

## Tutoriat 7

Stan Bianca-Mihaela, Stăncioiu Silviu

December 2020

CAND SILVIU DE LA ASC ARE NUMAI TAIETURI,  
TYPO-URI SI GRESELI IN MATERIALE CA SA  
PREGATEASCA STUDENTII PENTRU CURSUL  
STRUCTURI DE DATE DE PE SEMESTRUL 2



### Contents

- 1 Rezolvarea subiectului 3, punctul c)

2

<b>2 Recapitulare</b>	<b>6</b>
2.1 Examen 2016 . . . . .	6
2.1.1 Subiectul I . . . . .	6
2.1.2 Subiectul II . . . . .	12
2.1.3 Subiectul III . . . . .	17
<b>3 Subiect 3 rezolvat la curs</b>	<b>25</b>

## 1 Rezolvarea subiectului 3, punctul c)

Exemplul 1 [RESTANTA SEPRTEMBRIE 2020]

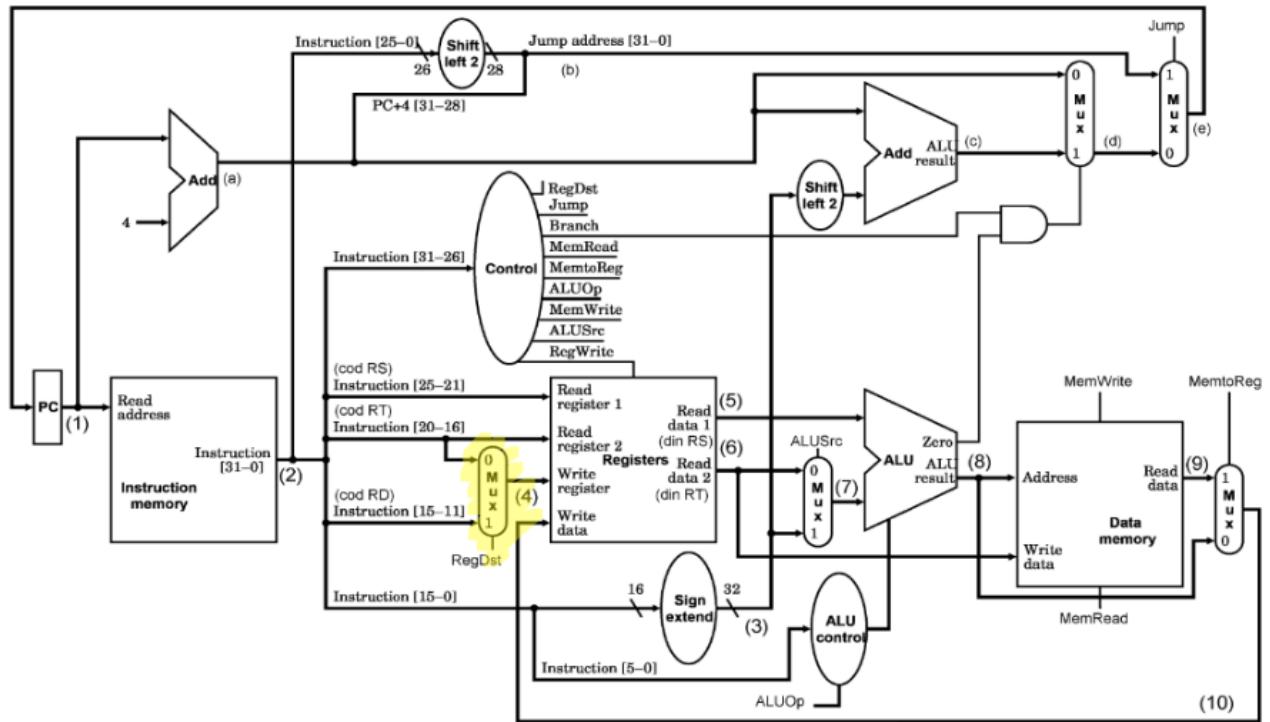
c) Adăugați procesorului implementarea instrucției: dlw rd, rs, rt  
care încarcă în registrul rd valoarea afărată în memorie la adresa care se obține adunând valorile din regiștrii rs și rt (adică efectueză  $rd := \text{mem}[rs + rt]$ ).

Pentru implementare, este suficientă adăugarea unei linii tabelului "Control" (codul op = - este o valoare nouă, nerelevantă). Completați această linie:

Instruction [31 - 26]	RegDst	ALU Src	Mem To Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
—										

Sa analizam putin cerinta si sa luam coloanele pe rand:

- RegDst: Ce facea RegDst? Sa ne amintim din procesor:



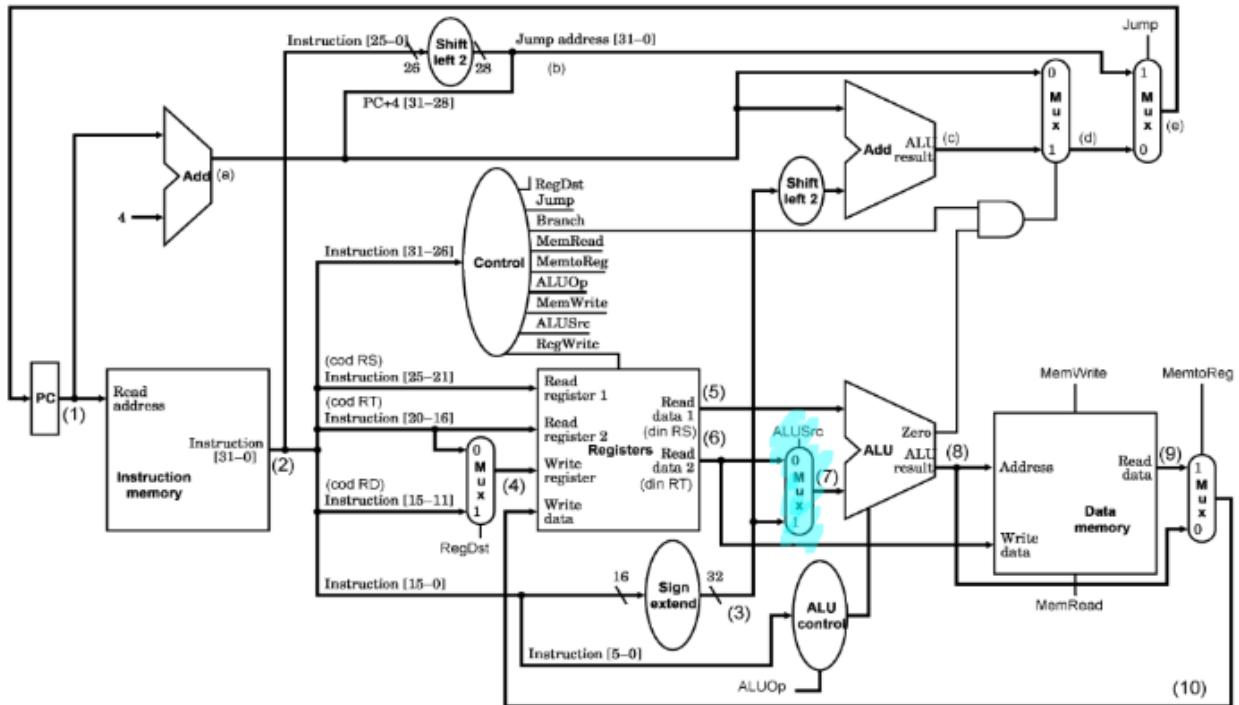
RegDst decide cine este registrul destinație.

- Daca avem o instructiune de tip R, registrul destinație este rd.  $\Rightarrow \text{RegDst}=1$
- Daca avem o instructiune de tip I, cum ar fi li \$t0, 5, registrul destinație este chiar rt. $\Rightarrow \text{RegDst}=0$
- Cand avem instructiuni de genul sw sau beq, nu avem niciun registru destinație.  $\Rightarrow \text{RegDst}=X$  (aka. niciuna dintre variante).

Acum sa ne gandim ce fel de instructiune trebuie sa construim noi. Vedem din prima ca avem un rd, deci RegDst=1.

Instruction [31 - 26]	RegDst	ALU Src	Mem To Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
—	1									

- ALUSrc: Ce facea ALUSrc?



Face diferenta dintre instructiunile care fac operatii pe valori si cele care fac operatii pe adrese de memorie.

- Pentru instructiunile de tip R si cele de tip branch o sa trimita ca input catre Unitatea Aritmetica si Logica rs si rt pentru ca ele fac operatii pe valori. (ALUSrc=0)
- Pentru instructiunile de tip I o sa trimita ca input catre Unitatea Aritmetica si Logica valoarea lui rs si valoarea imediata (imm) pentru ca vrem sa facem operatii pe niste adrese de memorie.(ALUSrc=1)

Din nou, noi avem o instructiune de tip R, deci ALUSrc va fi 0.

Instruction [31 - 26]	RegDst	ALU Src	Mem To Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
—	1	0								

- MemToReg: Are voie instructiunea noastra sa citeasca din memorie si sa scrie intr-un registru? Da,

in cerinta scrie ca instructiunea **incarca in rd** o valoare aflata la o **adresa de memorie**. => MemToReg=1.

Instruction [31 - 26]	RegDst	ALU Src	Mem To Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
—	1	0	1							

- RegWrite: Are voie instructiunea noastra sa scrie intr-un regisztr? Clar. => RegWrite=1.

Instruction [31 - 26]	RegDst	ALU Src	Mem To Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
—	1	0	1	1						

- MemRead: Are voie instructiunea noastra sa citeasca din memorie? Da. => MemRead=1.

Instruction [31 - 26]	RegDst	ALU Src	Mem To Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
—	1	0	1	1	1					

- MemWrite: Are voie instructiunea noastra sa scrie din memorie? Nu. => MemWrite=0.

Instruction [31 - 26]	RegDst	ALU Src	Mem To Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
—	1	0	1	1	1	0				

- Branch: E instructiunea noastra de tip branch? Nu. => Branch=0.

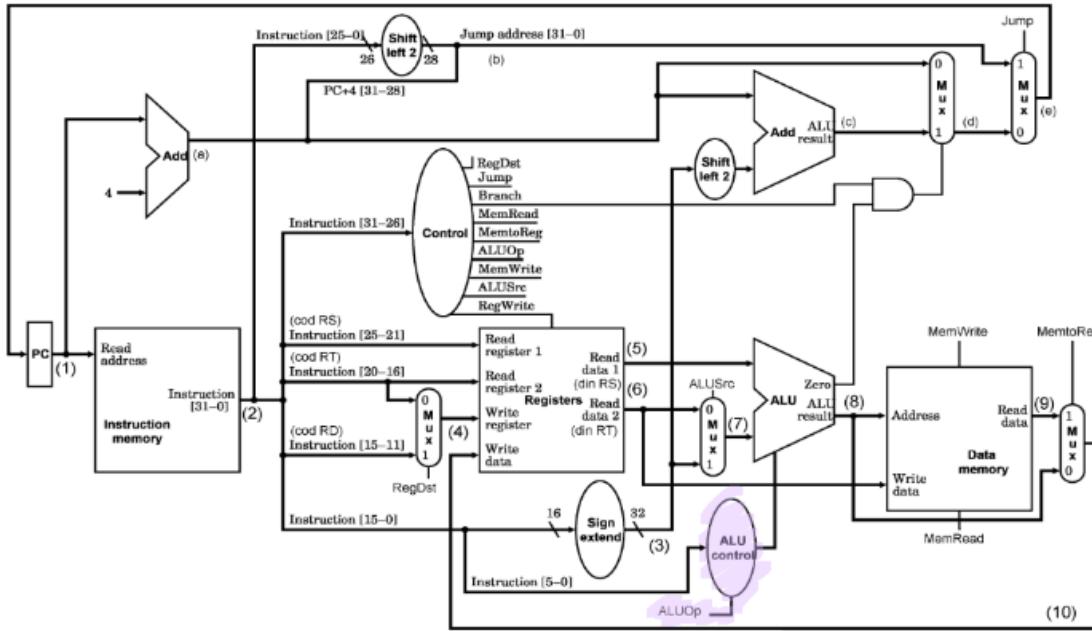
Instruction [31 - 26]	RegDst	ALU Src	Mem To Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
—	1	0	1	1	1	0	0			

- Jump: E instructiunea noastra de tip branch? Nu. => Jump=0.

Instruction [31 - 26]	RegDst	ALU Src	Mem To Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
—	1	0	1	1	1	0	0	0		

- ALUOp (o sa tratam si ALUOp1 si ALUOp2 intr-o singura sectiune pentru ca e aceeasi discutie pe ambele).

Vom face o analiza mai amanuntita pentru ALUOp. Stim ca ALUOp intra in ALUcontrol, care scoate ce fel de operatie va efectua Unitatea Aritmetica si Logica (ALU).



Dar ce mai exact inseamna aceste perechi de numere (ALUOp1, ALUOp2)?

- (ALUOp1=0, ALUOp2=0) : fa intotdeauna adunare (addi, lw, sw, etc.)
- (ALUOp1=0, ALUOp2=1) : fa intotdeauna scadere (beq, bne, etc.)
- (ALUOp1=1, ALUOp2=X) : fa o operatie determinata de campul Funct (din instructiunile de tip R)

exemplu: La add vrem sa facem adunare, la sub scadere, etc.

Acum ca stim asta, putem sa alegem valorile pe care le vrem noi. Instructiunea noastra trebuie sa:

- determine adresa de memorie de la care sa citeasca, prin adunare
- sa incarce in registrul rd valoarea gasita la acea adresa de memorie, tot prin adunare (daca ne uitam la valorile lui ALUOp1 si ALUOp2 din tabelele de ajutor avem perechea (0,0), care inseamna adunare).

Deci vom alege adunarea, adica ALUOp1=0, ALUOp2=0.

Instruction [31 - 26]	RegDst	ALU Src	Mem To Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
—	1	0	1	1	1	0	0	0	0	0

## 2 Recapitulare

Ne apucăm să rezolvăm subiecte date în alți ani la examenul de ASC. Vom începe cu subiectele date în 2016.

### 2.1 Examen 2016

#### 2.1.1 Subiectul I

Fie  $x = 73.75$  și  $y = 5.25$

- Convertiți  $x$  și  $y$  în baza 2.
- Convertiți mai departe  $x$  și  $y$  din baza 2 în baza 16, direct (fără a trece prin baza 10).
- Calculați  $x - y$  lucrând în baza 16.
- Convertiți rezultatul din baza 16 în baza 10.
- Calculați  $73.75 + (-5.25)$  folosind algoritmul de adunare în virgulă mobilă pentru formatul single (se va lucra cu reprezentările matematice în baza 2 în notație științifică, iar în final se va converti rezultatul în baza 10).
- Determinați reprezentarea internă ca single a lui  $x$ , binară (32 biți) și hexa (8 cifre hexa).
- Interpretați ca număr în baza 10 reprezentarea internă hexa în virgulă mobilă  $0x8C$ , considerând formatul dat de  $n = 8$  (dimensiunea locației) și  $k = 2$  (dimensiunea câmpului caracteristică).

Rezolvare:

a)

$$(73,75)_2 = ?$$

$$\begin{array}{r} 73 \\ 36 \\ 18 \\ 9 \\ 4 \\ 2 \\ 1 \end{array} \left| \begin{array}{l} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{array} \right. \Rightarrow (73)_2 = \overline{1001001}$$

$$0,75 \cdot 2 = 1,5 \quad | \Rightarrow (0,75)_2 = \overline{0,11}$$

$$0,5 \cdot 2 = 1,0 \quad | \Rightarrow (0,75)_2 = \overline{0,11}$$

$$\Rightarrow (73,75)_2 = \overline{1001001,11}$$

$$(5,25)_2 = ?$$

$$\begin{array}{r} 5 \\ 2 \\ 1 \end{array} \left| \begin{array}{l} 1 \\ 0 \\ 1 \end{array} \right. \Rightarrow (5)_2 = \overline{101}$$

$$0,25 \cdot 2 = 0,5 \quad | \Rightarrow (0,25)_2 = \overline{0,01}$$

$$0,5 \cdot 2 = 1,0 \quad | \Rightarrow (0,25)_2 = \overline{0,01}$$

$$\Rightarrow (5,25)_2 = \overline{101,01}$$

b)

$16 = 2^4 \Rightarrow$  grupăm câte 4

$$\underbrace{0100}_{\text{1}}, \underbrace{1001}_{\text{1}}, \underbrace{1100}_{\text{0}}$$

$$\begin{aligned} (0100)_2 &= (4)_{16} \\ (1001)_2 &= (9)_{16} \\ (1100)_2 &= (C)_{16} \end{aligned} \quad | \quad \Rightarrow x = (\overline{49, C})_{16}$$

$$\underbrace{0101}_{\text{1}}, \underbrace{0100}_{\text{0}}$$

$$\begin{aligned} (0101)_2 &= (5)_{16} \\ (0100)_2 &= (4)_{16} \end{aligned} \quad | \quad \Rightarrow y = (\overline{5, 4})_{16}$$

c) Vom înmulți cele două numere cu 16 pentru a scăpa de virgulă, iar rezultatul îl vom înmulți cu 16 pentru a obține răspunsul corect.

$$\begin{array}{r} 49C - \\ \underline{54} \\ \hline 448 \end{array} \Rightarrow x - y = (\overline{44, 8})_{16}$$

d)

$$(44,8)_{16}^{-1} = ?$$

$$\begin{aligned}(44,8)_{16}^{-1} &= 4 \cdot 16^1 + 4 \cdot 16^0 + 8 \cdot 16^{-1} = \\ &= 64 + 4 + \frac{8}{16} = 68,5\end{aligned}$$

e)

$$\begin{aligned}(\overline{1001001,11})_2 &= (\overline{1,00100111}) \cdot 2^6 \\(-\overline{101,01})_2 &= (-\overline{1,0101})_2 \cdot 2^2\end{aligned}$$

OVERFLOW / UNDERFLOW:

$$-126 \leq g \leq 127 \quad \checkmark$$

$$-126 \leq d \leq 127 \quad \checkmark$$

ROTATION

$$|p_x| = |00100111| = b = 23$$

$$|p_y| = |0101| = h = 23$$

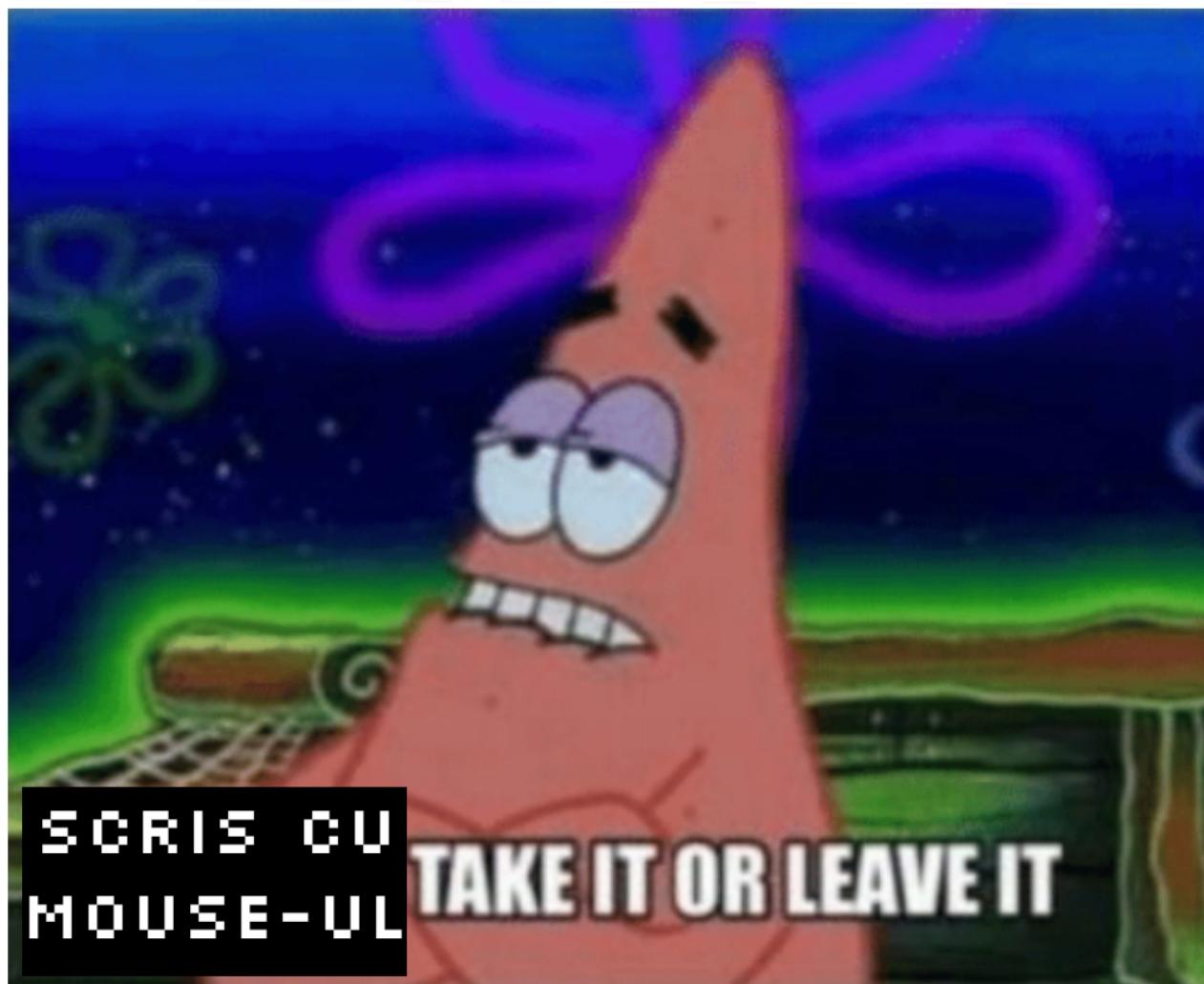
$2 < b \Rightarrow$  add c exp bei y da x

$$\Rightarrow y = -0,000\ 10101 \cdot 2^6$$

$$\begin{array}{r} 1,00100111 - \\ 0,00010101 \\ \hline 1,00010010 \end{array}$$

---

CAND NU AI TABLETA  
GRAFICA SI TI-E LENE SA  
SCRII TUTORIATUL IN LATEX



f)

Repräsentation (intern)

$$m=32, k=8, p=24$$

$$x = (\overline{1,00100111})_2 \cdot 2^6$$

$$\Rightarrow E = 6$$

$$f = 00100111 - \text{mantissa}$$

$$s=0 \text{ (positiv)}$$

$$c = E + \text{BIAS} = 6 + 127 = 133$$

TESTE

OVERFLOW / UNDERFLOW

$$-126 \leq c \leq 127 \quad \checkmark$$

ROUNDSIPE

$$|f| = 6 \leq 23 \quad \checkmark$$

$\Rightarrow x: 0 \underline{10000101} 00100111 \underbrace{00 \dots 0}_{23-8 = 0^4-\text{bit}} \underbrace{\text{mantissa}}$

Hexa: 42958000

g)

## INTERPRETARE 0x8C

$$m = 8 \quad k = 2$$

$$\begin{array}{l} b = \overline{1000} \\ c = \overline{1100} \end{array} \quad | \quad \Rightarrow \text{reprzentarea binară:} \\ 10001100$$

s : 1 (primul bit, mi se pare că avem un negativ)

c : 00 (următoarea  $k = 2$  cifru)

f : 01100 (mantina)

c = 00  $\Rightarrow$  format dinormalizat

$$\Rightarrow x = (-1)^s \cdot 2^{E_{\text{min}}} \cdot 0.f_p$$

$$E_{\text{min}} = -(2^{k-L} - 2) = 0$$

$$x = -\overline{0, f_p} = -\overline{0, 01100}$$

$$(x)_2^{-1} = -0,375$$

### 2.1.2 Subiectul II

Fie  $f : B_2^3 \rightarrow B_2^3$ ,  $f(x, y, z) = (f_1(x, y, z), f_2(x, y, z))$ , unde  $f_1, f_2 : B_2^3 \rightarrow B_2$ ,  $f_1(x, y, z) = y + \bar{x}z$ ,  $f_2(x, y, z) = 1$  dacă și numai dacă cel puțin două dintre variabilele x, y, z au valoarea 1.

a) Construiți tabelul de valori al lui  $f$  și scrieți  $f_1, f_2$  în FND și FNC.

b) Implementați  $f$  folosind un PROM.

c) Implementați  $f$  folosind un codificator.

d) Implementați  $f$  folosind multiplexori elementari, apoi reduceți la maximum numărul de multiplexori elementari, iar în final redesenați circuitul păstrând doar multiplexorii rămași.

e) Desenați un circuit 1-DS care, primind la intrare un sir de biți, scoare la ieșire 1 dacă și numai dacă cel puțin doi dintre ultimii 3 biți au valoarea 1.

f) Considerăm automatul  $(Q, X, Y, \delta, \lambda)$ , unde:

$$Q = \{0, 1\}^2, X = Y = \{0, 1\},$$

$$\delta(q_1, q_0, x) = (f_1(q_1, q_0, x), f_2(q_1, q_0, x)), \lambda(q_1, q_0, x) = f_1(q_1, q_0, x) \oplus f_2(q_1, q_0, x).$$

Implementați acest automat în maniera standard, folosind PLA și TFF.

Rezolvare:

a)

Tabelul pentru aceasta functie este:

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

FMD

$$f_1 = \underset{(2)}{\bar{x}} \cdot \underset{(3)}{\bar{y}} \cdot \underset{(4)}{z} + \underset{(3)}{\bar{x}} \cdot \underset{(5)}{y} \cdot \underset{(6)}{\bar{z}} + \underset{(4)}{\bar{x}} \cdot \underset{(6)}{y} \cdot \underset{(7)}{z} + \underset{(5)}{x} \cdot \underset{(6)}{y} \cdot \underset{(7)}{\bar{z}} + \underset{(6)}{x} \cdot \underset{(7)}{y} \cdot \underset{(7)}{z}$$

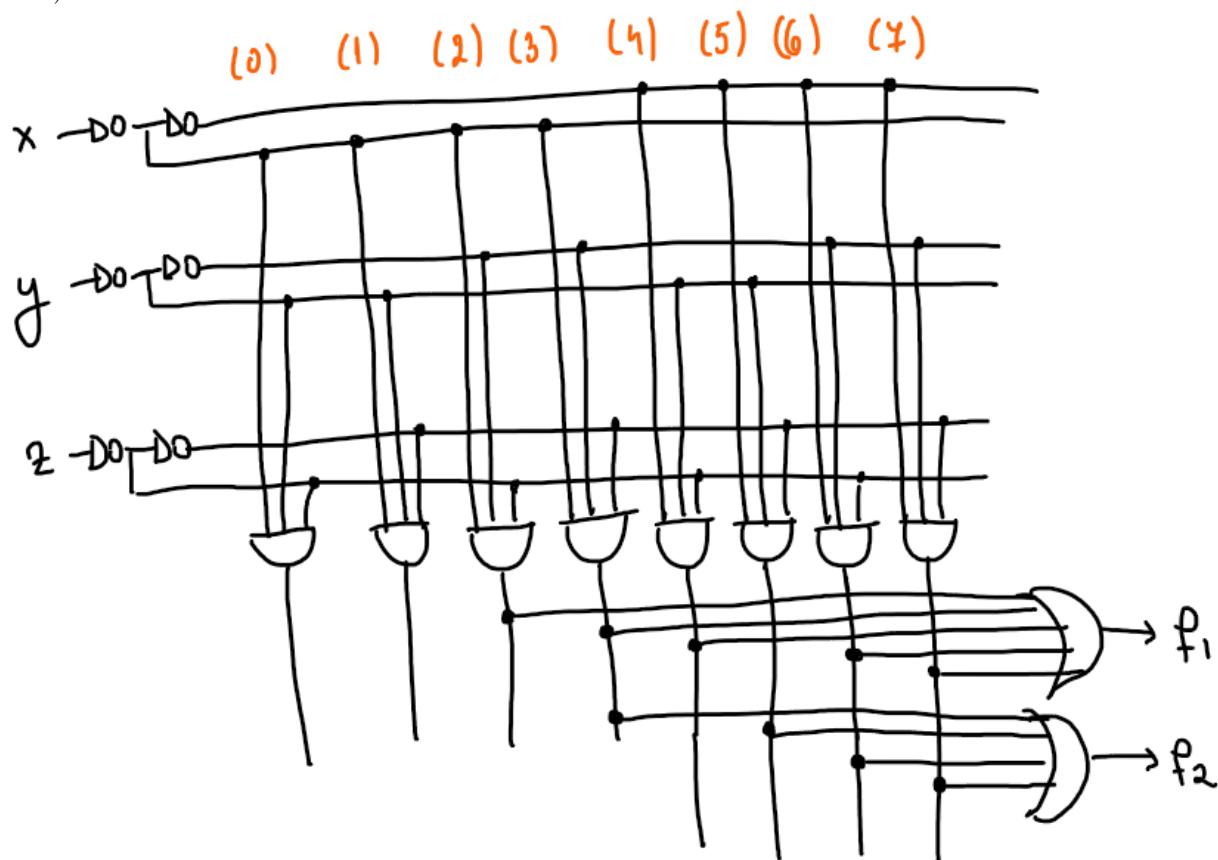
$$f_2 = \underset{(5)}{\bar{x}} \cdot \underset{(6)}{y} \cdot \underset{(7)}{z} + \underset{(6)}{x} \cdot \underset{(7)}{\bar{y}} \cdot \underset{(7)}{z} + \underset{(6)}{x} \cdot \underset{(7)}{y} \cdot \underset{(5)}{\bar{z}} + \underset{(7)}{x} \cdot \underset{(7)}{y} \cdot \underset{(5)}{\bar{z}}$$

FMC

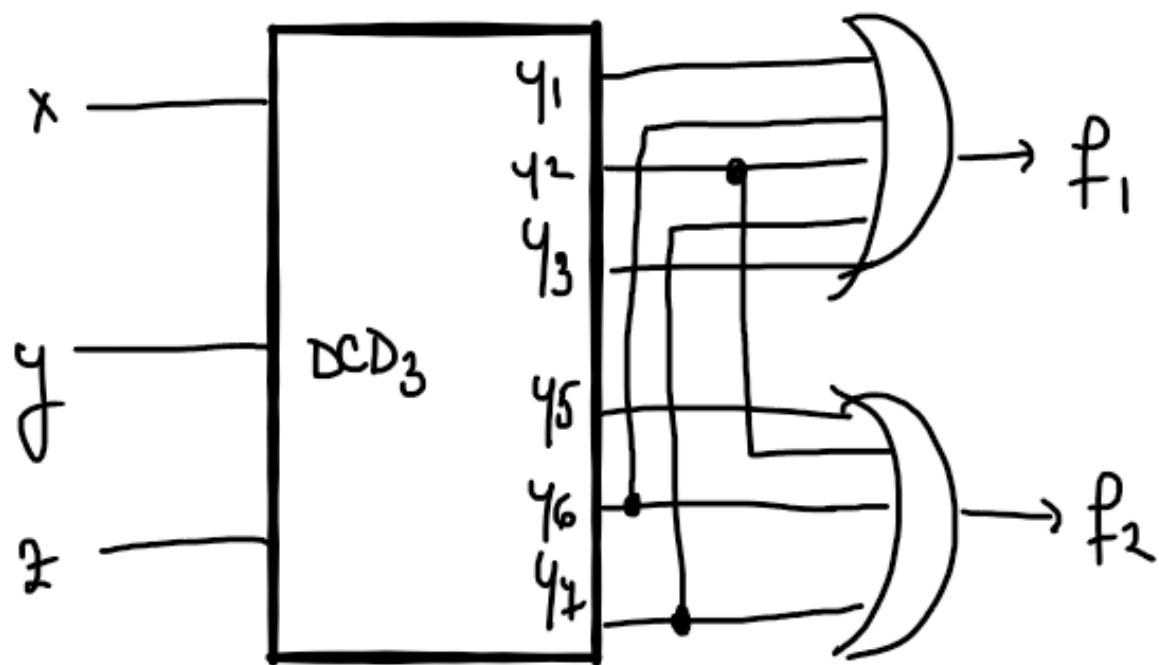
$$f_1 = \underset{(0)}{(x+y+z)} \cdot \underset{(4)}{(\bar{x}+y+z)} \cdot \underset{(5)}{(\bar{x}+\bar{y}+\bar{z})}$$

$$f_2 = \underset{(0)}{(x+y+z)} \cdot \underset{(1)}{(x+y+\bar{z})} \cdot \underset{(2)}{(x+\bar{y}+z)} \cdot \underset{(4)}{(\bar{x}+y+\bar{z})}$$

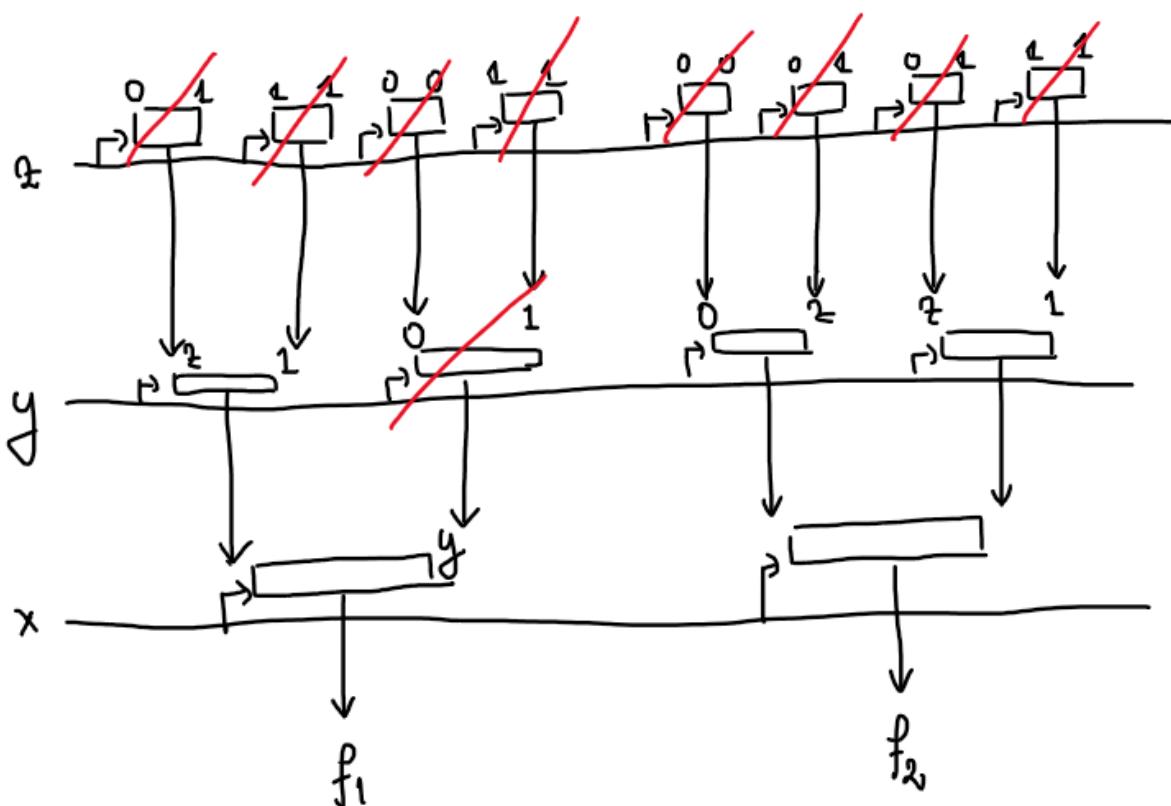
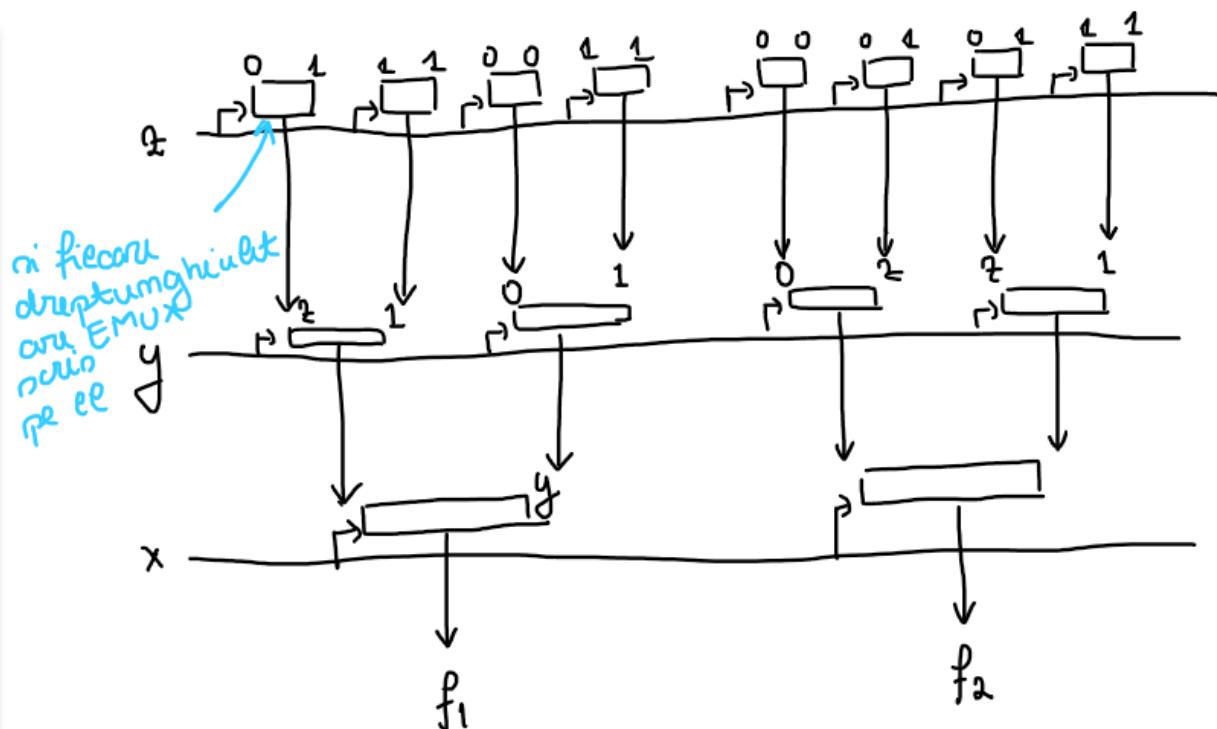
b)



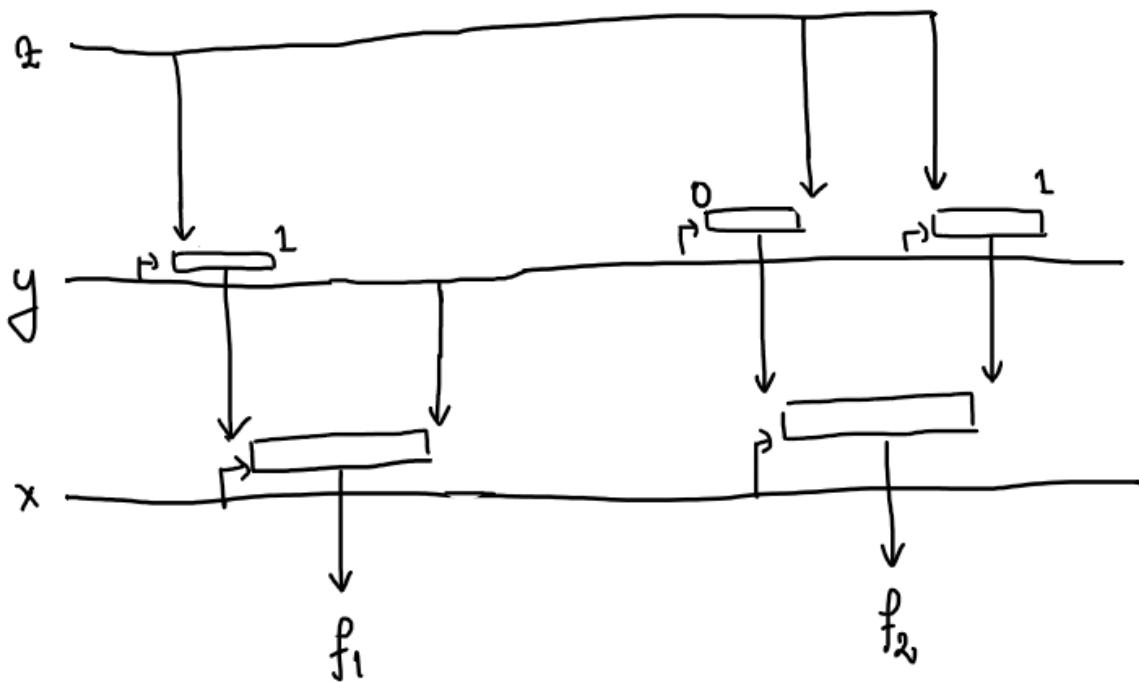
c)



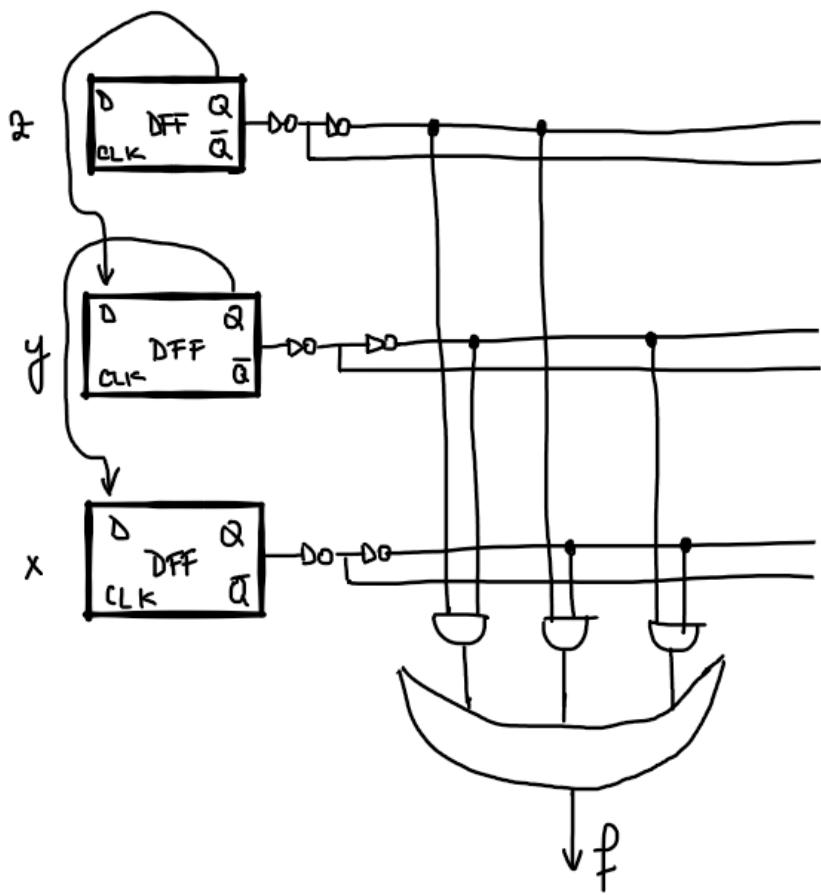
d)



Minimizarea multiplexorilor elementari:



e)



f) Punctul asta are automate, pe care o sa le faceti la Limbaje Formale si Automate. Anul trecut a zis ca nu o sa dea asa ceva, cel mai probabil asa va fi si la voi.

### 2.1.3 Subiectul III

Consideram implementarea procesorului MIPS cu 1 ciclu/ instructiune. Fie programul:

```
.data
x: .word 3
y: .word 4
.text
main:
la $t1, x
li $t2, 2
lw $t3, y
et:
sub $t3, $t3, $t2
sw $t3, 4($t1)
beq $t3, $t2, et
li $v0, 10
syscall
```

Presupunem ca in memorie instructiunea sw din program are adresa  $\alpha$ , iar variabila x are adresa  $\beta=0x10010000$ .  
a) Pentru instructiunile sw si beq din program scrieti campurile din reprezentarea lor interna (ex: op/rs/rt/imm, valorile se scriu in hexa); pentru beq din program scrieti reprezentarile ei binara (32 biti) si hexa (8 cifre hexa).

Incepem cu sw  $\$t3, 4(\$t1)$ . Ne uitam in pdf-ul cu instructiunile MIPS (gasiti un link la el si in tutoriatul 6 si in Teams, la Files). Vedem ca sw are forma sw  $\$t$ , offset( $\$s$ ) si encoding-ul:

1010 11ss ssst tttt iiiiiiiiiiiiiiiii

In tutoriatul 6 avem o poza cu scris de mana care ne spune ce cod are fiecare regisztr. Deci:

- $\$t3 = (11)_{16} = \$t = \overline{B}$  (in hexa)
- $\$t4 = (12)_{16} = \$s = \overline{C}$  (in hexa)
- offset =  $(4)_{16} = \overline{4}$  (in hexa)

Care e valoarea lui opcode in hexa?

$$\overline{(101011)}_2 = \overline{(2B)}_1 6$$

Deci reprezentarea in memorie a lui sw putem sa o scriem ca:

sw: 2B/C/B/4 (reprezentarea asta cu / respecta formatul din cerinta)

Cea pe biti este:

sw: 1010 1101 1000 1011 0000 0000 0000 0100 (in binar)

sw: AD8B0004 (in hexa).

Pentru beq (beq  $\$t3, \$t2, et$ ) stim ca are forma beq  $\$s, \$t$ , offset si encoding-ul:  
0001 00ss ssst tttt iiiiiiiiiiiiiiiii

Din nou, care sunt registri:

- $\$t3 = (10)_2 = \$t = \overline{01010}$  (pe 5 biti)
- $\$t2 = (11)_{16} = \$s = \overline{01011}$  (pe 5 biti)

- offset =  $(-3)_{16}$

Cum scriem -3 pe 16 biti? Complement fata de 2.

Reprezentarea lui 3 pe 16 biti: 0000 0000 0000 0011. Complementul fata de 1 si apoi adun 1.

$$\begin{array}{r}
 \boxed{N} \quad \underline{\text{0000} \quad \text{0000} \quad \text{0000} \quad \text{0011}} \\
 \underline{\text{1111} \quad \text{1111} \quad \text{1111} \quad \text{1100}} \quad + \\
 \underline{\hspace{10em}} \quad \underline{1} \\
 \underline{\hspace{10em}} \quad \underline{\text{1111} \quad \text{1111} \quad \text{1111} \quad \text{1101}}
 \end{array}$$

Deci offset =  $\overline{1111111111111100}$ .

Aici ne cere explicit reprezentarea pe biti (deci nu cea cu /). Aceasta va fi:

beq: 0001 0001 0100 1011 1111 1111 1100 (in binar)

beq: 114BFFFC (in hexa)

b) Completati tabelul urmator cu valorile obtinute la prima executare a instructiunilor sw si beq din program (am notat prescurtat: B=Branch, MR=MemRead, MW=MemWrite, iar Mem[y] inseamna continutul variabilei y); valorile se scriu hexa/formula, iar daca valoarea este necunoscuta/nedefinita o vom nota cu "?"; in coloanele PC si Mem[y] se vor trece valorile noi, de la sfarsitul fiecarei instructiuni execute:

Inainte sa trecem mai departe trebuie sa analizam ce se intampla la fiecare instructiune:

- Ce se intampla pana la sw?

x are valoarea 3

y are valoarea 4

\$t1 tine adresa lui x, adica  $\beta$

\$t2 are valoarea 2

\$in y pun valoarea lui \$t3, adica 2

din \$t3 se scade \$t2 si de pune la loc in \$t3, deci \$t3 are acum valoarea 2

- Ce se intampla pana la beq?

la sw am pus in y valoarea din \$t3, care este 2

Incepem cu valorile care pot fi luate din tabelele de ajutor:

	1	3	5	7	8	ALU zero	(d)	(e)	B	M2	MW	ALU Src	ALU Op (2b)	ALU Ctrl (3b)	PC	MemtoReg
Initial	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$\alpha$	5
sw	$\alpha$								0	0	1	1	00	010		
beq									1	0	0	0	X1	110		

na ignorati complet  
la intamplare  
cu game

Instruction	RegDst	ALUSrc	MemtoReg	Reg Write	Mem Read	Mem Write	Branch	
R-format	1	0	0	1	0	0	0	
lw	0	1	1	1	1	0	0	
sw	X	1	X	0	0	1	0	
beq	X	0	X	0	0	0	1	

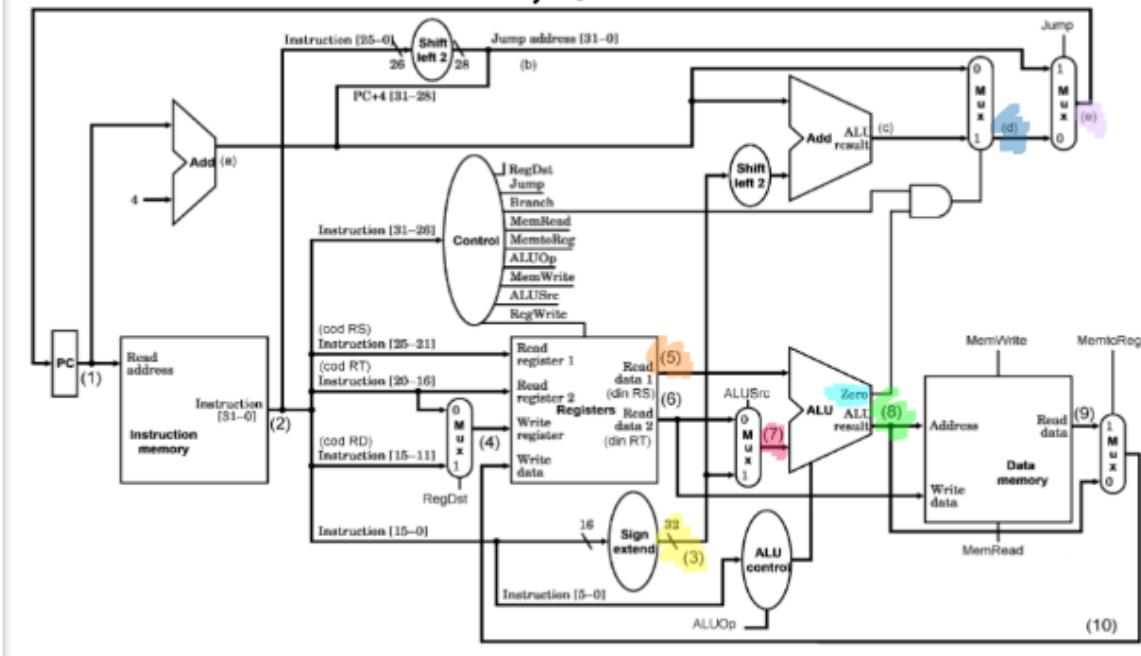
### ALU Control (slide 7.36)

lw/sw beq add sub and or R- format slt	ALUOp		Camp functie					Operatie	
	ALUOp <sub>1</sub>	ALUOp <sub>0</sub>	F5	F4	F3	F2	F1	F0	
	0	0	X	X	X	X	X	X	010 (+)
	X	1	X	X	X	X	X	X	110 (-)
	1	X	X	X	0	0	0	0	010 (+)
	1	X	X	X	0	0	1	0	110 (-)
	1	X	X	X	0	1	0	0	000 (and)
	1	X	X	X	0	1	0	1	001 (or)
	1	X	X	X	1	0	1	0	111 (slt)

Continuam cu celelalte valori. Le-am completat cu valorile deja in hexa. Sunt fix chestiile pe care le-am facut tutoriatul trecut, deci exercitiul asta ar trebui sa fie o verificare pentru voi.

	1	3	5	7	8	ALU zero	(d) (e)	$\beta$	MR	MW	ALU Src	ALU Op (3b)	ALU Cntr (3b)	PC	Mem[y]
Initial	—	—	—	—	—	—	—	—	—	—	—	—	—	$\alpha$	4
lw	$\alpha$	4	$\beta$	4	$\beta + 4$	?	$\alpha + 4$	$\alpha + 4$	0	0	1	1	00	010 $\alpha + 4$	2
beq	$\alpha + 4$	-3	2	2	0	1	$\alpha - 4$	$\alpha - 4$	1	0	0	0	x1	110 $\alpha - 4$	2

$(\alpha + 8) + (-3) \cdot 4 = \alpha - 4$



Analizam de ce am pus fiecare valoare pentru sw:

- 3: pe acolo intra valoarea lui offset, pe 32 de biti, care este 4
- 5:iese valoarea din rs, adica valoarea din \$t1, adica  $\beta$
- 7: AluSrc=1 => iese ce intra pe la 1, adica valoarea din (3)
- 8: se face operatia de adunare ( codul 010 corespunde operatiei de adunare ) intre (5) si (7). =>  $\beta + 4$
- ALU zero: nu avem nicio conditie, deci ALU zero nu stim ce valoarea are (?).
- (d): valoarea portii AND este 0, deci iese ce intra pe la 0.
- (e): sw nu este instructiune de tip J, deci iese ce intra pe la 0.
- PC: are valoarea lui (e)
- Mem[y]: in sw se pune la adresa  $\beta + 4$  valoarea din \$t3. y se afla chiar la adresa  $\beta + 4$ . Iar valoarea lui \$t3 este 2. Deci y are acum valoarea 2.

Acum, de ce am pus fiecare valoare pentru beq:

- 1: beq este fiz urmatoare instructiune dupa sw, deci de avanseaza doar cu 4
- 3: punem valoarea din offset, care este -3
- 5: punem valoarea din \$t3, adica 2
- 7: ALUSrc=0, deci pun valoarea din \$t2, adica 2
- 8: fac diferența (110 este codul pentru scadere) dintre valoarea din (5) și valoarea din (7)
- ALUzero: condiția era de egalitate (beq). Cum scaderea dintre cele 2 valori = 0 => termenii sunt egali => condiția se verifică => ALUzero=1.
- (d): poarta logica AND scoate 1, deci prin (d) ieșe ce intra prin 1. Prin 1 intra  $(\alpha + 8 + (-3) * 4) = \alpha - 4$ .
- (e): beq nu este instructiune de tip J => din (e) ieșe valoarea care intra din 0, adica (d).
- PC: in PC intra ce ieșe pe la (e).
- Mem[y]: nu s-a schimbat nimic la y.

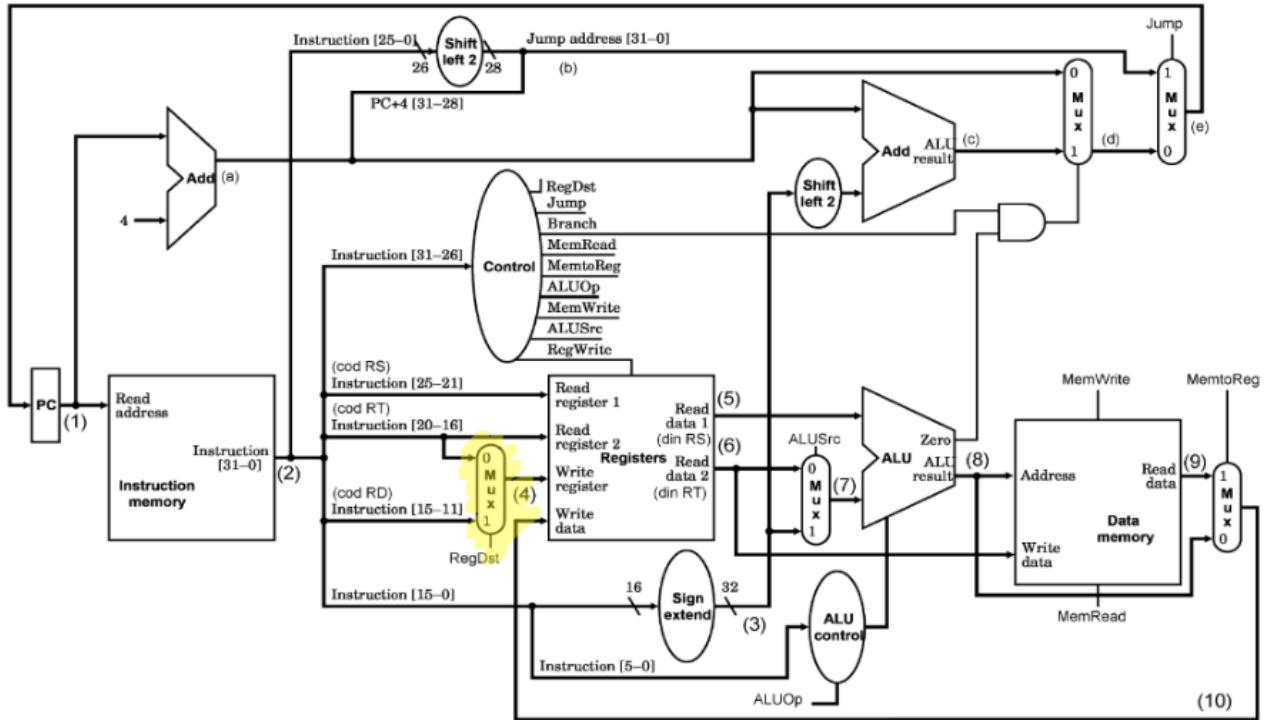
c) Adagati procesorului implementarea **subp rt, rs** care atribuie registrului rt diferența valorilor din registrii **rs** si **rt** (adica efectueaza  $rt := rs - rt$ ); se va folosi formatul I, cu op=0xFF si imm=0x0.

Pentru implementare, este suficienta adaugarea unei linii in taelul "Control". Completati aceasta linie:

Instruction [31 - 26]	RegDst	ALU Src	Mem To Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
—										

Incepem sa analizam:

- RegDst: Ce facea RegDst? Sa ne amintim din procesor:

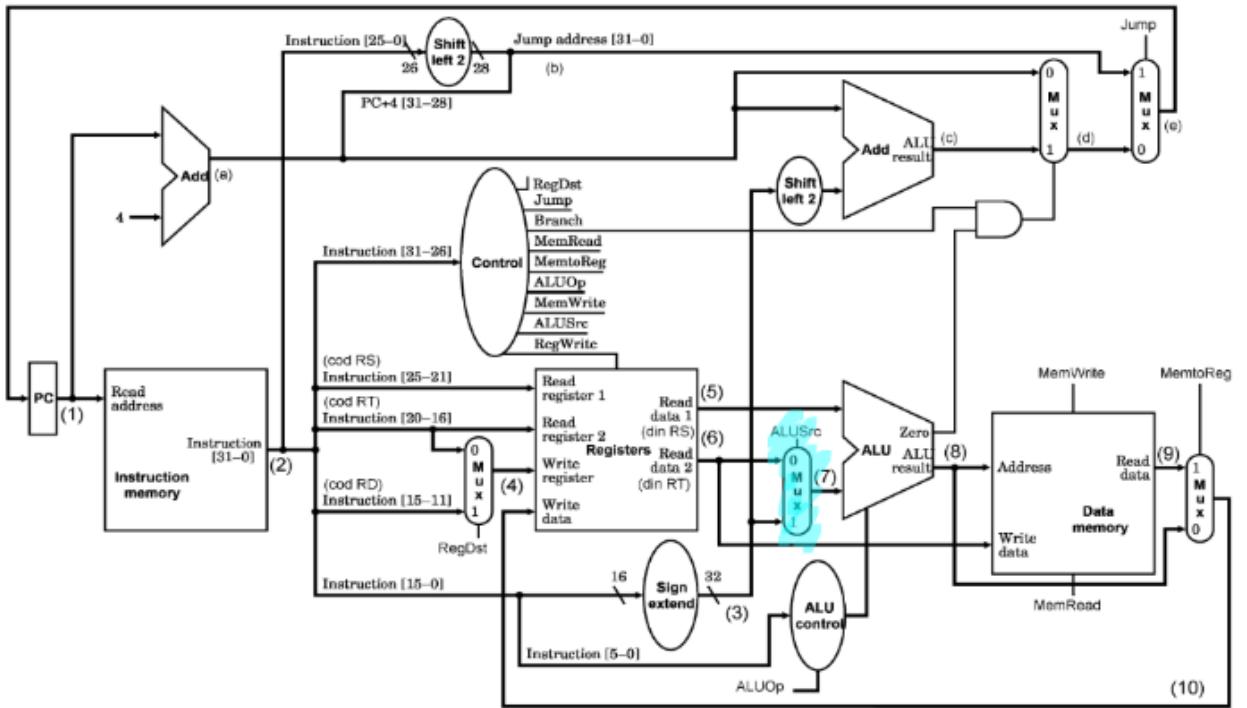


RegDst decide cine este registrul destinatie.

- Daca avem o instructiune de tip R, registrul destinatie este rd. => RegDst=1
- Daca avem o instructiune de tip I, cum ar fi li \$t0, 5, registrul destinatie este chiar rt.=> RegDst=0
- Cand avem instructiuni de genul sw sau beq, nu avem niciun registru destinatie. => RegDst=X (aka. niciuna dintre variante).

Acum sa ne gandim ce fel de instructiune trebuie sa construim noi. Vedem din prima ca registrul destinatie este rt. => RegDst=0

- ALUSrc: Ce facea ALUSrc?



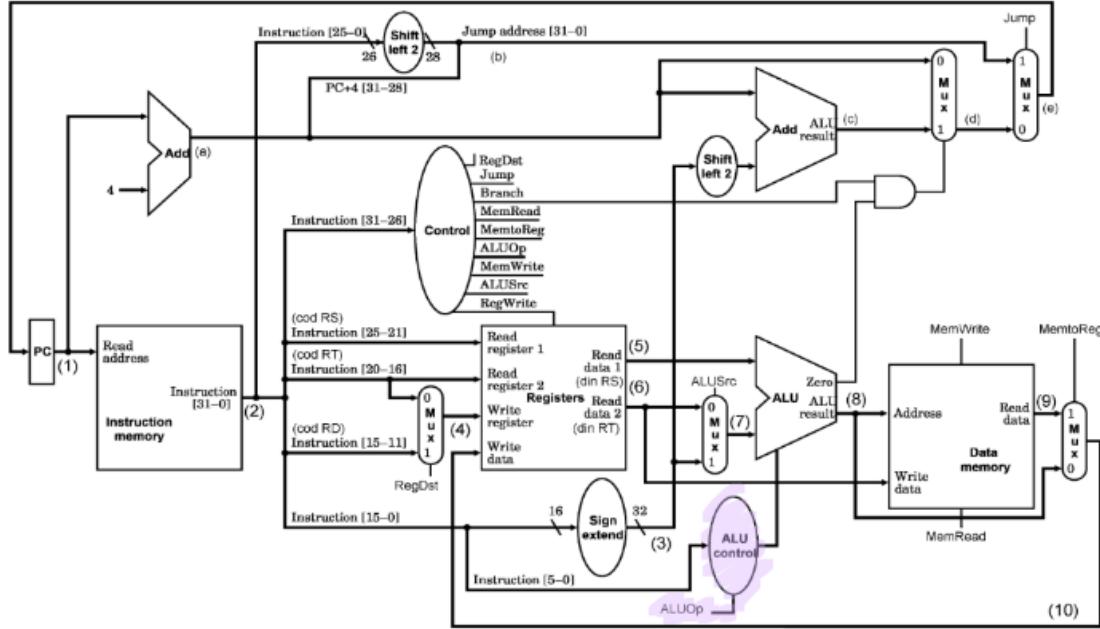
Face diferenta dintre instructiunile care fac operatii pe valori si cele care fac operatii pe adrese de memorie.

- Pentru instructiunile de tip R si cele de tip branch o sa trimita ca input catre Unitatea Aritmetica si Logica rs si rt pentru ca ele fac operatii pe valori. ( $ALUSrc=0$ )
- Pentru instructiunile de tip I o sa trimita ca input catre Unitatea Aritmetica si Logica valoarea lui rs si valoarea imediata (imm) pentru ca vrem sa facem operatii pe niste adrese de memorie. ( $ALUSrc=1$ )

Noi vrem sa facem diferenta dintre 2 alori, nu lucram cu adrese de memorie, deci alegem  $ALUSrc=0$ .

- MemToReg: Are voie instructiunea noastra sa citeasca din memorie si sa scrie intr-un registru? Nu, pentru ca nu are voie sa citeasca din memorie.  $MemToReg=0$ .
- RegWrite: Are voie instructiunea noastra sa scrie intr-un registru? Da.  $RegWrite=1$ .
- MemRead: Are voie instructiunea noastra sa citeasca din memorie? Nu.  $\Rightarrow MemRead=0$ .
- MemWrite: Are voie instructiunea noastra sa scrie din memorie? Nu.  $\Rightarrow MemWrite=0$ .
- Branch: E instructiunea noastra de tip branch? Nu.  $\Rightarrow Branch=0$ .
- Jump: E instructiunea noastra de tip branch? Nu.  $\Rightarrow Jump=0$ .
- ALUOp (o sa tratam si  $ALUOp1$  si  $ALUOp2$  intr-o singura sectiune pentru ca e aceeasi discutie pe ambele).

Vom face o analiza mai amanuntita pentru ALUOp. Stim ca ALUOp intra in ALUcontrol, care scoate ce fel de operatie va efectua Unitatea Aritmetica si Logica (ALU).



Dar ce mai exact inseamna aceste perechi de numere (ALUOp1, ALUOp2)?

- (ALUOp1=0, ALUOp2=0) : fa intotdeauna adunare (addi, lw, sw, etc.)
  - (ALUOp1=0, ALUOp2=1) : fa intotdeauna scadere (beq, bne, etc.)
  - (ALUOp1=1, ALUOp2=X) : fa o operatie determinata de campul Funct (din instructiunile de tip R)
- exemplu: La add vrem sa facem adunare, la sub scadere, etc.

Acum ca stim asta, putem sa alegem valorile pe care le vrem noi. Instructiunea noastra trebuie sa:

- sa faca diferența rs - rt, deci in Unitatea Aritmetica si Logica avem nevoie ori de ALUOp1=0, ALUOp2=1 (scadere intotdeauna), ori de ALUOp1=1, ALUOp2=X (fa ce operaie iti indica campul funct din reprezentarea in memorie a instructiunii). De ce nu putem alege ALUOp1=1, ALUOp2=X? Pentru ca instructiunea noastra **subp rt, rs** nu este de tip R => nu are in reprezentarea sa in memorie campul funct.
- sa puna rezultatul in rt (aici nu ne trebuie Unitatea Aritmetica si Logica, daca ne uitam in procesor, rezultatul scaderiiiese pe la (8), datorita lui MemToReg se transmite catre (10) si de acolo se duce direct in Write Data, adica valoarea care va fi incarcata in registrul destinatie).

=> Aleg ALUOp1=0, ALUOp2=1.

Instruction [31 - 26]	RegDst	ALU Src	Mem To Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
0xFF	0	0	0	1	0	0	0	0	0	1

### 3 Subiect 3 rezolvat la curs

La noi cel putin, al ultimul curs a rezolvat Draguliciun subiect 3, pe care l-a completat destul de interesant asa ca sa il facem si noi.

Bucata de cod utilizata pentru a completa tabelul este:

```
# x=2*y
.data
x: .space 4
y: .word 10
.text
main:
la $t0,x
lw $t1,4($t0)
add $t2,$t1,$t1
sw $t2,0($t0)
li $v0,10
syscall
```

Tabelul completat este:

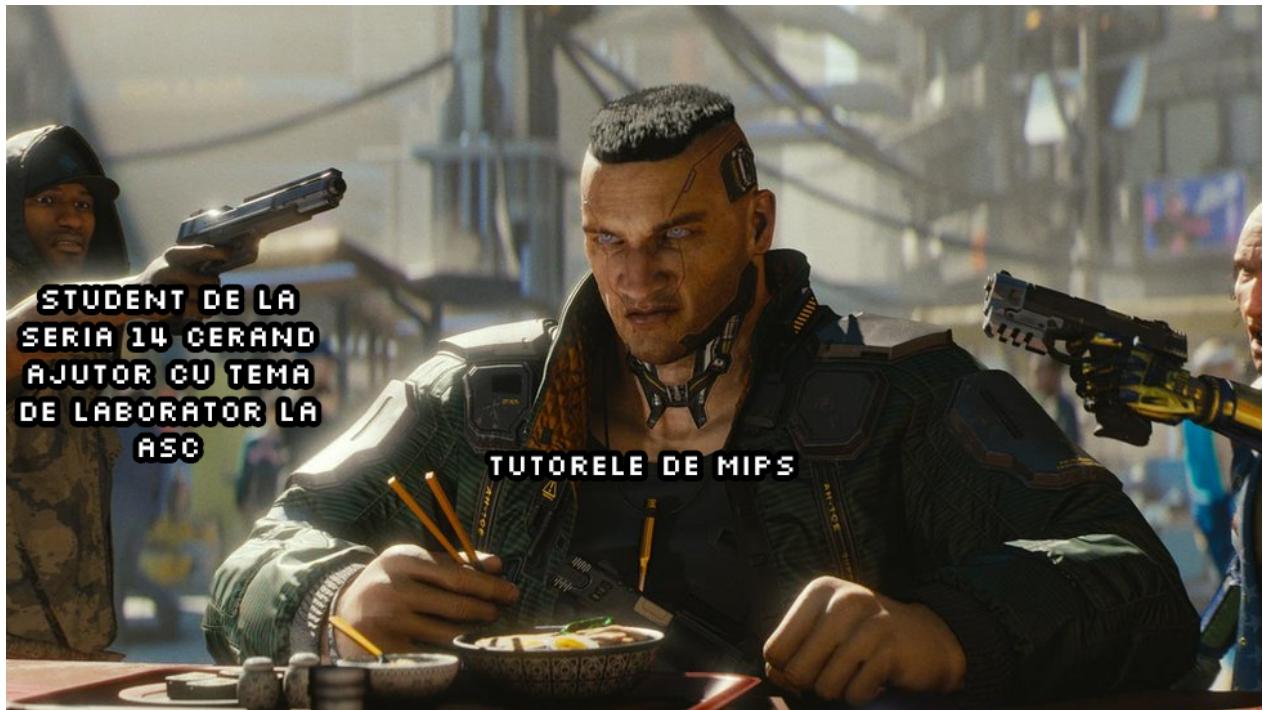
qc	op/RS/RT/RD/0/FCT op/RS/RT/immu	(3)	CTRL RegDst/ALUSrc/MemToReg/ RegWrite/ MemRead/ MemWrite/ Branch/Jump / ALUOp	ALU ctrl	(4)	(5)	(6)	(7)	ALU result (8)	ALU zero	(9)	(10)	(e)
lw	$\alpha$	23/8/9/h	h	010 +	g	p	?	h	p+h	?	A	A	$\alpha+h$
add	$\alpha+h$	0/9/9/A/0/20	5020	010 +	A	A	A	A	h	0	?	1h	$\alpha+8$
sw	$\alpha+b$	2B/8/A/0	0	010 +	?	p	1h	0	p	?	?	?	$\alpha+c$

Observatii:

- Valorile sunt scrise direct in hexa.
- A adaugat o coloana in plus **op/RS/RT/RD/0/FCT** (pentru instructiuni de tip R) si **op/RS/RT/immu** (pentru instructiuni de tip I). Fiecare variabila este scrisa in hexa, deci acel 23 de la lw inseamna  $2*16+3=35$ , care este fix opcode-ul de la lw: 100011)
- A adaugat o alta coloana: CTRL, care include toate variabilele din Unitatea de Control: RegDst, ALUSrc, MemToReg, RegWrite, MemRead, MemWrite, Branch, Jump, ALUOp.

# LUMEA DACA FIECARE PROGRAMATOR CODA IN MIPS





## References

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