1. Bazele aritmetice al calculatoarelor numerice

1.1. 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. Pentru un sistem poziţional ponderea unei cifre este dată atât de valoarea ei intrinsecă cât şi de poziţie. Un sistem nepoziţional, este acela în care ponderea nu este influenţată de poziţia cifrei.

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).

Exemple 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: *b=16*, 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 secvenţa de simboluri $x_{m-1}\ x_{m-2} \ldots x_2\ x_1\ x_0$ care verifică următoarele condiţii:

a. $$0 \leq x_i < b,\ \ i = m-1, m-2, \ldots, 2, 1, 0;\ \ x_{m-1} \neq 0; \qquad (1.1)$$

b. $$N = x_{m-1}b^{m-1} + x_{m-2}b^{m-2} + \ldots + x_2 b^2 + x_1 b^1 + x_0 b^0. \qquad (1.2)$$

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 $x_{m-1} \ldots x_1\ x_0 . x_{-1}\ x_{-2} \ldots$ care verifică următoarele condiţii:

a. $$0 \leq x_i < b,\ \ i = m-1, m-2, \ldots, 2, 1, 0, -1, -2, \ldots;\ \ x_{m-1} \neq 0; \qquad (1.3)$$

b. $$r = x_{m-1}b^{m-1} + \ldots + x_2 b^2 + x_1 b^1 + x_0 b^0 + x_{-1} b^{-1} + x_{-2} b^{-2} + \ldots. \qquad (1.4)$$

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.

1.2. Conversia unui număr dintr-o bază în alta

Existenţa şi utilizarea mai multor SN ridică problema conversiei dintr-un sistem în altul. Una din metodele frecvent utilizate o reprezintă împărţirea / înmulţirea numărului cu noua bază.

- *Vom examina pentru început conversia numerelor întregi.*

Fie numărul N exprimat în baza α :

$$(N)_\alpha = a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2} + \ldots + a_1\alpha^1 + a_0\alpha^0, \tag{1.5}$$

pentru care se doreşte exprimarea în baza β, respectiv

$$(N)_\alpha = x_{n-1}\beta^{n-1} + x_{n-2}\beta^{n-2} + \ldots + x_1\beta^1 + x_0\beta^0, \tag{1.6}$$

Algoritmul de conversie presupune determinarea coeficienţilor x_i , una din metode fiind cea a *împărţirilor succesive a numărului la noua bază,* conform următorului procedeu:

$$(N)_\alpha/\beta = \underbrace{x_{n-1}\beta^{n-2} + x_{n-2}\beta^{n-3} + \ldots + x_1\beta^0}_{\text{Câtul}\,(N_1)_\alpha} + \underbrace{x_0/\beta}_{\text{Restul}} \Rightarrow x_0\,, \tag{1.7}$$

$$(N_1)_\alpha/\beta = \underbrace{x_{n-1}\beta^{n-3} + x_{n-2}\beta^{n-4} + \ldots + x_2\beta^0}_{\text{Câtul}\,(N_2)_\alpha} + \underbrace{x_1/\beta}_{\text{Restul}} \Rightarrow x_1\,, \tag{1.8}$$

……..

$$(N_k)_\alpha/\beta = \underbrace{x_{n-1}\beta^{n-k-2} + \ldots + x_{k+1}\beta^0}_{\text{Câtul}(N_{k+1})_\alpha} + \underbrace{x_k/\beta}_{\text{Restul}} \Rightarrow x_k\,, \tag{1.9}$$

…….

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*).

- *În cazul numerelor reale de forma* $r = r_I + r_S$ partea întreagă r_I se converteşte potrivit procedurii prezentate, iar partea subunitară r_S prin înmulţiri succesive cu noua bază.

Fie numărul r_S exprimat în baza α :

$$(r_S)_\alpha = a_{-1}\alpha^{-1} + a_{-2}\alpha^{-2} + \ldots + a_{-m}\alpha^{-m}, \tag{1.10}$$

pentru care se doreşte exprimarea în baza β , respectiv

$$(r_S)_\alpha = x_{-1}\beta^{-1} + x_{-2}\beta^{-2} + \ldots + a_{-n}\alpha^{-n} + \ldots, \tag{1.11}$$

Coeficienţii x_{-i} se obţin prin înmulţiri succesive în (1.11) cu β, respectiv

$$(r_S)_\alpha \cdot \beta = x_{-1} + \underbrace{x_{-2}\beta^{-1} + \ldots + a_{-n}\beta^{-n+1} + \ldots}_{(r_{S1})_\alpha}, \Rightarrow x_{-1}; \tag{1.12}$$

$$(r_{S1})_\alpha \cdot \beta = x_{-2} + \underbrace{x_{-3}\beta^{-1} + \ldots + a_{-n}\beta^{-n+2} + \ldots}_{(r_{S2})_\alpha}, \Rightarrow x_{-2}; \quad (1.13)$$

……..

$$(r_{S\,k-1})_\alpha \cdot \beta = x_{-k} + \underbrace{x_{-k-1}\beta^{-1} + \ldots + a_{-n}\beta^{-n+k} + \ldots}_{(r_{S\,k})_\alpha}, \Rightarrow x_{-k}; \quad (1.14)$$

…….

Cifra x_{-1} reprezintă *CCMS* iar x_{-k} *CCMPS* a rezultatului. Procedura se opreşte în momentul partea fracţionară $r_{S\,k}$ a rezultatului unei înmulţiri devine zero, sau dacă s-a atins precizia de conversie specificată prin numărul de cifre al reprezentării.

Din prezentarea efectuată rezultă că numărul de poziţii binare pentru partea întreagă se poate determina apriori, în timp ce pentru partea fracţionară acest număr trebuie impus. Astfel numărul *n* de poziţii binare care se obţin la conversia unui număr zecimal *N* este:

$$n = \begin{cases} \log_2 N + 1 & \text{dacă } N = 2^k,\ k \in N \\ [\log_2 N] + 1 & \text{dacă } N \neq 2^k, k \in N \end{cases} \quad (1.15)$$

- *Conversiile binar - octal ; binar - hexazecimal* se realizează simplu datorită faptului că bazele acestora sunt puteri ale lui 2 aspect evidenţiat în tabelul 1.1.

Tabelul 1.1

Octal	Binar	Hexaze cimal	Binar	Hexaze cimal	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

Cifrele sistemului octal pot fi reprezentate prin combinaţii de câte trei biţi denumite *triade* iar ale sistemului hexazecimal prin combinaţii de câte patru biţi numite *tetrade.*

Regula de conversie: fiind dată reprezentarea în binar a unui număr real, reprezentarea în octal se obţine grupând câte trei cifrele binare începând de la separatorul fracţionar (punct, virgulă), spre stânga şi spre dreapta. După ce grupele extreme se completează (dacă este cazul cu zerouri nesemnificative), fiecare triadă se substituie cu echivalentul său octal. Aceiaşi regulă se aplică şi la conversia *binar-hexazecimal,* cu observaţia că în loc de *triade* se operează cu combinaţii de 4 biţi (*tetrade).*

Conversia inversă *octal / hexazecimal* $\rightarrow$ *binar* se face prin înlocuirea fiecărei cifre *octale / hexazecimale* cu *triada / tetrada* corespunzătoare.

1.3. 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.*

a) *Reprezentarea numerelor întregi* . Acestea se reprezintă de regulă în format cu virgulă fixă. De la început trebuie făcută precizarea că virgula (punctul zecimal) nu este prezentă în mod explicit în reprezentare. Sintagma *virgulă fixă* are în vedere o convenţie care presupune un loc fix al acesteia şi un acelaşi număr de cifre pentru număr.

Numărul de biţi (n) utilizaţi pentru o reprezentare determină numărul de valori reprezentabile (2^n). În tabelul 2.2 sunt prezentate valori tipice pentru n şi 2^n.

Domeniul finit de valori pentru numerele întregi, reprezentate în formatul cu virgulă fixă este:

$$D = [V_{\min}, V_{\max}] \cap Z , \tag{1.16}$$

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

Observaţie importantă.

Indiferent de format 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, interpretate diferit în cele două formate. *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.

Numerele întregi fără semn se reprezintă prin corespondentul lor binar *(codul direct)* numărul de biţi pentru reprezentarea unui număr N fiind

$$2^{n-1} \le N < 2^n . \tag{1.17}$$

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

a1semn – mărime;
a2complement faţă de 1;
a3complement faţă de 2.

a1) Reprezentarea numerelor în semn mărime

Considerând reprezentarea pe *n+1* biţi primul bit din stânga este asociat *semnului*, iar restul de *n* biţi conţin *mărimea* numărului egală cu reprezentarea binară a valorii $|N|$- figura 1.1.

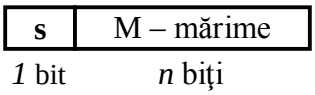


Fig. 1.1. Reprezentarea *semn – mărime* pe *n+1* biţi.

În privinţa semnului *s* care ocupă bitul *cel mai semnificativ* (***MSB –Most Significant Bit***) convenţia este următoarea:

$S = 0$, dacă $N \geq 0$;

$S = 1$, dacă $N < 0$.

a2) Reprezentarea numerelor în complement faţă de 1

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

Dacă N < 0, atunci s = 1 şi mărimea

$$M = C1(|N|) = 2^n - 1 - |N|, \tag{1.18}$$

unde *n este* numărul de biţi utilizaţi pentru reprezentarea mărimii.

Calculul complementului *C1* se poate face prin două metode şi anume:

1- utilizând definiţia, (1.18) respectiv

$S = 1 \quad M = 2^n - 1 - |N|.$

2 - prin *inversarea biţilor* reprezentării cu semn a valorii absolute $|N|$ a numărului *N.*

a3) Reprezentarea numerelor în complement faţă de 2

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

Dacă $N < 0$, atunci $s = 1$ şi mărimea

$$M = C2(|N|) = 2^n - |N|, \tag{1.19}$$

unde *n este* numărul de biţi utilizaţi pentru reprezentarea mărimii.

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

1- *utilizând definiţia,* respectiv

$S = 1 \quad M = 2^n - |N|.$ *s=1.*

2 - prin *inversarea biţilor* reprezentării cu semn a valorii absolute $|N|$ a numărului *N* la care se adaugă *1*;

3 - 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).

Observaţie

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

Reprezentările *semn – mărime şi C1* oferă două reprezentări distincte ale acestui număr ***(+0 şi -0)*** după cum urmează.

Semn - mărime	*C1*	*C2*
+0 = 0000 0000	*+0 = 0000 0000*	*+0 = 0000 0000*
- 0 = 1000 0000	*- 0 = 1111 1111*	*-0 = 0000 0000*

b. <u>*Reprezentarea numerelor reale.*</u> Numerele reale se pot reprezenta în *format virgulă fixă sau virgulă mobilă.*

b1) Reprezentarea în format virgulă fixă este asemănătoare reprezentării numerelor întregi. Se impun *n1* biţi pentru partea întreagă, respectiv *n2* biţi pentru cea fracţionară, rezultând o poziţie fix[pentru separatorul zecimal (punct sau virgulă) potrivit figurii 1.2.

s	M1 – mărime partea întreagă	M2 – mărime partea fracţionară
1	*n1* biţi	*n2* biţi

Fig. 1.2. Reprezentarea *format cu virgulă fixă* pentru numere reale.

b2) Reprezentarea în format virgulă fixă mobilă utilizează reprezentarea ştiinţifică

$$r = m \cdot 10^{E}, \tag{1.20}$$

căreia îi sunt asociate două componente:

• *E - exponentul* - indică ordinul de mărime al numărului;
• *m - mantisa* - arată mărimea exactă a numărului într-un anumit domeniu.

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.

Diferenţa faţă de formatul cu virgulă fixă constă în modul în care sunt interpretate aceste valori. 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.

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 2.3, a domeniilor de valori reprezentabile.

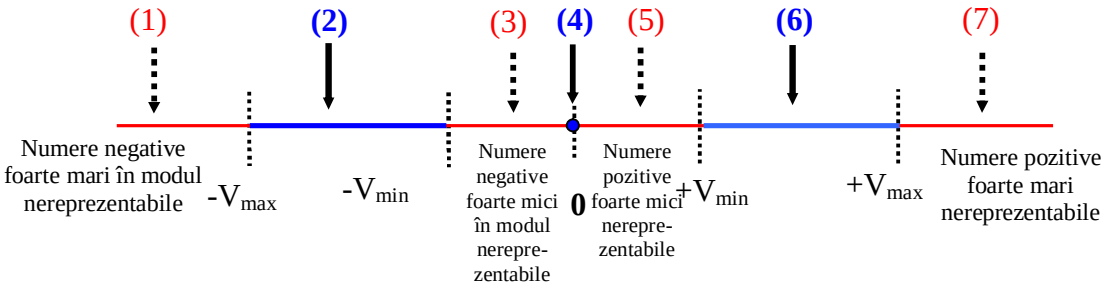


Fig. 1.3. Domeniile valorilor reale reprezentabile.

Zonele marcate cu cifre in figura 1.3 reprezintă următoarele categorii de numere:

1. numere negative foarte mari în modul nereprezentabile ;
2. numere negative reprezentabile
3. numere negative foarte mici în modul, nereprezentabile;
4. *numărul zero*;
5. numere pozitive foarte mici în valoare absolută nereprezentabile;
6. numere pozitive reprezentabile *;*
7. numere pozitive foarte mari in valoare absolută, nereprezentabile.

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 1.3 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 determina *precizia de reprezentare* în domeniile *2* şi *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, 4, 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 *caracteristică* în domeniul [0, +127]∩Z.

Reprezentarea standard în VM. Î anul 1985 s-a adoptat standardul *IEEE 754* referitor la reprezentarea numerelor în virgulă mobilă, standard acceptat de majoritatea firmelor producătoare de microprocesoare.

Standardul defineşte trei formate: *simpla precizie, dubla precizie, precizie extinsă,* tabelul 1.2 fiind prezentate caracteristicile primelor două formate.

Tabelul 1.2

Format	Biti	Semn	Exp.	Mantisa	Exces
Simplă precizie	32	1	8	23	127
Dublă precizie	64	1	11	52	1023

1.4. Operaţii aritmetice

În cele ce urmează se vor prezenta câteva elemente ce privesc realizarea operaţiilor aritmetice în cod binar.

Efectuarea oricărei astfel de operaţii se reduce la adunarea şi / sau scăderea numerelor binare conform regulilor următoare:

0+0 = 0 0-0 = 0
0+1 = 1 1-0 = 1
1+0 = 1 10-1 = 1+b
1+1 = 10=0+c 1-1 = 0

unde c (*carry*) este transportul la rangul superior, iar b (*borrow*) este împrumutul de la rangul superior.

Adunarea şi scăderea în reprezentarea *semn - mărime*. Cele două operaţii vor fi tratate unitar conform relaţiei:

$$op = x_S \oplus y_S \oplus s_{OP} \qquad (1.21)$$

unde: *op* reprezintă operaţia efectivă ce se va efectua între cei doi operanzi;
x_S, y_S - semnele celor doi operanzi (*1* pt. -, *0* pt. +);
s_{OP} - operaţia ce se doreşte a fi efectuată (*1* pt. -, *0* pt. +).

În ceea ce priveşte operaţia "⊕" (*sau exclusiv*) 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* se determină cu ajutorul tabelului 1.3.

Tabelul 1.3

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	-	-	-	-

Dacă *op=0*, se adună modulele celor doi operanzi semnul rezultatului fiind dat de semnul primului operand. Rezultatul este corect dacă nu se depăşeşte valoarea maximă pentru numărul respectiv de biţi.

Dacă *op=1*, se va efectua scăderea modulelor celor doi operanzi, semnul rezultatului fiind dat de semnul numărului mai mare în modul.

Adunarea şi scăderea în *C1*. Aceste operaţii se reduc la operaţia de adunare a numerelor reprezentate în C1. Cei doi operanzi se adună bit cu bit, inclusiv biţii de semn. Eventualul transport care rezultă la *rangul de semn* se va aduna la rangul cel mai puţin semnificativ. Dacă bitul de semn al rezultatului are valoarea 1 atunci rezultatul este în C1 (altfel este *semn – mărime*).

Adunarea şi scăderea în C2. Operaţiile se reduc la adunarea numerelor reprezentate în C2. Cei doi operanzi se adună bit cu bit inclusiv biţii de semn iar eventualul transport care rezultă la bitul de semn se neglijează. Dacă bitul de semn al rezultatului are valoarea 1 atunci rezultatul este în C2 (altfel este *semn – mărime*).

Înmulţirea numerelor binare. Majoritatea metodelor de înmulţire a numerelor binare se bazează pe procedeul adunării repetate. Vom exemplifica pentru produsul a două numere *x* şi *y* exprimate în *semn – mărime,* pentru care se parcurg următoarele etape:

a) se determină semnul produsului $s_P = s_x + s_Y$ unde s_P este semnul produsului iar s_x şi s_Y semnele celor doi factori $s_P, s_x, s_Y \in \{0, 1\}$ conform regulii:

$0 + 0 = 0$; $(+) \bullet (+) = (+)$
$0 + 1 = 1$; $(+) \bullet (-) = (-)$
$1 + 0 = 1$; $(-) \bullet (+) = (-)$
$1 + 1 = 0$; $(-) \bullet (-) = (+)$

b) se calculează modulul produsului prin însumarea produselor parţiale, respectiv

$$|p| = \sum_{k=1}^{n-1} 2^{-k} \cdot y_{-k} \cdot |x|$$

unde (1.22)

$$2^{-k} \cdot y_{-k} \cdot |x| = \begin{cases} 0 & \text{daca } y_{-k} = 0 \\ 2^{-k} \cdot |x| & \text{daca } y_{-k} \neq 0 \end{cases}$$

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 $\rightarrow$ deplasare *stânga* cu ***k*** poziţii (se adaugă zerouri în poziţiile nesemnificative din dreapta rămase libere);

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

Împărţirea numerelor binare. Împărţirea numerelor binare are la bază regulile *0 : 1 = 0, 1 : 1 = 1* (împărţirea cu zero nu are sens). După cum s-a văzut, operaţia de înmulţire poate fi redusă la o serie de adunări. În mod similar operaţia de împărţire poate fi redusă la o serie de scăderi.

Adunarea şi scăderea numerelor reprezentate în format virgulă mobilă. Aceste operaţii se execută în mai multe etape şi anume:

a) dacă cei doi exponenţi (caracteristici) nu coincid se aduc numerele la acelaşi cel mai mare - cel mai mare (în acest scop numărul cu exponentul mai mic va fi deplasat cu $D = E_{max} - E_{min}$ poziţii la dreapta);

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

c) se normalizează mantisa obţinută în urma adunării.

Fie două numere în *VM* baza 16

$A = (\pm 0.M_1)_{16} \cdot 16^{E1}$ şi $B = (\pm 0.M_2)_{16} \cdot 16^{E2}$ unde M_1 şi M_2 sunt mantisele iar E_1 şi E_2 (unde $E_1 > E_2$) sunt caracteristicile celor două numere.

Suma respectiv diferenţa celor două numere se calculează astfel:

$$A \pm B = ((\pm 0.M_1)_{16} \pm (\pm 0.M_2)_{16} \cdot 16^{D}) \cdot 16^{E1} \qquad (1.23)$$

Înmulţirea şi împărţirea numerelor reprezentate în format cu virgulă mobilă. Această operaţie presupune adunarea (scăderea) exponenţilor (caracteristicilor) şi înmulţirea (împărţirea mantiselor).

Dacă cele două numere sunt:

$A = (\pm 0.M_1)_{16} \cdot 16^{E1}$ şi $B = (\pm 0.M_2)_{16} \cdot 16^{E2}$ atunci produsul şi câtul vor fi :

$$A \cdot B = ((\pm 0.M_1)_{16} \cdot (\pm 0.M_2)_{16}) \cdot 16^{E1+E2}$$

$$A : B = ((\pm 0.M_1)_{16} : (\pm 0.M_2)_{16}) \cdot 16^{E1-E2}$$

Produsul mantiselor va determina un număr de lungime dublă faţă de cele două mantise din care se vor reţine numai jumătate din cifre (cele mai semnificative) după care se face rotunjirea.

1.5. Coduri numerice şi alfanumerice

Pentru stocarea, prelucrarea şi transmiterea informaţiei se utilizează diverse codificări care elimină erorile de reprezentare permiţând detecţia şi corecţia erorilor.

1.5.1. Coduri numerice (binar – zecimale)

Cifrele zecimale pot fi reprezentate atât în binar cât şi în alte codificări care prezintă diverse avantaje în utilizare cum ar fi: **BCD**, codurile ponderate **8421** şi **2421** şi neponderate **Gray** şi **Exces 3.**

- **Codul BCD.** Fiecare cifră zecimală a unui număr este reprezentată prin codul său binar codificarea numindu-se *zecimal codificat binar (BCD – Binary Coted Decimal).* Această codificare se deosebeşte evident de codificarea binară, următorul exemplu fiind relevant.

*Operaţii aritmetice în cod BCD***.** Din modul în care se efectuează codificarea în BCD rezultă că sunt valide numai combinaţiile cuprinse între *0000* şi *1001,* celelalte (între *1010* şi *1111)* fiind invalide. Se pune problema efectuării de calcule aritmetice cu numere reprezentate în BCD, dar utilizând o UAL care lucrează binar. De exemplu 36+94 ş 130_{10} ş 1100 1010_{BCD} unde apar combinaţiile interzise 1100 şi 1010. Eliminarea acestora şi revenirea în cod BCD se face prin procedeul de *ajustare zecimală* care constă în:

a. dacă în urma adunării rezultă o cifră hexa $S \geq A$ atunci se adună 6 (respectiv *0110* şi se obţine cifra BCD;

b. dacă în urma adunării rezultă cifra $S \leq 3$, dar cu transport spre rangul superior, atunci se adună 6 (respectiv *0110* şi se obţine cifra BCD.

Această corecţie va permite efectuarea adunării în bază 10 *şi nu* în bază 16 cum s-ar efectua dacă s-ar lua în consideraţie toate combinaţiile posibile de câte 4 biţi.

În ceea ce priveşte celelalte operaţii acestea se reduc la adunare sau utilizează în mod repetat adunarea.

O menţiune pentru înmulţirea (împărţirea) cu 10^k ,echivalente cu deplasarea numărului cu *k* tetrade spre dreapta sau spre stânga faţă de poziţia punctului zecimal, după cum *k* este un întreg pozitiv, respectiv negativ.

- **Coduri ponderate şi neponderate.** *Un cod ponderat* asociază fiecărei cifre zecimale o tetradă binară, iar ponderea fiecărui bit din tetradă este egală cu valoarea cifrei din denumirea codului. Valoarea cifrei zecimale codificate se obţine prin însumarea biţilor cuvântului de cod, ponderaţi cu valoarea corespunzătoare din denumirea codului.

La *codurile neponderate* nu este o legătură directă între poziţia şi ponderea unui anumit bit..

In tabelul 1.4 se prezintă câte două coduri din fiecare categorie menţionată.

Tabelul 1.4

Cifră zecimală	Coduri ponderate		Coduri neponderate	
	8421	2421	Gray	Exces 3
0	0000	0000	0000	0011
1	0001	0001	0001	0100
2	0010	0010	0011	0101
3	0011	0011	0010	0110
4	0100	0100	0110	0111
5	0101	1011	0111	1000
6	0110	1100	0101	1001
7	0111	1101	0100	1010
8	1000	1110	1100	1011
9	1001	1111	1101	1100

Referitor la codurile prezentate în tabelul 1.4 se pot formula observaţiile de mai jos.

a) La codul *2421* codurile primelor 4 cifre sunt identice cu ale codului 8421. Codurile cifrelor a căror sumă este 9 sunt complementare exemplu 2 cu 7, 3 cu 6, 4 cu 5 etc).

b) Codul *Gray* are proprietatea de adiacenţă în sensul că trecerea de la o cifră la următoarea se face prin modificarea unui singur bit.

c) Un cuvânt al codului *EXCES 3* se obţine din cuvântul corespunzător din 8421 la care se adaugă 0011 (adică 3 în binar). Avantajul acestui cod este că se poate face distincţie între absenţa informaţie şi codul cifrei zero.

Cel mai răspândit cod este 8421 care stă la baza codificării *zecimal – binare (BCD).*

1.5.2. Coduri alfanumerice

În afara numerelor întregi sau reale informaţia prelucrată de calculator mai poate conţine text format din caractere, adrese de memorie, informaţie de stare etc.

Textul reprezintă una din formele cele mai utilizate pentru manevrarea şi memorarea informaţiei. În prezent calculatorul este utilizat într-o mare varietate de aplicaţii care nu necesită neapărat calcule matematice cum ar fi: editare, redactare, grafică etc. Tot cu ajutorul caracterelor se introduc în memoria calculatorului programele sursă şi se obţin rezultatele.

În general, în mulţimea caracterelor alfanumerice intră cifre, litere şi caractere speciale. Caracterele pot fi tipăribile sau nu, în ultima categorie intrând:

- caractere de control pentru comunicaţia între dispozitive;
- caractere de control pentru deplasarea cursor;
- caractere cu semnificaţie nedefinită.

În prezent marea majoritate a platformelor de calcul utilizează codul IBM **ASCII** (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange). Codul ASCII se prezintă în variantele restrâns şi extins. Codul ASCII restrâns asociază fiecărui caracter câte o formaţie de 7 biţi iar cel extins câte una de 8 biţi. Rezultă că cele două coduri permit codificarea a 128 respectiv a 256 de caractere. Exemple de coduri ASCII: A: 41h, 2:32h, *:2Ah, ?:3Fh, etc.

1.5.3. Coduri pentru detectarea şi corectarea erorilor

Relativ la erorile care pot apare la transmisia sunt de menţionat două categorii de probleme

a. detectarea erorilor;
b. corectarea erorilor.

La rândul său detectarea erorilor comportă două tipuri de operaţii:

a1. detectarea stării de eroare;
a2. diagnoza erorii.

Prin detectarea *stării de eroare* se specifică existenţa unei erori fără a se preciza exact eroarea. *Diagnoza* presupune determinarea erorii, respectiv a biţilor care au fost transmişi eronat.

Importantă în detectarea erorilor de transmisie este *distanţa Hamming (DH)* definită ca numărul de poziţii binare prin care diferă două cuvinte de cod. De exemplu cuvintele binare $x = 01100$ şi $y = 00101$ diferă prin două poziţii binare, deci distanţa Hamming este $d = 2$. Fiind vorba de sesizarea lipsei coincidenţei, d se poate determina cu ajutorul funcţiei *SAU – EXCLUSIV*, respectiv $d = x \oplus y$. Semnificaţia *DH* este că sunt necesare d erori de un singur bit

pentru a converti un cuvânt în altul. De asemenea pentru a elimina *d* erori este nevoie de un cod cu *DH = d + 1*.

Cea mai simplă soluţie de semnalare a stării de eroare este adăugarea bitului de *paritate (bP).* Dacă de exemplu la codul 8421 pe 4 biţi se adaugă un bit de paritate se obţine un cod detector de erori pe 5 biţi. *DH* a acestui cod este 2 deoarece la transmisia greşită a unui bit este afectat şi bitul de paritate. Se poate utiliza convenţia de paritate *pară (PP) sau impară (PI)*. La *PI* numărul total de biţi **1** din codul 8421 împreună cu bitul de paritate este impar, iar la *PP* acest număr este par.

În practică este larg utilizată şi metoda *codului polinomial* cunoscut şi sub denumirea de *cod cu redundanţă ciclică* sau cod ***CRC (Cyclic Redundancy Code)***.

La transmiterea datelor între cele mai utilizate polinoame generatoare sunt următoarele :

$CRC\text{-}12 = x^{12} + x^{11} + x^3 + x^2 + x + 1$

$CRC\text{-}16 = x^{16} + x^{15} + x^2 + 1$

$CRC\text{-}CCITT = x^{16} + x^{12} + x^5 + 1$

Corectarea erorii presupune înlocuirea biţilor detectaţi ca fiind transmişi eronat. 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 pentru a asigura această distanţă se determină cu relaţia

$$2^k \geq m + k + 1, \qquad (1.24)$$

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

2. Bazele logice ale calculatoarelor numerice

2.1. Variabile şi funcţii 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.

2.1.1. Algebra booleană

Fie mulţimile $\boldsymbol{M}=\{x_1, x_2,\ldots,x_n,\}$ cu $x_i \in \boldsymbol{Z}$ şi $\boldsymbol{O}=\{+,\bullet\}$ (componentele mulţimii $\boldsymbol{O}$ sunt două operaţii care vor fi definite ulterior). Structura $A=(M,O)$ reprezintă o algebră dacă:

a) mulţimea M conţine cel puţin două elemente;

b) mulţimea M reprezintă parte stabilă în raport cu cele două operaţii respectiv $x_1+x_2 \in$M, $x_1 \bullet x_2 \in M$ pentru orice $x_1, x_2 \in M$;

c) cele două operaţii au următoarele proprietăţi:

- comutativitate:

$x_1+x_2 = x_2+x_1;$

$x_1 \bullet x_2 = x_2 \bullet x_1,$

- asociativitate:

$(x_1+x_2)+x_3 = x_1+ (x_2 + x_3)$;

$(x_1 \bullet x_2) \bullet x_3 = x_1 \bullet (x_2 \bullet x_3)$

- distributivitatea uneia faţă de cealaltă:

$(x_1+x_2) \bullet x_3 = x_1 \bullet x_3 + x_2 \bullet x_3$;

$x_1+(x_2 \bullet x_3)= x_1 \bullet x_2 + x \bullet_1 x_3.$

d) mulţimea conţine un element nul - ***0*** şi unul unitate -***1*** care constituie elemente neutre faţă de cele două operaţii şi pentru cvare sunt valabile proprietăţile:

$x_1+0=0+x_1=x_1;$

$x_1 \bullet 1=1 \bullet x_1=x_1$ unde $x_1 \in M$.

e) fiecărui element $x \in M$ îi corespunde un unic invers $\overline{x} \in M$ cu proprietăţile :

$\overline{x} \cdot x = 0$ (principiul contradicţiei)

$\overline{x} + x = 1$ (principiul terţului exclus)

Dacă elementele mulţimii M pot lua numai două valori ***(0 şi 1)*** structura de mai sus reprezintă o *algebră booleană.*

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 2.1.

Tabelul 2.1

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 deduc teoremele prezentate în tabelul 2.2 care se constituie în reguli de calcul în cadrul algebrei booleene.

Tabelul 2.2

Nr.	Denumire	Forma produs	Forma sumă
T1	Dublă negaţie (involuţia)	$\overline{\overline{x}} = x$	$\overline{\overline{x}} = x$
T2	Absorbţia	$x_1 \cdot (x_1 + x_2) = x_1$	$x_1 + x_1 \cdot x_2 = x_2$
T3	Elemente neutre	$x \cdot 0 = 0$	$x + 1 = 1$
T4	Idempotenţa (tautologia)	$x \cdot x \cdot \ldots \cdot x = x$	$x + x + \ldots + x = x$
T5	De Morgan	$\overline{x_1 \cdot x_2} = \overline{x_1} + \overline{x_2}$	$\overline{x_1 + x_2} = \overline{x_1} \cdot \overline{x_2}$

Oricare dintre cele 5 teoreme poate fi demonstrată utilizând axiomele cu ajutorul cărora s-a definit structura algebrei.

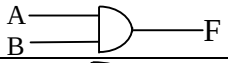
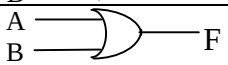
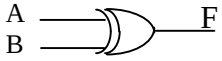
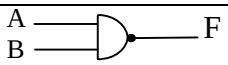
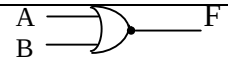
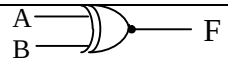
2.1.2. Funcţii logice importante

O funcţie $y=f(x_1, x_2, \ldots, x_n)$ reprezintă o funcţie logică dacă domeniul de definiţie este reprezentat de produsul cartezian $\{0,1\}^n$, cu alte cuvinte $\boldsymbol{f:\{0,1\}^n \rightarrow \{0,1\}}$.

Având în vedere această definiţie se poate spune că o funcţie logică (booleană) pune în corespondenţă o combinaţie binară asociată produsului cartezian cu una din valorile **0** sau **1**.

Domeniul de definiţie al unei funcţii logice de n variabile este format din 2^n puncte (combinaţii), iar numărul total de funcţii este de 2^{2^n}. De exemplu cu 2 variabile pot fi formate 16 funcţii, dintre care în tabelul 2.3 se prezintă cele mai importante.

Tabelul 2.3

Denumire funcţie	**Ecuaţie logică**	**Simbol**
ŞI	$F = A \cap B$	
SAU	$F = A \cup B$	
SAU EXCL.	$A \oplus B = A \bullet \overline{B} + \overline{A} \bullet B = \overline{A \otimes B}$	
NICI EXCL.	$A \otimes B = \overline{A \bullet B} + A \bullet B = \overline{A \oplus B}$	
ŞI- NU	$A \uparrow B = \overline{A \bullet B} = \overline{A} + \overline{B}$	
SAU - NU	$A \downarrow B = \overline{A + B} = \overline{A} \bullet \overline{B}$	

Funcţiile *ŞI, SAU, NU* se numesc *funcţii logice de bază* întrucât cu ajutorul lor se poate exprima orice altă funcţie logică. 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. Pentru funcţiile din tabelul 2.3 se prezintă tabelul de adevăr 2.4.

Tabelul 2.4

A	B	ŞI	SAU	SAU EXCL	NICI EXCL	ŞI - NU	SAU - NU
0	0	0	0	0	1	1	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	1	0	1	0	0

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 2.1 este reprezentată diagrama Karnaugh pentru o funcţie de trei variabile cu marcarea vecinătăţilor punctului *010*.

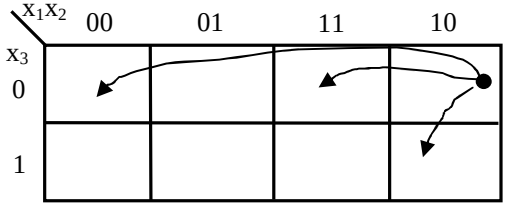


Fig. 2.1. Diagrama Karnaugh pentru o funcţie de trei variabile.

După cum se observă, trecerea de la o combinaţie la alta pe laturile diagramei Karnaugh se face prin modificarea unui singur bit.

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.*

2.1.3. 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.

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.

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.

2.2. Circuite logice 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 2.11. 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).

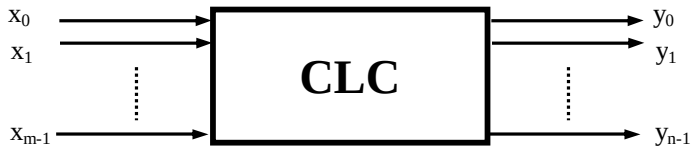


Fig. 2.2. Circuit logic combinaţional

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.

Sinteza *CLC* presupune parcurgerea următoarelor etape pentru stabilirea structurii circuitului:

- definirea funcţiilor logice;
- minimizarea acestora;
- obţinerea schemei circuitului.

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 unor CLC uzuale din structura unui calculator numeric.

2.2.1. Convertoare de cod

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 2.3 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.

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

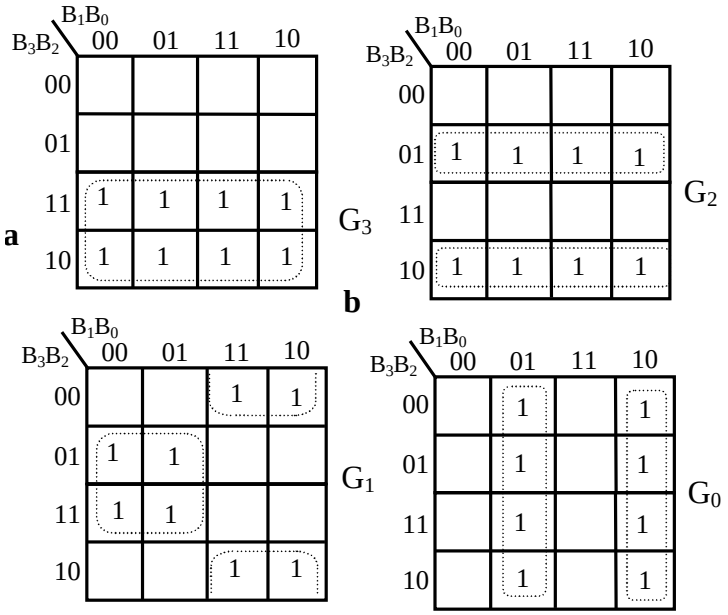


Fig. 2.3. Convertor de cod binar natural - Gray:
a - tabela de corespondenţă;
b - diagramele Karnaugh asociate

Făcând reducerile în diagramele Karnaugh rezultă:

$$G_3 = B_3$$

$$G_2 = \overline{B}_2 B_3 + \overline{B}_3 B_2 = B_2 \oplus B_3$$

$$G_1 = \overline{B}_1 B_2 + \overline{B}_2 B_1 = B_1 \oplus B_2$$

$$G_0 = \overline{B}_1 B_0 + \overline{B}_0 B_1 = B_1 \oplus B_0$$

În figura 2.4 se prezintă două variante de implementare ale relaţiilor de mai sus.

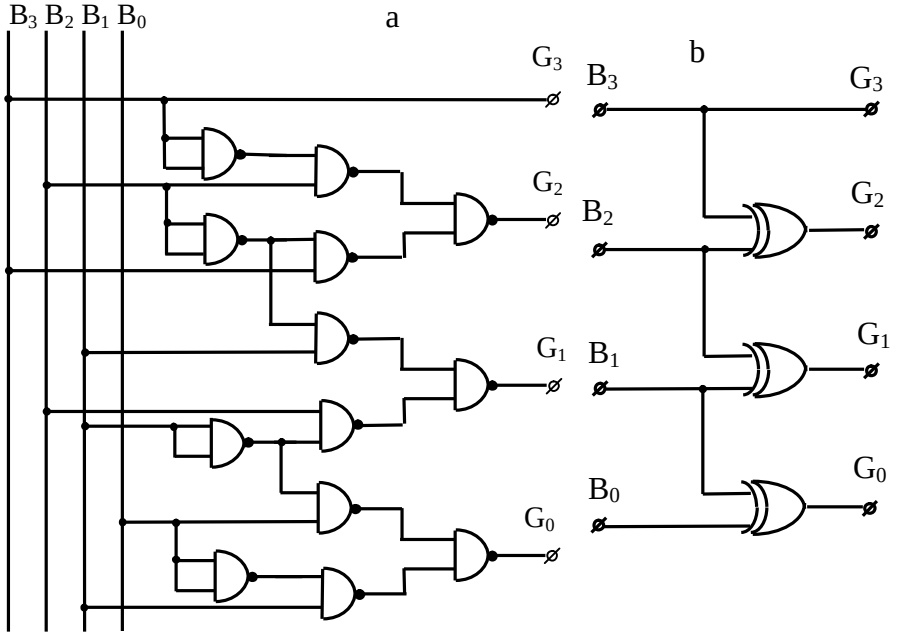
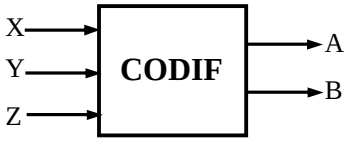


Fig. 2.4. Schema convertorului din cod binar natural în cod Gray:
a - realizarea cu porţi *NAND*; b - realizarea cu circuite *SAU EXCLUSIV* .

2.2.2. 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 ($m \le 2^n$). În figura 2.5 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

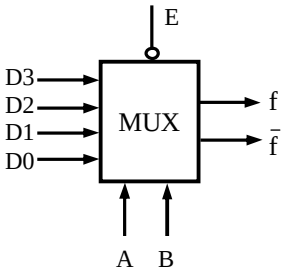
$$A = \overline{X} \cdot (Y \oplus Z)$$
$$B = \overline{Y} \cdot (X \oplus Z)$$

Fig. 2.5. Codificator cu m=3 şi n=2 (schema bloc, tabela de adevăr, funcţii logice.

Decodificatoarele sunt *CLC* care 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.

2.2.3. Multiplexoare şi demultiplexoare

Multiplexoarele sunt *CLC* care permit transferul datelor de la una din intrările selectate cu o adresă (cuvânt de selecţie) către o ieşire unică. Din punct de vedere funcţional *MUX* pot fi privite ca o reţea de comutatoare comandate. *MUX* pot fi analogice sau numerice, ultimele fiind specifice CN. În continuare se va face sinteza unui MUX 4:1 numeric şi implementarea cu porţi logice.



E	B	A	f = Canal
0	0	0	D3
0	0	1	D2
0	1	0	D1
0	1	1	D0
1	*	*	-

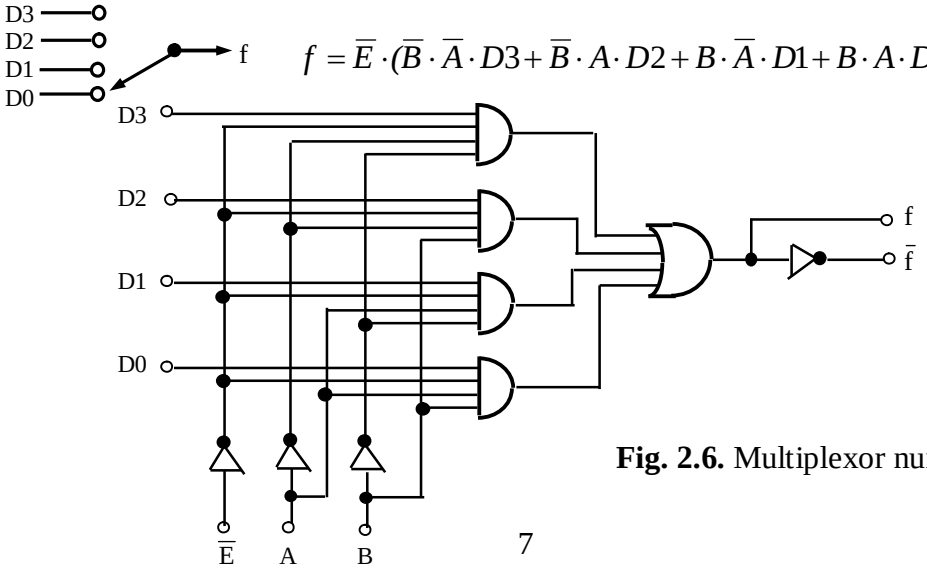
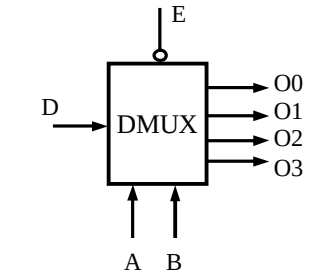
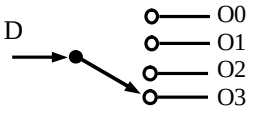


Fig. 2.6. Multiplexor numeric 4:1.

Demultiplexoarele sunt *CLC* care realizează transmiterea datelor de la o unică intrare către o ieşire selectabilă cu ajutorul unui cuvânt de selecţie (adresă). Ca şi *MUX* demultiplexoarele reprezintă practic o reţea de comutatoare comandate, putând fi numerice sau analogice. În figura 2.7 se prezintă elemente specifice sintezei unui DMUX numeric 1:4.



E	B	A	O0	O1	O2	O3
0	0	0	D			
0	0	1		D		
0	1	0			D	
0	1	1				D
1	*	*	-	-	-	-



$O0 = \overline{E} \cdot \overline{B} \cdot \overline{A} \cdot D \quad O2 = \overline{E} \cdot B \cdot \overline{A} \cdot D$

$O1 = \overline{E} \cdot \overline{B} \cdot A \cdot D \quad O3 = \overline{E} \cdot B \cdot A \cdot D$

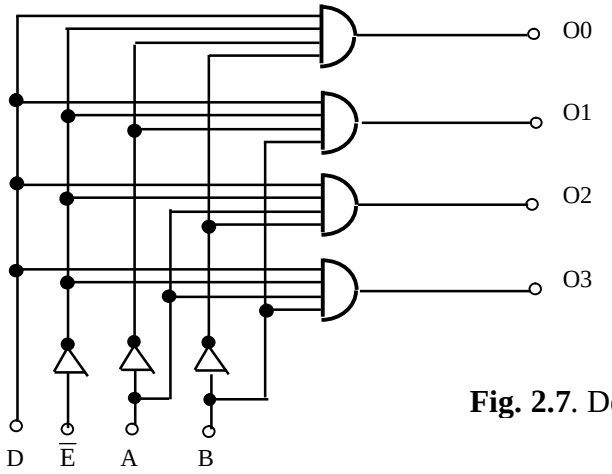


Fig. 2.7. Demultiplexor 1:4

2.2.4. Circuite de complementare

Circuitul de complementare este un *CLC* care funcţie de comenzile aplicate realizează una din următoarele funcţii:

- complementează faţă de unu biţii cuvântului de la intrare;
- lasă cuvântul de la intrare neschimbat;
- forţează în unu toţi biţii cuvântului de la ieşire;
- forţează în zero toţi biţii cuvântului de la ieşire.

În figura 2.8 se prezintă elementele aferente unui circuit de complementare pe 4 biţi. Din tabela de adevăr se obţin următoarele funcţii logice ale ieşirilor:

$$Y_1 = \bar{x}_1 \overline{A}\overline{B} + x_1 \overline{A}B + A\overline{B} = (\overline{x_1 \oplus B}) + A\overline{B}$$

$$Y_2 = \bar{x}_2 \overline{A}\overline{B} + x_2 \overline{A}B + A\overline{B} = (\overline{x_2 \oplus B}) + A\overline{B}$$

$$Y_3 = \overline{x}_3\overline{A}\overline{B} + x_3\overline{A}B + A\overline{B} = (\overline{x_3 \oplus B}) + A\overline{B}$$

$$Y_4 = \overline{x}_4\overline{A}\overline{B} + x_4\overline{A}B + A\overline{B} = (\overline{x_4 \oplus B}) + A\overline{B}$$

a căror implementare s-a realizat cu porţi *ŞI, SAU* şi *SAU-EXCLUSIV*.

Comenzi		I e ş i r i			
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_2	x_3	x_4
1	0	1	1	1	1
1	1	0	0	0	0

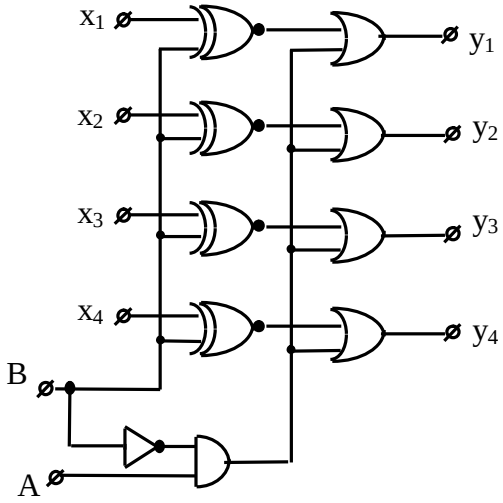
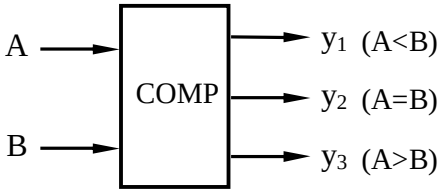


Fig. 2.8. Circuit de complementare pe 4 biţi.

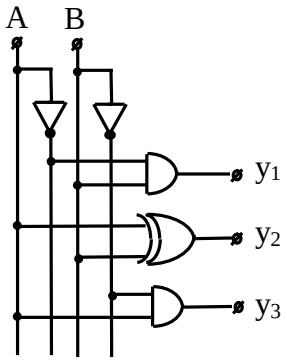
2.2.5. Comparatoare

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 2.9 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. 2.9. Comparator pe un bit.

Prin interconectarea mai multor comparatoare pe un bit se obţin comparatoare pentru cuvinte binare formate din mai mulţi biţi.

2.2.6. Detectoare de paritate

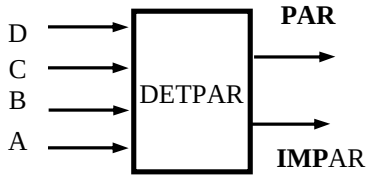
Detectoarele de paritate sunt *CLC* cu *n* intrări şi două ieşiri *PAR* şi *IMPAR* care sunt complementare. Ieşirea *PAR* are valoarea **1** atunci când numărul de valori logice **1** în combinaţia de la intrare este *par* şi **0** atunci când acest număr este *impar*.

În figura 2.10 se prezintă elementele aferente sintezei unui detector de paritate cu *n=4* intrări.

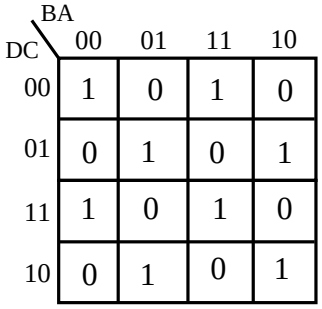
După cum se observă în tabela de adevăr funcţiile *PAR* şi *IMPAR* sunt complementare, respectiv IMPAR= $\overline{PAR}$. Din această cauză în figura 2.10 a fost reprezentată diagrama Karnaugh pentru funcţia *PAR*. Aşa cum reiese din diagramă, nu se poate opera nici o reducere asupra funcţiei care va 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



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. 2.10. 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

$$\overline{D}\,\overline{C} + DC = \overline{\overline{\overline{D}\,\overline{C} + DC}} = (\overline{\overline{\overline{D}\,\overline{C}}}) \cdot (\overline{\overline{DC}}) = \overline{(\overline{\overline{D}} + \overline{\overline{C}}) \cdot (\overline{D} + \overline{C})} = \overline{(D + C) \cdot (\overline{D} + \overline{C})} =$$

$$= \overline{D\overline{C} + \overline{D}C} = \overline{D \oplus C}$$

$$\overline{B}\overline{A} + BA = \overline{B \oplus A}$$

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 2.11.

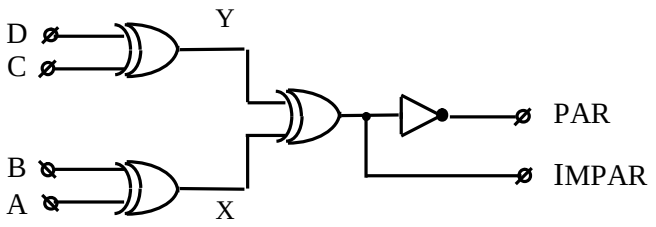


Fig. 2.11. Implementarea detectorului de paritate.

2.2.7. Sumatoare

Semisumatorul elementar, pentru care schema logică şi tabela de adevăr sunt prezentate în figura 2.12, adună două numere a câte un bit x_i, 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

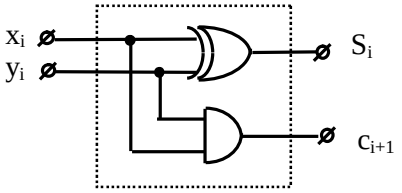


Fig. 2.12. Semisumatorul elementar.

Schema din fig. 2-22 a rezultat pe baza relaţiilor:

$$s_i = \bar{x}_i y_i + x_i \bar{y}_i = x_i \oplus y_i ;$$

$$c_{i+1} = x_i y_i .$$

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 2.5.

Tabelul 2.5

x_i	y_i	c_i	S	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 2.5 rezultă:

$$s_i = \bar{x}_i \bar{y}_i c_i + \bar{x}_i y_i \bar{c}_i + x_i \bar{y}_i \bar{c}_i + x_i y_i c_i = x_i \oplus y_i \oplus c_i$$

$$c_{i+1} = \bar{x}_i y_i c_i + x_i \bar{y}_i c_i + x_i y_i \bar{c}_i + x_i y_i c_i = c_i (x_i \oplus y_i) + x_i y_i$$

Relaţiile de mai sugerează obţinerea sumatorului elementar din două semisumatoare conform figurii 2.13.

Pentru adunarea a două cuvinte de *n* biţi este necesar să se înserieze *n* astfel de sumatoare ca în figura 2.14.

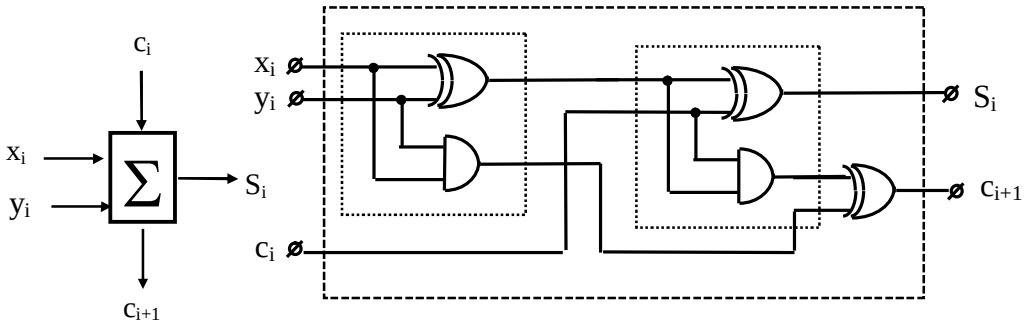


Fig. 2.13. Sumatorul elementar.

Cele două numere care urmează a se aduna se găsesc în registrele A şi B, iar rezultatul în registrul C. Transportul este depus într-un bistabil exterior care pentru un microprocesor este indicatorul de transport CY (Carry).

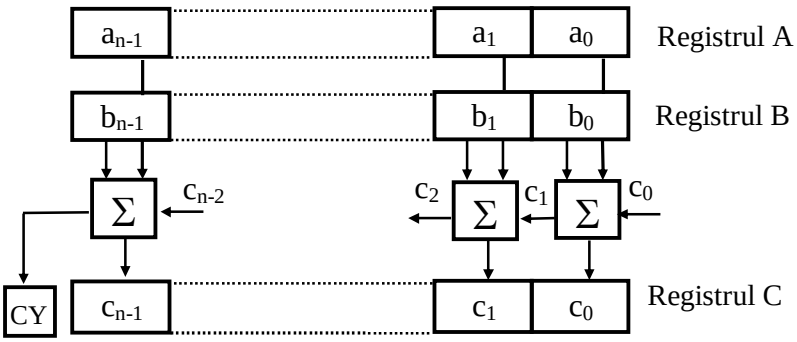


Fig. 2.14. Sumator pentru cuvinte de n biţi.

2.3. Circuite logice 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.

Modelul matematic al *CLS*, pentru un anumit moment de timp *t*, este definit de două seturi de ecuaţii care reflectă tranziţia *stărilor* şi pe cea de *ieşirilor* şi care pot fi grupate în cvintuplul

$C_S=(X,Y,Q,f,g)$,

unde:

$X=\{x_1,x_2,\ldots x_n\}$ este mulţimea variabilelor binare de intrare;

$Y=\{y_1,y_2,\ldots y_m\}$ - mulţimea variabilelor binare de ieşire;

$Q=\{q_1,q_2,\ldots q_p\}$ - mulţimea variabilelor binare de stare;

$f: X \times Q \to Q$ - funcţia de tranziţie a stărilor;

$g: X \times Q \to Y$ - funcţia de tranziţie a ieşirilor.

Funcţiile de tranziţie a *stărilor* respectiv *ieşirilor* sunt de forma

$$q_i^{'} = f(x_1, x_2, \ldots, x_n, q_1, q_2, \ldots, q_p) \quad i=1,2,\ldots,p$$

$$y_k = g(x_1, x_2, \ldots, x_n, q_1, q_2, \ldots, q_p) \quad k=1,2,\ldots,p$$

Funcţiile $q_i^{'}$ şi y_k reflectă procesele de modificare a *stărilor* respectiv *ieşirilor,* ambele dependente doar de intrări şi de starea actuală

Din punct de vedere al structurii CLS conţin elemente combinaţionale şi elemente de memorie, figura 2.15.

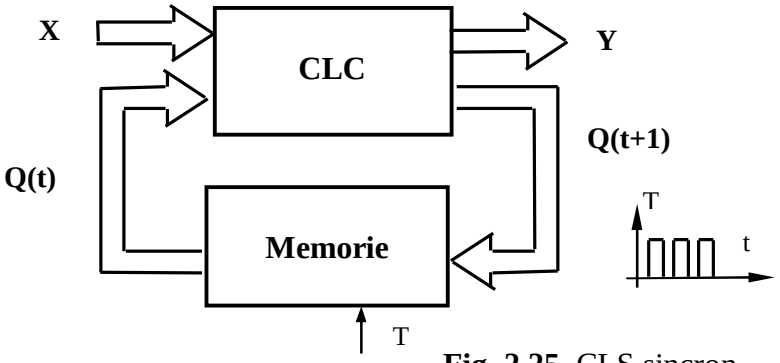


Fig. 2.25. CLS sincron.

În ceea ce priveşte modul de schimbare a stării elementelor de memorie există două tipuri de *CLS*:

• *CLS sincrone* la care modificarea stării se face sincron cu *un impuls de tact,* în funcţie de intrări şi de starea curentă;

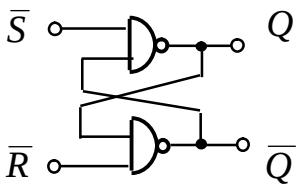
• *CLS asincrone* la care modificarea stării se produce la momente aleatoare depinzând numai de intrări şi de starea curentă.

În continuare vor fi prezentate pentru CLS care se regăsesc în structura unui CN cum ar fi: *bistabile, numărătoare, registre, circuite de memorie.*

2.3.1. Circuite basculante bistabile

Circuitele basculante bistabile (CBB) au două stări stabile la ieşire, iar prin aplicarea unor semnale de comandă trec dintr-o anume stare în starea complementară. Practic un *CBB* implementează un element de memorie care păstrează *un bit* de informaţie. Între cele mai răspândite *CBB* sunt cele de tip *RS, D, T şi JK.*

- *Bistabilul RS asincron* este format din două porţi NAND, fiecare având drept una din intrări ieşirea celeilalte (figura 2.26).



$\overline{S}$	$\overline{R}$	Q(t+1)	$\overline{Q}$ (t+1)
0	0	1	1 int
0	**1**	**1**	**0**
1	**0**	**0**	**1**
1	1(0)	0	1 ned
1(0)	1	1	0 ned

$\overline{S}$	$\overline{R}$	Q(t)	Q (t+1)
1	x	0	0
0	**1**	**0**	**1**
1	**0**	**1**	**0**
x	1	1	1

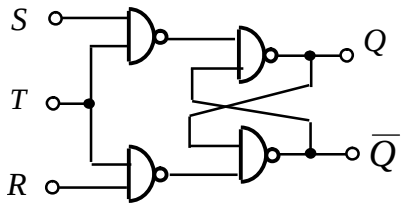
Fig. 2.26. B-RS asincron: schema logică, tabelele de adevăr si de excitatie.

Tabelul de adevăr defineşte *starea ieşirii funcţie de intrări* iar tabelul de excitaţie *defineşte intrarea care determină o anumită evoluţie a ieşirii* – intrările *S* (Set) şi *R* (Reset) sunt active pe *0*.

Combinaţia ***00*** la intrare este interzisă deoarece ieşirile sunt identice şi nu complementare. Combinaţia ***11*** la intrare păstrează starea anterioară *Q(t),* ieşirea fiind o nedeterminare faţă de intrare. Dacă înainte de ***11*** a fost la intrare ***10*** ieşirea este ***0,*** iar dacă înainte de ***11*** a fost ***01*** ieşirea este ***1.***

Se observă că pentru a memora ***1*** (pentru seta *CBB*) trebuie aplicată combinaţia $\overline{S}$ =0 , $\overline{R}$ =1 în timp ce combinaţia $\overline{S}$ =1 , $\overline{R}$ =0 determină memorarea cifrei binare ***0*** (resetarea *CBB*).

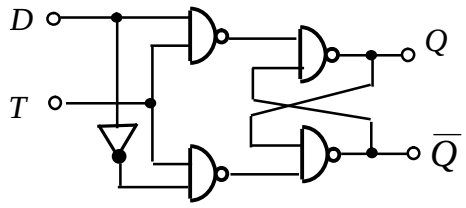
- *Bistabilul RS sincron.* La acest tip de CBB modificarea stării este determinată de un impuls de tact T (figura 2.17). Se observă că CBB RS sincron derivă din cel asincron prin adăugarea unor porţi suplimentare acţionate pe una dintre intrări de semnalul de tact.



T	S	R	Q(t+1)	$\overline{Q}$ (t+1)
1	0	0	Q(t)	$\overline{Q}$ (t)
1	**0**	**1**	**0**	**1**
1	**1**	**0**	**1**	**0**
1	1	1	ned	ned
0	x	x	Q(t)	$\overline{Q}$ (t)

Fig. 2.17. B-RS sincron: schema logică, tabelul de adevăr .

- *Bistabilul D* are o asemenea structură (figura 2.18) încât permite eliminarea stării de nedeterminare specifice *CBB* RS sincron, respectiv a acelei stări pentru care *S=R=1*.

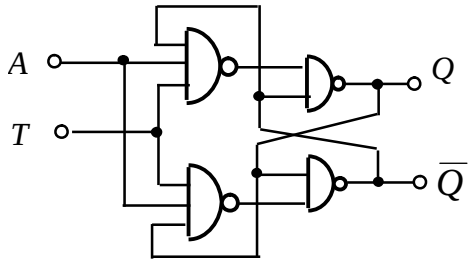


T	D	Q(t+1)	$\overline{Q}$ (t+1)
0	0	Q(t)	$\overline{Q}$ (t)
0	1	Q(t)	$\overline{Q}$ (t)
1	**0**	**0**	**1**
1	**1**	**1**	**0**

Fig. 2.18. Bistabilul D: schema logică, tabelul de adevăr .

Valoarea logică aplicată la intrarea *D* se transferă la ieşire doar la aplicarea semnalului de ceas (deci cu o întârziere de un tact).

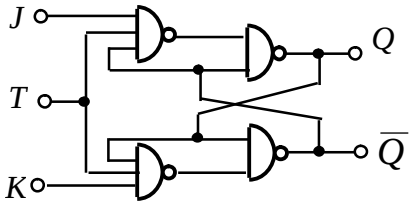
- *Bistabilul T* are proprietatea că schimbă starea la fiecare impuls de tact, dacă o intrare de validare *A* are ***1*** logic (figura 2.19). Dacă *A=0* starea bistabilului (respectiv ieşirea *Q*) rămâne neschimbată.



T	A	Q(t+1)	$\overline{Q}$ (t+1)
0	x	Q(t)	$\overline{Q}$ (t)
1	0	Q(t)	$\overline{Q}$ (t)
1	1	$\overline{Q}$ (t)	Q(t)

Fig. 2.19. Bistabilul T: schema logică, tabelul de adevăr .

- *Bistabilul JK* este un bistabil sincron care admite comenzi simultane pe ambele intrări fără a prezenta o stare instabilă. După cum se observă din figura 2.20 acesta are în plus faţă de *RS* două intrări de reacţie care sunt activate simultan cu semnalele de comandă. Conform tabelului de adevăr la combinaţia *J=0, K=1* ieşirea *Q=0*, iar la combinaţia *J=1, K=0* ieşirea *Q=1*. Circuitul funcţionează şi dacă pe ambele intrări se aplică ***1*** respectiv dacă *J=K=1*.



J	K	Q(t+1)	$\overline{Q}$ (t+1)
0	0	Q(t)	$\overline{Q}$ (t)
0	**1**	**0**	**1**
1	**0**	**1**	**0**
1	1	$\overline{Q}$ (t)	Q(t)

Fig. 2.20. CBB-JK sincron: schema logică, tabelul de adevăr .

2.3.2. Numărătoare

Numărătoarele sunt *CLS* care numără (contorizează) impulsurile aplicate la intrare şi memorează rezultatul.

În funcţie de sistemul de numeraţie folosit se întâlnesc numărătoare *binare, hexazecimale, decadice etc.* Un numărător care poate număra atât înainte cât şi înapoi se numeşte *reversibil*. Practic un numărător realizează, pentru un număr natural *N,* operaţia de identificare a claselor de resturi modulo C ($\overline{0}, \overline{1}, \ldots \overline{c-1}$).

De exemplu, un numărător *modulo 10* va avea aceeaşi stare *3* pentru oricare din următoarele numere aplicate la intrare. *N=3, 13, 23, 33, …, 103, 113, …, 203, 313*. Numărul maxim înscris într-un numărător modulo *c* este *c-1,* deoarece pentru *N=c,* acesta va indica zero.

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 registre de deplasare a căror ieşire este conectată la intrare.

2.3.3. Registre

Registrele sunt *CLS* destinate memorării vectorilor binari. Numărul de biţi egal cu numărul elementelor de memorie reprezintă *capacitatea registrului* sau *lungimea* cuvântului registru. În mod obişnuit registrele sunt constituite dintr-un set de bistabile şi o logică combinaţională auxiliară. Fiecare bit D_i al unui cuvânt binar este păstrat într-un bistabil B_i unde *i=0,1, …,n-1* (registrul are capacitatea de a memora *n* biţi iar *i* este rangul bistabilului *Bi*).

Registrele pot efectua o serie de operaţii cum ar fi:

a) **încărcarea** datelor serială sau paralelă – figurile 2.21 a,b;

b) **deplasare** date stânga sau dreapta - figurile 2.21 c,d;

c) **rotaţie** stânga sau dreapta - figurile 2.21 e,f;

d) **ştergere**.

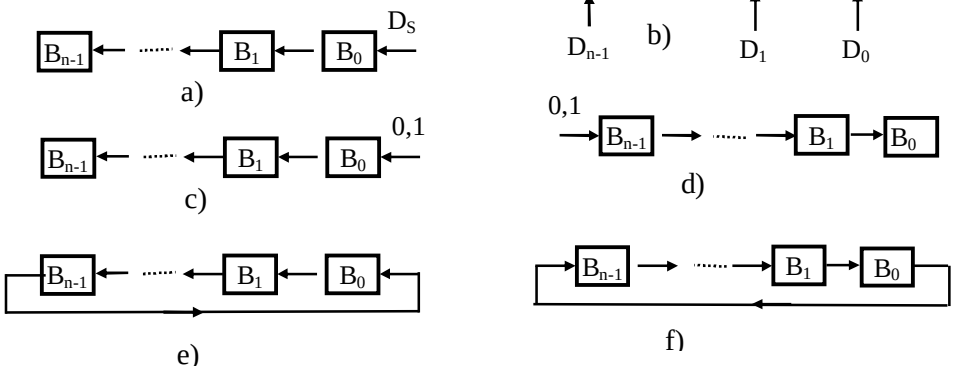


Fig. 2.21. Operaţii cu registre: D_{n-1} – MSB, D_0 – LSB.

- Încărcarea serială se realizează prin *n* impulsuri de tact:

`$B_i \leftarrow B_{i-1}$, $i=1,2,\ldots,n-1$, $R_0 \leftarrow D_S$,

unde *Ds* sunt biţii care se încarcă în B_0 după ce are loc deplasarea spre stânga. Un astfel de registru este folosit la un receptor la care datele sosesc serial pe o linie de 1 bit. Acestea sunt împachetate în cuvinte a câte *n* biţi şi transmise apoi paralel.

- Încărcarea paralelă se realizează într-un singur impuls de tact

$B_i \leftarrow D_i$, $i=0,1,\ldots,n-1$,

un exemplu de utilizare fiind încărcarea unui cuvânt de *n* biţi de pe o magistrală sau dintr-un alt registru.

- Deplasările stânga/dreapta sunt asemănătoare încărcării seriale, cu deosebirea că se execută un singur pas:

$B_i \leftarrow B_{i-1}$, $i=1,2,\ldots,n-1$, $B_0 \leftarrow$ *0 sau 1* (stânga),
$B_{i-1} \leftarrow B_i$, $i=1,2,\ldots,n-1$, $B_{n-1} \leftarrow$ *0 sau 1* (dreapta),

în bistabilele B_0 sau B_{n-1} încărcându-se *0* sau *1* funcţie de contextul utilizării registrului.

- Rotaţiile stânga/dreapta sunt asemănătoare deplasărilor cu deosebirea că bitul care părăseşte registrul reintră în registru (în B_0 la *rotaţia stânga* şi în B_{n-1} la *rotaţia dreapta):*

$B_i \leftarrow B_{i-1}$, $i=1,2,\ldots,n-1$, $B_0 \leftarrow B_{n-1}$ *(stânga),*
$B_{i-1} \leftarrow B_i$, $i=1,2,\ldots,n-1$, $B_{n-1} \leftarrow B_0$ *(dreapta).*

Pentru exemplificare în figura 2.22 se prezintă un registru pe 4 biţi care permite realizarea operaţiilor de ştergere, încărcare şi citire paralelă.

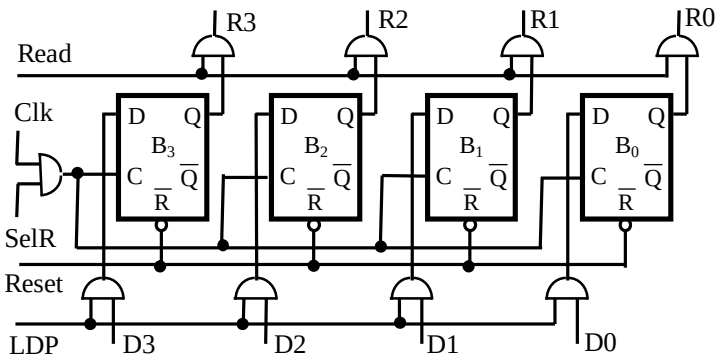


Fig. 2.22. Registru de 4 biti cu încărcare si citire paralelă.

Înscrierea celor 4 bistabile *D* ale registrului se face sincron cu impulsul de tact dacă semnalul de selecţie registru *SelR* este activ. Semnalul *LDP* validează intrările de pe liniile *D3…D0,* iar semnalul *Read* validează ieşirile *R3…R0* ale registrului. Ştergerea registrului (încărcare *CBB* cu *0*, se realizează prin activarea semnalului *Reset.*

În calculatoare *registrele* sunt utilizate la procesarea unor informaţii cum ar fi: *adrese, coduri de instrucţiuni, operanzi, rezultate parţiale sau definitive, informaţii de stare* etc. Una dintre cele mai importante operaţii o constituie transferul între registre. În continuare vor fi prezentate două modalităţi de transfer ilustrate în figurile 2.23 şi 2.24.

Transferul de la un registru sursă *C* la două registre destinaţie *A* şi *B* (figura 2.23) este validat prin activarea simultană a semnalelor *LoadA, LoadB* şi *ReadC.* Selecţia registrelor care se înscriu se realizează prin activarea semnalelor *SelA* şi *SelB.*

În cazul transferului de la două registre sursă *A* şi *B* la un registru destinaţie *C* (figura 2.24), ieşirile registrelor *A* şi *B* sunt reunite în porţile *SAU* de la intrarea registrului destinaţie *C*. Transferul *C-B*, de exemplu se realizează prin activarea semnalelor *ReadB, LoadC şi SelC.*

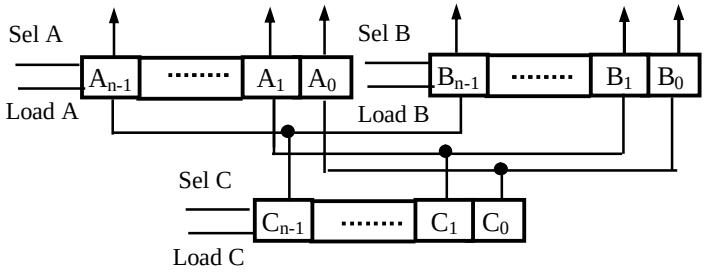


Fig. 2.23. Transfer : un registru sursă – două registre destinaţie.

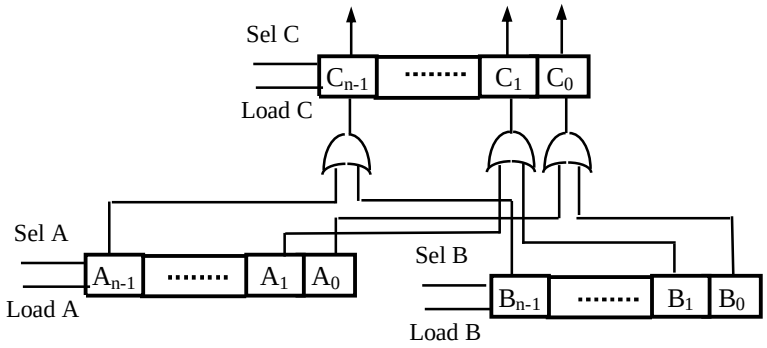


Fig. 2.24. Transfer : două registre sursă – un registru destinaţie.

2. Caracterizarea microprocesoarelor pe 8 biţi

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,** 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=adevarat.

Pornind de la această secvenţiere structura unui μP8 va fi descrisă pe 5 niveluri de detaliere .

2.1. Nivelul 1 de caracterizare

Nivelul 1 este asociat registrelor de date (RD) şi de adrese (RA). Aceste registre apar la interfaţa µP cu magistrala de date (MD) şi respectiv de adrese (MA), figura 2.1.

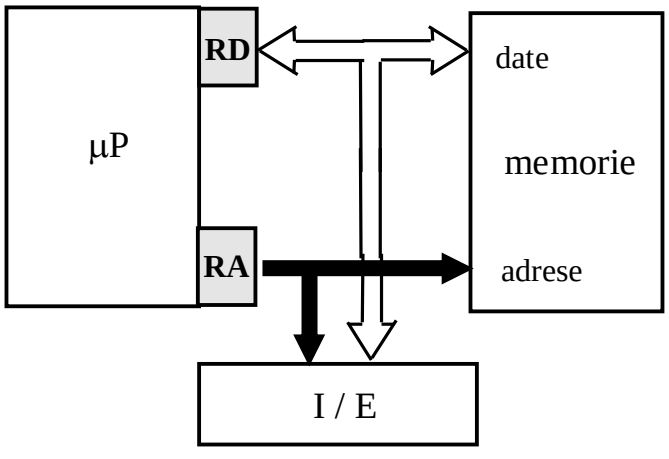


Fig. 2.1. Nivelul 1 de detaliere al unui µP8

RD este bidirecţional ca şi MD şi are lungimea egală cu a acesteia (8 biţi). O informaţie provenită din P este disponibilă unităţilor conectate la MD numai după înscrierea acesteia în RD. Invers, o informaţie destinată P este accesibilă acestuia tot numai după înscrierea în RD.

RA este unidirecţional şi are lungimea impusă de caracteristicile unităţii de control a adresării memoriei. RA 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 P şi periferic.

Atât RD cât şi RA sunt transparente pentru utilizator, acestea nefiind atribute de arhitectură.

2.2. Nivelul 2 de caracterizare

Acest nivel este înglobează registrele generale (RG). Acestea reprezintă practic memoria internă a P 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 2.2, accesul fizic la RG se face cu ajutorul unui multiplexor (MUX) pentru a citi un registru respectiv cu ajutorul unui demultiplexor (DMUX) pentru a înscrie un registru.

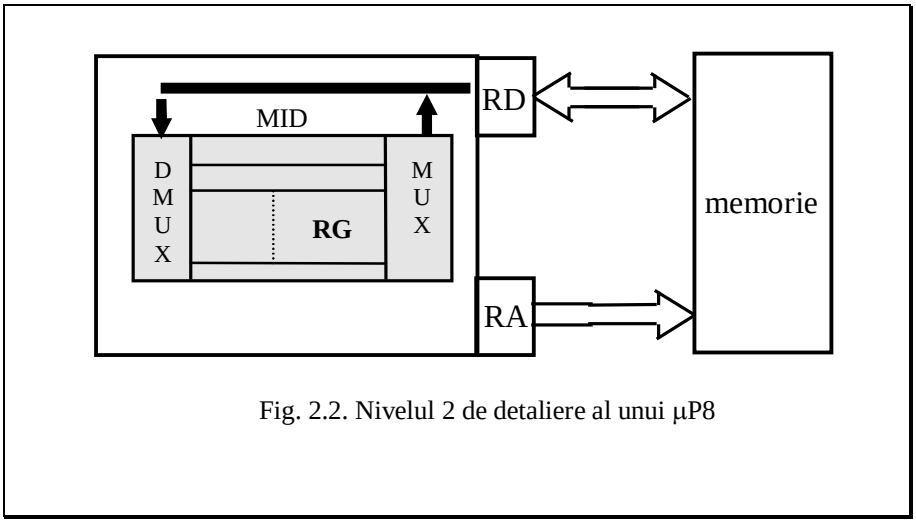


Fig. 2.2. Nivelul 2 de detaliere al unui μP8

După cum se observă din figura 2.2, legătura internă dintre RD şi RG este realizată prin *magistrala internă de date* (MID) care constituie o prelungire a magistralei de date (MD) a sistemului în interiorul P. 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 P. În ceea ce priveşte lătimea 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, P 8080 avea 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*.

2.3. Nivelul 3 de caracterizare

Nivelul 3 include unitatea aritmetică de procesare (UAP). Acest bloc funcţional, a cărui structură este prezentată în figura 2.3, reprezintă suportul activităţii de prelucrare a datelor.

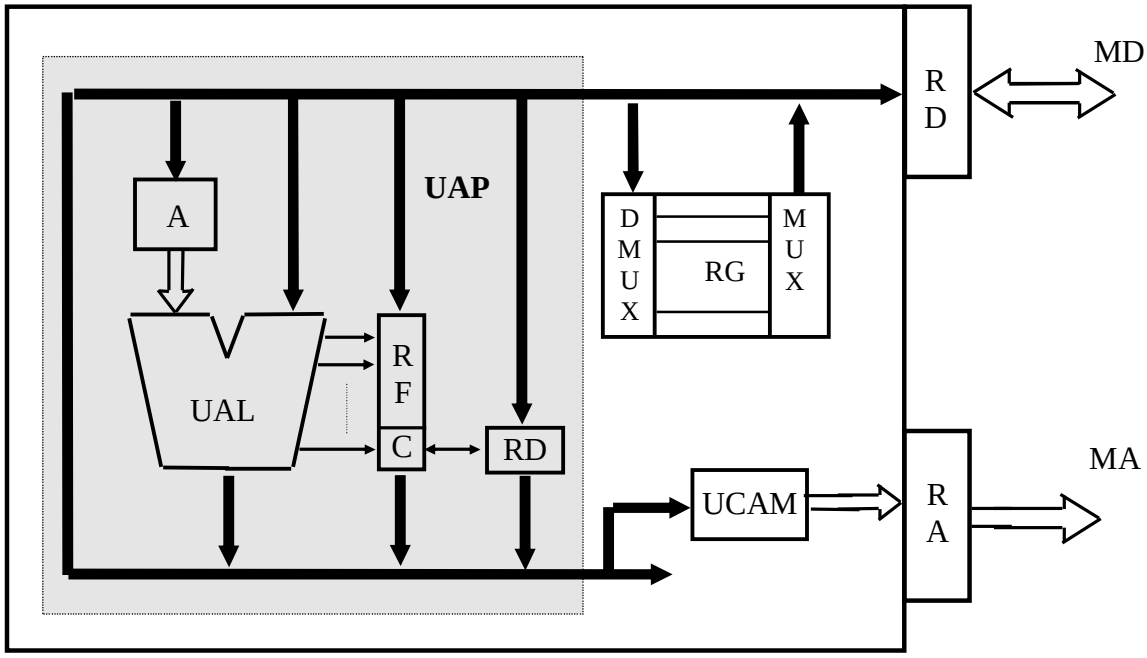


Fig. 2.3. Nivelul 3 de caracterizare a unui μP8:
UAL - unitate aritmetico - logică; A - registru acumulator; RF - registru al fanioanelor; C - fanion (indicator) de transport; RD - registru de deplasare; UCAM - unitate de control a aresării memoriei.

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 P 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 2.3.

Acumulatorul (A) este un registru asemănător cu cele din setul de RG. 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 2.3 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 câte unui acumulator şi registru (temporare), evidenţiate în figura 2.4. Cei doi operanzi sunt menţinuţi în

registrele temporare până când rezultatul prelucrării se înscrie în A. Cele două registre sunt complet transparente pentru utilizator, ele neconstituind elemente de arhitectură.

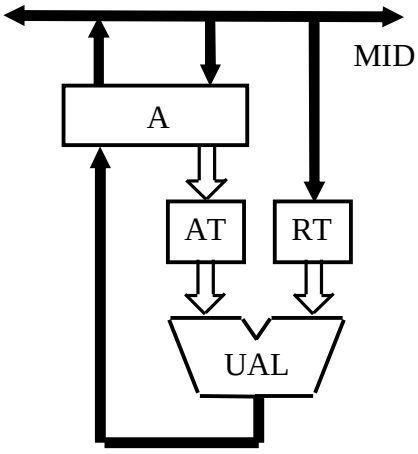


Fig. 2.4. Registre temporare aferente UAP:
AT - acumulator temporar;
RT - registru temporar.

Fanioanele (indicatorii de condiţie) reprezintă bistabili pentru memorarea unor condiţii speciale apăruta în funcţionarea P şi în special a UAL. Ele pot fi grupate într-un registru al fanioanelor, accesibil utilizatorului.

În figura 2.5 se reprezintă conţinutul acestui registru pentru P INTEL 8080, reprezentativ pentru clasa P8, 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) - este **1** în condiţiile existenţei unui transport dinspre bitul 3 spre bitul 4 al *A*, altfel AC este **0**.

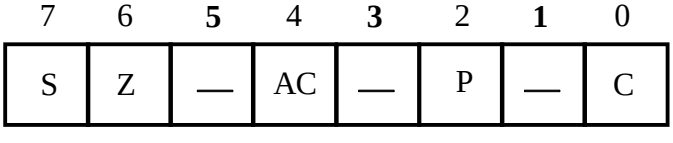


Fig. 2.5. Structura registrului fanioanelor pentru μP 8080.

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 P. 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) ocupă un loc aparte în structura oricărui P8 deoarece cu ajutorul lor se realizează operaţiile de înmulţire şi împărţire 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 2.6 sunt evidenţiate cele două maniere de realizare a deplasării. RD nu este un atribut de arhitectură, el este implicit utilizat de instrucţiunile specifice ale P, dar operatorul nu are acces nemijlocit la acest registru special.

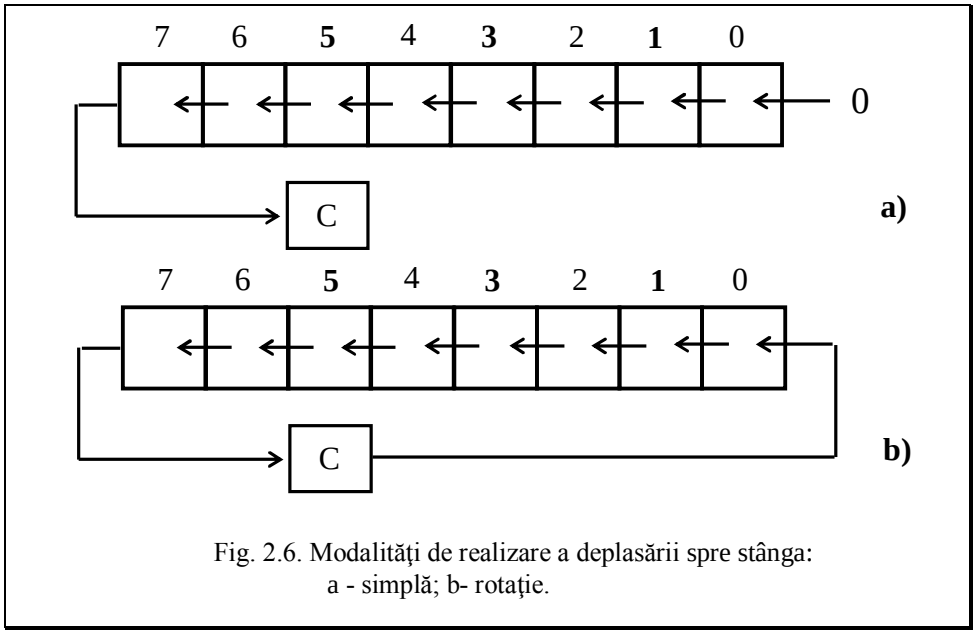


Fig. 2.6. Modalităţi de realizare a deplasării spre stânga:
a - simplă; b- rotaţie.

O secvenţă de utilizare implicită a RD ar putea fi următoarea:

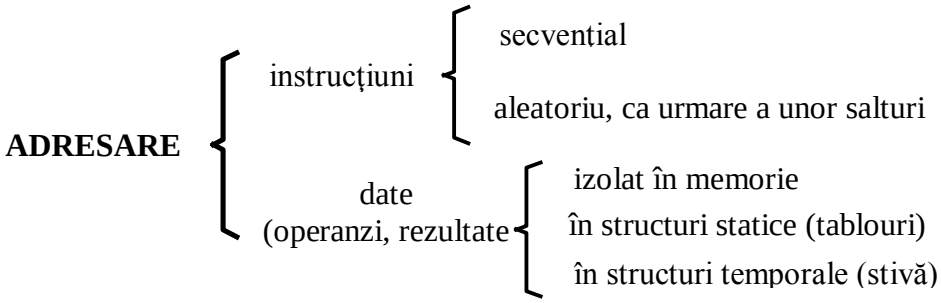
1 - se selectează registrul care conţine operandul;
2 - conţinutul acestuia este adus pe MID;
3 - se înscrie operandul în RD;
4 - se comandă acestuia funcţia de deplasare dorită;
5 - conţinutul RD (deplasat) este transferat pe MID;
6 - se selectează din nou registrul de la pasul 1;
7 - se înscrie în acesta rezultatul operaţiei de deplasare care ia locul operandului iniţial.

Funcţionarea RD este posibilă, dacă acesta este conectat la fanionul C, aspect evidenţiat şi de figura 2.3.

2.4. Nivelul 4 de caracterizare

Acest nivel defineşte unitatea de control a adresării memoriei (UCAM). Această unitate, a cărei conectare cu magistralele interne este evidenţiată în figura 2.7 realizează încărcarea unei adrese în RA în vederea localizării unei informaţii în memoria sau porturile calculatorului.

Structura sa derivă din funcţiile acesteia sintetizate 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 2.7 se prezintă o posibilitate de interconectare.

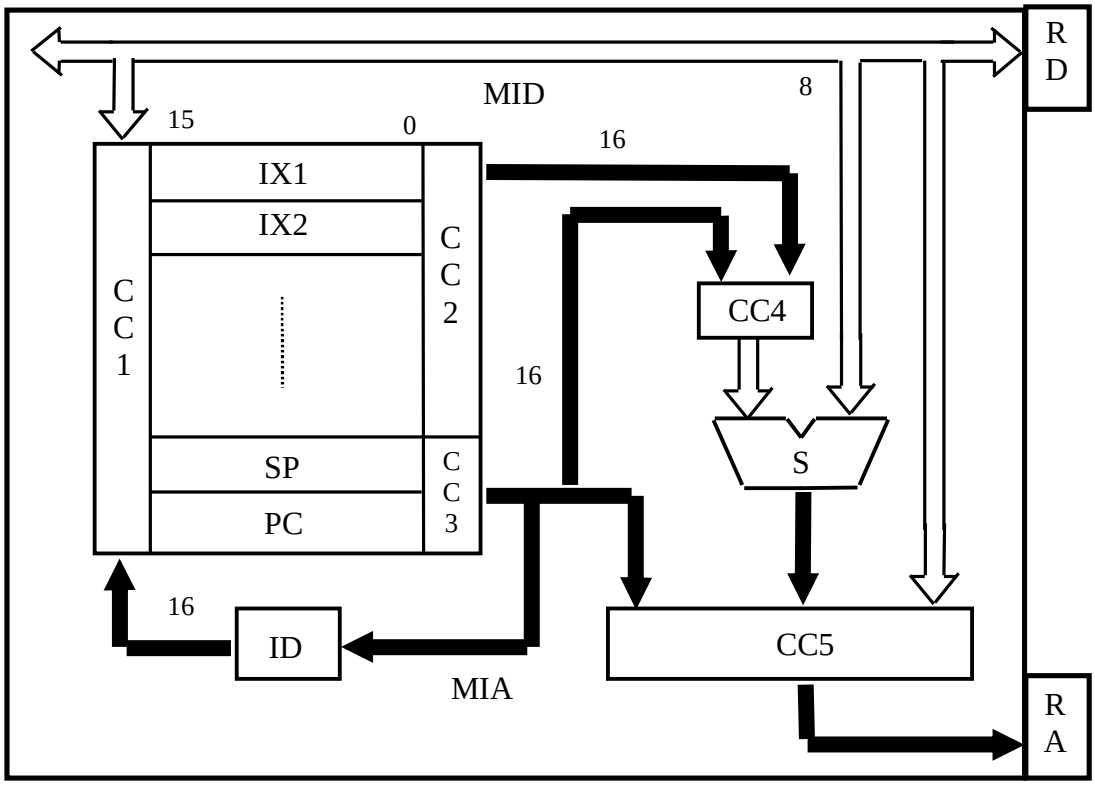


Fig. 2.7. Structura posibilă a unei unităţi de control a adresării memoriei:
PC - numărător de program; SP - registru indicator de stivă; IX - registre index; S - sumator; IDN - circuit de incrementare/decrementare; CC - circuite de conectare ; MIA - magistrală internă pentru formarea adresei.

Numărătorul de program (PC-Programm Counter) conţine adresa fizică (AF) a instrucţiunii ce urmează a fi executată. AF reprezintă forma sub care se furnizează o adresă pe MA 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}ş65536 octeţi ş 64KB (1 Koctet ş 1 Kbyte ş 1024 octeţ*i).

Adresarea secvenţială a memoriei presupune transmiterea conţinutului registrului PC (respectiv a adresei locaţiei adresate) pe calea CC3 - CC5 la RA, urmată de incrementarea PC de către circuitul ID.

Schema din figura 2.7 permite saltul în memoria program (*noua adresă însoţeşte codul instrucţiunii ce urmează a se executa*) conform următoarei secvenţe:

- RA se încarcă prin CC5 direct de pe MID cu adresa instrucţiunii care urmează a se executa;

- concomitent PC se va încărca prin CC1 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). Stiva (*Stack*) reprezintă o structură de date utilizată pentru păstrarea temporară a datelor. Stiva este 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 vrful stivei, încărcarea şi descărcarea acesteia putându-se efectua numai prin acest punct. După cum se observă din figura 2.8 prin operaţiile de încărcare descărcare se modifică adresa *vârfului* stivei.

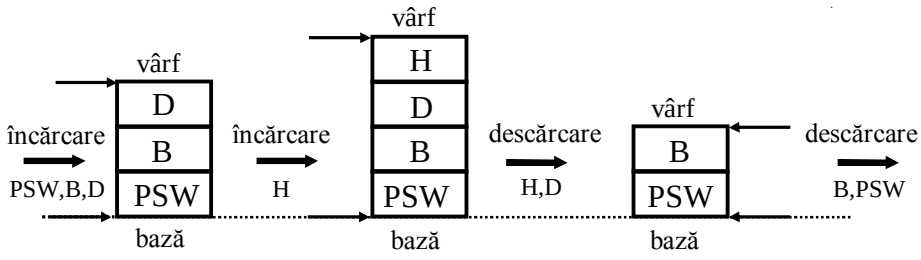


Fig. 2.8. Încărcarea şi descărcarea unei stive:
PSW - registrul stare program; B,C,D - registre duble.

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 2.8 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. Configurarea unei zone de memorie ca stivă se face prin înscrierea în SP a adresei bazei. Stiva se organizează astfel încât creşterea ei să se facă "în jos" adică în sensul descreşterii adreselor. 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 RA cu adresa din SP se face pe calea CC3 - CC5. Figura 2.9 constituie o ilustrare a operaţiunilor aferente lucrului cu stiva.

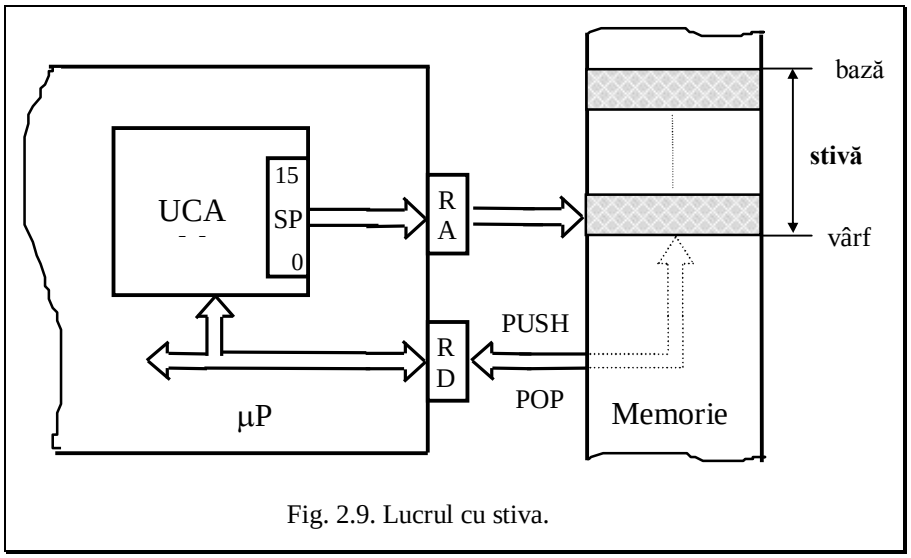


Fig. 2.9. Lucrul cu stiva.

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 □P 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 □P8 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 CC2 - CC4) 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 P8 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.

2.5. 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.* La rândul său fiecare etapă se descompune în acţiuni elementare care vor fi detaliate în continuare.

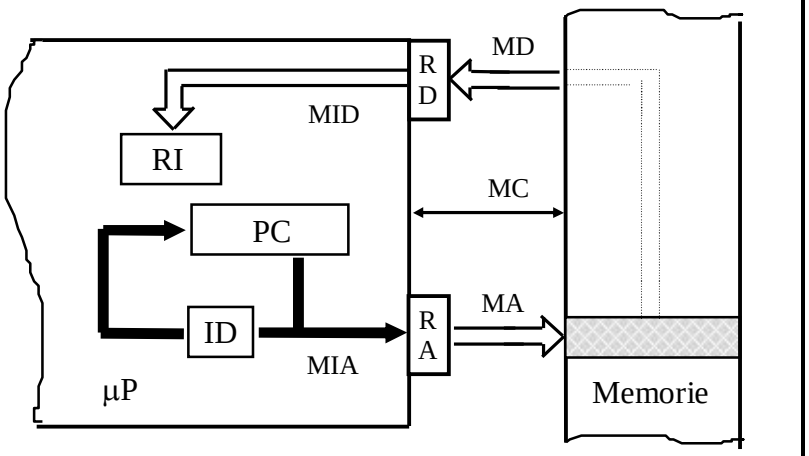


Fig. 2.10. Elemente implicate în extragerea unei instrucţiuni:
RI - registru de instrucţiuni; PC - numărător de program;
ID - circuit de incrementare/decrementare.

1. *Localizarea* şi aducerea în memorie a unei instrucţiuni, etapă la care participă elementele prezentate în figura 2.10, presupune parcurgerea următoarelor faze:

- încărcarea lui RA cu adresa din PC (în cazul uzual al parcurgerii secvenţiale a programului), în urma căreia adresa devine disponibilă pe MA;
- incrementarea lui 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 RD a conţinutului locaţiei identificate;
- transferul codului instrucţiunii, din RD 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.

Având în vedere caracterul de automat sincron al unui 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 UC□P8 a elementelor evidenţiate în figura 2.11. 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.

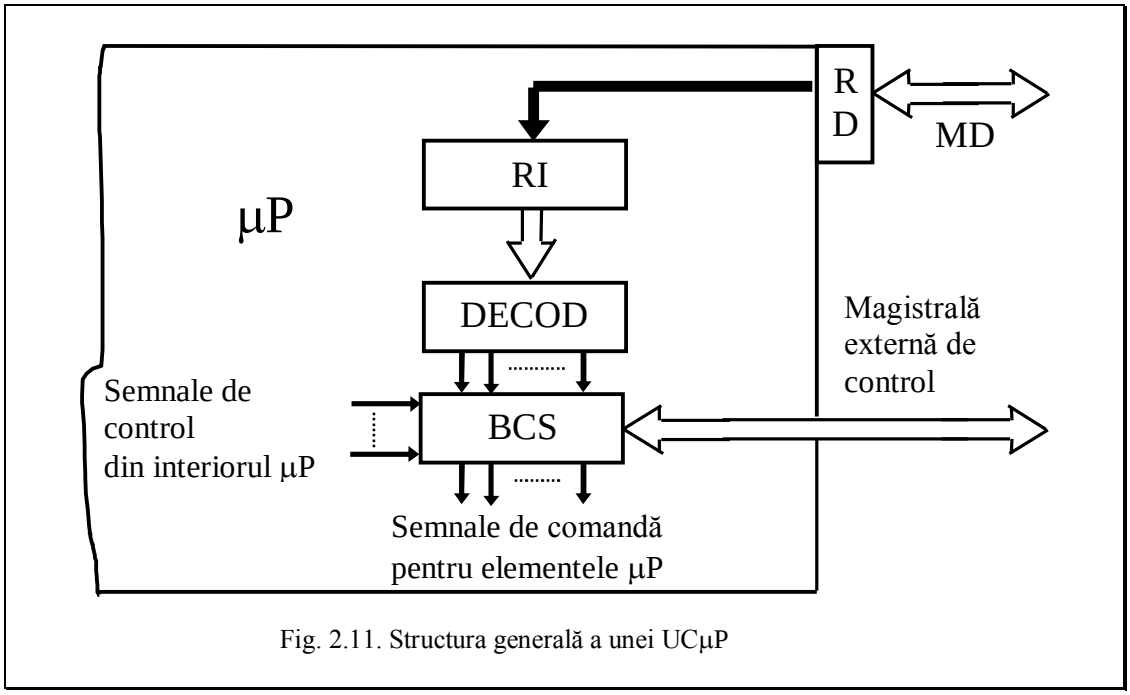


Fig. 2.11. Structura generală a unei UCμP

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.

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.

3. Caracterizarea funcţională a unui microprocesor pe 16 biţi

La patru ani de la prezentarea primului μP8 - I8080 (MD pe 8 biţi, MA pe 16 biţi), firma Intel realizează μP8086 (magistrală unică multiplexată: 16 linii de date, 20 linii de adresă)

Bazat tot pe conceptul clasic al maşinii von Neumann noul μP a preluat majoritatea atributelor de structură şi arhitectură specifice generaţiei precedente (μP8). Dincolo de dublarea capacităţii numărului liniilor de date, în concepţia şi realizarea μP pe16 biţi (μP16) apar elemente de noutate care vor fi prezentate în continuare pe baza schemei funcţionale din figura 3.1.

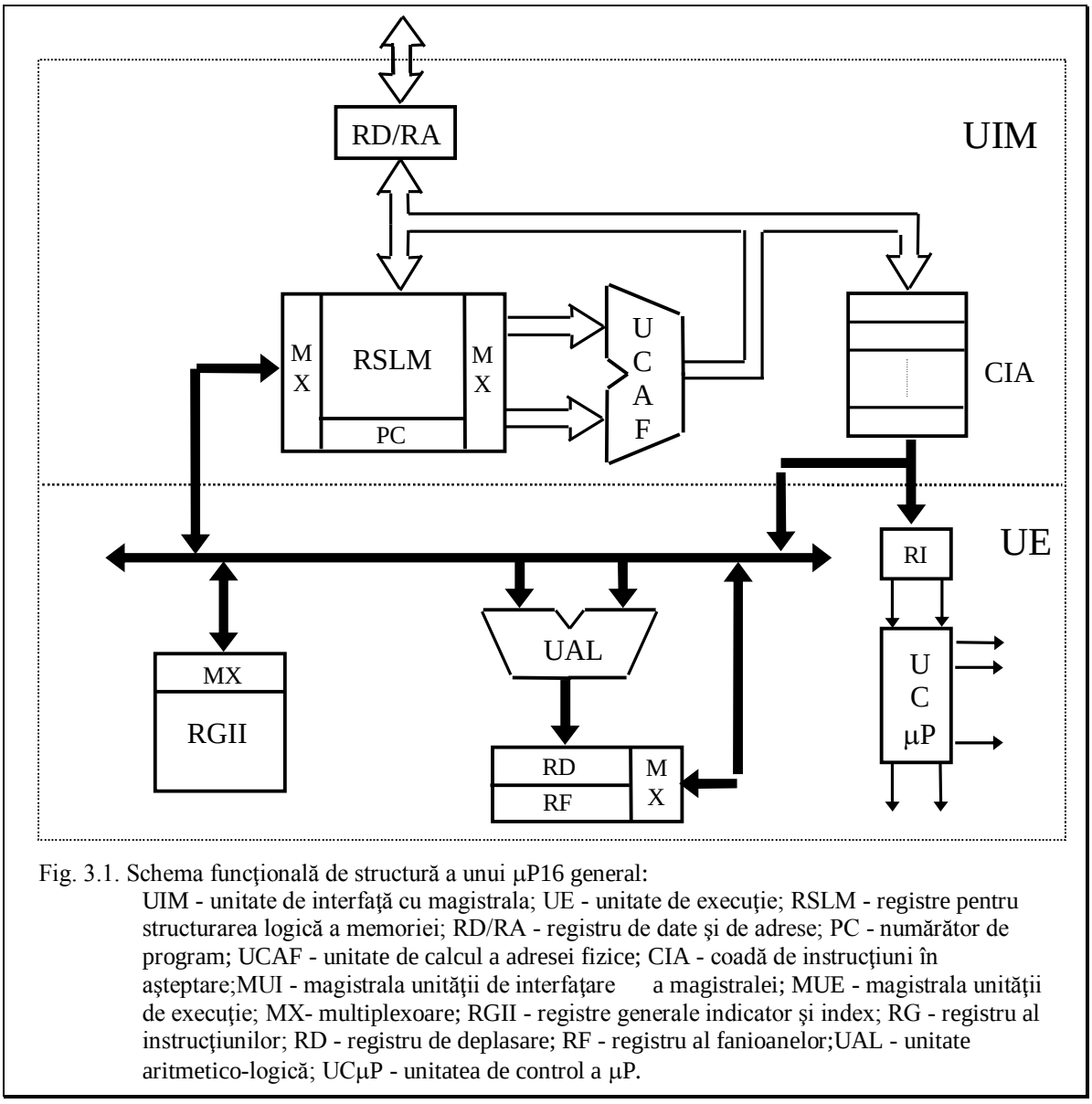


Fig. 3.1. Schema funcţională de structură a unui μP16 general:
UIM - unitate de interfaţă cu magistrala; UE - unitate de execuţie; RSLM - registre pentru structurarea logică a memoriei; RD/RA - registru de date şi de adrese; PC - numărător de program; UCAF - unitate de calcul a adresei fizice; CIA - coadă de instrucţiuni în aşteptare;MUI - magistrala unităţii de interfaţare a magistralei; MUE - magistrala unităţii de execuţie; MX- multiplexoare; RGII - registre generale indicator şi index; RG - registru al instrucţiunilor; RD - registru de deplasare; RF - registru al fanioanelor;UAL - unitate aritmetico-logică; UCμP - unitatea de control a μP.

Deosebirea esențială față de μP8 o constituie existența 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).**

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.

UIM are drept scop mărirea cantităţii de informaţie vehiculată pe magistrală 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 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 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.

1. Funcţiile şi structura unei unităţi centrale de procesare

Unitatea centrală de prelucrare (UCP) reprezintă un subsistem al unui sistem de calcul numeric care efectuează operaţiile esenţiale de prelucrare a datelor şi coordonează activitatea celorlalte subsisteme. Sintetic spus UCP asigură disponibilităţile de calcul şi de comandă a echipamentului numeric.

Pornind de funcţiile menţionate, UCP este structurată într-o *diviziune de calcul şi una de comandă.* Practic UCP este realizată sub forma unui procesor central care realizează funcţiile menţionate. Ultima generaţie de calculatoare a impus procesorul realizat pe un singur circuit integrat pe scară foarte largă (VLSI - Very Large Scale Integrated) numit *microprocesor* (μP).

1.1. Subsisteme ale Unităţii Centrale de procesare

Un sistem de calcul este constituit dintr-un ansamblu de componente funcţionale fizice (*hardware*) şi logice (*software de bază*) care interacţionează în scopul satisfacerii unor cerinţe, adesea contradictorii, ale utilizatorilor. Sistemul de calcul prelucrează date într-o formă specifică şi oferă rezultatele într-o formă accesibilă utilizatorilor. În figura 1.1 este ilustrat modelul topologic de bază al unui calculator care evidenţiază componentele fizice şi poziţia acestora în raport cu cele logice .

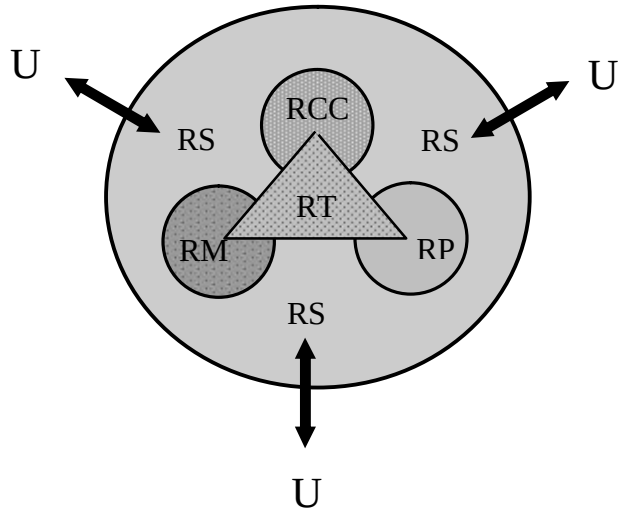


Fig. 1.1. Structura generală a unui sistem de calcul:
RCC - resurse de calcul şi de comandă; RM - resurse de memorare; RP - resurse de introducere şi de extragere a datelor (periferice); RT - resurse de transmisie a datelor; RS - resurse software; U - utilizatori.

După cum se observă componenta software apare ca o anvelopă a componentelor hardware, oferind exploatarea facilităţilor acestora şi interfaţa cu utilizatorii. Din punct de vedere conceptual, funcţionarea unui sistem de calcul poate fi văzută ca un flux de instrucţiuni a căror execuţie este iniţiată de introducerea datelor de prelucrat şi finalizată de obţinerea rezultatelor.

Pornind de la cele două fluxuri, *instrucţiuni şi date*, pot fi realizate patru tipuri de sisteme de calcul şi anume:

a) sisteme cu fluxuri de instrucţiuni şi date unice (**SISD** - **S**ingle **I**nstruction **S**ingle **D**ata);

b) sisteme cu flux de date unic şi mai multe fluxuri de date (**SIMD** - **S**ingle **I**nstruction **M**ultiple **D**ata);

c) sisteme cu mai multe fluxuri de instrucţiuni şi flux de date unic (**MISD** - **M**ultiple **I**nstruction **S**ingle **D**ata);

d) sisteme cu fluxuri de instrucţiuni şi date multiple (**MIMD** - **M**ultiple **I**nstruction **M**ultiple **D**ata).

Arhitectura **SISD** este reprezentată de sistemele clasice *monoprocesor* în timp ce arhitecturii **SIMD** îi sunt specifice sistemele cu *procesare paralelă.* În ceea ce priveşte arhitectura **MISD** aceasta este reprezentată de sistemele cu procesoare *"pipeline"*. Caracteristice pentru arhitectura **MIMD** sunt sisteme de calcul *multiprocesor* şi *reţelele de sisteme de calcul*. În continuare vor fi prezentate trăsături ale UCP aferente sistemelor SISD. Principalele componente ale unei astfel de UCP sunt *unitatea aritmetico-logică, unitatea de comandă şi memoria.*

Unitatea aritmetico-logică (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. În figura 1.2 este prezentată structura principială a unei UAL simple cu operanzii exprimaţi în virgulă fixă.

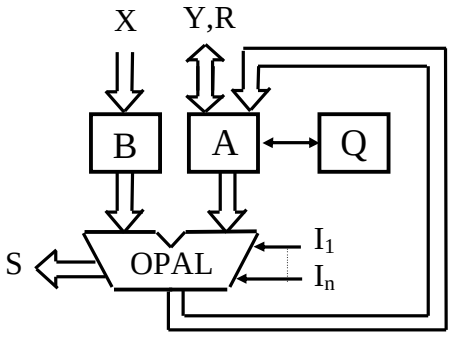


Fig. 1.2. Structura unui UAL în virgulă fixă

UAL este constituită dintr-un operator aritmetico-logic OPAL şi 3 registre A,B,Q. OPAL reprezintă un circuit combinaţional ce constă dintr-un sumator paralel şi un bloc asociat funcţiilor logice. Registrul A păstrează la un moment dat unul din operanzi sau rezultatul operaţiei, iar registrul B conţine al doilea operand. În ceea ce priveşte registrul Q acesta este destinat realizării de înmulţiri / împărţiri prin deplasări succesive. Selectarea operaţiei ce se va efectua la un moment dat se face cu ajutorul variabilelor $I_1,I_2,\ldots I_n$ aplicate de asemenea la intrarea OPAL. În afara rezultatului care se obţine în registrul A, la ieşirea OPAL se mai obţine un cuvânt de stare format din indicatori de condiţie.

Unitatea de comandă (UC) reprezintă un subsistem al UCP care generează succesiunea de semnale de comandă pentru coordonarea activităţii întregului sistem de calcul *(transferuri de informaţie, operaţii aritmetico-logice, citire, interpretare şi execuţie instrucţiuni*). În funcţie de modul de implementare se întâlnesc UC convenţionale (cablate) şi UC microprogramate.

UC *convenţională* reprezintă un automat secvenţial în care semnalele de stare testate de aceasta sunt considerate semnale de intrare, iar semnalele generate pentru controlul resurselor sistemului sunt semnale de ieşire.

UC *microprogramată* generează semnalele de comandă necesare funcţionării sistemului numeric prin citirea unor cuvinte de comandă stocate într-o memorie specială.

Realizarea funcţiilor UCP necesită conectarea acesteia cu celelalte unităţi ale sistemului de calcul. Schimbul de informaţie se realizează prin intermediul *magistralelor de comunicaţie* care reprezintă grupuri de interconexiuni cu funcţii similare ce leagă diversele secţiuni ale unui sistem de calcul. La nivelul unui sistem de calcul principial există două categorii de magistrale: *interne şi externe*. Magistralele interne conectează unităţi ale CPU iar cele externe interconectează modulele sistemului. Fiecare din cele două categorii de magistrale poate fi divizată funcţie de tipul informaţiei transferate în magistrale de *adrese*, de *date* şi de *comenzi*.

Funcţie de tipul transmisiei magistralele pot fi *paralele* (MP) sau seriale (MS). MP transferă toţi biţii unui cuvânt binar simultan, deci numărul de linii este egal cu lungimea cuvântului. MS sunt formate dintr-o singură linie prin care cuvintele binare se transmit bit cu bit. Practic, toate magistralele interne ale UCP şi cele care conectează UCP cu memoria şi cu interfeţele de intrare-ieşire sunt organizate ca MP. În ceea ce priveşte MS acestea sunt utilizate pentru a conecta unele periferice (mouse, imprimante etc.) sau pentru interconectarea sistemelor de calcul în cadrul reţelelor. În figura 1.3 este reprezentat modul de interconectare a principalelor module ale unui sistem de calcul prin intermediul magistralelor.

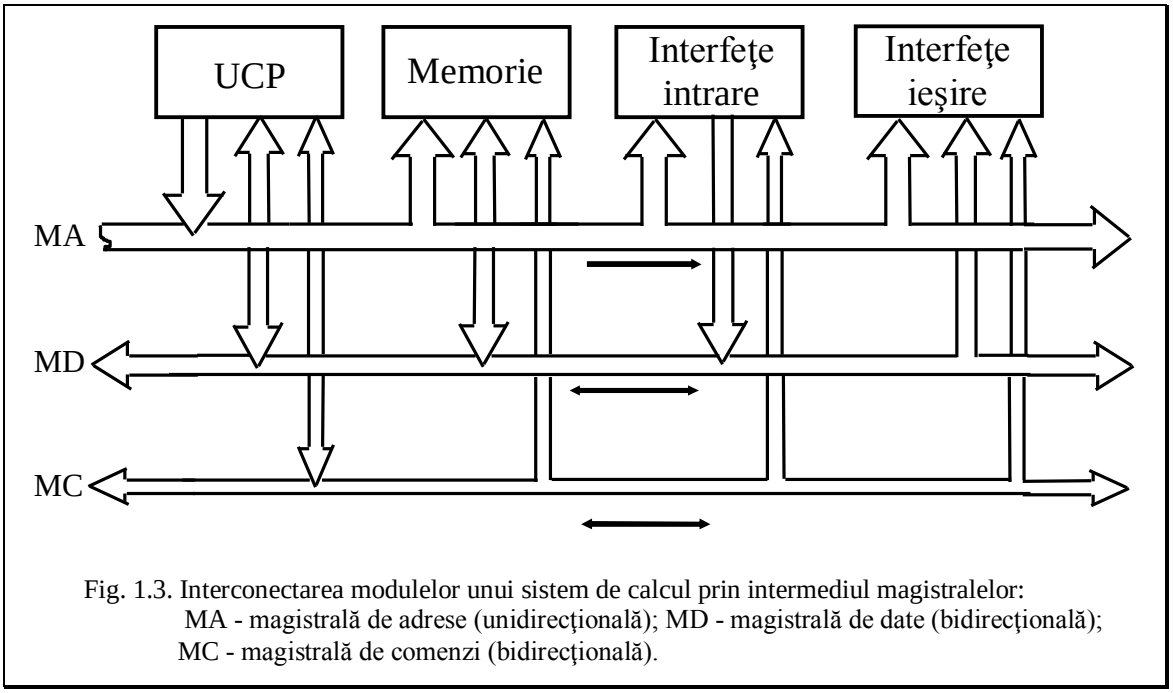


Fig. 1.3. Interconectarea modulelor unui sistem de calcul prin intermediul magistralelor:
MA - magistrală de adrese (unidirecţională); MD - magistrală de date (bidirecţională);
MC - magistrală de comenzi (bidirecţională).

Magistrala de date este destinată atât transferului unidirecţional de instrucţiuni de la memorie, cât şi a celui bidirecţional de date între UCP, memorie şi interfeţele de intrare-ieşire. Sensul transferului de informaţie este supervizat de către unitatea de comandă a UCP. Numărul de linii al magistralei de date şi implicit lungimea cuvântului de date (8, 16, 32, 64 biţi) constituie o caracteristică importantă a sistemului de calcul.

Magistrala de adrese este destinată transferului unidirecţional al cuvântului binar ce reprezintă adresa locaţiei de memorie al cărui conţinut urmează a fi citit. Numărul de biţi ce poate fi transferat prin magistrala de adrese determină o altă catacteristică a sistemului şi anume capacitatea memoriei ce poate fi adresată.

Magistrala de control vehiculează semnale care asigură coordonarea activităţilor într-un sistem de calcul. Numărul liniilor acestei magistrale este egal cu numărul semnalelor de control al UCP, dintre care pot fi menţionate: semnalul de ceas pentru sincronizare, semnalele de citire/scriere din şi în memorie sau de la interfeţele de intrare/ieşire etc.

1.2. Conceptul de microprocesor

Microprocesorul (μP) reprezintă un circuit logic 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, care se numeau 4004 şi 8008. Acestea aveau

lăţimea magistralei de date de 4 şi respectiv 8 biţi şi se consideră că aparţin generaţiei întâi de μP.

Anul 1976 marchează apariţia generaţiei a doua de μP, prin lansarea produselor 8080 al aceleiaşi firme şi Z80 al firmei Zilog. Este de remarcat faptul că Z80 a fost un μP cu performanţe deosebite pentru generaţia sa, în acest sens prefigurând într-o anumită măsură noi concepte specifice viitoarelor generaţii de μP.

Lansarea pe piaţă, de către firma Intel a μP 8086 marchează pe lângă trecerea la μP cu lungimea magistralei de date de 16 biţi apariţia generaţiei a treia. La scurt timp şi firmele Motorola şi Zilog lansează produsele M68000 şi Z8000 din cadrul aceleiaşi generaţii. O manevră interesantă a fost lansarea de către Intel în 1979 a μP 8088, care este identic în interior cu 8086 iar în exterior lucrează pe 8 biţi. Această apariţie venea în sprijinul deţinătorilor de sisteme pe 8 biţi care voiau să beneficieze de facilităţile oferite de noul produs pe 16 biţi. Consacrarea definitivă a produselor Intel se produce după ce firma IBM lansează microsistemele ***IBM PC-XT*** care folosesc μP 8086/8088.

În anul 1993 Intel lansează primul μP al generaţiei a cincea şi anume Pentium. Aceasta inaugurează seria μP pe 64 biţi cu o arhitectură puternic diferită faţă de μP precedente. Consideraţiile privind evoluţia μP au avut în vedere dispozitive ale căror caracteristici şi arii posibile de aplicaţie s-au dezvoltat pornind de la tipurile de bază lansate în anii ’70. Acestea sunt cunoscute ca *microprocesoare de uz general* sau pur şi simplu *microprocesoare*.

Pot fi identificate însă şi alte direcţii de dezvoltare spectaculoasă a dispozitivelor de procesare a datelor.

Procesoarele **RISC** (**R**educed **I**nstruction **S**et **C**omputer) prezintă un set redus de instrucţiuni în vederea creşterii vitezei de prelucrare. Ideea de bază a arhitecturii RISC constă în reducerea setului instrucţiunilor de bază la un minimum necesar, punând acces pe cele folosite mai des şi optimizându-le pentru cea mai rapidă execuţie posibilă. Restul instrucţiunile trebuie să poată fi executate prin combinarea celor selectate. Viteza de execuţie la un procesor RISC este de *1,5-1,7* ori mai mare decât la unul **CISC** (**C**omplex **I**nstruction **S**et **C**omputer).

O evoluţie interesantă a au **transputerele**, procesoare care pot asigura o prelucrare paralelă a datelor. Tot în categoria procesoarelor situate în afara uzului general trebuie menţionate circuitele **ASIC** (**A**pplication **S**pecific **I**ntegrated **C**ircuit) în rândul cărora un loc aparte îl ocupă procesoarele de semnal.

Dinamica extraordinară a tehnicii de calcul poate fi mai bine receptată, dacă amintim faptul că primul calculator elctronic a apărut în anul 1947, iar evoluţia acestora a fost marcată de progresul continuu al tehnologiei electronice. Calculatoarele la care UCP este realizată cu μP aparţin generaţiei a patra, celelalte generaţii având ca tehnologii dominante: tubul electronic pentru generaţia întâi, componentele statice (tranzistoare) discrete pentru generaţia a doua şi circuitele integrate pe scară redusă pentru generaţia a treia. Datorită unui raport performanţă/cost în continuă creştere, calculatorul penetrează în mediile cele mai diverse fiind considerat principalul factor favorizant al trecerii la societatea post-industrială informatizată.

Indiferent cărei generaţii aparţin, toate tipurile de calculatoare prezintă două caracteristici comune şi anume:

- lucrează pe baza unui program memorat;

- comunicarea programatorului cu calculatorul se realizează prin intermediul limbajelor artificiale, înlocuirea acestor limbaje cu cel natural reprezentând unul din obiectivele importante ale generaţiei a cincea de calculatoare.

Conceptul de calculator numeric cu program memorat a fost introdus de John von Neumann în anul 1947. Potrivit acestui concept instrucţiunile şi datele sunt memorate împreună şi sunt accesibile în acelaşi mod. În figura 1.4, care prezintă structura de bază a unui calculator numeric bazat pe o maşină von Neumann, este evidenţiat faptul că în memorie sunt stocate atât secvenţele de instrucţiuni ce constituie programul, cât şi datele (de intrare, intermediare, de ieşire). Prin secţiuni specifice calculatorul schimbă informaţie cu echipamentele periferice care pot fi atât convenţionale (tastatură, ecran, imprimantă etc.), cât şi de proces (traductoare şi elemente de execuţie).

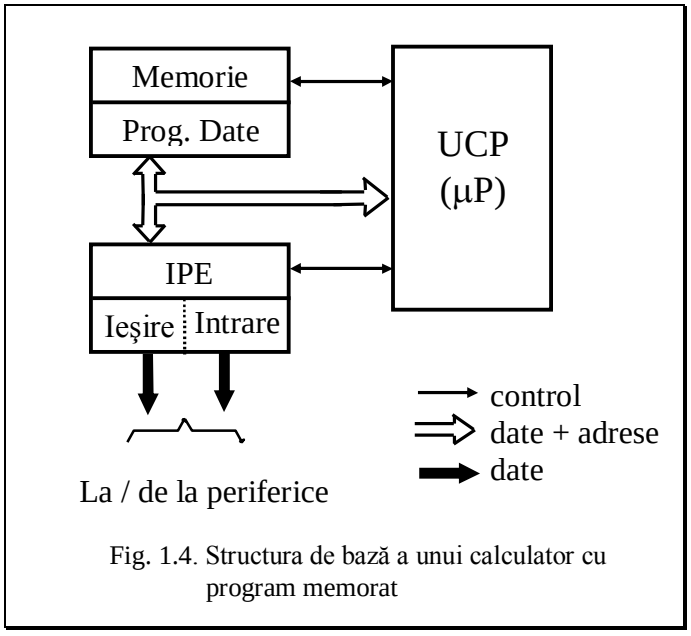


Fig. 1.4. Structura de bază a unui calculator cu program memorat

Practic un calculator numeric realizează cinci funcţii de bază 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 1.5 secţiunile **aritmetico-logică** şi **control** sunt integrate în μP.

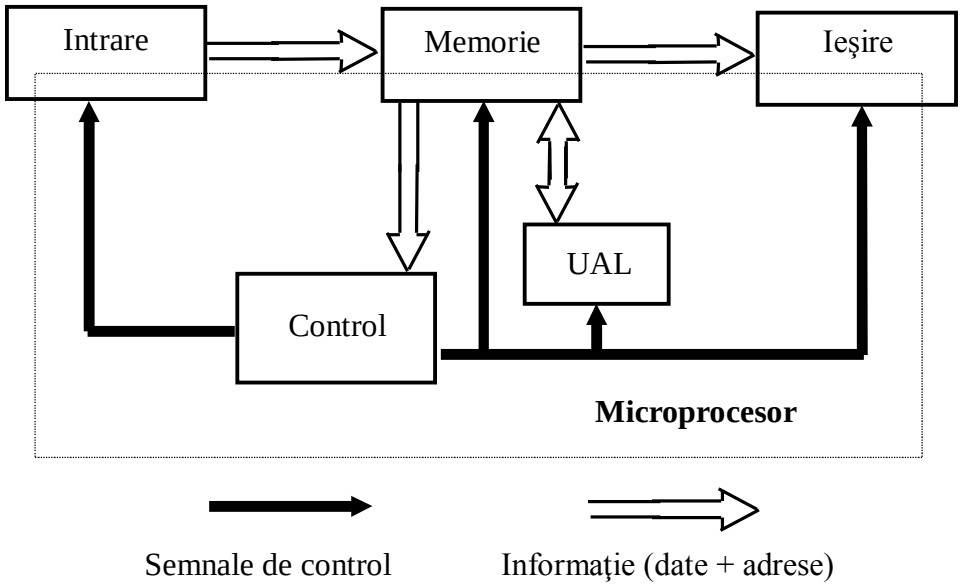


Fig. 1.5. Structura funcţională şi fluxul de informaţie într-un sistem cu microprocesor

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 1.6, 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.*

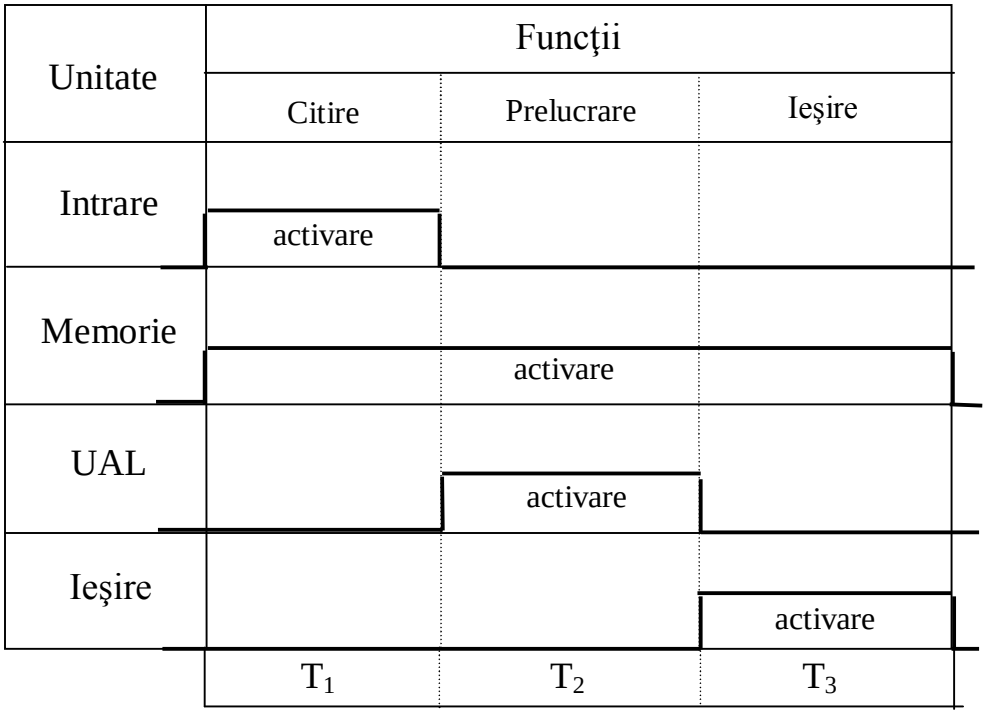


Fig. 1.6. Secvenţierea operaţiilor într-un sistem numeric cu microprocesor