10. OPERAȚIILE ARITMETICE

10.1 PROCESORUL ARITMETIC

Un procesor aritmetic reprezintă un dispozitiv capabil să efectueze operații simple sau complexe asupra unor operanzi furnizați în formate corespunzătoare. Ca exemple se pot da:

- Unitatea Aritmetică simplă;
- Incrementatorul;
- Dispozitivul pentru Transformata Fourier Rapidă etc.

Procesorul Aritmetic poate fi examinat atât din punctul de vedere al utilizatorului, cât şi al proiectantului.

10.1.1 Din *punctul de vedere al utilizatorului*, procesorul aritmetic reprezintă o cutie neagră, cu un număr de intrări şi ieşiri, capabilă să efectueze o serie de operații asupra unor operanzi cu formate specificate. Rezultatele se obțin într-un timp, care depinde de tipul operației şi de formatul operanzilor şi sunt însoțite de indicatorii de condiții.

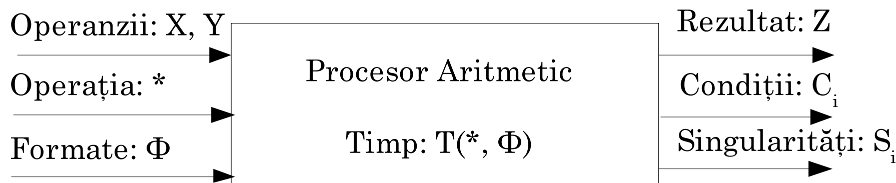


Figura 10.1. Procesorul aritmetic

Intrări:

- una sau mai multe mărimi numerice: X, Y;
- un simbol al operației, operatorul: *;
- un simbol de format: Φ.

Operanzii de la intrare sunt caracterizați prin trei proprietăți:

- aparțin unei mulțimi finite *M* de mărimi numerice, caracterizate printr-o gamă:

$$X_{min} \leq X \leq X_{max}$$

sunt cunoscute cu o precizie dată:

$$X - \Delta X_1 \leq X \leq X + \Delta X_h$$

sunt reprezentate cu ajutorul unor simboluri/cifre, în cadrul unui sistem de numerație, sub forma unor *n*-tupluri: $x_{n-1}, x_{n-2}, \dots, x_1, x_0$ care sunt interpretate ca mărimi/valori numerice, pe baza unor reguli date.

Operatorii sunt codificați cu ajutorul unor simboluri *, care corespund unui set redus sau extins de operații aritmetice:

$$* \in \{+, -, \times, :\}$$

Formatul. Atunci când sunt posibile mai multe formate, pentru reprezentarea operanzilor, acest lucru poate fi specificat la intrarea *format*, printr-un simbol dat ϕ.

Ieşiri:

- una sau mai multe mărimi numerice Z, care reprezintă rezultatul;
- unul sau mai multe simboluri C_i, reprezentând condițiile în care apare rezultatul;
- unul sau mai multe simboluri S_i, reprezentând singularități.

Ieşirile numerice posedă aceleaşi proprietăți ca şi operanzii de la intrare: gamă, precizie şi reprezentare.

Condițiile specifică caracteristici ale rezultatului: < 0, = 0, > 0 etc.

Singularitățile sunt asociate cu situațiile în care rezultatul obținut este invalid:

- *depăşire*: rezultatul depăşeşte posibilitățile hardware de reprezentare a numerelor, în *sistemul numeric dat;*
- *pierderea excesivă de precizie*, la operațiile în virgulă mobilă;
- *erori datorate hardware-lui*.

În aceste situații apare un pseudorezultat $Z(S_i)$, împreună cu singularitatea S_i, care sunt tratate atât prin hardware, cât şi cu ajutorul unor rutine specifice ale sistemului de operare.

Timpul de operare T(*) este dat pentru fiecare operație (*), efectuată de către procesor. Timpul de operare, în unele cazuri, poate fi variabil:

$$T(*)_{min} \leq T(*) \leq T(*)_{max}$$

Observații:

- definiția dată procesorului aritmetic cuprinde un spectru larg de cutii negre”, de la un contor simplu (ADD-ONE), până la un generator de funcții trigonometrice sau un procesor FFT.
- structura internă este specificata în termenii timpului asociat cu execuția diferitelor operații, cât şi cu formatul de reprezentare a datelor.

10.1.2 Din *punctul de vedere al proiectantului* interesează specificarea detaliată a structurii interne.

Această specificare trebuie să considere:
- algoritmii aritmetici (proiectarea aritmetică),
- structura logică a procesorului (proiectarea logică).

Proiectarea aritmetică pleacă de la specificațiile date de către utilizator şi le transforma în specificatii de operații aritmetice detaliate, la nivel de ranguri individuale/bit, în cadrul reprezentării concrete a datelor. Aceste specificatii, la nivel de rang individual, reprezintă, în continuare, datele inițiale (tabele de adevăr, diagrame etc) pentru proiectarea logică.

Proiectarea logică pleacă de la specificațiile furnizate de către proiectarea aritmetică şi, în cadrul unei tehnologii date, selecteaza tipurile de circuite logice, pe care le interconecteaza, în mod corespunzator, în vederea implementarii operațiilor aritmetice impuse de către algoritmii aritmetici. În cazul în care algoritmii aritmetici nu se pot executa într-un singur pas, se proiecteaza secvențe, constand în pasi aritmetici elementari, efectuati sub controlul unor semnale de comanda. Astfel, proiectantul la nivel logic trebuie să elaboreze, atât proiectul unității de executie, cât şi proiectul unității de comanda.

Specificațiile de tip “black box”, pentru proiectarea unui procesor aritmetic, se obțin prin transformarea specificatiilor date de către utilizator, astfel încât, ele să corespundă posibilităților de implementare.

În acest context trebuie să se aibe în vedere că:
- datele se reprezintă sub forma unor vectori binari;
- la baza circuitelor, care efectuează operațiile aritmetice, se afla circuite logice, ce opereaza cu semnale binare.

Având în vedere cele de mai sus:
- intrările X şi Y vor deveni:

  $X \equiv x_{n-1}\, x_{n-2} \dots x_1\, x_0,$

$Y \equiv y_{n-1}\, y_{n-2} \dots y_1\, y_0$

operatorul (*) va fi codificat printr-un cod de operație: $\Omega \equiv \omega_{n-1}\, \omega_{n-2} \dots \omega_1\, \omega_0$ care va indica atât operația, cât şi formatul.

- ieşirile reprezintă următorii vectori numerici:

$Z \equiv z_{n-1}\, z_{n-2} \dots z_1\, z_0$, rezultatul;

$C \equiv c_{p-1}\, c_{p-2} \dots c_1\, c_0$, indicatorii de condiții

$S \equiv s_{q-1}\, s_{q-2} \dots s_1\, s_0$, indicatorii de pseudorezultat.

În continuare se vor examina algoritmii operațiilor aritmetice în virgulă fixă şi în virgulă mobilă.

10.2 OPERAȚII ARITMETICE ÎN VIRGULĂ FIXĂ

10.2.1 ADUNAREA ŞI SCĂDEREA

Operațiile de adunare şi scădere ale numerelor în virgulă fixă se implementează, în majoritatea covârşitoare a cazurilor, cu numere reprezentate în complementul față de doi. Astfel, operațiile de adunare şi scădere se reduc la operația de adunare a codurilor complementare ale celor doi operanzi. Adunarea se efectuează rang cu rang, începând cu rangurile mai puțin semnificative, inclusiv rangurile de semn. Transportul, care apare la stânga rangului de semn, se neglijează.

Fie operanzii:

$x = \pm x_{n-2} \dots x_1\, x_0$

$y = \pm y_{n-2} \dots y_1\, y_0$

în condițiile :

$-2^{n-1} \le x \le 2^{n-1} - 1$

$-2^{n-1} \le y \le 2^{n-1} - 1$

La adunarea/scăderea celor doi operanzi, de mai sus, apar următoarele situații:

a) $x > 0, y > 0, dar\ x + y \le 2^{n-1} - 1$

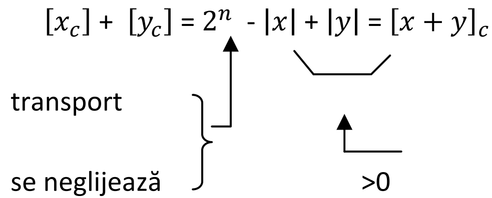
$[x]_c + [y]_c = |x| + |y| = [x + y]_c$

Exemplul 1:

$[0011]_c + [0010]_c = |0011| + |0010| = |0101| = [0101]_c$

b) $x < 0$, $y > 0$, dar $0 < x + y \le 2^{n-1} - 1$

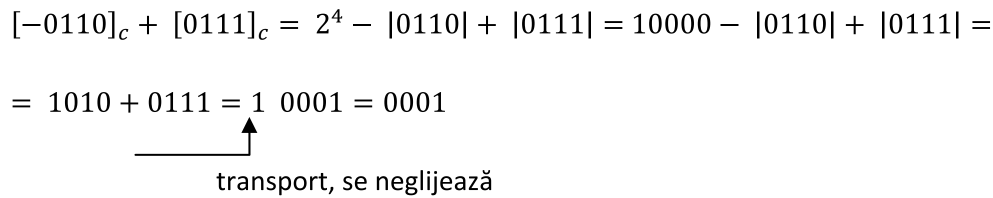
$[x_c] + [y_c] = 2^n - |x| + |y| = [x + y]_c$



Exemplul 2:

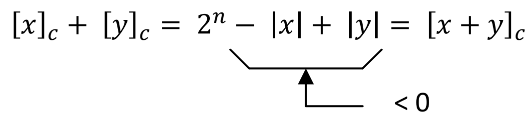
$[-0110]_c + [0111]_c = 2^4 - |0110| + |0111| = 10000 - |0110| + |0111| =$

$= 1010 + 0111 = 1\ 0001 = 0001$



c) $x < 0, y > 0, dar\ x + y < 0$

$[x]_c + [y]_c = 2^n - |x| + |y| = [x + y]_c$

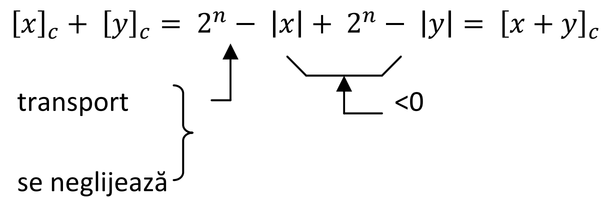


Exemplul 3:

$[-0111]_c + [0110]_c = 2^4 - |0111| + |0110| = 10000 - |0111| + |0110| = 1001 + 0110$
$= [1111]_c$

d) $x < 0, y < 0, dar\ |x| + |y| \le 2^{n-1} - 1$

$[x]_c + [y]_c = 2^n - |x| + 2^n - |y| = [x + y]_c$



Exemplul 4:

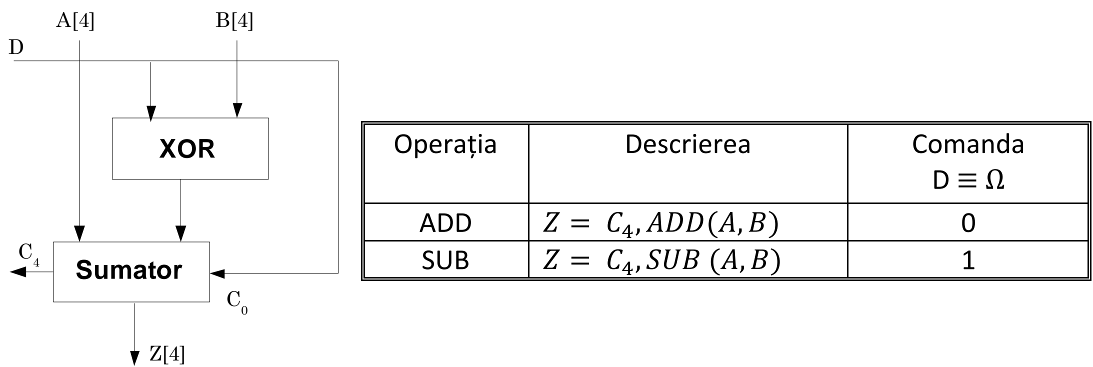
$[-0011]_c + [-0010]_c = 2^4 - |0011| + 2^4 - |0010| = 10000 - |0011| + 10000 -$

$- |0010| = 1101 + 1110 = 1 \quad 1011 = [1011]_c$

transport, se neglijează

Schema unui sumator/scăzător

Schema se bazează pe utilizarea a două circuite: XOR şi ADD aşa cum este prezentat în figura 10.2.



Operația	Descrierea	Comanda D ≡ Ω
ADD	$Z = C_4, ADD(A,B)$	0
SUB	$Z = C_4, SUB\ (A,B)$	1

Figura 10.2. Un sumator/scăzător

Codul de operație specifică operatorul: $D \equiv \Omega$ care ia valorile “0” pentru “+” şi “1” pentru ”-“

Poziționarea indicatorilor de condiții

Indicatorii de condiții specifică o serie de proprietăți ale rezultatului, care apare în registrul acumulator al rezultatului AC. De regulă, ei sunt stocați în bistabile notate cu mnemonice, care formează un registru, încorporat în cuvântul de stare al programului/procesului. Indicatorii de condiții specifică diverse situații:

- rezultat = 0 - mnemonica Z, ($Z \leftarrow \overline{\cup/AC}$);
- semnul rezultatului > 0 sau < 0 - mnemonica S, ($S \leftarrow AC_{n-1}$);
- apariția transportului, la stânga rangului de semn, mnemonica C,($C_n \leftarrow 1$);
- rezultatul verificării parității - mnemonica P, ($P \leftarrow \overline{\oplus/AC}$).

Poziționarea indicatorilor de pseudorezultat

Printre situațiile care conduc la un pseudorezultat, în cazul operării în virgulă fixă, este şi aceea când apare o depăşire.

Depăşirea se manifestă în condițiile în care cei doi operanzi care se adună au acelaşi semn. Dacă rezultatul obținut are un semn diferit de semnul comun, al celor doi operanzi, s-a înregistrat o depăşire.

Depăşirea poate constitui o cauză de întrerupere/suspendare a programului în cadrul căreia a apărut. Această situație este semnalizată sistemului de operare, în vederea luării unei decizii corespunzătoare.

Situația de depăşire se semnalează prin generarea unui semnal D egal cu *suma modulo doi* între transportul în rangul de semn şi transportul în afara rangului de semn, în cadrul registrului acumulator, al rezultataului: $D \leftarrow C_n \oplus C_{n-1}$

Implementări

Operația de adunare pe un singur rang se realizează prin generarea sumei şi a transportului, folosind circuitele logice necesare realizării fizice a expresiilor logice de mai jos:

$$C_{out_i} = \left(x_i \,.\, C_{in_1}\right) \cup \left(y_i \,.\, C_{in_i}\right) \cup \left(x_i \,.\, y_i\right)$$

$$sum_i = \left(x_i \,.\, \overline{y_i} \,.\, \overline{C_{in_i}}\right) \cup \left(\overline{x_i} \,.\, y_i \,.\, \overline{C_{in_i}}\right) \cup \left(\overline{x_i} \,.\, \overline{y_i} \,.\, C_{in_i}\right) \cup \left(x_i \,.\, y_i \,.\, C_{in_i}\right),$$

Aceasta se concretizează într-un circuit de tip “schema bloc”, denumit SUM, prezentat în continuare:

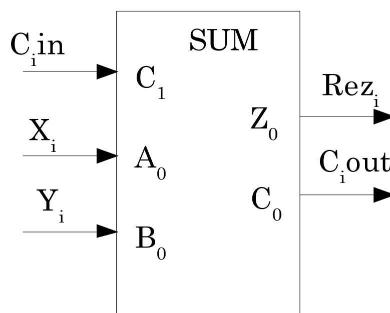


Figura 10.3. Modul sumator

Descrierea în Verilog a unui sumator cu 3 intrări şi două ieşiri este următoarea:

```
module fulladd(sum,carry,a,b,c);
        input a,b,c;
        output sum,carry;
        wire sum1;
        xor xor1(sum1,a,b);
        xor xor2(sum,sum1,c);
        and and1(c1,a,b);
        and and2(c2,b,c);
        and and3(c3,a,c);
        or or1(carry,c1,c2,c3);
endmodule
```

Compilarea acestei descreri la nivel de măşti conduce la rezultatul prezentat în figura 10.4:

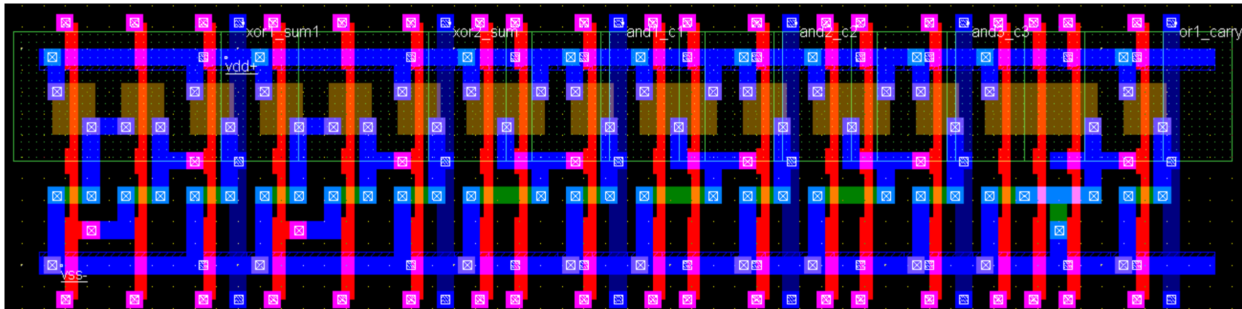


Figura 10.4. Descrierea la nivel de măşti

Cu ajutorul sumatorului la nivel de bit se poate realiza un sumator pe *n* biți aşa cum este prezentat în figura 10.5.

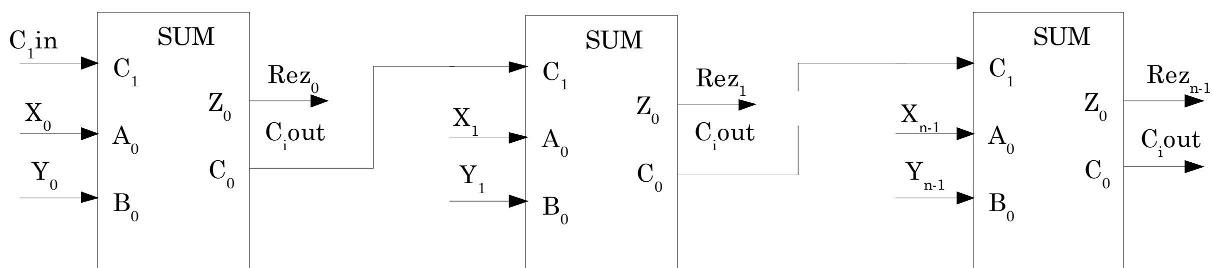


Figura 10.5. Sumator pe n biți

Sumatorul la nivel de bit poate fi extins şi cu operațiile logice SI/AND, SAU/OR, ca în figura 10.6, în cadrul căreia ieşirile sumatorului, circuitului AND şi circuitului OR sunt aplicate la intrările unui multiplexor MUX. Codul de operație, pentru selectarea operatorului, se forțează la intrarea *Sel*, a multiplexorului.

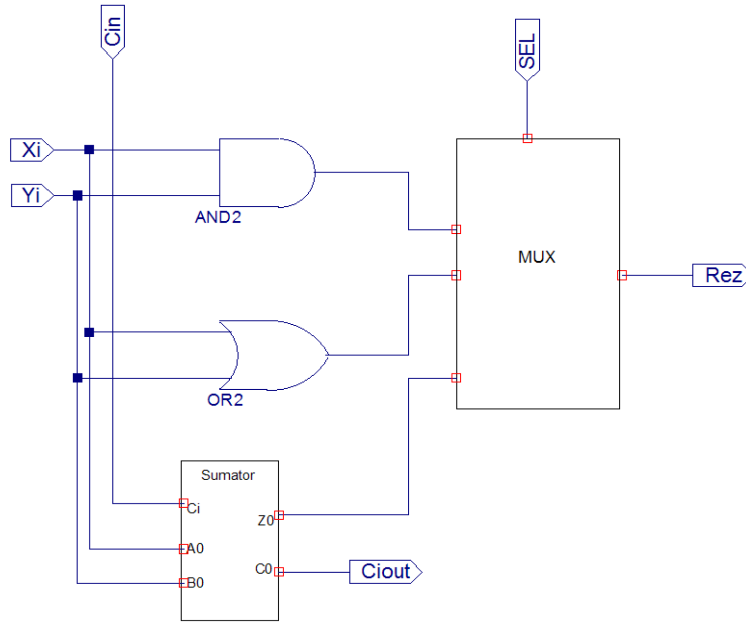


Figura 10.6. Sumatorul extins, la nivel de bit.

În condițiile în care se doreşte şi implementarea operației de scădere, este necesar să se creeze posibilitatea inversării intrării $y_{i,}$ după cum se prezinta în continuare, în figura 10.7.

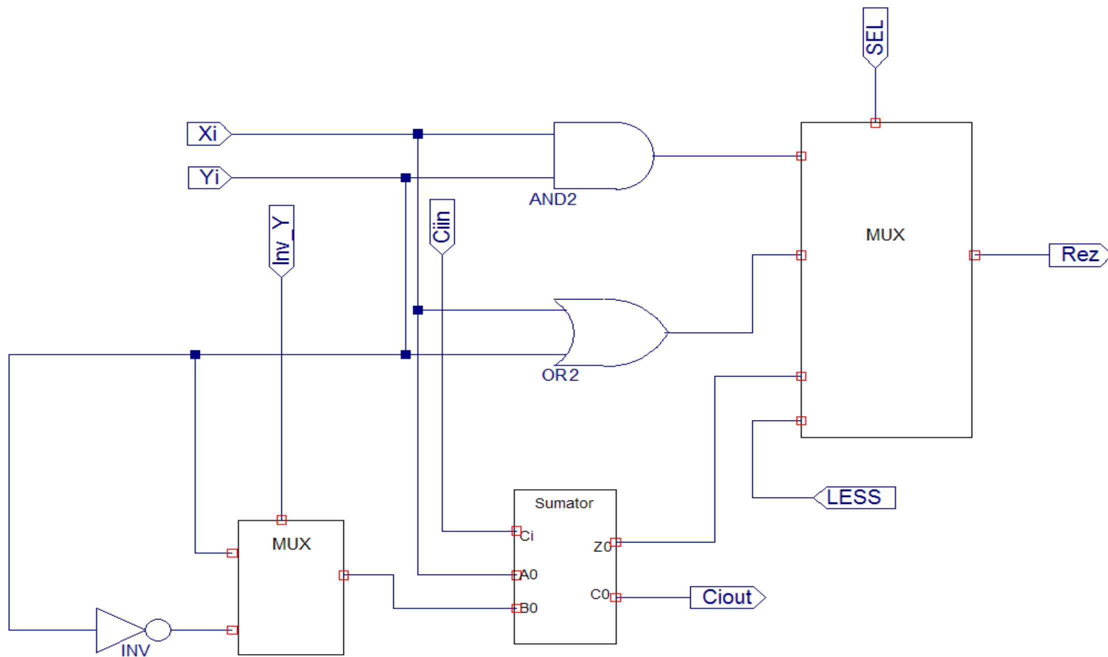


Figura 10.7. Implementare completă, cu adăugarea operației de scădere.

Inversarea lui *y* se realizează sub controlul semnalului de selecție *InvY*, aplicat la intrarea de selecție a celui de-al doilea multiplexor

Operațiile ADD, SUB, AND şi OR se regăsesc în Unitățile Aritmetice Logice ale oricărui procesor. Există unele procesoare care au implementată instrucțiunea “*set-on-less-than*”, ce se traduce prin “*forțează în* (1), *bitul cel mai puțin semnificativ al rezultatului, dacă operandul x este mai mic decât operandul y*”, în condițiile în care toți ceilalți biți ai rezultatului vor fi 0. Astfel, în figura de mai sus a apărut o nouă intrare la multiplexor “less”.

Structura poate fi completată cu detalierea schemei UAL pentru bitul cel mai semnificativ, în care se pune în evidență depăşirea aritmetică.

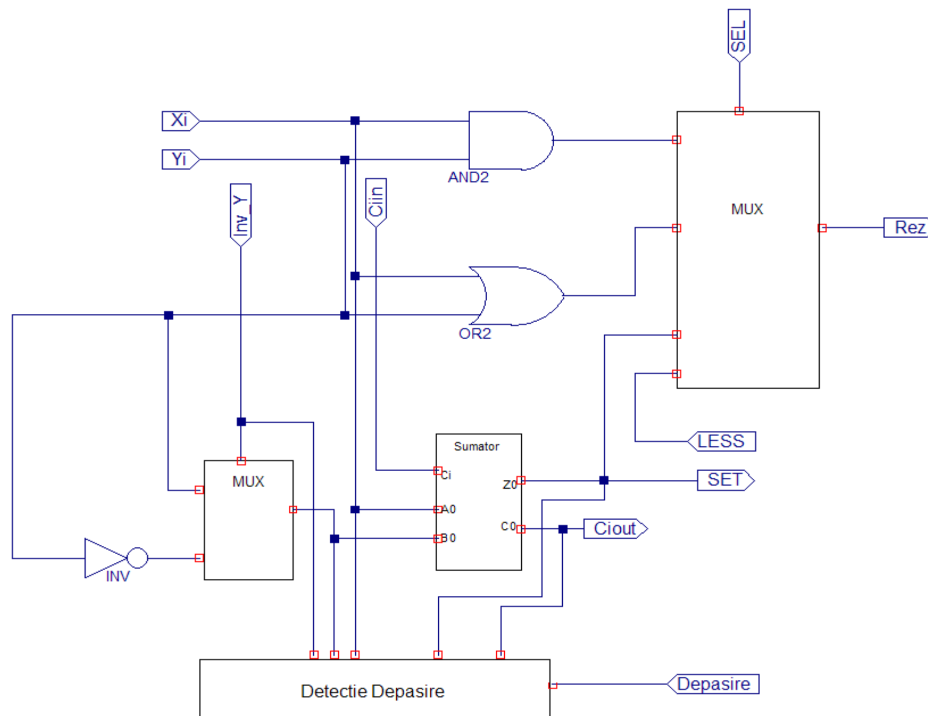


Figura 10.8. Detalierea unei scheme UAL cu evidențierea depăşirii aritmetice

Implementarea operației “*set-on-less-than”* presupune efectuarea scăderii lui *y* din *x*.

Dacă *x* < *y*, se va poziționa în 1 semnul rezultatului, adică Sum_{n-1}. Acest lucru trebuie să se reflecte însă în forțarea în 1 a bitului cel mai puțin semnificativ al rezultatului Rez_0 şi în zero a biților Rez_{n-1} … Rez_1. Această soluție este ilustrată în schema bloc din figura 10.9.

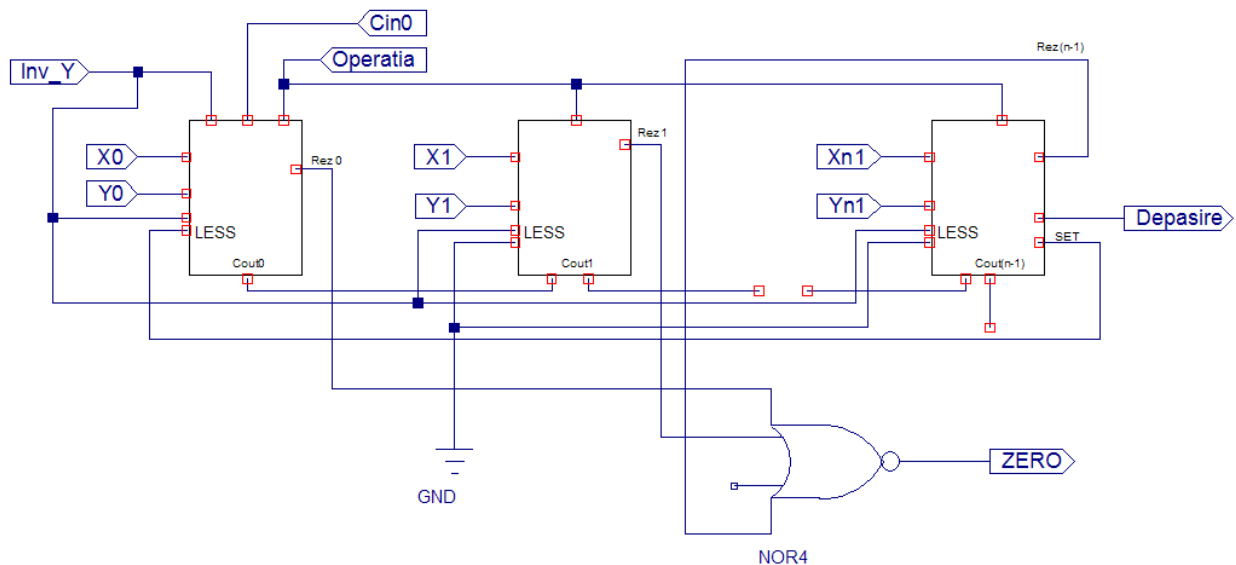


Figura 10.9. Implementarea operației “*set-on-less-than*”.

În figura 10.9 fost implementată şi poziționarea în “1” , a indicatorului de condiții, care specifică apariția unui rezultat egal cu “0”, la ieşirea unității arimetice logice. Bistabilul *Z* se poziționează în “1”.

10.2.2 SUMATOARE PERFORMANTE

Sumatorul cu transport succesiv

Tipul cel mai simplu de sumator paralel este sumatorul cu transport succesiv, obținut prin interconectarea unor sumatoare complete. Ecuațiile pentru transport şi sumă, la nivelul unui etaj, sunt date mai jos:

$$C_{out_i} = (x_i \,.\, C_{in_i}) \cup (y_i \,.\, C_{in_i}) \cup (x_i \,.\, y_i)$$

$$Sum_i = (x_i \,.\, \overline{y_i} \,.\, \overline{C_{in_i}}) \cup (\overline{x_i} \,.\, y_i \,.\, \overline{C_{in_i}}) \cup (\overline{x_i} \,.\, \overline{y_i} \,.\, C_{in_i}) \cup (x_i \,.\, y_i \,.\, C_{in_i})$$

Se poate observa că cele două funcții combinaționale se implementează pe două niveluri logice. În condițiile în care întarzierea pe un nivel logic este Δt, rezultă că întarzierea minimă pe rang va fi $2\Delta t$. Întrucât, în cazul cel mai defavorabil, transportul se propagă de la rangul cel mai puțin semnificativ până la ieşirea celui mai semnificativ rang, rezultă o întârziere egală cu $2n\Delta t$, unde *n* este numărul de ranguri. Este evident că o asemenea soluție nu este acceptabilă.

Sumatorul cu întârziere minimă

Considerând *ecuația sumei*, Sum_0, de la ieşirea celui mai puțin semnificativ rang, se încearcă stabilirea unei expresii a acesteia ca funcție de intrările pentru rangul curent $(n-1)$, cât şi de intrările pentru rangurile mai puțin semnificative $(n-2), \dots, 1, 0$. În acest mod se va obține o expresie logică implementabilă în două trepte.

$$Sum_0 = \left[x_0 \,.\, \overline{y_0} \,.\, \overline{C_{in_0}}\right] \cup \left[\overline{x_0} \,.\, y_0 \,.\, \overline{C_{in_0}}\right] \cup \left[\overline{x_0} \,.\, \overline{y_0} \,.\, C_{in_0}\right] \cup \left[x_0 \,.\, y_0 \,.\, C_{in_0}\right]$$

Pentru ieşirea, care reprezintă transportul, din rangul 0 s-a înlocuit C_{out_0} cu C_{in_1}, pentru a simplifica relațiile, care urmează a se obține.

$$C_{in_1} = \left[x_0 \,.\, C_{in_0}\right] \cup \left[y_0 \,.\, C_{in_0}\right] \cup \left[x_0 \,.\, y_0\right]$$

Pentru rangul următor 1, se obține următoarea expresie a sumei:

$$Sum_1 = \left[x_1 \,.\, \overline{y_1} \,.\, C_{in_1}\right] \cup \left[\overline{x_1} \,.\, y_1 \,.\, \overline{C_{in_1}}\right] \cup \left[\overline{x_1} \,.\, \overline{y_1} \,.\, C_{in_1}\right] \cup \left[x_1 \,.\, y_1 \,.\, C_{in_1}\right]$$

În expresia lui Sum_1 se va înlocui C_{in_1} cu componentele care-l alcătuiesc, conform formulei precedente. Calculul lui $\overline{C_{in_1}}$ conduce la următoarea expresie:

$$\overline{C_{in_1}} = \left[\overline{x_0} \,.\, \overline{C_{in_0}}\right] \cup \left[\overline{y_0} \,.\, \overline{C_{in_0}}\right] \cup \left[\overline{x_0} \,.\, \overline{y_0}\right]$$

Astfel, pentru Sum_1 se va obține o expresie formată prin adunarea logică a 12 produse logice de câte 4 variabile fiecare. Aceasta presupune utilizarea unei porți OR cu 12 intrări şi a 12 porți AND, cu câte 4 intrări. Extinzând procedeul la următoarele ranguri se vor obține expresii cu numeroşi termeni de tip produs, care, la rândul lor, vor avea multe componente. Un calcul simplu arată că, în cazul unui sumator pe 64 de biți, vor fi necesare circa 10^{12} porți, practic de nerealizat în prezent.

Principiul anticipării transportului

Întrucât sumatorul cu transport succesiv este lent, iar sumatorul cu întârziere minimă este imposibil de realizat, se caută o soluție intermediară. Aceasta grupează relațiile obținute la sumatorul cu întârziere minimă, astfel încât să se obțină dimensiuni rezonabile.

Ideea de bază este aceea de a caracteriza comportarea unui rang al sumatorului din punctul de vedere al *generării*/*propagării* unui transport. Astfel, rangul *j* al unui sumator generează, G_j, transport dacă are loc

relația: $G_j = x_j \, . \, y_j$. De asemenea, rangul *j* al unui sumator propagă, P_j, transport în situația următoare: $P_j = x_j \oplus y_j = (\overline{x_j} \, . \, y_j) \cup (x_j \, . \, \overline{y_j})$.

În aceste condiții se pot elabora noi expresii pentru transportul C_{j+1} şi pentru suma Sum_j, la nivelul fiecărui rang al sumatorului:

$$C_{j+1} = G_j \cup (P_j \, . \, C_j)$$
$$Sum_j = (P_j \, . \, \overline{C_j}) \cup (\overline{P_j} \, . \, C_j)$$

Astfel, un sumator complet va constă în două părți: partea *P/G* şi partea *Sum*

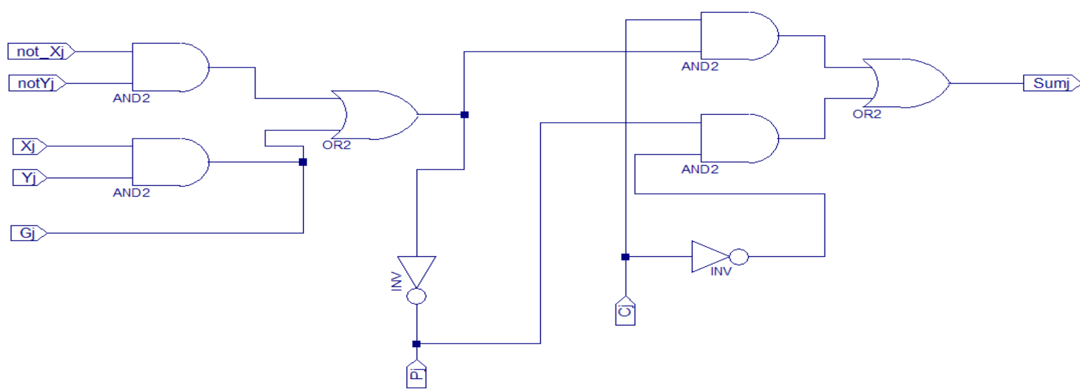


Figura 10.10. Sumatorul cu anticipare a transpportului

Relațiile de mai sus se pot structura la nivelul a 4 secțiuni ale unui sumator, bazat pe ideea menționată anterior. Începând cu rangul cel mai puțin semnificativ se obțin:

- pentru sume:

$$Sum_0 = (\overline{P_0} \, . \, C_0) \cup (P_0 \, . \, \overline{C_0})$$
$$Sum_j = (\overline{P_1} \, . \, C_1) \cup (P_1 \, . \, \overline{C_1})$$
$$Sum_j = (\overline{P_2} \, . \, C_2) \cup (P_2 \, . \, \overline{C_2})$$
$$Sum_j = (\overline{P_3} \, . \, C_3) \cup (P_3 \, . \, \overline{C_3})$$

şi:

- pentru transporturi:

$$C_1 = G_0 \cup (P_0 \, . \, C_0)$$
$$C_2 = G_1 \cup (P_1 \, . \, C_1) = G_1 \cup (P_1 \, . \, G_0) \cup (P_1 \, . \, P_0 \, . \, C_0)$$
$$C_3 = G_2 \cup (P_2 \, . \, C_2) = G_2 \cup (P_2 \, . \, G_1) \cup (P_2 \, . \, P_1 \, . \, G_0) \cup (P_2 \, . \, P_1 \, . \, P_0 \, . \, C_0)$$
$$C_4 = G_3 \cup (P_3 \, . \, C_3) = G_3 \cup (P_3 \, . \, G_2) \cup (P_3 \, . \, P_2 \, . \, G_1) \cup (P_3 \, . \, P_2 \, . \, P_1 \, . \, G_0) \cup (P_3 \, . \, P_2 \, . \, P_1 \, . \, P_0 \, . \, C_0)$$

Aceste relații sunt implementate în unitatea de anticipare a transportului *UAT*.

Pentru o secțiune de 4 biți, sumatorul cu transport anticipat este prezentat în figura 10.11.

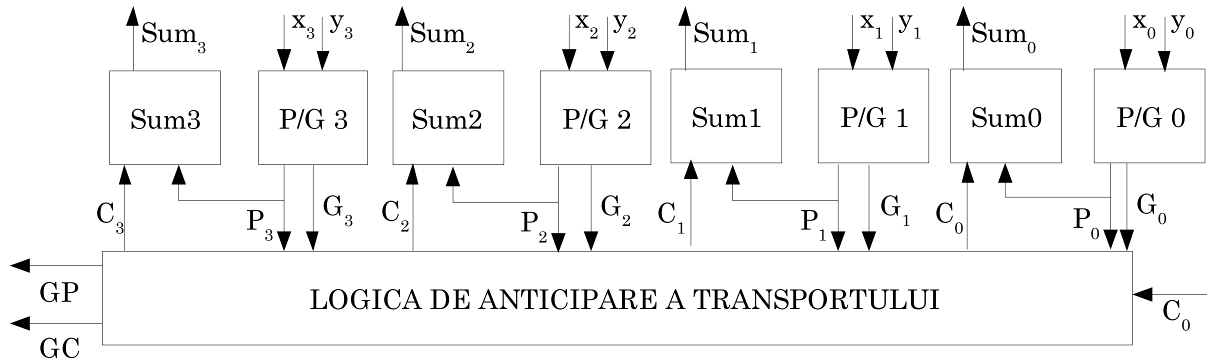


Figura 10.11. Secțiune pe 4 biți a unui sumator cu anticipare a transportului

Efectuând un calcul al întârzierii, se obține pe 4 biți de sumator o întârziere de $6\Delta t$, în comparație cu întarzierea de $8\Delta t$, pentru sumatorul cu transport succesiv. Din schema bloc, de mai sus, rezultă că *Logica de Anticipare a Transportului* mai generează două semnale la nivel de grup. Grupul constă într-un ansamblu de patru ranguri de sumator. Astfel, grupul poate genera transport (*GG* =1) sau poate propaga transport (*GP* = 1).

Ideea anticipării transportului se poate fi extinsă atât la nivel de grup, cât şi la nivel de secțiune (ansamblu de patru grupuri), realizând o structură piramidală de *UAT*-uri. Unitățile de anticipare la nivel de grup şi la nivel de secțiune sunt identice cu logica de anticipare a transportului la nivelul sumatorului cu patru biți.

Analiza performanțelor arată că în cazul unui sumator pe 16 biți, cu transport de grup, se înregistrează o întârziere de $8\Delta t$, față de întârzierea de $32\Delta t$, pentru sumatorul cu transport succesiv. În cazul unui sumator pe 64 de biți, în situația anticipării transportului pe secțiuni, se obține o întârziere maximă de $14\Delta t$, comparativ cu întarzierea de $128\Delta t$, pentru sumatorul cu transport succesiv.

Se poate aprecia că numărul de terminale de intrare şi iesire ale porților poate reprezenta un criteriu de cost, într-o implementare dată. Comparația cost/performanță pentru cele două implementări, de mai sus, arată că în cazul unui sumator pe 64 de biți, la o creştere a numărului de terminale cu 50%, pentru soluția cu transport anticipat, se obține o viteză de 9 ori mai mare, decât în cazul transportului succesiv.

Sumatorul cu salvare a transportului

În cazul adunării mai multor vectori binari simultan se poate recurge la o schemă mai rapida, constând în utilizarea mai multor sumatoare în cascadă.

Fie cazul adunării a patru numere de câte 4 biți, notate cu *a*, *b*, *e*, *f*. Rangurile numerelor *b*, *e*, *f* se vor aduna într-un *sumator cu salvare a transportului*, care va avea ieşiri pentru sumă şi transport. Acestea, la rândul lor, se vor aduna cu rangurile numărului *a*, într-un alt *sumator cu salvare a transportului*. Ieşirile reprezentând rangurile sumei şi rangurile transportului se vor aduna într-un sumator obişnuit.

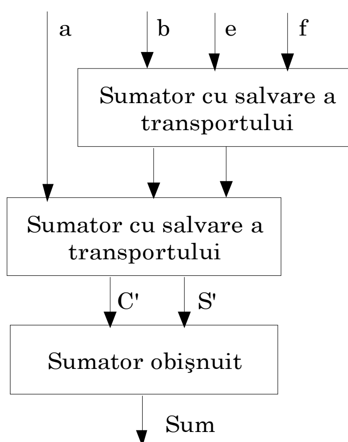


Figura 10.12. Utilizarea mai multor sumatoare în cascadă.

Detaliile de implementare sunt lăsate pe seama cititorului.

Comparație între câteva tipuri de sumatoare, pe *n* biți, în privința întârzierii şi a ariei ocupate:

Tip sumator	**Întârziere**	**Aria ocupata**
Transport succesiv	$2n\Delta t$	$9n$
Întârziere minimă	$2\Delta t$	$2^{(2n+1)}/(2n+1)$
Transport anticipat	$2(\log_2 n + 1).\Delta t$	$7n + 22\log_2 n$

10.2.3 ÎNMULȚIREA

Înmulțirea numerelor constă în generarea şi adunarea produselor partiale obținute prin înmulțirea rangului curent al înmultitorului cu deînmulțitul. În cazul numerelor binare, produsul parțial curent este egal cu deînmulțitul, deplasat spre stânga conform poziției rangului înmulțitorului, dacă bitul curent al înmulțitorului este unu, sau egal cu zero, dacă bitul curent al înmulțitorului este zero. Dacă produsul parțial curent este diferit de zero, el se adună la suma produselor parțiale anterioare.

Există şi posibilitatea ca, după fiecare adunare, suma produselor parțiale să fie deplasată spre dreapta cu un rang.

<u>Înmulțirea în cod direct</u>

Înmulțirea în cod direct/semn şi modul presupune tratarea separată a semnelor şi înmulțirea modulelor. Fie numerele X şi Y,

$X \equiv x_{n-1}\, x_{n-2} \dots x_1\, x_0$

$Y \equiv y_{n-1}\, y_{n-2} \dots y_1\, y_0$

care urmează să fie înmulțite, pentru a furniza produsul Z:

$Z \equiv z_{2n-2}\, z_{2n-3} \dots z_1\, z_0$

Semnul produsului z_{2n-1} va fi obținut astfel:

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

Produsele parțiale: $P_0, \dots, P_{n-3}, P_{n-2}$ se obțin după cum urmează:

$P_0 = |x| \,.\, y_0 \,.\, 2^0$

$P_1 = |x| \,.\, y_1 \,.\, 2^1$

…………………

$P_{n-2} = |x| \,.\, y_{n-2} \,.\, 2^{n-2}$

iar suma lor va conduce la produsul modulelor:

$z_{2n-3} \dots z_1\, z_0$

Fie cazul a două numere reprezentate în cod direct pe 5 biți:

- deînmulțitul $x = 11000$ şi
- înmulțitorul $y = 01000$

Pentru semn rezultă: $z_8 = 1 \oplus 0 = 1$,

în timp ce modulul produsului se va obține după cum se arată mai jos:

```
       1000
   × 1001
  ____________
       1000
     0000
    0000
   1000
  __________
  01001000
```

$z_7 \ldots z_1 z_0 = 01001000$

Astfel, produsul celor două numere va avea 9 biți:

$z_8 z_7 \ldots z_1 z_0 = 101001000.$

În exemplul de mai sus produsele parțiale s-au obținut prin deplasarea spre stânga a deînmulțitului, înmulțit cu rangul corespunzator al înmulțitorului.

Soluție paralelă.

O implementare combinațională presupune utilizarea sumatoarelor complete şi a unor porți AND. Astfel, se obține un sumator modificat, pentru un rang, conform schemei din figura 10.13.

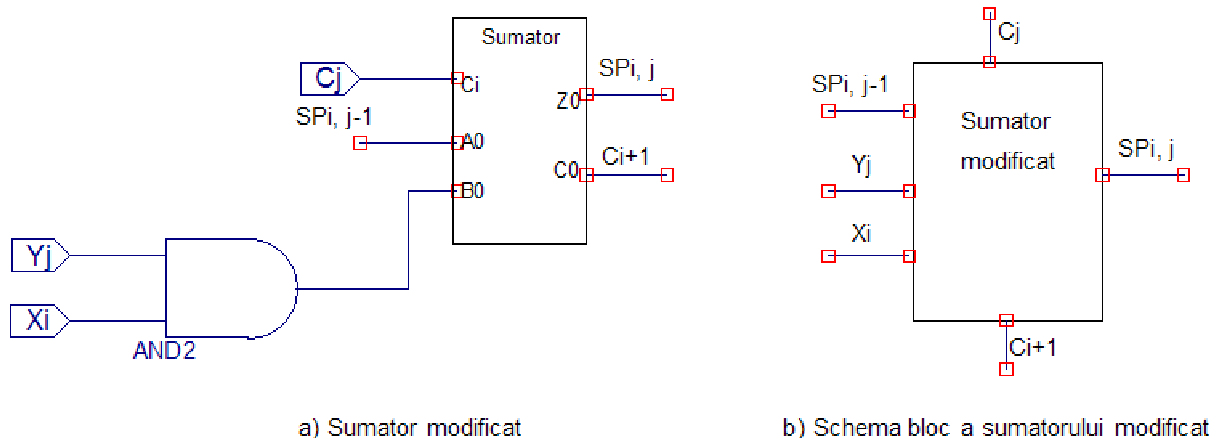


Figura 10.13. Un sumator modificat pentru un rang.

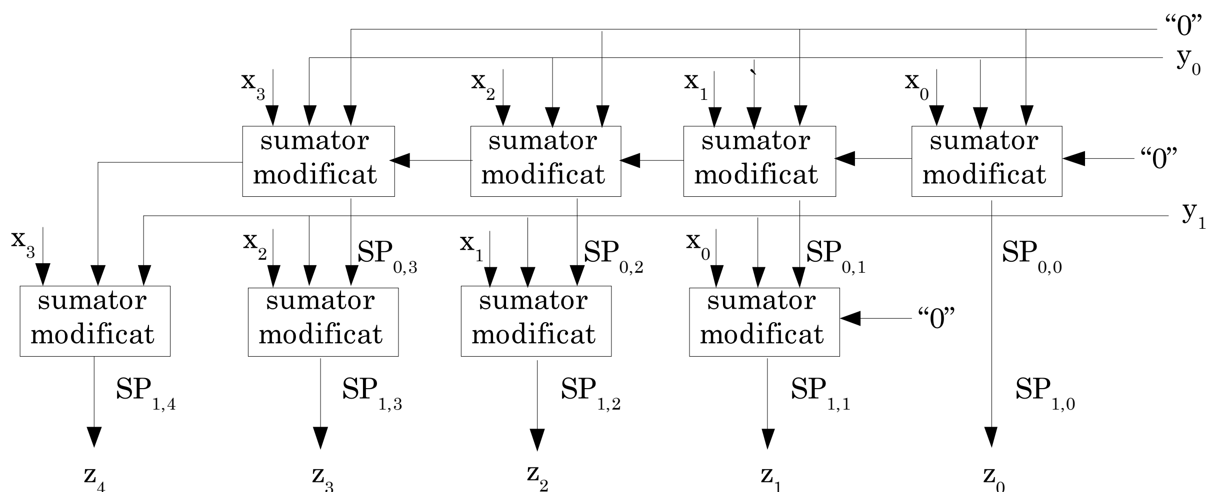


Figura 10.14. Primele două etaje ale dispozitivului paralel de înmulțire.

S-a notat cu SP_{ij} bitul *i* al sumei produselor parțiale *j.* Schema din figura 10.14 prezintă numai primele două etaje ale dispozitivului paralel de înmulțire.

Sumatorul modificat poate fi utilizat în cadrul unei structuri de înmulțire paralelă, ca în figura de mai sus. Din punctul de vedere al implementării algoritmului, se poate afirma că, în acest caz, este vorba de o “programare spațială”, care conduce la viteza ridicată de operare. Soluția necesită, pentru

implementare, $(n-1)^2$ sumatoare modificate. În cel mai defavorabil caz, rezultatul înmulțirii se obține după propagarea semnalelor prin $4(n-1)$ porți.

Se lasă pe seama cititorului elaborarea unei soluții bazate pe "sumatoare cu salvare a transportului" şi sumatoare cu transport anticipat.

Soluție serial - paralelă

În scopul reducerii cantității de hardware, dispozitivele de înmulțire se realizează sub forma unei structuri paralele hardware, pentru un pas de înmulțire, cu operare secvențială.

Dacă se notează *produsul parțial j* cu pp_j, atunci un pas de înmulțire va avea următoarea formă:

$$pp_j = pp_{j-1} + x \,.\, y_j \,.\, 2^j$$

O schemă bloc, pentru soluția menționată mai sus, este prezentată în continuare în figura 10.15:

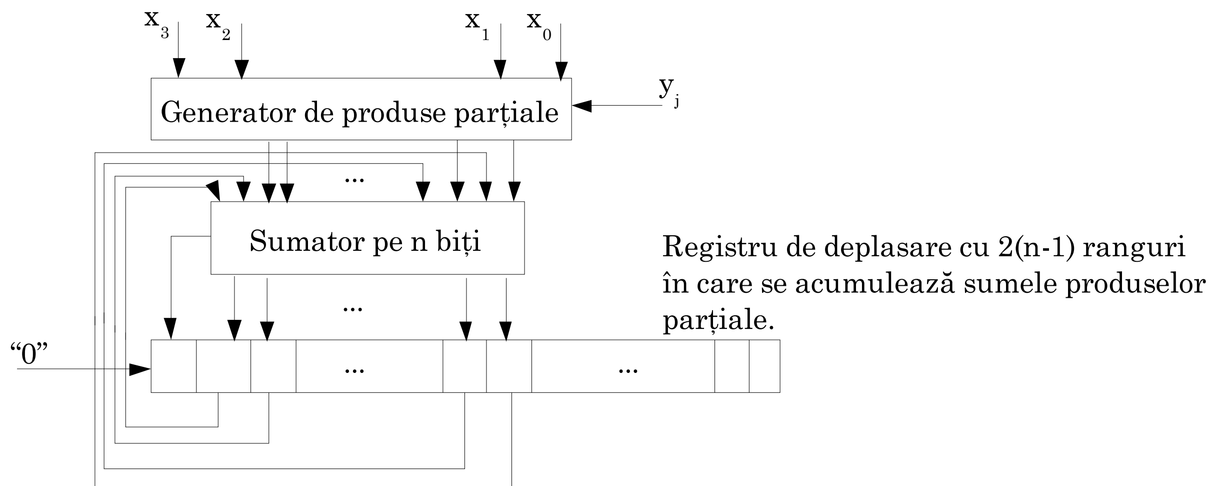


Figura 10.15. Soluție serial -paralelă pentru un dispozitiv de înmulțire.

Operarea se efectuează conform urmatoarei secvențe:

1. anulează conținutul registrului în care se acumulează sumele produselor parțiale;
2. inițializează numărul rangului bitului înmulțitorului *j* = 0;
3. formează produsul parțial $x \,.\, y_j$.
4. adună produsul parțial la jumătatea superioară a registrului sumei produselor parțiale;
5. efectuează *j* = *j* + 1 şi dacă *j* = *n* treci la 8.
6. deplasează la dreapta cu un rang conținutul registrului sumei produselor parțiale;

7. treci la pasul 3;
8. produsul cu $2(n-1)$ ranguri s-a obținut în registrul de deplasare.

Rezultatul se poate stoca fie sub forma unui singur cuvânt, păstrând jumătatea superioară, fie sub forma unui cuvânt dublu. În primul caz va fi afectată precizia.

<u>Înmulțirea numerelor în cod complementar</u>

Întrucât, în marea majoritate a cazurilor, numerele negative se reprezintă în calculatoare în codul complementar, în continuare vor fi examinate câteva metode de înmulțire în acest cod.

Mai întâi se vor examina *deplasările aritmetice la stânga şi la dreapta în cod complementar.*

Se consideră următoarele cazuri:

$x > 0$.

$[x]_c = 0\ x_{n-2} \dots\ x_1\ x_0,$

Deplasarea la stânga:

$[2x]_c = \ x_{n-2} \dots\ x_1\ x_0\ 0,$

Deplasarea la dreapta:

$[2^{-1}.x]_c = 00x_{n-2} \dots\ x_1,$

$x < 0$.

$[x]_c = 1\ \widetilde{x_{n-2}} \dots\ \widetilde{x_1}\ \widetilde{x_0},$

Deplasarea la stânga:

$[2x]_c = \ \widetilde{x_{n-2}} \dots\ \widetilde{x_0}\ 0,$

Deplasarea la dreapta:

$[2^{-1}.x]_c = 11\widetilde{x_{n-2}} \dots\ \ \widetilde{x_1}.$

Câteva metode pentru înmulțirea numerelor reprezentate în cod complementar.

Metoda 1.

- Se modifică , dacă este cazul, înmulțitorul şi deînmulțitul astfel încât înmulțitorul să fie pozitiv.
- Produsele parțiale se calculează în mod obişnuit.
- Deplasarea spre stânga/ dreapta a deînmulțitului/sumei produselor parțiale se realizează conform regulilor de deplasare în cod complementar.

***Metoda* 2.**

Această metodă presupune înmulțirea numerelor cu semn în maniera obişnuită, ca şi când ar fi vorba de numere întregi fără semn. Rezultatul va fi corect numai în cazul în care cei doi operanzi sunt pozitivi. În caz contrar sunt necesare corecții, care se încadrează în trei cazuri.

1. $x > 0$, $y > 0$.

$[x]_c \,.\, [y]_c = |x| \,.\, |y| = [x.y]_c$

2. $x < 0$, $y > 0$.

$[x]_c = 2^n - |x|$; $[y]_c = |y|$;

$[x]_c \,.\, [y]_c = 2^n \,.\, |y| - |x| \,.\, |y|$; *rezultat incorect*

$[x]_c \,.\, [y]_c = 2^{2n} - |x| \,.\, |y|$; *rezultat corect*

Rezultă necesitatea unei corecții egală cu: $2^{2n} - 2^n \,.\, |y|$, care se va aduna la rezultatul incorect.

3. $x > 0$, $y < 0$;

$[x]_c = |x|$; $[y]_c = 2^n - |y|$;

$[x]_c \,.\, [y]_c = 2^n \,.\, |x| - |x| \,.\, |y|$; *rezultat incorect*

$[x]_c \,.\, [y]_c = 2^{2n} - |x| \,.\, |y|$; *rezultat corect*

Rezultă necesitatea unei corecții egală cu: $2^{2n} - 2^n \,.\, |x|$, care se va aduna la rezultatul incorect.

4. $x < 0$, $y < 0$;

$[x]_c = 2^n - |x|$; $[y]_c = 2^n - |y|$;

$[x]_c \,.\, [y]_c = 2^{2n} - 2^n \,.\, |y| - 2^n \,.\, |x| + |x| \,.\, |y|$; *rezultat incorect*

$[x.y]_c = |x| \,.\, |y|$; *rezultat corect*

Rezultă necesitatea unei corecţii egală cu: $-2^{2n} + 2^n \,.\, |y| + 2^n \,.\, |x|$, care se va aduna la rezultatul incorect.

Metoda este greoaie şi are un caracter pur teoretic.

10.2.4 ALGORITMUL LUI BOOTH

În acest caz se pleacă de la observația că, valoarea unui număr *Y*, reprezentat în cod complementar, se poate calcula astfel:

$$Y = y_{n-1} \,.\, 2^{n-1} + y_{n-1} \,.\, 2^{n-2} + y_{n-3} \,.\, 2^{n-3} + \ldots + y_1 \,.\, 2^1 + y_0 \,.\, 2^0 = \sum_{i=0}^{n-1} (y_{i-1} - y_i) . 2^i$$

unde:

- y_{n-1} reprezintă rangul de semn, codificat cu 0/1 în cazul numerelor pozitive/negative;
- y_{-1} constituie rangul aflat la dreapta rangului 0, având inițial valoarea 0.

În aceste condiții valoarea produsului $X \times Y$ se va exprima după cum urmează:

$$X \times Y = \sum_{i=0}^{n-1} x \,.\, (y_{i-1} - y_i) . 2^i$$

Pe baza formulei de mai sus se poate calcula produsul parțial de rang i :

Tabelul 10.1. Produsul parțial

y_{i-1}	y_i	produs parțial
0	0	0
0	1	$-x \,.\, 2^i$
1	0	$x \,.\, 2^i$
1	1	0

Pentru implementarea hardware se consideră resursele corespunzătoare unei structuri orientate pe un singur acumulator:

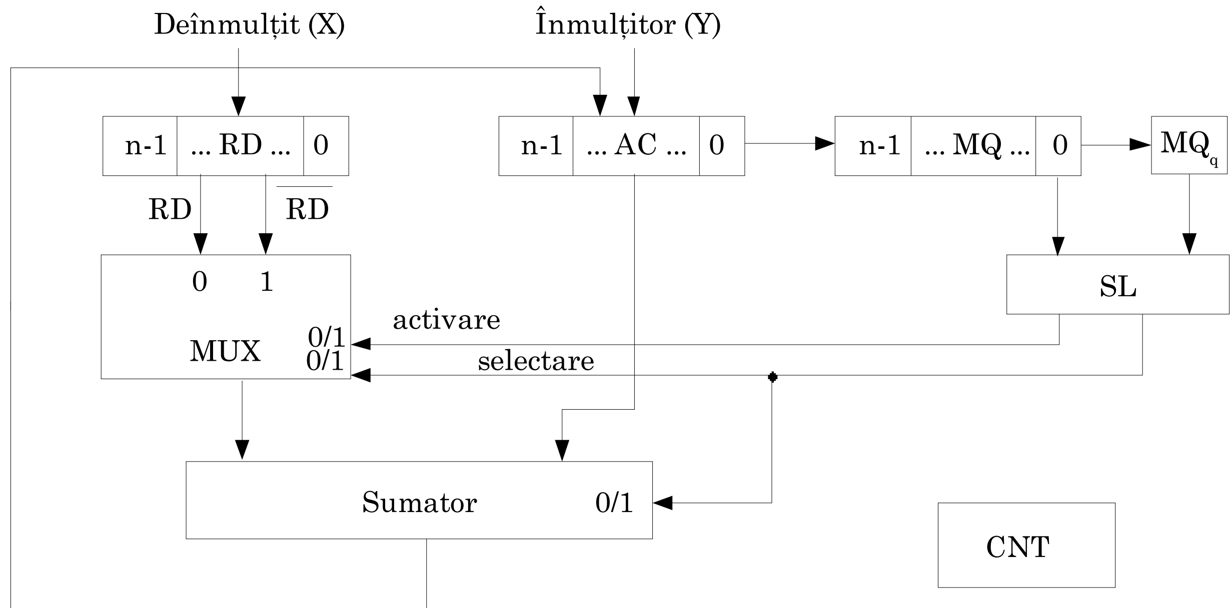


Figura 10.16. Implementarea hardware a algoritmului lui Booth

Resursele hardware:

- AC – registru acumulator;
- RD – registru de date al memoriei;
- MQ – registru de extensie al acumulatorului;
- MQ_q – registru de un bit, extensia lui MQ;
- CNT – contor de cicluri ($\rho\ CNT = \lceil \log_2 n \rceil$);
- SL – structura logică pentru activarea şi selectarea intrărilor multiplexorului, cât şi pentru controlul transportului la sumator;
- Sumator- sumator combinațional cu *n* ranguri binare.

Descrierea operării dispozitivului se poate face cu ajutorul următoarei organigrame din figura 10.17:

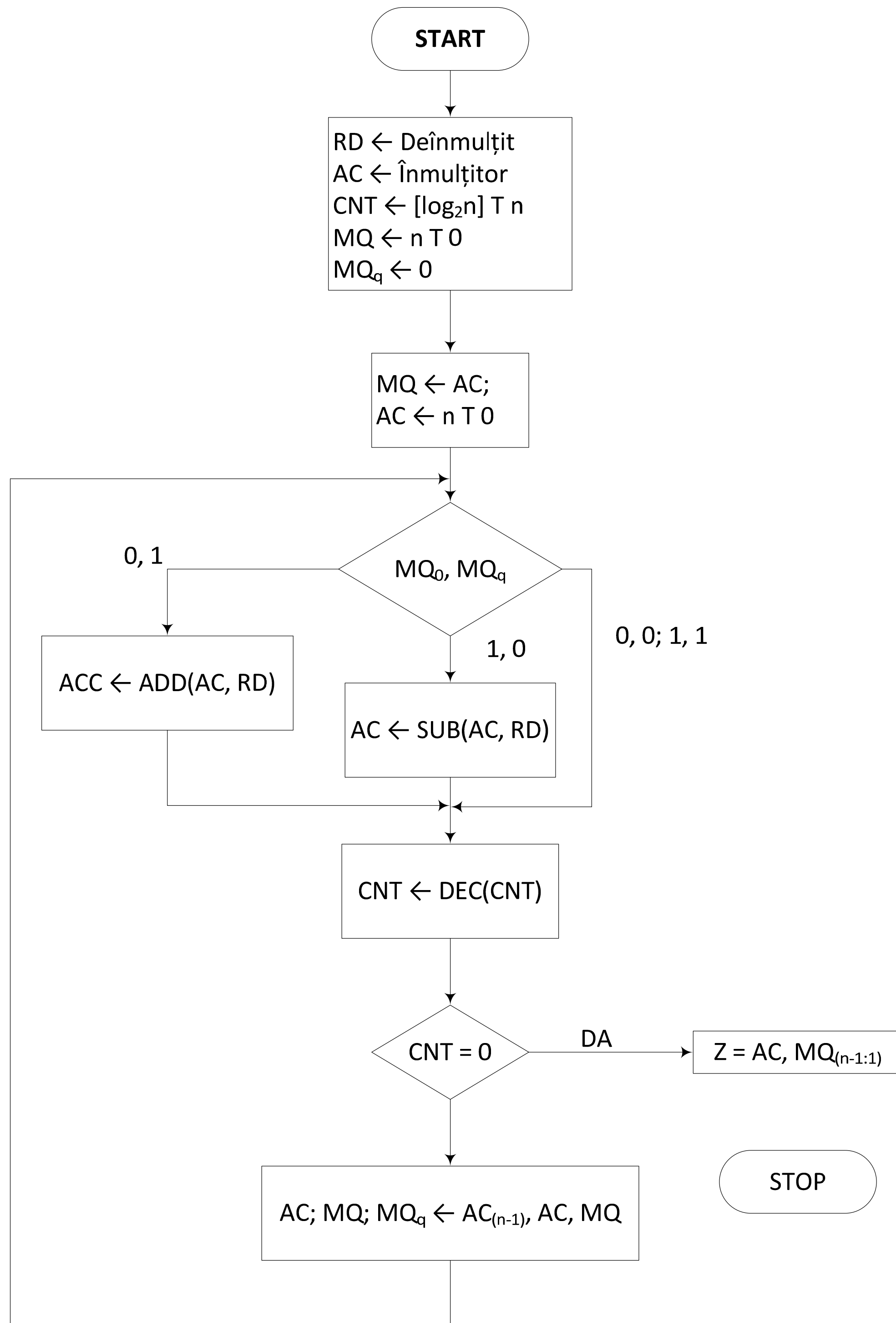


Figura 10.17. Organigrama operării dispozitivului.

În continuare se va prezenta un program Verilog pentru simularea unui dispozitiv de înmulțire.

```
//Simularea unui dispozitiv de înmulţire a numerelor reprezentate în
complementul faţă de doi, //folosind Algoritmul lui Booth
module inmultitorb;
parameter n = 8;
reg [n − 1 : 0] RD, AC, MQ;
reg [2 ∗ n − 1 : 0];
reg [2 : 0] CNT;
reg MQq;
initial begin: init
AC=-7; RD=-5; MQ=0; MQq=0; CNT = n-1;
$display("timp   RD   AC   MQ   MQq   CNT");
$monitor("%0d %b %b %b %b %b",$time, RD, AC, MQ, MQq, CNT);
wait (CNT==0) begin
   $monitor("Produs= %b",{AC[n-1], AC, MQ[n-1 : 1]});
   #1 $stop;
   end
end
always begin
   #1; MQ = AC;
   #1; AC = 0;
while (CNT > 0)
   begin
   case({MQ[0], MQq})
2'b00: begin
      #1;{AC, MQ, MQq} = {AC[n-1], AC, MQ};
      end
2'b01: begin
       #1; AC = AC + RD;
       #1;{AC, MQ, MQq} = {AC[n-1], AC, MQ};
      end
2'b10: begin
       #1; AC = AC - RD;
       #1; {AC, MQ, MQq} = {AC[n-1], AC, MQ};
      end
2'b11: begin
       #1; {AC, MQ, MQq} = {AC[n-1], AC, MQ};
      end
  endcase
```

```
   #1; CNT = CNT - 1;
   end
end
endmodule
```

//Cazul: RD = 5; AC = 7;

Timp	RD	AC	MQ	MQq	CNT
0	000101	00000111	00000000	0	111
1	00000101	00000111	00000111	0	111
2	00000101	00000000	00000111	0	111
3	00000101	11111011	00000111	0	111
4	00000101	11111101	10000011	1	111
5	00000101	11111101	10000011	1	110
6	00000101	11111110	11000001	1	110
7	00000101	11111110	11000001	1	101
8	00000101	11111111	01100000	1	101
9	00000101	11111111	01100000	1	100
10	00000101	00000100	01100000	1	100
11	00000101	00000010	00110000	0	100
12	00000101	00000010	00110000	0	011
13	00000101	00000001	00011000	0	011
14	00000101	00000001	00011000	0	010
15	00000101	00000000	10001100	0	010
16	00000101	00000000	10001100	0	001
17	00000101	00000000	01000110	0	001

```
Produs = 0000000000100011
Stop at simulation time 19
C1>$finish;
```

```
Top-level modules:
   inmultitorb
// Cazul: RD = 5; AC = -7;
Timp  RD        AC        MQ        MQq  CNT
0     00000101  11111001  00000000  0    111
1     00000101  11111001  11111001  0    111
2     00000101  00000000  11111001  0    111
3     00000101  11111011  11111001  0    111
4     00000101  11111101  11111100  1    111
5     00000101  11111101  11111100  1    110
6     00000101  00000010  11111100  1    110
7     00000101  00000001  01111110  0    110
8     00000101  00000001  01111110  0    101
9     00000101  00000000  10111111  0    101
10    00000101  00000000  10111111  0    100
11    00000101  11111011  10111111  0    100
12    00000101  11111101  11011111  1    100
13    00000101  11111101  11011111  1    011
14    00000101  11111110  11101111  1    011
15    00000101  11111110  11101111  1    010
16    00000101  11111111  01110111  1    010
17    00000101  11111111  01110111  1    001
18    00000101  11111111  10111011  1    001
Produs= 1111111111011101
Stop at simulation time 20
C1>$finish;
```

// Cazul: RD = -5; AC = -7;

Timp	RD	AC	MQ	MQq	CNT
0	11111011	11111001	00000000	0	111
1	11111011	11111001	11111001	0	111
2	11111011	00000000	11111001	0	111
3	11111011	00000101	11111001	0	111
4	11111011	00000010	11111100	1	111
5	11111011	00000010	11111100	1	110
6	11111011	11111101	11111100	1	110
7	11111011	11111110	11111110	0	110
8	11111011	11111110	11111110	0	101
9	11111011	11111111	01111111	0	101
10	11111011	11111111	01111111	0	100
11	11111011	00000100	01111111	0	100
12	11111011	00000010	00111111	1	100
13	11111011	00000010	00111111	1	011
14	11111011	00000001	00011111	1	011
15	11111011	00000001	00011111	1	010
16	11111011	00000000	10001111	1	010
17	11111011	00000000	10001111	1	001
18	11111011	00000000	01000111	1	001

Produs= 0000000000100011

Stop at simulation time 20

C1>

10.2.5 ÎMPĂRȚIREA

Ca regulă generală, împărțirea numerelor se realizează prin scăderea repetată a împărțitorului, deplasat spre dreapta cu un rang, din restul de la scăderea precedentă, până la obținerea unui rest egal cu zero sau mai mic decât împărțitorul; ca prim rest se consideră deîmpărțitul.

În general se practică două metode:

- metoda bazată pe refacerea ultimului rest pozitiv (metoda cu restaurare) şi
- metoda în care nu se reface ultimul rest pozitiv (metoda fără restaurare).

Pentru prima metodă se prezintă exemplul următor, în care deîmpărțitul este 22, iar împărțitorul este 9:

$22 : 9 = q_0, q_{-1}q_{-2}$

22	40	40
- 9	- 9	- 9
13	31	31
- 9	- 9	- 9
4	22	22
- 9	- 9	- 9
- 5	13	13
$q_0 = 2$	- 9	- 9
	4	4
	- 9	- 9
	- 5	- 5
	$q_{-1} = 4$	$q_{-2} = 4$

22 : 9 = 2,44...

Cea de-a două metoda pleacă de la ideea că, în loc de a scădea împărțitorul deplasat la dreapta cu un rang, din ultimul rest pozitiv, se va aduna împărțitorul deplasat la dreapt, la ultimul rest negativ. Metoda se va ilustra prin exemplul de mai jos:

22 : 9 = q_0, $q_{-1}q_{-2}$

22	-50	40
- 9	+ 9	- 9
13	- 41	31
- 9	+ 9	- 9
4	- 32	22
- 9	+ 9	- 9
- 5	- 23	13
$q_0 = 3$	+ 9	- 9
	- 14	4
	+ 9	- 9
	- 5	- 5
	+ 9	
	4	
	$q_{-1} = \overline{6}$	$q_{-2} =$

Catul va avea forma: $3, \overline{6}\ 5\ \overline{6}\ 5\ \overline{6}$ ……, în care termenii cu bară sunt negativi $[3, (-6)5\ (-6)5\ (-6)]$, ceea ce va impune efectuarea unei corecții care va aduce rezultatul la forma: 22 : 9 = 2,44444…

Ultima metodă necesită un număr mai mare de paşi, astfel încât, în continuare, se va examina implementarea metodei bazata pe restaurarea ultimului rest pozitiv.

<u>Algoritmul împărțirii numerelor reprezentate în complementul față de doi, cu restaurarea ultimului rest pozitiv</u>

Pentru ilustrarea algoritmului se vor considera următoarele resurse hardware:

- AC – acumulator;
- RD – registrul de date al memoriei;
- MQ – registrul de extensie a acumulatorului;
- CNT – registru incrementator, contor de cicluri.

Împărțitorul Y va fi încărcat în RD, iar deîmpărțitul X în AC. În MQ se va acumula câtul.

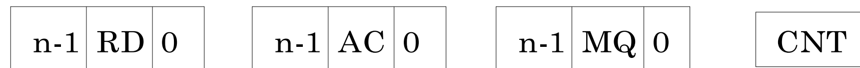


Figura 10.18. Ilustrarea resurselor hardware.

Descrierea algoritmului:

1. Se deplasează conținutul lui RD cu *p* ranguri spre stânga, în condițiile în care 2^p este prima încercare de multiplu binar al împărțitorului.

   Se verifică dacă RD = 0, în caz afirmativ operația se termină cu semnalizarea unei erori.

   Dacă RD $\neq$ 0, se încarcă CNT cu vectorul binar având valoarea p .

2. Dacă deîmpărțitul şi împărțitorul au semne identice/diferite se scade/se adună împărțitorul din/la deîmpărțit, pentru a obține restul.
3. a) Dacă restul şi deîmpărțitul au semne identice sau restul este egal cu zero, se introduce o unitate în bitul cel mai puțin semnificativ al lui MQ, iar restul va lua locul deîmpărțitului.

   b) Dacă restul şi deîmpărțitul au semne diferite şi restul curent nu este zero, se introduce un zero în bitul cel mai puțin semnificativ al lui MQ, fără a mai modifica deîmpărțitul.

4. Se deplasează conținutul registrului împărțitorului cu un rang spre dreapta, extinzând rangul de semn.

   Se decrementează CNT.

   Se deplasează MQ spre stânga cu un bit.

5. Se repetă paşii 2- 4 până când conținutul lui CNT devine egal cu zero.
6. Dacă deîmpărțitul şi împărțitorul au avut semne identice rezultatul se află în registrul MQ. În cazul în care semnele celor doi operanzi au fost diferite, rezultatul se obține prin complementarea conținutului registrului MQ.

Realizarea practică a algoritmului impune introducerea unor resurse hardware suplimentare, față de AC, RD, MQ, CNT şi anume:

- R[n] – registrul în care se obține restul curent;
- z[1] – bistabil în care se stocheaza informația (1/0) referitoare la semnele identice/diferite ale celor doi operanzi;
- e[1] – bistabil care semnalizează condiția de eroare/non-eroare (1/0);
- S – unitate logică combinațională, care generează semnalul de sfârşit al operațiilor de deplasare spre stânga în registrul RD; $S = RD_{n-1} . RD_{n-2} . \overline{RD_{n-3}} \cup \overline{RD_{n-1}} . RD_{n-2}$
- F- unitate logică combinațională, care calculează identitatea/neidentitatea semnelor operanzilor; $F = \overline{AC_{n-1} \oplus RD_{n-1}}$
- V – unitate logică combinațională, care verifică semnele operanzilor din acumulator (deîmpărțit) şi din registrul restului curent R; $V = \overline{AC_{n-1} \oplus R_{n-1}}$
- U – unitate logică combinațională, care verifică existența unui rest curent egal cu zero; $U = \overline{\cup / R}$

Structura generală, la nivel de schemă bloc, a dispozitivului de împărțire este dată în figura 10.19.

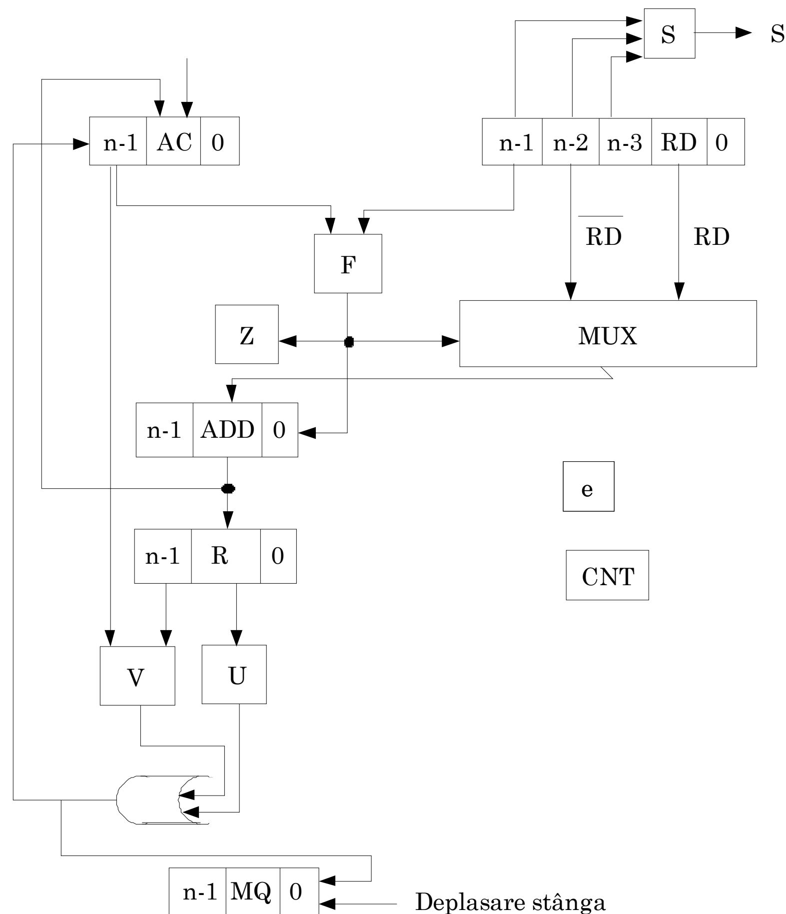


Figura 10.19. Structura generală, la nivel de schemă bloc, a dispozitivului de împărțire.

10.3 OPERAȚII ARITMETICE ÎN VIRGULĂ MOBILĂ

Operațiile aritmetice în virgulă mobilă vor fi examinate la nivelurile schemei bloc, pentru unitatea aritmetică, şi al organigramelor pentru adunare/scădere şi înmultire. În analiza care urmează se consideră operanzii de intrare X ,Y şi rezultatul Z, care vor avea următoarele formate:

X $\leftarrow$ xs, XE, XF ; Y $\leftarrow$ ys, YE, YF; Z $\leftarrow$ zs, ZE, ZF

SCHEMA BLOC A UNITĂȚII ARITMETICE ÎN VIRGULĂ MOBILĂ

Schema bloc a unității aritmetice în virgulă mobilă, în cazul de față, se bazează pe structura schemei unității aritmetice în virgulă fixă, la care s-au mai adaugat o serie de resurse, pentru manipularea exponenților (registre şi unitate aritmetică).

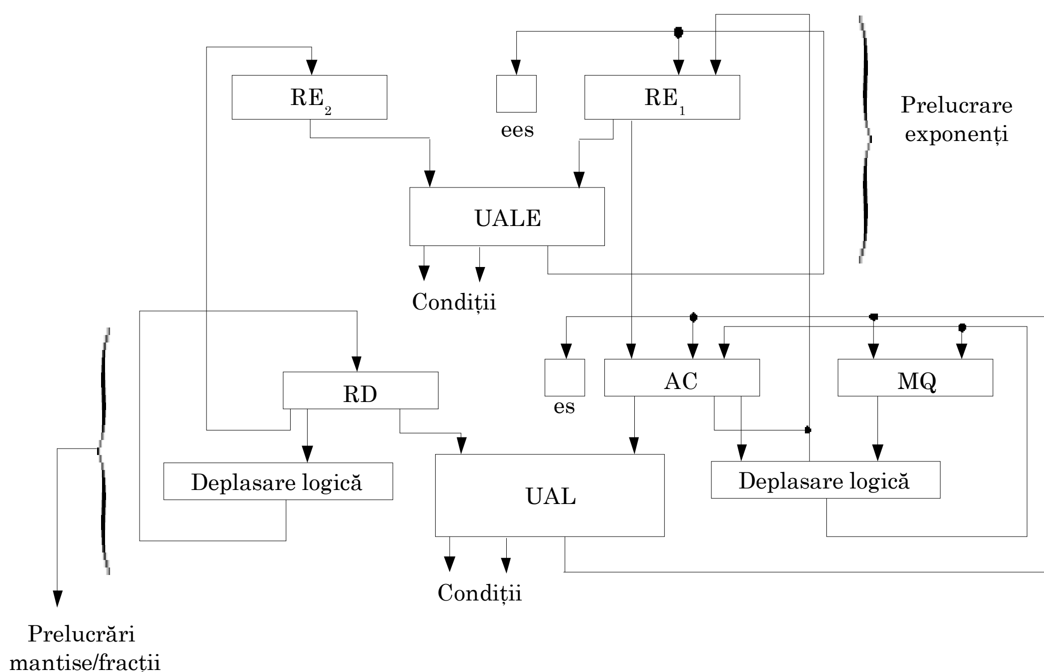


Figura 10.20. Schema bloc a unității aritmetice în virgulă mobilă

Partea care prelucrează exponenții conține următoarele resurse:

- RE_1 şi RE_2 - registre pentru exponenți;
- ees- bistabil de extensie a registrului exponentului la stânga;
- UALE- Unitate Aritmetică Logică pentru Exponenți.

În partea care prelucrează mantisele/fracțiile, față de resursele hardware pentru prelucrarea în virgulă fixă au mai apărut două circuite de deplasare logică și un bistabil de extensie a acumulatorului la stânga *es*. Cele două unități aritmetice logice sunt prevazute cu circuite logice pentru generarea indicatorilor de

condiții şi de eroare. Operanzii de prelucrat se află inițial în registrele AC şi RD. Din acestea se extrag exponenții (operația de despachetare), care sunt încărcați în RE_1 şi RE_2. Fracțiile sunt deplasate spre stânga în AC şi RD, pentru a beneficia de o precizie maximă. După terminarea operațiilor asupra exponenților şi fractiilor, are loc o inserție (operația de împachetare) a exponentului rezultatului în registrul AC, prin deplasarea fracției din AC spre dreapta.

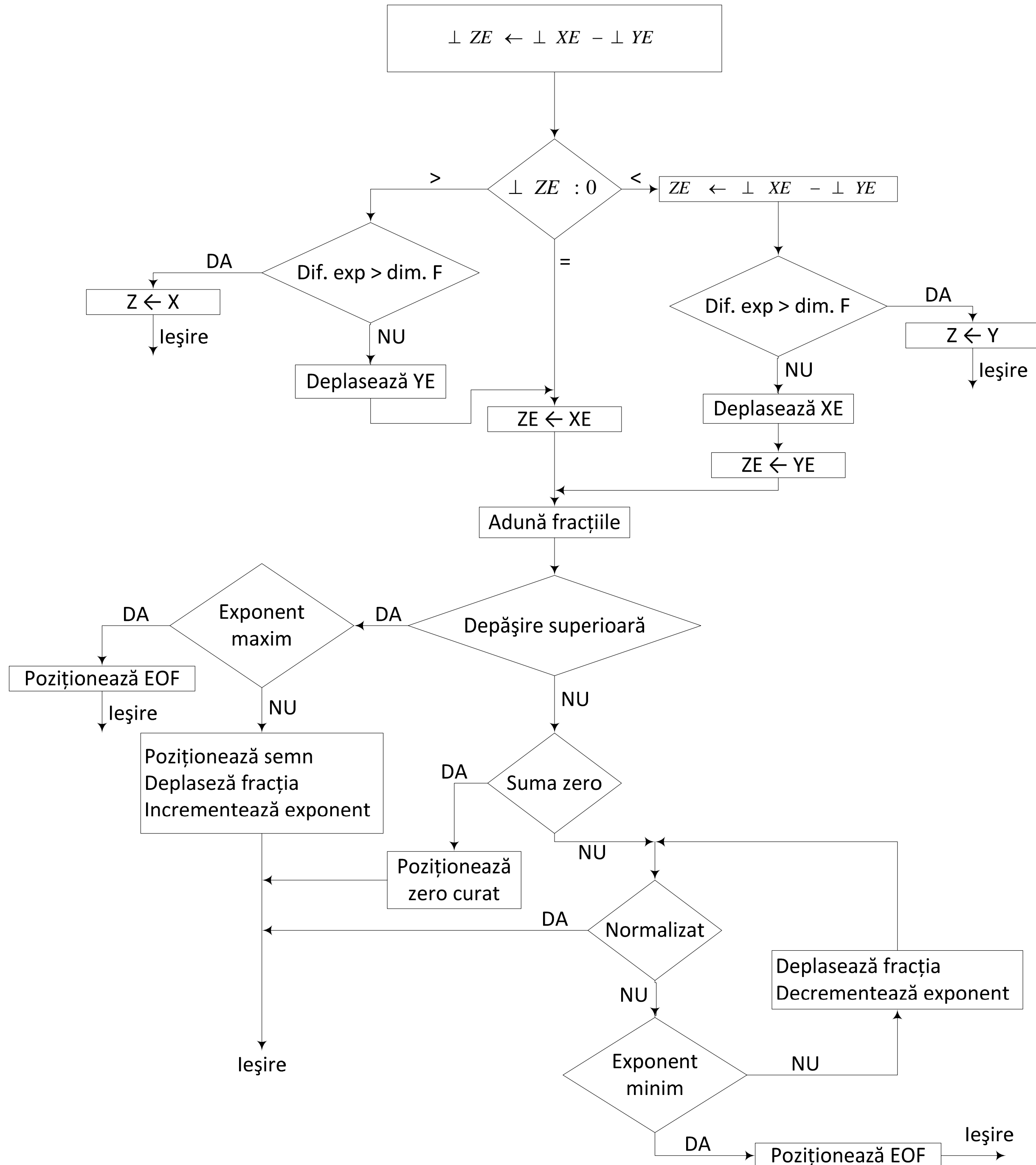


Figura 10.21. Organigrama operației adunare/scădere în virgulă mobilă

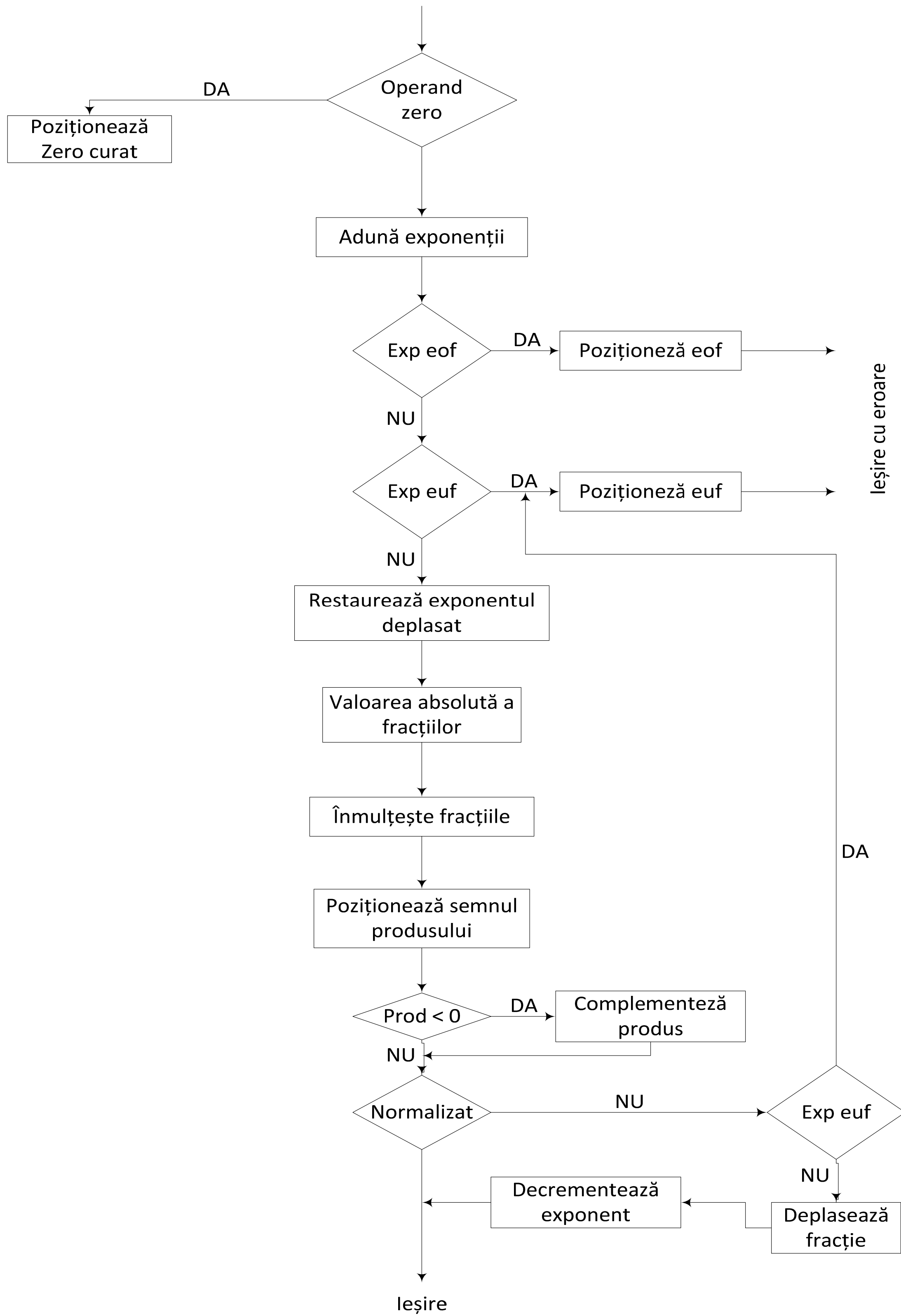


Figura 10.22. Organigrama operației de înmulțire în virgulă mobilă