

# Tutoriat 1 - Arhitectura sistemelor de calcul

Stan Bianca-Mihaela, Stăncioiu Silviu



# Contents

<b>1 [IMPORTANT] Transformarea din baza 10 in baza b</b>	<b>2</b>
1.1 Nr. intregi . . . . .	2
1.2 Fractii . . . . .	4
<b>2 [IMPORTANT] Transformarea din baza b in baza 10</b>	<b>6</b>
<b>3 [IMPORTANT] Trecerea din baza 2 într-o bază care este putere a lui 2 (și invers)</b>	<b>8</b>
<b>4 [IMPORTANT] Operații în baza b</b>	<b>10</b>
4.1 Adunarea . . . . .	10
4.2 Scaderea . . . . .	11
4.3 Inmultirea . . . . .	11
4.4 Impartirea . . . . .	13

## 1 [IMPORTANT] Transformarea din baza 10 in baza b

### 1.1 Nr. intregi

Cum se face transformarea lui x din baza 10 in baza b?

- impart x cu rest la b (  $x : b = c_1$  rest  $r_1$  )
- impart  $c_1$  cu rest la b (  $c_1 : b = c_2$  rest  $r_2$  )
- impart  $c_2$  cu rest la b (  $c_2 : b = c_3$  rest  $r_3$  ) . . .
- impart  $c_n$  cu rest la b (  $c_n : b = c_{n+1}$  rest  $r_{n+1}$  ) și  $c_{n+1} = 0$
- numarul x in baza b va fi concatenarea:  $r_{n+1}r_n...r_1$

Exemplul 1:  $(7954)_{16} = ?$

$$7954 : 16 = 497 \text{ rest } 2$$

$$497 : 16 = 31 \text{ rest } 1$$

$$31 : 16 = 1 \text{ rest } 15$$

$1 : 16 = 0$  rest  $1 \Rightarrow$  aici ne oprim

Luam resturile obtinute in ordine inversa: 1, 15, 1, 2.

Totusi, nu putem sa concatenam numerele direct. De ce? Concatenarea lor ar fi 1512. Dar cum stim ca numarul a fost obtinut prin concatenarea 1, 15, 1, 2 si nu prin concatenarea 1, 1, 5, 1, 2? Ca sa putem face diferenta, numerele de la 10 la 15, in baza 16, vor fi reprezentate prin literele de la A la F, asa cum se vede si in tabelul de mai jos:

<b>Binary Base-2</b>	<b>Decimal Base-10</b>	<b>Hexa- Decimal Base-16</b>	<b>Octal Base-8</b>	<b>BCD Code</b>	<b>Gray Code</b>
0000	0	0	0	0	0000
0001	1	1	1	1	0001
0010	2	2	2	2	0011
0011	3	3	3	3	0010
0100	4	4	4	4	0110
0101	5	5	5	5	0111
0110	6	6	6	6	0101
0111	7	7	7	7	0100
1000	8	8	10	8	1100
1001	9	9	11	9	1101
1010	10	A	12	---	1111
1011	11	B	13	---	1110
1100	12	C	14	---	1010
1101	13	D	15	---	1011
1110	14	E	16	---	1001
1111	15	F	17	---	1000

Deci noi vom concatena numerele 1, F, 1 si 2.  
 $\Rightarrow (7954)_{16} = \overline{1F12}$

**Exemplul 2:**  $(243)_2 = ?$

Pentru transformarea in baza 2 putem sa facem o mica "optimizare" in calcul. Scriem  $x \mid 1$  daca  $x$  e impar si  $x \mid 0$  daca  $x$  e par.

$243 \mid 1$ , pe linia urmatoare vom scrie catul lui 243 la impartirea cu 2

$$121 \mid 1$$

$$60 \mid 0$$

$$30 \mid 0$$

$$15 \mid 1$$

$$7 \mid 1$$

$$3 \mid 1$$

$$1 \mid 1$$

0 - am ajuns la 0 deci ne oprim

Retineti! sensul de parcurgere a resturilor este de **jos in sus**

Urmatorul pas este sa luam toate resturile in ordine inversa: 1, 1, 1, 1, 0, 0, 1, 1.

$\Rightarrow (243)_2 = \overline{11110011}$  Exercitii Sa se transforme:

- 712 in baza 5
- 9934 in baza 2
- 721 in baza 8
- 6290 in baza 16

## 1.2 Fractii

La ce ne referim prin fractie: numere cu virgula (pot sa aiba si perioada). Cum facem conversia din baza 10 in baza b pentru o fractie?

- Pentru partea intreaga a numarului, transformarea se face fix ca ala numere intregi (ex: pentru 167,89(3) transformam 167 in baza b ca la numere intregi, punem virgula si separat facem transformarea pentru ce e dupa virgula)
- Pentru partea fractionara, avem: perioada si neperioada.  
Mai intai transformam in fractie:
  - la numarator punem: partea fractionara minus neperioada

- la numitor punem: atati de 9 cate cifre are perioada, urmati de atati de 0 cate cifre are neperioada

Ca sa fie mai usor de inteleas procesul vom lua:

**Exemplul 1** : Transformati  $18,4(8)$  din baza 10 in baza 8.

$$(18, 7(45))_8 = ?$$

Facem conversia lui 18 in baza 8:

$$18 : 8 = 2 \text{ rest } 2$$

$$2 : 8 = 0 \text{ rest } 2$$

0 - ne oprim

$$\Rightarrow (18)_8 = \overline{22}$$

Facem conversia lui  $0,4(8)$  in baza 8 :  $\frac{48-4}{90} = \frac{44}{90} = \frac{22}{45}$

Acum inmultim succesiv cu baza b:

$$\frac{22}{45} * 8 = \frac{176}{45} = 3 + \frac{41}{45} \text{ (retinem ca aici partea intreaga a fost 3)}$$

$$\frac{41}{45} * 8 = \frac{328}{45} = 7 + \frac{13}{45} \text{ (retinem ca aici partea intreaga a fost 7)}$$

$$\frac{13}{45} * 8 = \frac{104}{45} = 2 + \frac{14}{45} \text{ (retinem ca aici partea intreaga a fost 2)}$$

$$\frac{14}{45} * 8 = \frac{112}{45} = 2 + \frac{22}{45} \text{ (retinem ca aici partea intreaga a fost 2)}$$

Am ajuns inapoi la fractia  $\frac{22}{45} \Rightarrow$  avem o perioada.

De data aceasta concatenam partile intregi de **sus** in .

$$\Rightarrow 0,4(8)_8 = 0,(3722)$$

$$\Rightarrow 18,4(8)_8 = \overline{22,(3722)}$$

**Exemplul 2** : Transformati  $215,65$  din baza 10 in baza 16.

$$215 : 16 = 13 \text{ rest } 7 \text{ (reamintim ca } 13_{16} = D)$$

$$13 : 16 = 0 \text{ rest } 13$$

0 - ne oprim

$$\Rightarrow (215)_{16} = \overline{D7}$$

$$0,65 * 16 = 10,4 \text{ retinem ca partea intreaga este } 10_{16} = A$$

$$0,4 * 16 = 6,4 \text{ retinem ca partea intreaga este } 6$$

0,4 - se repeta deja aceasta valoare => ne oprim si avem perioada pe 6

$$0,65_{16} = 0, A(6)$$
$$\Rightarrow (215, 65)_{16} = \overline{D7, A(6)} \text{ Exercitii Sa se transforme:}$$

- 82,61(3) in baza 8
- 6,(18) in baza 2
- 99,1(5) in baza 5
- 0,6(7) in baza 16

## 2 [IMPORTANT] Transformarea din baza b in baza 10

Cand avem de transformat din baza b in baza 10, impartim din nou numarul in 2 sectiuni:

- partea intreaga : plecam de la sfarsitul numarului, indexand de la 0
- partea fractionara: daca avem perioada transformam in fractie, daca nu, indexam de la -1 descrescator

**Exemplul 1:** Transformati 110110 din baza 2 in baza 10

- Plecam de la sfarsitul numarului cu un index de la 0:  
 $110110_0 -> 11011_10_0 -> 1101_21_10_0 -> \dots -> 1_51_40_31_21_10_0$
  - Inmultim fiecare cifra cu  $2^{index}$  :  
 $(\overline{110110})_2^{-1} = 1*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0 = 54$
- $$\Rightarrow (\overline{110110})_2^{-1} = 54$$

**Exemplul 2:** Transformati A25,1C(4) din baza 16 in baza 10.

- Plecam de la sfarsitul numarului cu un index de la 0:  
 $A25_0 -> A2_15_0 -> A_22_15_0$

- Înmultim fiecare cifra cu  $16^{index}$ :

$$(\overline{A25})_{16}^{-1} = A*16^2 + 2*16^1 + 5*16^0 = 10*16^2 + 2*16^1 + 5*16^0 = 2597$$

- Luam acum partea fractionara si o transformam in fractie (numator=partea fractionara-neperioada, numitor=atati de F (baza in care suntem -1 = 16-1) cate cifre are perioada urmatoare de atati de 0 cate cifre are neperioada):

$$(\overline{0,1C(4)})_{16}^{-1} = \left(\frac{\overline{1C4}-\overline{1C}}{\overline{F00}}\right)^{-1}_{16} = \frac{(\overline{1C4})_{16}^{-1} - (\overline{1C})_{16}^{-1}}{(\overline{F00})_{16}^{-1}} = \frac{1*16^2 + C*16^1 + 4*16^0 - 1*16^1 + C*16^0}{F*16^2 + 0*16^1 + 0*16^0} =$$

$$\frac{1*16^2 + 12*16^1 + 4*16^0 - 1*16^1 - 12*16^0}{F*16^2 + 0*16^1 + 0*16^0} = \frac{424}{3840} = 0,11041(6)$$

$$\Rightarrow (\overline{A25})_{16}^{-1} = 2597,11041(6)$$

**Exemplul 3:** Să se scrie în baza 10 următoarele numere:

- $(\overline{1101})_2^{-1} = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 13$
- $(\overline{A2C})_{16}^{-1} = 10 \cdot 16^2 + 2 \cdot 16^1 + 12 \cdot 16^0 = 2604$
- $(\overline{12.4})_5^{-1} = 1 \cdot 5^1 + 2 \cdot 5^0 + 4 \cdot 5^{-1} = 7 + \frac{4}{5} = \frac{39}{5} = 7.8$

**Exemplul 4:** Să se scrie în baza 10 următoarele numere:

- $(\overline{1010001011})_2$
- $(\overline{1001.1001})_2$
- $(\overline{513})_6$
- $(\overline{C1D})_{16}$
- $(\overline{ABC.ABC})_{16}$

### 3 [IMPORTANT] Trecerea din baza 2 într-o bază care este putere a lui 2 (și invers)



Fie n un număr reprezentat în baza 2 pe care vrem să-l reprezentăm în baza  $2^k$ . Putem grupa reprezentarea binară a numărului în bucăți de lungime k. Scriem fiecare bucătă în baza  $2^k$ , concatenam rezultatele și obținem reprezentarea în baza  $2^k$ .

**Exemplul 1:** Să se convertească  $(\overline{11011.101})_2$  în baza 16.

Grupăm numărul în bucăți de cate 4 deoarece  $16 = 2^4$ . Pentru partea întreagă vom începe împărțirea de la dreapta spre stânga, iar pentru partea fractionară vom începe de la stânga la dreapta. Astfel, vom obține gruparea: 0001 1011.1010. Observăm că se completează cu 0 pentru a obține bucătile de lungime 4. Acum transformăm bucătile în baza 16.

$$(\overline{0001})_2 = (\overline{1})_{16},$$

$$(\overline{1011})_2 = (\overline{B})_{16},$$

$$(\overline{1010})_2 = (\overline{A})_{16}.$$

$$\text{Prin urmare } (\overline{11011.101})_2 = (\overline{1B.A})_{16}$$

Pentru a reprezenta un număr scris într-o bază  $b = 2^k$  în baza 2, aplicam

procedeul invers. Luăm fiecare cifră din reprezentarea în baza b a numărului, o reprezentăm în baza 2, iar la final concatenăm ”bucătile” obținute.

**Exemplul 2 :** Să se convertească  $(\overline{1A.C})_{16}$  în baza 2.

Deoarece  $16 = 2^4$  știm că fiecare cifră din reprezentarea numărului în baza 2 va avea lungimea 4 (se completează cu 0-uri acolo unde este cazul). Luăm fiecare cifră și o scriem în baza 2.

$$(\overline{1})_{16} = (\overline{0001})_2,$$

$$(\overline{A})_{16} = (\overline{1010})_2,$$

$$(\overline{C})_{16} = (\overline{1100})_2,$$

Reprezentarea lui  $(\overline{1A.C})_{16}$  în baza 2 este concatenarea celor 3 rezultate obținute anterior, adică  $(\overline{00011010.1100})_2 = (\overline{11010.11})_2$  (am eliminat 0-urile redundante).

**Exemplul 3 :** Sa se transforme 6A,D din baza 16 in baza 2.

Fieindca  $16=2^4$  fiecare cifra din numarul in baza 16 reprezinta 4 cifre din numarul in baza 2.

$$(\overline{6A,D})_{16} = ?$$

$$6_{(16)} = \overline{0110}_{(2)}$$

$$A_{(16)} = \overline{1010}_{(2)}$$

$$D_{(16)} = \overline{1101}_{(2)}$$

$$\Rightarrow (\overline{6A,D})_{(16)} = \overline{01101010,1101}_{(2)} = \overline{1101010,1101}_{(2)}$$

**Exemplul 4 :** Transformati 1011001,101 din baza 2 in baza 8.

Cum  $8 = 2^3$  impart in bucatele de cate 3 cifre (pornind de la virgula) si adaug 0-uri la inceput si sfarsit daca este cazul.

$$\overline{1011001,101}_2 = ?$$

$$001_{(2)} = \overline{1}_{(8)}$$

$$011_{(2)} = \overline{3}_{(8)}$$

$$001_{(2)} = \overline{1}_{(8)}$$

$$101_{(2)} = \overline{5}_{(8)}$$

$$\Rightarrow (\overline{1011001}, \overline{101})_{(2)} = \overline{131}, \overline{5}_{(8)}$$

**Exemplul 5 :** Să se reprezinte în baza 2 următoarele numere:

- $(\overline{47.3})_8$
- $(\overline{113.3})_4$
- $(\overline{ABC.ABC})_{16}$
- $(\overline{AB.G})_{32}$

**Exemplul 6 :** Să se reprezinte numerele în următoarele baze:

- $(\overline{1011.1011})_2$  în baza 8
- $(\overline{111.111})_2$  în baza 4
- $(\overline{101010.010111})_2$  în baza 16
- $(\overline{101010.010111})_2$  în baza 32

## 4 [IMPORTANT] Operații în baza b

Operațiile aritmetice cu numerele scrise într-o bază  $b \geq 2$  oarecare se fac după reguli asemănătoare ca în baza 10, dar transportul și împrumutul trebuie să se facă la  $b$ , nu la 10. Pentru calculele de o cifră în baza  $b$  putem trece numerele în baza 10, facem acolo calculele, apoi trecem rezultatele în baza  $b$ .

### 4.1 Adunarea

**Exemplul 1 :** Să se calculeze  $(\overline{1011})_2 + (\overline{110})_2$  fără a trece prin baza 10.

$$\begin{array}{r} 1 & 0 & 1 & 1 \\ + & & 1 & 1 & 0 \\ \hline 1 & 0 & 0 & 0 & 1 \end{array}$$

Pe poziția unităților am avut  $\bar{1} + \bar{0} = \bar{1}$ , fără transport. Pe următoarea poziție am avut  $\bar{1} + \bar{1} = \bar{1}\bar{0}$  (adică numărul 2), s-a păstrat  $\bar{0}$  și s-a propagat  $\bar{1}$ ; pe următoarea poziție am avut  $\bar{0} + \bar{1} + \bar{1}$  (ultimul  $\bar{1}$  provenit din transport)  $= \bar{1}\bar{0}$ , s-a păstrat  $\bar{0}$  și s-a propagat  $\bar{1}$ ; etc.

## 4.2 Scaderea

**Exemplul 2 :** În baza 16, să se scadă BA - 9B.

$$\begin{array}{r} B \quad A \\ - \quad 9 \quad B \\ \hline 1 \quad F \end{array}$$

Pe poziția unităților avem  $\bar{A} - \bar{B} = 10 - 11 < 0$ ; de aceea, împrumutăm  $\bar{1}$  de pe poziția următoare și atunci calculul este  $1 \cdot 16 + 10 - 11 = 15 = \bar{F}$ . Pe poziția următoare avem  $\bar{B} - \bar{1} - \bar{9}$  (acel  $\bar{1}$  a fost cedat la împrumut)  $= 11 - 1 - 9 = 1 = \bar{1}$ .

## 4.3 Înmulțirea

**Exemplul 3 :** În baza 2, să se înmulțească 110.11 · 1.001.

$$\begin{array}{r} 1 \quad 1 \quad 0 \quad .1 \quad 1 \\ \cdot \quad \quad 1 \quad .0 \quad 0 \quad 1 \\ \hline 1 \quad 1 \quad 0 \quad .1 \quad 1 \\ 1 \quad 1 \quad 0 \quad .1 \quad 1 \\ \hline 1 \quad 1 \quad 1 \quad .1 \quad 0 \quad 0 \quad 1 \quad 1 \end{array}$$

Înmulțirea într-o bază oarecare se face tot după reguli asemănătoare ca în baza 10, dar pentru baza 2 aceste reguli se pot simplifica, deoarece singurele cifre sunt  $\bar{0}$  și  $\bar{1}$ , care desemnează respectiv 0 și 1, care sunt factor anulator, respectiv element neutru, la înmulțire; înmulțirea cu  $\bar{0}$  presupune scrierea unui rând de  $\bar{0}$ -uri, care nu contează la adunare și se pot omite, iar înmulțirea cu un  $\bar{1}$  revine la a scrie o copie a deînmulțitului; aşadar, pentru a face înmulțirea, este suficient să parcurgem înmulțitorul de la dreapta spre stânga și pentru fiecare  $\bar{1}$  întâlnit să mai scriem o copie a deînmulțitului, cu cifra unităților aliniată la acel  $\bar{1}$ , iar în final să adunăm rândurile scrise - ceea ce am făcut mai sus; în final, numărul de zecimale ale produsului este sună

numerelor de zecimale ale factorilor, la fel ca în cazul bazei 10.

#### 4.4 Impartirea

Exemplul 4 : Impartiti 10100,011 la 11 in lucrand cu operatiile in baza 2.

A handwritten binary division problem on a dotted grid background. The dividend is 10100,011. The divisor is 11. The quotient is 110,110 (01). The remainder is 1. A yellow arrow points from the first 1 in the quotient down to the first 1 in the remainder. A yellow circle highlights the remainder 1.

$$\begin{array}{r} 10100,011 \\ \overline{11} \end{array} \quad \begin{array}{r} 11 \\ \overline{110,110(01)} \\ -100 \\ \hline 11 \\ -11 \\ \hline 1 \end{array}$$

- Ca la impartirea normala, incep din stanga. Ma uit la prima cifra, clar 11 nu se cuprinde in 1 deci mai adaug o cifra din numar.
- Acum am 10 si 11. Clar 11 nu se cuprinde in 10 deci mai adaug o cifra.

- 101 și 11, acum putem să facem împărțirea. Fiind în baza 2, lucrurile stau destul de simplu: catul ori e 0 ori e 1. În cazul nostru catul e 1 și restul 10.
- Adaug următoarea cifra din număr și am 100 și 11.  $\Rightarrow$  catul 1, restul 1.
- Adaug următoarea cifra care e 0. 11 nu se cuprinde în 10  $\Rightarrow$  catul 0, restul 10.
- Înainte de a adăuga următoarea cifra, observ că întâlnesc virgula, pe care o adaug și la rezultat. Abia acum pot adăuga urmatorul 0.  $\Rightarrow$  100 împărțit la 11, cat 1 rest 1.
- Adaug următoarea cifra.  $\Rightarrow$  11 împărțit la 11, cat 1 rest 0.
- Adaug următoarea cifra, care este 1. 11 nu se cuprinde în 1  $\Rightarrow$  cat 0 rest 1.
- S-a terminat numarul, dar nu s-a terminat și împărțirea. Adaugam cati de 0 avem nevoie pana ajungem la restul 0 sau gasim perioada. Asadar, 10 împărțit la 11  $\Rightarrow$  cat 0 rest 10.
- Mai adaug un 0  $\Rightarrow$  100 împărțit la 11, cat 1 rest 1. Am ajuns înapoi la 1  $\Rightarrow$  avem perioada pe 01.

**Exercițiul 5 :** Să se efectueze următoarele calcule:

- $(\overline{1010})_2 + (\overline{1111})_2$
- $(\overline{F3})_{16} + (\overline{AB})_{16}$
- $(\overline{12})_8 + (\overline{46})_8$
- $(\overline{1000})_2 - (\overline{1})_2$

**Exercițiul 6 :** Să se efectueze următoarele calcule:

- $(\overline{110})_2 \cdot (\overline{10101})_2$
- $(\overline{AB})_{16} \cdot (\overline{3})_{16}$

## References

- [1] Dumitru Daniel Drăgulici. *Curs.*
- [2] Larisa Dumitrache *Tutoriat 2019*

## Tutoriat 2 - Arhitectura sistemelor de calcul

Stan Bianca-Mihaela, Stancioiu Silviu

November 2020

**ORICINE ESTE IMPOZIBIL SA FACI SI MATERIA DE  
SEMINAR SI CEA DE LABORATOR INTR-UN SINGUR  
TUTORIAT**

**TUTORII DE ASC:**



# 1 Reprezentarea numerelor în calculator

## 1.1 Noțiuni generale

Într-un calculator nu putem reprezenta numerele aşa cum o facem în matematică (sau cel puțin nu o putem face într-un mod performant), aşa că există câteva convenții legate de reprezentarea numerelor în calculator.

În calculator numerele sunt reprezentate binar folosind un număr finit de biți (un bit poate 1 sau 0). De regulă se folosește un multiplu de 8 biți pentru a reprezenta un număr.

Considerăm următoarele operații hardware la nivel de bit:

$a$	$b$	$a b$ (or)	$a \& b$ (and)	$a \wedge b$ (xor)	$a$	$\sim a$ (not)
0	0	0	0	0	0	1
0	1	1	0	1	1	0
1	0	1	0	1		
1	1	1	1	0		

Pe un număr reprezentat în binar pe un număr FINIT de biți putem aplica operații logice pe biți: *or*, *and*, *xor*, etc. Ele se efectuează apilcând bit cu bit operațiile logice respective, conform tabelului anterior.

Exemplu pentru operația de *and* logic pentru două numere reprezentate pe 8 biți:

$$\begin{array}{r} 11000101 \\ 01000110 \\ \hline 01000100 \end{array} \quad \&$$

Pe lângă operațiile logice pe biți, putem aplica și operațiile aritmetice binare (cele discutate în tutoriatul anterior, adica adunare, scadere, etc) asupra unui număr reprezentat binar pe un număr finit de biți.

În cazul adunării, dacă trebuie să facem transport pe o poziție mai mare decât numărul de biți pe care îl avem la dispoziție, acel transport se pierde. Iar la scădere, dacă e nevoie să ne împrumutăm de la o poziție mai mare decât numărul de biți pe care îl avem la dispoziție, atunci împrumutul se face "automat".

Exemplu pentru adunarea și scăderea a două numere reprezentate pe 8 biți:

$$\begin{array}{r} 11000101 \\ 01000110 \\ \hline 00001011 \end{array} \quad \oplus \quad \begin{array}{r} 00001011 \\ 01000110 \\ \hline 11000101 \end{array} \quad \ominus$$

Putem aplica și operații unare asupra acestor numere (complement față de 1 și

complement față de 2).

Complementul față de 1 înseamnă ca luăm fiecare bit din reprezentarea numărului și îl negăm (adică dacă e 1 îl transformăm în 0, iar dacă e 0 îl transformăm în 1).

Complementul față de 2 înseamnă ca prima oară calculăm complementul față de 1 al numărului, apoi adunăm 1 aritmetic (adunare ca în exemplele anterioare). Exemple pentru un număr reprezentat pe 8 biți:

$$\begin{array}{r} \text{~} \quad \boxed{01001100} \\ \text{~} \quad \hline 10110011 \end{array} \qquad \begin{array}{r} \text{~} \quad \boxed{01001100} \\ \text{~} \quad \hline 10110100 \end{array}$$

Observăm că pentru a scădea două numere, putem face suma dintre primul număr și complementul față de 2 al celui de-al doilea număr (TODO: luați un exemplu și verificați că e adevărat). De aceea, deși sunt diferite instrucțiunile pentru adunare și scădere din limbajul de asamblare, intern sunt realizate de același circuit, numit sumator.

## 1.2 Reprezentarea numerelor naturale ca întregi fără semn

În limbajul C, această reprezentare este folosită în cazul tipuilor de date întregi însotite de "unsigned":

unsigned char, unsigned short, unsigned int, unsigned long, unsigned long long  
Fie un tip de numere naturale pe  $n$  biți, atunci acel tip de date poate stoca toate numerele naturale cuprinse între 0 și  $2^{n-1}$  inclusiv.

Exemplu pe 8 biți: cea mai mică valoare care poate fi stocată pe 8 biți folosind această reprezentare este 0 (adică 00000000 intern), iar cea mai mare valoare care poate fi reprezentată este 255 (adică 11111111 intern, practic este cel mai mare număr care poate fi reprezentat cu 8 biți). După cum am discutat mai devreme (faptul că transportul se pierde după ultimul bit), rezultă că operațiile cu aceste numere se fac modulo  $2^n$ .

Exemplu: dacă avem numărul 255 (ambele stocate ca numere naturale fără semn pe 8 biți,  $[255]_8 = \overline{11111111}$ ) și adunăm la el  $[1]_8 = \overline{00000001}$ , operația internă care se execută este:  $11111111 + 00000001$ . Se va transportă un bit până la ultimul bit, unde se pierde, deci ne va rămane 00000000. Adică fix 0. 0 este egal cu 256 mod 256 (pentru că  $256 = 2^8$ ).

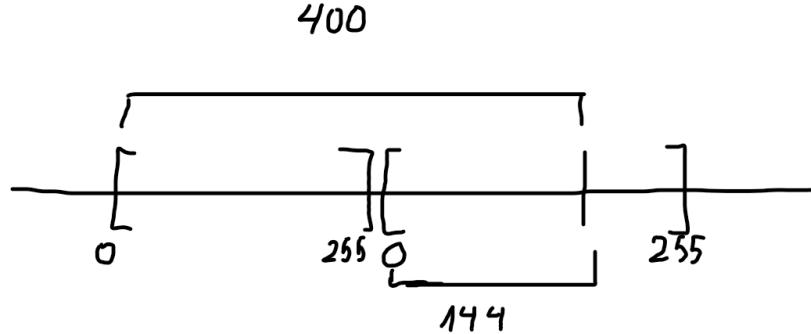
Alt exemplu: tot pe 8 biți lucrăm și avem de adunat 128 cu 129 ( $[128]_8 = \overline{10000000}$ ) și adunăm la el  $[129]_8 = \overline{10000001}$ , adică  $10000000 + 10000001$ . Folosind regulile de calcul de mai sus ne va da 00000001, adică 1.  $1 = (128 + 129) \text{ mod } 256 = 257 \text{ mod } 256$ .

Observație:

Atunci când suma a două numere reprezentate în acest mod depășește val-

oarea maximă suportată de tipul de date, rezultatul va fi strict mai mic decât cel puțin unul din termenii adunării. Datorită acestei proprietăți putem verifica dacă s-a produs overflow în timpul adunării.

Alt exemplu: tot pe 8 biți avem  $x = 200$  și  $y = 200$ . Dacă facem  $z = x + y$ , vom avea că  $z$  are valoarea 144. Să vedem și în binar ce se întâmplă: avem  $x, y = 11001000$ , deci  $x + y = 11001000 + 11001000 = 10010000$  deoarece s-a pierdut transportul. Putem să și vizualizăm acest exemplu:



Practic pentru orice operație ne putem imagina că avem o axă imaginată cu intervale de valori cuprinse între 0 și  $2^n - 1$ . Fiecare operație ne ducă mai în stânga sau în dreapta axei. Punctul în care ajungem este rezultatul calculului, evident, o valoare între 0 și  $2^n - 1$  inclusiv.



Secvența de cod:  $x = 255; ++x;$  face ca  $x$  să devină 0 (cum am văzut și într-un exemplu anterior)

Secvența de cod:  $x = 0; -x;$  face ca  $x$  să devină 255

Secvența de cod: `for (x = 0; x < 256; ++x)` este un ciclu infinit deoarece  $x$  mereu va fi mai mic decât 256 (256 nu poate fi reprezentat pe 8 biți folosind această reprezentare), deci condiția  $x \neq 256$  este mereu adevărată. Practic se va cicla la infinit printre valorile 0, 1, ... 255.

De notat că la efectuarea testului  $x < 256$  operanții sunt convertiți la tipul *int*, iar comparația se face în cadrul tipului *int*.

### 1.3 Reprezentarea numerelor întregi în complement față de 2

În limbajul C, reprezentarea în complement față de 2 se folosește în cazul tipurilor întregi ne-însotite de 'unsigned':

char, signed char, short, signed short, int, signed int, long, signed long, long

long, signed long long. Un tip de date cu semn pe  $n$  biți poate ține numere întregi cu valori cuprinse între  $-2^{n-1}$  și  $2^{n-1} - 1$ . Dacă avem un număr cuprins în acest interval și vrem să-l reprezentăm folosind această reprezentare, avem două variante:

1. Dacă numărul este pozitiv, atunci pur și simplu scriem reprezentarea lui binară.
2. Dacă numărul este negativ scriem complementul său față de 2.

Exemplu: În limbajul C, pentru tipul char avem:  $n = 8$ .

El poate ține valori întregi cuprinse între -128 și 127. Modul de reprezentare a câtorva valori:

127: 01111111

126: 01111110

1: 00000001

0: 00000000

-1: 11111111

-2: 11111110

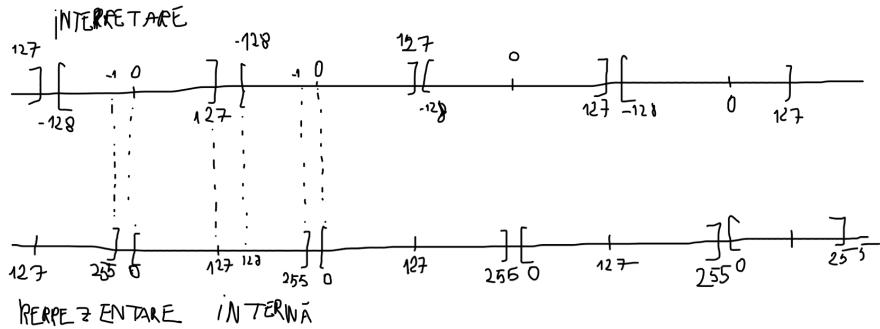
-127: 10000001

-128: 10000000

Observație:

Numerele negative au valoarea celui mai semnificativ bit egală cu 1, pe când numerele pozitive au valoarea aceluui bit egală cu 0.

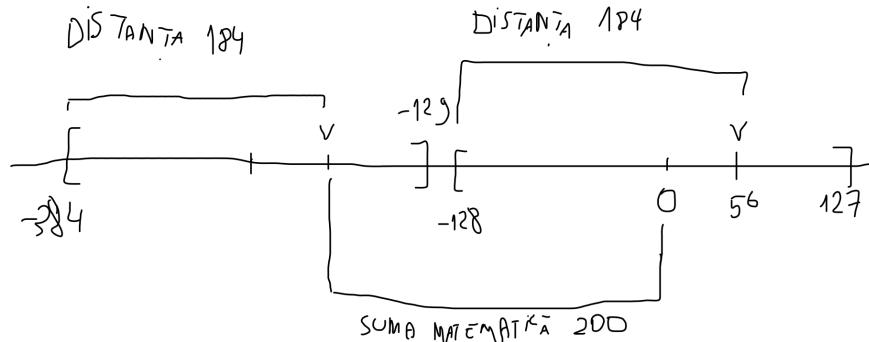
Asemănător cu reprezentarea intuitivă de la numerele unsigned, aici ne putem imagina două axe infinite de "intervale cu valori întregi". Una cu valori cuprinse între  $-2^{n-1}$  și  $2^{n-1} - 1$ , iar cealaltă cu valori cuprinse între 0 și  $2^n - 1$ . Practic avem o bijectie astfel: pe axa cu numere negative numerele între 0 și  $2^{n-1} - 1$  sunt mapate tot pe același valori de pe cealaltă axă, iar numerele negative sunt mapate pe "intervalul" dintre  $2^{n-1}$  și  $2^n - 1$  pe cealaltă axă. Prima axă este cea a interpretării (adică valoarea pe care o interpretăm noi că o are numărul), iar cea de a doua axă este reprezentarea lui în calculator. (adică dacă transformăm numerele de pe cea de a doua axă în baza 2, obținem reprezentarea lor internă în calculator). Pentru 8 biți avem:



Operațiile se fac tot  $2^n$ , dar translatat (deplasamentul se măsoară față de începutul intervalului  $\{-2^{n-1}, \dots, 2^{n-1} - 1\}$ ).

Exemplu: În limbajul C, avem:

```
char x, y, z; /* n = 8, M = {-128, ..., 127} */
x = -100; y = -100; z = x + y;
printf("%d", (int)z); /* afișează 56 */
```



Exemplu: Fie  $x = 32$ ,  $y = 41$ . Calculați  $z = x - y$  folosind reprezentarea în complement față de 2 pe 8 biți.

Avem  $n = 8$ , deci multimea valorilor pe care le putem stoca este  $M = \{-128, \dots, 127\}$ .

Deoarece  $x$  și  $y$  sunt pozitive și aparțin mulțimii de mai sus, le vom reprezenta direct în baza 2. Deci  $x = 00100000$  și  $y = 00101001$ .

Calculăm complementul lui  $y$  față de 2, pentru asta va trebui ca prima oară să calculăm complementul față de 1, deci vom avea: 11010110. Acum adunăm 1 și avem 11010111. Tocmai am obținut complementul lui  $y$  față de 2. Acum trebuie doar să adunăm  $x$  cu rezultatul obținut, avem:  $00100000 + 11010111 = 11110111$ . Acest număr înseamnă 247. Deoarece cel mai semnificativ bit

este 1, înseamnă că avem un rezultat negativ, deci pentru a afla valoarea rezultată îl scoatem pe z din următoarea ecuație:  $256 + z = 247$ . În acest caz ne va da că  $z = -9$ . Deci calculul pentru situații de genul este  $2^n + z = \text{valoare\_interpretata\_din\_binar}$ . O intuiție în acest caz ar fi să ne uităm pe diagramă și să ne gândim că 256 înseamnă 0 (și chiar asa și înseamnă după bijecția stabilită de noi). Ne gândim care este diferența între origine și rezultatul nostru, iar această diferență este chiar diferența din diagrama interpretării.

Alt mod de a interpreta acest exercițiu este să ne imaginăm că prelungim intervalul pe axa noastră infinită, facem calculul matematic, calculăm diferența dintre rezultatul nostru și începutul intervalului în care pică rezultatul, iar apoi să adunăm această diferență la capătul de început al intervalului nostru. (exemplul anterior luat din cursul lui Drăgulici). În cazul acesta nu este nevoie de aşa ceva pentru că oricum -9 se află în range-ul în care acceptăm valori.

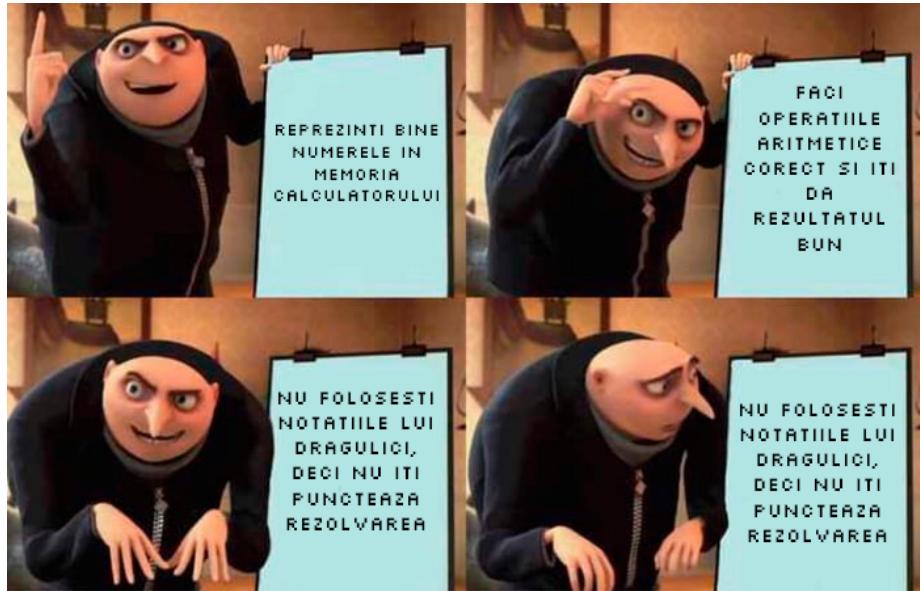
Dacă aveți la examen un exercițiu asemănător, urmați metoda de rezolvare cu  $2^n + z = \text{valoare\_interpretata\_din\_binar}$ , nu a doua metodă de rezolvare pe care v-am prezentat-o.

**Exemplu:** În limbajul C considerăm următoarea secvență de cod:

```
char x; /* p = 8 */
short y; /* q = 16 */
x = -127; /* se reprezintă -127 ca întreg cu semn pe 8 biți */
y = x; /* se extinde reprezentarea din x de la 8 biți la 16 biți, prin propagarea
bitului de semn*/
```

Verificați că valoarea reprezentată în y este tot -127.

Avem:  $[-127]_8^s = (256 - 127)_2^8$  (deoarece  $-127 < 0$ )  $= (129)_2^8 = 10000001$ . (Acesta sunt notațiile lui Drăgulici pentru treaba asta, vom discuta mai multe pe tema asta când ne vom apuca de rezolvat modele de examen.). Prin propagarea bitului de semn la 16 biți, obținem:  $111111110000001 = ((2^{16} - 1) - (2^7 - 1) + 1)_2^{16} = (65535 - 127 + 1)_2^{16} = (65409)_2^{16}$ . Aceasta este reprezentarea ca întreg cu semn a unui număr  $z < 0$ , deoarece bitul 15 (de semn) este 1.  
Atunci  $[z]_{16}^s = (65536 + z)_2^{16} = (65409)_2^{16}$ , deci  $65536 + z = 65409$ , deci  $z = -127$ .



Exercitiul 1: [RESTANTA SEPT 2020] Calculati  $z=x-y$ , ( $x=112$ ,  $y=37$ ) folosind reprezentarea in complement fata de 2 pe 8 biti. Se vor explicita reprezentarile lui  $x$  si  $y$ , complementul fata de 1 si cel fata de 2 al reprezentarii lui  $y$ , obtinerea reprezentarii lui  $z$ , interpretarea acestia ca numar in baza 10.

$$[x]_8 = [112]_8 = 01110000$$

$$[y]_8^s = [37]_8 = \overline{00100101}$$

complementul fata de 1 al lui  $y$  este:  $\overline{11011010}$

adaugam 1 la complementul fata de 1 si obtinem complementul fata de 2:  
 $1 + \overline{11011010} = \overline{11011011}$

$$\Rightarrow [z]_8^s = \overline{01001011} = [2^6 + 2^4 + 2^3 + 2^1 + 2^0]_8 = [75]_8$$

$\Rightarrow$  reprezentarea lui  $z$  ca numar in baza 10 este: 75

Exercitiul 2: Calculati  $z=x-y$  ( $x=12$ ,  $y=67$ ) folosind reprezentarea in complement fata de 2 pe 8 biti.

\* incercati si cealalta varianta

## 2 MIPS (Ne pare rău seria 15, nu știm x86 assembly, chiar am vrea sa știm și sa va putem ajuta să înțelegeți)



### 2.1 Noțiuni generale

Limbajul de programare MIPS assembly este modul prin care programăm un procesor care are arhitectura MIPS. Această arhitectură este una de tip RISC (Reduced Instruction Set Computer), spre deosebire de arhitectura x86 care este de tip CISC (Complex Instruction Set Computer). Procesoarele de tip MIPS sunt mici, simple și consumă puțin curent, de aceea acestea se găsesc de regulă pe dispozitive precum: teste de sarcină, frigidere, mașini de spălat, etc. Procesoarele MIPS sunt procesoare pe 32 de biți (4 bytes).

Intr-adevăr, probabil vor fi puține situații (most likely nu vor fi deloc) în care veți coda în MIPS în cariera voastră de programatori, dar înțelegerea acestui limbaj vă va prinde foarte bine pe viitor. MIPS este un limbaj de asamblare foarte ușor (comparativ cu x86 sau x64), iar dacă îl veți învăța vă va fi usor după să treceți la limbaje de asamblare precum x86 sau x64 assembly (pe acestea trebuie să le știți dacă vreți să lucrați la antiviruși, să analizați alte programe, să modătați jocuri, etc).

### 2.2 Lucrul cu registrii

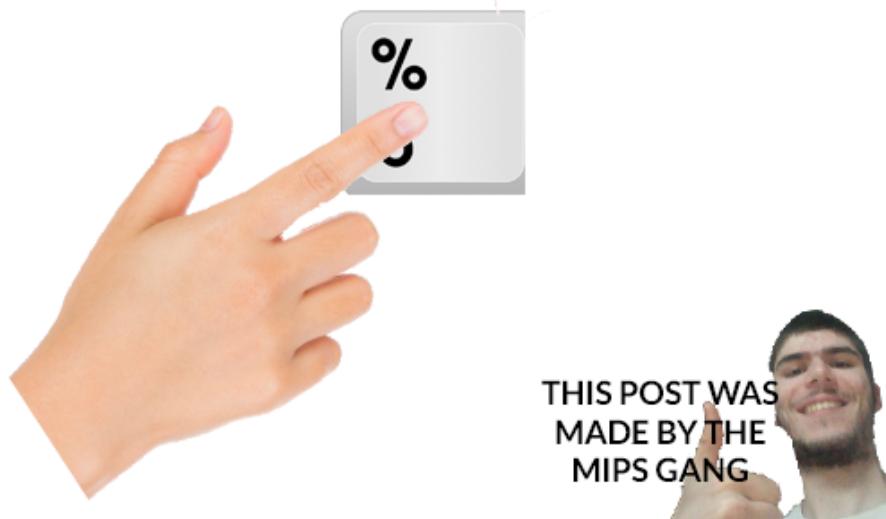
În limbiile de programare high level precum C/C++ sau Python suntem obișnuiți să lucrăm cu variabile pentru a realiza operații/ a stoca date. În limbajele de asamblare, lucrul cu memoria nu este atât de "direct". Aici pentru a lucra cu memorie, această memorie trebuie mai întâi copiată în registrii de pe procesor,

apoi se fac operațiile folosind regiștrii de pe procesor, iar la final se copiază înapoi în memoria RAM rezultatele. (cam asta ar fi explicația la lucrul cu regiștrii pe scurt, vom vedea în cateva exemple mai jos cum se lucrează)

### Regiștri MIPS

Când scriem cod în MIPS, prefixăm cu \$ fiecare registru pe care îl folosim (în x86 at&t assembly prefixăm cu %).

**IMAGINE PREFIXING YOUR  
REGISTERS WITH %  
INSTEAD OF \$**



În MIPS avem următorii regiștri:

- \$zero
- \$at
- \$a0-\$a4
- \$v0-\$v1 (în \$v0 vom pune codul pentru apelurile de sistem)

- \$t0-\$t7, \$t8, \$t9 (sunt registri temporari, de regulă cu ei vom lucra în main)
- \$s0-\$s8 (registri salvați, de regulă vom lucra cu ei în interiorul procedurilor)
- \$sp, \$fp (registri pentru controlul stivei)
- \$ra

### Tipuri de date MIPS

Acstea sunt folosite pentru a declara date ”statice” la începutul programului. Ne putem imagina că ele sunt stocate direct în RAM, iar noi trebuie să le copiem în registri pentru a lucra cu ele, iar apoi din registri să întoarcem înapoi valorile în ”memoria RAM”. Tipurile de date sunt prefixate cu caracterul punct '.' în MIPS.

Tipurile:

- .word - tip de date pe 32 de biți (4 bytes), folosit pentru a stoca întregi
- .byte - tip de date pe un byte (8 biți) folosit pentru a stoca caractere sau valori booleene
- .ascii
- .asciiz - folosit pentru a ține siruri de caractere. Pe acesta îl vom folosi, nu pe .ascii
- .space *dim* - declară o zonă liberă de memorie de dimensiunea *dim* octeți.

Exemple de date declarate:

Datele se declară în felul următor:  
nume: tip valoare\_implicită

Exemplu:

```
x:.word 5
y:.byte 'a'
```

```
str:.asciiiz "mesaj"  
sum:.space 4  
res_byte:.space 1
```

Instructiuni pentru transferarea datelor din memorie in registri (si invers):

Instructiuni pentru incarcarea valorilor in registri:

lw \$reg, mem\_word (load word, incarcă in registrul *reg*, valoarea aflată in memorie in *mem\_word*)

lb \$reg, mem\_byte (la fel ca mai sus, doar că aici este load byte, deci încarcă un singur byte, nu 4)

la \$reg, adr\_mem (load address, incarcă in registrul *reg* adresa de memorie a lui *adr\_mem*. Deci instructiunile de mai sus incără in registri valoarea de la adresa de memorie, iar acestă instrucțiune încarcă adresa de memorie. Instrucțiunea aceasta este utilă dacă vrem să iterăm printr-un array. Încarcăm adresa de memorie a array-ului într-un registru, iar apoi incrementăm valoarea din registru. Pentru a accesa valorile din array, folosim *lw* și *lb*, iar ca parametru 2 punem adresa de memorie incrementată)

li \$reg, const (încarcă o valoarea constantă *const* în registrul *reg*)

Instructiuni pentru salvarea valorilor din registri in memorie:

sw \$reg, mem (store word, salvează valoarea din registrul *reg* la adresa *mem*)

sb \$reg, mem (store byte, la fel ca mai sus, doar că aici salvează un byte, nu un word)

Exemple de folosire a acestor instructiuni:

```
lw $t0, x # încarcă in registrul t0 valoarea din x (adică 5)  
lb $t1, y # încarcă in registrul t1 valoarea din y (adică 'a')  
la $t2, str # încarcă in registrul t2 adresa de memorie a lui str  
li $t3, 15 # încarcă in registrul t3 valoarea constantă 15  
sw $t4, sum # salvează valoarea din registrul t4 la adresa de memorie a lui sum  
sb $t5, res_byte # salvează valoarea din registrul t5 la adresa de memorie a lui res_byte (în cazul asta e un singur byte, nu 4 cum erau la sw)
```

### 2.3 Instrucțiuni aritmetice

add/ addu \$dest, \$src1, \$src2 (aduna numerele din registrii *src1* și *src2* și salvează rezultatul în registrul *dest*. Diferența dintre cele două este că add ne poate ajut să detectăm dacă a fost overflow la adunare printr-un trap, pe când addu este folosit pentru numere unsigned, deci putem verifica dacă rezultatul este mai mare decât cele două numere, iar dacă nu este, e clar că a fost overflow. Nu ne vom concentra pe aceste detalii la tutoriat și probabil vom folosi add în mai toate situațiile)

addi \$dest, \$src, const (adună numărul din registrul *src* cu constanta *const* și salvează rezultatul în registrul \$dest)

Observație: Putem face add \$t0, \$t0, \$t1 direct pentru a face operația  $t_0 = t_0 + t_1$ . Un procesor nu poate face această operație direct, deci în spate, compilatorul de MIPS face niște "magie neagră".

sub/ subu \$dest, \$src1, \$src2 (asemanător cu add/ addu. sub realizează scăderea dintre valorile din registrul *src1* și *src2*, iar rezultatul îl salvează în registrul *dest*. subu este pentru unsigned)

mul \$dest, \$src1, \$src2 (realizează operația  $dest = src1 \cdot src2$ )

div \$dest, \$src1, \$src2 (realizează operația  $dest = src1 \text{ div } src2$ . Adică împarte cu rest pe *src1* la *src2* și salvează cîtuț în registrul *dest*)

rem \$dest, \$src1, \$src2 (realizează operația  $dest = src1 \text{ mod } src2$ . Adică împarte cu rest pe *src1* la *src2* și salvează restul în registrul *dest*)

### 2.4 IO + apeluri sistem

Apelurile de sistem nu le putem imagina ca pe niște funcții predefinite de sistemul de operare pe care programul nostru le poate apela pentru a interacționa cu mediul în care se execută. Procesoarele MIPS nefiind folosite pe calculatoare, de regulă apelurile de sistem sunt transmise direct către hardware fără ca vreun sistem de operare să fie intermediar. De exemplu pentru un test de sarcină se poate face un apel de sistem pentru a face un beep în cazul în care sunt 2 bare la rezultat. Practic atunci când facem un apel de sistem pierdem controlul asupra programului, hardware-ul/ sistemul de operare decide când să execute acel apel de sistem, iar apoi decide când să redea controlul programului pentru a-și putea continua execuțarea (se spune execuțare, nu execuție, execuție e atunci când omori pe cineva). Ideea astă de apeluri de sistem este foarte bine să o înțelegeti de acum pentru că astă este în fiecare limbaj de asamblare, nu doar în MIPS,

iar în anul 2 va trebui să vă definiți voi propriile apeluri de sistem la cursul de "Sisteme de operare". Dacă vreți să învățați de acum mai multe despre apeluri de sistem, vă puteți uita pe laboratoarele de la cursul de "Sisteme de operare": <https://cs.unibuc.ro/pirofti/so.html>

Pentru a face un apel de sistem este nevoie să încărcăm în registrul \$v0 codul funcției de sistem (nu este ca în C să apelam o funcție după nume, aici fiecare funcție are un număr ca id, care trebuie încărcat în acel registrul.). De regulă se folosește registrul \$a0 pentru a transmite parametrii. După ce valorile necesare au fost setate, se scrie instrucțiunea *syscall*.

### Coduri pentru apeluri de sistem

1 - PRINT INT (afișează un număr întreg, se încarcă în \$a0 valoarea de afișat, în \$v0 valoarea 1 (codul pentru apelul se sistem), iar apoi se scrie *syscall*)

4 - PRINT STRING (afișează un sir de caractere, se încarcă în \$a0 adresa de memorie a sirului de caractere, în \$v0 valoarea 4 (codul pentru apelul se sistem), iar apoi se scrie *syscall*)

5 - READ INT (citește un număr de la tastatură, se încarcă în \$v0 valoarea 5 (codul pentru apelul de sistem), apoi se scrie *syscall*. După ce a reușit să citească numărul de la tastatură, sistemul ne va returna valoarea citită în registrul \$v0)

10 - EXIT (dacă facem *li v0, 10*, iar apoi *syscall* este echivalentul lui *return 0*; din C. Încheiem executarea programului.)

In tutoriatele viitoare vom aborda și alte apeluri de sistem, dar pentru acest tutoriat, aceste 4 apeluri ne sunt suficiente.

### Exemple de apeluri de sistem

```
# PRINT STRING
la $a0, str # încarc în $a0 adresa sirului de caractere afișat
li $v0, 4
syscall

# READ INT
li $v0, 5
syscall
move $t0, $v0 # în $v0 am primit valoarea citită de la tastatură, deci o copiem în
registrul $t0 pentru a lucra cu ea. Sintaxa pentru instrucțiunea move este: move
```

`$regd, $regs`. Instrucțiunea copiază valoarea din registrul `regs` în registrul `regd`.

```
# EXIT # echivalentul lui return 0; din C
li $v0, 10
syscall
```

## 2.5 Structura unui program în MIPS

```
1  # Comentariile în MIPS încep cu #
2  # Ele sunt ignorate de către compilator
3  # Ele sunt valabile până la sfârșitul rândului
4  .data
5  |    # zonă pentru declararea datelor
6  .text
7  |    #cod pentru proceduri
8  main:
9  |    #cod main
10 |    li $v0, 10
11 |    syscall
```

## 2.6 Exerciții și probleme

Problema 1: Se dă un număr întreg  $n$  stocat în memorie, să se calculeze  $(n \div 3) - 2$  și să se afișeze pe ecran rezultatul.

```
1  .data
2  |    n:.word 100756      # numărul declarat în memorie
3  .text
4  main:
5
6      lw $t0, n          # incarcă în registrul $t0 valoarea din n
7      div $t0, $t0, 3     # realizează împărțirea La 3
8      |                   # observați că nu există instrucțiunea divi
9      |                   # deci pentru a realiza împărțirea La o constantă
10 |                   # este suficient ca al doilea parametru al instrucțiunii
11 |                   # să fie o constantă
12      sub $t0, $t0, 2     # scădem 2
13      |                   # din nou, nu există subi, deci putem folosi o constantă
14      |                   # pentru al doilea parametru
15      move $a0, $t0        # copiază în $a0 valoarea pe care vrem să o afișăm
16      li $v0, 1             # incarcă în $v0 codul pentru print int
17      syscall              # realizează apelul de sistem
18
19      li $v0, 10            # exit
20      syscall
```

## 2.7 Branch instructions

Sunt instrucțiuni care în urma unei condiții decid dacă să continue executarea la linia curentă sau să continue executarea la altă linie din program. Atunci când calculatorul execuțiază un program și se află la o linie de cod, el ține într-un registru un pointer către linia curentă. Acel pointer poate fi modificat, iar programul să se execute la altă linie. Structura unei instrucțiuni de branch este următoarea:  $b_{\_} \$src1, \$src2, et$ .  $b_{\_}$  este instrucțiunea (vom vedea mai jos care sunt instrucțiunile și ce condiții implică fiecare).  $\$src1$  și  $\$src2$  sunt cele două valori pe baza cărora instrucțiunea decide dacă va continua executarea de la linia de cod la care se află eticheta  $et$ .

Instrucțiunile:

bne - (branch not equal, echivalentul lui  $!=$  din C. Practic, dacă facem  $bne \$src1, \$src2, et$ , echivalentul în C ar fi  $if(src1 != src2) goto et;$ , unde  $et$  este un label pus la o linie de cod din program)

beq - (branch equal, echivalentul lui  $==$  din C)

ble - (branch less than or equal, echivalentul lui  $\leq$  din C)

blt - (branch less than, echivalentul lui  $<$  din C)

bge - (branch less than or equal, echivalentul lui  $\geq$  din C)

bgt - (branch greater than, echivalentul lui  $>$  din C)

Salt necondiționat: jump (j)

j et (sare la eticheta  $et$  direct, necondiționat)

## 2.8 Exerciții și probleme

Problema 1: Să se afișeze pe ecran toate valorile de la 0 la  $n-1$  (inclusiv) cu n citit de la tastatură.

```

1  .data
2
3      n: .space 4          # numarul nostru (alocam 4 bytes pentru el)
4      # putem folosi si word in loc
5      # (cum am facut la problema anterioara)
6      sp:.asciiz " "       # spatiu pentru a putea delimita numerele cand
7      # le afisam pe ecran
8
9  .text
10
11 main:
12
13     li $v0, 5            # READ INT
14     syscall
15     move $t0, $v0,        # copiaza in $t0 valoarea citita
16     sw $t0, n             # salveaza in memorie valoarea citita
17
18     li $t1, 0              # $t1 va fi counterul nostru, deci il initializam cu 0
19     loop:
20         beq $t1, $t0, exit # cand conterul ($t1) este egal cu n (valoarea din $t0),
21         # e clar ca am afisat toate numerele de la 0 la n-1
22         # deci ne ducem la label-ul exit pentru a inchide programul
23         move $a0, $t1        # PRINT INT (copiaza in $a0 valoarea curenta pentru a o afisa)
24         li $v0, 1            # PRINT STRING
25         syscall
26
27         li $v0, 4            # PRINT STRING
28         la $a0, sp           # Incarcă în $a0 pointer către sirul de caractere
29         # care conține un singur spatiu
30         syscall             # afisează un spatiu pe ecran pentru a putea delimita
31         # numerele
32
33         addi $t1, $t1, 1      # incrementează conterul
34         # echivalentul lui i++ din C
35         j loop               # continuă iterarea
36         # adică continua executarea de la linia
37         # unde este label-ul loop
38
39 exit:                      # daca am ajuns aici inseamna ca beq din loop
40         # ne-a adus aici, deci am afisat toate numerele de la
41         # 0 la n-1
42
43     li $v0, 10             # exit
44     syscall

```

Problema 2: Se citește  $n$  de la tastatură. Să se calculeze utilizând structura repetitivă următoarea sumă:

$$\sum_{i=1}^{n-1} (i \text{ div } 3) - 2$$

```

1  .data
2  .text
3  main:
4
5      li $v0, 5
6      syscall
7      move $t0, $v0
8
9      li $t1, 1
10     li $t3, 0
11
12     loop:
13         bge $t1, $t0, exit    # am folosit bge pentru a nu avea loop infinit
14             # în cazul în care primim o valoare mai mică decât 1
15             # de la intrare
16
17         move $t2, $t1          # calculează termenul  $a_i$  în t2
18         div $t2, $t2, 3
19         sub $t2, $t2, 2
20
21         add $t3, $t3, $t2    # îl adună pe  $a_i$  la sumă
22             # suma va fi ținută în registrul $t3
23
24         addi $t1, $t1, 1
25
26         j loop
27
28     exit:
29
30     li $v0, 1                # afișează rezultatul sumei
31     move $a0, $t3
32     syscall
33
34     li $v0, 10
35     syscall

```

Problema 3: Se citește  $n \in N$  și  $n$  numere naturale. Să se calculeze următoarea sumă:

$$\sum_{i=1}^n (a_i \text{ div } 3) - 2$$

, unde  $a_i$  este al  $i$ -ulea număr citit de la tastatură (după ce s-a citit numărul  $n$ ).

```

1  .data
2  .text
3  main:
4
5      li $v0, 5
6      syscall
7      move $t0, $v0
8
9      li $t1, 0          # de data asta pornim de la 0,
10     |                 | # nu de la 1 cum era la problema anterioara
11     li $t3, 0
12
13     loop:
14         bge $t1, $t0, exit
15
16         li $v0, 5          # citim termenul a_i și il salvăm în $t2
17         syscall
18         move $t2, $v0
19
20         div $t2, $t2, 3    # nu mai avem nevoie de valoarea citită
21         |                 | # deci putem calcula termenul general direct în $t2
22         sub $t2, $t2, 2
23
24         add $t3, $t3, $t2  # adună termenul calculat la sumă
25
26         addi $t1, $t1, 1
27
28         j loop
29
30     exit:
31
32     li $v0, 1
33     move $a0, $t3
34     syscall
35
36     li $v0, 10
37     syscall

```

Aceasta a fost prima parte din problema care s-a dat anul trecut în prima zi de Advent Of Code. În tutoriatele următoare vom mai rezolva probleme date la Advent Of Code în MIPS.

Pont: Faceți probleme de la Advent Of Code în MIPS și în Python pentru a lua 10 la examenele de laborator de la aceste materii. Link problemă: <https://adventofcode.com/2019/day/1>



## References

- [1] Dumitru Daniel Drăgulici. *Curs Arhitectura Sistemelor de Calcul*.
- [2] Larisa Dumitracă. *Tutoriat 2019*
- [3] Bogdan Macovei. *Laboratoare 2019/2020*
- [4] Paul Irofti. *Laborator Sisteme de Operare*
- [5] Advent Of Code. *Day 1 2019*

# Tutoriat 3 - Arhitectura sistemelor de calcul

Stan Bianca-Mihaela, Stăncioiu Silviu

November 2020



# 1 Formatul intern in virgula mobila

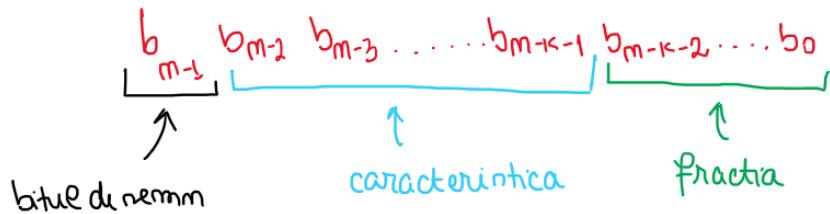
Ce vom invata in tutoriatul acesta? Care este reprezentarea interna a numerelor cu virgula (fara perioada).

Folosim standardul "IEEE 754".

In primul rand, avem 2 dimensiuni:

- $n$  = numarul total de biti
- $2k \leq n - 2$ , numarul de cifre pe care le are caracteristica in reprezentarea interna

Cum interacioneaza  $k$  si  $n$  in reprezentarea numarului?



In functie de acestei 2 parametri se mai definesc:

- $p = n - k$  (precizia = numarul de cifre din mantisa + 1)
- mantisa,  $\rho$ , a carei lungime este  $p-1$  biti
- exponentul minim :  $E_{min} = -2^{k-1} + 2$
- exponentul maxim sau BIAS :  $E_{max} = 2^{k-1} - 1$
- caracteristica :  $c = E + \text{BIAS} \Rightarrow 1 \leq c \leq 2^k - 2$

Fiecare numar va avea un  $E$  calculabil (vedem imediat cum il calculam). In functie de acesta, numerele se impart in 3 categorii:

- nereprezentabile:  $E > E_{max}$
- cu reprezentare normalizata:  $E \in [E_{min}, E_{max}]$
- cu reprezentare denormalizata :  $E < E_{min}$

NOTATIE STIINTIFICA vs. NOTATIE STIINTIFICA NORMALIZATA

- O reprezentare in virgula mobila este notatie stiintifica daca are o singura cifra inainte de virgula.
- O reprezentare in virgula mobila este in notatie stiintifica normalizata daca are o singura cifra inainte de virgula si aceasta este nenula.

Cum arata aceste reprezentari in virgula mobila?

### 1.1 FORMATUL NORMALIZAT

$$x_1 = (-1)^s * 2^E * \overline{1, \rho}$$

unde  $s = 0$  daca  $x_1$  e pozitiv si  $1$  daca  $x_1$  e negativ



### 1.2 FORMATUL DENORMALIZAT

$$x_2 = (-1)^s * 2^{E_{min}} * \overline{0, \rho} \text{ cu } \rho! = 0$$



### 1.3 $\pm 0$

$$x_3 = (-1)^s * 2^{E_{min}} * \overline{0, 0}$$



### 1.4 $\pm \infty$



## 1.5 NaN (not a number)

$x_5 :$  

## 2 Tipuri de formate

Acum ca stim ca n si k sunt variabilele cheie (in functie de ele se definesc toate celelalte) putem sa identificam anumite tipuri de formate:

### 2.1 Formatul single - pentru numere normalize

Numarul trebuie sa apartina intervalului  $[2^{-126}, 2^{127}]$ .

- n=32
- k=8 =>  $E_{max} = BIAS = 2^{k-1} - 1 = 127$  si  $E_{min} = -126$
- p=n-k=24 =>  $|\rho| = 23$  (numarul de cifre al mantisei)

### 2.2 Formatul double - pentru numere normalize



Numarul trebuie sa apartina intervalului  $[2^{-1022}, 2^{1023}]$ .

- n=64
- k=11 =>  $E_{max} = BIAS = 2^{k-1} - 1 = 1023$  si  $E_{min} = -1022$

- $p=n-k=53 \Rightarrow |\rho| = 52$  (numarul de cifre al mantisei)

**Exemplul 1 :** Transformati numarul 9,75 in format single.

Suntem in formatul single, deci  $n=32$ ,  $k=8$  si  $p=24$ .

$x=9,75$

E clar ca  $x \notin \{\pm 0, \pm \infty, NaN\}$ .

$9=(1001)_2$

$0,75 * 2 = 1,50$

$0,50 * 2 = 1$

am ajuns la 0, ne oprim

$\Rightarrow 9,75 = (\overline{1001,11})_2$

Ce facem mai departe? Noi vrem sa avem doar un singur 1 inainte de virgula.

Deci mutam virgula cu 3 pozitii mai in fata.

$1001,11 = 1,00111 * 2^3$

Identificam acum variabilele:

- Cine e E?  $E=3$  (puterea la care e 2 cand numarul este normalizat)
- Cine e mantisa?  $00111$  (ce e dupa virgula)
- Ce valoare are s, bitul de semn? Semnul lui x e pozitiv, deci  $s=0$
- Cine este caracteristica, c?  $c=E+BIAS=3+127=130$

Inainte sa scriem numarul in reprezentarea sa interna, mai avem de facut 2 verificari:

- TESTUL DE OVERFLOW / UNDERFLOW

$$E \in [E_{min}, E_{max}] \Leftrightarrow E \in [-126, 127]$$

In cazul nostru,  $E=3, -126 \leq 3 \leq 127$  deci se verifica. ✓

- TESTUL DE ROTUNJIRE

lungimea mantisei  $\leq p - 1$

In cazul nostru, lungimea mantisei este  $|\rho| = 5$  si  $5 \leq 23$  ✓

Vreau acum sa scriu reprezentarea interna a lui x in formatul single.

s: 0 ( $x > 0$ )

c:  $(130)_2 = \overline{10000010}$  (daca c nu avea  $k=8$  cifre, completam cu 0-uri)

$\rho$ :  $00111...0$  (am completat cu 0-uri la final pana s-au facut 23 de cifre)

In final, x: 01000001000111000000000000000000 (reprezentarea lui 9,75 intern, pe 32 de biti)

Daca ne cere sa il scriem si in hexa?

x: 0100 0001 0001 1100 0000 0000 0000

Stim ca 4 cifre din binar inseamna o cifra din hexa.

$\Rightarrow x:411C0000$  (HEXA)



**Exemplul 3** : Transformati numarul -0,125 in format particular: n=8, k=3.

Calculam valorile:

- p=n-k=5
- lungimea mantisei =  $|\rho|=p-1=4$
- $E_{min} = -2^{k-1} + 2 = -2$
- $E_{max} = BIAS = 2^{k-1} - 1 = 3$

x=-0,125

$x \notin \{\pm 0, \pm \infty, NaN\}$

$0,125 * 2 = 0,250$

$0,250 * 2 = 0,5$

$0,50 * 2 = 1,0$

am ajuns la 0, ne oprim

$\Rightarrow -0,125 = (\overline{0,001})_2$

Ce facem mai departe? Noi vrem sa avem un 1 inainte de virgula. Deci mutam virgula cu 3 pozitii mai in spate.

$0,001 = 0,001 * 2^{-3}$

Identificam acum variabilele:

- Cine e E? E=-3 (puterea la care e 2 cand numarul este normalizat)
- Cine e mantisa? 0 (ce e dupa virgula)
- Ce valoare are s, bitul de semn? Semnul lui x e negativ, deci s=1
- Cine este caracteristica, c?  $c=E+BIAS=-3+3=0$

Facem cele 2 verificari:

- TESTUL DE OVERFLOW / UNDERFLOW  
 $E \in [E_{min}, E_{max}] \Leftrightarrow E \in [-2, 3]$   
 In cazul nostru, E=-3 deci nu se verifica  $\Rightarrow$  avem un format denormalizat.  
 Crestem E la  $E_{min} \Rightarrow x = \overline{-0,1} * 2^{-2}$
- TEST ROTUNJIRE  
 lungimea mantisei  $\leq p-1$   
 In cazul nostru, lungimea mantisei este  $|\rho| = 1$  si  $1 \leq 4$

Vrem acum sa scriem reprezentarea interna a lui x in formatul particular.

s: 1 ( $x < 0$ )

c: 000 (are 3 cifre pentru ca k=3, si toate sunt 0 pentru ca suntem in format denormalizat)

$\rho$ : 1000 (am completat cu 3 0-uri pentru ca lungimea mantisei trebuie sa fie 4)

In final, x: 10001000 (reprezentarea lui -0,125 intern, pe 8 de biti)  
 Daca ne cere sa il scriem si in hexa?  
 x: 1000 1000  
 Stim ca 4 cifre din binar inseamna o cifra din hexa.  
 =>x:88 (HEXA)

**Exemplul 4:** [RESTANTA SEPT 2020] Interpretati ca numar in baza 10 reprezentarea interna hexa in format single. 0xC1C80000.

0x din fata inseamna ca este reprezentarea in hexa. =>  $x : C1C80000(HEXA)$   
 $C = \overline{1100}$   
 $1 = \overline{0001}$   
 $C = \overline{1100}$   
 $8 = \overline{1000}$   
 $0 = \overline{0000}$

Reprezentarea in binar a lui x: 1100 0001 1100 1000 0000 0000 0000 0000.

Identificam cine sunt s, c si  $\rho$ .  
 s: 1 (primul bit din reprezentarea in binar)  
 c: 10000011 (urmatoarele 8 cifre pentru ca stim ca lungimea lui c este k si k=8 in format single)  
 $\rho$ : 100 1000 0000 0000 0000 (restul cifrelor din reprezentarea in binar)

Observam ca  $c \notin \{0...0, 1...1\}$  care corespund lui  $\pm 0$ , respectiv  $\pm \infty$ . => suntem in format normalizat.

$$x = (-1)^s * 2^E * \overline{1, \rho}$$

Cine este E? Stim ca  $c=E+BIAS$ , iar BIAS in single este 127.

=> Transform E din baza 2 in baza 10.

$$(10000011)_2^{-1} = 1 * 2^0 + 1 * 2^1 + 1 * 2^7 = 1 + 2 + 128 = 131$$

$$\Rightarrow E = c - BIAS = 131 - 127 = 4$$

Mai departe transform si  $\rho$  din baza 2 in baza 10.

$$(0, 1001)_2^{-1} = 1/2 + 1/16 = 9/16 = 0,5625$$

Acum avem toate datele, putem sa inlocuim in x.

$$x = (-1)^1 * 2^4 * \overline{1, 1001} = -16 * 1,5625 = -25$$

### Exemplul 5 ADUNAREA IN VIRGULA MOBILA

[RESTANTA MAI 2020] Calculati  $82,375 + (-1,75)$  folosind algoritmul de adunare in virgula mobila pentru formatul single (se va lucra cu reprezentarile matematice in baza 2 in notatie stiintifica, iar in final sa va converti rezultatul in baza 10).

$$x=82,375$$

$$y=-1,75$$

Il transformam pe x in baza 2:

$$82|0$$

$$41|1$$

$$20|0$$

$$10|0$$

$$5|1$$

$$2|0$$

$$1|1$$

0, ne oprim

$$0,375 * 2 = 0,750$$

$$0,75 * 2 = 1,5$$

$$0,5 * 2 = 1,0$$

0, ne oprim

$$\Rightarrow (82,375)_2 = \overline{1010010,011}$$

Il transform si pe y in baza 2:

$$0,75 * 2 = 1,5$$

$$0,5 * 2 = 1,0$$

0, ne oprim

$$\Rightarrow (-1,75)_2 = -\overline{1,11}$$

Eu le vreau pe ambele in notatie stiintifica:

$$\Rightarrow x = \overline{1010010,011} = \overline{1,010010011} * 2^6$$

Facem cele 2 verificari:

- TESTUL DE OVERFLOW / UNDERFLOW

$$E \in [E_{min}, E_{max}] \Leftrightarrow E \in [-126, 127]$$

In cazul nostru, E=6,  $-126 \leq 6 \leq 127$  deci se verifica. ✓

- TESTUL DE ROTUNJIRE

lungimea mantisei  $\leq p - 1$

In cazul nostru, lungimea mantisei este  $|\rho| = 9$  si  $9 \leq 23$  ✓

$$\Rightarrow y = \overline{1,11} = \overline{1,11} * 2^0$$

Facem cele 2 verificari:

- TESTUL DE OVERFLOW / UNDERFLOW

$$E \in [E_{min}, E_{max}] \Leftrightarrow E \in [-126, 127]$$

In cazul nostru,  $E=0, -126 \leq 0 \leq 127$  deci se verifica. ✓

- TESTUL DE ROTUNJIRE

lungimea mantisei  $\leq p - 1$

In cazul nostru, lungimea mantisei este  $|\rho| = 2$  si  $2 \leq 23$  ✓

Vrem acum sa facem adunarea dintre x si y. Ca sa putem sa facem asta este nevoie ca x si y sa aiba acelasi E.

$$\begin{aligned} 0 < 6 &\Rightarrow \text{trebuie sa aduc exponentul lui } y \text{ la nivelul lui } x \\ \Rightarrow y &\text{ devine: } 0,00000111 * 2^6 \end{aligned}$$

Acum putem face adunarea cu numarul negativ  $\Leftrightarrow$  scaderea:

$$\begin{array}{r} 1 ,0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 \\ 0 ,0 0 0 0 0 1 1 1 1 1 0 \\ \hline 1 ,0 1 0 0 0 1 0 0 0 0 1 0 0 1 0 1 \end{array}$$

$$\Rightarrow x + y = \overline{1,010000101} * 2^6$$

Facem din nou cele 2 verificari:

- TESTUL DE OVERFLOW / UNDERFLOW

$$E \in [E_{min}, E_{max}] \Leftrightarrow E \in [-126, 127]$$

In cazul nostru,  $E=6, -126 \leq 6 \leq 127$  deci se verifica. ✓

- TESTUL DE ROTUNJIRE

lungimea mantisei  $\leq p - 1$

In cazul nostru, lungimea mantisei este  $|\rho| = 9$  si  $9 \leq 23$  ✓

Deci rezultatul este:  $\overline{1,010000101} * 2^6 = \frac{1010000101}{2^9} * 2^6 = \frac{645}{8} = 80,625$

## Exemplul 6 INMULTIREA IN VIRGULA MOBILA

Inmultiti in format single x=7,75 si y=-0,5.

$$x_2 = \overline{111,11} = \overline{1,1111 * 2^2}$$

$$y = -\overline{0,1} = -\overline{1} * 2^{-1}$$

Facem verificari:

- TESTUL DE OVERFLOW / UNDERFLOW

$$E \in [E_{min}, E_{max}] \Leftrightarrow E \in [-126, 127]$$

In cazul nostru, E=2 si E=-1,  $-126 \leq 2 \leq 127$ ,  $-126 \leq -1 \leq 127$  deci se verifica. ✓

- TESTUL DE ROTUNJIRE

$$\text{lungimea mantisei} \leq p - 1$$

In cazul nostru, lungimea mantisei este  $|\rho| = 4$  si  $4 \leq 23$ ,  $|\rho| = 0$  si  $0 \leq 23$  ✓

Observam ca nu mai trebuie sa aducem exponentii la aceeasi valoare ca la adunare.

Acum putem sa afcem inmultirea:

$$\begin{array}{r} 1 ,1 & 1 & 1 & 1 \\ 1 ,0 & 0 & 0 & 0 \\ \hline 1 ,1 & 1 & 1 & 1 \end{array} \Rightarrow x*y = -\overline{1,1111} *$$

$$2^{2-1} = -\overline{1,1111} * 2^1$$

Facem din nou cele 2 verificari:

- TESTUL DE OVERFLOW / UNDERFLOW

$$E \in [E_{min}, E_{max}] \Leftrightarrow E \in [-126, 127]$$

In cazul nostru, E=1,  $-126 \leq 1 \leq 127$ , deci se verifica. ✓

- TESTUL DE ROTUNJIRE

$$\text{lungimea mantisei} \leq p - 1$$

In cazul nostru, lungimea mantisei este  $|\rho| = 4$  si  $4 \leq 23$  ✓

Deci rezultatul este:

$$-\overline{1,1111} * 2^1 = -\frac{\overline{1111}}{2^4} * 2^1 = -\frac{31}{8} = -3,875.$$

### **3 MIPS - Tablouri unidimensionale (de elemente întregi)**

**APPLE LANSEAZA LAPTOPURI CU  
PROIECȚII FACUTE DE EI PE  
ARHITECTURA ARM**



#### **3.1 Declarare**

Se declară asemănător cu datele de tip .word din tutoriatul trecut, atât că acum se scrie și lista de valori inițiale după, separare prin virgulă.

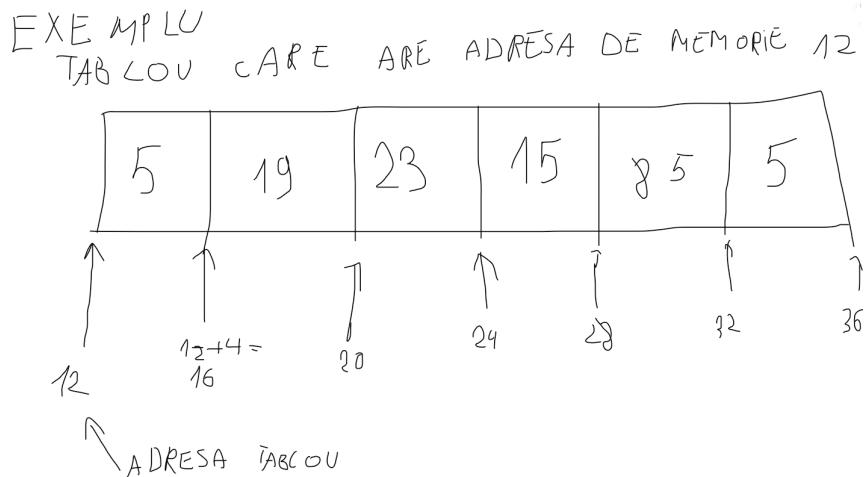
Exemplu declarare:

v:.word 5, 19, 23, 15, 85

n:.word 5

### 3.2 Indexare și adrese de memorie

Tablourile noastre declarate sunt practic o zonă de memorie în calculator unde se află valorile noastre. Noi putem afla un număr care ne spune exact unde se află în memorie începutul tabloului nostru (numit adresa de memorie). Dacă citim 4 bytes (adică un word) care se află la acea adresă de memorie (numărul nostru), atunci vom obține exact numărul de pe prima poziție din tablou. Dacă citim ce se află la poziția numarul\_nostru + 4, vom obține exact numărul de pe a doua poziție din tablou... and so on and so forth.



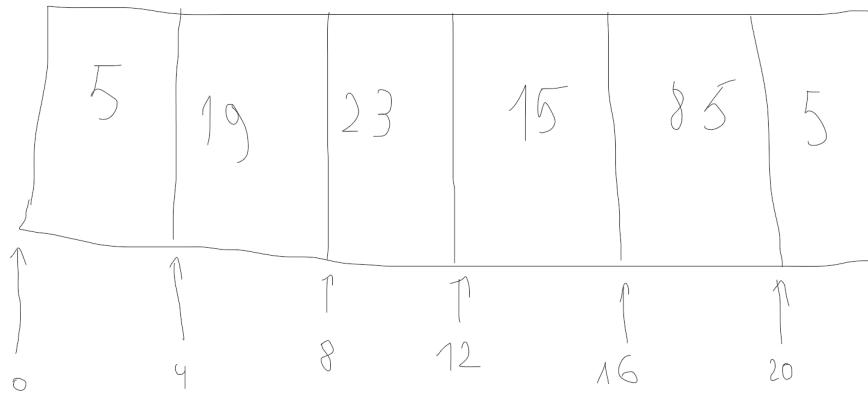
Exemplu

Să se încarce în registrul \$t1 valoarea de pe a 3-a poziție din tabloul v. la \$t0, v # Luăm adresa de memorie a lui v și o punem în registrul t0  
`addi $t0, $t0, 8` # adunăm 4 de două ori la adresa respectivă (primul 4 pentru # a ajunge la poziția a două, iar a doilea 4 pentru a ajunge la poziția a 3-a)  
`lw $t1, 0($t0)` # aici ia valoarea care se află la adresa de memorie \$t0 și o copiază # în registrul \$t1. (nu îi putem da direct `lw $t1, $t0`, pentru că instrucțiunea # lw vrea să primească o adresă de memorie pentru argumentul 2, nu un #registru.)

Acum, presupunând că avem vectorul v declarat în prima parte a materialului, atunci registrul \$t1 va avea valoarea 23.

Aceasta este una din metodele de a lucra cu tablouri, este cea mai "raw" metodă în care se lucrează exact cu adrese de memorie. Ce e de reținut e cum se ia valoarea numerică de la o adresă de memorie pe care o avem ținută într-un registru și cum se găsește adresa de memorie a unui element într-un tablou.

În continuare, o să facem același lucru, doar că printr-o metodă mai simplă, iar de acum, vom folosi această metodă pentru a lucra cu tablouri. Ne putem gândi că indexăm un array ca în alte limbi de programare, iar în cazul acesta, putem ignora adresa de început a array-ului și doar să luăm elementele de la un anumit index. Diferența dintre ce vom face acum și indexarea din celelalte limbi de programare constă în faptul că nu vom putea direct să spunem poziția de unde vrem elementul, ci va trebui să dăm poziția înmulțită cu 4 (deoarce un word ocupa 4 bytes). Așadar, ne putem gândi că primul element are indexul 0, al doilea are indexul 4, al treilea are indexul 8 și tot așa.



Folosind acest approach, putem accesa elementele de la o poziție dată în două moduri.

1. Dacă avem un indice care este o constantă.

Să zicem că vrem să copiem valoarea de la a 3-a poziție în registrul \$t1, atunci vom face:

```
la $t0, v # Luăm adresa de memorie a lui v și o punem în registrul t0
lw $t1, 8($t0) # Luăm valoarea de pe a 3-a poziție și o copiem în registrul
# $t1. Observăm că acest approach este foarte asemănător cu primul approach
# prezentat, singura diferență fiind că aici nu incrementăm adresa de
# memorie, ci îi spunem direct cu cât să incrementeze adresa de memorie
# atunci când încarcăm valoarea.
```

2. Dacă avem un indice care este ținut într-un registru.

Aceasta metodă o vom folosi cel mai frecvent (cel puțin acum la început).

Considerăm tot exemplul de mai sus, în care vrem să copiem valoarea de pe a 3-a poziție în registrul \$t1.

li \$t0, 8 # punem în \$t0 indexul pentru a 3-a poziție.

lw \$t1, v(\$t0) # Luăm valoarea de pe a 3-a poziție și o copiem în registrul

```
# $t1. Observăm că de această dată nu am mai luat deloc adresa de memorie a
# tabloului, ci pur și simplu atunci când am copiat valoarea în registrul $t1,
# i-am spus de unde să ia valoarea de la indexul nostru, adică din tabloul v.
```

Acum putem vedea asemănarea dintre această metodă și alte limbaje de programare. În C ce am facut acum s-ar traduce în: `int t1 = v[t0];` (o diferență de notat ar fi ca în C indexul nu este multiplu de 4), care seamănă cât de cât cu această ultimă scriere din MIPS.

Observație:

Toate cele 3 metode prezentate sunt echivalente. Adică ele îi spun calculatorul să ia o valoare de la o adresă de memorie. Singurul lucru care diferă este scrierea. Dacă facem un calcul, ne va ieși că în toate cele 3 metode, argumentul 2 al lui `lw` are valoarea egală cu adresa de memorie a celui de-al treilea element. În primul caz avem  $0 + \text{valoarea\_din\_t0}$ , unde în `$0` avem valoarea adresei de memorie a lui  $v + 8$ . Deci această sumă dă: adresa de memorie a lui  $v + 8$ , adică fix adresa de memorie a celui de-al treilea element. În al doilea caz avem  $8 + \text{valoarea\_din\_t0}$ , unde în `$0` avem valoarea adresei de memorie a lui  $v$ . Deci această sumă dă: adresa de memorie a lui  $v + 8$ . În ultimul approach avem adresa de memorie a lui  $v + \text{valoarea\_din\_t0}$ , unde în `$0` avem indexul pentru poziția a 3-a, adică 8. Deci suma este: adresa de memorie a lui  $v + 8$ . Deci toate cele 3 sume sunt egale.

Observație:

La ultimele 2 approach-uri putem avea și indici negativi (evidenț, care în modul sunt divizibili cu 4). Dacă de exemplu avem într-un registru `$t0` adresa de memorie a lui  $n$  (declarat mai sus). Atunci  $-4(\$t0)$  ar fi adresa de memorie a ultimului element din tabloul  $v$ , adică 85. (altă observație care se face este că toate lucrurile din memorie se află unele după altele.)

### 3.3 Exerciții

Problema 1: Se dă un vector în memorie, Să se afișeze pe ecran elementele lui.

```

1  .data
2    v:.word 5, 19, 25, 13, 8 # vectorul nostru din memorie
3    n:.word 5                 # lungimea vectorului
4    sp:.asciiz " "
5  .text
6  main:
7    lw $t0, n                # încarcă Lungimea vectorului în registrul $t0
8    li $t1, 0                 # counterul nostru pentru loop
9    li $t2, 0                 # indicele de unde vom accesa vectorul
10   |                         | # nu îl folosim pe $t1 pentru asta
11   |                         | # deoarece avem nevoie ca indicele nostru pentru vector
12   |                         | # să fie multiplu de 4
13
14  loop:
15    beq $t0, $t1, exit
16
17    lw $a0, v($t2)          # copiază elementul de la poziția $t2 din vector
18    |                         | # în registrul $a0 pentru a o afisa pe ecran
19    li $v0, 1
20    syscall
21
22    la $a0, sp
23    li $v0, 4
24    syscall
25
26    addi $t1, $t1, 1        # incrementează counterul
27    addi $t2, $t2, 4        # incrementează indicele nostru pentru vector.
28    |                         | # este incrementat cu 4 bytes deoarece un word înseamnă
29    |                         | # 4 bytes
30
31    j loop
32
33  exit:
34    li $v0, 10
35    syscall

```

Observație: Ce se întâmplă dacă schimbăm valoarea lui  $n$  din 5 în 6?  
Încercați.

Problema 2: Se dă  $n \in N$  stocat în memorie, Să se citească un vector de  $n$  elemente numere întregi.

```

1  .data
2  |   v:.space 28          # 7 elemente, adică 28 de bytes
3  |   n:.word 7
4  .text
5  main:
6
7      lw $t0, n
8      li $t1, 0
9      li $t2, 0
10
11     loop:
12         beq $t0, $t1, exit
13
14         li $v0, 5
15         syscall
16         sw $v0, v($t2)      # salvează valoarea citită în vectorul v
17         | | | | |           # La poziția $t2
18
19         addi $t1, $t1, 1
20         addi $t2, $t2, 4
21
22         j loop
23
24     exit:
25
26     li $v0, 10
27     syscall

```

Bonus: Să se și afișeze vectorul citit.

Problema 3: Se citesc  $n \in N$  și un vector de  $n$  numere întregi. Să se afișeze pe ecran elementele pare prin două parcurgeri (una pentru citire și una pentru afișare).

```

1  .data
2      v:.space 400          # vector de 100 de elemente (400 = 100 * 4)
3      sp:.asciiz " "
4  .text
5  main:
6
7      li $v0, 5
8      syscall
9      move $t0, $v0
10
11     li $t1, 0
12     li $t2, 0
13
14     read:                 # citește vectorul (ca la problema anterioară)
15
16         beq $t0, $t1, solve
17
18         li $v0, 5
19         syscall
20         sw $v0, v($t2)|
21
22         addi $t1, $t1, 1
23         addi $t2, $t2, 4
24
25         j read
26
27     solve:                 # afisează pe ecran
28
29     li $t1, 0              # t1 va fi counterul nostru care va merge din 4 în 4 poziții
30     li $t2, 0              # indexul curent din vector (merge tot din 4 în 4 poziții
31             |               |               |               |               |
32             |               |               |               |               |               #
33             |               |               |               |               |               # deci va fi incrementat cu 16, adică 4 * 4)
34
35     loop:
36
37         beq $t0, $t1, exit  # dacă am parcurs vectorul ne oprim
38
39         lw $t3, v($t2)      # Luăm valoarea din array la poziția noastră și o ținem în $t3
40
41         rem $t4, $t3, 2      # calculăm restul împărțirii la 2 în $t4 pentru a verifica
42             |               |               |               |
43             |               |               |               |               # dacă numărul este par
44
45         beq $t4, 0, par      # dacă e par, îl vom afișa
46         j cont                # dacă nu, continuăm Loop-ul în mod normal
47         par:
48
49             move $a0, $t3      # copiază în a $a0 valoarea pară pentru a o afișa
50             li $v0, 1
51             syscall            # afisează valoarea
52
53             la $a0, sp
54             li $v0, 4
55             syscall            # afisează spațiu între rezultate
56
57         cont:
58
59             addi $t1, $t1, 1
60             addi $t2, $t2, 4
61
62             j loop
63
64     exit:
65
66     li $v0, 10
67     syscall

```

Observație: Pentru a simula un if putem face în felul următor:

```
b__ ceva_condiție, label_if
j continuare # dacă condiția nu este adevărată, programul continuă în mod
# normal.
label_if:
# aici vom avea codul din if
continuare:
# aici va continua programul indiferent dacă s-a executat sau nu codul din if.
```

Problema 4: Se citește un număr natural  $n$ ,  $n \leq 200$ , urmat de  $n$  numere întregi. Să se parcurgă vectorul (adică cele  $n$  numere întregi citite) din 4 în 4 poziții, iar atunci când este întâlnită una din valorile 1 sau 2, să se afișeze valorile de pe următoarele 3 poziții din vector. Când este întâlnită valoarea 99, programul trebuie să se opreasă. Se garantează că inputul este corect. (adică nu te va pune sa afișezi pe ecran valori care sunt în afara vectorului)

Exemplu:

Input:

```
12
1
9
10
3
2
3
11
0
99
30
40
50
```

Output:

```
9 10 3
3 11 0
```

Explicație: Se citește prima oară numărul de elemente din vector, adică 12. După care se citesc 12 valori. Prima valoare din vector este 1, deci trebuie să afișăm următoarele 3 valori. Trecem la a 4-a poziție din vector, unde valoarea este 2, deci iar trebuie să afișăm următoarele 3 valori. Apoi vom ajunge la valoarea 99, deci programul trebuie să se opreasă.

```

1 .data
2   v:.space 800           # vector de 200 de elemente (800 = 200 * 4)
3   sp:.asciz " "
4   nl:.asciz "\n"         # pentru a afișa rânduri noi intre răspunsuri
5 .text
6 main:
7
8   li $v0, 5
9   syscall
10  move $t0, $v0
11
12  li $t1, 0
13  li $t2, 0
14
15 read:                 # citește vectorul (ca la problema anterioară)
16
17  beq $t0, $t1, solve
18
19  li $v0, 5
20  syscall
21  sw $v0, v($t2)
22
23  addi $t1, $t1, 1
24  addi $t2, $t2, 4
25
26  j read
27
28 solve:                # rezolvă problema
29
30  li $t1, 0             # t1 va fi counterul nostru care va merge din 4 în 4 pozitii
31  li $t2, 0             # indexul curent din vector (merge tot din 4 în 4 poziții
32                                # deci va fi incrementat cu 16, adică 4 * 4)
33
34 loop:
35
36  bge $t1, $t0, exit   # dacă am ieșit din vector, ne oprim (teoretic nu ar fi
37                                # nevoie de această linie dacă avem garantat
38                                # că există mereu un 99 care să ne opreasă)
39
40  lw $t3, v($t2)        # luăm valoarea din array la poziția noastră și o ținem în $t3
41
42  beq $t3, 99, exit     # dacă am găsit 99, programul se oprește
43
44  beq $t3, 1, afis_case
45  beq $t3, 2, afis_case
46
47  j cont_loop           # dacă am ajuns aici, nicio condiție nu este îndeplinită
48                                # deci continuă loop-ul normal
49
50 afis_case:              # dacă am ajuns aici, e clar că $t3 este egal fie cu 1, fie cu 2
51
52  move $t4, $t2          # Luăm poziția la care suntem în vector și o salvăm în $t4
53
54  addi $t4, 4             # incrementăm poziția pentru a afișa elementul următor
55  lw $a0, v($t4)          # încărcăm în $a0 valoarea elementului următor pentru a o afișa
56  li $v0, 1
57  syscall                # afișăm valoarea pe ecran
58
59  la $a0, sp              # afișăm spațiu pe ecran
60  li $v0, 4
61  syscall
62
63  addi $t4, 4             # facem la fel și pentru următorul număr care trebuie afișat
64  lw $a0, v($t4)
65  li $v0, 1
66  syscall
67
68  la $a0, sp
69  li $v0, 4
70  syscall
71
72  addi $t4, 4             # La fel și pentru al treilea
73  lw $a0, v($t4)
74  li $v0, 1
75  syscall
76
77  la $a0, nl              # punem un rând nou
78  li $v0, 4
79  syscall
80
81 cont_loop:               20
82
83  addi $t1, $t1, 4        # continuă loop-ul, adică incrementează poziția curentă cu 4
84  addi $t2, $t2, 16        # incrementează indexul cu 16. (4 * 4)
85
86  j loop
87
88 exit:
89  li $v0, 10
90  syscall
91
92
93
```

**Problema 5:** Se citește un număr natural  $n$ ,  $n \leq 200$ , urmat de  $n$  numere întregi. Să se parcurgă vectorul (adică cele  $n$  numere întregi citite) din 4 în 4 poziții, iar atunci când este întâlnită una din valorile 1 sau 2, se consideră următorile valori astfel:  $a =$  numărul de pe poziția următoare în vector,  $b =$  numărul care urmează după  $a$ ,  $c =$  numărul care urmează după  $b$ . Dacă valoarea întâlnită în vector este 1, atunci să se realizeze următoarea operație din C:  $v[c] = v[a] + v[b]$ ; Dacă valoarea întâlnită în vector este 2, atunci să se realizeze urmatoarea operație din C  $v[c] = v[a] * v[b]$ ; Când se întâlnește valoarea 99, programul trebuie să se opreasă. Înainte de a parcurge vectorul, să se pună în vector la poziția 2 valoarea 12, iar la poziția 3, valoarea 2. Se cere să se afișeze valoarea de la poziția 0 din vector după ce programul își termină executarea. Este garantat că datele de intrare sunt valide.

Exemplu (în care nu am schimbat valorile de pe pozitiile 2 si 3 din vector pentru a fi mai usor de ilustrat programul, în rezolvare nu trebuie omis acest pas):

Input:

```
12
1
9
10
3
2
3
11
0
99
30
40
50
```

Output:

```
3500
```

**Explicație:** Ca și data trecută prima oară este întâlnit 1, deci  $a = 9$ ,  $b = 10$ ,  $c = 3$ . Deoarece avem valoarea 1, facem adunarea numerelor de la pozițiile 9, respectiv 10, și o salvăm la poziția 3. Vom avea:  $v[3] = v[10] + v[9]$ ; deci la a 3 a poziția 3 nu vom mai avea valoarea 3, ci valoarea  $v[10] + v[9]$ , adică  $30 + 40$ , adică 70. Acum întâlnim valoarea 2, deci  $a = 3$ ,  $b = 11$ ,  $c = 0$ . Deoarece avem valoarea 2, facem înmulțirea numerelor de la pozițiile 3, respectiv 11 și o salvăm la poziția 0. Vom avea  $v[0] = v[3] * v[11]$ ; deci la prima poziție nu vom mai avea valoarea 1, ci valoarea  $v[3] * v[11]$ , adică  $70 * 50 = 3500$ . Întâlnim 99, deci ne oprim și afișăm valoarea de pe prima poziție, adică 3500.

```

1      .data
2          v1: space 800
3          sp: ascii " "
4          nl:ascii "\n"
5      .text
6      main:
7          li $v0, 5
8          syscall
9          move $t0, $v0
10         li $t1, 0
11         li $t2, 0
12
13         read:
14             beq $t0, $t1, solve
15
16             li $v0, 5
17             syscall
18             sw $v0, v($t2)
19
20             addi $t1, $t1, 1
21             addi $t2, $t2, 4
22
23             j read
24
25         solve:
26
27             la $t1, v           # încarcă în $t1 adresa de memorie a vectorului
28             li $t2, 12          # pune într-un registru valoarea 12
29             sw $t2, 4($t1)       # pune pe a doua poziție din vector valoarea 12
30
31             li $t2, 2           # pune pe a treia poziție din vector valoarea 2
32             sw $t2, 8($t1)       # pune pe a patra poziție din vector valoarea 2
33
34             li $t1, 0           # pune pe a cincea poziție din vector valoarea 0
35
36             li $t2, 0           # pune pe a sasea poziție din vector valoarea 0
37
38             loop:
39
40                 bge $t1, $t0, exit
41
42                 lw $t3, v($t2)
43
44                 beq $t3, 99, exit
45
46                 beq $t3, 1, adunare    # acum avem 2 cazuri, unul pentru adunare, unul pentru înmulțire
47                 j cont1            # dacă nu a apărut 1, poate a apărut 2, deci mergem mai jos să verificăm
48
49             adunare:
50
51                 move $t4, $t2        # obține cele 3 numere a, b, c ca în programul anterior
52                 addi $t4, $t4, 4
53                 lu $t5, v($t4)
54
55                 addi $t4, $t4, 4
56                 lu $t6, v($t4)
57
58                 addi $t4, $t4, 4
59                 lu $t7, v($t4)
60
61                 add $t5, $t5, $t5    # înmulțește pozitia lui a cu 4 (pentru că așa "indexăm" noi)
62                 add $t5, $t5, $t5    # două adunări replate înseamnă o înmulțire cu 4
63
64                 lw $t5, v($t5)      # obține valoarea din vector de la poziția a
65
66                 add $t6, $t6, $t6    # la fel cum am făcut pentru a, facem și pentru b
67                 add $t6, $t6, $t6
68
69                 lu $t6, v($t6)
70
71                 add $t5, $t5, $t6    # facem adunarea
72
73                 add $t7, $t7, $t7    # la fel, înmulțim pozitia cu 4 pentru a putea
74                 add $t7, $t7, $t7    # accesa în vectorul nostru
75
76                 sw $t5, v($t7)      # salvează rezultatul la poziția c
77
78                 j cont2            # ne ducem la sfârșitul loop-ului pentru că
79                 # stîm că nu vom intra pe cazul în care
80                 # valoarea este 2
81
82             cont1:
83
84                 beg $t3, 2, inmultire # dacă valoarea citită e 2, este cazul pentru înmulțire
85
86                 j cont2            # dacă valoarea nu e nici 1, nici 2, atunci continuă loop-ul
87
88             inmultire:
89                 # codul pentru v(c) = v(a) * v(b), asemănător cu cel de adunare
90                 # majoritatea codului se repetă, probabil s-ar putea face mai elegant
91                 # deși atunci succesiunea de branch-uri ar deveni confuză
92                 # deci, în scop didactic vom avea "copy paste" aici.|
93
94                 move $t4, $t2
95                 addi $t4, $t4, 4
96                 lu $t5, v($t4)
97
98                 addi $t4, $t4, 4
99                 lu $t6, v($t4)
100
101                 addi $t4, $t4, 4
102                 lu $t7, v($t4)
103
104                 add $t5, $t5, $t5
105                 add $t5, $t5, $t5
106
107                 lw $t5, v($t5)
108
109                 add $t6, $t6, $t6
110                 add $t6, $t6, $t6
111
112                 lw $t6, v($t6)
113
114                 mul $t5, $t5, $t6    # înmulțește cele două numere
115
116                 add $t7, $t7, $t7
117                 add $t7, $t7, $t7
118
119                 sw $t5, v($t7)
120
121             cont2:
122
123                 addi $t1, $t1, 4
124                 addi $t2, $t2, 16
125
126                 j loop
127
128             ~ exit:
129
130                 lw $a0, v           # ia valoarea de pe prima poziție și o afisează
131                 li $v0, 1
132                 syscall
133
134                 li $v0, 10
135                 syscall

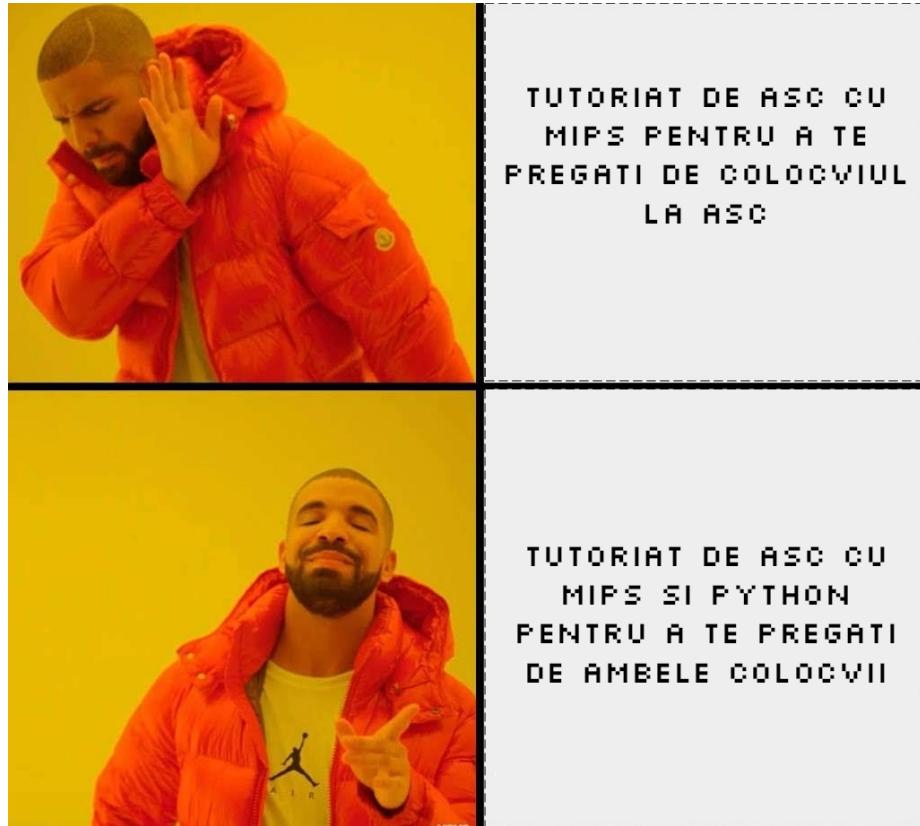
```

Fun fact: Aceasta este prima parte din problema care a fost dată anul trecut la Advent of Code în cea de a doua zi. Acolo vectorul este dat ca numere separate prin virgulă. Pentru a ne fi ușor să interpretăm inputul în MIPS am schimbat metoda de input (n citit, urmat de alte n numere). Pentru a converti inputul acestei probleme de AOC la input ușor de citit în MIPS, putem folosi următorul script în Python:

```

3  infile = open("raw_input.txt")           # deschide fisierul din care citim
4  outfile = open("processed_input.txt", "a") # deschide fisierul in care scriem
5
6  numbers = [int(x) for x in infile.readline().split(",")] # parseaza vectorul de numere
7
8  outfile.write(str(len(numbers)) + "\n")          # scrie numarul de elemente din vector
9
10 for number in numbers:                   # parcurge array-ul
11     outfile.write(str(number) + "\n")          # scrie numerele separate prin randuri noi
12
13 outfile.close()                         # inchide fisierul in care scriem
14 infile.close()                          # inchide fisierul din care citim
15                                         # La colocviu la PA sa nu uitati sa
16                                         # inchideti fisierele
17                                         # cei care corecteaza au si lucrui de genul
18                                         # in vedere

```



Problema 6: Se dă o matrice declarată în memorie de forma:

2	a:.word 1, 2, 3, 4
3	.word 5, 6, 7, 8
4	.word 9, 10, 11, 12
5	n:.word 3
6	m:.word 4

Să se afișeze pe ecran.

```

1   .data
2       a:.word 1, 2, 3, 4
3           .word 5, 6, 7, 8
4               .word 9, 10, 11, 12
5       n:.word 3
6       m:.word 4
7       sp:.asciiz " "
8       nl:.asciiz "\n"
9   .text
10  main:
11
12      lw $t0, n          # incarc in $t0 numărul de liniii
13      lw $t1, m          # incarc in $t1 numărul de coloane
14
15      li $t2, 0          # contor pentru iterarea liniilor
16
17      print:             # "for" pentru liniii
18
19          beq $t2, $t0, exit
20
21          li $t3, 0          # counter pentru iterarea coloanelor
22
23          print_line:        # "for" pentru coloane
24
25              beq $t3, $t1, end_line
26
27              move $t4, $t2          # calculează poziția în matrice
28
29              mul $t4, $t4, $t1          # poziția e dată de formula  $p = (L * m) + c$ 
30
31              add $t4, $t4, $t3          # unde L este linia, iar c este coloana
32
33              add $t4, $t4, $t4          # inmulțim $t4 cu 4 pentru a fi pozitie validă
34              add $t4, $t4, $t4          # în tabloul nostru
35
36              lw $a0, a($t4)          # incarc valoarea din tablou în registrul $a0
37              li $v0, 1          # pentru a o afișa pe ecran
38              syscall
39
40              la $a0, sp
41              li $v0, 4
42              syscall
43
44              addi $t3, $t3, 1
45              j print_line
46
47          end_line:
48
49              la $a0, nl
50              li $v0, 4
51              syscall
52
53              addi $t2, $t2, 1
54
55              j print
56
57      exit:
58
59      li $v0, 10
60      syscall

```

### **3.4 Mai multe exerciții**

Problema 1: Se citesc  $n \in N^*$  și un vector de  $n$  numere naturale. Să se afișeze maximul și toate pozițiile pe care apare.

Problema 2: Se citesc  $n \in N^*$  și doi vectori  $v, w \in Z$  ordonați crescător. Să se interclaseze cei doi vectori într-un vector  $z$  și să se afișeze  $z$ .

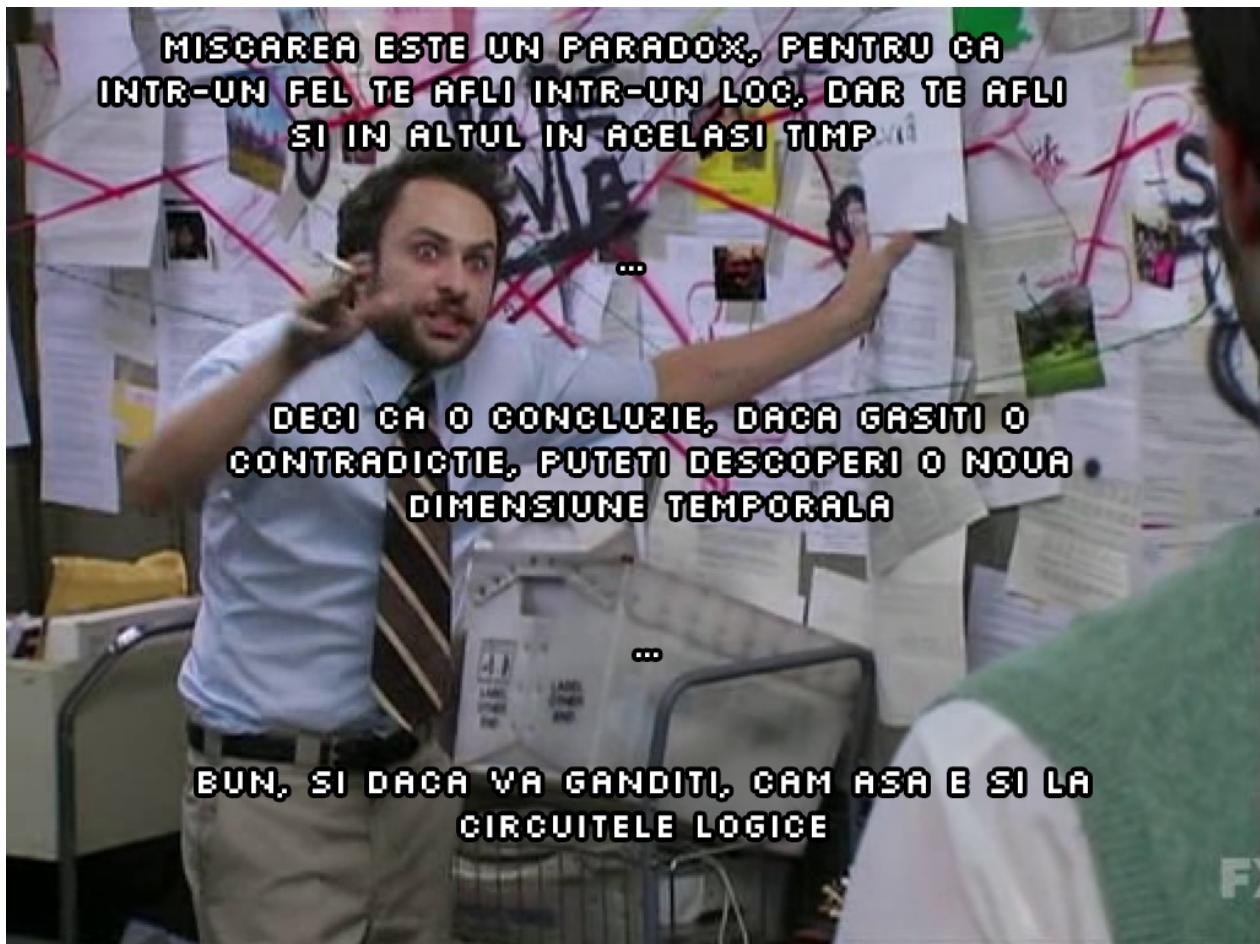
## **References**

- [1] Dumitru Daniel Drăgulici. *Curs Arhitectura Sistemelor de Calcul*.
- [2] Larisa Dumitrashe. *Tutoriat 2019*
- [3] Bogdan Macovei. *Laboratoare ASC 2019/ 2020*
- [4] Advent Of Code. *Day 2 2019*

# Tutoriat 4

Stan Bianca-Mihaela, Stăncioiu Silviu

January 12, 2021



## Contents

<b>1</b>	<b>Algebra Booleană</b>	<b>2</b>
1.1	Operatii si proprietati . . . . .	2
1.2	Functiile booleene . . . . .	4
<b>2</b>	<b>Porti logice</b>	<b>5</b>
<b>3</b>	<b>MIPS</b>	<b>12</b>
3.1	Siruri de caractere . . . . .	12
3.2	Lucrul cu numere în virgulă mobilă . . . . .	15

# 1 Algebra Booleana

## 1.1 Operatii si proprietati

Vom incepe cu algebra booleana cu care probabil suntem deja familiari de la Logica. Ca un mic twist, la ASC avem notatiile:

- $+$   $\Leftrightarrow \vee$  (SAU)
- $\cdot$   $\Leftrightarrow \wedge$  (SI)
- $\overline{\phantom{x}}$   $\Leftrightarrow \neg$  (NEGATIA)

$x$	$y$	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

$x$	$y$	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

$$\overline{0} = 1$$

$$\overline{1} = 0$$

Dupa cum stim, pe o algebra booleana  $B_2$  (aka unde avem doar multimea de valori 0,1) avem anumite axiome:

### 1. ASOCIATIVITATEA

$$(x+y)+z=x+(y+z)$$

$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$

### 2. COMUTATIVITATEA

$$x+y=y+x$$

$$x \cdot y = y \cdot x$$

### 3. ABSORBTIA

$$x + (x \cdot y) = x \cdot (1 + y) = x \text{ (fiindca } 1 + y = 1 \text{ sau "ceva" va da mereu 1)}$$

$$x \cdot (x + y) = x \text{ (daca } x \neq 0 \Rightarrow \text{rezultatul e 0, daca } x = 1 \Rightarrow \text{rezultatul e 1 indiferent de } y)$$

### 4. DISTRIBUTIVITATEA

$$x \cdot (y + z) = x \cdot y + x \cdot z$$

$$x + (y \cdot z) = (x + y) \cdot (x + z)$$

### 5. MARGINIREA

$$0+x=x$$

$$0 \cdot x = 0$$

$$1+x=1$$

$$1 \cdot x = x$$

## 6. COMPLEMENTAREA

$$x + \bar{x} = 1$$

$$x \cdot \bar{x} = 0$$

Pe langa acestea, mai definim si cateva operatii derivate:

- **IMPLICATIA**

$$x \rightarrow y = \bar{x} + y$$

- **DIFERENTA**

$$x - y = x \cdot \bar{y}$$

- **ECHIVALENTA**

$$x \leftrightarrow y = (x \rightarrow y) \cdot (y \rightarrow x) = (\bar{x} + y) \cdot (\bar{y} + x)$$

- **XOR**

$$x \oplus y = x \cdot \bar{y} + \bar{x} \cdot y$$

- **NXOR**

$$x * y = \overline{x \oplus y} \text{ (am notat cu * pentru ca NXOR nu are un semn propriu-zis din cate stiu eu)}$$

- **NAND**

$$x * y = \overline{x \cdot y} \text{ (am notat cu * pentru ca NAND nu are un semn propriu-zis din cate stiu eu)}$$

- **NOR**

$$x * y = \overline{x + y} \text{ (am notat cu * pentru ca NOR nu are un semn propriu-zis din cate stiu eu)}$$

Aici avem un tabel de adevar cu cele mai importante operatii:

x	y	$x \cdot y$	x NAND y	$x + y$	x NOR y	$x - y$	$x \rightarrow y$	$x \oplus y$	x NXOR y
0	0	0	1	0	1	0	1	0	1
0	1	0	1	1	0	0	1	1	0
1	0	0	1	1	0	1	0	1	0
1	1	1	0	1	0	0	1	0	1

Si cateva proprietati:

## 1. IDEMPOTENTA

$$x + x = x$$

$$x \cdot x = x$$

## 2. LEGEA DUBLEI NEGATII

$$\bar{\bar{x}} = x$$

## 3. LEGILE LUI MORGAN

$$\overline{x + y} = \bar{x} \cdot \bar{y}$$

$$\overline{x \cdot y} = \bar{x} + \bar{y}$$

$$\overline{\bar{x} + \bar{y}} = x \cdot y$$

$$\overline{\bar{x} \cdot \bar{y}} = x + y$$

#### 4. ABSORBTIA BOOLEANA

$$x + \bar{x} \cdot y = x + y$$

$$x \cdot (\bar{x} + y) = x \cdot y$$

#### 5. UNICITATEA COMPLEMENTULUI

$$x+y=1 \text{ si } x \cdot y = 0 \Rightarrow y = \bar{x}$$

$$x+y=0 \Rightarrow x = y = 0$$

$$x \cdot y = 1 \Rightarrow x = y = 1$$

## 1.2 Functiile booleene

O functie booleana este o functie de forma:

$$f : B_2^n \rightarrow B_2^k \Leftrightarrow f : \{0, 1\}^n \rightarrow \{0, 1\}^k$$

Pentru aceste functii putem defini FND si FNC cu care probabil cuntem familiarii de la logica.

- FND (FUNCTIA NORMALA DISJUNCTIVA) aka unde da functia 1

$$(.) + (.) + \dots + (.) = 1$$

- FNC (FUNCTIA NORMALA CONJUNCTIVA) aka unde da functia 0

$$(+) \cdot (+) \cdot \dots \cdot (+) = 0$$

**Exemplul 1 :** [RESTANTA SEPTEMBRIE 2020]

Fie  $f : B_2^3 \rightarrow B_2^2$ ,  $f(x, y, z) = (f_1(x, y, z), f_2(x, y, z))$ , unde:

- $f_1(x, y, z) = (x \oplus y)(y \oplus z)$
- $f_2(x, y, z) = 1$  d.d. secventa  $x, y, z$  este crescatoare (adica  $x \leq y \leq z$ ).

Construiti tabelul de valori al lui  $f$  si scrieti  $f_1$  si  $f_2$  in FND si FNC.

index	x	y	z	$x \oplus y$	$y \oplus z$	$f_1(x, y, z)$	$f_2(x, y, z)$
(0)	0	0	0	0	0	0	1
(1)	0	0	1	0	1	0	1
(2)	0	1	0	1	1	1	0
(3)	0	1	1	1	0	0	1
(4)	1	0	0	1	0	0	0
(5)	1	0	1	1	1	1	0
(6)	1	1	0	0	1	0	0
(7)	1	1	1	0	0	0	1

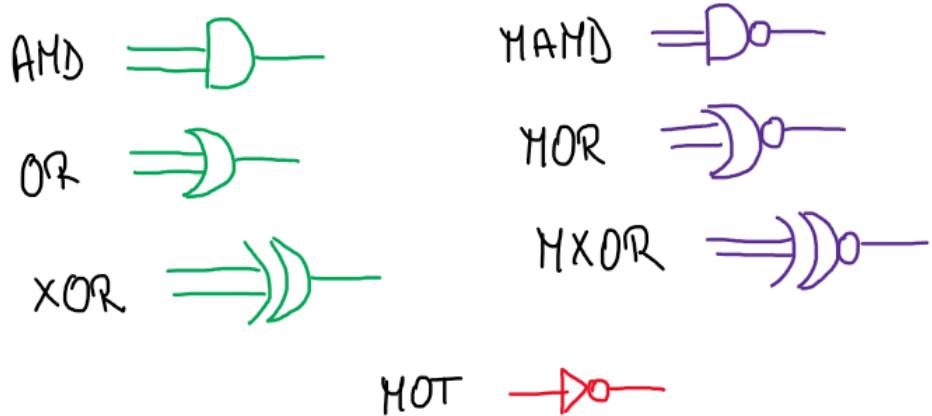
FND:

- $f_1 = \bar{x} \cdot y \cdot \bar{z} + x \cdot \bar{y} \cdot z \quad (2)+(5)$
- $f_2 = \bar{x} \cdot \bar{y} \cdot \bar{z} + \bar{x} \cdot \bar{y} \cdot z + \bar{x} \cdot y \cdot z + x \cdot y \cdot z \quad (0)+(1)+(3)+(7)$

FNC:

- $f_1 = (x + y + z) \cdot (x + y + \bar{z}) \cdot (x + \bar{y} + z) \cdot (\bar{x} + y + z) \cdot (x + y + \bar{z}) \cdot (x + y + z) \quad (0) \cdot (1) \cdot (3) \cdot (4) \cdot (6) \cdot (7)$
- $f_2 = (x + \bar{y} + z) \cdot (\bar{x} + y + z) \cdot (\bar{x} + \bar{y} + z) \cdot (\bar{x} + \bar{y} + \bar{z}) \quad (2) \cdot (4) \cdot (5) \cdot (6)$

## 2 Porti logice



### IMPORTANT!

Orice circuit se poate implementa folosind doar porti NOR, respectiv NAND. Cum? Stim ca orice circuit se poate implementa doar cu porti AND, OR si NOT => daca aratam ca putem sa implementam aceste 3 porti doar cu porti NOR, respectiv NAND, am demonstrat ca orice circuit se poate implementa doar cu porti NOR, respectiv NAND.

### Pentru NOR:

- NEGATIA:

$$\bar{x} = x \text{ NOR } x$$

- AND:

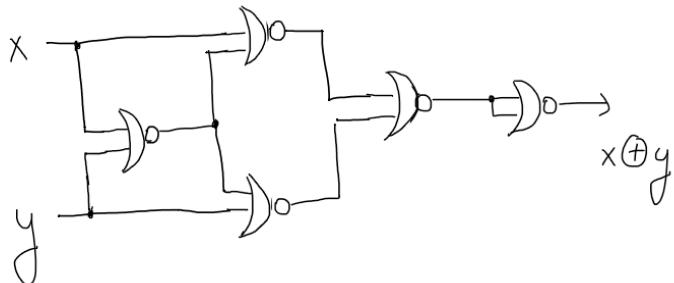
$$x \cdot y = \overline{\overline{x} \cdot \overline{y}} = \overline{x} + \overline{y} = \overline{x} \text{ NOR } \overline{y} = (x \text{ NOR } x) \text{ NOR } (y \text{ NOR } y)$$

- OR:

$$x + y = \overline{\overline{x} \cdot \overline{y}} = \overline{x} \text{ NOR } \overline{y} = (x \text{ NOR } y) \text{ NOR } (x \text{ NOR } y) \quad (1)$$

- XOR: Prin anumite "optimizari" in calcul putem obtine o formula cu doar 5 NOR-uri (vezi Exemplul 4 pentru demonstratie):

$$\begin{aligned} x \oplus y &= [(x \text{ NOR } (x \text{ NOR } y)) \text{ NOR } (y \text{ NOR } (x \text{ NOR } y))] \\ &\quad \text{NOR } [(x \text{ NOR } (x \text{ NOR } y)) \text{ NOR } (y \text{ NOR } (x \text{ NOR } y))] \end{aligned} \quad (2)$$



Pentru NAND:

- NEGATIA:

$$\bar{x} = \overline{x \cdot x} = x \text{ NAND } x$$

- AND:

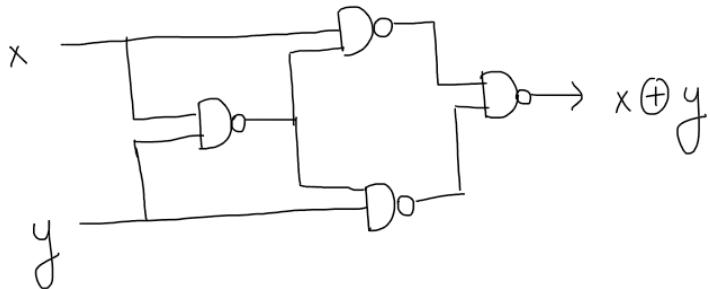
$$x \cdot y = \overline{\overline{x} \cdot \overline{y}} = \overline{x \text{ NAND } y} = (x \text{ NAND } y) \text{ NAND } (x \text{ NAND } y)$$

- OR:

$$x + y = \overline{\overline{x} + \overline{y}} = \overline{\overline{x} \cdot \overline{y}} = \overline{x \text{ NAND } y} = (x \text{ NAND } x) \text{ NAND } (y \text{ NAND } y)$$

- XOR: Prin anumite "optimizari" in calcul, putem sa ajungem la o formula cu doar 4 NAND-uri (vezi Exemplul 5 pentru demonstratie):

$$x \oplus y = ((x \text{ NAND } y) \text{ NAND } x) \text{ NAND } ((x \text{ NAND } y) \text{ NAND } y)$$

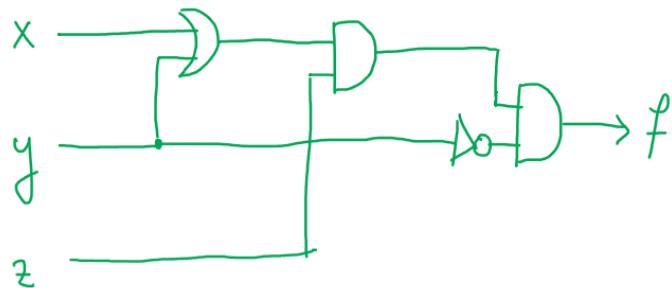


Ca sa intelegem mai bine cum se lucreaza cu aceste porti logice, vom face cateva exemple:

Exemplul 1 : Reprezentati prin porti logice functia:

$$f : \mathbb{B}_2^3 \rightarrow \mathbb{B}_2$$

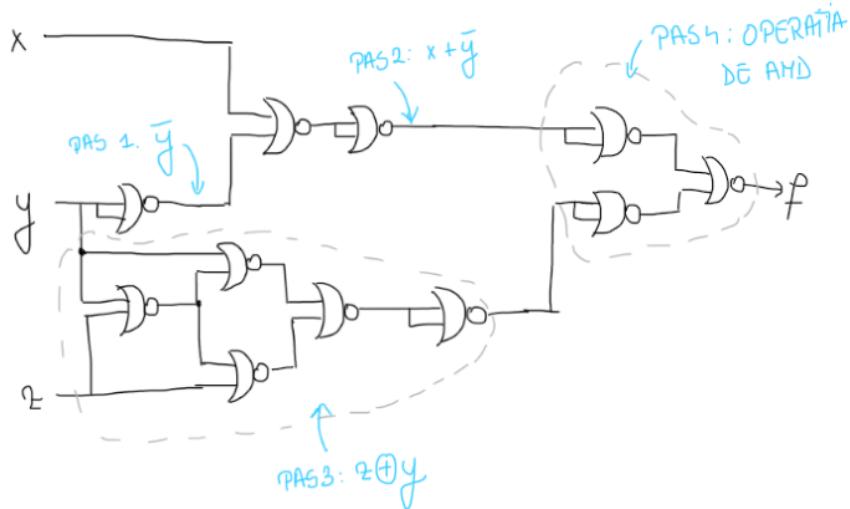
$$f(x, y, z) = (x + y) \cdot z \cdot \bar{y}$$



**Exemplul 2 :** Implementati functia  $f : B_2^3 \rightarrow B_2^1$ ,  $f(x) = (x + \bar{y}) \cdot (z \oplus y)$  doar cu porti NOR si apoi doar cu porti NAND.

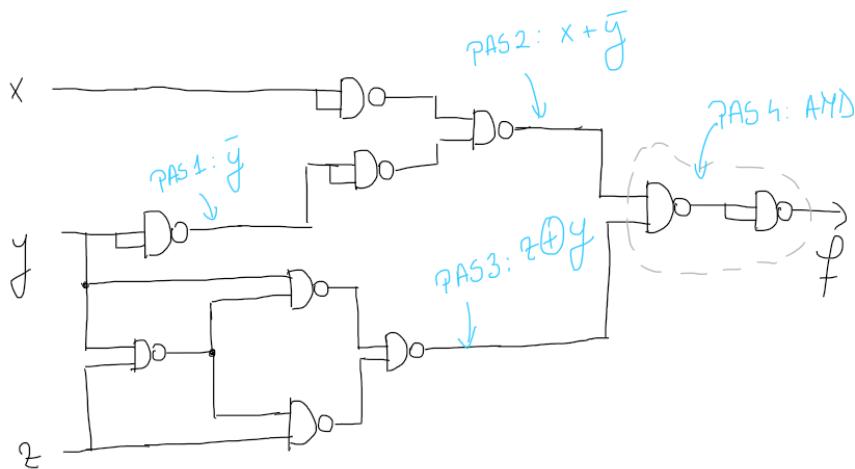
Varianta cu NOR:

$$f : B_2^3 \rightarrow B_2^1, f(x, y, z) = (x + \bar{y}) \cdot (z \oplus y)$$



Varianta cu NAND:

$$f : B_2^3 \rightarrow B_2^1, f(x, y, z) = (x + \bar{y}) \cdot (z \oplus y)$$



**Exemplul 3 [RESTANTA SEPTEMBRIE 2020]**

Implementati poarta NXOR printr-un circuit care contine doar porti NOR.

Prima metoda care ne vine in minte este sa expimam x NXOR y =  $\bar{x} \cdot y + x \cdot \bar{y}$  si sa inlocuim operatiile de NEGATIE, AND si OR asa cum stim din formulele pentru NOR.

Aceasta rezolvare este corecta, dar rezulta in:

- 3 porti NOR de la NEGATIE

- 3+3 porti NOR de la AND

- 3 porti NOR de la OR

=> in total vom avea 12 porti NOR, ceea ce nu e ideal.

Dragulici nu specifica ca numarul de porti logice sa fie minim, dar cu siguranta nu se supara daca rezolvati cu numar minim. Si o sa va ajute si pe voi sa nu va incurcati, pentru ca a desena 12 porti doar pentru un XNOR nu e usor.

Asa ca vom incerca sa ajungem la numarul minim de porti cu care se poate reprezenta NXOR.

Keep in mind, la examen daca vreti sa folositi varianta "optimizata" cu 5 porti, trebuie sa aratati cum ati ajuns acolo.

$$x \text{ NXOR } y = \overline{x \oplus y}$$

$$x \text{ NOR } y = \overline{x + y}$$

Stim ca:

$$x \oplus y = x \cdot \bar{y} + y \cdot \bar{x}$$

↓

$$\overline{x \oplus y} = \overline{x \cdot \bar{y} + y \cdot \bar{x}}$$

Stim ca  $x \cdot \bar{x} = 0$  și  $y \cdot \bar{y} = 0$

$\Rightarrow$  pot sa adaug  $x \cdot \bar{x}$  și  $y \cdot \bar{y}$

$$\begin{aligned} \overline{x \oplus y} &= \overline{x \cdot \bar{y} + y \cdot \bar{x} + x \cdot \bar{x} + y \cdot \bar{y}} = \\ &= \overline{\cancel{x \cdot \bar{y}}} + \overline{\cancel{y \cdot \bar{x}}} \end{aligned}$$

$$A = \overline{x \cdot \bar{y}}$$

$$A = \overline{\overline{A}} = \overline{\overline{x \cdot \bar{y}}} = \overline{\overline{x}} + \overline{\overline{x \cdot \bar{y}}} =$$

$$= \overline{x + (\overline{x} \cdot \bar{y})} = x \text{ NOR } (\overline{x} \cdot \bar{y}) =$$

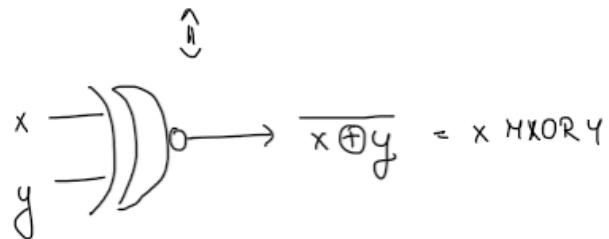
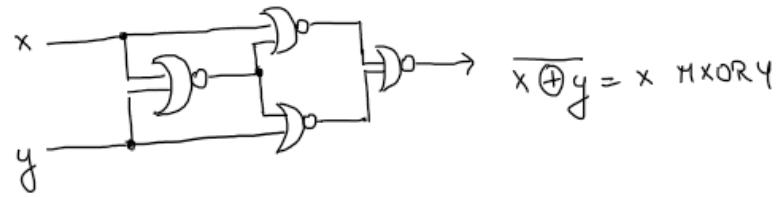
$$\Rightarrow A = x \text{ NOR } (x \text{ NOR } y)$$

$$\text{Analog } B = \overline{y \cdot \bar{x}}$$

$$\overline{x \oplus y} = \overline{[x \text{ NOR } (x \text{ NOR } y)] + [y \text{ NOR } (x \text{ NOR } y)]} =$$

$$= (x \text{ NOR } (x \text{ NOR } y)) \text{ NOR } (y \text{ NOR } (x \text{ NOR } y))$$

Iar diagrama va fi:



Exemplul 4 Implementati poarta XOR printr-un circuit care contine doar porti NOR.

$$x \oplus y = \bar{x} \cdot y + x \cdot \bar{y}$$

Stim ca  $x \cdot \bar{x} = 0$

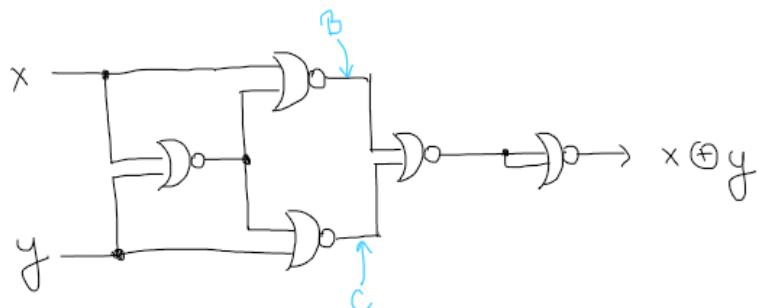
$$y \cdot \bar{y} = 0$$

$$\begin{aligned} x \oplus y &= \bar{x} \cdot y + x \cdot \bar{y} + x \cdot \bar{x} + y \cdot \bar{y} = \\ &= \bar{x} \cdot (x + y) + \bar{y} \cdot (x + y) = \\ &\stackrel{A=\bar{A}}{=} \overline{\bar{x} \cdot (x+y)} + \overline{\bar{y} \cdot (x+y)} = \\ &\stackrel{\text{de Morgan}}{=} \overline{\bar{x} + (\bar{x}+y)} + \overline{\bar{y} + (\bar{x}+y)} = \\ &\stackrel{A=\bar{A}}{=} \overline{[x + (\bar{x}+y)]} + \overline{[y + (\bar{x}+y)]} \end{aligned}$$

$$\begin{aligned} x \oplus y &= B + C = \overline{B + C} = \overline{B \text{ NOR } C} = \\ &= (\overline{B \text{ NOR } C}) \text{ NOR } (\overline{B \text{ NOR } C}) \end{aligned}$$

$$\begin{aligned} \text{unde } B &= \overline{x + (\bar{x}+y)} = x \text{ NOR } (x \text{ NOR } y) \\ C &= \overline{y + (\bar{x}+y)} = y \text{ NOR } (x \text{ NOR } y) \end{aligned}$$

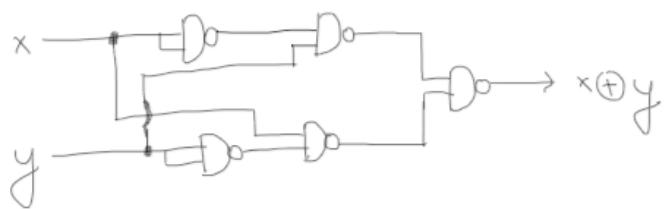
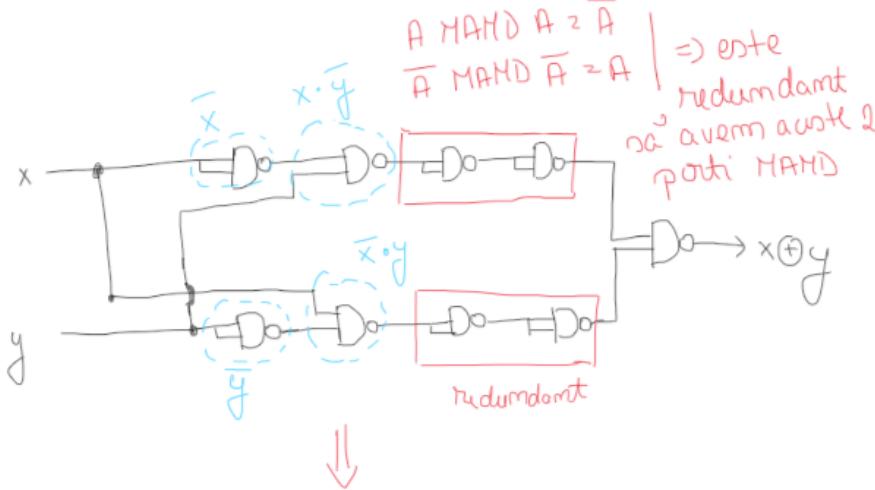
Iar diagrama va fi:



**Exemplul 5** Implementati poarta XOR printr-un circuit cu numar minim de porti si doar cu porti NAND.

$$\begin{aligned}
 x \oplus y &= x \cdot \bar{y} + \bar{x} \cdot y = \overline{\overline{x \cdot \bar{y}} + \overline{\bar{x} \cdot y}} = \\
 &\rightarrow \overline{(x \cdot \bar{y}) \cdot (\bar{x} \cdot y)} = \\
 &= \overline{x \cdot (y \text{ NAND } y)} \cdot \overline{(\bar{x} \text{ NAND } x) \cdot y} = \\
 &= \overline{(x \text{ NAND } (\bar{y} \text{ NAND } y)) \text{ NAND } (\bar{x} \text{ NAND } (\bar{y} \text{ NAND } y))} \cdot \\
 &\quad \cdot \overline{((\bar{x} \text{ NAND } x) \text{ NAND } y) \text{ NAND } ((x \text{ NAND } \bar{x}) \text{ NAND } y)}
 \end{aligned}$$

Chiar daca aceasta nu este forma cu numar minim de porti logice, vom desena diagrama pentru ca asa ne va fi foarte usor sa observam unde se realizeaza redundanta.

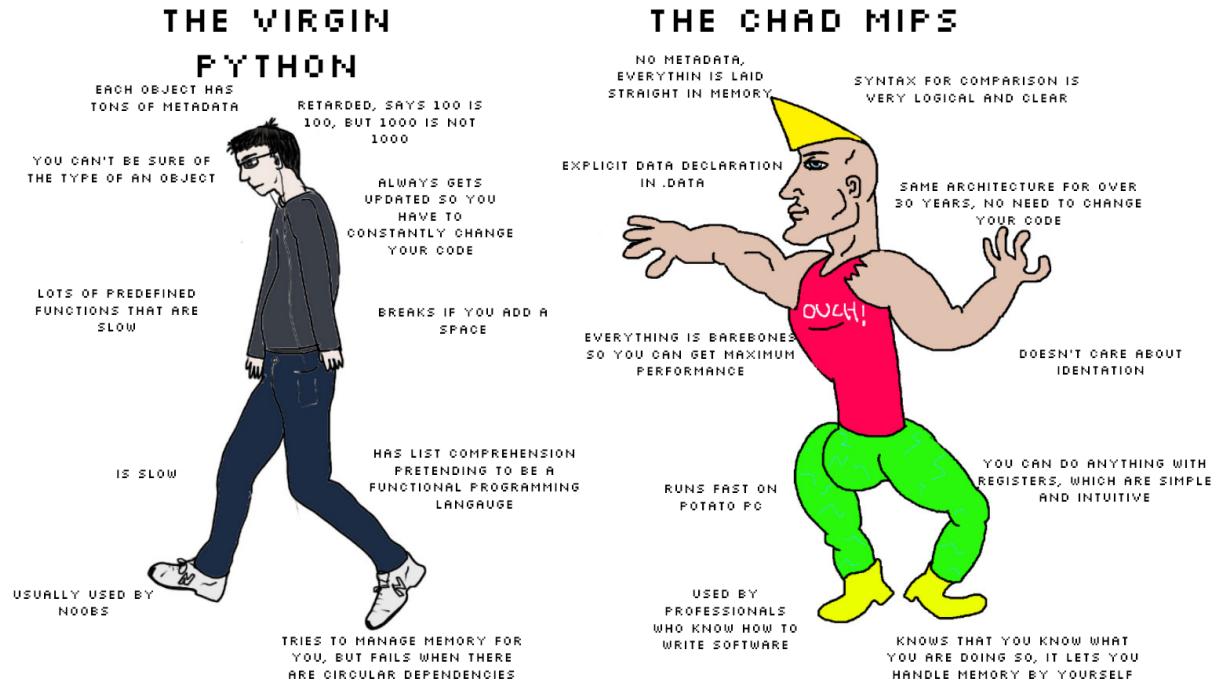


$$\Rightarrow x \oplus y = ((x \text{ NAND } y) \text{ NAND } ((y \text{ NAND } y) \text{ NAND } x))$$

**Exemplul 6** Implementati poarta NXOR printr-un circuit cu numar minim de porti si doar cu porti NAND.

Avand atatea exemple mai sus, incercati exercitiul acesta singuri.

### 3 MIPS



#### 3.1 Siruri de caractere

Sunt tablouri unidimensionale de un byte și sunt finalizate cu caracterul '\0'. Această convenție de a termina șirurile de caractere cu '\0' este comună în multe limbaje de programare. În felul acesta se pot parcurge caracterele, iar când se ajunge la capătul șirului să se opreasă, fără a mai fi nevoie de o variabilă auxiliară în care să se țină lungimea șirului.

Pentru a verifica dacă un caracter este egal cu '\0' se poate folosi beqz.

Exemplu:

Problema 1 Să se afișeze pe ecran lungimea unui șir de caractere.

```

1 .data
2     str:.asciiiz "Sir de caractere" # sirul de caractere
3 .text
4 main:
5
6     li $t0, 0                      # counterul
7     lb $t1, str($t0)               # iau primul caracter din sir
8
9 loop:
10
11    beqz $t1, exit                # daca am ajuns la sfarsitul sirului ne oprim
12
13    addi $t0, 1
14    lb $t1, str($t0)               # ia urmatorul caracter din sir
15    j loop
16
17 exit:
18
19    move $a0, $t0
20    li $v0, 1
21    syscall
22
23    li $v0, 10
24    syscall

```

## PRINT BYTE

Dacă avem un caracter stocat într-un registru îl putem afișa pe ecran. Apelul de sistem pentru asta are codul 11. În \$a0 vom copia caracterul pe care vrem să-l afișăm.

Exemplu PRINT BYTE:

```

1 .data
2     ch:. byte 'a'
3 .text
4 main:
5
6     lb $a0, ch # copiem in $a0 caracterul pe care vrem sa-l afisam
7     li $v0, 11 # punem in $v0 codul pentru apelul de sistem, adica 11
8     syscall    # afiseaza caracterul pe ecran
9
10    li $v0, 10
11    syscall

```

printbyte.s

## READ STRING

Avem apel de sistem pentru citirea sirurilor de caractere de la tastatură. Codul pentru acest apel de sistem este 8.

Pentru a citi un sir de caractere de la tastatură trebuie să facem următoarele lucruri:

- Punem în registrul \$a0 adresa de memorie la care vrem să reținem sirul de caractere
- Punem în registrul \$a1 dimensiunea maximă a sirului.
- Punem în registrul \$v0 codul pentru apelul de sistem, adică 8.

Exemplu READ STRING:

```

1 .data
2     str:. space 100 # sir de caractere de lungime 99
3             # lungimea este 99, nu 100
4             # deoarece ultimul caracter
5             # este '\0'
6 .text
7 main:
8
9     la $a0, str   # citeste sirul de caractere
10    li $a1, 99    # pune in $a0 adresa de memorie la
11        # care vreau sa retin sirul de caractere
12    li $v0, 8      # dimensiunea maxim a sirului
13    syscall        # codul pentru apelul de sistem este 8
14
15
16    la $a0, str   # afiseaza sirul de caractere
17    li $v0, 4      #
18    syscall        # exit
19
20    li $v0, 10
21    syscall

```

readstring.s

Mai multe exerciții

Problema 1: Se citește un sir de caractere de dimensiune maximă 99. Să se afișeze pe ecran caracterele de pe poziții pare.

Problema 2: Se dă un sir de caractere la nivel de memorie, să se modifice sirul adăugând un + 1 pe codul ASCII al fiecărui element.

Exemplu: "abc xyz" -> "bcd!yz{"

Problema 3: Prima parte din problema din prima zi de advent of code din 2017:

<https://adventofcode.com/2017/day/1>

### 3.2 Lucrul cu numere în virgulă mobilă



## Regiștri

În MIPS avem regiștri  $\$f0-\$f31$  dedicati pentru lucrul cu numere în virgulă mobilă. Fiecare regisztru este pe 32 de biți, deci poate ține un single. Pentru double, sunt folosiți doi regiștri (unul după altul). De exemplu dacă am vrea să ținem un double în regiștrii  $\$f0$ , respectiv  $\$f1$ , 32 de biți vor fi ținuți în  $\$f0$ , iar restul vor fi ținuți în  $\$f1$ .

## Tipuri de date

Se declară ca și celelalte tipuri de date în .data.

Pentru numerele în virgulă mobilă avem tipurile:

- .float - tip de date pe 32 de biți folosit pentru a stoca numere single
- .double - tip de date pe 64 de biți folosit pentru a stoca numere double

## Exemple de date declarate

```
a:.float 10.54  
b:.double 20.31
```

## Instrucțiuni pentru transferarea datelor din regiștri și invers

Când lucrăm cu single și double, de regulă postfixăm numele instrucțiunii noastre cu .s, respectiv .d. Din cauza faptului că pentru double sunt folosiți doi regiștri, instrucțiunile pentru double funcționează doar pe regiștrii cu număr par (ex:  $\$f0$ ,  $\$f2$ ,  $\$f4$ , etc). În unele implementări de MIPS, regula rămâne valabilă și pentru instrucțiunile pentru numere single.

### Instrucțiuni pentru încărcarea valorilor în regiștri:

l.s/ l.d  $\$fd$ , *mem* (load single/ load double, încarcă în registrul  $\$fd$ , valoarea aflată în memorie în *mem*. În cazul la double, va fi folosit și următorul regisztru pentru a se stoca în el. Este valabil pentru orice instrucțiune de transfer al datelor pe double)

li.s/ li.d  $\$fd$ , *const* (Încarcă o valoarea constantă *const* în registrul  $\$fd$ )

### Instrucțiuni pentru salvarea valorilor din regiștri în memorie:

s.s/ s.d  $\$fs$ , *mem* (store signle/ store double, salvează valoarea din registrul  $\$fs$  la adresa *mem*)

## Exemple de folosire a acestor instrucțiuni:

```
l.s $f0, a # încarcă în registrul $f0 valoarea din a (adică 10.54)  
l.d $f2, b # încarcă în regiștrii $f2 și $f3 valoarea din b (adică 20.31)  
li.s $f4, 15.3 # încarcă în registrul $f4 valoarea constantă 15.3  
li.d $f6, 21.9 # încarcă în registrul $f6 valoarea constantă 21.9  
s.s $f4, a # salvează valoarea din registrul $f4 la adresa de memorie a lui a
```

s.d \$f6, b # salvează valoarea din registrul \$f6 (respectiv \$f7) la adresa de memorie a lui b.  
instructiuni aritmetice

add.s/ add.d *dest, src1, src2* (adună numerele din registri *src1* și *src2* și salvează rezultatul în registrul *dest*. și la aceste instructiuni rămâne valabil comportamentul pentru double, sunt folosiți câte doi registri)

sub.s/ sub.d *dest, src1, src2* (asemănător cu add.s/ add.d, realizează scăderea dintre valoarele din registrul *src1* și *src2*, iar rezultatul îl salvează în registrul *dest*. sub.d este pentru double)

mul.s/ mul.d *dest, src1, src2* (realizează operația  $dest = src1 \cdot src2$ )

div.s/ div.d *dest, src1, src2* (realizează operația  $dest = src1 / src2$ )

neg.s/ neg.d *dest, src* (realizează operația  $dest = -src$ )

abs.s/ abs.d *dest, src* (realizează operația  $dest = \|src\|$ )

IO + apeluri de sistem

Coduri pentru apeluri de sistem:

2 - PRINT SINGLE (afisează pe ecran un număr single, se încarcă în \$f12 valoarea de afișat, în v0 valoarea 2 (codul pentru apelul de sistem), iar apoi se scrie syscall)

3 - PRINT DOUBLE (afisează pe ecran un număr double, se încarcă în \$f12, respectiv \$f13 valoarea de afișat, în v0 valoarea 3 (codul pentru apelul de sistem), iar apoi se scrie syscall)

6 - READ SINGLE (citește un single de la tastatură, se încarcă în v0 valoarea 6 (codul pentru apelul de sistem), apoi se scrie syscall. După ce a reușit să citească numărul de la tastatură, sistemul ne va returna valoarea citită în registrul \$f0)

7 - READ DOUBLE (citește un double de la tastatură, se încarcă în v0 valoarea 7 (codul pentru apelul de sistem), apoi se scrie syscall. După ce a reușit să citească numărul de la tastatură, sistemul ne va returna valoarea citită în registri \$f0, respectiv \$f1)

Exemple de apeluri de sistem

```
# READ SINGLE
```

```
li $v0, 6
```

```
syscall mov.s $f2, $f0 # În $f0 am primit valoarea citită de la tastatură. O vom copia în registrul $f2 cu ajutorul lui mov.s (folosim mov.d în cazul în care avem double). Sintaxa pentru mov.s este mov.s regd, regs. Instrucțiunea copiază valoarea din registrul regs în
```

registru *regd* (analog pentru *mov.d*).

## Exerciții și probleme

Problema 1: Se dau două numere de tip single stocate în memorie, să se interschimbe valoările celor două numere și să se afișeze pe ecran noile valori.

```
1 .data
2
3     x:.float 11.4      # x este un single declarat in memorie cu valoarea implicita 11.4
4     y:.float 34.5      # y -> single cu valoarea 34.5
5
6 .text
7 main:
8
9     l.s $f0, x          # incarc in $f0 valoarea din x
10    l.s $f2, y           # incarc in $f2 valoarea din y
11        # am ales $f2 in loc de $f1
12        # deoarece in unele medii de lucru
13        # nu putem folosi $f-urile impare
14        # nici pentru operatii cu single
15
16    s.s $f0, y           # pun in y valoarea din f0 (adica ce era inainte in x)
17    s.s $f2, x           # pun in x valoarea din f2 (adica ce era inainte in y)
18
19    li $v0, 2             # codul pentru a afisa un single pe ecran
20    mov.s $f12, $f2        # pun in $v12 valoarea din f2 (adica ce avem acum in x)
21    syscall               # afisez pe ecran
22
23    li $v0, 11            # afisez spatiu intre rezultate
24    li $a0, ','           # pun in a0 caracter pentru spatiu
25        # in unele medii (spim) putem face direct
26        # fara a mai declara in memorie spatiul
27    syscall
28
29    li $v0, 2
30    mov.s $f12, $f0        # acum afisez valoarea din $f0 (adica ce avem acum in y)
31    syscall
32
33    li $v0, 10
34    syscall
```

problema1\_1.s

## Branch instructions

Sunt asemănătoare cu cele de la word-uri, deși diferă puțin sintaxa și modul de lucru. Fiecare instrucțiune de verificare a condițiilor începe cu ”c.”, este urmată de ”eq”, ”lt” sau ”le”, iar apoi este urmată de ”.s” sau ”.d” (în funcție dacă folosim single sau double).

Instrucțiunile:

c.eq.s/ c.eq.d fs, ft (comparison equal, verifică dacă fs și ft sunt egale (deși nu aş recomanda să faceți asta vreodată cu float-uri... în niciun limbaj de programare). Dacă sunt egale, atunci setează bitului de condiție (parte din procesor) valoarea 1, altfel valoarea 0)

c.lt.s/ c.lt.d fs, ft (comparison less than, verifică dacă fs este mai mic decât ft, iar dacă este, setează valoarea bitului de condiție valoarea 1, altfel valoarea 0)

c.le.s/ c.le.d fs, ft (comparison less than or equal, verifică dacă fs este mai mic sau egal cu

ft, iar dacă este, setează valoarea bitului de condiție valoarea 1, altfel valoarea 0)

Asta nu e tot ce trebuie să facem, aceste instrucțiuni doar au modificat valoarea bitului de condiție. noi acum trebuie ne ducem la un branch dacă această valoare este 1 sau 0.

Pentru asta avem următoarele instrucțiuni:

bc1t label (continuă executarea de la linia unde este *label*, dacă bitul de condiție are valoarea 1)

bc1f label (continuă executarea de la linia unde este *label*, dacă bitul de condiție are valoarea 0)

## Exerciții și probleme

Problema 1: Să se afișeze pe ecran toate valorile pozitive mai mici sau egale cu n (single citit de la tastatură) pornind de la 0 și incrementând cu un step de 0.1.

```
1 .data
2
3     n:.float 0.0          # numarul n, il vom tine si in memorie (doar in scop didactic)
4
5 .text
6 main:
7
8     li $v0, 6            # citeste n
9     syscall
10    s.s $f0, n           # vom pastra in $f0 valoarea lui n
11
12    li.s $f2, 0.1         # pasul cu care vom incrementa counterul
13    li.s $f4, 0.0         # counterul nostru, initializat cu 0
14
15 loop:
16
17    c.le.s $f0, $f4        # daca counterul nostru este >= f0, atunci am terminat
18    # deci seteaza bitului de conditie valoarea 1
19    bc1t exit              # daca bitul de conditie este 1, am terminat
20    # din pacate nu avem branching ca la word-uri
21    # si nici nu putem verifica daca un numar e mai mare(sau egal)
22    # cu altul, trebuie sa verificam daca sunt mai mici(sau egale)
23
24    li $v0, 2              # afiseaza numarul curent
25    mov.s $f12, $f4
26    syscall
27
28    li $v0, 11
29    li $a0, '
30    syscall
31
32    add.s $f4, $f4, $f2 # incrementeaza counterul cu step
33    # adica $t4 = $t4 + $t2
34
35    j loop
36
37 exit:
38
39    li $v0, 10
40    syscall
```

## Conversii

Putem copia bit cu bit valorile din registri normali în registri în virgulă mobilă (și invers) folosind următoarele instrucțiuni:

`mtc1 rs, fd` (copiază din registrul normal *rs*, în registrul pentru floating point *fd*)

`mfc1 rd, fs` (copiază din registrul pentru floating point *fs*, în registrul normal *rd*)

Aceste instrucțiuni nu fac conversie între tipuri, ci doar copiază bit cu bit valorile. Pentru a converti avem instrucțiunea `cvt`, care se postfixează cu `".dest.sursa"`, unde *dest* și *sursa* reprezintă tipul către care vrem să convertim, respectiv tipul de la care convertim. Dest și sursă pot avea valorile `"s"`, `"d"`, respectiv `"w"` (pentru word).

Sintaxa: `cvt.dest.sursa ds, sr ->` convertește valoarea din *sr* (de tipul sursa) în valoare de tip *dest* și salvează rezultatul în registrul *ds*.

## Exerciții și probleme

Problema 1: Se citește un număr de tip single de la tastatură, să se convertească la word, iar apoi să se afișeze pe ecran rezultatul.

```

1 .data
2 .text
3 main:
4
5     li $v0, 6
6     syscall
7
8     cvt.w.s $f2, $f0 # convertim pe $f0 (citit) în word, și tinem rezultatul în $f2
9         # noi am vrea să-l tinem în $t0 pentru că este word, dar compilerul nu
10        # ne lasă
11        # astăzi trebuie să punem un registru pentru float-uri ca destinație
12        # punem rezultatul din $f2 în $t0 (în $t0, este reprezentat intern ca
13        # word, dar este în registru de float, deci îl copiem într-un registru
14        # pentru word-uri)
15
16     li $v0, 1          # afișăm pe ecran rezultatul
17     move $a0, $t0
18     syscall
19
20     li $v0, 10
21     syscall

```

## Mai multe exerciții

Problema 1: Se citesc  $n \in N^*$  și  $n$  numere de tip single. Să se calculeze media lor aritmetică.

## References

- [1] Dumitru Daniel Drăgulici. *Curs Arhitectura Sistemelor de Calcul.*
- [2] Larisa Dumitrache. *Tutoriat 2019*
- [3] Bogdan Macovei. *Laboratoare ASC 2019/ 2020*
- [4] Advent Of Code. *Day 1 2017*