UNITATEA ARITMETICO-LOGICĂ

Contents

I. STUDIU BIBLIOGRAFIC

UAL realizează toate operațiile aritmetice și logice specificate prin instrucțiuni asupra datelor, rezultatele fiind salvate în memorie sau furnizate spre exterior.

Datele vor fi încărcate în registrele UCP, acestea fiind conectate la UAL prin căi de date. Instrucțiunile se decodifică și se transmit semnalele necesare pentru efectuarea operației respective. După execuția instrucțiunii UAL va transmite un semnal spre unitatea de comandă pentru a anunța finalul instrucțiunii. Rezultatele sunt stocate în registre.

Datele le voi reprezenta in 32 de biti.

A. ADUNARE, SCĂDERE ÎN C2

➢ Se aduna biţii corespunzători ai celor două numere și transportul de la rangul anterior

3 = 0011	-3 = 1101	-3 = 1101	3 = 0011
+ 2 = 0010	+-2 = 1110	+ 2 = 0010	+ -2 = 1110
- - - -	- - - -	- - - -	- - - -

For a full adder

A	B	Cin	SUM	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Inputs			Outputs	
A	B	$Borrow_{in}$	Diff	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

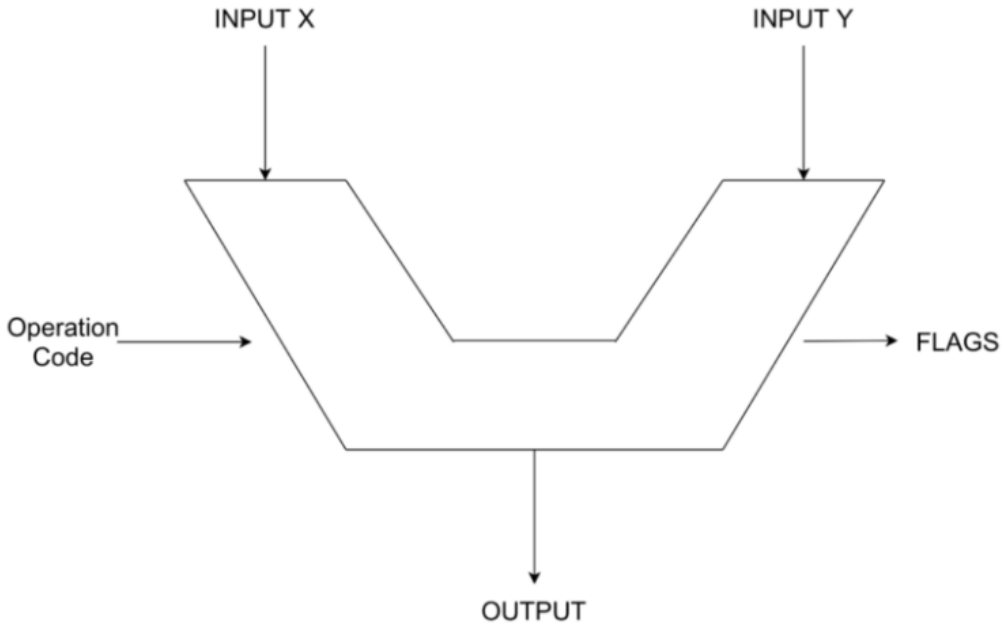
➢ Se obține negativul scăzătorul, apoi sunt adunate cele două numere

3-	3 = 0011	3+	3 = 0011	-3 -	-3 = 1101	-3 +	-3 = 1101
2	2 = 0010	(-2)	-2 = 1110	-2	-2 = 1110	2	2 = 0010
?	- - - -	1	1= 0001	?	- - - -	-1	-1 = 1111

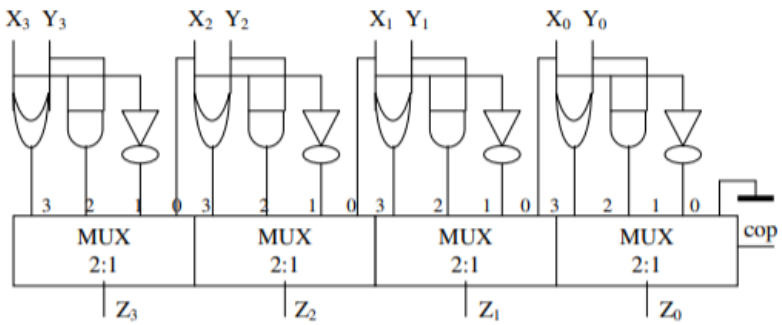
Voi folosi un sumator pe 32 biți cu Carry in și Carry out.

B. ȘI, SAU, NU

➢ Ne folosim de o unitate logică care va genera rezultatul oricărei instrucțiuni logice pe baza unui multiplexor ce operează pe codul operației.
➢ Și, sau, nu – cu porți logice.



C. ROTAȚIE STÂNGA/DREAPTA



➢ Mai sus avem schema pentru deplasarea la stânga cu o poziție. În acst desen pe poziția 0 s-a conectat 0 logic pentru că la deplasarea în stânga, în dreapta va veni un 0 logic.
➢ Cum am nevoie de rotire nu de deplasare, pe poziția 0 voi pune ultimul bit.
➢ La fel voi proceda și la rotirea spre dreapta doar ca desenul va fi pe dos.

D. ÎNMULȚITOR ȘI ÎMPĂRȚITOR

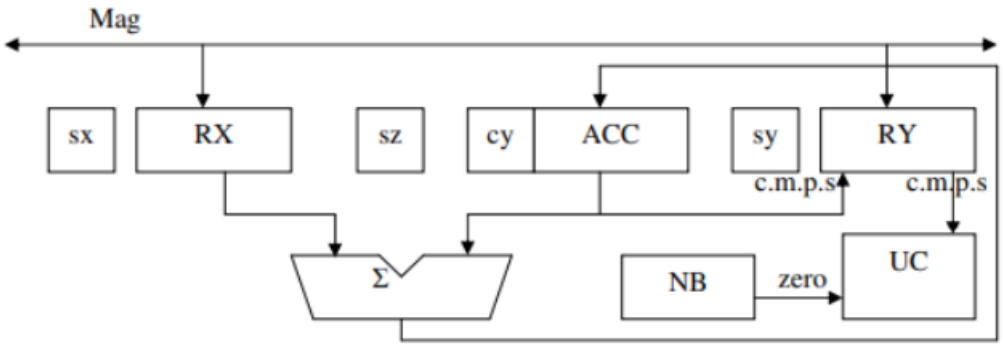


Fig.4.2.6 Unitate de inmultire pentru cod direct.

RX-modul deînmulțit
RY-modul înmulțitor
sx-bistabil pentru semn deînmulțit
sy-bistabil pentru semn înmulțitor
sz-bistabil pentru semnul rezultatului
cy-bistabil pentru memorarea transportului sumatorului
ACC-registru acumulator pentru păstrarea rezultatelor parțiale. La final conține cei mai semnificativi biți pentru modulul produsului
NB-numărător ce contorizează pașii de înmulțire (număr pași=număr biți din modulul operanzilor)
UC-unitatea de comandă
Inițial ACC=0, iar sx, RX, sy și RY se încarcă cu valorile celor doi operanzi.
Registrele cy, ACC si RY sunt registre cu deplasare, informația se poate deplasa cu o poziție spre dreapta, bitul cel mai puțin semnificativ va trece din ACC în RY.
La fiecare pas, se testează bitul curent al înmulțitorului ce e adus prin deplasări în poziția celui mai puțin semnificativ bit din RY.

- Dacă e 1 se adună RX cu ACC, această sumă fiind salvată în ACC, iar apoi se deplasează cu o poziție spre dreapta CY, ACC și RY.
- Dacă e 0 se execută doar deplasarea

Numaratorul de biti este initializat la inceputul operatiei cu valoarea n-1 (numarul de biti pentru reprezentarea modulului fiecarui operand) si este decrementat la fiecare ACC RY NB UC sx RX sz cy sy 6 pas. Când acesta ajunge în zero, anunță unitatea de comanda prin activarea unui semnal (zero) încheierea operației de înmultire. Rezultatul este obținut pe lungime dublă în ACC, RY.

prezentata in figura 7

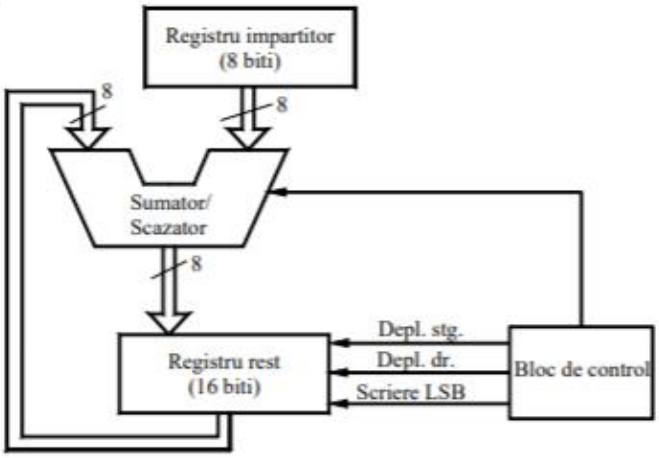


Fig. 8 – Structura hardware pentru algoritmul de impartire cu refacerea restului

Registrul rest = 2 * lungime operanzi, primul bit e bit de semn
Jumătatea superioară conține deîmpărțitul, jumătatea inferioară se încarcă inițial cu 0, iar la final, prima jumătate va conține restul și a doua jumătate câtul.
Împărțitorul este stocat pe toată perioada operației în registrul împărțitor.
Blocul sumator/scazator face diferența sau suma între jumătatea superioară a registrului rest și registrul împărțitor.
Registrul rest realizează deplasări spre stânga (cmsb va fi 0/1 în funcție de semnul scăderii dintre jumătatea superioară a reg. rest și reg. împărțitor).
Jumătatea superioară a reg. rest poate realiza și deplasări la drepata (completare cu 0).
Etapele împărțirii:

- Se încarcă jumătatea superioară a reg. rest cu 0 și jumătatea inferioară cu deîmpărțitul.
- Împărțitorul se încarcă în registrul împărțitor.
- Registrul rest se deplasează spre stânga.
- Se scade împărțitorul din jumătatea superioară a reg. rest.
- Dacă bitul de semn e 1, cifra câtului e 0 (deîmpărțit mai mic decât împărțitorul (trebuie refăcut deimpartitul astfel incat se aduna impartitorul la rezultatul din jumatatea superioara a registrului rest, pentru a reface deimpartitul).
- Deplasăm spre stânga restul și se completează bitul cel mai puțin semnificativ cu 0. Dacă bitul de semn e 0 atunci cifra câtului este 1. Se deplasează registrul rest spre stânga și se completează bitul cel mai puțin semnificativ cu 1.
- Se repetă operațiile de n ori (n = lungimea operanzilor).
- Registrul rest va conține restul și câtul după cum am menționat mai sus.

II. PLANIFICARE PE SĂPTĂMÂNI

SĂPTĂMÂNA	TASK
SĂPTĂMÂNA 3	ADUNARE. SCĂDERE, ȘI, SAU, NU
SĂPTĂMÂNA 4	ROTAȚII ȘI ÎNMULȚIRE
SĂPTĂMÂNA 5	ÎMPĂRȚIRE
SĂPTĂMÂNA 6	VERIFICARE ȘI TESTARE
SĂPTĂMÂNA 7	PREDARE

III. PROPUNERE DE PROIECT

Voi respecta instructiunile mentionate in cerinta proiectului, acele instructiuni specificate mai sus. Operațiile pentru ALU le voi realiza pe 32 de biți, după cum am menționat mai sus.

IV. Analiza

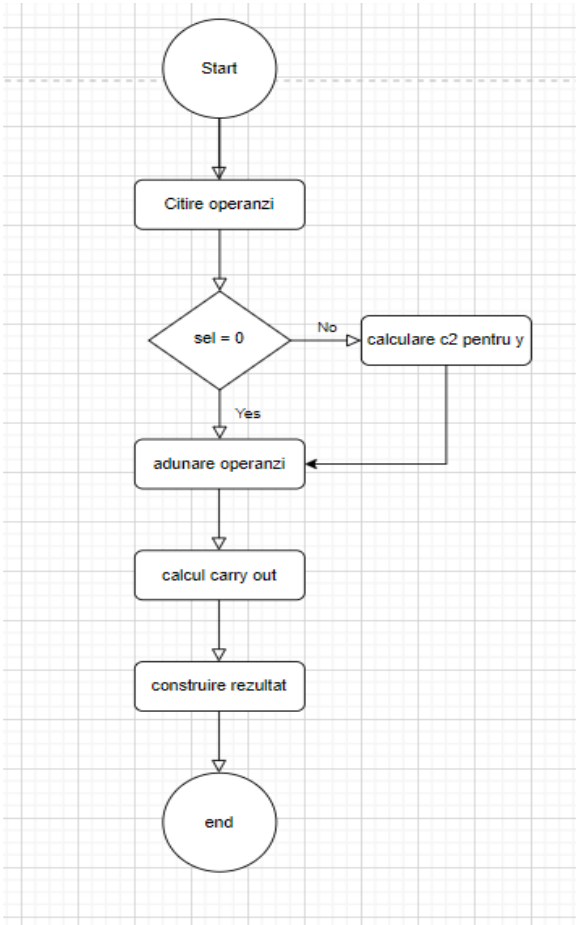
A. Adunarea

Este cea mai utilizată operație, toate celelalte operații bazându-se pe ea, deci implementarea eficientă a acesteia influențează direct toate celelalte operații.

O să avem un sumator pe 32 de biți, adică 32 de sumatoare pe un bit. Întârzierea va fi 32 * 3 * întârzierea pe poartă. Întârzierea pe poartă = 10 ns (TTL), de unde rezultă că întârzierea va fi de aproximativ 1000 ns.

Sumatorul pe un bit are 3 intrări (xi, yi și ci-1) și două ieșiri (si și ci).

- $S_i = x_i \oplus y_i \oplus C_i$
- $C_i = x_i y_i + (x_i \oplus y_i)\, C_{i-1}$



B. Scăderea

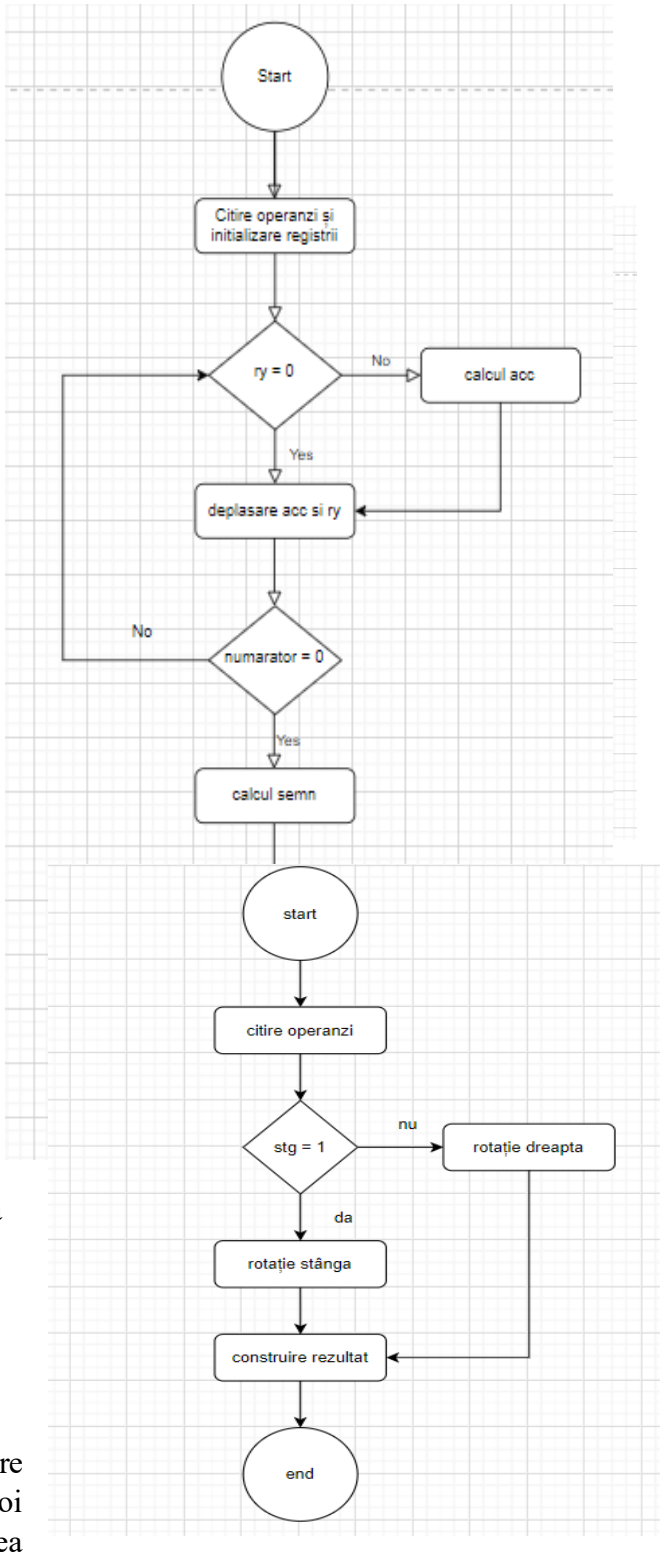
Scăderea rerezintă o adunare cu complementul față de 2 al celui de-al doilea operand. De aceea voi avea o singură componentă pentru adunare și scădere, cu un semnal special de selecție (sel): operația de adunare se va realiza pentru sel = 0, iar cea de scădere pentru sel = 1.

Pentru calcularea complementului față de 2 voi nega biții celui de-al doilea operand și voi adăuga 1.

C. Și, Sau, Nu

Pentru operațiile logice, voi avea o unitate logică care va conține un multiplexor, ce va activa operația necesară. Intrările vor fi x, y (32 de biți) și op (2 biți), iar ieșirea va fi rez (32 biți). Muxul va fi controlat de un semnal pe 2 biți (op) și va funcționa astfel:

- Pentru op = 00 => x and y
- Pentru op = 01 => x or y
- Pentru op = 10 => not x



D. Rotație stânga/dreapta

Operația de rotație se va realiza astfel:

- Voi avea un semnal stg, pentru stg = 1 voi realiza rotație la stânga și pentru stg = 0 voi realiza operația de rotație la dreapta.
- La stânga: voi memora bitul 31 și voi deplasa biții spre stânga cu o poziție pană la bitul 30 inclusiv. La final, bitul memorat va trece pe poziția 0.
- La dreapta: voi memora bitul 0 și voi deplasa biții spre dreapta cu o poziție, până la bitul 1 inclusiv. La final, bitul memorat va trece pe poziția 31.

E. Înmulțire

Operația de înmulțire o voi realiza cu adunare repetată. Voi shifta rezultatul parțial la dreapta și voi pune produsul parțial în același loc, pentru a putea folosi un sumator pe 32 de biți în loc de unul pe 64.

RX- pentru x
RY- pentru y
sx-bistabil pentru semn deînmulțit
sy-bistabil pentru semn înmulțitor
sz-bistabil pentru semnul rezultatului
cy-bistabil pentru memorarea transportului sumatorului

ACC-registru acumulator pentru păstrarea rezultatelor parțiale. La final conține cei mai semnificativi biți pentru modulul produsului

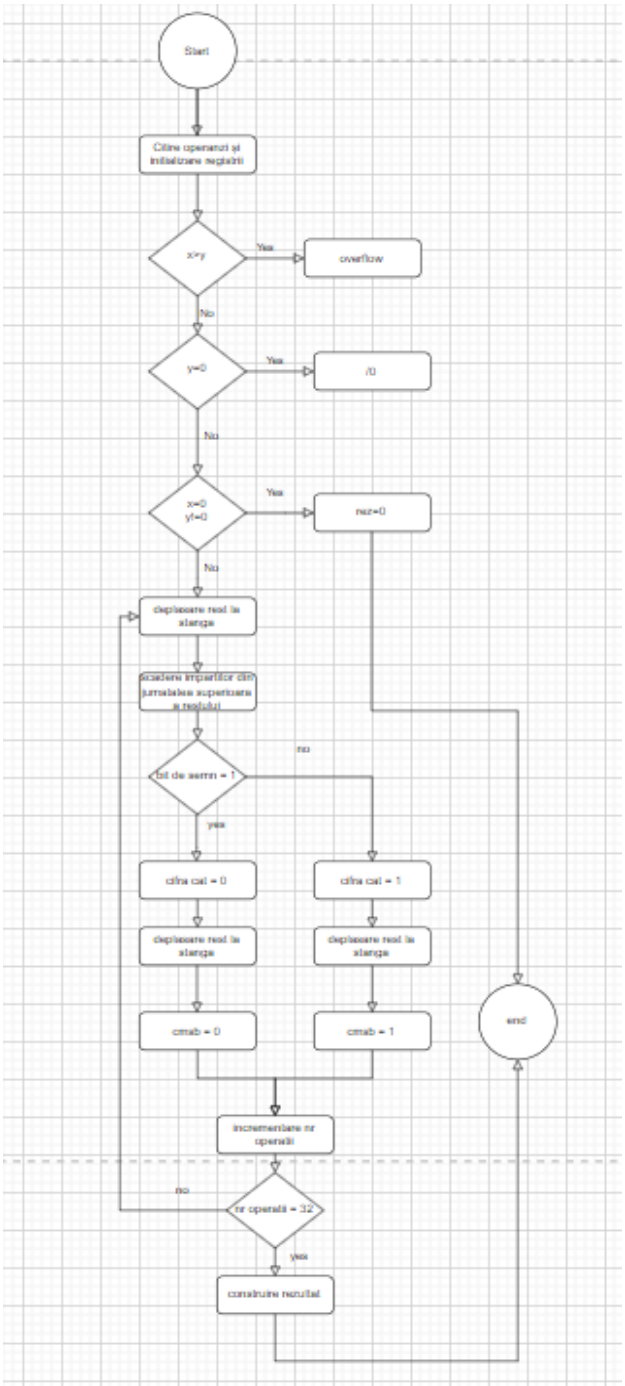
NB-numărător ce contorizează pașii de înmulțire (număr pași=număr biți din modulul operanzilor)

UC-unitatea de comandă

Pași:

- Inițial ACC=0, iar sx, RX, sy și RY se încarcă cu valorile celor doi operanzi.
- Registrele cy, ACC si RY sunt registre cu deplasare, informația se poate deplasa cu o poziție spre dreapta, bitul cel mai puțin semnificativ va trece din ACC în RY.
- La fiecare pas, se testează bitul curent al înmulțitorului ce e adus prin deplasări în poziția celui mai puțin semnificativ bit din RY.
- Dacă $RY_0 = 1$, ACC = RX + RY, iar apoi se deplasează cu o poziție spre dreapta CY, ACC și RY.
- Dacă $RY_0 = 0$ se execută doar deplasarea lui ACC și RY
- Număratorul de biți este initializat la inceputul operatiei cu valoarea n-1 (31) si este decrementat la fiecare ACC RY NB UC sx RX sz cy sy 6 pas. Când acesta ajunge în zero, anunță unitatea de comanda prin activarea unui semnal (zero) încheierea operației de înmultire. Rezultatul este obținut pe lungime dublă în ACC, RY.
- Semnul rezultatului se calculează astfel: $sz = sx \oplus sy$

F. Împărțire



Registrul rest = 2 * lungime operanzi, primul bit e bit de semn

Jumătatea superioară conține deîmpărțitul, jumătatea inferioară se încarcă inițial cu 0, iar la final, prima jumătate va conține restul și a doua jumătate câtul.

Împărțitorul este stocat pe toată perioada operației în registrul împărțitor.

Blocul sumator/scazator face diferența sau suma între jumătatea superioară a registrului rest și registrul împărțitor, în funcție de valoarea semnalului sel, după cum am menționat mai sus.

Registrul rest realizează deplasări spre stânga (cmsb va fi 0/1 în funcție de semnul scăderii dintre jumătatea superioară a reg. rest și reg. împărțitor).

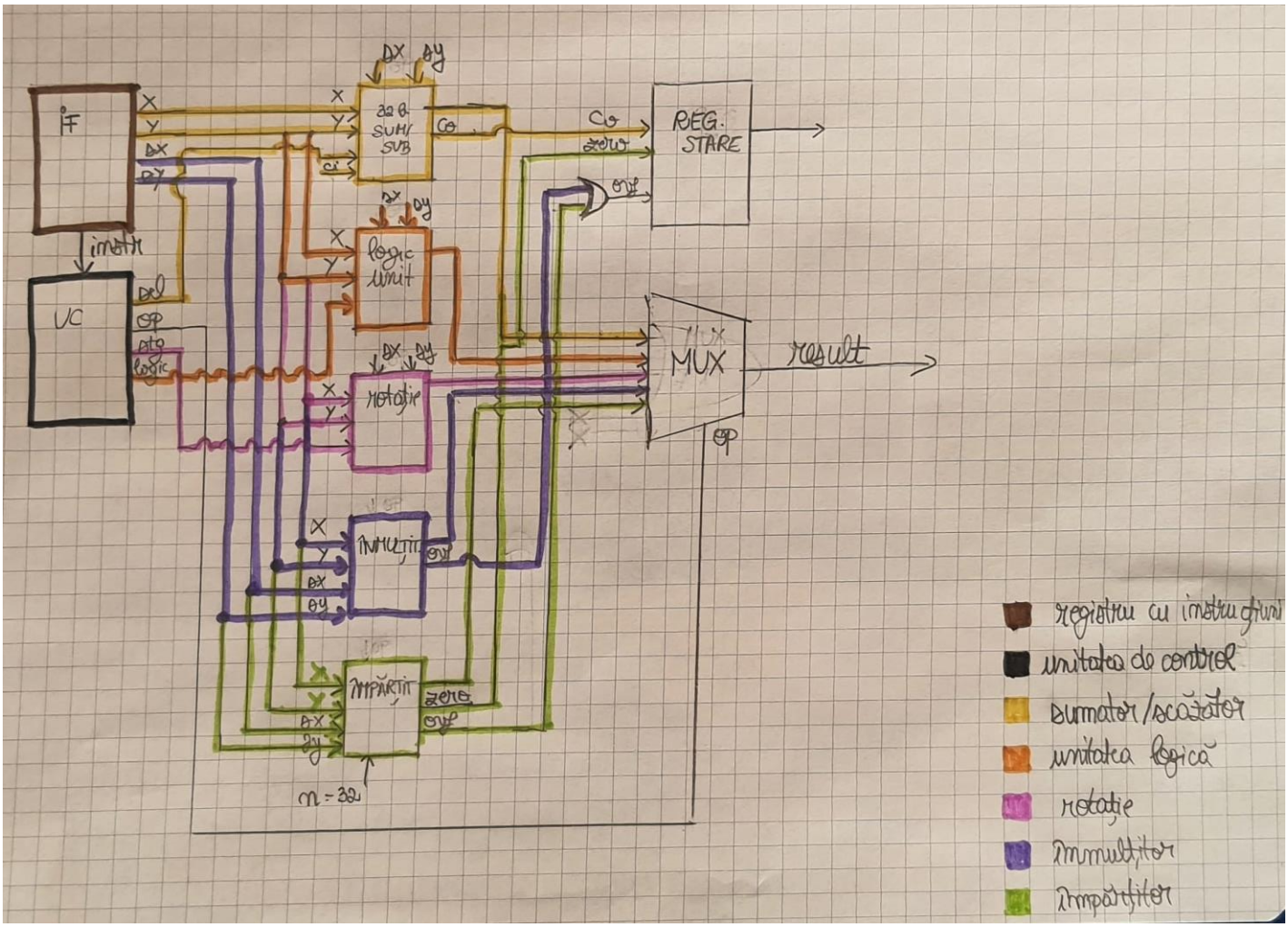
Jumătatea superioară a registrului rest poate realiza și deplasări la drepata (completare cu 0).

Etapele împărțirii:

- Se încarcă jumătatea superioară a registrului rest cu 0 și jumătatea inferioară cu deîmpărțitul.
- Împărțitorul se încarcă în registrul împărțitor.
- Dacă deîmpărțitul >= împărțitorul avem overflow (flag overflow = 1)
- Daca împărțitorul = 0 avem împărțire la 0 (flag 0 = 1)
- Dacă deîmpărțitul = 0 și împărțitorul != 0 avem rezultatul 0.
- Registrul rest se deplasează spre stânga.
- Se scade împărțitorul din jumătatea superioară a registrului rest.
- Dacă bitul de semn e 1, cifra câtului e 0 (deîmpărțit mai mic decât împărțitorul (trebuie refăcut deîmpartitul astfel incat se aduna impartitorul la rezultatul din jumatatea superioara a registrului rest, pentru a reface deimpartitul).
- Deplasăm spre stânga restul și se completează bitul cel mai puțin semnificativ cu 0. Dacă bitul de semn e 0 atunci cifra câtului este 1. Se deplasează registrul rest spre stânga și se completează bitul cel mai puțin semnificativ cu 1.
- Se repetă operațiile de n ori (n = 32).
- Registrul rest va conține restul și câtul după cum am menționat mai sus.

V. Design

A. Design ALU



În desen se pot observa legăturile dintre componente, respectiv cum interacționează între ele.

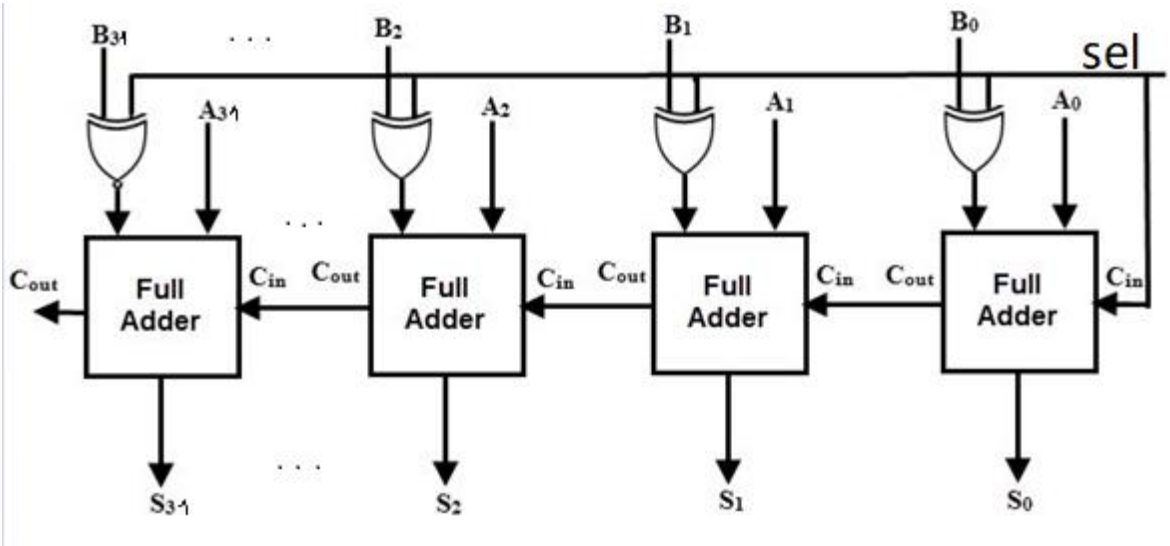
Registrul de instrucțiuni va conține un set de instrucțiuni ce se vor executa, va separa operanzii de instruncțiunea propriu-zisă și va trimite operanzii și semnul lor spre componentele reprezentative pentru fiecare operație din ALU. Codul instrucțiunii va fi trimis spre unitatea de control.

Unitatea de control va transmite spre componentele de calcul codul operațiilor, spre componenta ce realizează rotația semnalul stg, spre unitatea logica semnalul stg și spre sumatorul scăzător semnalul sel.

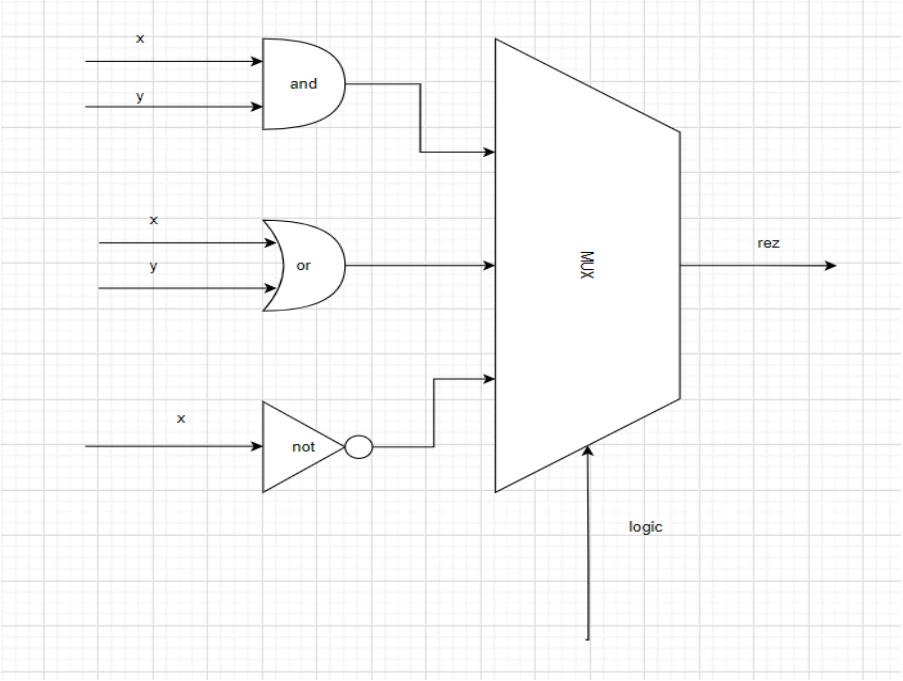
Registrul stare va primi semnalul de carry out, semnificând overflow pentru adunare, semnalul zero, semnificând împărțirea la 0 și semnalul overflow, pentru overflow la împărțire sau înmulțire.

Multiplexorul final va selecta, în funcție de codul operației, rezultatul corespunzător.

Sumator/Scăzător

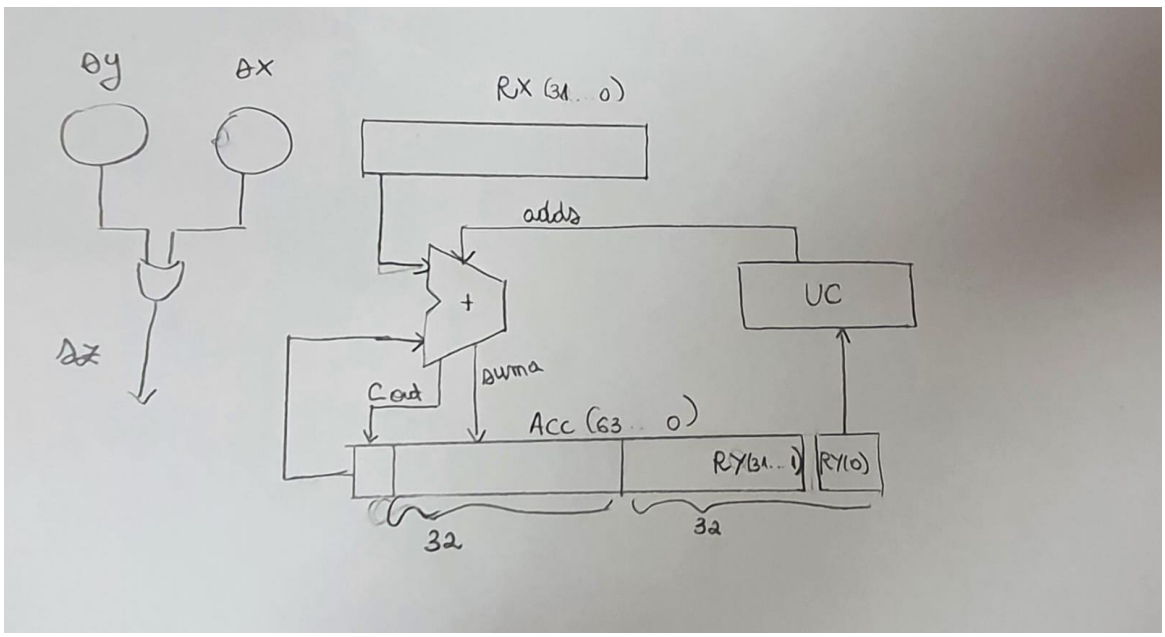


Acesta este alcătuit din 32 sumatoare pe un bit, iar dacă sel = 1 se va nega fiecare bit al celui de-al doilea opeand și se va adăuga 1 (porțile de sau exclusiv și sau exclusiv negat).



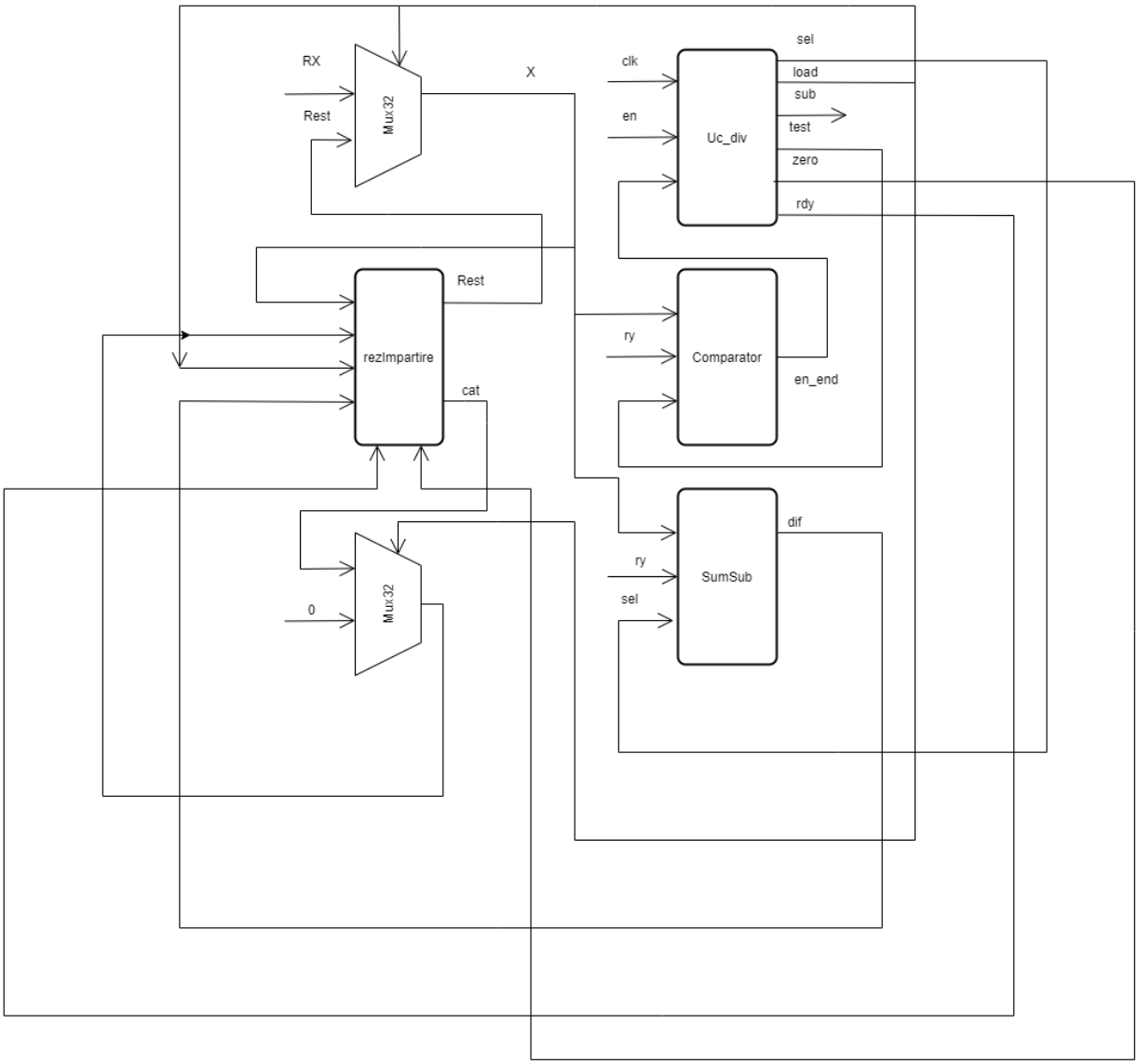
Unitatea logică

Aceasta este alcătuită din 3 porți logice și un mux care va selecta, în funcție de codul operației rezultatul corespunzător.



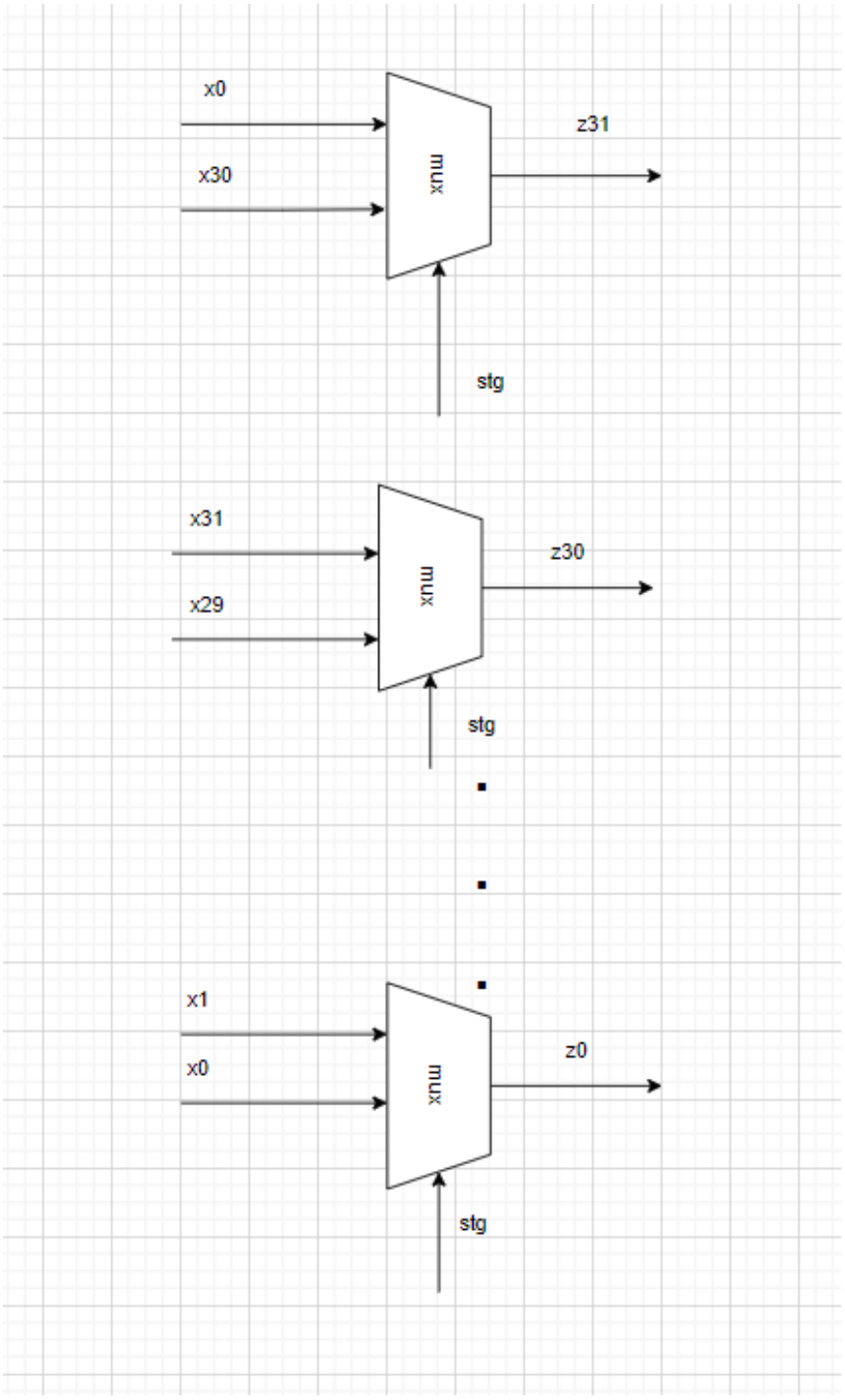
Înmulțitor

Această componenta conține registrele rx, ry pentru operanzi, un sumator, sx și sy pentru semnele operanzilor, un numărător și refistrul acc pentru rezultat și cy pentru semnul rezultatului.



Împărțitor

Acesta este alcătuit dintr-un sumator/scazător, un registru pentru împărțitor și unul pentru rest și se va folosi de unitatea de control.



Rotație

Această componenta conține 31 de multiplexoare care, în funcție de semnalul stg vor selecta bitul corespunzător pentru a efectua rotaâia la stânga, respectiv la dreapta.

B. Unitatea de control

OPERAȚIE	*INSTRUCȚIUNE*	*SEL*	*OP*	*STG*	*LOGIC*
+	0000	0	000	X	XX
-	0001	1	001	X	XX

AND	0010	X	010	X	00
OR	0011	X	010	X	01
NOT	0100	X	010	X	10
ROTATE RIGHT	0101	X	011	0	XX
ROTATE LEFT	0110	X	011	1	XX
*	0111	X	100	X	XX
/	1000	X	101	X	XX
SHIFT LEFT	1001	X	110	1	XX
SHIFT RIGHT	1010	X	110	0	XX
COMP	1010	X	111	X	XX
-	1011	X	XXX	X	XX
-	1100	X	XXX	X	XX
-	1110	X	XXX	X	XX
-	1111	X	XXX	X	XX

VI. Implementare

Aplicația a fost implementată într-o manieră structurală, operațiile fiind controlate de unitatea de control descrisă mai sus. Unitatea aritmetico logică este compusă dintr-o memorie de instrucțiuni, o componentă pentru decodificarea instrucțiunilor, o unitate de comandă, o unitate logică, un sumator scăzător, o componentă pentru realizarea operației de rotație, un înmulțitor, un împărțitor și un registru de stare. Rezultatul corect este selectat în funcție de opcode, iar valoarea semnalului care activează posibilitatea de a trece la intrucțiunea următoare este, de asemenea, obținută cu ajutorul opcode-ului.

```
clk<=not clk after 20 ns;

c1: InstrF port map(clk,rdy,reset, instruction,pc_o);
c2: ID port map(instruction,opcode,x,y);
c3: UC port map(opcode,sel,op1,stg,logic);
c4: LU port map(x,y,op1,logic,sx,sy,signL,rdyLU,rezLU);
c5: SumSub port map(x,y,sel,op1,Carry,rdySumSub,rezS);
c6: Rotate port map(y,stg,op1,rdyRotate,rezRotate);
c7: inmultitor port map(op1,x,sx,y,sy,clk,start,ovf,zero,rdyInmultitor,rezMul,signM);
c8: RegStare port map(zero,ovf,sign,Carry);
```

Figura 6.1 Componente

```
process(op1,rdyInmultitor,rdySumSub,rdyLU,rdyRotate,signs,signl,signm)
begin
   case op1 is
   when "000"=>start<='0';sz<=signs;sign<=signs;rdy<=rdySumSub;
   when "010"=>start<='0';sz<=signl;sign<=signl;rdy<=rdyLU;
   when "011"=>start<='0';rdy<=rdyRotate;
   when "100"=>start<='1';sz<=signm;sign<=signm;rdy<=rdyInmultitor;
   when others=>rdy<='1';
   end case;
end process;

process(op1,rezS,rezLu,rezRotate,rdy,rezMul)
begin
   if rdy='1' then
       case op1 is
       when "000"=>rezultat<=rezS;
       when "010"=>rezultat<=rezLu;
       when "011"=>rezultat<=rezRotate;
       when "100"=>rezultat<=rezMul;sz<=signm;sign<=signm;
       when others=>rezultat<="00000000000000000000000000000000000000000000000000000000000000000";
       end case;
   end if;
end process;
```

Figura 6.2 Selectare rezultat și activare IF

InstrF reprezintă memoria de instrucțiuni și conține, pe lângă instrucțiuni și modul în care se trece la instrucțiunea următoare în funcție de clk și enable.

```
type tRom is array(0 to 255) of STD_LOGIC_VECTOR(67 downto 0);
signal mem: tRom:=(

B"0000_00000000000000000000000000011_00000000000000000000000000000110",     --7+6
B"0001_00000000000000000000000001111_00000000000000000000000000001111",   --15-15
B"0010_00000000000000000000000001111_00000000000000000000000000000100",   --15 and 4
B"0011_00000000000000000000000001111_00000000000000000000000000010001",   --15 or 17
B"0100_00000000000000000000000010101_00000000000000000000000000010101",   -- not 21
B"0101_00000000000000000000000000000_00000000000000000000000000011111",   --rr 31
B"0110_00000000000000000000000000000_00000000000000000000000000011100",   --rl 28
B"0111_00000000000000000000000001000_00000000000000000000000000000101",    --8*5
B"1000_00000000000000000001111101000_00000000000000000000001111101000",    --2000/1000
others=>B"0000_00000000000000000000000000000_00000000000000000000000000000000");
```

Figura 6.3 Memoria de instrucțiuni

O instrucțiune este constituită, în această ordine din: 4 biți pentru operație (0000 – adunare, 0001 scădere ...), 32 de biți pentru primul operand și 32 de biți pentru al doilea operand (unde este necesar), în total având 68 de biți.

```
begin
    PC_o<=Q+'1';
    PCi<=Q+'1';
    process(clk,reset,en,PCi)   --PC
    begin
    if reset='1' or PCi="00000000000000000000000000001001" then
      Q<="00000000000000000000000000000000";
    elsif rising_edge(clk) then
       if en='1' then
            Q<=PCi;
      end if;
    end if;
    end process;

    process(Q)
    begin --ROM
         i<=mem(conv_integer(Q(4 downto 0)));
    end process;
```

Figura 6.4 Selectarea instrucțiunii următoare

ID reprezintă componenta care are rolul de a decodifica instrucțiunea în opcode, primul operand și al doilea operand după cum se vede și în figura 6.5.

```
opcode<=instr(67 downto 64);
a<=instr(63 downto 32);
b<=instr(31 downto 0);
```

Figura 6.5 Instruction Decoder

UC reprezintă unitatea de control și are rolul de a inițializa semnalele care controlează restul componentelor după cum se vede în procesul din figura 6.6.

```
process(I)
begin
    case I is
    when "0000" => sel<='0';op<="000";stg<='X';logic<="XX"; --adunare
    when "0001" => sel<='1';op<="000";stg<='X';logic<="XX"; --scadere
    when "0010" => sel<='X';op<="010";stg<='X';logic<="00"; --si
    when "0011" => sel<='X';op<="010";stg<='X';logic<="01"; --sau
    when "0100" => sel<='X';op<="010";stg<='X';logic<="10"; --not
    when "0101" => sel<='X';op<="011";stg<='0';logic<="XX"; --rotate right
    when "0110" => sel<='X';op<="011";stg<='1';logic<="XX"; --rotate left
    when "0111" => sel<='X';op<="100";stg<='X';logic<="XX"; --*
    when "1000" => sel<='X';op<="101";stg<='X';logic<="XX"; --/
    when "1001" => sel<='X';op<="110";stg<='1';logic<="XX"; --shift left
    when "1010" => sel<='X';op<="110";stg<='0';logic<="XX"; --shitf right
    when "1011" => sel<='X';op<="111";stg<='X';logic<="XX"; --comp
    when others => sel<='X';op<="XXX";stg<='X';logic<="XX";
    end case;
end process;
```

Figura 6.6 Unitatea de control

LU reprezintă componenta care realizează operațiile logice: negare, și, sau. Operațiile au fost realizate cu ajutorul unui proces care, în funcție de valoarea semnalului sel oferă rezultatul corespunzător. Pentru sel = 00 se realizează și, pentru sel=01 se realizează sau și pentru sel = 10 se realizează negația.

```
process(A,B,op,sA,sB,sel)
begin
    if op="010" then
        case sel is
        when "00" => rez(31 downto 0)<=A and B; sRez<=sA and sB;rdy<='1';
        when "01" => rez(31 downto 0)<=A or B; sRez<=sA or sB;rdy<='1';
        when "10" => rez(31)<=not A(31); sRez<=not sA;
        rez(30)<=not A(30);rez(29)<=not A(29);rez(28)<=not A(28);rez(27)<=not A(27);rez(26)<=not A(26);
        rez(25)<=not A(25);rez(24)<=not A(24); rez(23)<=not A(23); rez(22)<=not A(22);rez(21)<=not A(21);
        rez(20)<=not A(20);rez(19)<=not A(19);rez(18)<=not A(18);rez(17)<=not A(17);rez(16)<=not A(16);
        rez(15)<=not A(15);rez(14)<=not A(14);rez(13)<=not A(13); rez(12)<=not A(12);rez(11)<=not A(11);
        rez(10)<=not A(10); rez(9)<=not A(9); rez(8)<=not A(8);rez(7)<=not A(7); rez(6)<=not A(6);
        rez(5)<=not A(5); rez(4)<=not A(4);rez(3)<=not A(3);rez(2)<=not A(2);rez(1)<=not A(1); rez(0)<=not A(0);
        rdy<='1';
        when others => rez<=X"FFFFFFFFFFFFFFFF";rdy<='0';
        end case;
        rez(63 downto 32)<="00000000000000000000000000000000";
    end if;
end process;
```

Figura 6.7 Unitatea logică

SumSub reprezintă un sumator scăzător pe 32 de biți realizat prin cascadarea a 32 de sumatoare scăzătoare pe un bit care, pentru sel = 0 realizeaza adunarea și pentrusel = 1 scăderea.

```
process(sel,A,B,ci,op)
begin
    if op="000" then
        if sel='0' then
            S<=A xor B xor ci;
            Co<= (A and B) or ((A and ci) xor (B and ci));
        else
           S<=A xor B xor ci;
           Co<=((not A) and (B or Ci)) or (B and Ci);
        end if;
    end if;

end process;
```

Figura 6.8 Sumator – Scăzător pe un bit

Rotate reprezintă componenta care realizează rotația prin intermediul a 32 de multiplexoare pe un bit care, în funcție de valoarea semnalului stg aleg un anumit bit

dintre cele două intrări și îl scriu pe ieșire. Astfel, pentru stg = 1 se va realiza rotația la stânga și pentru stg = 0 rotația la dreapta.

```
begin
process(A,B,s)
begin
        case s is
          when '0'=>
          rez<=A;
          when '1'=>
          rez<=B;
          when others=>rez<='X';
        end case;

end process;
```

Figura 6.9 Multiplexor folosit pentru rotație

```
c1: Mux1bitS port map(en,X(0),X(30),stg,rez(31));
```

Figura 6.10 Exemplu de instanțiere a multiplexorului folosită în Rotate

RegStare conține flaguri penrtu overflow, semn, zero și carry, acestea fiind inițializate cu ieșirile de la componentele ALU în funcție de operația realizată.

```
entity RegStare is
Port ( zero: in std_logic;
       overflow: in std_logic;
       sign: in std_logic;
       Carry: in std_logic
 );
end RegStare;
```

Înmulțitorul a fost realizat cu mai multe componente: sumator scăzător, unitate de control pentru înmulțire (fig 6.11) , registru pe 32 de biți (fig 6.12), rezÎnmulțire (fig 6.13) și flags (fig 6.14). Unitatea de control generează valori pentru semnalele de intrare necesare componentelor înmulțitorului, acestea depinzând de etapa curentă a înmulțirii (start, initreg, verifb0, adder sau shifter) și de valoarea lui Q care se decrementează la fiecare shiftare realizată. Acesta a fost inițializat cu valoarea 32 pentru a realiza înmulțirea în conformitate cu algoritmul descris într-un capitol anterior. Registrul este folosit pentru stocarea primului operand, rezÎnmulțire generează rezultatul final după aplicare a repetată a unor serii de shiftări și/sau adunări în funcție de valoarea ultimului bit al celui de al doilea operand, după cum am menționat mai sus. Componenta flags este folosită pentru a returna valorile semnalelor ce vor fi folosite în registrul de stare menționat mai sus.

```
c1: SumSub port map(RA,RB,'0',"000",Carry_out,r,Add_out);
c2: uc_mul port map(op,clk,lsb,starts,LOADs,SHIFTs,ADDs,rdyS);  --corectam cu start
c3: registru port map(RX,LOADs,clk,RA);
c4: rezInmultire port map(clk,RY,ADD_out(31 downto 0),CARRY_out,ADDs,LOADs,SHIFTs,lsb,RB,ACCs);
c6: flags port map(ACCs(31 downto 0),zero1,ovf1);
c7: flags port map(ACCs(63 downto 32),zero2,ovf);
```

Figura 6.11 a Componentele înmulțitorului

```
process(clk)
begin
if rising_edge(clk) then
   case s is
   when start => if enable='1' and op ="100" then
                 s<=initreg;
                 else s<=start;
                 end if;

   when initreg => s<=verifb0; zero<='0';
   when verifb0 => if b0='1' then --si adunam si shiftam
                       s<=adder;
                   else
                      s<=shifter;  --doar shiftam
                   end if;
   when adder => s<=shifter;
   when shifter => if Q="00000" then  Q<="11111"; s<=start;zero<='1';
                   else Q<=Q-'1'; s<=verifb0;  zero<='0';  --nu s-au executat 31 de repetari incrementam contorul
                   end if;
   end case;
end if;
sum<='1' when s=adder else '0';
shift<='1' when s=shifter else '0';
load<='1' when s=initreg else '0';
```

Figura 6.11

```
process(load)
begin
  if rising_edge(load) then
     if load = '1' then
        Q<=D;
     end if;
  end if;
end process;
```

Figura 6.12

```
process(clk)
begin
if rising_edge(clk) then
   if ld ='1' then
       temp_register (64 downto 32) <= (others => '0');
       temp_register(31 downto 0) <= B;    --incarcam B in registrul ACC
   end if;
   if add='1' then
       temp_Add<='1';
   end if;

   if shift='1' then --shiftam
         if temp_add='1' then
         --stocam rezultatul sumei si shiftam la dreapta
           temp_add<='0';
           temp_register<='0' & cout & suma & temp_register(31 downto 1);

         else
            --doar shiftam
            temp_register<='0' & temp_register(64 downto 1);
         end if;
    end if;
   end if;
rez_o<=temp_register(63 downto 0);
lsb<=temp_register(0);
rb<=temp_register(63 downto 32);
```

Figura 6.13

```
process(a)
begin
if A="00000000000000000000000000000000" then
  z<='1';
else z<='0';
end if;
o<=A(0) or A(1)or A(2) or A(3)or A(3)or A(4)or A(5)or A(6)or A(7)or A(8)or A(9)or A(10)or A(11)or A(12)or A(13)or A(14)or
end process;
zero<=z;
ovf<=o;
```

Figura 6.14

Împărțitorul a fost realizat cu ajutorul unui sumator scăzător care a fost folosit pentru a calcula restul, scăzându-se din primul număr al doilea, la fiecare activare a semnalului sub generat de unitatea de control. Algoritmul continuă cât timp primul operand este mai mare sau egal cu al doilea. Calcularea câtului a fost realizată prin intermediul incrementării unui semnal la fiecare scădere realizată. Unitatea de comandă pentru împărțire generează semnale pentru fiecare etapă a algoritmului (start, initreg, substracter, verif și ends). Componenta rezImpărțire realizează calculul final, adică calculul câtului și al restului. Componenta mux_32 este folosită pentru selectarea intrării în rezImparire (0 pentru load = 1 sau câtul anterior pentru load = 0) sau pentru selectarea intrării rx a sumatorului scăzător (rx pentru load = 1 sau rest pentru load = 0).

Codul componentelor poate fi observat în figurile de mai jos.

```
entity div is
  Port ( RX: in std_logic_vector(31 downto 0);
  sx: in std_logic;
  RY: in std_logic_vector(31 downto 0);
  sy: in std_logic;
  clk: in std_logic;
  start: in std_logic;
  ovf: out std_logic;
  zero: out std_logic;
  rdy: out std_logic;
  rez: out std_logic_vector(63 downto 0);
  sz: out std_logic
   );
end div;

  c4: mux_32 port map(RX,restS,rdys,load,X);  --selectam A la inceput, iar in rest diferenta pentru intrare in re
  c5: comparator port map(X,RY,test,en_end);
  c1: UC_div port map(clks,start,en_end,load,sub,test,zzero,sel,srdy,op);
  c2: rezImpartire port map(clks,cats2,X,RY,dif(31 downto 0),sub,carry,load,cat,rests);
  c3: SumSub port map(X,RY,sel,op,carry,rdys,dif);
  c7: mux_32 port map("00000000000000000000000000000000",cat,'1',load,cats2);
```

Figura 6.15 – Împărțitorul

```
if rising_edge(clk) then
   if en_end = '1' then s<=endS; end if;
   case s is
   when start => if enable='1' then s<=initreg; end if;
   when initreg => s<=verif;
   when verif => if en_end = '1' then s<=endS; else s<=substracter;end if;--
   when substracter => if en_end = '1' then s<=endS; else s<=verif; end if;
   when endS => r<='1';
   end case;
end if;
end process;
zero<='1' when s=start else '0';
load<='1' when s=start else '0';
sub<='1'  when s=substracter else '0';
sel<='1' when s=substracter else 'U';
op<="000" when s=substracter else "111";
test<='1' when s=verif else '0';
rdy<='1' when s=endS else '0';
end Behavioral;
```

Figura 6.16 – Unitatea de control

```
begin
process(clk,sub)
begin
if rising_edge(clk) then
   if ld ='1' then
       --prima jumatate va fi catul si ultima restul
       temp_register(31 downto 0) <= A;    --restul
   end if;
   if sub='1' then
       temp_register(31 downto 0)<=dif; --diferenta
       rb<=cont+1;
   else rb<=cont; --nu mai crestem numarul de operatii efectuate
   end if;

end if;
end process;

rez_o<=temp_register(31 downto 0);
```

Figura 6.17 – RezImpartire

```
begin
process(A,B,load)
begin
        if load='0' then rez<=B;
        else
        rez<=A;
        end if;

end process;
```

Figura 6.18 – Mux_32

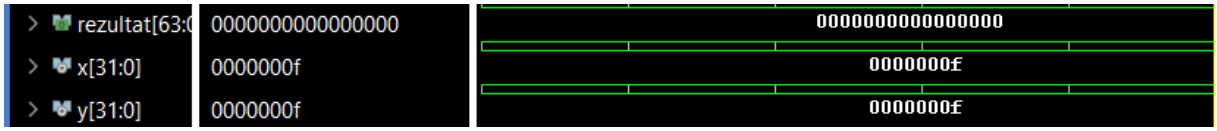
VII. Testare și validare

Exemple pentru adunare și scădere:

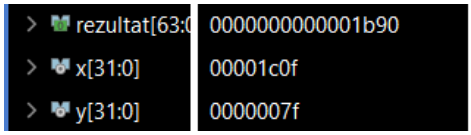
3+6 = 9

16899 + 262 = 17161

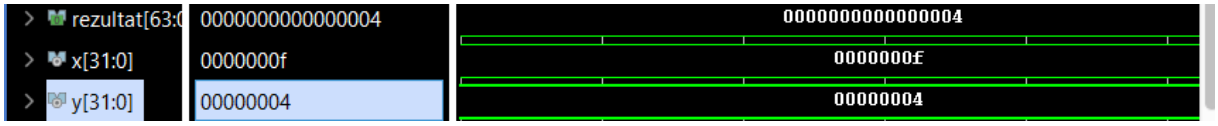
15 - 15 = 0



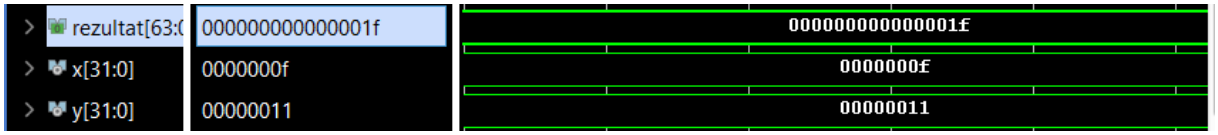
7183 – 127 = 7056



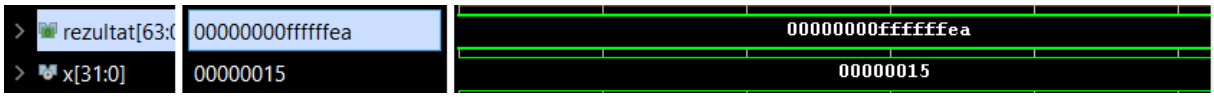
Exemple pentru LU:



15 and 4 = 4

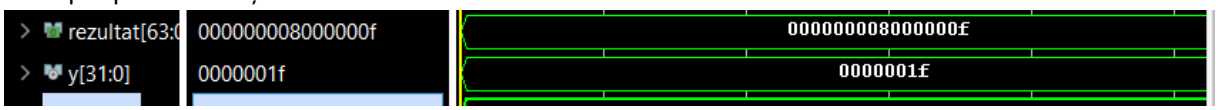


15 or 17 = 31

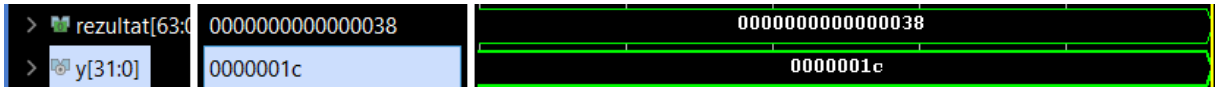


Not 15 = 4.294.967.274

Exemple pentru rotații

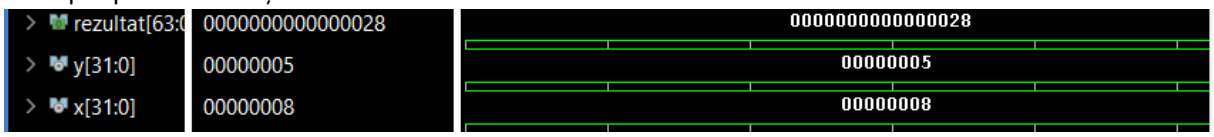


Rotație la dreapta 31 = 2.147.483.663



Rotație la stânga 28 = 56

Exemple pentru înmulțire



5 * 8 = 40

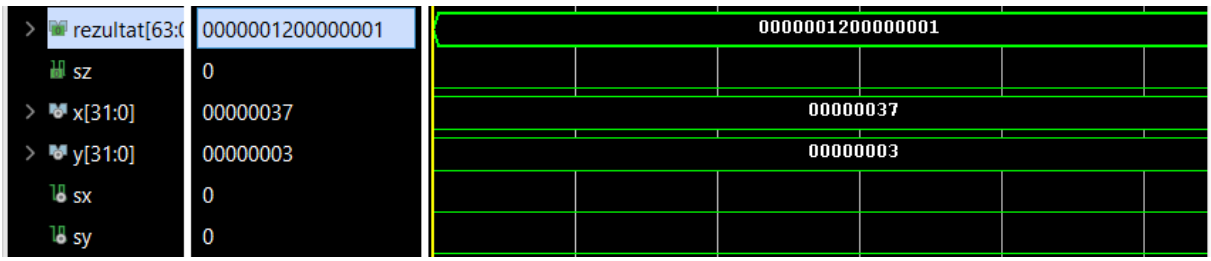
O simulare detaliată a înmulțitorului poate fi observată în figurile de mai jos.

16 * 3 = 48


Name
Value
start
ovf
zero
rdy
ACC[63:0]
ADDs
Add_o...3:0]
Carry_out
LOADs
accS[63:0]
RA[31:0]
RB[31:0]
Rxs[31:0]
Rys[31:0]
SHIFTs
rdyS
clk
lsb
starts
zero1
zero2

Exemple de impartiri:

55 / 3 = 18 rest 1



9 / 2 = 4 rest 1

VIII. BIBLIOGRAFIE:

[1] https://users.utcluj.ro/~baruch/ac/curs/UAL.pdf
[2] https://andrei.clubcisco.ro/1ii/cursuri/UAL..PDF
[3] https://www.baeldung.com/cs/arithmetic-logic-unit
[4] https://allaboutfpga.com/vhdl-code-for-4-bit-alu/
[5] Curs-AC
[6] http://retele.elth.ucv.ro/Popescu%20Daniela/Calculatoare%20de%20bord/Prelegere%202%20Dinca%20Liviu%20-%20Notiuni%20de%20arhitectura%20calculatoarelor.pdf