Seminar 2 - Logică digitală. CIRCUITE COMBINAŢIONALE. UNITATEA ARITMETICO-LOGICĂ. DETERMINAREA LATENŢEI.

student Ștefan-Alexandru JURA
prof. dr. ing. Oana AMĂRICĂI-BONCALO

22 Martie 2024

1 Scurt breviar teoretic.

Scopul acestui seminar este de a aprofunda noţiunile legate de circuite combinaţionale, studiate în cadrul cursului de Logică Digitală, prin intermediul implementării unei ALU (Unitate Aritmetico-Logică). În continuare se prezintă, succint, noţiunile necesare pentru sinteza acestui dispozitiv.

1.1 Circuite combinaţionale.

Circuitele combinaţionale sunt circuite fără memorie, fără feedback, în care **ieşirile depind numai de intrări.** Acestea se sintetizează folosind porţi logice şi expresii logice aferente ieşirilor, care sunt funţie de intrare:

$$\text{ieşiri} = f(\text{intrări})$$

Principalele porţi logice sunt AND, OR, NAND, NOR, XOR/EXOR (”sau-exclusiv”) şi XNOR, iar minimizarea funcţiilor logice (aducerea la forma cea mai simplă şi eficientă din punctul de vedere al perfomanţei) se face cu ajutorul unor metode consacrate, precum cea a hărţilor Veich-Karnaugh, studiată în seminarul trecut.

1.2 Unitatea aritmetico-logică (Arithmetic Logic Unit - ALU)

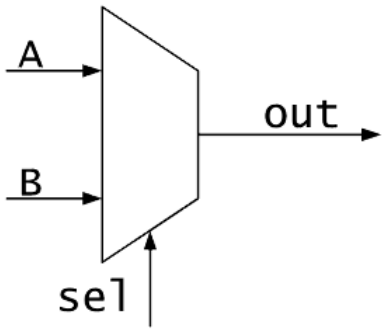
Unitatea aritmetico-logică este ”creierul” unui calculator, **un dispozitiv care realizează operaţii aritmetice (adunare, scădere, înmulţire, împărţire) şi operaţii logice (precum AND, OR etc.)**, numele dispozitivului fiind bazat pe operaţiile realizate. Sau, informal,

ALU n. [Arthritic Logic Unit or (rare) Arithmetic Logic Unit] A random-number generator supplied as standard with all computer systems. Stan Kelly-Bootle, The Devil's DP Dictionary, 1981 :)
Cu scop didactic, se va prezenta implementarea ALU pentru operanzi pe n biţi.

2 Proiectarea ALU.

2.1 Proiectarea blocului de extensie logică.

Aşa cum reiese din nume, ***rolul acestui bloc este de a realiza operaţii logice (AND, OR etc.)***. Pentru a putea *SELECTA* operaţia dorită, este nevoie de un dispozitiv combinaţional numit **MULTIPLEXOR sau SELECTOR**, care are n intrări şi $\log_2 n$ intrări de seleţie. Pentru a ilustra funcţionarea acestui dispozitiv, vom lua ca exemplu un multiplexor 2-la-1 (2 intrări la o ieşire).



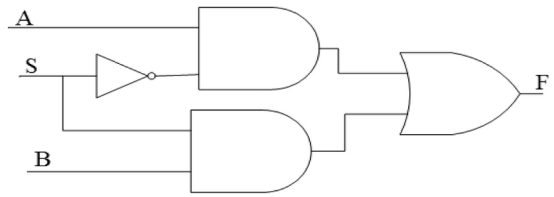
Intrarea A va avea indexul 0, iar B indexul 1. În functie de valoarea selectorului, dacă acesta este 0, la ieşirea *out* vom avea valoarea lui A, iar dacă acesta este 1, la ieşirea *out* vom avea valoarea lui B. Tabelul de adevăr pentru un multiplexor (abreviat MUX) este:

sel (S)	A	B	out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

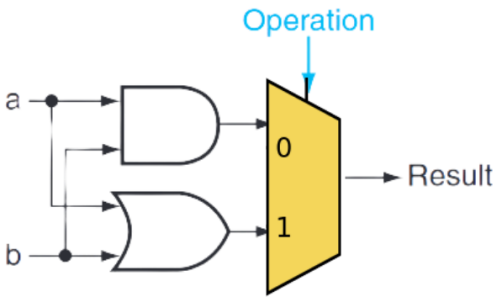
În urma minimizării folosind diagrame Veich-Karnaugh, se obţine

$$out = \overline{S}A + SB$$

Circuitul logic care descrie MUX-ul $2:1$ este



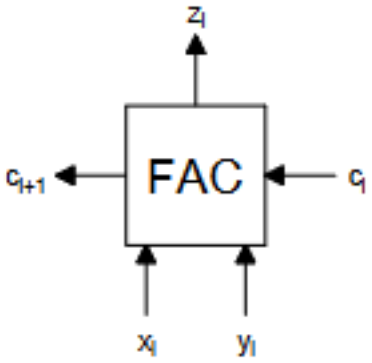
Deci, un multiplexor ne va ajuta să selectăm ce operatie dorim să facem. Prin urmare, un bloc de extensie logică care realizează operaţiile "AND" şi "OR" arată astfel:



Semnalul "Operation" este cel ce selectează operaţia. Dacă acesta este 0, se va realiza "AND" între biţii a şi b, altfel se va realiza "OR". De asemenea, se mai pot adăuga alte operaţii logice, precum complementarea biţilor, identitatea (adică bitul rămâne la fel), NOR, NAND etc.

2.2 Proiectarea blocului de extensie aritmetică.

În cadrul acestui bloc, se vor implementa operaţiile de adunare şi scădere combinaţionale. Implementarea înmulţirii şi împărţirii în sistemele de calcul va fi discutată în cadrul cursurilor de "Arhitectura calculatoarelor" şi "Calculatoare numerice", anul al II-lea, semestrul I, respectiv semestrul II.
Pentru proiectarea sumatorului, ne vom folosi de FAC (Full Adder Cell). Acesta este un circuit combinaţional care are ca intrări 2 biţi si un transport (carry in), iar la ieşire are suma şi carry out. Informal, carry out-ul poate fi asemănat cu cifra obţinută la trecerea peste ordin în cazul adunării numerelor zecimale. Un FAC se reprezintă astfel



Tabelul de adevăr pentru un FAC este:

Inputs			Outputs	
x_i	y_i	c_i	z_i	c_{i+1}
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

Folosind diagrame Karnaugh, scoatem expresiile pentru z_i (care este suma

biţilor de la rangul i) şi pentru c_{i+1} (care este carry out-ul). Harta Karnaugh pentru sumă este:

c_i \ x_iy_i	00	01	11	10
0		1		1
1	1		1	

Se observă că nu se poate minimiza funcţia. Se obţine aşa-zisa "tablă de şah", care se minimizează folosind proprietăţi ale algebrei boolene:

$$z_i = \overline{x_i}\,\overline{y_i}c_i + \overline{x_i}y_i\overline{c_i} + x_i\overline{y_i}\,\overline{c_i} + x_iy_ic_i$$

Dăm factor comun pe x_i şi $\overline{x_i}$ şi obţinem:

$$z_i = \overline{x_i}(\overline{y_i}c_i + y_i\overline{c_i}) + x_i(\overline{y_i\overline{c_i}} + y_ic_i)$$

Ştim că $A \oplus B = \overline{A}B + A\overline{B}$, deci

$$z_i = \overline{x_i}(y_i \oplus c_i) + x_i(\overline{y_i \oplus c_i})$$

Şi se observă uşor că

$$z_i = x_i \oplus y_i \oplus c_i$$

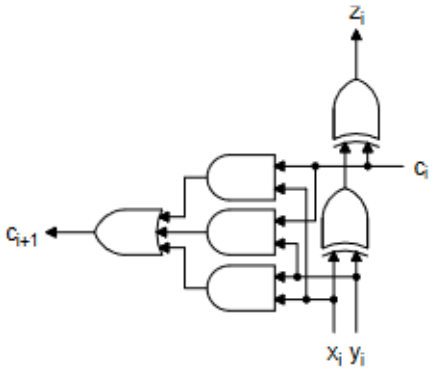
Pentru c_{i+1}, obţinem următoarea hartă Karnaugh:

c_i \ x_iy_i	00	01	11	10
0			1	
1		1	1	1

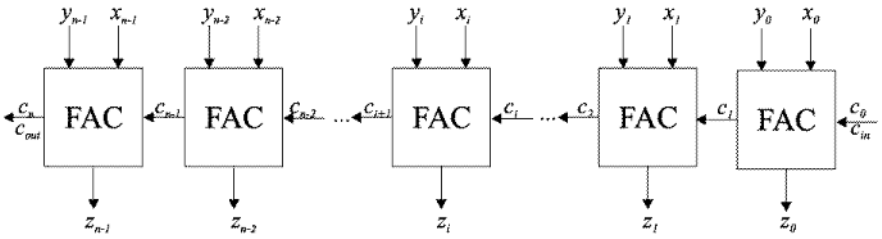
Minimizând, se obţine că

$$c_{i+1} = x_iy_i + x_ic_i + y_ic_i$$

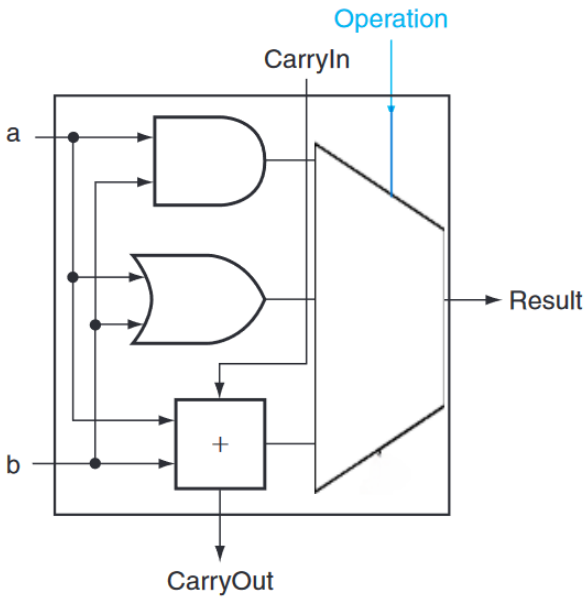
Circuitul logic aferent lui FAC este:



Cum noi dorim să însumăm mai mulţi biţi, vom avea nevoie, practic, de mai multe celule de tip FAC (fiecare celula FAC adună câte 2 biţi, câte unul de la fiecare operand). Se obţine, astfel, un **sumator paralel cu propagarea serială a carry-ului, denumit Ripple Carry Adder (RCA).** Arhitectura RCA pe n biţi este:



Această operaţie de adunare va fi adaugată în ALU, care acum va arăta aşa:

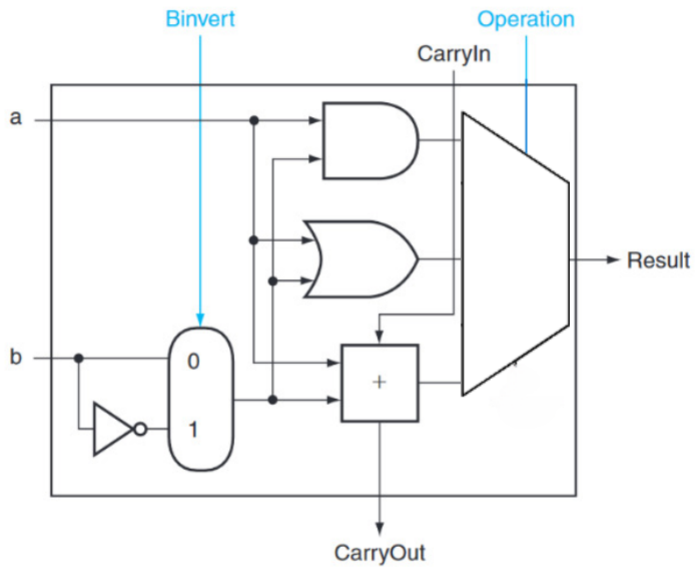


unde prin "+" se înţelege o celulă de tip FAC.
De asemenea, pentru realizarea operaţiei de scădere, ţinem cont de următorul aspect:

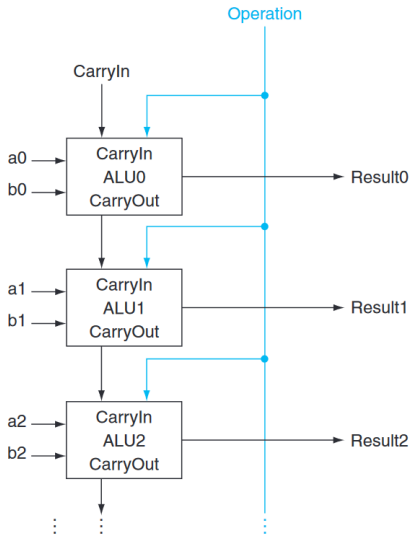
$$X - Y = X + (-Y) = X + \overline{Y} + 1$$

adică, practic, scăderea reprezintă adunarea unui operand cu complementul de 2 a celuilalt operand. Pentru a realiza acest lucru, vom mai adăuga un semnal,

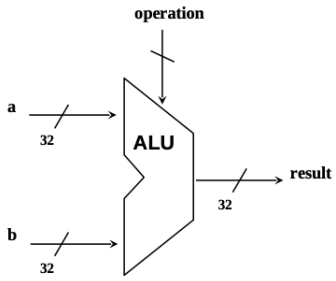
"Binvert", care complementeaza bitul B şi se va realiza scăderea, setând carry in pe 1. Noua arhitectura de ALU pe 1 bit este:



Aceasta este o celulă care face operaţiile pe 1 bit (între 1 bit de la un operand şi celălalt bit de la celălalt operand). Dacă dorim să avem operaţiile acestea pentru numere pe n biţi, trebuie să interconectăm mai multe celule ALU, astfel:



Simbolul unui ALU pe 32 de biţi este următorul:



Observaţie. Când utilizăm sumatorul, există posibilitatea apariţiei unui **overflow**, care are expresia:

$$overflow = c_n \oplus c_{n-1}$$

Aşadar, ALU va avea şi un semnal de ieşire suplimentar, overflow, ca fiind EXOR între ultimii 2 biţi de carry.

3 Implementarea ALU în Verilog HDL.

Primul pas este realizarea design-ului. Succint sunt prezentate etapele necesare.

1. Definirea Cerințelor.
Înainte de a începe scrierea codului, este important să înțelegem ce funcții va trebui să îndeplinească ALU:

• Operații aritmetice: adunare și scădere.
• Operații logice: AND, OR.
• Acceptă doi operanzi de 32 de biți și un cod de operație de 2 biți.
• Generează un rezultat de 32 de biți și un semnal de carry out.

2. Structura Codului Verilog pentru ALU.
Structura de bază a ALU în Verilog va include: Declarația Modulului: Definește interfața ALU, inclusiv intrările (operanzii și codul operației) și ieșirile (rezultatul și carry out).
Logica ALU: Implementează operațiile specificate folosind o structură case într-un bloc always @(*) pentru a alege operația bazată pe codul operației.

3. Implementarea Operațiilor.
Pentru fiecare operație (adunare, scădere, AND, OR), scriem logică specifică: Adunare și Scădere: Se pot utiliza operatorii + și - pentru a calcula rezultatul și carry out. Dar, pentru implementarea acestui ALU, se doreşte implementarea specifică a sumatorului, adică se realizează un alt modul pentru sumator şi se instanţiază în cadrul modulului principal.
AND și OR: Utilizăm operatorii & și | pentru a calcula rezultatul. Carry out nu este relevant pentru aceste operații.

Mai jos este un exemplu orientativ pentru testbench-ul aferent acestui ALU. Cazurile nu sunt tratate exhaustiv, ci sunt luate nişte seturi de valori comentate în cadrul codului si sunt testate toate combinaţiile posibile dintre acestea (asemănător produsului cartezian). Pentru a fi în totalitate corect, ar trebui tratate şi cazuri speciale, precum overflow-ul (a se realiza în cadrul temei).

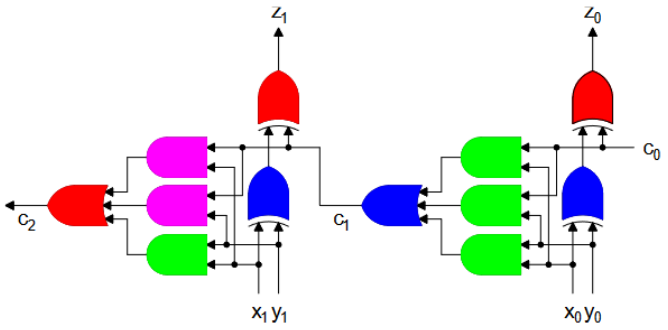
```
1  `timescale 1ns / 1ps
2
3  module alu_tb;
4
5  reg [31:0] a, b;
6  reg [1:0] op_code;
7  wire [31:0] result;
8  wire carry_out;
9
10 // Presupunand ca avem un modul ALU cu denumirea `alu`
11 alu uut (
12     .a(a),
13     .b(b),
14     .op_code(op_code),
15     .result(result),
16     .carry_out(carry_out)
17 );
18
19 initial begin
20     // Valorile specificate pentru testare
```

```
21      int test_values[5];
22      test_values[0] = 32'd0; // Valoare absoluta minima
23      test_values[1] = 32'd1; // Valoare absoluta minima + 1
24      test_values[2] = $random % (32'hFFFFFFFE - 2) + 2; // O
    valoare aleatoare
25      test_values[3] = 32'hFFFFFFFE; // Valoare absoluta
    maxima - 1
26      test_values[4] = 32'hFFFFFFFF; // Valoare absoluta
    maxima
27
28      // Parcurgem toate combinatiile de valori si operatii
29      for (op_code = 0; op_code < 4; op_code = op_code + 1)
    begin
30          foreach (test_values[a_idx]) begin
31              a = test_values[a_idx];
32              foreach (test_values[b_idx]) begin
33                  b = test_values[b_idx];
34                  #10; // Asteptam 10 nanosecunde pentru
    vizualizare
35
36                  // Afisare rezultate test
37                  $display("Time: %t, OpCode: %b, A: %h, B: %h
    , Result: %h, Carry: %b",
38                           $time, op_code, a, b, result,
    carry_out);
39              end
40          end
41      end
42
43      $finish; // Finalizeaza simularea
44  end
45
46  endmodule
```

Listing 1: ALU Test Bench Example

4 Determinarea latenţei.

Întârzierile sau latenţele în cazul porților logice se referă la **timpul necesar ca schimbările de la intrările unei porți logice să se propage și să producă o schimbare la ieșirea acesteia.** Definim şi noţiunea de **cale critică** ca fiind **calea semnalelor din circuit corespunzătoare întârzierii maxime de propagare a semnalelor.** Notând cu d o unitate de timp (de exemplu, nanosecunde - ns), pentru porţile logice standard vom avea următoarele latenţe:
• porţile primitive, precum AND, OR, au latenţa $1d$;
• invertoarele nu au întârzieri, deci se spune că au latenţa $0d$;
• porţile SAU-EXCLUSIV au întârzierea $2d$. **De ce? :)**
Exemplu pentru determinarea căii critice: RCA pe 2 biti.



Întârizerea unui segment RCA pe n biți:
$D^{c_{out}}_{RCA} = 2nd$
$D^{z}_{RCA} = 2nd$

Cum raţionăm: pornim de la semnalele de intrare, să zicem de la x_0 şi y_0. La momentul iniţial, semnalele au latenţa $0d$. Ambele trec printr-o poartă EXOR, care duce la latenţa $2d$. Acum, ieşirea porţii EXOR intră, alături de semnalul c_0, într-o altă poartă EXOR care are ca ieşire z_0. Cum determinăm aici latenţa? Ne uităm la intrările porţii şi adunăm latenţa acesteia (care este de $2d$) cu **latenţa maximă dintre cele 2 semnale de intrare**, în cazul nostru semnalul primei porţi EXOR dintre x_0 şi y_0. Se obţine, astfel, pentru un bit de sumă, latenţa maximă de $2d$. Se procedează analog şi pentru c_{out} şi se obţine latenţa tot $2d$. Prin induţie matematică, se demonstrează că latenţa maximă a unui RCA pe n biţi, atât pentru bitul de sumă, cât şi pentru bitul de carry out, este $2nd$, unde n este numărul de biţi pe care este construit RCA-ul.

5 Temă de casă pentru bonus.

1. * Modelarea unui ALU pe 32 de biţi în Verilog şi testbench-ul aferent acestuia.
2. Să se calculeze latenţa maximă pentru următoarele circuite combinaţionale:
a) majority voter;
b) ALU pe 8 biţi;
c) multiplexor 4 la 1.
3. Să se proiecteze un circuit digital care are ca intrări 2 numere, X şi Y, pe 4 biţi fiecare, fără semn, şi care are ca ieşire 1 dacă suma acestora este număr par şi 0 altfel. Se va preciza şi calea critică.
4. Să se arate cum s-a ajuns la condiţia de overflow în cazul adunării.

Bibliografie

[1] David A. Patterson, John L. Hennessy (2014) *Computer Organisation and Design: The Hardware/Software Interface, Fifth Edition*, Elsevier.

[2] Mircea Vlăduţiu (2012) *Computer Arithmetic: Algorithms and Hardware Implementations*, Springer.

[3] Oana Amăricăi-Boncalo (2024) *Curs Logică Digitală.*

[4] Flavius-Gabriel Opriţoiu (2024) *Curs Arhitectura Calculatoarelor.*