cereri (demand driven).

Acest tip de control *se caracterizează prin faptul că paralelismul este exploatat fără a avea un control explicit*. Ca bază teoretică acest tip de control are analiza funcţiilor şi calculul lambda. Programele sunt şiruri de expresii sau aşa zişii arbori de analiză gramaticală - "parse trees". *Controlul se face prin flux de cereri care de fapt constă din execuţia acestor expresii prin reducerea expresiilor la operaţiile componente din acestea.*

*exp:* (A + B) * (C + D)

poate fi scrisă în forma poloneză inversă poate fi scrisă:

* + AB + CD

pentru a fi evaluată prin reducere succesivă.

Atât în calculatoarele "data flow" cât şi în calculatoare cu control prin flux de cereri există un aşa zis "overhead" organizatoric care până la urmă poate conduce la o *ineficienţă faţă de metodele de comandă prin flux de comenzi.*

Avantajul acestor două tipuri de structuri este că oferă o abstractizare a metodelor de programare şi de exemplu oferă un suport foarte bun pentru implementarea limbajelor funcţionale, "functional language".

Dezavantajul celor două tipuri de structuri constă în aceeea că este nevoie de comunicaţii atât pentru date cât şi pentru programe ceea ce constituie o limitare datorită creşterii "overhead"-ului de comunicaţie.

Obs: Cu cât numărul de procesoare este mai mare cu atât clasa de aplicaţii care va utiliza eficient acest sistem va fi mai mică.

### 3.1.1.4 Control prin flux de comenzi

Strategiile de control în sistemele cu MS conduc la diferite tipuri de structuri, ca de exemplu SIMD, MIMD.

O altă caracteristică importantă este *distribuirea controlului*, adică controlul local raportat la controlul global.

Altă caracteristică este *distribuirea puterii de prelucrare* - numărul de procesoare poate fi cuprins între zeci şi zeci de mii de procesoare cu performanţe cuprinse într-o gamă foarte largă.

În ceea ce priveşte exploatarea paralelismului, cuantificarea cantităţii de paralelism este imposibilă (nedecidabilitatea paralelismului maxim) - nu există procedură care să spună dacă s-a găsit paralelismul maxim. Cantitatea de paralelism ce poate fi valorificată depinde de fiecare aplicaţie în parte.

**Distribuirea puterii de calcul**

Unul dintre aspectele cele mai importante în proiectarea unui sistem distribuit cu prelucrare paralelă este distribuirea puterii de calcul între componentele sistemului.

În ceea ce priveşte distribuirea puterii de prelucrare putem avea:

- sistem de tip multicalculatoare;
- reţele de procesoare.

La una dintre extreme putem considera calculatoare slab cuplate, de genul reţelelor locale de calculatoare care pot fi integrate în structuri paralele de tipul "hipercub", sau structuri heterogene de tip multiprocesor.

La cealaltă extremă se situează reţele formate dintr-un număr foarte mare de procesoare foarte simple.

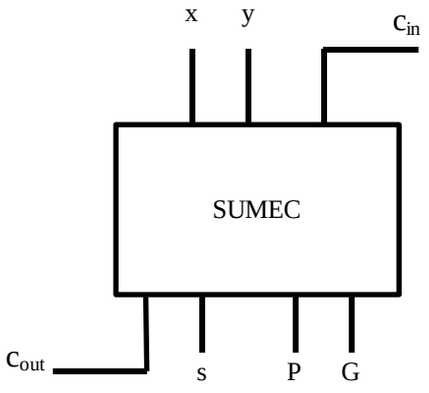
Să considerăm de exemplu un sumator elementar complet pe un bit (SUMEC). Tabelul de adevăr şi ecuaţiile sunt prezentate în tabelul de mai jos:

| x | y | $C_{in}$ | S | $C_{out}$ | P | G |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 |

$S = x \oplus y \oplus C_{in}$ $\quad P = x \oplus y$

$C_{out} = C_{in}(x \oplus y) + xy$ $\quad G = x.y$

Schema sumatorului elementar complet:




Se observă că pot fi aplicate diferite modalităţi de creştere a vitezei de prelucrare a acestei reţele de procesoare simple. O abordare similară se poate face şi pentru înmulţire.

O primă soluţie constă în legarea în cascadă a mai multor sumatoare elementare pentru a obţine un sumator paralel cu transport succesiv. Considerând $t_{SE}$ întîrzierea dată de două porţi NAND şi o sumă modulo doi care implementează un sumator elementar complet, adunarea serială a n biţi se va face cu o întîrziere totală de:

$T_s = nt_{SE}$

Pentru sumatorul cu transport succesiv:

$T_P = t_{SE} + 2(n-1)t_p$ unde $t_p$ este întîrzierea pe o poartă NAND.

Pentru un sumator paralel cu transport succesiv:

$T_{TS} = 2t_{SE} + (\log_2 n-1)t_a$ unde $t_a$ este întîrzierea pe unitatea de anticipare.

## 3.2 Procesoare de tip reţea (Array Processors)

Reprezintă principala modalitate de creştere a vitezei unui calculator prin MS.

Un procesor de tip reţea (PR) este o reţea sincronă de procesoare paralele. Ca structură, PR sunt formate din mai multe elemente de prelucrare interconectate între ele şi o unitate de comandă care controlează fluxul de instrucţiuni şi fluxul de date deci comunicaţiile între elementele de prelucrare. Având în vedere fluxul de instrucţiuni şi cel de date care reflectă structura PR, acestea sunt exemple de *structuri SIMD*. De fapt, sistemele SIMD includ mai multe tipuri de structuri din care fac parte şi:

- Procesoare de tip reţea (PR);
- Procesoare asociative (PA)

*PR se caracterizează prin faptul că memoria fiecărui element de prelucrare şi a unităţii de comandă sunt de tip RAM, iar la PA memoria este de tip CAM* (Content Addressable Memory).

### 3.2.1 Organizarea unor structuri SIMD de tip PA

Structurile SIMD de tip PA se caracterizează prin următoarele:

- au o unitate de comandă cu o memorie proprie;
- există o magistrală de comenzi între unitatea de comandă şi elementele de prelucrare;
- există o magistrală de date care leagă toate elementele de memorie;
- există o reţea de interconectare a elementelor de prelucrare;
- există reţea de interconectare între echipamentele de prelucrare comandate de unitatea

de comandă;

- există o magistrală externă folosită pentru conectarea la:
  - subsisteme de I/E;
  - memorie externă;
  - sistem gazdă (host system) care va folosi ca un coprocesor procesorul asociativ.

În Figura 3.3 am prezentat o *configuraţie SIMD de tip I*.

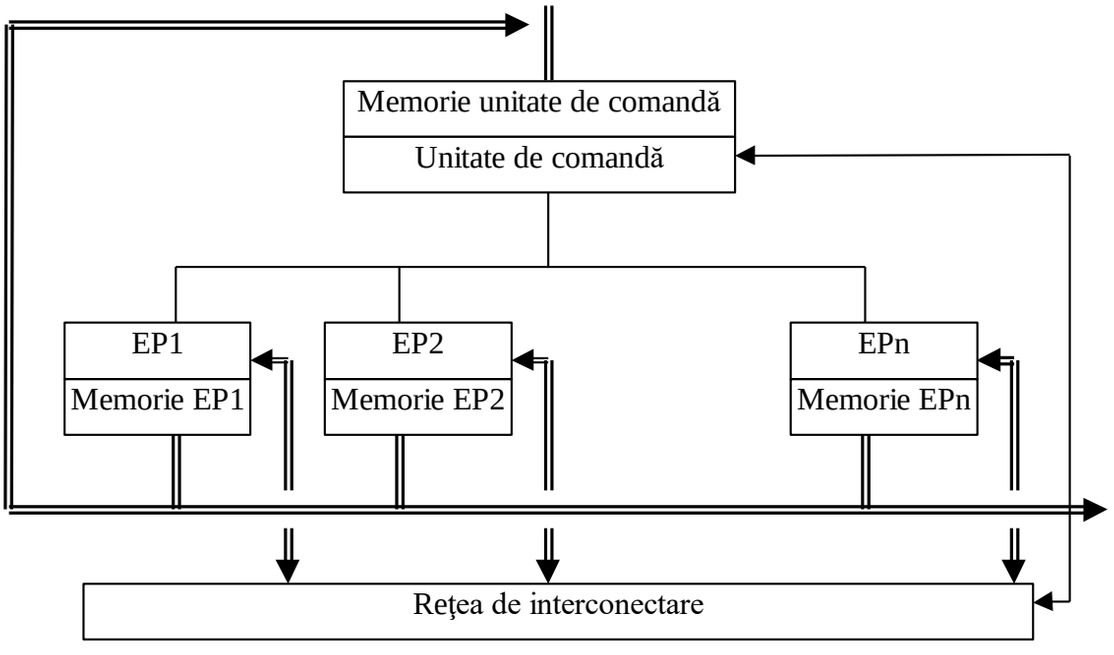



Figura 3.3 Configuraţie SIMD de tip I

Fiecare EP are propria sa memorie care este accesibilă şi unităţii de comandă şi sistemului gazdă.

Programele sunt încărcate din exterior prin intermediul magistralei externe în memoria unităţii de comandă.

Un exemplu de sistem SIMD este ILLIAC IV care poate efectua 80 - 100 MFLOPS.

Unitatea de comandă decodifică instrucţiunile utilizatorului şi determină locul unde se vor executa aceste instrucţiuni.

Exemplu: Unitatea de comandă execută toate instrucţiunile de control şi instrucţiunile de prelucrare a operanzilor de tip scalar. Înstrucţiunile vectoriale sunt distribuite la elementele de prelucrare asigurându-se astfel paralelismul spaţial. Datele distribuite se pot încărca din memoria elementelor de prelucrare prin magistrala de date sau pot fi transmise prin "broadcast" pe magistrala de comenzi direct la elementele de prelucrare.

Există un vector de mascare care asigură controlul individual al fiecărui element de prelucrare. Astfel, fiecare element de prelucrare poate să fie activat/dezactivat la nivel de ciclu instrucţiune.

Legătura pe care elementele de prelucrare o au cu sistemul gazdă asigură printre altele:

- gestiunea resurselor întregului sistem (Host + Unitate comandă + EP)
- intrări/ieşiri (gestiunea echipamentelor periferice şi supervizarea operaţiilor de I/E).

Un alt tip de structură - *configuraţie SIMD de tip II* se caracterizează printr-o separare a elementelor de prelucrare de memoriile acestora, aşa cum se arată în Figura 3.4.

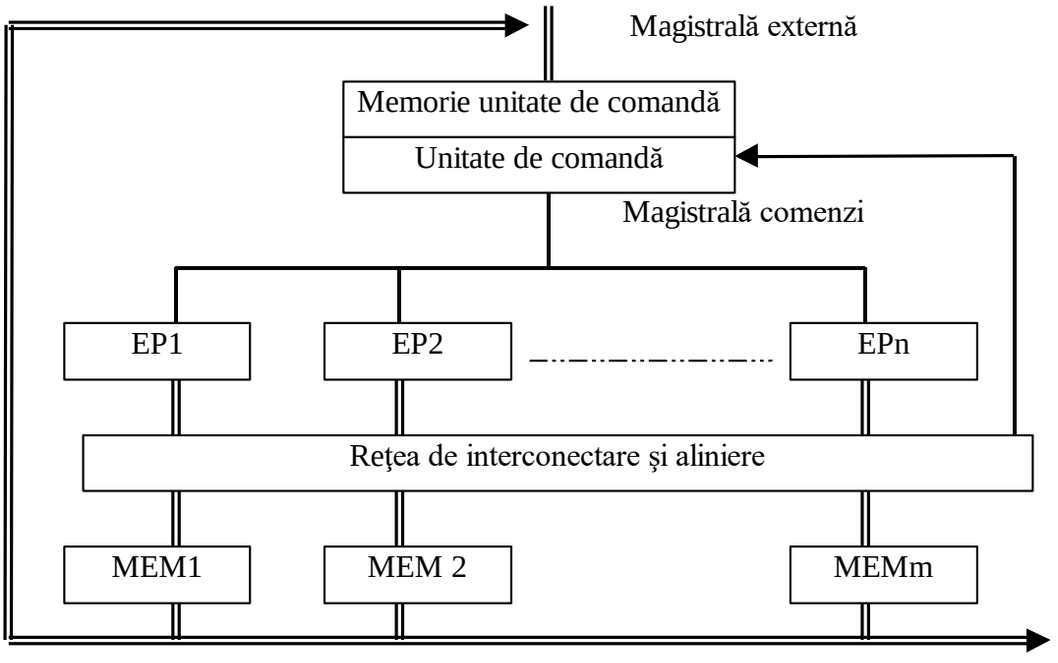



**Figura 3.4** Configuraţie SIMD de tip II

Exemplu: BSP (Burroughs Scientific Processor) are o viteză aprox. 50 MFLOP.
Structura acestora se caracterizează prin:

- memoria este paralelă spre deosebire de structura anterioară;
- memoria este modulară;
- reţeaua de interconectare a procesoarelor a fost înlocuită cu reţeaua de conectare a memoriilor care asigură şi conectarea dintre procesoare şi memorii;
- număr de procesoare diferite de numărul de module de memorie;
- unitatea de comandă are o memorie proprie de 256 Kcuvinte (la BSP) cu ciclul de acces de 160ns;
- unitatea de comandă este dotată cu un procesor scalar care are o structură de tipul cu registre generale având 16 registre generale a 48 biţi;
- frecvenţa ceasului la BSP este de 80ns, 12,5MHz iar secţiunea de reţea a calculatorului asigură citirea operanzilor din memoria modulară - paralelă, alinierea operanzilor cu UAL-urile din elementele de prelucrare, execuţia operaţiilor specificate de fluxul de instrucţiuni, alinierea rezultatelor cu memoria, memorarea rezultatelor în memoria paralelă.

Operaţii cu vectori implementate la BSP

1°. Operaţii vector-scalar

$A \leftarrow B$ op s.

- Reducere în simplă precizie

$s \leftarrow A_0$ op $A_1$ op ... op $A_{N-1}$ $s \leftarrow$ ¤/A

- Reducere în dublă precizie (operanzii sunt în dublă precizie)
- Operaţie secvenţă

$A_i \leftarrow A_1 + A_2 + ... A_i$

- Operaţie de compresie a unui vector
- Operaţie de expandare a unui vector
- Operaţie "merge"

$A \leftarrow B$ or C

- Operaţie de citire aleatoare

$A \leftarrow B(I)$

- Operaţie de scriere aleatoare

$A(I) \leftarrow B$

- Produs scalar

$s \leftarrow A_1 * B_1 + A_2 * B_2 + ... + A_n * B_n$

- Operaţie de recurenţă

$A_1 \leftarrow C_1$

$A_i \leftarrow A_{i-1} * B_i + C_i$

2°. Operaţii de transfer de date - fac schimbul de date între memoria unităţii de comandă şi memoria paralelă.

O altă maşină de tip RP este *Connection Machine (CM)* proiectată la M.I.T. ca o maşină de prelucrare paralelă pentru aplicaţii de IA implementate în VLSI (Figura 3.5). Este o maşină de tip bit-serial. Specific acestei maşini este reţeaua de conectare foarte complexă asigurând posibilitatea programării unei conexiuni între oricare două procesoare. Fluxul datelor este similar cu cel din reţele locale prin comutare de pachete. Datele sunt sub formă de pachete, fiecare pachet având adrese destinaţie. Topologia sistemului CM este cea de un cub cu 12 dimensiuni.

Specific lui CM este că algoritmul de rezolvare a problemei se mapează pe o structură virtuală de procesoare cu multiplicare spaţială.

CM a fost implementată de "Thinking Machine Corporation" cu 65536 procesoare bit-serial. Fiecare procesor are 4K de memorie. Unitatea de comandă este microprogramată iar CM este utilizat ca un coprocesor pentru un calculator gazdă.

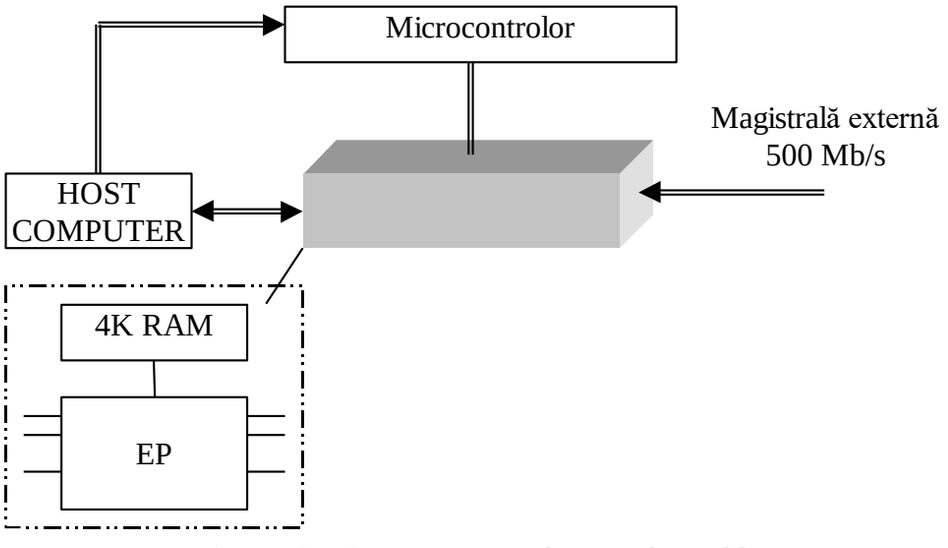



**Figura 3.5** Connection Machine - schema bloc

Procesoarele sunt implementate VLSI în tehnologie CMOS. Consumul este de 1W/4MHz, capsula de 48 pini şi 50000 tranzistori. O placă conţine 512 procesoare şi memorie externă. Întreg sistemul disipă 12KWatt.

Un element de prelucrare (Figura 3.6) este realizat dintr-o UAL cu două ieşiri având 256 de funcţii.

Programarea se face în CM - LISP sau LISP - SIMD.

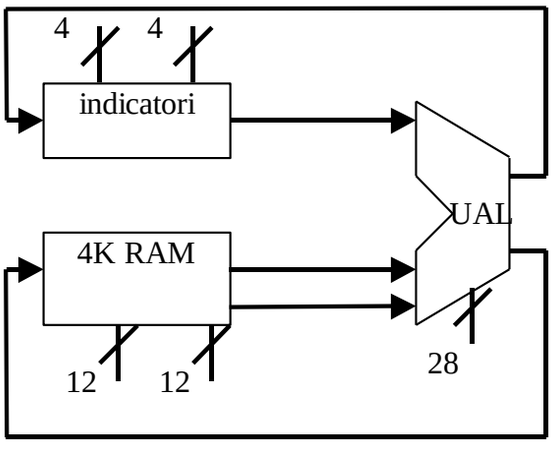



**Figura** 3.6 Element de prelucrare

O altă structură de tip RP de tip SIMD este o *reţea de transputere*. Transputerul este un microprocesor apărut în Anglia în 1985 realizat de INMOS. Sunt cel puţin trei variante de transputere:

T 212 - magistrală de date şi 16 biţi adresă

T 414

T 800 - pe 32 de biţi

**Caracteristici T 414:**

- procesor de tip RISC cu frecvenţa tactului de 20MHz;
- magistrală de date şi adrese de 32 biţi;
- 25% din arie este ocupată de partea de prelucrare, 25-30% este ocupată de 2Ko de memorie RAM static, 25% pentru interfaţa cu magistrala I/O şi 20-25% pentru legăturile specifice transputerului (LINKS) care reprezintă nişte legături de mare viteză (10Mb/s);
- 4 legături seriale.

Un procesor de tip transputer din punct de vedere al legăturilor apare ca în figura 3.7 având 4 legături ce ar putea fi folosite pentru conectarea la cei 4 vecini.

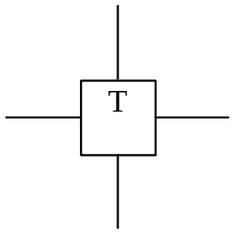



**Figura** 3.7 Legături transputer

**Caracteristici T800:**

- Procesor RISC cu 4 Ko memorie RAM statică,
- magistrală de date şi adrese pe 32 biţi,
- controler pe cele 4 legături cu exteriorul;
- frecvenţă de tact 20 sau 30MHz;
- are incorporat un procesor de virgulă mobilă pe 64 biţi.

O caracteristică importantă la T414 şi T800 o constituie
comunicaţia cu circuitele asociate fixate în exterior care se face la o viteză fixată (ex: 5MHz) pe toate componentele externe asociate.

Schema bloc a unui astfel de transputer (Figura 3.8).

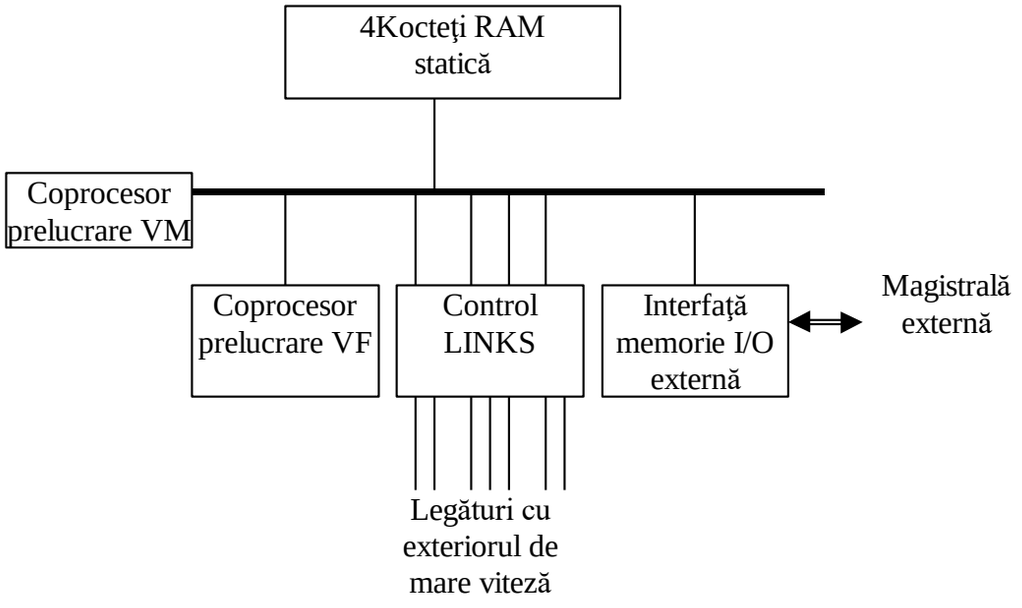



**Figura** 3.8. Schema bloc transputer

Legăturile de mare viteză sunt seriale cu un protocol propriu.

S-au realizat aplicaţii mixte între transputere şi alte procesoare (ex: cu 88000, I860).

Pentru programarea transputerelor se foloseşte limbajul OCCAM care este un limbaj de tip paralel. Se afirmă că odată scris programul pentru OCCAM nu interesează numărul de transputere. Ideea este că un program în OCCAM este alcătuit din mai multe procese care pot funcţiona paralel. Programatorul le defineşte şi totodată defineşte relaţia de precedenţă dintre procese (paralelismul este specificat de programator). În momentul execuţiei unui program format din mai multe procese sistemul de exploatare face planificarea în execuţie a proceselor în funcţie de relaţiile de precedenţă stabilite de programator.

O configuraţie de tip "array" realizată cu transputere este următoarea (Figura 3.9):

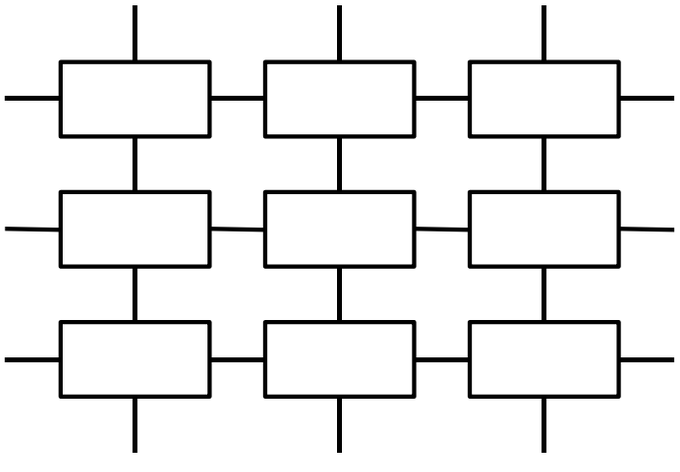


**Figura 3.9** Configuraţie tip "array" realizată cu transputere

### 3.3 Sisteme paralele de tip multiprocesor

### 3.3.1 Granularitatea paralelismului

Pentru a exploata paralelismul la nivel de task-uri se utilizează sistemul multiprocesor în diferite configuraţii.

În general un *sistem multiprocesor* este prin definiţie o colecţie de procesoare autonome, fiecare fiind capabil să execute un proces secvenţial la un moment dat.

Dacă task-urile executate de toate procesoarele sunt identice, rezultă o structură SIMD din punct de vedere funcţional.

Dacă sistemul suportă execuţia unei colecţii eterogene de taskuri din punct de vedere funcţional sistemul este de tip MIMD.

Există o confuzie între sistemul multicalculator şi sistemul multiprocesor.

*Un sistem multiprocesor* este prin definiţie un sistem de calcul unitar în care o colecţie de procesoare autonome funcţionează concurent şi cooperează la realizarea funcţiilor logice şi de calcul sub controlul unui executiv unitar şi distribuit.

*Un sistem multicalculator* constă din mai multe calculatoare autonome interconectate care comunică sau nu pentru realizarea prelucrărilor cerute.

Obs: În cadrul unui sistem multiprocesor, procesoarele sunt universale (nefiind dedicate funcţional pentru un anumit task).

Comunicaţia între procesoare în sistem multiprocesor se bazează pe următoarele principii:

1. Comunicaţie prin memorie comună
2. Comunicaţie prin transfer de mesaje

### 3.3.2 Caracteristici ale sistemului multiprocesor

**1°. Raportul performanţe/cost funcţie de complexitate**

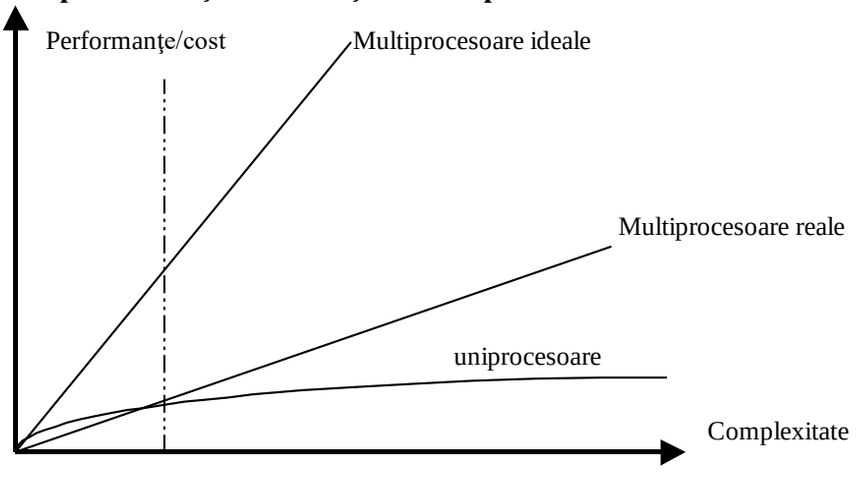



**Figura** 3.10 Raportul performanţe/cost în funcţie de complexitate

**2°. Structura eterogenă** -procesoarele nu trebuie neapărat să fie de acelaşi tip.

**3°. Modularitatea** -se asigură reconfigurabilitate, expandabilitate, mentenabilitate

**4° Fiecare procesor să poată funcţiona autonom ca un calculator universal** putând prelua taskurile altui procesor asigurând astfel degradarea lentă ("graceful degradation")

**5°. Organizarea memoriei** - trebuie să ţină cont de următoarele aspecte:

- fiecare procesor să aibă acces la întreaga memorie a sistemului;
- capacitatea de adresare a procesoarelor e în general mai mică decât spaţiul de memorie

al întregului sistem;
- asigurarea protecţiei memoriei pentru asigurarea integrităţii datelor.

**6°. Siguranţa în funcţionare** - la sistemele multiprocesor se defineşte prin alte două atribute:
- **disponibilitate** - capacitatea sistemului de a răspunde prompt la cererile utilizatorului.
- **integritate** - capacitatea sistemului de a menţine consistenţa stării întregului sistem în condiţiile apariţiei unor defecte.

**7°. Reconfigurabilitatea** - există mai multe modalităţi de a o aborda în sisteme multiprocesor, de exemplu:
- **reconfigurabilitatea structurală** - ca proces de refacere a stării sistemului la apariţia unor defecte; acestea se realizează prin distribuirea structurală a componentelor şi distribuirea arhitecturală a funcţiilor;
- **reconfigurabilitatea arhitecturală şi structurală** - conduce la schimbarea caracteristicilor de bază ale sistemului, de exemplu:
  - lungimea cuvântului de memorie,
  - lungimea vectorului de prelucrare în UAL,
  - lungimea registrelor,

  în funcţie de cerinţele de prelucrare.

**8°. Programarea** - pentru programarea sistemului multiprocesor trebuie avute în vedere următoarele:
- decompoziţia algoritmilor: se poate face manual sau semiautomat;
- controlul şi sincronizarea proceselor concurente, care sunt diferite de sistemele multiprogramate;
- planificarea pentru execuţie a taskurilor, constând în maparea taskurilor pe structura multiprocesor pentru a asigura optimizarea utilizării resurselor;
- rezolvarea structurilor de blocare.

Având în vedere structura de interconectare a procesoarelor există două modele importante de sisteme multiprocesor:

a) multiprocesoare cu cuplaj strîns (interconectate)

b) multiprocesoare cu cuplaj slab (interconectate)

### 3.3.3 Multiprocesoarele strîns interconectate

Acestea comunică printr-o memorie principală partajată şi o caracteristică a lor constă în faptul că rata de transfer între procesoare este de acelaşi ordin de mărime cu rata de transfer a memoriei.

Fiecare procesor are (poate avea) o memorie locală între procesor şi memorie existând o conectivitate completă implementată prin:
- prevederea unor reţele de conexiune între procesoare şi memorie (similară cu un sistem SIMD sau reţele de procesoare)
- implementarea unei memorii multiport.

O limitare în dezvoltarea acestor sisteme prin creşterea numărului de procesoare o constituie degradarea performanţelor, accesul concurent al mai multor procesoare la acelaşi modul de memorie. Gradul conflictelor la accesul concurent poate fi redus crescând gradul de întreţesere al acceselor concurente, deci micşorând granularitatea structurilor de date accesate şi crescând panta de acces la memorie.

O altă limitare este introdusă de reţeaua de interconectare procesor-memorie.

### 3.3.4 Multiprocesoare slab cuplate

În aceste structuri nu apar conflicte de acces la memorie ca la sistemele cu cuplaj strîns deoarece fiecare procesor este prevăzut cu resurse locale, intrări/ieşiri şi memorie cu capacitate suficient de mare pentru a asigura majoritatea acceselor la instrucţiuni şi date.

Procesorul care asigură atât prelucrările aritmetice şi logice cât şi funcţiile de comandă locale împreună cu memoria locală, interfeţele şi echipamentele de I/O, formează un *modul calculator (Cm).*

Procesele care se execută pe aceste Cm diferite, comunică între ele printr-un schimb de mesaje folosind un sistem de transfer de mesaje (MTS - Message Transfer System). Gradul de cuplare între Cm-uri poate fi foarte slab conducând în acest caz la sisteme distribuite mai degrabă decât la sisteme multiprocesor.

Gradul de cuplare este determinat în special de topologia căilor de comunicaţie asociate sistemului de transfer de mesaje. Eficienţa sistemelor slab cuplate este direct dependentă de interacţiunile dintre taskuri. Spre deosebire de sistemele slab cuplate, structura sistemelor puternic cuplate poate asigura un grad ridicat de interacţiune între taskuri în condiţii de performanţe ridicate.

Exemplu de procesor neierarhizat slab cuplat (Figura 3.12)

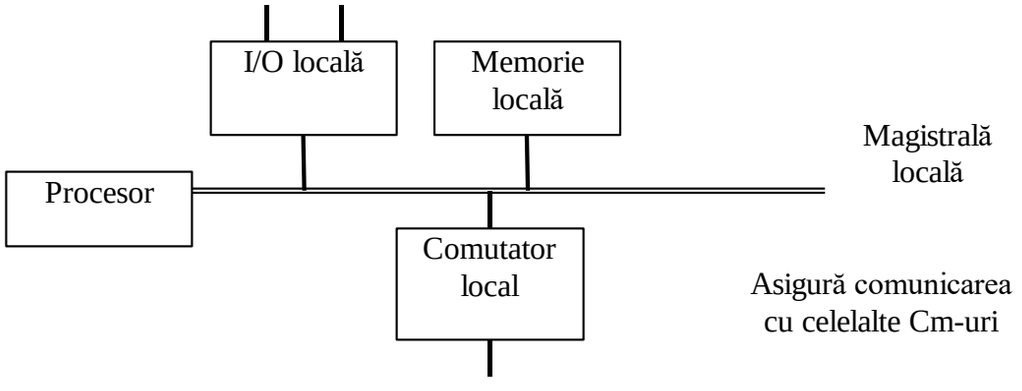



**Figura 3.11** Schemă bloc Cm

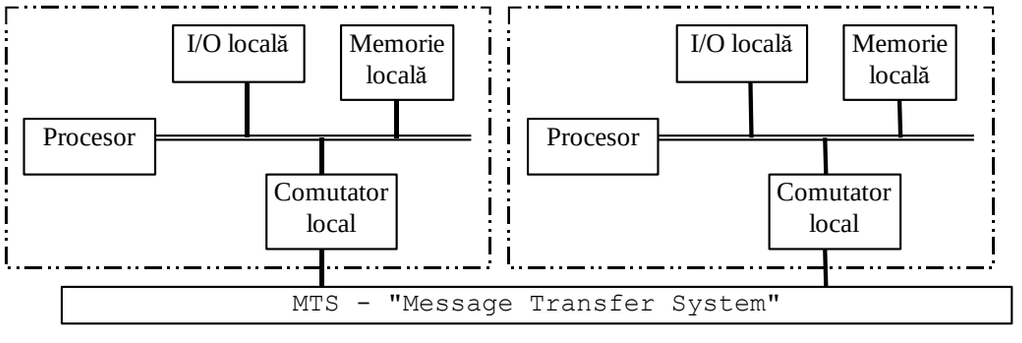



**Figura 3.12** Procesor neierarhizat slab cuplat

În această structură de sistem de cuplare slabă neierarhic, dacă cererile de la două sau mai multe module calculator intră în coliziune, în cazul accesului unui segment fizic din MTS, arbitrul asociat va selecta una din cererile simultane în concordanţă cu o disciplină de rezolvare a priorităţilor selectată.

Pînă la servirea unei cereri, cererile simultane vor fi întîrziate astfel încât Comutatorul local poate dispune de o memorie de mare viteză care poate fi utilizată pentru stocarea mesajelor transferate. De asemenea MTS poate dispune de o memorie tampon rapidă de comunicaţie accesibilă

procesoarelor modulelor calculator.

Un Cm poate fi implementat în VLSI sub forma de circuit integrat.

În sistemele slab cuplate neierarhizate sistemul de transfer al mesajelor poate fi o magistrală unică ca de exemplu MULTIBUS sau UNIBUS sau un sistem de interconectare cu memorie partajată realizată printr-un set de module de memorie şi o reţea de interconectare procesor-memorie sau o memorie multiport.

Prin utilizarea unei memorii de comunicaţie fie că este concentrată sau distribuită se consideră nişte porturi logice de comunicaţie care pot fi accesate de procesoare.

Procesele pot comunica cu alte procese alocate aceluiaşi procesor sau cu taskuri alocate altor procesoare (Figura 3.13).

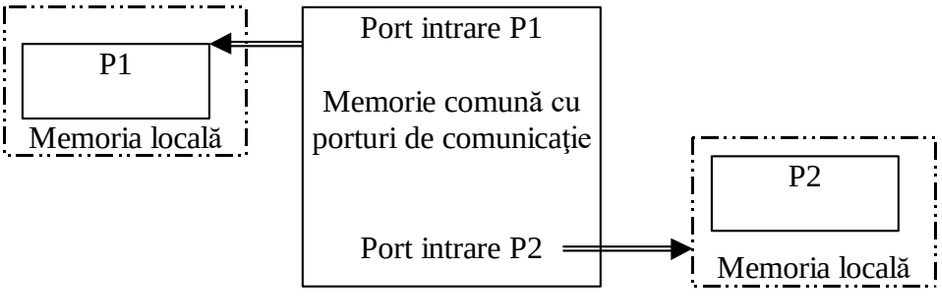



**Figura 3.13** Comunicaţiile între procese

## 3.4 Arhitectura şi structura sistemului Cm*

### 3.4.1 Structura lui Cm*

Cm* este un sistem multiproesor slab cuplat. Este format din module calculator bazate pe LSI 11. Structura funcţională este formată din 50 procesoare LSI 11 având următoarea structură (Figura 3.14)

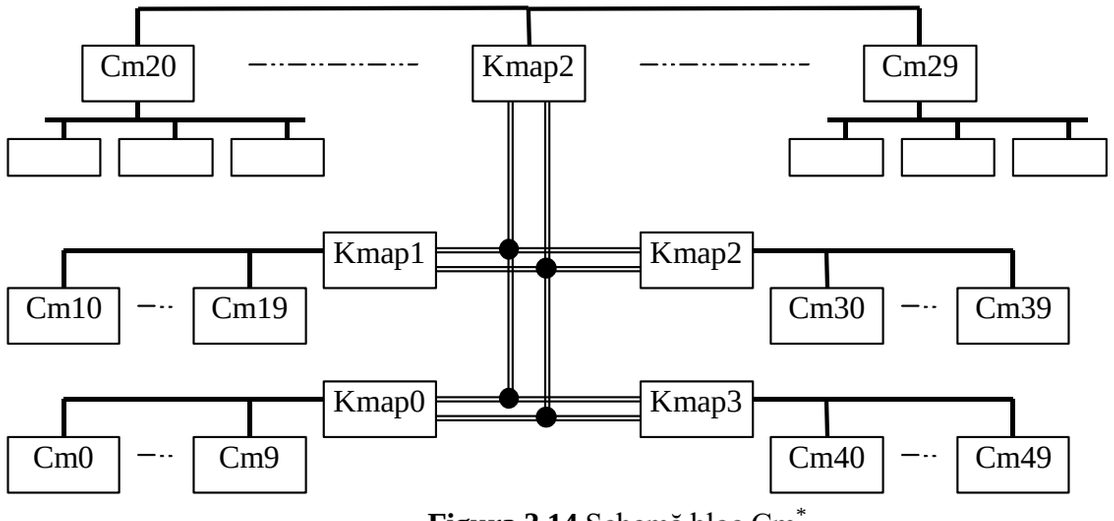



**Figura 3.14** Schemă bloc Cm*

Fiecare $K_{map}$ are două conexiuni de legătură cu exteriorul şi o magistrală locală (Figura 3.15).

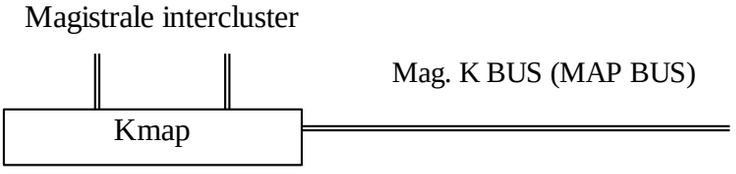



**Figura 3.15** Conexiuni $K_{map}$

Mai există o magistrală pentru fiecare Cm care poate avea memorie şi diferite echipamente periferice.

Fiecare Cm include un comutator local analog CCA-ului prezentat, denumit aici Slocal (Switch local).

Astfel, un Cm apare în structura prezentată în Figura 3.16:

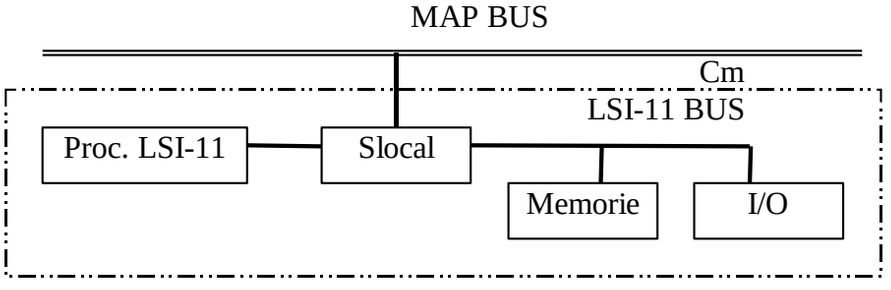



Figura 3.16 Structura Cm

Modulele de tipul Cm interconectate printr-o magistrală de tip MAP BUS formează un CLUSTER (Figura 3.17):

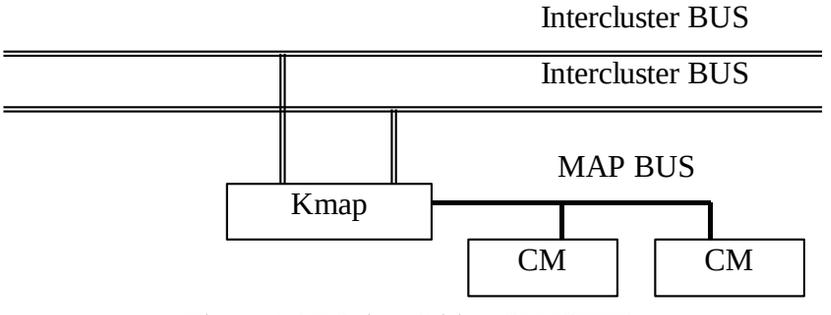



Figura 3.17 Schemă bloc CLUSTER

Au fost experimentate şi alte configuraţii de sistem (Fig 3.18):

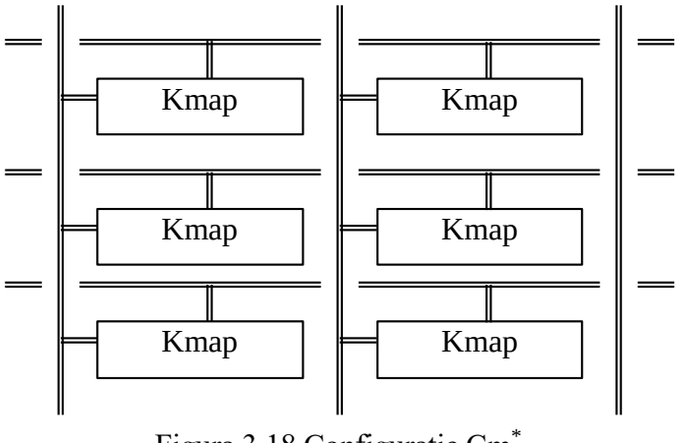



Figura 3.18 Configuraţie Cm*

### 3.4.2 Accesul la resursele locale

Switch-ul local din fiecare Cm interceptează şi dirijează cererile de acces de la procesorul local la memorie şi I/E locale sau la distanţă (prin intermediul lui MAPBUS).

De asemenea Slocal acceptă cererile la resursele locale (memorii, I/O) din partea altor module externe. Pentru a realiza filtrarea cererilor locale şi externe, Slocal asigură o translatare a adreselor având în vedere faptul că memoria este paginată în pagini de câte 4Ko şi că magistrala de adrese la LSI 11 este de 18 biţi. De asemenea în cuvântul de stare program la LSI 11 există un bit pentru selectarea contextului folosit în cazul nostru pentru selectarea spaţiului în tabele de mapare (Figura 3.19).

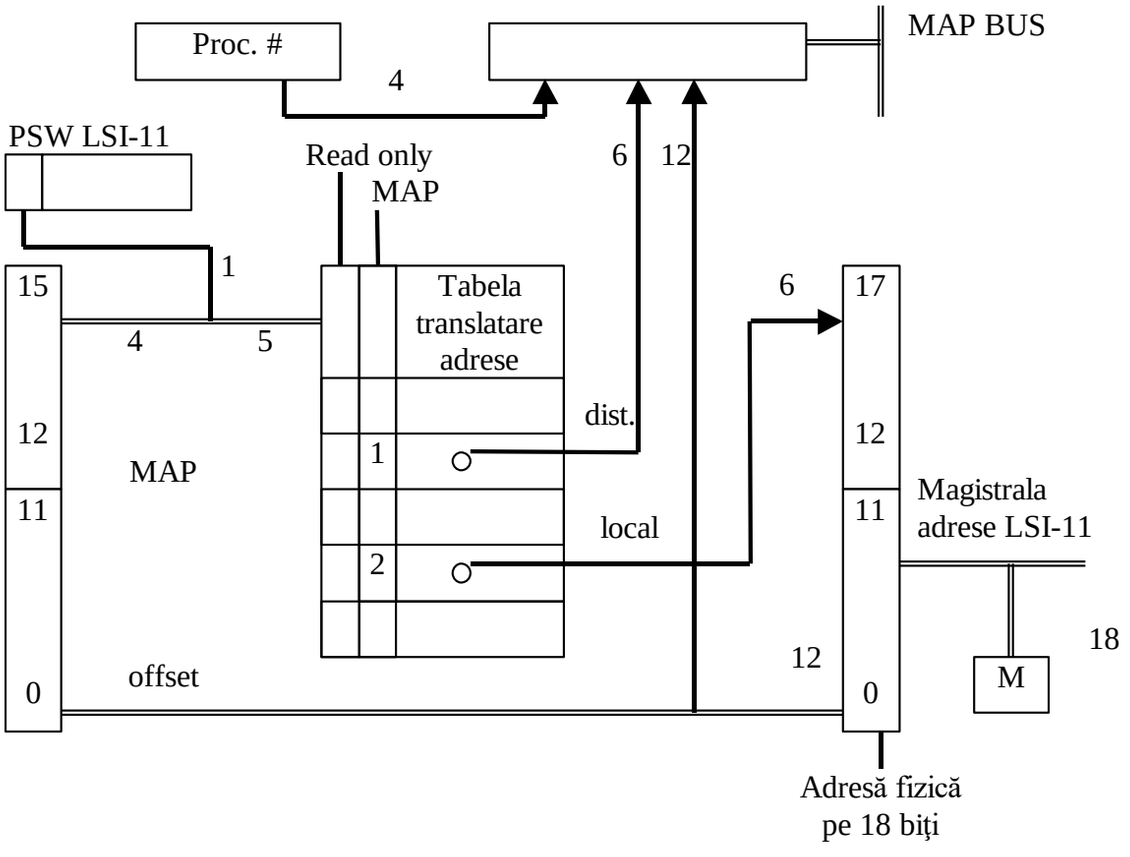



Figura 3.19 Mecanismul de adresare pentru referinţe la memoria locală

Se observă că se poate implementa un sistem multiprocesor cu memorie globală formată din memoriile locale.

Slocal asigură translatarea adreselor folosind cei mai semnificativi 4 biţi (12-15) ai adresei generate de procesor împreună cu bitul X din PSW-ul lui LSI 11. Cu aceşti biţi se face accesul la tabele de mapare, pe de o parte pentru a stabili dacă adresa este locală (0 - în primul bit) sau la distanţă (1 - în primul bit). Adresa virtulă a unei referiri nelocale este formată prin concatenarea câmpurilor dată de partea cea mai puţin semnificativă a adresei procesorului, adresa paginii de la distanţă dată de tabela de mapare a adreselor şi de numărul de identificare al procesului sursă.

Slocal anunţă $K_{map}$-ul asociat la care este legat că doreşte un transfer şi $K_{map}$ citeşte prin MAP-BUS adresa virtuală trimisă asamblată de către Slocal. Mai multe Cm-uri pot fi conectate la un MAP-BUS partajând acelaşi $K_{map}$. $K_{map}$ este un procesor de mare viteză având ca funcţie principală maparea adreselor virtuale în adrese fizice şi dirijarea datelor între switch-urile locale ale Cm-urilor din acelaşi cluster sau la distanţă. Modulele calculator sunt conectate în "clustere" ierarhice prin două nivele de magistrale. Conectarea în clustere şi deci obţinerea unor structuri ierarhizate facilitează cooperarea

Cm-urilor din acelaşi cluster şi asigură facilităţi hardware pentru execuţia în regim de multiprelucrare a unui grup de procese cu cuplaj strîns care cooperează între ele. Orice referinţă nelocală la distanţă este recepţionată de $K_{map}$ din clusterul respectiv care apoi intră în conexiune cu $K_{map}$-ul asociat clusterului din care face parte memoria ţintă.

Performanţele sistemului sunt direct determinate de modalităţile de interconectare a clusterelor, de vitezele de comunicaţie interclustere, de vitezele procesoarelor $K_{map}$.

Deoarece două $K_{map}$ nu trebuie să fie neapărat interconectate direct rezultă o structură de interconectare foarte complexă.

În ceea ce priveşte performanţele la sistemul realizat se apreciază că dacă se ia ca bază timpul de acces la o memorie locală de către un procesor local rezultă:

- accesul la o memorie din acelaşi cluster - de 3 ori mai mare
- accesul la o memorie din alt cluster dar care sunt conectate - de 3 ori mai lent.

Spaţiul total de memorie (spaţiul virtual de adresare) este de 28 biţi (256Mo). Acest spaţiu e divizat în segmente de maximum 4Ko.

Programele când adresează memoria se referă la segmente indirect prin intermediul capabilităţilor care sunt grupuri de două cuvinte de 16 biţi conţinând numele global al segmentului şi specificând drepturile de acces la segment.

Memoria unui Cm este împărţită în 16 pagini de câte 4Ko.

Este asigurat un mecanism care permite unui program să asocieze orice capabilitate pentru oricare segment la care are acces cu oricare pagină din spaţiul de adresare imediată.

### 3.4.3 Mecanismul de referinţe intermodule

Dacă bitul MAP din Slocal (tabela de translatare) este 1, referinţa este nelocală şi această referinţă este rezolvată de către $K_{map}$. Segmentul referit poate să fie local clusterului $K_{map}$ respectiv sau poate fi localizat în alt cluster. Dacă segmentul este local are loc o referinţă intercluster şi în acest caz având în vedere că toate interfeţele procesorului local şi $K_{map}$ cu magistralele sunt interfaţate cu registre rapide, $K_{map}$ va folosi informaţia din aceste registre rapide pentru a face translatarea adreselor din adresă procesor (aşa cum este emisă de un LSI 11) în adresă virtuală şi apoi adresă fizică pentru a face accesul efectiv la memorie. Astfel $K_{map}$ poate translata rapid din numărul paginii referite de procesor într-o adresă fizică constând din numărul Cm care conţine locaţia fizică şi o adresă locală de 18 biţi.

#### 3.4.3.1 Etapele de acces la memorie în interiorul unui cluster (Figura 3.20)

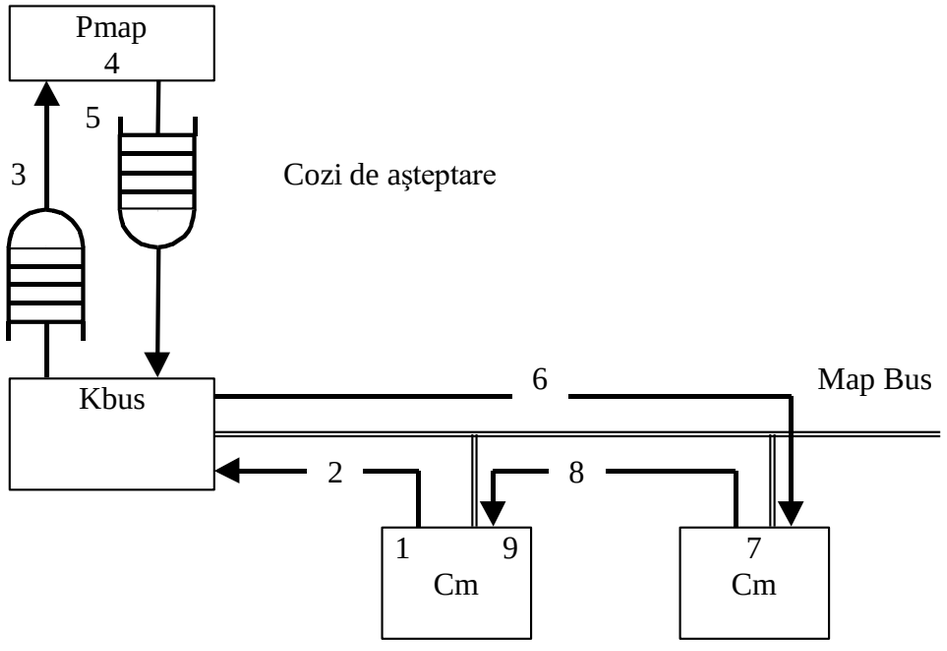



Figura 3.20 Etapele de acces la memorie în interiorul unui cluster

Un $K_{map}$ se compune din:

- $K_{bus}$ - se ocupă de controlul accesului pe MAP BUS
- $P_{map}$ - este procesorul microprogramat care efectuează toate tranzacţiile
- $K_{link}$

1. Procesorul Cm master care vrea să comunice iniţiază un acces la memoria nelocală;
2. $K_{bus}$ - citeşte adresa virtuală de la Cm master;
3. $K_{bus}$ - activează un context (creează o structură de date specifică tranzacţiei) şi îl trimite în coadă de aşteptare pentru $P_{map}$;
4. $P_{map}$ - tratează contextul şi face translatarea adresei;
5. $P_{map}$ - trimite într-o coadă de răspuns cererea pentru ciclul de memorie către modulul Cm slave;
6. $K_{bus}$ - trimite adresa fizică către Cm slave (destinaţie);
7. Prin intermediul lui Slocal, Cm destinaţie efectuează ciclul de acces la memorie prin furt de ciclu;
8. Rezultatul accesului la memorie este transmis Cm-ului care a iniţializat ciclul, sub controlul lui $K_{bus}$;
9. Cm - master continuă execuţia

Translatarea adresei se face deci de către $K_{bus}$ într-o adresă fizică de 18 biţi. După aceasta se execută o a doua tranzacţie la MAP BUS pentru a trece această adresă şi un bit suplimentar care indică tipul operaţiei (scriere/citire) pentru Slocal destinaţie. Dacă operaţia este de citire, datele sunt trecute direct de la Cm destinaţie la Cm sursă şi invers dacă operaţia este scriere datele trec de la Cm sursă la cel destinaţie. Slocal destinaţie efectuează operaţia prin acces direct la memorie prin furt de ciclu. La terminarea operaţiei Slocal destinaţie emite o cerere de achitare transfer la $K_{map}$ pentru a înştiinţa terminarea operaţiei. Are apoi loc un al treilea ciclu de acces la MAP BUS pentru transferarea datelor la procesorul care a iniţiat operaţia în cazul unei citiri şi de asemenea pentru a-l înştiinţa pe acesta despre terminarea referinţei astfel încât procesorul master să ajungă la pasul S adică să continue prelucrările, el între timp fiind într-o stare de aşteptare (Figura 3.21).

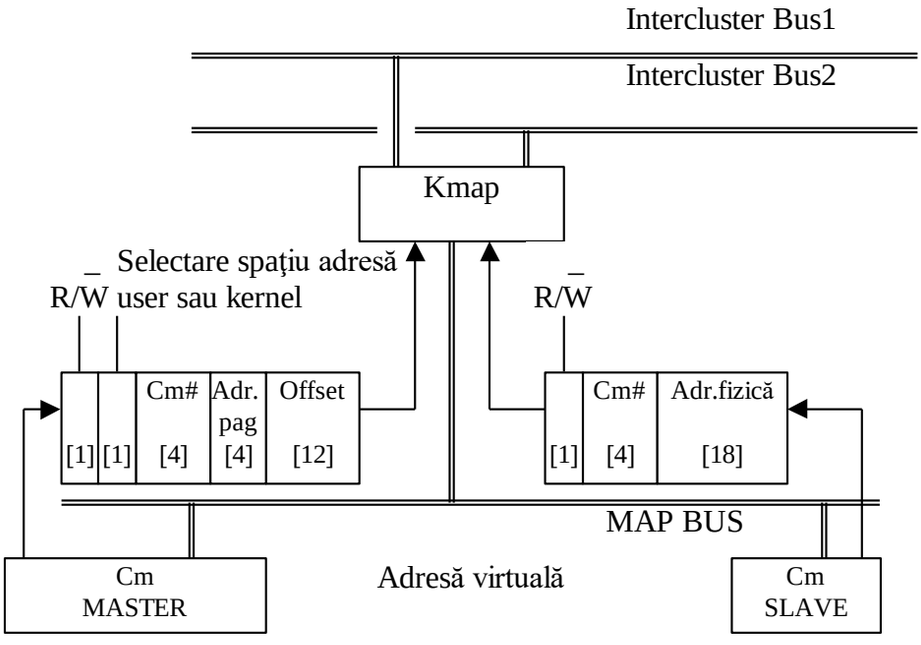



Figura 3.21 Comunicaţii în cadrul aceluiaşi cluster

### 3.4.3.2 Mecanismul de referinţă inter-clustere

O altă posibilitate apare atunci când $K_{map}$ primeşte o adresă de mapat care generează o adresă fizică care nu este locală clusterului. În acest caz, informaţia este memorată în $K_{map}$ pentru pagina care este referită şi care de fapt ne va da numele unui segment de 16 biţi, numărul clusterului care conţine memoria fizică alocată segmentului şi 2 biţi folosiţi pentru a extinde bitul de citire/scriere la un cod de operaţi pe 3 biţi. Această informaţie este concatenată cu cei mai puţin semnificativi 2 biţi de offset ai procesorului iniţial pentru a forma întreaga adresă virtuală a obiectului care este referit (Figura 3.22)

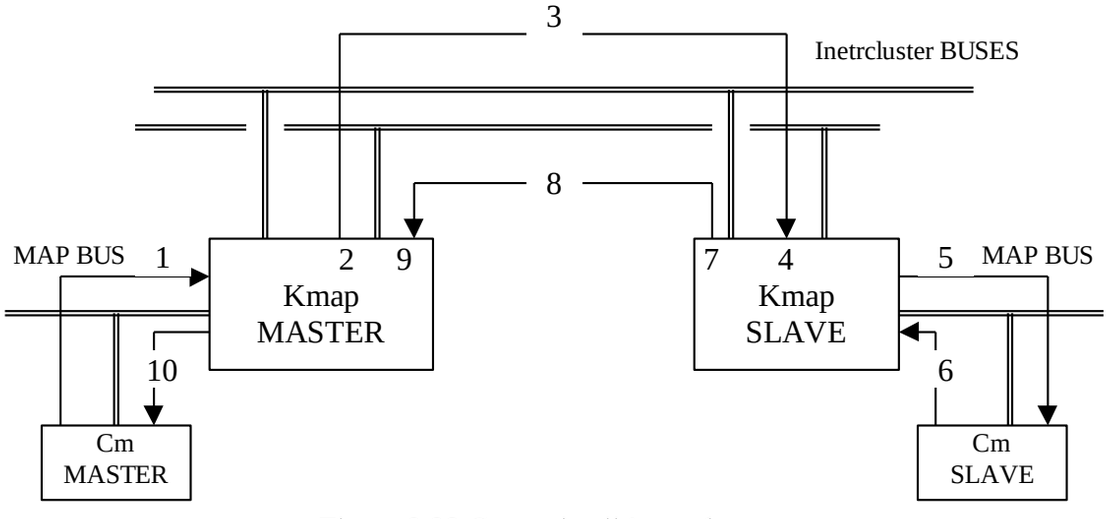



Figura 3.22 Comunicaţii între clustere

1. $K_{map}$-master primeşte o cerere de transfer de la Cm master
2. $K_{map}$-master pregăteşte mesajul intercluster
3. Mesajul intercluster este trimis pe intercluster BUS
4. $K_{map}$-slave decodifică cererea primită
5. Cererea de ciclu memorie este trimisă la Cm slave

6. Cm-slave reîntoarce răspunsul către $K_{map}$-slave
7. $K_{map}$-slave pregăteşte mesajul de răspuns intercluster
8. $K_{map}$-slave trimite mesajul de răspuns intercluster către $K_{map}$-master
9. $K_{map}$-master decodifică şi interpretează mesajul primit
10. Rezultatul este trimis la Cm-master

Structurile de date ce se transmit între clustere sunt prezentate în Figura 3.23.

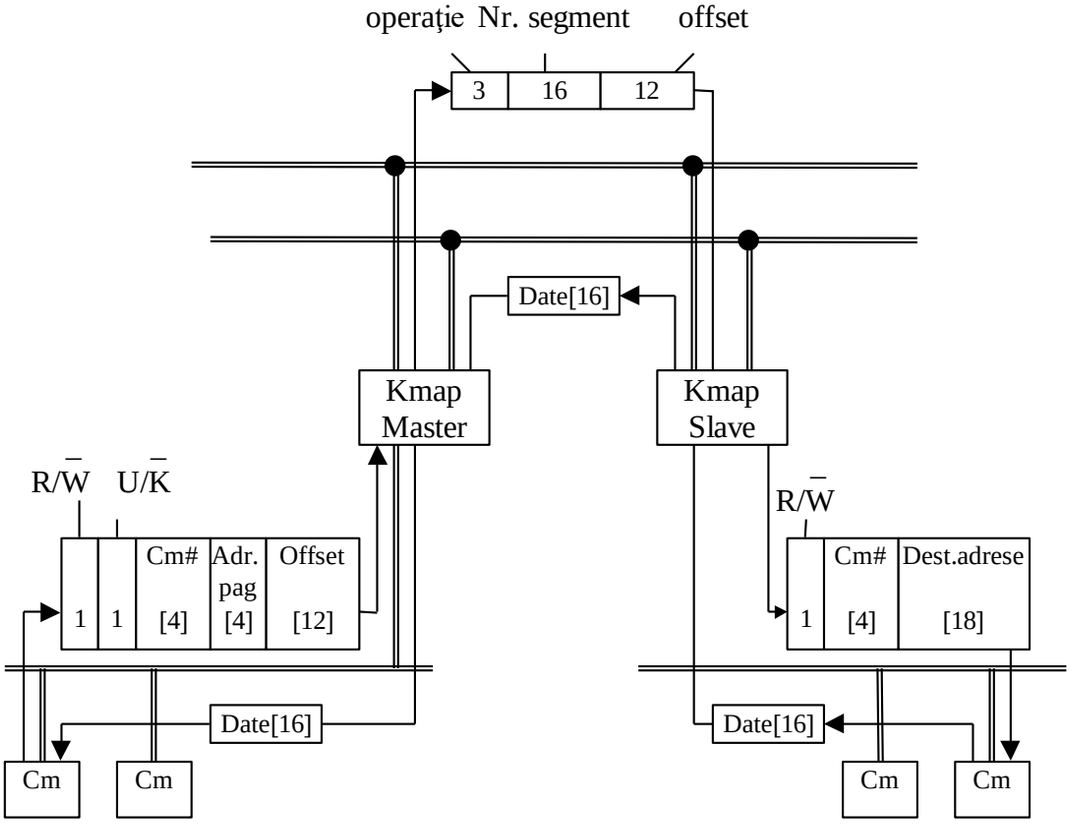



Figura 3.23 Structurile de date ce se transmit între clustere

Adresa virtuală împreună cu datele de la procesor (dacă este vorba de scriere) sunt trimise de-a lungul unei magistrale intercluster la $K_{map}$-ul clusterului care conţine segmentul respectiv. Dacă nu există o magistrală directă care să lege cele două $K_{map}$-uri, în acest caz mesajul va fi trimis de la un $K_{map}$ la altul până când ajunge la clusterul destinaţie, $K_{map}$ având memorie tampon pentru a trata mai multe tranzacţii concurente. $K_{map}$-ul destinaţie va mapa adresa virtuală în adresă fizică în interiorul clusterului propriu.

Tranzacţiile la MAP BUS vor fi executate şi vor permite trecerea adresei fizice şi datelor la un Slocal care la rândul său efectuează operaţia de întoarcere a unui semnal de recunoaştere transfer şi eventual datele (pentru operaţia de citire) înapoi la $K_{map}$-destinaţie. Cu excepţia nivelului magistralei memoriei locale, unde se foloseşte comutare de circuite, toate comunicaţiile se efectuează prin comutare de pachete; rezultă că magistralele sunt alocate numai pentru perioade de transfer pentru ciclul respectiv. De asemenea datele sunt memorate în fiecare interfaţă fără a stabili un circuit alocat continuu de la sursă la destinaţie. Aceasta permite o utilizare mai bună a magistralei şi evită interblocările în alocarea magistralei. Toate transferurile sunt asincrone de tip întrebare-răspuns cu interblocare completă, deci orice operaţie trebuie să se termine printr-un răspuns de achitare transfer şi există implementate tehnici de supraveghere a transferurilor pentru a evita blocările în cazul unor adrese inexistente. $K_{map}$ sunt responsabile de asigurarea acestor funcţii. Translatarea completă adresă procesor - adresă virtuală - adresă fizică este făcută numai în cazul referinţelor intercluster. O caracteristică importantă a structurii de adresare e aceea că întotdeauna există exact un $K_{map}$ care face

translatarea adresei virtuale în adresă fizică pentru un segment dat; rezultă că sunt asigurate unicitatea şi consistenţa accesului. Aceste cerinţe, ca toate referinţele la un segment dat să fie controlate de un singur $K_{map}$, simplifică operaţiile de mutare a segmentelor, de control al capabilităţilor de acces şi de asemenea face posibilă implementarea excluderii mutuale.

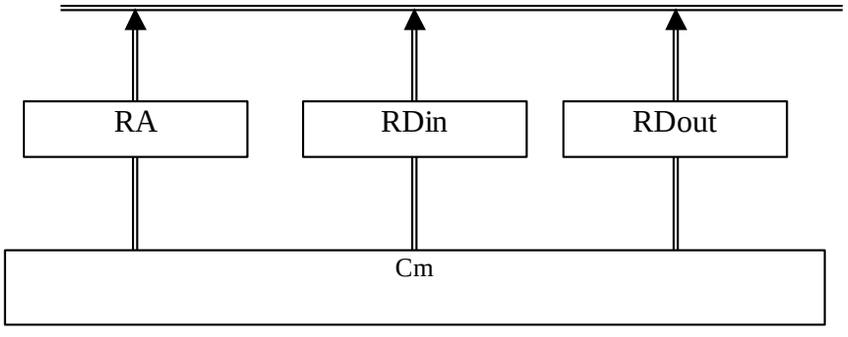



Figura 3.24

### 3.4.4 Controlul mecanismului de comunicare

#### 3.4.4.1 Asigurarea concurenţei în interiorul mecanismului de mapare

Pentru proiectarea mecanismelor de comunicare între Cm-uri s-au avut în vedere următoarele:

1. Timpul pentru o tranzacţie pe MAPBUS » 0,5 microsecunde
2. Timpul necesar într-un $K_{map}$ pentru translatarea unei adrese » 1÷ 2 microsecunde
3. Timpul de transfer intercluster pe o magazie intercluster de » 2 ÷ 4 microsecunde
4. Timpul ca un Slocal să execute o operaţie la memorie » 3 ÷ 4 microsecunde

Referindu-ne la mecanismele pentru accesele nelocale se observă că timpul cel mai mare e consumat în accesele la memoria locală de către Slocal deoarece memoria lui LSI 11 este lentă. Pentru a analiza tranzacţiile controlate de $K_{map}$ s-a folosit noţiunea de context pentru a descrie o tranzacţie.

3.4.4.2 **Operaţiile care necesită excluderea mutuală** (exemplu: translatarea unei adrese din adresă virtuală în adresă fizică)

Se vor implementa în Cm ca referinţe la memorie la nişte segmente speciale care determină pe $K_{map}$ să efectueze aceste operaţii într-un mod de lucru protejat. În general aceste operaţii necesită câteva referinţe ale lui $K_{map}$ la memorie.

Dacă în acelaşi timp $P_{map}$ trebuie folosit pentru alte operaţii de mapare, trebuie să existe posibilitatea de salvare şi restaurare a contextului furnizînd în structura lui $P_{map}$ un număr de registre ce permit efectuarea de până la 8 tranzacţii concurente.

### 3.4.5 Structura lui $K_{MAP}$

$K_{map}$ se compune din trei procesoare distincte: $K_{bus}$, $P_{map}$ şi LINK (Figura 3.25).

**$K_{bus}$** : este în principal unitatea de comandă a magistralei de mapare MAP BUS controlând toate tranzacţiile pe această magistrală.

**$P_{map}$** : procesor de mapare. Face toate translatările de adresă şi menţine o memorie tampon

folosită pentru a creşte viteza operaţiei de mapare. El poate accepta simultan 8 procese.

**LINK** : Legătură interclustere: Supraveghează transmiterea mesajelor între clustere.

În mare, structura acestora este prezentată în Figura 3.24.

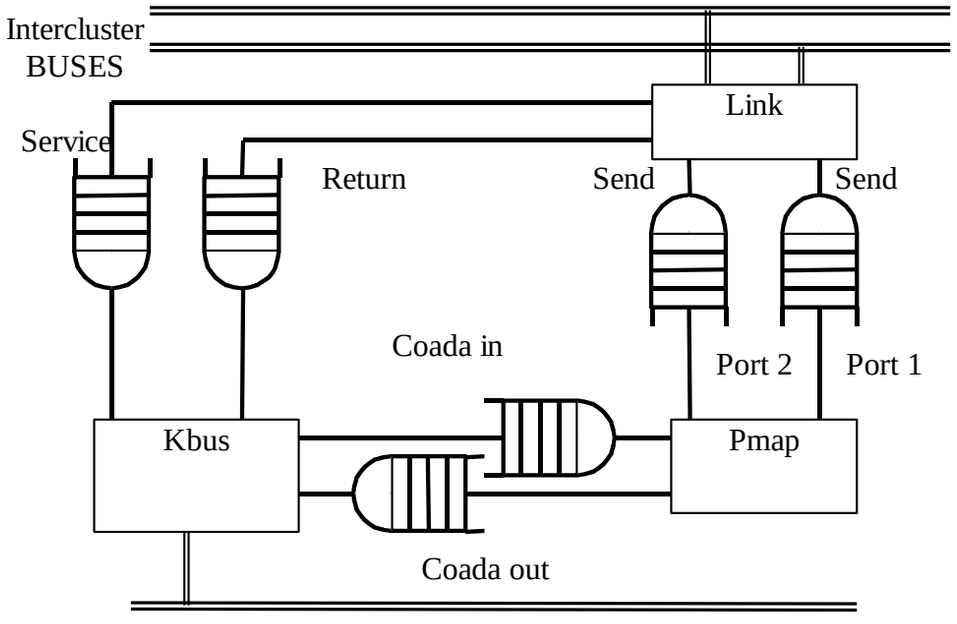



Figura 3.25 Schema bloc $K_{MAP}$

Există trei tipuri de comunicaţie:

- cu comutare de circuite (circuit switching)
- cu comutare de pachete (Packet switching)
- cu comutare de mesaje (Message switching)

Folosirea cozilor care specifică comutarea de pachete, duce la:

- creşterea productivităţii sistemului
- evitarea blocărilor

Un **$K_{map}$** prelucrează 8 *tranzacţii concurente*. Unei tranzacţii îi spunem *context* (putem avea cel mult 8 contexte active la un moment dat). Fiecărui context i se asociază *8 registre generale* şi *4 registre de legătură la subrutină* (se permit 4 nivele de adîncime de salt la subrutină - în Pmap).

### 3.4.5.1 Comunicaţia $K_{bus}$ - $P_{map}$

$K_{bus}$ menţine starea a 8 contexte $P_{map}$ şi alocă aceste contexte unor noi cereri de serviciu pe care le primeşte $K_{bus}$. Numărul contextului şi alte informaţii de stare sunt plasate în coada RUN pentru a sesiza $P_{map}$ că acel context este gata de execuţie. Procesorul de mapare $P_{map}$ activează contextul următor ştergând numărul acestuia din coada RUN şi pornind execuţia lui printr-un salt de la o adresă din microprogram determinată de biţii de stare citiţi din RUN (se execută în funcţie de starea curentă). La activarea unui context nou se efectuează translatarea adresei procesor şi se plasează o cerere de referinţă la memoria principală în coada OUT. Între timp $K_{bus}$ a fost liber să citească alte cereri de serviciu sau să efectueze funcţii cerute în prealabil de către $P_{map}$. La activare (furnizarea cererii în coadă OUT) se execută deci o schimbare a contextului în $P_{map}$ pentru a dezactiva contextul curent până la terminarea referinţei la memorie şi pentru a activa următorul context din coada RUN.

$K_{bus}$ preia cererea din coada OUT şi transferă adresa şi datele (dacă e cazul) la Slocal destinaţie apoi prelucrează alte cereri în timp ce se efectuează accesul la memoria cerută. Când referinţa la memorie este terminată $K_{bus}$ fie citeşte confirmarea de terminare (scriere) sau citeşte şi datele (citire) şi plasează contextul din nou în coada RUN reactivându-l sau trimite confirmarea la procesul care a făcut iniţial cererea de serviciu, terminând astfel operaţia cerută. În acest caz marchează contextul respectiv ca fiind liber pentru realocare la o nouă cerere de serviciu.

Observaţii:

Faptul că un context rămâne alocat la fiecare referinţă nelocală până la terminarea accesului indiferent dacă mai multe $K_{map}$-uri sunt implicate în prelucrarea tranzacţiei crează posibilitatea ca la detectarea unei erori contextul să poată fi reactivat (el există alocat) punând la dispoziţia informaţiei de stare pentru tratarea erorilor.

Astfel se asigură practic comunicaţia $K_{bus}$, $P_{map}$, magistrală şi procesorul ataşat pe magistrala $K_{map}$.

### 3.4.5.2 Comunicaţia Link-$P_{map}$

Aceasta este similară cu cea $K_{bus}$-$P_{map}$; $P_{map}$ trimite într-o coadă Send o cerere de trimitere a unui mesaj interclustere. În acest moment suspendă contextul iniţiator al mesajelor intercluster şi îl trece într-o stare de aşteptare. Când se obţine un mesaj de răspuns pentru contextul respectiv acesta este plasat în coadă "Return" şi LINK determină $K_{bus}$ să reactiveze contextul în coada RUN.

Când se primeşte un mesaj intercluster (trimis dinafară, iniţiat de un context din alt $K_{map}$) de către unul dintre cele două posturi ale LINK acesta e plasat în coada "Service" şi este emisă o cerere către $K_{bus}$ pentru a aloca un context pe care să-l plaseze în coada RUN şi aşa mai departe.

### 3.4.5.3 Legătura $K_{bus}$-$MAP_{bus}$

Există o mare varietate de taskuri pe care trebuie să le execute $K_{bus}$ inclusiv o tratare inteligentă a erorilor. $K_{bus}$ este realizat sub forma unui procesor microprogramat cu o memorie de control ROM de 256 cuvinte a 40 biţi; $K_{bus}$ are un ciclu al microinstrucţiunii de 100 ns sincronizat cu tactul de 150 ns al lui $P_{map}$.

Schema bloc a lui $K_{bus}$ (Figura 3.26)

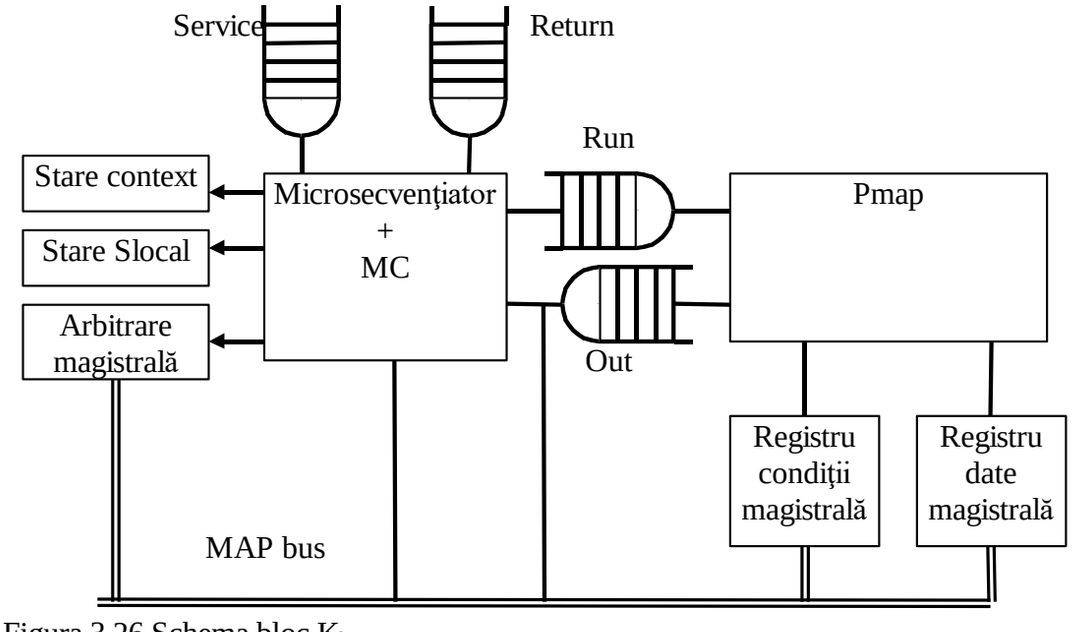



Figura 3.26 Schema bloc $K_{bus}$

**Fluxul informaţiei**

$MAP_{bus}$ - magistrală de comunicaţii cu 38 de linii, dintre care 20 bidirecţionale pentru transmitere adrese şi date între Slocal şi $K_{bus}$-ul unui cluster. $K_{bus}$ controlează toate transferurile de pe magistrala $MAP_{bus}$ deci el specifică sursa şi destinaţia pentru fiecare ciclu de transfer ca şi biţii de stare care specifică conţinutul liniilor de magazie (adrese, date, stări, etc).

Maparea $MAP_{bus}$ este sincronă cu $K_{bus}$ care generează toate semnalele de strob - activare

pentru transmiterea datelor. Fiecare Slocal este prevăzut cu linii de cerere-răspuns individuale, secţiunea de control a magistralei şi arbitrare magistrală interoghează acele linii într-o schemă de priorităţi psoudo "Round Robin". $K_{bus}$ controlează cozile şi registrele folosite pentru comunicarea cu $P_{map}$.

Coada RUN conţine 8 intrări (înregistrări) de câte 8 biţi (nu se poate dep__i niciodată, având în vedere că pot fi activate numai 5 contexte) fiecare activare conţinând număr context (3biţi) şi 5biţi de stare.

Coada OUT conţine 4 intrări (înregistrări a câte 39 biţi fiecare).

$P_{map}$ scrie în această coadă pentru a cere operaţiile ce trebuie executate de $K_{bus}$. Deoarece sunt numai 4 intrări, fiecare scriere trebuie precedată de o inspectare a stării (ca să nu existe depăşire). Fiecare înregistrare a cozii OUT conţine un cod operaţie pentru a selecta una din cele 32 operaţii ce pot fi executate de $K_{bus}$ şi adresa, datele, alte informaţii semnificative pentru operaţia cerută. De asemenea sunt încărcate 2 registre de către $K_{bus}$ pentru fiecare context $P_{map}$ activat. Aceste registre sunt scrise de către $K_{bus}$ şi citite de $P_{map}$. Registrul date magistrală conţine ultimul cuvânt de date de la $MAP_{bus}$ pentru contextul activ, celălalt registru conţine informaţii de comandă şi stare. $K_{bus}$ este responsabil de alocarea contextelor şi deci menţine starea fiecărui context în acest scop. De asemenea $K_{bus}$ menţine 2 biţi suplimentari de stare pentru fiecare context care sunt utilizaţi într-o schemă time-out pentru evitarea sistemului de blocare în cazul în care configuraţia de terminare a unei operaţii la memoria principală sau de trimitere mesaj intercluster nu este terminată în cel mult 2 milisec. Dacă un context suspendat rămâne astfel mai mult de 2ms este reactivat implicit poziţionându-se biţii de stare pe eroare.

$K_{bus}$ menţine de asemenea 9 biţi de stare pentru fiecare Slocal din cluster în care se vede dacă Slocal e ocupat de o referinţă la memorie cerută de $P_{map}$, şi dacă este aşa, ce trebuie făcut cu informaţiile întoarse la sfîrţitul tranzacţiei. Această stare este reiniţializată ori de câte ori se reiniţializează o referinţă la memoria sistemului respectiv şi este folosită şi pentru a se asigura că 2 contexte diferite, nu încearcă simultan să ceară un acces la memorie prin acelaşi Slocal.

#### 3.4.5.4 Arhitectura lui $P_{map}$ - procesor de mapare a adresei

Procesorul de mapare al lui $K_{map}$, **$P_{map}$** este un procesor microprogramat orizontal cu prelucrare pe 16 biţi (Figura 3.27). El ocupă o poziţie centrală în structura lui $K_{map}$ şi coordonează activitatea celorlalte elemente. Are o structură în BA şi ciclu de 150 ns. Are o memorie de control RAM bipolară de 1Koctet x 80 biţi extensibilă la 4K x 80 biţi. Memoria de control se numeşte şi WCS (Writable Control Store).

$P_{map}$ foloseşte de asemenea un RAM rapid de 5k x 16 biţi pentru memorarea informaţiilor care descriu capabilităţile active şi descriptorii de segmente. În plus faţă de translatarea adresei nelocale din cluster, $P_{map}$ implementează o serie de primitive ale sistemului de operare critice din punct de vedere timp şi de asemenea culege informaţii statistice pentru îmbunătăţirea soluţiilor de proiectare. Schema bloc este organizată în jurul unei structuri cu 3 magistrale:

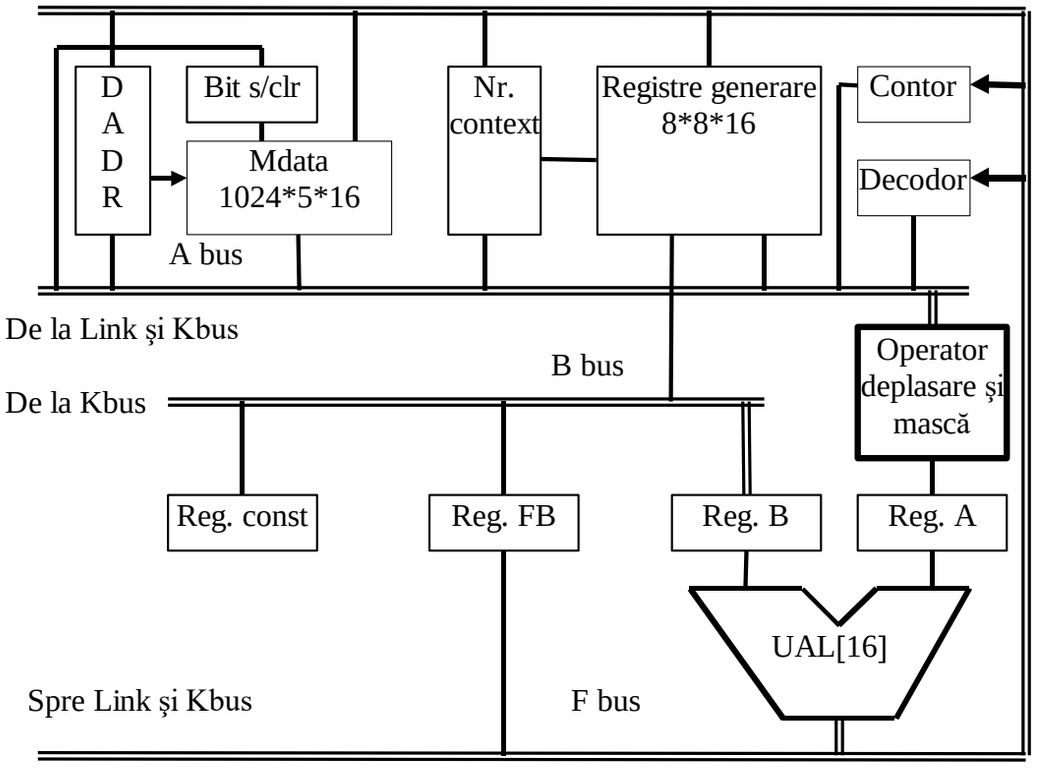



Figura 3.27 Schema bloc $P_{map}$

Cele trei magistrale ($A_{bus}$, $B_{bus}$, $F_{bus}$) sunt cu trei stări: A şi B pentru operanzi, F pentru rezultate, $K_{bus}$ şi LINK având şi ele acces la aceste magistrale. Se utilizează registre tampon pentru a asigura o funcţionare de tip bandă de asamblare, unitatea de deplasare şi mascare asigură prelucrări la nivel de bit a operandului A din UAL, lucru necesar ţinând cont că $P_{map}$ prelucrează pachete de informaţii despre descriptori de segment, mesaje intercluster, capabilităţi etc. Operaţiile sunt de rotire, deplasare, mascare cu una din 32 de măşti de 16 biţi prevăzute într-o memorie PROM. Pentru a asigura operaţia de mapare a adreselor, $K_{map}$ trebuie să aibă acces rapid la informaţiile necesare pentru translatarea adresei virtuale în adresă fizică; aceste informaţii se referă în principal la capabilităţile active şi descriptorii de segment care la un moment dat pot ajunge la 448 simultan într-un cluster. Nu s-a utilizat o memorie asociativă adresabilă prin conţinut datorită capacităţii mari necesare. Utilizând algoritmi eficienţi de alocare a memoriei pentru diferite tabele, utilizarea unor tehnici de memorare sub formă de tabele de dispersie cu diferite procedee de dispersare folosite pentru memoria descriptorilor oferă o soluţie acceptabilă.

Memoria de date e împărţită în 1024 înregistrări expandabile la 4K, fiecare înregistrare conţinând 5 cuvinte de 16 biţi.

Această structură se potriveşte cu structura descriptorilor de segmente şi a informaţiilor de stare asociate în memoria tampon. Memoria e adresabilă pe cuvinte, adresa înregistrării fiind furnizată de registrul de adrese (DADR) iar în cadrul înregistrării se foloseşte un indice de cuvânt de 3 biţi furnizat de un câmp din microinstrucţiunea curentă. Deci în momentul pregătirii unei adrese de înregistrare cuvintele individuale se adresează fără o pierdere de timp pentru calculul adresei. Datele scrise în memoria de date pot fi preluate de pe $A_{bus}$ sau $F_{bus}$. De asemenea s-a prevăzut un operator de setare/resetare de biţi având în vedere structura câmpurilor descriptorilor de segment şi de capabilităţi. De asemenea pentru a permite paralelismul, memoria de date este astfel construită încât permite citirea unui cuvânt în timp ce se citeşte un alt cuvânt.

**Microsecvenţiatorul lui $P_{map}$**

O caracteristică a funcţionării lui este posibilitatea de salturi multiple având în vedere numărul

mare de condiţii de test. De exemplu servirea unei cereri va necesita tastarea stării cererii, a tipului operaţiei, a tipului segmentului, a condiţiilor de violare a protecţiei, a stării descriptorilor (exemplu: blocaţi pentru excludere mutuală), localizarea segmentului etc.

Există în microinstrucţiune un câmp de 2 biţi care selectează modul de ramificare:

- în 2 direcţii
- în 4 direcţii
- în 16 direcţii

şi două câmpuri de 3 biţi care controlează 6 multiplexoare de tip 8:1 pentru condiţii de test.

**Structura microsecvenţiatorului** (Figura 3.28)

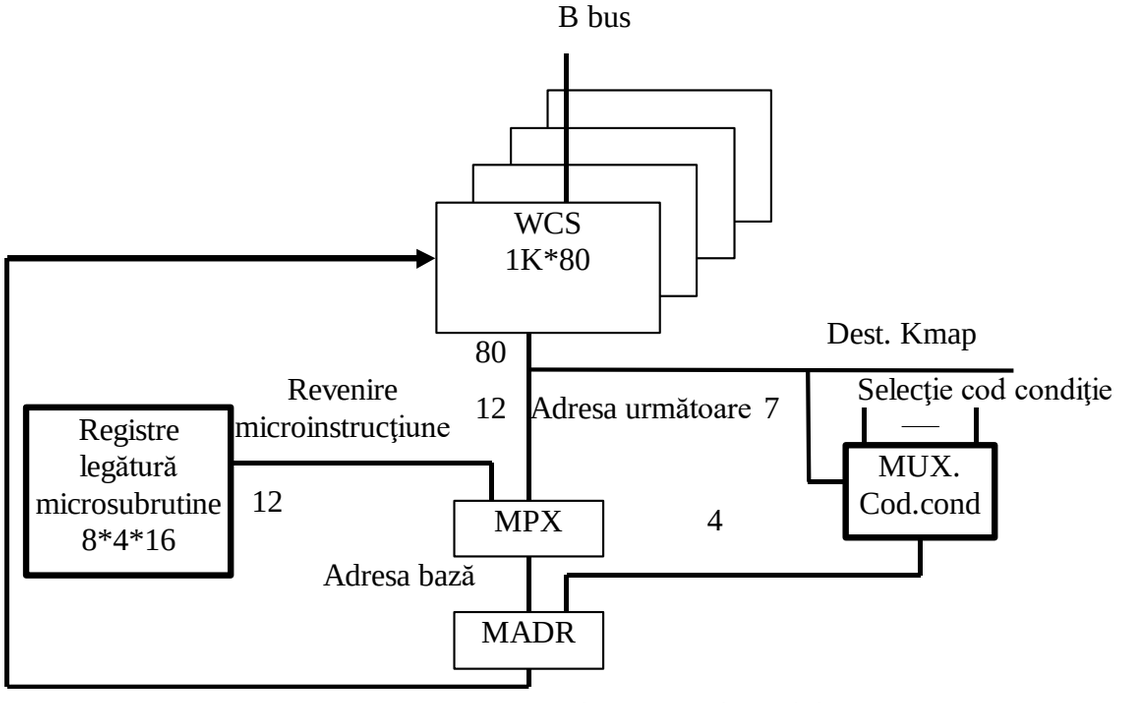



Figura 3.28 Structura microsecvenţiatorului

S-a implementat ramificaţia multiplă printr-un "sau" între codurile de condiţii selectate printr-un câmp din microinstrucţiune cu adresa microinstrucţiunii următoare obţinute dintr-un alt câmp din microinstrucţiune. Pe lângă aceasta, există un mecanism de condiţionare care poate interzice o ramificaţie potenţială în 16 direcţii în sensul că dacă această condiţie este adevărată să execută o ramificaţie la cea de a 16-a adresă fără a se ţine cont de codul de condiţie selectat - permite tratarea urgentă a unor situaţii de excepţie.

### 3.4.5.5 Descrierea unui context în $K_{map}$

$K_{map}$ conţine 64 registre generale şi 32 registre de legătură cu subrutinele, permiţând folosirea exclusivă de către fiecare context a 8 registre generale şi 4 registre de legătură la subrutine. Numărul contextului curent care este memorat în registrul număr context selectează înregistrarea curentă de 8+4 registre generale. În mod normal acest registru este încărcat din coada RUN când se execută schimbarea de contexte.

Pentru a asigura operaţia de diagnosticare, $P_{map}$ poate încărca direct registrul număr de context şi dacă este nevoie, prin microprogram, se poate face acces la registrul fiecărui context.

Fiecare context poate face acces la 4 subrutine imbricate (nested subroutines). Prin convenţie registrul de subrutine 0 este folosit pentru memorarea adresei de reactivare a unui context suspendat. Biţii de stare ai unui context din coada RUN indică modul de reactivare al unui context, de exemplu

de la adresa din registrul 0 sau explicit de la una din cele 16 adrese de salt posibile din memoria de control. De fapt cele 16 adrese pot da 16 intrări distincte în memoria de control. Astfel se poate asigura tratarea unui mod de prelucrare specific stării contextului când a fost activat.

#### 3.4.5.6 Structura Link şi a magistralei intercluster

Link asigură comunicarea intercluster conectând $P_{map}$ la 2 magistrale intercluster prin 2 porturi de legătură. Comunicaţia se face prin mesaje scurte schimbate între $K_{map}$-uri (comunicaţie prin comutare de pachete). Mesajele sunt memorate într-o memorie RAM de mesaje care este gestionată în comun de $K_{map}$ şi cele două porturi ale magistralei intercluster.

Îndicatorii de mesaje trec într-un sistem automat de cozi şi permit regăsirea mesajelor din memoria de mesaje. Mesajele sunt trimise de obicei direct de la sursă la clusterul destinaţie dar pot fi dirijate şi prin clustere intermediare permiţând astfel topologii arbitrare cu numai 2 magistrale intercluster pentru fiecare $P_{map}$. Drumul mesajelor este controlat de microprogramele din $P_{map}$.

În proiectarea Link-ului s-a urmărit asigurarea unei comunicaţii intercluster rapide fără blocări şi cu degrevarea $P_{map}$-ului (Figura 3.29).

### 3.4.6 Comunicaţii prin transmitere de mesaje

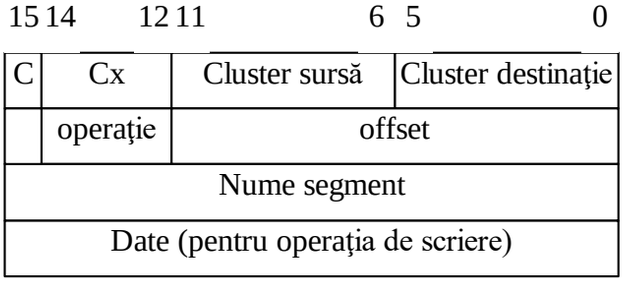
#### 3.4.6.1 Protocolul magistralei intercluster

Magistrala intercluster conţine 26 linii (16 date, 2 de paritate, 8 control). Comunicaţia pe magistrală este asincronă de tip întrebare-răspuns cu interblocare completă cu rata de transfer de 450 ns/cuvânt. Controlul magistralei este trecut ciclic între porturile care iniţiază transferul implementându-se o schemă de prioritate de tip "round-robin". Mesajele intercluster sunt sub formă de pachete din 1-8 cuvinte de 16 biţi. Cele mai utilizate formate de mesaje:

a) - mesaj direct

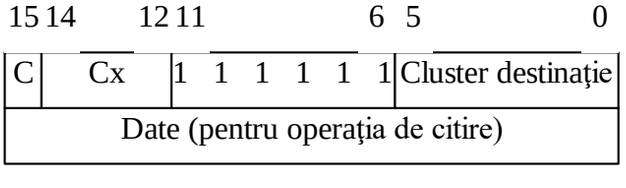
b) - mesaj de răspuns

a) Mesajul direct este format din 4 cuvinte de 16 biţi:




Câmpul de constante al mesajului conţine câte un identificator de 6 biţi pentru clusterele sursă şi destinaţie, CX-ul (numărul contextului care trimite mesajul) şi bitul complex (C). De fapt acest bit este un bit de indirectare spre alte structuri de mesaje (ex: mesaje în caz de eroare sau mesaj de transfer pe blocuri de cuvinte etc.)

b) Mesajul de răspuns este format din 2 cuvinte de 16 biţi



Un mesaj de răspuns are conţinutul câmpului sursă acelaşi pentru orice mesaj. Numărul de context al sursei este trimis cu mesajul pentru a permite reactivarea directă a contextului sursă suspendat.

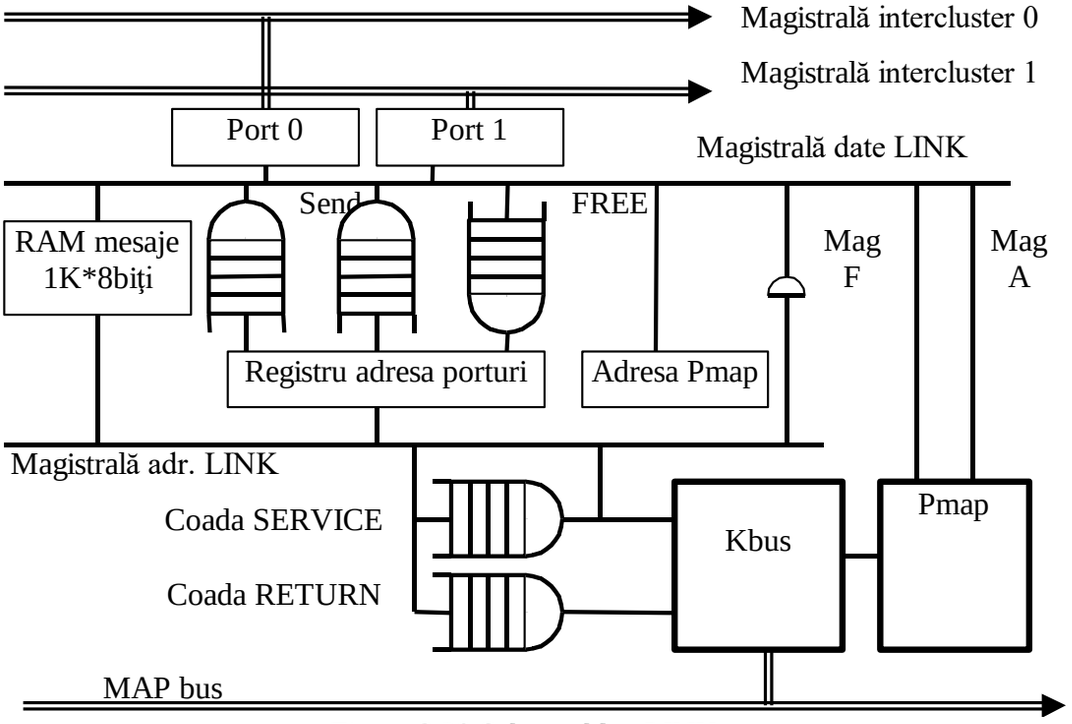



Figura 3.29 Schema bloc LINK

**Coada Free** păstrează intrările libere din memoria de mesaje.
**Registrul adrese pointeri** - trimite nişte pointeri către memoria de mesaje.

Spaţiul bufferelor pentru mesaje este asigurat de RAM-ul de mesaje de 1K x 18 divizat în 128 de înregistrări de câte 8 cuvinte fiecare (pentru ca o singură dată să trebuiască generată adresa după care în cadrul acestei înregistrări prin cuvântul de micro cod se va selecta direct). Aceste 8 cuvinte sunt suficiente pentru a evita apariţia unor situaţii de blocare în sisteme de dimensiuni rezonabile.

$P_{map}$ are posibilitatea de a accesa RAM-ul de mesaje şi de asemenea RAM-ul de mesaje este accesibil direct de către porturi. Mai multe contexte pot folosi LINK în mod concurent fără pericolul apariţiei unor interferenţe ţinând cont că fiecare context adresează independent bufferului de mesaje. Un context poate folosi numărul lui de context pentru a avea acces la un buffer rezervat care este folosit numai pentru operaţia de creare de *mesaje directe* şi de asemenea în care să primească *mesaje de răspuns* de la alte clustere.

În structura LINK există de asemenea registrul de adresă $P_{map}$ (pentru fiecare context câte un registru) pentru tratarea mesajelor directe trimise de alte clustere şi care implică activarea unui nou context în $P_{map}$-ul curent.

Cuvintele din înregistrarea selectată sunt selectate de un câmp din microprogramul lui $P_{map}$. Fiecare din cele două porturi are un registru de adrese şi un registru contor de cuvinte (nefigurate pe schemă) pentru accesul la RAM-ul de mesaje. În mod curent Link-ul gestionează 5 cozi de aşteptare, două dintre ele sunt cozi SEND (una pentru fiecare port) folosite pentru $P_{map}$ pentru a cere transmiterea de mesaje directe.

Pentru a cere transmiterea unui mesaj pe o magistrală intercluster, $P_{map}$ plasează o adresă buffer de mesaj în coada adecvată apoi activează contextul.

Coada Free păstrează adresele tuturor intrărilor libere în bufferele de mesaje.

Coada SERVICE e folosită de Link pentru a informa pe $K_{bus}$ şi Link în legătură cu adresa mesajelor venite de la alte clustere.

Coada RETURN - prin care Link cere lui $K_{bus}$ reactivarea contextelor la primirea unor

răspunsuri la mesajele directe trimise de clusterul curent.

Toate cele 5 cozi sunt implementate ca partiţii ale unei singure memorii bipolare de 1K x 19.

Link funcţionează sincron cu $P_{map}$ cu tact de 150ns. Pentru diagnosticare $P_{map}$ are acces la aproape toate stările interne ale lui Link şi poate executa toate microciclurile interne executate de aceste porturi.

### 3.4.6.2 Tranzacţia de mesaje interclustere (Figura 3.30)

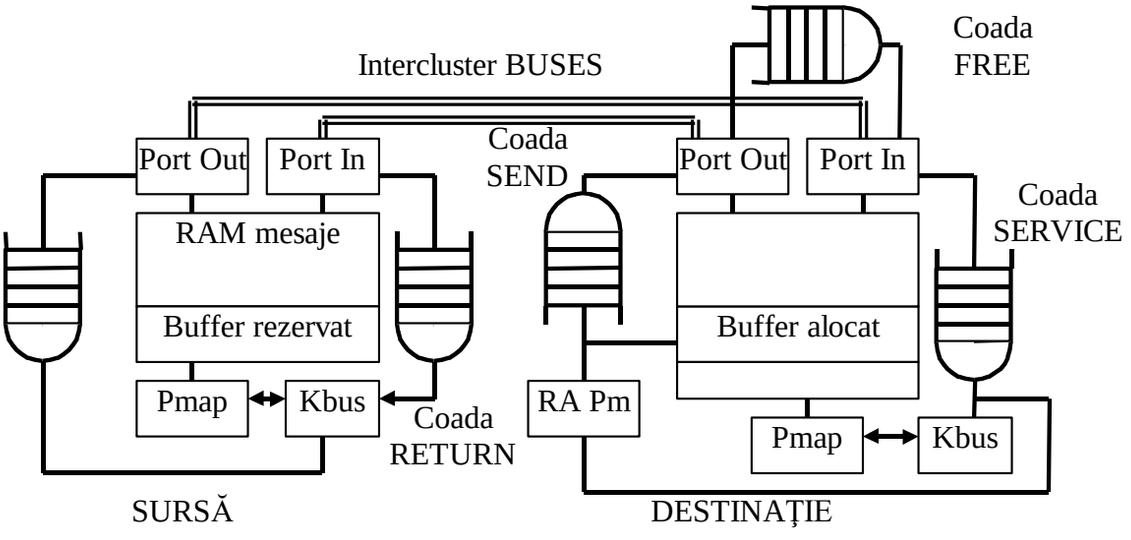



Figura 3.30 Tranzacţia de mesaje interclustere

$P_{map}$ de la clusterul sursă crează mesajul direct într-un buffer rezervat din memoria de mesaje. Apoi adresa bufferului este plasată în coada SEND adecvată portului pe care vrem să transmitem mesajul (denumit acum Port Out). Link scoate pointerul mesajului din coada SEND, îl introduce în registrul de adrese PORT, ocupă magistrala intercluster corespunzătoare şi transferă cuvânt cu cuvânt pe magistrala intercluster mesajul de la RAM-ul de mesaje propriu la RAM-ul de mesaje al LINK-ului destinat. La destinaţie portul receptor obţine un buffer de la coada FREE şi dacă mesajul este primit complet fără eroare atunci pointerul său este plasat în coada SERVICE. În caz contrar, mesajul este ignorat şi la sursă se va genera mesajul time-out pentru că nu se va primi mesaj de răspuns. Coada service de la destinaţie cere lui $K_{bus}$ să aloce un context $P_{map}$ liber pentru a servi mesajul. Contextul respectiv include şi biţi de stare pentru a lansa microprogramul de tratare adecvat. Mai departe contextul adecvat va transfera indicatorul mesajului din coada SERVICE în registrul adresă $P_{map}$ şi prelucrează mesajul făcând referinţele la adresa necesară în memoria principală. Mai departe $P_{map}$ crează un mesaj de răspuns în acelaşi buffer setând câmpul sursă cu 1 peste tot pentru a indica faptul că este mesaj de răspuns. Dacă a fost citire se va adăuga şi cuvântul de date. Pointerul mesajului de răspuns complet este trecut în coada SEND şi trimis pe magistrala intercluster. După aceea pointerul bufferului respectiv este trecut în coada FREE. La clusterul sursă care a iniţiat tranzacţia, mesajul de răspuns este plasat în bufferul rezervat pentru contextul chemător ce a iniţializat tranzacţia. Numărul şi starea acestui context sunt plasate în coada RETURN şi contextul este reactivat pentru a trimite datele sau a prelua datele de la Cm-ul care a iniţiat cererea şi astfel se încheie tranzacţia.

### 3.4.7 Mecanismul de acces bazat pe capabilităţi

### 3.4.7.1 Modul de adresare la memorie a unui proces

Spaţiul virtual de adresare este împărţit în $2^{16}$ segmente, fiecare segment este definit de un descriptor de segment. Tipul de segment standard este un vector liniar de locaţii de memorie. Descriptorul de segment specifică adresa fizică de bază a segmentului şi lungimii.

Dimensiunea unui segment este cuprinsă între 2 octeţi şi 4 Ko. Există tipuri de segmente care sunt mai mult decât simple locaţii de memorie. Astfel, referinţele la segmente pot invoca operaţii speciale. Segmentele pot avea proprietăţi corespunzătoare stivelor cozilor sau altor structuri de date. Anumite segmente n-au asociată nici o locaţie de memorie, iar referinţa la un astfel de segment va invoca o operaţie de comandă. Pentru fiecare tip de segment se pot defini 8 operaţii distincte. Pentru segmentele obiţnuite operaţiile sunt "read" şi "write". Conceptual, segmentele nu sunt nicioadată adresate direct ci prin intermediul *capabilităţilor. O capabilitate* este formată din două cuvinte care conţin *numele segmentului* şi *câmpul de drepturi*. Fiecare bit din câmpul de drepturi indică permisiunea operaţiei corespunzătoare asupra segmentului. Pentru a asigura un suport eficient de schimbare a contextului, transmiterea de mesaje, este necesar ca microprogramul din $K_{map}$ să ştie structura unui proces definit la nivelul sistemului de operare. Fiecare proces este reprezentat printr-un *cadru* (environment). Un astfel de cadru este o structură de date pe 3 nivele compusă din segmente de diferite tipuri.

Registrul cadru utilizator este pointerul către o astfel de structură pe 3 nivele.

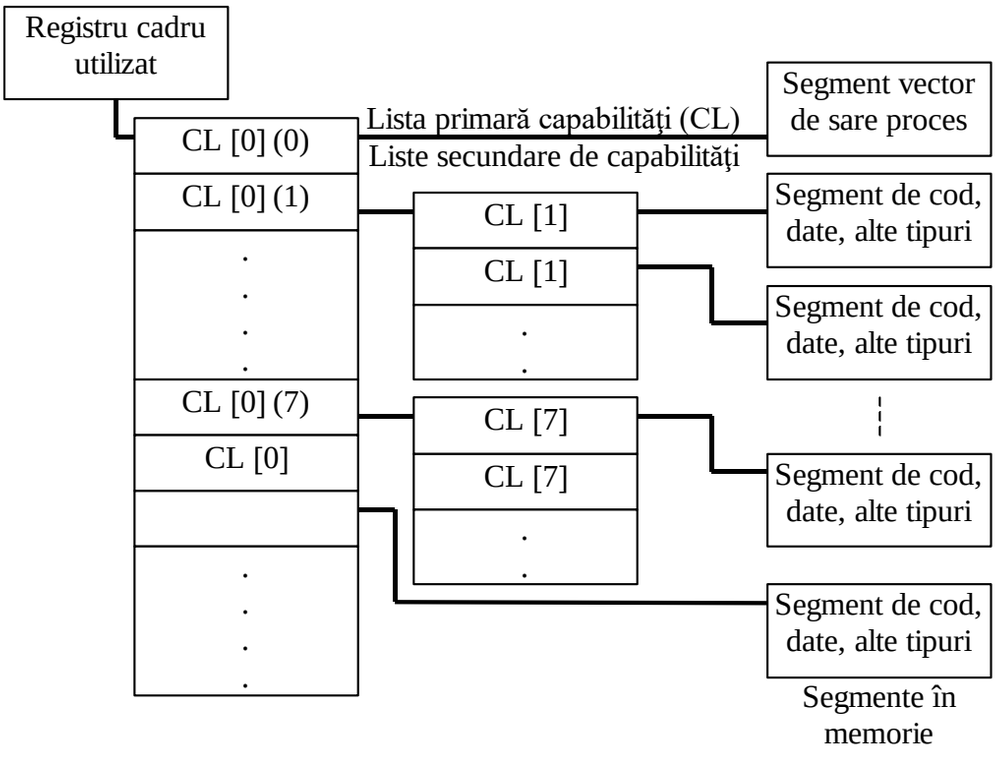



Figura 3.31

Primul nivel conţine lista primară de capabilităţi (CL[0]). Prima intrare în această listă indică segmentul ce reprezintă vectorul de stare a procesului în timp ce acesta este în stare de aşteptare.

Întrările CL[0](1) ÷ CL[0](7) conţin capabilităţi pentru listele secundare de capabilităţi referite ca CL[1] ÷ CL[7]. Întrările din acestea, cât şi cele rămase libere din lista primară conţin capabilităţi pentru segmente accesibile de către proces în timpul execuţiei.

Existenţa a până la 8 liste de capabilităţi duce la utilizarea în comun a unor segmente sau seturi de segmente de către procese ce cooperează.

Un proces poate accesa numai acele segmente pentru care el are capabilităţi şi poate efectua

numai acele operaţii permise prin capabilităţi.

### 3.4.7.2 Generarea adresei virtuale

Spaţiul de adresare de 64Kocteţi al procesorului LSI 11 este împărţit in 16 pagini de câte 4Ko fiecare, fiecare pagină constituind o fereastră în spaţiul de adrese virtual de $2^{28}$ octeţi şi poate fi independent legată de un segment oarecare în spaţiul de adresă virtual.

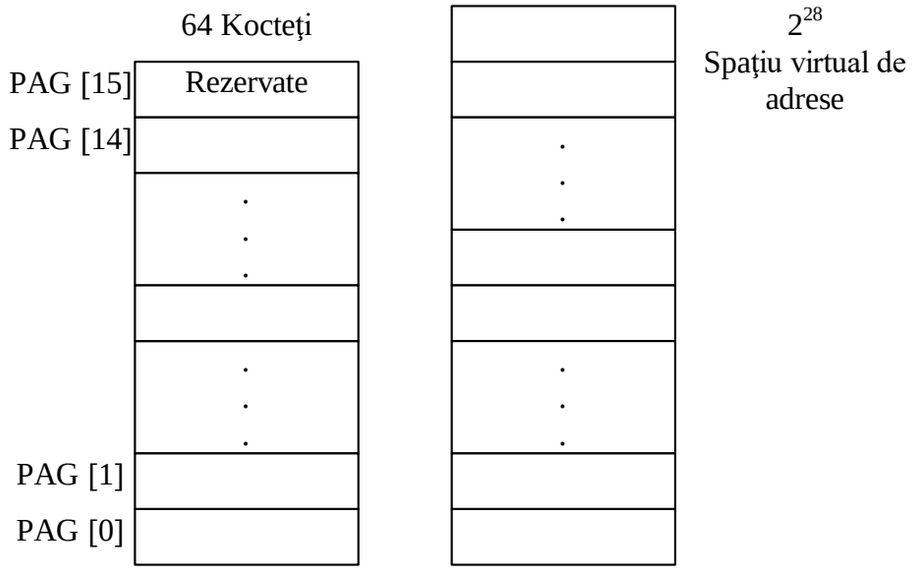



Figura 3.32 Generare adresă virtuală

Pagina a 15-a în spaţiul de 64Ko este rezervată pentru interacţiunea directă a programului din memoria locală cu $K_{map}$. În pagina a 15-a se definesc:

- 15 pseudoregistre - *registre ferestre* (window register). Ele definesc legătura între spaţiul de adresare imediat al procesorului şi elementele din spaţiul virtual. Acest lucru se face "direct" prin intermediul capabilităţilor. Fiecare registru fereastră conţine un index spre o capabilitate în structura listei de capabilităţi a procesului curent (în curs de execuţie).

Un index către o listă de capabilităţi conţine un câmp de 3 biţi care selectează una din cele 8 liste de capabilităţi şi un offset în lista de capabilităţi selectată.

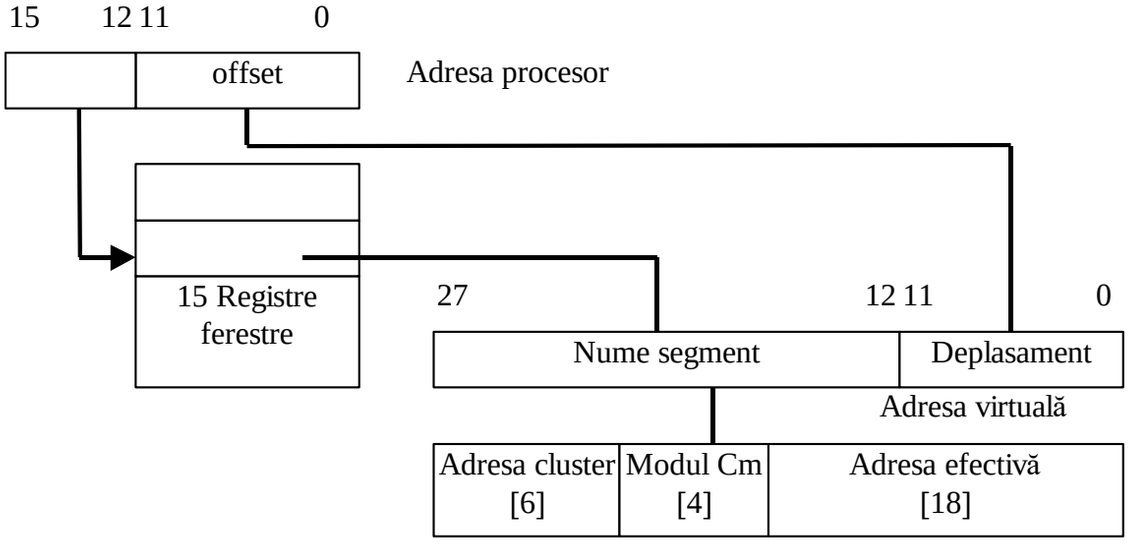


Figura 3.33

Corespondenţa dintre pagini şi segmentele din spaţiul virtual se stabileşte printr-o simplă scriere a unui index de capabilitate într-un registru fereastră din pagina 15.

Această operaţie de scriere în registrele ferestre şi de legătură cu segmentele din spaţiul virtual este sigură datorită faptului că un proces poate referi numai acele segmente pentru care el are asociate capabilităţi. Scrierea unui cuvânt într-un registru fereastră, automat activează $K_{map}$. $K_{map}$ citeşte capabilitatea selectată prin câmpul corespunzător în registrul fereastră (această capabilitate e citită din memoria principală) şi o plasează în memoria tampon de capabilitate a $K_{map}$. Apoi $K_{map}$ îşi ajustează tabelele interne astfel încât referirile ulterioare la paginile respective vor fi dirijate către segmentul specificat de capabilitate prin numele acestuia. Dacă se întâmplă ca acest segment să fie local, $K_{map}$ poate schimba registrul de realocare din Slocal punând bitul de spaţiu pe zero (pentru a nu mai fi trimise referirile la segment către $K_{map}$ ele făcându-se acum la viteza maximă). Din motive de cost-performanţe Slocal nu posedă un hardware mai complex pentru testarea limitei de segment, astfel că fără asistenţa lui $K_{map}$ Slocal poate accesa doar segmente de 4K.

### 3.4.7.3 Translatarea adresei procesor în adresă virtuală

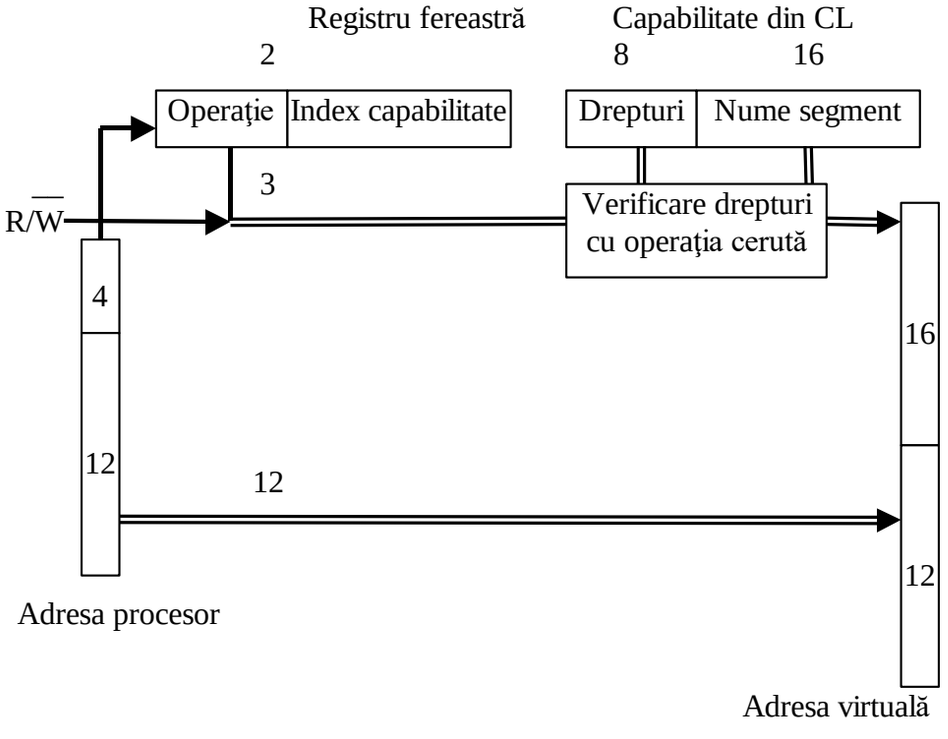



Figura 3.34 Translatare adresă procesor în adresă virtuală

Translatarea în adresa virtuală de 28 biţi se face prin selectarea de către cei 4 biţi mai semnificativi ai unui registru fereastră din cele 15 registre care conţine un index către o capabilitate din liste de capabilitate (Figura 3.34).

Numele segmentului din capabilitatea selectată este concatenat cu cei 12 biţi rămaşi din adresa procesor pentru a forma adresa de 28 biţi. Bitul read/write este concatenat cu cei 2 biţi de operaţie pentru a forma un cod de operaţie de 3 biţi care va selecta şi testa capabilitatea. Dacă operaţia nu este permisă se forăează o intrerupere de eroare.

### 3.4.7.4 Translatarea adresei virtuale în adresă fizică

Depinde de locul segmentului selectat în reţea şi de tipul segmentului. Pentru operaţia de *citire/scriere în acelaşi cluster* translatarea se desfăşoară astfel (Figura 3.35):

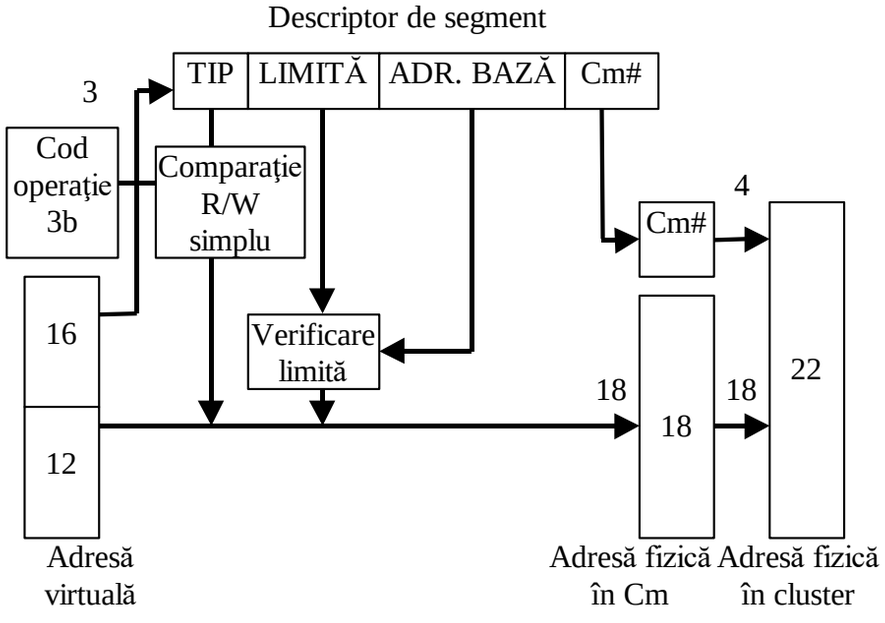



Figura 3.35 Translatare adresă virtuală în adresă fizică

Numele segmentului este utilizat pentru selectarea descriptorului de segment corespunzător. Descriptorul de segment conţine limita segmentului care va fi comparată cu offsetul din adresa virtuală şi în cazul depăşirii limitei se generează o întrerupere de eroare. Offsetul se adună cu adresa de bază a segmentului pentru a forma o adresă de 18 biţi în spaţiul de 256Kocteţi al unui cluster.

Dacă adresa virtuală se referă la un segment din alt cluster numele segmentului este utilizat pentru accesul indirect la un descriptor. Referinţa indirectă indică în ce cluster este localizat segmentul după care $K_{map}$ trimite adresa virtuală la acel cluster prin intermediul magistralei intercluster.

## 3.5 PROGRAMAREA SISTEMELOR MULTIPROCESOR

În multiprogramare sunt mai multe programe ce se execută cvasisimultan folosind acelaşi procesor.

Sistemul de operare în regim de multiprogramare se caracterizează prin schema de alocare şi gestiune a resurselor, protecţia memoriei şi a datelor, rezolvarea structurilor de blocare prin prevenire, detectare, mecanismele de iniţiere şi terminare a taskurilor, planificare pentru execuţie a taskurilor.

La sistemele de operare în multiprelucrare mai trebuie luate în considerare:

- tehnicile specifice de repartizare a resurselor la diferite procesoare;
- planificarea pentru execuţie în sisteme multitasking, în sisteme multiprocesoare;
- creşterea fiabilităţii prin degradarea lentă în cazul defectării unor procesoare - sunt necesare scheme de reconfigurare (pentru creşterea fiabilităţii cât şi pentru creşterea performanţelor)
- mecanismele pentru exploatarea paralelismului (care sunt pe de o parte la nivel sistem de operare şi pe de alta la nivel de limbaje).

Performanţele sistemului de operare la sistemul multiprocesor mai depind de:

- schimbul de comunicaţie între procesoare;

- mecanismele de sincronizare între procesoare;
- strategii de utilizare a resurselor.

### 3.5.1 Tipuri de sisteme de operare pentru sisteme multiprocesor

#### 3.5.1.1 Sisteme de operare de tip master-slave

În astfel de configuraţii există un procesor care este funcţional dedicat pentru a controla starea şi funcţionarea celorlalte procesoare

**Caracteristici**:

- rutinele executivului sistemului de operare se execută pe acelaşi procesor desemnat ca procesor master
- execuţia concurentă a funcţiilor supervizorului şi a programului utilizator
- supervizorul şi rutinele aferente trebuie să fie reentrante
- controlul centralizat de către master simplifică problema conflictului accesului şi utilizării resurselor comune
- structura este mai rigidă dar mai u_or de implementat
- sistemul are o sensibilitate la funcţionarea defectuoasă a procesorului master
- sistemele slave trebuie prevăzute cu posibilitatea de a lansa rutine din sistemul de operare pentru a asigura menţinerea integrităţii sistemului în cazul unor defecţiuni la master
- sunt adecvate structurilor dedicate în care sistemele slave au funcţii (drepturi) mai reduse, în sisteme eterogene (în care funcţiile de control sunt alocate unui procesor cu performanţe mai ridicate).

#### 3.5.1.2 Sisteme de operare distribuite

Se utilizează în general la structuri omogene sau în cazul structurilor distribuite eterogene cu cuplaj slab (de exemplu reţele locale).

Fiecare procesor din structură păstrează o copie a nucleului sistemului de operare. Partajarea resurselor se face în general la un nivel mai ridicat (de exemplu partajare de fiţiere, structuri de date).

**Principalele caracteristici**:

- fiecare procesor îşi satisface propriile cerinţe de resurse (memorie, I/O). Există un nucleu integrator care asigură caracterul de unicitate al sistemului de operare care poate să fie multiplicat în fiecare procesor sau poate să fie reentrant astfel încât să existe o singură copie executată de diferite procesoare
- structuri de date (de exemplu diferite tabele de alocare a resurselor) proprii dar există şi structuri de date comune (de exemplu pentru cereri concurente de acces la resurse)
- toleranţă la defectarea unor procesoare prin asigurarea reconfigurabilităţii sistemului în cazul apariţiei unor defecţiuni. Cu cât această reconfigurabilitate este mai mare, cu atât şi redundanţele ce trebuie prevăzute sunt mai mari, funcţia nucleului integrator privind sincronizarea funcţionării procesoarelor constituie un "overhead" mai ridicat - afectează performanţele procesului.

### 3.5.1.3 Sistemul de operare cu supervizor flotant

Pentru o structură omogenă (procesoare de acelaşi tip universale astfel încât oricare din ele să poată executa funcţiile sistemului de operare).

Caracteristica principală - supervizorul migrează de la un procesor la altul, deşi mai multe procesoare pot executa simultan rutine ale supervizorului. Supervizorul este în mare parte reentrant.

Pentru asigurarea revenirii într-o stare de defectare a procesorului supervizor la un moment dat este necesară prevederea unor informaţii de stare care să fie păstrate de către unul sau mai multe procesoare.

Funcţia de master este transferată de la un procesor la altul după o schimbare de tip rulare prin rotaţie sau după o secvenţă oarecare în funcţie de încărcarea procesoarelor la un moment dat.

Acest tip de sistem de operare permite o folosire mai eficientă a resurselor eliberate de procesoarele slave, schimbarea de priorităţi este flexibilă şi poate fi aleasă în funcţie de utilizarea în comun a unor resurse.

## 3.5.2 Sisteme de operare pentru Cm*

Există trei sisteme mai importante:
Smap
Medusa
StarOS
care pot fi executate şi simultan pe calculator.

### 3.5.2.1 Specificaţii de proiectare

La dezvoltarea sistemului de operare pentru Cm* s-au avut în vedere două aspecte:

**1.** Realizarea unui mediu pentru scrierea şi testarea software-ului orientat maşină este una din modalităţile de dezvoltare software pentru o maşină nouă. Astfel a fost realizat sistemul Cm*Host care este un sistem orientat pe gestiunea resurselor implementat pe un PDP11/10, conectat prin linii seriale la componentele sistemului Cm*.

**2.** Dezvoltarea sistemului de operare multiprocesor constând în principal din implementarea prin microprogramarea $K_{map}$-urilor a primitivelor sistemului de operare.

**Funcţiile de bază ale sistemului Cm*Host sunt:**

**- asigurarea securităţii datelor** (sistemul Host protejează sistemul Cm* împotriva utilizării neautorizate prin implementarea unui sistem de acces şi contabilitate. Orice utilizare a unei resurse din Cm* înseamnă furnizarea unui număr de identificare şi a unei parole)

**- protecţia** - resursele sistemului Cm* sunt folosite concurent de către programele care rulează pe el la un moment dat. Datorită dimensiunilor sistemului rezultă că există situaţii când lucrează concurent mai multe proiecte sau chiar mai multe sisteme de operare. Această simultaneitate a funcţionării lui Cm* este asigurată prin utilizarea unor seturi disjuncte de resurse şi prin asigurarea unui sistem de protecţie pentru decuplarea între ei a utilizatorilor sistemului.

**- controlul resurselor** - sistemul Cm*Host implementează diferite comenzi pentru controlul utilizării resurselor cum ar fi: încărcarea şi lansarea în execuţie a programelor, oprirea sau funcţionarea pas cu pas individuală a unui procesor în vederea testării hard-soft, gestiunea bufferelor pentru comunicaţii inter-procesor, comunicaţiile pe linii seriale între Host şi resursele controlate de Cm*:

**- comunicaţii** - sistemul Cm*Host permite utilizatorului sistemului să monitorizeze ieşirile

simultane de la resursele sistemului Cm* de exemplu dirijînd toate mesajele spre terminalul propriu al Host-ului, lucru foarte util pentru testarea funcţionării în regim de multiprelucrare.

### 3.5.2.2 Sistemul de operare StarOS

S-au proiectat mai multe sisteme de operare pentru Cm* care diferă atât ca principii de bază cât şi ca implementare.

**1° Convenţii de proiectare** a sistemului StarOS

**a) mesaje intercluster**

Fiecare $K_{map}$ este responsabil de controlul tuturor referinţelor la segmentele din clusterul propriu. Dacă un $K_{map}$ dintr-un cluster de la distanţă trebuie să facă o referinţă acesta trebuie să trimită un mesaj spre $K_{map}$-ul ce conţine segmentul respectiv; $K_{map}$-ul local face toate translatările de adrese, testarea depăşirii limitelor etc. Una din problemele importante este *excluderea mutuală*. Deoarece excluderea mutuală în Cm* se face prin descriptori de segment din memoria tampon de 1K x 80 biţi, aceşti descriptori trebuie să fie unici. Pentru a evita blocarea unui context din $K_{map}$ nu poate aloca în mod exclusiv un descriptor de segment în memoria tampon şi apoi să aştepte un eveniment de la alt cluster. Mai mult $K_{map}$ nu poate avea acces exclusiv la 2 descriptori simultan exceptând anumite situaţii speciale.

**b) Contextul nr. 7 nu are voie să aştepte un eveniment intercluster** -rezultă că astfel se garantează că există cel puţin un contex liber în fiecare cluster, evitând astfel apariţia unor situaţii de blocare. *Un context la care aşteaptă alt context nu poate aştepta un eveniment intercluster* (nu este permisă aşteptarea pe 3 nivele).

În configuraţiile curente toate Kmap-urile sunt conectate la ambele magistrale intercluster; nu există situaţii de retransmitere a mesajelor.

**2° Referinţele la memorie**

Toate referinţele la segmentele din sistem se fac prin descriptori din memoria tampon. Dacă $K_{map}$ doreşte mai multe accese la un segment trebuie să facă acces la descriptor de fiecare dată.

**3° Alte considerente de implementare a limbajelor pentru Cm***

Primitivele de bază ale sistemului sunt implementate în microcodul $K_{map}$. Pentru menţinerea consistenţei microprogramelor care sunt mari, un anumit concept este implementat în microprogram într-un singur loc (o singură dată).

Anumite rutine din memoria de comandă pot apela alte rutine care efectuează funcţiile dorite exceptând cazurile în care din motive de performanţe nu se permite imbricarea microsubrutinelor.

Dacă o rutină cere deblocarea unui descriptor (lock - blocare segment; unlock - deblocare segment) şi se detectează o eroare în timpul efectuării operaţiei descriptorul va fi deblocat (necondiţionat).

Constantele sunt furnizate prin încărcarea unor registre de constante. Din motive de performanţe unele sunt cablate.

**4° Caracteristici ale sistemului de operare StarOS**

StarOS este un sistem de operare orientat pe obiecte.

Fiecare obiect din sistem are un anumit tip iar comportamentul obiectului este determinat de funcţiile definite ca parte a tipului obiectului respectiv.

Obiectele pot conţine indicatori (pointeri) protejaşi spre alte obiecte; obiectele pot fi compuse într-o structură arbitrară. Rezultatul referinţei la un obiect nu depinde de locaţia unde se află acesta, de cine şi când a fost creat obiectul sau dacă obiectul este referit de un program utilizator sau de un program sistem.

StarOS oferă un sistem puternic de comunicare între procese implementat prin transfer de mesaje. Mesajele pot fi date sau indicatori spre obiecte.

StarOS este construit pe baza a două clase de funcţii:

- *o clasă formînd nucleul sistemului de operare* care este necesar pentru execuţia oricărui program. Funcţiile nucleului sunt *sincrone* în sensul că un program care apelează o funcţie a nucleului aşteaptă până la terminarea ei. Fiecare funcţie a nucleului este invocată printr-o anumită referire la memorie.
- *o clasă dată de setul de funcţii asincrone* - ele pot fi apelate concurent cu continuarea execuţiei programului chemător, cu semnalizare la încheierea execuţiei funcţiei, dar aceste funcţii pot fi apelate şi sincron. (Obs: Funcţiile asincrone sunt invocate prin transmitere de mesaje).

Toate funcţiile definite de sistem sau de utilizator în afara funcţiilor nucleului sunt funcţii asincrone.

O sarcină este dusă la îndeplinire prin crearea de către StarOS a unor subtaskuri denumite TASKFORCES care sunt colecţii de procese care prin cooperare realizează sarcina respectivă.

StarOS însuşi este un astfel de TASKFORCE.

Toate funcţiile necesare creării de noi subtaskuri sunt disponibile atât utilizatorului cât şi lui StarOS.

Toate informaţiile din sistemul StarOS (programe, date, procese, etc) sunt conţinute în obiecte. Fiecare obiect este de un anumit tip şi fiecărui tip i se asociază un set de funcţii prin care se precizează modalităţile în care informaţiile încapsulate într-un obiect sunt accesibile sau pot fi modificate. De exemplu: singurele operaţii permise asupra unui obiect de tip stivă sunt PUSH şi POP. Asupra unui obiect de tip "cutie poştală" se pot executa numai funcţiile SEND şi RECEIVE.

Nucleul StarOS implementează nişte obiecte de tip REPREZENTARE care se utilizează pentru construirea altor structuri. Fiecare obiect de tip "reprezentare" este implementat ca un sector continuu localizat într-un Cm.

Nucleul implementează funcţiile specifice pentru aceste obiecte de tip reprezentare care la rândul lor sunt funcţii sincrone. Atât StarOS cât şi programele utilizatorului pot defini noi tipuri abstracte de obiecte. Realizarea fizică a fiecărui obiect abstract este un obiect specific unic de tip reprezentare.

Funcţiile specifice obiectelor de tip abstract sunt asincrone.

**5° Apelarea unor funcţii şi adresarea capabilităţilor**

Fiecare obiect are un nume unic prin care se localizează descriptorul obiectului. Descriptorul este o înregistrare dintr-un obiect de tip "directory" în care se specifică tipul de reprezentare, dimensiunea, tipul abstract (dacă este cazul) şi locaţia fizică a obiectului descris. Orice referire la un obiect trebuie să specifice o capabilitate pentru acel obiect.

De fapt o capabilitate este numele obiectului.

Capabilitatea este o structură care conţine numele efectiv al obiectului şi o listă de drepturi sub forma unui vector binar pentru obiectul denumit cu numele respectiv. Fiecare drept din câmpul de drepturi dă posibilitatea de a invoca o funcţie specifică tipului pentru acel obiect (Figura 3.36).

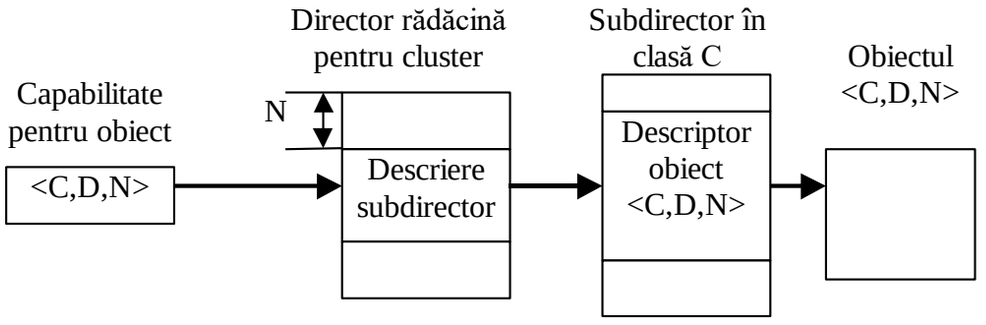



Figura 3.36 Indicarea unui obiect de către o capabilitate

Capabilitatea care indică un obiect oarecare ne duce la un director "rădăcină pentru clusterul C" în care vom găsi o descriere de un subdirector în clusterul "C" unde vom găsi un descriptor pentru obiectul <C, D, N>

O capabilitate conţine trei câmpuri:

C - clusterul în care este conţinut obiectul;

D - arată directorul (subdirectorul) în care se găseşte descriptorul pentru obiect;

N - reprezintă indexul descriptorului în directorul rădăcină.

Singura adresă (singura modalitate) prin care se poate face acces la un obiect este capabilitatea. Capabilităţile stau la baza controlului tuturor autorităţilor (a tot ce este autorizat să se facă) din StarOS.

Ele sunt protejate prin furnizarea unor funcţii specifice de manipulare a lor.

Capabilităţile nu sunt obiecte dar ele sunt conţinute în obiecte.

Un proces este un tip de obiect care poate fi asignat unui procesor pentru execuţie. Procesul are acces la obiectele desemnate (denumite) de capabilităţile memorate în obiectul proces şi la obiectele denumite de capabilităţile memorate în orice obiect la care procesul are acces.

Exemplu (Figura 3.37):

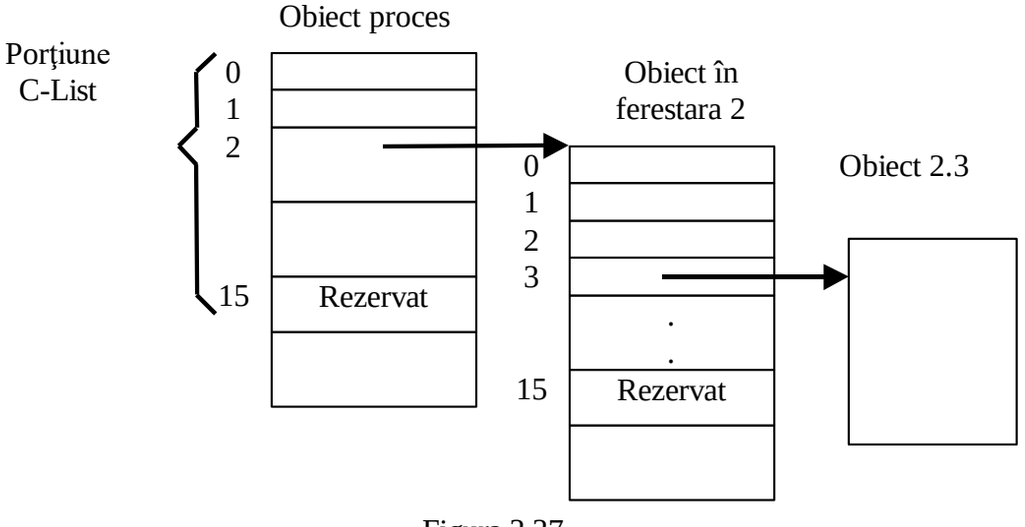



Figura 3.37

Structura listelor de capabilitate şi translatarea adreselor pentru accesul la segmente este cea realizată la structura hardware a lui Cm*. Pentru gestiunea obiectelor s-a definit un modul din StarOS denumit OBJECT-MANAGER care efectuează:

- alocarea memoriei;
- întreţinerea descriptorului în directorul obiecte.

Funcţiile acestuia sunt asincrone.

Există un alt modul în sistemul de operare denumit "Garbage Collector" (colector de spaţiu disponibil). Acesta conlucrează cu nucleul StarOS şi Object Manager având ca funcţii:

- colectarea spaţiului disponibil;
- adăugarea spaţiului disponibil la cel deţinut de Object Manager pentru reutilizare.

Nucleul sistemului de operare este implementat în microprogram prin mai multe clase de primitive definite corespunzător.

**Primitive pentru alocarea resurselor**

Există două primitive:

StartWait

EndWait

care asigură un context care cere o resursă de la sistem că nu va fi "distrus prin înfometare". (De exemplu dacă ar aştepta la un descriptor în starea blocat ("locked") aceasta ar duce la înfometarea procesului care aşteaptă).

Fiecărei resurse sau grup de resurse pentru care un context intră în competiţie i se asociază un cuvânt de sincronizare care menţine starea contextului relativ la acea resursă. Un context care doreşte să intre în competiţie pentru o resursă cheamă procedura StartWait transmiţându-i adresa din memoria RAM a cuvântului de sincronizare. Din StartWait se revine după ce determină că acest context poate concura pentru resursa dorită în condiţii de siguranţă fără a bloca alt context (Obs: toate operaţiile se fac în $K_{map}$). Apoi contextul încearcă să obţină resursa apelând procedura SwapOut şi intră în stare de aşteptare până o obţine. După ocuparea resursei cheamă procedura ENDWait transmiţînd din nou adresa cuvântului de sincronizare marcând faptul că acest context a ocupat *recent* resursa respectivă (pentru a nu permite ca un proces să ocupe de mai multe ori resursa în timp ce alte procese aşteaptă fără a le fi atribuită resursa) (Figura 3.38).

Obs: în fiecare $K_{map}$ este un astfel de nucleu.

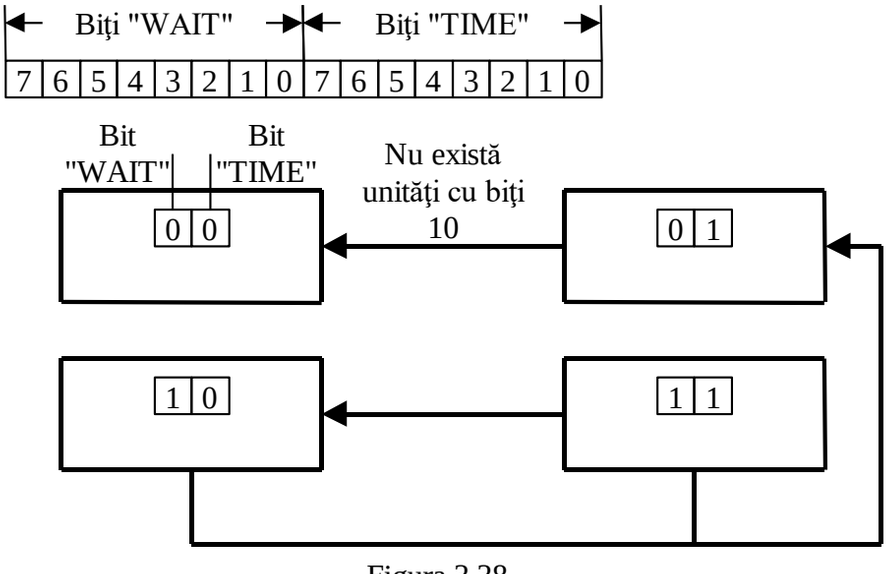



Figura 3.38.

Un proces cu "00" nu a avut resursa recent şi nici nu o doreşte.

Când procesul intenţionează/doreşte să ocupe resursa trece în starea cu biţii WAIT şi TIME în starea 11 în care "*aşteaptă resursa, dar nu de foarte mult timp*". Tranzacţia în starea 10 se face dacă nu există alte contexte cu biţii 10. Starea cu biţii 10 înseamnă că aşteaptă *ocupare resursă de un timp mai îndelungat*. Starea 01 înseamnă că contextul a avut recent resursa şi ajunge în această stare când contextul primeşte resursa.

Un context poate fi în una din următoarele stări:

00 - contextul nu a deţinut recent resursa şi nici nu doreşte să o obţină;
01 - contextul a utilizat recent resursa şi nu poate să o mai utilizeze până ce bitul TIME va trece în 0 marcând faptul că aşteaptă de mult timp;
10 - contextul aşteaptă de mult timp pentru ocuparea resursei;
11 - sistemul aşteaptă de nu prea mult timp.

### 3.5.3 Programarea sistemelor multiprocesor

Resursele de calcul asigurate de un sistem multiprocesor influenţează algoritmii proiectantului şi structura programelor. Programele trebuie astfel structurate încât să utilizeze cât mai avantajos resursele oferite de sistem. Din acest punct de vedere programele diferă faţă de implementarea lor pe sisteme uniprocesor. De aceea tipul resurselor de calcul determină mediul în care se execută şi se depanează programele.

Pentru programarea sistemelor multiprocesor trebuie ţinut cont de următoarele aspecte:

**1°**. Prezenţa procesoarelor multiple sugerează opţiunea prelucrării paralele care lărgeşte spaţiul soluţiilor alternative la o problemă dată. Prezenţa procesoarelor multiple sugerează opţiunea prelucrării paralele care lărgeşte spaţiul soluţiilor alternative la o problemă dată. Este dificil de dezvoltat procedee generale de descompunere a problemelor pentru a obţine soluţii paralele. Exploatarea paralelismului se face în general prin soluţii specifice atât clasei de probleme cât şi resurselor necesare (disponibile). Există totuşi strategii generale care pot fi urmărite de către programatorul ce dezvoltă aplicaţii paralele:

- *abordarea unei partiţii* - aplicabilă la probleme în care toate datele urmează a fi prelucrate în aproximativ acelaşi mod şi în care datele pot fi partiţionate în segmente ce sunt prelucrate cvasiindependent. Acelaşi algoritm de prelucrare a datelor este executat pe mai multe procesoare, fiecare având acces la partea de date pe care o prelucrează. Dacă este necesar, se adaugă o *fază de postprocesare* cu un algoritm specific pentru a combina rezultatele din fiecare partiţie (strategie asemănătoare cu SIMD) De exemplu, în aplicaţii grafice se împarte imaginea în cadrane iar fiecare procesor trebuie să recunoască imaginea din cadranul său urmînd apoi o postprocesare a rezultatelor obţinute;

- *strategie pipe-line* - dedicarea unui procesor pentru fiecare funcţie cerută, datele ce trebuie prelucrate fiind astfel structurate încât trec prin procesoare asemănător structurilor de calcul în BA.

Partiţionarea şi pipe-line-ul nu sunt singurele strategii de descompunere a unei probleme în subprobleme. Trebuie avut în vedere că descompuneri diferite duc deseori la cerinţe diferite privind resursele necesare.

**2°**. Multiplicitatea modulelor de memorie primară şi modul în care comunică procesoarele într-un sistem multiprocesor. Datele în memoria primară sunt accesibile de către fiecare procesor în mod direct prin *segmente partajate* sau printr-un mecanism de *comunicare a mesajelor* (Ex: comunicare prin cutie poştală) sau *printr-un program intermediar*.

Posibilitatea utilizării partajate a memoriei înseamnă că două procese pot comunica direct şi eficient chiar dacă din motive de alocare a resurselor ele se execută pe procesoare separate. Dacă prin arhitectura sistemului o astfel de partajare nu este posibilă, programul trebuie adaptat pentru a folosi mecanismul de comunicare implementat în arhitectura respectivă.

În Cm*, memoria principală deşi omogenă ca adresabilitate este organizată într-o ierarhie de performanţe. Astfel, în paranteze apare costul relativ pentru a accesa fiecare nivel de memorie:

- memoria locală (într-un Cm) (1)
- memoria intracluster (3)
- memorie intercluster (8)

Presupunem că toate operaţiile asupra unor segmente importante se fac având în vedere că segmentele respective se află simultan în memoria principală, această organizare ierarhică generează

probleme de plasare a segmentelor şi de *asignare procese-procesoare*.

Un segment ar trebui să fie plasat în memorie şi procesele să fie asignate procesoarelor astfel încât să minimizeze costul accesului la memorie şi al prelucrării. De exemplu ar trebui ca 2 procese să fie multiplexate pe acelaşi Cm pentru ca fiecare să acceseze cu cost minim segmentele partajate în memoria locală a acelui Cm.

În Cm* cele trei nivele de ierarhizare a memoriei constituie dificultatea principală privind deciziile de plasare a segmentelor în memoria globală a sistemului.

Un sistem, care prin arhitectura sa permite separarea aspectelor privind comportarea funcţională a unui program şi performanţele sale optime care se datoresc plasării optime a segmentelor, este posibil de optimizat. Pentru ca această optimizare să fie posibilă, modul în care un program numeşte un cuvânt nu trebuie să fie funcţie de locul în care este plasat acel cuvânt. De exemplu, în Cm* programele se execută corect dar cu performanţe diferite dacă se ignoră plasarea segmentelor şi asignarea procesoarelor, aceasta deoarece prin hardware se asigură un spaţiu virtual de adresare omogen care acoperă toată memoria fizică. Un program ce se execută corect pe o configuraţie arbitrară se va executa fără nici o modificare, dacă se modifică configuraţia. Spre deosebire de aceasta, într-o arhitectură de tip reţea de procesoare plasarea şi asignarea procesoarelor afectează comportarea funcţională la fel ca şi performanţele, deoarece datele dintr-un nod pot fi inaccesibile dintr-un alt nod.

**3°**. Problema spaţiului redus de adresare. Este generată de utilizarea unor procesoare cu cuvinte scurte. De exemplu LSI 11 din componenta lui Cm* adresează direct 64K pentru un proces dat la un moment dat. Dacă un procesor necesită mai mult de 64K este necesară o tehnică de *alterare a adresabilităţii* care se poate realiza fie prin *mutarea segmentului* fie prin *schimbarea mapării sau translatării adresei*. Costul dinamic al acestor operaţii este ridicat. O alternativă la aceste soluţii este descompunerea programului astfel încât fiecare subprogram să necesite o memorie mai mică decât memoria adresabilă direct.

**4°**. Fiabilitatea unui sistem multiprocesor.

Probabilitatea unei căderi în sistem creşte odată cu numărul de componente şi de conexiuni în sistem. Există cel puţin două strategii pentru a reduce rata căderilor care duc la pierderea funcţională a unui procesor:

- reproiectarea şi creşterea fiabilităţii fiecărui procesor;
- sistemul multiprocesor şi software-ul asociat să fie astfel realizate încât să fie tolerante la căderile componentelor (Ex:dublerea componentelor individuale - prin soft adecvat se poate asigura testarea şi diagnosticarea componentelor menţinând funcţionarea sistemului în condiţiile de degradare lentă a performanţelor).

**5°**. Extensibilitatea arhitecturilor multiprocesor atât în ce priveşte extensia cu memoria secundară sau a posibilităţilor de interconectare.

De asemenea, anumite programe se pot executa numai pe un anumit procesor care are ataşat echipament de I/O necesar.

În Cm* zonele tampon trebuie să fie plasate în memoria modulului la care este ataşat echipamentul respectiv.

**6°**. Manipularea obiectelor.

Sarcina unui programator depinde de posibilitatea de a defini şi manipula obiectele abstracte necesare.

De exemplu, atributele memoriei sistemului: programatorul crează obiecte de date de diferite dimensiuni având în vedere că este convenabil ca ele să poată să fie denumite, memorate independent. O memorie cu segmente de lungime variabilă permite conceptualizarea obiectelor de date, fiecare

obiect putând fi reprezentat compact într-un segment separat. El poate fi definit modificat sau chiar distrus independent de celelalte obiecte. Prin contrast, dacă arhitectura nu permite decât segmente de dimensiune fizică, programatorul trebuie să asigure împachetarea obiectelor mai mici într-un segment, respectiv partiţionarea unui obiect în mai multe segmente. Impachetarea mai multor obiecte într-un segment trebuie să ţină cont şi de durata de viaţă a obiectelor. Necesitatea de a defini obiectele prin offset, în segmentele ce le implementează, crează dificultate pentru programator.

O altă simplificare a sarcinii programatorului, este ca sistemul de operare să-i furnizeze diferite tipuri de obiecte de date împreună cu operaţiile asociate care definesc comportarea particulară a acelui obiect (de exemplu în limbajele orientate obiect există biblioteci de obiecte predefinite).

De asemenea, programatorul este protejat de diferite erori dacă el poate efectua numai operaţiile asociate obiectului. La Cm* există segmente care includ structuri de tip stivă care pot fi tratate doar ca stive şi nu pot fi citite arbitrar de la o adresă oarecare, nici măcar citirea vîrfului stivei printr-o operaţie simplă de citire. La Cm* se implementează ideea că un program nu trebuie să aibă mai multe privilegii decât este strict necesar pentru a-şi realiza funcţiile, interzicând operaţii, altfel legale asupra unor obiecte, care nu sunt necesare efectuării acelor funcţii.

**7°**. Metodologie de dezvoltare software pentru sistemele multiprocesor. Obiectivul programatorului este construirea cu cost minim a unui software conform specificaţiilor, eficienţa într-o formă adecvată între_inerii şi dezvoltării ulterioare.

Comparând productivitatea proiectelor software, se constată dependenţa de complexitate astfel: măsurând numărul de instrucţiuni obiect documentate şi depanate de un programator de nivel mediu ca o funcţie de complexitate şi de mărimea programului. Se constată că pentru programe mici (de aproximativ 5000 instrucţiuni) productivitatea era de 115 instrucţiuni/zi. Pentru programe mai mari de aproximativ 200.000 instrucţiuni productivitatea era de aproximativ 15 instrucţiuni/zi şi de 3 instrucţiuni/zi pentru un sistem extrem de complex.

Mărimea unui program este doar un parametru care determină complexitatea lui dar ea nu este nici măcar dominantă. Se poate arăta că un program în timp real a cărui execuţie depinde de evenimente interne şi externe este de 3 ori mai costisitor decât un program de aceeaşi dimensiune neutilizat în timp real.

Au fost propuse mai multe metodologii de programare pentru software de dezvoltare de complexitate mai redusă. Toate recomandă:

- modularizare;
- partiţionare soluţii la problemă.

În principiu se urmăreşte minimizarea interdependenţei între module şi stabilirea exactă a acestor interdependenţe. Unele metodologii impun ca toate programele să fie secvenţiale reducând complexitatea, soluţie netolerată pentru aplicaţiile paralele pe sisteme multiprocesor. Soluţiile paralele sunt impuse de aplicaţii mari în constrîngeri de timp care nu pot fi rezolvate pe sisteme uniprocesor.

O soluţie paralelă pe un sistem multiprocesor e implementată ca un set de procese, fiecare proces implicând o secvenţă separată de execuţie. Acest set de procese cooperează pentru a efectua un anumit task (în terminologie Cm* setul de procese se numeşte "task force"). Membrii unui "task force" necesită mecanisme pentru partajarea datelor, sincronizare şi comunicare într-o memorie determinstă.

### 3.5.4 Aspecte privind programarea lui CM*

Nucleul SO Cm* (StarOS) gestionează un singur cluster Cm* asigurând adresarea prin capabilitate, alocarea memoriei, declararea modulelor soft, gestiunea proceselor şi a task force-urilor, transmisia mesajelor, multiplexarea procesoarelor (planificarea pentru execuţie a proceselor), manipularea întreruperilor şi a capcanelor. Nucleul StarOS este *distribuit* şi orice funcţie nucleu implementată în software poate fi executată pe orice procesor, adică fiecare procesor execută programele autonom.

Pentru a descreşte timpul mediu de acces la memorie 8 Kocteţi din codul nucleului cei mai des executaşi din StarOS sunt multiplicaţi în fiecare Cm.

Modulele soft pentru manipularea întreruperilor, a proceselor, comutarea proceselor şi pentru comunicaţii între procese sunt de asemenea duplicate.

Un mod de calcul are două spaţii de adresare:

- nucleu (Kernel);
- utilizator.

Orice operaţii implementate de nucleul StarOS se execută în nucleu ca o măsură de protecţie.

Referinţele prin adresare directă la segmentele locale pot fi satisfăcute la nivelul modulului. Referinţele la distanţă (referinţe mapate) necesită asistenţa $K_{map}$.

Dacă un segment este local sau la distanţă, acest lucru nu se reflectă în program. În $Cm^*$ o capabilitate este mărginită la un segment unic deoarece ea defineşte un descriptor de segment ce conţine descrierea fizică a acelui segment (tip, mărime, cluster, Cm-ul, offsetul începutului de segment).

Astfel mutarea unui segment chiar dacă există capabilităţi multiple pentru acel segment necesită schimbarea descriptorului de segment şi nu a capabilităţilor. Majoritatea operaţiilor de manipulare a capabilităţilor sunt implementate în $K_{map}$. Parametrii capabilităţilor referitoare la aceste operaţii trebuie ţinuţi pentru eficienţă într-o listă primară de capabilităţi sau cel mult în una secundară.

#### 3.5.4.1 Operaţii cu capabilităţi şi costurile lor

**1°. Stergere capabilitate**

- costul: 89 microsecunde;
- şterge o capabilitate dintr-o listă de capabilităţi, Clist;
- invocarea ei porneşte şi procesul de colectare a spaţiului disponibil pentru a regăsi şi trece în lista spaţiului disponibil memoria alocată acelui segment, numai în condiţii în care capabilitatea ştearsă este unică pentru acel segment (în sensul că există şi alte capabilităţi pentru acel segment)

**2°. Copiază capabilitatea**

- costul: 127 microsecunde;
- copiază o capabilitate dintr-o înregistrare a unei liste Clist într-o altă înregistrare a unei liste Clist.

**3°. Transferă capabilitatea**

- costul: 70 microsecunde;
- mută o capabilitate dintr-o locaţie a unei Clist în alta eliberând locaţia originală.

**4°. Citeşte capabilitatea**

- costul: 40 microsecunde;
- citeşte conţinutul capabilităţii pentru a extrage informaţii ce se referă la segmentul de date.

**5°. Citeşte/scrie cuvânt**

- cost: 40 microsecunde;
- face acces la cuvinte individuale în segmente care nu necesită să fie direct adresabile, adică în segmente al căror nume nu este înscris într-un registru fereastră.

**6°. Încarcă segment**

- cost: 23 microsecunde;
- încarcă capabilitatea specificată pentru segmentul respectiv în registrul fereastră (ea activează

un segment prin capabilitatea lui);

- operaţiile arătate nu necesită ca segmentul să fie adresabil direct (numele său să fie încărcat într-un registru fereastră). Această primitivă este una din operaţiile critice, mai ales atunci cânt este folosită pentru rezolvarea spaţiului unic de adresare pe care îl are un Cm.

Este un execrciţiu util să se facă o comparaţie între primitivele de lucru cu capabilităţi pentru CM* şi primitivele similare ale lui 80386. să se compare costul acestor primitive cu operaţiile de bază. Operaţiile de manipulare a capabilităţilor prin microprograme la Cm* sunt comparabile cu operaţiile elementare din punct de vedere cost, de exemplu o instrucţiune tipică MOV are un cost de aproximativ 8 microsecunde, dacă operaţiile sunt în memoria locală Cm şi aproximativ 15 microsecunde pentru operanzi la distanţă dar în acelaşi cluster. 80386 are registre interne de capabilităţi, dar 386 face şi swaping între registre interne şi memoria principală.

De exemplu: FADD are un cost de aprox. 58 microsecunde;
MUL are un cost de aprox. 31 microsecunde;
LOAD are un cost de aprox. 28 microsecunde.

### 3.5.4.2 Facilităţi de dezvoltare soft în StarOS

Un programator consideră programele din punct de vedere *static* (datele şi codul specificate pentru a asigura anumite operaţii) şi *dinamic* (în termenii execuţiei codului respectiv).

Construirea unui sistem de programe complex necesită modularizarea sau subâmpărţirea programelor respective în module. Este necesar să se definească foarte bine interfeţele dintre module şi este bine să se minimizeze comunicaţiile între module.

Modulul este unitatea de specificare; prin modul se specifică un set de funcţii care asigură un serviciu pe care-l pot folosi şi alte module. Această idee a fost dezvoltată în continuare sub numele Client-Server: modulele care oferă servicii altor module, apelul făcându-se la un serviciu şi nu la un modul.

**Modulul în StarOS** este unitatea de bază de construcţie a programelor şi pentru a executa funcţii aferente unui modul este necesară prezenţa altor module de care acesta depinde.

Modulul este *unitatea de adaptare a soft-ului la cerinţele utilizatorului*, înţelegând prin adaptare fie specificarea, codificarea respectiv instalarea unui serviciu nou sub forma unui modul nou sau modificări ale modulului existent. Se poate impune pentru adaptarea unui modul la o altă aplicaţie schimbarea implementării care se consideră neadecvată sau chiar a specificaţiilor, caz în care este afectat numai un modul, sau schimbarea se poate reflecta în schimbarea specificaţiilor funcţionale ale modulului de unde rezultă schimbări în modulele de care acestea depind.

De asemenea, având ca bază de plecare modulul, se pot efectua planificarea resurselor şi deciziilor de alocare. Modulele pot fi privite ca grupând segmente pentru plasarea în memorie a acestora şi asignarea proces-procesor.

Un modul constă dintr-o colecţie de obiecte de date şi cod. În cazul în care părţile de cod ce implementează funcţiile specifice modulului sunt interdependente, de exemplu funcţia utilizează subrutine partajate, toate segmentele de cod din grupul pe care le grupează acest modul este bine să fie plasate în acelaşi modul de memorie, în memoria locală spre a fi executate de procesorul local.

Dacă un sistem software este dezvoltat ca o mulţime de module mici prin asignarea procesor-procese se poate face o echilibrare a încărcării procesoarelor atât ca memorie alocată cât şi în ce priveşte execuţia mai eficientă a funcţiilor.

StarOS este construit din module, existând module care definesc funcţii pentru a crea noi module.

Modulele definite pot funcţiona în spaţiul utilizator sau în spaţiul sistem. De exemplu, un încărcător binar ce funcţionează în spaţiul utilizator acceptă specificaţiile segmentului de cod şi date împreună cu specificaţiile funcţiilor ce urmează a fi apelate şi cheamă StarOS pentru a crea un modul obiect. Un modul obiect de acest gen este reprezentat de o listă de capabilităţi CList care conţine

capabilităţi pentru segmente de cod şi date care sunt globale tuturor apelurilor de funcţii ale modulului ca şi capabilităţile pentru acele module ale căror funcţii pot fi invocate de către acest modul. În plus este creat un segment special de date care conţine un vector de descriere a funcţiilor modulului şi se plasează pentru el o capabilitate în CList. De asemenea pentru fiecare funcţie există un descriptor care descrie punctele de intrare, numărul de parametri aşteptaţi şi dimensiunea stivei necesare în timpul execuţiei (Figura 3.39).

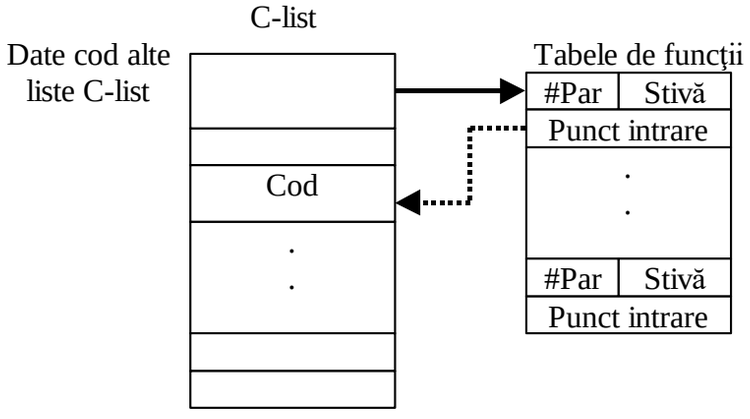


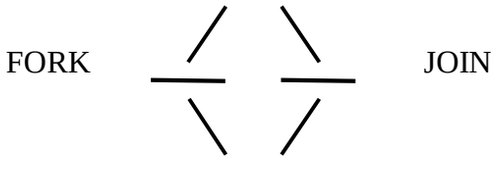

Figura 3.39.

Din punct de vedere *dinamic* în StarOS, unitatea de execuţie este *procesul elementar* (environment) care este construit ca rezultatul unei apelări de funcţii. Un astfel de proces elementar constă dintr-o listă primară de capabilităţi care defineşte spaţiul numerelor de capabilitate şi a informaţiilor de stare a execuţiei. Această listă primară de capabilităţi (LPC) asignată procesului elementar conţine o capabilitate pentru modulul ce implementează funcţia dată, astfel asigurând accesul procesului elementar la toate segmentele globale modulului. Fiecare proces elementar are o stivă proprie de execuţie şi capabilităţi pentru segmentele transferate ca parametri la invocarea funcţiei.

Funcţiile dintr-un modul pot fi invocate prin mecanism CALL specific sistemelor cu multiprogramare dar uniprocesor, fie prin mecanism FORK oferit de nucleu.

CALL crează un proces elementar nou şi suspendă procesul elementar care a apelat până când se execută un RETURN de la procesul elementar chemat.

FORK, faţă de CALL, crează un proces elementar care se execută concurent cu creatorul său. Învocând o operaţie JOIN un proces elementar îşi poate anunţa creatorul, care eventual îl aşteaptă să se termine, că s-a terminat execuţia.




După execuţia unui RETURN sau JOIN acel proces elementar care le-a executat este distrus. Complexitatea acestor operaţii n-a permis implementarea lor în microcod în $K_{map}$.

Un proces StarOS este realizat ca o stivă de procese elementare legate prin CALL în afara primului proces. FORK crează primul proces elementar dintr-un proces descendent, astfel că un TaskForce este un arbore de procese legate prin JOIN.

În cadrul acestui sistem multiprocesor, un TaskForce operează pentru a îndeplini o anumită sarcină de prelucrare. Elementul de bază privind multiplexarea execuţiei nu este procesul, ci procesul elementar.

**1° Asignarea procesoarelor la procese elementare**

Procesul elementar este unitatea de execuţie a unei funcţii şi deci trebuie făcută planificarea pentru execuţie pentru a desemna procesorul sau procesoarele ce execută procesul elementar. Deoarece majoritatea referinţelor la memorie se efectuează la segmentul de cod, un segment elementar în Cm* se execută ineficient dacă codul său nu este local procesorului. Pentru a realiza aceasta, modulele StarOS înregistrează o valoare preferenţială de memorie utilizată pentru planificarea pentru execuţie care este iniţializată astfel pentru a indica Cm-ul în care este rezident codul respectiv.

Procesele elementare sunt alocate pentru execuţia procesoarelor folosind un mecanism de cozi de execuţie (cozi de procese elementare). Astfel, când se crează un proces elementar i se asociază o coadă de execuţie bazată pe adresa preferenţială de memorie a modulului al cărui cod îl va executa procesul elementar creat.

Fiecărui procesor i se asociază o secvenţă de cozi de execuţie. Pentru a lansa în execuţie un proces elementar, procesorul buclează pe secvenţa sa de cozi de execuţie pentru a scoate din fiecare coadă un proces elementar. Acest mecanism de planificare pentru execuţie poate fi iniţializat în mai multe moduri.

De exemplu fiecare procesor poate fi asociat cu o coadă de execuţie proprie sau particulară şi deci fiecare proces elementar se poate executa numai pe acel procesor. O alternativă constă în a asocia fiecărui procesor o coadă de execuţie preferată şi una secundară astfel că atunci când coada de execuţie preferată devine goală va servi procesul elementar din coada secundară. Dacă se defectează un procesor şi memoria lui rămâne intactă rămâne un alt procesor care va servi coada preferată a procesorului căzut în condiţiile în care coada lui este goală.

**2° Comunicaţie şi sincronizare**

StarOS implementează mai multe mecanisme de comunicaţie şi sincronizare între membrii unui TaskForce. Principalele mecanisme sunt prin *partajare de cutii poştale* şi *bazate pe partajarea segmentelor de tip citire scriere* (în genul semafoarelor).

În StarOS nu există un mijloc de a se asigura corectitudinea unei proceduri, deci a unui protocol. Datorită faptului că segmentele referite prin $K_{map}$ pot fi de mărime variabilă, două procese elementare pot partaja minimum de informaţii de control necesare prin acelaşi mecanism ca şi informaţia de date fără o risipă a spaţiului de memorie.

De asemenea, un proces elementar poate fi limitat doar la operaţia de citire (read only) la un segment partajat prin restrîngerea corespunzătoare a capabilităţilor la segmentul partajat.

Nucleul StarOS asigură operaţia de prelucrare a unor variabile cu valori întregi pentru sincronizare, variabile denumite **locks** (semafoare).

Operaţia indivizibilă DecrWord care ar fi echivalentul primitivei P (Wait) decrementează valoarea care precede decrementul este mai mare decât zero.

Operaţia indivizibilă Înc Word incrementează o valoare întreagă semafor.

Sincronizarea lui Cm* se bazează pe funcţiile executate de către $K_{map}$. Având în vedere că orice referinţă mapată la un cuvânt implică $K_{map}$-ul din acelaşi cluster cu cuvântul referit. $K_{map}$-ul menţine semaforul lui propriu pentru fiecare segment şi el este responsabil de asigurarea indivizibilităţii operaţiilor pentru semafoare.

Chiar dacă un $K_{map}$ lucrează în regim de multiprogramare a cererilor, o cerere poate fi făcută indivizibilă deoarece cererile concurente care se referă la acelaşi segment vor fi supuse unui proces de excludere mutuală de către $K_{map}$. Astfel $K_{map}$ foloseşte această tehnică pentru a asigura indivizibilitatea operaţiilor Inc Word, Decr Word şi manipulare de capabilitate, operaţii ce necesită 40 microsecunde fiind implementate în microcod.

O altă metodă de comunicaţie la StarOS o constituie cutiile poştale.

Cutiile poştale sunt segmente partajate iar accesul la acestea se face numai pe baza drepturilor de acces stabilite de capabilităţile ce descriu segmentele care formează cutiile poştale.

Următoarele funcţii sunt primitive de acces la cutiile poştale:

a) Conditional Send
b) Conditional Receive

care se execută numai dacă:

pentru a: cutia poştală este goală;
pentru b: cutia poştală este plină

Alte două primitive sunt:

- Unconditional Send
- Unconditional Receive

care se termină totdeauna, iar nucleul sistemului de operare garantează efectuarea operaţiilor, chiar dacă procesul elementar ce a cerut efectuarea comenzilor s-a blocat.

De fapt transmisia de mesaje este o transmitere de capabilităţi rezultă că se pot transmite structuri arbitrare de mesaje.

Chiar şi cozile de lucru pentru planificarea pentru execuţie sunt accesate tot prin transmisia de mesaje. Dacă un procesor caută un nou proces elementar pentru execuţie el execută un "Conditional Receive" şi dacă coada respectivă nu este goală rezultă că va primi capabilitatea prin care va face acces la procesul respectiv.

Accesul la cutiile poştale se face doar prin operaţii specifice prezentate. Nu este posibil ca un segment asociat unei cutii poştale să fie încărcat într-un registru fereastră pentru a fi scris/citit. Un proces elementar are acces la o cutie poştală doar dacă are dreptul de acces.

În mod curent operaţia de transmitere a mesajelor e implementată într-un modul soft care este multiplicat în fiecare modul calculator.

Procesele elementare din spaţiul utilizator invocă operaţii oferite de sistemul de operare prin nişte "capcane" în spaţiul nucleului (echivalentul INT din 8086).

Secvenţele de cod care execută aceste întreruperi sunt în nucleu şi la terminarea lor controlul este reîntors în spaţiul utilizator, procesului elementar care a provocat întreruperea.

Există mai multe aplicaţii realizate pe acest nucleu:

- programe de QuickSort
- programe PDE (ecuaţii cu derivate parţiale)

### 3.5.4.3 Aplicaţii pentru CM*

**Harpy** (sistem de recunoaştere a vorbirii) - semnalele analogice de la traducătoarele vocale sunt digitizate şi apoi se extrag aşa zisele *gesturi vocale* (nişte amprente vocale ale semnalului recepţionat). După extragerea gesturilor vocale ele se compară cu nişte amprente de referinţă independente de vorbitor şi se generează vectori de probabilitate ca acestea să coincidă cu vectorii de referinţă.

Sistemul Harpy cunoaşte sarcina ce trebuie îndeplinită, inclusiv o descriere gramaticală pentru proprietăţile ce trebuie recunoscute.

Înformaţia combinată, fonetică şi gramaticală, este compilată într-o reţea în care nodurile sunt gesturile fonetice extrase iar arcele direcţionate sunt ponderate şi reprezintă probabilitatea ca un gest să urmeze altuia.

Paralelismul este exploatat prin algoritmi de căutare a drumului cu probabilitatea cea mai mare pentru recunoaştere, parcurgere care se face printr-o metodă euristică în paralel pe procesoarele existente în sistem.

Există un proces care face o preprocesare constând din extragerea vectorilor de probabilităţi.

Sistemul a fost implemetat sub forma a trei tipuri de module program, modulul de legătură şi comunicaţie (extrage un vector de probabilitate), modul de interfaţă şi coordonare, modul master şi modul slave care face căutare în graf.

# Cuprins


**4. CONTROLUL ACCESULUI PE MAGISTRALE ..... 4-1**

4.1 METODE DE CONTROL AL ACCESULUI PE MAGISTRALE ..... 4-2
*4.1.1 Unitate de comandă a accesului la magistrală prin înlănţuire serială ..... 4-2*
*4.1.2 Unitate de comandă a accesului la magistrală prin interogare ..... 4-3*
*4.1.3 Unitate de comandă a accesului la magistrală prin cereri independente ..... 4-4*
4.2 TEHNICI DE COMUNICAŢIE PE MAGISTRALĂ ..... 4-5
*4.2.1 Comunicaţii sincrone ..... 4-6*
*4.2.2 Comunicaţii asincrone ..... 4-6*
4.2.2.1 Comunicaţii asincrone cu comandă într-un singur sens ..... 4-6
4.2.2.2 Comunicaţii asincrone de tip întrebare - răspuns ..... 4-7
4.3 MODALITĂŢI DE TRANSFER A DATELOR ..... 4-10
4.4 EXEMPLU DE SISTEM MULTIPROCESOR CU MAGISTRALĂ UNICĂ MULTIPLEXATĂ ÎN TIMP ..... 4-11
4.5 INTERCONECTAREA ÎN SISTEMELE CU RESURSE MULTIPLE ..... 4-13
4.6 PERMUTĂRI FUNDAMENTALE ..... 4-16
*4.6.1 Permutarea de bază ..... 4-16*
4.6.1.1 Permutarea de baza k_inferior ..... 4-17
4.6.1.2 Permutarea de baza k_superior ..... 4-17
*4.6.2 Permutarea cu intercalare perfectă ..... 4-18*
4.6.2.1 Permutarea cu intercalare perfectă k_inferior ..... 4-19
4.6.2.2 Permutarea cu intercalare perfectă k_superior ..... 4-19
*4.6.3 Permutare cu negare bit ..... 4-21*
4.6.3.1 Permutarea fluture k_inferior ..... 4-23
4.6.3.2 Permutarea fluture k_superior ..... 4-24
*4.6.4 Permutarea cu ordine inversă ..... 4-24*
4.6.4.1 Permutarea cu ordine inversă k_inferior ..... 4-25
4.6.4.2 Permutarea cu ordine inversă k_superior ..... 4-26
*4.6.5 Permutarea cu incrementare modulo ..... 4-26*
4.6.5.1 Permutarea cu incrementare modulo k_inferior ..... 4-27
4.6.5.2 Permutarea cu incrementare modulo k_superior ..... 4-28
4.7 REŢELE DE COMUTARE IERARHICE ..... 4-29
*4.7.1 Reţele de comutare de tip DELTA ..... 4-29*
*4.7.2 Performanţele reţelelor de comutare ..... 4-32*
4.7.2.1 Analiza reţelei de comutare de tip cross bar ..... 4-33
4.7.2.2 Analiza unei reţele de tip DELTA $a^n$ x $b^n$ ..... 4-35

# 4. Controlul accesului pe magistrale

**Obiective**

## 4.1 Metode de control al accesului pe magistrale

Pentru asigurarea comunicaţiilor pe magistrale în condiţiile respectării celor trei reguli de transfer trebuie să fie prevăzute unităţi de control a accesului pe magistrală. Aceste unităţi pot să constituie module fizice separate sau pot să fie distribuite la nivelul modulelor conectate pe magistrală. Se obţin astfel două tipuri de magistrale :

- magistrale cu control centralizat;
- magistrale cu control distribuit.

Indiferent de modul de implementare, centralizat sau distribuit există trei metode principale de realizare a unităţilor de comandă pentru accesul la magistrale :

- unităţi de comandă cu control prin înlănţuire serială;
- unităţi de comandă cu control prin interogare;
- unităţi de comandă cu control prin cereri independente.

După modul în care participă la efectuarea transferurilor pe magistrală modulele unui sistem sunt de două tipuri :

- module "MASTER" care pot să iniţieze şi să controleze transferurile pe magistrală cu alte module;
- module "SLAVE" care pot doar să răspundă la cererile de transfer iniţiate de modulele de tip "MASTER".

În continuare se vor considera magistrale care acceptă mai multe module MASTER, denumite magistrale multi - MASTER.

În schemele de principiu prezentate în continuare nu sunt prezentate liniile de date şi adrese pentru simplificarea desenelor.

### 4.1.1 Unitate de comandă a accesului la magistrală prin înlănţuire serială

Schema de principiu pentru interconectarea modulelor unui calculator numeric printr-o magistrală cu control centralizat prin înlănţuire serială este prezentată în figura 4.1.

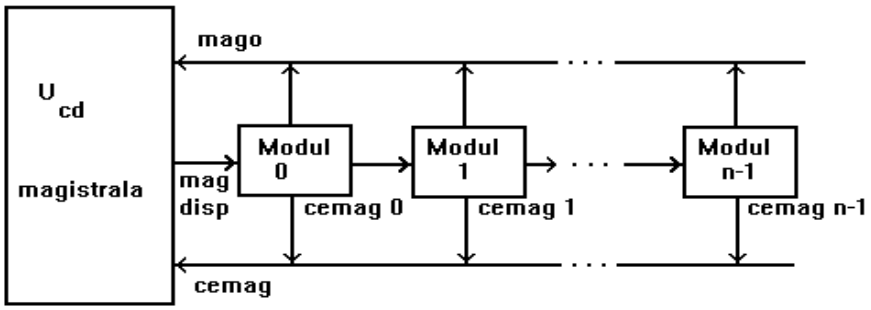



**Figura 4.1** Control centralizat cu înlănţuire serială

Fiecare modul poate să lanseze o cerere de acces pe magistrală activând linia comună *cemag*. Dacă Ucd sezizează activarea liniei *cemag* în condiţiile în care semnalul *mago* este dezactivat activează linia *magdisp* care merge prin înlănţuire serială la toate modulele. Un modul i care primeşte linia *magdisp* activată şi nu doreşte acces, deci are *cemag i* dezactivată, transmite mai departe *magdisp* activată. Primul modul j care sesizează trecerea în starea activă a semnalului *magdisp* şi doreşte acces, adică a activat *cemag j*, trimite mai departe *magdisp* dezactivată, activează linia comună *mago*, dezactivează linia *cemag j* şi începe transferul. Activarea liniei *mago* menţine activă linia *magdisp* până la dezactivarea liniei *mago* ca urmare a

terminării unui ciclu de transfer. În momentul în care Ucd sezizează dezactivarea liniei *mago* va dezactiva semnalul *magdisp,* activând-o eventual dacă semnalul *cemag* este activ. Deci *mago* şi *magdisp* sunt activate şi dezactivate la fiecare ciclu de transfer.

Prin eliminarea liniei *mago* şi conectarea liniei *cemag* la intrarea *magdisp* de la Modul 0 se obţine schema cu control distribuit din figura 4.2.

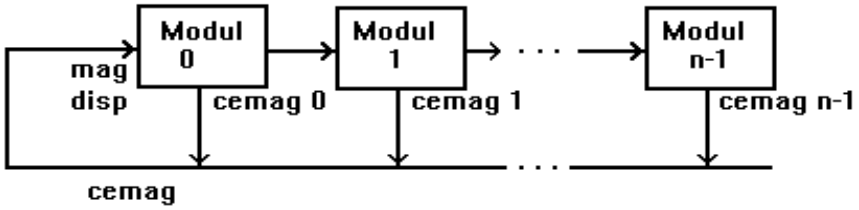



Figura **4.2** Control distribuit cu înlănţuire serială

Un modul i activează linia *cemag i* numai dacă la intrarea sa *magdisp* este inactivă. Dacă modulul j a activat *cemag j* şi primeşte la intrare *magdisp* activă, transmite mai departe *magdisp* dezactivată şi începe transferul menţinând activă *cemag j* până la terminarea ciclului de transfer. În perioada transferului numai modulele j+1,...,n-1 având semnalul *magdisp* inactiv la intrare pot să genereze cereri de acces. Dezactivarea liniei *cemag j* dezactivează linia *magdisp* dacă nu mai există alte cereri *cemag j+1:n-1* active. Dacă există, primul modul după modulul j care are *cemag* activă va prelua controlul accesului la magistrală.

Se observă că la schema centralizată priorităţile sunt fixe. Modulul 0 are prioritatea cea mai mare iar modului n-1 cea mai mică. La schema distribuită prioritatea cea mai mare în ciclul următor o are modulul imediat următor celui care a controlat magistrala în ciclul curent. Schema de priorităţi este de tip rulare prin rotaţie (round robin).

Avantajele principale ale schemelor cu înlănţuire serială sunt:

- simplitate;
- cost scăzut;
- numărul liniilor de comandă nu depinde de numărul de module;
- extensibilitate;
- reconfigurabilitate.

Dezavantajul principal este inerent schemelor de tip "serial", la apariţia unui defect, pe linia *magdisp* de exemplu schema nu mai funcţionează. La schema centralizată priorităţile fiind fixe, trebuie avut în vedere că un modul aflat în apropierea Ucd care utilizează intens magistrala poate să blocheze modulele cu prioritate mai mică.

### 4.1.2 Unitate de comandă a accesului la magistrală prin interogare

Schema de principiu pentru controlul centralizat prin interogare este prezentată în figura4.3.

Un modul i care doreşte acces activează linia *cemag i*. La detectarea activării liniei *cemag*, dacă semnalul *mago* este dezactivat, unitatea de comandă porneşte interogarea numărând pe liniile *conint* - contor de interogare.

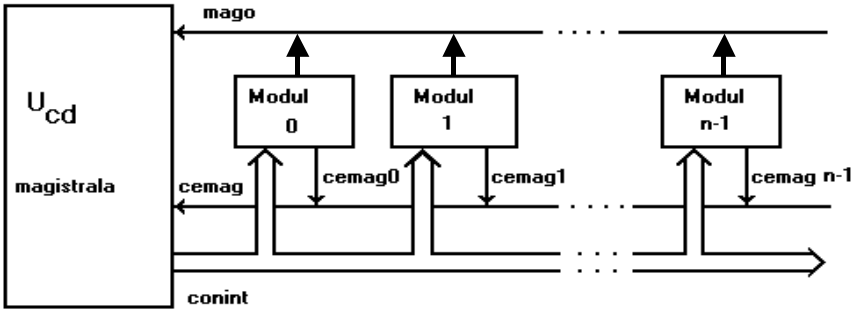



**Figura 4.3** Control centralizat prin interogare

Primul modul j care îşi detectează propria adresă pe liniile *conint*, activează linia *mago,* dezactivează linia *cemag j* iar unitatea de comandă sesizând activarea liniei *mago* opreşte numărarea pe liniile *conint*. La terminarea ciclului de transfer curent modulul j dezactivează *mago* şi concurenţa pentru ocuparea magistralei poate reâncepe. Ucd poate începe numărarea pe liniile *conint* pornind de la zero pentru a implementa o schemă cu priorităţi fixe sau continuând de la modulul j+1 pentru a realiza o schemă de priorităţi de tipul rulare prin rotaţie. Desigur că numărarea se poate executa într-o secvenţă oarecare pentru a se realiza scheme de priorităţi mai complicate. Schema este uşor de implementat dar numărul de linii de comandă este mare iar extensibilitatea este dificil de realizat.

Schema de control prin interogare de tip distribuit este prezentată în figura 4.4.

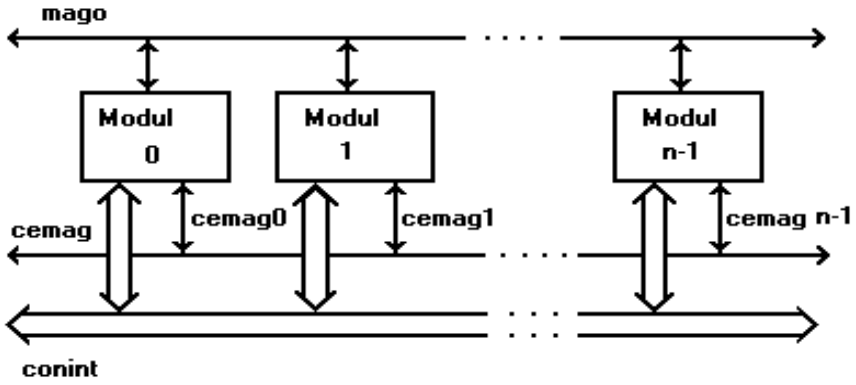



**Figura 4.4** Control distribuit prin interogare

Fiecare modul trebuie să poată executa funcţiile unităţii de comandă. Iniţial un modul oarecare execută controlul magistralei. Toate liniile de comandă sunt bidirecţionale. Un modul care deţine controlul magistralei la sfârţitul transferului examinează linia *cemag.* Dacă este activă porneşte interogarea, într-o secvenţă corespunzătoare algoritmului de priorităţi ales. Fiecare modul care doreşte să ocupe magistrala activează linia *cemag* numai dacă *mago* este inactivă şi compară codul de pe liniile *conint* cu propria-i adresă. Când îşi recunoaşte adresa pe liniile de interogare, activează linia *mago*, dezactivează linia *cemag* şi începe transferul. La activarea liniei *mago* modulul care efectua interogarea opreşte numărarea pe liniile *conint.* La terminarea transferului modulul respectiv preia funcţia de control a accesului la magistrală în ciclul următor. Schema distribuită are avantajul unei degradări lente în cazul defectării unor module.

### 4.1.3 Unitate de comandă a accesului la magistrală prin cereri independente

Schema centralizată cu cereri independente este prezentată în figura 4.5.

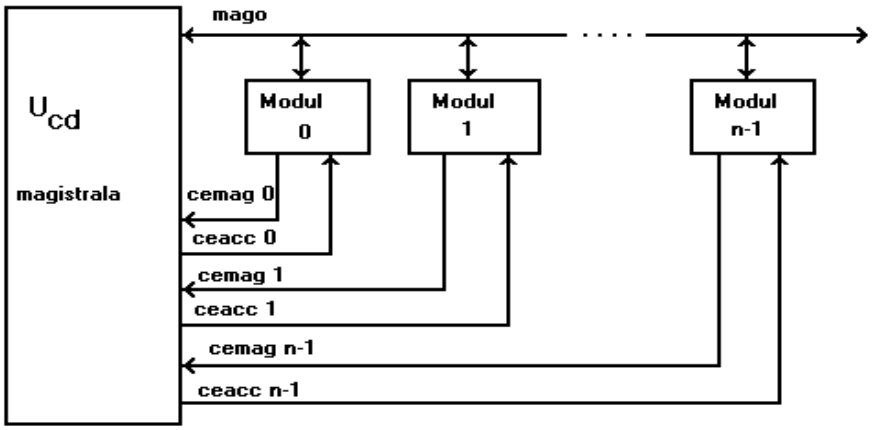



**Figura 4.5** Control distribuit cu cereri independente

Fiecare modul care doreşte acces la magistrală activează linia *cemag* asociată. Unitatea de comandă analizează aceste cereri şi pe baza unui algoritm de priorităţi prestabilit selectează modulul i cu prioritatea cea mai mare şi activează linia *ceacc i* corespunzătoare. Modulul i recepţionează *ceacc i* şi în momentul în care *mago* este inactivă, activează linia *mago*, dezactivează linia *cemag i* şi începe transferul. La terminarea ciclului de acces se dezactivează linia *mago* şi începe un nou ciclu de acces la magistrală.

Schema distribuită este prezentată în figura 4.6.

Liniile *cemag* sunt bidirecţionale. Fiecare modul care doreşte să facă acces la magistrală activează linia *cemag* corespunzătoare şi le testează pe celelalte. Dacă se identifică pe el ca fiind cu prioritatea cea mai mare când *mago* este inactivă, activează *mago*, dezactivează *cemag* şi începe transferul. La terminarea transferului *mago* este dezactivată.

După ocuparea magistralei de către un modul MASTER începe un dialog între modulele implicate în transfer pentru a realiza comunicaţia efectivă pe magistrală.

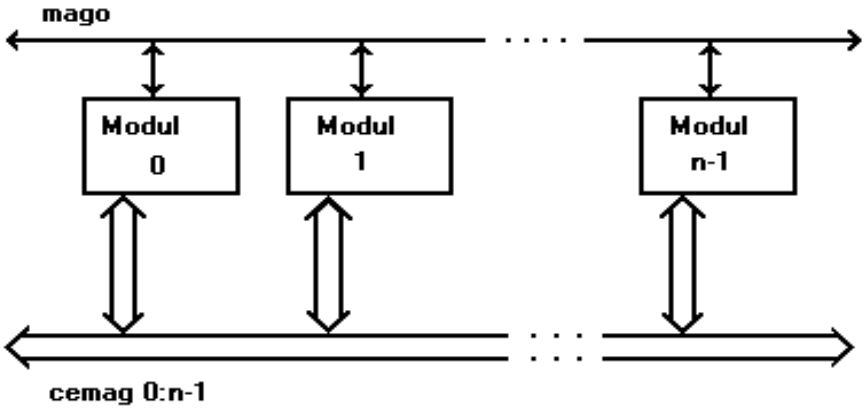



**Figura 4.6.** Schema distribuită cu cereri independente

## 4.2 Tehnici de comunicaţie pe magistrală

După ocuparea magistralei comunicaţia efectivă între modulele implicate în transfer

poate să fie de următoarele tipuri:

- sincronă;
- asincronă;

### 4.2.1 Comunicaţii sincrone

Trebuie precizat că nu este vorba de comunicaţii sincrone la distanţă între două calculatoare ci de comunicaţia pe magistrală între modulele unui calculator.

Un ciclu de transfer pe magistrală este divizat în intervale sincrone cu tactul unităţii de comandă. Unui modul i se poate asocia un astfel de interval (subciclu) în mod dedicat. Dacă un modul nu doreşte transfer, intervalul asociat nu este utilizat.

Dacă fiecărui modul MASTER i se asociază în mod dedicat un subciclu, unitatea de comandă a acesului la magistrală trebuie doar să sincronizeze cererile de acces cu subciclul asociat.

De exemplu să considerăm un sistem multi MASTER cu terminal grafic incorporat în care memoria de ecran este în acelaşi spaţiu de adrese cu memoria operativă, ciclul magistralei este împărţit în două subcicluri. Într-un subciclu au acces în mod concurent modulele MASTER iar în celălalt subciclu are acces interfaţa grafică pentru reîmprospătarea imaginii pe ecran.

Deci în cazul comunicaţiei sincrone un ciclu magistrală durează întodeauna acelaşi număr de perioade de tact iar acţiunile în fiecare perioadă sunt bine determinate şi sincrone cu tactul.

### 4.2.2 Comunicaţii asincrone

Comunicaţiile asincrone apar atunci când două module implicate într-un transfer evoluează independent, cu generatoare de tact distincte, fără o corelare prealabilă a evoluţiei lor în timp.

După modul în care modulele schimbă între ele semnale de dialog pentru efectuarea corectă a transferurilor asincrone se disting două tipuri de comunicaţii asincrone :

- fără răspuns sau cu comandă într-un singur sens;
- de tip întrebare - răspuns.

#### 4.2.2.1 Comunicaţii asincrone cu comandă într-un singur sens

Modulul care iniţiază transferul trimite un semnal de comandă şi consideră că celalt modul răspunde în mod corect. Este necesară cunoaşterea foarte bună a comportării în timp a celor două module. La rândul lor comunicaţiile asincrone fără răspuns pot să fie de două tipuri:

- controlul efectuat de modulul sursă;
- controlul efectuat de modulul destinaţie.

Comunicaţia cu controlul la sursă este prezentată în Fig.4.7

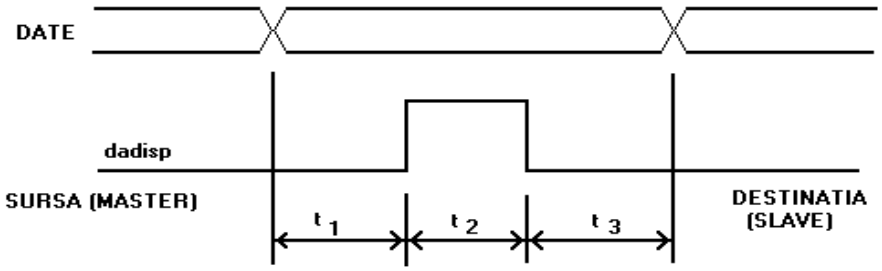



**Figura 4.7** Comunicaţia asincronă cu control la sursă

În acest caz t1, t2 şi t3 trebuie să fie bine precizate în funcţie de caracteristiciile magistralei şi comportarea în timp a celor două module. Dezavantajul acestei tehnici de comunicaţie constă în faptul că nu este posibil controlul validităţii transferului la receptie.

Comunicaţia cu controlul la destinaţie este prezentată în figura 4.8.

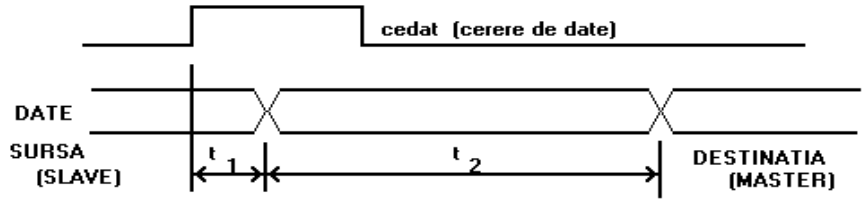



**Figura 4.8** Comunicaţia asincronă cu control la destinaţie

Modulul sursă primeşte *cedat* şi activează datele pe magistrală, dar modulul destinaţie trebuie să ştie valoarea intervalelor t1 şi t2 ceea ce nu este foarte simplu pentru orice modul SLAVE posibil.

În cazul sistemelor cu structură eterogenă în care nu se cunosc bine vitezele relative ale modulelor MASTER şi SLAVE se utilizează comunicaţia de tip întrebare-răspuns.

### 4.2.2.2 Comunicaţii asincrone de tip întrebare - răspuns

Prin utilizarea unui număr mai mare de linii de comandă se poate implementa un dialog mai sigur între cele două module care comunică conform schemei din figura 4.9.

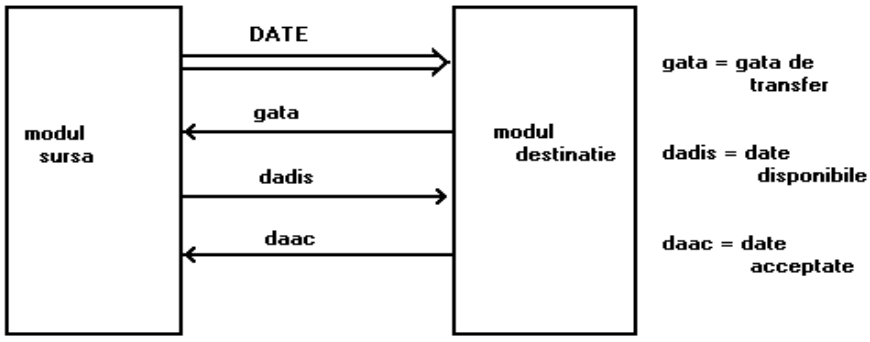



**Figura 4.9** Comunicaţia de tip întrebare răspuns

Prin activarea semnalului *gata* modulul destinaţie semnalează că este pregatit pentru a primi date. Modulul sursă văzând că *gata* este activ activează datele şi semnalul *dadis*. Modulul destinaţie preia datele apoi activează semnalul *daac* pentru a comunica modulului sursă încheierea ciclului de transfer.

În funcţie de modul în care se intercondiţionează cele trei semnale de dialog se obţin trei tehnici de comunicaţie asincronă de tip întrebare-răspuns:

- fără interblocare;
- cu semiinterblocare;
- cu interblocare completă.

Să considerăm organigramele parţiale pentru unităţile de comandă şi diagramele de timp prezentate în figura 4.10.

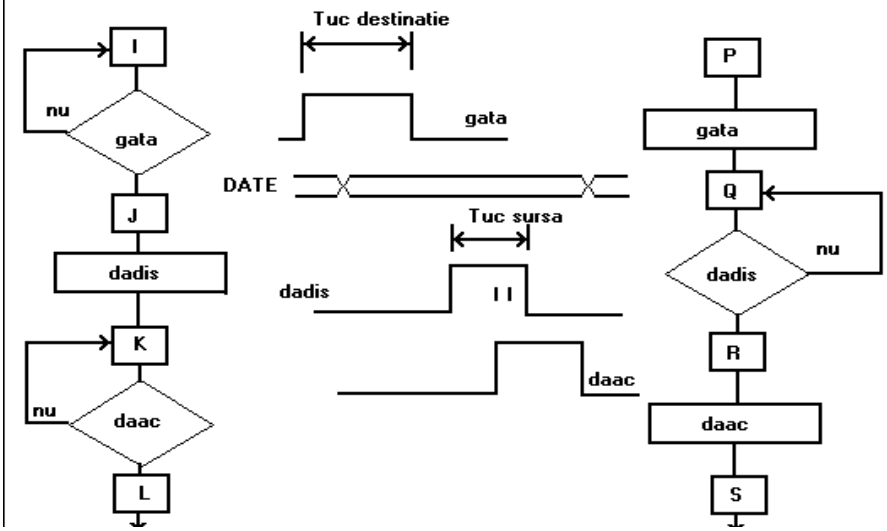



**Figura 4.10** Comunicaţia asincronă de tip întrebare-răspuns fără interblocare

Semnalele de ieşire sunt activate de către fiecare unitate de comandă pe durata unei perioade a tactului propriu presupunând că acest lucru este detectat de către cealaltă unitate care testează aceste semnale. Dacă perioadele semnalelor de tact ale celor două unităţi de comandă diferă iar registrele de stare sunt realizate cu elemente bistabile de tip D pot să apară următoarele situaţii de blocare:

**a) Tuc sursă < Tuc destinaţie**

UC destinaţie activează semnalul *gata* şi trece în starea Q asteptând activarea semnalului *dadis*. UC sursă detectează activarea semnalului *gata*, activează *dadis* în starea J pe durata Tuc sursă şi apoi trece în starea K unde aşteaptă activarea lui *daac*. Este posibil ca semnalul *dadis* să nu fie detectat de către destinaţie ceea ce conduce la blocarea sistemului în stările K şi Q.

Această situaţie se încadrează în

**1° Comunicaţia de tip întrebare- răspuns fără interblocare**

Soluţia este menţinerea activată a semnalului *dadis* până ce este luat în considerare de către UC destinaţie. Organigramele şi diagramele de timp sunt prezentate în figura 4.11.

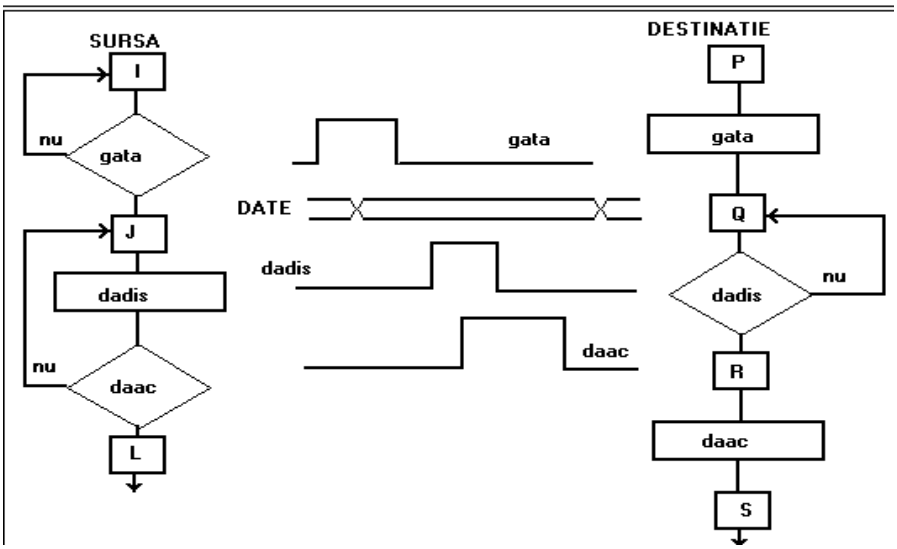



**Figura 4.11** Comunicaţie asincronă de tip întrebare-răspuns cu semiblocare

**b) Tuc sursă > Tuc destinaţie**

În acest caz este posibil ca UC sursa să nu sesizeze activarea semnalului gata şi sistemul se blochează în stările I şi Q sau sesizează activarea semnalului *gata* dar nu sesizează semnalul *daac.* Această restricţie poate să conducă la blocarea temporară a modulului sursă în starea I în timp ce modulul destinaţie preia aceleaşi date de mai multe ori.

Această situaţie se încadrează în

**2° Comunicaţia asincronă de tip întrebare-răspuns cu semiinterblocare**

Soluţia corectă se obţine prin interblocarea semnalelor *gata* şi *daac*.

**3° Comunicaţia asincronă de tip întrebare-răspuns cu interblocare completă**

Organigramele şi diagramele de timp sunt prezentate în figura 4.12.

Această schemă funcţionează corect indiferent de vitezele relative ale celor două unităţi de comandă şi poate fi utilizată într-o structura multi MASTER eterogenă.

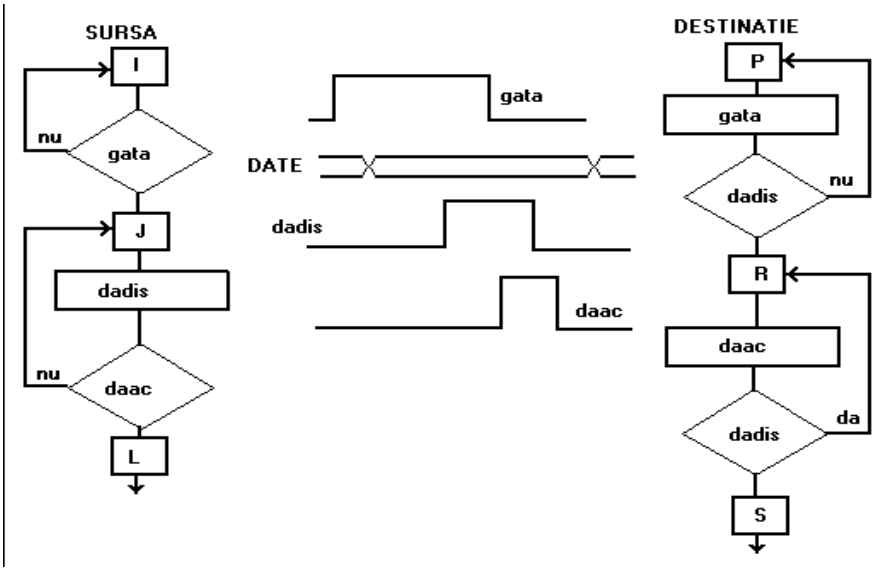



**Figura 4.12.** Comunicaţia asincronă de tip întrebare-răspuns cu interblocare completă

Dacă funcţiile semnalelor *gata* şi *daac* se reunesc într-un singur semnal se obţine din Fig 4.9 o schemă de comunicaţie de tip întrebare-răspuns cu numai două semnale de dialog. Deasemenea schema din Fig.4.9 poate să fie generalizată pentru o comunicaţie MASTER-SLAVE indiferent dacă modulul MASTER este sursă sau destinaţie. Astfel modulul MASTER emite o cerere de transfer şi informaţii privind sensul transferului, pe care le menţine active până la primirea unui semnal de achitare transfer de la SLAVE. Modulul SLAVE menţine activ semnalul de achitare transfer până la dezactivarea semnalului cerere transfer de către modulul MASTER.

## 4.3 Modalităţi de transfer a datelor

Utilizând o anumită metodă de control a accesului la magistrală, un modul MASTER ocupă magistrala şi utilizând o tehnică de comunicaţie pe magistrală adecvată, intră în dialog cu modulul SLAVE pentru a efectua transferul efectiv al datelor.

Transferul de date efectiv se poate desfăşura astfel :

- un singur cuvânt;
- un bloc de lungime fixă;
- un bloc de lungime variabilă;
- un singur cuvânt sau bloc de lungime fixă;
- un singur cuvânt sau bloc de lungime variabilă.

Alegerea unei modalităţi de transfer a datelor adecvată se face în funcţie de caracteristicile magistralei, tehnica de comunicaţie pe magistrală, caracteristiciile aplicaţiilor, etc.

Dacă magistrala asigură o viteză de transfer suficient de mare pentru a asigura servirea multiplexată a tuturor cererilor de transfer se preferă transferul cuvânt cu cuvânt. Dacă însă există module critice cu un timp de aşteptare limită mai mic decât ciclul magistralei trebuie aleasă o modalitate de transfer în blocuri de lungime fixă sau variabilă. În acest caz pe durata transferului întregului bloc magistrala este exclusiv alocată acestui transfer blocând alte

transferuri. Dacă această situaţie nu este tolerată de sistem soluţia este prevederea unui număr mai mare de magistrale care să asigure transferuri concurente.

## 4.4 Exemplu de sistem multiprocesor cu magistrală unică multiplexată în timp

Sistemul are o structură modulară, împărţirea făcându-se pe criterii funcţionale şi constructive. Considerăm că modulele din structura sistemului sunt de două tipuri:

- module "MASTER" care pot să iniţieze şi să controleze execuţia unui transfer de informaţii pe magistrală. Modulele din această categorie sunt de tip: UCP, DMA, canale de I/E, etc;
- module "SLAVE" care pot numai să răspundă la cererile de transfer din partea modulelor MASTER. Modulele din această categorie sunt de tip: memorii, interfeţe de I/E, etc.

Fiecare modul este conectat la o magistrală numită magistrala sistemului. În funcţie de complexitatea modulelor, în componenţa acestora poate să intre şi o magistrală internă.

Rezultă că schema bloc a sistemului are structura prezentată în figura 4.13.

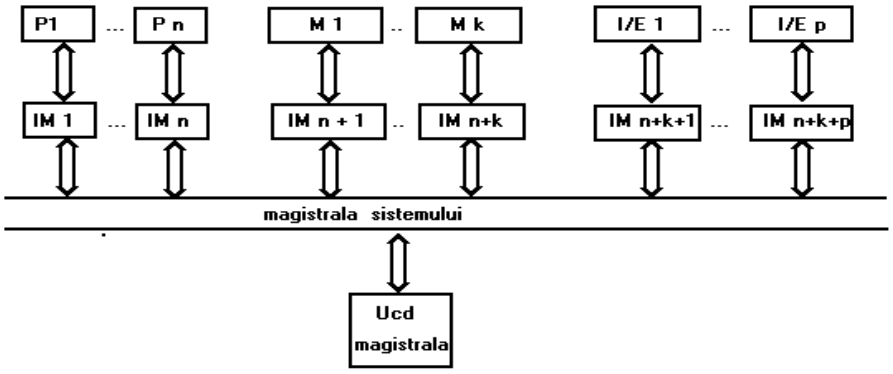



**Figura 4.13**. Schema bloc a sistemului multiprocesor

În figura 4.13. P1,..., Pn reprezintă module MASTER (UCP, DMA, canale de I/E), M1,..., Mk reprezintă module de memorie globală accesibile tuturor modulelor MASTER, iar I/E 1,..., I/E p reprezintă interfeţe pentru echipamente periferice, iar IM i reprezintă interfeţe de cuplare la magistrală.

Să considerăm că magistrala sistemului satisface următoarele cerinţe:

- liniile de interconectare sunt pasive ;
- unitatea de comandă a magistralei utilizează metoda de control centralizat cu cereri independente;
- tehnica de comunicaţie pe magistrală este asincronă de tip întrebare răspuns cu interblocare completă;
- pe magistrală se poate transfera un bloc de câte un cuvânt sau blocuri de lungime variabilă. Cu alte cuvinte, magistrala poate să fie ocupată şi eliberată pentru fiecare cuvânt care se transferă sau după ocuparea magistralei de către un modul MASTER acesta poate să păstreze controlul pentru transferul unui bloc de cuvinte, blocând accesul altor module MASTER la magistrală.

Fiecare modul de tip UC este dotat cu o serie de resurse locale (memorii, şi interfeţe de intrare/ieşire) reducându-se în acest mod interferenţele pe magistrala sistemului şi obţinându-se

o creştere importantă a productivităţii sistemului. Un astfel de modul are o structură internă similară celei prezentate în Figura4.14.

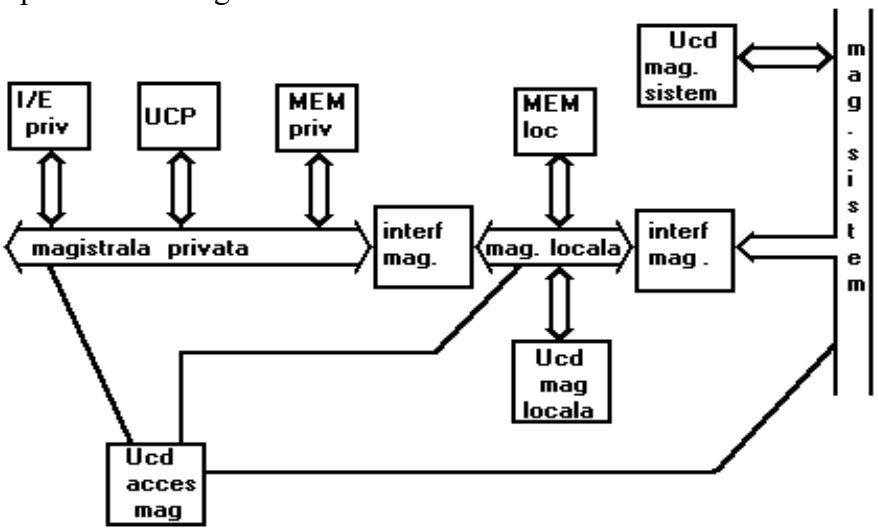



**Figura 4.14**. Structura internă a unui modul MASTER de tip UC

În figura 4.14 nu au mai fost reprezentate interfeţele de cuplare a modulelor la magistrale.

Se observă că un modul UC conţine resurse accesibile în mod exclusiv numite resurse private şi resurse accesibile şi celorlalte module din sistem numite resurse locale. În afara acestor resurse UCP are acces şi la o serie de resurse globale prin intermediul magistralei sistemului. Existenţa magistralei private distincte de magistrala locala rezultă din necesitatea de a permite UCP să realizeze accese la anumite resurse în timp ce din exterior alte module MASTER fac acces la memoria locală. În acest mod accesul la resurse se realizează printr-o ierarhie de magistrale asa cum se arată în Figura4.15.


Fig 4.15 Ierarhia de magistrale pentru un modul de tip UC

Accesul la resurse se face prin străpungerea nivelelor de magistrale respective în modul următor:

- UCP are acces la resursele private prin intermediul magistralei private. Acest acces se poate desfăşura în paralel cu alte transferuri care au loc pe magistrala sistemului sau cu accesul dinspre magistrala sistemului la memoria locală;
- UCP poate să facă acces la memoria locală prin intermediul magistralei private şi a magistralei locale în paralel cu transferurile care au loc pe magistrala sistemului;
- UCP poate să facă acces la memoria locală a altui modul de tip UC prin intermediul magistralei private, a magistralei locale proprii, a magistralei globale a sistemului şi a

magistralei locale a modulului respectiv.

Se observă că la memoria locală poate să facă acces atît UCP-ul cât şi alte module MASTER prin intermediul magistralei sistemului, corespunzător memoria locală trebuie să fie biport.

Dacă UCP doreşte să realizeze un transfer va genera o cerere de acces sub forma unor semnale de comandă. Pe baza acestor semnale de comandă şi a adresei emise de către UCP unitatea de comandă a magistralei locale determină dacă transferul se referă la resurse locale, private sau globale.

- dacă accesul se referă la o resursă privată, atunci transferul se va efectua utilizând numai magistrala privată;
- dacă accesul se referă la o resursă locală, atunci unitatea de comandă a magistralei locale va determina momentul în care se poate efectua transferul;
- dacă transferul se referă la o resursă globală, atunci unitatea de comandă a accesului la magistrala sistemului va determina momentul de timp la care se poate efectua transferul care va avea loc prin cooperarea cu unitatea de comandă a magistralei locale.

În situaţia în care la o resursă apar cereri concurente (de la UC şi de la magistrala sistemului) se va acorda prioritate accesului UCP.

## 4.5 Interconectarea în sistemele cu resurse multiple

În sistemele cu resurse multiple şi arhitectură paralelă, SIMD sau MIMD un rol deosebit în stabilirea performanţelor îl ocupă modul de interconectare a resurselor.

În structurile cu arhitectură paralelă este necesar să se asigure comunicaţia între procesoare şi accesul procesoarelor la module de memorie sau la module de intrare /ieşire. Comunicaţia trebuie să se desfăşoare în aşa fel încât fluxul de date şi comenzi să asigure maximum de paralelism.

Conexiunile între procesoare, între procesoare şi modulele de memorie sau modulele de intrare ieşire se realizează prin structuri de interconectare sau **reţele de comutare**.

Presupunând că este necesar sa interconectăm N resurse de un anumit tip cu M resurse de un alt tip, asigurând maximum de conexiuni paralele, max(N,M), se poate concepe o reţea de interconectare care asigură un set de conexiuni între mulţimea de resurse de intrare şi mulţimea resurselor de ieşire, Figura4.16.

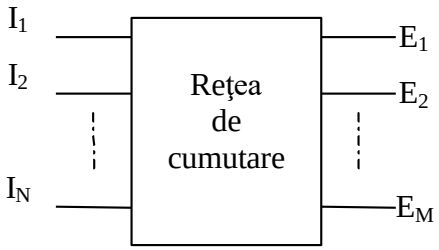



Figura4.16 Reţea de comutare N x M

- Considerând că pot avea loc interconectări care respectă regulile:
- o intrare poate fi conectată la oricare ieşire;

- o intrare poate fi conectată simultan la mai multe ieşiri;
- o ieşire poate fi conectată numai la o intrare la un moment dat;

Reţeaua de comutare trebuie să realizeze $N^M$ corespondenţe.

O reţea care asigură toate cele $N^M$ conexiuni constituie o reţea de conectare generalizată ( GCN - generalized connection network).

Dacă se stabilesc conexiunile astfel încât corespondenţa să fie bijectivă, reţeaua de conectare trebuie să realizeze (max(N,M))! corespondenţe. O astfel de reţea este cunoscută sub numele de reţea de comutare (CN connection network).

O reţea de conectare GCN poate fi reprezentată ca un graf în care arcele conectează perechile de resurse.

Reţeaua de conectare generalizată GCN se poate realiza:

- stabilind legături directe între intrări şi ieşiri (pe un singur nivel);
- prin utilizarea unor subreţele de dimensiuni reduse conectate ierarhic pe mai multe nivele;

În ambele situaţii trebuie să se aibă în vedere doi parametri:

- întârzierea prin GCN, definită ca numărul maxim de comutatoare elementare care asigură conexiunea intrare/ ieşire;
- numărul total de comutatoare elementare.

Pentru a proiecta o reţea de comutare se utilizează componenta de bază de tip cross-bar.

Un comutator elementar 2x2 de tip cross-bar asigură conexiunea intrărilor cu ieşirile în funcţie de semnalele de comandă, Figura4.17:

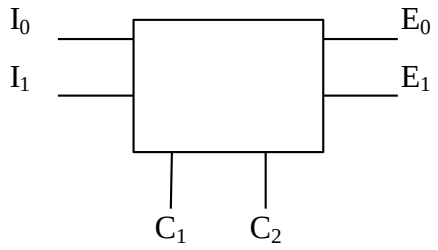



Figura4.17 Comutator elementar 2x2

Un comutator elementar de tip cross-bar asigură următoarele conexiuni:

-legătura directă;

-legătura interschimabată;

-broadcast superior;

-broadcast inferior;

aşa cum se arată în Figura4.18:

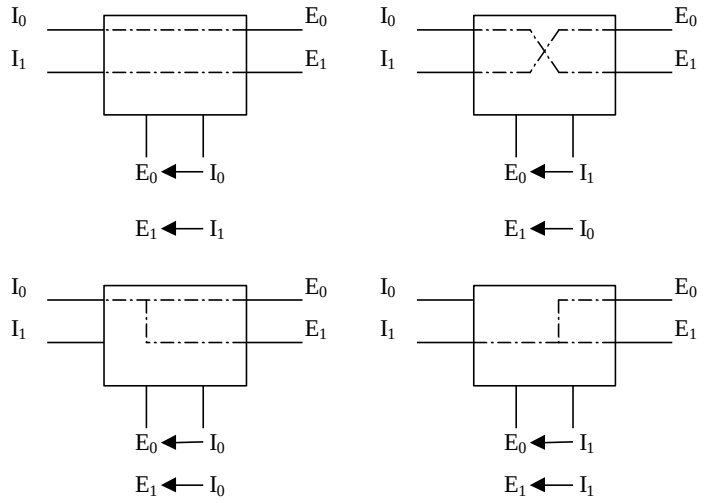



Figura4.18 Conexiuni asigurate de cross-bar

Unitatea de control a semnalelor de comandă tebuie să aibă în vedere ca o ieşire să fie conectată la o singură intrare la un moment dat.

Un comutator N x M de tip cross-bar asigură conectarea oricărei intrări $I_i$ la oricare ieşire $E_j$ are structura din Figura4.19.

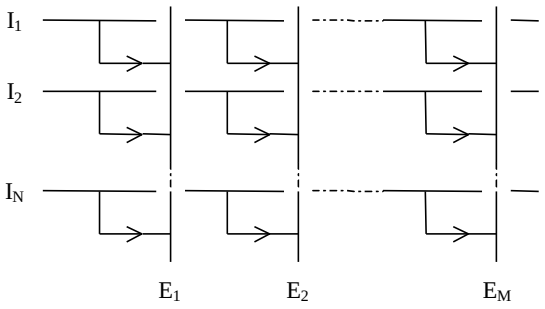



Figura4.19 Comutator cross-bar N x M

Aşa cum am precizat, unitatea de comandă rejectează mai multe cereri la aceeaşi ieşire.

O reţea de comutare de tip cross bar N x M, cu n intrări şi m ieşiri, necesită un număr de N x M comutatoare elementare, iar întârzierea este 1.

Din punctul de vedere al timpului de comutare este cea mai eficientă posibilitate de conectare. Are însă dezavantaje în ceea ce priveşte numărul mare de comutatoare elementare şi inflexibilitatea de dezvoltare ulterioară.

S-au dezvoltat reţele de comutare structurate pe mai multe nivele ierarhice, care reduc complexitatea în ceea ce priveşte numărul de comutatoare dar în schimb creşte timpul de comutare.

Înainte de a dezvolta reţele de comutare ierarhizate pe mai multe nivele vom analiza permutările fundamentale, care stau la baza acestora, şi care asigură comutarea în structurile paralele de tip SIMD cu o complexitate de O(n log n).

## 4.6 Permutări fundamentale

Presupunem că avem n resurse. O permutare specifică conexiunea între aceste resurse şi se poate defini ca o funcţie bijectivă pe o mulţime ordonată. Sunt cunoscute mai multe posibilităţi de a realiza permutările:

- permutare de bază (base line permutation);
- permutare cu intercalare perfectă (shuffle permutation);
- permutare cu negare bit (negate bit permutation);
- permutare fluture (butterfly);
- permutare cu ordine inversă (bit reversal permutation);
- permutare cu incrementare modulo (increment permutation);

Definirea acestora se poate face, aşa cum a propus Flanders, pe baza reprezentării binare a adreselor resurselor din mulţime.

Să considerăm reprezentarea binară a unei adrese:

ADR = ( $a_n$ $a_{n-1}$ ... $a_1$ $a_0$ )

### 4.6.1 Permutarea de bază

Permutarea de bază este realizată prin rotirea ciclică cu o poziţie la dreapta, a reprezentării binare a adresei. Rezultatul deplasării constituie adresa resursei cu care se va interconecta.

Permutarea de bază se defineşte :

p_baza(ADR) = p_baza($a_n a_{n-1}$ ... $a_1 a_0$) = ($a_0 a_n$ ... $a_2 a_1$)

Un exemplu de interconectare între resurse utilizând permutarea de bază se prezintă în Figura4.20.

Permutarea de bază poate avea două forme în funcţie de biţii de adresă care participă la deplasarea ciclică.

Sunt definite două sub permutări :

- permutare de baza k_inferior ( k-th sub base line permutation) în care participă, la deplasarea ciclică, cei mai puţini semnificativi k biţi ai adresei;
- permutare de baza k_superior ( k-th super base line permutation) în care participă, la deplasarea ciclică, cei mai semnificativi k biţi ai adresei;

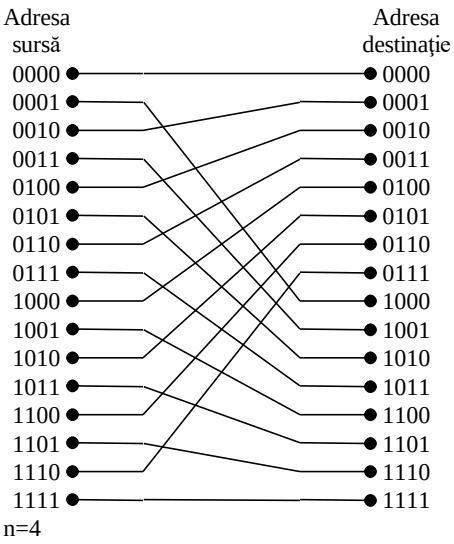



**Figura 4.20** Permutare de bază

### 4.6.1.1 Permutarea de baza k_inferior

Această permutare se defineşte astfel:

$p\_baza\_k\_inf(ADR) = p\_baza\_k\_inf(a_n a_{n-1} \ldots a_{k+1} a_k a_{k-1} \ldots a_1 a_0)$

$= (a_n a_{n-1} \ldots a_{k+1} a_k a_0 a_{k-1} \ldots a_2 a_1)$

şi implică numai cei mai puţin semnificativi k biţi ai adresei sursă. Exemplu în Figura 4.21.

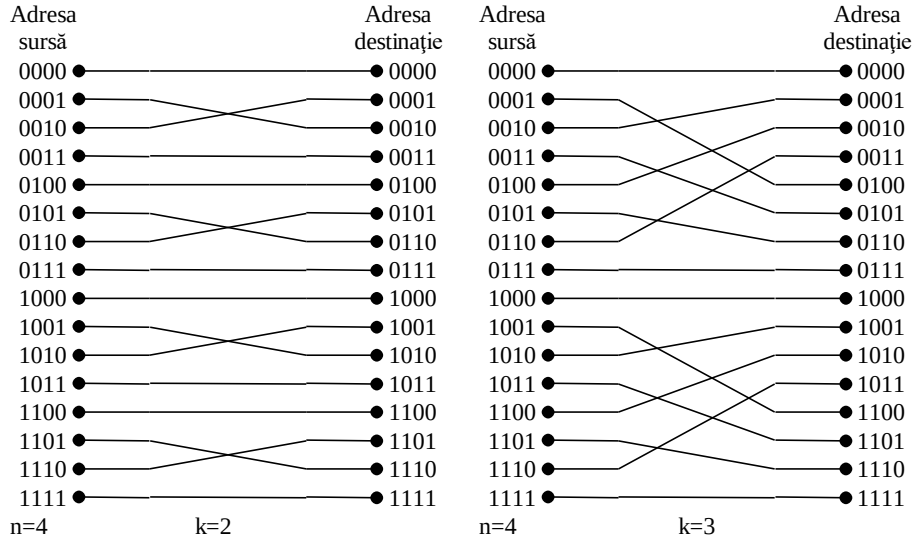



**Figura 4.21** Permutare de bază (k inferior)

### 4.6.1.2 Permutarea de baza k_superior

Această permutare se defineşte astfel:

$p\_baza\_k\_sup(ADR) = p\_baza\_k\_sup(a_n a_{n-1} \ldots a_{n-k+2} a_{n-k+1} a_{n-k} \ldots a_1 a_0)$

$= (a_{n-k+1} a_n \ldots a_{n-k+2} a_{n-k} \ldots a_1 a_0)$

şi implică numai cei mai semnificativi k biţi ai adresei sursă. Exemplu în Figura 4.22.

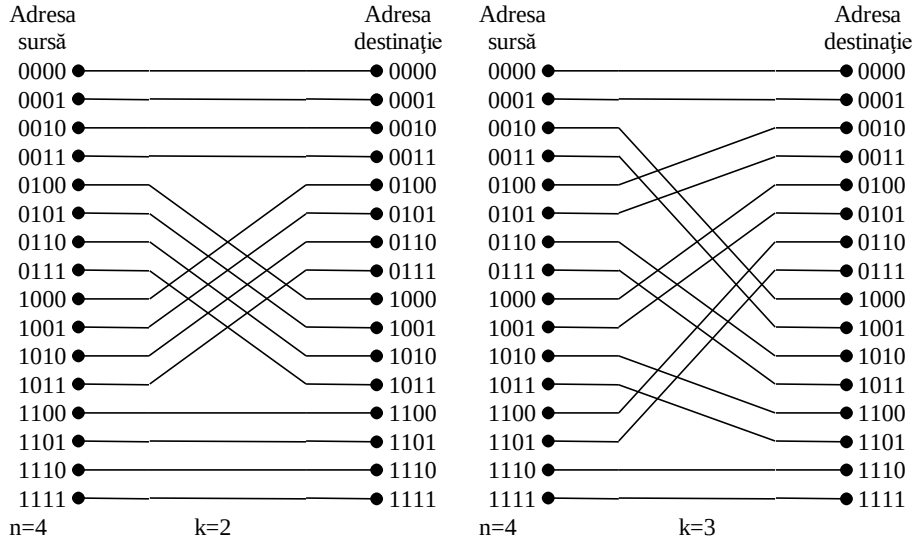



**Figura 4.22** Permutare de bază (k superior)

## 4.6.2 Permutarea cu intercalare perfectă

Permutarea cu intercalare perfectă este realizată prin rotirea ciclică cu o poziţie la stânga, a reprezentării binare a adresei. Rezultatul deplasării constituie adresa resursei cu care se va interconecta.

Permutarea cu intercalare perfectă se defineşte :

p_int_perf(ADR) = p_int_perf($a_n a_{n-1} \ldots a_1 a_0$) = ($a_{n-1} \ldots a_0 a_n$)

Un exemplu de interconectare între resurse utilizând permutarea cu intercalare perfectă se prezintă în Figura 4.23.

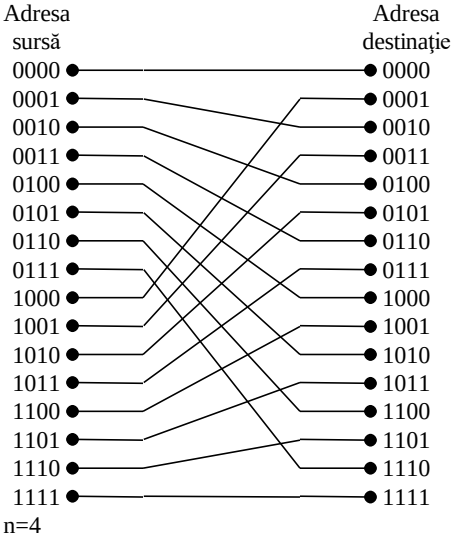



**Figura 4.23** Permutare cu intercalare perfectă

Permutarea cu intercalare perfectă poate avea două forme în funcţie de biţii de adresă

care participă la deplasarea ciclică.

Sunt definite două sub permutări :

- permutare cu intercalare perfectă k_inferior ( k-th sub shuffle permutation) în care participă, la deplasarea ciclică, cei mai puţin semnificativi k biţi ai adresei;
- permutare cu intercalare perfectă k_superior ( k-th super shuffle permutation) în care participă, la deplasarea ciclică, cei mai semnificativi k biţi ai adresei;

### 4.6.2.1 Permutarea cu intercalare perfectă k_inferior

Această permutare se defineşte astfel:

p_int_perf_k_inf(ADR) =

= p_int_perf_k_inf($a_n a_{n-1} \ldots a_{k+1} a_k a_{k-1} \ldots a_1 a_0$)

= ($a_n a_{n-1} \ldots a_{k+1} a_k a_{k-2} a_{k-3} \ldots a_1 a_0 a_{k-1}$)

şi implică numai cei mai puţin semnificativi k biţi ai adresei sursă. Exemplu în Figura 4.24.

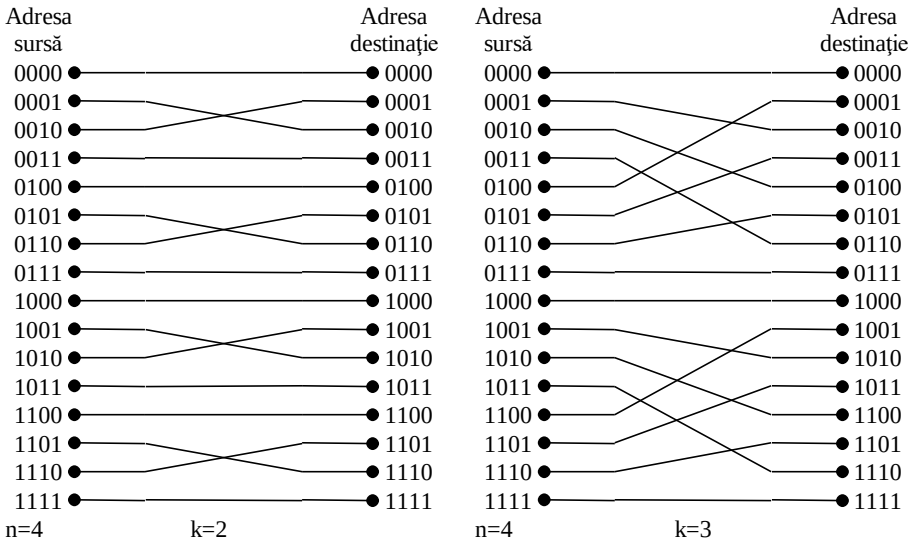



**Figura 4.24** Permutare cu intercalare perfectă (k inferior)

### 4.6.2.2 Permutarea cu intercalare perfectă k_superior

Această permutare se defineşte astfel:

p_int_perf_k_sup(ADR) =

= p_int_perf_k_sup($a_n a_{n-1} \ldots a_{n-k+2} a_{n-k+1} a_{n-k} \ldots a_1 a_0$)

= ($a_{n-1} a_{n-2} \ldots a_{n-k+2} a_{n-k+1} a_n a_{n-k} \ldots a_1 a_0$)

şi implică numai cei mai semnificativi k biţi ai adresei sursă. Exemplu în Figura 4.25.

Pentru a asigura o permutare mai generală, care să fie utilizată în reţele de comutare, structurată pe nivele de tip delta se poate defini o intercalare perfectă de tip q.

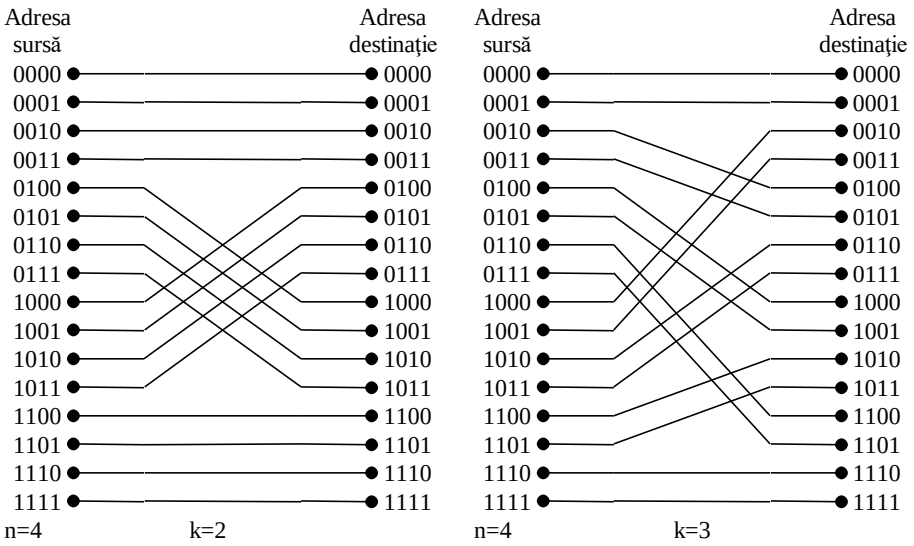



**Figura 4.25** Permutare cu intercalare perfectă (k superior)

Această permutare asigură o interconectare a q•c resurse şi se defineşte astfel:

p_int_perf_q•c(ADR) = $q \bullet V_{ADR} = [V_{ADR} / c] \bmod q \bullet c$

pentru $0 \le V_{ADR} \le q \bullet c - 1$
unde $V_{ADR}$ reprezintă valoarea adresei.

Altfel spus:

$$p\_int\_perf\_q \bullet c(ADR) = \begin{cases} q \bullet V_{ADR} \bmod (q \bullet c - 1) & \text{pentru} \quad 0 \le V_{ADR} \le q \bullet c - 1 \\ V_{ADR} & \text{pentru} \quad V_{ADR} \ge q \bullet c - 1 \end{cases}$$

Exemplu în Figura4.26.

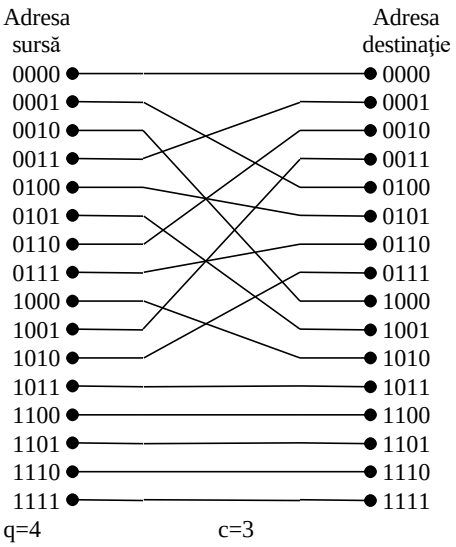



**Figura 4.26** Permutare cu intercalare perfectă de tip qc

### 4.6.3 Permutare cu negare bit

Permutarea cu negare bit este realizată prin negarea bitului cel mai puţin semnificativ, din reprezentarea binară a adresei. Rezultatul negării constituie adresa resursei cu care se va interconecta.

Permutarea cu negare bit se defineşte :

$$p\_neg(ADR) = p\_neg(a_n a_{n-1} \ldots a_1 a_0) = (a_n a_{n-1} \ldots\ldots a_1 \overline{a}_0)$$

Un exemplu de interconectare între resurse utilizând permutarea cu negare bit se prezintă în Figura 4.27.

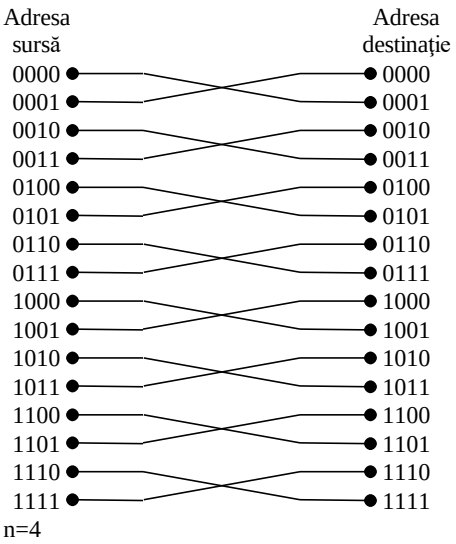



**Figura 4.27** Permutare cu negare bit

Permutarea cu negare bit poate avea o formă generalizată în care se neagă oricare din biţii reprezentării binare a adresei.

Este definită o permutare k bit în care se inversează bitul k din reprezentarea binară a adresei resursei care se interconectează.

Această permutare se defineşte astfel:

$$p\_neg(ADR) = p\_neg(a_n a_{n-1} \ldots a_{k+1} a_k a_{k-1} \ldots a_1 a_0) =$$
$$= (a_n a_{n-1} \cdots\cdots a_{k+1} a_k \overline{a_{k-1}} \cdots\cdots a_1 a_0)$$

Exemplu în Figura 4.28

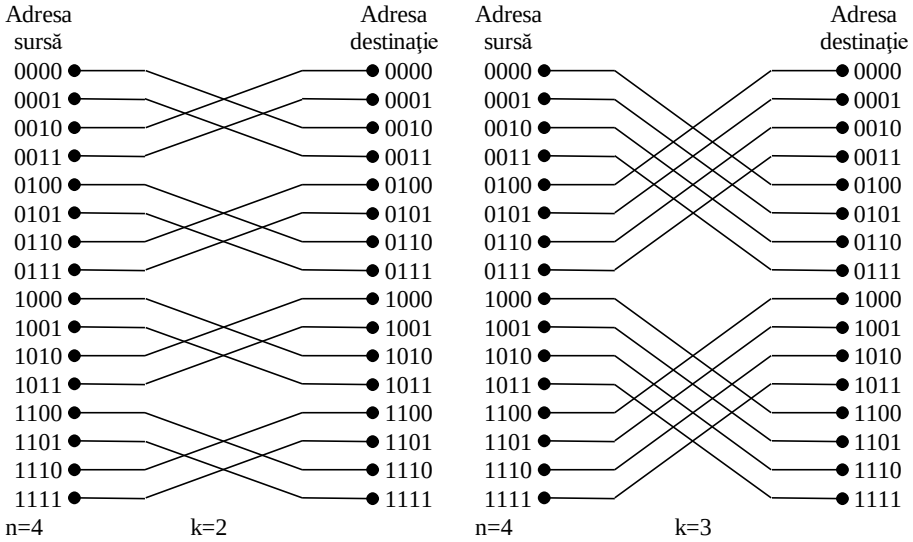



**Figura 4.28** Permutare cu negare de bit generalizată

### 4.7.4 Permutarea fluture

Permutarea fluture este realizată prin interschimbarea bitului cel mai puţin semnificativ al reprezentării binare a adresei cu bitul cel mai semnificativ. Rezultatul interschimbării constituie adresa resursei cu care se va interconecta.

Permutarea fluture se defineşte :
$p\_fluture(ADR) = p\_fluture(a_n a_{n-1} \ldots a_1 a_0) = (a_0 a_{n-1} \ldots a_1 a_n)$

Un exemplu de interconectare între resurse utilizând permutarea fluture se prezintă în Figura 4.29.

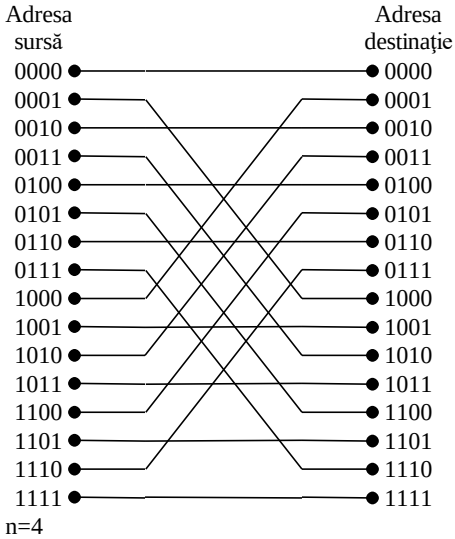



**Figura 4.29** Permutare fluture

Permutarea fluture poate avea două forme în funcţie de biţii de adresă care participă la interschimbare.

Sunt definite două sub permutări :
- permutare fluture k_inferior ( k-th sub butterfly permutation) în care participă, la interschimbare, bitul $a_{k-1}$ şi bitul $a_0$;
- permutare fluture k_superior ( k-th super butterfly permutation) în care participă, la interschimbare, bitul $a_n$ şi bitul $a_{n-k+1}$.

#### 4.6.3.1 Permutarea fluture k_inferior

Această permutare se defineşte astfel:
$p\_fluture\_k\_inf(ADR) = p\_fluture\_k\_inf(a_n a_{n-1} \ldots a_{k+1} a_k a_{k-1} \ldots a_1 a_0)$
$= (a_n a_{n-1} \ldots a_{k+1} a_k a_0 a_{k-2} \ldots a_2 a_1 a_{k-1})$
şi implică numai cei mai puţin semnificativi k biţi ai adresei sursă. Exemplu în Figura 4.30.

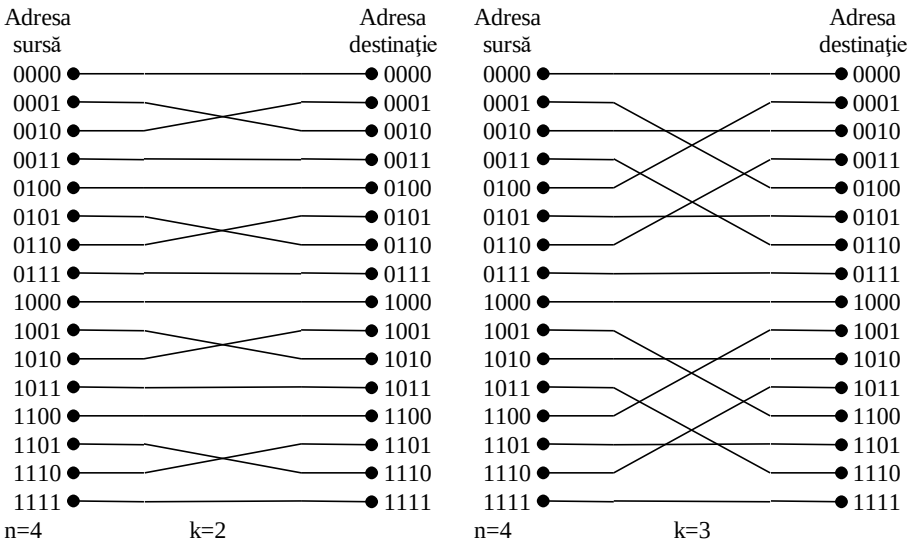



**Figura 4.30** Permutare fluture (k inferior)

### 4.6.3.2 Permutarea fluture k_superior

Această permutare se defineşte astfel:

$p\_fluture\_k\_sup(ADR) = p\_fluture\_k\_sup(a_n a_{n-1} \ldots a_{n-k+2} a_{n-k+1} a_{n-k} \ldots a_1 a_0)$

$= (a_{n-k+1} a_{n-1} \ldots a_{n-k+2} a_n a_{n-k} \ldots a_1 a_0)$

şi implică numai cei mai semnificativi k biţi ai adresei sursă. Exemplu în Figura4.31.

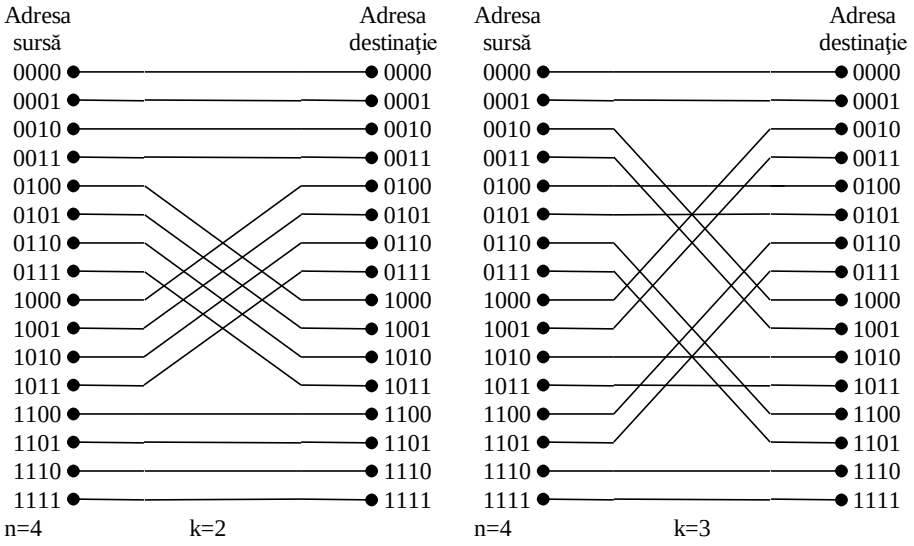



**Figura 4.31** Permutare fluture (k superior)

### 4.6.4 Permutarea cu ordine inversă

Permutarea cu ordine inversă este realizată prin inversarea ordinii biţilor din reprezentarea binară a adresei. Rezultatul inversării ordinii constituie adresa resursei cu care se va interconecta.

Permutarea cu ordine inversă se defineşte :

p_ord_inv(ADR) = p_ord_inv($a_n a_{n-1} \ldots a_1 a_0$) = ($a_0 a_1 \ldots a_{n-1} a_n$)

Un exemplu de interconectare între resurse utilizând permutarea cu ordine inversă se prezintă în Figura 4.32.

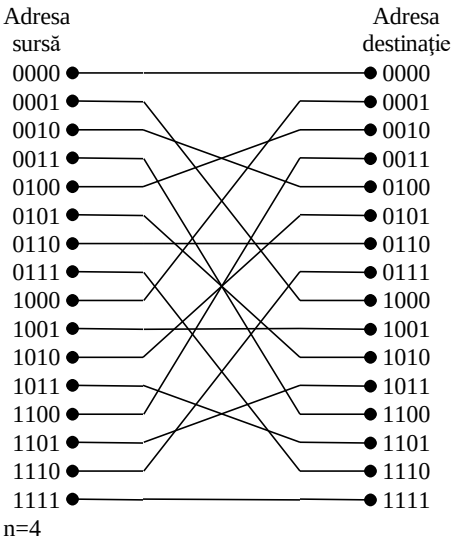



**Figura 4.32** Permutare cu ordine inversă

Permutarea cu ordine inversă poate avea două forme în funcţie de biţii de adresă căror ordine este inversată.

Sunt definite două sub permutări :

- permutare cu ordine inversă k_inferior ( k-th sub bit reversal permutation) în care participă, la inversarea ordinii, cei mai puţin semnificativi k biţi ai adresei;
- permutare cu ordine inversă k_superior ( k-th super bit reversal permutation) în care participă, la inversarea ordinii, cei mai semnificativi k biţi ai adresei;

### 4.6.4.1 Permutarea cu ordine inversă k_inferior

Această permutare se defineşte astfel:

p_ord_inv_k_inf(ADR) = p_ord_inv_k_inf($a_n a_{n-1} \ldots a_{k+1} a_k a_{k-1} \ldots a_1 a_0$)

= ($a_n a_{n-1} \ldots a_{k+1} a_k a_0 a_1 \ldots a_{k-2} a_{k-1}$)

şi implică numai cei mai puţin semnificativi k biţi ai adresei sursă. Exemplu în Figura 4.33.

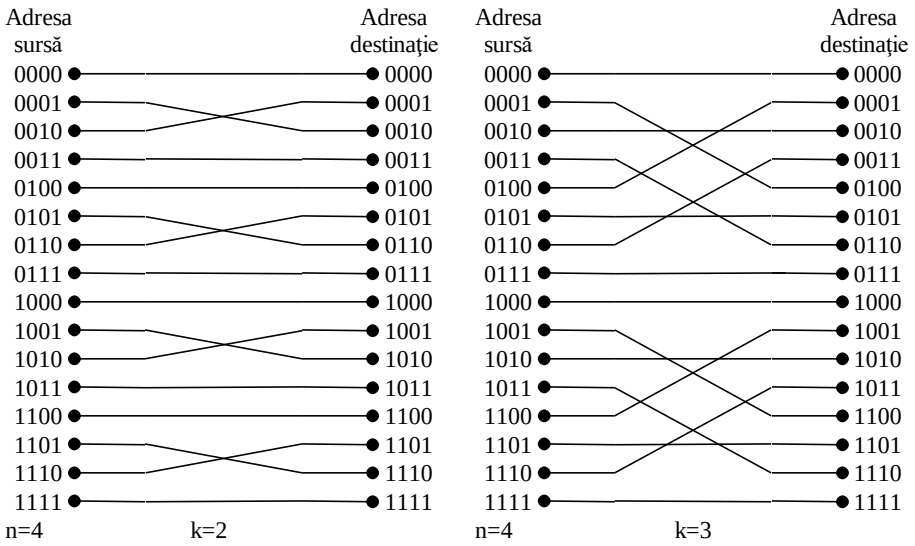



**Figura 4.33** Permutare cu ordine inversă (k inferior)

### 4.6.4.2 Permutarea cu ordine inversă k_superior

Această permutare se defineşte astfel:

p_ord_inv_k_sup(ADR) _ p_ord_inv_k_sup($a_n a_{n-1} \ldots a_{n-k+2} a_{n-k+1} a_{n-k} \ldots a_1 a_0$)
= ($a_{n-k+1} a_{n-k+2} \ldots a_{n-1} a_n a_{n-k} \ldots a_1 a_0$)

şi implică numai cei mai semnificativi k biţi ai adresei sursă. Exemplu în Figura 4.34.

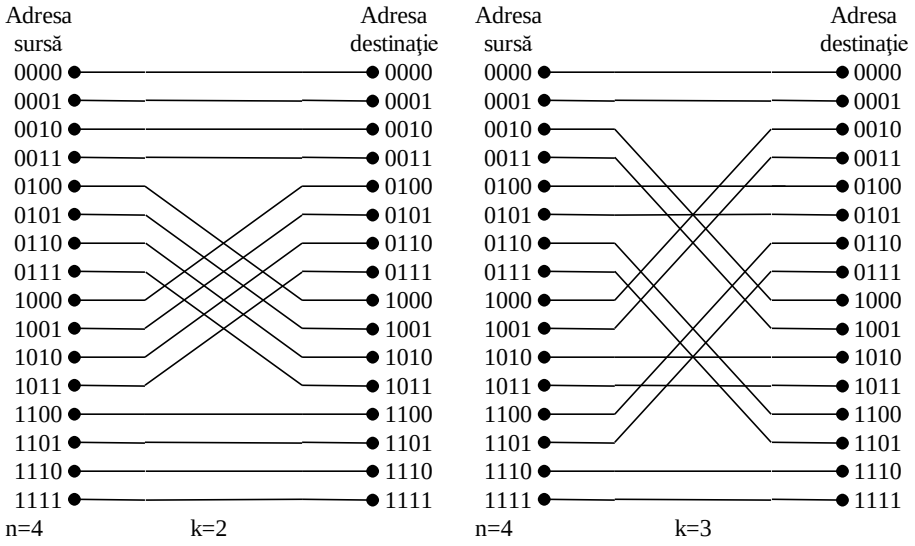



**Figura 4.34** Permutare cu ordine inversă (k superior)

### 4.6.5 Permutarea cu incrementare modulo

Permutarea cu incrementare modulo este realizată prin incrementarea modul $2^{n+1}$, a

reprezentării binare a adresei. Rezultatul incrementării constituie adresa resursei cu care se va interconecta.

Permutarea cu incrementare modulo se defineşte :

$p\_inc\_mod(ADR) = (V_{ADR} + 1) \bmod 2^{n+1}$

Un exemplu de interconectare între resurse utilizând permutarea cu incrementare modulo se prezintă în Figura 4.35.

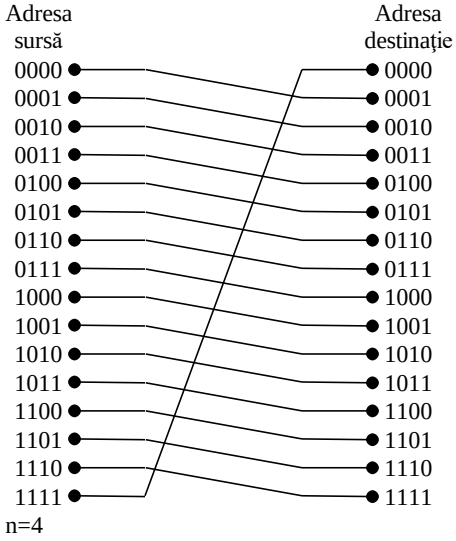



**Figura 4.35** Permutare cu incrementare modulo

Permutarea cu incrementare modulo poate avea două forme în funcţie de biţii de adresa care participă la incrementare modulo. Sunt definite două sub permutări :

- permutare cu incrementare modulo k_inferior ( k-th sub increment modulo permutation) în care participă, la incrementare modulo , cei mai puţin semnificativi k biţi ai adresei;
- permutare cu incrementare modulo k_superior (k-th super increment modulo permutation) în care participă, la incrementare modulo, cei mai semnificativi k biţi ai adresei;

### 4.6.5.1 Permutarea cu incrementare modulo k_inferior

Această permutare se defineşte astfel:

$p\_inc\_mod\_k\_inf(ADR) = (V_{ADR} + 1) \bmod 2^k + [V_{ADR} / 2^k] \bullet 2^k$

şi implică numai cei mai puţin semnificativi k biţi ai adresei sursă.
Exemplu în Figura 4.36.

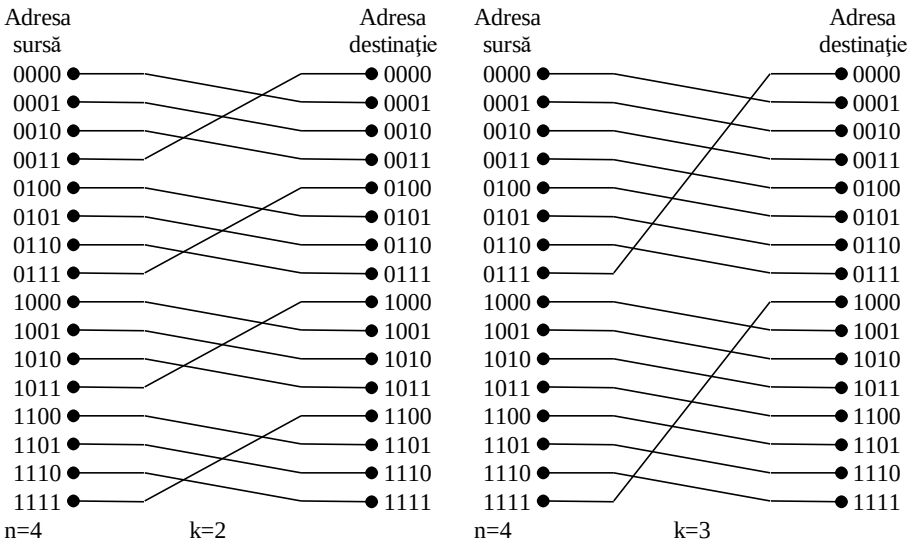



**Figura 4.36** Permutare cu incrementare modulo (k inferior)

### 4.6.5.2 Permutarea cu incrementare modulo k_superior

Această permutare se defineşte astfel:

$p\_inc\_mod\_k\_sup(ADR) = (V_{ADR} + 2^{n-k}) \bmod 2^n$

şi implică numai cei mai semnificativi k biţi ai adresei sursă. Exemplu în Figura 4.37.

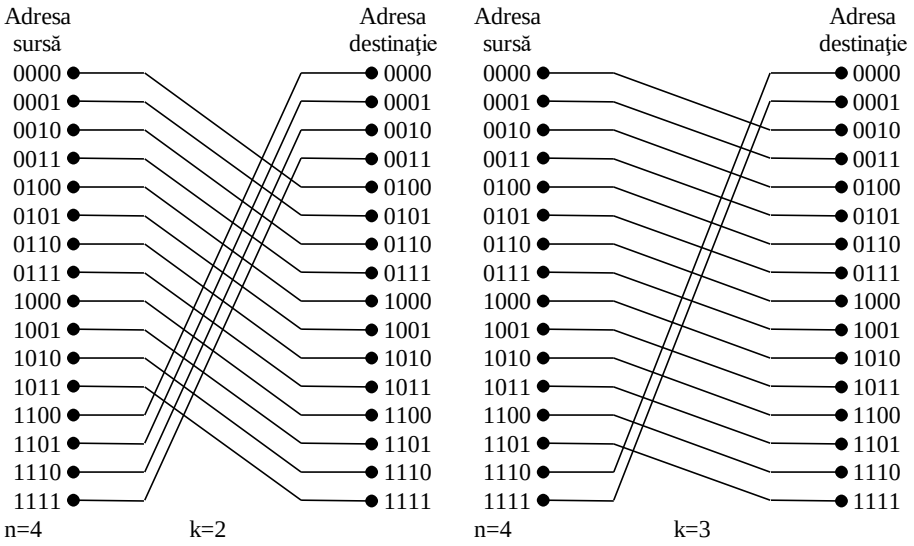



**Figura 4.37** Permutare cu incrementare modulo (k superior)

## 4.7 Reţele de comutare ierarhice

Beizer a propus o construcăie de comutare ierarhică de complexitate 4n log n - 2n, în care reţeaua de comutare cu n intrări este sintetizata din 2 subreţele cu n/2 intări plus 4n comutatore elementare.

Această clasă de reţele de comutare, denumită şi clasa de reţele de comutare de tip Benes, este bazată pe simularea unui cross bar de n x n cu două reţele cross bar de n/2 x n/2, Figura4.39.

Această simulare a fost utilizată pentru dezvoltarea a două tipuri de reţele de comutare.

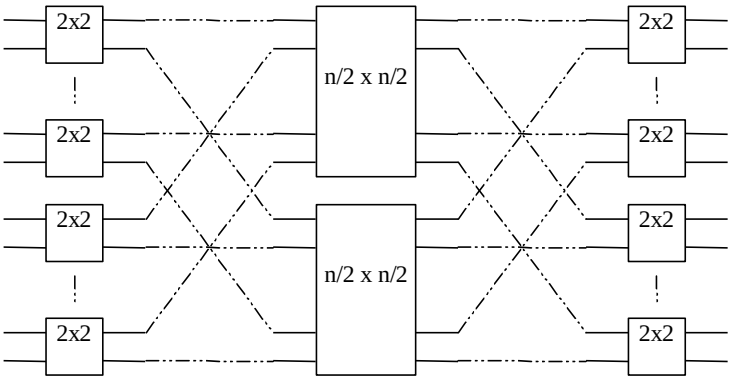



**Figura. 4.39** Reţea de comutare Beizer (Benes)

În Figura4.40 se prezintă structura de comutare pentru o reţea ierarhică generalizată pentru 8x8 resurse.

### 4.7.1 Reţele de comutare de tip DELTA

Pentru a realiza un comutator ierarhic de tip DELTA cu $a^n$ intrări şi $b^n$ ieşiri, se utilizează comutatoare de tip cross bar a x b şi reţele de permutare de tip intercalare perfectă.

Reţeaua de tip DELTA se realizează pe n nivele ierarhice.

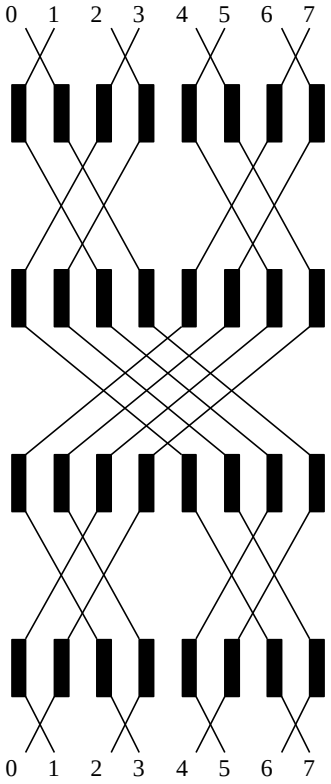



**Figura. 4.40** Reţea de comutare 8x8 generalizată

Pe primul nivel se folosesc $a^{n-1}$ reţele cross bar a x b, şi asigură conectarea a $a^n$ intrări la $a^{n-1}\bullet b$ ieşiri. Aceasta implică să avem la nivelul următor (nivelul doi) $a^{n-1}\bullet b$ intrări fiind necesare $a^{n-2}\bullet b$ module cross bar. În general la nivelul i sunt utilizate $a^{n-i}\bullet b^{i-1}$ module cros bar de dimensiune a x b.

Numărul total de module cross bar a x b, necesar pentru implementarea reţelei de comutare de tip DELTA este:

$(a^n - n^n)(a-b)$ pentru $a \neq b$

$n\bullet b^{n-1}$ pentru $a = b$

În Figura4.41 se prezintă schema generală a unui comutator de tip DELTA.

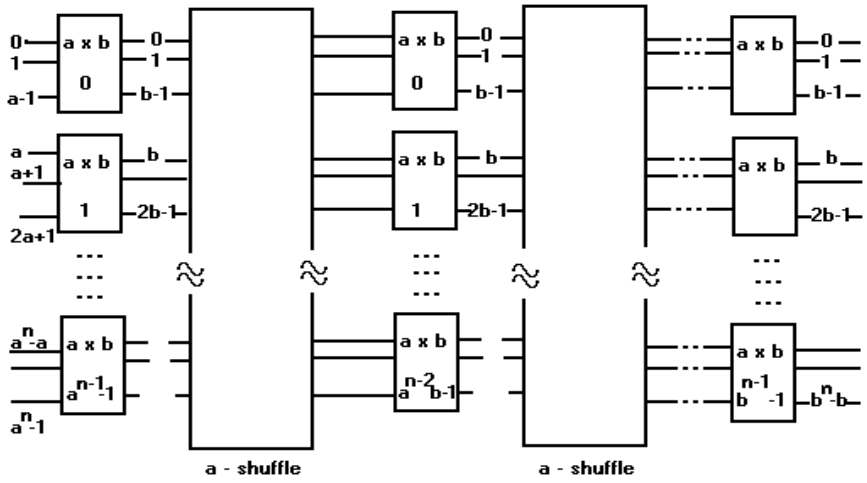


**Figura4.41** Reţea DELTA $a^n$ x $b^n$

Dacă destinaţia este $D=(d_{n-1}\ d_{n-2}\ ...\ d_1\ d_0)_b$ unde $0 \le d_i \le b$, atunci cifrele de reprezentare a adresei destinaţie în baza b vor controla modulul cross bar a x b se pe nivelele ierarhice. Astfel cifra $d_i$ controlează modulul de pe nivelul (n-i). Funcţia intercalării perfecte de tip a shuffle este de a conecta ieşirea unui nivel cu intrarea nivelului următor.

În Figura4.42 se prezintă un exemplu de reţea de comutare de tip DELTA, $4^2$ x $3^2$, care conectează 16 procesoare la 9 memorii, iar în Figura4.43 o reţea de interconectare a 8 procesoare cu 8 module de memorie realizată cu o reţea de tip DELTA $2^3$ x $2^3$.

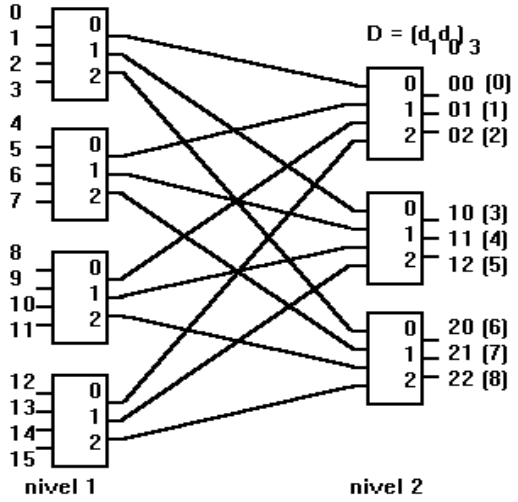



**Figura4.42** Exemplu de reţea de comutare de tip DELTA, $4^2x3^2$

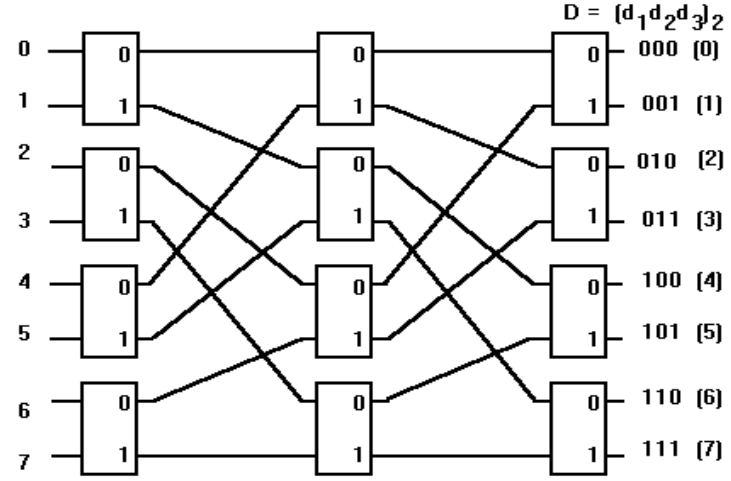



**Figura 4.43** Reţea de interconectare a 8 procesoare cu 8 module de memorie realizată cu o reţea de tip DELTA $2^3$ x $2^3$

Se observă ca biţii de reprezentare a destinaţiei în baza b stabilesc modul de control al modulelor cross bar intermediare de pe nivelele reţelei. Astfel pentru conectarea oricărui procesor la modulul de memorie cu adresa 3, $011_2$, la diverse momente de timp, în Figura4.43 primul nivel de comutatoare elementare asigură transferul spre ieşirea 0, al doilea nivel spre ieşirea 1, iar al treilea nivel spre ieşirea 1.

Sunt situaţii în care este necesară ca pe lîng_ o conectare a oricărui procesor de intrare la

oricare modul de memorie, să se realizeze o permutare identică adică în mod curent fiecare procesor i să fie cuplat la memoria i. Acest lucru este necesar atît pentru a considera o alocare cât de cât preferenţială a unui modul de memorie la un procesor, cât şi pentru procesul de testare /depanare. Această conexiune se realizează prin reţeaua de comutare din Figura4.44 în care procesorul 1 se conectează preferenţial cu modulul de memorie 1, procesorul 2 se conectează preferenţial cu modulul de memorie 2, etc.

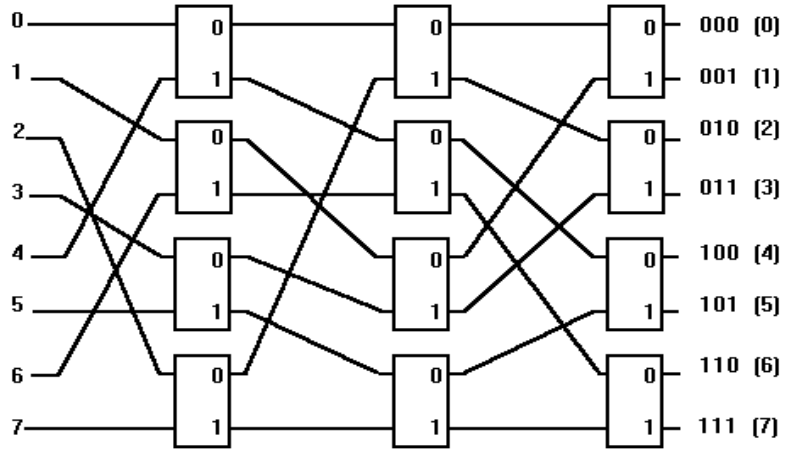



**Figura 4.44**

### 4.7.2 Performanţele reţelelor de comutare

Vom analiza performanţele unei reţele de comutare care asigură interconectarea a p procesoare şi m memorii.

Analiza performanţelor se va face pe cele două implementări ale reţelei de comutare şi anume: reţeaua de tip cross bar şi reţeaua de tip DELTA. Analiza se va face în vederea stabilirii ratei de transfer cu memoria.

În cele ce urmează vom utiliza noţiunile:

- *rata de servire* : media cererilor de acces la memorie acceptate într-un ciclu:
- *ciclu de servire* : timpul necesar pentru o cerere de acces la memorie să se propage prin logica de comutare la care se adaugă timpul de acces efectiv la memorie şi timpul de propagare inversă.

Pentru ambele tipuri de reţele de comutare vom cosidera aceleaşi presupuneri şi anume:

- nu vom face deosebire între ciclul de citire şi cel de scriere;
- fiecare procesor va genera cereri independente pentru un acces la un module de memorie. Se consideră că aceste cereri sunt uniform distribuite în cadrul tuturor modulelor de memorie;
- la începutul fiecărui ciclu fiecare procesor P generează o nouă cerere de acces cu probabilitatea r. Putem considera r media cererilor generate de fiecare procesor într-un ciclu;
- cererile neacceptate într-un ciclu sunt ignorate. Pentru simplificarea analizei presupunem că cererile de la ciclul următor sunt considerate independente de cele neacceptate în ciclul anterior (în realitate orice cerere neacceptată la un ciclu este menţinută în ciclul următor);

4.7.2.1 Analiza reţelei de comutare de tip cross bar

Presupunem un comutator de tip cross bar ce asigură interconectarea a *p* procesoare cu *m* memorii, de dimensiune pxm.

În reţeaua de comutare de tip cross bar se consideră că două cereri de acces sunt în conflict numai dacă se referă la acelaşi modul de memorie. Vom analiza mai degrabă conflictul la acelaşi modul de memorie decât conflictul de transfer prin reţeaua de comutare.

Fie :

r - probabilitatea ca un procesor să genereze o cerere de acces în cadrul unui ciclu;

q(i) - probabilitatea ca i cereri să sosească în timpul unui ciclu;

În acest caz:

$q(i) = C_p^i \, r^i (1-r)^{p-i}$ (1)

E(i) - numărul de cereri acceptate de reţeaua p x m în timpul unui ciclu, considerând că toate cererile au apărut în cadrul acestui ciclu.

Pentru evaluarea lui E(i) ne bazăm pe faptul că i cereri independente pot fi deservite în $m^i$ posibilităţi. Dacă presupunem că un modul nu este adresat, numărul de posibilităţi ar fi $(m-1)^i$

În acest fel $m^i - (m-1)^i$ este numărul de posibilităţi în care un modul oarecare este întotdeauna cerut.

Probabilitatea ca un modul oarecare să fie adresat este :

$(m^i - (m-1)^i)/ m^i$

Pentru fiecare modul de memorie, dacă acesta este adresat de mai multe procesoare simultan, reţeaua de comutare va accepta o singură cerere la un moment dat. Astfel:

$E(i)= ((m^i - (m-1)^i)/ m^i)*m$

Deci:

$E(i) = (1-((m-1)/m)^i)*m$ (2)

Rata de transfer cu memoria B(p,m) este definită ca media numărului de cereri acceptate şi se poate exprima astfel:

$B(p,m) = \sum_{0 \le i \le p} E(i) \bullet q(i)$

Forma simplificată este :

$B(p,m) = m - m \bullet (1-r/m)^p$ (3)

Probabilitatea de acceptare a unei cereri oarecare $P_A$ este definită astfel:

$P_A = B(p,m)/(r \bullet p) = m/(r \bullet m) - m/(r \bullet m) \bullet (1-r/m)^p$ (4)

În stabilirea relaţiei (4) s-a considerat că cererile sunt independente. În realitate o cerere rejectată la un anumit ciclu va fi transferată în ciclul următor mărind astfel rata de transfer.

Să analizăm ce se întâmplă în cazul în care numărul de resurse interconectate,

procesoare şi memorii, creşte foarte mult. Vom calcula limita când m -> ∞ .

$$\lim_{m\to\infty} (1-r/m)^p = e^{-r\bullet p/m}$$

Pentru valori foarte mari ale lui p şi m :

$$B(p,m) \approx m(1-e^{-r\bullet p/m}) \qquad (5)$$

iar

$$P_A \approx m/(r\bullet p)\bullet(1-e^{-r\bullet p/m}) \qquad (6)$$

Valorile exprimate de relaţiile (5) şi (6) sunt în limita unor erori de 1% dacă p şi m sunt ≥ 30 , şi în limita unor erori de 5% dacă p şi m sunt ≥ 8.

Pentru a analiza comportarea unui procesor vom utiliza un graf Marcov. Un procesor va avea două stări posibile:

- starea ACTIV (A), când cererea a fost acceptată;
- starea ASTEPTARE (W), când cererea a fost rejectată.

Fie :

$q_A$ probabilitatea ca procesorul să fie în starea A

$q_W$ probabilitatea ca procesorul să fie în starea W

$P_A$ probabilitatea de a fi acceptată cererea, adică procesorul să treacă din starea W în starea A.

În acest caz putem defini :

$$q_A = P_A / ( P_A + r\bullet(1-P_A))$$

$$q_W = 1 - q_A$$

Graful Marcov asociat comportării unui procesor este prezentat în Figura4.45.

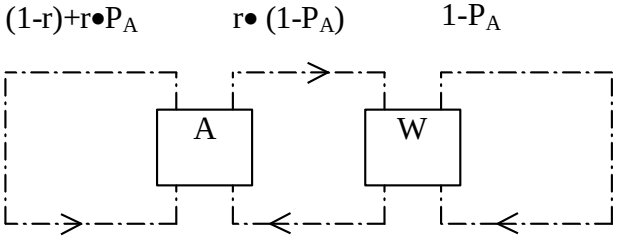



Figura 4.45 Graful Marcov pentru dinamica ratei de cereri r

S-a considerat că rata de cereri r este o mărime statică. În realitate rata de cereri r va fi definită mai precis dacă se iau în considerare şi conflictele de acces. Modulele de memorie vor primi un număr de cereri r' care este mai mare decât r datorită conflictelor de acces.

În acest caz :

$$r' = r\bullet q_A + q_W = r / (r+P_A\bullet(1-r)) \qquad (7)$$

$$P_A = m / (r'\bullet p) \bullet (1 - (1 - r'/m)^p) \qquad (8)$$

Relaţiile (7) şi (8) definesc un proces iterativ. Rata de transfer r' poate fi iniţializată cu r pentru începutul procesului iterativ.

Putem considera ca $P_A$ reprezintă o măsură a cererilor de acces nerezolvate. Dacă $P_A$ are o valoare mare rezultă un număr redus de stări de aşteptare.

Dacă considerăm $\overline{w}$ media numărului de cicli de aşteptare pentru o cerere, şi presupunem că o cerere care este rejectată de i ori va aştepta i cicli putem defini :

$$\overline{w} = \sum^{x} i \bullet (1-P_A)^i \bullet P_A = (1-P_A)/P_A \qquad (10)$$

În practică cererile care produc conflict de acces la un modul nu sunt rejectate ci sunt introduse într-o coadă de aşteptare şi servite pe baza unei discipline FIFO. Presupunerea cu rejecţia s-a făcut numai pentru a simplifica modelul.

### 4.7.2.2 Analiza unei reţele de tip DELTA $a^n$ x $b^n$

Presupunem reţeaua de tip DELTA de dimensiune $a^n$ x $b^n$ realizată din module de tip cross bar a x b. Cu această reţea de comutare interconectăm $a^n$ procesoare la $b^n$ module de memorie. Aplicăm rezultatele prezentate la reţeaua de tip cross bar în cadrul nivelelor ierarhice ale reţelei DELTA. Aşa cum s-a arătat anterior fiecare nivel al reţelei DELTA este controlat de biţii de reprezentare a adresei destinaţie codificată în baza b. De exemplu în cadrul unui nivel oarecare i, reţeaua cross bar respectivă utilizează biţii $d_{n-i}$ ai cererii şi acest bit nu mai este utilizat de nici un alt nivel în cadrul reţelei. De asemenea nivelul curent i, nu va folosi nici un alt bit de adresa în afară de $d_{n-1}$. În acest fel cererile la orice modul a x b sunt independente şi uniform distribuite.

Fiind dată rata de cereri r la fiecare intrare a din reţeaua de comutare a x b, considerând p=a şi m=b, relaţia (3) devine:

$$B(a,b) = b - b\bullet(1-r/b)^a \qquad (11)$$

Împărţind relaţia (11) cu numărul de ieşiri b, se obţine rata de cereri la fiecare ieşire din reţeaua a x b curentă.

În acest fel pentru fiecare nivel intermediar a reţelei DELTA, rata de cereri pentru fiecare ieşire $r_{out}$ este o funcţie de rata de cereri la intrare $r_{in}$ şi este stabilită de relaţia:

$$r_{out} = 1 - (1 - r_{in} / b)^a$$

Rata de cereri la ieşirea unui nivel constituie rata de cereri de intrare la nivelul următor. În particular rata de cereri la ieşirea nivelului ultim n, determină rata de transfer a reţelei de tip DELTA.

Fie $r_i$ rata de cereri la ieşirea unui nivel i. Rata de cereri $B(a^n,b^n)$ a unei reţele de tip DELTA generată de fiecare procesor este :

$$B(a^n,b^n) = b^n \bullet r_n$$

unde:

$$r_i = 1 - (1- r_{i-1}/b)^a \quad \text{şi } r_0 = r$$

Probabilitatea ca o cerere să fie acceptată este :

$P_A = b^n \bullet r_n / ( a^n \bullet r)$ (12)

Relaţiile (4) şi (12) specifice reţelelor de comutare de tip cross bar respectiv DELTA nu se pot compara direct. Vom considera un exemplu în care avem:

- pentru reţeaua de tip cross bar:
  - p x p crossbar, considerând p procesoare;
  - r=1 rata de cereri din partea fiecărui procesor.
- pentru reţeaua de tip DELTA, două cazuri:
  - a=2, b=2, $2^n$ x $2^n$
  - a=4, b=4, $4^n$ x $4^n$