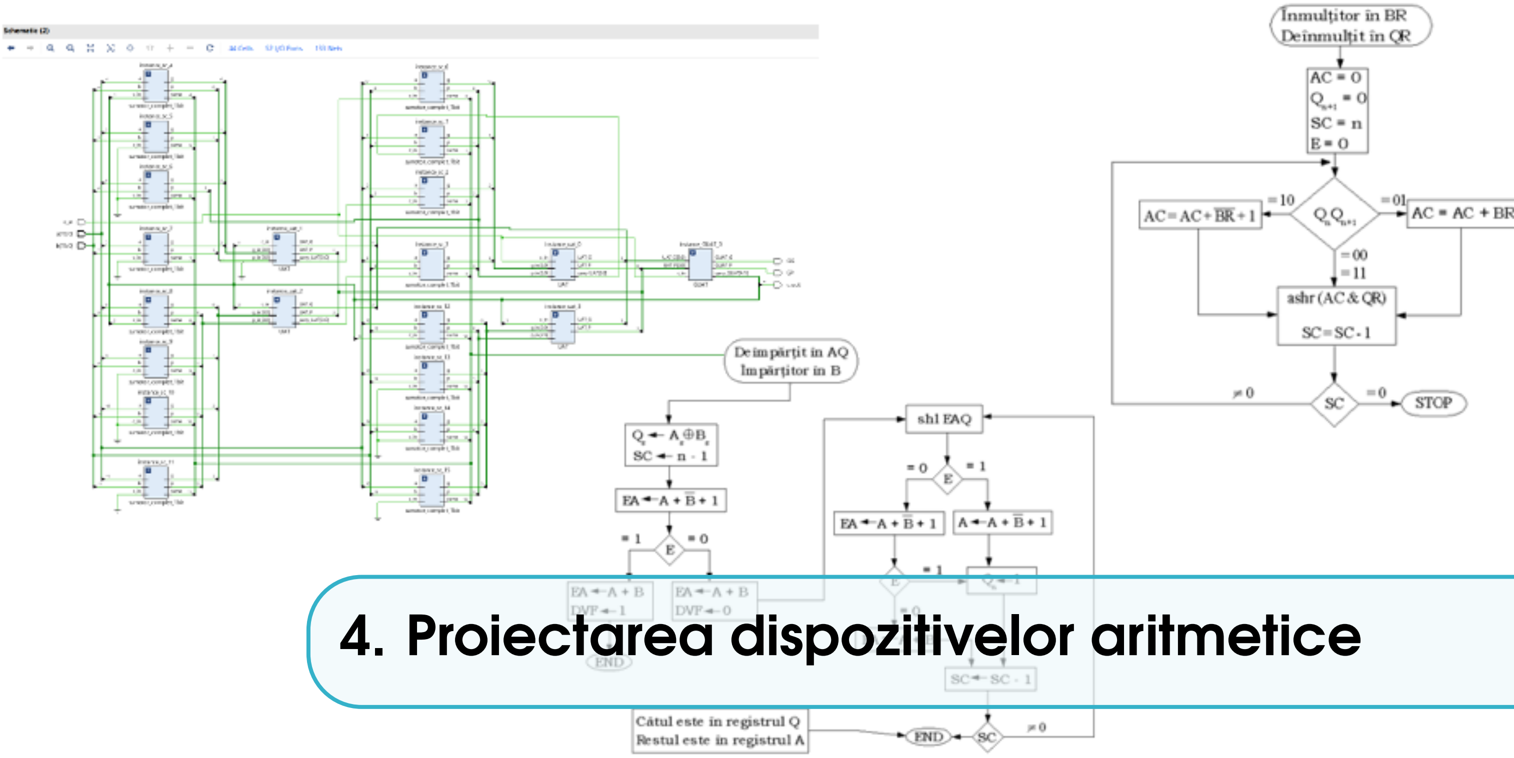


4. Proiectarea dispozitivelor aritmetice

4.1 Prezentare teoretică

Scopul acestui laborator este acela de a prezenta sumatorul **carry look ahead** precum şi o modalitate de implemetare a operaţiilor de adunare, scădere, înmulţire şi împărţire în virgulă fixă. Reprezentarea numerelor în virgulă fixă este descrisă în standardul IEEE 754 şi se presupune cunoscută.

Cuvintele calculatoarelor sunt compuse din biţi, deci ele pot fi reprezentate ca numere binare. Numărul de biţi din care este compus un cuvânt se alege în funcţie de precizia cu care se doreşte realizarea operaţiilor în calculator. Spre exemplu, un cuvânt de 32 biţi oferă o gamă de reprezentare cuprinsă între 0 şi $2^{32} - l$ (4.294.967.295).

Numerele într-un calculator pot fi fără semn sau fără semn. În cazul în care un număr este reprezentat fără semn, el se codieră număr pozitiv. Orice calculator foloseşte pentru numerele binare cu semn reprezentarea în cod complementar faţă de 2. Această reprezentare are avantajul că toate numerele negative au 1 în bitul cel mai semnificativ, ea devenind standardul universal pentru aritmetica de întregi a calculatoarelor.

În cazul în care se lucrează cu numere binare de dimensiuni mari: 32, 64 sau 128 biţi, este recomandată citirea/scrierea lor utilizând o bază mai mare decât baza 2 şi care să asigure ulterior conversia rapidă a numărului într-un număr binar.

Deoarece aproape toate dimensiunile de date de calculator sunt multipli de 4, utilizarea numerelor hexazecimale pentru operaţii de citire/scriere este o soluţie des întâlnită. Conversia din binar în hexazecimal se realizează mecanic prin înlocuirea fiecărui grup de patru cifre binare printr-o singură cifră hexazecimală şi invers.

4.1.1 Proiectarea unei unităţi aritmetice / logice

Unitatea Aritmetică/Logică (UAL) este dispozitivul care efectuează operaţiile aritmetice şi logice. Operaţiile aritmetice şi logice executate de către orice calculator sunt: *adunarea*, *scăderea*, *ŞI la nivel de bit*, *SAU la nivel de bit*.

În calculator, adunarea se efectuează prin adunarea celor două numere bit cu bit de la dreapta la stânga, transportul trecând la următoarea cifră din stânga.

Scăderea foloseşte adunarea: înainte de a fi adunat, scăzătorul este transformat însă în negativul său.

Se va proiecta un circuit care implementeză operaţii aritmetice şi logice pe l bit (UAL de 1 bit), urmând ca acest circuit să fie multiplicat de 32 de ori, pentru a obţine o unitate aritmetică/logică, care lucrează cu numere binare de lungime 32 biţi (UAL de 32 biţi).

Cele mai simple operaţii de implementat sunt operaţiile ŞI şi SAU, care necesită din punct de vedere al resurselor hardware, doar o poartă ŞI cu două intrări, o poartă SAU cu două intrări şi un multiplexor, aşa cum se poate observa în figura: 4.1.a.)

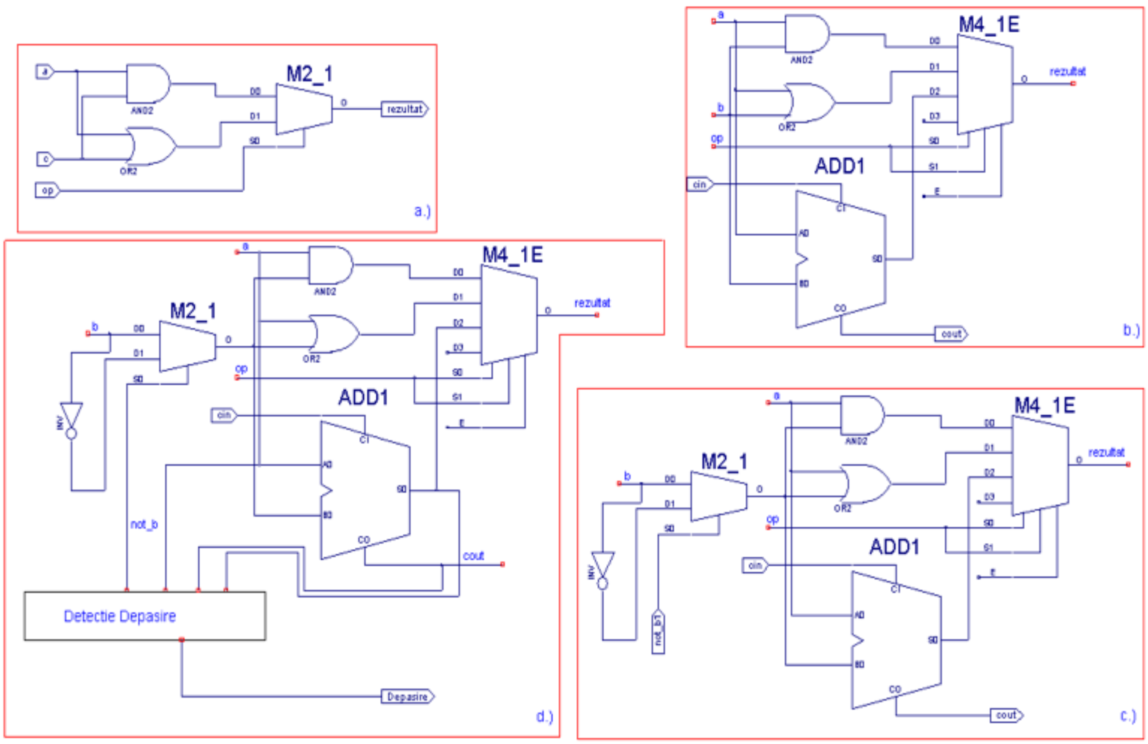


Figura 4.1: UAL de 1 bit care realizează operaţiile *Şl* şi *SAU*.

Următoarea operaţie care va fi implementată este operaţia de *adunare*. Pentru aceasta se va utiliza un sumator de l bit care va fi adăugat logicii deja existente. Unitatea aritmetică logică de l bit care realizează operaţiile: *ŞI*, *SAU* şi adunare este prezentată în figura 4.1.b.)

Pentru implementarea operaţiei de scădere mai trebuie adăugat un multiplexor şi un inversor resurselor hardware deja existente aşa cum se poate observa în figura 4.1.c. Conform acestei figuri, dacă semnalul not_b = l şi semnalul cin = l, atunci se obţine scăderea în cod complementar faţă de 2 a lui b din a.

Tot în figura 4.1.c.) se poate observa că, pentru realizarea operaţiilor logice sau al operaţiei de adunare, este suficient ca semnalele not_b şi cin să fie 0. Din această cauză se pot combina aceste două semnale într-o singură linie de control numită not_b1, ca în figura 4.1.d.)

În cazul în care se adună doi operanzi cu acelaşi semn pot să apară cazuri de depăşire. Acestea trebuie evidenţiate pentru a elimina situaţiile în care rezultatul obţinut poate fi citit/interpretat eronat. O verificare simplă a depăşirii la adunare este să se constate dacă semnalul cin spre cel mai semnificativ bit este diferit de semnalul cout de la cel mai semnificativ bit. Schema finală pentru un

UAL de 1 bit în care sunt semnalate situaţiile de posibilă depăşire este prezentată în 4.1.d.)

Un circuit UAL de 32 de biţi se realizeză printr-o înlănţuire de 32 de unităţi UAL de 1 bit. În cazul în care se doreşte introducerea instrucţiunilor de ramificaţie condiţionată, circuitului UAL de 32 biţi i se mai adaugă un inversor şi o poartă logică OR cu 32 de intrări, aşa cum se poate observa în figura 4.2.

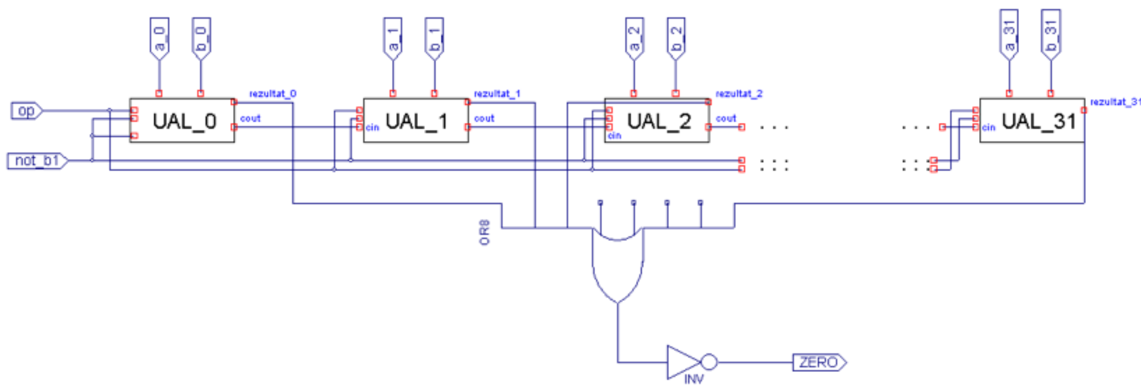


Figura 4.2: UAL de 32 biţi.

Aceste instrucţiuni determină o ramificare a controlului, fie dacă două registre sunt egale în conţinut, fie dacă ele sunt diferite. Verificarea egalităţii cu ajutorul circuitului UAL se realizează prin scăderea lui b din a şi apoi testarea rezultatului cu zero. Astfel, dacă se adaugă hardware pentru a se verifica dacă rezultatul este 0, se poate testa egalitatea. Metoda cea mai simplă de calcul presupune efectuarea unui *SAU* între toate ieşirile, apoi acest semnal să fie scos printr-un inversor.

$$ZERO = \overline{rezultat_{31} + rezultat_{30} + \ldots + rezultat_1 + rezultat_0} \tag{4.1}$$

4.1.2 Sumatorul cu anticiparea transportului

În vederea determinării factorilor *generare* şi *propagare* transport se porneşte de la ecuaţia care determină transportul de intrare pentru sumatorul 1.

$$c_1 = b_0 * c_0 + a_0 * c_0 + a_0 * b_0 \tag{4.2}$$

Factorizând ecuaţia 4.2, se obţine următoarea ecuaţie:

$$c_{i+1} = b_i * c_i + a_i * c_i + a_i * b_i = a_i * b_i + c_i * (a_i + b_i) \tag{4.3}$$

Dacă în această ecuaţie se notează termenul $a_i * b_i$ cu g_i şi $a_i + b_i$ cu p_i atunci ecuaţia 4.3 devine:

$$c_{i+1} = g_i + p_i * c_i \tag{4.4}$$

Termenii g_i şi p_i se numesc *generare transport* şi *propagare transport.* Dacă în ecuaţia 4.4 se presupune că $g_i = 1$, atunci rezultă că $c_{i+1} = 1$. Aceasta înseamnă că sumatorul generează un transport către sumatorul următor, independent de valoarea intrării c. Anolog, dacă se presupune în ecuaţia 4.4, $g_i = 0$ şi $p_i = 1$ atunci rezultă că $c_{i+1} = c_i$, adică sumatorul propagă transportul de la intrare la ieşire.

Exemplu: Să se determine ecuţiile pentru un sumator cu anticiparea transportului care realizează suma a două numere binare de 4 biţi.

Soluţie: Ecuaţiile sunt următoarele:

$$
\begin{aligned}
c_1 &= g_0 + p_0 * c_0 \\
c_2 &= g_1 + p_1 * g_0 + p_1 * p_0 * c_0 \\
c_3 &= g_2 + p_2 * g_1 + p_2 * p_1 * g_0 + p_2 * p_1 * p_0 * c_0 \\
c_4 &= g_3 + p_3 * g_2 + p_3 * p_2 * g_1 + p_3 * p_2 * p_1 * g_0 + p_3 * p_2 * p_1 * p_0 * c_0
\end{aligned}
\tag{4.5}
$$

Aşa cum se observă şi în exemplul prezentat, această formă simplificată conduce la un volum considerabil de circuite logice chiar şi pentru un sumator de 16 biţi. Pentru a înlătura acest inconvenient este nevoie să se treacă la un nivel superior de abstractizare.

Pentru a mări viteza, în cazul folosirii unui sumator de 16 biţi, este necesară efectuarea anticipării transportului cu sumatoare de 4 biţi. Noile semnale, P şi G, vor semnifica propagarea şi generarea transportului la nivel de bloc. Ecuaţiile la nivel de bloc pentru transportul de intrare al fiecărui grup de 4 biţi al sumatorului de 16 biţi sunt prezentate în ecuaţiile 4.6, dar ele sunt asemănătoare cu ecuaţiile 4.5.

$$
\begin{aligned}
C_1 &= G_0 + P_0 * c_0 \\
C_2 &= G_1 + P_1 * G_0 + P_1 * P_0 * c_0 \\
C_3 &= G_2 + P_2 * G_1 + P_2 * P_1 * G_0 + P_2 * P_1 * P_0 * c_0 \\
C_4 &= G_3 + P_3 * G_2 + P_3 * P_2 * G_1 + P_3 * P_2 * P_1 * G_0 + P_3 * P_2 * P_1 * P_0 * c_0
\end{aligned}
\tag{4.6}
$$

unde:

$$
\begin{aligned}
P_0 &= p_3 * p_2 * p_1 * p_0 \\
P_1 &= p_7 * p_6 * p_5 * p_4 \\
P_2 &= p_{11} * p_{10} * p_9 * p_8 \\
P_3 &= p_{15} * p_{14} * p_{13} * p_{12} \\
G_0 &= g_3 + p_3 * g_2 + p_3 * p_2 * g_1 + p_3 * p_2 * p_1 * g_0 \\
G_1 &= g_7 + p_7 * g_6 + p_7 * p_6 * g_5 + p_7 * p_6 * p_5 * g_4 \\
G_2 &= g_{11} + p_{11} * g_{10} + p_{11} * p_{10} * g_9 + p_{11} * p_{10} * p_9 * g_8 \\
G_3 &= g_{15} + p_{15} * g_{14} + p_{15} * p_{14} * g_{13} + p_{15} * p_{14} * p_{13} * g_{12}
\end{aligned}
\tag{4.7}
$$

Implementarea hardware este prezentată în figura 4.4.

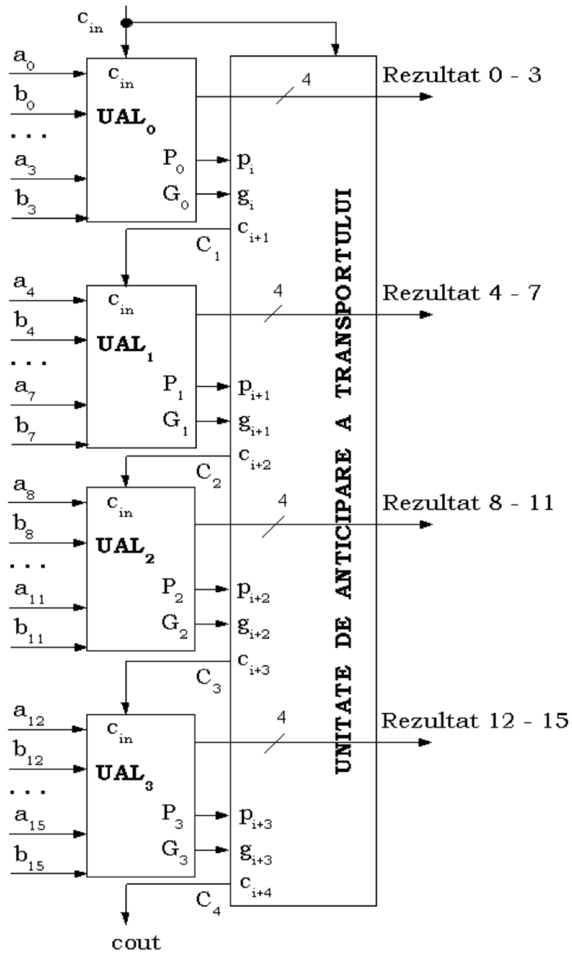


Figura 4.3: Sumator pe 16 biţi

Exemplu: Să se determine valorile g_i, p_i, P_i şi G_i ale următoarelor două numere de 16 biţi: a = 0001 1010 0011 0011 şi b = 1110 0101 1110 1011. Să se determine şi valoarea termenului C_4.

Soluţie: Se determină întâi valorile $g_i = a_i * b_i$ şi $p_i = a_i + b_i$.

$$\begin{aligned}
a &= 0001101000110011 \\
b &= 1110010111101011 \\
g_i &= 0000000000100011 \\
p_i &= 1111111111111011
\end{aligned} \tag{4.8}$$

Determinarea semnalelor P_3, P_2, P_1 şi P_0 sunt simple operaţii ŞI între propagările nivelului inferior.

$$
\begin{aligned}
P_3 &= 1*1*1*1 = 1\\
P_2 &= 1*1*1*1 = 1\\
P_1 &= 1*1*1*1 = 1\\
P_0 &= 1*0*1*1 = 0\\
G_0 &= 0+(1*0)+(1*0*1)+(1*0*1*1) = 0+0+0+0 = 0\\
G_1 &= 0+(1*0)+(1*1*1)+(1*1*1*0) = 0+1+0+0 = 1\\
G_2 &= 0+(1*0)+(1*1*0)+(1*1*1*0) = 0+0+0+0 = 0\\
G_3 &= 0+(1*0)+(1*1*0)+(1*1*1*0) = 0+0+0+0 = 0\\
C_4 &= 0+(1*0)+(1*1*1)+(1*1*1*0)+(1*1*1*0*0) = 0+0+1+0+0 = 1
\end{aligned}
\tag{4.9}
$$

Se observă că la adunarea numerelor a şi b de 16 biţi există un transport de ieşire.

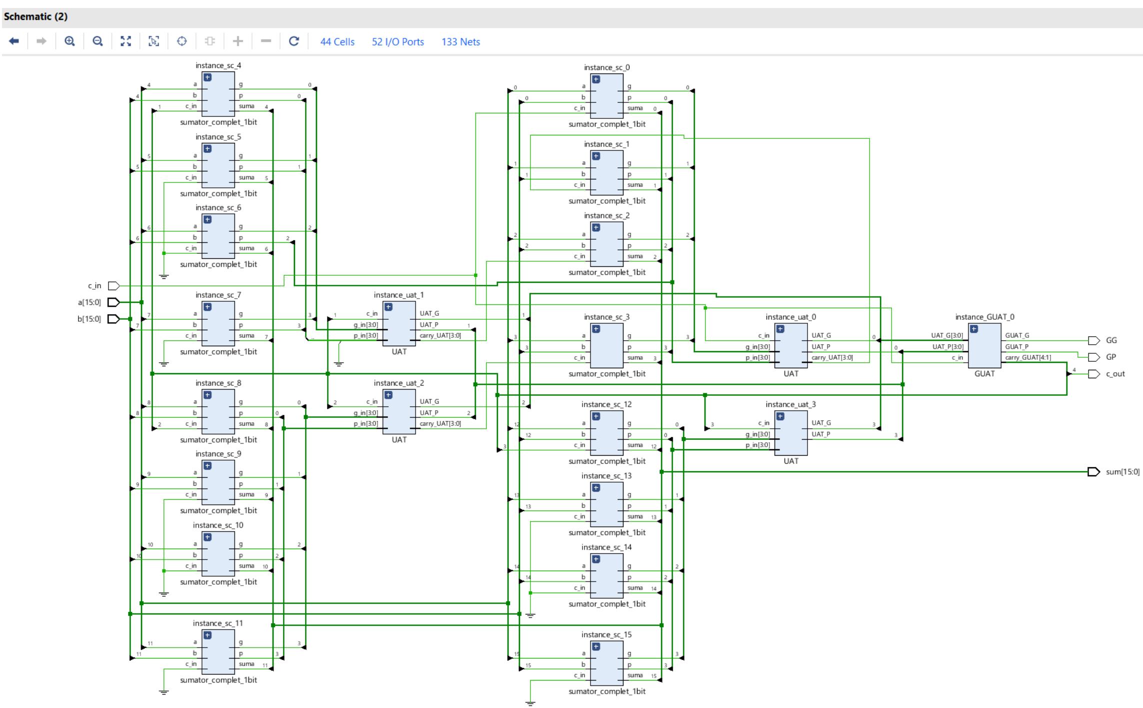


Figura 4.4: Implementarea unui sumator pe 16 biţi

4.1.3 Algoritmul de înmulțire al lui Booth

Acest algoritm oferă o modalitate de a înmuți două numere binare întregi cu semn folosind reprezenarea cod complementar față de 2.

Algoritmul exploatează faptul că secvenţele de 0 din înmulţitor necesită doar deplasare, iar secvenţele de 1 din cadrul înmulţitorului, cuprinse între rangurile 2^k şi 2^m, pot fi tratate ca $2^{k+1} - 2^m$.

Exemplu: Se consideră numărul binar 001 110 (+14). Acest număr conţine o secvenţă de 1 între poziţiile 2^3 şi 2^1. Rezultă k = 3 şi m = 1. Numărul poate fi reprezentat ca: $2^{k+1} - 2^m = 2^4 - 2^1 = 16 - 2 = 14$.

Înmulţirea $M * 14$ (unde M reprezintă înmulţitorul iar 14 deînmulţitul) poate fi realizată ca: $M * 2^4 - M * 2^1$. Se observă că produsul poate fi obţinut prin deplasarea înmulţitorului binar M de 4 ori la stânga şi scăderea lui M deplasat stânga, odată.

Algoritmul lui Booth necesită examinarea biţilor înmulţitorului şi deplasarea produsului parţial. Înainte de a deplasa produsul parţial, înmulţitorul poate fi adunat/scăzut la/din produsul parţial conform regulilor:

1. $Q_n = 0$, $Q_{n+1} = 0$ sau $Q_n = 1$, $Q_{n+1} = 1$ - produsul parțial nu se modifică;
2. $Q_n = 1$, $Q_{n+1} = 0$ - se scade înmulțitorul din produsul parțial;
3. $Q_n = 0$, $Q_{n+1} = 1$ - se adună nmulțitorul la produsul parțial.

În figura 4.5 sunt prezentate resursele hardware necesare implementării acestui algorithm precum şi biţii Q_n şi Q_{n+1}.

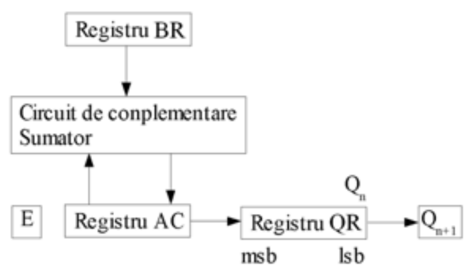


Figura 4.5: Resursele hardware necesare implementării algoritmului lui Booth.

Semnificaţia resurselor folosite în figura 4.5 este următoarea:

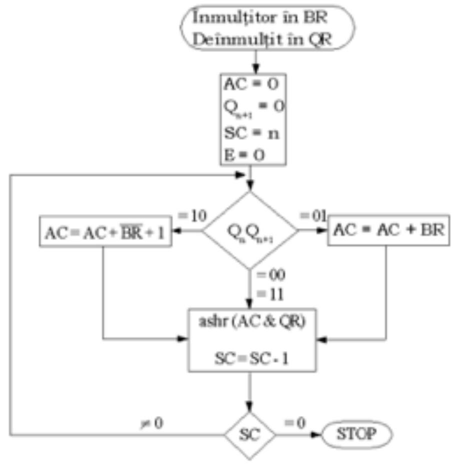
- **Registru BR** – registru care menţine valoarea înmulţitorului;
- **E** – registru de 1 bit folosit pentru memorarea valorii transportului rezultat în urma operaţiei de adunare;
- **Registru AC** – registru auxiliar de lucru;
- **Registrul QR** – registru care menţine valoarea deînmulţitului

Q_n reprezintă cel mai puţin semnificativ bit al deînmulţitului memorat în **QR**. Un registru suplimentar Q_{n+1} este adăugat la **QR** pentru a facilita inspectarea celor 2 biţi ai deînmulţitului. Acest registru este iniţializat cu 0.

Un exemplu numeric pentru algoritmul lui Booth este prezentat în tabelul din figura 4.6a. Algoritmul lui Booth este prezentat sub formă de organigramă în figura 4.6b. Exemplul numeric prezentat presupune că înmulţitorul are valoarea -9 şi deînmulţitul are valoarea -13 (-9 x -13).

Q_n	Q_{n+1}	$\overline{BR}+1=01001$	AC	QR	Q_{n+1}	SC
		Inițial	0000	10011	0	101
1	0	se scade BR	01001			
			01001			
		ashr	00100	11001	1	100
1	1	ashr	00010	01100	1	011
0	1	se adună BR	10111			
			11001			
		ashr	11100	10110	0	010
0	0	ashr	11110	01011	0	001
1	0	se scade BR	01001			
			00111			
		ashr	00011	10101	1	000

(a) Exemplu. ashr = deplasare aritmetică la dreapta



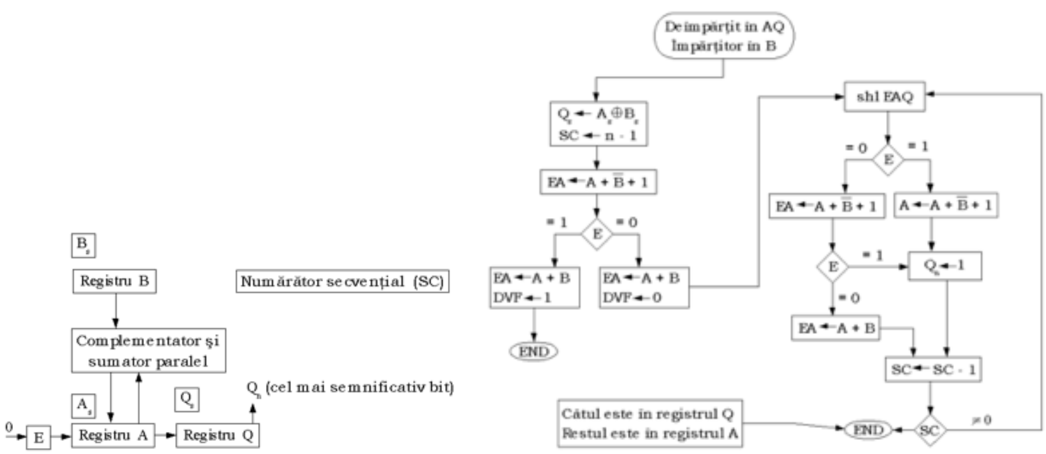
(b) Înmulţirea a două numere binare cu semn

Figura 4.6: Algoritmul lui Booth

4.1.4 Împărțirea numerelor binare în virgulă fixă

Uzual, operaţia de împărţire este realizată folosind comparări succesive, deplasări şi operaţii de scădere. Împărţirea binară este mai simplă decât împărţirea zecimală, deoarece biţii care formează câtul sunt 1 sau 0 şi de aceea nu este necesar să se estimeze de câte ori deîmpărţitul sau restul parţial este cuprins în împărţitor.

Resursele hardware necesare implementării operaţiei de împărţire sunt cele prezentate în figura 4.7a. Algoritmul hardware de împărţire a două numere binare în virgulă fixă este prezentat sub formă de organigramă în figura 4.7b.



(a) Resursele hardware necesare (b) Organigrama

Figura 4.7: Operația de împărțire