LABORATOR 3 – PROIECTAREA DISPOZITIVELOR ARITMETICE ÎN VIRGULĂ FIXĂ

1. Prezentarea teoretică

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

Cuvintele calculatoarelor sunt compuse din biți, deci ele pot fi reprezentate ca numere binare. Numărul de biți din care este compus un cuvânt se alege în funcție de precizia cu care se doreşte realizarea operațiilor în calculator. Spre exemplu, un cuvânt de 32 biți 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 față de 2. Această reprezentare are avantajul că toate numerele negative au l 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 biți, 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 operații de citire/scriere este o soluție 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.

2. 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 implementează 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: 1.a.)

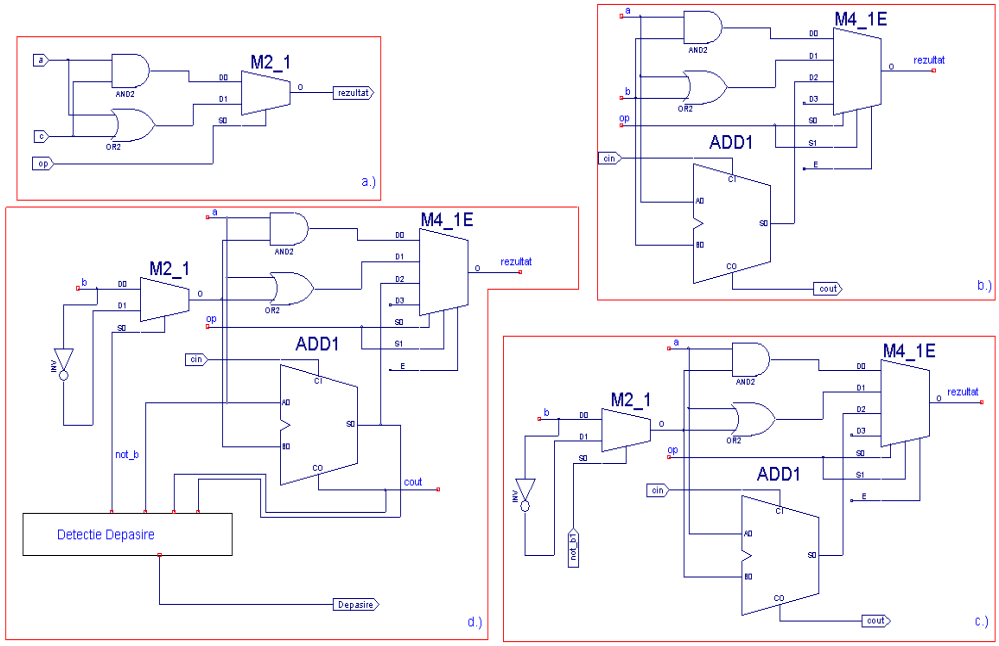

Figura 1 – UAL de 1 bit care realizează operaţiile *ŞI* şi *SAU*

Următoarea operație care va fi implementată este operația 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 operațiile: *ŞI*, *SAU* şi *adunare* este prezentată în figura 3.1.b.)

Pentru implementarea operației de scădere mai trebuie adăugat un multiplexor şi un inversor resurselor hardware deja existente aşa cum se poate observa în figura 3.1.c. Conform acestei figuri, dacă semnalul *not_b* = l şi semnalul *cin* = l, atunci se obține scăderea în cod complementar față de 2 a lui *b* din *a.*

Tot în figura 3.1 c.) se poate observa că, pentru realizarea operațiilor logice sau al operației 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 3.1.d.)

În cazul în care se adună doi operanzi cu acelaşi semn pot să apară cazuri de depăşire. Acestea trebuie evidențiate pentru a elimina situațiile în care rezultatul obținut 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 l bit în care sunt semnalate situațiile de posibilă depăşire este prezentată în figura 3.1.d.)

Un circuit UAL de 32 de biți se realizează printr-o înlănțuire de 32 de unități UAL de l bit. În cazul în care se doreşte introducerea instrucțiunilor de ramificație condiționată, circuitului UAL de 32 biți i se mai adaugă un inversor şi o poartă logică *OR* cu 32 de intrări, aşa cum se poate observa în figura2.

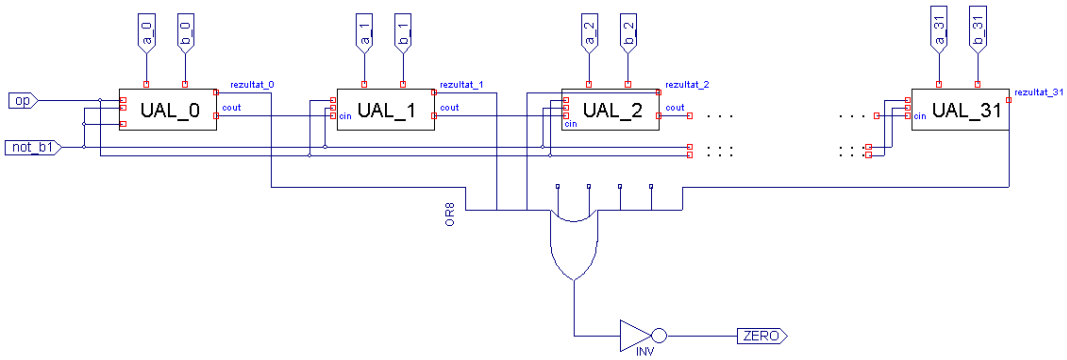

Figura 2 – UAL pe 32 biti

Aceste instrucțiuni determină o ramificare a controlului, fie dacă două registre sunt egale în conținut, fie dacă ele sunt diferite. Verificarea egalității 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.

$$ZERO = \overline{rezultat_{31} + rezultat_{30} + ... + rezultat_1 + rezultat_0} \quad (3.1)$$

3. Sumatorul cu anticiparea transportului

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

$$cin_1 = b_0 \cdot cin_0 + a_0 \cdot cin_0 + a_0 \cdot b_0 \quad (3.2)$$

Factorizând ecuația 3.2, se obține următoarea ecuație:

$$c_{i+1} = b_i \cdot c_i + a_i \cdot c_i + a_i \cdot b_i = a_i \cdot b_i + c_i \cdot (a_i + b_i) \quad (3.3)$$

Dacă în această ecuație se notează termenul $a_i \cdot b_i$ cu g_i şi $a_i + b_i$ cu p_i atunci ecuația 3.3 devine:

$$c_{i+1} = g_i + p_i \cdot c_i \quad (3.4)$$

Termenii g_i şi p_i se numesc *generare transport* şi *propagare transport.* Dacă în ecuația 3.4 se presupune că g_i = l, atunci rezultă că c_{i+1} = 1. Aceasta înseamnă că sumatorul generează un transport către sumatorul următor, independent de valoarea intrării *cin.* Analog, dacă se presupune în ecuația 3.4, 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 ecuațiile pentru un sumator cu anticiparea transportului care realizează suma a două numere binare de 4 biți.

***Soluție**:* Ecuațiile sunt următoarele:

$$
\begin{aligned}
c_1 &= g_0 + p_0 \cdot c_0 \\
c_2 &= g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_0 \\
c_3 &= g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot c_0 \\
c_4 &= g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 + p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot c_0
\end{aligned}
\qquad (3.5)
$$

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 biți. 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 biți, este necesară efectuarea anticipării transportului cu sumatoare de 4 biți. Noile semnale, *P* şi G, vor semnifica propagarea şi generarea transportului la nivel de bloc. Ecuațiile la nivel de bloc pentru transportul de intrare al fiecărui grup de 4 biți al sumatorului de 16 biți sunt prezentate în ecuațiile 3.6, dar ele sunt asemănătoare cu ecuațiile 3.5.

$$
\begin{aligned}
C_1 &= G_0 + P_0 \cdot c_0 \\
C_2 &= G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot c_0 \\
C_3 &= G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot c_0 \\
C_4 &= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot c_0
\end{aligned}
\qquad (3.6)
$$

unde:

$$
\begin{aligned}
P_0 &= p_3 \cdot p_2 \cdot p_1 \cdot p_0 \\
P_1 &= p_7 \cdot p_6 \cdot p_5 \cdot p_4 \\
P_2 &= p_{11} \cdot p_{10} \cdot p_9 \cdot p_8 \\
P_3 &= p_{15} \cdot p_{14} \cdot p_{13} \cdot p_{12} \\
G_0 &= g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 \\
G_1 &= g_7 + p_7 \cdot g_6 + p_7 \cdot p_6 \cdot g_5 + p_7 \cdot p_6 \cdot p_5 \cdot g_4
\end{aligned}
\qquad (3.7)
$$

$$G_2 = g_{11} + p_{11} \cdot g_{10} + p_{11} \cdot p_{10} \cdot g_9 + p_{11} \cdot p_{10} \cdot p_9 \cdot g_8$$

$$G_3 = g_{15} + p_{15} \cdot g_{14} + p_{15} \cdot p_{14} \cdot g_{13} + p_{15} \cdot p_{14} \cdot p_{13} \cdot g_{12}$$

Implementarea hardware este prezentată în figura 3.3.

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

Soluție : Se determină întâi valorile $g_i = a_i \cdot b_i$ şi $p_i = a_i + b_i$.

a: 0001 101000110011

b: 11100101 1110 1011

g_i : 0000000000100011

p_i : 1111 1111 1111 1011

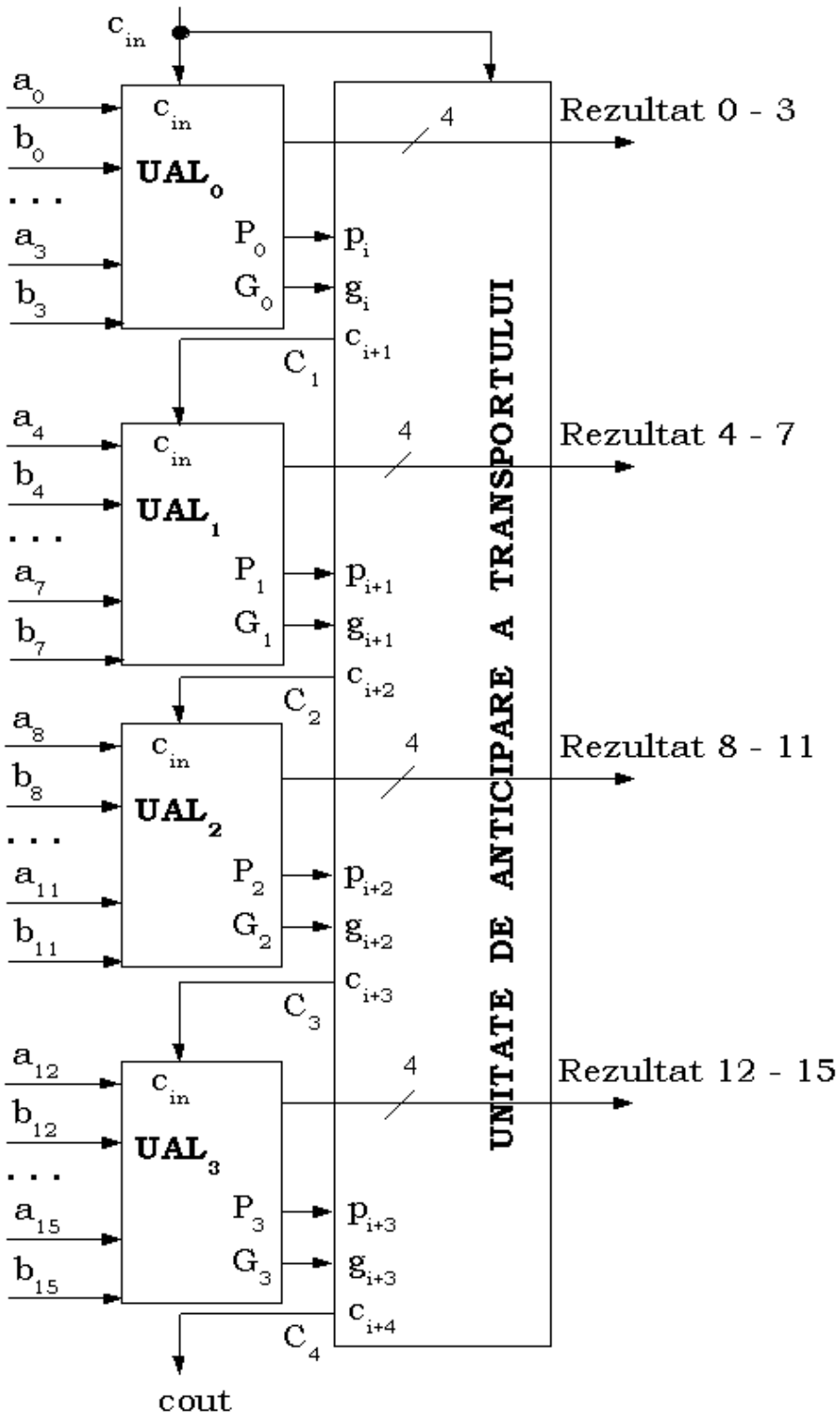

Figura 3 – Sumator pe 16 biti

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

$$P_3 = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

$$P_2 = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

$$P_1 = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

$$P_0 = 1 \cdot 0 \cdot 1 \cdot 1 = 0$$

$$G_0 = 0 + (1 \cdot 0) + (1 \cdot 0 \cdot 1) + (1 \cdot 0 \cdot 1 \cdot 1) = 0 + 0 + 0 + 0 = 0 \qquad (3.8)$$

$$G_1 = 0 + (1 \cdot 0) + (1 \cdot 1 \cdot 1) + (1 \cdot 1 \cdot 1 \cdot 0) = 0 + 0 + 1 + 0 = 1$$

$$G_2 = 0 + (1 \cdot 0) + (1 \cdot 1 \cdot 0) + (1 \cdot 1 \cdot 1 \cdot 0) = 0 + 0 + 0 + 0 = 0$$

$$G_3 = 0 + (1 \cdot 0) + (1 \cdot 1 \cdot 0) + (1 \cdot 1 \cdot 1 \cdot 0) = 0 + 0 + 0 + 0 = 0$$

$$C_4 = 0 + (1 \cdot 0) + (1 \cdot 1 \cdot 1) + (1 \cdot 1 \cdot 1 \cdot 0) + (1 \cdot 1 \cdot 1 \cdot 0 \cdot 0) = 0 + 0 + 1 + 0 + 0 = 1$$

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

4. Algoritmul de înmulțire al lui Booth

Acest algoritm oferă o modalitate de a înmulți două numerelor binare întregi cu semn folosind reprezentarea cod complementar față de 2.

Algoritmul exploatează faptul că secvențele de 0 din înmulțitor necesită doar deplasare, iar secvențele de l din cadrul înmulțitorului, cuprinse între rangurile $2^k - 2^m$, pot fi tratate ca $2^{k+1} - 2^m$.

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

Înmulțirea M $\times$ 14 (unde M reprezintă înmulțitorul iar 14 deînmulțitul) poate fi realizată ca: M x 2^4 - $M \times 2^1$. Se observă că produsul poate fi obținut prin deplasarea înmulțitorului binar M de 4 ori la stânga şi scăderea lui M deplasat stânga, odată.

Algoritmul lui Booth necesită examinarea biților înmulțitorului şi deplasarea produsului parțial. Înainte de a deplasa produsul parțial, înmulțitorul poate fi adunat/scăzut la/din produsul parțial 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 înmulțitorul la produsul parțial.

În figura 4 sunt prezentate resursele hardware necesare implementării acestui algorithm precum şi biții Q_n şi Q_{n+1}.

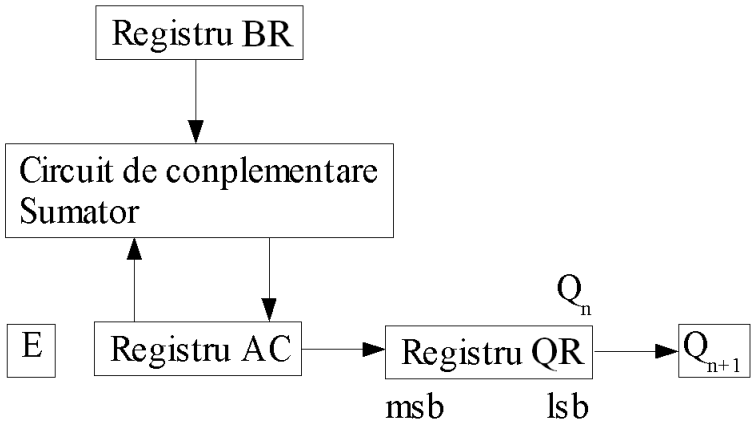

Figura 4 – Resursele herdware necesare implementării algoritmului lui Booth

Semnificația resurselor folosite în figura 4 este următoarea:

- **Registru BR** – registru care menține valoarea înmulțitorului;
- **E** – registru de 1 bit folosit pentru memorarea valorii transportului rezultat în urma operației de adunare;
- **Registru AC** – registru auxiliar de lucru;
- **Registrul QR** – registru care menține valoarea deînmulțitului

Q_n reprezintă cel mai puțin semnificativ bit al deînmulțitului memorat în QR. Un registru suplimentar Q_{n+1} este adăugat la QR pentru a facilita inspectarea celor 2 biți ai deînmulțitului. Acest registru este inițializat cu 0.

Un exemplu numeric pentru algoritmul lui Booth este prezentat în tabelul de mai jos. Algoritmul lui Booth este prezentat sub formă de organigramă în figura 5. Exemplul numeric prezentat presupune că înmulțitorul are valoarea -9 şi deînmulțitul 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

Algoritmul lui Booth. Exemplu. **ashr** = deplasare aritmetică la dreapta

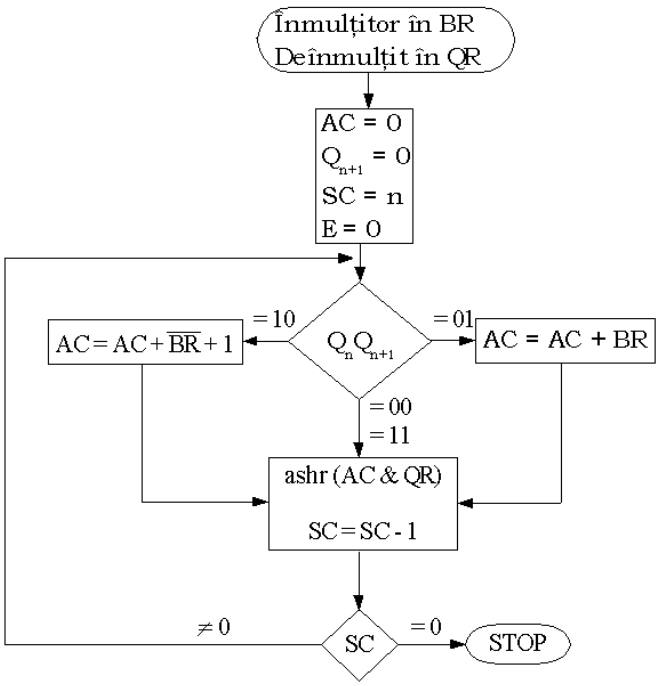

Figura 5 – Algoritmul lui Booth de înmulțire a două numere bianre cu semn

5. Împărțirea numerelor binare în virgulă fixă

Uzual, operația de împărțire este realizată folosind comparări succesive, deplasări şi operații de scădere. Împărțirea binară este mai simplă decât împărțirea zecimală, deoarece biții care formează câtul sunt 1 sau 0 şi de aceea nu este necesar să se estimeze de câte ori deîmpărțitul sau restul parțial este cuprins în împărțitor.

Resursele hardware necesare implementării operației de împărțire sunt cele prezentate în figura 3.6. Algoritmul hardware de împărțire a două numere binare în virgulă fixă este prezentat sub formă de organigramă în figura 7.

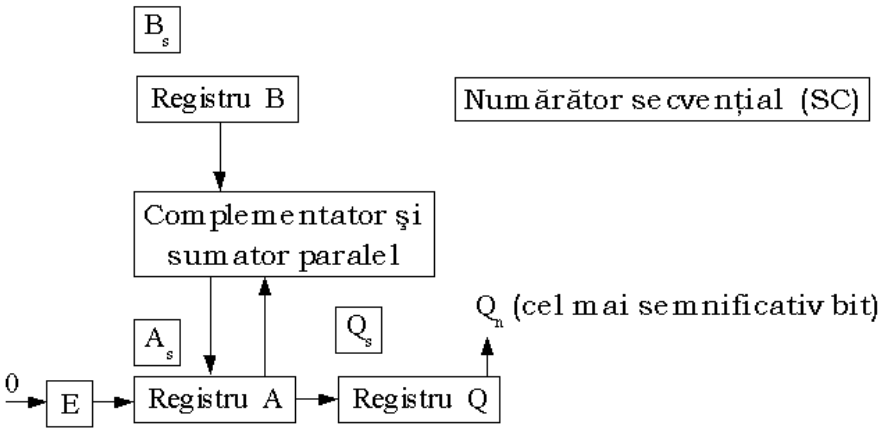

Figura 6 – Resursele hardware necesare operației de împărțire

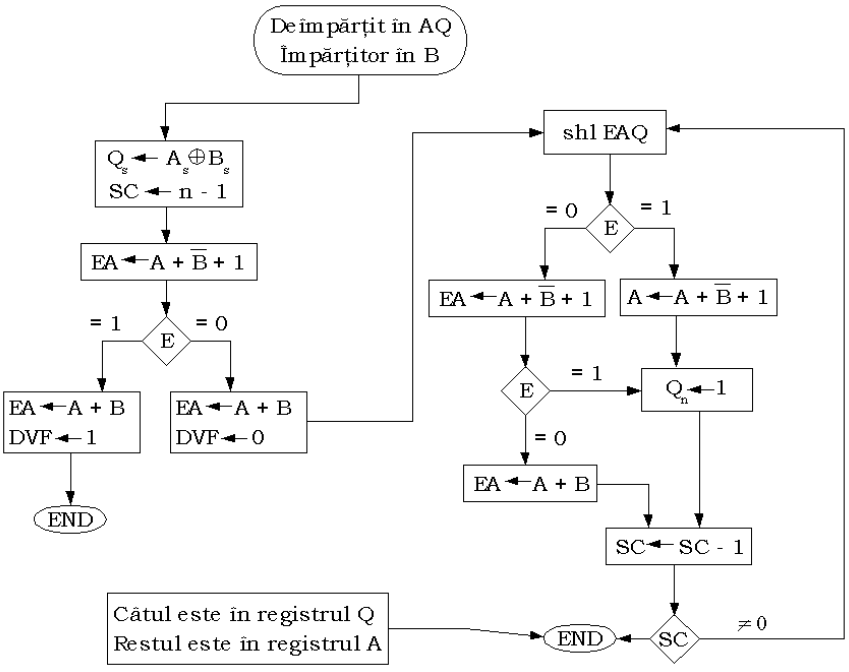

Figura 7 – Organigrama operației de împărțire

6. Desfășurarea lucrării

1. Se va implementa în Verilog şi simula folosind simulatorul ModelSim o unitate aritmetică logică cu dimensiunea de 32 biți. Schema bloc este prezentată în figura 3.2.

2. Se va proiecta, implementa în Verilog şi testa funcționarea unui sumator cu transport anticipat (***carry look ahead***), care operează cu numere de 8 sau 32 de biți.

3. Se va implementa în Verilog şi testa cu ajutorul simulatorului ModelSim algoritmul de înmulțire al lui Booth.

4. Se va implementa în Verilog şi testa cu ajutorul simulatorului ModelSim algoritmul de împărțire prezentat în figura 3.7