1. STRUCTURI ŞI ARHITECTURI DE CALCULATOARE NUMERICE

1.1. Evoluţia calculatoarelor

1.1.1. Introducere
1.1.2. Generaţii de calculatoare
1.1.3. Forţe tehnologice şi economice

1.2. Modalităţi de reprezentare a calculatoarelor

1.2.1. Maşina von Neumann
1.2.2. Reprezentarea funcţională
1.2.3. Reprezentarea structurală

1.3. Clasificări arhitecturale

1.3.1. Clasificarea după controlul execuţiei
1.3.1.1. Calculatoare SISD
1.3.1.2. Calculatoare MISD
1.3.1.3. Calculatoare SIMD
1.3.1.4. Calculatoare MIMD

1.3.2. Clasificarea după organizarea spaţiului de adresă
1.3.2.1. Arhitecturi cu spaţiul de adresă al memoriei partajat
1.3.2.2. Arhitecturi cu transfer de mesaje

1. STRUCTURI ŞI ARHITECTURI DE CALCULATOARE NUMERICE

Calculatoarele au determinat o a treia revoluţie a civilizaţiei şi anume *revoluţia informaţiei* primele două fiind revoluţia agriculturii şi cea industrială. Calculatorul reprezintă un unicat între uneltele inventate de om în care creşterea performanţelor a fost însoţită în permanenţă de o reducere a preţurilor. Se apreciază că dacă un asemenea progres ar fi avut loc în industria automobilelor, astăzi s-ar putea ajunge de la New York la San Francisco în 5 secunde cu un preţ de 50 de cenţi.

Revoluţia calculatoarelor continuă şi se pare că nu există limite care să nu poată fi depăşite. Progresele în echipamente (*hardware)* au permis programatorilor dezvoltarea de aplicaţii *(software)* performante capabile să confere calculatorului valenţe şi implicări de neimaginat în urmă cu circa 25 de ani (Internet, e-comerce, poşta electronică etc.).

Obiectul prezentului capitol constă în prezentarea unor elemente definitorii care privesc structura şi arhitectura calculatoarelor numerice *(CN).*

1.1. Evoluţia calculatoarelor

Prezentarea propriuzisă a problematicii *CN* va fi precedată de o incursiune în istoria mijloacelor de calcul insistându-se asupra celor bazate pe tehnologiile electronice.

1.1.1. Introducere

Calculatorul numeric reprezintă un sistem fizic capabil să rezolve probleme prin execuţia unor instrucţiuni primite sub forma unui program. La nivelul unui ***CN*** pot fi derulate mai multe tipuri de prelucrări şi anume:

1. prelucrări de date;
2. prelucrări de informaţii;
3. prelucrări de cunoştinţe;
4. prelucrări inteligente (inteligenţă artificială),

ilustrate în structura piramidală din figura 1.1.

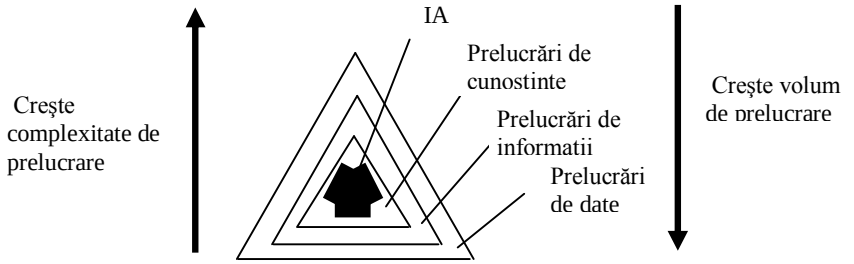


Fig. 1.1. Structura piramidală a tipurilor de prelucrări la nivelul unui *CN*.

1. *Spaţiul datelor.* Materialul prelucrat la baza piramidei poate fi considerat ca un spaţiu de **date** mutual independente. Acest spaţiu care este cel mai mare spaţiu al obiectelor prelucrate în CN include caractere, simboluri şi/sau reprezentări multidimensionale ale acestora, numere în diverse formate.

2. *Spaţiul informaţiilor.* Termenul de *informaţie* se asociază în acest context unei colecţii de date conectate printr-o anumită relaţie sau structură sintactică (sintaxa reprezintă un set de reguli care guvernează alcătuirea propoziţiilor într-un limbaj). Spaţiul informaţiilor se constituie într-un subspaţiu al datelor.

3. *Spaţiul cunoştinţelor.* În cadrul acestui spaţiu, care formează un subspaţiu al spaţiului informaţiilor, informaţiile sunt legate între ele printr-o structură semantică (semantica reprezintă un set de reguli care permit atribuirea de înţelesuri propoziţiilor într-un limbaj)

4. *Spaţiul prelucrărilor de tip inteligenţă artificială.* În cadrul acestui spaţiu (subspaţiu al spaţiului cunoştinţelor) se lucrează cu baze de cunoştinţe, reguli de inferenţă (raţionament) sau cu alte mijloace specifice domeniului *inteligenţei artificiale.*

După cum se va vedea mai departe toate calculatoarele realizate până în prezent evoluează pe baza unui program anterior memorat, program realizat sub forma unei secvenţe de instrucţiuni aferente unui limbaj artificial.

În ceea ce priveşte limbajele de programare acestea pot fi mai apropiate de maşina care le execută sau de utilizatorul uman. Gradul de apropiere se cuantifică în nivelul de percepţie al respectivului limbaj şi în capacitatea de *manevrare a instrucţiunilor aferente.*

Componentele fizice ale unui CN (circuitele electronice) nu pot recunoaşte şi executa decât un număr limitat de instrucţiuni. Instrucţiunile care pot fi înţelese şi executate direct (fără a necesita *translatare sau interpretare)* sunt instrucţiuni maşină iar limbajul corespunzător este *limbajul maşină* pe care îl vom nota *L1***.** Limbajul *L1* cu toate că permite comunicarea utilizatorului cu maşina este greu de folosit, iar în aplicaţiile de dimensiuni mari chiar imposibil. În aceste condiţii este necesară crearea unui nou limbaj, pe care îl vom nota cu *L2***,** mult mai apropiat de modul natural de gândire şi de operare al omului.

Din cele prezentate rezultă că utilizatorul poate scrie programe atât în *L1* cât şi în *L2*, dar calculatorul va executa întotdeauna instrucţiuni aferente limbajului **L1** pentru care a fost proiectat fizic. Pentru execuţia unui program scris în limbajul *L2* există două *tehnici* şi anume:

1. **translatarea (traducerea)** care presupune înlocuirea fiecărei instrucţiuni din *L2* cu instrucţiuni *L1*, rezultând un program în *L1* care va putea fi executat direct de maşină;

2. **interpretarea** care presupune analizarea fiecărei instrucţiuni din programul scris în *L2* şi execuţia ei printr-o secvenţă echivalentă de instrucţiuni din *L1*.

Având în vedere că utilizatorul lucrează cu o maşină căreia i se adresează în *L2* dar care execută în *L1*, să o numim **maşină virtuală** pentru a o deosebi de **maşina reală** căreia utilizatorul i se adresează în *L1*, iar execuţia se face tot în *L1*. *În general o* ***maşină virtuală*** *este o maşină capabilă să execute programe scrise în limbaje de nivel superior celui accesibil nivelului fizic.* Raţiunea de a fi a maşinii virtuale rezultă pe de o parte din dificultatea realizării fizice a unei maşini capabile să execute direct programe scrise în *L2*, iar pe de altă parte, din dificultatea utilizării directe a limbajului *L1*.

Dacă şi programarea în *L2* este dificilă se poate crea un alt limbaj *L3*, execuţia unui program scris în *L3* putându-se realiza prin aceleaşi două tehnici, respectiv:

1. traducerea programului într-un program echivalent scris în *L2*;

2. interpretarea fiecărei instrucţiuni din *L3* prin instrucţiuni din *L2***.**

Se poate spune că maşina virtuală având limbajul *L3* are la bază maşina virtuală cu limbajul *L2***.** Metoda se poate extinde pentru diferite limbaje şi maşini din ce în ce mai performante, numite în literatura de specialitate simplu **niveluri.**

Un calculator alcătuit din *n* niveluri conceptuale poate fi văzut ca *n* maşini virtuale distincte fiecare maşină având propriul său limbaj. Programele scrise în *L2, L3, …,Ln* trebuie să fie interpretate de un *interpretor* având un nivel mai mic sau să fie *translatate (traduse)* într-un limbaj inferior. Programatorul care are programele scrise pentru o maşină virtuală de nivelul *n* nu este interesat de translatoarele sau interpretoarele aflate la un nivel inferior.

În concluzie se poate spune că un calculator poate fi văzut ca o suită de niveluri, fiecare nivel înglobându-le pe cele precedente. În acest sens un nivel prezintă un anumit grad de abstractizare şi conţine diverse obiecte şi operaţii cu aceste obiecte. În sens ascendent creşte *complexitatea prelucrării* iar în sens descendent creşte *volumul materialului prelucrat.*

Pentru fiecare nivel maşină se defineşte **arhitectura calculatorului** ca fiind *totalitatea tipurilor de date, operaţiilor şi facilităţilor vizibile şi accesibile programatorilor.*

Structura calculatorului stabileşte şi defineşte componentele necesare realizării funcţiilor specificate. În principiu se poate vorbi de o abordare structurală şi în cadrul fiecărui nivel dar în general aceasta are în vedere maşina în ansamblul său. După cum se va vedea cvasitotalitatea calculatoarelor realizate până în prezent respectă structura definită de von Neumann în 1945 (se va reveni).

În afara noţiunilor de *arhitectură şi structură* CN mai sunt caracterizate şi prin noţiunile de *hardware, software, firmware.*

- Noţiunea **hardware** încadrează totalitatea componentelor fizice (electronice, electrice, mecanice etc.) aferente unui sistem de calcul. Practic caracteristicile *hardware* definesc maşina de nivel 0 care execută programul maşinii de nivel 1.
- Noţiunea **software** încadrează totalitatea programelor şi procedurilor care conferă unui calculator capacitatea de a executa sarcini specifice.

- Noţiunea **firmware** încadrează componente software nemodificabile încorporate de către fabricanţi în anumite dispozitive electronice.

Calculatorul reprezintă obiectul de studiu al ***Ştiinţei Calculatoarelor (Computer Science).*** Acest studiu are în vedere *proiectarea, modul de lucru şi utilizarea acestora* în cursul prelucrării datelor. Domeniul acoperit de *Ştiinţa Calculatoarelor* începe cu programarea şi arhitectura calculatoarelor şi ajunge la *Inteligenţă artificială* şi *Robotică.*

Legată nemijlocit de *Ştiinţa calculatoarelor* este *Ingineria calculatoarelor* care se ocupă cu studiul conceptelor care stau la baza proiectării componentelor *hardware* destinate calculatoarelor.

1.1.2. Generaţii de calculatoare

Un calculator modern reprezintă un sistem complex care înglobează componente *electronice, magnetice, electromecanice, electrono-optice, etc.* Urmărirea evoluţiei CN poate fi analizată din perspectiva asigurării resurselor, ilustrate în figura 1.2 pentru realizarea funcţiilor importante pentru acesta, şi anume:

- resurse de calcul şi comandă *(RCC)*;
- resurse de memorare *(RM);*
- resurse de introducere şi extragere a datelor *(RI/E);*
- resurse de comunicaţie *(RT);*
- resurse de programare ***(RP).***

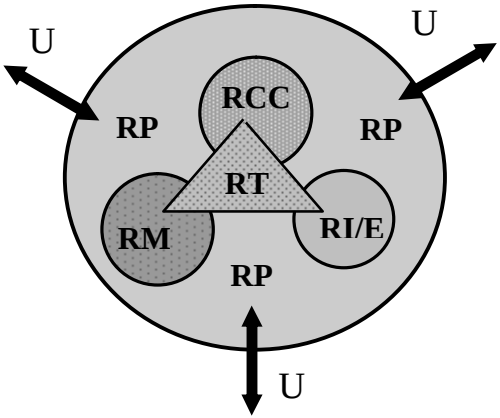


Fig. 1.2. Structura generală a unui sistem de calcul.

În evoluţia instrumentelor destinate realizării de operaţii aritmetice (în care sunt incluse şi *CN*) pot fi identificate 4 etape concretizate în 4 tipuri de maşini şi anume:

I. maşini manuale;
II. maşini mecanice;
III. maşini electromecanice;
IV. maşini electronice.

Evoluţia calculatoarelor a fost determinată de evoluţii în tehnologiile *furnizoare* de echipamente. Istoricii domeniului au delimitat mai multe *generaţii* marcate de salturi importante în tehnologiile majoritare utilizate. Pornind de la influenţa deosebită a electronicii în ceea ce priveşte performanţele calculatoarelor, toţi precursorii calculatoarelor electronice au fost grupaţi în aşa numita *Generaţie 0*, urmând ca *Generaţiile 1,2,3,4*.... să fie asociate numai calculatoarelor electronice. În continuare vor fi punctate aspecte remarcabile aferente evoluţiei calculatoarelor.

- **G0 - generaţia zero – (antichitate – 1945)**

În această generaţie sunt încadrate dispozitivele *manuale, mecanice, electromecanice.*

Maşini manuale . În rândul acestora pot fi incluse dispozitive de tip *riglă sau abac* (abacul este cunoscut în China de circa 2500 de ani).

Maşini mecanice. Meritul de a fi realizat prima maşină de calcul îi revine savantului francez **Blaise Pascal** când avea 19 ani**.** Această maşină realizată în 1642, era antrenată manual, conţinea angrenaje mecanice cu roţi dinţate şi putea efectua numai adunări şi scăderi. Unul dintre limbajele de nivel înalt importante a primit numele PASCAL, care în acelaşi timp este şi acronim pentru ***P**rogramme **A**pplique a la **S**élection et la **C**ompilation **A**utomatique de la **L**ittérature*) . Treizeci de ani mai târziu, în 1671, maşina lui Pascal a fost îmbunătăţită de către marele matematician german **Baron Gotfried Wilhelm von Leibnitz** prin adăugarea operaţiilor da înmulţire şi împărţire.

În 1830 **Charles Babbage** profesor de matematică la Universitatea Cambridge a construit *maşina analitică.* Aceasta conţinea patru părţi: memoria (magazia), unitatea de calcul, intrarea (cititorul de cartele perforate) şi ieşirea (perforatorul de cartele), toate în întregime mecanice. Marea noutate a acestei maşini era posibilitatea de prelua datele de intrare de pe cartele perforate şi de a întoarce rezultatele tot pe un asemenea

suport. Memoria putea stoca 1000 de cuvinte de 50 de cifre pentru variabile şi rezultate. Unitatea de calcul prelua operanzii din memorie, efectua diverse operaţii asupra lor (+, -, *, /) şi întorcea rezultatul tot în memorie. Acest mod de secvenţiere a operaţiilor conferă maşinii lui Babbage calitatea de prim precursor al calculatoarelor numerice de mai târziu. Un aspect interesant de relevat este acela că prima programatoare a acestei maşini a fost *Ada Lovelace,* fiica poetului englez *Lord Byron* .

Maşini electromecanice. Un pionier în domeniu este studentul german **Konrad Zuse**, care la sfârşitul anilor 1930 a construi mai multe maşini cu relee electromagnetice, care din păcate au fost distruse în timpul războiului.

Puţin mai târziu **John Atanasoff** de la *Colegiul Statului IOWA* şi **George Stibbitz** au proiectat de asemenea maşini de calcul cu componente electromecanice. Maşina lui **Atanasoff** folosea aritmetica binară şi avea memoria formată din capacităţi care trebuiau periodic reîncărcate. Din păcate această maşină nu a devenit operaţionaqlă niciodată. Calculatorul lui **Stibbitz** deşi mai primitiv decât al **Atanasoff** a funcţionat**, demonstraţia** având loc în 1940 la *Colegiul Darmouth.*

Între acestea importantă este maşina Mark 1 (*Automatic Sequence Controlled Calculator*) construită în 1944 de prof. *Howard Aiken* (Universitatea Harward) împreună cu specialişti de la IBM şi Bell Telephone, după un plan echivalent maşinii lui Babbage. Maşina conţinea relee, selectoare mecanice, cablaje). Memoria era de 72 de numere a câte 23 de cifre, efectua o adunare în 0,4 secunde şi o înmulţire în 6 secunde. Ca suporturi pentru introducerea/extragerea datelor utiliza cartele şi bandă perforată.

- **G1 - generaţia a întâia de CN – tuburile cu vid (1945 – 1955)**

Circuitele logice aferente *CN* din *G1* erau realizate cu tuburi electronice caracterizat de un consum energetic ridicat. Conţineau o unică memorie realizată cu tambur magnetic. Periferia era extrem de redusă şi se reducea la un cititor/perforator de hârtie. Reperoriul de instrucţiuni se reducea la 10-20 de instrucţiuni simple care de regulă defineau limbajul *L1* (cel mult *L2*), iar raportul timpilor în care se realiza o înmulţire respectiv o adunare era 20/1. După cum se observă

erau schimbări minore faţă de maşinile realizate de Aiken sau chiar Babbage, dar saltul semnificativ îl marca pătrunderea în era electronicii.

Cel de-al doilea război momdial a grăbit apariţia calculatoarelor, constituind din acest punct de vedere un factor favorizant. În consecinţă primele calculatoare de G1 aui avut o destinaţie militară şi au fost reprezentate de maşinile **ENIGMA** realizată de Germania pentru codificarea mesajelor ce urmau a fi transmise respectiv **COLLOSUS** realizat de armata britanică pentru deecodificarea mesajelor germane interceptate. La proiectarea sa a participat şi matematicianul englez *Alan Turing.*

În 1946 a devenit operaţional calculatorul **ENIAC – *E**lectronic **N**umerical **I**ntegrator **A**nd **C**omputer* realizat la University of Pensylvania de *John Mauchley şi J. Presper Eckert* şi destinat tot aplicaţiilor militare (calcularea caracteristicilor traiectoriilor balistice). Aflat în funcţiune până în anul 1955 ENIAC avea 18000 tuburi, 1500 de relee consuma 140 kW şi cântărea 30 de tone, fiind structurat în 30 de unităţi separate, care procesau în paralel datele. Din punct de vedere al arhitecturii conţinea 20 de registre a câte 10 cifre zecimale destinate rezultatelor parţiale sau finale. Performanţele sale erau "uluitoare" pentru acea vreme, putând efectua 5000 de operaţii/secundă, rezolvând în 20 de secunde probleme a căror soluţionare manuală necesita două zile.

Urmaşul lui **ENIAC** a fost calculatorul **EDVAC - *E**lectronic **D**iscrete **V**ariable **A**utomatic **C**omputer (realizat tot de către Mauchley şi Eckert)*** care elimina unităţile paralele de procesare şi conţinea numai 4000 de tuburi.

În timp ce *Mauchley şi Eckert* lucrau la **EDVAC**, unul dintre cei care fuseseră implicaţi în proiectul **ENIAC** şi anume matematicianul ungaro-american John von Neumann s-a mutat la Institutul Princeton pentru Studii Avansate în scopul de a construi propria versiune de **EDVAC** şi anume maşina **IAS.** El a descoperit că programarea calculatoarelor care conţin un număr mare de comutatoare şi cabluri era dificilă, lipsită de flexibilitate şi ca urmare ineficientă. El a înţeles că aceste neajunsuri pot fi eliminate prin memorarea programului în formă digitală împreună cu datele. De asemenea

a înlocuit aritmetica zecimală serială de la ENIAC[1] cu aritmetica binară paralelă. Proiectarea de bază pe care a descris-o prima dată într-un articol publicat în anul 1947 este cunoscută ca **maşina von Neumann.** Aceasta a fost folosită pentru prima dată la **EDSAC** care a reprezentat primul calculator cu program memorat.

Primul calculator destinat aplicaţiilor civile a fost **UNIVAC 1** lansat în 1951 de Eckert and Mauchly. Una din primele aplicaţii în care a fost implicat a fost predicţia, pe baza unui eşantion rezultatelor alegerilor prezidenţiale din SUA câştigate de Dwight Eisenhower.

Alte realizări care se pot menţiona în cadrul G1:

- **Whirlwind I** realizat la MIT (**M**assachusetts **I**nstitute of **T**echnology) şi destinat aplicaţiilor de conducere în timp real;
- **IBM 701** cu memorie de 2 kcuvinte a 36 de biţi şi două instrucţiuni pe cuvânt;
- **IBM 704** cu memorie de 4 kcuvinte a câte 36 biţi, instruccţiuni pe câte un cuvânt, procesare în virgulă mobilă (IBM – International Business Machine).

- **G2 - generaţia a doua de CN – componente discrete (1955 – 1965)**

În 1948 *John Bardeen, Walter Brattain şi William Shokley* de la Bell Telephone Laboratories au inventat tranzistorul, invenţie pentru care în 1956 au primit premiul Nobel pentru fizică. Apariţia componentelor electronice fără purtători mobili de sarcină electrică au condus la apariţia unei noi generaţii de *CN* căreia i se pot evidenţia următoarele caracteristici:

- utilizarea dispozitivelor semiconductoare (diode şi tranzistoare cu Ge şi apoi cu Si) ce ofereau un gabarit redus, putere disipată mai mică, siguranţă în funcţionare mai ridicată, eliminarea fenomenelor de radiaţie;
- memorii pe inele de ferită (din 1959) cu timpi de acces de ordinul a 2-12 μs (de 1000 de ori mai rapidă decât memoria pe tambur magnetic);
- interconectarea componentelor era realizată pe cablaj imprimat, aspect ce a permis modularizarea şi implicit uşurarea activităţii de întreţinere şi depanare;
- perfecţionări în domeniul echipamentelor periferice prin apariţia unităţilor de discurilor şi bandă magnetică, primele imprimante etc.;

[1] Fiecare rang zecimal era reprezentat de către 10 tuburi dintre care unul era activ şi 9 inactive.

- performanţele cresc în condiţiile reducerii costurilor (trimp înmulţire / timp adunare = 10/1 ;
- apar primele versiuni ale limbajelor de nivel înalt
  - FORTRAN (**FOR**mula **TRAN**slation) dezvoltat între anii 1954-1958 de Jim Backus şi destinat aplicaţiilor ştiinţifice;
  - COBOL (**CO**mmon **B**usiness-**O**riented **L**anguage**)** dezvoltat între anii 1959-1961 la solicitarea Departamentului Apărării a SUA şi destinat aplicaţiilor în care se prelucrează volume mari de date;
  - ALGOL (**ALGO**rithmic **L**anguage) lansat în 1959, a fost primul limbaj structurat, utilizat pe scară largă în Europa.

Primul CN cu tranzistoare a fost **TX-0** (***T**ranzistorised **eX**perimental computer **0**)*, realizat la MIT ca o continuare a calculatorului Whirlwind.

În 1961 firma **DEC** (**D**igital **E**quipment **C**orporation) a lansat calculatorul **PDP-1 (P**rogrammable **D**ata **P**rocessor) cu o memorie internă de 4 Kcuvinte a 18 biţi şi ciclul instrucţiune de 5 μs. O premieră absolută a lui PDP 1 o constituie otilizarea ecranului cu tub catodic (**CRT – C**athode **R**ay **T**ube) cu posibilitatea de control a fiecărui punct de pe ecranul cu rezoluţie de 512x512.

După câţiva ani tot DEC a introdus calculatorul PDP-8 caracterizat de o magistrală unică la care erau conectate următoarele module (figura 1.3) : *UCP, memoria, consola, unităţile de itrare/ieşire pe bandă de hârtie, alte dispozitive de I/E.* A fost primul CN realizat în mare serie, fiind vândute în jur de 50000 de sisteme.

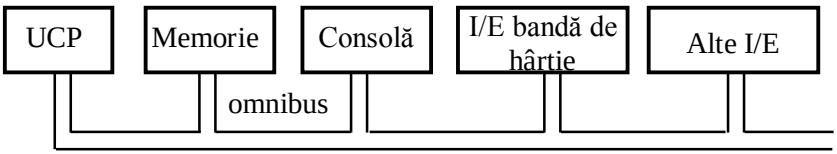


Fig. 1.3. Structura modulară a calculatorului PDP 8.

În 1964 DEC a lansat calculatorul **CDC 6600** (*proiectant Seymour Cray* - cel care mai târziu avea să realizeze seria de calculatoare *CRAY*).

IBM s-a impus prin calculatoarele **IBM 7094** (32 kcuvinte de 36 biţi, ciclu instrucţiune de 2 μs) şi în special prin calculatorul comercial **IBM 1401.**

În România au existat preocupări de dezvoltare a unor CN unicat din primele două generaţii cum ar fi **DACICC**, **MECIPT**, **ANCA**, **CETA** la Institutul de Fizică Atomică, Institutele Politehnice din Timişoara, Cluj, Bucureşti etc.

- **G3 - generaţia a treia de CN – circuite integrate (1965 – 1980)**

Apariţia acestei generaţii a fost determinată de inventarea circuitelor integrate[2] (*CI*), care grupau pe o pastilă de siliciu mai multe componente. Integrarea a făcut posibilă construirea unor calculatoare mai mici, mai rapide şi mai ieftine decât predecesoarele lor cu tranzistoare.

Principalele caracteristici ale CN din această generaţie sunt:

- utilizarea *CI* pe scară redusă (**SSI** - **S**mall **S**cale **I**ntegration) cu până la 100 tranzistoare / chip;

- utilizarea memoriilor semiconductoare cu timpi de acces dre ordinul a 0,5 – 1,75 μs;

- memorii externe de mare capacitate: discuri de masă capabile să stocheze până la 1 MB de informaţie.

Sistemele care s-au impus atenţiei din această generaţie au fost **IBM 360** şi **PDP 11.**

Principalele caracteristici ale lui **IBM 360** au fost: *ciclul instrucţiune de 250 nsec, memoria de până la 512 kcuvinte a câte un octet,spaţiul de aresare 2^{16} octeţi (16 Mocteţi), registre de lucru interne pe 32 de biţi, portabilitatea programelor în limbaj de asamblare pentru diferite versiuni.* Este primul calculator pe vcare s-a aplicat *multiprogramarea,* care permite existenţa în memorie şi execuţia simultană (sau cvasisimultană) a mai multor programe. Maşina 360 a fost, de asemenea, prima care putea emula (simula) alte calculatoare.

Calculatorul **PDP 11** realizat de DEC avea multe trăsături apropiate de IBM 360 (cu deosebirea că procesa cuvinte pe 16 biţi) însă avea un raport performanţă/cost mult mai bun. A fost considerat cel mai performant calculator al acestei generaţii, având un succes enorm în special în universităţi.

[2] Robert Noice în anul 1958.

În România s-au produs calculatoarele **FELIX C** la ICE Bucureşti. Primul calculator de proces a fost **FELIX C32P**. Începând cu 1977 s+au produs calculatoarele **INDEPENDENT şi CORAL** compatibile cu PDP 11.

- **G4 - generaţia a patra de CN – circuite VLSI (după 1980)**

Apariţia acestei generaţii a fost posibilă datorită perfecţionării în primul rând a tehnologiilor din industria electronică şi se caracterizează prin următoarele aspecte importante:

- utilizarea masivă a circuitelor integrate pe scară foarte largă (***VLSI* – V**erry **L**arge **S**cale **I**ntegration) – şi în primul rând a ***microprocesoarelor*** care integrează milioane de tranzistoare pe un cip şi care prezintă timpi de comutaţie de ordinul ns.;

- dezvoltarea de noi tipuri de memorii (MOS, magnetice, holografice) şi echipamente periferice orientate pe sesizarea primară a datelor;

- interconectarea calculatoarelor în reţele însoţită de întrepătrunderea industriilor de calculatoare şi de telecomunicaţii;

- apariţia şi dezvoltarea mediilor de programare complexe cu puternice facilităţi grafice.

Odată cu generaţia a IV-a calculatorul devine un instrument individual de lucru, accesibil ca preţ şi cu performanţe care erau de neimaginat pentru vechile generaţii.

Primele calculatoare personale se vindeau sub formă de kituri (realizate în jurul microprocesorului 8080), fără nici un fel de software. Mai târziu sistemul de operare *CP/M*[3], orientat pe operarea diskului flexibil care permitea comenzi de la tastatură a devenit destul de răspândi pe sisteme cu 8080. Un alt calculator timpuriu a fost Apple, şi mai târziu Apple II, proiectate şi realizate de Steve Jobs şi Steve Wozniak în celebrul lor garaj.

Întrucât microprocesorul (μP) a avut un cuvânt greu de spus în ceea ce priveşte performanţele generaţiei a IV-a în continuare vor fi prezentate

[3] Realizator Gary Kidall.

câteva elemente referitoare la evoluţia μP produse de două dintre cele mai importante firme şi anume **Intel** şi **Motorola.**

În continuare, din considerente istorice, sunt menţionate câteva repere semnificative ale evoluţiei microprocesoarelor din nfamilia Intel. Primul μP au fost 4004 pe 4 biţi în 1971 şi 8008 pe 8 biţi în 1972. Acestea au fost practic primele **CPU** (**C**entral **P**rocessing **U**nit) realizate într-un singur circuit integrat. Datorită succesului înregistrat au fost realizate noi tipuri de μP, referite în *Tabelul 1.1*.

Tabelul 1.1

Cip	Data	Mhz	Tranz.	Mem.	Observaţii
4004	4/1971	0,108	2300	640	Primul μP pe 4 biţi
8008	4/1972	0,108	3500	16 KB	Primul μP pe 8 biţi
8080	4/1974	2	6000	64 KB	Prima UCP 8 biţi pe un cip
8086	6/1978	5-10	29000	1 MB	Prima UCP 16 biţi pe un cip
8088	6/1979	5-8	29000	1 MB	Folosit în IBM PC XT
80286	2/1982	8-12	134000	16 MB	Folosit în IBM PC AT
80386	10/1985	16-33	275000	4 GB	Prima UCP 32 biţi pe un cip
80486	4/1989	25-100	1200000	4 GB	Cache de 8KB încorporat
Pentium	3/1993	60-233	3100000	4 GB	Două benzi de asamblare
Pent. Pro	3/1995	150-200	5500000	4 GB	Două niveluri de cache
Pent. II	5/1997	233-400	7500000	4 GB	Pent. Pro + MMX

Firma IBM a utilizat μP 8088 în primul calculator personal **IBM-PC/XT (**e**X**tended **T**echnology) iar 80286 în calculatoarele **IBM-PC/AT (A**dvanced **T**echnology), care a devenit standard în industria calculatoarelor personale. Setul de instrucţiuni cuprindea instrucţiunile procesoarelor precedente la care au fost adăugate altele mai performante, asigurându-se astfel compatibilitatea în jos.

Procesorul **80386** are atât magistrala de date cât şi registrele pe 32 biţi şi un spaţiu de adresare mult mai mare, păstrându-se în continuare regula compatibilităţii în jos.

Procesorul **80486** este o îmbunătăţire a lui 80386 în ce priveşte viteza, datorită încorporării coprocesorului de virgulă mobilă, a controlorului de memorie şi a unei memorii cache de 8kB.

Pentium realizează trecerea la arhitectura *superpipeline* în condiţiile lungimii magistralei de date de 64 kB şi a posibilităţii de conectare a mai multor μP (maximum 4) într-un sistem.

Versiunea iniţială a IBM PC era echipată cu sistemul de operare MS-DOS produs de Microsoft Corporation, o companie mică pe vremea aceea.

Pentru a răspunde versiunilor din ce în ce mai puternice de microprocesoare realizate de Intel, Microsoft a lansat sistemul de operare OS/2 care avea o interfaţă grafică similară cu cea de la Apple Macintosh. Întrucât OS/2 nu s-a impus, Miocrosoft a dezvoltat Windows care, în versiuni succesive, a acaparat piaţa. Astfel două companii mici *Intel şi Microsoft* au detronat firma *IBM,* cu o îndelungată tradiţie şi mult mai multe resurse.

Şi în ceea ce priveşte calculatoarele de G4 sunt de notat realizări din România cum ar fi: **M18, M18B, M118 (**8080**), Felix PC (**8086**), PRAE, aMIC, HC85 (**Z80**).**

Urmărind evoluţia calculatoarelor realizate până în prezent se desprind două trăsături comune importante:

- toate evoluează în baza unui program memorat;
- limbajul de programare este artificial.

Spre sfârşitul anilor 80 s-a încercat o formulare a cerinţelor pentru generaţia a V-a. Potrivit acestor cerinţe arhitectura de bază a unei maşini de G5 urma să cuprindă următoarele elemente importante:

- o interfaţă inteligentă care să permită dialogul pe bază de limbaj natural (voce, sunete, imagini, informaţie grafică);
- maşina pentru rezolvarea de probleme (realizează raţionamente care să permită rezolvarea problemei fără cunoaşterea prealabilă a algoritmului;
- baza de cunoştinţe cu un volum imens, în care căutarea să se facă foarte rapid.

În *Tabelul 1.2* se prezintă sintetic un scurt istoric al dezvoltării din domeniul calculatoarelor, pânî în anul 2000.

Tabelul 1.2

AN	NUME	SUBIECT
0	**1**	**2**
1642	B. Pascal	Prima maşină de calculat mecanică
1834	C. Babbage	Maşina analitică de la Cambridge cu instrucţiuni pe cartelă perforată
1904	J. A. Flemming	Dioda
1906	Lee De Forest	Trioda
1936	K. Zuse	Primul calculator cu relee - Berlin

1943	Guv. Britanic cu concursul lui A. Turing	Primul calculator electronic cu tuburi cu vid – COLOSSUS
1944	H. Aiken	Primul calculator AMERICAN – MARK 1 (72 cuvinte a 23 cifre, ciclu maşină de 6 sec.) începutul erei electronicii
1946	J. Mauchly şi J.P. Eckert	Debutul generației I de calculatoare – ENIAC, Univ. Pennsylvenia (18.000 tuburi electronice, 1.500 relee., 600 comutatoare, 30 tone, 150 KW)
1948	J. Bardeen, W Shockley, W. Brittain	Tranzistorul
1948	N. Wiener	Cibernetica
1949	M. Wilkes, cu arhitectură propusă de J. Von Neumann	Primul calculator cu program înregistrat, EDSAC
1951	MIT	Primul calculator în timp real, WHIRLWIND
1951	J. Mauchly şi J.P. Eckert	Primul calculator comercializat – UNIVAC
0	1	2
1952	John von Neumann	Calculatorul IAS, Princeton, va răspândi arhitectura de bată von Neumann
1955	Laboratoarele Lincoln	Primul calculator echipat cu tranzistoare – TX - 0
1956-1960	Firma IBM	Calculatorul IBM 704 Circuite integrate
1960-1961	Firma DEC	Primul minicalculator. PDP-1, practic începutul generației a doua de calculatoare (tranzistorizate) şi a miniinfomaticii
1961	Firma IBM	Maşina de gestiune 140 I
1962	Firma IBM	Primul calculator ştiințific puternic (pe 36 de biți) – IBM 7094
1963	Firma BORROUGHS	Calculatorul B 5000 – primul calculator dedicat unui limbaj de programare ALGOL 60
1964	Firma IBM	Prima familie de calculatoare IBM 360;

		firma IBM prima supremaţie în informatică; multiprogramare; începutul generaţiei a treia (cu circuite integrate)
1964	Firma CDC	Primul calculator paralel - 6600
1965	Firma DEC	Primul calculator cu producţie de masă (pe 12 biţi). PDP -8
1970	Firma DEC	Minicalculatorul PDP-11, firma DEC preia supremaţia în miniinformatică. Circuite integrate pe scară largă
1970 - 1971	Firma INTEL	Primul microprocesor de 4 biţi - 4004
1972	Firma INTEL	Primul microprocesor de 8 biţi – 8008
1974	Firma CRAY	Primul supercalculator CRAY-1
1978	Firma INTEL	Microprocesorul 8080 prima unitate centrală pe un cip
1978	Firma DEC	Primul superminicalculator de 32 biţi-VACS
1979	Firma MOTOROLA	Primul procesor al familiei 680x0 – 68000
0	1	2
1980	Firma IBM	Începutul generaţiei a patra de calculatoare (cu circuite VLS1); era calculatoarelor personale, PC
1982	Firma INTEL	Microprocesorul de 16 biţi – 80286
1983	SUA	Tehnologia LAN)Local Area Network) este folosită pe scară largă
1984	Firma MOTOROLA	Primul microprocesor de 32 biţi – 68020
1985	Firma INTEL	Microprocesorul de 32 biţi – 80386
1987	Firma MOTOROLA	Microprocesorul de 32 biţi cu unitate de gestiune a memoriei – 68030
1989	Firma INTEL	Microprocesorul de l32 biţi cu coprocesor şi memorie cache - 80486
1990 - 1991	Thinking Machines Co. CM2 şi CM5	Începutul generaţiei a cincea de calculatoare (preluare masiv paralelă)
1992	Laboratorul CERN Geneva	Serviciul de informaţii www (World Wide Web) oferit pe Internet

1993	Firma INTEL	Microprocesorul cu structură superscalară – Pentium
1995	Firma Sun Microsystems	Tehnologia JAVA asigură interactivitatea serviciului www
1996	Firma CYRIX	Microprocesorul în arhitectură superpipe /line/ 6x86 P200+
1997	Firma AMD	Microprocesoarele RISC – K5, K6
1998	Firma AMD	Microprocesorul K6-2 3D
1998	Firma INTEL	Microprocesorul Mendocino 333/MHz
1999	Firma INTEL	Microprocesorul Pentium III/600MHz
0	1	2
1999	Firma AMD	Microprocesorul Athlon/K7
2000	Firma AMD	Microprocesorul Sharptooth/K6-3

1.1.3. Forţe tehnologice şi economice

După cum s-a mai arătat evoluţia nici unei alte ramuri industriale nu este atât de rapidă ca cea din industria calculatoarelor. Rata progresului tehnologic în domeniu poate fi modelată cu ajutorul unei observaţii numită **legea lui Moore (Moore's Law)** descoperită în 1965 de către Gordon Moore co-fondator şi preşedinte al firmei Intel. Referindu-se la memorii Moore a observat că fiecare nouă generaţie apărea la 3 ani distanţă de precedenta. Cum fiecare nouă generaţie avea o capacitate de patru ori mai mare decât predecesoarea, el a realizat că numărul ce tranzistoare pe cip creştea în ritm constant şi a prezis că această creştere va continua şi în deceniile următoare[4]. Uzual legea lui Moore este cunoscută *ca dublarea numărului de tranzistoare pe chip la fiecare 18 luni.* Pentru exemplificare în figura 1.4 se prezintă un grafic (scară semilogaritmică) asociat legii lui Moore pentru perioada 1965-1995.

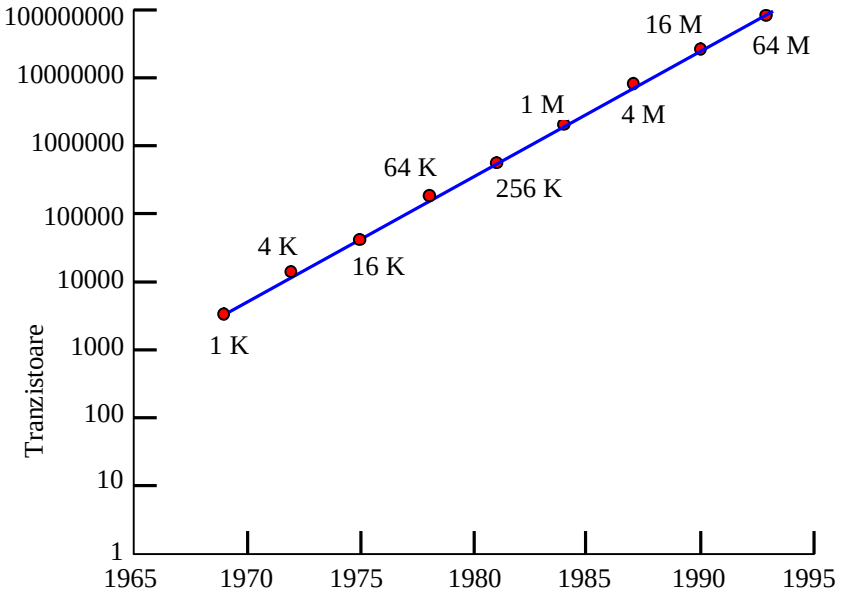


Fig. 1.4. Legea lui Moore pentru memorii (creştere anuală cu 60 % a numărului de tranzistoare pe cip.

[4] Se apreciază că legea va fi valabilă până către anul 2020 când tranzistoarele în accepţiunea curentă vor fi alcătuite din prea puţini atomi pentru ca să prezinte siguranţă în funcţionare.

Chiar dacă legea lui Moore a fost asociată mult timp cu numărul de biţi dintr-un cip de memorie, aceasta poate fi aplicată la fel de bine şi pentru numărul de tranzistoare integrat pe un microprocesor. Pentru edificare în figura 1.5 se prezintă evoluţia acestui număr pentru o parte din microprocesoarele familiei Intel.

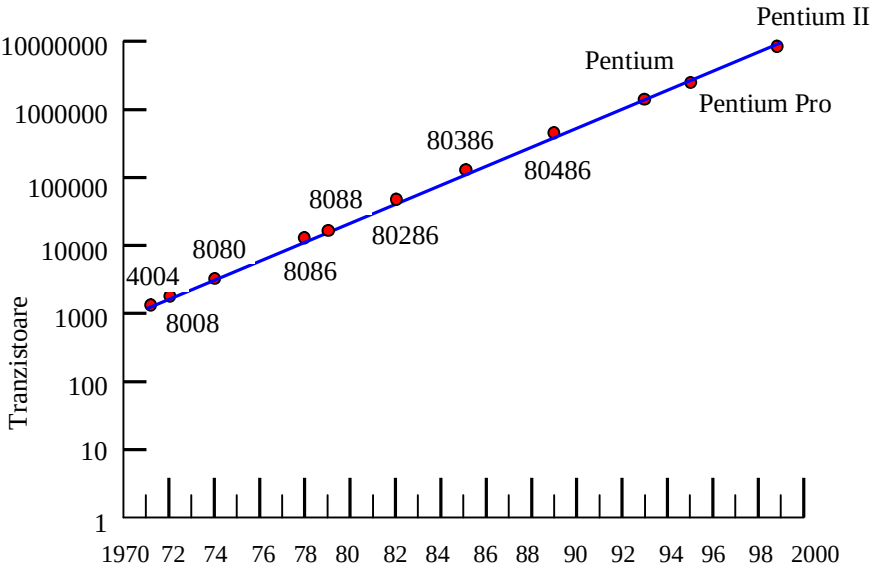


Fig. 1.5. Legea lui Moore pentru cipurile *UCP* (microprocesoare).

Legea lui Moore a creat ceea ce unii economişti numesc **cerc virtuos (virtous circle),** care succint se prezintă astfel:

- progresele tehnologice (nr. tr. / cip) conduc la produse mai bune şi mai ieftine;
- preţurile mici determină noi aplicaţii (nimeni nu şi-a pus problema să dezvolte jocuri pentru calculatoare care costau milioane de USD);
- noile aplicaţii duc la noi pieţe şi apar noi companii care să profite de ele;
- aceste companii se concurează (determină o competiţie);
- această competiţie determină o cerere de tehnologii performante astfel încât să-i elimine pe ceilalţi (cerc complet)

Tot în sfera acestor consideraţii de natură tehnologică şi economică este cazul să amintim prima lege a *software-lui a lui Nathan*[5]. Conform acestei legi *software-ul* se comportă ca un gaz, în sensul că îşi creşte volumul pentru a ocupa tot spaţiul care i se pune la dispoziţie. Software-ul continuând în permanenţă să acumuleze opţiuni, creează o nevoie constantă de procesoare mai rapide, memorii mai mari şi capacităţi de I/E sporite.

Legat de evoluţia calculatoarelor o menţiune specială trebuie făcută în ceea ce priveşte telecomunicaţiile, creşterea exponenţială a Internet-ului fiind o consecinţă imediată.

1.2. Modalităţi de reprezentare a calculatoarelor

1.2.1. Maşina von Neumann

Un important volum al calculatoarelor calculatoarelor realizate până în prezent se bazează pe arhitectura propusă de ***von Neumann***, care după cum s-a menţionat a făcut parte din echipele de realizare a primelor calculatoare.

Într-un articol publicat în 1947 von Neumann, e expus principiile de realizare a unui calculator numeric. Conform acestor principii un *CN* trebuie să posede următoarele elemente:

- un *mediu de intrare* prin intermediul căruia să poată fi introdus un număr practic nelimitat de date şi de instrucţiuni;
- o *memorie* unde să fie depuşi operanzii şi instrucţiunile (în aceiaşi formă) şi de unde rezultatele să poată fi preluate în ordinea dorită;
- o *secţiune de calcul* care să fie capabilă să execute operaţii aritmetice şi logice asupra datelor din memorie;
- un *mediu de ieşire* prin intermediul căruia să poată fi comunicat utilizatorului un număr practic nelimitat de rezultate;
- *unitate de comandă* capabilă să interpreteze instrucţiunile citite din memorie şi pe baza informaţiilor furnizate de către secţiunea de calcul să fie capabilă să decidă între mai multe variante de desfăşurare a operaţiilor.

[5] Nathan Myhrvold – director la Microsoft.

Pe baza acestor cerinţe poate fi realizat modelul funcţional al unui *CN* , model ilustrat în figura 1.6.

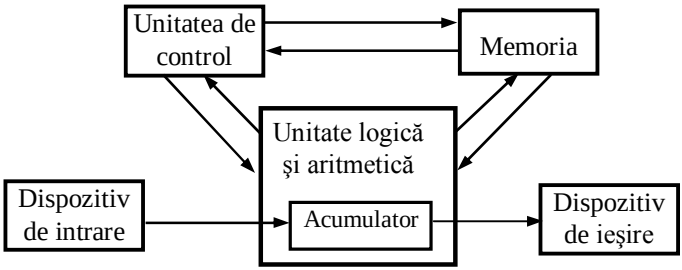


Fig. 1.6. Modelul funcţional al maşinii lui John von Neumann.

La nivelul unui CN sunt evidenţiate două fluxuri care interacţionează şi anume: *fluxul de instrucţiuni şi fluxul de date.* Calculatoarele realizate conform principiilor *von Neumann* se numesc convenţionale şi sunt controlate de fluxul de instrucţiuni[6].

În general un calculator poate fi investigat din punct de vedere *funcţional sau structural.* În cele ce urmează se vor face referiri la cele două modalităţi de reprezentare.

1.2.2. Reprezentarea funcţională

Din punct de vedere funcţional un calculator se poate prezenta în forma generală prin tripletul

$$\langle I, E, C \rangle \qquad (1.1)$$

în care I este mulţimea intrărilor;

E – mulţimea ieşirilor;

C - o submulţime a produselor carteziene între elementele mulţimii I şi cele ale mulţimii E respectiv $C \subset I \times E$ [7].

[6] În prezent se studiază realizarea a noi tipuri de calculatoare neconvenţionale controlate *fluxul de date sau fluxul de cereri.* În primul caz sunt amorsate, la un moment dat toate operaţiile din program pentru care sunt disponibile datele implicate. În al doilea caz necesitatea unui rezultat activează toate operaţiile de calcul asociate.

[7] Practic C realizează aplicaţii din mulţimea intrărilor I în mulţimea ieşirilor E.

În contextul, reprezentării funcţionale se defineşte *arhitectura unui calculator numeric* prin cuadruplul

$$A = \langle PI, PE, RG, I \rangle \qquad (1.2)$$

unde $PI = \{PI0,...,PIi\}$ este mulţimea porturilor de intrare;

$PE = \{PE0,...,PEj\}$ - mulţimea porturilor de ieşire;

$RG = \{RG0,...,RGk\}$ - mulţimea registrelor generale din unitatea de execuţie;

$I = \{I0,...,Il\}$ - setul instrucţiunilorcalculatorului.

Porturile de intrare şi de ieşire sunt utilizate pentru schimbul de date cu lumea înconjurătoare, prin inermediul echipamentelor periferice. În ceea ce priveşte *registrele* acestea sunt utilizate pentru stocarea diferitelor variabile de stare.

Un prim exemplu de arhitectură ilustrată în figura 1.7, se referă la microprocesorul pe 8 biţi 8080,

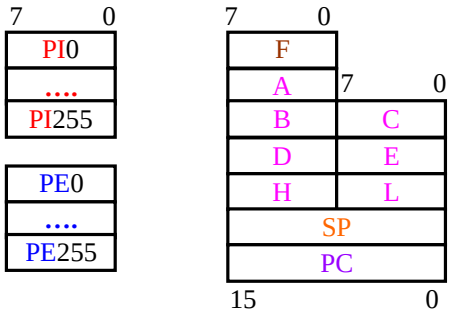


Fig. 1.7. Registrele microprocesorului Intel 8080.

- În ceea ce priveşte porturile, microprocesorul are 256 porturi de intrare pe 8 biţi (PI0…PI255) şi tot atâtea de ieşire (PE0…PE255).

- Registrele sunt reprezentate de registre generale şi specializate după cum urmează:

  - ***A*** este registrul *acumulator* principal utilizat de majoritatea instrucţiunilor aritmetice şi logice;

- ***B, C, D, E, H, L*** sunt registre de uz general, care pot fi utilizate ca registre de 8 biţi sau alipite (concaternate) două câte două ca registre de 16 biţi;

- ***F*** este registrul indicatorilor de condiţii furnizaţi de unitatea de execuţie după fiecare operaţie;

- ***SP*** este indicatorul de stivă, care permite accesul la structura LIFO de tip *stivă* organizată în memoria calculatorului;

- ***PC*** este contorul de program care conţine adresa instrucţiunii care urmează să se execute, permiţând astfel adresarea instrucţiunilor în memorie.

- Repertoriul de instrucţiuni al microprocesorului 8080 cuprinde 78 instrucţiuni de bază pe 1, 2 sau 3 octeţi.

În încheierea acestei scurte prezenări arhitecturale câteva precizări care privesc structura registrului indicatorilor de copndiţii *F*. Aceste are dimensiunea de 7 biţi şi conţine următorii indicatori a căror dipunere este ilustrată în figura 1.8:

- *S* – indicator pentru semnul rezultatului;
- *Z* – indicator pentru rezultat zero;
- *AC* – indicator pentru transportul între tetradele octetului rezultat;
- *P* – indicator privind paritatea numărului der biţi *1* din rezultat;
- *CY* – indicator de transport în afara bitului de semn al rezultatului.

7	***6***	***5***	***4***	***3***	***2***	***1***	***0***
S	Z	-	AC	-	P	-	CY

Fig. 1.8. Structura registrului *indicatorilor de* condiţii la microprocesorul 8080.

Al doilea exemplu are în vedere arhitectura microprocesorului 8086, care a fost primul microprocesor cu lungimea cuvântului de date de 16 biţi şi care putea adresa 1 MB de memorie (cuvânt de adresă de 20 de biţi).

- Microprocesorul 8086 poate adresa 2x65536 porturi pe un octet sau 2x32768 porturi pe câte 16 biţi potrivit reprezentării din figura 1.9.

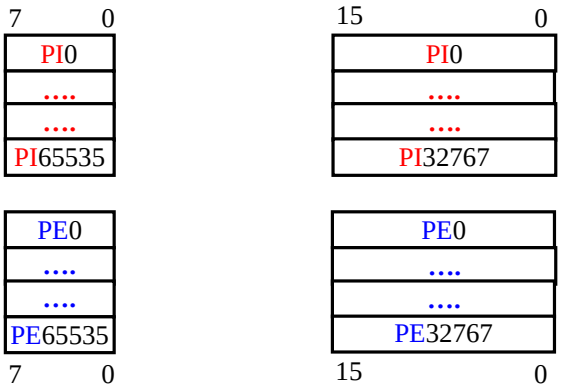


Fig. 1.9. Spaţiul porturilor microprocesorului 8086: PI – porturi de intrare; PE – porturi de ieşire.

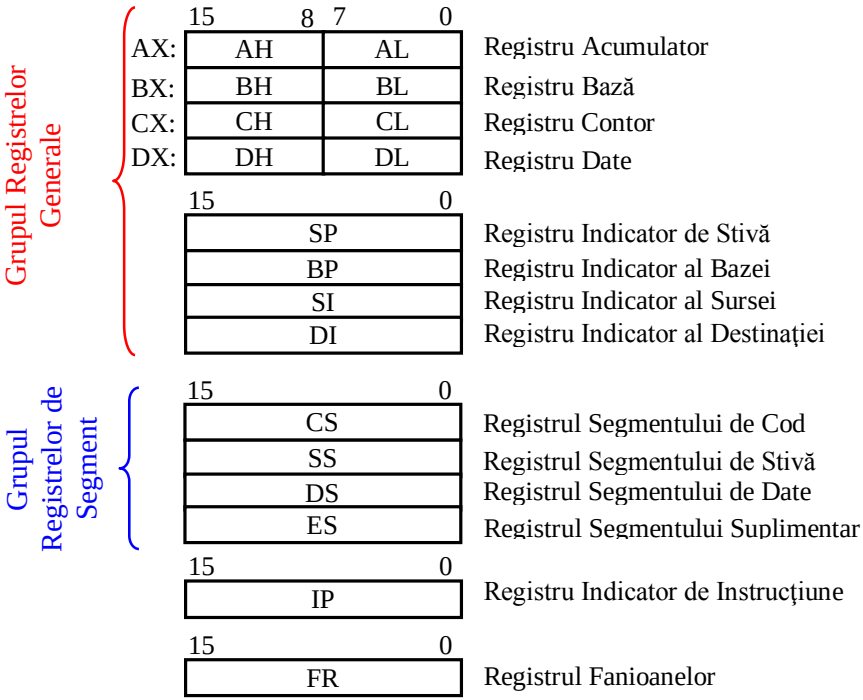


Fig. 1.10. Registrele microprocesorului 8086.

- Mulţimea registrelor conţine patru grupuri şi anume:

- grupul registrelor generale;
- grupul registrelor de segment;
- registrul indicator de instrucţiune;
- registrul fanioanelor,

a căror componenţă este ilustrată în figura 1.10.

În ceea ce priveşte registrul fanioanelor acesta are structura prezentată în figura 1.11.

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
-	-	-	-	**O**	**D**	**I**	**T**	**S**	**Z**	-	**A**	-	**P**	-	**C**

Fig. 1.11. Structura registrului fanioanelor microprocesorului 8086: *O* – depăşire aritmetică; *D* – direcţia de explorare a şirurilor; *I* – activare/dezactivare întreruperi; *T* – capcana pentru lucrul pas cu pas; *S* – semn; *Z* – zero; *A* – transport auxiliar; *P* – paritate; *C* – transport.

1.2.3. Reprezentarea structurală

Ideea de bază a *reprezentării structurate* are în vedere faptul că un calculator reprezintă un sistem realizat din componente (*primitive)* funcţionale. Interconectarea componentelor este astfel realizată încât permite calculatorului executarea operaţiilor specifice de prelucrare a informaţiei.

Din punct de vedere structural sistemul de calcul poate fi divizat în:

- unitatea de intrare (UI);
- unitatea centrală (UCe);
- unitatea de ieşire (UE),

interconectate ca în figura 1.12.

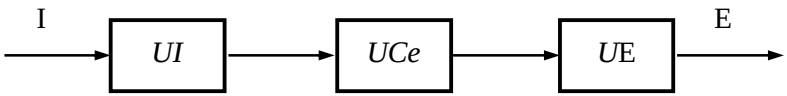


Fig. 1.12. Unităţile structurale principale ale unui calculator numeric.

- *UI* şi *UE* permit conectarea sistemului la echipamentele periferice primare, care îndeplinesc roluri de *traductoare şi elemente de execuţie.* Această asociere este dictată de faptul că pe intrări se *preia informaţie din mediul extern*, iar pe ieşiri *se intervine asupra mediului extern.*

- *UCe* asigură execuţia programului şi stocarea codului (programului) .

Înţelegerea funcţionării unui sistem complex cum este calculatorul implică o creştere a nivelului de detaliere ilustrată în figura 1.12. Se observă că pe lângă evidenţiera părţilor componente ale celor trei unităţi structurale de bază sunt prezente şi liniile de semnal între module.

Astfel la nivelul unităţilor *UI şi UE* pot fi întâlnite următoarele entităţi:

- subsistemul de intrare *SI*;
- subsistemul de ieşire *SE;*
- echipamentele periferce de intrare *EPI;*
- echipamentele periferce de ieşire *EPE.*

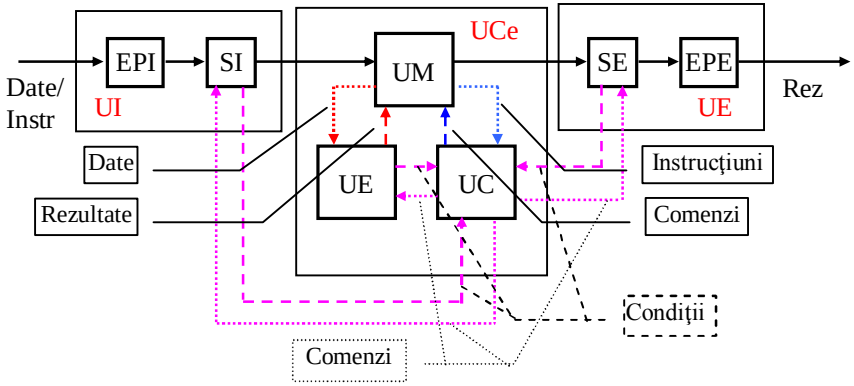


Fig. 1.12. Detalierea unităţilor structurale principale ale unui calculator numeric.

EPI au rolul de a prelua datele de intrare (eventual stocate pe anumite suporturi) şi de a le aduce la forma acceptată de *SI* (ca natură fizică, format de reprezentare etc.). *EPE* preiau datele prelucrate (sub forma rezultatelor) şi le aduc la o formă accesibilă utilizatorilor sau le transpun pe un anumit suport fizic.

La nivelul *UCe* se disting următoarele elemente funcţionale:

- unitatea de memorie *UM;*
- unitatea de execuţie *UE;*
- unitatea de comandă *UC.*

- *Unitatea de memorie* stochează datele iniţiale, codul (programul), rezultatele iniţiale şi finale. *UM* are o organizare liniară, care constă constă în locaţii (celule) al căror conţinut poate fi referit prin specificarea unei adrese, potrivit reprezentării din figura 1.13.

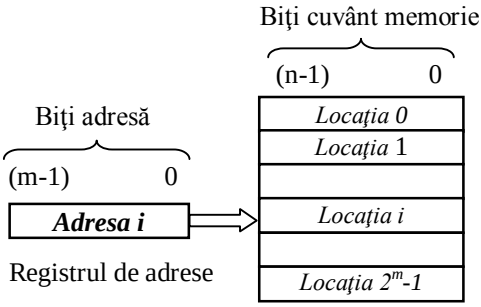


Fig. 1.13. Organizarea liniară a memoriei unui *C*N.

Din figura 1.13 rezultă că prin intermediul unui cuvânt de adresă de *m* biţi pot fi accesate 2^m locaţii de memorie, numerotate între *0* şi 2^m*-1*.

- *Unitatea de execuţie* realizează sub controlul *UC* o succesiune de operaţii aritmetice şi logice asupra datelor preluate din *UM* sau memoria locală proprie. *UE* este implementată sub forma unor registre generale *RG*.. După fiecare operaţie *UE* actualizează valorile unor indicatori de condiţie, care reflectă caracteristicile rezultatului curent (<0, >0, =0, paritate, transport, depăşirea, etc.).

- *Unitatea de comandă* procesează fluxul de instrucţiuni care constituie programul. Aceasta furnizează semnale de coordonare pentru celelalte unităţi în conformitate cu cerinţele programului. *UC* se poate implementa sub formă convenţională (ca automat secvenţial) sau în formă microprogramată (cu stocarea semnalelor de comandă în manieră statică într-o memorie rapidă).

Sintetizând se poate observa că:

- ansamblul *UE+UC=P* (Procesor sau Unitate Centrală de Prelucrare – *UCP sau CPU);*
- ansamblul *P+M=UCe* (Unitate Centrală);

- ansamblul *UCe+SI+SE+Software=Sistem de calcul.*

În ceea ce priveşte instrucţiunile care operează asupra datelor, acestea se împart în două categorii:

- instrucţiuni operaţionale de transfer, inclusiv instrucţiunile de I/E;
- instrucţiuni cu caracter de decizie, care modifică secvenţa de execuţie a programului condiţionat sau nu.

Instrucţiunile operaţionale conţin mai multe câmpuri, dintre care primul este codul operaţiei *COP* aşa cum rezultă şi din figura 1.14.

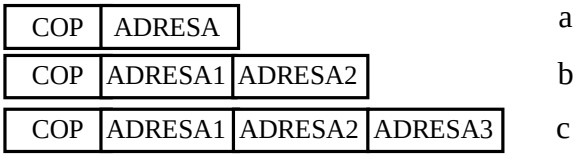


Fig. 1.14. Structuri de instrucţiuni: a- cu o adrese; b-cu două adrese; c-cu trei adrese.

În primul caz instrucţiunea conţine adresa unui singur operand, cel de-al doilea (în cazul operaţiilor care implică doi operanzi) fiind deja adus în *UE,* unde rămâne în mod obişnuit şi rezultatul.

În cazul al doilea sunt prezente adresele celor doi operanzi (sursă şi destinaţie) rezultatul fiind de regulă transmis la adresa operandului destinaţie.

Câmpul *Adresa 3* în al treilea caz poate conţine adresa la care se depune rezultatul.

1.3. Clasificări arhitecturale

În cele ce urmează vor fi abordate două criterii de clasificare şi anume:

- după mecanismul de control al execuţiei;
- după organizarea spaţiului de adresare a memoriei.

1.3.1. Clasificarea după controlul execuţiei

Funcţionarea oricărui calculator presupune existenţa a două fluxuri care interacţionează şi anume : *fluxul de instrucţiuni şi fluxul de date.* Fluxul de instrucţiuni – expresia programată a unui algoritm indică ce anume trebuie să execute calculatorul la fiecare pas. Fluxul datelor (reprezentând datele de intrare în algoritm) este procesat de instrucţiuni în vederea obţinerii rezultatelor.

După natura acestor fluxuri (*unice sau multiple)* pot fi identificate patru clase de calculatoare şi anume:

- Calculatoare cu fluxuri unice de date şi de instrucţiuni (*Single Instruction Stream, Single Data Stream – **SISD);***
- Calculatoare cu fluxuri multiple de instrucţiuni şi flux unic de date *(Multiple Instruction Stream, Single Data Stream – **MISD);***
- Calculatoare cu flux unic instrucţiuni şi fluxuri multiple de date *(Single Instruction Stream, Multiple Data Stream – **SIMD);***
- Calculatoare cu fluxuri multiple de instrucţiuni şi fluxuri multiple de date *(Multiple Instruction Stream, Multiple Data Stream – **MIMD);***

Clasificarea de mai sus, introdusă de Michael Flinn în anul 1972 şi are în vedere mecanismul de control al execuţiei. În arhitecturile cu un unic flux de instrucţiuni, o singură unitate de control lansează instrucţiuni către toate unităţile procesare (*execuţie).* În cazul arhitecturilor cu mai multe fluxuri de instrucţiuni, unităţile de procesare lucrează independent sub controlul unui flux de instrucţiuni propriu.

1.3.1.1. Calculatoare SISD

Un calculator din această clasă este caracterizat de un singur flux de instrucţiuni *FI* pe care unitatea de procesare (execuţie) îl primeşte de la unitatea de control (comandă), după cum se poate vedea în figura 1.15.

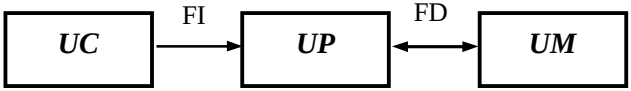


Fig. 1.15. Calculator *SISD*: *UC* – Unitate de Comandă (Control); *UP* – Unitate de Procesare (Execuţie); *UM* – Unitate de Memorie; *FI* – Flux de Instrucţiuni; *FD* – Flux de Date.

La fiecare pas *UC* lansează o instrucţiune care operează asupra unei date din fluxul de date *FD* obţinut din unitatea de memorie *UM*. Un astfel de calculator este un calculator pur secvenţial care respectă modelul *von Neumann.* De exemplu pentru calculul sumei a *n* numere *UC* accesează *UM* de *n* ori consecutiv şi de fiecare dată primeşte un număr, efectuându-se *n-1* adunări

Principalul neajuns al acestei arhitecturi constă în viteza de procesare care la rândul ei este determinată de frecvenţa ceasului. Este evident că nici o realizare tehnologică nu va putea face perioada ceasului nulă. În consecinţă modul strict secvenţial de tratare a operaţiilor impus de arhitectura von Neumann plafonează la un moment dat viteza de procesare. Această situaţie este cunoscută sub numele *gâtul sticlei lui Neumann (Neumann Bottleneck).* Spargerea acestei limitări se realizează prin introducerea arhitecturilor de tip neserial (*respectiv arhitecturile paralele*).

1.3.1.2. *Calculatoare MISD*

La aceste tipuri de calculatoare, pentru care în figura 1.16 se prezintă o structură de principiu, mai multe unităţi de procesare , fiecare cu propria unitate de comandă *UC,* primesc un unic flux de date de la unitatea de memorie *UM*. La fiecare pas, asupra unei date primite din memorie sunt executate operaţii diferite de către toate procesoarele simultan, în funcţie de instrucţiunea primită de la propria unitate de control.

Paralelismul structurii *MISD* este dictat de faptul că fiecare *UP* execută operaţii diferite la acelaşi moment de timp, aspect evidenţiat şi de exemplele de mai jos.

Exemplul 1.1.

Se cere să se determine dacă un număr natural *z* are şi alţi divizori în afara celor improprii, respectiv *unu şi numărul însuşi.*

Soluţia secvenţială constă în testarea repetată a divizibilităţii cu toţi divizorii posibili. Problema poate fi rezolvată într-un singur pas cu o arhitectură *MISD* care conţine un număr de procesoare egal cu numărul de divizori potenţiali ai numărului *z*. Toate *UP* primesc *z* ca dată de intrare, încearcă divizibilitatea cu divizorul asignat şi memorează rezultatul obţinut. Se poate determina astfel într-un singur pas dacă *z* este număr prim. Având în vedere este greu de realizat câte un calculator pentru fiecare număr, o

abordare mai realistă presupune un număr fix de procesoare, care poate fi mai mic decât numărul de divizori potenţiali. În aceste condiţii unei *UP* îi revine misiunea de a trata (în manieră secvenţială) o submulţime de divizori. Cu toate că nu este o procesare pur paralelă, timpul de obţinere a rezultatului este inferior timpului în care problema s-ar rezolva pe o arhitectură *SISD*.

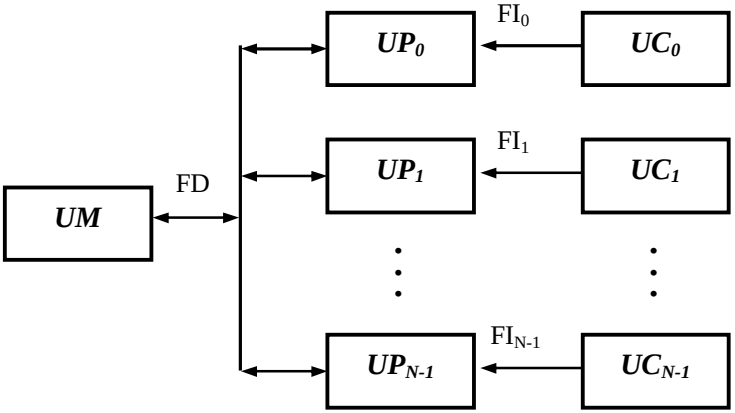


Fig. 1.16. Calculator *MISD*: $UC_0 \ldots UC_{N-1}$– Unităţi de Comandă (Control); $UP_0 \ldots UP_{N-1}$ – Unităţi de Procesare (Execuţie); *UM* – Unitate de Memorie; $FI_0 \ldots FI_{N-1}$ – Fluxuri de Instrucţiuni; *FD* – Flux de Date.

Exemplul 1.2.

Se presupune existenţa mai multor clase de obiecte *Ci* cu proprietăţi comune. Să se clasifice un obiect *z*, respectiv să se determine clasa de apartenenţă a acestuia.

Ca şi în exemplul precedent, soluţia secvenţială constă în testarea repetată a încadrării obiectului în toate clasele posibile. Rezolvarea eficientă a problemei se poate pe o arhitectură *MISD* care conţine un număr de procesoare egal cu numărul de clase posibile. Toate *UP* primesc obiectul *z* ca intrare, şi testează simultan apartenenţa acesteia la clasa specifică fiecărei *UP*. Se poate determina astfel într-un singur pas clasa *Ci* în care poate fi încadrat obiectul *z*. Dacă numărul de procesoare este inferior celui al claselor *Ci* atunci se asociază fiecăruia o submulţime de clase pentru care se efectuează secvenţial testule de apartenenţă. Cu toate că nu este o procesare pur paralelă, timpul de obţinere a rezultatului este inferior timpului în care problema s-ar rezolva pe o arhitectură *SISD*.

1.3.1.3. *Calculatoare SIMD*

După cum se observă din figura 1.17 un calculator din această clasă constă dintr-un număr *N* de *UP* care procesează un flux unic de instrucţiuni *FI* lansat de o unică *UC*.

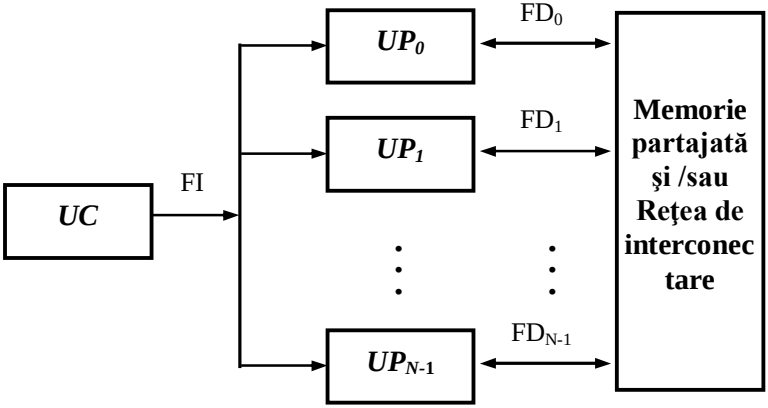


Fig. 1.17. Calculator *SIMD: UC*– Unitate de Comandă (Control);
UP_0… UP_{N-1} – Unităţi de Procesare (Execuţie);
FD_0… FD_{N-1} – Fluxuri de Date; *FI* – Flux de Instrucţiuni.

Fiecare *UP* care posedă câte o *UAL, UC şi UM* locale lucrează sincron în sensul că la fiecare pas execută aceiaşi instrucţiune asupra unei date diferite din fluxuri de date multiple[8]. Rezolvarea problemelor pe un astfel de calculator presupune transferul de date şi rezultate intermediare între procesoarele componente. Aceste comunicaţii se realizează prin intermediul unei reţele de interconectare sau printr-o *memorie partajată şi reţea de interconectare.*

O caracteristică importantă a calculatoarelor *SIMD* este aceea că presupun o sincronizare automată între procesoare la fiecare ciclu instrucţiune. Din acest motiv aceste calculatoare sunt utile pentru execuţia programelor paralele care necesită *sincronizări frecvente* şi mai puţin recomandate pentru programele care necesită *decizii frecvente*, după cum rezultă din exemplul 1.3.

[8] Pe baza acestor considerente calculatoarele *SIMD* mai sunt cunoscute şi sub denumirea de tablouri de procesoare *(processor arrays).*

Exemplul 1.3.

Instrucţiunea de decizie

```
if (b==0)
        c=a;
else
        c=a/b;
```

este executată în doi paşi. În primul pas toate procesoarele care au *b==0* execută instrucţiunea *c=a* iar toate celelalte sunt inactive. În al doilea pas, partea *else* a instrucţiunii este executată de toate procesoarele care au *b≠0,* iar toate procesoarele care au fost inactive la primul pas devin acum inactive.

Având în vedere această observaţie, inclusiv exemplul de mai sus, se poate concluziona că sistemele *SIMD* sunt eficiente pentru execuţia programelor care implică operaţii asupra unor masive de date, respectiv a *programelor data-paralele.*

1.3.1.4. Calculatoare MIMD

Calculatoarele *MIMD* sunt calculatoare paralele la care un procesor poate executa un program diferit de programele executate de celelalte procesoare. După cum reiese din figura 1.18, un asemenea calculator constă din mai multe procesoare *(UP)* fiecare cu propria unitate de control *(UC)* şi partiţie de memorie.

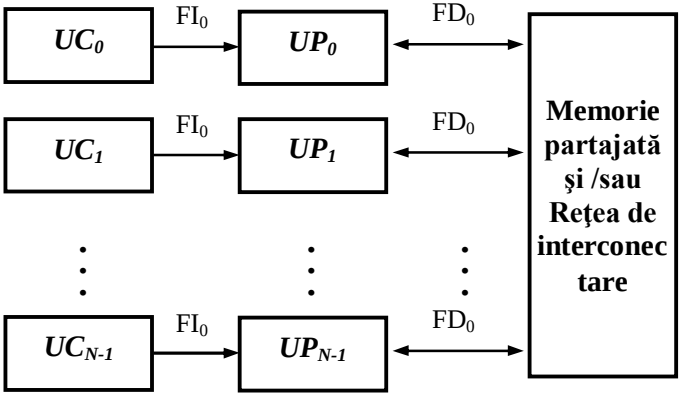


Fig. 1.18. Calculator *MIMD*: $UC_0 \ldots UC_{N-1}$– Unităţi de Comandă (Control); $UP_0 \ldots UP_{N-1}$ – Unităţi de Procesare (Execuţie); $FD_0 \ldots FD_{N-1}$ – Fluxuri de Date; $FI_0 \ldots FI_{N-1}$ – Fluxuri de Instrucţiuni.

După cum se observă din figura 1.18 fiecare procesor operează sub controlul unui flux de instrucţiuni lansat de propria unitate de control asupra unui propriu flux de date. Ca şi în cazul unui calculator *SIMD*, comunicaţia se poate realiza *printr-o reţea de interconectare* sau *printr-o memorie partajată şi reţea de interconectare.*

UP individuale aferente unui calculator *MIMD* sunt de o complexitate superioară celor aferente unui calculator *SIMD* deoarece posedă câte o unitate de control proprie. În condiţiile acestei complexităţi costul lor este mai scăzut deoarece sunt procesoare de uz general spre deosebire de cele aferente calculatoarelor *SIMD* care sunt proiectate special.

În continuare se prezintă un exemplu edificator pentru marea flexibilitate în execuţie a calculatoarelor *MIMD*.

Exemplul 1.4.

Fie programul unui joc de strategie *(exemplu şahul)* care generează un arbore de decizie *(game tree)*. Rădăcina arborelui este configuraţia curentă a jocului *(respectiv poziţia pe tablă),* de la care programul trebuie să execute o mutare. Poziţiile obţinute prin fiecare mutare posibilă constituie fiii rădăcinii şi în acelaşi timp nodurile următorului nivel. Această ierarhie continuă până la un număr predefinit de nivele.

Fiecărui nod frunză al acestui arbore i se asociază o valoare de optimalitate, programul trebuind să determine calea care conduce către cea mai bună poziţie, considerând că partenerul joacă perfect.

Având în vedere că din fiecare poziţie sunt posibile mai multe mişcări, arborii de decizie tind să devină foarte mari şi de aceea sunt generaţi pe măsură ce se caută soluţia pentru o poziţie dată. Explorarea acestor arbori se face printr-un algoritm de căutare în adâncime *(depth-first search),* cu excluderea *(cut-off)* acelor căi despre care se cunoaşte că nu pot conduce către o soluţie mai bună decât cele deja explorate.

Modalitatea de implementare a unui astfel de program pe un calculator *MIMD,* constă în distribuirea subarborilor rădăcinii procesoarelor componente, care le explorează în paralel. În cursul căutării procesoarele pot interschimba între ele subarbori de căutare , astfel încât un procesor care a terminat propriul sbarbore să nu rămână inactiv, ci să primească de la

un altul o parte a subarborelui acestuia. Problema nu poate fi rezolvată pe un calculator *SIMD* deoarece la fiecare pas de execuţie instrucţiunile pot diferi de la un procesor la altul [9].

1.3.2. Clasificarea după organizarea spaţiului de adresă

Rezolvarea unei probleme pe un sistem cu mai multe procesoare presupune comunicaţia între acestea. După modul în care se realizează comunicarea există următoarele categorii de arhitecturi paralele:

- arhitecturi cu spaţiul de adresă al memoriei partajat;
- arhitecturi cu transfer de mesaje.

1.3.2.1. Arhitecturi cu spaţiul de adresă al memoriei partajat

Aceste arhitecturi cunoscute ca *Shared Address Space Architectures - SASA* asigură suport hardware de citire şi scriere al tuturor procesoarelor la o singură memorie partajată. Calculatoarele *MIMD* care se încadrează în această categorie ca *multiprocesoare (multiprocessors).* Având în vedere gradul avansat de partajare a resurselor acestea sunt considerate *sisteme puternic cuplate (tightly coupled systems).* Există mai multe modele de *SASA* şi anume:

- *UMA – Uniform Memory Access;*
- *NUMA – Nonuniform Memory Access;*
- *COMA – Cache Only Memory Access,*

asupra cărora se vor face în continuare referiri succinte.

- ***Modelul cu acces uniform la memorie (UMA)***

În cadrul modelului *UMA* memoria fizică este împărţită uniform de toate procesoarele, care au timp de acces egal la toate cuvintele din memoria partajată – figura 1.19.

[9] De exemplu în timp ce un procesor evaluează un nod frunză, un al doilea poate să execute excludere a unei căi, al treilea poate efectua revenirea la un arbore pentru căutarea unei noi căi, etc.

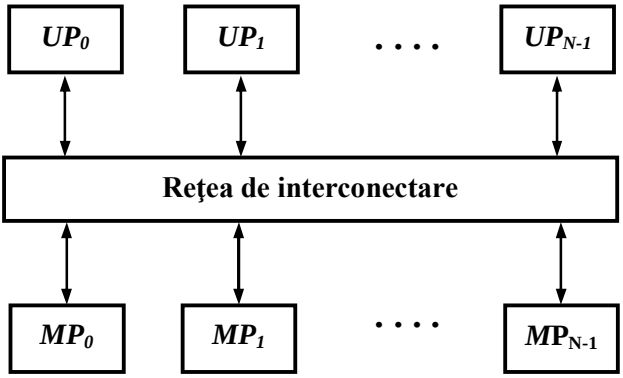


Fig. 1.19. Modelul *UMA* de *SASA* : UP_0… UP_{N-1}– Procesoare; MP_0… MP_{N-1} – *Memorie partajată.*

Un neajuns important al modelului *UMA* este acela că banda de comunicaţie a reţelei de interconectare trebuie să fie foarte mare deoarece în fiecare ciclu instrucţiune, fiecare procesor poate solicita accesul la o locaţie din memoria partajată prin intermediul acestei reţele.

Această complexitate poate determina o creştere a timpului de acces l memorie, întrucât se pot ivi situaţii în care datele să traverseze mai multe nivele ale reţelei de interconectare.

- ***Modelul cu acces neuniform la memorie (NUMA)***

La un asemenea model timpul de acces variază în funcţie de localizarea cuvântului în memorie.

O primă variantă a modelului *NUMA*, reprezentată în figura 1.20, este cea în care fiecare procesor UP_i are o memorie locală ML_i în care sunt memorate programele şi datele locale ale fiecărui procesor. În afara nivelului memoriilor locale mai există cel memoriei globale format din modulele MGP_i. În aceste memorii sunt memorate datele partajate accesibile tuturor procesoarelor prin intermediul unei reţele de interconectare.

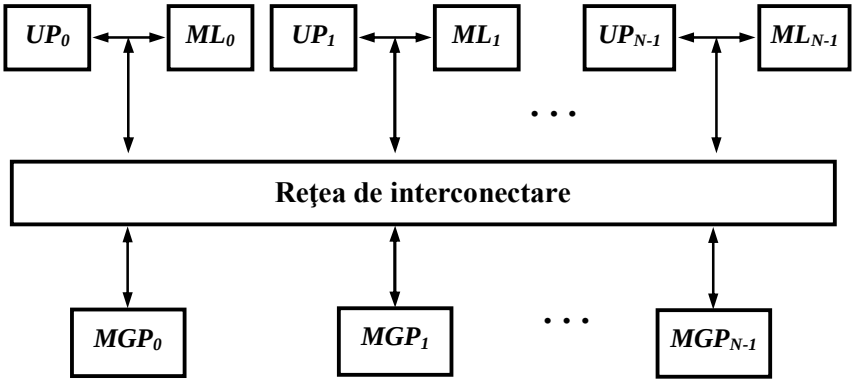


Fig. 1.20. Modelul *NUMA* de *SASA* : UP_0… UP_{N-1}– Procesoare; MGP_0… MGP_{N-1} – Memorie globală partajată; ML_0… ML_{N-1} – Memorie locală nepartajată;

O a doua variantă de model *NUMA* , ilustrată în figura 1.21, este cea în care memoria este distribuită fizic sub forma modulelor de memorie partajată *(MPD_i)*. Din punct de vedere logic însă mulţimea tuturor acestor procesoare formează un spaţiu de adrese unic accesat de toate procesoarele. Timpul de acces la memorie este neuniform deoarece timpul de acces la propria memorie este inferior timpului de acces la memoriile aferente altor noduri datorită timpilor suplimentari necesari traversării reţelei de interconectare.

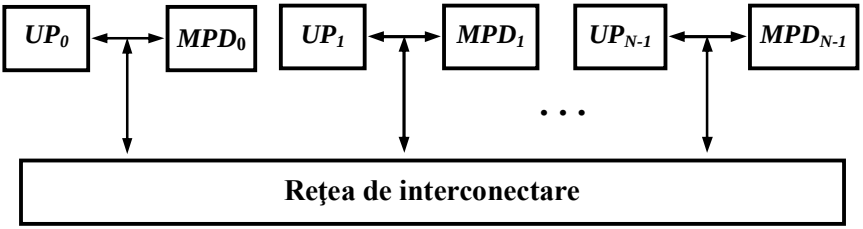


Fig. 1.21. Modelul *NUMA* de *SASA* : UP_0… UP_{N-1}– Procesoare; MPD_0… MPD_{N-1} – Memorie partajată distribuită.

- ***Modelul cu acces partajat prin memoria cache (COMA)***

La acest model, memoria ataşată fiecărui procesor este un modul de memorie *cache,* dispus ca în figura 1.22. Toate aceste module formează împreună un spaţiu global adresabil de către toate procesoarele.

Accesul la un modul de memorie *cache* (MC_i) nelocal (respectiv care este conectată nemijlocit la un alt procesor) necesită un bloc director BD_i care menţine consistenţa datelor din memoria *cache*. Noţiunea de *director* este utilizată aici în sens de catalog al datelor socate în cache.

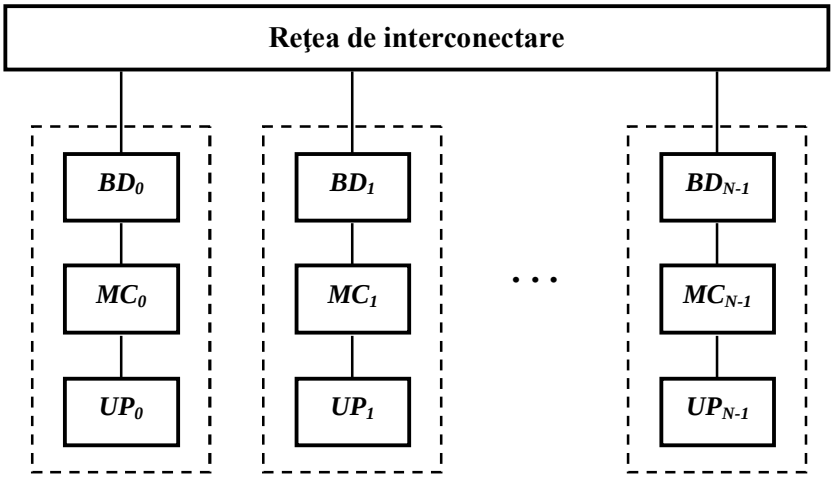


Fig. 1.22. Modelul *COMA* de *SASA* : UP_0… UP_{N-1}– Procesoare; MC_0… MC_{N-1} – Memorii cache locale; BD_0… BD_{N-1} – Blocuri Director.

1.3.2.2. Arhitecturi cu transfer de mesaje

În arhitecturile cu transfer de mesaje *Messages Passing Architecture – MPA* procesoarele sunt cuplate printr-o reţea de interconectare. După cum se observă din figura 1.23 fiecare procesor Up_i are o memorie locală ML_i , la care însă are acces numai respectivul procesor.

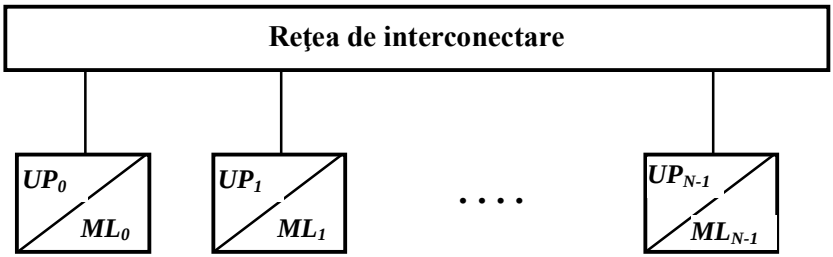


Fig. 1.23. Arhitectura cu transfer de mesaje: UP_0… UP_{N-1}– Procesoare; ML_0… ML_{N-1} –Memorii locale.

Procesoarele pot interacţiona numai prin intermediul *mesajelor* transferate prin reţeaua de interconectare. Datorită acestei organizări această arhitectură este cunoscută ca o arhitectură cu memorie distribuită şi spaţii multiple de adresă. Un calculator *MIMD* orientat pe comunicare prin transfer de mesaje este denumit *multicalculator (multicomputer)*. Datorită existenţei memoriilor locale *multicalculatoarele* sunt considerate sisteme slab cuplate *(loosely – coupled systems)* deoarece resursele sunt partajate într-o măsură redusă.

Abordările anterioare permit o clasificare a calculatoarelor *MIMD* ca fiind cele mai expresive sisteme cu procesare paralelă, clasificare ilustrată în figura 1.24.

Pot fi identificate patru subclase arhitecturale dacă se au în vedere:

- modul de organizare a spaţiului de adresă al memoriei (*spaţiu de adresă unic, partajat al memoriei sau spaţii multiple de adresă*);
- modul de amplasare fizică a memoriei (*centralizat sau distribuit)*.

Dintre cele patru subclase, cea a multicalculatoarelor cu cu memorie centralizată, nu poate fi acoperită deocamdată cu un sistem realizat fizic.

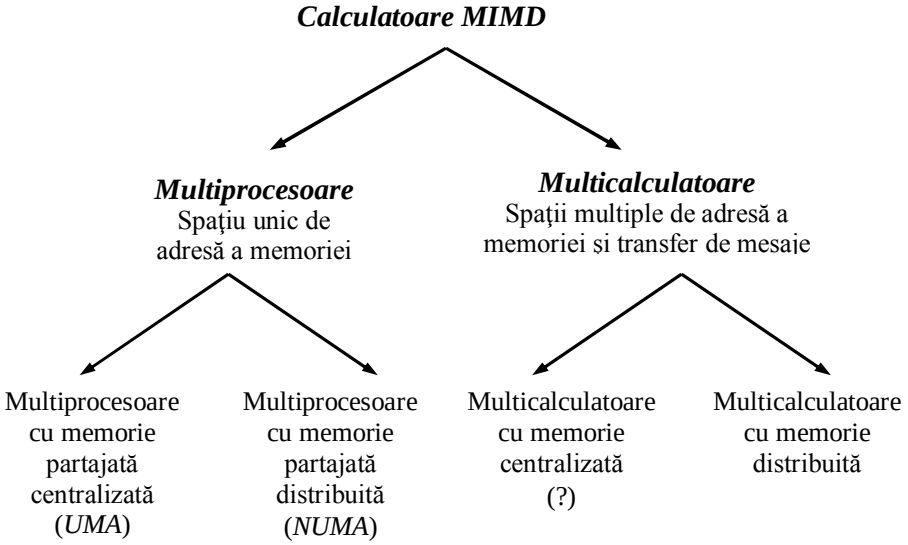


Fig. 1.24. Clasificarea calculatoarelor *MIMD.*

Referitor la calculatoarele paralele se întâlnesc mai multe clasificări, termeni şi definiţii destul de diferite funcţie de autori sau de momentul când au fost propuse.

S. Akl atribuie denumirea de *sistem distribuit (distributed system)* unui calculator paralel *MIMD* cu memorie distribuită în care există şi o distribuţie fizică a unităţilor acestuia.

După *Tanenbaum* un *sistem distribuit* reprezintă o colecţie de calculatoare independente, care apar utilizatorului ca un singur calculator. Conform aceluiaşi autor, în categoria multicalculatoarelor cu memorie distribuită sunt incluse şi reţelele locale *(LAN – Local Area Network.*

Din cele expuse rezultă că este dificil de făcut o separaţie între *sistem distribuit* şi *sistem paralel.* Totuşi se poate spune că un *calculator paralel este utilizat pentru rezolvarea unei singure aplicaţii într-un timp mai scurt (datorită multiplicării resurselor) iar un sistem distribuit poate fi accesat şi folosit de către mai mulţi utilizatori simultan.*

2. BAZELE ARITMETICE ALE CALCULATOARELOR NUMERICE

2.1. Introducere

2.2. Sisteme de numeraţie, conversii şi operaţii

2.2.1. Sisteme de numeraţie

2.2.2. Conversia unui număr dintr-o bază în alta

2.2.2.1. Metoda substituţiei

2.2.2.2. Metoda împărţirii la / înmulţirii cu noua bază

2.2.3. Operaţii aritmetice în binar octal şi hexazecimal

2.2.3.1. Operaţii aritmetice în SN binar

2.2.3.2. Operaţii aritmetice în SN octal

2.2.3.3. Operaţii aritmetice în SN hexazecimal

2.3. Reprezentarea numerelor în calculator

2.3.1. Reprezentarea numerelor în virgulă fixă

2.3.1.1. Reprezentarea unui număr prin mărime şi semn

2.3.1.2. Reprezentarea unui număr în complement faţă de 1 (cod invers)

2.3.1.3. Reprezentarea unui număr în complement faţă de 2 (cod complementar)

2.3.1.4. Reprezentarea în exces

2.3.2. Reprezentarea numerelor în virgulă mobilă

2.3.3. Coduri numerice şi alfanumerice

2.3.3.1. Coduri numerice

2.3.3.2. Coduri alfanumerice

2.3.3.3. Coduri detectoare şi corectoare de erori

2. BAZELE ARITMETICE ALE CALCULATOARELOR NUMERICE

Pentru înţelegerea aspectelor legate de structura şi funcţionalitatea sistemelor numerice de calcul se impune deţinerea unor cunoştinţe ce privesc atât prelucrarea informaţiei numerice cât şi mijloacele de implementare.

În acest capitol se prezintă unele aspecte legate de reprezentarea numerelor şi operaţii aritmetice specifice calculatoarelor numerice.

2.1. Introducere

După cum s-a arătat resursele importante ale unui calculator numeric sunt reprezentate de cele de *calcul şi comandă, memorare , intrare – ieşire şi comunicaţii.* În ceea ce priveşte memoria internă aceasta este realizată din dispozitive cu două stări stabile iar elementele de procesare a informaţiei sunt bazate pe circuite logice care operează pe baza logicii bivalente.

Aceste motive recomandă reprezentarea datelor în binar sub forma unor succesiuni de unităţi şi zerouri. Există numeroase posibilităţi pentru reprezentarea datelor, care diferă prin expresibilitate, cost de implementare, efortul de conversie de la un format la altul, etc.

Calculatoarele conţin *elemente de stocare a datelor de* tipul registrelor, cu un număr finit de elemente (ranguri) care afectează precizia calculului. Uzual se spune că reprezentarea este cu **precizie finită** aspect care presupune prezenţa erorilor de procesare.

Tipurile uzuale de date care se folosesc la nivelul *hardware* al unui calculator numeric sunt:

- **Bit:** 0, 1;
- **Şir de biţi:** secvenţe de biţi cu următoarele lungimi uzuale:
  - *tetrada:* 4 biţi;

  - *octet / byte:* 8 biţi;
  - *semicuvânt:* 16 biţi;
  - *cuvânt:* 32 biţi;
  - *dublu cuvânt:* 64 de biţi,

- **Caracter:**
  - ASCII: cod de 7 biţi,

- **Zecimal:**
  - cifrele zecimale *0 –9* codificate binar $\mathbf{0000_2 - 1001_2}$ (două cifre zecimale pot reprezentate în doi octeţi sau pot fi împachetate într-un singur octet);

- **Întreg (Virgulă Fixă):**
  - fără semn;
  - cu semn, reprezentare în:
    - cod direct (semn şi modul);
    - cod invers (complement faţă de *unu*);
    - cod complementar (complement faţă de *doi*);

- **Real (Virgulă Mobilă):**
  - precizie simplă;
  - precizie dublă;
  - precizie extinsă.

2.2. Sisteme de numeraţie, conversii şi operaţii

2.2.1. Sisteme de numeraţie

Un *sistem de numeraţie (SN)* este format din totalitatea regulilor de reprezentare a numerelor cu ajutorul unor simboluri numite *cifre*.

SN sunt de două tipuri: poziţionale şi nepoziţionale. *SN* poziţionale mai sunt cunoscute şi sub denumirea de *SN ponderate* întrucât valoarea unei cifre depinde atât de *semnificaţia sa intrinsecă* cât şi de *poziţia* acesteia în respectivul număr. De exemplu, pentru numărul **1111** reprezentat în SN zecimal fiecare cifră **1** are o altă pondere (mii, sute, zeci, unităţi). Un exemplu de sistem nepoziţional, în care ponderea nu este influenţată de poziţia cifrei este sistemul *roman*.

Datorită simplităţii de reprezentare şi efectuare a calculelor, în sistemele numerice se folosesc în exclusivitate sistemele poziţionale, un

asemenea sistem fiind caracterizat prin *bază* care reprezintă numărul total de simboluri (cifre).

Exemplu de baze uzuale:
Sistemul zecimal, b=10, simboluri: 0,1,2,3,4,5,6,7,8,9;
Sistemul binar, b=2, simboluri: 0,1;
Sistemul octal, b=8, simboluri: 0,1,2,3,4,5,6,7;
Sistemul hexazecimal, simboluri: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

Pentru un număr întreg $N \geq 0$, reprezentarea în baza *b* este un **n – tuplu**

$N = x_{n-1}\, x_{n-2} \ldots x_2\, x_1\, x_0$ care verifică următoarele două condiţii:

a : $0 \leq x_i < b$, i = n-1,…,0; $x_{n-1} \neq 0$;

b : $N = x_{n-1}b^{n-1} + \ldots + x_1 b + x_0$.

Exemplul 2.1.

b = 10, $N = 4\,523_{10}$ = $4x10^3+5x10^2+2x10+3$

b = 8, $N = 573_8$ = $5x8^2+7x8+3$

b = 2, $N = 101001_2$ = $1x2^5+0x2^4+1x2^3+0x2^2+0x2^1+1$.

Numerele reale au o reprezentare asemănătoare, însă conţin punctul fracţionar (sau virgula) care separă partea întreagă de cea fracţionară.

Pentru un număr real $r \geq 0$, reprezentarea în baza *b* este secvenţa de simboluri **(m – tuplul)**

$r = x_{n-1}\, x_{n-2} \ \ldots\ x_1\, x_0 \,.\, x_{-1}\, x_{-2} \ldots\ x_{-m}$ care verifică următoarele relaţii:

a: $0 \leq x_i < b$, i = n-1,…,0,-1,-2,…,-m; $x_{n-1} \neq 0$;

c> $r = x_{m-1}b^{m-1} + \ldots + x_1 b + x_0 + x_{-1}b^{-1} + x_{-2}b^{-2} + \ldots + x_{-n}b^{-n}$

Exemplul 2.2.

b=10, $N=154.643_0=1x10^2+5x10+4+3x10^{-1}+2x10^{-2}$

b=8, $N=623.45_8=6x8^2+2x8+3+4x8^{-1}+5x8^{-2}$

b=2, $N=101.011_2=1x2^2+0x2+1+0x2^2+1x2^1+1$.

Pornind de la faptul că la baza realizării unui sistem numeric de calcul stau dispozitivele cu două stări stabile, rezultă că SN binar (care necesită numai două cifre, **0** şi **1**) este cel mai potrivit pentru prelucrarea, codificarea şi transmiterea informaţiei în aceste echipamente. SN ale căror

baze reprezintă puteri ale lui 2 prezintă de asemenea proprietăţile sistemului binar, motiv pentru care sunt frecvent utilizate în tehnica de prelucrare automată a datelor (în special SN octal şi SN hexazecimal). în ceea ce priveşte SN zecimal acesta este cu precădere utilizat în anumite faze ale operaţiilor de intrare- ieşire.

2.2.2. Conversia unui număr dintr-o bază în alta

Într-un sistem de calcul datele sunt de regulă reprezentate în mai *multe sisteme de numeraţie.* Astfel datele de intrare şi cele de ieşire (rezultatele) sunt de regulă reprezentate în baza *10* în timp ce în memorie şi în unitatea de procesare baza de reprezentare este *2.*

Sunt situaţii în care datele care se prelucrează se reprezintă în baza *10,* cifrele zecimale fiind reprezentate prin *tetrade binare.* Pentru reducerea efortului de procesare datele se reprezintă în memorie în bazele *8* sau *16.* Existenţa şi utilizarea mai multor SN ridică problema conversiei dintr-un sistem în altul. În continuare vor fi prezentate două metode frecvent utilizate şi anume:

a - *metoda substituţiei;*

b – *metoda împărţirii la / înmulţirii cu baza.*

2.2.2.1. Metoda substituţiei

Fie numărul *N* reprezentat în baza α

$$N_\alpha = a_{n-1}\alpha^{n-1} + a_{n-2}\alpha^{n-2} + \ldots + a_0\alpha^0 + a_{-1}\alpha^{-1} + \ldots + a_{-m}\alpha^{-m}, \quad (2.1)$$

pentru care se doreşte conversia în baza β.

Metoda substituţiei presupune scrierea numărului N_α în forma

$$\begin{aligned} N_\alpha &= (a_{n-1})_\alpha (10)_\alpha^{n-1} + (a_{n-2})_\alpha (10)_\alpha^{n-2} + \\ &\cdots + (a_0)_\alpha (10)_\alpha^0 + (a_{-1})_\alpha (10)_\alpha^{-1} \cdots + (a_{-m})_\alpha (10)_\alpha^m \end{aligned} \quad (2.2)$$

şi conversia în baza β pentru fiecare cifră $(a_i)_\alpha$ şi $(10)_\alpha$.

Exemplul 2.3.

a. Să se realizeze conversia în baza 2 a numărului $N = (17)_{16}$.

Aplicând relaţia (2.2) se obţine

$$N = (17)_{16} = (1)_{16} \times (10)_{16}^{1} + (7)_{16} \times (10)_{16}^{0}$$

dar

$$(1)_{16} = (1)_2, (7)_{16} = (111)_2, (10)_{16} = (16)_{10} = (10000)_2$$

În aceste condiţii numărul *N* se va scrie

$$N = (17)_{16} = (1)_2 \times (10000)_2 + (111)_2 = (10000)_2 + (111)_2 =$$
$$= (10111)_2 = (0001\ 0111)_2$$

b. Să se realizeze conversia în baza 2 a numărului $N = (54)_8$.

Aplicând relaţia (2.2) se obţine

$$N = (52)_8 = (5)_8 \times (10)_8^{1} + (2)_8 \times (10)_8^{0}$$

dar

$$(5)_8 = (101)_2, (2)_8 = (10)_2, (10)_8 = (8)_{10} = (1000)_2$$

În aceste condiţii numărul *N* se va scrie

$$N = (52)_8 = (101)_2 \times (1000)_2 + (10)_2 = (101000)_2 + (10)_2 =$$
$$= (101010)_2 = (101\ 010)_2$$

c. Să se realizeze conversia în baza 10 a numărului $N = (25)_8$.

Aplicând relaţia (2.2) se obţine

$$N = (25)_8 = (2)_8 \times (10)_8^{1} + (5)_8 \times (10)_8^{0}$$

dar

$$(2)_8 = (2)_{10}, (5)_8 = (5)_{10}, (10)_8 = (8)_{10}$$

În aceste condiţii numărul *N* se va scrie

$$N = (25)_8 = (2)_{10} \times (8)_{10} + (5)_{10} = 2 \times 8 + 5 = (21)_{10}$$

d. Să se realizeze conversia în baza 10 a numărului $N = (10100)_2$.

Aplicând relaţia (2.2) se obţine

$$N=(10100)_2=(1)_2\times(10)_2^4+(0)_2\times(10)_2^3+(1)_2\times(10)_2^2+$$
$$+(0)_2\times(10)_2^1+(0)_2\times(10)_2^0=1\times 2^4+1\times 2^2=(20)_{10}$$

dar

$$(2)_8=(2)_{10},(5)_8=(5)_{10},(10)_8=(8)_{10}$$

În aceste condiţii numărul *N* se va scrie

$$N=(25)_8=(2)_{10}\times(8)_{10}+(5)_{10}=2\times 8+5=(21)_{10}$$

e. Să se realizeze conversia în baza 10 a numărului N = $(BF4.15)_{16}$.

Aplicând relaţia (2.2) se obţine

$$N=(BF4.15)_{16}=(B)_{16}\times(10)_{16}^2+(F)_{16}\times(10)_{16}^1+(4)_{16}\times(10)_{16}^0+$$
$$+(1)_{16}\times(10)_{16}^{-1}+(5)_{16}\times(10)_{16}^{-2}$$

dar

$$(B)_{16}=(11)_{10},(F)_8=(15)_{10},(4)_{16}=(4)_{10},$$
$$(1)_{16}=(1)_{10},(5)_{16}=(5)_{10},(10)_{16}=(16)_{10}$$

În aceste condiţii numărul *N* se va scrie

$$N=(BF4.15)_{16}=(11)_{10}\times(16)_{10}^2+(15)_{10}\times(16)_{10}^1+$$
$$+(4)_{10}\times(16)_{10}^0+(1)_{10}\times(16)_{10}^{-1}+(5)_{10}\times(16)_{10}^{-2}=$$
$$11\times 256+15\times 16+4+1\times 0.0625+5\times 0.00390625=$$
$$=2816+240+4+0.0625+0.01953125=(3060.08203125)_{10}$$

Din analiza exemplelor 2.a şi 2.b se desprinde concluzia că din bazele care sunt puteri ale lui ***2*** conversia se poate realiza *mecanic* înlocuind cifrele numărului cu combinaţiile binare corespunzătoare. În *Tabelul 2.1* se prezintă combinaţiile binare pentru cifrele sistemului octal *(triade)* respectiv hexazecimal *(tetrade)*.

Tabelul 2.1

Octal	Binar	Hexazecimal	Binar	Hexazecimal	Binar
0	000	0	0000	8	1000
1	001	1	0001	9	1001
2	010	2	0010	A	1010
3	011	3	0011	B	1011
4	100	4	0100	C	1100
5	101	5	0101	D	1101
6	110	6	0110	E	1110
7	111	7	0111	F	1111

În ceea ce priveşte conversia din binar într-o bază putere a lui **2** aceasta se poate realiza tot mecanic. În continuare se va exemplifica procedeul pentru conversiile *binar – octal* şi *binar – hexazecimal.*

Conversia binar - octal

Fie

$r = x_{n-1}\ldots x_1x_0 \bullet x_{-1}\, x_{-2}\ldots\, x_{-m}$

reprezentarea binară a unui număr real *r* oarecare.

Acelaşi număr scris ca o combinaţie liniară a puterilor lui 2 este:

$$r = x_{n-1}2^{n-1}+x_{m-2}2^{\,m-2}+\ldots x_12^1+x_02^0+x_{-1}2^{-1}+x_{-2}2^{-2}+\ldots..\, x_{-m}2^{-m}\ .$$

Deoarece o cifră octală se reprezintă printr-o grupare de trei biţi, grupăm expresia în triade:

$$r = (x_{3k+2}2^{3k+2} + x_{3k+1}2^{3k+1} + x_{3k}2^{3k})+\ldots.$$
$$+ (x_22^2 + x_12^1 + x_02^0) + (x_{-1}2^{-1} + x_{-2}2^{-2} + x_{-3}2^{-3})+\ldots.$$

sau

$$r = 2^{3k}(x_{3k+2}2^2 + x_{3k+1}2 + x_{3k}2)+\ldots.$$
$$+ (x_22^2 + x_12 + x_0) + 2^{-3}(x_{-1}2^2 + x_{-2}2 + x_{-3})+\ldots.$$

respectiv

$$r = 8^ky_k +\ldots+ 8^0y_0 + 8^{-1}y_{-1}+ \ldots \text{ cu}$$
$$0 \le\ y_i=x^{3i+2}2^2 + x^{3i+1}2 + x^{3i} < 8.$$

Rezultatul demonstraţiei de mai sus permite formularea următoarei ***reguli de conversie:*** fiind dat un număr real în formă binară , reprezentarea sa în octal se obţine grupând câte trei cifrele binare începând de la marca fracţionară (punct, virgulă), spre stânga respectiv dreapta. După ce grupele

extreme se completează (dacă este cazul cu zerouri nesemnificative), fiecare triadă se înlocuieşte cu echivalentul său octal.

Conversia binar - hexazecimal

Fie

$r = x_{n-1}\ldots x_1x_0 \bullet x_{-1}\, x_{-2}\ldots\, x_{-m}$

reprezentarea binară a unui număr real *r* oarecare.

Acelaşi număr scris ca o combinaţie liniară a puterilor lui 2 este:

$$r = x_{n-1}\,2^{n-1}+x_{n-2}\,2^{\,m-2}+\ldots x_1 2^1+x_0 2^0+x_{-1}2^{-1}+x_{-2}\,2^{-2}+\ldots..\; x_{-m}2^{-m}\;\;.$$

Deoarece o cifră hexazecimală se reprezintă printr-o grupare de patru biţi, grupăm expresia în tetrade:

$$\begin{aligned} r = {} & (x_{4k+3}2^{4k+3} + x_{4k+2}2^{4k+2} + x_{4k+1}2^{4k+1} + x_{4k}2^{4k})+\ldots. \\ & + (x_4 2^4 + x_3 2^3 + x_2 2^2 + x_1 2 + x_0) + \\ & + (x_{-1}2^{-1} + x_{-2}2^{-2} + x_{-3}2^{-3} + x_{-4}2^{-4})+\ldots. \end{aligned}$$

sau

$$\begin{aligned} r = {} & 2^{4k}(x_{4k+3}2^3 + x_{4k+2}2^2 + x_{4k+1}2 + x_{4k}2)+\ldots. \\ & + (x_3 2^3 + x_2 2^2 + x_1 2 + x_0) + 2^{-4}(x_{-1}2^3 + x_{-2}2^2 + x_{-3}2 + x_{-4})+\ldots. \end{aligned}$$

respectiv

$$r = 16^k y_k +\ldots+ 16^0 y_0 + 16^{-1}y_{-1}+ \ldots \text{ cu}$$

$$0 \le y_i = x^{4i+3}2^3 + x^{4i+2}2^2 + x^{4i+1}2 + x^{4i} < 16.$$

Ca şi în cazul precedent, rezultatul demonstraţiei de mai sus permite formularea următoarei ***reguli de conversie:*** fiind dat un număr real în formă binară , reprezentarea sa hexazecimală se obţine grupând câte *patru* cifre binare începând de la marca fracţionară (punct, virgulă), spre stânga respectiv dreapta. După ce grupele extreme se completează (dacă este cazul cu zerouri nesemnificative), fiecare tetradă se înlocuieşte cu echivalentul său hexazecimal.

După cum s-a mai spus, conversiile inverse *octal / hexazecimal* $\rightarrow$ *binar* se realizează prin înlocuirea fiecărei cifre *octale / hexazecimale* cu *triada / tetrada* corespunzătoare.

Exemplul 2.4.

a. Să se convertească în binar numerele $A=(135.72)_8$ şi $B=(5A23.B5D)_{16}$.

$(A)_2 = 001 \mid 011 \mid 101.111 \mid 010$

$(B)_2$ = 0101 | 1010 | 0010 | 0011.1011 | 0101 | 1101

b. Să se convertească în octal şi în hexazecimal numărul binar

$(C)_2$ =1011010.11011.

b1. $(C)_2$ = 001 011 010 **.** 110 110
Pe baza corespondenţei din tabelul 2.1 rezultă $(C)_8$=132.66.

b2. $(C)_2$=0101 1010 . 1101 1000
după care se face corespondenţa în conformitate cu *Tabelul 2.1* şi rezultă $(C)_{16}$=5A.D8.

2.2.2.2. Metoda împărţirii la / înmulţirii cu noua bază

Fie numărul real *N* reprezentat în baza α

$$N_\alpha = a_{n-1}\alpha^{n-1} + a_{n-2}\alpha^{n-2} + \ldots + a_0\alpha^0 + a_{-1}\alpha^{-1} + \ldots + a_{-m}\alpha^{-m}, \quad (2.3)$$

pentru care se doreşte conversia în baza β.

Pornind de la reprezentarea numerelor prin intermediul coeficienţilor dezvoltării polinomiale, în raport cu baza, conversia se poate realiza prin operaţii repetate de *împărţire / înmulţire la / cu* noua bază, după cum se converteşte partea întreagă respectiv cea fracţionară.

Deoarece se vor trata separat conversiile părţilor întreagă respectiv fracţionară, numărul se va considera descompus sub forma

$$N_\alpha = N_{\mathbf{I}\alpha} + N_{\mathbf{F}\alpha} \quad (2.4)$$

în care $N_{\mathbf{I}\alpha}$ şi $N_{\mathbf{F}\alpha}$ sunt părţile întreagă respectiv fracţionară ale numărului real N_α .

Conversia numerelor întregi

Fie partea întreagă N_I , exprimată în baza α, a unui număr real *N*:

$$N_{\mathbf{I}\alpha} = a_{n-1}\alpha^{n-1} + a_{n-2}\alpha^{n-2} + \ldots + a_1\alpha + a_0\alpha^0,$$

care în baza β poate fi exprimată astfel

$$N_{\mathbf{I}\alpha} = x_{n-1}\beta^{n-1} + x_{n-2}\beta^{n-2} + \ldots + x_1\beta + x_0\beta^0 ,$$

în care coeficienţii x_i urmează a se determina.

Determinarea acestor coeficienţi prin metoda împărţirilor succesive presupune împărţirea repetată a numărului la noua bază şi reţinerea resturilor conform procedeului de mai jos.

$$N_{I\alpha} / \beta = \underbrace{x_{n-1}\beta^{n-2} + x_{n-2}\beta^{n-3} + \ldots + x_1\beta^0}_{\text{Câtul} = (N_1)_{I\alpha}} + \underbrace{x_0/\beta}_{\text{Restul}} \quad \rightarrow x_0 ,$$

$$(N_1)_{I\alpha} / \beta = \underbrace{x_{n-1}\beta^{n-3} + x_{n-2}\beta^{n-4} + \ldots + x_2\beta^0}_{\text{Câtul} = (N_2)_{I\alpha}} + \underbrace{x_1/\beta}_{\text{Restul}} \quad \rightarrow x_1 ,$$

……………………………………………………..

$$(N_k)_{I\alpha} / \beta = \underbrace{x_{n-1}\beta^{n-k-2} + \ldots\ldots + x_{n-k-1}\beta^0}_{\text{Câtul} = (N_{k+1})_{I\alpha}} + \underbrace{x_k/\beta}_{\text{Restul}} \quad \rightarrow x_k .$$

………………………………………………………

După cum se observă în urma primei împărţiri rezultă x_0, (*cifra cea mai puţin semnificativă a rezultatului - CCMPS*), apoi x_1 ş.a.m.d. Procedeul continuă până când câtul devine mai mic decât noua bază β, ultimul rest fiind coeficientul x_{n-1}, (*cifra cea mai semnificativă a rezultatului -CCMS*).

Din cele prezentate rezultă că procesul de determinare a reprezentării în noua bază este un proces iterativ cu un număr necunoscut de paşi la începutul conversiei, întrucât se utilizează drept criteriu de *stop* obţinerea unui cât mai mic decât noua bază.

Dacă baza de plecare este **10,** iar conversia se face în baza $\boldsymbol{\beta}$, atunci numărul de iteraţii, respectiv numărul de cifre în noua reprezentare poate fi determinat înainte de începerea conversiei potrivit relaţiei

$$n = \begin{cases} \log_\beta N + 1 & \text{daca } N = \beta^k,\ k \in N \\ [\log_\beta N] + 1 & \text{daca } N \neq \beta^k, k \in N \end{cases} \qquad (2.5)$$

Conversia numerelor fracţionare

Fie partea fracţionară N_F, exprimată în baza α, a unui număr real N:

$N_{\mathbf{F}\alpha} = a_{-1}\alpha^{-1} + a_{-2}\alpha^{-2} + \ldots + a_{-m}\alpha^{-m}$,

care în baza β poate fi exprimată astfel

$N_{\mathbf{F}\alpha} = x_{-1}\beta^{-1} + x_{-2}\beta^{-2} + \ldots + x_{-m}\beta^{-m}$,

în care coeficienţii x_{-i} urmează a se determina.

Determinarea acestor coeficienţi prin metoda înmulţirilor succesive presupune înmulţirea repetată a numărului fracţionar cu noua bază şi reţinerea părţilor întregi ale produselor conform procedeului de mai jos.

$$N_{\mathbf{F}\alpha} \times \beta = \underbrace{x_{-1} + x_{-2}\beta^{-1} + \ldots + x_{-m}\beta^{-m+1}}_{(N_1)_{\mathbf{F}\alpha}} \rightarrow x_{-1} ,$$

$$(N_1)_{\mathbf{F}\alpha} \times \beta = \underbrace{x_{-2} + x_{-3}\beta^{-1} + \ldots + x_{-m}\beta^{-m+2}}_{(N_2)_{\mathbf{F}\alpha}} \rightarrow x_{-2} ,$$

……..

$$(N_{k-1})_{\mathbf{F}\alpha} \times \beta = \underbrace{x_{-k} + x_{-k-1}\beta^{-1} + \ldots + x_{-m}\beta^{-m+k}}_{(N_k)_{\mathbf{F}\alpha}} \rightarrow x_{-k} ,$$

…….

Cifra x_{-1} reprezintă CCMS iar x_{-k} CCMPS a rezultatului. Procedura se opreşte în momentul partea fracţionară $(N_k)_{F\alpha}$ a rezultatului unei înmulţiri devine zero, sau dacă s-a atins precizia de conversie specificată.

Exemplul 2.5.

a. Să se convertească în binar numărul N_{10} = 98.03125 (numărul de iteraţii admis pentru comversia părţii fracţionare: 8).

1. Se determină numărul de poziţii binare n_I necesare pentru conversia părţii întregi:

Deoarece 98 nu este o putere întreagă a lui 2

$$n_I = [\log_2 98]+1=[6.6147]+1=7 \text{ biţi}$$

2. Se converteşte partea întreagă: N_I =98

$$98 = 2 \times 49 + 0 \rightarrow \quad a_0 = 0$$

$$49 = 2 \times 24 + 1 \rightarrow \quad a_1 = 1$$
$$24 = 2 \times 12 + 0 \rightarrow \quad a_2 = 0$$
$$12 = 2 \times 6 + 0 \rightarrow \quad a_3 = 0$$
$$6 = 2 \times 3 + 0 \rightarrow \quad a_4 = 0$$
$$3 = 2 \times 1 + 1 \rightarrow \quad a_5 = 1$$
$$1 = 2 \times 0 + 1 \rightarrow \quad a_6 = 1$$

Rezultă pentru partea întreagă $(N_I)_2 = 1100010$.

3. Se converteşte partea fracţionară $(N_F)_{10} = 0.03125$.

$$0.03125 \times 2 = 0.0625 \rightarrow \quad a_{-1} = 0$$
$$0.06250 \times 2 = 0.1250 \rightarrow \quad a_{-2} = 0$$
$$0.12500 \times 2 = 0.2500 \rightarrow \quad a_{-3} = 0$$
$$0.25000 \times 2 = 0.5000 \rightarrow \quad a_{-4} = 0$$
$$0.50000 \times 2 = 1.0000 \rightarrow \quad a_{-5} = 1$$

Rezultă pentru partea fracţionară $(N_F)_2 = 0.00001$

Pentru numărul N rezultă $(N)_2 =$ **1100010.0001**.

4. Verificarea se realizează procedându-se la conversia inversă, cu *metodei substituţiei*:

$(N)_{10} = 1x2^6+1x2^5+0x2^4+0x2^3+0x2^2+1x2^1+0x2^0+0x2^{-1}+0x2^{-2}+$
$+0x2^{-3}+0x2^{-4}+1x2^{-5}=64+32+2+0.03125 = 98.0315$

b. Să se convertească în octal numărul $N_{10} = 234.046875$ (numărul de iteraţii admis pentru comversia părţii fracţionare: 6).

1. Se determină numărul de poziţii octale n_I necesare pentru conversia părţii întregi:

Deoarece 234 nu este o putere întreagă a lui 8

$$n_I = [\log_8 234]+1=[2.623]+1=3 \text{ cifre}$$

2. Se converteşte partea întreagă: $N_I = 234$

$$234 = 8 \times 29 + 2 \rightarrow \quad a_0 = 2$$
$$29 = 8 \times 3 + 5 \rightarrow \quad a_1 = 5$$
$$3 = 8 \times 0 + 3 \rightarrow \quad a_2 = 3$$

Rezultă pentru partea întreagă $(N_I)_8 = 352$

3. Se converteşte partea fracţionară $(N_F)_{10} = 0.046875$:

$$0.046875 \times 8 = 0.375 \rightarrow \quad a_{-1} = 0$$

$$0.375000 \times 8 = 3.000 \rightarrow \qquad a_{-2} = 3$$

Rezultă pentru partea fracţionară $(N_F)_8=0.03$

Pentru numărul N rezultă $(N)_2$=**352.03**

2.2.3. Operaţii aritmetice în binar octal şi hexazecimal

Operaţiile aritmetice elementare într-un SN cu baza o putere a lui 2 urmează aceleaşi reguli ca în zecimal, cu luarea în consideraţie a principiului de numărare specific fiecărui SN.

Este de menţionat că numărătoarea în sistemul zecimal este realizată prin utilizarea consecutivă a celor zece simboluri *(0, 1, 2, …, 9)*; epuizarea acestora conduce la introducerea simbolului ***1*** pe o poziţie mai la stânga şi reluarea pe poziţia iniţială a secvenţei de simboluri *(10, 11, 12, …, 19)*. Procesul continuă în mod similar prin adăugarea cifrelor **2, 3 , …, 9.** Când se ajunge la 9 în poziţia secundă se continuă cu adăugarea unei noi poziţii la stânga ş.a.m.d.

2.2.3.1. Operaţii aritmetice în SN binar

După cum s-a arătat pentru reprezentarea unui număr în *SN* binar se folosesc simbolurile **0 şi 1.** Respectând regula de numărare zecimală, în condiţiile operării în exclusivitate cu cele două simboluri se obţine următoarea secvenţă:

0, 1,
10, 11,
100, 101, 110, 111
1000, 1001, 1010,…

În *Tabelul 2.2* sunt prezentate regulile pentru adunare şi înmulţire în binar.

Tabelul 2.2

Adunare			*Înmulţire*		
+	0	1	**x**	0	1
0	**0**	**1**	0	**0**	**0**
1	**1**	**10**	1	**0**	**1**

În continuare se prezintă câte un exemplu pentru fiecare dintre operaţiile elementare (*adunare, scădere, înmulţire şi împărţire).*

Adunare	*Scădere*
1011011.10110+	1101110.10111-
1101111.01101	0001011.01110
11001011.00011	1100011.01001

Înmulţire	*Împărţire*
1011.11x	11101110.111 | 1101
111.01	1101 | **10010.011**
101111	0001111
000000	1101
101111	0010011
101111	1101
101111	001101
10011100011	1101
	0000
1001110.0011	

2.2.3.2. Operaţii aritmetice în SN octal

După cum s-a arătat pentru reprezentarea unui număr în SN octal se utilizează simbolurile **0, 1, 2, …,7.** Aplicând regula de numărare zecimală, în condiţiile operării în exclusivitate cu cele opt simboluri se obţine următoarea secvenţă:

0, 1,2,3,4,5,6,7
10, 11,12,13,…,17
20,21,23,23,…,27
…..……..……..

În *Tabelul 2.3* sunt prezentate regulile pentru adunare şi înmulţire în octal.

Tabelul 2.3

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

x	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

Ca şi în cazul precedent, în cele ce urmează se prezintă câte un exemplu pentru fiecare dintre operaţiile elementare (*adunare, scădere, înmulţire şi împărţire).*

Adunare

```
 257.432 +
 144.613
---------
 424.245
```

Scădere

```
 4123.516 -
  567.307
---------
 3334.207
```

Înmulţire

```
 1753.216 x
      4.3
---------
 5701652
 7655070
---------
 104452552

 10445.2552
```

Împărţire

```
 76321.542 : 23.11

 7632154.2 | 2311
 7133      | 3205.32
 =4771
  4622
  =14754
   13755
   ==7772
     7133
     =6370
      4622
      1546
```

2.2.3.3. Operaţii aritmetice în SN hexazecimal

Reprezentarea unui număr în SN cu baza 16 (hexazecimal) implică utilizarea simbolurilor **0, 1, 2, …,7, 8, 9, A, B, C, D, E, F.** Prin aplicarea regulii de numărare zecimală, în condiţiile operării cu cele 16 simboluri specifice rezultă următoarea secvenţă:

0, 1, 2, 3, 4, 5, 6 ,7, 8, 9, A, B, C, D, E, F
10, 11,12,13,…,1D, 1E, 1F
20,21,23,23,…,2f
……..

În *Tabelele 2.4 şi 2.5* sunt prezentate regulile pentru adunare şi înmulţire în hexazecimal.

Tabelul 2.4

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Tabelul 2.5

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D

4	0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

În continuare se prezintă câte un exemplu pentru fiecare dintre operaţiile elementare (*adunare, scădere, înmulţire şi împărţire)* efectuate în SN hexazecimal.

Adunare

```
 7FA4 +
 521D
------
 D1C1
```

Scădere

```
 C412 -
 AFB5
------
 145D
```

Înmulţire

```
 14A.2F  x
    B.1
---------
  14A2F
 E3005
---------
 E44A7F
```

E44.A7F

Împărţire

```
 A57BF | 3A
 74    |-----
 317   | 2DA6.9A
 2F2
 =25B
  244
  =17F
   15C
    230
    20A
     260
     244
      1C
```

2.3. Reprezentarea numerelor în calculator

Uzual un echipament de calcul numeric preia datele şi oferă rezultatele într-o formă accesibilă utilizatorului. În ceea ce priveşte prelucrarea, aceasta presupune exprimarea datelor într-o formă specifică procesorului. În consecinţă există două formate de reprezentare a numerelor în calculator şi anume *formatul intern* şi *formatul extern.*

Pentru reprezentarea internă se utilizează SN binar în mai multe forme diferenţiate de soluţia aleasă pentru *indicarea poziţiei virgulei şi semnului numărului.* În acest sens poziţia fixă sau variabilă a delimitatorului fracţionar (punct sau virgulă) determină reprezentarea în *virgulă fixă* sau respectiv în *virgulă mobilă.*

2.3.1. Reprezentarea numerelor în format virgulă fixă

În calculator se operează cu numere de lungime fixă, numărul de poziţii binare fiind determinat de lungimea registrelor de memorare.

În formatul *virgulă fixă,* virgula (punctul zecimal) nu este reprezentată fizic în calculator, poziţia sa fiind stabilită prin proiectare şi neputând fi schimbată. Uzual virgula se consideră plasată în faţa celei mai semnificative cifre a numărului, în aceste condiţii numerele cu care se operează fiind *subunitare.*

Observaţie importantă

Numărul de biţi pe care se reprezintă un număr este *finit şi fix,* stabilindu-se în faza de proiectare a calculatorului. Din acest motiv în calculator nu se poate reprezenta decât un număr finit de valori, care pot fi interpretate diferit în cele două formate (intern sau extern). *Numerele care se pot reprezenta în calculator se numesc* ***numere cu precizie finită (NPF)*** *şi au proprietăţi diferite faţă de numerele din matematică.*

În cazul acestora poate apărea depăşirea capacităţii de memorare deoarece *NPF* au un domeniu finit de valori, în sensul că nu pot exista numere oricât de mari sau oricât de mici. Depăşirile pot fi detectate hardware sau software. Următorul exemplu demonstrează că în cadrul NPF nu este valabilă proprietatea de *asociativitate.*

Fie o maşină cu următorul format admisibil:

S			

în care se pot reprezenta numere întregi cuprinse între –999 şi +999.

Fie a = 600, b=500, c=400

În aritmetica clasică a+(b-c) = (a+b)-c respectiv 600+(500-400) =700 şi (600+500)-400=700 asociativitatea fiind respectată.

În cazul NPF (calculatoare) a+(b-c) = 600+(500-400) = 700, iar (a+b)-c = $\underbrace{(600+500)}_{1100}$ - 400 = eroare.
1100 - depăşire

Numerele întregi fără semn se reprezintă prin corespondentul lor binar (codul direct) numărul de biţi pentru reprezentarea unui număr N fiind $2^{n-1} \le N < 2^n$.

În general domeniul finit de valori pentru numerele întregi, reprezentate în formatul cu virgulă fixă este:

$$D = [V_{\min}, V_{\max}] \cap Z, \qquad (2.6)$$

unde V_{min} *şi* V_{max} sunt cea mai mică respectiv cea mai mare valoare care se pot reprezenta pe *n* biţi. Cele 2^n valori distincte pot constitui reprezentări ale unor numere întregi *pozitive sau negative.*

Pentru numerele cu semn există trei reprezentări mai des utilizate şi anume:

1. *semn – mărime;*
2. *complement faţă de 2;*
3. *complement faţă de 1.*

2.3.1.1. Reprezentarea unui număr prin mărime şi semn

Scrierea unui număr prin mărime şi semn (sau în cod direct) este dată de relaţia

$$N = a_n 2^n + \sum_{i=-m}^{n-1} a_i 2^i \qquad (2.7)$$

unde a_n este bitul de semn, iar a_i sunt biţii numărului *N*.

În privinţa semnului a_n care ocupă bitul *cel mai semnificativ* (**MSb** –**M**ost **S**ignificant **b**it) convenţia este următoarea:

$a_n = 0$, dacă $N \ge 0$;

$a_n = 1$, dacă $N < 0$.

Considerând reprezentarea pe *m+n* biţi, conform relaţiei (2.7) primul bit din stânga este asociat **semnului**, iar restul de *n+m-1* biţi conţin **mărimea** numărului egală cu reprezentarea binară a valorii $|N|$ - figura 2.1.

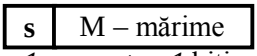


Fig. 2.1. Reprezentarea *semn – mărime* pe *n+m* biţi.

Exemplul 2.6.

a) + 9 = 0 1001
valoarea numărului
bitul de semn

b) - 6 = 1 0110
valoarea numărului
bitul de semn

c) -0.6875 = 1 1011
valoarea numărului
bitul de semn

Reprezentarea *semn – mărime* prezintă avantajul apropierii de scrierea naturală, însă din punctul de vedere al realizării calculelor prezintă unele dezavantaje. Acestea se referă în primul rând la necesitatea examinării înainte de realizarea unei operaţii (adunare sau scădere) a bitului de semn. Pe de altă parte, unitatea aritmetică se poate mult simplifica dacă este orientată numai către operaţia de adunare. Acest deziderat se poate realiza prin alegerea convenabilă a reprezentării numerelor, recomandabilă fiind reprezentarea în C2 (complement faţă de 2).

Un alt dezavantaj important este legat de reprezentarea numărului zero care este afectată de semn respectiv

$$[+0]_d = \mathbf{0}\ 000\ldots\ 000\ ;$$
$$[-\ 0]_d = \mathbf{1}\ 000\ldots\ 000\ .$$

Această dublă reprezentare face ca un număr *N* întreg reprezentat în cod direct pe *n* ranguri să poată lua valori în gama

$$-2^{n-1}+1 \le N \le 2^{n-1}-1\ , \qquad (2.8)$$

ceea ce, de exemplu, face ca pentru

- n=8 domeniul de reprezentare să fie $D = [-127,+127] \cap Z$;

- n=16 domeniul de reprezentare să fie $D = [-32767,+32767] \cap Z$.

2.3.1.2. Reprezentarea unui număr în complement faţă de 1 (cod invers)

Pentru numere pozitive N ≥ 0 reprezentarea în complement faţă de *1* este identică cu reprezentarea *semn – mărime.*

Dacă N < 0, atunci bitul de semn s = 1 şi

$$C1(|N|) = 2^{n-1} - 1 - |N| , \qquad (2.9)$$

unde *n-1 este* numărul de biţi utilizaţi pentru reprezentarea mărimii (fără semn).

Calculul C(1) se poate face prin două metode şi anume:

a. *utilizând definiţia,* respectiv
   $s = 1 \qquad C1(|N|) = 2^{n-1} - 1 - |N|$;
b. prin *inversarea biţilor* reprezentării cu semn a valorii absolute $|N|$ a numărului *N.*

Exemplul 2.6.

Fie n-1 = 7 şi N = -123; să se calculeze prin cele două metode C1(N)

a. |N| = 123; C1(|N|) = $2^7 - 1 - 123 = 128 - 1 - 123 = 4$ =0000 100
   C1(N) = 1 0000100 = $(1x2^7 + 1x2^2)_{10} = (132)_{10}$
b. $(|N|)_d$ = 0 1111011
   C1(N) = 1 0000100 (s-a inversat fiecare bit).

În ceea ce priveşte reprezentarea numărului zero şi în acest caz aceasta este afectată de semn respectiv

$[+0]_i$ = **0** 000… 000 ;

$[-0]_d$ = **1** 111… 111 .

Această dublă reprezentare face ca un număr *N* întreg reprezentat în cod direct pe *n* ranguri să poată lua valori în aceiaşi gamă ca în cazul codului direct.

2.3.1.3. Reprezentarea unui număr în complement faţă de 2 (cod complementar)

Pentru numere pozitive N > 0 reprezentarea în complement faţă de 2 este identică cu reprezentările *semn – mărime* şi în *cod complementar faţă de 1.*

Dacă N < 0, atunci s = 1

$$C2(|N|) = 2^{n-1} - |N| , \qquad (2.10)$$

unde *n-1 este* numărul de biţi utilizaţi pentru reprezentarea mărimii (fără semn).

Calculul *C2* se poate face prin trei metode şi anume:

a. *utilizând definiţia,* respectiv

$s = 1 \qquad C2(|N|) = 2^{n-1} - |N|$;

b. prin *inversarea biţilor* reprezentării cu semn a valorii absolute $|N|$ a numărului *N* la care se adaugă 1, în poziţia cea mai puţin semnificativă;

c. prin *analiza* de la dreapta la stânga a reprezentării cu semn a valorii absolute $|N|$ a numărului *N*. Primii biţi de 0 şi primul bit de 1 se lasă neinversaţi, apoi se inversează toţi biţii, inclusiv bitul de semn (dacă primul bit întâlnit este *1* se lasă neschimbat numai acesta).

Exemplul 2.7.

1. Fie n-1 = 7 şi N = -123; să se calculeze prin cele trei metode C2(N)

a. |N| = 123; M = 2^7 – 123 = 128 – 123 = 5
C2(N) = 1 0000101

b. $(|N|)_d$ = 0 1111011
C1(N) = 1 0000100 + (s-a inversat fiecare bit).
1
C2(N) = 1 0000101

c. 1. $(|N|)_d$ = 0 1111011

2. se analizează biţii de la dreapta spre stânga şi se inversează

C2(N) = *1 000010* **1** ← bit neschimbat
biti inversati

2. Fie n-1 = 7 şi N = -96; să se calculeze prin cele trei metode C2(N)

a. |N| = 96; M = 2^7 – 96 = 128 – 96 = 32
C2(N) = 1 0100000

b. $(|N|)_d$ = 0 1100000
C1(N) = 1 0011111 + (s-a inversat fiecare bit).
1
C2(N) = 1 0100000

c. 1. (|N|) = *0 1* **100000**

2. se analizează biţii de la dreapta spre stânga şi se inversează

C2(N) = *1 0* **100000** ← biţi neschimbaţi
biţi inversaţi

Reprezentarea în *complement faţă de 2* este cea mai utilizată pentru numerele algebrice datorită în primul rând faptului că elimină ambiguităţile legate de reprezentarea numărului zero.

În *C2* numărul zero are o reprezentare unică şi anume

$[0]_c$ = **0** 000… 000 .

Această unică reprezentare face ca un număr *N* întreg reprezentat în cod complementar pe *n* ranguri să poată lua valori în gama

$$-2^{n-1} \le N \le 2^{n-1}-1 \quad , \qquad (2.11)$$

ceea ce va implica, de exemplu, pentru

- n=8 domeniul de reprezentare să fie $D=[-128,+127]\cap Z$;
- n=16 domeniul de reprezentare să fie $D=[-32768,+32767]\cap Z$.

2.3.1.4. Reprezentarea în exces

Necesitatea acestui tip de reprezentare este impusă de anumite avantaje pe care le oferă tratarea numerelor fără semn. Ideea care stă la baza acestei reprezentări cunoscută şi sub denumirea de *reprezentare deplasată* constă în adunarea la fiecare număr a unui *exces* reprezentat de

regulă de cel mai mic număr întreg negativ care se poate reprezenta în calculator.

De exemplu în reprezentarea C2 pe un octet domeniul

$$D = [-128, +127] \cap Z$$

se transformă prin adunarea excesului *128* în

$$D_{exces128} = [0, +255] \cap Z .$$

În aceste condiţii celui mai mic număr negativ $(-128)_{10}$ îi va corespunde $(00000000)_2$ iar celui mai mare număr pozitiv $(+127)_{10}$ îi va corespunde $(11111111)_2$. Această reprezentare simplifică efectuarea operaţiei de comparare a numerelor, ceea ce este esenţial pentru realizarea *hardware-ului* implicat în operarea exponenţilor la reprezentarea în virgulă mobilă.

*

* *

După cum s-a văzut prin reprezentarea în format virgulă fixă se precizează un număr de ranguri la stânga *virgulei* pentru partea întreagă, respectiv la dreapta acesteia pentru partea fracţionară. În aceste condiţii gama şi precizia de reprezentare sunt determinate de numărul de ranguri. De exemplu pe 40 de ranguri pot fi manevrate numere de ordinul *trilioanelor* (10^{12}) precizia fiind de o trilionime.

Solicitări suplimentare ar putea fi rezolvate prin creşterea numărului de ranguri, aspect care nu este posibil întrucât acest număr este fixat pentru un anumit tip de calculator.

2.3.2. Reprezentarea numerelor în virgulă mobilă

Necesitatea reprezentării în calculator a numerelor foarte mari sau foarte mici, cu o precizie ridicată a condus la reprezentarea în virgulă mobilă (*VM*) .

Reprezentarea în format VM utilizează reprezentarea ştiinţifică

$$r = m \times b^E \qquad (2.12)$$

căreia îi sunt asociate trei componente:

- b – baza în care se reprezintă numărul;
- E – exponentul; care indică ordinul de mărime al numărului;

- *m* – mantisa asociată mărimii exacte a numărului într-un anumit domeniu.

Exemplul 2.8
$6.342 = 0.6342 \times 10^{1} = 6.342 \times 10^{0}$;
$0.0057 = 0.57 \times 10^{-2} = 5.7 \times 10^{-3}$;
$5760 = 0.576 \times 10^{4} = 5.76 \times 10^{3}$.

Considerăm că pentru reprezentarea unui număr în virgulă mobilă se utilizează *n* biţi din care *e* pentru exponent (care determină intervalul de valori) şi *m* biţi pentru mantisă (care determină precizia reprezentării). După cum se ştie cu *n* biţi se pot reprezenta 2^n numere reale.

Între numere reale din matematică şi numerele reale reprezentate în formatul cu virgulă mobilă există diferenţe legate de domeniul finit şi mulţimea discretă de valori. Domeniul finit de valori este determinat de capacitatea fizică limitată a memoriei calculatorului.

Pentru a vedea cum se poate determina acest domeniu vom considera o maşină ipotetică zecimală (*baza de numeraţie este 10)* în care:

- mantisa *m* este subunitara si pozitiva respectiv, $0 \le m < 1$;
- mantisa se reprezintă ca un număr cu semn şi trei cifre, în domeniul $0.1 \le m < 1$ sau zero;
- exponentul se reprezintă ca un număr cu semn şi două cifre.

Structura unui număr cu caracteristicile expuse precum şi valorile pozitive minimă şi maximă sunt:

	mantisa	**exponent**
Structura:	± □□□	± □□

$-V_{max} = -\,9\ 9\ 9 \;+\; 9\ 9 \;=\; -\ 0.999 \times 10^{+99}$
$-V_{min} = -\,1\ 0\ 0 \;-\; 9\ 9 \;=\; -\ 0.100 \times 10^{-99}$
$+V_{min} = +\,1\ 0\ 0 \;-\; 9\ 9 \;=\; +\ 0.100 \times 10^{-99}$;
$+V_{max} = +\,9\ 9\ 9 \;+\; 99 \;=\; +\ 0.999 \times 10^{+99}$.

Numărul de mantise pozitive distincte este Nm = 1800 {900 pentru $m \in [-999 , -100] \cap Z$ si 900 $m \in [+100 , +999]\} \cap Z$, iar numărul de exponenţi este Ne = 199 (toate valorile întregi intre –99 si +99 inclusiv 0), deci rezultă că în acest format se pot reprezenta în total N numere (pozitive şi negative)

$$N = Nm \times Ne + 1 = 1800 \times 199 + 1 = 358201 \text{ numere.}$$

O reprezentare a valorilor limite pe axa reală permite evidenţierea situaţiilor in care se produce depăşire. Determinarea valorilor V_{min} şi V_{max} permite identificarea pe axa reală, figura 2.2, a domeniilor de valori reprezentabile.

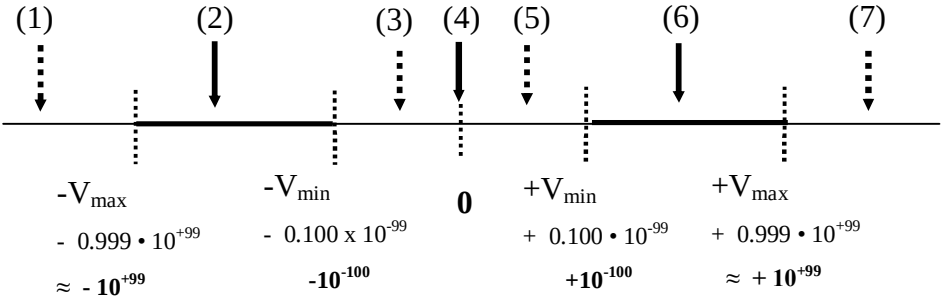


Fig. 2.2. Domeniile valorilor reale reprezentabile în *VM*.

Zonele marcate cu cifre in figura 2.2 reprezintă următoarele categorii de numere:

1. numere negative mai mici decât - $0.999 \times 10^{+99}$ ($\approx -10^{+99}$) , dar foarte mari in valoare absolută;
2. numere negative intre - $0.999 \times 10^{+99}$ şi - 0.100×10^{-99} (-10^{+99}, -10^{-100});
3. numere negative foarte mici cu valoare absolută mai mică decât 0.100×10^{-99} (10^{-100});
4. *zero*;
5. numere pozitive foarte mici cu valoare absolută mai mică decât 0.100×10^{-99}(10^{-100});
6. numere pozitive intre + 0.100×10^{-99} si + $0.999 \times 10^{+99}$ ($+10^{-100}$, -10^{+99});
7. numere foarte mari in valoare absolută, pozitive mai mari decât + $0.999 \times 10^{+99}$ ($+10^{+99}$);

Domeniul de reprezentare va fi format din zonele (**2** – *numere negative*) şi (**6** - *numere pozitive*) la care se adaugă zona **4** reprezentată de numărul zero. Dacă în urma unei operaţii aritmetice rezultă o valoare în zonele (**1**) sau (**7**) apare *depăşirea flotantă* (inferioară dacă *rezultat* < $-V_{min}$ sau superioară dacă *rezultat* > $+V_{max}$). Dacă rezultatul se situează în zonele (3) sau (5) se semnalizează imposibilitatea reprezentării pentru ca numerele sunt prea mici (negative – zona **3**, pozitive – zona **5**).

Caracterul discret al reprezentării în virgulă mobilă este dat de faptul că mulţimile din zonele (**2**) şi (**6**) ale schemei din figura 2.2 sunt mulţimi finite. Având în vedere că mulţimea numerelor reale este o mulţime continuă (între orice două numere reale se găseşte o infinitate de asemenea numere) rezultă că nu există o corespondenţă biunivocă între această mulţime şi mulţimea numerelor reprezentabile în virgulă mobilă.

Numărul de cifre al mantisei determină *precizia de reprezentare* în domeniile **2** si **6** in timp ce numărul de cifre al exponentului afectează *mărimea aceloraşi* domenii.

Mantisa se reprezintă uzual in forma normalizată (*prima cifră din stânga diferită de zero)* si intr-una din bazele **2, 8, 16.**

Pentru a nu folosi exponenţi negativi se introduce noţiunea de **caracteristică.** Aceasta este egală cu exponentul cu semn deplasat, astfel încât să ia valori intr-o mulţime de numere pozitive. De exemplu pentru un *exponent* in domeniul [-64, +63]∩Z, o deplasare cu +64 va conduce la o *caracteristica* in domeniul [0, +127]∩Z.

Stabilirea formatului de reprezentare a numerelor reale în virgulă mobilă a constituit obiectul a numeroase studii care s-au concretizat în elaborarea în anul 1985 a standardului IEEE[1] 754, când au fost alese două forme standard:

- *formatul scurt* sau *de bază* (figura 2.3) , pe un cuvânt de 32 de biţi care asigură o viteză mai mare de operare;

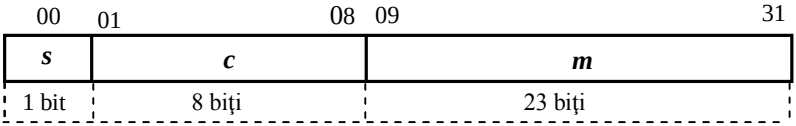


Fig. 2.3. Formatul scurt în *VM* al Standardului IEEE 754.

- *formatul lung* sau *dublu de bază* (figura 2.4), pe un cuvânt de 64 de biţi care asigură o precizie mai mare de lucru.

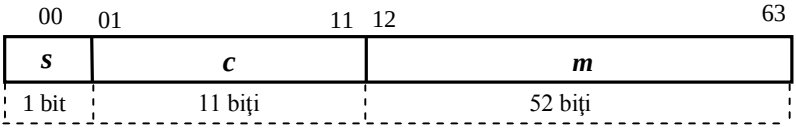


Fig. 2.4. Formatul lung în *VM* al Standardului IEEE 754.

[1] IEEE - Institute of Electrical and Electronics Engineers www.ieee.org

În formatul scurt *(simplu de lucru)* semnificaţiile sunt următoarele:

- bitul 00: *s = Semnul mantisei* (*s = 1* – mantisă negativă);
- biţii 01- 08: *c = Caracteristică (exponent deplasat cu* $2^7 - 1 = 127$);
- biţii 09 - 31: *f = Mantisa normalizată (fracţia)* [2].

Valoarea unui număr *r* reprezentat într-un asemenea format poate fi:

- dacă $c = 255$ şi $m \neq 0$ atunci r = NAN (nu este un număr – **N**ot **A** **N**umber);
- dacă $c = 255$ şi $m = 0$ atunci $r = (-1)^s \times \infty$;
- dacă $0 < c < = 255$ atunci; $r = (-1)^s \times 2^{e-(2^7-1)} \times (1.m)$;(2.13)
- dacă $c = 0$ şi $m \neq 0$ atunci ; $r = (-1)^s \times 2^{-126} \times (0.m)$
- dacă $c = 0$ şi $m = 0$ atunci $r = (-1)^s \times 0 (zero)$.

În particular se obţin următoarele reprezentări în format scurt

0 00000000 00000000000000000000000 = +0
1 00000000 00000000000000000000000 = -0

0 11111111 00000000000000000000000 = +∞
1 11111111 00000000000000000000000 = -∞

0 11111111 00000100000000000000000 = NaN
1 11111111 00100010001001010101010 = NaN

0 10000000 00000000000000000000000 = +1 * 2**(128-127) * 1.0 = 2

0 00000001 00000000000000000000000 = +1 * 2**(1-127) * 1.0 = 2**(-126)
0 00000000 10000000000000000000000 = +1 * 2**(-126) * 0.1 = 2**(-127)
0 00000000 00000000000000000000001 = +1 * 2**(-126) * 0.00000000000000000000001 = 2**(-149) (Cea mai mică valoare pozitivă)

[2] când e≠0, se presupune un bit egal cu **1** la stânga lui *f;* virgula se consideră plasată între acest bit şi primul bit explicit al mantisei.

Conform relaţiei (2.13) mărimea numerelor reprezentate este în gama

$$. - 2^{-126} \times (1.0) \le r \le 2^{127} \times \left(2 - 2^{-23}\right)$$

În formatul lung *(dublu de bază)* semnificaţiile sunt următoarele:
- bitul 00: *s = semnul mantisei* (*c = 1 –* mantisă negativă);
- biţii 01 - 11: *e = Caracteristică (exponent deplasat cu* $2^{10} - 1 = 1023$);
- biţii 12 - 63: *fm= mantisa normalizată/ fracţia* [3].

Exemplul 2.9.

Să se reprezinte în formatul scurt (baza 2) numărul r = -348.5625_{10}

a. se converteşte *r* in baza 2

348 = 174 x 2 + 0
174 = 87 x 2 + 0
87 = 43 x 2 + 1
43 = 21 x 2 + 1
21 = 10 x 2 + 1 **r_I = 101011100**
10 = 5 x 2 + 0
5 = 2 x 2 + 1
2 = 2 x 1 + 0
1 = 0 x 2 + 1

0.5625 x 2 = **1**.125
0.1250 x 2 = **0**.250
0.2500 x 2 = **0**.500 **r_F = 0.1001**
0.5000 x 2 = **1**.000

r =1010 11100.1001

b. se trece rezultatul in forma normalizată

r =1.**0101 1100** 1001 0000 0000 000 x 2^8 (pentru a se obţine mantisa pe 23 de biţi s-a completat două grupe de câte 4 şi o grupă cu 3 de zero;

c. se reprezintă în VM

s = 1 (negativ)

[3] idem 2.

$c = 127 + 8 = 135_{10} = 87_{16} = 1000\ 0111_2$
m= 0101 1100 1001 0000 0000 000

Rezultă reprezentarea in virgulă mobilă
r = 1 1000 0111 0101 1100 1001 0000 0000 000

d. Verificare

$r = (-1)^1 \times 2^{135-127} \times 1.\ 0101\ 1100\ 1001\ 0000\ 0000\ 000$

$1.\ 1101\ 1100\ 1001\ 0000\ 0000\ 000 = 1+2^{-1}+2^{-2}+2^{-4}+2^{-5}+2^{-6}+2^{-9}+2^{-12}$
$= 1+0.25+0.0625+0.03125+0.015625+0.001953125+$
$+0.000244140625=1.361572265625$
$r = (-1)^1 \times 2^8 \times 1.361572265625=-256x1.361572265625=348.5625$

La analiza reprezentării în *VM* interesează următoarele aspecte:
- numărul numerelor reprezentabile;
- numărul care are valoarea/mărimea cea mai mare;
- numărul diferit de zero care are valoarea/mărimea cea mai mică;
- determinarea celei mai mari distanţe dintre două numere succesive;
- determinarea celei mai mici distanţe dintre două numere succesive.

2.3.3. Coduri numerice şi alfanumerice

După cum s-a arătat calculatorul prelucrează numere reprezentate în SN binar în timp ce utilizatorul uman operează curent cu numere exprimate în SN zecimal, cu cuvinte specifice unei anumite limbi, cu anumite semne grafice, etc. Pentru translatarea între celor două maniere de reprezentare, evident fără alterarea conţinutului informaţional se utilizează *codificarea*.

A codifica elementele unei mulţimi A prin elementele unei alte mulţimi B înseamnă a realiza o corespondenţă între fiecare element $a \in A$ o secvenţă de elemente $b \in B$.

Un *cod* este o funcţie $f : A \to I$ unde I este mulţimea secvenţelor care se pot forma cu elemente din B. În ceea ce priveşte mulţimea A acesta se numeşte *alfabet* iar elementele acesteia (cifre, litere, semne de punctuaţie etc.) sunt caractere. Codurile în care sunt reprezentate numai

numere se numesc *coduri numerice* iar cele în care pe lângă numere conţin şi litere, caractere speciale etc. se numesc coduri *alfanumerice.*

Având în vedere reprezentarea internă în cod binar, respectiv $B=\{0,1\}$, în continuare vor fi tratate coduri care codifică litere şi cifre prin combinaţii de elemente ale mulţimii *B.*

Dacă mulţimea *A* are un număr de *N* caractere, un caracter poate fi codificat printr-o secvenţă de *n* elemente din *B* dacă

$$N \le 2^n \qquad (2.13)$$

Exemplul 2.10

Să se determine numărul minim de biţi care pot codifica cele zece cifre ale SN zecimal.

Din relaţia (2.13) rezultă $n \ge \log_2 N$ respectiv $n \ge \log_2 10 = \frac{1}{\log 2} = 3.322$. În aceste condiţii rezultă că numărul minim de biţi utilizaţi pentru codificarea unei cifre zecimale este n=4.

2.3.3.1. Coduri numerice

Codurile *binar-zecimale* care asigură codificarea binară a cifrelor zecimale pot fi *ponderate sau neponderate.*

Codurile ponderate asociază fiecărei cifre zecimale o tetradă binară, în care fiecare rang are o anumită pondere. Pentru exemplificare în *Tabelul 2.6* se prezintă o serie de coduri numerice ponderate cu o frecvenţă mai ridicată de utilizare.

Tabelul 2.6

Zecimal	*8421*	*2421*	*4221*	*5421*	*7421*
0	0000	0000	0000	0000	0000
1	0001	0001	0001	0001	0001
2	0010	0010	0010	0010	0010
3	0011	0011	0011	0011	0011
4	0100	0100	0110	0100	0100
5	0101	1011	1001	1000	0101
6	0110	1100	1100	1001	0110
7	0111	1101	1101	1010	0111
8	1000	1110	1110	1011	1001
9	1001	1111	1111	1100	1010

- *Codul 8421* este un cod în care cifrele binare de la stânga la dreapta au respectiv ponderile 2^3, 2^2,2^1, 2^0.
- *Codurile 2421 şi 4221* se caracterizează prin utilizarea ponderii 2 în două poziţii distincte ale tetradei, iar tetradele care reprezintă cifre zecimale a căror sumă este egală cu 9 sunt complementare. O altă particularitate constă în faptul că primele cinci tetrade au în prima poziţie ***0*** iar ultimele cinci au în aceiaşi poziţie cifra ***1.***
- *Codul 5421*este caracterizat de faptul că cifrele 5-9 se deosebesc de cifrele 0-4 numai prin valoarea primului bit.
- *Codul 7421* utilizează ponderile 7, 4, 2, 1. Aceste ponderi introduc o ambiguitate în ceea ce priveşte reprezentarea cifrei 7 (0111 sau 1000). Pentru înlăturarea acesteia se introduce o restricţie şi anume considerarea combinaţiei cu cei mai mulţi biţi ***1*** respectiv 0111.

Codurile neponderate nu presupun existenţa câte unei ponderi care să fie asociată fiecărei tetrade binare. În Tabelul 2.7 sunt prezentate câteva coduri neponderate create pornind de la considerente impuse de utilizarea lor.

Tabelul 2.7

Zecimal	*Exces 3*	*Gray*	*2 din 5*
0	0011	0000	00011
1	0100	0001	00101
2	0101	0011	00110
3	0110	0010	01001
4	0111	0110	01010
5	1000	0111	01100
6	1001	0101	10001
7	1010	0100	10010
8	1011	1100	10100
9	1100	1101	11000

- *Codul Exces 3* se obţine din codul 8421 pri adunarea la fiecare tetradă a cifrei 3 (0011) în binar. Rezultă un cod cu

proprietatea de autocomplementare[4] din care s-a eliminat combinaţia 0000[5] .

- *Codul Gray* se caracterizează prin aceea că trecerea de la echivalentul binar al unei cifre zecimale la următorul se face prin modificarea unui singur rang binar.
- *Codul 2 din 5* utilizează pentru codificarea cifrelor zecimale cinci poziţii binare, în condiţiile în care fiecare combinaţie trebuie să conţină *câte doi biţi unitari.* Această restricţie asigură unicitatea reprezentării şi creează posibilitatea controlului asupra transmisiei informaţiilor codificate în acest mod.

2.3.3.2. Coduri alfanumerice

Numărul limitat al caracterelor alfanumerice (spre deosebire de numerele reale la care gama este infinită) permite ca un întreg set de caractere să fie reprezentat pe un număr limitat de biţi. În practică s-au impus două sisteme de codificare a caracterelor alfanumerice ASCII şi EBCDIC.

*Codul ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange)* utilizează pentru reprezentarea caracterelor alfanumerice câte 7 biţi pe caracter.

Tabelul 2.8

HEX	DEC	CHR	CTRL	HEX	DEC	CHR	HEX	DEC	CHR	HEX	DEC	CHR
00	0	NUL	^@	20	32	SP	40	64	@	60	96	`
01	1	SOH	^A	21	33	!	41	65	A	61	97	a
02	2	STX	^B	22	34	"	42	66	B	62	98	b
03	3	ETX	^C	23	35	#	43	67	C	63	99	c
04	4	EOT	^D	24	36	$	44	68	D	64	100	d
05	5	ENQ	^E	25	37	%	45	69	E	65	101	e
06	6	ACK	^F	26	38	&	46	70	F	66	102	f
07	7	BEL	^G	27	39	'	47	71	G	67	103	g
08	8	BS	^H	28	40	(	48	72	H	68	104	h
09	9	HT	^I	29	41	)	49	73	I	69	105	i
0A	10	LF	^J	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	^K	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	^L	2C	44	,	4C	76	L	6C	108	l

[4] Două tetrade ale căror echivalente zecimale însumate dau 9 sunt complementare.
[5] Pentru a face distincţie între valoarea 0 şi absenţa unui semnal se exclude combinaţia 0000.

0D	13	CR	^M	2D	45	□	4D	77	M	6D	109	m
0E	14	SO	^N	2E	46	.	4E	78	N	6E	100	n
0F	15	SI	^O	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	^P	30	48	0	50	80	P	70	112	p
11	17	DC1	^Q	31	49	1	51	81	Q	71	113	q
12	18	DC2	^R	32	50	2	52	82	R	72	114	r
13	19	DC3	^S	33	51	3	53	83	S	73	115	s
14	20	DC4	^T	34	52	4	54	84	T	74	116	t
15	21	NAK	^U	35	53	5	55	85	U	75	117	u
16	22	SYN	^V	36	54	6	56	86	V	76	118	v
17	23	ETB	^W	37	55	7	57	87	W	77	119	w
18	24	CAN	^X	38	56	8	58	88	X	78	120	x
19	25	EM	^Y	39	57	9	59	89	Y	79	121	y
1A	26	SUB	^Z	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC		3B	59	;	5B	91	[	7B	123	{
1C	28	FS		3C	60	<	5C	92	\	7C	124	|
1D	29	GS		3D	61	=	5D	93	]	7D	125	}
1E	30	RS		3E	62	>	5E	94	^	7E	126	~
1F	31	US		3F	63	?	5F	95	_	7F	127	DEL

Există posibilitatea reprezentării a 2^7 coduri, asignarea acestora la caractere valide fiind evidenţiată în *Tabelul 2.8.*

Codurile 00h – 1Fh şi 7Fh nu sunt afişabile fiind caractere de control utilizate pentru transmisie, controlul afişării/tipăririi, cât şi pentru alte scopuri, semnificaţiile acestora fiind cuprinse în *Tabelul 2.9.*

Tabelul 2.9

CHR	Semnificaţie	CHR	Semnificaţie	CHR	Semnificaţie
NUL	**NUL**l	**FF**	**F**orm **F**eed	**CAN**	**CAN**cel
SOH	**S**tart **O**f **H**eading	**CR**	**C**arriage **R**eturn	**EM**	**E**nd of **M**edium
STX	**S**tart of **TeX**t	**SO**	**S**hift **O**ut	**SUB**	**SUB**stitution
ETX	**E**nd of **TeX**t	**SI**	**S**hift **I**n	**ESC**	**ESC**ape
EOT	**E**nd **O**f **T**ransmission	**DLE**	**D**ata **L**ink **E**scape	**FS**	**F**ile **S**eparator
ENQ	**ENQ**uiry	**DC1**	**D**evice **C**ontrol **1**	**GS**	**G**roup separator
ACK	**ACK**nowledge	**DC2**	**D**evice **C**ontrol **2**	**RS**	**R**ecord separator
BEL	**BEL**l	**DC3**	**D**evice **C**ontrol **3**	**US**	**U**nit separator
BS	**B**ack**S**pace	**DC4**	**D**evice **C**ontrol **4**	**SP**	**SP**ace
HT	**H**orizontal **T**abulation	**NAK**	**N**egative **AcK**nowledge	**DEL**	**DEL**ete
LF	**L**ine **F**eed	**SYN**	**SYN**chronous idle		

VT	**V**ertical **T**abulation	**ETB**	**E**nd of **T**ransmission **B**lock		

Toate celelalte caractere – *litere, cifre, semne de punctuaţie, spaţiul* – sunt reprezentabile. Tratarea cifrelor 0 – 9 este similară cu a literelor mici şi mari ceea ce simplifică manipularea caracterelor[6].

Codul EBCDIC (Extended Binary Coded Decimal InterChange) lansat de IBM, utilizează pentru reprezentarea caracterelor alfanumerice câte 8 biţi pe caracter, permiţând codificarea a 256 caractere.

2.3.3.3. Coduri detectoare şi corectoare de erori

În timpul transmisiei şi prelucrării datelor pot apare erori datorate, în primul rând, zgomotelor care afectează nivelurile fizice de tensiune specifice semnalelor digitale. Pentru detectarea şi eventual corectarea acestor erori se utilizează coduri cu proprietăţi speciale. Aceste coduri conţin pe lângă biţii utili o serie de biţi de redundanţi (de control) cu ajutorul cărora se identifică şi se corectează eventualele erori de transmisie.

Codurile detectoare de erori au următoarea proprietate: *apariţia unei singure erori transformă un cuvânt valid intr-un cuvânt invalid*. O metodă pentru detecţia erorilor este *metoda bitului de paritate* a cărui idee de baza constă în adăugarea unei cifre binare in plus la fiecare cuvânt al unui cod dat, pentru a face ca numărul de biţi de 1 din fiecare cuvânt să fie impar sau par.

Este util sa definim *distanta*[7] dintre 2 coduri cuvânt ca numărul de cifre care trebuie sa fie schimbate intr-un cuvânt pentru a rezulta alt cuvânt. (de exemplu distanţa Hamming a codului ASCII este **1).**

Un cod este cod detector de erori daca distanta sa minimă este mai mare sau egala cu **2**. De exemplu adăugarea unui bit de paritate plasat la stânga codului normal ASCII al unui caracter va determina pentru acest cod o distanţă Hamming egală cu 2. Acest bit se calculează ca sumă *modulo 2*[8]

[6] Pentru a transforma reprezentarea alfanumerică a unei cifre în valoarea sa numerică se scade 30h iar pentru a transforma codul unei majuscule în codul corespunzător al minusculei se adună 20h.

[7] Această distanţă este cunoscută ca *distanţa Hamming*.

[8] $0 \oplus 0 = 0\ 0 \oplus 1 = 1\ 1 \oplus 0 = 1\ 1 \oplus 1 = 0$

în cazul *parităţii pare* şi cu valoarea negată a acesteia în cadrul parităţii impare procedeu ilustrat în figura 2.5.

PI	PP	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Caracter
0	**1**	1	1	0	0	1	0	0	**d**
1	**0**	1	1	0	0	1	0	1	**e**

Biţi de paritate Codul ASCII

Fig. 2.5. Adăugarea bitului de paritate la codul ASCII: PI – paritate impară; PP – paritate pară.

Prin acest procedeu se va obţine o tabelă a codurilor ASCII cu 256 de caractere dintre care jumătate sunt invalide. La recepţie se recalculează bitul de paritate iar la neconcordanţă se va solicita retransmisia, care nu este întotdeauna o soluţie convenabilă.

În practică este larg utilizată şi metoda ***codului polinomial*** cunoscut şi sub denumirea de **cod cu redundanţă ciclică** sau cod *CRC* ***(Cyclic Redundancy Code)***. *CRC* sunt bazate pe tratarea şirurilor de biţi ca polinoame cu coeficienţi 0 şi 1. De exemplu biţii cuvântului cu 6 biţi 110001 se reprezintă prin polinomul

$$P(x) = 1 \bullet x^5 + 1 \bullet x^4 + 0 \bullet x^3 + 0 \bullet x^2 + 0 \bullet x^1 + 1 \bullet x^0 = x^5 + x^4 + 1$$

Aritmetica polinomială în acest caz este *aritmetica modulo 2,* potrivit regulilor teoriei algebrice. Nu există transport la adunare sau împrumut la scădere, adunările şi scăderile fiind identice, operându-se cu funcţia *SAU-EXCLUSIV*.

De exemplu

1 0 0 1 +	1 0 0 1 -
0 1 0 1	0 1 0 1
1 1 0 0	1 1 0 0

În cele ce urmează se va suînţelege că fiecare polinom este ataşat unui cadru de biţi , iar fiecare cadru de biţi are ataşat un polinom. Din acest motiv când se vorbeşte de operaţii aplicate se subînţelege *operaţii aplicate coeficienţilor acestora,* respectiv diverselor cadre de biţi.

La utilizarea acestei metode, pentru a transmite un cadru (formaţie) de biţi *Q* căruia i se ataşează un polinom *Q(x)* emiţătorul şi receptorul

convin asupra unui polinom generator *P(x)* a cărui lungime trebuie să fie mai mică decât a lui *Q(x)*.

La transmisie se procedează astfel:

1. se adaugă la *Q r* biţi (r este gradP) obţinându-se *cadrul îmbunătăţit QI;*
2. se împarte polinomul *CI(x),* ataşat cacadrul îmbunătăţit *C,* la P(x);
3. restul rezultat *R* (care are întotdeauna *r* sau mai puţini biţi) se scade *modulo 2* din cadrul îmbunătăţit.
4. cadrul *T* corespunzător polinomului diferenţa T(x) (divizibil modulo2 cu P(x)) este cel care se transmite.

La recepţie se efectuează împărţirea T(x):P(x), şi dacă rezultă rest nul, rezultă că transmisia s-a efectuat corect. Refacerea lui Q(x) în caz că nu sunt erori presupune separarea în T(x) a ***r cifre*** din zona nesemnificativă.

Exemplul 2.11

Fie cadrul de biţi **C=1101101** sau **Q(x)=1***x^6+**1*** x^5+ **1***x^3+ **1***x^2+1

Polinomul generator este **P(x)** = $\mathbf{x^3+ x+1}$**cu** cadrul **P = 1011**

Să se determine cadrul transmis T (cu polinomul asociatT(x)), din care să se refacă apoi Q(x) respectiv cadrul de biţi iniţial C.

1. Se adaugă 3 biţi (gradul lui P(x) este 3) la cadrul iniţial obţinându-se cadrul îmbunătăţit

CI = 1101101*000*

2. Se împarte cadrul îmbunătăţit la P(x) (CI:P)

```
1 1 0 1 1 0 1 0 0 0 | 1 0 1 1
1 0 1 1             | 1 1 1 1 1 0 1
- 1 1 0 1
  1 0 1 1
  - 1 1 0 0      c â t = 1 1 1 1 1 0 1
    1 0 1 1
    - 1 1 1 1    r e s t = 0 1 1 1
      1 0 1 1
      - 1 0 0 0
        1 0 1 1
```

```
- - 1 1 0 0
    1 0 1 1
  ---------
  0 1 1 1
```

3. Se scade restul R din cadrul îmbunătăţit CI şi rezultă polinomul care se transmite, respectiv T(x).

```
1 1 0 1 1 0 1 0 0 0 -
0 0 0 0 0 0 0 1 1 1
-------------------
1 1 0 1 1 0 1 1 1 1
```

T = 11011011111

4. Verificarea la recepţie, presupune efectuarea împărţirii T:P, respectiv T(x):P(x)

```
1 1 0 1 1 0 1 1 1 1 | 1 0 1 1
1 0 1 1             | -------
- 1 1 0 1           | 1 1 1 1 1 0 1
  1 0 1 1
  - 1 1 0 0      cât = 1 1 1 1 1 0 1
    1 0 1 1
    - 1 1 1 1  rest = 0 0 0 0
      1 0 1 1
      - 1 0 0 1
        1 0 1 1
        - - 1 0 1 1
            1 0 1 1
            -------
            0 0 0 0
```

5. Restul fiind **0000** rezultă că transmisia s-a efectuat corect, iar refacerea cadrului iniţial eliminând ultimele 3 cifre din T(x), respectiv

C = **1101101 sau** $Q(x)=\mathbf{1}*x^6+\mathbf{1}*x^5+\mathbf{1}*x^3+\mathbf{1}*x^2+1$

La transmiterea datelor se folosesc, printre altele, următoarele trei polinoame generatoare (Standard IEEE 802-3):

CRC - 8: $P(x) = x^8 + x^7 + x^6 + x^4 + x^2 + 1$

CRC-16 IBM: $P(x) = x^{16} + x^{15} + x^2 + 1$

CRC - 32: $P(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

După cum s-a menţionat, la recepţia unui mesaj eronat se cere retransmisia acestuia. Pentru a evita retransmisia se apelează la *codurile corectoare de erori* .

Un cod este **cod corector de erori** *(CCE)* atunci când cuvântul de cod corect poate fi întotdeauna dedus din cuvântul eronat. Între *CCE* un loc aparte revine *codului Hamming*, care în cazul unei corecţii singulare are distanţa **3.** Numărul minim de biţi de control necesari pentru a asigura această distanţă se determină cu relaţia

$$2^k \geq m + k + 1, \qquad (2.14)$$

unde m este numărul de biţi de informaţie, iar k - numărul de biti de control.

Vom explica în continuare construcţia unui cod Hamming cu $m = 7$ (mesaj original in ASCII). Din relatia lui Hamming (2.14) rezulta $k = 4$, astfel încât 4 biţi de control trebuie adăugaţi celor 7 biţi de informaţie, rezultând un cod ASCII pe 11 biţi cu 4 biţi redundanţi.

Asignarea celor 4 biţi redundanţi la cuvintele originale se va face astfel încât să poată fi identificată *o eroare la un singur bit*. Fiecărui bit din cuvântul codificat, incluzând şi biţii redundanţi de verificare îi este asignată combinaţia de biţi $C_8C_4C_2C_1$, corespunzător *Tabelului 2.10*. În acest tabel echivalentul zecimal al combinaţiilor binare reprezintă poziţia biţilor care sunt verificaţi începând cu poziţia 1. În cuvântul de 11 biţi care se transmite C_1 ocupă poziţia **1**, C_2 poziţia **2** a doua ş.a.m.d. , conform reprezentării din figura 2.6.

Tabelul 2.10

C_8	C_4	C_2	C_1	**Poziţie bit verificat**
D0	0	0	1	**1**
0	0	1	0	**2**
0	0	1	1	**3**
0	1	0	0	**4**
0	1	0	1	**5**
0	1	1	0	**6**
0	1	1	1	**7**

1	0	0	0	**8**
1	0	0	1	**9**
1	0	1	0	**10**
1	0	1	1	**11**

11	10	9	8	7	6	5	4	3	2	1
B_7	B_6	B_5	**C_8**	B_4	B_3	B_2	**C_4**	B_1	**C_2**	**C_1**

Fig. 2.6. Structura unui cuvânt ASCII cu patru biţi de verificare:
B_i – biţi utili; C_j - biţi de verificare.

Modul de amplasare a biţilor de verificare în poziţii corespunzând puterilor lui 2, se numeşte *Cod de Corectare a unei Erori Singulare (CCES) sau Single Error Correcting (SEC),* şi facilitează procesul de localizare a erorii.

Valorile celor 4 biţi de verificare sunt determinaţi funcţie de paritatea (pară sau impară avută în vedere) prin evaluarea sumelor *modulo 2* S_1, S_1, S_4, S_8, conform relaţiilor de mai jos:

$$S_1 = C_1 \oplus B_1 \oplus B_2 \oplus B_4 \oplus B_5 \oplus B_7 \quad (2.15)$$

$$S_2 = C_2 \oplus B_1 \oplus B_3 \oplus B_4 \oplus B_6 \oplus B_7 \quad (2.16)$$

$$S_4 = C_4 \oplus B_2 \oplus B_3 \oplus B_4 \quad (2.17)$$

$$S_8 = C_8 \oplus B_5 \oplus B_6 \oplus B_7 \quad (2.18)$$

Sumele S_i trebuie să realizeze paritatea pară pentru poziţiile care au **1** în coloanele aferente biţilor de verificare C_i după cum urmează:

$S_1 = 0$ pentru grupul de biţi {1,3,5,7,9,11} ;
$S_2 = 0$ pentru grupul de biţi {2,3,6,7,10,11} ;
$S_4 = 0$ pentru grupul de biţi {4,5,6,7} ;
$S_8 = 0$ pentru grupul de biţi {8,9,10,11} .

Din relaţiile (2.15)…(2.18) rezultă (impunând tipul paritatăţii) valorile care se vor transmite pentru biţii de control C_1, C_2, C_4, C_8. La recepţie se evaluează sumele, iar când acestea corespund tipului parităţii transmise, transmisia se consideră corectă. În cazul când una sau mai multe

sume nu corespund tipului de paritate, există un bit eronat a cărui poziţie se determină prin adunarea indicilor sumelor eronate[9].

În cele ce urmează se va considera un exemplu pentru care se va presupune paritate pară (suma 0 la un număr par de biţi 1).

1. Să se determine biţii reduntanţi de verificare la transmisia codului ASCII al caracterului ***a*** (97_{10} = 1100001_2 sau 61_{16} = 1100001_2)

11	10	9	8	7	6	5	4	3	2	1
B_7	B_6	B_5	**C_8**	B_4	B_3	B_2	**C_4**	B_1	**C_2**	**C_1**
1	1	0	**?**	0	0	0	**?**	1	**?**	**?**

$$S_1 = C_1 \oplus B_1 \oplus B_2 \oplus B_4 \oplus B_5 \oplus B_7 = 0 \rightarrow C_1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 = 0 \rightarrow C_1 = 0$$
$$S_2 = C_2 \oplus B_1 \oplus B_3 \oplus B_4 \oplus B_6 \oplus B_7 = 0 \rightarrow C_2 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 = 0 \rightarrow C_2 = 1$$
$$S_4 = C_4 \oplus B_2 \oplus B_3 \oplus B_4 = 0 \rightarrow C_4 \oplus 0 \oplus 0 \oplus 0 = 0 \rightarrow C_4 = 0$$
$$S_8 = C_8 \oplus B_5 \oplus B_6 \oplus B_7 = 0 \rightarrow C_8 \oplus 0 \oplus 1 \oplus 1 = 0 \rightarrow C_8 = 0$$

astfel încât se transmite următoarea formaţie de 11 biţi

11	10	9	8	7	6	5	4	3	2	1
B_7	B_6	B_5	**C_8**	B_4	B_3	B_2	**C_4**	B_1	**C_2**	**C_1**
1	1	0	**0**	0	0	0	**0**	1	**1**	**0**

2. Să se determine ce caracter a fost transmis dacă a fost recepţionată următoarea formaţie de 11 biţi **10010111001** (7 biţi ai codului ASCII şi 4 biţi de verificare

11	10	9	8	7	6	5	4	3	2	1
B_7	B_6	B_5	**C_8**	B_4	B_3	B_2	**C_4**	B_1	**C_2**	**C_1**
1	0	0	**1**	0	1	1	**1**	0	**0**	**1**

[9] Se are în vedere faptul că un bit n este verificat de către biţii ale căror poziţii însumate dau n (de exemplu bitul 7 este verificat de către biţii din poziţiile 1, 2, 4 deoarece 1+2+4=7.

Se evaluează sumele *Si*

$$S_1 = C_1 \oplus B_1 \oplus B_2 \oplus B_4 \oplus B_5 \oplus B_7 = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 1 \rightarrow \textit{impară}$$
$$S_2 = C_2 \oplus B_1 \oplus B_3 \oplus B_4 \oplus B_6 \oplus B_7 = 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 0 \rightarrow \textit{pară}$$
$$S_4 = C_4 \oplus B_2 \oplus B_3 \oplus B_4 = 1 \oplus 1 \oplus 1 \oplus 0 = 1 \rightarrow C_4 \rightarrow \textit{impară}$$
$$S_8 = C_8 \oplus B_5 \oplus B_6 \oplus B_7 = 1 \oplus 0 \oplus 0 \oplus 1 = 0 \rightarrow \textit{pară}$$

de unde rezultă că S_1 şi S_4 sunt impare, un bit a fost transmis eronat a cărui poziţie este **1+4=5** deci bitul numărul 5 trebuie corectat astfel încât formaţia de 11 biţi devine

11	10	9	8	7	6	5	4	3	2	1
B_7	B_6	B_5	**C_8**	B_4	B_3	B_2	**C_4**	B_1	**C_2**	**C_1**
1	0	0	**1**	0	1	0	**1**	0	**0**	**1**

Se extrag biţii de control din poziţiile 1, 2, 4, 8 astfel încât rezultă

11	10	9	8	7	6	5	4	3	2	1
B_7	B_6	B_5	**C_8**	B_4	B_3	B_2	**C_4**	B_1	**C_2**	**C_1**
1	0	0	-	0	1	0	-	0	-	-

Respectiv

7	6	5	4	3	2	1
B_7	B_6	B_5	B_4	B_3	B_2	B_1
1	0	0	0	1	0	0

Vectorul binar este $(1000100)_2 = (44)_{16} = (68)_{10}$ care este codul ASCII al caracterului **D**.

Observaţie

În cazul apariţiei a două erori acestea pot fi detectate, însă numai una poate fi corectată. Pentru a corecta 2 erori distanţa Hamming minimă trebuie să fie 5. În general pentru a detecta **p** erori distanţa Hamming trebuie să fie **p+1** , iar pentru corectarea a **p** erori această distanţă trebuie să fie **2p+1.**

2.4. Operaţii aritmetice în virgulă fixă

2.4.1. Adunarea şi scăderea binară

Cele două operaţii vor fi tratate pentru reprezentarea în codul *direct,* în mod similar putând fi tratate şi operaţiile în codurile *invers* şi *complementar.*

Cele două operaţii vor fi tratate unitar conform relaţiei:

$$op\_ef = s_x \oplus s_y \oplus s_{op} \qquad (2.19)$$

unde :

op_ef reprezintă operaţia efectivă ce se va efectua între cei doi operanzi;

s_x, s_y - semnele celor doi operanzi (*1* pentru -, *0* pentru +);

s_{OP} - operaţia ce se doreşte a fi efectuată (*1* pentru -, *0* pentru +).

În ceea ce priveşte operaţia "⊕" (*sau exclusiv*) după cum s-a mai arătat aceasta este definită astfel:

$0 \oplus 0 = 0$;
$0 \oplus 1 = 1$;
$1 \oplus 0 = 1$;
$1 \oplus 1 = 0$.

Pe baza celor prezentate, valoarea operatorului *op_ef* se determină cu ajutorul *Tabelului 2.11.*

Tabelul 2.11

x_s	y_s	S_{op}	Op	x_s	y_s	S_{op}	Op
0	0	0	0	+	+	+	+
0	0	1	1	+	+	-	-
0	1	0	1	+	-	+	-
0	1	1	0	+	-	-	+
1	0	0	1	-	+	+	-
1	0	1	0	-	+	-	+
1	1	0	0	-	-	+	+
1	1	1	1	-	-	-	-

În cazul în care prin evaluarea semnelor rezultă *scădere,* scăzătorul va putea fi înlocuit prin complementul său faţă de 2 (C2), operaţia fiind

practic înlocuită cu o operaţie de adunare[10], în *Tabelul 2.12* fiind prezentate cele 4 situaţii posibile.

Tabelul 2.12

Situaţii posibile	$(+A)+(+B)$	$(+A)+(-B)$	$(-A)+(+B)$	$(-A)+(-B)$
Operaţia de executat	$A+B$	$A+\overline{B}$	$\overline{A}+B$	$A+B$

Adunarea numerelor se execută asupra cifrelor numerelor *A* şi *B* în conformitate cu regulile adunării binare. Semnul sumei *C* precum corectitudinea rezultatului obţinut se stabilesc în funcţie de absenţa (*Tabelul 2.13*) sau nu a (*Tabelul 2.14*) transportului de la *CCMS*[11] la bitul de semn.

Tabelul 2.13

Semnul rezultatului *Cs*	$(+A)+(+B)$ +(0)	$(+A)+(-B)$ -(1)	$(-A)+(+B)$ -(1)	$(-A)+(-B)$ -(1)
Semnificaţia sumei	Adevărata mărime *C*	Complementul C2	Complementul C2	Adevărata mărime *C*
Corecţie	Nu (rezultat corect)	Recomplementare (rC2[12])	Recomplementare (rC2)	Nu (rezultat corect)

Tabelul 2.14

Semnul rezultatului *Cs*	$(+A)+(+B)$ +(0)	$(+A)+(-B)$ +(0)	$(-A)+(+B)$ +(0)	$(-A)+(-B)$ -(1)
Semnificaţia sumei	Adevărata mărime *C*	Adevărata mărime *C*	Adevărata mărime *C*	Adevărata mărime *C*
Corecţie	Eroare de depăşire	Nu (rezultat corect)	Nu (rezultat corect)	Eroare de depăşire

[10] Acest lucru este posibil deoarece $A-B=A+(0-B)=A+\overline{B}$

[11] *CCMS –**C**ifra **C**ea **M**ai **S**emnificativă*

[12] *rC2* – recomplementare, operaţiile se fac invers celor de la obţinerea *C2* (se scade *1*, după care complementează).

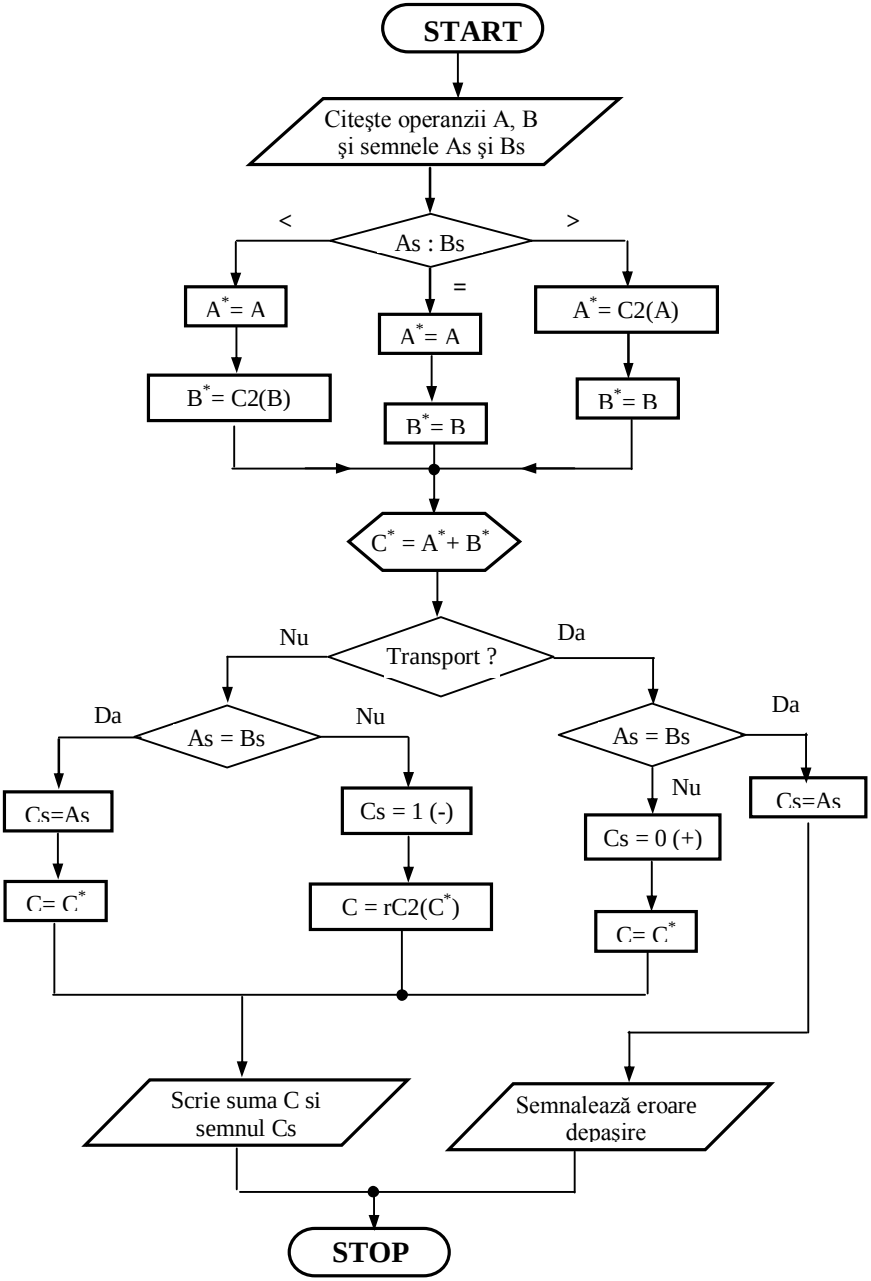


Fig. 2.7. Schema logică a adunării în cod direct.

Algoritmul descris în *Tabelele 2.13 şi 2.14* poate fi transpus în schema logică simplificată din figura 2.7.

Exemplul 2.12.

Fie $A=(10/16)_{10}= (0.625)_{10}$ şi $B=(7/16)_{10}= (0.4375)_{10}$ reprezentate în binar pe cinci biţi (patru pentru valoare, unul pentru semn) prin

$A=(\mathbf{0}1010)_2$ şi $B=(\mathbf{0}0111)_2$

Să se calculeze $C_1=A+B$, $C_2=A-B$, $C_3=-A+B$, $C_4=-A-B$

$C_1^* = A + B$

0	1	0	1	0	+
0	0	1	1	1	
0	0	0	0	1	
1					

S-a generat transport de la CCMS la bitul de semn – numerele A şi B fiind de acelaşi semn rezultatul este **incorect** deoarece se depăşeşte capacitatea de reprezentare pe 4 biţi - într-adevăr $(0.625)_{10}+ (0.4375)_{10}= (\mathbf{1}.0625)_{10}$ – obţinându-se un număr supraunitar.

$C_2^* = A - B =$
$= A + C2(B)$

0	1	0	1	0	+
1	1	0	0	1	
	0	0	1	1	
1					

$C_2 = C_2^*$ ***0*** 0 0 1 1

S-a generat transport de la CCMS la bitul de semn – numerele A şi B fiind de semne contrare rezultatul este **corect** încadrându-se în capacitatea de reprezentare pe 4 biţi - într-adevăr $(0.625)_{10}$ - $(0.4375)_{10}= (\mathbf{0}.1675)_{10}$ – obţinându-se un număr subunitar respectiv 1/8 + 1/16 = 3/16

$C_3^* = -A + B =$
$= C2(A) + B$

1	0	1	1	0	+
0	0	1	1	1	
1	1	1	0	1	

$C_3 = rC2(C_2^*)$ ***1*** 0 0 1 1

Nu s-a generat transport de la CCMS la bitul de semn – rezultatul pe 4 biţi obţinându-se prin recomplementare -(1/8+1/16)=-3/16.

$C_4^* = -A - B$

1	1	0	1	0	+
1	0	1	1	1	
	0	0	0	1	
1 ←					

S-a generat transport de la CCMS la bitul de semn – numerele A şi B fiind de acelaşi semn rezultatul este **incorect** deoarece se depăşeşte capacitatea de reprezentare pe 4 biţi - într-adevăr $-(0.625)_{10} - (0.4375)_{10} = -(\mathbf{1}.0625)_{10}$ – obţinându-se un număr supraunitar în valoare absolută.

2.4.1. Înmulţirea binară

Operaţia de înmulţire a numerelor constă în generarea şi adunarea *produselor parţiale* obţinute prin înmulţirea rangului curent al înmulţitorului cu deînmulţitul.

În cazul numerelor binare produsul parţial poate fi:

- chiar deînmulţitul deplasat spre stânga conform poziţiei rangului împărţitorului, dacă bitul asociat acestui rang este **1**;
- 0 dacă bitul asociat rangului curent al împărţitorului este **0.**

Dacă produsul parţial este diferit de zero, acesta se va aduna la suma produselor parţiale anterioare.

În continuare vor fi tratată înmulţirea numerelor exprimate în cod direct, înmulţirea cu puteri ale lui **2** şi înmulţirea cu mai multe cifre..

2.4.1.1. Înmulţirea în cod direct

Fie operanzii exprimaţi în cod direct:

$X = x_{n-1}\, x_{n-2} \ldots\ldots\ldots x_2\, x_1\, x_0$

$Y = y_{n-1}\, y_{n-2} \ldots\ldots\ldots y_2\, y_1\, y_0$

prin căror înmulţire se va obţine produsul

$Z = z_{2n-2}\, z_{2n-3} \ldots\ldots\ldots z_2\, z_1\, z_0$

Semnul produsului se determină ca sumă modulo 2 a reprezentărilor pentru semnele operanzilor

$$z_{2n-2} = x_{n-1} \oplus y_{n-1} \tag{2.20}$$

unde z_{2n-2} , x_{n-1} şi y_{n-1} , sunt semnele produsului şi ale celor doi factori. Prin aplicarea relaţiei (2.20) se obţin cazurile particulare evidenţiate mai jos.

$$\begin{aligned} &0 \oplus 0 = 0 \quad ; \quad (+)\cdot(+) = + \\ &0 \oplus 1 = 1 \quad ; \quad (+)\cdot(-) = - \\ &1 \oplus 0 = 1 \quad ; \quad (-)\cdot(+) = - \\ &1 \oplus 1 = 0 \quad ; \quad (-)\cdot(-) = + \end{aligned} \tag{2.21}$$

Produsele parţiale $P_0, P_1, \ldots, P_{n-3}, P_{n-2}$ se determină după cum urmează:

$$\begin{aligned} P_0 &= |x| \cdot y_0 \cdot 2^0 \\ P_1 &= |x| \cdot y_1 \cdot 2^1 \\ &\ldots\ldots\ldots\ldots \\ P_{n-2} &= |x| \cdot y_{n-2} \cdot 2^{n-2} \end{aligned} \tag{2.22}$$

iar suma lor va conduce la produsul modulelor respectiv

$$|Z| = z_{2n-3} \ldots\ldots\ldots z_2\, z_1\, z_0$$

Dacă este necesară aducerea numărului la lungime simplă, aceasta se poate realiza prin *trunchiere şi rotunjire* .

Exemplul 2.13.

1. Se dau numerele X = -27, Y = 23 şi se cere să se reprezinte în codul *semn-mărime*, iar apoi să se efectueze produsul Z = X • Y, utilizând metoda adunărilor repetate.

$(X)_2 = 1\ 11011$
$(Y)_2 = 0\ 10111$
Semnele celor două numere sunt $x_5 = 1$, $y_5 = 0$ de unde rezultă
$z_{10} = x_5 \oplus y_5 = 1 \oplus 0 = 1$ (negativ)

Având în vedere că n = 6 poziţii binare, inclusiv bitul de semn rezultă că modulul va avea 10 poziţii)

Se înmulţesc cele două module:

11011 x 10111		deînmulţit înmulţitor
0000**11011**	$\|x\| \cdot 2^0$	(x→stânga 0 poziţii)
0000**11011**0	$\|x\| \cdot 2^1$	(x→stânga 1 poziţie)
000**1101100**	$\|x\| \cdot 2^2$	(x→ stânga 2 poziţii)
0000000000	0 pentru că $y_3 = 0$	
0**1101**10000	$\|x\| \cdot 2^4$	(x→stânga 4 poziţii)
1001101101		produs

Produsul reprezintă suma calculată simultan pentru toate cele cinci produse parţiale. Având în vedere că în calculator se execută la un moment dat numai suma a două numere, înmulţirea se poate efectua conform etapizării prezentate mai jos.

11011x 10111	deînmulţit înmulţitor
0000000000	suma iniţială
11011	primul produs parţial
0000011011	prima sumă
11011	al doilea produs parţial
0001010001	a doua sumă
11011	al treilea produs parţial
0010111101	a treia sumă
00000	al patrulea produs parţial
0010111101	a patra sumă
11011	al cincilea produs parţial
1001101101	produs

Prin ambele metode rezultă $|Z|$ = **1001101101** respectiv

$Z = 1$ **1001101101**.

Verificare: $x \cdot y = -621$.

$p = -(1 \cdot 2^9 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1) =$

$= -(512 + 64 + 32 + 8 + 4 + 1) = -621.$

2. Se dau numerele X = 0.8125, Y = -0.6875 şi se cere să se reprezinte în *semn-mărime*, iar apoi să se efectueze produsul Z = X • Y, utilizând metoda adunărilor repetate.

$(X)_2 = 0$ **0.1101**
$(Y)_2 = 1$ **0.1011**

Semnele celor două numere sunt $x_4 = 0$, $y_4 = 1$ de unde rezultă $z_8 = x_4 \oplus y_4 = 0 \oplus 1 = 1$ (negativ)

Se înmulţesc apoi cele două module.

.1101 x		deînmulţit
.1011		înmulţitor
0000**1101**	$\|x\| \cdot 2^0$	(x→stânga 0 poziţii)
000**1101**0	$\|x\| \cdot 2^1$	(x→stânga 1 poziţie)
00000000	0 pentru că $y_2 = 0$	
0**1101**000	$\|x\| \cdot 2^3$	(x→stânga 3 poziţii)
0.10001111		produs

Alt mod (clasic)

```
    1101 x
    1011
   ------
    1101
   1101
  1101
 --------
0.10001111
```

Verificare: 0.8125 • (-0.6875) = -0.5585937

$$1\ \mathbf{0.10001111} = -(1 \cdot 2^{-1} + 1 \cdot 2^{-5} + 1 \cdot 2^{-6} + 1 \cdot 2^{-7} + 1 \cdot 2^{-8}) =$$

$$= -(0.5 + 0.03125 + 0.015625 + 7.8125 \cdot 10^{-3} + 3.90625 \cdot 10^{-3}) = -0.5585937$$

În acest caz se poate face trunchiere şi rotunjire. Pentru exprimarea pe 5 biţi (din care unul pentru semn) se îndepărtează ultimii 4 biţi. Dacă cel mai semnificativ bit îndepărtat este 1 rezultatul se rotunjeşte (se adaugă 1).

p = 1.1000|1111 MSb p ≅ 1.1000 +
1
1.1001

Verificare $Z = -(1 \cdot 2^{-1} + 1 \cdot 2^{-4}) = -(0.5 + 0.0625) = -0.5625$.

Eroarea relativă de rotunjire va fi

$$e_r = \frac{|0.5585937 - 0.5625|}{0.5585937} \cdot 100 = 0.7\%$$

2.4.2.2. Înmulţirea cu puteri ale lui 2

O categorie specială de înmulţire o reprezintă înmulţirea cu puteri ale bazei 2 respectiv cu 2^k, care presupune deplasări după cum urmează:

k > 0 → deplasare *stânga* cu **k** poziţii (se adaugă zerouri în poziţiile nesemnificative din dreapta rămase libere);

k < 0 → deplasare *dreapta* cu **k** poziţii (se adaugă zerouri în poziţiile semnificative rămase libere).

Exemplul 2.14

1. $28 \bullet 2^2 = 112$

7	6	5	4	3	2	1	0
0	**0**	**0**	**1**	**1**	**1**	**0**	**0**

Stânga 2 poziţii ←

		7	6	5	4	3	2	1	0
0	0	**0**	**1**	**1**	**1**	**0**	**0**	***0***	***0***

Zerouri adăugate

Verificare $1 \bullet 2^6 + 1 \bullet 2^5 + 1 \bullet 2^4 = 64 + 32 + 16 = 112$

2. $28 / 2^2 = 28 \bullet 2^{-2} = 7$

7	6	5	4	3	2	1	0
0	**0**	**0**	**1**	**1**	**1**	**0**	**0**

Dreapta 2 poziţii →

7	6	5	4	3	2	1	0		
0	***0***	**0**	**1**	**1**	**1**	**0**	**0**	0	0

Zerouri adăugate

Verificare $1 \bullet 2^2 + 1 \bullet 2^1 + 1 \bullet 2^0 = 4 + 2 + 1 = 7$

2.4.2.3. Înmulţirea cu mai multe cifre

Timpul afectat realizării unei operaţii de înmulţire poate fi redus prin micşorarea numărului de paşi. Una din metodele utilizate în acest sens este efectuarea înmulţirii cu mai multe cifre simultan.

Presupunem că avem de efectuat produsul $\boldsymbol{P = A \bullet B}$ şi considerăm grupe de câte ***k*** cifre binare consecutive ale înmulţitorului ***B.*** Aceste cifre se pot găsi în $\boldsymbol{2^k}$ combinaţii evidenţiate în Tabelul 2.15. Funcţie de valorile combinaţiilor şi de poziţia grupei, înmulţirea cu ***k*** cifre presupune adunarea

deînmulţitului ***A*** sau a unui multiplu al său aşa cum rezultă din *Tabelul 2.15.*

Tabelul 2.15

Nr. comb.	Combinaţie	Operaţie executată
1	00…000	-
2	00…001	Se adună ***A***
3	00…010	Se adună 2***A***
⋮	⋮	⋮
2^k	11…111	Se adună $(2^k-1)A$

Valorile ***A***, 2***A***, $(2^k-1)A$ se adună deplasate spre stânga începând cu CCMPS[13] a grupei. Este evident că prin această procedură numărul de paşi în care se efectuează înmulţirea se reduce de ***k*** ori, timpul reducându-se corespunzător.

Exempluul 2.15

Fie numerele A=00001011 şi B=00001101. Să se efectueze produsul $\boldsymbol{P = A \bullet B}$ aplicând înmulţirea cu două cifre simultan.

Pentru *două cifre Tabelul 2.15* devine.

Tabelul 2.16

Nr.	Comb.	Operaţie executată
1	00	-
2	01	Se adună ***A***
3	10	Se adună 2***A***
4	11	Se adună ***3A***

A = 00001011 2A = 00010110 3A = 00100001

A					1	0	1	1	x
B					1	1	0	1	
					1	0	1	1	Prima grupă 01
	1	0	0	0	0	1			A doua grupă 11
P	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	

[13] CCMPS – Cifra Cea Mai Puţin Semnificativă

Verificare $A = 1 \cdot 2^3 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 2 + 1 = 11$

$B = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^0 = 8 + 4 + 1 = 13$

$P = 1 \cdot 2^7 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 =$

$= 128 + 8 + 4 + 2 + 1 + = 143 = 11 \cdot 13$

2.4.3 Împărţirea binară

Dacă operaţia de înmulţire consta practic într-o adunare repetată, cea de împărţire se poate realiza prin scăderea repetată a împărţitorului. Sunt însă şi metode speciale din care în continuare va fi prezentat cea *cu comparare,* alte metode importante fiind cele *cu restaurare şi fără restaurare.*

Împărţirea binară prin metoda comparării. Dacă se consideră două numere *A (deîmpărţit)* şi *B (împărţitor),* pentru ca împărţirea să se poată efectua trebuie respectate următoarele două condiţii:

a- *împărţitorul B* să fie diferit de ***0 (zero);***

b- $B > A$ (câtul va rezulta în acest fel subunitar)[14].

Efectuarea împărţirii ***A:B*** presupune determinarea a două numere ***Q (cîtul)*** şi $\boldsymbol{R_n}$ *(al n-lea rest)* astfel încât să fie satisfăcută relaţia

$$A = B \cdot Q + R_n \qquad (2.23)$$

Conform acestei metode cifrele câtului se determină prin *comparări* succesive ale împărţitorului ***B*** cu resturile parţiale $\boldsymbol{R_i}$, în urma comparării putând apărea următoarele situaţii:

a- $\boldsymbol{B < R_i} \rightarrow$ cifra corespunzătoare a câtului este ***1*** şi se execută o *scădere a împărţitorului din restul parţial* (R_i *- B)*;

b- $\boldsymbol{B > R_i} \rightarrow$ cifra corespunzătoare a câtului este ***0*** şi nu se execută *scăderea împărţitorului din restul parţial*;

Considerând lungimile deîmpărţitului ***A*** şi împărţitorului ***B*** ca fiind de fiind de ***2n*** respectiv ***n*** biţi, câtul *Q* va rezulta cu lungimea de *n* biţi care

[14] Dacă aceasta condiţie nu este îndeplinită se va transforma împărţitorul, iar după efectuarea împărţirii se va ajusta corespunzător câtul.

se vor determina în ***n*** paşi conform procedurii prezentate în cele ce urmează.

Cifra de rang ***i*** se va calcula cu relaţia de recurenţă de mai jos, care se va aplica de n ori, conform schemei logice din figura 2.9.

$$R_i = 2 \cdot R_{i-1} - q_i \cdot B \,.^{15} \qquad (2.24)$$

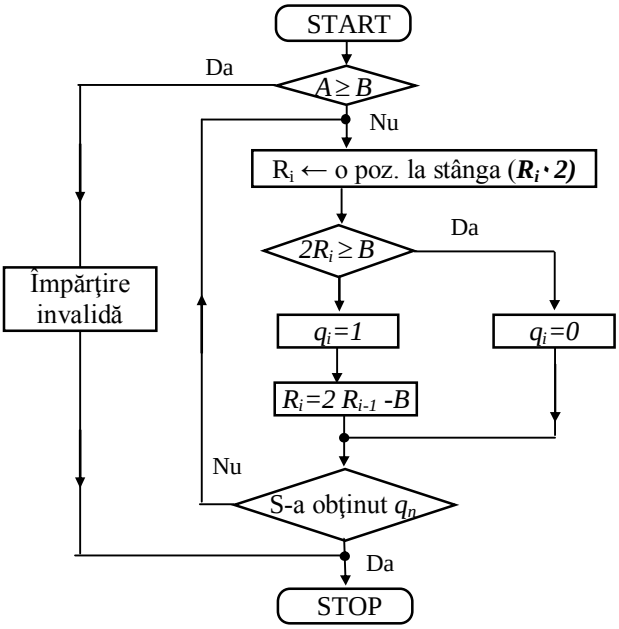


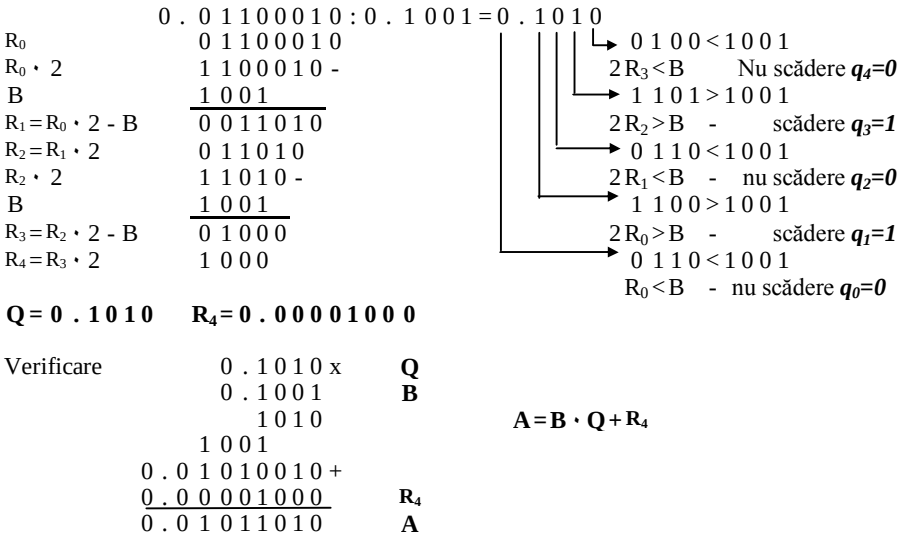
Fig. 2.8. Algoritmul de împărţire prin comparare.

Procesul iterativ se opreşte atunci când o nouă deplasare spre stânga a restului ar determina valoarea zero, sau după un număr fixat de paşi

[15] Primul rest parţial $R_0 = A$ se compară cu ***B.*** Dacă $B < R_0$ se scade ***B*** din R_0 iar cifra câtului este ***1;*** în caz contrar scăderea nu se execută iar cifra câtului este *0*. În ambele situaţii rezultatul obţinut se deplasează cu o poziţie la stânga devenind restul parţial R_1.

Exemplul 2.16

Să se efectueze împărţirea *A:B* unde $A = 0.01100010$ şi $B = 0.1001$.



Verificare

$$\begin{array}{rl} 0.1010 \times & \mathbf{Q} \\ 0.1001 & \mathbf{B} \\ \hline 1010 & \\ 1001 & \\ \hline 0.01010010 + & \\ 0.00001000 & \mathbf{R_4} \\ \hline 0.01011010 & \mathbf{A} \end{array} \qquad \mathbf{A = B \cdot Q + R_4}$$

Împărţirea s-a oprit după 5 paşi deoarece prin deplasarea lui R_4 la stânga s-ar obţine 0000B.

2.5. Operaţii aritmetice în virgulă mobilă

2.5.1. Adunarea şi scăderea numerelor reprezentate în virgulă mobilă

Aceste operaţii se execută în mai multe etape şi anume:

a) se compară caracteristicile celor două numere şi la necoincidenţă se aduc numerele la aceiaşi caracteristică - cea mai mare (în acest scop numărul cu caracteristica mai mică va fi deplasat cu $\boldsymbol{D = C_{max} - C_{min}}$ poziţii la dreapta);

b) se adună (scad) mantisele în codul în care sunt reprezentate;

c) se normalizează mantisa obţinută în urma adunării.

Fie două numere *A* şi *B* reprezentateîn VM (baza *b*)

$$A = (\pm M_1)_b \cdot b^{C_1} \ , \ B = (\pm M_2)_b \cdot b^{C_2}$$

unde M_1 şi M_2 sunt mantisele iar C_1 şi C_2 (unde $C_1 > C_2$) sunt caracteristicile celor două numere.

Suma, respectiv diferenţa celor două numere se calculează astfel:

$$A \pm B = ((\pm M_1)_b \pm (\pm M_2)_b \cdot b^{-D}) \cdot b^{C_1} \qquad (2.25)$$

potrivit schemei logice din figura 2.12.

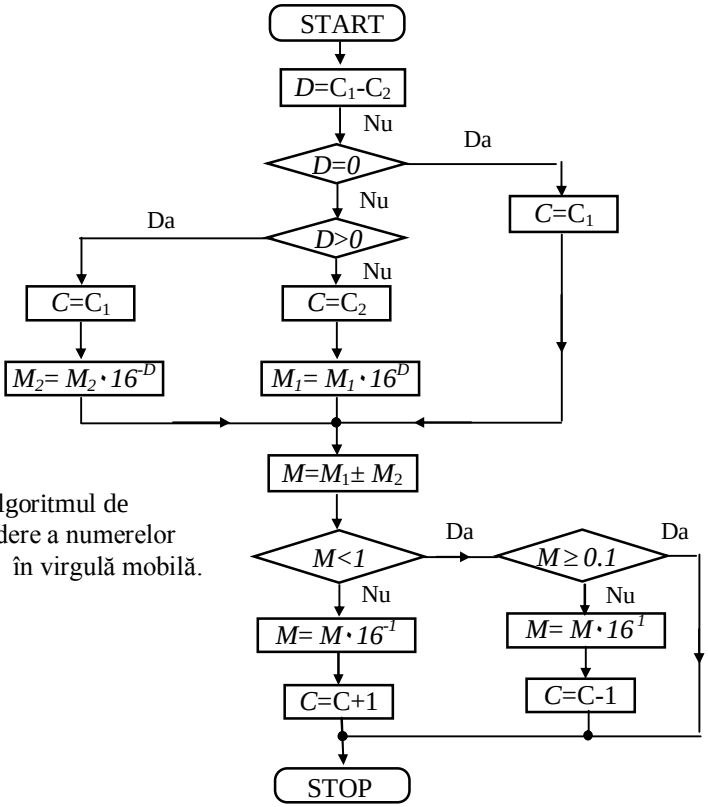


Fig. 2.12. Algoritmul de adunare/scădere a numerelor reprezentate în virgulă mobilă.

Se observă că dacă $M>1$ sau $M<0.1$ se impune normalizarea mantisei.

Exemplul 2.17.

Fie $A = 0.1F7 \cdot 16^3$ şi $B = 0.B54 \cdot 16^{-1}$ (baza $b=16$). Să se calculeze $F_1=A+B$ şi $F_2=A-B$.

Pentru datele din exemplu $C_1 = 3$, $C_2 = -1$ şi $D = C_1 - C_2 = 4$ şi $A = 0.1F8 \cdot 16^3$ şi B = 0.0000B54 • 16^3

Rezultă:

$$F_1 = A + B = (0.1F7+0.B54 \cdot 16^{-4}) \cdot 16^3 =$$
$$= (0.1F7 + 0.0000B54) \cdot 16^3 = 0.1F70B54 \cdot 16^3$$

şi

$$F2 = A - B = (0.1F7 - 0.0000B52) \cdot 16^3 =$$
$$= (0.1F7 + 0.0000B54) \cdot 16^3 = 0.1F6F4AC \cdot 16^3$$

2.5.2. Înmulţirea numerelor reprezentate în virgulă mobilă

Înmulţirea a două numere reprezentate în virgulă mobilă, presupune *adunarea* caracteristicilor şi *înmulţirea* mantiselor.

Pentru două numere *A* şi *B* reprezentate în VM (baza *b*)

$$A = (\pm M_1)_b \cdot b^{C_1} \ , \ B = (\pm M_2)_b \cdot b^{C_2}$$

unde M_1 şi M_2 sunt mantisele iar C_1 şi C_2 sunt caracteristicile celor două numere, produsul celor două numere se calculează astfel:

$$P = A \cdot B = ((\pm M_1)_b \cdot (\pm M_2)_b) \cdot b^{C_1+C_2} \qquad (2.26)$$

Înmulţirea mantiselor şi adunarea caracteristicilor se poate realiza potrivit procedurilor prezentate anterior. În ceea ce priveşte normalizarea rezultatului, aceasta se realizează în acelaşi mod cu normalizarea rezultatului adunării/scăderii numerelor reprezentate în virgulă mobilă. Mantisa rezultată va avea un număr dublu de cifre, din care se vor reţine numai jumătate dacă se doreşte respectarea formatului.

Observaţie. Din suma caracteristicilor trebuie scăzut excesul, altfel rezultatul este viciat. Regula numai cu adunarea caracteristicilor este valabilă numai pentru exponenţi (exces =0).

Exemplu

Fie $A = 0.1F7 \cdot 16^3$ şi $B = 0.B54 \cdot 16^{-1}$ (baza $b=16$). Să se calculeze $P=A \cdot B$.

$M=0.1F7 \cdot 0.B54=0.16420C$
$C = C1+C2 = 3 – 1= 2$

M1		0	.	1 F 7	x
M2		0	.	B 5 4	
				7 D C	
			9 D 3		
		1 5 9 D			
	0	.	1 6 4 2 0 C		

P = 0. 1 6 4 2 0 C · 16^2

2.5.3. Împărţirea numerelor reprezentate în virgulă mobilă

Împărţirea a două numere reprezentate în virgulă mobilă, presupune *scăderea* caracteristicilor şi *împărţirea* mantiselor.

Pentru două numere *A* şi *B* reprezentate în VM (baza *b*)

$$A = (\pm M_1)_b \cdot b^{C_1} \ , \ B = (\pm M_2)_b \cdot b^{C_2}$$

unde M_1 şi M_2 sunt mantisele iar C_1 şi C_2 sunt caracteristicile celor două numere, produsul celor două numere se calculează astfel:

$$Q = A : B = ((\pm M_1)_b : (\pm M_2)_b) \cdot b^{C_1 - C_2} \qquad (2.27)$$

Împărţirea mantiselor şi scăderea caracteristicilor se poate realiza potrivit procedurilor prezentate anterior. În ceea ce priveşte normalizarea rezultatului, aceasta se realizează în acelaşi mod cu normalizarea rezultatului adunării/scăderii numerelor reprezentate în virgulă mobilă.

Observaţie. La diferenţa caracteristicilor trebuie adunat excesul, altfel rezultatul este viciat. Regula numai cu scăderea caracteristicilor este valabilă numai pentru exponenţi (exces =0).

Exemplul 2.18.

Fie $A = 0.1F7 \bullet 16^3$ şi $B = 0.B54 \bullet 16^{-1}$ (baza *b=16).* Să se calculeze $Q = A:B =$.

M=0.1F7:0.B54=0.2C67 rest 0.000D4
C = C1-C2 = 3 + 1= 4

```
1 F 7 0 | B 5 4
1 6 A 8 | 0. 2 C 6 7
= 8 C 8 0
  8 7 F 0
  -------
  = 4 9 0 0
    4 3 F 8
    -------
    = 5 0 8 0
      4 F A C
      -------
      = = D 4
```

Q= 0. 2 C 6 7 · 16^4

2.6. Operaţii aritmetice cu numere zecimale codificate binar

Procesarea numerelor zecimale la nivelul calculatoarelor numerice se realizează uzual utilizând codificarea binară a acestora utilizând coduri ponderate sau neponderate.

2.6.1. Codificarea binară a numerelor zecimale

Reprezentarea cu semn presupune plasarea bitului corespunzător în poziţia cea mai semnificativă[16]. După bitul de semn se consideră punctul zecimal, astfel că se vor considera reprezentate numai *numere subunitare.*

Se utilizează trei moduri de reprezentare şi anume:
- mărime şi semn;
- complement faţă de 9;
- complement faţă de 10.

Ca şi în cazul aritmeticii binare pentru numerele pozitive cele trei reprezentări coincid. Pentru numerele negative, se determină

[16] Este valabilă convenţia ***1****pentru numere negative,* ***0*** *pentru numere pozitive.*

complementele faţă de ***9*** respectiv faţă de ***10***, în cazul reprezentărilor în cod complementar.

2.6.2. Adunarea şi scăderea în cod binar-zecimal

Algoritmii prezentaţi la adunarea numerelor binare prezentaţi anterior rămân valabili şi pentru numerele zecimale şi se *aplică anumite corecţii*.

Necesitatea corecţiilor rezultă din faptul că din cele 16 combinaţii posibile ale codului cu patru biţi codul *BCD* nu utilizează decât 10 şi că în urma adunării sau scăderii se pot genera combinaţii invalide care trebuie eliminate.

În cazul codului ***8421*** toate combinaţiile cuprinse între *1010* şi *1111* sunt invalide şi pot fi eliminate dacă oricărei tetrade mai mare decât *1001 (cifra 9)* i se adaugă tetrada *0110 (cifra 6).* Aceiaşi tetradă se adaugă şi dacă în urma adunării rezultă tetradele *0000, 0001, 0010* şi transport la rangul superiore. Necesitatea acestor corecţii rezultă din tabla adunării BCD prezentată mai jos.

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	**A**
2	2	3	4	5	6	7	8	9	**A**	**B**
3	3	4	5	6	7	8	9	**A**	**B**	**C**
4	4	5	6	7	8	9	**A**	**B**	**C**	**D**
5	5	6	7	8	9	**A**	**B**	**C**	**D**	**E**
6	6	7	8	9	**A**	**B**	**C**	**D**	**E**	**F**
7	7	8	9	**A**	**B**	**C**	**D**	**E**	**F**	***10***
8	8	9	**A**	**B**	**C**	**D**	**E**	**F**	***10***	***11***
9	9	**A**	**B**	**C**	**D**	**E**	**F**	***10***	***11***	***12***

Această corecţie face ca *operaţia de adunare să se efectueze în baza 10 şi nu 16.*

În cazul scăderii se parcurge următoarea secvenţă:

1– fiecare cifră zecimală se exprimă printr-o tedradă binară în cod 8421;

2– se efectuează scăderea poziţie cu poziţie; dacă pentru o poziţie nu este necesar împrumut de la tedrada următoare, atunci rezultatul este corect;

3– dacă se impune un împrumut se execută următoarea secvenţă:

3.1 – se scade 1 de la poziţia următoare;

3.2 – se adună $16_{10}=10000_2$ la descăzutul scăderii curente;

3.3 – se scade $6_{10} = 0110_2$ din rezultat pentru a se efectua corecţia întrucât împrumutul a fost 16_{10} şi nu 10_{10} .

Exemplul 2.19

1. Fie $A_{10} = 789$ şi $B_{10} = 165$.

Să se determine utilizând reprezentarea semn mărime a codului *BCD* operaţia $C_{BCD}=A_{BCD} + B_{BCD}$.

$C_{10} = A_{10} + B_{10} = 789 + 165 = 954$

A_{BCD-MS} =	0111	1000	1001	+	
B_{BCD-MS} =	0001	0110	0101		
	1000	**1110**+	**1110**+		← Corecţii necesare, deoarece tetradele
		0110	**0110**		corespunzătoare rangurilor 10^{-2} şi 10^{-3}
		0100	0100		sunt mai mari decât 1001
		1←	**1**←		
$C1_{BCD-MS}$ =	0 1001	0101	0100		← Se adună transportul generat

C_{10} = 954

2. Fie $X_{10} = 532$ şi $Y_{10} = 178$.

Să se determine utilizând reprezentarea semn mărime a codului *BCD* operaţia $Z_{BCD}=X_{BCD} - Y_{BCD}$.

$Z_{10} = X_{10} - Y_{10} = 532 - 178 = 354$

X_{BCD-MS} =	0101	0011	0010	-	
B_{BCD-MS} =	0001	0111	1000		
	0100→	**10010**→	**10010**-		← Scădere 0011 – 0001 = 0010 Corecţii
		0111	**1000**		Împrumut 10000 + 0010 = 10010
	0100 -	1011 -	1010-		Se efectuează scăderile 1
	0001	**0**110	**0**110		Se efectuează corecţiile
$C1_{BCD-MS}$ =	0011	0101	0100		

Z_{10} = 354

2.6.3. Înmulţirea în cod binar-zecimal

În toate cazurile semnul produsului se stabileşte ca sumă modulo 2 a celor doi biţi de semn ai factorilor.

În continuare va fi prezentată o metodă de înmulţire a numerelor reprezentate în mărime-semn precum şi înmulţirea cu puteri întregi ale bazei 10.

2.6.3.1. Înmulţirea cu 10^k

O categorie specială de înmulţire a numerelor zecimale codificate binar o reprezintă înmulţirea cu puteri ale bazei 10 respectiv cu 10^k, care presupune deplasări după cum urmează:

k>0 → deplasare *stânga* cu **k** tetrade; (se adaugă zerouri în tetradele nesemnificative din dreapta rămase libere);

k<0 → deplasare *dreapta* cu **k** poziţii. (se adaugă zerouri în tetradele semnificative rămase libere).

În urma deplasărilor valoarea bitului de semn rămâne neschimbată şi lungimea numărului constantă. În aceste condiţii unele tetrade se pierd iar altele se completează cu zero.

Exemplul 2.20.

Să se convertească în *BCD* mărime-semn numărul $A_{10} = 0.385$şi să se determine $C1 = A \cdot 10^2$ şi $C2 = A \cdot 10^{-2}$

A_{BCD-MS} = 0 0011 1000 0101	
$C1_{BCD-MS}$ = 0 0101 0000 0000 **$C1_{10}$ = 0.5 0 0**	C1=A·100 Rezultatul real C1=38.5 Se pierd tetradele 1 şi 2 iar tetradele 2 şi 3 se completează cu zero
$C2_{BCD-MS}$ = 0 0000 0000 0011 **$C2_{10}$ = 0.0 0 3**	C2=A·0.01 Rezultatul real C2=0.00385 Se pierd tetradele 2 şi 3 iar tetradele 1 şi 2 se completează cu zero

Din exemplul de mai sus se observă că apar trunchieri, iar pentru ca rezultatul final să nu fie afectat se impune salvarea tetradelor care sunt eliminate.

2.6.3.2. Metoda adunării repetate

Această metodă este asemănătoare cu cea utilizată la înmulţirea binară. Deînmulţitul va fi adunat cu el însuşi de un număr de ori egal cu cifra corespunzătoare a înmulţitorului (0…9) pentru a forma produsul parţial. Acesta se deplasează cu o poziţie (tetradă) spre stânga şi începe ciclul de adunări corespunzător următoarei cifre a înmulţitorului. Procesul se încheie cu ciclul de adunări corespunzătoare celei mai semnificative cifre a înmulţitorului.

Exemplul 2.21.

Fie $A_{10} = 0.431$*(deînmulţit)* , şi $B_{10} = 0.232$ *(înmulţitor).*

Să se determine $P=A \cdot B$ folosind metoda adunării repetate în cadrul reprezentării BCD.

P=0.431·0232=0.099992

		A	0 1 0 0	0 0 1 1	0 0 0 1	x
		B	0 0 1 0	0 0 1 1	0 0 1 0	
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	= prod. partial initial
			0 1 0 0	0 0 1 1	0 0 0 1	= A
			0 1 0 0	0 0 1 1	0 0 0 1	= A
0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	0 1 1 0	0 0 1 0	=P1-primul prod. part.
		0 1 0 0	0 0 1 1	0 0 0 1		= A
0 0 0 0	0 0 0 0	0 1 0 0	1 0 1 1	0 1 1 1	0 0 1 0	
			0 1 1 0			corectie
0 0 0 0	0 0 0 0	0 1 0 1	0 0 0 1	0 1 1 1	0 0 1 0	
		0 1 0 0	0 0 1 1	0 0 0 1		= A
		1 0 0 1	0 1 0 0	1 0 0 0	0 0 1 0	
		0 1 0 0	0 0 1 1	0 0 0 1		= A
0 0 0 0	0 0 0 0	1 1 0 1	0 1 1 1	1 0 0 1	0 0 1 0	=P2-al doilea prod. part.
		0 1 1 0				corectie
	0 0 0 1	0 0 1 1	0 1 1 1	1 0 0 1	0 0 1 0	
	0 1 0 0	0 0 1 1	0 0 0 1			= A
	0 1 0 1	0 1 1 0	1 0 0 0	1 0 0 1	0 0 1 0	
	0 1 0 0	0 0 1 1	0 0 0 1			= A
0 0 0 0	1 0 0 1	1 0 0 1	1 0 0 1	1 0 0 1	0 0 1 0	Rezultat final

P= 0 . 0 9 9 9 9 2

2.6.4. Împărţirea în cod binar-zecimal

Metodele sunt similare cu cele utilizate la împărţirea binară cu toate că împărţirea zecimală este mai complicată decât cea binară. Această complexitate este dictată în primul rând de faptul că pentru cifrele câtului există zece posibilităţi (tetrade) în timp ce în cazul binar sunt numai două asemenea posibilităţi (cifrele binare 0 şi 1).

În continuare se prezintă ideile care stau la baza metodei comparării, care presupune comparaţii succesive cu multipli ai împărţitorului. Procesul de comparaţie se opreşte la cel mai mare multiplu care este mai mic decât deîmpărţitul. Acesta se scade din deîmpărţit, iar ordinul multiplului devine cifră a câtului. O variantă a acestei metode, exemplificată mai jos, presupune scăderea repetată a împărţitorului din deîmpărţit, procesul oprindu-se atunci când împărţitorul a devenit mai mare decât restul parţial. În aceste condiţii numărul de scăderi devine cifră a câtului.

Exemplul 2.22.

D e î m p ă r ţ i t A = 0 . 1 7 6
Î m p ă r ţ i t o r B = 0 . 7 3 1

0 .1 7 6 : 0 . 7 3 1 = 1 7 6 : 7 3 1

1 7 6 - Prima scădere - 555 < 731 q_0=0
7 3 1
- 5 5 5

1 7 6 0 - Prima scădere 1029>731
7 3 1
1 0 2 9 - A doua scădere 298<731 q_1=2
7 3 1
2 9 8 0 - Prima scădere 2249>731
7 3 1
2 2 4 9 - A doua scădere 1518>731
7 3 1
1 5 1 8 - A treia scădere 1518>731
7 3 1
7 8 7 - A patra scădere 298<731 q_1=4
7 3 1
= 5 6 0 - Prima scădere 560 < 731 q_1=0

0 . 1 7 6 = 0 . 7 3 1 x 0 . 2 4 + 0 . 0 0 0 5 6
Q = 0 . 2 4 R = 0 . 0 0 0 5 6

3. BAZELE LOGICE ALE CALCULATOARELOR NUMERICE

3. BAZELE LOGICE ALE CALCULATOARELOR NUMERICE

3.1. Funcţii şi operaţii logice

Caracteristica esenţială a tuturor generaţiilor de calculatoare numerice realizate până în prezent o constituie natura discretă a operaţiilor pe care acestea le efectuează.

Considerente de ordin tehnologic impun utilizarea în construcţia calculatorului a dispozitivelor cu două stări care condiţionează codificarea informaţiei şi efectuarea calculelor în sistem binar.

Analiza şi sinteza circuitelor de comutaţie aferente calculatoarelor numerice utilizează ca principal instrument matematic algebra logică (booleană).

În continuare se prezintă unele elemente atât ale algebrei logice cât şi ale unor circuite logice fundamentale.

3.1.1. Algebra booleană

Ca *structură* algebra booleană se defineşte prin următoarele entităţi:

- o mulţime suport;
- o mulţime de operatori;
- un grup de axiome.

Mulţimea suport este mulţimea binară, respectiv $M_2 = \{0,1\}$;

Mulţimea de operatori O conţine doi operatori *binari* şi unul *unar*. Operatorii binari sunt + şi • care se mai numesc operatori de *disjuncţie (sau)* respectiv de *conjuncţie* (*şi*). În ceea cea ce priveşte operatorul unar acesta este operatorul de negaţie $\bar{}$ *(nu).*

Grupul de axiome include pentru $M_2 = \{0,1\}$ un grup de şapte axiome care vor fi detaliate în continuare.

1. *Axioma de închidere,* conform căreia mulţimea M_2 este parte stabilă, în raport cu cele două operaţii respectiv

$x+y \in M_2$ *si* $x \cdot y \in M_2 \; (\forall) x,y \in M_2$.

2. *Axioma de asociativitate*

$$x+(y+z)=(x+y)+z \quad (\forall) x,y,z \in M_2$$
$$x \cdot (y \cdot z)=(x \cdot y) \cdot z \quad (\forall) x,y,z \in M_2.$$

3. *Axioma de comutativitate*

$$x+y=y+x \quad (\forall) x,y \in M_2$$
$$x \cdot y = y \cdot x \quad (\forall) x,y \in M_2.$$

4. *Axioma de absorbţie*

$$x+x \cdot y = x \quad (\forall) x,y \in M_2$$
$$x \cdot (x+y)=x \quad (\forall) x,y \in M_2.$$

5. *Axioma de distributivitate*

$$x+(y \cdot z)=(x+y) \cdot (x+z) \quad (\forall) x,y,z \in M_2$$
$$x \cdot (y+z)=x \cdot y + x \cdot z \quad (\forall) x,y,z \in M_2.$$

6. *Axioma elementelor neutre*

$$x+0=0+x=x \quad (\forall) x \in M_2 \quad 0 \textit{ element neutru pentru operatia} +$$
$$x \cdot 1 = 1 \cdot x = x \quad (\forall) x \in M_2. \quad 1 \textit{ element neutru pentru operatia} \cdot$$

7. *Axioma elementului invers (complementului)*

$$(\forall) x \in M_2, (\exists) \bar{x} \in M_2 \textit{ unic cu}$$
$$\left| x+\bar{x}=1, \quad \textit{(principiul tertului exclus);} \right.$$
$$x \cdot \bar{x} = 0. \quad \textit{(principiul contradictiei).}$$

La definirea axiomatică a algebrei s-au folosit notaţiile +, •, $\overline{x}$ pentru cele două legi de compoziţie, respectiv pentru elementul invers. În logică şi tehnică există denumiri şi semnificaţii specifice, evidenţiate în *Tabelul 3.1.*

Tabelul 3.1.

Matematică		**Logică**		**Tehnică**	
Denumire	Simbol	Denumire	Simbol	Denumire	Simbol
Prima operaţie	+	Disjuncţie	$\cup$	SAU	$\cup$
A doua operaţie	$\bullet$	Conjuncţie	$\cap$	ŞI	$\cap$
Element invers	$\overline{x}$	Negaţie	$\neg x$	NU	$\overline{x}$

Pornind de la axiome se pot deduce teoremele prezentate în *Tabelul 3.2* care se constituie în reguli de calcul în cadrul algebrei booleene.

Tabelul 3.2.

Nr.	**Denumire**	**Forma produs**	**Forma sumă**
T1	Dublă negaţie (involuţia)	$\overline{\overline{x}} = x \mid (\forall) x \in M_2$	$\overline{\overline{x}} = x \mid (\forall) x \in M_2$
T2	Elemente neutre	$x \cdot 0 = 0 \mid (\forall) x \in M_2$	$x + 1 = 1 \mid (\forall) x \in M_2$
T3	Idempotenţa (tautologia)	$x \cdot x \cdot \ldots \cdot x = x \mid (\forall) x \in M_2$	$x + x + \ldots + x = x \mid (\forall) x \in M_2$
T4	De Morgan	$\overline{x \cdot y} = \overline{x} + \overline{y} \mid (\forall) x, y \in M_2$	$\overline{x + y} = \overline{x} \cdot \overline{y} \mid (\forall) x, y \in M_2$

Oricare dintre cele 4 teoreme poate fi demonstrată utilizând axiomele cu ajutorul cărora s-a definit structura algebrei.

3.1.2. Variabile şi funcţii logice

O funcţie $y = f(x_1, x_2, ..., x_2)$ reprezintă o funcţie logică dacă domeniul de definiţie este reprezentat de produsul cartezian *{0,1}*^{n}, cu alte cuvinte $f : \{0,1\}^n \rightarrow \{0,1\}$.

O funcţie logică (booleană) pune în corespondenţă o combinaţie binară asociată produsului cartezian cu una din valorile **0** sau **1**. Acest aspect este evidenţiat în figura 3.1 unde combinaţia binară este reprezentată

de starea a *n* comutatoare interconectate iar valoarea funcţiei de starea de conducţie sau nu a grupării.

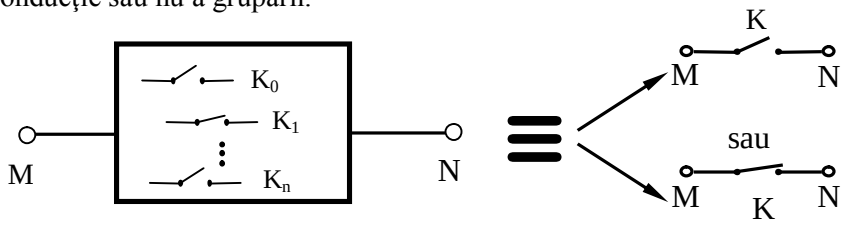


Fig. 3.1. Ilustrarea definirii unei funcţii logice.

În reprezentarea din figura 3.1 se consideră că unui comutator deschis i se asociază valoarea logică ***0,*** iar unuia închis valoarea logică ***1***. Existenţa unei căi de curent între bornele *M* şi *N* depinde de modul de interconectare al celor *n* comutatoare precum şi de starea fiecăruia. Această dependenţă este exprimată matematic prin intermediul funcţiei logice $y = f(x_1, x_2, ..., x_2)$.

Deoarece o funcţie logică (booleană) pune în corespondenţă o combinaţie binară asociată produsului cartezian cu una din valorile *0* sau *1*, rezultă că într-un *SN* binar , cu *n* variabile se pot forma $m = 2^n$ combinaţii[1]. Acestea la rândul lor se pot combina în $N = 2^m = 2^{2^n}$ moduri, rezultând posibilitatea definirii a $N = 2^{2^n}$ funcţii cu *n* variabile.

Exemplul 3.1.

Cu variabila binară $x \in \{0,1\}$ se pot forma $N = 2^{2^1} = 4$ funcţii logice şi anume:

$f_0(x) = 0;$

$f_1(x) = 1;$

$f_2(x) = x;$

$f_3(x) = \bar{x}.$

[1] În general, într-un *SN* cu baza *b* se pot forma b^n combinaţii.

Conform precizărilor de mai sus cu două variabile se pot forma $N = 2^{2^2} = 16$ 16 funcţii logice ale căror valori sunt sistematizate în *Tabelul 3.3*. Un asemenea tabel se numeşte *tabel de adevăr* în care prin convenţie valoarea *0* corespunde unei afirmaţii de tip *FALS*, iar valoarea *1* unei afirmaţii de tip *ADEVĂRAT*.

Tabelul 3.3

x	y	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Din analiza *Tabelului 3.3* se observă că perechile de funcţii $f_k - f_{15-k}$ sunt complementare pentru $k = 0,1,\ldots,15$. În continuare vor fi prezentate funcţiile frecvent utilizate.

1. Funcţiile $f_{15}(x,y)=1 \; si \; f_0(x,y)=0$ sunt funcţiile constante *1* şi *0*.
2. Funcţiile $f_{14}(x,y) \; si \; f_1(x,y)$ sunt funcţiile logice *SAU* respectiv *SAU-NU*.
3. Funcţiile $f_6(x,y)=y \; si \; f_9(x,y)=\overline{y}$ sunt funcţiile directă şi inversă referitoare la variabila *y*.
4. Funcţiile $f_{12}(x,y)=x \; si \; f_3(x,y)=\overline{x}$ sunt funcţiile directă şi inversă referitoare la variabila *x*.
5. Funcţiile $f_4(x,y) \; si \; f_{11}(x,y)$ sunt funcţiile logice *ŞI* respectiv *ŞI-NU*.
6. Funcţiile $f_{10}(x,y) \; si \; f_5(x,y)$ sunt funcţiile logice *SAU EXCLUSIV* respectiv *SAU EXCLUSIV-NU*.

Observaţii

1. Pentru o parte din funcţiile de mai sus este larg utilizată terminologia normală şi/sau abreviată din limba engleză, conform corespondenţei din *Tabelele 3.4 şi 3.5*.

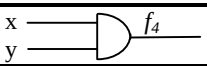
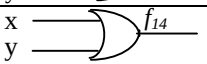
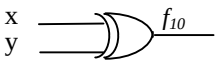
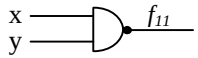
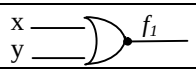
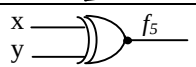
Tabelul 3.4

Funcţie	Denumire		
	Limba română	Limba engleză	
		Normală	Abreviată
f_1	SAU – NU	NOT - OR	**NOR**
f_4	ŞI	AND	**AND**
f_5	SAU-EXCLUSIV-NU	NOT – EXCLUSIVE-OR	**NXOR**
f_{10}	SAU-EXCLUSIV	EXCLUSIVE - OR	**XOR**
f_{11}	ŞI – NU	NOT - AND	**NAND**
f_{14}	SAU	OR	**OR**

2. Funcţiile ŞI *(END)*, SAU *(OR)*, NU *(NOT)* corespunzând operatorilor din definirea *algebrei boolene* ca structură, se numesc *funcţii logice de bază* întrucât cu ajutorul lor se poate exprima orice altă funcţie logică.

3. În tabelul 3.4 nu au fost incluse funcţiile NOT referitoare la cele două variabile.

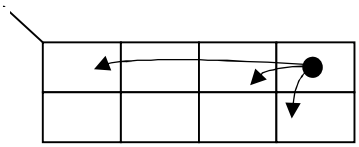
Tabelul 3.5

Denumire funcţie	**Ecuaţie logică**	**Simbol**
AND	$f_4 = x \cap y = x \cdot y$	
OR	$f_{14} = x \cup y = x + y$	
XOR	$f_{10} = x \oplus y = x \cdot \bar{y} + \bar{x} \cdot y$	
NAND	$f_{11} = x \uparrow y = \overline{x \cdot y}$	
NOR	$f_1 = x \downarrow y = \overline{x + y}$	
NXOR	$f_5 = x \otimes y = x \cdot y + \bar{x} \cdot \bar{y}$	

Ilustrarea semnificaţiei operatorilor logici se poate realiza prin diagrame Venn, tabele de adevăr, diagrame Karnaugh, scheme cu comutatoare etc.

Reprezentarea cea mai comodă şi pretabilă formalizării este cea realizată cu ajutorul tabelelor de adevăr, o astfel de reprezentare fiind cea din *Tabelul 3.3.*

Reprezentarea cu ajutorul *diagramei Karnaugh* constă în marcarea punctelor domeniului de definiţie într-o diagramă plană şi precizarea valorilor funcţiei în fiecare din aceste puncte. De exemplu în figura 3.2 este reprezentată diagrama Karnaugh pentru o funcţie de trei variabile cu marcarea vecinătăţilor punctului *010*.



După cum se observă, trecerea de la o combinaţie la alta pe laturile diagramei Karnaugh se face prin modificarea unui singur bit.

Exemplul 3.2.

a) Să se demonstreze axiomele absorbţiei.
$x \cdot (x + y) = x$; $x + x \cdot y = x$.

a.1) $x \cdot (x + y) = x \cdot x + x \cdot y = x + x \cdot y = x \cdot (1 + y) = x \cdot 1 = x$;

a.2) $x + x \cdot y = x \cdot (1 + y) = x \cdot 1 = x$

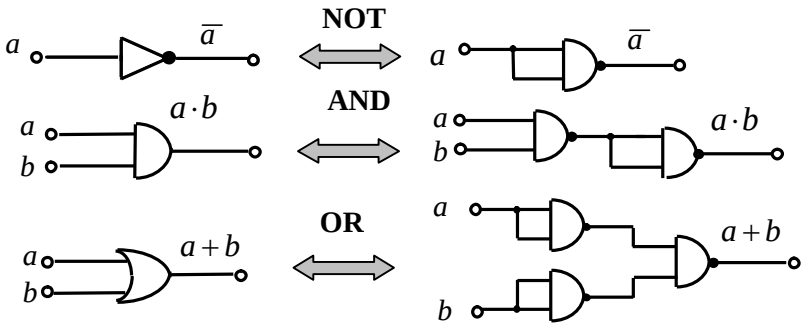
b) Să se implementeze cu porţi NAND funcţiile logice *NOT, AND, OR.*

b.1) $\bar{a} = \overline{a \cdot a} = a \uparrow a$;

b.2) $a \cdot b = \overline{\overline{a \cdot b}} = \overline{a \uparrow b} = (a \uparrow b) \uparrow (a \uparrow b)$;

b.3) $a + b = \overline{\overline{a + b}} = \overline{\bar{a} \cdot \bar{b}} = (a \uparrow a) \uparrow (b \uparrow b)$.

Rezultă implementarea prezentată în schemele de mai jos.



c) Să se implementeze cu porţi *NAND* funcţia *EXCLUSIVE OR*.

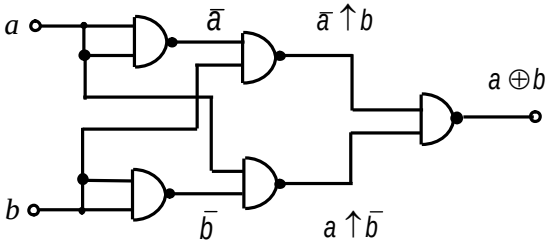
$a \oplus b = \overline{a} \cdot b + a \cdot \overline{b}$

$\overline{a} \cdot b = \overline{\overline{\overline{a} \cdot b}} = \overline{\overline{a} \uparrow b}$; $a \cdot \overline{b} = \overline{\overline{a \cdot \overline{b}}} = \overline{a \uparrow \overline{b}}$

Ţinând cont de rezultatele anterioare se obţine echivalenţa

$a \oplus b = [(a \uparrow a) \uparrow b)] \uparrow [a \uparrow (b \uparrow b)]$

evidenţiată în reprezentarea de mai jos



O funcţie logică se poate reprezenta dezvoltat în două forme şi anume:

- *forma disjunctivă canonică (FDC),* cu utilizarea constituenţilor unităţii;
- *forma conjunctivă canonică (FCC),* cu utilizarea constituenţilor lui zero.

FDC presupune exprimarea funcţiei ca o *disjuncţie de conjuncţii* (reuniune de intersecţii) în care variabilele care au valoarea ***0*** *se consideră* ***negate***.

FCC presupune exprimarea funcţiei ca o *conjuncţie de disjuncţii* (intersecţie de reuniuni) în care variabilele care au valoarea ***1*** *se consideră negate*.

Exemplul 3.3.

Să se determine *FDC* şi *FCC* pentru funcţia

$y = (x_1 \cap \bar{x}_3) \cup (\bar{x}_1 \cap x_3)$.

Din motive de simplificare a scrierii semnul $\cup$ va fi înlocuit cu + iar semnul $\cap$ cu $\bullet$.

1. Determinarea *FDC*

a) *prin utilizarea proprietăţilor operaţiilor logice*.

Dacă *x* este o variabilă logică se ştie că $\bar{x} + x = 1 \; si \; x \cdot 1 = x$, astfel încât rezultă

FDC: $y = (x_2 + \bar{x}_2) \cdot x_1 \cdot \bar{x}_3 + (x_2 + \bar{x}_2) \cdot \bar{x}_1 \cdot x_3$

$y = x_1 \cdot x_2 \cdot \bar{x}_3 + x_1 \cdot \bar{x}_2 \cdot \bar{x}_3 + \bar{x}_1 \cdot x_2 \cdot x_3 + \bar{x}_1 \cdot \bar{x}_2 \cdot x_3$

b) prin utilizarea diagramelor Karnaugh.

Pentru fiecare termen se completează cu **1** *n-k+1* poziţii în diagramă, unde *n* este numărul de variabile iar *k* este numărul de variabile asociat termenului. Dacă funcţia conţine de exemplu x_1x_2 atunci se pune **1** în căsuţele corespunzătoare combinaţiilor $x_1x_2x_3$ şi $x_1x_2\bar{x}_3$. Pentru funcţia y din exemplu diagrama Karnaugh este prezentată în figura 3.4.

x_3 \ x_1x_2	00	01	11	10
0	**0**	**0**	**1**	**1**
1	**1**	**1**	**0**	**0**

Fig. 3.3. Diagrama Karnaugh pentru determinarea *FCD*

Prin interpretarea datelor din figura 3.4 rezultă:

FDC: $y = x_1 \cdot x_2 \cdot \bar{x}_3 + x_1 \cdot \bar{x}_2 \cdot \bar{x}_3 + \bar{x}_1 \cdot x_2 \cdot x_3 + \bar{x}_1 \cdot \bar{x}_2 \cdot x_3$.

Se observă că rezultatele obţinute prin cele două metode coincid.

2. Determinarea *FCC*

a) prin utilizarea proprietăţilor operaţiilor logice.

Dacă x este o variabilă logică se ştie că $\bar{x} \cdot x = 0$ *si* $x + 0 = x$.

FCC: $y = x_1 \cdot \bar{x}_1 + x_1 \cdot \bar{x}_3 + x_3 \cdot \bar{x}_3 + \bar{x}_1 \cdot x_3$;

$$y = x_1 \cdot (\bar{x}_1 + \bar{x}_3) + x_3 \cdot (\bar{x}_1 + \bar{x}_3);$$
$$y = (x_1 + x_3) \cdot (\bar{x}_1 + \bar{x}_3).$$

Dacă *u* şi *v* sunt două variabile logice atunci

$$(u + v) \cdot (u + \bar{v}) = u \cdot u + u \cdot \bar{v} + v \cdot u + v \cdot \bar{v} = u + u \cdot (v + \bar{v}) = u + u = u$$

În aceste condiţii

$$x_1 + x_3 = (x_1 + x_2 + x_3) \cdot (x_1 + \bar{x}_2 + x_3);$$
$$\bar{x}_1 + \bar{x}_3 = (\bar{x}_1 + x_2 + \bar{x}_3) \cdot (\bar{x}_1 + \bar{x}_2 + \bar{x}_3);$$
$$y = (x_1 + x_2 + x_3) \cdot (x_1 + \bar{x}_2 + x_3) \cdot (\bar{x}_1 + x_2 + \bar{x}_3) \cdot (\bar{x}_1 + \bar{x}_2 + \bar{x}_3).$$

b) prin utilizarea diagramelor Karnaugh. Respectând regula de la determinarea *FDC* (numai că în loc de unu se trece zero), *FCC* rezultă din figura 3.3.

FCC:

$$y = (x_1 + x_2 + x_3) \cdot (x_1 + \bar{x}_2 + x_3) \cdot (\bar{x}_1 + x_2 + \bar{x}_3) \cdot (\bar{x}_1 + \bar{x}_2 + \bar{x}_3).$$

Se observă că cele două obţinute pentru *FCC* coincid.

3.1.3. Minimizarea funcţiilor logice

Minimizarea unei funcţii booleene implică reducerea la minimum a numărului de variabile şi a simbolurilor de funcţii implicate în reprezentarea acesteia.

Metodele de minimizare pot fi încadrate în două categorii: *analitice* şi *grafice.*

Metodele analitice constau în principal din calcule efectuate în funcţia dată pe baza axiomelor şi teoremelor algebrei binare. Principiul metodei va fi ilustrat pe funcţia de trei variabile din exemplul precedent.

Pentru început se va considera pentru funcţia *FDC*

$$y = x_1 \cdot x_2 \cdot \bar{x}_3 + x_1 \cdot \bar{x}_2 \cdot \bar{x}_3 + \bar{x}_1 \cdot x_2 \cdot x_3 + \bar{x}_1 \cdot \bar{x}_2 \cdot x_3$$

Aplicând distributivitatea pentru primii şi respectiv pentru ultimii doi termeni rezultă:

$$y = x_1 \cdot \bar{x}_3 \cdot (x_2 + \bar{x}_2) + \bar{x}_1 \cdot x_3 \cdot (x_2 + \bar{x}_2);$$

respectiv

$$y = x_1 \cdot \bar{x}_3 + \bar{x}_1 \cdot x_3.$$

Minimizarea se poate realiza pornind şi de la FCC

$$y = (x_1 + x_2 + x_3) \cdot (x_1 + \bar{x}_2 + x_3) \cdot (\bar{x}_1 + x_2 + \bar{x}_3) \cdot (\bar{x}_1 + \bar{x}_2 + \bar{x}_3).$$

Asociind primii doi şi respectiv ultimii doi factori şi efectuând produsele rezultă:

$$(x_1 + x_2 + x_3) \cdot (x_1 + \bar{x}_2 + x_3) = (x_1 + x_3) \cdot (x_1 + x_3) +$$
$$+ (x_1 + x_3) \cdot \bar{x}_2 + (x_1 + x_3) \cdot x_2 + x_2 \cdot \bar{x}_2 =$$
$$= (x_1 + x_3) \cdot (x_2 + \bar{x}_2) = x_1 + x_3$$

$$(\bar{x}_1 + x_2 + \bar{x}_3) \cdot (\bar{x}_1 + \bar{x}_2 + \bar{x}_3) = (\bar{x}_1 + \bar{x}_3) \cdot (\bar{x}_1 + \bar{x}_3) +$$
$$+ (\bar{x}_1 + \bar{x}_3) \cdot \bar{x}_2 + (\bar{x}_1 + \bar{x}_3) \cdot x_2 + x_2 \cdot \bar{x}_2 =$$
$$= (\bar{x}_1 + \bar{x}_3) \cdot (x_2 + \bar{x}_2) = \bar{x}_1 + \bar{x}_3$$

$y = (x_1 + x_3) \cdot (\bar{x}_1 + \bar{x}_3)$, respectiv

$y = x_1 \cdot \bar{x}_3 + \bar{x}_1 \cdot x_3$. .

Metodele grafice presupun constituirea unor tabele sau matriţe de combinaţii, din care prin grupări şi asocieri corespunzătoare rezultă reduceri. Din categoria acestor metode, în continuare se vor face referiri la cea care utilizează diagrama Karnaugh.

După cum s-a văzut, două celule adiacente într-o diagramă Karnaugh diferă prin valoarea unei singure variabile. Dacă termenilor din două asemenea celule li se aplică proprietatea de distributivitate şi principiul terţului exclus se elimină variabila care îşi schimbă valoarea.

Referitor la acest procedeu de reducere şi implicit de minimizare pot fi formulate următoarele observaţii:

a) un grup de 2^m celule vecine ocupate cu unităţi permite eliminarea a *m* variabile;

b) pentru reducere, fiecare celulă trebuie să facă parte dintr-o grupare, dar poate fi inclusă în mai multe;

c) cel mai avansat grad de simplificare se obţine dacă unităţile dintr-o diagramă Karnaugh sunt grupate într-un număr minim de grupări fiecare grup conţinând un număr minim de unităţi;

d) pentru a putea aplica în mod succesiv proprietatea de distributivitate şi teorema terţului exclus, numărul unităţilor din grupările formate trebuie să fie o putere întreagă a lui 2.

Reguli similare pot fi deduse şi pentru deducerea formei conjunctive minime. În acest caz, în diagrama Karnaugh se vor grupa zerourile. Se va scrie apoi disjuncţia grupurilor de zerouri vecine, iar forma minimă va fi conjuncţia grupurilor de coordonate. Metoda grafică de minimizare cu ajutorul diagramelor Karnaugh va fi ilustrată prin următorul exemplu.

Exemplul 3.4.

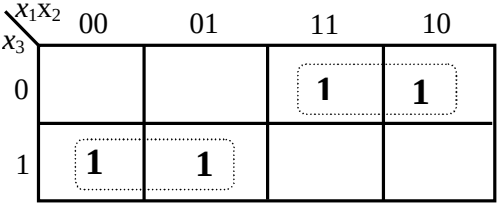
Fie funcţia logică de trei variabile $y = f(x_1, x_2, x_3)$ cunoscută prin intermediul tabelei de adevăr.

x_1	0	0	0	0	1	1	1	1
x_2	0	0	1	1	0	0	1	1
x_3	0	1	0	1	0	1	0	1
y	0	1	0	1	1	0	1	0

Să se determine formele minime disjunctivă şi conjunctivă cu ajutorul diagramelor Karnaugh.

a) determinarea *formei minime disjunctive.*

Se construieşte diagrama Karnaugh, unde se reprezintă constituenţii unităţii:

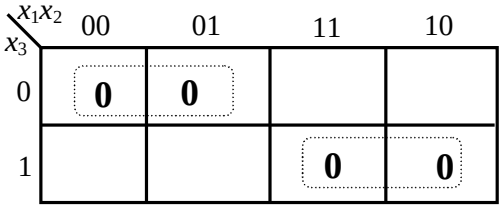


Din grupele încercuite se elimină variabila x_2 atât pentru grupa din linia superioară cât şi pentru cea din linia inferioară astfel încât rezultă pentru forma minimă disjunctivă (disjuncţie de conjuncţii) expresia:

$$y_D = x_1 \cdot \overline{x}_3 + \overline{x}_1 \cdot x_3.$$

b) determinarea *formei minime conjunctive.*

Se construieşte diagrama Karnaugh, unde se reprezintă constituenţii lui zero:



Ca şi în situaţia precedentă din grupele încercuite se elimină variabila x_2 astfel încât rezultă pentru forma minimă (conjuncţii de disjuncţii) expresia:

$$y_C = (x_1 + x_3) \cdot (\overline{x}_1 + \overline{x}_3).$$

Din tabela de adevăr prezentată mai jos rezultă echivalenţa celor două forme minime obţinute.

x_1	x_3	$\overline{x}_1$	$\overline{x}_3$	$\overline{x}_1 \cdot x_3$	$x_1 \cdot \overline{x}_3$	$\overline{x}_1 + \overline{x}_3$	$x_1 + x_3$	y_D	y_C
0	0	1	1	0	0	1	0	0	0
0	1	1	0	1	0	1	1	1	1
1	0	0	1	0	1	1	1	1	1
1	1	0	0	0	0	0	1	0	0

Etapa care succede minimizării este aceea a implementării funcţiei logice. Această implementare se realizează cu elemente de comutaţie de diverse tipuri cum ar fi: contacte şi relee, porţi logice etc.

O manieră de implementare o reprezintă utilizarea porţilor *AND-OR-NOT*, figura 3.4.a şi *NAND* figura 3.4.b.

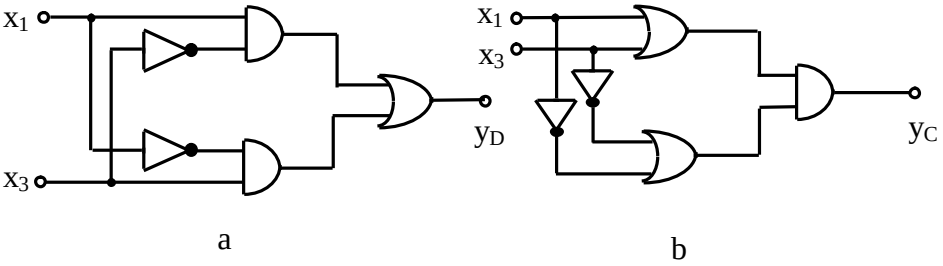


Fig. 3.4. Implementarea funcţiilor logice minimizate cu porţi *AND_OR_NOT* : a - forma minimă disjunctivă; b - forma minimă conjunctivă.

Utilizând rezultatele obţinute în exemplul 3.2.b, rezultă reprezentările din figurile 3.5.a şi 3.5.b.

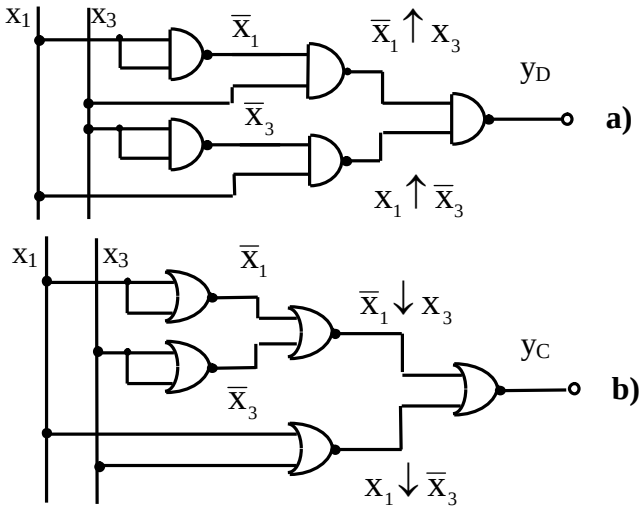


Fig. 3.5. Implementarea funcţiilor logice minimale: a - minim disjunctive cu porţi *NAND*; b - minim conjunctive cu porţi *NOR*.

3.2. Proiectarea structurilor combinaţionale

Caracteristica principală a circuitelor logice combinaţionale (*CLC)* o reprezintă dependenţa mărimilor de ieşire ale acestora *numai* de combinaţiile aplicate la intrare, nu şi de timp.

Schema bloc a unui *CLC* este prezentată în figura 3.6. Acesta dispune de intrările x_0, x_1, … x_{m-1} şi generează în exterior ieşirile y_0, y_1, …, y_{n-1}. Funcţionarea CLC poate fi descrisă cu ajutorul unei funcţii logice (de comutaţie).

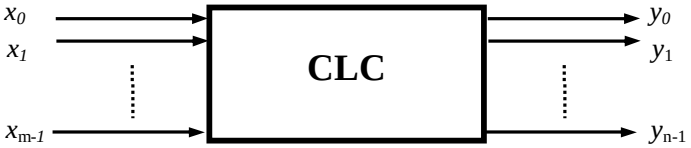


Fig. 3.6. Circuit logic combinaţional.

Pe baza reprezentării din figura 3.6 se pot defini funcţiile de tranziţie a ieşirilor pentru un *CLC* după cum urmează:

$$\begin{aligned} y_0 &= g_0(x_0, x_1, x_2, ..., x_{m-1}); \\ y_1 &= g_1(x_0, x_1, x_2, ..., x_{m-1}); \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ y_{n-1} &= g_{n-1}(x_0, x_1, x_2, ..., x_{m-1}). \end{aligned} \qquad (3.1)$$

Modelul matematic al unui *CLS* este reprezentat de tripletul

$$C_C = (X, Y, g), \qquad (3.2)$$

unde - $X = \{x_0, x_1, x_2, ..., x_{m-1}\}$ este mulţimea variabilelor binare de intrare;

- $Y = \{y_0, y_1, y_2, ..., y_{n-1}\}$ este mulţimea variabilelor binare de ieşire;
- $g : X \to Y$ este funcţia de tranziţie a ieşirilor– relaţia (3.1).

În mod uzual tratarea *CLC* implică soluţionarea a două tipuri de probleme reprezentate de *analiza* şi *sinteza* acestor circuite.

Analiza *CLC* pleacă de la cunoaşterea schemei acestuia şi urmăreşte stabilirea funcţionării, concretizată prin tabela de adevăr sau prin

scrierea expresiilor variabilelor de ieşire funcţie de cele de intrare respectiv a funcţiilor de tranziţie a ieşirilor din relaţia (3.1) .

Sinteza *CLC* presupune parcurgerea următoarelor etape pentru stabilirea structurii circuitului:

- definirea funcţiilor logice;
- minimizarea acestora;
- obţinerea schemei circuitului.

Operaţia de sinteză se reprezintă practic proiectarea *CLC*.

În structura unui calculator numeric se întâlnesc numeroase tipuri de *CLC* între care reprezentative sunt: *convertoarele de cod, codificatoarele şi decodificatoarele, multiplexoarele şi demultiplexoarele, comparatoarele, detectoarele şi generatoarele de paritate, ariile logice programabile, memoriile şi circuitele aritmetice.*

În continuare vor fi prezentate elemente privind sinteza (proiectarea) unor *CLC* uzuale din structura unui calculator numeric.

3.2.1. Convertoare de cod

B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

a

b

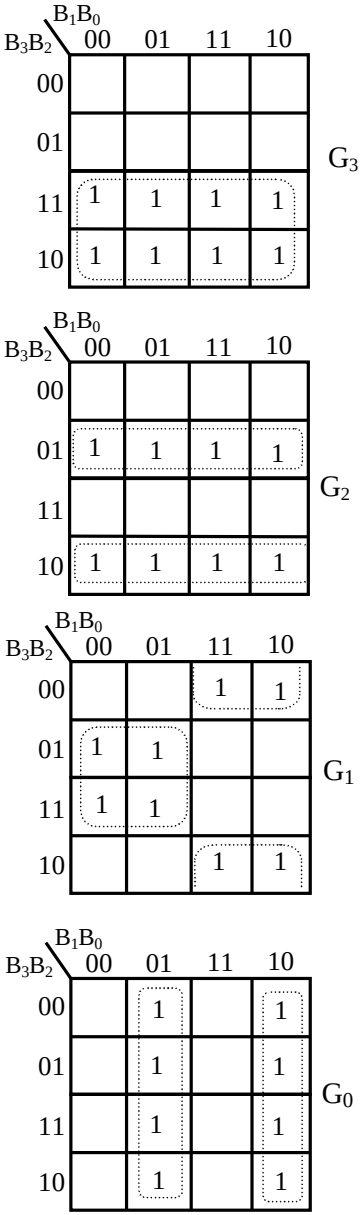


Fig. 3.7. Elemente aferente proiectării unui convertor de cod *binar natural – Gray:* a – tabela de adevăr; b – diagramele Karnaugh.

Convertoarele de cod sunt *CLC* care permit trecerea dintr-un cod binar în altul. Sinteza unui asemenea *CLC* se va exemplifica pentru un convertor *din cod binar în cod Gray*. În figura 3.7 se prezintă elementele aferente sintezei acestui tip de convertor, în care $B_3\ B_2\ B_1\ B_0$ reprezintă cuvântul binar aplicat la intrare, iar $G_3\ G_2\ G_1\ G_0$ cuvântul binar obţinut la ieşire.

Efectuând reducerile în diagramele Karnaugh rezultă relaţiile de implementare:

$$\begin{aligned} G_3 &= B_3\,; \\ G_2 &= \overline{B}_3 \cdot B_2 + B_3 \cdot \overline{B}_2 = B_3 \oplus B_2 \\ G_1 &= \overline{B}_2 \cdot B_1 + B_2 \cdot \overline{B}_1 = B_2 \oplus B_1 \\ G_0 &= \overline{B}_1 \cdot B_0 + B_1 \cdot \overline{B}_0 = B_1 \oplus B_0\,. \end{aligned} \qquad (3.3)$$

În figura 3.8 se prezintă două variante de implementare ale relaţiilor (3.3).

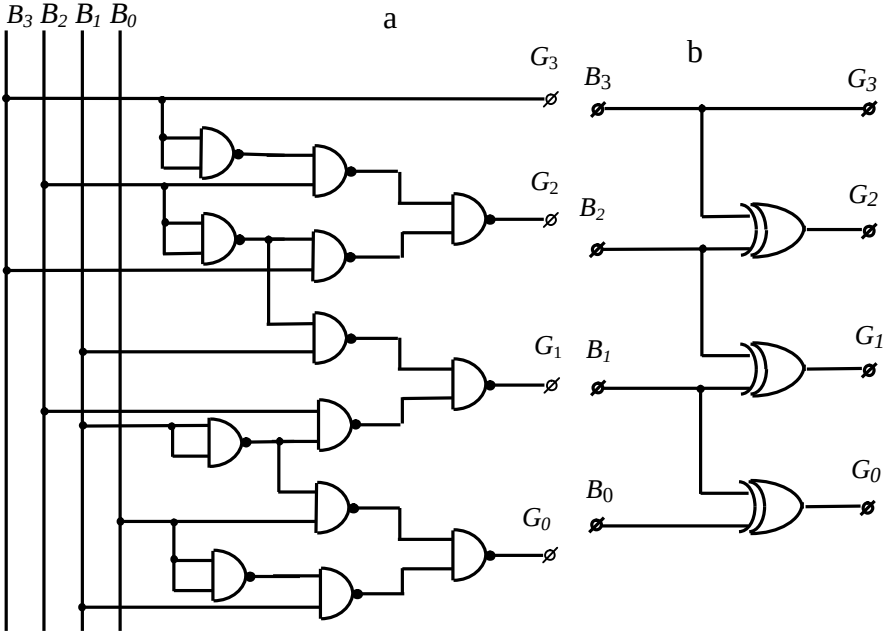


Fig. 3.8. Schema convertorului din cod binar natural în cod Gray:
a - realizarea cu porţi *NAND*; b - realizarea cu circuite *EXCLUSIVE OR*.

3.2.2. *Codificatoare şi decodificatoare*

Codificatoarele sunt *CLC* la care activarea unei intrări, dintr-un grup de *m,* conduce la apariţia unui cuvânt de cod la ieşire format din *n* biţi *($m \leq 2^n$)*. În figura 3.9 se prezintă elemente aferente unui codificator cu *m=3* şi *n=2*.

X	*Y*	*Z*	*A*	*B*
0	0	0	0	0
1	0	0	0	1
0	1	0	1	0
0	0	1	1	1

$A = \overline{X} \cdot (Y \oplus Z)$

$B = \overline{Y} \cdot (X \oplus Z)$

Fig. 3.9. Codificator cu m=3 şi n=2 (schema bloc, tabela de adevăr, funcţii logice.

Decodificatoarele sunt *CLC* la care se activează una sau mai multe ieşiri funcţie de cuvântul de cod aplicat la intrare. Decodificarea este necesară în aplicaţii care se referă la adresarea memoriilor, afişarea numerică, multiplexarea datelor etc. În continuare se vor prezenta elemente aferente decodificatorului BCD – zecimal.

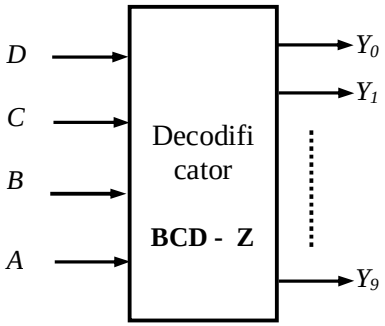


Fig. 3.9.a. Decodificator *BCD-Z* :schema bloc, tabela de adevăr.

D	**C**	**B**	**A**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1
1	0	1	0	**C O M B I N A | I I**									
1	0	1	1										
1	1	0	0										
1	1	0	1										
1	1	1	0	**I N T E R Z I S E**									
1	1	1	1										

Decodificatorul BCD-Z are ca intrări cele patru ranguri ale reprezentării în binar a cifrelor zecimale iar ca ieşiri 10 variabile care corespund celor 10 cifre zecimale. Fiecare din ieşiri va trebui să aibă valoarea **1** atunci când combinaţia valorilor logice de intrare corespunde reprezentării binare a cifrei zecimale asociate ieşirii respective. În figura 3.9 se prezintă elementele aferente sintezei decodificatorului.
După cum se observă combinaţiile interzise sunt rejectate, apariţia oricăreia dintre ele duce la stabilirea tuturor ieşirilor în **1**.

$$y_0 = \overline{D}\,\overline{C}\,\overline{B}\,\overline{A} \quad y_3 = \overline{D}\,\overline{C}BA \quad y_6 = \overline{D}CB\overline{A}$$
$$y_1 = \overline{D}\,\overline{C}\,\overline{B}A \quad y_4 = \overline{D}C\overline{B}\,\overline{A} \quad y_7 = \overline{D}CBA$$
$$y_2 = \overline{D}\,\overline{C}B\overline{A} \quad y_5 = \overline{D}C\overline{B}A \quad y_8 = D\overline{C}\,\overline{B}\,\overline{A} \quad y_9 = D\overline{C}\,\overline{B}A$$

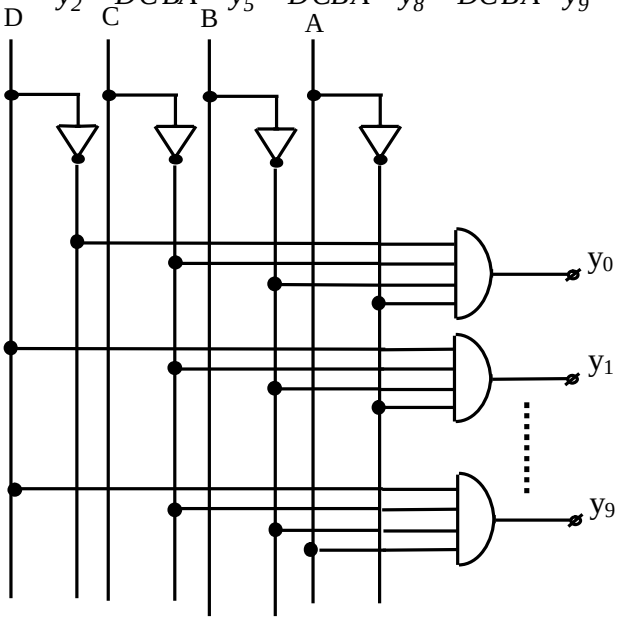


Fig. 3.9.b. Decodificatorul *BCD-Z*: funcţii logice şi implementare

3.2.3. *Multiplexoare şi demultiplexoare*

Multiplexoarele sunt *CLC* care permit transferul datelor de la una din intrări către o ieşire unică. Selecţia se face pe baza unui cuvânt de selecţie (adresă). Din punct de vedere funcţional *MUX* pot fi privite ca o

reţea de comutatoare comandate. *MUX* pot fi analogice sau numerice, ultimele fiind specifice calculatoarelor numerice.

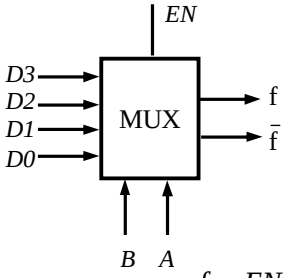
Numărul de biţi *n* ai cuvântului de selecţie pentru un multiplexor cu *N* intrări şi intrare de activare *(ENABLE)* este dat de relaţia

$$n = log_2 N + 1 \quad , \tag{3.4}$$

care în cazul în care *N* este putere a lui 2, respectiv $N = 2^k$ devine

$$n = log_2 2^k + 1 = k + 1 . \tag{3.5}$$

În continuare se va face sinteza unui *MUX* 4:1 numeric realizat cu porţi logice, pentru care aplicând relaţia (3.5) rezultă *n=2.*



EN	*B*	*A*	*f = Canal*
1	0	0	*D3*
1	0	1	*D2*
1	1	0	*D1*
1	1	1	*D0*
0	*	*	-

$$f = EN \cdot (\overline{B} \cdot \overline{A} \cdot D3 + \overline{B} \cdot A \cdot D2 + B \cdot \overline{A} \cdot D1 + B \cdot A \cdot D0)$$

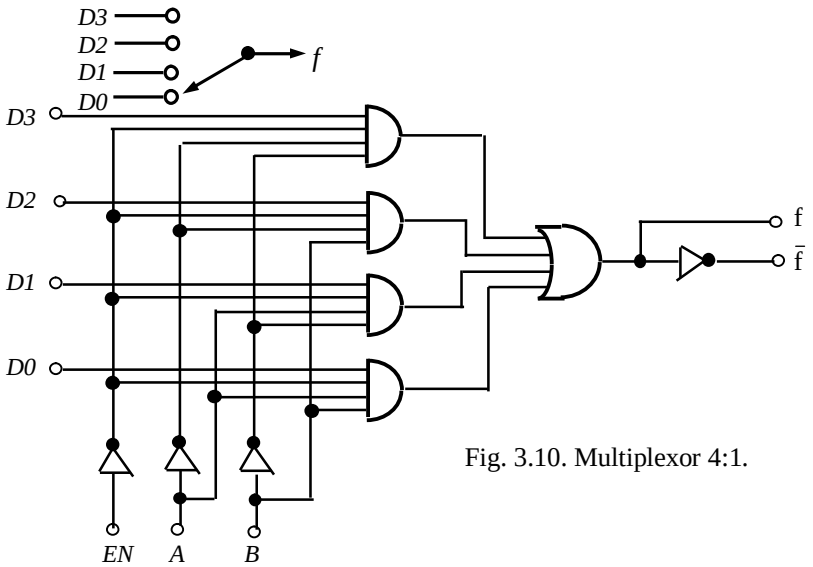
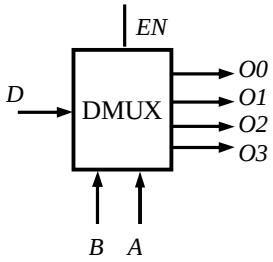


Fig. 3.10. Multiplexor 4:1.

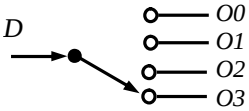
Demultiplexoarele sunt *CLC* care realizează transmiterea datelor de la o unică intrare către o ieşire selectabilă cu ajutorul unui cuvânt de selecţie (adresă), a cărei mărime se determină tot cu ajutorul relaţiilor (3.4) şi (3.5) ,în care *N* reprezintă numărul canalelor de ieşire. Ca şi *MUX* demultiplexoarele reprezintă practic o reţea de comutatoare comandate, putând fi numerice sau analogice. În figura 3.11 se prezintă elemente specifice sintezei unui *DMUX* numeric 1:4.



EN	*B*	*A*	*O0*	*O1*	*O2*	*O3*
1	0	0	D			
1	0	1		D		
1	1	0			D	
1	1	1				D
0	*	*	-	-	-	-

$$O0 = EN \cdot \overline{B} \cdot \overline{A} \cdot D \quad O2 = \overline{E} \cdot B \cdot \overline{A} \cdot D$$

$$O1 = EN \cdot \overline{B} \cdot A \cdot D \quad O3 = \overline{E} \cdot B \cdot A \cdot D$$



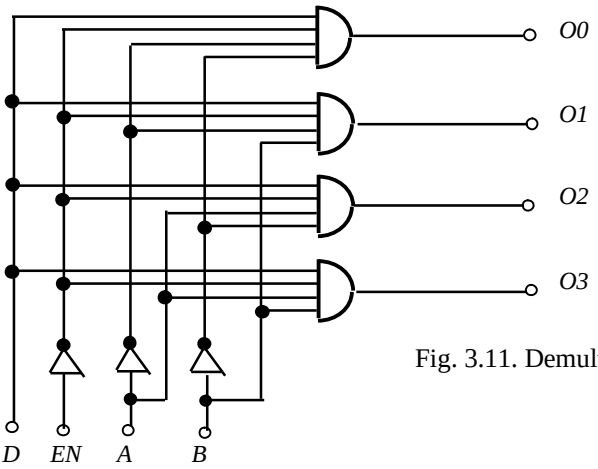


Fig. 3.11. Demultiplexor 1:4.

3.2.4. *Circuite de complementare*

Circuitul de complementare este un *CLC* care funcţie de comenzile aplicate poate realiza una dintre următoarele funcţii:

- complementarea faţă de unu a biţilor cuvântului de la intrare;
- lăsarea neschimbată cuvântul de la intrare;
- forţarea în unu a biţilor cuvântului de la ieşire;
- forţează în zero a biţilor cuvântului de la ieşire.

În figura 3.12 se prezintă elementele aferente unui circuit de complementare pe 4 biţi, pentru care din tabela de adevăr s-au obţinut următoarele funcţii logice ale ieşirilor:

$$y_1 = \overline{x}_1 \cdot \overline{A} \cdot \overline{B} + x_1 \cdot \overline{A} \cdot B + A \cdot \overline{B} = \overline{A} \cdot (\overline{x}_1 \cdot \overline{B} + x_1 \cdot B) + A \cdot \overline{B} = \overline{A} \cdot (\overline{x_1 \oplus B}) + A \cdot \overline{B}$$

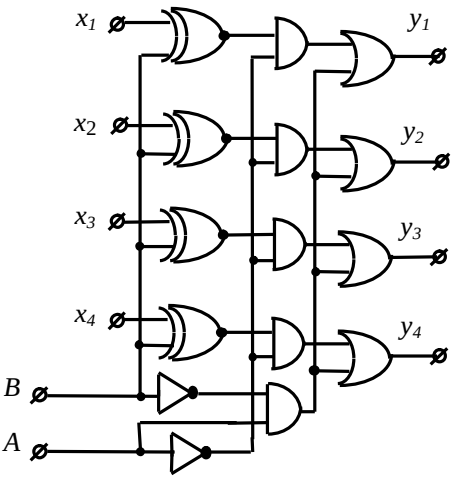
$$y_2 = \overline{x}_2 \cdot \overline{A} \cdot \overline{B} + x_2 \cdot \overline{A} \cdot B + A \cdot \overline{B} = \overline{A} \cdot (\overline{x}_2 \cdot \overline{B} + x_2 \cdot B) + A \cdot \overline{B} = \overline{A} \cdot (\overline{x_2 \oplus B}) + A \cdot \overline{B}$$

$$y_3 = \overline{x}_3 \cdot \overline{A} \cdot \overline{B} + x_3 \cdot \overline{A} \cdot B + A \cdot \overline{B} = \overline{A} \cdot (\overline{x}_3 \cdot \overline{B} + x_3 \cdot B) + A \cdot \overline{B} = \overline{A} \cdot (\overline{x_3 \oplus B}) + A \cdot \overline{B}$$

$$y_4 = \overline{x}_4 \cdot \overline{A} \cdot \overline{B} + x_4 \cdot \overline{A} \cdot B + A \cdot \overline{B} = \overline{A} \cdot (\overline{x}_4 \cdot \overline{B} + x_4 \cdot B) + A \cdot \overline{B} = \overline{A} \cdot (\overline{x_4 \oplus B}) + A \cdot \overline{B}$$

Comenzi		Ieşiri			
A	*B*	y_1	y_2	y_3	y_4
0	0	$\overline{x}_1$	$\overline{x}_2$	$\overline{x}_3$	$\overline{x}_4$
0	1	x_1	x_2	x_3	x_4
1	0	1	1	1	1
1	1	0	0	0	0

Fig. 3.12. Circuit de complementare pe 4 biţi.

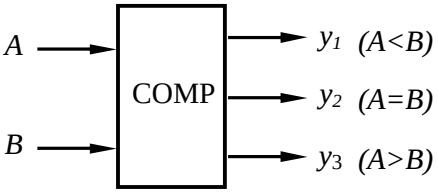


3.2.5. *Comparatoare*

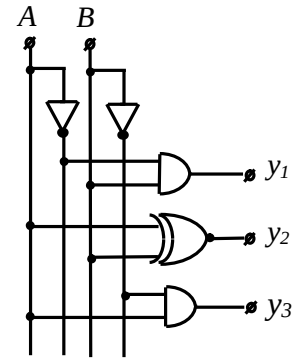
Comparatoarele numerice sunt *CLC* care permit determinarea relaţiei existente între două numere. Ieşirile unui comparator sunt

reprezentate de *trei funcţii* care corespund tipului de relaţie existent între numerele aplicate la intrare *(<,=,>)*.

În figura 3.13 sunt prezentate elemente aferente sintezei unui comparator pe un bit.



A	*B*	y_1	y_2	y_3
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0



$$y_1 = \overline{A} \cdot B$$
$$y_2 = \overline{A} \cdot \overline{B} + A \cdot B = \overline{A \oplus B}$$
$$y_3 = A \cdot \overline{B}$$

Fig. 3.13. Comparator pe un bit.

Prin interconectarea mai multor comparatoare pe un bit se pot obţine comparatoare pentru cuvinte binare formate din mai mulţi biţi.

3.2.6. *Detectoare de paritate*

Detectoarele de paritate sunt *CLC* cu *n* intrări şi două ieşiri *PAR* şi *IMPAR* care sunt complementare. Ieşirea *PAR* are valoarea **1** atunci când numărul de valori logice **1** în combinaţia de la intrare este *par* şi **0** atunci când acest număr este *impar*.

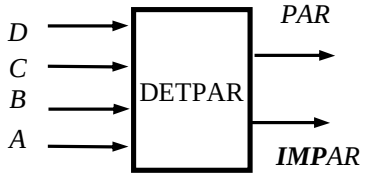
În figura 3.14 se prezintă elementele aferente sintezei unui detector de paritate cu *n=4* intrări.

După cum se observă în tabela de adevăr funcţiile *PAR* şi *IMPAR* sunt complementare, respectiv $IMPAR = \overline{PAR}$. Din această cauză în figura 3.14 a fost reprezentată diagrama Karnaugh pentru funcţia *PAR*. Aşa cum

reiese din diagramă, nu se poate opera nici o reducere asupra funcţiei care va fi:

$$PAR = \overline{D}\,\overline{C}\,\overline{B}\overline{A} + \overline{D}\,\overline{C}BA + \overline{D}C\overline{B}A + \overline{D}CB\overline{A} + D\overline{C}\,\overline{B}A + D\overline{C}B\overline{A} + DC\overline{B}\overline{A} + DCBA$$

În relaţia de mai sus prin aplicarea proprietăţilor operaţiilor logice rezultă:



a

DC \ BA	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

c

D	*C*	*B*	*A*	*PAR*	*IMP*
0	0	0	0	1	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	0	1
1	0	0	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	1	0

b

Fig. 3.14. Detector de paritate: a – schema bloc; b - tabela de adevăr; c - diagrama Karnaugh a funcţiei *PAR*.

$$PAR = \overline{D}\,\overline{C}(\overline{B}\overline{A} + BA) + \overline{D}C(\overline{B}A + B\overline{A}) + D\overline{C}(\overline{B}A + B\overline{A}) + DC(\overline{B}\overline{A} + BA)$$

$$PAR = (\overline{D}\,\overline{C} + DC)(\overline{B}\overline{A} + BA) + (\overline{D}C + D\overline{C})(\overline{B}A + B\overline{A})$$

Dar

$$\overline{D}\overline{C} + DC = \overline{\overline{\overline{D}\overline{C} + DC}} = \overline{(\overline{\overline{D}\overline{C}})\cdot(\overline{DC})} = \overline{(\overline{\overline{D}} + \overline{C})\cdot(\overline{D} + \overline{C})} = \overline{(D + C)\cdot(\overline{D} + \overline{C})} =$$

$$= \overline{D\overline{C} + \overline{D}C} = \overline{D \oplus C}$$

$$\overline{BA} + BA = \overline{B \oplus A}$$

Notăm

$$D \oplus C = C \oplus D = Y$$

$$B \oplus A = A \oplus B = X$$

Rezultă:

$$PAR = \overline{YX} + YX = \overline{Y \oplus X} = \overline{(D \oplus C) \oplus (B \oplus A)}$$

$$IMPAR = \overline{PAR} = (D \oplus C) \oplus (B \oplus A)$$

relaţii a căror implementare se prezintă în figura 3.15.

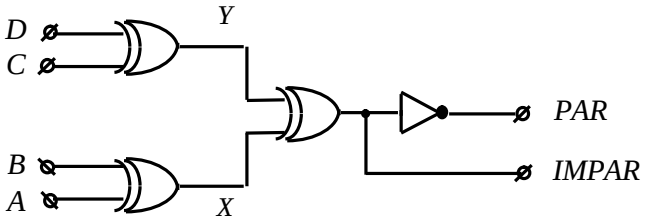


Fig. 3.15. Logigrama pentru detectorul de paritate.

3.2.7. *Sumatoare*

Semisumatorul elementar, pentru care schema logică şi tabela de adevăr sunt prezentate în figura 3.16, adună două numere a câte un bit x_i şi y_i şi generează la ieşire 2 biţi: suma S_i şi transportul c_i către rangul următor.

x_i	y_i	S_i	c_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

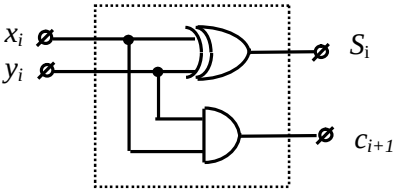


Fig. 3.16. Semisumatorul elementar.

Schema din figura 3.16 a rezultat pe baza relaţiilor:

$$S_i = x_i \cdot \overline{y}_i + \overline{x}_i \cdot y_i = x_i \oplus y_i$$

$$c_{i+1} = x_i \cdot y_i \quad .$$

Sumatorul elementar este un *CLC* care adună două numere binare x_i, y_i cu un transport de intrare c_i, generând la ieşire doi biţi: suma S_i şi transportul c_{i+1} către rangul superior, conform tabelului 3.6.

Tabelul 3.6

x_i	y_i	c_i	S_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Din tabelul 3.6 rezultă:

$$S_i = \overline{x}_i \cdot \overline{y}_i \cdot c_i + \overline{x}_i \cdot y_i \cdot \overline{c}_i + x_i \cdot \overline{y}_i \cdot \overline{c}_i + x_i \cdot y_i \cdot c_i$$

$$c_{i+1} = \overline{x}_i \cdot y_i \cdot c_i + x_i \cdot \overline{y}_i \cdot c_i + x_i \cdot y_i \cdot \overline{c}_i + x_i \cdot y_i \cdot c_i$$

sau după efectuarea unor calcule:

$$S_i = x_i \oplus (y_i \oplus c_i)$$

$$c_{i+1} = x_i \cdot y_i + c_i (x_i \oplus c_i).$$

Relaţiile de mai sugerează obţinerea sumatorului elementar din două semisumatoare conform figura 3.17.

Pentru adunarea a două cuvinte de *n* biţi este necesar să se înserieze *n* astfel de sumatoare ca în figura 3.18.

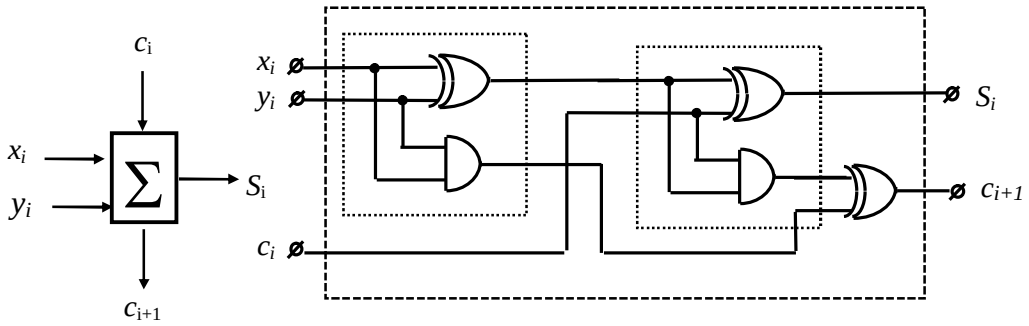


Fig. 3.18. Sumatorul elementar

Cele două numere care urmează a se aduna se găsesc în registrele *X* şi *Y*, iar rezultatul în registrul *S*. Transportul este depus într-un bistabil exterior care pentru un microprocesor este indicatorul de transport *CY* (Carry).

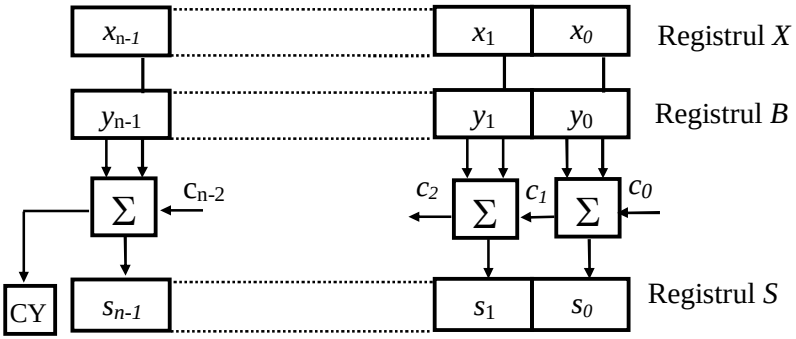


Fig. 3.18. Sumator pentru cuvinte de n biţi.

3.3. Proiectarea structurilor secvenţiale

Circuitele logice secvenţiale *(CLS)* sunt circuite ale căror mărimi de ieşire, la un moment dat, depind atât de combinaţia mărimilor de intrare, cât şi de starea sa, respectiv de succesiunea stărilor prin care respectivul circuit a trecut. Pornind de la această definiţie rezultă că la un *CLS* (spre deosebire de un *CLC)* există legături de reacţie de la intrări către ieşiri şi în plus nu se mai neglijează timpul de transfer al semnalelor (întârzierile), nici pe calea directă nici pe cea de reacţie. Aşa cum reiese din figura 3.19 un *CLS* conţine în structura sa un *CLC* şi un grup de elemente de întârziere Δ_0, $\Delta_1 \ldots \Delta_{p-1}$ situate pe calea de reacţie.

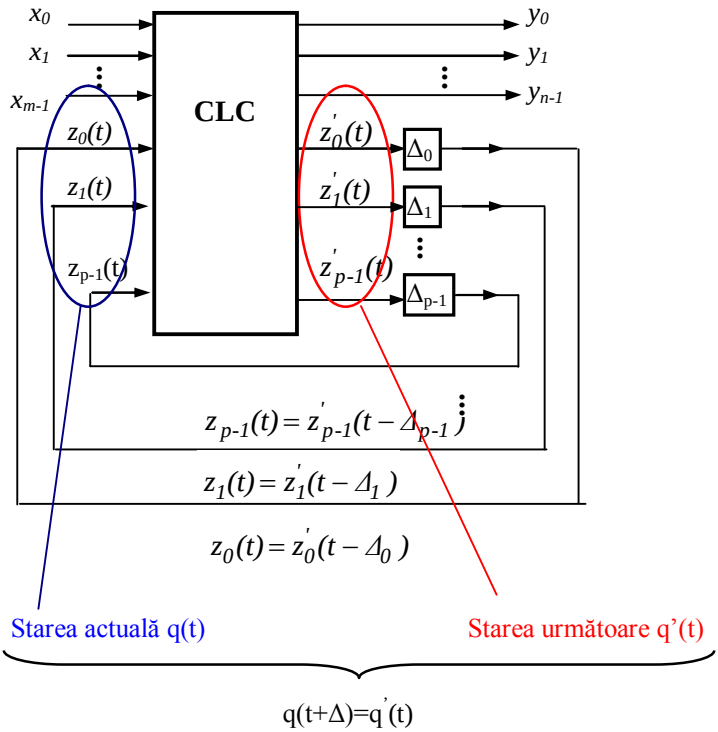


Fig. 3.19. Structura unui *CLS* asincron.

La un moment oarecare de timp *t,* circuitului *CLS* i se asociază *starea* sa internă care poate fi exprimată de valorile variabilelor secundare de intrare în *CLC* respectiv $z_0(t)$, $z_1(t)$,…, $z_{p-1}(t)$. Cuvântul format din cele *p* variabile secundare de intrare la un moment dat (*din mulţimea de* 2^p*cuvinte)* reprezintă cuvântul de stare *q(t) al CLS* sau starea curentă *(actuală)* a acestuia.

Intrările secundare $z_0(t)$, $z_1(t)$,…, $z_{p-1}(t)$ reprezintă ieşirile secundare $z'_{p-1}, ...,z'_1,z'_0$ întârziate cu timpii $\Delta_{p-1}, ...\Delta_1,\Delta_0$. Considerând întârzierile Δ_i egale, se poate spune că starea internă curentă va deveni stare următoare după un timp Δ respectiv $q(t+\Delta)=q'(t)$.

Pe baza reprezentării din figura 3.19 se pot defini secţiunile *modelului matematic* pentru un moment *t,* aferent unui *CLS* după cum urmează:

- *funcţii asociate tranziţiei ieşirilor,* care reflectă transferul semnalelor de la intrare la ieşire în funcţie de starea curentă respectiv:

$$y_0 = g_0(x_0,x_1,x_2,...,x_{m-1},z_0,z_1,z_2,...,z_{p-1});$$

$$y_1 = g_1(x_0,x_1,x_2,...,x_{m-1},z_0,z_1,z_2,...,z_{p-1});$$

……

$$y_{n-1} = g_1(x_0,x_1,x_2,...,x_{m-1},z_0,z_1,z_2,...,z_{p-1}). \qquad (3.6)$$

- *funcţii asociate tranziţiei stărilor,* care reflectă transferul stării actuale *q(t)* pentru un anumit cuvânt de intrare respectiv:

$$z'_0 = f_0(x_0,x_1,x_2,...,x_{m-1},z_0,z_1,z_2,...,z_{p-1});$$

$$z'_1 = f_1(x_0,x_1,x_2,...,x_{m-1},z_0,z_1,z_2,...,z_{p-1});$$

…… (3.7)

$$z'_{p-1} = f_1(x_0,x_1,x_2,...,x_{m-1},z_0,z_1,z_2,...,z_{p-1}).$$

Sintetic modelul matematic al unui *CLS* este reprezentat de cvintuplul

$$C_s = (X,Y,Q,f,g), \qquad (3.8)$$

unde - $X = \{x_0, x_1, x_2, ..., x_{m-1}\}$ este mulţimea variabilelor binare de intrare;

- $Y = \{y_0, y_1, y_2, ..., y_{n-1}\}$ este mulţimea variabilelor binare de ieşire;
- $Q = \{q_0, q_1, q_2, ..., q_{2^p-1}\}$ este mulţimea stărilor *CLS*;
- $f : X \times Q \to Q$ este funcţia de tranziţie a stărilor – relaţia (3.7);
- $g : X \times Q \to Y$ este funcţia de tranziţie a ieşirilor– relaţia (3.6).

Un *CLS* de tipul celui prezentat la care starea *următoare* devine stare curentă numai după un timp Δ determinat de întârzierile inerente ale *CLC* se numeşte *CLS asincron (CLSA).* Deoarece mulţimea funcţiilor (respectiv a cuvintelor) care se pot forma pe cele trei mulţimi este finită, *CLS* se numeşte cu stări finite. Un neajuns al *CLSA* este reprezentat de faptul că acestea pot fi uneori instabile.

Acest neajuns este întotdeauna eliminat dacă pe linia de reacţie întârzierea Δ este înlocuită cu un circuit de memorie, potrivit reprezentării din figura 3.20. Această memorie va fi citită la intervale de timp , adică va introduce o întârziere, care poate fi controlată cu T – *perioada generatorului de tact.* Un *CLS* la care starea următoare $q'(t)$ devine stare prezentă numai după aplicarea impulsului de ceas, respectiv $q(t+T) = q'(t)$ se numeşte *CLS sincron (CLSS).*

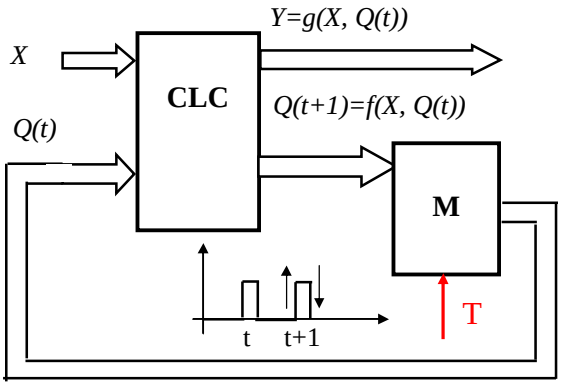


Fig. 3.20. Structura unui *CLS* sincron.

După cum se observă din figura 3.20, pentru mulţimea cuvintelor stării următoare se utilizează notaţia *Q(t+1),* care semnifică faptul că un

cuvânt din mulţimea stărilor actuale *Q(t)* devine cuvântul stării următoare numai după trecerea *unei unităţi de timp respectiv a unei perioade de ceas*[2] Tranziţia stării *viitoare în curentă* se poate realiza fie pe frontul crescător, fie pe cel descrescător al impulsului de tact, fronturi notate în figura 3.20 cu săgeţi orientate în sus respectiv în jos.

Eliminarea *hazardului combinaţional* [3] specific structurii din figura 3.20 se realizează prin introducerea unei memorii şi pentru ieşiri, astfel încât generarea lui *Y* să se facă tot numai după momentele de tact *kT (k=1, 2, 3,…).* Structura obţinută, ilustrată în figura 3.21, este cunoscută sub denumirea de *automat secvenţial de tip Mealy.*

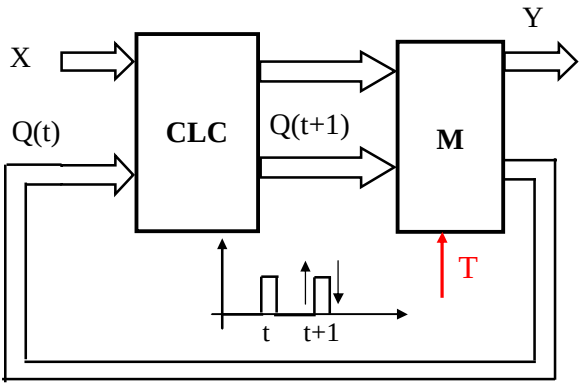


Fig. 3.21. Structura unui automat secvenţial de tip *Mealy.*

În situaţia în care ieşirile sunt funcţii numai de stările actuale respectiv,

$$Y = g(Q)\,, \qquad (3.9)$$

se obţine *CLSS* cu structura ilustrată în figura 3.22 cunoscut ca *ca automat secvenţial de tip Moore.* Este de menţionat faptul că că există algoritmi care permit transformarea în ambele sensuri a celor două tipuri de automate însă prezentarea lor nu face obiectul prezentei lucrări.

---

[2] Se obişnuieşte ca această notaţie să se utilizeze şi pentru celelalte mulţimi X respectiv Y.

[3] Acest hazard se datorează faptului că propagarea prin CLC nu este instantanee şi prin urmare, cuvântul corect Y la ieşire apare numai după trecerea celui mai lung timp de propagare prin reţeaua combinaţională. Cuvintele care apar anterior acestui moment sunt considerate eronate.

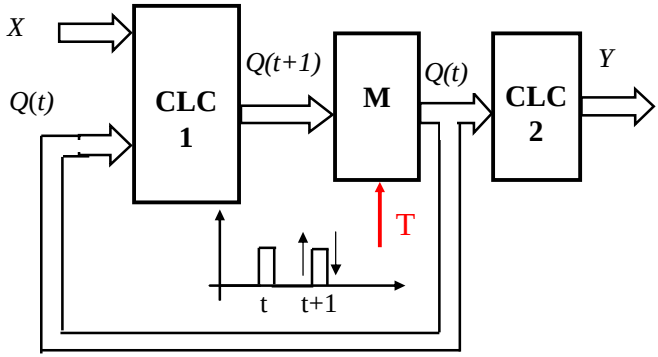


Fig. 3.22. Structura unui automat secvenţial de tip *Moore*.

Există mai multe modalităţi de descriere a evoluţiei unui *CLS* cu modelul (3.8), dar o importanţă aparte prezintă *tabelele de tranziţii, grafurile de tranziţii şi organigramele* [4].

- Un *tabel de tranziţii* exprimă toate transformările stării următoare şi ale ieşirii, relaţiile (3.7) şi (3.6) sub formă tabelară. În figura 3.23 sunt ilustrate asemenea reprezentări pentru un automat *Mealy* respectiv *Moore*.

X / Q	x_0	...	x_{m-1}
q_0	$f(x_0,q_0)$; $g(x_0,q_0)$	...	$f(x_{m-1},q_0)$; $g(x_{m-1},q_0)$
q_1	$f(x_0,q_1)$; $g(x_0,q_1)$	...	(x_{m-1},q_1); $g(x_{m-1},q_1)$
...	...	...	...
$q_2^p{}_{-1}$	$f(x_0,q_2^p{}_{-1})$; $g(x_0,q_2^p{}_{-1})$	...	$f(x_{m-1},q_2^p{}_{-1})$; $g(x_{m-1},q_2^p{}_{-1})$

a)

X / Q	x_0	...	x_{m-1}	Y
q_0	$f(x_0,q_0)$; $g(x_0,q_0)$	...	$f(x_{m-1},q_0)$ $g(x_{m-1},q_0)$	$g(q_0)$
q_1	$f(x_0,q_1)$; $g(x_0,q_1)$	...	(x_{m-1},q_1) $g(x_{m-1},q_1)$	$g(q_1)$
...	...	...	...	...
$q_2^p{}_{-1}$	$f(x_0,q_2^p{}_{-1})$ $g(x_0,q_2^p{}_{-1})$	...	$f(x_{m-1},q_2^p{}_{-1})$ $g(x_{m-1},q_2^p{}_{-1})$	$g(q_2^p{}_{-1})$

b)

Fig. 3.23. Reprezentarea cu ajutorul *tabelelor de tranziţii* a automatelor sincrone: a – de tip *Mealy*, b – de tip *Moore*.

La automatul *Mealy* – figura 3.23 a pe prima linie se înscriu toate mărimile de intrare (respectiv x_i) , iar pe prima coloană toate stările

[4] Acestora li se mai pot adăuga şi diagramele de timp ale semnalelor aferente *CLS*.

circuitului (respectiv q_j). La intersecţia unei coloane corespunzătoare unei intrări x_i cu o linie corespunzătoare unei stări q_j se înscrie starea următoare dată de funcţia *f* respectiv ieşirea dată de funcţia *g*. Dacă *CCLSS* este de tip *Moore* atunci ieşirea este dependentă numai de stare şi *tabelul de tranziţii* arată ca în figura 3.23 b.

- Un *graf de tranziţii* constituie o reprezentare grafică a modelului matematic al unui *CLS*. Stările circuitului sunt trecute în cerculeţe care sunt nodurile grafului. Între două stări q_j şi q_k există un arc orientat, de la q_j la q_k dacă şi numai dacă există o mărime de intrare x_i astfel încât $f(x_i, q_j) = q_k$ şi se obţine o mărime de ieşire $y_i = g(x_i, q_j)$. Dacă se îndeplineşte condiţia pe arc se înscrie perechea x_i, y_i şi graful obţinut corespunde unui automat *Mealy*. La automatele *Moore* întrucât ieşirile depind numai de stări acestea nu se mai înscriu pe arce ci în interiorul cerculeţelor aferente nodurilor. Pentru exemplificare, în figura 3.24 se prezintă două exemple de grafuri pentru cele două tipuri de automate.

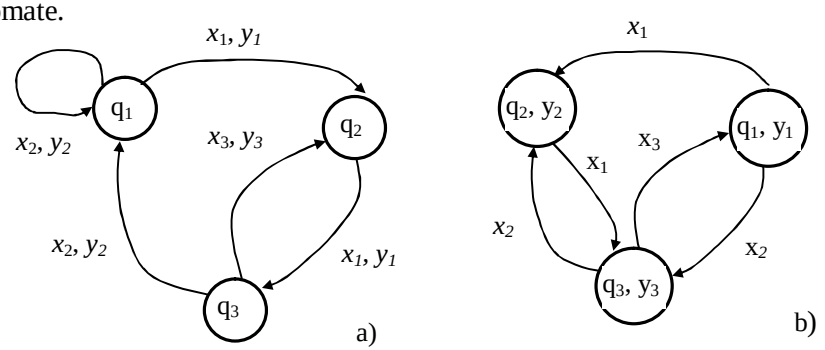


Fig. 3.24. Reprezentarea cu ajutorul *grafurilor de tranziţii* a automatelor sincrone: a – de tip *Mealy*, b – de tip *Moore*.

- Utilizarea *organigramelor* permite o reprezentare eficientă e evoluţiei unui automat secvenţial. Este posibilă evidenţierea tranziţiei între stări cu materializarea intrărilor şi ieşirilor funcţie de realizarea unor condiţii. Pentru exemplificare în figura 3.25 se prezintă fragmente de scheme logice pentru câte un automat *Mealy* respectiv *Moore*.

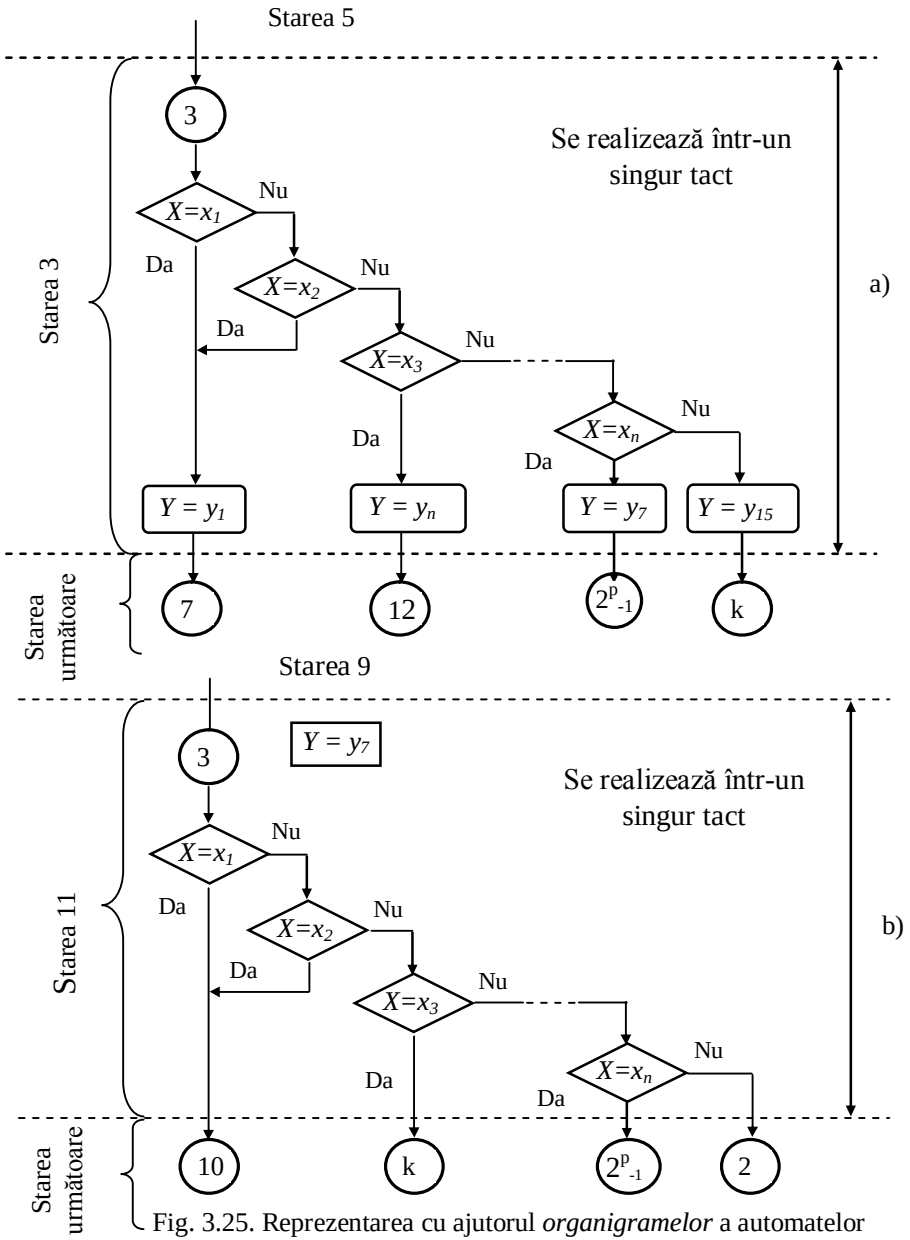


Fig. 3.25. Reprezentarea cu ajutorul *organigramelor* a automatelor sincrone: a – de tip *Mealy*, b – de tip *Moore*.

De exemplu pentru automatul *Mealy* (figura 3.25 a) la aplicarea unui impuls de tact se trece din starea 5 în starea 3. În această stare se testează cuvântul de intrare X , iar în funcţie de configuraţia biţilor acestuia rezultă un anumit cuvânt de ieşire şi o anumită stare următoare. Astfel pentru un cuvânt de intrare $X=x_3$ rezultă ieşirea $Y = y_n$ şi starea următoare *12* la care se trece însă la următorul impuls de ceas.

La automatul *Moore* – figura 3.25 b, când se aplică un impuls de tact se trece din *starea 9* în *starea 11,* stare în care se generează independent de cuvântul de intrare ieşirea $Y = y_7$. În funcţie de structura cuvântului de intrare X la următorul impuls de tact se trece la starea următoare.

Din cele expuse se desprinde concluzia că *CLC* este un caz particular de *CLS* la care mulţimea stărilor din relaţia (3.8) este mulţimea vidă respectiv $Q = \Phi$. În aceste condiţii funcţia de tranziţie a stărilor dispare, iar cea de tranziţie a ieşirilor devine

$$g : X \to Y , \tag{3.10}$$

relaţie care arată că domeniul de definiţie al funcţiei g se reduce la spaţiul intrărilor.

Cu aceste precizări se poate spune că modelul matematic al unui *CLC* este reprezentat de tripletul

$$C_C = (X, Y, g), \tag{3.11}$$

care exprimă faptul că *un CLS* cu mulţimea stărilor vidă se reduce la un *CLC.* În continuare vor fi prezentate elemente care privesc *CLS* cu o prezenţă semnificativă în structura oricărui *CN.*

3.3.1. Circuite basculante bistabile

Circuitele basculante bistabile[5] *(CBB)* au două stări stabile la ieşire, iar prin aplicarea unor semnale de comandă trec dintr-o anume stare în starea complementară. *CBB* sunt *CLS* realizate pe baza unor structuri fundamentale de tipul celor prezentate în figura 3.26, completate cu alte elemente în vederea obţinerii unei comportări specifice. Circuitul este format din două porţi *NOR* sau *NAND (partea de CLC)* între care există încrucişate prin intermediul întârzierilor Δ_1 şi Δ_2 asociate propagărilor prin porţi. Aceste reacţii realizează practic legătura între starea următoare şi cea

[5] Trigger – în limba engleză

curentă. Biţii cuvântului de ieşire y_1, y_2 se notează cu $\overline{Q}$ respectiv Q şi corespund biţilor stării curente z_1 *şi* z_2 . Deoarece $y_1 = z_1 = \overline{Q}$ şi $y_2 = z_2 = Q$ rezultă că *CBB* este un automat de tip *Moore*.

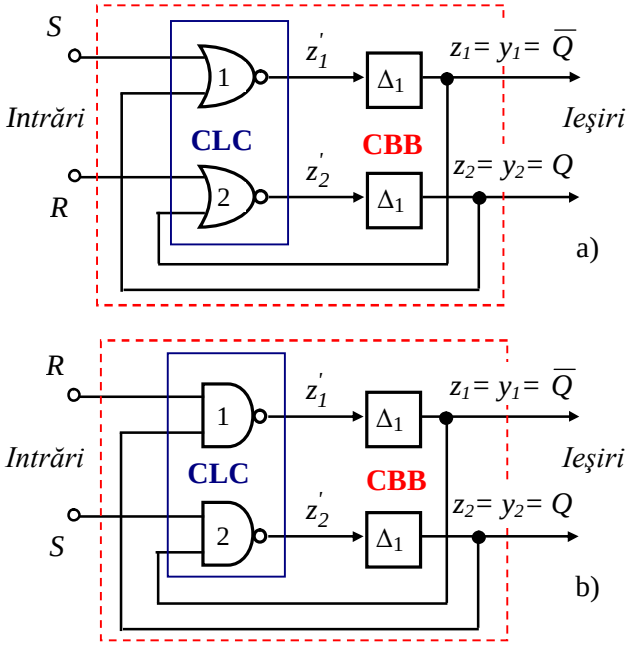


Fig. 3.26. *CBB* ca structuri fundamentale de *CLS:* a - cu porţi *NOR*, b – cu porţi *NAND*.

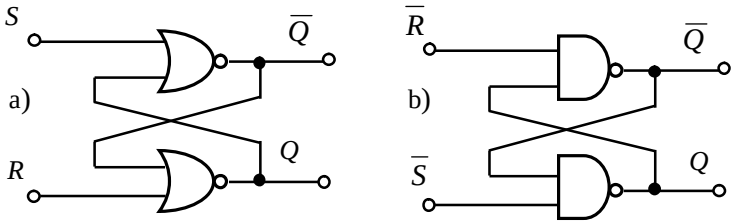
Explicarea funcţionării *CBB* cu structurile din figura 3.26 se va face ţinându-se cont de tabelele de adevăr ale funcţiilor *NOR* şi *NAND* [6]. Să presupunem că la intrarea principală de scriere *S* a porţii 1 din varianta *a* se aplică ***1*** iar la intrarea principală de ştergere *R* a porţii 2 se aplică ***0***. După un timp Δ_1 ieşirea $\overline{Q}$ va deveni *0* şi se va aplica la intrarea secundară a porţii *2*. Dacă ieşirea *Q* a acesteia nu este ***1***, atunci după un timp Δ_2 va deveni ***1*** (deoarece la ambele intrări se aplică *0*). Ieşirea porţii 2 se va aplica pe intrarea secundară a porţii 1, *întărind* comanda *S=1*. Din acest moment

[6] Singurele combinaţii care vor genera ***1*** respectiv ***0*** la ieşirile unor porţi *NOR* respectiv ***NAND*** vor fi ***00*** respectiv ***11.***

(deci după $t > \Delta_1 + \Delta_2$[7] de la aplicarea *S=1)* intrarea se poate anula respectiv se poate face *S=0,* ieşirile rămânând neschimbate. Se observă că parcurgând un contur *de forma cifrei opt* se obţine un semnal de reacţie care întăreşte sau poate substitui comanda dată, starea circuitului fiind *Q=1* ,*(*$\overline{Q}$ =0). Această reacţie pozitivă (un defazaj de 360 între cele două intrări – principală şi secundară- ale unei porţi se obţine datorită negaţiilor şi fiecăreia dintre porţi şi a legăturilor încrucişate..

În mod similar se poate explica şi funcţionarea *CBB* realizat cu porţi *NAND* din figura 3.26 b cu observaţia că fixarea ieşirii *Q=1* se va face prin *S=0* (deoarece o poartă *NAND* va avea ieşirea în ***1*** când una dintre intrări este în ***0***. Deoarece la acest *CBB* ieşirile sunt active în ***0*** şi nu în ***1*** (ca la cel cu porţi *NOR*) cele două intrări se mai notează $\overline{S}$ şi $\overline{R}$.

Din prezentarea modului de funcţionare, se poate spune că practic un *CBB* implementează un element de memorie care păstrează *un bit* de informaţie. Cele două structuri analizate sunt reprezentate în mod curent ca în figura 3.27, în care întârzierile se presupun fără a fi evidenţiate.



R	S	Q_t	Q_{t+1}	$\overline{Q}_{t+1}$
0	0	0	**0**	1
0	0	1	**1**	0
0	1	0	**1**	0
0	1	1	**1**	0
1	0	0	**0**	1
1	0	1	**0**	1
1	1	0	*	*
1	1	1	*	*

$\overline{R}$	$\overline{S}$	Q_t	Q_{t+1}	$\overline{Q}_{t+1}$
0	0	0	*	*
0	0	1	*	*
0	1	0	**0**	1
0	1	1	**0**	1
1	0	0	**1**	0
1	0	1	**1**	0
1	1	0	**0**	1
1	1	1	**1**	0

Fig. 3.27. Structuri fundamentale de *CBB* şi tabele de excitaţie*:* a - cu porţi *NOR,* b – cu porţi *NAND.*

[7] În mod obişunit întârzierile de propagare sunt egale astfel încât condiţia se va scrie $t > 2\Delta$.

Ţinând cont de tabelul de adevăr din figura 3.27a rezultă următoarele caracteristici pentru *CBB* realizat cu porţi *NOR:*

a) aplicarea semnalului *zero* la ambele intrări nu modifică starea bistabilului;

b) aplicarea semnalului $S=1$ simultan cu $R=0$ determină $Q_{t+1}=1$ indiferent de starea anterioară Q_t (această trecere a *CBB* în starea $Q=1$ se numeşte *poziţionare – **Set** în limba engleză)*;

c) aplicarea semnalului $R=1$ simultan cu $S=0$ determină $Q_{t+1}=0$ indiferent de starea anterioară Q_t (această trecere a *CBB* în starea $Q=0$ se numeşte *ştergere – **Reset** în limba engleză)*;

d) aplicarea simultană a semnalelor $R=S=1$ generează o *stare nedeterminată* şi de aceea această combinaţie este interzisă prin impunerea condiţiei $R \cdot S = 0$ astfel încât cel puţin o intrare să fie *zero*.

Similar poate fi analizat *CBB* realizat cu porţi *NAND* a cărui schemă şi tabelă de adevăr sunt reprezentate în figura 3.79 b, pe baza căruia pot fi formulate următoarele caracteristici:

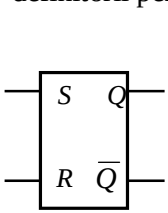
e) aplicarea semnalului *unu* la ambele intrări nu modifică starea bistabilului;

f) aplicarea semnalului $\overline{R}=1$ simultan cu $\overline{S}=0$ determină $Q_{t+1}=1$ indiferent de starea anterioară Q_t (această trecere a *CBB* în starea $Q=1$ se numeşte *poziţionare – **Set** în limba engleză)*;

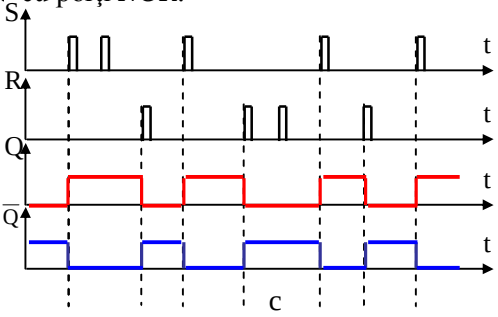
g) aplicarea semnalului $\overline{R}=0$ simultan cu $\overline{S}=1$ determină $Q_{t+1}=0$ indiferent de starea anterioară Q_t (această trecere a *CBB* în starea $Q=0$ se numeşte *ştergere – **Reset** în limba engleză)*;

h) aplicarea simultană a semnalelor $\overline{R}=\overline{S}=0$ generează o *stare nedeterminată* şi de aceea această combinaţie este interzisă prin impunerea condiţiei $\overline{R}+\overline{S}=1$ astfel încât cel puţin o intrare să fie *unu*.

CBB analizat este cunoscut sub numele bistabil *RS asincron* denumit şi *RS latch*[8] , în figura 3.28 prezentându-se unele elemente definitorii pentru varianta realizată cu porţi *NOR*.



R	*S*	*Q*
0	0	Nemod.
0	1	**1**
1	0	**0**
1	1	Ned.



a b c

Fig. 3.28. Circuit bistabil *RS asincron:* a – simbol; b – tabelă de adevăr simplificată; c – diagramă de semnale.

O altă variantă de bistabil *RS* este cea sincronă la care pe lângă intrările de date *R* şi *S* este prezentă şi o intrare de sincronizare (*tact, clock*) notată cu *CLK*. La un *CBB RS sincron* starea *Q(t)* se consideră anterioară aplicării impulsului de sincronizare iar *Q(t+1)*posterioară acestuia. Este de menţionat că la *CBB asincron* starea următoare poate fi considerată cea obţinută după un timp $t \geq 2\Delta$ de la aplicarea unui semnal pe intrările de date.

La un *CBB* sincron datele se pot transfera de la intrare la ieşire *pe frontul sau pe palierul impulsului de sincronizare.* La comutarea pe front datele se pot transfera pe frontul crescător (simbol ▷ la borna *CLK*) sau pe front descrescător (simbol ○▷ la borna *CLK*) . Înaintea frontului pozitiv al impulsului de tact şi după frontul negativ al acestuia sunt specificaţii timpii de *prestabilire* [9]şi de *menţinere* [10] în care nu trebuie să se schimbe datele pe intrări. La *CBB* cu *comutare pe palier* datele sunt transferate intervalul de timp în care impulsul de tact are valoarea *unu.*

O structură eficientă de *CBB* este reprezentată de structura *Master – Slave,* ilustrată în figura 3.29 a. Un circuit *RS master – slave* este format din două bistabile *RS* dintre Q_M cu rol de *Master (stăpân)* şi Q_S cu rol de

[8] Latch – zăvor (engl)
[9] Set-up time (engl) > 20 ns
[10] Hold time (engl) > 5 ns

Slave (sclav). Bistabilul Q_M este validat pe frontul pozitiv al impulsului de sincronizare, iar Q_S pe frontul negativ al acestuia. Cu alte cuvinte, pe frontul pozitiv datele sunt transferate la ieşirea Q_M , iar pe frontul negativ la ieşirea Q_S, aşa cum reiese şi din figura 329 b, în care se prezintă o diagramă de timp pentru un bistabil *RS sincron de tip MS.*

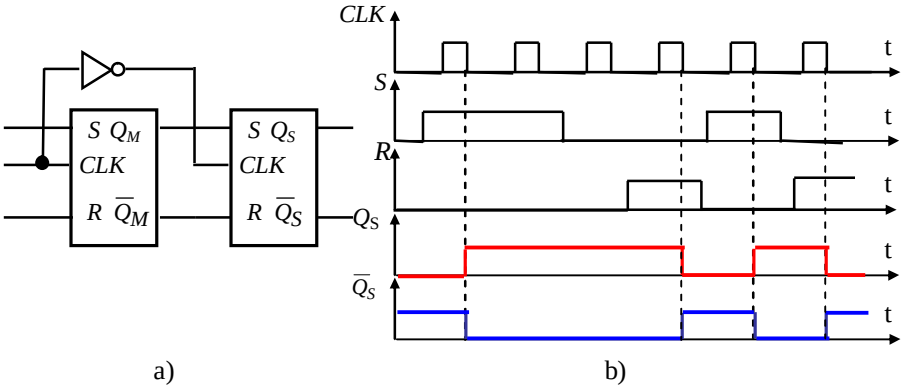
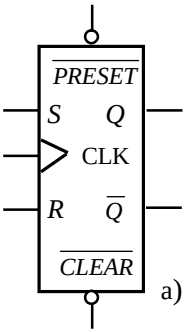


Fig. 3.29. Circuit bistabil *RS sincron MS:* a – structură;
b – diagramă de semnale.

Bistabilele sincrone, indiferent de tip, sunt prevăzute în general cu *intrări prioritare* de scriere (*PRESET)* respectiv de ştergere *(CLEAR).* Aceste intrări sunt asincrone şi modifică ieşirile, neutralizând intrările de date şi de tact. De regulă aceste intrări sunt active în starea *zero* , aspect evidenţiat şi de simbolizarea din figura 3.30.



S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Ned.

b)

Q(t)	S	R	Q(t+1)
0	0	x	0
0	1	0	1
1	0	1	0
1	x	1	1

x – indiferent 0 sau 1

c)

$\overline{PRESET}$	$\overline{CLEAR}$	Q	$\overline{Q}$
0	0	Interzis	
0	1	1	0
1	0	0	1
1	1	Nemod.	

d)

Fig. 3.30. Circuit bistabil *RS sincron* cu basculare pe front crescător*:* a – simbol;
b – tabelă de adevăr simplificată; c-tabelă de excitaţie; d – tabelă de adevăr pentru intrări prioritare.

În figurile 3.27 şi 3.30 apar *tabele de excitaţie* care indică valorile semnalelor ce trebuie aplicate la intrările de date [11] pentru a se realiza o trecere dorită de la starea *Q(t)* la o stare *Q(t+1).*

În afara bistabilelor *RS* o largă utilizare cunosc *CBB* de tip *JK, D şi T,* care vor fi tratate în cele ce urmează

- Bistabilul JK provine dintr-un circuit *RS* la care se adaugă două bucle de reacţie totală prin intermediul unor porţi *AND.* În figura 3.83, se prezintă structura logică a unui *CBB* de tip *JK* bazat pe porţi *NOR* în variantă asincronă şi sincronă.

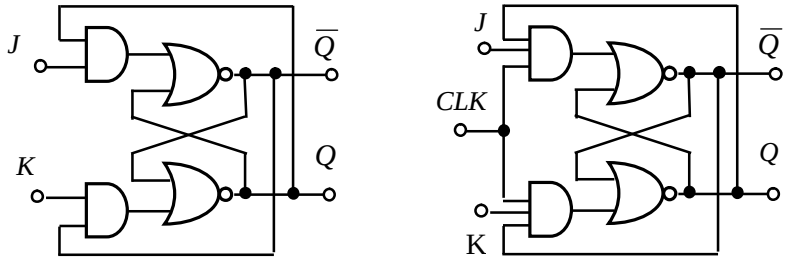
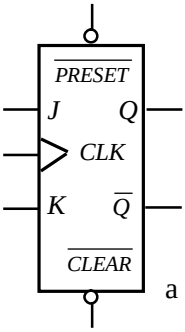


Fig. 3.31. Structura logică a bistabilului *JK:* a – varıanta asincronă; b – varianta sincronă .

Principalul avantaj al acestui *CBB* constă în eliminarea nedeterminării la combinaţia de intrare ***11.***

Intrările de date sunt *J (pentru scriere)* şi *K (pentru ştergere)* cărora li se pot adăuga intrările asincrone prioritare *PRESET* şi *CLEAR* active în *zero.* Tabelul de adevăr pentru intrările prioritare, figura 3.32b, ale bistabilului *JK* coincide cu cel al bistabilului *RS* (figura 3.30d).

[11] Cu validarea prin impulsul de tact la *CBB* sincrone şi fără impuls de tact la *CBB* asincrone.



a

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q}$(t)

b

J	K	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

c

Q(t)	J	K	Q(t+1)
0	0	x	0
0	1	0	1
1	0	1	0
1	x	1	1

x – indiferent 0 sau 1

d

Fig. 3.32. Circuit bistabil *JK sincron:* a – simbol; b – tabelă de adevăr; c-tabelă de excitaţie dezvoltată; d – tabelă de excitaţie redusă.

Se observă că primele trei linii ale tabelei de adevăr coincid cu primele trei linii ale tabelei de adevăr pentru bistabilul *RS* (figura 3.30 b). Pentru combinaţia ***11,*** la aplicarea impulsului de tact bistabilul *JK* basculează în starea opusă $\overline{Q}(t)$. Explicarea funcţionării bistabilului *JK* se poate face şi cu ajutorul diagramelor de timp prezentate în figura 3.33.

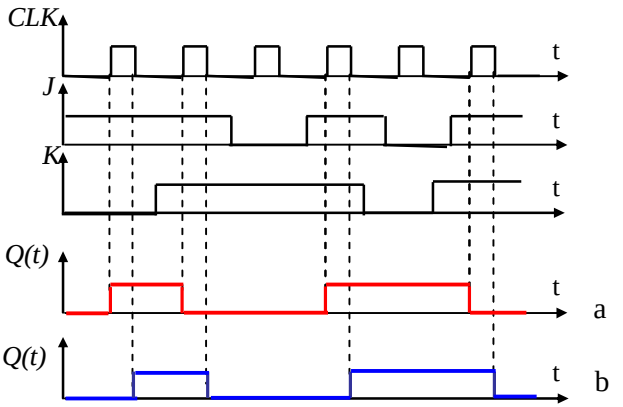
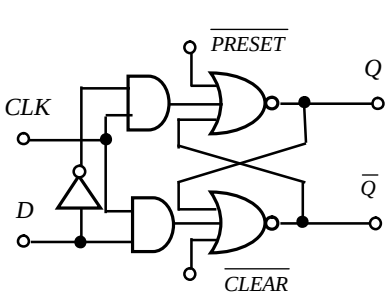
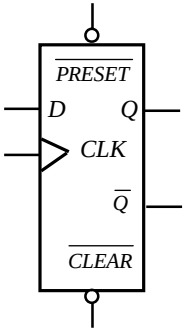


Fig. 3.33. Diagrame de timp pentru un *CBB de tip JK:*
a - cu basculare pe front pozitiv; b - cu basculare pe front negativ.

- Bistabilul D are o asemenea structură (figura 3.34) încât permite de asemenea eliminarea stării de nedeterminare specifice *CBB RS* sincron, respectiv a acelei stări pentru care *S=R=1*.



a)



b)

D	*Q(t+1)*
0	**0**
1	**1**

c)

Q(t)	*D*	*Q(t+1)*
0	0	0
0	1	**1**
1	0	**0**
1	1	1

d)

Fig. 3.34. Bistabilul *D* : a - structura logică; b – simbol; c – tabelă de adevăr; d – tabelă de excitaţie.

Acest bistabil are o intrare de date *D* şi una de sincronizare *CLK* la care se pot adăuga cele două intrări prioritare *PRESET* şi *CLEAR*. Valoarea aplicată pe intrarea de date se transferă la ieşire numai la aplicarea impulsului de sincronizare (deci cu o întârziere de un *tact* [12]).

Din figura 3.86 se observă că tabela de adevăr corespunde liniilor doi şi trei din tabelele de adevăr ale bistabilelor *RS* şi *JK* . Această observaţie creează posibilitatea transformării acestor tipuri de *CBB* în bistabile de tip *D* prin conectarea între cele două intrări de date a unui inversor.

Funcţionarea bistabilului *D* se reflectă şi în diagrama de timp din figura 3.35, în care s-a considerat varianta *Master Slave* la care comutarea se face pe frontul negativ al impulsului de sincronizare.

[12] Numele bistabilului *D – Delay* are în vedere tocmai această întârziere.

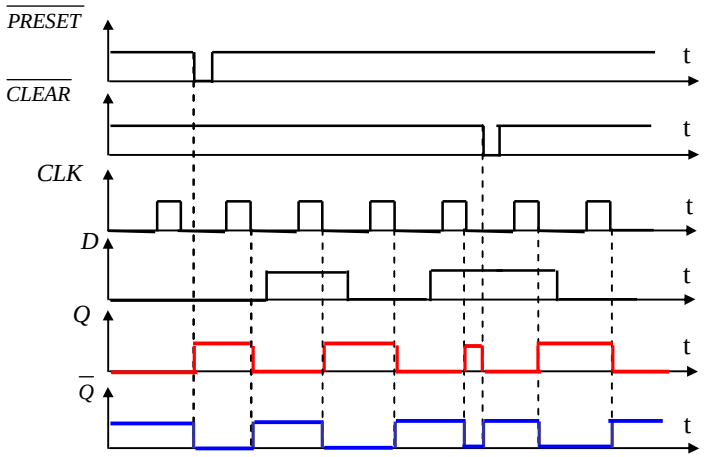
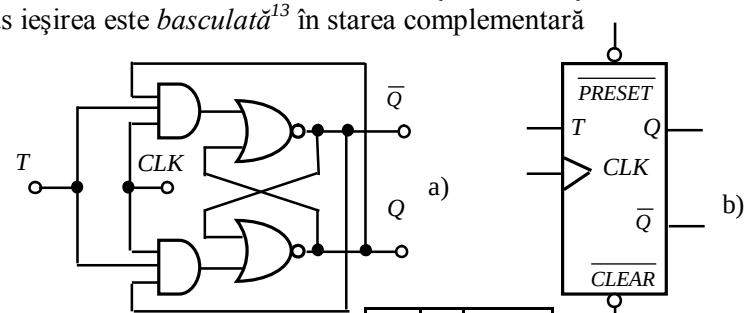


Fig. 3.35. Diagrame de timp pentru un *CBB de tip D Master – Slave.*

- Bistabilul T reprezintă un bistabil *JK* cu o singură intrare de date *T* obţinută prin unirea intrărilor *J* şi ***K***, aşa cum rezultă din figura 3.36. După cum se observă din tabelele de adevăr şi de excitaţie la fiecare impuls de ceas ieşirea este *basculată*[13] în starea complementară



T	Q(t+1)
0	Q(t)
1	$\overline{Q}$(t)

c)

Q(t)	T	Q(t+1)
0	0	0
0	1	1
1	1	0
1	0	1

d)

Fig. 3.36. Bistabilul *T* : a - structura logică; b – simbol; c – tabelă de adevăr; d – tabelă de excitaţie.

[13] Numele bistabilului *T – Toogle (basculare)* are în vedere tocmai această funcţionare.

Se observă că bistabilul comută în starea complementară numai în urma aplicării unui impuls de tact, revenind în starea iniţială după două impulsuri (când *T=1).* Această proprietate face ca *CBB* de tip *T* să fie utilizaţi pe larg la realizarea numărătoarelor sau a altor circuite care necesită o divizare cu 2 a numărului de impulsuri aplicate la intrare.

Pentru exemplificarea funcţionării acestui tip de bistabil în figura 3.37 se prezintă o diagramă de timp a semnalelor la care bascularea se produce pe frontul pozitiv.

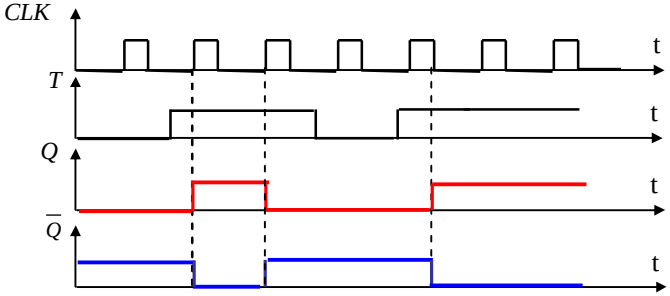


Fig. 3.37. Diagrame de timp pentru un *CBB de tip T.*

Referitor la utilizarea practică a *CBB* sunt utile următoarele precizări:

- terminalele de intrare neutilizate se vor conecta la ***1** logic;*
- conectarea la masă a intrărilor *PRESET sau CLEAR* va forţa ieşirea directă în starea ***1*** respectiv ***0;***
- la bistabilul *RS* realizat cu porţi *NOR* cel puţin una dintre intrările *R şi S* trebuie să fie conectate la masă pentru a se realiza condiţia $R \cdot S = 0$ *;*
- la bistabilul *RS* realizat cu porţi *NAND* cel puţin o intrare, $\overline{R}$ sau $\overline{S}$, trebuie să fie lăsată în aer pentru a se realiza condiţia $\overline{R} + \overline{S} = 1$*;*
- conectarea intrărilor *J, K* la ***1*** logic vor determina comutarea la fiecare impuls de tact;
- comutarea se produce în general pe frontul pozitiv, iar în implementare *MS* pe cel negativ al impulsului de sincronizare;
- există posibilitatea de a realiza toate tipurile de *CBB* cu ajutorul unui singur tip (exemplu *JK).*

3.3.2. Registre

Registrele sunt *CLS* destinate memorării cuvintelor binare şi transferării acestora la cerere. Numărul de biţi, egal cu numărul elementelor de memorie, reprezintă *capacitatea registrului* sau *lungimea* cuvântului registru. După modul în care informaţia poate fi introdusă şi extrasă registrele se împart în patru categorii şi anume:

- registre serie;
- registre paralel;
- registre serie-paralel;
- registre paralel-serie,

cu structurile prezentate în figura 3.38.

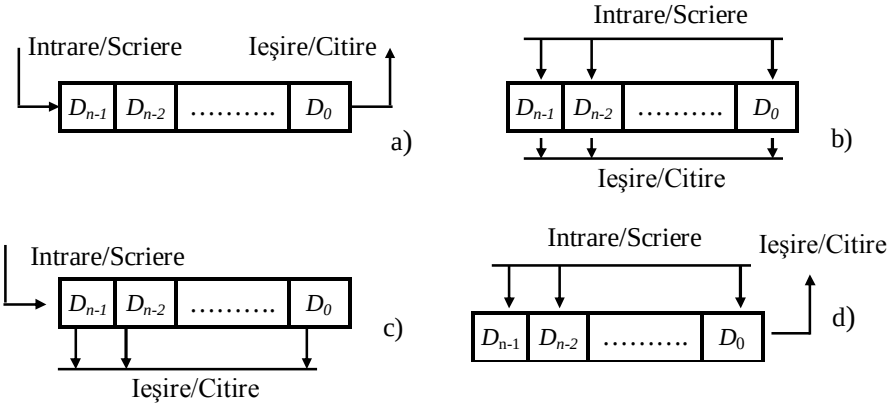


Fig. 3.38. Tipuri de registre: a – serie; b – paralel;
c – serie/paralel; d – paralel/serie.

Registrele serie permit introducerea şi extragerea succesivă, bit cu bit, a datelor. Comanda se realizează cu ajutorul impulsurilor de sincronizare, fiind necesar câte un impuls pentru fiecare cifră binară.

Registrele paralel permit introducerea şi extragerea simultană a biţilor pentru toate rangurile. Din punctul de vedere al comenzii ambele operaţii se desfăşoară pe durata unui impuls de sincronizare.

Registrele serie-paralel denumite şi convertoare serie-paralel permit introducerea bit cu bit a datelor şi extragerea simultană a biţilor întregului cuvânt memorat.

Registrele paralel - serie denumite şi convertoare paralel- serie permit introducerea simultană a biţilor cuvântului care se memorează, urmând ca extragerea să se efectueze succesiv, bit cu bit.

Pentru ilustrarea organizării unui registru în figura 3.39 se prezintă elemente aferente unui registru paralel pe 4 biţi realizat cu bistabili de tip *D*.

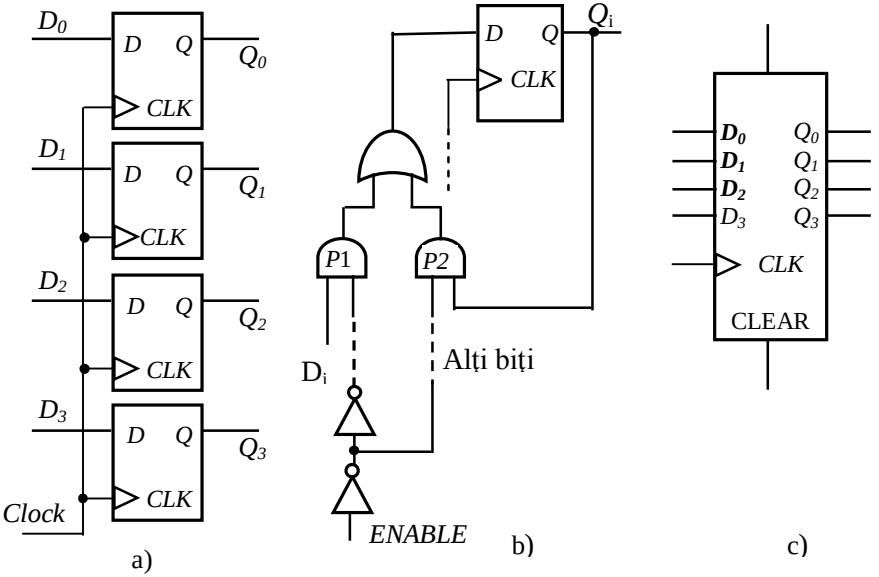


Fig. 3.39. Registru paralel pe 4 ranguri: a – structură; b – organizarea unei celule cu posibilitate de activare; c – simbol de reprezentare.

Cuvântul de intrare de 4 biţi $D_3\ D_2\ D_1\ D_0$ va fi înscris în bistabile la aplicarea impulsului de sincronizare *(CLK)*. Într-un sistem un registru este identificat printr-o *adresă* şi ca urmare fiecare celulă trebuie să prezinte o logică de autorizare la activarea comenzii *ENABLE,* aşa cum este indicat în figura 3.39 b. Când registrul este adresat *(ENABLE = 1)* pe frontul pozitiv al impulsului de tact se va înscrie bitul D_i prezent la intrarea porţii *P1*. Când registrul nu este adresat (*ENABLE=0)* pe frontul pozitiv se va înscrie bitul Q_i prezent la intrarea porţii *P2*. Se observă în acest caz că informaţia nu se modifică la aplicarea impulsului de tact. Din simbolul prezentat în figura 3.39 c se desprinde ideea că pe lângă intrarea de activare un registru poate prezenta şi intrare de ştergere.

Deoarece la nivelul unui microprocesor informaţia este procesată la nivel de cuvânt, în structura acestuia trebuie să existe un număr de registre cu funcţii bine precizate. În cele ce urmează vor fi punctate unele aspecte cu caracter general referitoare la funcţionalitatea registrelor.

- După cum s-a arătat în capitolul precedent prin deplasarea stânga – dreapta a unui cuvânt binar se pot realiza înmulţiri respectiv împărţiri cu puteri ale lui 2. Registrele care permit efectuarea acestor operaţii se numesc *registre de deplasare*[14], în figura 3.40 fiind prezentat un asemenea registru pe 4 biţi implementat cu bistabile *JK*.

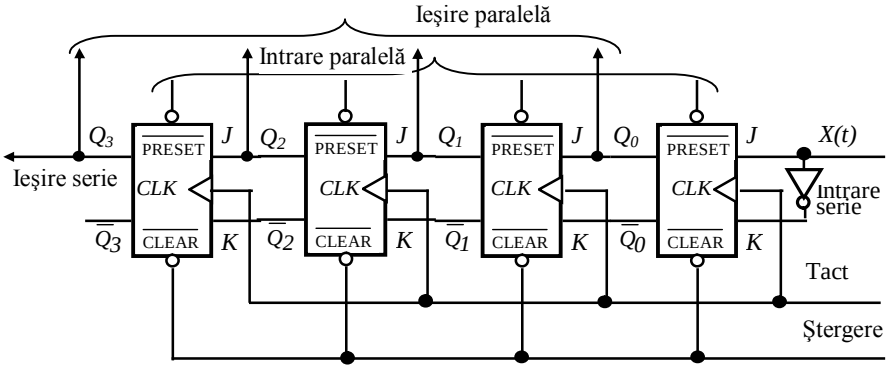


Fig. 3.40. Registru de deplasare pe patru biţi.

Bistabilul Q_0, datorită complementării intrărilor se transformă practic într-un bistabil *D*, astfel încât după aplicarea fiecărui impuls de ceas în această celulă va fi memorat bitul de pe intrarea serie *X(t)*. Întrucât, după cum se observă din figura 3.40 circuitele bistabile sunt conectate în cascadă, după aplicarea fiecărui impuls de tact starea unei celule va coincide cu starea celulei care o precede anterioare aplicării impulsului. Această *propagare* poate fi descrisă prin următoarele relaţii

$$Q_{i+1}(t+1) = Q_i(t) \cdot CLK \; ; \qquad (3.12)$$

$$Q_0(t+1) = X(t) \cdot CLK \quad , \qquad (3.13)$$

în care *CLK* este variabila asociată impulsului de ceas [15], iar indicele *i* reflectă numărul celulei.

[14] Shift Register – în limba engleză
[15] CLK=1 la prezenţa impulsului.

Funcţionarea registrului poate fi înţeleasă urmărind tabela de adevăr din figura 3.93 construită în pentru cuvântul *1011 [X(T)=1, X(2T)=0, X(3T)=1, X(4T)=1]*. Se observă că începând cu al 5-lea impuls şi până la al 8-lea la ieşirea Q_3 se pot citi succesiv biţii introduşi la intrarea *X(t)*.

Nr. tact	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	1	0	1
4	1	0	1	1
5	0	1	1	0
6	1	1	0	0
7	1	0	0	0
8	0	0	0	0

Fig. 3.41. Exemplu de secvenţă pentru un registru de deplasare pe patru biţi.

Registrul oferă şi posibilitatea încărcării paralele pe intrările prioritare *PRESET* urmând ca apoi să se efectueze deplasarea bit cu bit. Se creează posibilitatea utilizării registrului în calitate de convertor *paralel-serie* sau invers, ilustrată prin reprezentarea simbolică din figura 3.42 a.

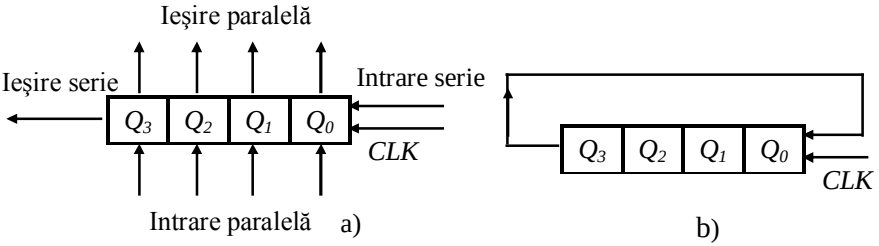


Fig. 3.42. Reprezentări simbolice: a – registru de deplasare; b – registru inel.

Un registru de deplasare poate fi transformat în *registru inel* dacă ieşirea se conectează la intrare potrivit reprezentării din figura 3.42 b. Într-un asemenea registru informaţia poate fi reciclată, un cuvânt înscris ajungând în aceiaşi poziţie după un număr de impulsuri de tact egal cu dimensiunea registrului.

- Registrele aferente microprocesorului, în calitate de unitate centrală de procesare concentrează nivelul celei mai rapide memorii aferente unui calculator. Între registrele microprocesorului au loc curent transferuri condiţionate sau nu de date.

De exemplu un transfer condiţionat de forma

$$(RA \cup RB)cond(x,y) \leftarrow RC \quad ; \tag{3.14}$$

presupune transferul conţinutului registrului *RC* după cum urmează:

- $RA \leftarrow RC$ *dacă* $x = 1$;
- $RB \leftarrow RC$ *dacă* $y = 1$;
- $RA \leftarrow RC$ *şi* $RB \leftarrow RC$ *dacă* $x = y = 1$;
- $RA \xleftarrow{nu} RC$ *şi* $RB \xleftarrow{nu} RC$ *dacă* $x = y = 0$.

Implementarea relaţiei (3.14) se poate face cu ajutorul unei reţele combinaţionale de tipul celei ilustrate în figura 3.43.

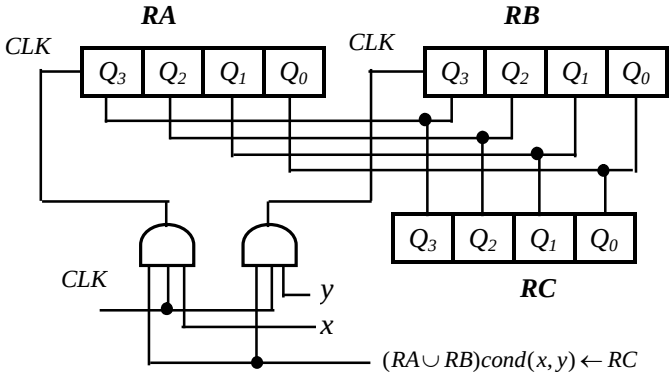


Fig. 3.43. Transferul condiţionat de la o sursă la două destinaţii.

În situaţia unui mare număr de registre transferurile directe sau prin intermediul unei reţele combinaţionale sunt dificil de realizat, o soluţie alternativă fiind conectarea la o magistrală, soluţie indicată în figura 3.44. Transferul trebuie astfel organizat încât la un moment dat să transmită un singur emiţător, numărul receptorilor nefiind limitat.

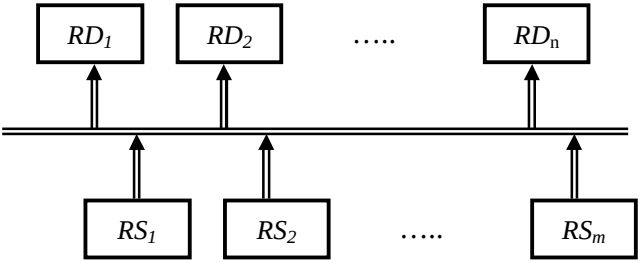


Fig. 3.44. Conectarea registrelor prin intermediul unei magistrale:
RD – registre destinaţie; *RS* - registre sursă.

Având în vedere existenţa celor două tipuri de registre, pentru transferul datelor se execută instrucţiuni de tipul $magistrală \leftarrow RS$ respectiv $RD \leftarrow magistrală$.

- O altă utilizare importantă a registrelor este în cadrul porturilor de intrare-ieşire. Accepţiunea de *port* este asociată unui punct prin care *UCP (microprocesorul)* realizează schimbul de date cu exteriorul. În mod obişnuit un echipament periferic este comandat de un controler care implică necesitatea unuia sau mai multor registre de interfaţare cunoscute sub denumirea de *registre port*. În figura 3.45 sunt prezentate soluţii de utilizare a registrelor pe 8 biţi ca porturi de intrare şi ieşire.

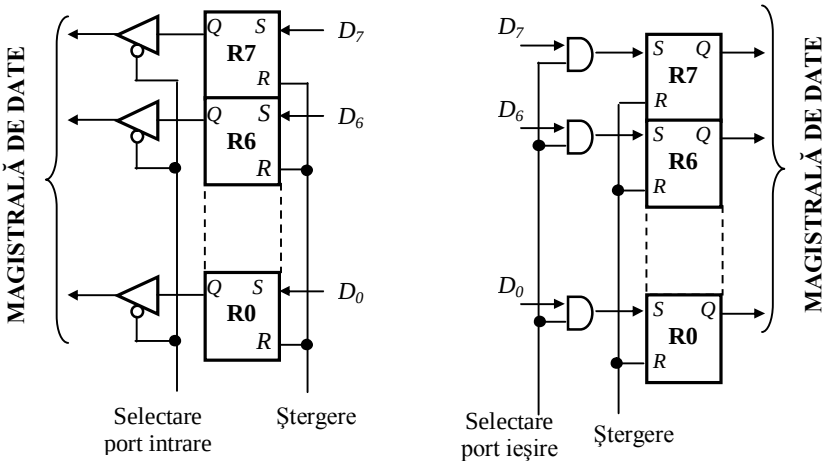


Fig. 3.45. Utilizarea registrelor ca porturi: a – conectarea ca port de intrare; b – conectarea ca port de ieşire.

3.3.3. Numărătoare

Numărătoarele sunt *CLS* care numără (contorizează) impulsurile aplicate la intrare şi memorează rezultatul. Uzual numărătoarele se clasifică în funcţie de următoarele caracteristici:

- ***modul de codificare a informaţiei:*** *binar, binar-zecimal, hexazecimal, exces 3, etc;*
- ***modul de comutare a elementelor de memorie:*** *sincrone, asincrone;*
- ***sensul de numărare:*** *directe, inverse, reversibile, ciclice.*

Practic un numărător realizează, pentru un număr natural *N,* operaţia de identificare a claselor de resturi modulo *C* ($\hat{0}, \hat{1}, \ldots c\hat{-}1$). În acest sens *modulul de numărare* este reprezentat de numărul de stări distincte ale numărătorului. De exemplu, un numărător *modulo 10* (cu *10* stări distincte va avea aceeaşi stare, de exemplu *5,* pentru oricare din următoarele numere aplicate la intrare. *N=5, 15, 25, 35, …, 105, 115, …, 205, 315, etc*. Numărul maxim înscris într-un numărător modulo *c* este *c-1*, deoarece pentru *N=c*, acesta va indica zero (în cazul numărătorului modulo *10,* acest număr este *9)*.

Numărătoarele asincrone sunt cele la care informaţia de la intrare se propagă spre ieşire pas cu pas. Numărătoarele sincrone sunt caracterizate prin aceea că toţi bistabilii care le compun basculează simultan funcţie de informaţiile aplicate la intrare şi de semnalul de tact.

Numărătoarele în buclă sunt practic registre de deplasare a căror ieşire este conectată la intrare.

În continuare, ca exemplu, se prezintă un numărător asincron modulo 10 cu aducere la zero. Tehnica aducerii la zero, în cadrul realizării unui numărător modulo *c*, constă în următoarele:

- se lasă numărătorul să evolueze normal până la starea *c-1*;

- la atingerea stării c se aplică un impuls de ştergere tuturor celulelor numărătorului.

Determinarea numărului de bistabili necesari pentru realizarea unui numărător se face din relaţia $2^n \geq c$, astfel încât pentru $c = 10$ rezultă $n=4$ bistabili.

Pentru numărătorul modulo 10, în figurile 3.46 şi 3.47 evoluţia respectiv diagramele de semnal asociate ieşirilor celor 4 bistabile.

Nr. Imp.	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Q_3	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
Q_2	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1
Q_1	0	0	1	1	0	0	1	1	0	0	0	0	1	1	0	0
Q_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Y	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0

Fig. 3.46. Evoluţia numărătorului decadic cu aducere la zero: Q_3, Q_2, Q_1, Q_0- ieşirile bistabilelor, Y – ieşirea de activare a ştergerii.

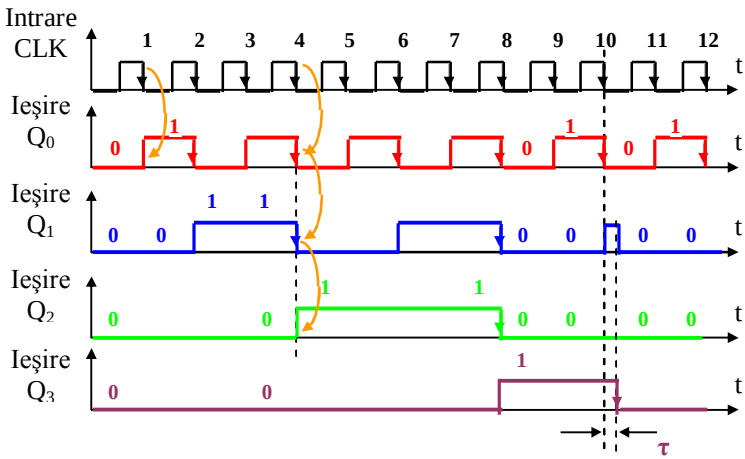


Fig. 3.47. Diagrame de timp pentru numărătorul decadic cu aducere la zero.

Din structura cuvântului binar asociat stării **10** rezultă $Y = \overline{Q_1 \cdot Q_3}$, astfel încât pentru numărător rezultă schema de realizare cu bistabile *JK* din figura 3.48.

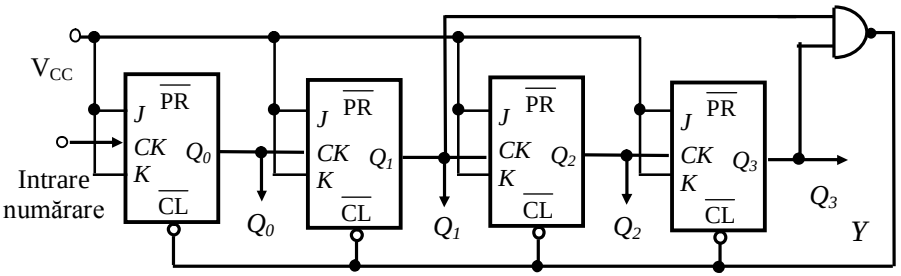


Fig. 3.48. Numărător modulo 10 cu aducere la zero realizat cu CBB de tip JK.

În mod similar se pot obţine şi alte numărătoare asincrone utilizând tabelul de evoluţie a stărilor numărătorului şi sintetizând în mod corespunzător circuitul de aducere la *zero*.

Numărătoarele asincrone prezintă dezavantajul întârzierilor cumulate ale bistabilelor, care vor limita frecvenţa de lucru. Acest neajuns este eliminat prin utilizarea numărătoarelor *sincrone* la care bascularea tuturor *CBB* se produce concomitent. Cu titlu de exemplu în figura 3.49 se prezintă schema unui numărător decadic sincron realizat cu *CBB* de tip *JK*.

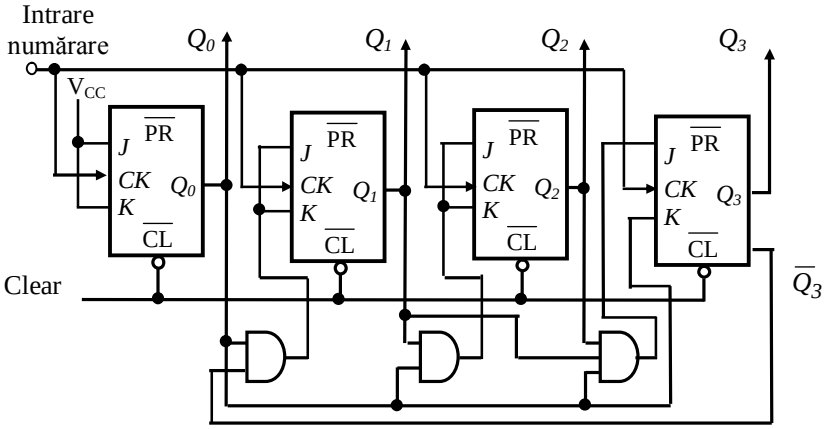


Fig. 3.49. Numărător sincron modulo 10 realizat cu *CBB* de tip *JK*.

3.3.4. Maşini cu algoritm de stare

O maşină cu algoritm de stare ***(Algorithmic State Machine – ASM)*** reprezintă un automat secvenţial de tip *Mealy* sau *Moore* la care algoritmul de tranziţie a stărilor este implementat de o structură combinaţională de tip

ROM sau *PLA*. În figura 3.50 este prezentată o structură principială de *ASM* realizată cu o memorie *ROM* de 32 de octeţi.

Starea actuală memorată într-un registru format din 4 bistabile [16]de tip *D* ale căror ieşiri se aplică intrărilor A_2 … A_5 ale memoriei *ROM*. Cuvântul de ieşire D_0 … D_7 este divizat în D_0 … D_3 – *cuvântul de comandă* şi D_4 … D_7 – *cuvântul pentru starea următoare* $q'(t)$. În fiecare dintre cele 16 stări există posibilitatea testării celor două intrări x_1 şi x_2.

La proiectarea unei *ASM* se are în vedere faptul că aceasta este destinată comenzii unui proces. Implementarea *ASM* presupune înscrierea programului de comandă format din cuvinte binare[17], în secţiunea *ROM* sau *PLA*. Un exemplu de proces este cel aferent unei *RALU (Registers + Arithmetic Unit)*.

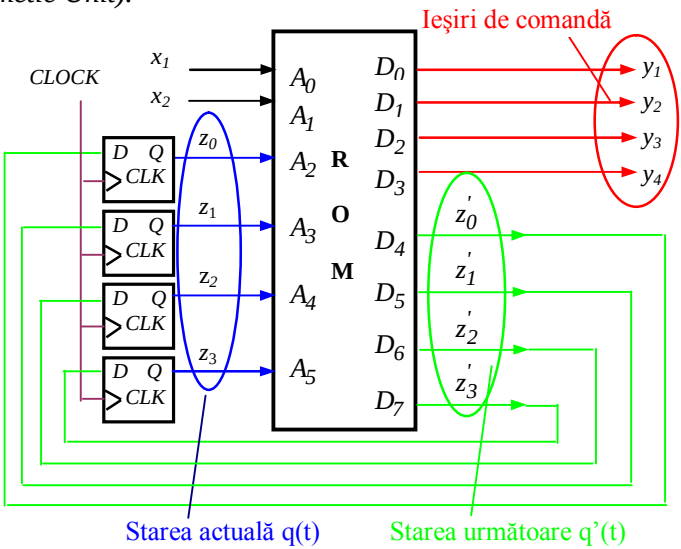


Fig. 3.50. Structura principială a unei maşini cu algoritm de stare.

Ansamblul *RALU+ASM* poate defini structura principială a unui microprocesor ilustrată în figura 3.51. Sarcinile procesorului decurg din instrucţiunile care sunt înscrise sub forma unui program într-o memorie exterioară procesorului.

[16] Pot fi realizate 16 stări distincte.

[17] Succesiunea de instrucţiuni binare se numeşte *microprogram* iar operaţiunea de înscriere *microprogramare* .

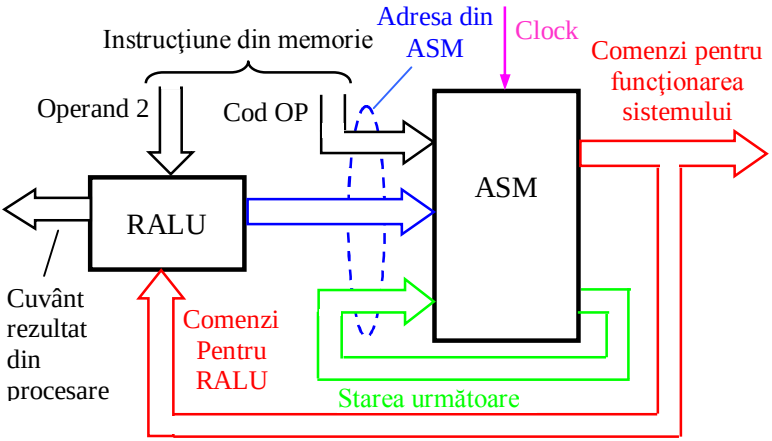


Fig. 3.51. Structura de principiu a unui procesor.

Se consideră că o instrucţiune conţine *Codul Operaţiei – OP* care este introdus în *ASM* şi un al doilea operand care este introdus în *RALU*[18]. Pe baza cuvântului asociat stării următoare, a condiţiilor primite de la *RALU* şi a *OP* se generează adresa din *ASM*. La această adresă se găseşte o microinstrucţiune a cărei execuţie presupune generarea semnalelor de comandă către unităţi subordonate.

[18] Se consideră că primul operand a fost adus anterior într-un registru special al *RALU* şi anume *acumulatorul.*

4. SUBSISTEMELE UNUI CALCULATOR

4. SUBSISTEMELE UNUI CALCULATOR

4.1. Magistrale de comunicaţie

4.1.1. Elementele definitorii ale unei magistrale

Din punct de vedere conceptual magistrala ***(M)*** reprezintă un mediu comun de comunicaţie între componentele unui sistem de calcul. Fizic magistrala este alcătuită dintr-un set de linii de semnal care facilitează transferul de date şi sincronizarea între componentele sistemului.

În prezent pot fi identificate două clase mari de M şi anume:

- *magistrale de sistem* dezvoltate pentru conectarea *UCP* la celelalte componente ale sistemului (exemplu: *MULTIBUS, ISA, EISA, PCI)***;**
- *magistrale specializate* destinate optimizării transferului de date cu un anumit tip de echipamente periferice (exemplu: ***VESA, SCSI, GPIB).***

O magistrală se compune dintr-un set de semnale şi un set de reguli care guvernează transferul de informaţii şi accesul la mediul de comunicaţie.

Informaţiile pot fi: *date, adrese, informaţii de control şi sincronizare,*magistralele corespunzătoare putând fi denumite *magistrale de date, adrese sau comenzi.*

Regulile se referă la:

- caracteristicile fizice şi electrice ale componentelor conectate pe magistrale (exemplu: niveluri de tensiune, curenţi, încărcare, tip conectori etc.);
- secvenţa de generare a semnalelor necesare pentru efectuarea unui transfer;
- timpi limită pentru diferitele faze ale unui transfer şi timpi de menţinere a unui anumit semnal;
- intercondiţionările funcţionale şi temporale între diferite tipuri de semnale.

În funcţie de numărul semnalelor utilizate pentru transferul de date magistralele pot fi de două tipuri:

- *magistrale seriale (MS);*
- *magistrale paralele (MP).*

MS se utilizează rar ca mijloc de comunicaţie între componentele de bază ale unui calculator (unitate centrală, memorie, interfeţe de I/E), datorită vitezei de transfer relativ scăzute.

Întrucât magistralele interne aferente unui calculator sunt practic în exclusivitate *MP* se vor face referiri la aceste tipuri de *M*. În accepţiunea *clasică* o asemenea *M* conţine următoarele tipuri de semnale:

- *semnale de date* cu următoarele caracteristici:

  - semnale bidirecţionale utilizate pentru transferul de date şi de instrucţiuni;

  - la un moment dat o singură unitate poate să emită pe liniile de date;

  - numărul liniilor de date (8, 16, 32, 64) determină dimensiunea maximă a cuvântului de date care poate fi transferat la un moment dat şi implicit viteza medie de transfer a magistralei;

- *semnale de adresă* cu următoarele caracteristici:

  - semnale unidirecţionale utilizate pentru specificarea adresei modulului destinaţie sau sursă;

  - numărul de linii de adresă determină spaţiul maxim de adresare permis de magistrală (de exemplu o magistrală cu 24 linii de adresă permite adresarea într-un spaţiu cu 2^{24}*=16 Mlocaţii*);

- *semnale de comandă* – utilizate pentru specificarea direciei de transfer (exemplu dinspre sau spre procesor) şi a tipului de modul adresat (exemplu: modul de memorie, modul de *I/E*, etc.);

- *semnale de control* – utilizate pentru reglarea condiţiilor de transferare a datelor (exemplu: temporizarea deschiderii/închiderii amplificatoarelor de magistrală);

- *semnale de întrerupere* – permit semnalizarea unor evenimente interne sau externe şi implicit determină întreruperea execuţiei programului curent;

- *semnale de tact* – utilizate pentru sincronizare şi pentru generarea unor semnale de frecvenţă prestabilită;

- *semnale de control a accesului* – folosite pentru arbitrarea şi controlul accesului pe magistrală (în cazul magistralelor *multimaster);*
- *semnale de alimentare* – folosite pentru alimentarea modulelor sistemului.

Magistralele pot fi clasificate după mai multe criterii şi anume:

a. după modul de lucru (în raport cu semnalul de tact):

- *magistrale sincrone* – la care ciclurile de transfer sunt direct corelate cu semnalul de tact.

- *magistrale asincrone* – la care nu există o legătură directă între evoluţia în timp a unui ciclu de transfer şi tactul sistemului (majoritatea magistralelor actuale - *ISA, EISA, MULTIBUS* etc.– lucrează pe acest principiu);

b. după numărul de unităţi master conectate la magistrală:

- *magistrale unimaster* – există un singur modul master pe magistrală şi în consecinţă nu este necesar un mecanism de arbitrare (*modul master* – poate iniţia un transfer cu magistrala, *modul slave* – modul care poate fi comandat în timpul unui ciclu de transfer, neavând elemente pentru controlul magistralei);

- *magistrale multimaster* – permit conectarea mai multor unităţi master pe acelaşi tronson de magistrală; în acest sens magistrala va trebui să conţină semnale de arbitrare şi un protocol de transfer al controlului pe magistrală (exemplu *MULTIBUS*);

c. după modul de realizare a transferului de date:

- *magistrale cu transfer prin cicluri (secvenţiale)* – ciclurile de transfer se desfăşoară secvenţial, la un moment dat cel mult un ciclu este în desfăşurare (majoritatea magistralelor folosesc acest principiu de transfer);

- *magistrale tranzacţionale* – transferul de date se desfăşoară prin *tranzacţii (*o tranzacţie este divizată în mai multe faze, iar mai multe tranzacţii se pot desfăşura simultan cu condiţia să se afle în faze diferite).

4.1.2. Magistrale sincrone

După cum s-a menţionat la aceste tipuri de magistrale ciclurile de transfer sunt direct corelate cu semnalul de tact. Pentru a explica modul în care se realizează transferul vom considera o magistrală la care citirea unui cuvânt din memorie necesită trei *cicluri magistrală* T_1, T_2, T_3 cu o durată *3T, T* fiind perioada ceasului (figura 4.1).

În ciclul T_1 *Unitatea Centrală de Procesare* (*UCP)* depune adresa cuvântului pe liniile de adrese. După stabilizarea adresei la noua valoare, se activează semnalele $\overline{MREQ}$ (**M**emory **REQ**uest – cerere acces la memorie) şi $\overline{RD}$ (**R**ea**D** operaţie de citire). Memoria decodifică adresa în ciclul T_2 şi depune data pe magistrală în ciclul T_3. Pe frontul descrescător al ceasului din ciclul T_3, *UCP* strobează (citeşte) liniile de date, memorând valoarea într-un registru intern. După citire *UCP* dezactivează semnalele $\overline{MREQ}$ şi $\overline{RD}$, după care, de la frontul crescător al ceasului poate începe un nou ciclu.

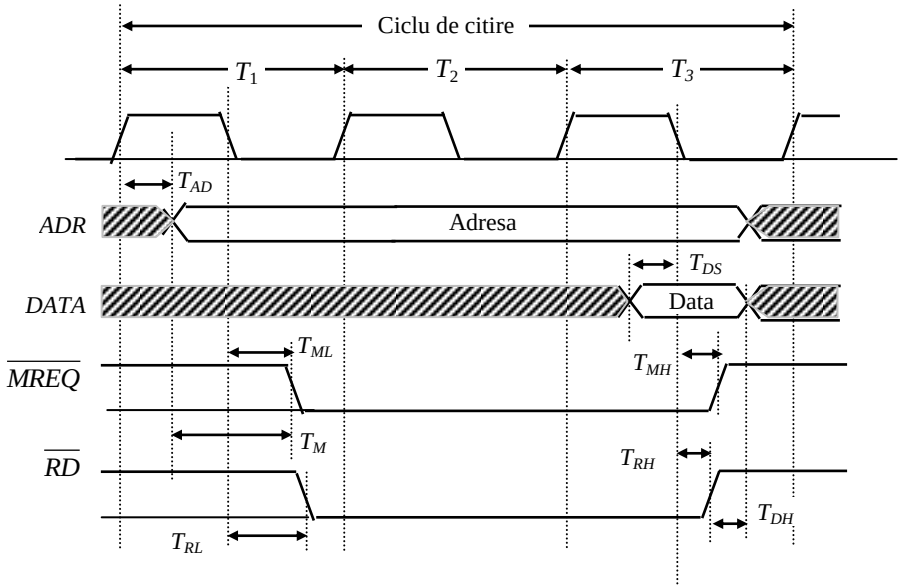


Fig. 4.1. Ciclul de citire la o magistrală sincronă.

Semnificaţiile timpilor marcaţi în figura 4.1 sunt următoarele:

- T_{AD} este intervalul de timp de la începutul ciclului T_1 până la depunerea adresei (este limitat superior);

- T_{DS} (*Data Setup)* – intervalul de la depunerea datei până la frontul descrescător al ceasului din ciclul T_3 (este limitat inferior);

- T_M - intervalul de la depunerea adresei până la activarea semnalului $\overline{MREQ}$ (este limitat inferior) ; cest timp este important dacă semnalul $\overline{MREQ}$ se utilizează la selecţia circuitului de memorie, deoarece anumite memorii necesită un timp de stabilizare a adresei până la selecţie;

- T_{ML}, T_{RL} – intervalele de la frontul descrescător al ceasului din ciclul T_1 până la activarea semnalelor $\overline{MREQ}$ respectiv $\overline{RD}$ (cei doi timpi sunt limitaţi superior pentru ca cele două semnale să fie activate într-un anumit timp de la mijlocul ciclului T_1;

- T_{MH}, T_{RH} – timpul după care trebuie dezactivate semnalele $\overline{MREQ}$ şi $\overline{RD}$ după citirea datei (limitaţi superior);

- T_{DH} – timpul cât trebuie menţinută data pe magistrală după dezactivarea semnalului $\overline{RD}$.

Pe lângă ciclurile de citire şi scriere, unele magistrale permit şi transferuri pe blocuri. Dacă se lansează o cerere de citire a unui bloc, dispozitivul *master* indică celui *slave* numărul de octeţi care urmează a fi transferaţi (de exemplu în ciclul T_1). Dispozitivul *slave* transmite un octet în timpul fiecărei ciclu, până când contorul asociat transferului ajunge la zero.

Magistralele sincrone prezintă unele dezavantaje cum ar fi:

- posibile întârzieri în situaţia în care un transfer nu se termină după un număr întreg de cicluri (trebuie să se aştepte sfârşitul de ciclu);

- existenţa unei unice viteze (dacă la o magistrală se conectează dispozitive cu viteze diferite viteza trebuie aleasă după dispozitivul cel mai lent, cele rapide în raport cu acesta fiind întârziate);

- după alegerea duratei unui ciclu de magistrală este dificil să se utilizeze dispozitive cu îmbunătăţirile tehnologice (de exemplu dacă în timp vor fi disponibile memorii mai rapide, cu toate că vor putea fi utilizate ele vor funcţiona la aceiaşi viteză ca cele vechi deoarece protocolul M cere memoria să depună datele cu T_{DS} înaintea frontului căzător al ceasului din ciclul T_3.

4.1.3.Magistrale asincrone

Magistralele asincrone elimină neajunsurile magistralelor sincrone deoarece în locul semnalului de ceas se utilizează un protocol logic între emiţător şi receptor ***(handshake)***. În figura 4.2 este ilustrată diagrama de semnale aferentă unui protocol de citire din memorie.

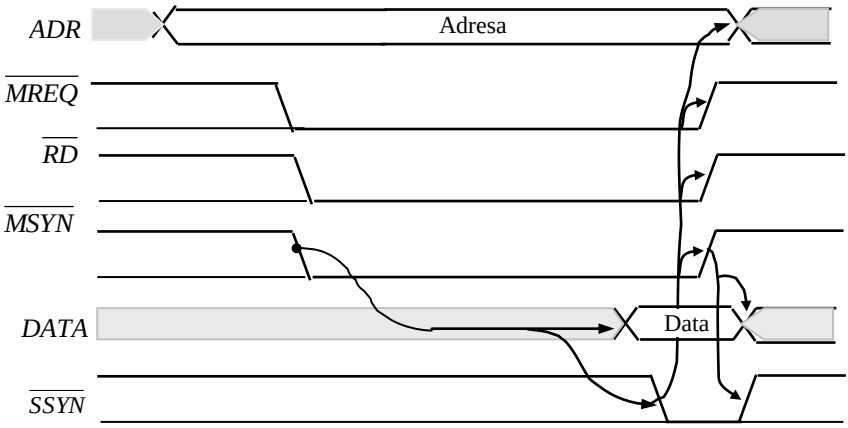


Fig. 4.2. Operaţie de citire la o magistrală asincronă.

Pentru operaţia de citire, după depunerea adresei şi activarea semnalelor $\overline{MREQ}$ şi $\overline{RD}$ dispozitivul *master* activează semnalul $\overline{MSYN}$ (***M**aster **SYN**chronization).* După detectarea acestui semnal dispozitivul *slave* depune data şi activează apoi semnalul $\overline{SSYN}$ (***S**lave **SYN**chronization)* semnalând *disponibilitatea* datei.

Dispozitivul *master* detectează activarea semnalului $\overline{SSYN}$ memorează data şi apoi dezactivează liniile de adrese, împreună cu semnalele $\overline{MREQ}$, $\overline{RD}$ şi $\overline{MSYN}$, dezactivarea acestuia din urmă marcând terminarea ciclului.

Dispozitivul *slave* detectează dezactivarea lui $\overline{MSYN}$ şi la rândul dezactivează şi invalidează datele, ajungându-se în starea iniţială cu toate semnalele dezactivate în aşteptarea unui nou ciclu.

Sintetic acţiunile aferente protocolului sunt:

1. se activează $\overline{MSYN}$;
2. se activează $\overline{SSYN}$ ca răspuns la activarea lui $\overline{MSYN}$;
3. $\overline{MSYN}$ este dezactivat ca răspuns la activarea lui $\overline{SSYN}$;
4. $\overline{SSYN}$ este dezactivat ca răspuns la activarea lui $\overline{MSYN}$.

Cu toate că magistralele asincrone rezolvă o parte din neajunsurile celor sincrone, totuşi cele mai răspândite sunt acestea din urmă. Motivul este că necesite mai puţine linii, realizarea este mai simplă şi în consecinţă sunt mai ieftine.

4.2. Unitatea centrală de procesare *(UCP)*

Unitatea centrală de prelucrare reprezintă cel mai important subsistem al unui sistem de calcul numeric care efectuează operaţiile esenţiale de prelucrare a datelor şi coordonează activitatea celorlalte subsisteme.

4.2.1. Structura şi funcţiile UCP

Sintetic spus *UCP* asigură disponibilităţile de calcul şi de comandă a echipamentului numeric. Pornind de funcţiile menţionate, *UCP* este structurată în trei diviziuni şi anume *memorare, calcul, comandă.*

Diviziunea de memorare este reprezentată de *registrele generale (RG)* care au rolul unor locaţii rapide de memorie în care se păstrează date utilizate de către program, rezultate intermediare, operanzi etc.

De asemenea *RG* pot fi implicate în operaţii elementare cum ar fi: rotaţie, deplasare, incrementare, decrementare, încărcare paralelă sau serială, ştergere.

În principiu RG pot fi organizate în două moduri şi anume:

- registre *conectate direct;*
- registre *conectate prin magistrala internă a UCP* (figura 4.3).

Primul mod de organizare permite transferul simultan al datelor între mai multe registre, în timp ce al doilea tip permite efectuarea la un moment dat a unui singur transfer ceea ce atrage după sine ocuparea magistralei interne a procesorului. În rândul *RG* poate fi încadrat şi registrul *Acumulator* utilizat în operaţiile aritmetico-logice şi care păstrează un

operand şi rezultatul operaţiei. După cum se va vedea, din punct de vedere funcţional Acumulatorul poate fi mai bine încadrat în *Unitatea aritmetică de procesare.*

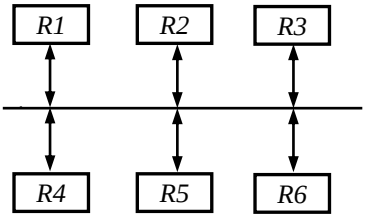


Fig. 4.3. Organizarea registrelor generala la o *UCP*.

Diviziunea de calcul (Unitatea Aritmetică şi Logică – UAL)

UAL reprezintă un subsistem al *UCP* care efectuează operaţiile aritmetice şi logice elementare. Operaţiile pot fi efectuate *paralel*, când se prelucrează simultan toţi biţii unui cuvânt, sau *paralel - serie* când se prelucrează simultan biţii unei zone a cuvântului transmis serial. Durata unei operaţii (cu excepţia înmulţirii şi împărţirii) este egală cu durata unui ciclu al procesorului.

UAL preia operanzii pe care îi prelucrează, din memorie sau din registre, rezultatele fiind transmise înapoi în memorie sau în mediul extern prin intermediul subsistemului de intrare-ieşire. În general o *UAL* primeşte doi operanzi şi un cod al operaţiei şi furnizează un rezultat şi eventuale informaţii despre acesta (condiţii / indicatori), conform reprezentării din figura 4.4.

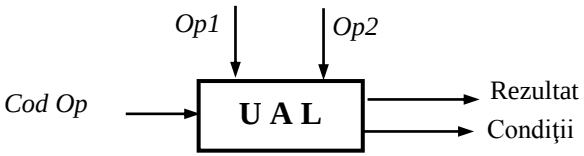


Fig. 4.2. Caracterizarea informaţională a unei *UAL*.

Elementul de bază al unei *UAL* este un sumator ale cărui caracteristici determină performanţele întregii unităţi. În figura 4.5 se prezintă o schemă principială de structură a unei *UAL* în virgulă fixă.

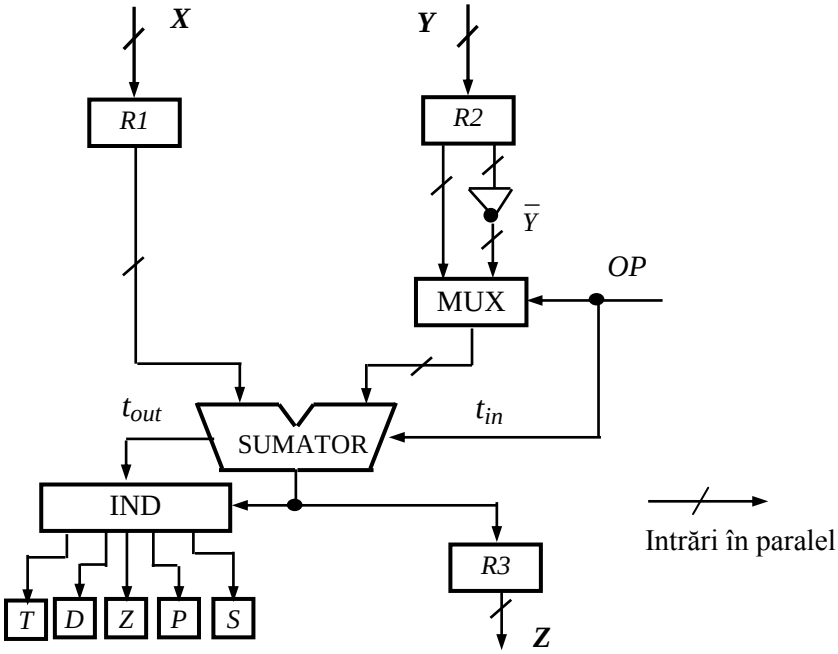


Fig. 4.5. Structura principială a unei UAL.

Cei doi operanzi *X* şi *Y* sunt furnizaţi paralel şi încărcaţi în registrele *R1* şi *R2*. Funcţie de valoarea bitului *OP* se poate realiza operaţia *X+Y* sau *X-Y* astfel:

- dacă *OP = 0* se face adunarea *X+Y+0*, caz în care intrarea de transport în sumator este nulă (t_{in} = *OP = 0*);
- dacă *OP = 1* se face adunarea *X+Y+1* (t_{in} = *OP = 1*), deci ţinând cont că numerele sunt reprezentate în *complement faţă de 2,* se face scăderea *X-Y*.

Furnizarea lui *Y* sau *–Y* se face prin intermediul multiplexorului *MUX*. În schemă mai există o logică de tip combinaţional *IND* care permite poziţionarea bistabililor *T, D, Z, P, S* numiţi indicatori de condiţie.

- ***T*** (***T****ransport*) – *T=1* dacă a apărut un transport la o operaţie de adunare sau un împrumut la o operaţie de scădere (la rangul cel mai semnificativ).
- ***D*** (***D****epăşire*) – *D=1* dacă rezultatul este prea mare în modul pentru a putea fi reprezentat; dacă rezultatul este corect *D=0*.
- ***Z*** (***Z****ero*) – *Z=1* dacă rezultatul este nul, altfel *Z=0*.
- ***P*** (***P****aritate)* – *P=1* dacă rezultatul conţine un număr par de biţi *1*, altfel *P=0*.
- ***S*** (***S****emn)* – *S=1* dacă rezultatul este negativ, altfel *S=0*.

În ceea ce priveşte operaţia de înmulţire, pentru un număr redus se poate realiza cu o logică de tip combinaţional, următorul exemplu fiind edificator pentru $X = x_2\, x_1\, x_0$ şi $Y = y_2\, y_1\, y_0$.

$$
\begin{array}{cccc}
 & x_2 & x_1 & x_0 \\
\cdot & y_2 & y_1 & y_0 \\
\hline
 & x_2y_0 & x_1y_0 & x_0y_0 \\
 & x_2\,y_1 & x_1y_1 & x_0y_1 \\
x_2y_2 & x_1y_2 & x_0y_2 & \\
\hline
\end{array}
$$

Obţinerea produselor de mai sus se poate face într-o structură combinaţională de tipul celei din figura 4.6.

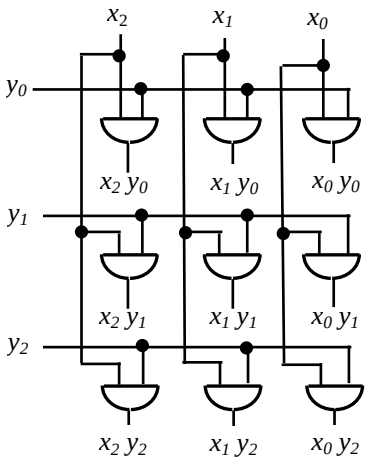


Fig. 4.6. Structură pentru calculul produselor parţiale.

Produsele parţiale $x_i\, y_j$ ($i=1...3,\ j=1...3$) se aplică unei reţele de sumatoare de tipul celei reprezentate în figura 4.7, rezultând în final produsul pe 6 biţi.

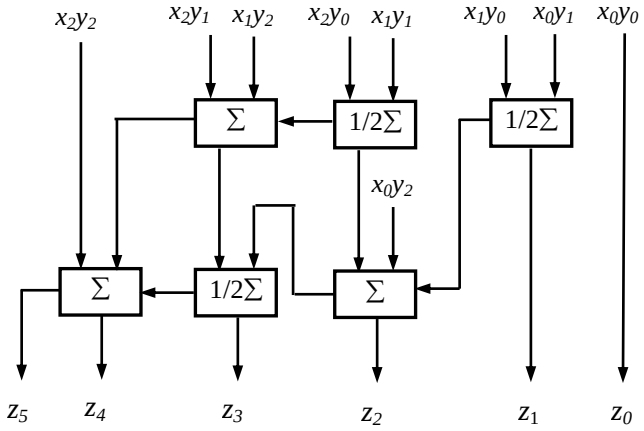


Fig. 4.7. Reţea de sumatoare pentru obţinerea produsului pe 6 biţi.

<u>Diviziunea de comandă (Unitatea de Comandă – UC)</u>

UC realizează supervizarea resurselor sistemului de calcul, supervizare care presupune:

- *obţinerea de informaţii* în legătură cu starea acestor resurse;
- *generarea semnalelor de comandă* în vederea execuţiei instrucţiunilor.

Exercitarea funcţiilor, determină prezenţa principială în *UC* a următoarelor elemente:

- ***RI*** (**R**egistrul de **I**nstrucţiuni);
- ***RS*** (**R**egistrul de **S**tare);
- ***NP*** (**N**umărătorul de **P**rogram);
- ***GF*** (**G**eneratorul de **F**aze);
- ***DI*** (**D**ecodificatorul de **I**nstrucţiuni);
- ***GT*** (**G**eneratorul de **T**act);
- ***BCC*** (**B**locul **C**ircuitelor de **C**omandă).

Toate operaţiile elementare (***microoperaţii)*** care se execută pe parcursul unei instrucţiuni sunt comandate de către semnalele generate de

către *BCC* (***microcomenzi***). Microcomenzile sunt trimise *elementelor de execuţie* din structura calculatorului respectiv: *registre, UAL, memorie, porturi, etc.* care realizează operaţii elementare cum ar fi: *înscriere, validare ieşire, ştergere, deplasări, etc.* Execuţia unei instrucţiuni, presupune o succesiune de microoperaţii, iar toate microoperaţiile care se execută la un moment dat se constituie într-o ***fază a instrucţiunii*.**

Celelalte entităţi evidenţiate în structura ***UC*** participă la execuţia unei instrucţiuni după cum urmează:

- ***NP*** conţine adresa instrucţiunii aflate în execuţie;
- ***RI*** conţine codul (parţial sau total) instrucţiunii care urmează a se executa;
- ***GF*** construieşte succesiunea de faze necesară pentru execuţia instrucţiunii al cărui cod se găseşte în *RI*; generarea fazei următoare în cadrul unei succesiuni este determinată de trei elemente şi anume:
  - a - faza curentă;
  - b - tipul instrucţiunii;
  - **a)** conţinutul registrului de stare ***RS;***
- ***RS*** păstrează informaţii despre modul de efectuare a operaţiilor comandate de către *BCC;*
- ***GT*** generează cadenţă schimbărilor de stare pentru toate circuitele secvenţiale din structura calculatorului.

Execuţia unei instrucţiuni se face în cadrul următoarei secvenţe de paşi care se constituie în *ciclul de extragere – decodificare - execuţie***:**

1 **–** extrage codul instrucţiunii curente din memorie (de la adresa conţinută în ***NP***) şi îl depune în ***RI***;

2 - modifică conţinutul ***NP,*** în sensul înscrierii în acesta a adresei următoarei instrucţiuni ce urmează a se executa;

3 **–** determină tipul instrucţiunii al cărui cod a fost extras în *RI***;**

4 – dacă instrucţiunea necesită date din memorie, se determină adresele acestora;

5 – în situaţia în care sunt necesare date acestea sunt extrase în registrele generale ale *UC***;**

6 – execută instrucţiunea;

7 – înscrie rezultatele instrucţiunii în locul specificat de instrucţiune;

8 – se reia pasul ***unu*** pentru următoarea instrucţiune.

Formatul instrucţiunilor şi moduri de adresare

Instrucţiunile se găsesc în memorie şi pot ocupa una sau mai multe locaţii de memorie, din care prima conţine întotdeauna codul operaţiei *CO* (figura 4.8). În comentariile care vor urma se vor avea în vedere instrucţiunile specifice limbajului de asamblare.

În general *codul instrucţiunii* conţine următoarele informaţii necesare interpretării şi execuţiei sale de către *UC*:

a. ***codul operaţiei*** specifică tipul operaţiei efectuate de către instrucţiune: *transfer, deplasare, operaţie aritmetică sau logică etc.;*

b. ***operanzii*** asupra cărora operează instrucţiunea (în instrucţiune se specifică *datele, adresa lor, sau modul în care se determină adresa);*

c. ***modul*** în care se efectuează instrucţiunea (de exemplu se specifică modul de interpretare a informaţiei referitoare la operanzi).

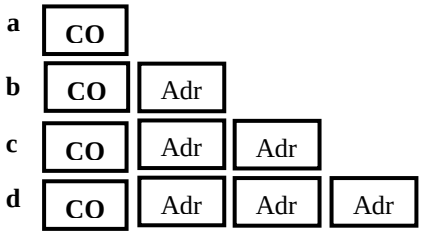


Fig. 4.8. Formate de instrucţiuni: a – fără adresă, b,c,d – cu una două sau trei adrese.

- *Instrucţiunile fără adresă* operează cu date din registre implicite (exemplu registrul acumulator).
- *Instrucţiunile cu o adresă* operează cu date din acumulator ca registru implicit şi un registru sau o dată din memorie.
- *Instrucţiunile cu două adrese* sunt de tipul registru-registru, registru-memorie, sau memorie-memorie (operează cu date din două registre, un registru şi memorie sau din memorie).
- *Instrucţiunile cu trei adrese* specifică pe lângă adresele celor doi operanzi şi pe cea a rezultatului, care poate fi plasat de asemenea în registre sau memorie.

Modurile de adresare indică practic maniera de specificare a operanzilor sau a adresei rezultatului. În continuare vor fi prezentate câteva moduri de adresare.

- *Adresarea imediată*

Codul instrucţiunii conţine valoarea operandului numit în acest caz ***operand imediat.*** Operandul este adus din memorie *imediat* împreună cu codul instrucţiunii, în care este inclus imediat.

- *Adresarea directă*

Codul instrucţiunii conţine adresa efectivă (sau offsetul în cadrul existenţei segmentelor) la care se află operandul în memorie.

- *Adresarea registru*

Acest mod de adresare reprezintă o variantă a *adresării directe,* în care operandul (operanzii) sunt conţinuţi în registrele de uz general ale microprocesorului în loc de memorie. Deoarece numărul de registre de uz general este relativ redus, adresa unui registru se poate codifica pe un câmp de 3-4 biţi funcţie de numărul de registre. În acest fel *adresele* pot fi incluse în chiar codul instrucţiunii, ceea ce face ca instrucţiunea să fie scurtă (un singur cuvânt) aspect cu consecinţe benefice în ceea ce priveşte vitezele de execuţie.

- *Adresarea indirectă*

În cazul *adresării indirecte* câmpul adresă din instrucţiune specifică adresa locaţiei de memorie sau registrului care conţine nu operandul *ci adresa operandului.*

- *Adresarea indexată*

La adresarea indexată adresa datei care se va utiliza de către instrucţiune se obţine prin adunarea conţinutului *registrului index* specificat cu o valoare numerică (constantă) de asemenea specificată.

Adresarea indexată este utilizată la prelucrarea unor structuri de date plasate succesiv în memorie. Constanta reprezintă adresa de început a structurii de date, iar conţinutul registrului index este adresa relativă în structură a fiecărui octet sau cuvânt.

- *Adresarea bazată*

În cadrul adresării bazate adresa operandului se obţine ca sumă dintre o constantă şi conţinutul unui registru special numit *registru bază,* adresa conţinută de acesta fiind numită *adresă de bază.* Spre deosebire de

adresarea indexată nu are loc autoincrementarea sau autodecrementarea registrului de bază, conţinutul acestuia (*adresa de bază)* putându-se face numai explicit prin program.

- *Adresarea stivă*

Stiva reprezintă o zonă de memorie în care se înscriu date potrivit strategiei *LIFO* (***L**ast **I**nput **F**irst **O**utput*). O stivă este caracterizată de două adrese importante şi anume *Baza şi Vârful stivei.* Această ultimă adresă este înscrisă într-un registru dedicat al microprocesorului numit ***Stack Pointer*.** Instrucţiunile de lucru cu stiva ***Push*** *(pentru depunere)* şi ***Pop*** *(pentru extragere)* nu necesită un câmp de adresă şi ca urmare execuţia acestora este rapidă.

- *Adresarea combinată*

În programe se utilizează combinaţii ale modurilor de adresare prezentate cum ar fi: *bazată indexată, indirectă indexată,* etc.

4.2.2. Elemente de arhitectura a microprocesoarelor pe 8 biţi

4.2.2.1. Conceptul de microprocesor

Microprocesorul *(μP)* reprezintă o unitate centrală de procesare realizată pe un singur circuit logic *(chip).* Practic *μP* este un sistem de o maximă complexitate, reprezentat din punct de vedere funcţional printr-o structură formată *din elemente secvenţiale şi combinaţionale.* Caracteristica specifică a unui *μP*, în raport cu o configuraţie clasică, este reprezentată de flexibilitatea sa la nivel software. Aceasta face ca implementarea unei structuri logice pentru o aplicaţie dată să nu reprezinte o intervenţie la nivel hardware ci să se reducă la implementarea unui program specific.

Termenul de *microprocesor* a fost introdus de firma Intel în anul 1972 şi este legat de realizarea de către această firmă a primelor μP.

Pentru a putea caracteriza un *μP* se reamintesc funcţiile de bază ale unui calculator numeric şi anume:

- funcţia de *INTRARE* care permite conectarea lumii exterioare la sistem;

- funcţia de *IEŞIRE* care permite conectarea sistemului la lumea exterioară;

- funcţia de *MEMORARE* care asigură păstrarea datelor şi a instrucţiunilor programului;

- funcţia *ARITMETICO - LOGICĂ* ce permite efectuarea operaţiilor de calcul (aritmetice şi logice) în sistem;

- funcţia de *CONTROL* care grupează totalitatea operaţiilor de secvenţializare şi coordonare în cadrul sistemului.

Celor cinci funcţii le corespund tot atâtea secţiuni care nu sunt însă integral acoperite de către *UCP (μP).* În general secţiunile de intrare-ieşire şi memoria sunt în mare măsură exterioare μP. După cum se observă din figura 4.9 secţiunile ***aritmetico-logică*** şi ***control*** sunt integrate în *μP*.

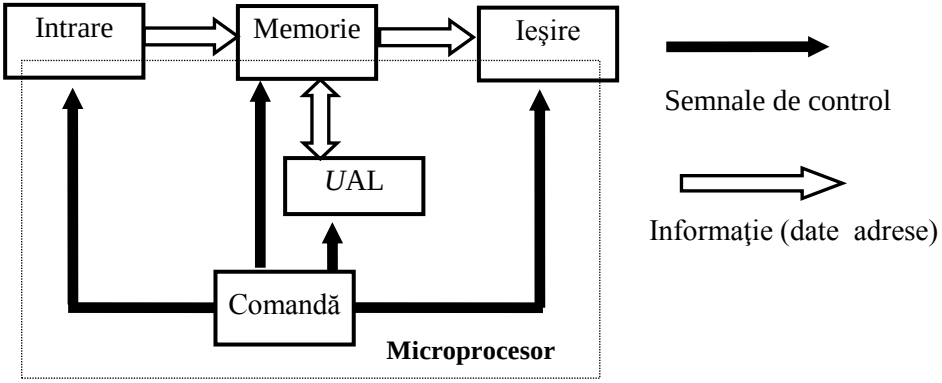


Fig. 4.9. Structura funcţională şi fluxul de informaţie într-un sistem cu microprocesor

Uzual, informaţia este adusă prin intermediul funcţiei de *INTRARE* în memoria sistemului. Din memorie, informaţia (instrucţiuni şi date) este citită şi decodificată, executându-se secvenţial instrucţiunile programului. Rezultatele sunt apoi transferate, prin intermediul funcţiei *IEŞIRE* în afara sistemului. Toate operaţiile, a căror înlănţuire temporală este redată simplificat în figura 4.10, sunt coordonate prin atribute ale funcţiei *CONTROL*.

În intervalul T_1 informaţia (instrucţiuni şi date) se memorează activându-se secţiunile *intrare şi memorie*. În T_2 codul este preluat din memorie şi executat iar în T_3 cu ajutorul funcţiei *ieşire* rezultatele sunt transferate în memorie.

După cum s-a arătat, totalitatea informaţiei (instrucţiuni, date, rezultate) se memorează şi se prelucrează în formă binară. Ele trebuie să fie *interpretabile* în mod unic de către *μP*, prezentându-se din acest motiv într-un format specific. Acest format conţine o combinaţie de lungime fixă de simboluri binare care constituie *cuvinte de instrucţiuni* respectiv *cuvinte de date.*

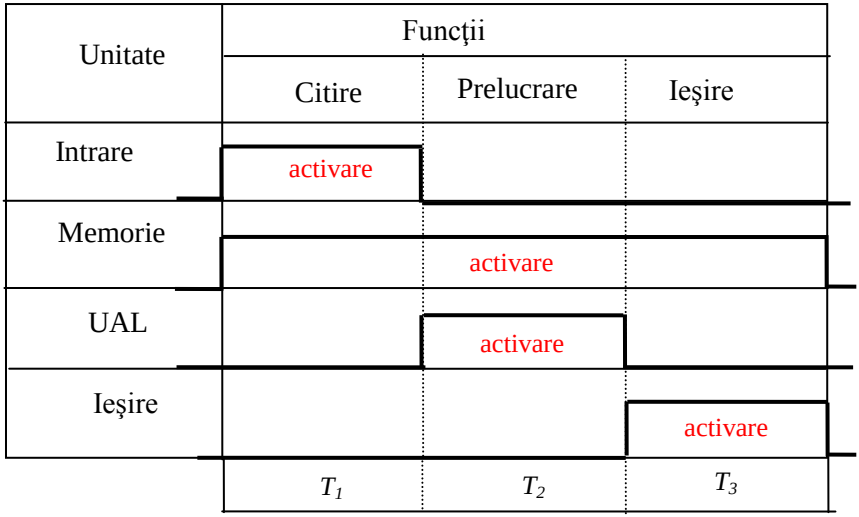


Fig. 4.10. Secvenţierea operaţiilor într-un sistem numeric cu microprocesor

În continuare se va descrie schema bloc funcţională a unui μP standard al cărui cuvânt de date are lungimea 8 biţi *(μP8).* Noţiunea *standard* se referă la faptul că *μP* îndeplineşte rolul *UCP* într-o maşină de tip von Neumann. După cum s-a arătat asemenea maşină este structurată în trei unităţi (*UAL, memorie, dispozitive de intrare / ieşire*) care comunică între ele printr-o unică magistrală cu secţiuni de *date, adrese şi comenzi*.

Presupunând existenţa unui program stocat în memorie al cărui sfârşit este marcat de o variabilă logică *SFP* iniţializată cu **1,** structura *μP8* rezultă din necesitatea execuţiei programului conform următorului algoritm:

Repetă

- adresare şi aducere din memorie a codului;
- decodificare instrucţiune;
- execuţie instrucţiune;

până când SFP=0.

Pornind de la această secvenţiere structura unui μP8 va fi descrisă pe 5 niveluri de detaliere.

4.2.2.2. Nivelul 1 de caracterizare

Acest nivel corespunde registrelor tampon de date (RTD) şi adrese (RTA). Acestea apar la interfaţa *μP* cu magistrala de date *(MD)* şi respectiv de adrese *(MA)*, figura 4.11.

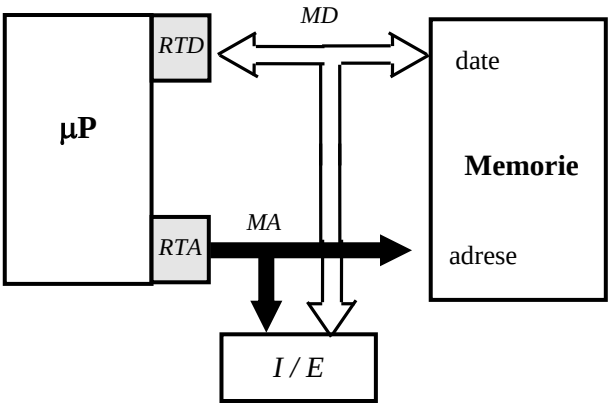


Fig. 4.11. Nivelul 1 de detaliere al unui μP8.

RTD este bidirecţional ca şi *MD* şi are lungimea egală cu a acesteia (8 biţi). O informaţie provenită din *μP* este disponibilă unităţilor conectate la *MD* numai după înscrierea acesteia în *RTD*. Invers, o informaţie destinată *μP* este accesibilă acestuia tot numai după înscrierea în *RTD*.

RTA este unidirecţional şi are lungimea impusă de caracteristicile unităţii de control a adresării memoriei. *RTA* are rolul de a menţine ferm pe *MA* adresa furnizată de *UCP* până la localizarea corectă a informaţiei în memorie sau în porturile intrare-ieşire. În acest context *portul* reprezintă o adresă de memorie care identifică circuitul fizic utilizat la transferul informaţiei între *μP* şi periferic.

Atât *RTD* cât şi *RTA* sunt transparente pentru utilizator, acestea nefiind atribute de arhitectură.

4.2.2.3. Nivelul 2 de caracterizare

Acestui nivel îi sunt asociate registrele generale (RG). Acestea reprezintă practic memoria internă a *μP* şi constituie nivelul de memorie cel mai rapid adresabilă într-un sistem. Funcţia lor este de stocare temporară a datelor (operanzi şi rezultate). După cum se observă din figura 4.12, accesul fizic pentru citire din şi pentru scriere în *RG* se face prin intermediul unui multiplexor *(MUX)* şi a unui demultiplexor *(DMUX)* cu ajutorul cărora se selectează registrul dorit.

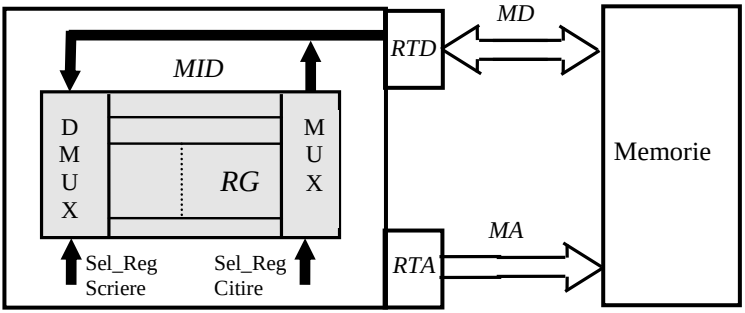


Fig. 4.12. Nivelul 2 de detaliere al unui *μP8.*

După cum se observă din figura 4.12, legătura internă dintre *RTD* şi *RG* este realizată prin *magistrala internă de date* (*MID*) care constituie o prelungire a magistralei de date (*MD*) a sistemului în interiorul *μP*. La *MID* se vor conecta toate blocurile interne care au acces la informaţia vehiculată prin *MD*.

Numărul de *RG* şi lăţimea *MID* constituie criterii de performanţă pentru orice *μP*. În ceea ce priveşte lăţimea *MID*, nu este obligatoriu ca aceasta să fie egală cu a *MD* externe. *RG* sunt în totalitate la dispoziţia utilizatorului ele constituind *elemente de arhitectură.*

De exemplu *μP* 8080[1], are un număr de 6 *RG* pe câte 8 biţi, *B,C,D,E,H,L* care pot fi utilizate ca atare dar şi în perechi, pentru a forma 3 registre de 16 biţi (*B-C* denumit registrul *B*, *D-E* denumit registrul *E şi H-L* care constituie registrul *H*.

[1] Microprocesor de referinţă pe 8 biţi realizat de firma Intel.

4.2.2.4. Nivelul 3 de caracterizare

Acest nivel corespunde unităţii aritmetice de procesare (UAP). Acest bloc funcţional, a cărui structură este prezentată în fig. 4.13, reprezintă suportul activităţii de prelucrare a datelor.

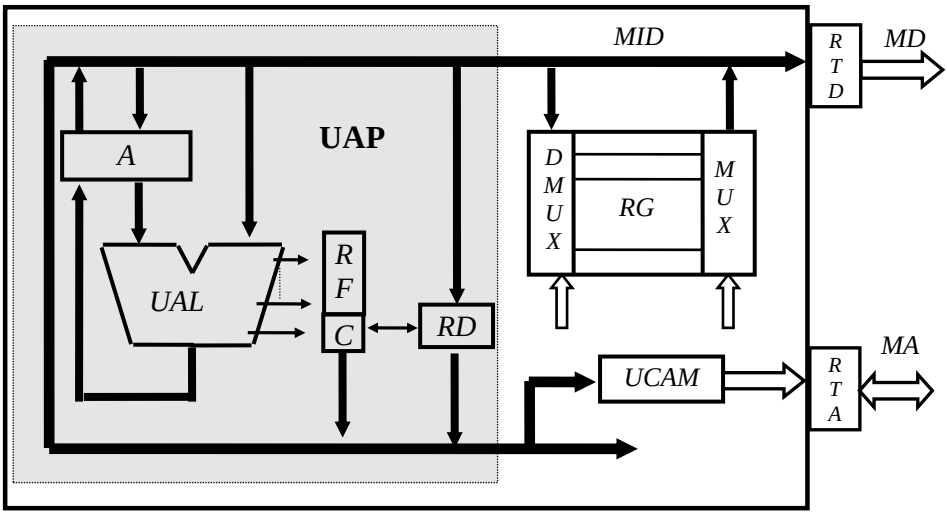


Fig. 4.13. Nivelul 3 de caracterizare a unui *μP8*:
UAL - unitate aritmetico - logică; *A* - registru acumulator; *RF* - registru al fanioanelor; *C* - fanion (indicator) de transport; *RD* - registru de deplasare; *UCAM* - unitate de control a adresării memoriei.

După cum s-a arătat, *UAL* reprezintă un circuit combinaţional care asigură realizarea unor funcţii aritmetice (*adunare, scădere, complementare faţă de 2, incrementare, decrementare, ajustare zecimală etc*.) şi logice (*SI, SAU, NICI, NUMAI, SAU EXCLUSIV, complementare faţă de 1, etc*).

Tipul şi numărul funcţiilor realizate de *UAL* constituie un criteriu de performanţă al *μP* care se reflectă într-un atribut de arhitectură şi *anume subsetul de instrucţiuni de prelucrare a datelor*. În afara intrărilor şi ieşirilor de date, UAL mai are şi intrări de selecţie a funcţiilor, care însă nu sunt reprezentate în figura 4.13.

Acumulatorul (A) este un registru asemănător cu cele din setul de registre generale. Prin definiţie *A* conţine un operand al *UAL* şi în urma efectuării prelucrării, rezultatul. Având în vedere această dublă

funcţionalitate a *A*, precum şi caracterul combinaţional al *UAL*, structura din figura 4.13 este nefuncţională datorită faptului că *UAL* se alimentează la infinit cu rezultatul propriei prelucrări. Eliminarea acestei situaţii critice se face prin includerea în structură a unui acumulator şi a unui registru, ambele temporare, evidenţiate în figura 4.14. Cei doi operanzi sunt menţinuţi în registrele temporare până când rezultatul prelucrării se înscrie în acumulator. Cele două registre temporare sunt complet transparente pentru utilizator, ele neconstituind elemente de arhitectură.

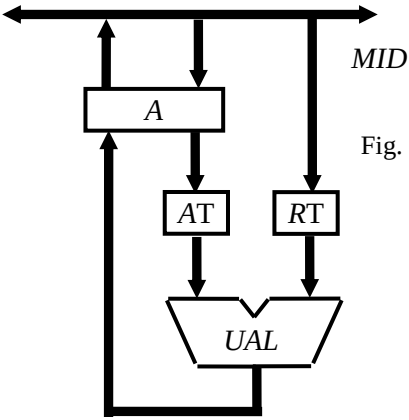


Fig. 4.14. Registre temporare aferente *UAP*:
AT - acumulator temporar;
RT - registru temporar.

Fanioanele (indicatorii de condiţie) reprezintă bistabili pentru memorarea unor condiţii speciale apăruta în funcţionarea μP şi în special a *UAL*. Ele pot fi grupate într-un registru al fanioanelor, accesibil utilizatorului.

În figura 4.15 se prezintă conţinutul acestui registru pentru *μP* INTEL 8080, reprezentativ pentru clasa μP8, în care semnificaţiile fanioanelor sunt următoarele:

- ***S*** **(***SIGN***)** - are valoarea **1** dacă rezultatul prelucrării din *A* este pozitiv (bitul cel mai semnificativ este **1**), altfel *S* are valoarea **0**;

- ***Z*** **(***ZERO***)** - este pus în **1** dacă în urma prelucrării, *A* conţine valoarea **0**, altfel Z este **0**;

- ***P*** **(***PARITY***)** - are valoarea **1** dacă în urma execuţiei unei instrucţiuni, *A* conţine un număr par de **1**, altfel P este **0**;

- ***C*** **(***CARRY***)** - este **1**, dacă din *A* s-a efectuat un transport, altfel C este **0**.

- ***AC*** **(***AUXILIARY CARRY***)**[2] - este **1** în condiţiile existenţei unui transport dinspre bitul 3 spre bitul 4 al *A*, altfel *AC* este **0**.

7	6	5	4	3	2	1	0
S	*Z*	-	*AC*	-	*P*	-	*C*

Fig. 4.15. Structura registrului fanioanelor pentru μP 8080.

Fanioanele joacă un rol important în structurarea programelor, întrucât instrucţiunile de salt condiţionat testează starea acestora. *RF* constituie element de arhitectură, iar numărul de fanioane un criteriu de performanţă pentru μP. Uzual *A* şi *RF* se asamblează într-un unic registru de 16 biţi cunoscut sub denumirea de *registru* ***PSW*** (***P***rogramm ***S***tatus ***W***ord).

Registrul de deplasare (RD) permite realizarea rapidă a operaţiilor de înmulţire (deplasare dreapta) şi împărţire (deplasare stânga) cu puteri ale lui 2. Deplasările presupun de regulă salvarea bitului extrem (0 sau 7) în fanionul *C*. O variantă interesantă o reprezintă *rotaţia* în care conţinutul fanionului *C* este înscris în celălalt bit extrem al *RD*. În figura 4.16 sunt evidenţiate cele două maniere de realizare a deplasării.

RD nu este un atribut de arhitectură, el este implicit utilizat de instrucţiunile specifice ale μP, dar nu are acces nemijlocit la acest registru special.

O secvenţă de utilizare implicită a *RD* ar putea fi următoarea:

1 - se selectează registrul care conţine operandul;

2 - conţinutul acestuia este adus pe *MID*;

3 - se înscrie operandul în *RD*;

4 - se comandă acestuia funcţia de deplasare dorită;

5 - conţinutul *RD* (deplasat) este transferat pe *MID*;

6 - se selectează din nou registrul de la pasul ***1***;

7 - se înscrie în acesta rezultatul operaţiei de deplasare care ia locul operandului iniţial.

[2] Indicatorul *AC* este utilizat pentru corecţia zecimală la adunarea a două numere în cod *BCD* în sensul celor prezentate în Capitolul 2.

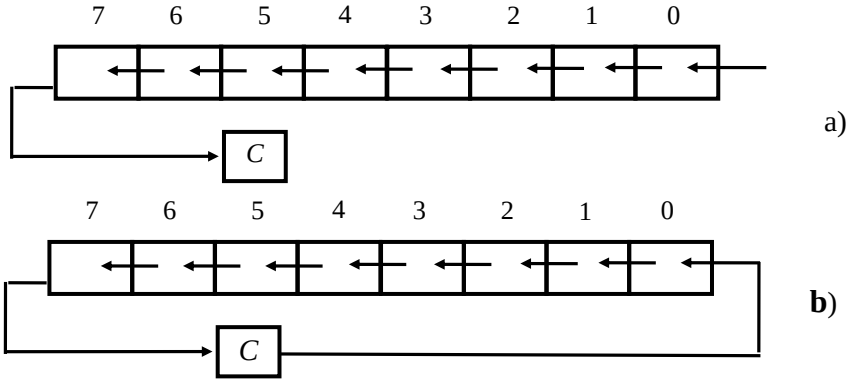
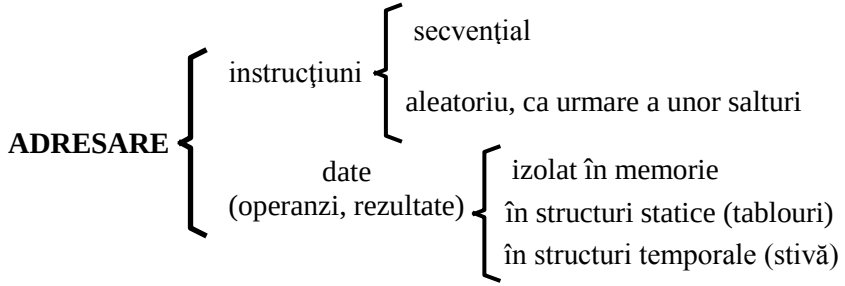


Fig. 4.16. Modalităţi de realizare a deplasării spre stânga:
a - simplă; b- rotaţie.

Funcţionarea *RD* este posibilă, dacă acesta este conectat la fanionul *C*, aspect evidenţiat şi de fig. 4.13.

4.2.2.5. Nivelul 4 de caracterizare

Acest nivel este dedicat unităţii de control a adresării memoriei (UCAM). Această unitate realizează încărcarea unei adrese în *RTA* în vederea localizării unei informaţii în memoria sau porturile calculatorului. Formarea adresei este în strânsă concordanţă cu modul de parcurgere a programului şi cu localizarea codului şi operanzilor, aşa cum reiese din reprezentarea de mai jos.



Realizarea acestor funcţii, implică existenţa în structura UCAM, în totalitate sau nu, a următoarelor elemente principale:

- numărător de program;
- indicator de stivă;
- registru index,

pentru care în figura 4.17 se prezintă o posibilitate de interconectare.

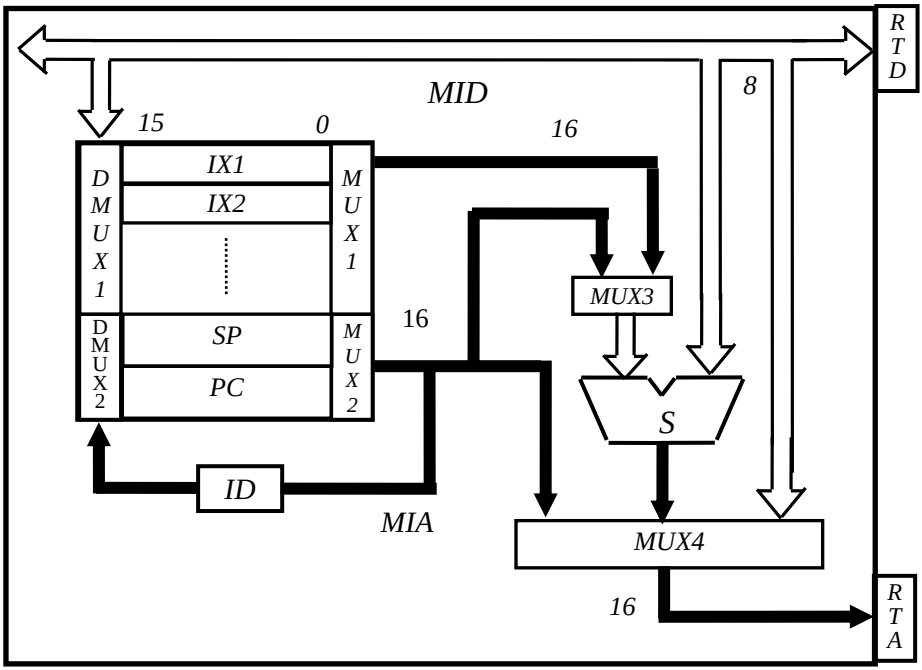


Fig. 4.17. Structura posibilă a unei unităţi de control a adresării memoriei: *PC* – registru numărător de program; *SP* - registru indicator de stivă; *IX* – registre index; *S* - sumator; *ID* - circuit de incrementare/decrementare; *MUX, DMUX* – multiplexoare, demultiplexare; *MIA* - magistrală internă de adrese.

- *Numărătorul de program (PC-Programm Counter)* conţine adresa fizică (*AF)* a instrucţiunii care va fi executată. *AF* reprezintă forma sub care se furnizează o adresă pe magistrala de adrese pentru a se face identificarea fizică a locaţiei de memorie vizate.

Lungimea *PC* impune capacitatea maximă a memoriei ce poate fi adresată în sistem. Uzual pentru un *μP8*, *PC* are 16 biţi de unde rezultă o hartă a memoriei direct adresabile de 2^{16}*=65536 octeţi = 64KB (1 Koctet = 1 Kbyte = 1024 octeţ*i).

Adresarea secvenţială a memoriei presupune transmiterea conţinutului registrului *PC* (respectiv a adresei locaţiei adresate) pe calea *MUX2 – MUX4* la *RTA*, urmată de incrementarea *PC* de către circuitul *ID*.

Pentru efectuarea unui salt *RTA* se încarcă prin *MUX4* cu noua adresă (*care însoţeşte codul instrucţiunii ce urmează a se executa)*. Concomitent *PC* se va încărca prin DMUX2 cu aceeaşi adresă, de unde îşi va continua funcţionarea secvenţială.

PC nu este un atribut de arhitectură, programatorul neputând modifica direct conţinutul acestuia.

- *Indicatorul de stivă (SP - Stack Pointer).* După cum s-a arătat stiva (*Stack*) reprezintă o structură de date utilizată pentru păstrarea temporară a datelor, organizată pe principiul ***LIFO*** (**L**ast **I**nput - **F**irst **O**utput — ultimul intrat - primul ieşit). Poziţia ocupată în stivă de ultimul element introdus constituie vârful stivei, încărcarea şi descărcarea acesteia putându-se efectua numai prin acest punct.

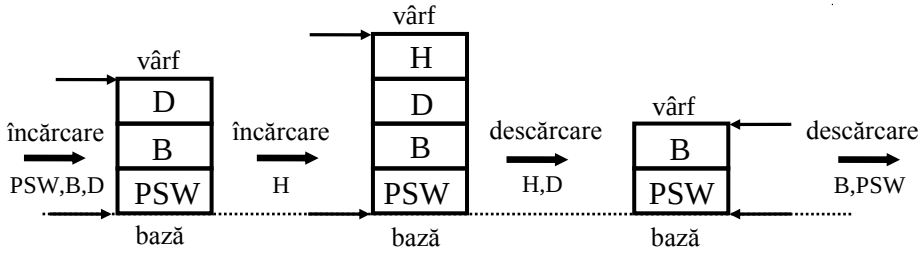


Fig. 4.18. Încărcarea şi descărcarea unei stive:
PSW - registrul stare program; *B,C,D* - registre duble.

Încărcarea şi descărcarea stivei se realizează prin instrucţiuni specifice (*PUSH* pentru scriere în stivă, *POP* pentru extragere din stivă). De exemplu pentru situaţia din figura 4.18 succesiunea de instrucţiuni este următoarea:

PUSH PSW;	înscrie în stivă conţinutul *PSW*
PUSH B;	idem *B*
PUSH D;	idem *D*
PUSH H;	idem *H*
.	
.	
POP H;	extrage din stivă conţinutul *H*
POP D;	idem *D*
POP B;	idem *B*
POP PSW;	idem *PSW*

(reamintim că *PSW, B, D, H* sunt registre duble pe câte 16 biţi).

SP conţine adresa curentă a vârfului stivei (prima locaţie liberă) . Configurarea unei zone de memorie ca stivă se face prin înscrierea în *SP* a adresei bazei, respectiv a adresei de la care începe stiva. Stiva se organizează astfel încât creşterea ei să se facă "în jos" adică în sensul descreşterii adreselor[3]. La fiecare înscriere în stivă, *SP* este *decrementat* iar la fiecare extragere, acesta este *incrementat*, astfel încât în orice moment va conţine adresa primei locaţii disponibile din memoria stivă. Incrementarea/ decrementarea *SP* este realizată cu ajutorul circuitului *ID* iar încărcarea lui *RTA* cu adresa din *SP* se face pe calea *MUX2 – MUX4.* Figura 4.19 constituie o ilustrare a operaţiunilor aferente lucrului cu stiva.

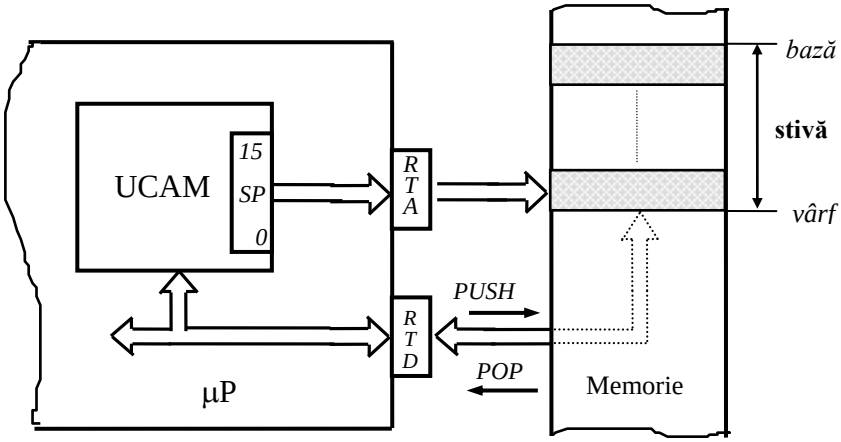


Fig. 4.19. Lucrul cu stiva.

[3] Această tehnică este utilizată pentru a nu se putea depăşi spaţiul declarat pentru stivă.

În afara stocării temporare a datelor, stiva mai este utilizată în mecanismele de apelare a subprogramelor şi de răspuns la cererile de întrerupere. Registrul *SP* constituie un element de arhitectură, utilizatorul având posibilitatea să definească iniţial baza stivei. Există *µP* la care stiva este implementată *hardware* cu ajutorul unor registre speciale. Acest tip de stivă prezintă avantajul accesului rapid şi dezavantajul limitării severe a mărimii. În cazul stivei hardware *SP* devine transparent pentru utilizator întrucât nu mai este necesară o definire a bazei stivei.

- *Registrele index* sunt opţionale în structura unui µP8 standard şi permit localizarea rapidă a informaţiei într-un bloc pe baza adresei fizice, care se poate obţine prin efectuarea unei operaţii de adunare:

$$AF_{element} = AF_{bază} + deplasament,$$

unde *AF* reprezintă adresa fizică. Această operaţie este realizată în sumatorul *S* iar cei doi termeni sunt furnizaţi de un registru index (adresa de bază pe calea *MUX2 - MUX4*) şi de *MID*. Deplasamentul reprezintă adresa relativă a unui element în cadrul tabloului şi însoţeşte codul instrucţiunilor care folosesc date astfel structurate. Mărimea deplasamentului indică dimensiunea maximă a tabloului ce poate fi construit în memorie. Uzual pentru µP8 deplasamentul are *8 biţi,* astfel încât se pot construi maxim $2^8 = 256$ elemente. Adresarea indexată permite localizarea rapidă printr-o singură instrucţiune a unui element de tablou şi creează premisele dezvoltării de noi metode pentru obţinerea adresei fizice prin calcul.

Sintetic *UCAM* realizează:

- adresarea secvenţială care se realizează prin încărcarea *RTA* cu adresa din *PC*;
- salturi, prin încărcarea *RA* cu o adresă furnizată pe *MID* (adresă cu care se încarcă evident şi *PC*);
- adresarea datelor elementare prin încărcarea *RTA* cu adresa ce însoţeşte codul instrucţiunii (această adresă nu se înscrie în *PC*);
- adresarea datelor elementare în structuri de tip tablou (adresa are două componente: bază şi deplasament);
- accesul în stivă (*RTA* se încarcă cu adresa din *SP*).

4.2.2.6. Nivelul 5 de caracterizare

Acest nivel este asociat unităţii de control a μP (UCμP8). După cum s-a arătat execuţia unui program comportă pentru fiecare instrucţiune parcurgerea următoarelor etape: *localizare, decodificare, execuţie propriu-zisă.*

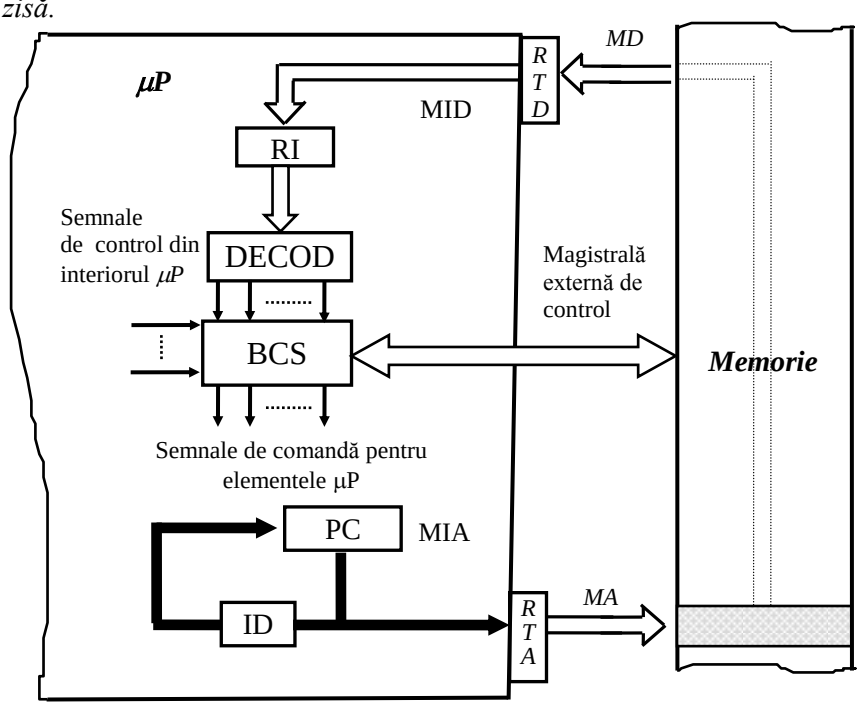


Fig. 4.20. Elemente implicate în extragerea unei instrucţiuni:
RI - registru de instrucţiuni; *PC* - numărător de program; *DECOD* – decodificator instrucţiuni. *BCS* – bloc de control şi sincronizare ;
ID – circuit de incrementare/decrementare.

1. *Localizarea* şi aducerea în memorie a unei instrucţiuni, (la care participă elementele ilustrate în figura 4.20), implică parcurgerea următoarelor faze:

- încărcarea *RTA* cu adresa din *PC* (în cazul uzual al parcurgerii secvenţiale a programului), în urma căreia adresa devine disponibilă pe *MA*;

- incrementarea *PC* pentru a se crea posibilitatea accesării următoarei locaţii de memorie;

- generarea pe magistrala de control a unui semnal de citire din memorie (*READ*);

- transferul pe *MD* şi de aici în *RTD* a conţinutului locaţiei identificate;

- transferul codului instrucţiunii, din *RTD* prin intermediul *MID*, într-un registru denumit *registru de instrucţiuni (RI).*

2. *Decodificarea* presupune recunoaşterea şi interpretarea conţinutului RI urmată de iniţierea acţiunilor aferente execuţiei.

3. *Execuţia* propriu-zisă implică activarea diverselor blocuri ale µP într-o ordine prestabilită şi/sau schimburi de informaţie cu memoria şi/sau porturile de intrare - ieşire.

Secvenţa de acţiuni elementare este dependentă de semantica fiecărei instrucţiuni. Coordonarea derulării în timp a fiecărei etape şi faze este asigurată de către *UCµP8*. O primă sarcină a acesteia o reprezintă stabilirea formatului instrucţiunii în funcţie de codul primit. Se are în vedere desfăşurarea în locaţii de memorie succesive a întregii informaţii necesare execuţiei instrucţiunii. De exemplu, pentru µP 8080 al cărui repertoriu conţine 256 instrucţiuni, formatul unei instrucţiuni poate conţine:

- obligatoriu pe primul octet codul instrucţiunii;

- opţional un operand pe 8 biţi, jumătatea inferioară a unui operand sau adrese pe 16 biţi;

- opţional jumătatea superioară a operandului sau a adresei pe 16 biţi.

Cu alte cuvinte, o instrucţiune a acestui *µP* poate ocupa *1, 2 sau 3 octeţi* în memorie. *UCµP8* decide dacă este cazul să mai aducă din memorie 1 sau 2 octeţi sau să declanşeze imediat după primirea codului execuţia instrucţiunii.

Având în vedere caracterul de automat sincron al *µP*, desfăşurarea în timp a etapelor şi fazelor unei instrucţiuni este nemijlocit legată de frecvenţa impulsurilor de sincronizare. Legat de execuţia în timp a instrucţiunilor se definesc următoarele noţiuni:

- *starea* ca timp maxim de efectuare a unei acţiuni elementare, reprezintă o durată egală cu perioada impulsurilor de sincronizare;

- *ciclul maşină* grupează mai multe acţiuni elementare în vederea conturării unei etape din execuţia unei instrucţiuni. Un ciclu maşină are mai multe stări şi are o semnificaţie strict funcţională fiind impus de necesităţi de sistematizare a activităţii a µP.

Atribuţiile legate de supervizarea funcţionării corecte a ansamblului de elemente reunite în structura unui *μP* impun prezenţa în cadrul UCμP8 a elementelor evidenţiate în figura 4.20. *RI* este conectat la *MID*, de unde preia codul instrucţiunii curente pe care îl transmite decodificatorului. Acesta identifică instrucţiunea din setul de instrucţiuni potenţial executabile de către *μP*. Informaţia rezultată la ieşirea decodificatorului este transmisă blocului de control şi sincronizare (*BCS*). Acesta este un automat finit microprogramat, care generează semnale de comandă pentru elementele implicate în execuţia instrucţiunii curente.

Microprogramul *BCS* va trebui să ţină cont, printre altele de:

- setul de instrucţiuni al *μP*;
- semantica fiecărei instrucţiuni;
- sistematizarea desfăşurării în timp a fiecărei instrucţiuni pe stări şi cicluri maşină;
- formatul fiecărei instrucţiuni;
- structura fizică concretă a blocurilor μP;
- semnalele de control necesare sau impuse μP (in interiorul sau din exteriorul său pe magistrala externă de comenzi).

BCS nu constituie un element de arhitectură şi prin urmare utilizatorul nu are acces la microprogramul acestuia. Datorită acestui fapt μP standard nu pot fi adaptate exact cerinţelor unei sarcini concrete prin optimizarea la nivel de microprogram a instrucţiunilor sale.

4.2.3. Elemente de arhitectura a microprocesoarelor pe 16 biţi

Din analiza efectuată pentru *μP8* a reieşit că pe lângă neajunsul unei lungimi de numai 8 biţi a *MD*, acestea sunt caracterizate de absenţa oricărui paralelism în realizarea celor trei faze aferente execuţiei unei instrucţiuni. Aceasta înseamnă că practic extragerea codului pentru o nouă instrucţiune nu se poate face decât după încheierea execuţiei instrucţiunii curente.

Lipsa paralelismului face ca *μP8* să prezinte următoarele particularităţi importante:

a) secţiunea resurselor de prelucrare nu este utilizată decât în faza de execuţie;
b) magistrala de adrese a este utilizată doar în faza de extragere.

La generaţia a III-a de *μP* caracterizate în primul rând de o lăţime a MD de 16 biţi *(μP16*) apar primele elemente de paralelism între fazele de *extragere şi execuţie,* ilustrat în figura 4.21.

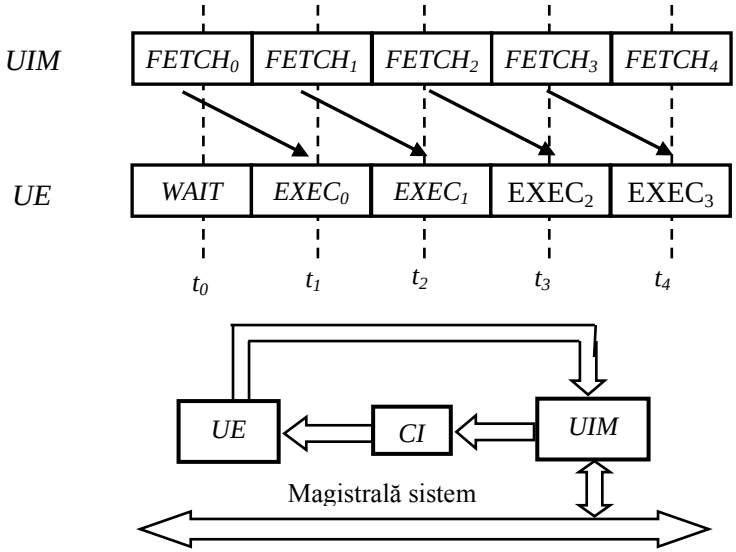


Fig. 4.21. Ilustrarea paralelismului la *μP16*.

Deosebirea esenţială faţă de *μP8* o constituie existenţa a două *procesoare specializate* care lucrează în paralel, pe care firma Intel, care a lansat primul *μP16,* le-a denumit *unitate de execuţie (UE)* **şi** *unitate de interfaţă cu magistrale (UIM).*

După cum se observă din figura 4.21 la momentul t_i *UIM* realizează operaţia $FETCH_i$ iar *UE* operaţia $EXEC_{i-1}$ **(**execută codul extras la tactul anterior).

Pentru structura principială a unui *μP16,* reprezentată în figura 4.22, rămân valabile o parte din consideraţiile făcute pentru *μP8*, astfel încât în cele ce urmează vor fi prezentate numai *aspecte specifice.*

❖ ***UE*** are ca principală sarcină execuţia instrucţiunilor, pe care împreună cu operanzii le primeşte prin intermediul *UIM* şi nu direct din memorie. Rezultatele prelucrării sunt trimise în memorie sau la porturi tot prin intermediul *UIM*.

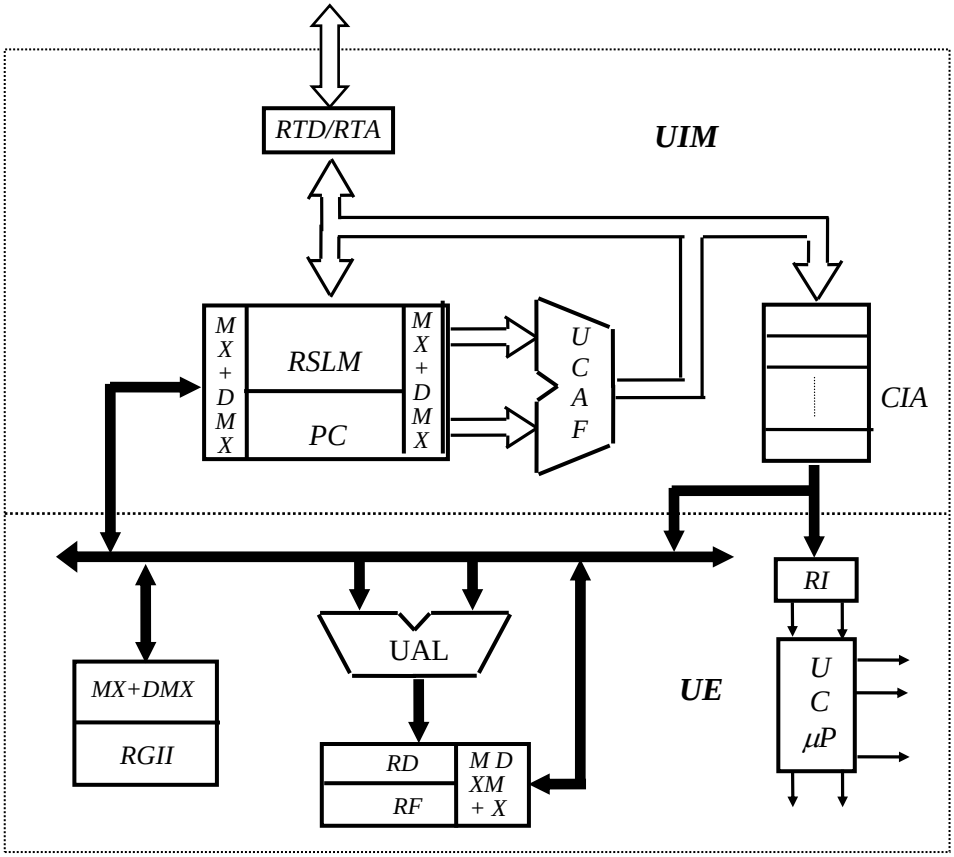


Fig. 4.22. Schema funcţională de structură a unui *μP16* general:
UIM - unitate de interfaţă cu magistrala; *UE* - unitate de execuţie; *RSLM* - registre pentru structurarea logică a memoriei; RTD/RTA – registru tampon de date şi de adrese; *PC* - numărător de program; *UCAF* - unitate de calcul a adresei fizice; *CIA* - coadă de instrucţiuni în aşteptare; *MUI* - magistrala unităţii de interfaţare a magistralei; *MUE* - magistrala unităţii de execuţie; *MX+DMX*- multiplexoare şi demultiplexoare; *RGII* - registre generale indicator şi index; *RI* - registru al instrucţiunilor; *RD* - registru de deplasare; *RF* - registru al fanioanelor; *UAL* - unitate aritmetico-logică; *UCμP* - unitatea de control a *μP*.

❖ ***UIM*** are drept scop mărirea cantităţii de informaţie vehiculată pe magistralele externe în unitatea de timp. Printre altele, această funcţie presupune: furnizarea adreselor pentru instrucţiuni şi pentru date, calculul adreselor, realizarea structurării logice a memoriei, încărcarea cozii cu instrucţiuni, de unde vor fi preluate şi executate de către *UE*.

❖ ***Setul de registre generale*** este completat cu registre de tip indicator şi index. Pentru fiecare din aceste registre există atât o utilizare implicită sugerată de fabricant, cât şi una alternativă. În contextul versatilităţii registrelor generale, *UAL* nu mai are asociat un registru acumulator dedicat, oricare din RG putând îndeplini acest rol.

❖ ***UCAM*** nu mai apare ca un bloc unitar, funcţiile fiind descentralizate astfel:

- registrele indicator şi index se găsesc în *UE*;
- numărătorul de program este asociat unui bloc de registre destinat structurării logice a memoriei;
- în *UE* apare un bloc special pentru calculul adreselor (UCAF) care dezvoltă sumatorul destinat acestui scop în structura unui µP8.

❖ **Paralelismul în funcţionare** al *UE* şi *UIM* este asigurat de un bloc de registre asociat *cozii de instrucţiuni* . Aceasta (organizată pe principiul *FIFO*) se alimentează de către *UIM* şi se descarcă în *UE*.

❖ **Existenţa unei magistrale externe unice** multiplexate conduce la existenţa unui unic registru tampon de date şi de adrese.

Sintetic, saltul calitativ care va conferi *µP16* şi noi atribute de arhitectură are în vedere următoarele aspecte:

- existenţa a două procesoare care lucrează în paralel;
- versatilitatea funcţiilor registrelor;
- existenţa blocului pentru calculul adreselor;
- existenţa cozii de instrucţiuni;
- posibilitatea de structurare logică a memoriei.

4.3. Memoria calculatoarelor

După cum s-a arătat, funcţionarea oricărui sistem de calcul este condiţionată de existenţa *memoriei*, destinată păstrării *datelor ce urmează a fi prelucrate, a programelor şi a rezultatelor.* Pe lângă funcţia intrinsecă menţionată, memoria este implicată în toate fazele aferente execuţiei unei aplicaţii codificate sub forma unui program. Informaţia transmisă calculatorului prin intermediul unităţilor de intrare este păstrată în *memorie*. De aici informaţia se transferă altor module funcţionale în vederea prelucrării. Rezultatele parţiale sau finale au destinatar *memoria* de unde pot deveni disponibile utilizatorului prin intermediul echipamentelor periferice de ieşire.

Spaţiul de memorie al unui calculator poate fi văzut ca un şir de locaţii consecutive, fiecare locaţie fiind identificabilă prin intermediul unei *adrese*. Din punct de vedere fizic memoria este formată din elemente care prezintă *două* stări cărora li se asociază simbolurile *0 şi 1* numite cifre binare. În cele ce urmează vor fi prezentate principalele trăsături funcţionale ale tipurilor de memorie aferente fiecărui nivel al subsistemului memorie.

4.3.1. Structura ierarhică a memoriei

Organizarea ierarhică a memoriei implică o structurare piramidală a acesteia pe patru niveluri, ilustrate în figura 4.23. Din analiza acestei figuri reiese că de la vârf spre bază creşte dimensiunea memoriei concretizată în *capacitatea de memorare*[4]. Acelaşi mod de variaţie este specific şi *timpului de acces* [5] în timp ce *costurile de memorare pe bit* cresc de la baza spre vârful piramidei.

[4] *Capacitatea* reprezintă numărul total de locaţii aferent unui nivel ierarhic al subsistemului memorie. În mod obişnuit mărimea unei locaţii de memorie este de *un octet.*

[5] *Timpul de acces* reprezintă intervalul dintre momentul lansării adresei şi cel în care informaţia referită devine disponibilă.

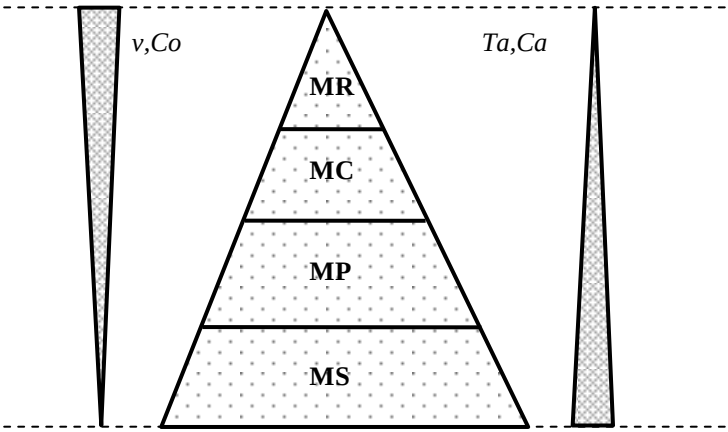


Fig. 4.23. Structura ierarhică piramidală a memoriei în cadrul unui sistem de calcul: *MR* – memorie în registre; *MC* - cache; *MP* - principală; *MS* - secundară; *Ta* - timp de acces; *Ca* - capacitate; *v* - viteză; *Co* - cost.

- *Registrele interne* ale *UCP (MR)* reprezintă cea mai rapidă şi cea mai mică memorie din sistem. *Registrele* sunt conectate direct la unităţile funcţionale de prelucrare cum ar fi *UAL*, unitatea de calcul a adresei etc. *Registrele generale* constituie nivelul *0* al memoriei de date, iar registrul de instrucţiuni reprezintă acelaşi nivel pentru memoria de cod.

- *Memoria cache (MC)* îmbunătăţeşte viteza de procesare, deoarece conţine perechi adresă-dată. Din punctul de vedere al capacităţii şi timpului de acces memoria cache se situează între registre şi memoria primară.

- *Memoria principală (primară, internă) (MP)* este memoria în care se află programele sau secvenţe din programele active şi datele necesare lor, precum şi o parte a sistemului de operare. Secţiunile inactive ale programului, datele aferente şi restul sistemului de operare se află în nivelul următor, cel secundar al memoriei.

- Memoria secundară *(MS)* conţine copii ale tuturor programelor şi datelor necesare sistemului. Dispozitivul fizic ce implementează acest nivel al memoriei este discul magnetic, sau un alt dispozitiv orientat pe transfer de date la nivel de bloc.

Pentru păstrarea programelor şi datelor un timp îndelungat poate fi identificat un subnivel al memoriei secundare denumit *memorie*

suplimentară. Acestui nivel îi sunt specifice de asemenea medii magnetice şi optice cum ar fi:, casete cu bandă magnetică, *CD-ROM, DVD-ROM,* etc.

4.3.2. Memoria principală

Memoria principală *(MP)*conţine informaţia aferentă programului în execuţie şi date cu care operează acesta. Pentru a se realiza o viteză ridicată de transfer *MP* este conectată direct la magistralele sistemului, după cum reiese din figura 4.24. Adresa locaţiei referite este păstrată în ***R**egistrul **I**ntern de **A**drese al **M**emoriei (RIAM),* iar datele citite/scrise din/în memorie se păstrează în ***R**egistrul **I**ntern de **D**ate (RIDM).* Prezenţa acestor registre este absolut necesară deoarece magistralele nu trebuie reţinute pe toată durata transferului în/din memorie. Circuitele de control ale memoriei (*CCM*) generează semnalele de selecţie şi comandă a modulelor de memorie (reprezentate în figura 4.24 cu linie întreruptă).

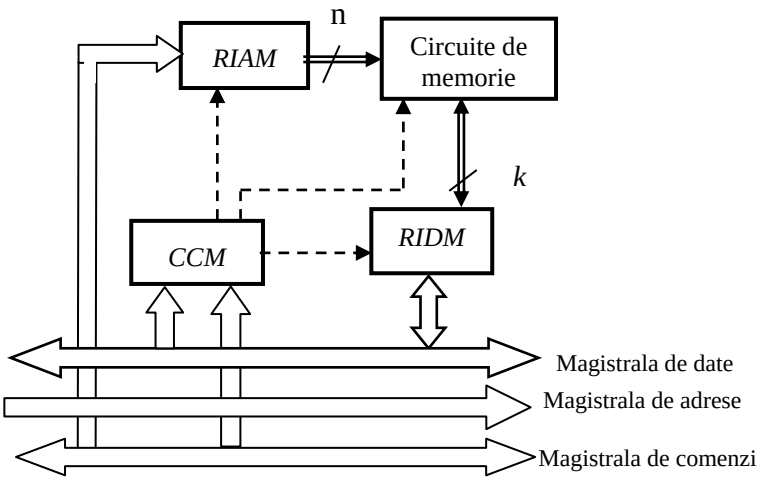


Fig. 4.24. Schema bloc a unei unităţi de memorie.

MP este realizată în diverse tehnologii cum ar fi: *circuite semiconductoare, (RAM, ROM, EPROM), inele de ferită, bule magnetice, etc.* Din punctul de vedere al accesului, memoria principală conţine secţiuni care pot fi atât citite cât şi înscrise şi secţiuni care pot fi numai citite. În continuare se vor prezenta trăsăturile importante ale celor două tipuri de memorie.

Memoria cu acces aleator ***RAM*** *(**R**andom **A**ccess **M**emory)* poate fi accesată atât pentru citire, cât şi pentru scriere. Întrucât informaţia memorată se pierde la deconectarea tensiunii de alimentare acest tip de memorie se numeşte *volatilă.* Memoria *RAM* stochează *informaţii variabile*, în sensul posibilităţii de modificare de către *UCP*, sub controlul programului, a conţinutului oricărei locaţii. Memoria *RAM* poate fi de două feluri: *statică şi dinamică.*

Memoria RAM statică utilizează drept celule elementare de un bit circuite basculante de tip bistabil şi se caracterizează prin persistenţa informaţiei înscrise, în sensul că nu este necesară o reîmprospătare periodică a acesteia. Din punctul de vedere al consumului energetic, acesta este ridicat însă timpul de acces este scurt comparativ cu memoriile *RAM* dinamice. Un modul de memorie *RAM* (*chip*) *static* prezintă structura principială evidenţiată în figura 4.25.

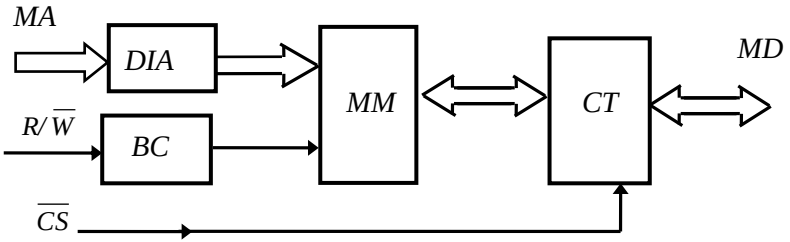


Fig. 4.25. Structura unui modul RAM static:
DIA - decodificator intern de adrese; *BC* - bloc de control; *MM* - matrice de memorie; *CT* - circuit tampon; *MA* - magistrală de adrese; *MD* - magistrală de date; $R/\overline{W}$ - semnal de citire / scriere; $\overline{CS}$ - semnal de selecţie.

DIA permite pe baza adresei furnizate de *UCP* selecţia unică a celulei de memorie referite, pentru care semnalul $R/\overline{W}$ va indica sensul transferului de informaţie. Corespunzător sensului de transfer, se va comanda circuitului tampon *CT* preluarea datelor de pe *MD* sau plasarea acestora pe *MD.* Semnalul $\overline{CS}$ (*Chip Select*) permite activarea modulului în cazul în care acesta este integrat într-o structură complexă.

Memoria RAM dinamică reţine informaţia prin încărcarea cu sarcină electrică a capacităţii porţii unui tranzistor *MOS*. Datorită descărcării în timp a capacităţii, informaţia trebuie supusă periodic unei operaţii de reîmprospătare (*refresh*). Reîmprospătarea, a cărei frecvenţă depinde de constanta de timp a circuitului (uzual 1-2 ms) constă în

efectuarea unei operaţii de citire. În raport cu memoria *RAM* statică, memoria *RAM* dinamică prezintă avantajul unui consum energetic redus şi al unei densităţi ridicate pe chip (datorită numărului redus de tranzistori pe celulă).

Structura de principiu a memoriei *RAM* dinamice este asemănătoare cu a memoriilor *RAM* statice în ceea ce priveşte organizarea şi conectarea celulelor. În principiu se menţine structura de tip matriceal, o celulă fiind identificată prin linie (*row*) şi coloană (*column*). Pentru a scurta timpul destinat reîmprospătării, acesta se realizează ori de câte ori se selectează o linie.

Memoria ROM Aceste memorii, care pe parcursul execuţiei programului nu pot fi înscrise, se împart în trei categorii şi anume:

- memorii *ROM* care nu pot fi înscrise de utilizator;

- memorii *ROM* programabile o singură dată de către utilizator (*PROM-Programmable ROM*);

- memorii *ROM* care pot fi scrise în mod repetat de către utilizator *(EPROM - Erasable PROM*).

Memoriile ROM propriuzise sunt programate de către producător conform funcţiei indicate de utilizator. Acest tip de memorie este rentabil în condiţiile unei producţii de mare serie. Cel mai cunoscut exemplu de utilizare a memoriei *ROM* este în calitate de suport al secţiunii *BIOS* [6] a sistemului de operare.

Memoriile PROM pot fi programate de către utilizator prin distrugerea selectivă, conform informaţiei ce se memorează a unor microfuzibile situate în nodurile matricei de memorie Având în vedere ireversibilitatea înscrierii, acest procedeu a fost denumit *ardere* a programului iar dispozitivul aferent *arzător de memorii PROM*.

Memoriile EPROM oferă posibilitatea programării repetate. Acestea reţin informaţia ca o sarcină electrică într-o celulă *MOSFET*, putând fi şterse prin expunerea la o sursă de radiaţii ultraviolete sau prin metode electrice. Memoriile *Flash ROM* care sunt utilizate din ce în ce mai mult ca suporturi amovibile sunt memorii *EEPROM* care permit ştergerea selectivă**.**

[6] *BIOS* (*Basic Input Output System*). furnizează servicii esenţiale pentru funcţiile de intrare – ieşire ale sistemului de calcul.

4.3.3. Memoria cache

Memoria cache (MC) îmbunătăţeşte viteza de procesare, deoarece conţine perechi adresă-dată. Din punctul de vedere al capacităţii şi timpului de acces memoria cache se situează între registre şi memoria primară.

În momentul în care *UCP* doreşte să acceseze memoria lansează adresa pe magistrala de adrese. Această adresă este interceptată de circuitul de control al *MC* care în cazul în care o găseşte între perechile memorate transmite data direct către microprocesor, situaţia fiind apreciată ca *nimerire de cache (cache hit).*

Dacă adresa cerută nu este identificată în *MC (ratare de cache - cache miss),* atunci această adresă este transferată memoriei principale. La apariţia datei pe magistrala de date, aceasta este înscrisă alături de adresa sa şi în *MC,* astfel încât la o nouă referire să poată fi găsită rapid. Un parametru important al *MC* este rata de nimerire (*hit rate*), adică procentul de încercări de citire reuşite din totalul încercărilor.

4.3.4. Memoria secundară

Memoria secundară *(MS)* este constituită din suporturile aferente unor dispozitive periferice care asigură o viteză de transfer ridicată, dar mult mai mică decât viteza de lucru a memoriei principale. Această viteză redusă este compensată de o capacitate de memorare peste cea a *MP*. Dintr-un anumit punct de vedere se poate considera că *MS* are practic o capacitate nelimitată (de exemplu capacitatea memoriei realizate pe discuri optice depinde de dimensiunea arhivei de discuri, respectiv de numărul discurilor utilizate ca suport de memorare.

În continuare se vor face scurte consideraţii referitoare la unele tipuri de suporturi specifice *MS*.

- *Banda magnetică (**M**agnetic **T**ape - **MT**),* care a fost printre primele suporturi utilizate ca *MS,* reprezintă practic o panglică confecţionată dintr-un material plastic numit *mylar* care este acoperit cu un material magnetic. Informaţia este organizată pe 9 piste longitudinale dintre care 8 pentru date şi una pentru bitul de paritate transversală, citirea şi scrierea datelor fiind realizate de către nouă capete magnetice.

În cazul benzii magnetice accesul este secvenţial iar cantitatea minimă de informaţie ce se poate transfera printr-o operaţie de citire/scriere este cea conţinută într-o înregistrare fizică. Accesul secvenţial presupune citirea tuturor înregistrărilor situate între cea curentă şi cea referită, cu

consecinţe directe asupra timpului de acces care nu este constant ci poate varia între fracţiuni de secundă şi sute de secunde. Cu toată această viteză redusă, banda magnetică este utilizată pentru păstrarea unor arhive de date (de exemplu salvările de date care se fac periodic în sistemele informatice – *backup*).

- *Discul magnetic rigid (**H**ard **D**isk - **HD**)* este format dintr-un disc din metal acoperit cu material magnetic pe una sau ambele feţe. Mai multe discuri fixate pe un ax formează o *pilă* de discuri (figura 4.26). Pe fiecare disc informaţia este organizată pe un număr de cercuri concentrice numite *piste*. Pistele cu acelaşi diametru de pe toate discurile, formează un *cilindru.*

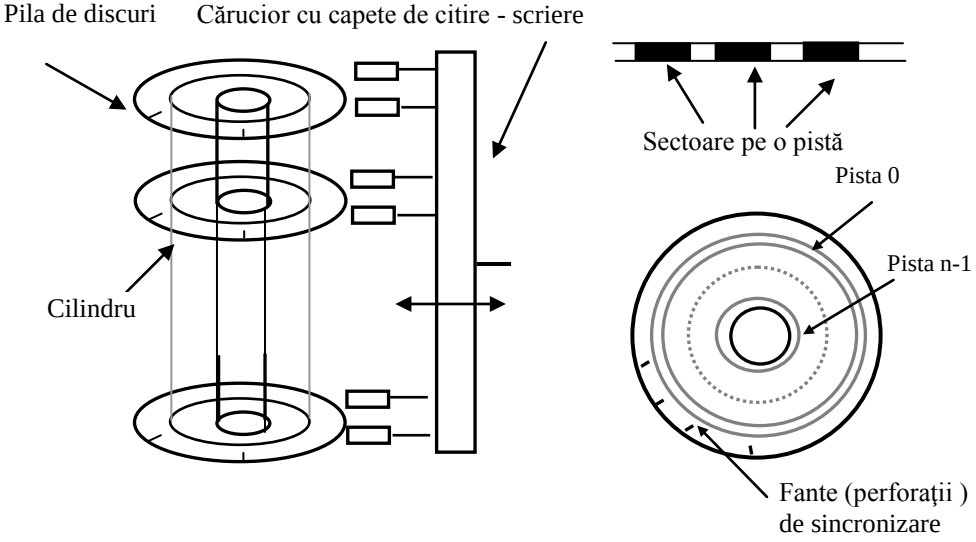


Fig. 4.26. Structura principială a unui disc rigid şi organizarea informaţiei pe disc.

Pistele sunt împărţite în *sectoare*, care reprezintă blocurile fizice în care se codifică datele. Un sector are un identificator, respectiv un număr unic prin care sectorul este referit. Un disc conţine între 40 şi câteva sute de piste, o pistă conţine între 10 şi 100 de sectoare, iar un sector între 128 şi 1024 de octeţi (tipic 512).O unitate de *HD* cu *n* discuri conţine un cărucior cu *2n capete de citire/scriere* (*CCS*) care se deplasează radial de la o pistă la alta.

Prin rotaţia discurilor se formează o pernă de aer între capete şi suprafaţa magnetică, astfel încât citirea/scrierea se efectuează fără contact direct între cele două elemente. Pilele de disc formează un aşa numit *hard-disk, disk dur sau disk rigid.* Cel mai utilizat model este discul *Winchester,* format dint-o pilă de disc încapsulată, cu capacităţi uzuale de sute de *GB.*

4.4. Subsistemul intrare - ieşire

Alături de sistemele *unitate centrală de prelucrare* şi *memorie* un alt sistem important al unui calculator numeric este *sistemul de intrare ieşire (SIE),* care facilitează conectarea acestuia la mediul extern reprezentat de utilizatori. În continuare vor fi prezentate unele aspecte care privesc structura *SIE* şi modalităţile de transfer a datelor în cadrul acestora**.**

4.4.1. Structura unui sistem de intrare – ieşire

Pentru realizarea funcţiilor specifice un *SIE* are o structură modulară de tipul celei din figura 4.27.

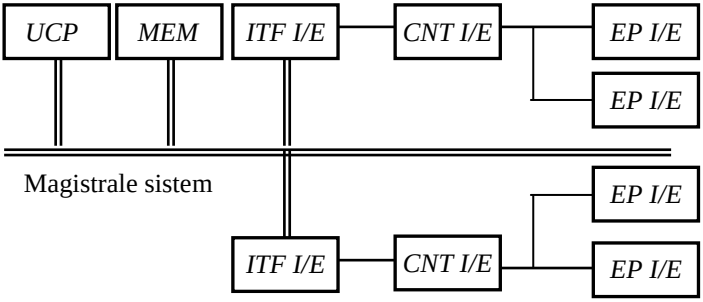


Fig. 4.27. Structura unui sistem de intrare – ieşire: *ITF I/E* – interfaţă de intrare – ieşire; *CNT* – controler intrare - ieşire; *EP I/E* – echipament periferic de intrare – ieşire.

După cum se observă în structura unui SIE sunt incluse următoarele elemente:

- interfeţe de I/E;
- controlere de I/E;
- echipamentele periferice de I/E;
- magistralele de comunicaţie;
- interfeţele dintre aceste componente.

Conectarea indirectă echipamentelor periferice de *I/E* la magistralele sistemului este justificată de următoarele argumente:

- marea varietate de periferice bazate pe diferite principii de funcţionare;
- diferenţe semnificative între ratele de transfer a echipamentelor periferice şi cea a memoriei;
- forme diferite de reprezentare a datelor în memorie şi în echipamentele periferice.

O interfaţă de *I/E* are rolul de a acţiona prin intermediul controlerelor asupra echipamentelor periferice. Îndeplinirea acestui rol induce pentru o interfaţă de I/E următoarele funcţii:

- control şi sincronizare;
- comunicaţia cu *UCP*;
- comunicaţia cu dispozitivele externe;
- memorarea temporară a datelor;
- detecţia erorilor.

În orice moment de timp, *UCP* poate comunica cu unul sau mai multe dispozitive externe. Resursele interne, ca memoria internă şi magistrala sistem, trebuie partajate între mai multe activităţi, inclusiv operaţiile de I/E ale datelor. Funcţia de *I/E* necesită deci o operaţie de *control şi sincronizare,* pentru coordonarea fluxului de date între resursele interne şi dispozitivele externe. De exemplu, controlul transferului de date de la un dispozitiv extern la *UCP* poate cuprinde următoarele etape:

- *UCP* interoghează interfaţa de *I/E* pentru a testa starea dispozitivului solicitat;
- interfaţa de *I/E* returnează starea dispozitivului;
- dacă dispozitivul este operaţional şi este pregătit pentru transmisia datelor, *UCP* solicită transferul datelor, printr-o comandă adresată interfeţei de *I/E*;
- interfaţa de *I/E* preia un octet sau cuvânt de date de la dispozitivul de *I/E*;
- datele sunt transferate de la interfaţa de *I/E* la *UCP*.

Din enumerarea de mai sus rezultă că interfaţa de *I/E* trebuie sa permită comunicaţia atât cu *UCP* şi cu dispozitivul extern reprezentat de controlerul de *I/E*.

4.4.2. Transferul datelor în sistemul intrare – ieşire

În cadrul sistemelor de *I/E* transferurile de date se pot realiza printr-una din următoarele metode:

- transfer programat;
- transfer prin întreruperi;
- transfer prin acces direct la memorie ;
- transfer prin canale de I/E**,**

metode care vor fi pe scurt prezentate în continuare.

Transferul programat. Prin transfer programat, *UCP* controlează în totalitate operaţia de intrare – ieşire asigurând următoarea secvenţă:

- detectarea stării perifericului;
- transmiterea unei comenzi de citire (intrare) sau scriere (ieşire);
- transferul datelor.

Pentru execuţia unei instrucţiuni de *I/E, UCP* transmite o adresă, (aferentă interfeţei de *I/E* şi dispozitivului extern) şi o comandă de *I/E* care poate fi: de *control*, de *test*, de *citire* sau de *scriere*. Comenzile de *control* sunt utilizate pentru activarea perifericelor şi pentru specificarea operaţiilor care urmează a se executa.

Prin comenzile de *test* se realizează verificarea diferitelor condiţii de stare asociate unei interfeţei de *I/E* şi perifericului specificat. În acest sens *UCP* va trebui, de exemplu, să determine dacă:

- perifericul este operaţional şi disponibil;
- ultima operaţie lansată este terminată;
- operaţia s-a executat cu sau fără erori.

Comanda de *citire* determină obţinerea unui octet sau cuvânt de date la periferic si depunerea acestuia într-o memorie tampon (registrul de date al interfeţei modulului de *I/E*). O comandă de *scriere* asigură preluarea de către interfaţa modulul de I/E a unui octet sau cuvânt de pe magistrala de date şi transferul acestuia către periferic.

Transferul programat se mai numeşte şi transfer în *buclă de aşteptare* deoarece după iniţierea transferului *UCP* execută o buclă de aşteptare până la finalizarea acestuia. Această aşteptare constituie un dezavantaj al transferului programat, datorită unei utilizări ineficiente procesorului. Aceste caracteristici

recomandă utilizarea acestei metode pentru transferul unui singur octet (cum ar fi un caracter de la tastatură).

Transferul prin întreruperi. *Întreruperea* reprezintă suspendarea temporară a execuţiei unui program, ca urmare a producerii la un moment de timp impredictibil a unui eveniment exterior acestei execuţii. Evenimentului produs îi corespunde un semnal de cerere de întrerupere, care este tratat de către *sistemul de întreruperi* al calculatorului. Dacă sistemul de întreruperi este activat, se apelează o *rutină de tratare a întreruperii*, asociată evenimentului care a generat întreruperea.

După execuţia rutinei de tratare, programul întrerupt este reluat în contextul anterior întreruperii, context care cuprinde:

- adresa de revenire;
- indicatorii de condiţii şi de stare ai *UCP*;
- registrele interne utilizate de rutina de tratare.

În cazul transferului prin întreruperi, *UCP* iniţiază transferul (transmite o comandă interfeţei de *I/E*), după care continuă execuţia programului. Interfaţa va emite o cerere de întrerupere către *UCP* atunci când este pregătită pentru transferul datelor. Ca răspuns la întrerupere *UCP* va executa transferul, după care va continua programul întrerupt.

Interfaţa specifică transferului prin întreruperi este mai complexă, iar viteza este mai mică decât în cazul celui programat. Viteza redusă este cauzată de faptul că tratarea întreruperii presupune execuţia unor instrucţiuni suplimentare în raport cu transferul programat

Transferul prin acces direct la memorie (DMA[7]). După cum s-a văzut, transferul programat, chiar dacă se efectuează prin întreruperi, necesită intervenţia *UCP* pentru transferul datelor între memorie şi o interfaţă de I*/E,* pentru fiecare octet sau cuvânt şi prezintă următoarele două dezavantaje:

- viteza de transfer este limitată de timpul în care *UCP* poate testa şi deservi un dispozitiv de *I/E*;
- încărcarea *UCP* cu gestionarea transferului , întrucât este necesară execuţia unui anumit număr de instrucţiuni pentru fiecare transfer.

[7] *DMA* – Direct Memory Access

Prin *DMA* transferul se realizează direct între un dispozitiv de *I/E* şi memorie, implicarea *UCP* fiind mult diminuată. Procesorul iniţiază transferul stabilind printr-o secvenţă de instrucţiuni următoarele elemente:

- adresa de început a zonei de memorie unde/de unde se efectuează transferul;
- lungimea blocului care se transferă;
- sensul transferului.

Transferul prin *DMA* necesită o interfaţă adecvată (controler *DMA*) care să poată prelua controlul magistralelor sistemului de calcul în următoarele situaţii:

- efectuarea unui transfer de date cu memoria, atunci când acestea nu sunt utilizate de *UCP*;
- suspendarea temporară de către *UCP* a programului în execuţie.

Având în vedere aceste posibilităţi de *rechiziţionare* a magistralei s-au impus două metode de efectuare a transferului DMA:

- prin *furt de ciclu (cycle stealing),* respectiv prin utilizarea intervalelor de timp în care *UCP* nu face acces la memorie (metoda se numeşte astfel deoarece interfaţa *DMA fură* un ciclu de memorie de la UCP, activitatea sa nefiind influenţată de operaţiile *DMA*);
- *in rafală (data break),* care presupune suspendarea programului executat de *UCP* în timpul transferului şi trecerea magistralei în starea de înaltă impedanţă.

Transferul prin canale de I/E. Aceste canale sunt procesoare specializate care reprezintă forme dezvoltate ale interfeţelor de *I/E* [8] . Procesoarele de *I/E* degrevează şi mai mult decât *DMA* procesorul de sarcinile specifice operaţiilor de *I/E*. Necesitatea acestor procesoare se impune mai ales în cadrul perifericelor rapide când furturile de ciclu pot ajunge să satureze magistrala. De asemenea chiar dacă se va genera o singură întrerupere la fiecare bloc transferat, timpul consumat de *UCP* pentru execuţia rutinei de tratare va diminua performanţele calculatorului în ansamblu.

După cum se observă din figura 4.28, în care se prezintă structura unui calculator cu canal de I/E, canalul asigură o cale pentru transferul

[8] Un procesor de *I/E* cu o largă de răspândire a fost INTEL 8089.

datelor dispozitivele de I/E şi memorie. În ceea ce priveşte comunicaţia canalului cu memoria aceasta se poate realiza prin mecanism DMA. În situaţia în care procesoarele de I/E au o memorie locala proprie, acestea devin calculatoare specializate care pot controla o paletă largă de echipamente de I/E, cu intervenţii minime din partea UCP.

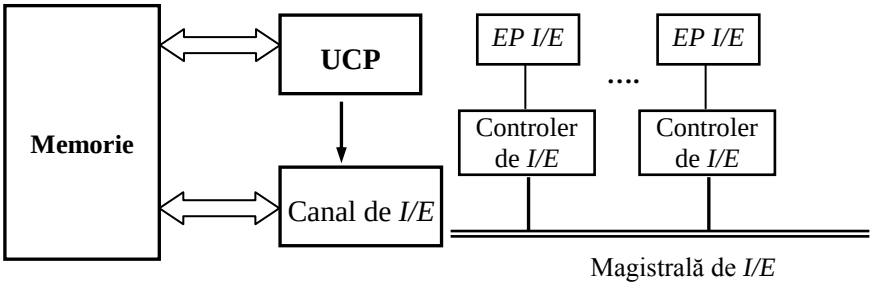


Fig. 4.28. Conectarea unui canal de *I/E* la memorie şi *UCP*:
EP I/E – echipamente periferice de intrare – ieşire.

Iniţierea unui transfer aparţine *UCP* care transmite o comandă procesorului de *I/E* pentru a executa un program specific din memorie. Secvenţa de operaţii de I/E va fi executată de procesorul specializat fără intervenţia UCP. Aceasta va fi *înştiinţată* printr-un semnal de întrerupere asupra încheierii întregii secvenţe.

4.4.3. Echipamente periferice de intrare – ieşire

Echipamentele periferice de intrare – ieşire *(EP I/E*) au rolul de a asigura comunicarea între unitatea centrală şi mediul exterior, prin intermediul unor unităţi de interfaţă. *EP I/E* permit realizarea următoarelor funcţii semnificative:

- introducerea datelor, a programelor şi a comenzilor în memoria calculatorului;
- afişarea, tipărirea sau redarea rezultatelor prelucrării într-o formă specificată de utilizator;
- înregistrarea, stocarea şi păstrarea volumelor mari de informaţii pe suporturi de memorie externă, în vederea unor prelucrări şi utilizări ulterioare;
- supravegherea şi posibilitatea intervenţiei utilizatorului pentru restabilirea funcţionării corecte a sistemului de calcul.

Potrivit acestor funcţii, echipamentele periferice se pot grupa astfel:

- *echipamente periferice de intrare* prin intermediul cărora se asigură introducerea datelor, a programelor, transmiterea unor comenzi manuale, citirea unor imagini etc. (în această categorie sunt incluse: tastatura, mouse-ul, creionul optic, tableta digitală, scanner-ul, cititorul de coduri de bare, cititorul de cartele magnetice, captatorul de sunete şi/sau imagini etc.);

- *echipamente periferice de ieşire*, care servesc la redarea rezultatelor prelucrărilor, a mesajelor, a programelor şi a altor informaţii (în această categorie sunt incluse monitoarele, imprimantele, echipamentele audio/video, etc.);

- *echipamente periferice de memorare***,** care au rolul de a stoca o mare cantitate de informaţie, pentru un timp nedeterminat, în vederea utilizării ulterioare (în această categorie sunt incluse: unităţile de: disc/bandă magnetică, disc optic, digital video disc, memorie flash, etc.);

- *echipamente de transport la distanţă***,** care au rolul de a oferi o formă adecvată semnalului care se emite/recepţionează prin suportul fizic reprezentat de cablu electric, unde electromagnetice, fibră optică (în această categorie este inclus modemul).

Întrucât echipamentele periferice de memorare au fost prezentate în cadrul sistemului memorie, în cele ce urmează vor fi prezentate câteva echipamente periferice din celelalte categorii.

Tastatura. Componentă a configuraţiei minimale a oricărui calculator, tastatura serveşte pentru introducerea caracterelor alfanumerice şi a comenzilor. După modul de dispunere a tastelor alfabetice, tastaturile sunt de două tipuri şi anume:

- tastatura de tip *anglo-saxon* la care primele taste alfabetice sunt cele care corespund literelor *Q W E R T Y*;

- tastatura de tip francez la care la care primele taste alfabetice sunt cele care corespund literelor *A Z E R T Y* .

Tastatura se comportă, în timpul lucrului, ca un mic calculator, în sensul că are capacitatea de a memora temporar o linie de date, o linie de comandă sau de instrucţiuni de program şi permite efectuarea corecturilor necesare, înainte de transmiterea acestora în memoria internă a calculatorului (înainte de acţionarea tastei *ENTER).* Acest lucru este posibil pentru că tastatura are un microprocesor propriu şi o memorie *RAM* .

Fiecare tastă are asociat un cod numeric, care este un cod ASCII. Procesorul tastaturii poate sesiza momentul apăsării unei taste şi momentul eliberării sale, putând genera în mod repetat codul tastei menţinute în poziţia apăsat. Este de menţionat faptul că există posibilitatea schimbării codului intern propriu al tastaturii prin comenzi de configurare, în funcţie de particularităţile ţării în care se utilizează respectiva tastatură *(regional settings)*.

Mouse-ul. Acesta reprezintă un dispozitiv de intrare care determină la mişcarea sa pe o suprafaţă plană deplasarea unui *cursor* pe ecran. Pentru a transmite comenzi mouse-ul este prevăzut cu două sau trei butoane[9], iar pentru deplasarea ferestrei active pe verticală cu o roată de *scroll.* Rezultă că mouse-ul este util numai în măsura în care pe ecran există afişate anumite opţiuni din care se pot selecta şi activa cele necesare pe parcursul unei sesiuni de lucru.

La apariţia sa acest periferic era de tip mecanic, în care mişcarea unei bile de cauciuc (situată în partea inferioară a mouse-ului) era convertită în semnale a căror finalitate era reprezentată de o deplasare a cursorului pe ecran. În prezent o largă răspândire o are mouse-ul optic care permite detectarea mişcării cu ajutorul unei perechi de diode electroluminiscente şi a unui suport reflectorizant.

Scanner-ul. În categoria echipamentelor pentru captarea imaginilor, în vederea unor prelucrări pe calculator, cel mai răspândit este scanner-ul. Acesta preia cu ajutorul unor senzori[10] imagini, desene şi texte de pe hârtie pe care le digitizează şi le transmite calculatorului, care le memorează sub forma unor fişiere. Prelucrarea ulterioară poate consta în finisarea contururilor, redimensionare, mutare, rotire, colorare, umbrire, suprapunere etc.

Monitorul. Cunoscut şi ca *display* acesta reprezintă componenta care împreună cu tastatura realizează consola face parte din configuraţia de bază a oricărui calculator personal. Monitorul este destinat afişării, pe ecran, a informaţiilor de natură alfanumerică şi/sau grafică.

---

[9] O apăsare scurtă a unui buton este cunoscută sub denumirea de *click,* iar mişcarea mouse-ului cu un buton apăsat (având ca efect marcarea unui text, mutarea unor obiecte sau ferestre, copierea, etc.) se numeşte *drag (glisare)*

[10] Senzorii scanner-ului se numesc celule CCD (Charge Coupled Device), care sunt de fapt condensatori încărcaţi electric şi sensibili la lumină.

După tehnologia de construcţie şi principiul de afişare monitoarele sunt de două tipuri şi anume:

- monitoare cu cristale lichide (*LCD*[11]);
- monitoare cu tub catodic. (*CRT*[12]).

Indiferent cărui tip aparţin, monitoarele sunt diferenţiate de următoarele caracteristici mai importante:

- calitatea grafică a afişării;
- dimensiunea ecranului (diagonala) şi dimensiunile imaginii afişate;
- numărul de culori;
- viteza de lucru;
- gradul de nocivitate al radiaţiilor pe care le emite.

În mod obişnuit monitoarele suportă două tipuri de afişări a informaţiei şi anume:

- afişarea în *modul text;*
- afişarea în *modul grafic.*

Afişarea în *modul text (alfanumeric)* se realizează la nivel de caracter, ecranul fiind considerat o matrice cu 2000 de zone-caracter (80 de coloane şi 25 de linii. În fiecare zonă se poate afişa un singur caracter din 256 posibile, conform codificării ASCII extinse. Aşadar în modul text entitatea minimă care poate fi afişată este *caracterul.*

În *modul grafic*, ecranul este văzut ca o matrice de elemente de imagini (*pixeli)* fiecare pixel putând fi caracterizat la monitoarele color prin trei elemente de culoare: *roşu, verde, şi albastru*. Obţinerea nuanţelor de culoare se realizează prin variaţia intensităţii de iluminare a pixelilor. În acest mod *calitatea grafică* este asigurată de doi factori şi anume *definiţia şi calitatea.*

Definiţia monitorului este dată de dimensiunea punctelor ce formează imaginea. Cu cât dimensiunea unui punct este mai mică[13], cu atât densitatea este mai ridicată şi în consecinţă definiţia este mai bună.

Rezoluţia desemnează numărul maxim de puncte ce pot fi afişate pe suprafaţa unui ecran (deci dimensiunile matricei asociate ecranului).

---

[11] *LCD* – Liquid Crystal Display

[12] *CRT* – Cathode Ray Tube

[13] La actualele monitoare o valoare standard a diametrului unui pixel este de *0,28 mm.* .

Dimensiunea ecranului unui monitor este reprezentată de mărimea diagonalei exprimată în inches[14]. Dimensiunile mai frecvent întâlnite sunt de 17; 19 inches, tendinţa actuală fiind spre cele de 19 inches în tehnologie *LCD* care se încadrează mai bine în normele ergonomice şi de consum redus de energie.

Creionul optic (light pen) este un dispozitiv utilizat pentru activarea (punctarea) unei poziţii pe ecran. În condiţiile existenţei unui soft adecvat cu ajutorul acestui dispozitiv poate fi scris un text pe ecran.

Tableta grafică şi digitizorul servesc la introducerea în memoria calculatorului a coordonatelor unor puncte de pe un desen deja realizat, raportate la un sistem de referinţă ataşat suprafeţei de lucru. La tableta de lucru suprafaţa utilă se încadrează în formatele *A3* sau *A4*, în timp ce digitizoarele pot opera pe suprafeţe mult mai mari. Pentru preluarea coordonatelor aceste echipamente sunt prevăzute cu dispozitive de tip creion sau lupă care se pot deplasa pe suprafaţa care trebuie digitizată. Pentru transmiterea la calculator a poziţiei şi a unor comenzi acestor dispozitive li se adaugă un număr de butoane.

Imprimanta. Imprimantele fac parte din categoria echipamentelor periferice noninteractive şi asigură obţinerea rezultatelor sub formă de document tipărit.

Principalele elemente care permit caracterizarea unui anumit tip de imprimantă sunt următoarele:

- principiul de funcţionare;
- viteza de tipărire;
- dimensiunea liniei tipărite;
- rezoluţia tipăririi ;
- dimensiunea memoriei proprii;
- existenţa unui limbaj propriu (POSTSCRIPT);
- fiabilitatea şi costul.

În mod obişnuit imprimantele sunt clasificate după principiul de funcţionare, respectiv după mecanismul de tipărire. Conform acestui criteriu, cele mai răspândite tipuri de imprimante sunt:

[14] 1 inch = 25,4 mm

- imprimantele matriceale;
- imprimantele cu jet de cerneală;
- imprimantele laser.

Imprimantele matriceale utilizează un mecanism de tipărire format dintr-un set de ace montate în capul de imprimare, care în momentul primirii impulsurilor lovesc o bandă tuşată, numită *ribbon*. Rezoluţia unei asemenea imprimante este dată de numărul de ace, cele mai răspândite fiind cu 9, 18 sau 24 de ace. Viteza de tipărire specifică acestor imprimante este de 150-400 caractere pe secundă.

Imprimantele cu jet de cerneală utilizează dispozitive electronice şi electromecanice care permit preluarea cernelei dintr-un rezervor special (cartuş) şi pulverizarea sa printr-un sistem de duze. Cerneala are o mare capacitate de a se usca rapid (are proprietăţi sicative ridicate).

Imprimantele laser asigură cel mai bun raport performanţă cost. Principiul de funcţionare al acestora se bazează pe polarizarea electrostatică cu ajutorul unui fascicul laser a unui cilindru special. Acesta la rândul lui atrage şi se încarcă pe suprafaţa sa cu o pulbere de grafit fin numită toner, pulbere care este depusă apoi pe hârtie. În continuare hârtia este supusă unui tratament termic pentru fixare.

Viteza de tipărire diferă de la un tip de imprimantă la altul, depinzând în primul rând de principiul de tipărire. O imprimantă laser asigură o viteză de tipărire între 5 şi 20 pagini pe minut sau chiar mai mult, în funcţie de gradul de umplere a documentului care se tipăreşte şi de calitatea imprimantei, în timp ce o imprimantă matriceală obişnuită imprimă cu o viteză medie sub 5 pagini pe minut. Calitatea grafică a tipăririi depinde, de rezoluţia imprimantei care se exprimă la fel prin numărul de puncte (dots) pe inches. Cea mai bună rezoluţie este asigurată de imprimantele laser (în medie 600 dpi), urmată de imprimanta cu jet de cerneală.

În mod obişnuit imprimantele dispun de o memorie internă proprie care serveşte pentru stocarea informaţiilor aflate în aşteptarea tipăririi. Când se generează o comandă de tipărire, programul de aplicaţie transmite şi informaţiile către imprimantă iar aceasta le stochează în propria memorie după care începe tipărirea. Dacă volumul informaţiilor de tipărit depăşeşte capacitatea memoriei proprii atunci transferul acestora către imprimantă se face treptat, astfel că programul de aplicaţii va ţine sistemul ocupat până la terminarea tipăririi.