

Seminar 2 - Logică digitală.

CIRCUITE COMBINAȚIONALE. UNITATEA ARITMETICO-LOGICĂ. DETERMINAREA LATENȚEI.

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

22 Martie 2024

1 Scurt breviar teoretic.

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

1.1 Circuite combinaționale.

Circuitele combinaționale sunt **circuite fără memorie, fără feedback, în care ieșirile depind numai de intrări**. Acestea se sintetizează folosind porți logice și expresii logice aferente ieșirilor, care sunt funcție de intrare:

$$\text{ieșiri} = f(\text{intrări})$$

Principalele porți logice sunt AND, OR, NAND, NOR, XOR/EXOR ("sau-exclusiv") și XNOR, iar minimizarea funcțiilor logice (aducerea la forma cea mai simplă și eficientă din punctul de vedere al performanței) se face cu ajutorul unor metode consacrate, precum cea a hărților 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 operații aritmetice (adunare, scădere, înmulțire, împărțire) și operații logice (precum AND, OR etc.)**, numele dispozitivului fiind bazat pe operațiile realizate. Sau, informal,

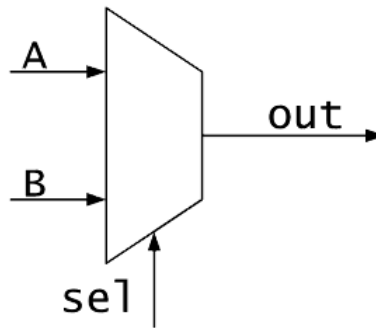
ALU n. [Arthritic Logic Unit or (rare) Arithmetic Logic Unit] A random-number generator supplied as standard with all computer systems. Stan Kelly-Bootle, The Devil's DP Dictionary, 1981 :)

Cu scop didactic, se va prezenta implementarea ALU pentru operanzi pe n biți.

2 Proiectarea ALU.

2.1 Proiectarea blocului de extensie logică.

Așa cum reiese din nume, *rolul acestui bloc este de a realiza operații logice (AND, OR etc.)*. Pentru a putea *SELECTA* operația dorită, este nevoie de un dispozitiv combinațional numit **MULTIPLEXOR** sau **SELECTOR**, care are n intrări și $\log_2 n$ intrări de selecție. Pentru a ilustra funcționarea acestui dispozitiv, vom lua ca exemplu un multiplexor 2-la-1 (2 intrări la o ieșire).



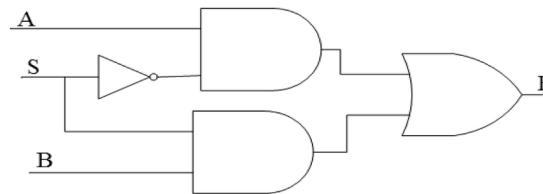
Intrarea A va avea indexul 0, iar B indexul 1. În funcție de valoarea selectorului, dacă acesta este 0, la ieșirea *out* vom avea valoarea lui A , iar dacă acesta este 1, la ieșirea *out* vom avea valoarea lui B . Tabelul de adevăr pentru un multiplexor (abreviat MUX) este:

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

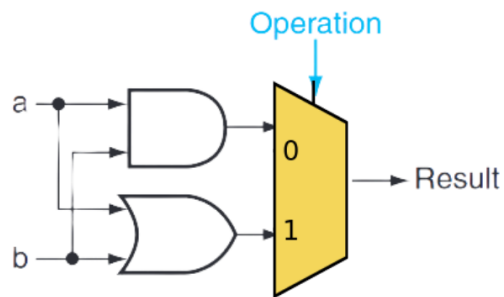
În urma minimizării folosind diagrame Veich-Karnaugh, se obține

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

Circuitul logic care descrie MUX-ul 2 : 1 este



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

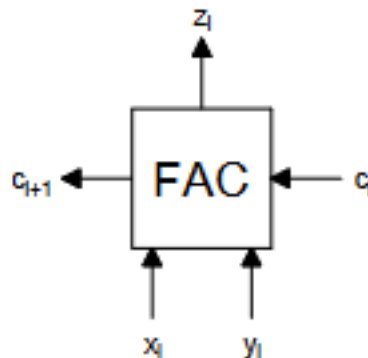


Semnalul "Operation" este cel ce selectează operația. Dacă acesta este 0, se va realiza "AND" între biții a și b , altfel se va realiza "OR". De asemenea, se mai pot adăuga alte operații logice, precum complementarea biților, identitatea (adică bitul rămâne la fel), NOR, NAND etc.

2.2 Proiectarea blocului de extensie aritmetică.

În cadrul acestui bloc, se vor implementa operațiile de adunare și scădere combinaționale. Implementarea înmulțirii și împărțirii în sistemele de calcul va fi discutată în cadrul cursurilor de "Arhitectura calculatoarelor" și "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 combinațional care are ca intrări 2 biți și un transport (carry in), iar la ieșire are suma și carry out. Informal, carry out-ul poate fi asemănat cu cifra obținută 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

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

		$x_i y_i$			
		00	01	11	10
c_i	0		1		1
	1	1		1	

Se observă că nu se poate minimiza funcția. Se obține așa-zisa ”tablă de șah”, care se minimizează folosind proprietăți ale algebrei boolene:

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

Dăm factor comun pe x_i și $\overline{x_i}$ și obținem:

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

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

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

Și se observă ușor că

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

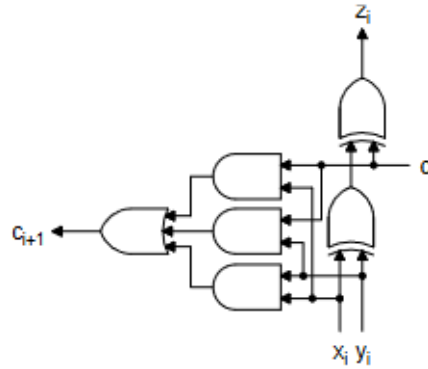
Pentru c_{i+1} , obținem următoarea hartă Karnaugh:

		$x_i y_i$			
		00	01	11	10
c_i	0			1	
	1		1	1	1

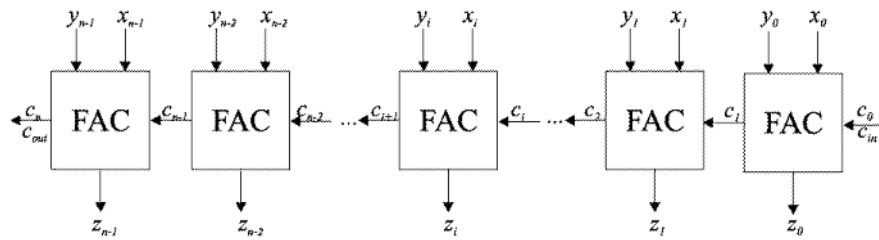
Minimizând, se obține că

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

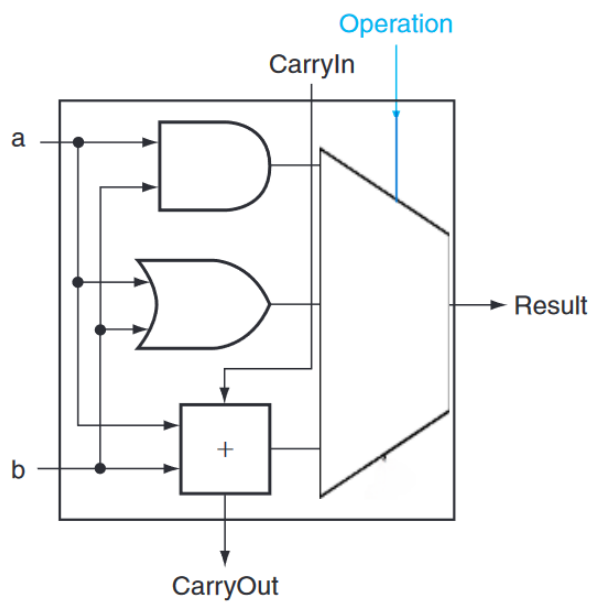
Circuitul logic aferent lui FAC este:



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



Această operație de adunare va fi adăugată în ALU, care acum va arăta așa:



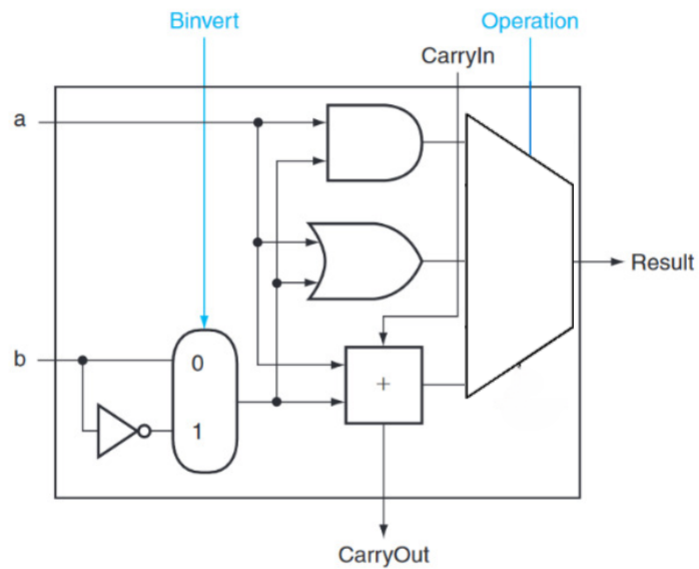
unde prin "+" se înțelege o celulă de tip FAC.

De asemenea, pentru realizarea operației de scădere, ținem cont de următorul aspect:

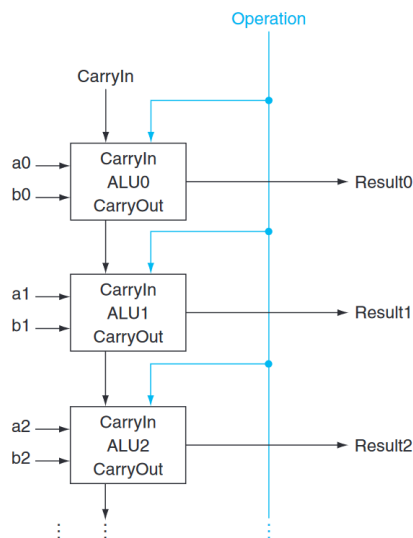
$$X - Y = X + (-Y) = X + \bar{Y} + 1$$

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

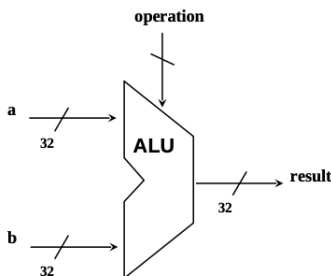
”Binvert”, care complementează bitul B și se va realiza scăderea, setând carry in pe 1. Noua arhitectură de ALU pe 1 bit este:



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



Simbolul unui ALU pe 32 de biți este următorul:



Observație. Când utilizăm sumatorul, există posibilitatea apariției unui **overflow**, care are expresia:

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

Așadar, ALU va avea și un semnal de ieșire suplimentar, overflow, ca fiind EXOR între ultimii 2 biți 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 dorește implementarea specifică a sumatorului, adică se realizează un alt modul pentru sumator și se instanțiază 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 niște seturi de valori comentate în cadrul codului și sunt testate toate combinațiile posibile dintre acestea (asemănător produsului cartezian). Pentru a fi în totalitate corect, ar trebui tratate și 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; // 0
    valoare aleatoare
25     test_values[3] = 32'hFFFFFFFE; // Valoare absoluta
    maxima - 1
26     test_values[4] = 32'hFFFFFFF; // 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

```

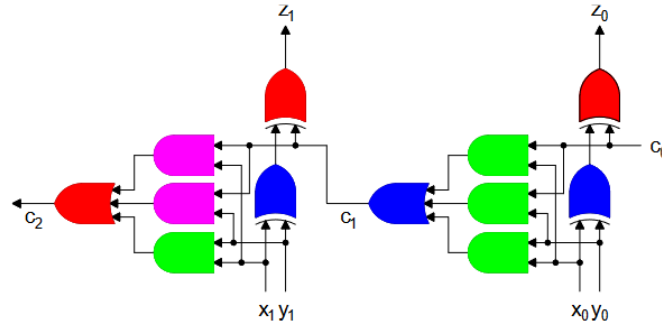
Listing 1: ALU Test Bench Example

4 Determinarea latenței.

Întârzierile sau latențele 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 și noțiunea 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 porțile logice standard vom avea următoarele latențe:

- porțile primitive, precum AND, OR, au latența $1d$;
- invertoarele nu au întârzieri, deci se spune că au latența $0d$;
- porțile SAU-EXCLUSIV au întârzierea $2d$. **De ce? :**)

Exemplu pentru determinarea căii critice: RCA pe 2 biti.



Întârizarea unui segment RCA pe n biți:

$$D_{RCA}^{c_{out}} = 2nd$$

$$D_{RCA}^z = 2nd$$

Cum raționăm: pornim de la semnalele de intrare, să zicem de la x_0 și y_0 . La momentul inițial, semnalele au latența $0d$. Ambele trec printr-o poartă EXOR, care duce la latența $2d$. Acum, ieșirea porții EXOR intră, alături de semnalul c_0 , într-o altă poartă EXOR care are ca ieșire z_0 . Cum determinăm aici latența? Ne uităm la intrările porții și adunăm latența acestora (care este de $2d$) cu **latența maximă dintre cele 2 semnale de intrare**, în cazul nostru semnalul primei porți EXOR dintre x_0 și y_0 . Se obține, astfel, pentru un bit de sumă, latența maximă de $2d$. Se procedează analog și pentru c_{out} și se obține latența tot $2d$. Prin inducție matematică, se demonstrează că latența maximă a unui RCA pe n biți, atât pentru bitul de sumă, cât și pentru bitul de carry out, este $2nd$, unde n este numărul de biți pe care este construit RCA-ul.

5 Temă de casă pentru bonus.

- * Modelarea unui ALU pe 32 de biți în Verilog și testbench-ul aferent acestuia.
- Să se calculeze latența maximă pentru următoarele circuite combinaționale:
 - majority voter;
 - ALU pe 8 biți;
 - multiplexor 4 la 1.
- Să se proiecteze un circuit digital care are ca intrări 2 numere, X și Y, pe 4 biți fiecare, fără semn, și care are ca ieșire 1 dacă suma acestora este număr par și 0 altfel. Se va preciza și calea critică.
- Să se arate cum s-a ajuns la condiția 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 Vlăduțiu (2012) *Computer Arithmetic: Algorithms and Hardware Implementations*, Springer.
- [3] Oana Amăricăi-Boncalo (2024) *Curs Logică Digitală*.
- [4] Flavius-Gabriel Oprițoiu (2024) *Curs Arhitectura Calculatoarelor*.