[1] UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

**DOCUMENTAȚIE PROIECT**

**la disciplina Structura Sistemelor de Calcul**

# UAL: Proiectarea unei unități aritmetico-logice

**Student: Pode Dana-Ioana**
**Grupa: 30232**

**An academic : 2023 – 2024**

**CUPRINS**



## 1.Introducere

[1]

1.1 **Context**

Scopul proiectului este de a proiecta și implementa o Unitate Aritmetică-Logică (UAL) pe 32 de biți, capabilă să efectueze diverse operații pe datele binare. Aceste operații includ:

1.Adunarea și scăderea utilizând reprezentarea în complementul față de doi.

2.Incrementarea și decrementarea numerelor binare.

3.Operații logice precum ȘI, SAU și NU.

4.Negarea bit cu bit a numerelor binare.

5.Rotirea la stânga și la dreapta a datelor binare.

Scopul proiectului meu este de a proiecta și implementa o Unitate Aritmetică-Logică (UAL) cu capacitatea de a efectua diverse operații pe date binare. Operațiile includ:

1.Adunare și scădere folosind reprezentarea în complement față de doi.

2.Incrementarea și decrementarea numerelor binare.

3.Operații logice precum ȘI, SAU și NU.

4.Negarea bit cu bit a numerelor binare.

5.Rotirea la stânga și la dreapta a datelor binare.

UAL este o componentă crucială care poate fi integrată în diverse sisteme și dispozitive ce necesită capacități de calcul. Poate servi ca parte esențială a unui procesor sau microprocesor, permițând executarea unei game variate de operații aritmetice și logice. În plus, poate fi folosită ca dispozitiv periferic conectat la un dispozitiv principal, primind comenzi pentru efectuarea de operații precum adunare, scădere, operații logice și rotații pe date binare.

Utilizarea unui acumulator facilitează stocarea eficientă și manipularea datelor de intrare și a rezultatului, permițând UAL să proceseze datele în mod eficient și precis. Proiectarea și implementarea UAL au ca scop furnizarea unei soluții versatile și fiabile pentru efectuarea de operații aritmetice și logice fundamentale pe date binare într-o gamă de aplicații informatice.

1.2 **Specificații**

Proiectul va fi implementat în mediul de proiectare Vivado, un mediu puternic și flexibil de proiectare a circuitelor, dezvoltat de Xilinx. Această platformă va asigura o implementare eficientă și optimizată a UAL, iar arhitectura de 32 de biți va permite manipularea și procesarea eficientă a datelor cu precizie și fiabilitate. Implementarea în Vivado va permite, de asemenea, o integrare ușoară a UAL într-o gamă largă de dispozitive și sisteme, asigurând o funcționare stabilă și performanțe ridicate într-un spectru divers de aplicații.

1.3 **Obiective**

Obiectivele includ optimizarea performanțelor pentru a asigura o execuție eficientă a operațiilor, gestionarea precisă a erorilor pentru a asigura funcționarea stabilă și fiabilă a UAL, precum și documentarea detaliată a întregului proces de proiectare și implementare. În plus, se va dezvolta o interfață intuitivă pentru utilizatori, facilitând introducerea datelor și vizualizarea clară a rezultatelor operațiilor desfășurate de UAL. Dispozitivul poate efectua operația în funcție de alegerea utilizatorului, iar după finalizarea tuturor etapelor operației, rezultatul trebuie afișat pe display-urile cu 7 segmente.

## 2. **Studiu bibliografic**

Unitatea aritmetico-logice(UAL) este un circuit digital esențial responsabil de executarea operațiilor aritmetice, logice și de deplasare în cadrul unui sistem informatic. Servind ca un bloc de construcție fundamental al unității centrale de prelucrare (CPU), chiar și cele mai simple microprocesoare integrează UALpentru a gestiona sarcini precum menținerea temporizatoarelor. În calculul contemporan, atât CPU-urile, cât și unitățile de procesare grafică (GPU) găzduiesc UAL-uri puternice și intricate, iar un singur component include adesea mai multe astfel de circuite pentru a stimula puterea de calcul.

Tipuri de operații calculate de UAL:

1. aritmetice: operații de bază: adăugare, scădere, decrement, înmulțire, împărțire, modulo
2. logice: AND, OR, NOT, inclusiveOR, exclusiceOR, rotație stânga si dreapta

Adunarea este operația este cea mai utilizată. Implementarea eficientă a operatiunii de adunare influențează direct toate celelalte operațiuni, astfel: scăderea reprezintă adăugarea complementului, împărțirea reprezintă scăderea repetitivă și adăugare iar multiplicarea adăugare repetitivă.

Pentru implementarea adunării am ales sa folosesc ***Ripple Carry Adder.*** Acesta poate fi descris in următorii pași:

1. Inițializarea bitului de transport la 0.
2. Adunarea biții celor două numere împreună cu transportul.
3. Adunarea biților corespunzători din cele două numere și transportul rezultat din pasul anterior.
4. Actualizarea valorii bitului de transport pentru următorul pas.
5. Obținerea rezultatului final al adunării și bitul de transport, dacă este necesar.

Scăderea implică transformarea numărului care urmează să fie scăzut în complementul său față de 2, apoi aplicarea aceluiași algoritm ca la adunare, în care se adaugă complementul la numărul de la care se scade. Pașii pentru efectuarea scăderii folosind complementul față de 2:

1. transformarea numărului de scăzut în complementul față de 2
2. adăugarea scăzătorului la complementul față de 2 al numărului de scazut, folosind același algoritm ca și în adunare.

Multiplicarea cu algoritmul lui **Both** constă în următorii pași:

1. Inițializează A, Q și Q^(-1) la 0 și numărul de iterații la n.

2. Bazat pe valorile lui Q0 și Q^(-1), efectuează următoarele:

a. Dacă Q0, Q^(-1) = 0, 0, atunci decalează A, Q și Q^(-1) spre dreapta și decrementează numărul de iterații cu 1.

b. Dacă Q0, Q^(-1) = 0, 1, atunci adună A și B și stochează rezultatul în A, decalează A, Q și Q^(-1) spre dreapta și decrementează numărul de iterații cu 1.

c. Dacă Q0, Q^(-1) = 1, 0, atunci scade B din A și stochează rezultatul în A, decalează A, Q și Q^(-1) spre dreapta și decrementează numărul de iterații cu 1.

d. Dacă Q0, Q^(-1) = 1, 1, atunci decalează A, Q și Q^(-1) spre dreapta și decrementează numărul de iterații cu 1.

Se repetă pasul 2 până când numărul de iterații nu mai este 0.

Algoritmul de **împărțire**:

1. se încarcă primul operand în A și Q, iar cel de-al doilea operand în B.
2. se calculează suma dintre AS și BS și se scrie rezultatul în QS.

-dacă AS = 1, se face complementarea lui A și Q.

-dacă BS = 1, se face complementarea lui B.

3. se fac următoarele teste:

A ≥ B, ceea ce poate indica un overflow.

B = 0, semnalează o împărțire cu 0.

A = 0 și Q < B, rezultatul este 0.

4. se face o deplasare la stânga pentru A și Q și se introduce 0 în Q0.

5.se scade B din A și se introduce rezultatul înapoi în A.

-dacă AS = 0 (rezultat pozitiv), se face o deplasare la stânga pentru A și Q și se introduce 1 în Q0.

-altfel (AS = 1, rezultat negativ), se adaugă B la A, se face o deplasare la stânga pentru A și Q, și se introduce 0 în Q0.

6. se repetă pasul 5 de n ori.

7. se rotunjește rezultatul. Dacă A ≥ B, se adaugă 1 la complementul lui Q.

8. dacă QS = 1, se face complementarea lui Q.

Pentru implementarea *operațiilor logice* pe biți, este necesară manipularea a două intrări, fiecare pe 32 de biți, și generarea unei ieșiri, de asemenea, pe 32 de biți. Cu toate acestea, în cazul operației "Nu logic" (NOT), este suficientă o singură intrare. Procesul presupune utilizarea unui ciclu de la 0 la 31, în care, prin aplicarea operațiilor logice "SAU" (OR), "ȘI" (AND), "ȘI-NU" (NAND) și "NU" (NOT), se realizează operațiile corespunzătoare între fiecare bit al celor două intrări, iar apoi se plasează bitul rezultant pe poziția specifică din ieșire. Astfel, operațiile logice sunt aplicate pe fiecare pereche de biți pentru a obține ieșirea finală pe 32 de biți.

## 3.**Analiza**

Datele vor fi reprezentate in Complement față de doi (C2). Acesta are mai multe avantaje:

1. Scăderea poate folosi aceeași logică ca și adunarea. Operația de scădere poate fi efectuată prin adunarea unui număr cu complementul său față de 2, nu este necesară o logică separată pentru operațiile de adunare și scădere.
2. Bitul de semn poate fi tratat ca un bit normal al numărului în adunare. Complementul față de 2 permite reprezentarea atât a numerelor pozitive, cât și a celor negative folosind aceeași convenție binară, bitul de semn poate fi tratat în mod uniform în operațiile de adunare, fără a necesita o logică specială pentru manipularea semnului.

| Număr | $+8_{10}$ | $-1_{10}$ | $-34_{10}$ | $-128_{10}$ | $-129_{10}$ |
|---|---|---|---|---|---|
| *Pasul 1.a* | $0000\ 1000_2$ | $0000\ 0001_2$ | $0010\ 0010_2$ | $1000\ 0000_2$ | $1000\ 0001_2$ |
| *Pasul 1.b* | - | $1111\ 1110_2 +$ | $1101\ 1101_2 +$ | $0111\ 1111_2 +$ | $0111\ 1110_2 +$ |
| *Pasul 2* | - | 1 | 1 | 1 | 1 |
| *Rezultat* | $0000\ 1000_{c2}$ | $1111\ 1111_{c2}$ | $1101\ 1110_{c2}$ | $1000\ 0000_{c2}$ | $0111\ 1111_{c2}$ |
| *Observații* | *nr. pozitiv, nu se aplică cei 2 pași* | | | | *bit de semn = 0 în c2, nu se poate reprezenta pe 8 biți* |

Tabel 3.1. Procesul de obținere a reprezentării în complement față de 2, exemple pe 8 biți

*1.Adunarea*

Ripple Carry Adder este implementat de mai multe sumatoare complete în serie, fiecare ieșire de transport fiind conectată la intrarea următorului. Acesta este un sumator paralel și este folosit pentru adunarea unui număr de n biți. Are avantajul simplității și costului redus, dar la costul unei viteze reduse.

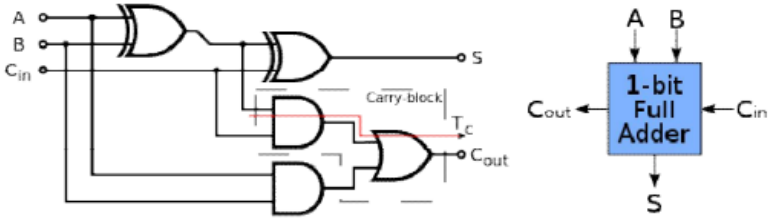


Fig 3.3: Full adder circuit and block diagram.

*2.Scăderea*

Add/Sub = 0 => adunare

Add/Sub = 1 => scădere

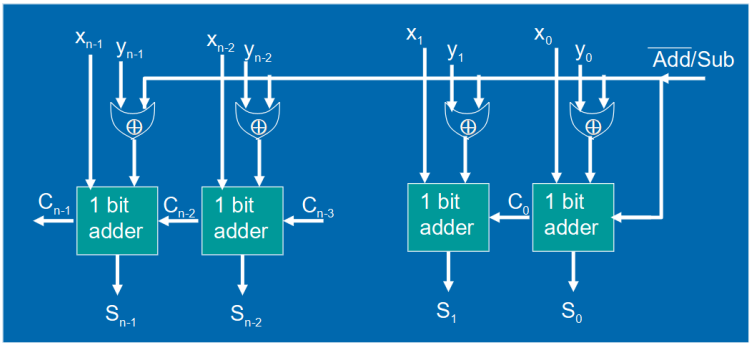


Fig 3.4: Sumator si scazator pe n biti

*3.Multiplicare*

Un algoritm de multiplicare începe prin inițializarea operanzilor în registre și ștergerea unui acumulator. În timpul procesului, în cazul numerelor negative, se verifică valoarea bitului de la coada

multiplului. Dacă acesta este 0, se realizează o deplasare spre dreapta a registrului, iar dacă este 1, se adună valoarea registrului B la acumulatorul A, urmată de o deplasare spre dreapta a ambelor registre. Acest proces continuă până când bitul de la penultima poziție a multiplului devine bitul cel mai puțin semnificativ al multiplicatorului. După finalizarea operațiilor, se efectuează verificarea bitului Q0 pentru a evalua rezultatul.

Îmbunătățirea acestui algoritm o reprezintă algoritmul lui **Both**. Acesta multiplica numere în complement față de 2( nu mai este necesara complementarea inițiala). Pentru secvențe lungi de 0 și 1 sunt necesare numai operațiuni de shiftare.

Exemple pentru secvențe de 1:

1111 = 10000 -1;

11.1111 = 100.000 – 1

O secventa de 1 se poate schimba intr-o secventa de zerouri minus 1.

Numai tranzițiile de la 0 la 1 sau 1 la 0 necesită adăugare sau scădere. Dacă doi biți sunt consecutivi în al doilea operand sunt:

0 si 0 - shift shiftare la dreapta

0 si 1 – aduna operand si shiftare la dreapta

1 si 1 – shiftare la dreapta

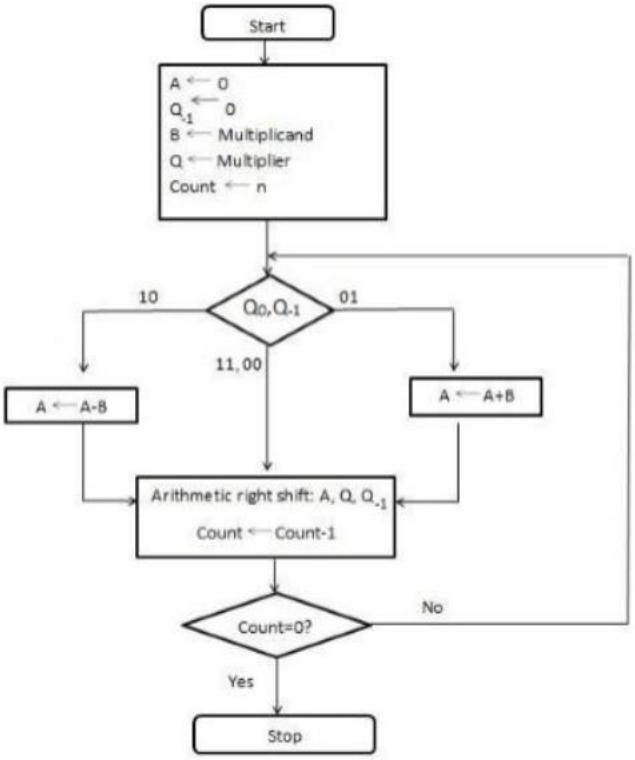


Fig. 3.5. The flowchart of Both Algoritm

*4.Împărțire*

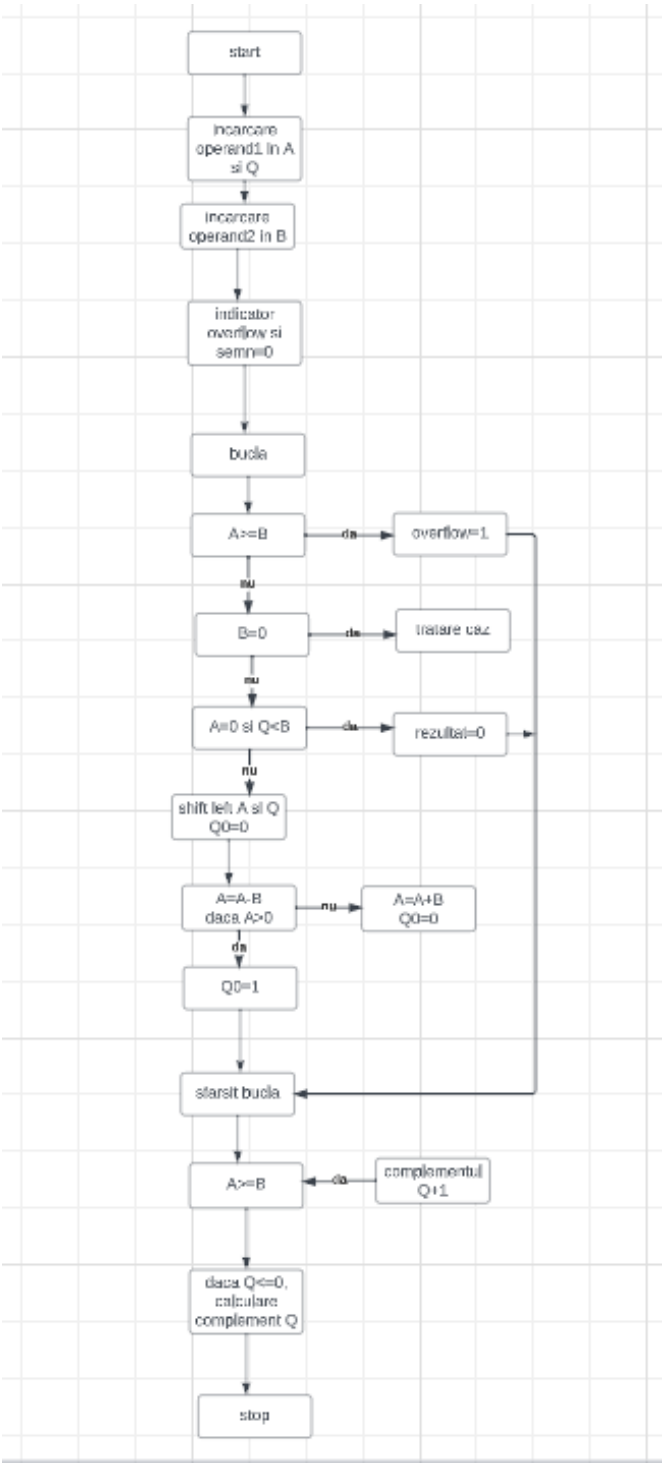


Fig4.1 Flowchart impărțire

*5.Operații logice*

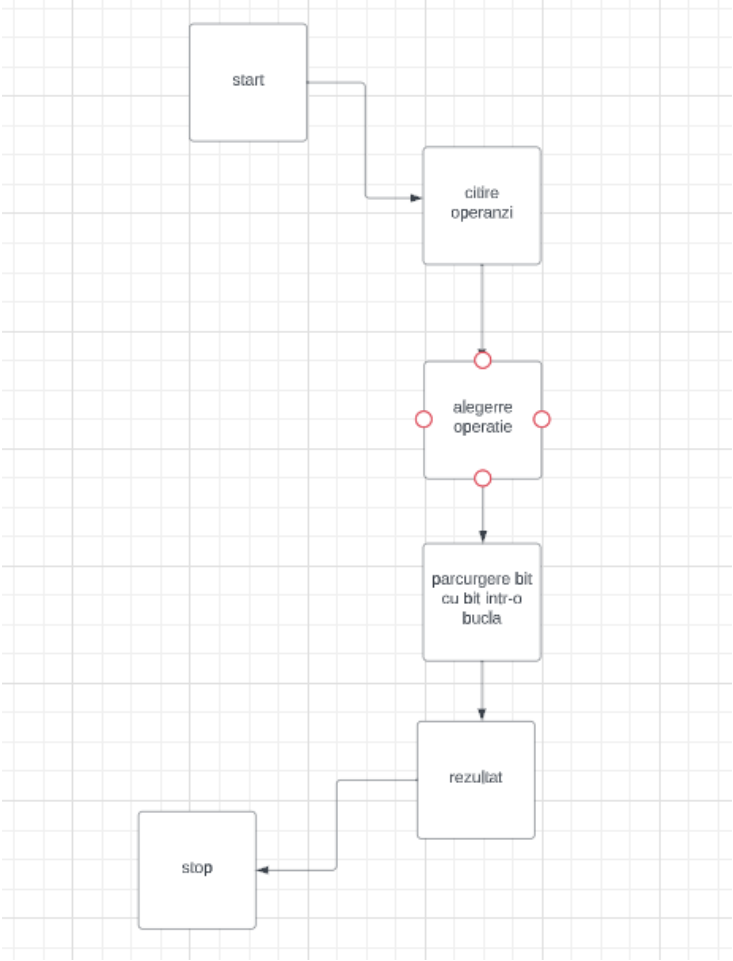


Fig 5.1. FlowChart realizarea operaților logice

4. **Proiectare**

**Adunarea**

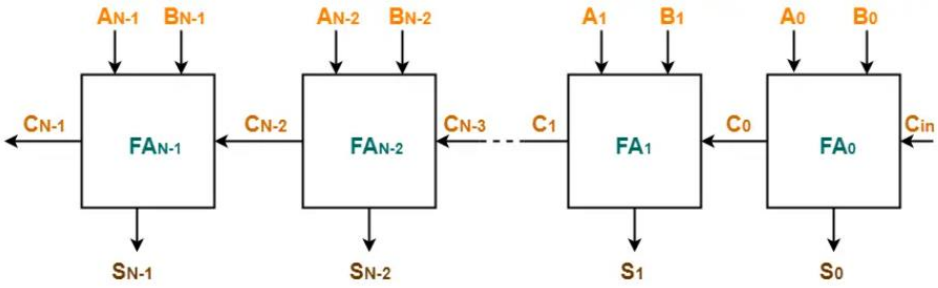


Fig 4.1 Ripple Carry Adder pentru numere pe 32 bit , pentru N=32

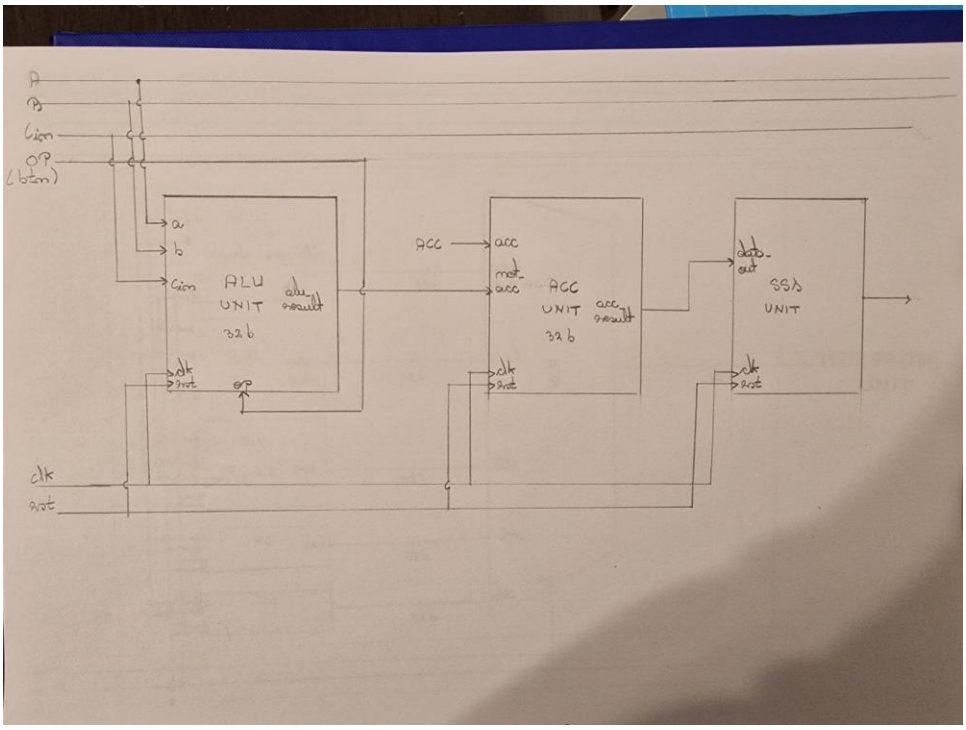

Fig. 4.2 Schema block a unitatii de calcul pe 32 bit

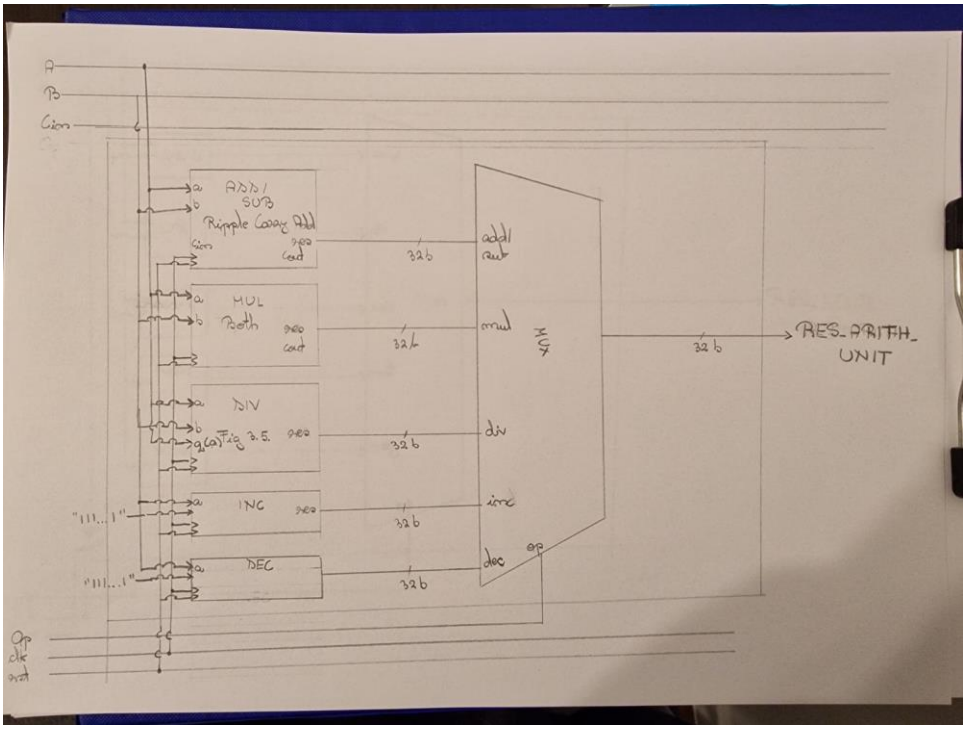

Fig. 4.3 Schema unitatii responsabile de operatiile aritmetice a ALU

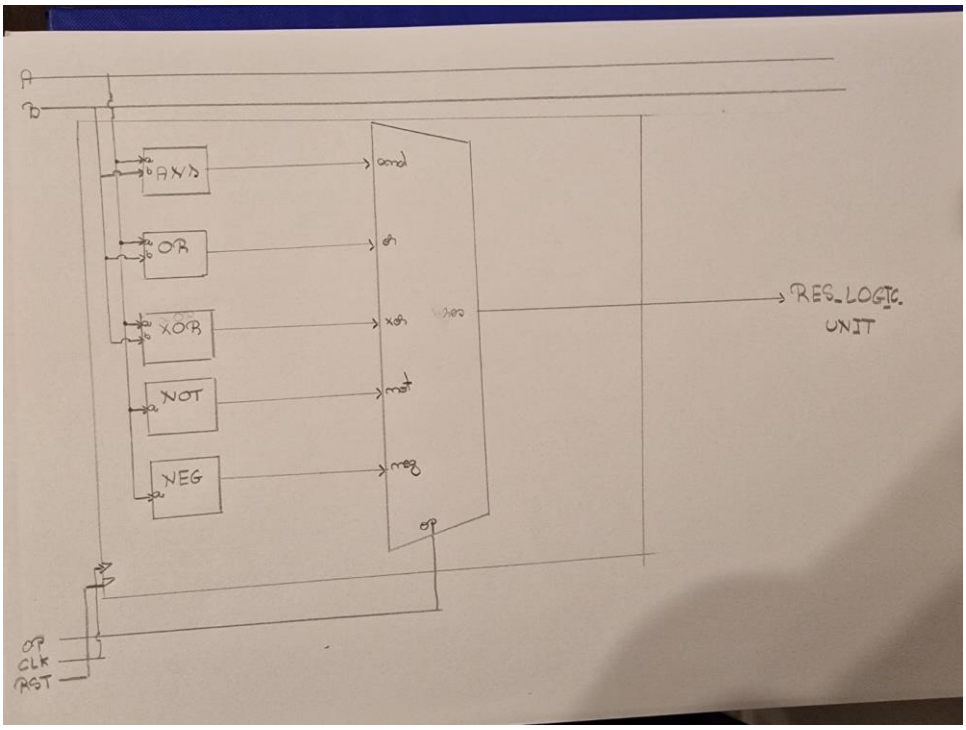


Fig. 4.4 Schema unitatii responsabile de operatiile logice din ALU

## 8.Resurse bibliografice

1. Florin Oniga, ” DE LA BIT LA PROCESOR. Introducere în arhitectura calculatoarelor”, 2019

2. D. A. Patterson, J. L. Hennessy, “Computer Organization and Design: The Hardware/Software Interface”, 5 th edition, ed. Morgan–Kaufmann, 2013.

3 ALU in VHDL. Online. Available: https://support.xilinx.com/s/question/0D52E00006hpLA4SAM/alu-in-vhdl?language=en_US

4. Dr. Malti Bansal Assistant Professor Department of Electronics and Communication Engineering Delhi Technological University , ”Implementation of 32-Bit Arithmetic Logic Unit on Xilinx using VHDL”. Online. Available: https://www.academia.edu/28277751/Implementation_of_32_Bit_Arithmetic_Logic_Unit_on_Xilinx_using_VHDL

5. Priyavrat Bhardwaj 1 , Aditya Anant Bansode2, International Journal of Innovative Research in Science, Engineering and Technology, Vol. 7, Issue 1, January 2018. Online. Available: http://www.ijirset.com/upload/2018/january/30_Design.pdf