# Proiectarea unei unități aritmetico-logice

*Structura Sistemelor de Calcul*

Student: Neacșu Ioana-Alina
Grupa: 30231

Facultatea de Automatică
Și Calculatoare

17 Ianuarie 2024

# Cuprins

# 1 Introducere

## 1.1 Context

Scopul acestui proiect este de a proiecta și implementa o unitate aritmetico-logică (UAL) capabilă să efectueze diverse operații aritmetice și logice. UAL-ul reprezintă o componentă esențială în orice sistem de calcul, gestionând sarcinile de bază necesare pentru procesarea datelor.

## 1.2 Obiective

UAL-ul va fi proiectat în VHDL și implementat într-un proiect Vivado pentru simulare și testare. Acesta va efectua o varietate de operații aritmetice și logice și va utiliza un acumulator pentru stocarea operanzilor și a rezultatelor. Un circuit suplimentar pentru înmulțire și împărțire va fi de asemenea integrat, oferind o funcționalitate completă din punct de vedere aritmetic.

În mod specific, UAL-ul va suporta următoarele operații:

1. Adunare și scădere în complement față de doi (C2)
2. Incrementare și decrementare
3. Negarea pe biți
4. Rotații la stânga și la dreapta pe biți
5. Înmulțire și împărțire prin intermediul unor circuite suplimentare

UAL-ul va fi testat în mod riguros într-un mediu de simulare pentru a asigura corectitudinea funcționării și performanței sale, inclusiv prin utilizarea unui testbench pentru validarea funcționalității.

# 2 Studiu bibliografic

## 2.1 Ce este o unitate aritmetico-logică?

Unitatea Aritmetico-Logică (UAL) reprezintă unul dintre blocurile fundamentale ale unui procesor și este responsabilă pentru efectuarea operațiilor aritmetice, logice și de deplasare-rotire. Aceste operații sunt esențiale pentru funcționarea corectă și eficientă a oricărei unități centrale de procesare (CPU), asigurând manipularea datelor în cadrul programelor.

## 2.2 Algoritmul Shift and Add pentru Înmulțire

Algoritmul Shift-and-Add este o metodă simplă pentru efectuarea înmulțirii binare. Acesta se bazează pe ideea adunării și deplasării (shift) unor valori pentru a calcula produsul a două numere binare, similar modului în care funcționează înmulțirea pe hârtie pentru numerele zecimale.

## 2.3 Algoritmul Restoring Division

Algoritmul de împărțire cu restaurare (restore division) este o metodă eficientă de calculare a câtului și restului în operațiile de diviziune binară. Această abordare implică un proces iterativ în care se compară dividendul cu divizorul și se efectuează operații de subtracție. Această tehnică este similară cu metoda tradițională de împărțire pe hârtie, unde se scrie dividendul și se subtrage divizorul în mod repetat.

## 2.4 Registru de acumulare

Un registru de acumulare (accumulator register) este o componentă esențială în arhitectura calculatoarelor, utilizat pentru a stoca rezultatele intermediare ale operațiilor aritmetice și logice. Acest registru funcționează ca un spațiu temporar unde se efectuează calculul efectiv, iar datele pot fi adunate, scăzute sau manipulate fără a necesita acces constant la memorie.

## 2.5 Sumator

Un sumator este un circuit digital esențial care efectuează operații de adunare binară între două sau mai multe numere. În general, un sumator simplu (de 1 bit) adună doi biți de intrare și un bit de transport (carry) de la operația anterioară, generând un rezultat și un bit de transport nou. Sumatorii sunt extinși pentru a lucra cu numere de mai mulți biți, formând sumatori cu mai multe etaje (n-bit adder), care permit adunarea numerelor binare mai mari. De asemenea, sumatorii sunt folosiți în mod indirect și pentru alte operații aritmetice, cum ar fi scăderea (prin adunarea complementului).

Există mai multe arhitecturi de sumatoare care pot fi utilizate pentru a aduna numere binare de dimensiuni mari într-un mod cât mai eficient. Fiecare tip de sumator are un mod diferit de a gestiona problema transportului (carry), un element esențial care influențează viteza de calcul a operației de adunare.

1. **N-Bit Ripple-Carry Adder (RCA)** este cea mai simplă formă de sumator, în care transportul rezultat din fiecare adunare de biți se propagă secvențial de la primul bit la ultimul.
2. **Carry-Select Adder (CSA)** pre-calculează două rezultate posibile pentru fiecare bloc (unul presupunând că transportul este 0 și altul presupunând că transportul este 1). Odată ce transportul real este cunoscut, rezultatul corect este selectat printr-un multiplexer.
3. **Carry-Skip Adder (CSKA)** este un sumator proiectat să reducă întârzierea de propagare a transportului din RCA. Ttransportul poate "sări" peste anumite blocuri de sumare dacă toate biții dintr-un bloc sunt 1 (cazul când transportul nu trebuie să fie recalculat). Transportul se propagă direct la următorul bloc, reducând astfel întârzierile.

| Adder | Delay Propagare | Arie |
|---|---|---|
| RCA | O(n) | 144 |
| CSA | O(sqrt(n)) | 306 |
| CSKA | O(sqrt(n)) | 150 |

Tabela 1: Comparație între Întârzierea de Propagare și Aria necesară pentru diferite tipuri de sumatoare pe 16 biți, conform studiului realizat de Sağlam și Kaçar (2019).

# 3 Analiza proiectului

## 3.1 Algoritm pentru înmulțire

*Principiul de funcționare*

1. Algoritmul pornește cu două numere: multiplicandul (operandul care trebuie înmulțit) și multiplicatorul (operandul care determină de câte ori se adună multiplicandul).
2. Multiplicatorul este analizat pe fiecare bit, iar dacă bitul curent este 1, multiplicandul este adăugat la un registru de rezultat. Indiferent de valoarea bitului, multiplicandul este apoi deplasat la stânga, iar multiplicatorul este deplasat la dreapta.
3. Acest proces se repetă pentru fiecare bit al multiplicatorului până când acesta devine 0.
4. La final, registrul de rezultat conține produsul celor două numere.

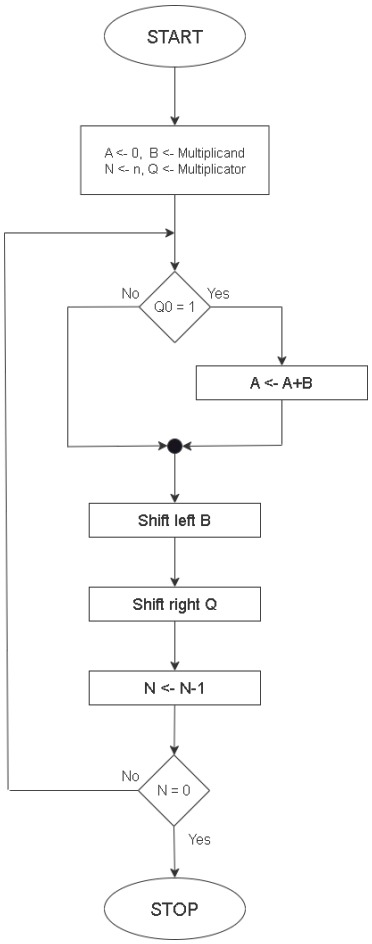


Figura 1: Algoritmul Shift and Add pentru înmulțire

## 3.2 Algoritm pentru împărțire

*Principiul de funcționare*

1. Algoritmul începe cu două numere: deîmpărțitul (numărul care trebuie împărțit) și divizorul (numărul cu care se face împărțirea).
2. În fiecare pas, se deplasează (shift) deîmpărțitul la stânga. Se adaugă un bit 0 la sfârșitul deîmpărțitului pentru a păstra formatul. Se compară divizorul cu porțiunea de bit a deîmpărțitului care este acoperită de divizor. Dacă divizorul este mai mic sau egal cu porțiunea deîmpărțitului, divizorul este scăzut din porțiunea respectivă, iar un bit 1 este adăugat la cât.
3. Procesul se repetă, deplasând deîmpărțitul din nou la stânga și repetând pașii de comparare și scădere pentru fiecare bit al rezultatelor.
4. La final, algoritmul produce câtul și restul, care reprezintă rezultatul împărțirii.

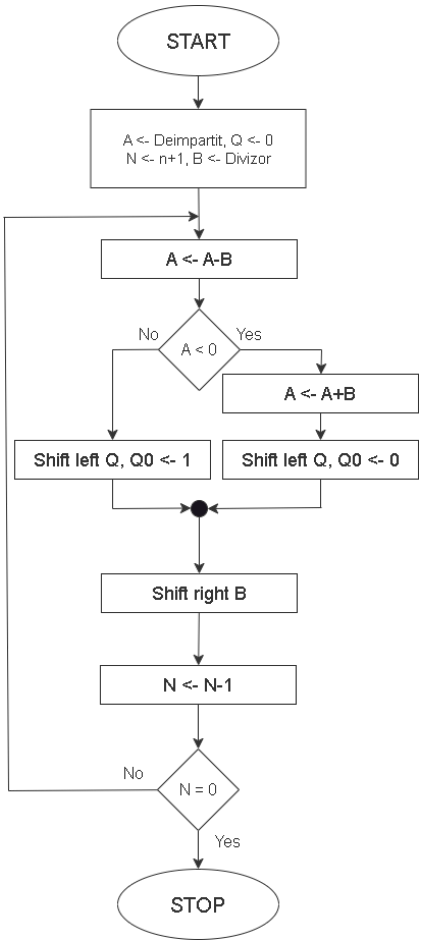


Figura 2: Algoritmul Restoring Division pentru împărțire

### 3.3 Carry-Skip Adder

*Principiul de funcționare*

1. Împărțirea în blocuri: CSKA împarte adunarea într-o serie de blocuri mai mici, fiecare bloc fiind format dintr-un grup de biți. În interiorul fiecărui bloc, adunarea se face printr-un sumator Ripple-Carry.
2. Generarea semnalului de transport: Fiecare bloc de sumare generează un semnal de transport la finalul calculelor sale. Semnalul de transport depinde de operațiile de adunare efectuate în blocul respectiv.
3. Sărirea transportului: Dacă toți biții dintr-un bloc sunt egali cu 1, transportul poate "sări" direct peste acel bloc, fiind transmis către următorul bloc. Aceasta se întâmplă deoarece, în cazul în care toate pozițiile dintr-un bloc sunt 1, transportul este garantat să fie propagat la următorul bloc fără modificări.
4. Reducerea întârzierii: Prin omiterea recalculării transportului în anumite cazuri, CSKA reduce semnificativ întârzierea de propagare a transportului, ceea ce crește viteza totală a sumatorului.

## 4 Design

### 4.1 Sumator Carry-Skip pe 32 biți

Figura 3 reprezintă schema bloc pentru un sumator Carry-Skip pe 32 de biți, împărțit în patru blocuri de câte 8 biți fiecare. Fiecare bloc de 8 biți are rolul de a efectua operația de adunare, iar blocurile sunt conectate într-o structură de tip carry-skip, care optimizează propagarea semnalului de transport (carry) prin omisiunea acestuia în anumite etape, reducând astfel timpii de execuție ai adunării.

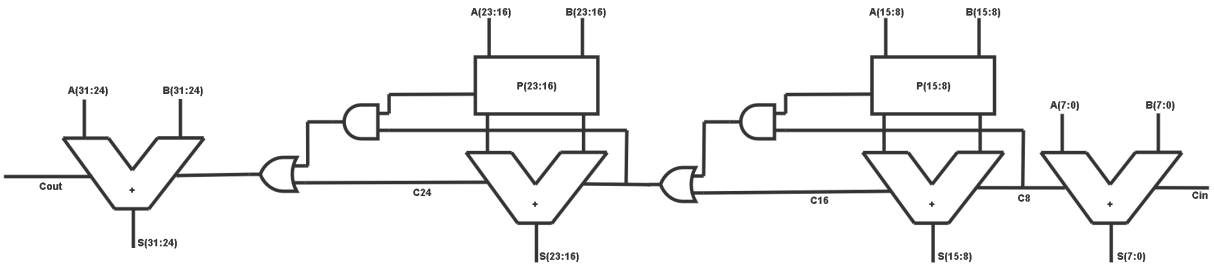


Figura 3: Design Sumator Carry-Skip

### 4.2 Unitatea Aritmetico-Logică

În figura 4 este reprezentată schema bloc pentru UAL care integrează mai multe componente funcționale pentru a efectua operații aritmetice, logice și de rotație. UAL este formată din următoarele componente:

1. Acumulator: Reprezintă o memorie internă care stochează valoarile input-urilor și a rezultatului.
2. Unitate aritmetică: Este responsabilă pentru operațiile aritmetice (adunarea, scăderean, înmulțirea și împărțirea).
3. Unitate logică: Este responsabilă pentru operațiile logice (AND, OR, NOT).

4. Unitate de rotație: Efectuează operațiile de rotație pe biți (rotirea la stânga sau la dreapta).
5. Multiplexor (MUX): Selectează unul dintre rezultatele generate de unitățile aritmetică, logică și de rotație pentru a-l transmite.
6. Unitate de Control (UC): Semnalele pentru selecția operației, a rezultatului și enable pentru fiecare componentă.

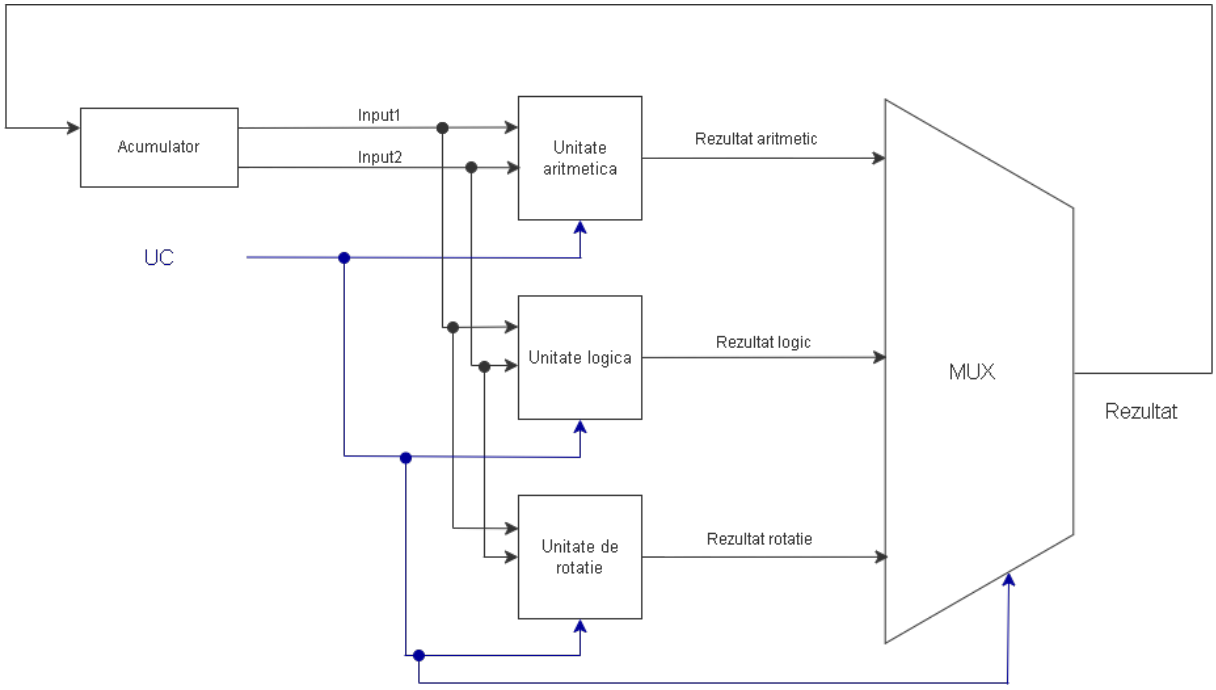


Figura 4: Design ULA

### 4.3 Shift and Add Multiplication

Figura 5 reprezintă schema bloc pentru algoritmul de multiplicare de tip Shift-and-Add Multiplication. Aceasta include următoarele componente principale: ALU pentru adunare, registrul A (Produs) pentru stocarea produsului curent, registrul B (Multiplicand) pentru stocarea valoarea multiplicandului, registrul Q(Multiplicator) pentru stocarea multiplicatorului și Unitatea de Control pentru coordonarea procesului.

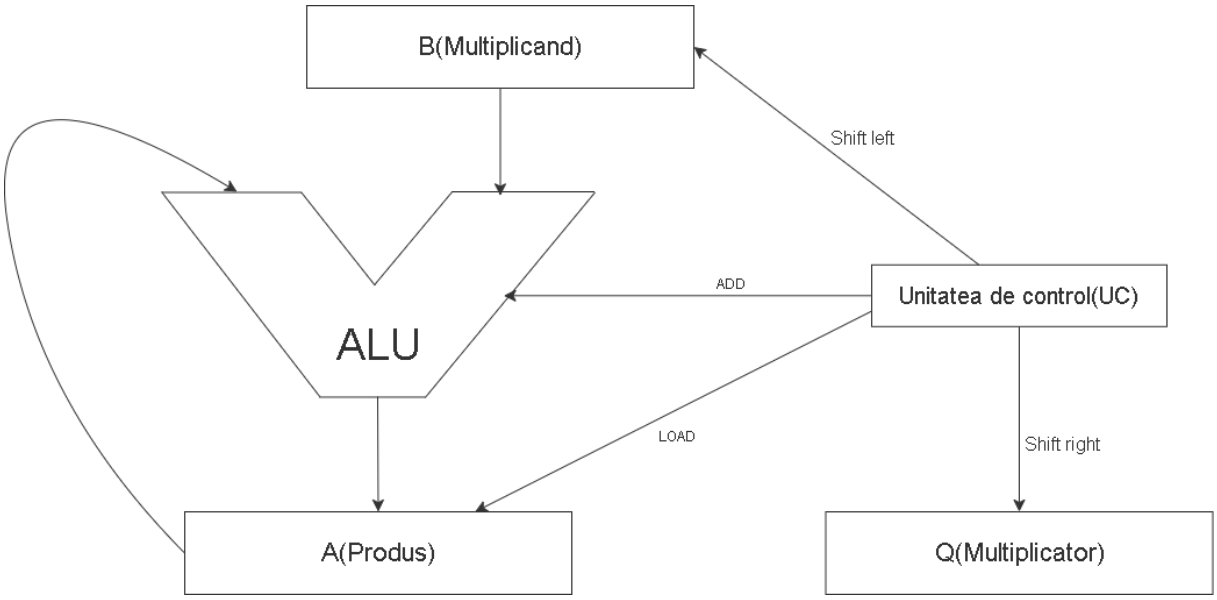


Figura 5: Design Înmulțire

### 4.4 Restoring Division

Figura 6 reprezintă schema bloc pentru algoritmul de diviziune de tip restoring division. Aceasta include următoarele componente principale: ALU pentru scădere și de restaurare (adunare), registrul A (Deîmpărțit, Rest) pentru stocarea dividendului curent și, la final, restul, registrul B (Divizor) pentru stocarea divizorului utilizat, registrul Q (Cât) care conține câtul rezultat si Unitatea de Control pentru coordonarea procesului.

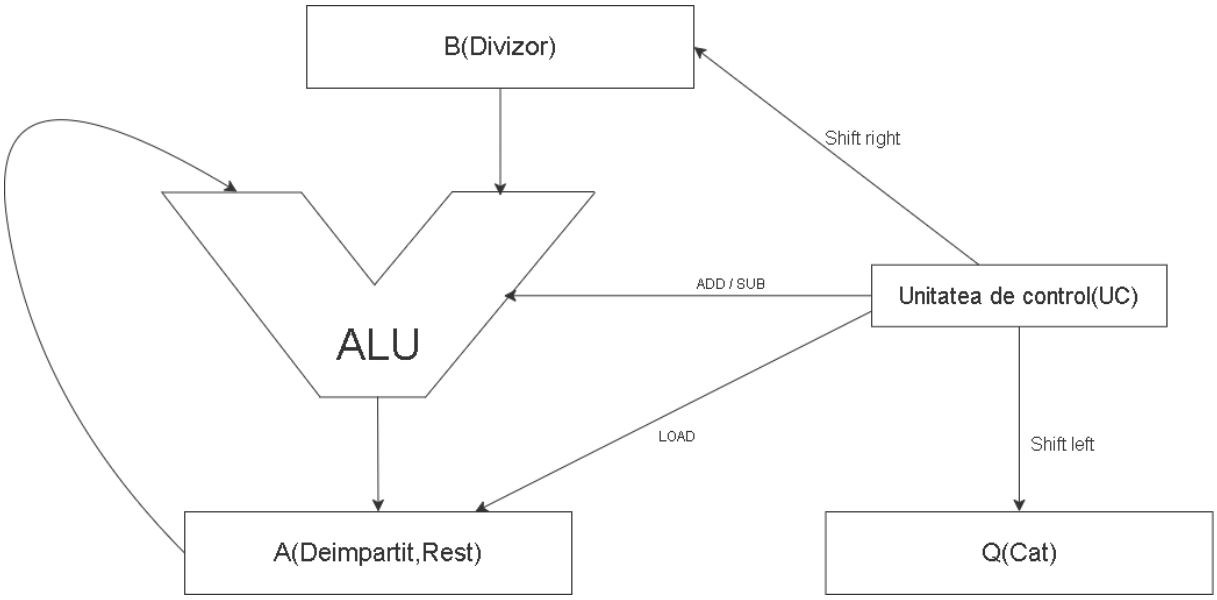


Figura 6: Design Împărțire

S-au implementat următoarele:

1. Sumator pe 32 biti
2. Unitate de rotație
3. Unitate logică

# 5 Bibliografie

## Referințe


[1] Kourav, S., Shah, S., & Shah, S. (2020). *Design and Implementation of 64-Bit Arithmetic Logic Unit on FPGA Using VHDL*. Indian Institute of Technology Allahabad. Retrieved from ResearchGate.

[2] Sağlam, B. N., & Kaçar, F. (2019). Design and Simulation of 64 Bit FPGA Based Arithmetic Logic Unit. *Electrica*, 19(2), 158-165.

[3] Vijay, V., Sreevani, M., Rekha, E. M., Moses, K., Pittala, C. S., Shaik, K. A. S., Koteshwaramma, C., Sai, R. J., & Vallabhuni, R. (2022). *A Review On N-Bit Ripple-Carry Adder, Carry-Select Adder And Carry-Skip Adder. Journal of VLSI Circuits and Systems*, 4(1), 27-32.

[4] Multiplication Algorithm in Signed-Magnitude Representation [Accessed October 2024], `https://www.geeksforgeeks.org/multiplication-algorithm-in-signed-magnitude-representation`

[5] Restoring Division Algorithm for Unsigned Integer [Accessed October 2024], `https://www.javatpoint.com/restoring-division-algorithm-for-unsigned-integer`

[6] What is an accumulator? Definition from TechTarget [Accessed October 2024], `https://www.techtarget.com/whatis/definition/accumulator`