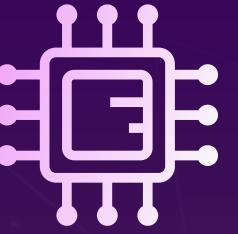


# 8-BIT ALU SIMULATION

Echipa:

- Iordăchescu Vlad-Alexandru
- Hondola Paul
- Golya Carmen-Mihaela
- Bardan Elena-Bianca



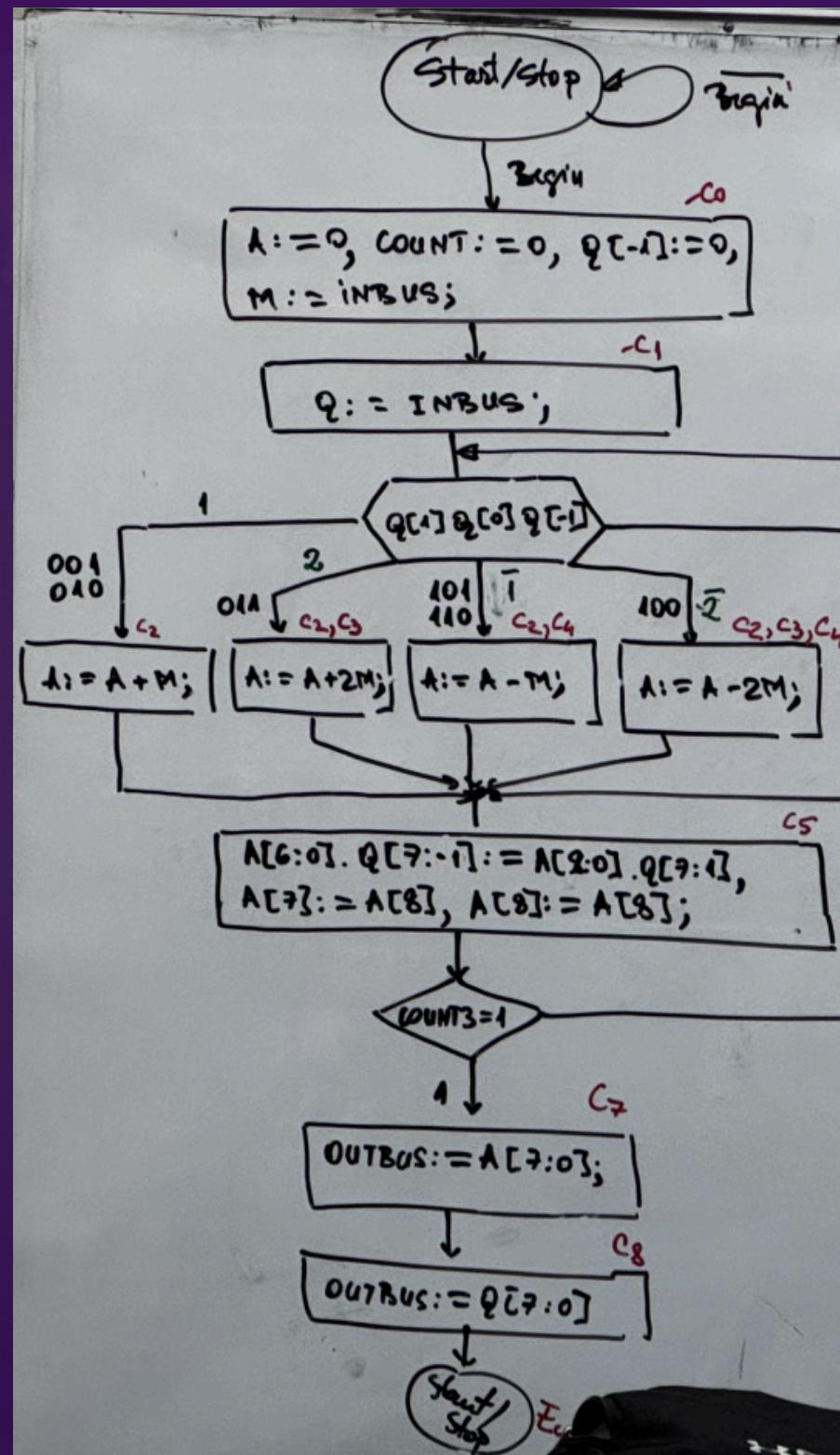
# ~OBIECTIVUL PROIECTULUI~

Proiectarea și simularea structurală a unei unități logice aritmetice (ALU) pe 8 biți, utilizând un limbaj de descriere hardware (HDL – Verilog sau VHDL), astfel încât să fie capabilă să efectueze patru operații aritmetice fundamentale – adunare, scădere, înmulțire și împărțire – folosind module hardware distincte și algoritmi standard de calcul implementați structural, fără cod comportamental.





# ~ALGORITMII IMPLEMENTAȚI~



## ◆ Adder/Subtractor

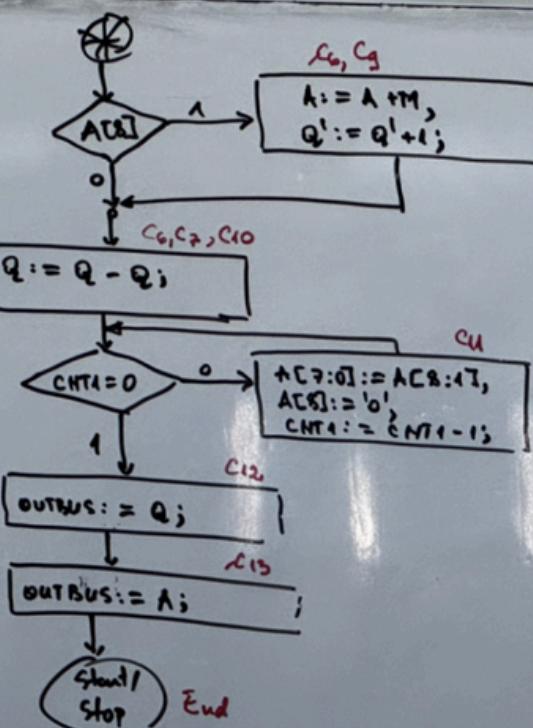
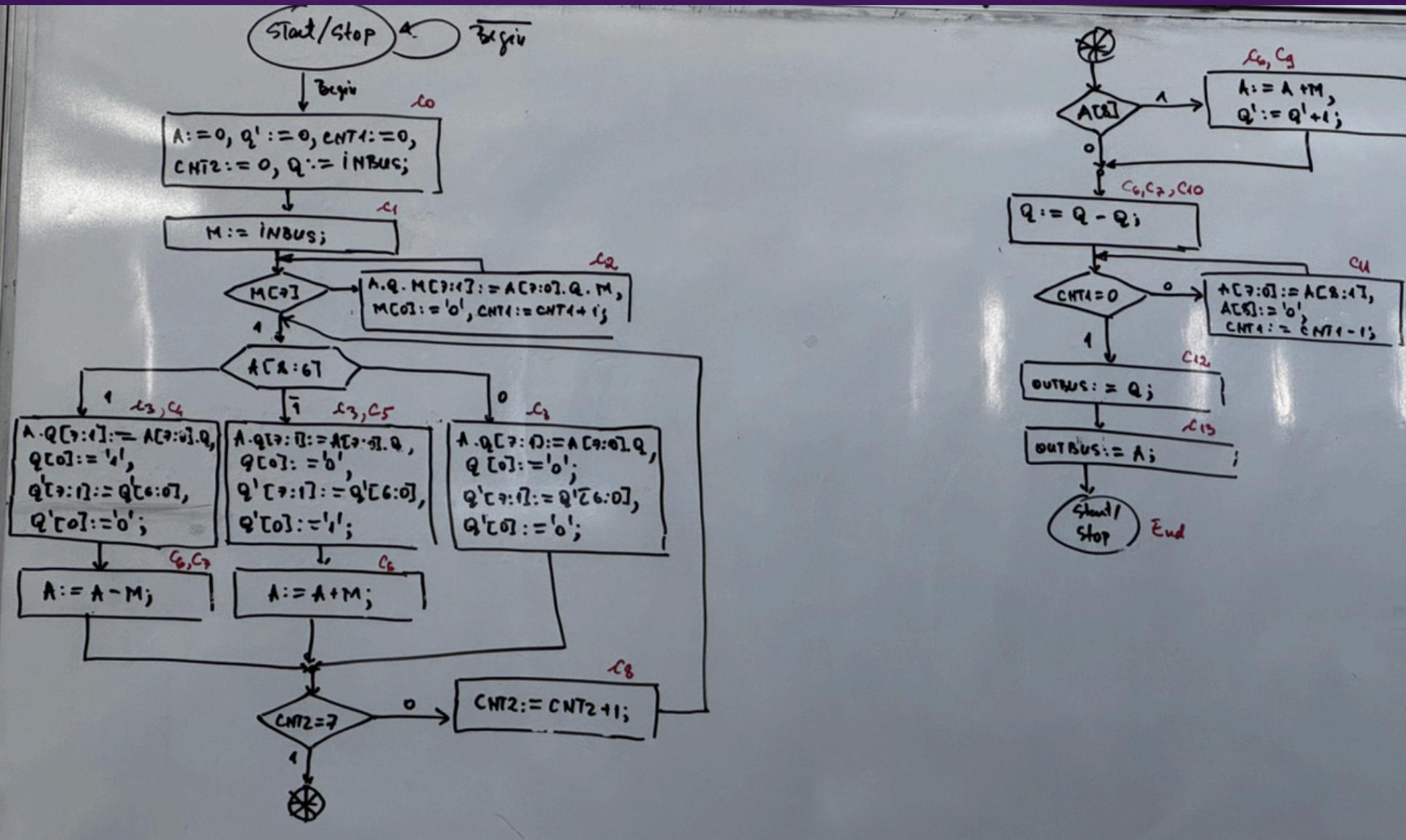
- Implementat structural folosind Ripple Carry Adder(RCA).
- Operația de scădere se realizează prin complementarea celui de-al doilea operand și adăugarea unei unități (operatie de tip 2's complement).
- RCA este simplu de implementat, dar lent pentru numere mari din cauza propagării transportului de la LSB la MSB.

## ◆ Înmulțire – Booth Radix-4 Algorithm

- Algoritm de înmulțire pentru numere cu semn (reprezentare în complement față de doi).
- Reduce numărul de operații aritmetice prin gruparea a câte două biți consecutivi și analizarea combinației  $Q[i+1], Q[i], Q[i-1]$ .
- Operațiile posibile: 0,  $\pm M$ ,  $\pm 2M$  (unde  $M$  este multiplicatorul).
- Necesită un registru  $A$  de 9 biți și un registru  $Q$  de 9 biți (inclusiv bitul  $Q[-1]$ ).



# ~ALGORITMII IMPLEMENTAȚI~



## ◆ Împărțire – SRT Radix-2 Algorithm

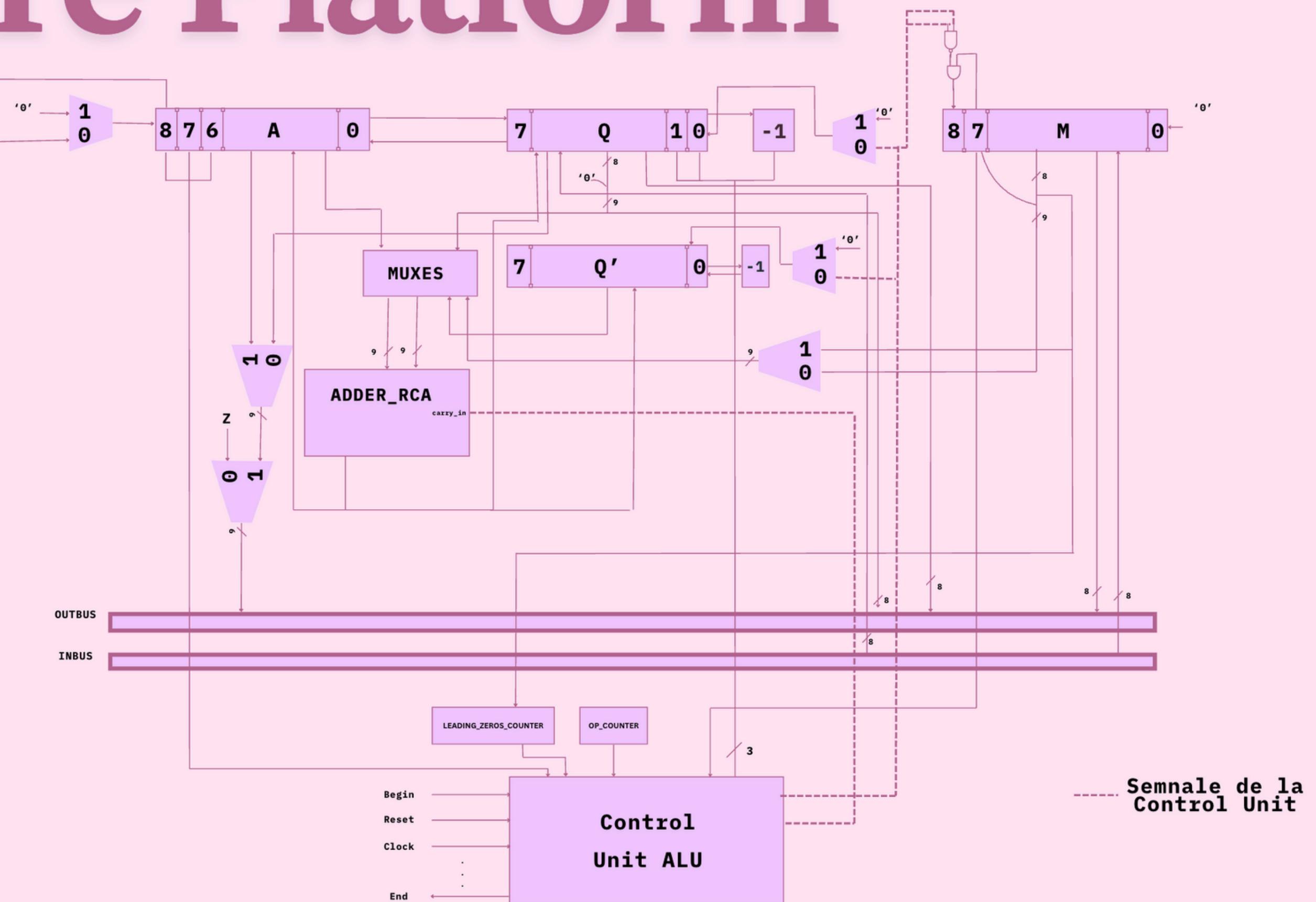
- Algoritm de diviziune pentru numere întregi fără semn.
- Utilizează o metodă bazată pe semnul bitilor superioiri din registrul A pentru a decide dacă se adună sau se scade M (divizorul).
- Folosește setul redundant de cifre în baza 2 pentru formarea cîrului.
- Necesită un registru A de 9 biți și Q de 8 biți; M este extins la 9 biți pentru a reprezenta  $2M$ .
- Algoritmul este eficient și potrivit pentru implementări datorită operațiilor simple de shift și adunare/scădere.

# Hardware Platform

A custom-designed ALU capable of addition, subtraction, multiplication with Radix-4 algorithm, and division with SRT-2 algorithm.

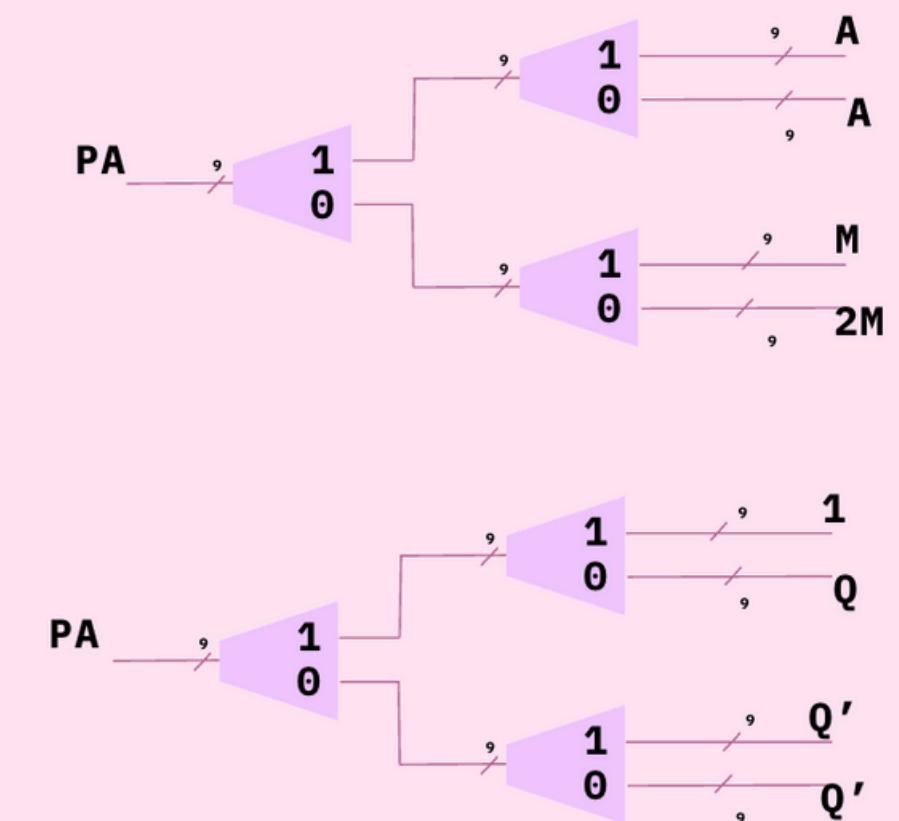
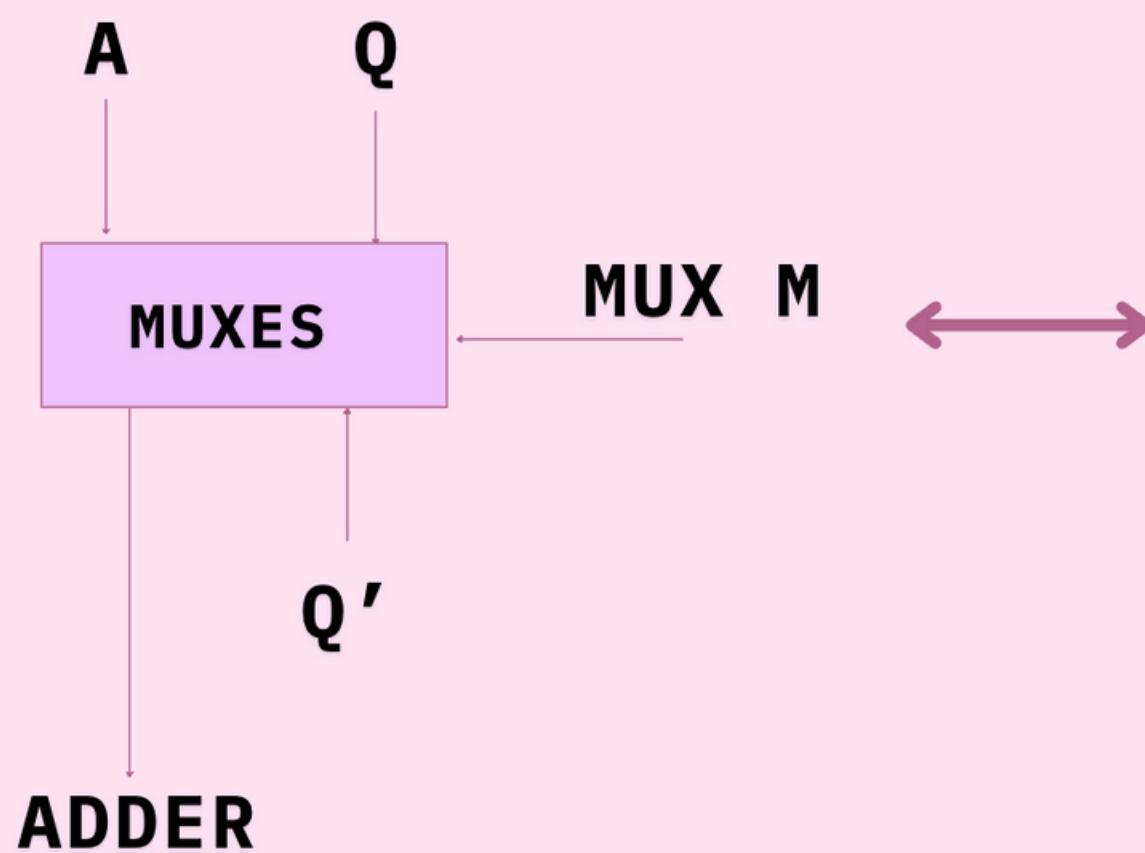
The platform features:

- A centralized Control Unit that manages the operations.
- RCA for arithmetic processing.
- Registers, multiplexers, logic gates.

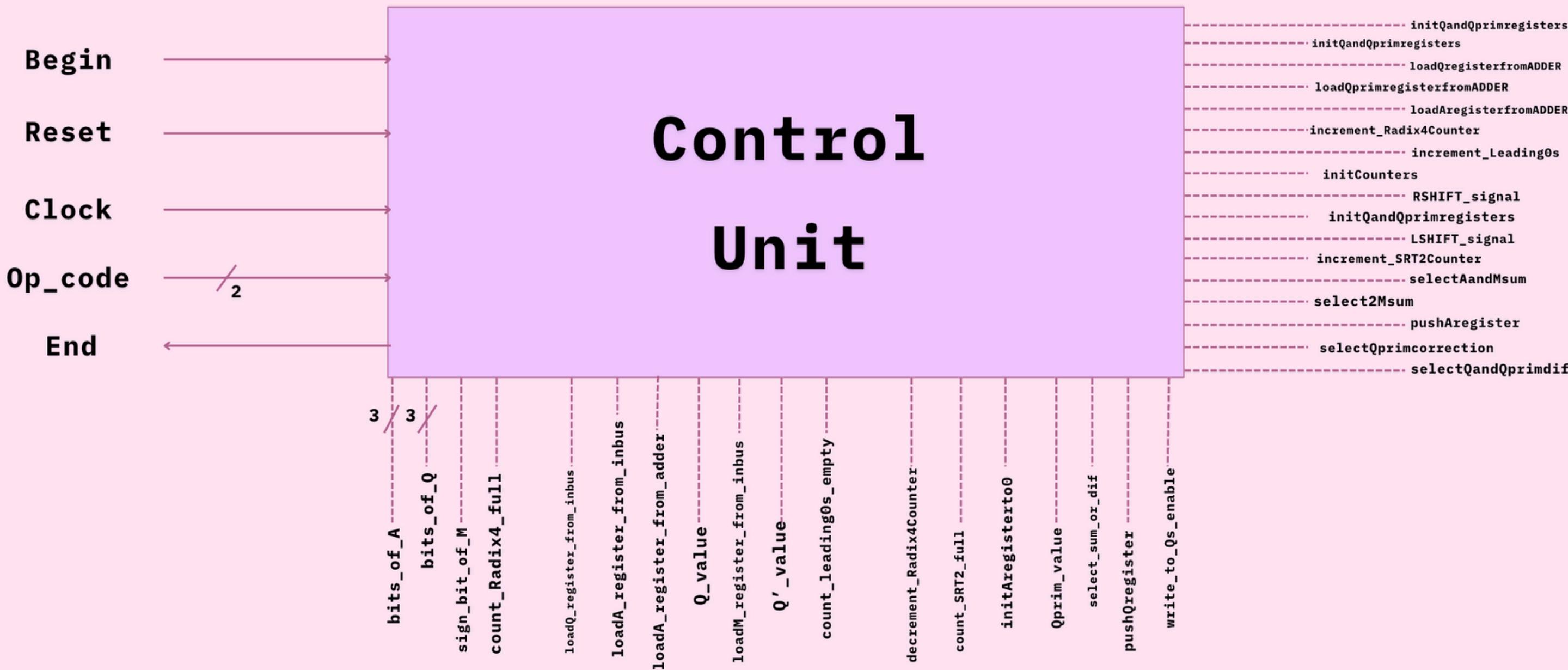


# Black-box explained

To improve clarity, we have separated this series of multipliers from the main schematic. Depending on the operation selected, the Control Unit sends signals to activate the appropriate inputs, enabling the corresponding multiplier functionality within the ALU.

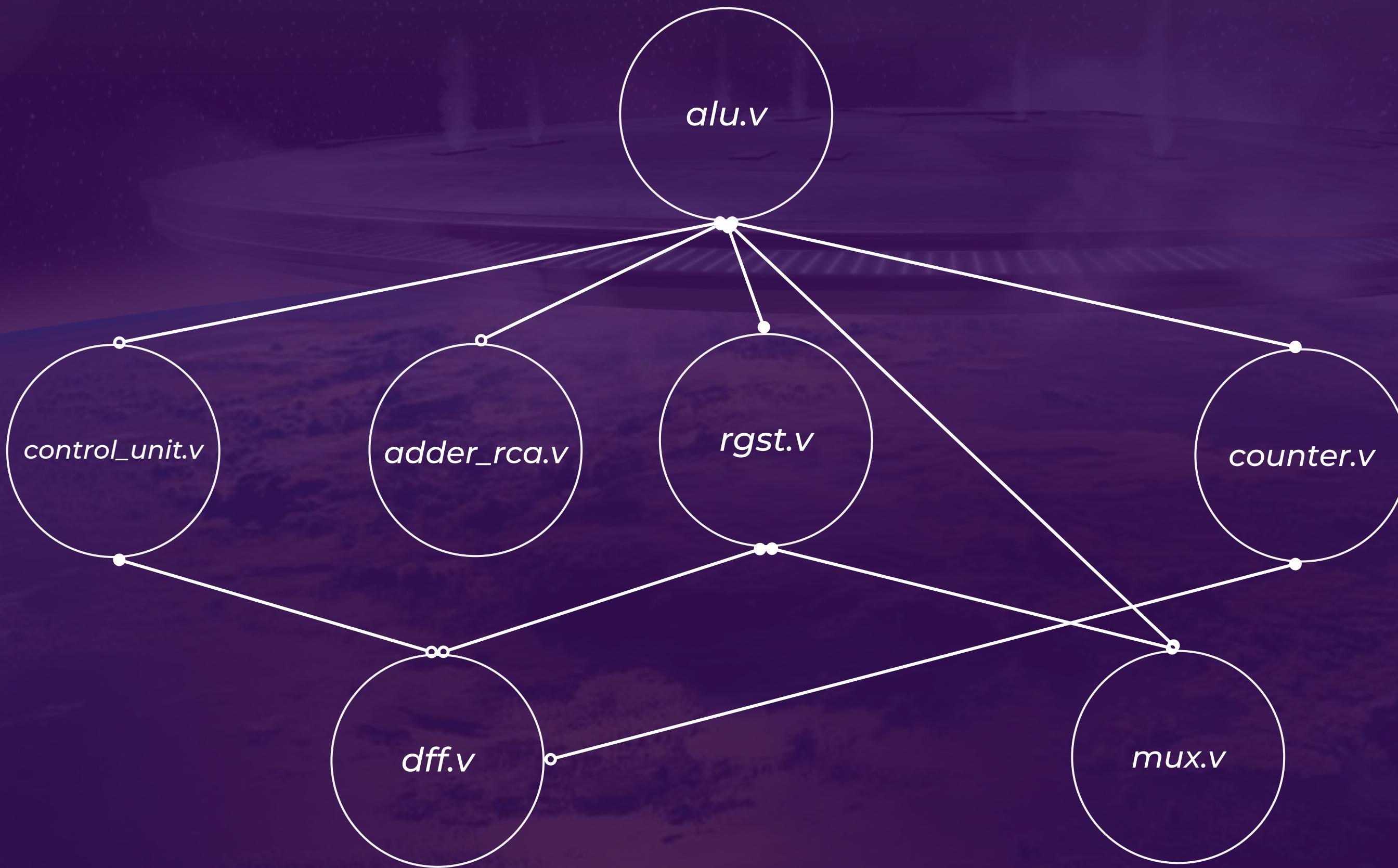


# Signals of the Control Unit





# ~MODULELE PRINCIPALE~





# ~HDL CODE LISTINGS~

## control\_unit.v

```
assign next_state[LOADA] = reset & act_state[IDLE] & BEGIN & ( ( ~op_code[1] ) | ( op_code[1] & op_code[0] ) ); // for add, sub, div
assign next_state[LOADQ] = reset & ((BEGIN & act_state[IDLE] & (op_code[1] & ~op_code[0])) // for mul
| (act_state[LOADA] & op_code[1] & op_code[0])); // for div
assign next_state[LOADM] = reset & ((act_state[LOADA] & ~op_code[1]) // for add, sub
| act_state[LOADQ]); // for for mul, div
```

## alu.v

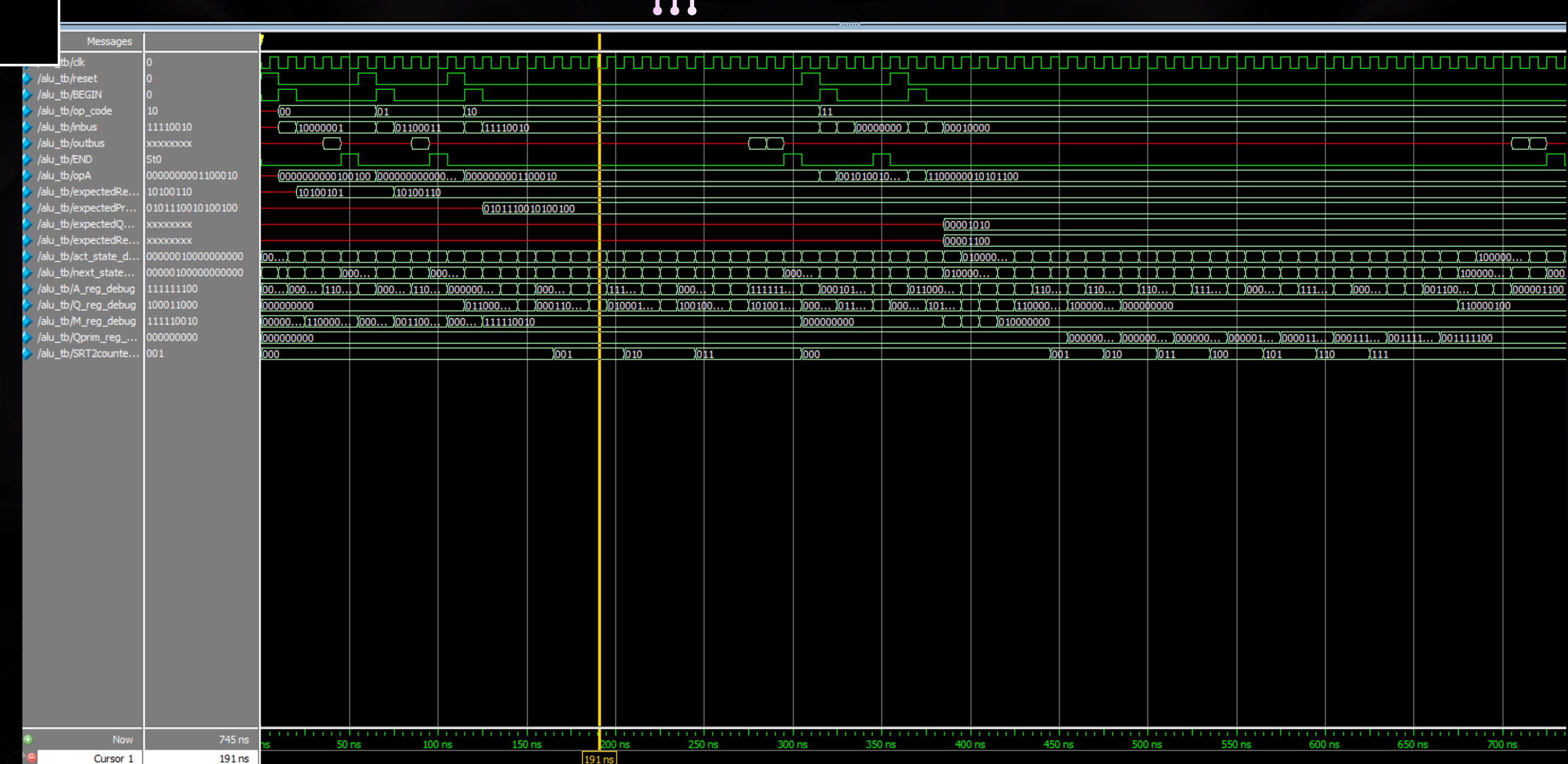
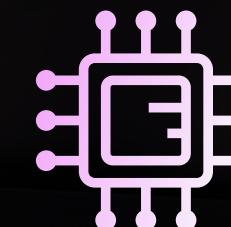
```
wire [8:0] operand_A, operand_B, adder_SUM;
wire adder_carry_in, adder_carry_out;

assign adder_carry_in = select_sum_or_dif;

adder_rca #(9) adder (
    .x(operand_A),
    .y(operand_B),
    .carry_in(addер_carry_in),
    .carry_out(addер_carry_out),
    .sum(addер_SUM)
);
```

- ALU: OPERATII (SUMA/DIFERENȚĂ) SELECTATE PE BAZA SEMNALULUI DE CONTROL. CODUL EVIDENȚIAZĂ LOGICA SIMPLĂ ȘI CLARĂ.
- CONTROL UNIT: IMPLEMENTARE STRUCTURALĂ. NEXT\_STATE DETERMINAT EXPLICIT DIN SEMNALELE DE CONTROL – UȘOR DE URMĂRIT ȘI TESTAT.

# ~SIMULARE~





# ~DISCUTIA IMPLEMENTARII~

În dezvoltarea ALU-ului pe 8 biți, au apărut dificultăți în faza inițială, când am încercat să implementăm module separate pentru fiecare operație. Această abordare modulară a dus la o complexitate crescută și probleme de sincronizare între module.

După testări, am decis să integrăm toate operațiile într-un singur modul ALU, ceea ce a simplificat designul, a redus interconexiunile și a îmbunătățit controlul semnalelor. Sincronizarea semnalelor de control, mai ales în gestionarea flip-flop-urilor de stare, a reprezentat o altă provocare majoră, necesitând o atenție sporită pentru a asigura funcționarea corectă în toate condițiile.

În final, integrarea într-un singur modul ALU a dus la un design eficient și stabil, capabil să realizeze toate operațiile cerute, cu o logică de control îmbunătățită și cu posibilitatea de extindere ulterioară.



VĂ MULTUMIM!