

# Laboratorul 7 - Calculatorul Didactic: Unitatea Aritmetică-Logică

În cadrul acestui laborator se continuă implementarea în Verilog a procesorului didactic prezentat la [curs](#) și în [laboratorul 6](#).

## Unitatea aritmetică-logică

Unitatea aritmetică-logică ([Fig. 1](#)) este responsabilă de efectuarea operațiilor aritmetice și logice în timpul execuției instrucțiunilor. Operațiile primesc unul sau doi operanzi, iar UAL-ul în afară de producerea rezultatului setează și o serie de indicatori de condiții (eng. *flags*) rezultați în urma operațiilor. Operațiile disponibile în UAL derivă din instrucțiunile prezente în setul de instrucțiuni al procesorului didactic, însă nu au neapărat o corespondență 1-la-1 cu acestea. Unele operații sunt folosite în mai multe instrucțiuni, iar unele instrucțiuni folosesc mai multe operații. UAL-ul trebuie însă proiectat în așa fel astfel încât să cuprindă toate operațiile necesare în execuția instrucțiunilor disponibile în procesorul didactic.

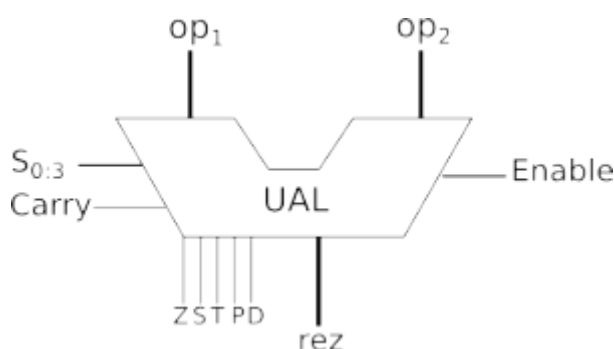


Fig. 1: Unitatea aritmetică-logică

Operanzii pe 16 biți sunt  $op1$  și  $op2$ , iar cei 4 biți  $S$  selectează operația ce va fi efectuată. Rezultatul este pus pe magistrală prin activarea semnalului *Enable*. Acesta este dezactivat de instrucțiunile care nu au nevoie de fapt de rezultatul operației, ci doar de indicatori (ex: *cmp*, *test*). Operațiile de adunare și scădere folosesc și un bit de carry/borrow reprezentat prin semnalul *Carry*. Acesta este activat selectiv de instrucțiunile ADD/ADC (*add with carry*) și SUB/SBB (*subtract with borrow*), precum și alte instrucțiuni, pentru a obține rezultatul dictat de semantica instrucțiunii.

## Descrierea generală a registrului care conține indicatorii de condiții ("IND")

Registrul de indicatori constituie o grupare a unor bistabili cu funcții individuale, poziționați la execuția instrucțiunilor, în funcție de rezultatul din unitatea aritmetică logică. Registrul IND permite alegerea unei secvențe de execuție următoare unei instrucțiuni aritmetice/logice, în funcție de rezultatul operației.

## Descrierea detaliată a indicatorilor de condiții

- **T/C** (*transport, eng: carry*): Este setat (poziționat în "1" ) dacă în urma unei adunări rezultă un transport dinspre rangul cel mai semnificativ, altfel T este șters (trecut în "0"). Transportul T este setat dacă în urma unei scăderi rezultă un împrumut în cel mai semnificativ bit al rezultatului, altfel este șters. Poate fi interpretat ca depășire în instrucțiunile cu numere întregi fără semn. Poate fi utilizat în instrucțiunile ADC și SBB pentru a efectua operații aritmetice în precizie multiplă. Poate fi testat cu instrucțiuni de salt condiționat.
- **S** (*semn*):
  - La execuția operațiilor aritmetice și logice, indicatorul S este setat la valoarea bitului cel mai semnificativ al rezultatului (bitul de semn).
    - Pentru numere cu semn (în absența depășirii)  $S=0$  indică rezultat pozitiv iar  $S=1$  indică rezultat negativ.
    - În cazul operațiilor cu numere fără semn S poate fi ignorat deoarece în acest caz specifică cel mai semnificativ bit al rezultatului.
  - Poate fi testat cu instrucțiuni de salt condiționat.
- **Z** (*zero*):
  - Este setat dacă în urma unei operații aritmetice sau logice se obține rezultat egal cu 0, altfel Z este șters.
  - Poate fi testat cu instrucțiuni de salt condiționat pentru a dirija secvența de execuție a instrucțiunilor în funcție de valoarea rezultatului.
- **P** (*paritate*):
  - Indicatorul de paritate P este setat dacă în urma execuției unei operații aritmetice sau logice rezultatul conține un număr par de biți egali cu 1, altfel P este șters.
  - Poate fi testat cu instrucțiuni de salt condiționat.
- **D/O** (*depășire, eng: overflow*):
  - Indicatorul de depășire D (*eng. overflow*) este setat dacă în urma execuției unei operații aritmetice rezultatul este un număr pozitiv prea mare sau un număr negativ prea mic pentru a putea fi reprezentat în operandul destinație (exclusiv bitul de semn); altfel D este șters.
  - Poate fi interpretat ca depășire în instrucțiunile cu numere întregi cu semn și poate fi testat cu instrucțiuni de salt condiționat.
  - Poate fi ignorat în operațiile aritmetice cu numere întregi fără semn.
  - Exemple:
    - În cazul unei operații cu semn, pe 4 biți:  $0100 + 0100 = 1000$  (indicatorul de depășire e setat)
    - $1000 + 1000 = 0000$  (indicatorul de depășire e setat)
    - $0110 + 1001 = 1111$  (indicatorul de depășire este șters)
    - $0100 + 0110 = 1010$  (indicatorul de depășire este setat, iar cel de transport este șters)
  - Detalii despre situații în care apare depășirea și cum se detectează găsiți [aici](#) și [aici](#).

### Observații referitoare la indicatorii de condiții:

- pentru operații între numere întregi fără semn contează *indicatorul de transport/carry* (T/C), pentru cele cu semn contează *indicatorul de depășire/overflow* (D/O).
- *indicatorul de paritate* (P) este 1 în caz ca avem un număr par de biți, altfel este 0 (deci paritate impară, altfel se numea paritate pară - 0 pentru număr par de biți, 1 pentru număr impar). Metoda de determinare a acestuia constă în aplicarea unui *xor negat* între biții cuvântului (dacă era paritate pară aplicam xor simplu).
- unele operații au efect asupra tuturor indicatorilor de condiție (exemplu ADD), altele însă afectează doar o parte dintre aceștia.

## Descriere operații

- **ADC:**
  - ADC este un Full Adder, în care intrările sunt cei doi operanzi și carry-ul.
  - Flag-uri setate: T/C, S, Z, P, D/O.
- **SBB1:**
  - SBB1 scade operandul *op2* și bitul *Carry* din operandul *op1*.
  - Flag-uri setate: T/C, S, Z, P, D/O.
- **SBB2:**
  - SBB2 scade operandul *op1* și bitul *Carry* din operandul *op2*.
  - Flag-uri setate: T/C, S, Z, P, D/O.
- **NOT:**
  - NOT inversează individual fiecare bit al operandului.
  - Flag-uri setate: S, Z, P (T/C și D/O sunt întotdeauna 0 în cazul operațiilor logice).
- **AND:**
  - AND efectuează "ȘI" logic între biții celor doi operanzi.
  - Flag-uri setate: S, Z, P (T/C și D/O sunt întotdeauna 0 în cazul operațiilor logice).
- **OR:**
  - OR efectuează "SAU" logic între biții celor doi operanzi.
  - Flag-uri setate: S, Z, P (T și D sunt întotdeauna 0 în cazul operațiilor logice).
- **XOR:**
  - XOR efectuează "SAU-exclusiv" între biții celor doi operanzi.
  - Flag-uri setate: S, Z, P (T și D sunt întotdeauna 0 în cazul operațiilor logice).
- **SHL/SAL:**
  - SHL/SAL realizează deplasarea la stânga cu o poziție a operandului.
  - În bitul cel mai puțin semnificativ se introduce zero. Deplasarea logică și aritmetică la stânga cu o poziție produc același rezultat.
  - Flag-uri setate: S, Z, P, T (conține întotdeauna bitul deplasat în afară), D (e setat pe 1 doar dacă în urma operației bitul cel mai semnificativ - de semn - și-a schimbat valoarea, altfel e 0)
- **SHR:**
  - SHR deplasează la dreapta biții operandului, introducând zero în bitul cel mai semnificativ.
  - Flag-uri setate: S, Z, P, T (conține întotdeauna bitul deplasat în afară), D (e setat pe 1 doar dacă în urma operației bitul cel mai semnificativ - de semn - și-a schimbat valoarea, altfel e 0)
- **SAR:**
  - SAR deplasează aritmetic la dreapta biții operandului.
  - Deplasarea se face cu extensia bitului de semn (bitul de semn rămâne neschimbat iar bitul cel mai semnificativ de date preia conținutul bitului de semn).
  - Flag-uri setate: S, Z, P, T (conține întotdeauna bitul deplasat în afară), D (e setat pe 1 doar dacă în urma operației bitul cel mai semnificativ - de semn - și-a schimbat valoarea, altfel e 0)

Sintaxa Verilog pentru operatorii aritmetici și logici este prezentată în [laboratorul 2](#).

## Operațiile de shiftare

Operațiile de shiftare se împart în: shiftare logică, shiftare aritmetică, shiftare circulară fără carry, shiftare circulară cu carry. În acest laborator vom trata primele două tipuri de operații.

## Shiftare logică

O shiftare logică nu ține cont de semnul operandului. În cazul shiftării logice se ține cont doar de ordinea biților, iar pozițiile care rămân libere sunt umplute cu zerouri.

În Fig. 2 și Fig. 3 se poate observa modul de execuție al shiftărilor logice.

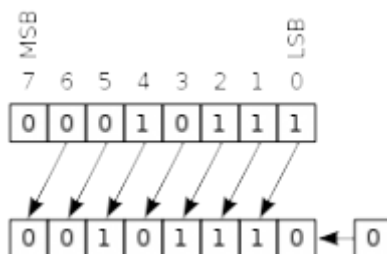


Fig. 2: Shiftare logică/aritmetică la stânga

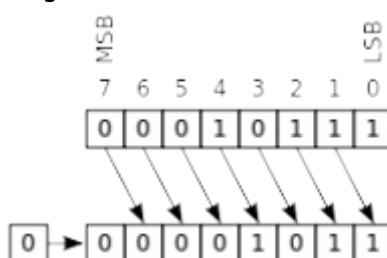


Fig. 3: Shiftare logică la dreapta

Iar în Fig. 4 și Fig. 5 găsiți un exemplu practic de efectuare a shiftărilor logice spre stânga (SHL) și spre dreapta (SHR).

$11001011 \ll 2 = 00101100$

$11001011 \ll 3 = 01011000$

Fig. 4: Exemplu de shiftare logică/aritmetică la stânga

$11001011 \gg 2 = 00110010$

$11001011 \gg 3 = 00011001$

Fig. 5: Exemplu de shiftare logică la dreapta

În Verilog, operatorii de shiftare logică sunt "<<" și ">>".

## Shiftare aritmetică

Spre deosebire de shiftarea logică spre dreapta (SHR), shiftarea aritmetică spre dreapta (SAR) nu umple spațiile rămase libere cu zerouri. În cazul SAR, spațiile rămase libere se umplu cu valoarea bitului cel mai semnificativ, care se replică de câte ori este nevoie (vezi Fig. 6).

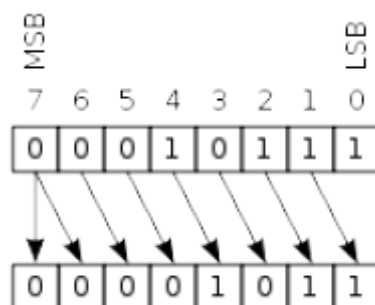


Fig. 6: Exemplu de shiftare aritmetică la dreapta

În Verilog, operatorii de shiftare aritmetică sunt "<<<" și ">>>". ⚠️ Aceștia au efectul scontat doar dacă variabila a fost declarată signed (ex: reg signed [15:0] a;).

Shiftarea logică spre stânga (SHL) și shiftarea aritmetică spre stânga (SAL) se efectuează în același mod (vezi Fig. 3). Se păstrează, însă, ambele mnemonici (SHL / SAL) pentru a se putea păstra contextul folosirii acestora, logic sau aritmetic.

Rezultatul shiftării cu  $n$  biți la stânga este echivalent cu înmulțirea cu  $2^n$ . Shiftarea la dreapta cu  $n$  biți este echivalentă cu împărțirea la  $2^n$ .

## Modulul UAL

Descărcați scheletul de cod și completați conținutul **modulului alu.v**, folosind intrările și ieșirile prezentate mai jos și în Fig. 1.

```
module alu(
    input oe,
    input [3:0] opcode,
    input [width-1 : 0] in1,
    input [width-1 : 0] in2,
    input carry,
    output [width-1 : 0] out,
    output [flags_width-1 : 0] flags
);
```

Codurile de identificare ale operațiilor (cei 4 biți S) sunt definite în modulul alu.v din scheletul de laborator și în codul de mai jos.

```
`define ADC      0
`define SBB1     1
`define SBB2     2
`define NOT      3
`define AND      4
`define OR       5
`define XOR      6
`define SHL      7
```

```
`define SHR      8
`define SAR      9
```

## Exerciții

Exercițiile din acest laborator constau în implementarea operațiilor și indicatorilor de condiții în modulul *alu.v* și apoi folosirea acestuia pentru a executa faza de trecere la instrucțiunea următoare.

1. **(2p)** Implementați operația *ADC* (eng. *ADd with Carry*) în cadrul unității aritmetice-logice. ⚠ **Nu uitați indicatorii de condiții!**
  - Implementarea operației fără indicatorii de condiții oferă un punctaj de 50%.
  - Hint: descrierea indicatorilor se găsește în tabelul de la pag. 9 din [curs](#) sau în [textul](#) laboratorului.
2. **(2p)** Implementați în unitatea de comandă faza de trecere la instrucțiunea următoare, prin incrementarea Contorului Program. Folosiți-vă de operația implementată la exercițiul anterior.
  - Hint: pentru realizarea de către UC a unui transfer de date între resursele calculatorului revedeți [soluția](#) laboratorului anterior.
3. **(4×1.5p)** Implementați următoarele operații în cadrul unității aritmetice-logice. ⚠ **Nu uitați indicatorii de condiții!**
  1. *SBB1* (eng. *SuBstract with Borrow*; calculează  $in1 - in2 - carry$ )
  2. *NOT*
  3. *XOR* (eng. *eXclusive OR*)
  4. *SAR* (eng. *Shift Arithmetic Right*)
    - Implementarea operației fără indicatorii de condiții oferă un punctaj de 50%.
    - Demonstrarea funcționării corecte pentru acordarea punctajului se face prin crearea unui fișier de test pentru modulul *alu* asemănător [tutorialului](#).
    - Hint: descrierea indicatorilor pentru fiecare operație se găsește în tabelul de la pag. 9 și descrierile de la pag. 10 și 11 din [curs](#) sau în [textul](#) laboratorului.

## Resurse

- [Schelet de cod](#)
- [Soluție laborator](#) (disponibilă începând cu 16.11.2019)
- [PDF laborator](#)
- [Cheat-sheet calculator didactic](#)
- [Arhitectura calculatorului didactic](#)

From:  
<https://elf.cs.pub.ro/ac/wiki/> - **AC Wiki**

Permanent link:  
<https://elf.cs.pub.ro/ac/wiki/lab/lab07>

Last update: **2019/11/10 20:30**

