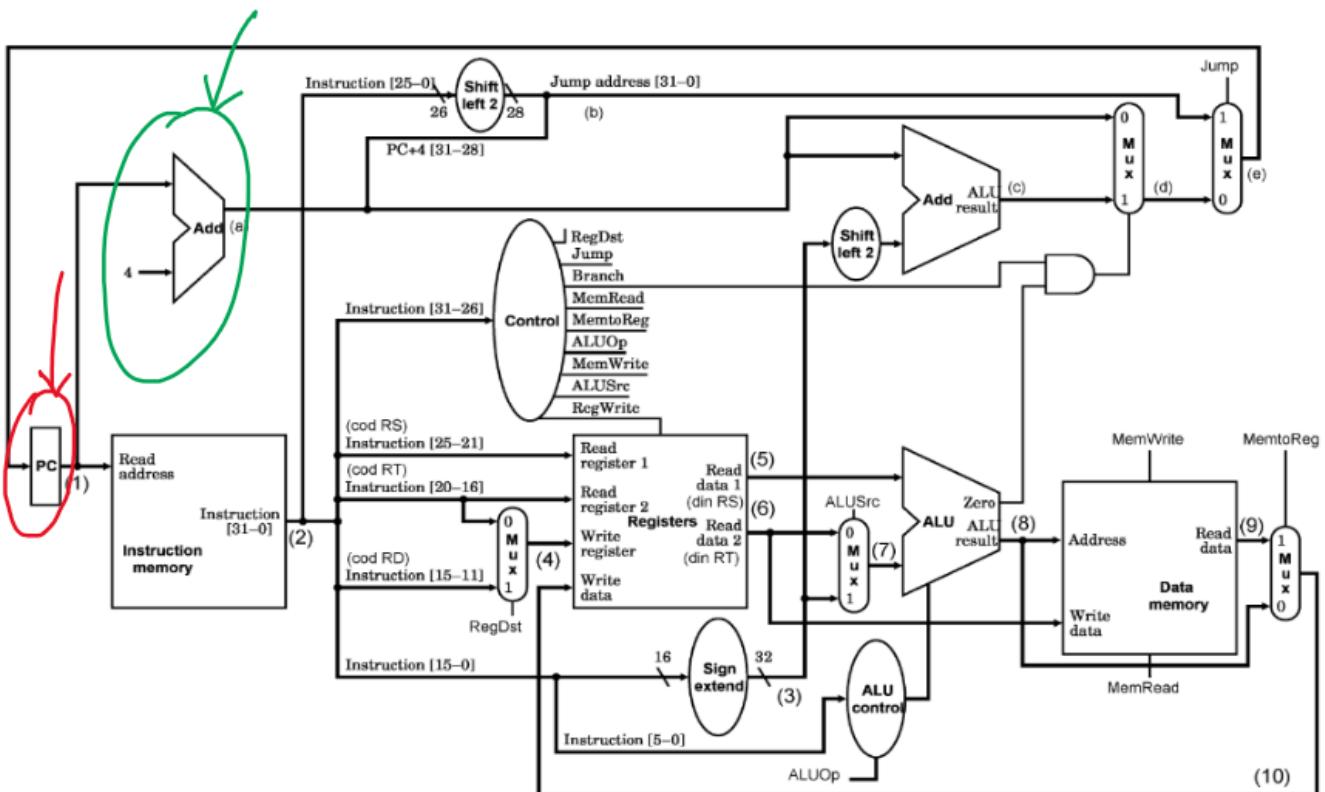


### ALU Control (slide 7.36)

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

#### 2.2.1 PC

Acum ca avem asta in minte, sa o luam cu prima chestie de la stanga la dreapta: PC.

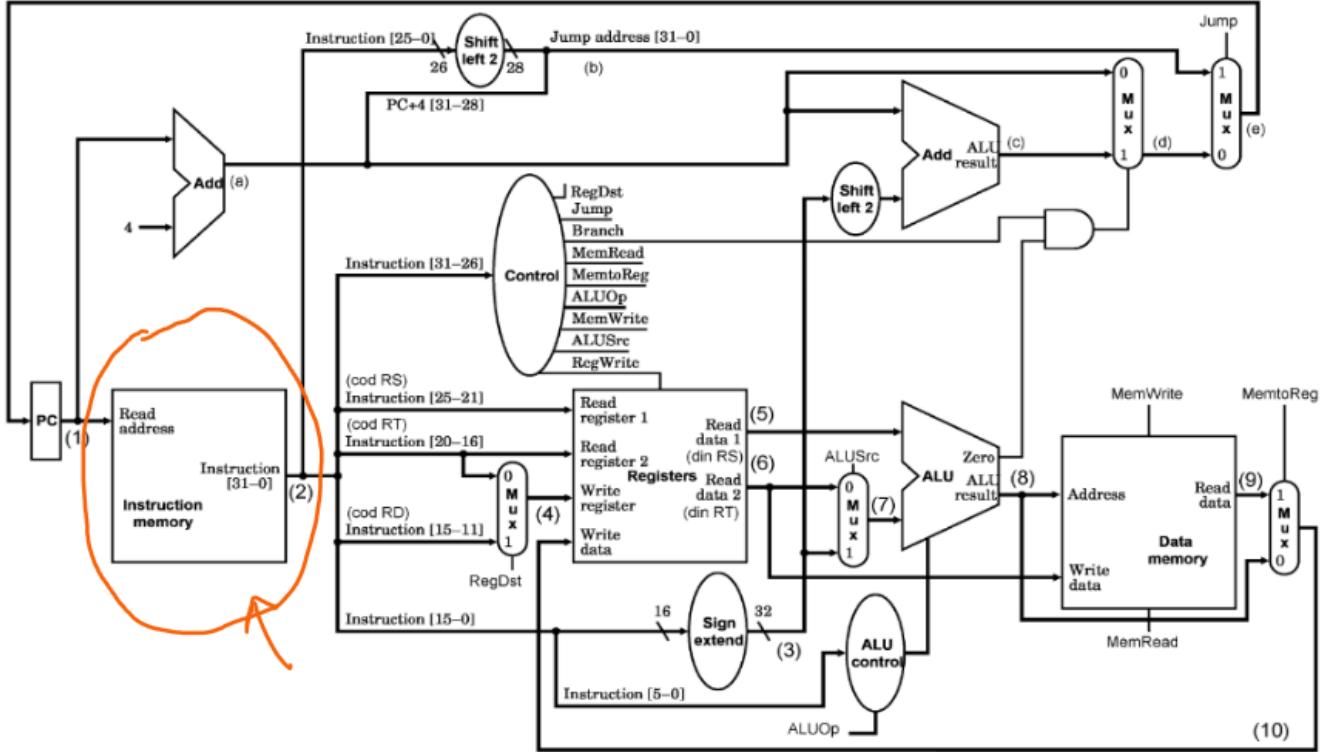


Ne putem gandi la PC ca la un pointer catre urmatoarea instructiune. De fiecare data cand se executa o instructiune, sa "incrementeaza" acest "pointer" pentru a arata catre urmatoarea instructiune. Incrementarea se face cu 4. De ce 4? Reprezentarea instructiunilor MIPS in calculator are 32 biti. Un byte are 8 biti.  $\Rightarrow$  Reprezentarea instructiunilor MIPS in calculator are 4 bytes.

Asadar, in schema, PC este cel incercuit cu rosu, iar incrementarea printr-un adder(sumator) este incercuita cu verde.

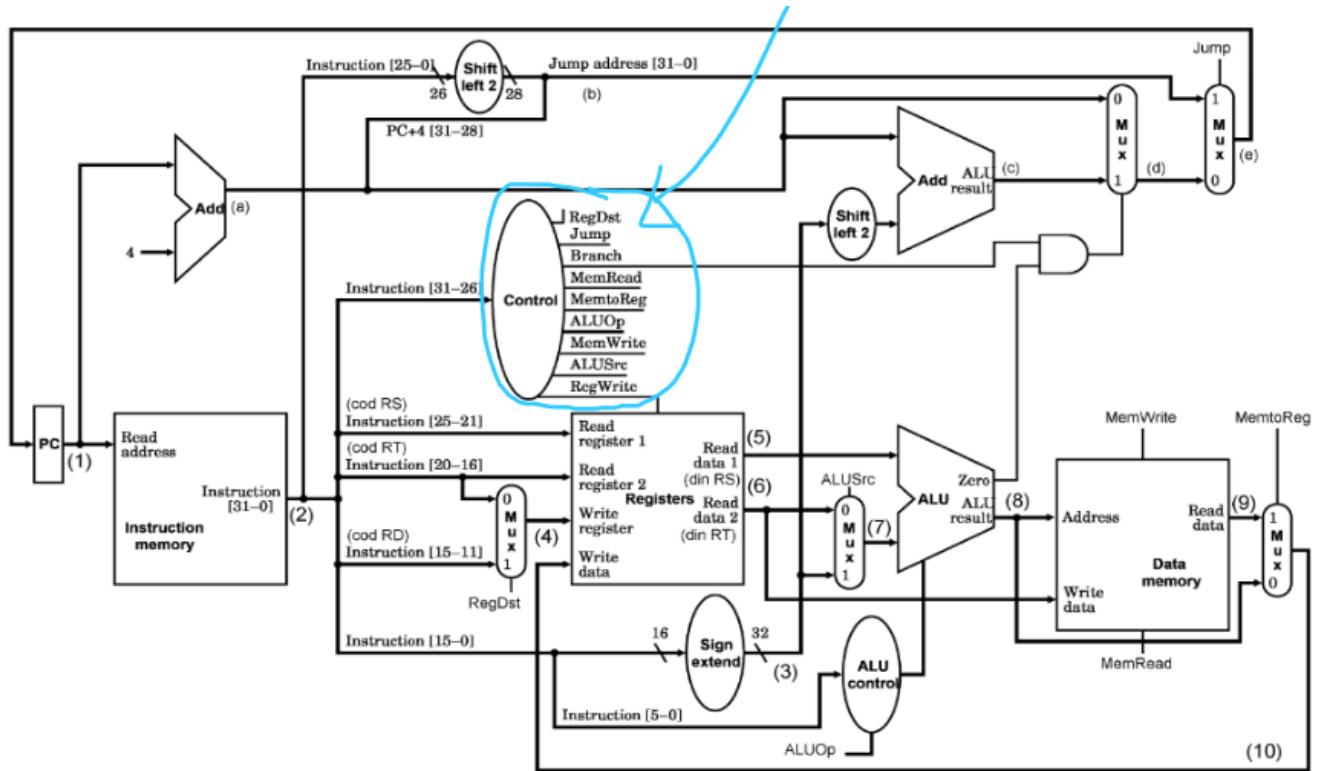
**Observatie!** In timpul executarii instructiunii, PC "arata" catre urmatoarea instructiune. De aceea, la 2.1 Reprezentarea interna a instructiunilor procesorului MIPS, trebuie sa adaugam acel -1.

## 2.2.2 Citirea instructiunii din memorie



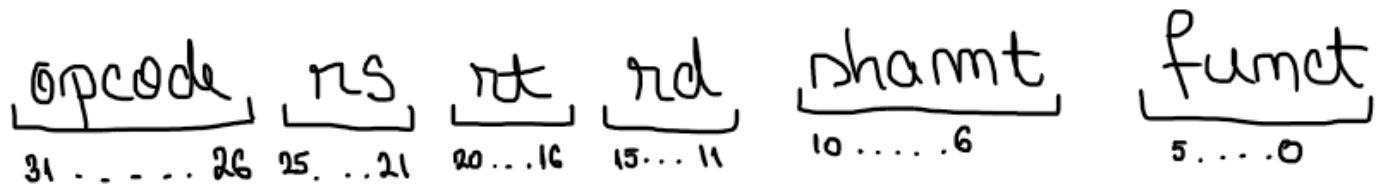
Citirea instructiunii din memorie este incercuita cu portocaliu. Vedem ca avem un patrat care se numeste chiar "Instruction memory" in care intra "Read address" (deci PC ii spune acestui bloc de la ce adresa sa citeasca instructiunea). Blocul are ca output o instructiune, pe 32 de biti.

### 2.2.3 Identificarea tipului de instructiune



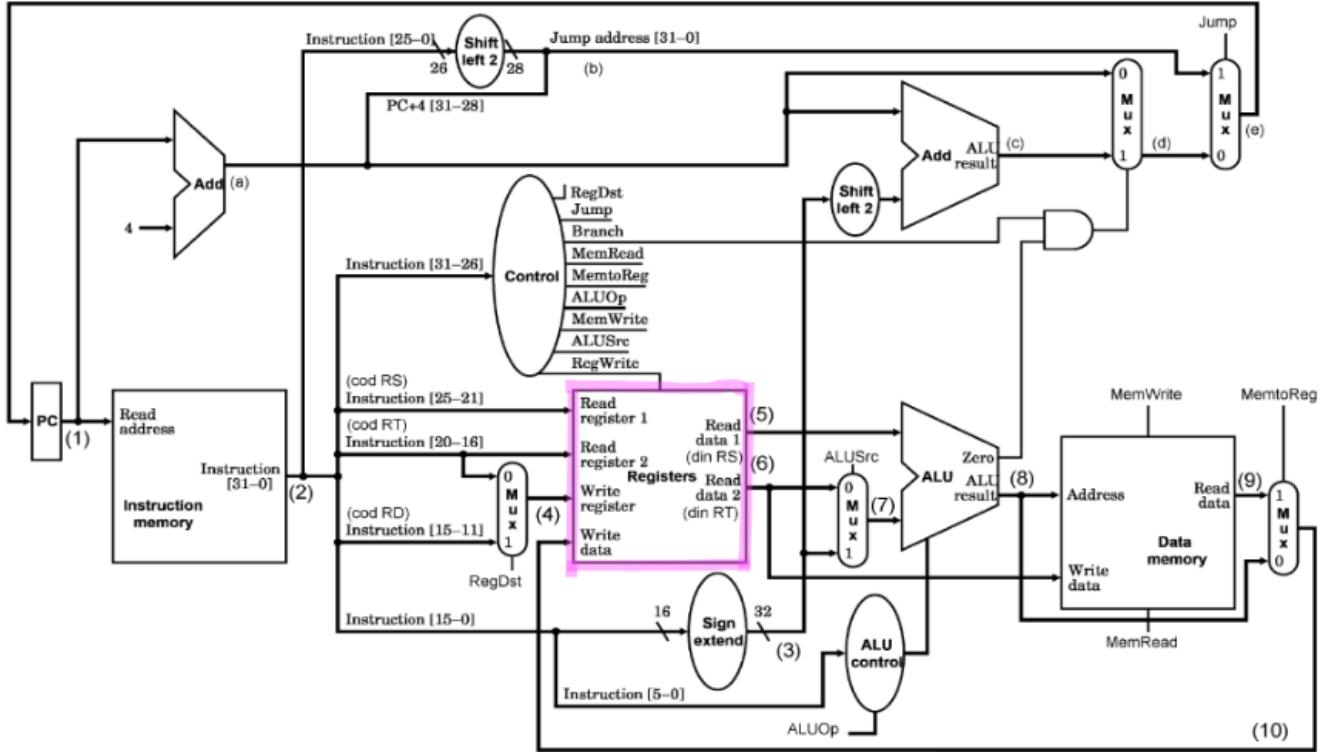
Identificarea tipului de instructiune se face prin structura incircuita cu albastru. Aceasta se numeste Control Unit. Ce parte din cei 32 de biti ai unei instructiuni ne spune noua ce fel de instructiune avem? Opcode. Acesta e comun tuturor tipurilor de instructiuni (R, I, J) si se afla mereu in primii 6 biti.

Observam ca in Control Unit intra bitii de la 31 la 26. Dar noi am spus ca opcode este mereu in primii 6 biti. Ne dam seama deci, ca asta este doar o chetiere de notatie. In memorie, fiecare bit este indexat de la 31 la 0.(putem sa ne gandim ca ultimul bit este cel mai nesemnificativ bit si de aceea are index 0).



Prin Opcode identificam ce fel de instructiune avem, iar prin datele pe care le stim noi despre instructiuni (pe care le vom identifica din tabelele de ajutor) vom afla valorile tuturor acelor variabile: RegDst, Jump, Branch, MemRead, MemtoReg, ALUOp, MemWrite, ALUSrc, RegWrite.

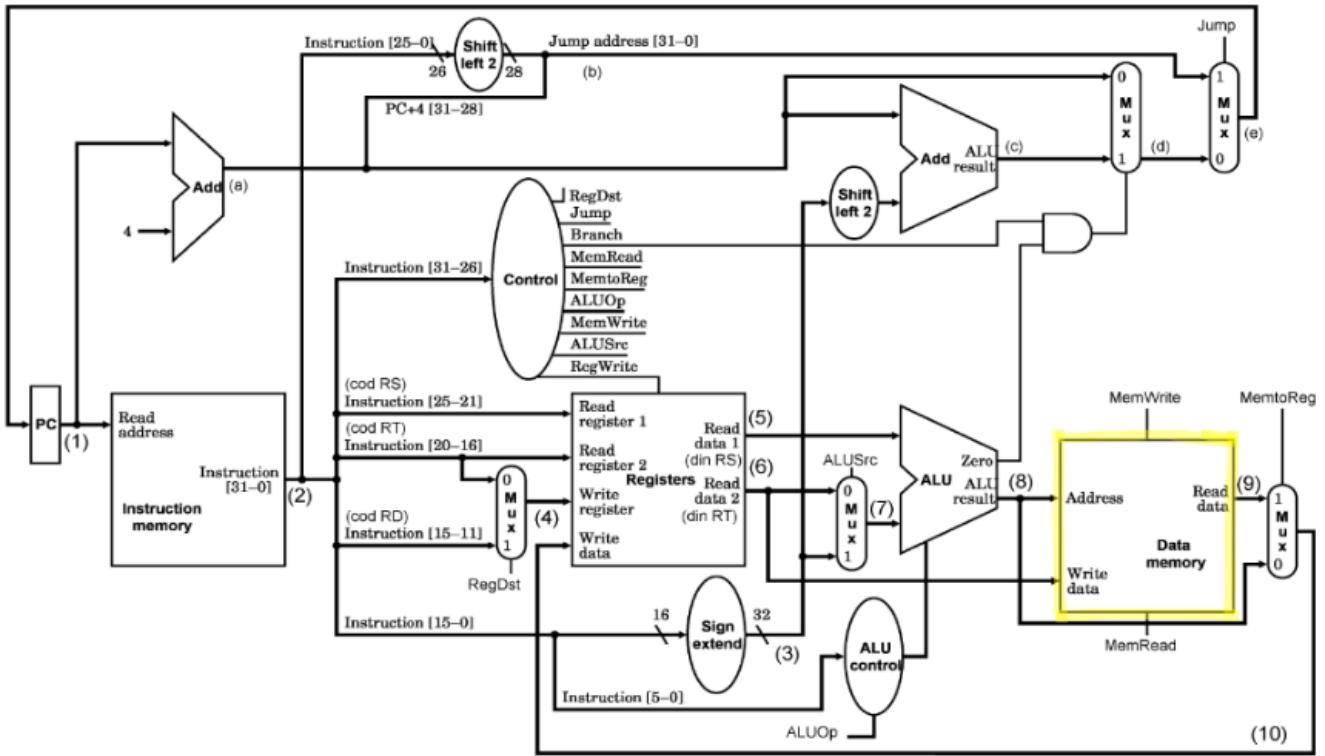
## 2.2.4 Citirea registrilor



Trecem la citirea registrilor.

- RegWrite: care, după cum vedem din tabel, are valoarea 1 pentru instrucțiunile de tip R( $\$d=\$s+\$t$ ) și pentru lw (evident, se "scrive" în registrul  $\$t$ )
- Read register 1: citește care este registrul  $\$s$  (poate să fie  $\$t0$ ,  $\$v2$ , etc.).
- Read register 2: citește care este registrul  $\$t$  (poate să fie  $\$t0$ ,  $\$v2$ , etc.).
- Write register: Aici se complica puțin lucrurile. Avem mai multe cazuri, în funcție de tip:
  - Pentru instrucțiuni de tip R, registrul  $\$d$  este cel în care se scrie (ex: pt add  $\$d=\$s+\$t$ ) de aceea el se numește Write register. Ce intra în Write register este determinat de un multiplexor. Rezultatul multiplexorului este determinat de RegDst. Dacă ne uitam în tabel, RegDst este 1 doar pentru instrucțiunile de tip R (evident, pentru că celelalte tipuri de instrucțiuni nu au un registru  $\$d$ ). => Din multiplexor o să iasa codul lui  $\$d$ , și acolo va fi scris rezultatul instrucțiunii.
  - Pentru instrucțiuni de tip I: nu vom vorbi despre toate instrucțiunile de tip I pentru că unele dintre ele rulează 2 instrucțiuni în loc de una. Pentru mai multe detalii vezi Exemplul 10. În tabele avem menționate că instrucțiuni de tip I doar: lw, sw și beq. Pentru lw și sw, rezultatul este stocat în  $\$t$ .
- Read data 1 ca output: citește **valoarea** din registrul  $\$s$ .
- Read data 2 ca output: citește **valoarea** din registrul  $\$t$ .
- Write data: valoarea care trebuie scrisă în Write Register.

## 2.2.5 Scrierea in memorie / citirea din memorie

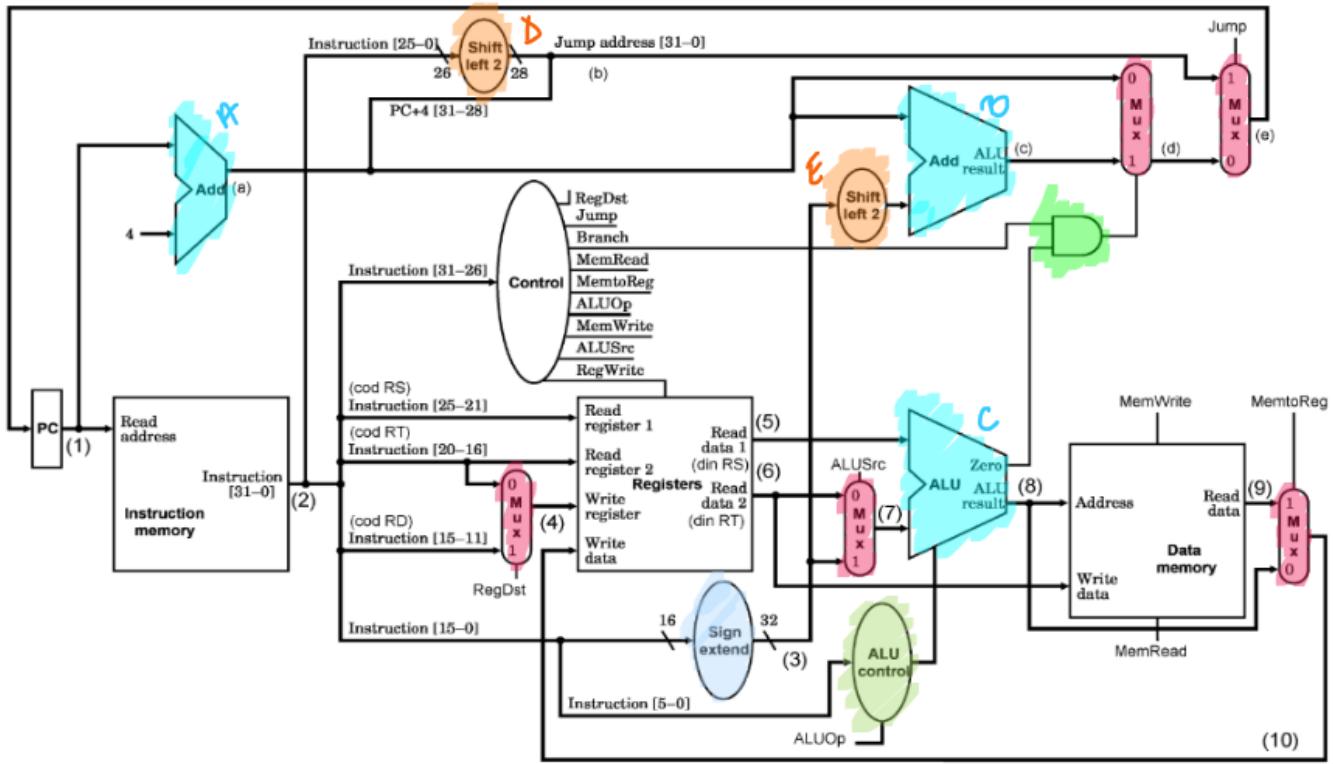


Pentru citirea/scrierea in memorie (aka lw si sw).

## 2.2.6 Executarea instructiunii

Inainte sa trecem la exemple, sa identificam restul de structuri din procesor:

- ALU control
- multiplexor
- shiftare pe biti la stanga
- extindere de semn
- unitate aritmetica si logica
- portata AMD



- ALU control: Primeste codul ALUOp si il transforma intr-o functie (operatie):

<u>ALU control input</u>	<u>Function</u>
000	AND
001	OR
010	add
110	subtract
111	set on less than

- Multiplexor elementar: primeste 2 input-uri X si Y notate cu 0 si 1 si in functie de valoarea de adevar input-ului de decizie Z scoate valoarea lui X sau a lui Y (pentru mai multe detalii vezi tutoriat 5)
- Shiftare pe biti la stanga cu 2: exemplu - 1010010 shiftat la stanga cu 2 este 10100100. Observam ca

initial avea 7 biti, acum are 9. De aceea, putem observa si la D-ul de pe desen(cu portocaliu), ca intra 26 de biti si ies 28.

- Extindere de semn: Din cardinalitatea care intra si iese din Sign extend observam ca: intra un cod pe 16 biti si iese unul pe 32 de biti.

Eu aveam un numar pe 16 biti si vreau sa il scriu pe 32. Cum fac? Ii extind bitul de semn in fata lui. Care era bitul de semn? Primul.

Deci pentru 10010001 pe 8 biti, extinderea sa la 16 biti este 111111110010001.

- Unitatea aritmetica si logica (ALU): este un circuit care aplica unor operanzi numere intregi pe n biti o operatie aritmetica sau logica selectata printr-un cod numeric.

Este exact ce vedem ca se intampla la C. Avem 2 input-uri, unul de la (5) si unul de la (7). Pe ele se aplica o operatie selectata printr-un cod numeric. Care cod numeric? ALUOp, care intra in ALU control. ALU control transforma acest cod intr-o functie, pe care o trimit ca input "de decizie" catre Unitatea aritmetica si logica.

exemplu: Vedem ca in tabel add are ALUOp=010. Functia in care il transforma ALU control este adunarea. Deci Unitatea aritmetica si logica va scoate ca rezultat (5)+(7).

Dar si B (cu albastru deschis) este este un ALU. Doar ca acum nu mai intra un input de decizie in el, ci scrie deja pe el Add. S-a decis deja ca acest ALU va face o adunare a celor 2 input-uri.

- Poarta AND: daca nu suntem familiarizati, vezi tutoriat 4.

### 2.2.7 Unitatea de control

Activity	Signal	Purpose
PC Update	Branch	Combined with a condition test boolean to enable loading the branch target address into the PC.
	Jump	Enables loading the jump target address into the PC (only appears in Figure 4.24 in Patterson and Hennessey).
Source Operand Fetch	ALUSrc	Selects the second source operand for the ALU (rt or sign-extended immediate field in Patterson and Hennessey).
ALU Operation	ALUOp	Either specifies the ALU operation to be performed or specifies that the operation should be determined from the function bits.
Memory Access	MemRead	Enables a memory read for load instructions.
	MemWrite	Enables a memory write for store instructions.
Register Write	RegWrite	Enables a write to one of the registers.
	RegDst	Determines how the destination register is specified (rt or rd in Patterson and Hennessey).
	MemtoReg	Determines where the value to be written comes from (ALU result or memory in Patterson and Hennessey).

Acum ca stim ce fac toate aceste structuri, trecem la exemple:

#### Exemplul 8 [RESTANTA SEPTEMBRIE 2020]

Considerăm implementarea procesorului MIPS cu 1 ciclu / instrucțiune (vezi verso). Fie fragmentul de program:

```
.data          1a $t3, x
x: .word 5      et:
.text          lw $t4, 0($t3)
main:          sub $t2, $t2, $t4
              li $t2, 5           beq $t2, $0, et
```

b) Completă tabelul următor cu valorile obținute la prima executare a instrucțiunilor lw și beq din program; valorile se vor scrie hexa/formulă, iar dacă valoarea este necunoscută/nedefinită se va nota "?"; în coloanele PC și \$t2 se vor trece valorile noi, de la sfârșitul executării instrucțiunilor respective:

	1	4	5	7	8	ALU zero	10	(d)	(e)	Branch	Mem To Reg	ALU Op (2b)	ALU Ctrl (3b)	Reg Write	PC	\$t2	
Initial	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$\alpha$	5
lw	$\alpha$																
beq																	

Incepem sa completam tabelul in certinta, folosindu-ne de datele din tabelele de ajutor.

	1	4	5	7	8	ALU zero	10	(d)	(e)	Branch	Mem To Reg	ALU Op (2b)	ALU Ctrl (3b)	Reg Write	PC	\$t2	
Initial	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$\alpha$	5
lw	$\alpha$										0	1	00	010	1		
beq																	

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

	ALUOp		Camp functie						Operatie
	ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
lw/sw	0	0	X	X	X	X	X	010	(+)
beq	X	1	X	X	X	X	X	110	(-)
add	1	X	X	X	0	0	0	010	(+)
sub	1	X	X	X	0	1	0	110	(-)
and	1	X	X	X	0	1	0	000	(and)
or	1	X	X	X	0	0	1	001	(or)
slt	1	X	X	X	1	0	1	111	(slt)

Am colorat cu aceeasi culoare perechile (valoare completata in tabelul din cerinta, de unde am luat acea valoare).

Acum ca am completat campurile pe care le putem lua direct din tabelele de ajutor, incepem "executia". Identificam valorile registrilor.

Reamintim ca lw are forma: lw \$t, offset(\$s) si noi avem instructiunea lw \$t4, 0(\$t3)

- \$s=\$t3=11

- $\$t = \$t4 = 12$

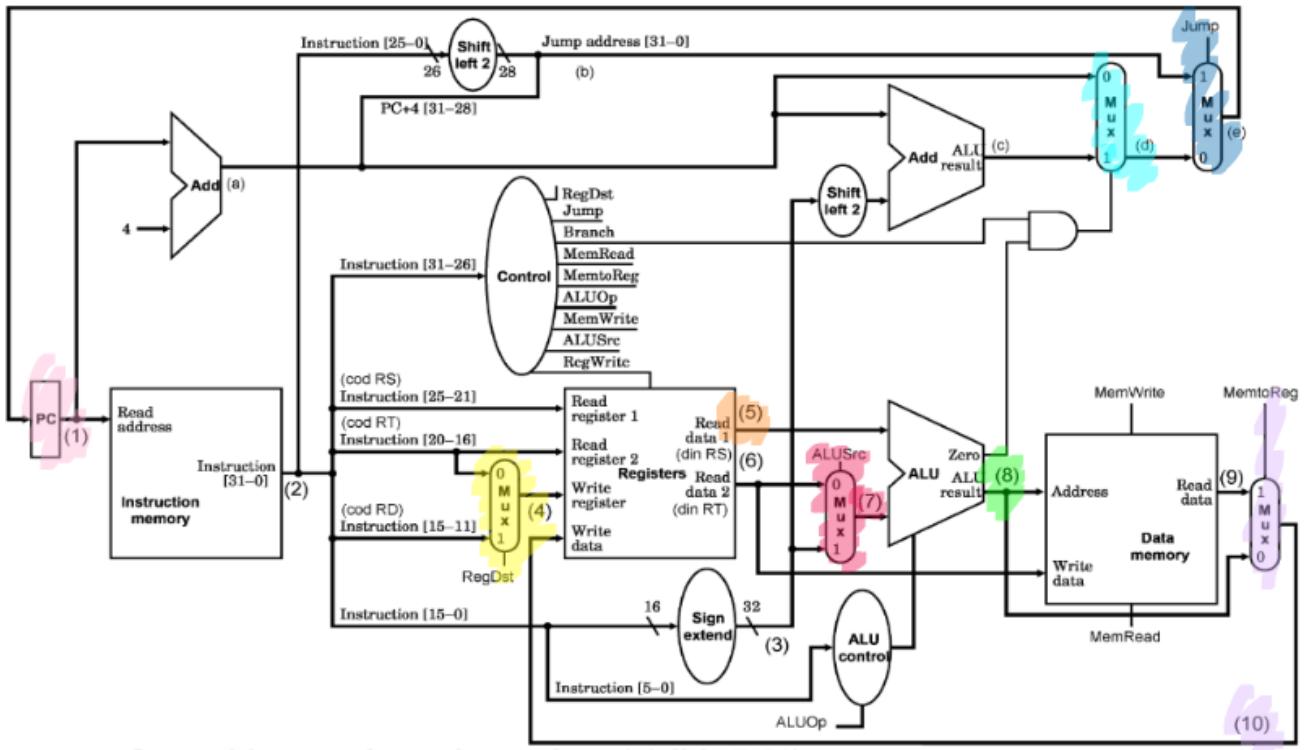
- offset=0

Mai departe, identificam ce valori se află în acești registri.

Aveți un  $x$  care inițial este 5,  $\$t2$  primește valoarea 5.  $\$t3$  primește ca valoarea adresa lui  $x$ .  $\$t4$  primește valoarea de la "punctul 0" din  $\$t3$ , adică valoarea lui  $x \Rightarrow$

- valoarea din  $\$s$  (adică din  $\$t3$ ) este adresa de memorie a lui  $x$  pe care o notăm cu  $\beta$
- valoarea din  $\$t$  (adică din  $\$t4$ ) este 5

	1	4	5	7	8	ALU zero	10	(d)	(e)	Branch	Mem To Reg	ALU Op (2b)	ALU Ctrl (3b)	Reg Write	PC	\$t2
Initial	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$\alpha$	5
lw	$\alpha$	12	β	0	β	?	5	$\alpha + \beta$	$\alpha + \beta$	0	1	00	010	1	$\alpha + \beta$	5
beq																



Să parcurgem ce am completat și de ce:

- (1): era deja completată cu  $\alpha$ . 1 este de fapt valoarea din PC care indică către instrucțiunea curentă.  
Nu stim la ce adresa de memorie se află instrucțiunea curentă, astăzi este notată cu  $\alpha$ .

- (4): Valoarea din 4iese dintr-un multiplexor. Input-ul de decizie este RegDst. Ne uitam in tabelele de ajutor sa vedem care este valoarea lui RegDst pentru lw. Observam ca valoarea este 0.  $\Rightarrow$  (4)=ce intra in dreptul lui 0 in multiplexor. Deasupra la "Instruction [20-16]" scrie "(cod RT)" care ne indica ca valoarea de la Instruction [20-16] este chiar codul lui \$t, adica 12.  $\Rightarrow$  valoarea lui (4) este 12.
- (5): Ne amintim de la sectiunea Identificarea tipului de instructiune ca Read data 1 ne citeste **valoarea** stocata in registrul \$s (nu codul registrului, valoarea din el). Asa cum am vazut mai sus, valoarea din \$s este adresa de memorie a lui x pe care nu avem de unde sa o stim, asa ca o notam cu  $\beta$ .
- (7): Avem din nou un multiplexor care depinde de data aceasta de ALUSrc. Ne uitam in tabelele de ajutor care este valoarea lui ALUSrc pentru lw. ALUSrc=1  $\Rightarrow$  Iese din multiplexor de a intrat pe la 1.  
Ce a intrat pe la 1? [Instruction 15-0], care este chiar offset in instructiunile de tip I, caruia i s-a aplicat un sign extend. offset avea valoarea 0 (pe 16 biti) $\Rightarrow$  in urma lui sign extend avem valoarea 0 pe 32 de biti.
- (8): Avem ALU (Arithmetic and Logical Unit). Ce operatie efectueaza? Trebuie sa ne uitam la ce iese din ALUcontrol, adica ne uitam in tabela de ajutor la ALUCtrl pentru lw. Vedem ca operatia este adunare. Deci se face adunarea intre valorile din (5) si (7). Deci rezultatul este  $\beta$ .
- ALU zero este o "variabila" care se foloseste in cazul instructiunilor de tip branch. Atunci cand am o instructiune de tip branch, valoarea lui ALUzero imi spune daca conditia din branch este indeplinita sau nu. Cum Instructiunea noastra momentan este lw, valoarea lui ALUzero este ? (nu am nicio conditie pusa, deci nu am cum sa stiu valoarea sa de adevar).
- (10): Ce iese din multiplexorul a carui valoare este determinata de MemToReg. Pentru lw MemToReg are valoarea 1  $\Rightarrow$  Iese ce intra la Read data. Ce intra la Read data? Daca ne amintim ca lw citeste ceva de la o adresa de memorie si pune acea valoarea intr-un registru, lucrurile sunt destul de clare. Deci in (10) vom avea valoarea citita la 0(\$t3), adica valoarea lui x, adica 5.
- (d): Rezultatul este determinat de o poarta AND intre Branch si ALUzero. Branch stim sigur ca are valoarea 0 pentru lw, deci si rezultatul portii va fi 0. Deci in (d) avem valoarea care intra in multiplexor la 0, care este  $\alpha$  la care se adauga un 4 prin adder.  $\Rightarrow$  (d)= $\alpha+4$
- (e): Un multiplexor determinat de valoarea lui Jump. Jump pentru lw e 0  $\Rightarrow$  iese ce intra din (d)  $\Rightarrow$   $\alpha + 4$
- PC: Observam ca im PC intra fix ce a iesit din (e)  $\Rightarrow$  PC= $\alpha+4$
- \$t2: Trebuie sa ne intoarcem putin la codul de MIPS si sa ne amintim ce se intampla acolo. \$t2 primește valoarea 5. Apoi avem peratia de lw pe \$t3 si \$t4, deci nu au treaba cu \$t2. Deci valoarea lui \$t2 ramane aceeasi.

Incepem sa completam si randul pentru beq, incepand cu valorile pe care le luam direct din tabelele de ajutor.

	1	4	5	7	8	ALU zero	10	(d)	(e)	Branch	Mem To Reg	ALU Op (2b)	ALU Ctrl (3b)	Reg Write	PC	St2
Initial	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$\alpha$	5
lw	$\alpha$									0	1	00 010	1			
beq										1	X	X 1 110	0			

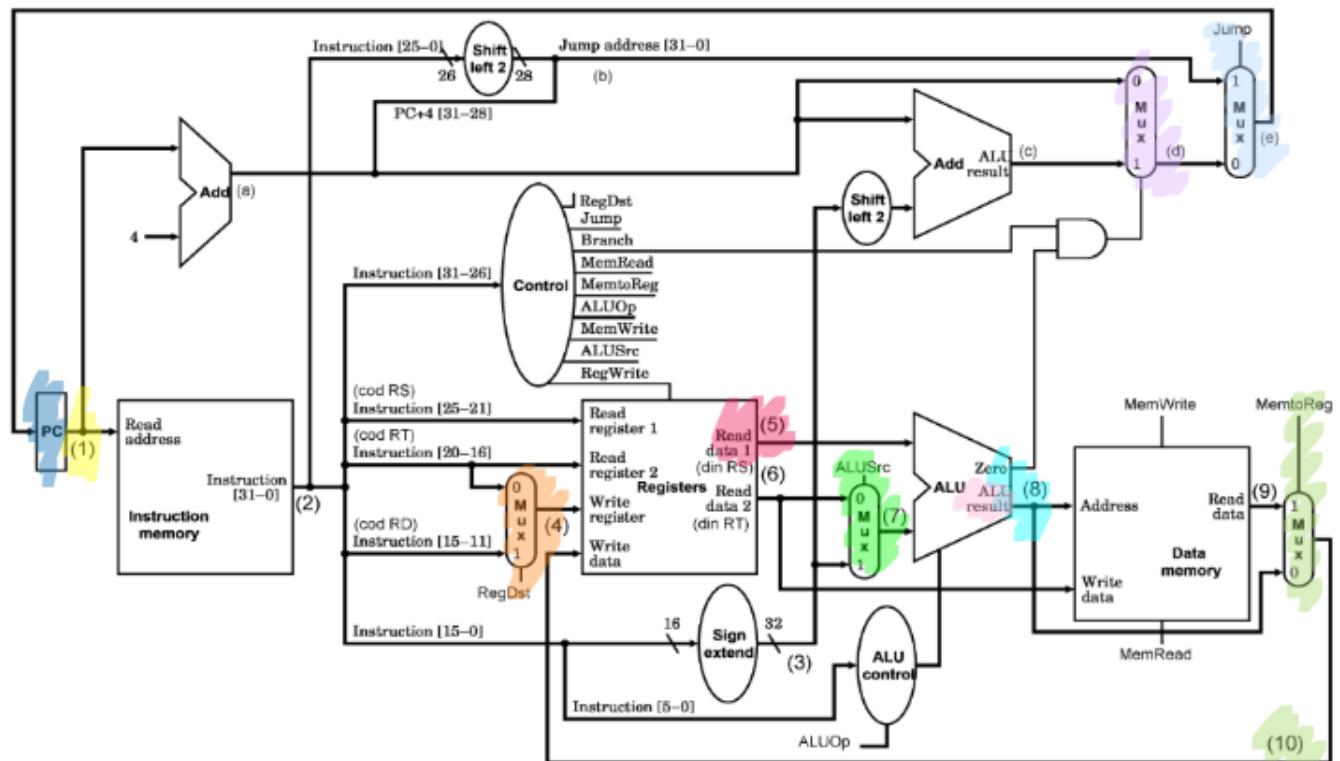
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp	ALUCtrl
R-format	1	0	0	1	0	0	0	010	000
lw	0	1	1	1	1	0	0	010	000
sw	X	1	X	0	0	1	0	010	000
beq	X	0	X	0	0	0	1	010	000

### ALU Control (slide 7.36)

lw/sw beq add sub and 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 (+)
beq	X	1	X	X	X	X	X	X	110 (-)
add	1	X	X	X	0	0	0	0	010 (+)
sub	1	X	X	X	0	0	1	0	110 (-)
and	1	X	X	X	0	1	0	0	000 (and)
or	1	X	X	X	0	1	0	1	001 (or)
R-format	1	X	X	X	1	0	1	0	111 (slt)

Continuam cu restul valorilor:

	1	4	5	7	8	ALU zero	10	(d)	(e)	Branch	Mem To Reg	ALU Op (2b)	ALU Ctrl (3b)	Reg Write	PC	\$t2
Initial	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$\alpha$	5
lw	$\alpha$	12	12	0	12	?	5	alpha+4	0	1	00	010	1	$\alpha+4$	5	
beq	$\alpha+8$	?	0	0	0	1	?	$\alpha$	$\alpha$	1	X	X1110	0	$\alpha$	0	



Să parcurgem ce am completat și de ce:

- (1): Înainte lw PC avea valoarea  $\alpha$ . Înainte de sub a avut valoarea  $\alpha+4$ . Înainte de beq va avea valoarea  $\alpha+8$ .
- (4): RegDst are valoarea X. Deci nu stim ce ieșe pe la 4. Si nici nu avem nevoie sa stim. Pentru beq, nu avem registru destinatie, deci nu ne intereseaza ce intra pe la 4.
- (5): Ne amintim de la sectiunea Identificarea tipului de instructiune ca Read data 1 ne citeste **valoarea** stocata in registrul \$s (nu codul registrului, valoarea din el). Care e valoarea din \$s?

Considerăm implementarea procesorului MIPS cu 1 ciclu / instrucțiune (vezi verso). Fie fragmentul de program:

```

.data
    .word 5
.text
main:
    li $t2, 5
    la $t3, x
    et:
    lw $t4, 0($t3)
    sub $t2, $t2, $t4
    beq $t2, $0, et

```

Să beq are forma: beq \$s, \$t, imm. La sub am scăzut din \$t2 5, deci \$t2 are acum valoarea 0. => (5)=0

- (7): Avem din nou un multiplexor care depinde de data aceasta de ALUSrc. Ne uitam in tabelele de ajutor care este valoarea lui ALUSrc pentru beq. ALUSrc=0 => Iese din multiplexor de a intrat pe la 0.  
Ce a intrat pe la 0? Valoarea ui \$t, care era tot 0 => (7)=0
- (8): Avem ALU (Arithmetic and Logical Unit). Ce operatie efectueaza? Trebuie sa ne uitam la ceiese din ALUcontrol, adica ne uitam in tabela de ajutor la ALUCtrl pentru beq. Vedem ca operatia este scadere. Deci se face scaderea intre valorile din (5) si (7). Deci rezultatul este 0.
- Acum chiar avem o instructiune de tip branch, deci ALUzero are sens. Instructiunea noastra este beq, adica branch if equal. Deci ALUzero este 1 daca valorile din (5) si (7) sunt egale, si 0 daca nu sunt egale. In procesor, testam egalitatea prin scadere: doua numere sunt egale daca rezultatul scaderii lor este zero. Asadar, pentru beq:  
ALUzero=1 daca (8)=0  
ALUzero=0 daca (8) ≠ 0 In cazul nostru valoarea lui (8) chiar este 0 => ALUzero=1

**Atentie!** Functia de mai sus nu este valabila pentru orice instructiune de tip branch. De exemplu, pentru bne (branch if not equal, adica opusul lui beq) functia va deveni:

ALUzero=0 daca (8)=0

ALUzero=1 daca (8) ≠ 0

Un alt exemplu ar fi pentru blez( branch if less than or equal to 0 ). Aceasta instructiune are forma blez \$s, offset. Pentru aceasta instructiune, functia va deveni:

ALUzero=0 daca (5)>0

ALUzero=1 daca (5) ≤ 0. De ce (5)? Pentru ca in (5) intra valoarea din rs, iar instructiunea noastra blez testeaza daca valoarea din rs este mai mica sau egala cu 0.

- (10): Ceiese din multiplexorul a carui valoare este determinata de MemToReg. Pentru beq MemToReg are valoarea X => Nu stim ceiese pe la (10).
- (d): Rezultatul este determinat de o poarta AND intre Branch si ALUzero. Branch stim sigur ca are valoarea 1 pentru beq, si ALUzero=1. Deci in (d) avem valoarea care intra in multiplexor la 1. Ce intra in 1? Ceiese din acel ALU cu operatia de adunare. Pe sus intra valoarea lui PC+4, adica  $(\alpha + 8) + 4 = \alpha + 12$ . Pe jos intra imm shiftat cu 2 biti la stanga. imm=-3. Shiftarea la stanga cu 2 inseamna o inmultire cu 4. => Pe sus intra = -12. => Ceiese din ALU este  $(\alpha + 12) - 12 = \alpha => (\alpha) = \alpha$
- (e): Un multiplexor determinat de valoarea lui Jump. Jump pentru beq e 0 => ieiese ce intra din (d) =>  $\alpha$
- PC: Observam ca im PC intra fix ce a iesit din (e) => PC= $\alpha$
- \$t2: Trebuie sa ne intoarcem putin la codul de MIPS si sa ne amintim ce se intampla acolo. \$t2 primește valoarea 5. Apoi avem operatia de lw pe \$t3 si \$t4, deci \$t4 este 5. Apoi avem un sub in care \$t2=0. In bew nu se schimba valoarea lui \$t2, doar o compara. Deci valoarea lui \$t2 ramane 0.

Mai departe, ne amintim ca cerinta era ca valorile sa fie scrise in tabel in hexa. Deci tabelul completat in hexa este:

	1	4	5	7	8	ALU zero	10	(d)	(e)	Branch	Mem To Reg	ALU Op (2b)	ALU Ctrl (3b)	Reg Write	PC	\$t2
Initial	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$\alpha$	5
lw	$\alpha$	C	p	0	p	?	5	$\alpha\#$	$\alpha\#$	0	1	00	010	1	$\alpha\#$	5
beq	$\alpha\#$	?	0	0	0	1	?	$\alpha$	$\alpha$	1	X	x1	110	0	$\alpha$	0

Observam ca pentru ALUOp si ALUCtrl nu transformam valorile in hexa pentru ca scrie ca trebuie sa fie pe 2 biti, respectiv pe 3 biti (am verificat si asa a completat si Dragulici la curs).

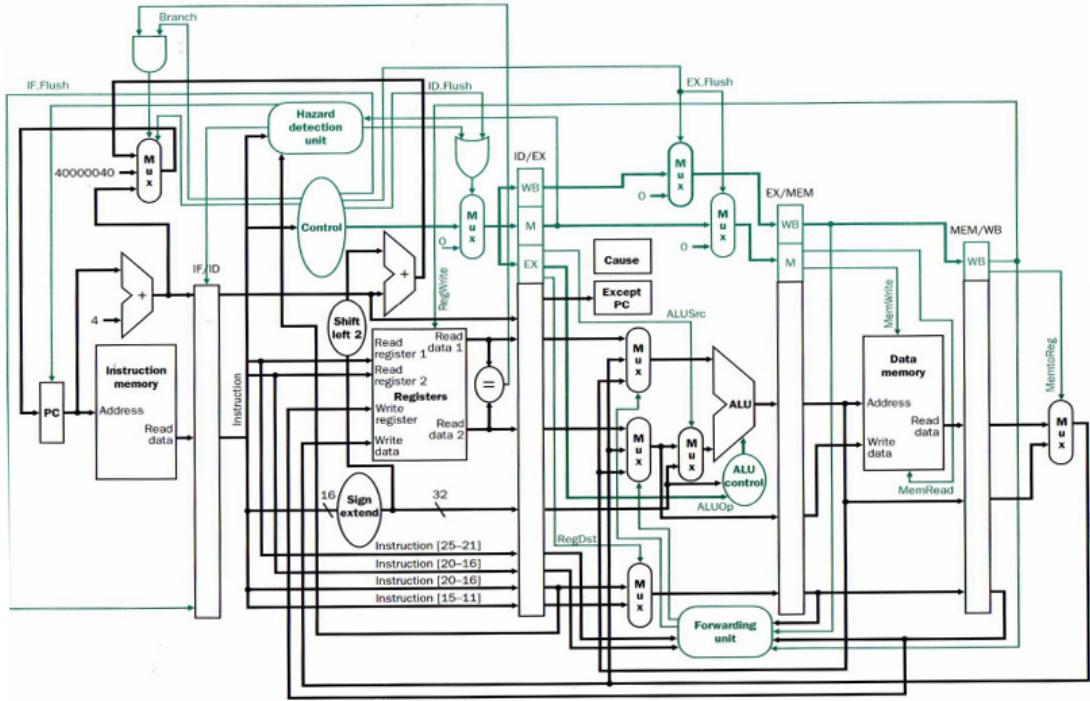
**Exemplul 9** De ce spunem ca instructiunile de tip I pot sa ruleze 2 instructiuni in loc de una?  
Luam ca exemplu:

li \$t0, 1234567890

Transformarea lui 1234567890 in baza 2 este: 1001001100101100000001011010010. Stim ca instructiunile de tip i au 16 biti la dispozitie pentru a tine valoarea immediate. $\Rightarrow$  numarul nostru nu incape in imm.

Deci procesorul va face 2 operatii:

lui \$t0, 0x4996  
ori \$t0, \$t0, 0x2d2



THE MIPS ARCHITECTURE OR  
SOMETHING, I DON'T KNOW,  
PLEASE WISHLIST AND FOLLOW  
PALMRIDE ON STEAM.

## References

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

## 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>24</b>

## 1 Rezolvarea subiectului 3, punctul c)

Exemplul 1 [RESTANTA SEPRTEMBRIE 2020]

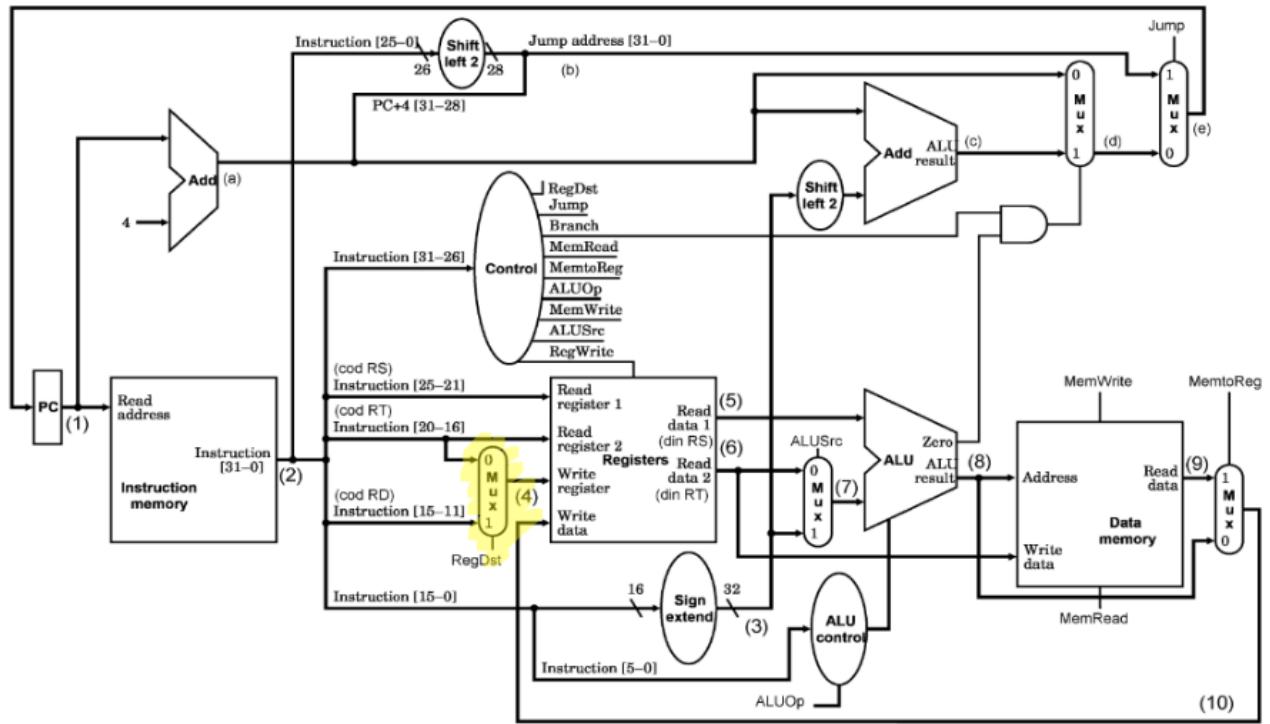
c) Adăugați procesorului implementarea instrucției: dlw rd, rs, rt  
care încarcă în registrul rd valoarea afărată în memorie la adresa care se obține adunând valorile din regiștrii rs și rt (adică efectueză  $rd := \text{mem}[rs + rt]$ ).

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:



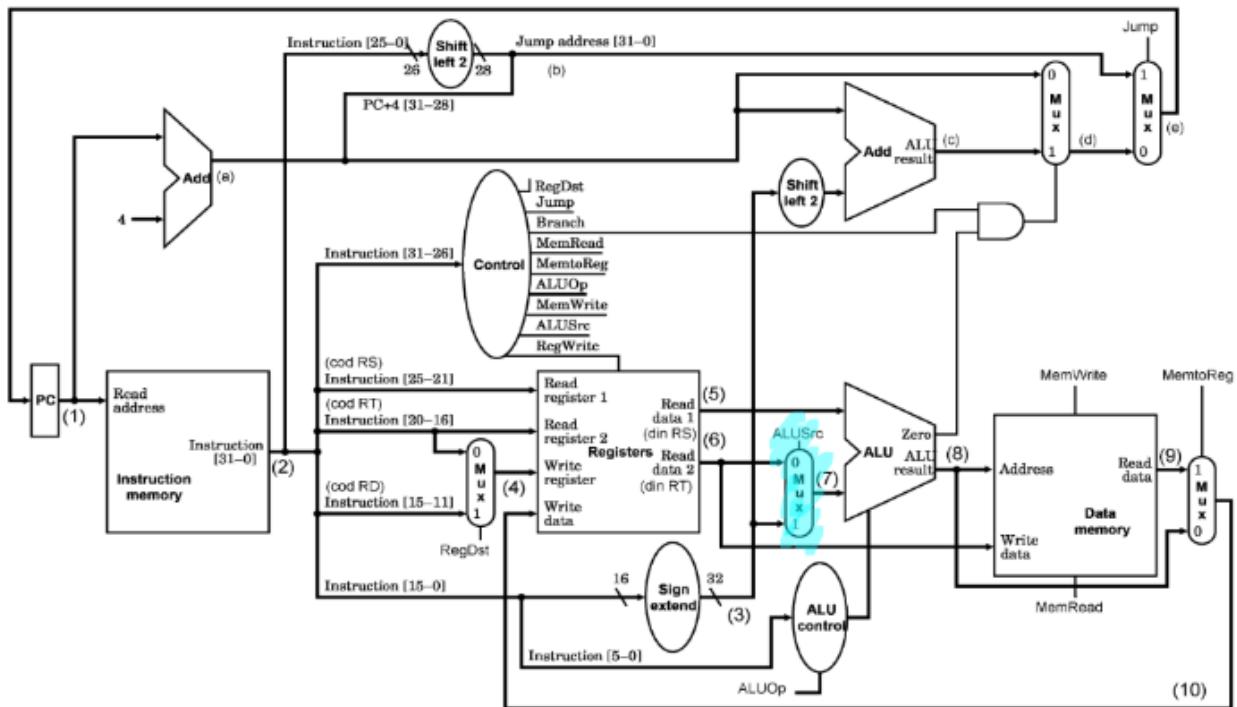
RegDst decide cine este registrul destinatie.

- 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 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 registru? 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.

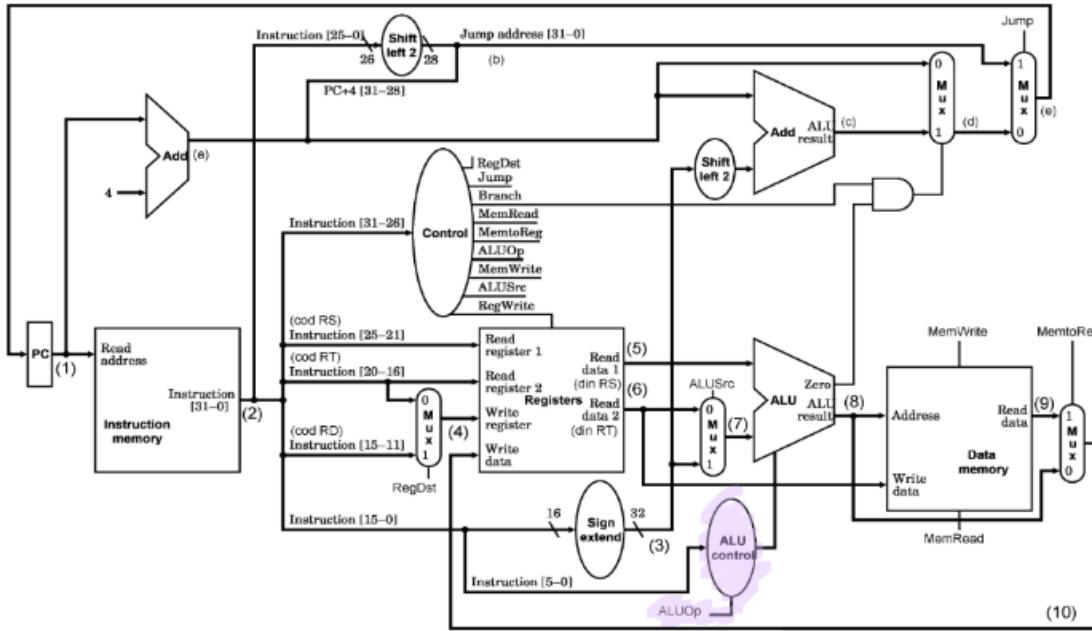
Instruction [31 - 26]	RegDst	ALU Src	Mem To Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
—	1	0	1	1	1	0	0			

- 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)

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

$$\begin{array}{r} 73 \\ 36 \\ 18 \\ 9 \\ 4 \\ 2 \\ 1 \end{array} \left| \begin{array}{l} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{array} \right. \Rightarrow (73)_2 = \overline{1001001}$$
$$0,75 \cdot 2 = 1,5 \quad | \Rightarrow (0,75)_2 = \overline{0,11}$$
$$0,5 \cdot 2 = 1,0 \quad | \Rightarrow (0,25)_2 = \overline{0,01}$$
$$\Rightarrow (73,75) = \overline{1001001,11}$$

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

$$\begin{array}{r} 5 \\ 2 \\ 1 \end{array} \left| \begin{array}{l} 1 \\ 0 \\ 1 \end{array} \right. \Rightarrow (5)_2 = \overline{101}$$
$$0,25 \cdot 2 = 0,5 \quad | \Rightarrow (0,25)_2 = \overline{0,01}$$
$$0,5 \cdot 2 = 1,0 \quad | \Rightarrow (0,25)_2 = \overline{0,01}$$
$$\Rightarrow (5,25)_2 = \overline{101,01}$$

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

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

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

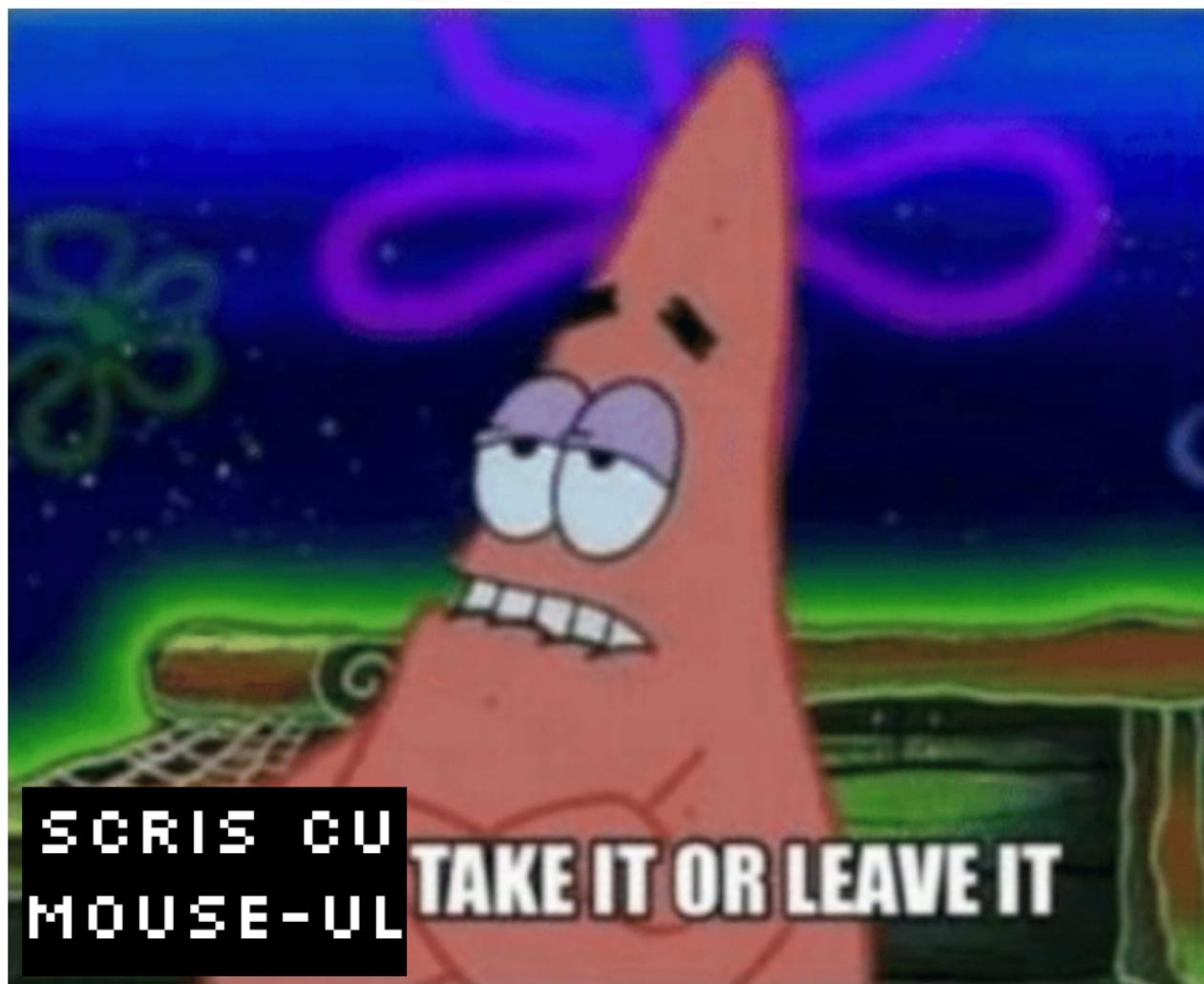
$2 < b \Rightarrow$  add c exp bei y da x

$$\Rightarrow y = -0,00010101 \cdot 2^6$$

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

---

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: 42938000

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

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

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

$$x = -\overline{0.f} = -\overline{0,01100}$$

$$(x)_{\frac{1}{2}} = -0,375$$

### 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

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

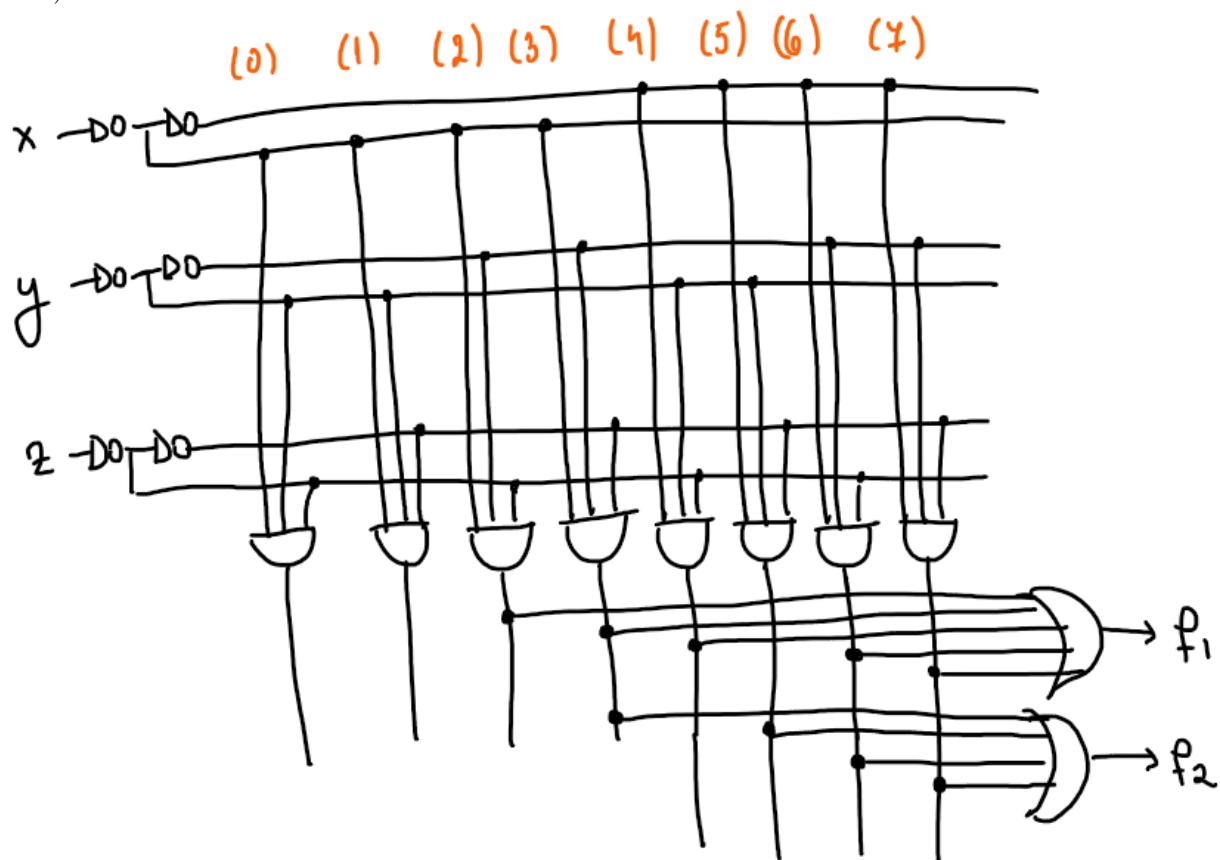
$$f_2 = \underset{(5)}{\bar{x}} \cdot \underset{(6)}{y} \cdot \underset{(7)}{z} + \underset{(6)}{x} \cdot \underset{(5)}{\bar{y}} \cdot \underset{(7)}{z} + \underset{(6)}{x} \cdot \underset{(7)}{y} \cdot \underset{(5)}{\bar{z}} + \underset{(7)}{x} \cdot \underset{(7)}{y} \cdot \underset{(5)}{\bar{z}}$$

FMC

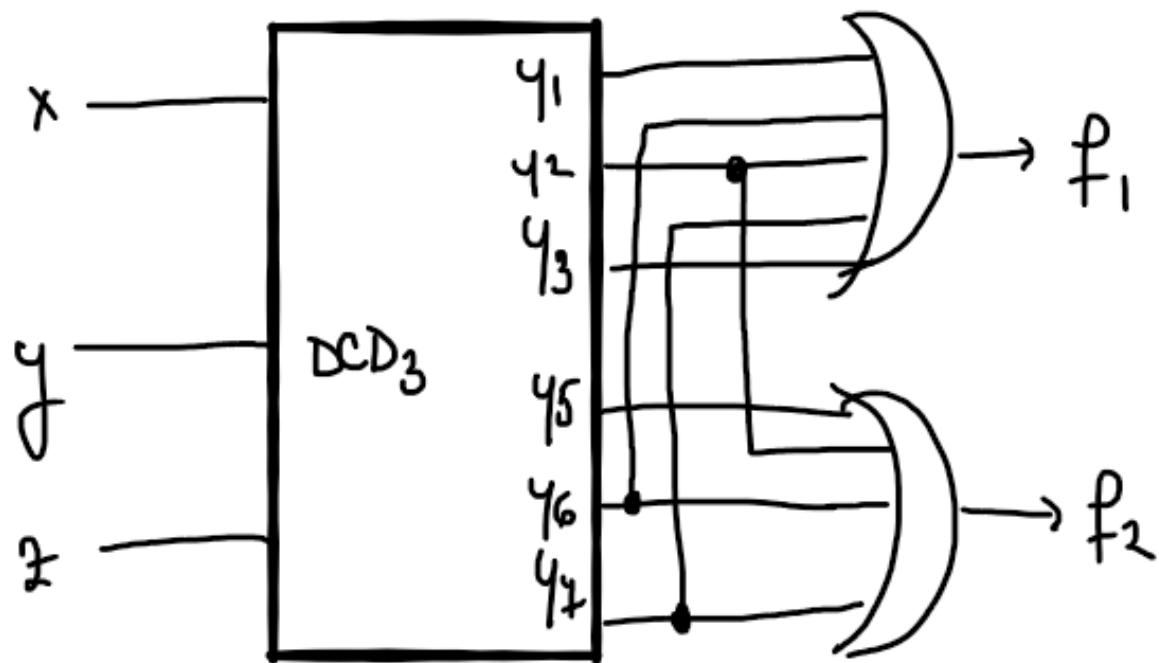
$$f_1 = \underset{(0)}{(x+y+z)} \cdot \underset{(4)}{(\bar{x}+y+z)} \cdot \underset{(5)}{(\bar{x}+\bar{y}+\bar{z})}$$

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

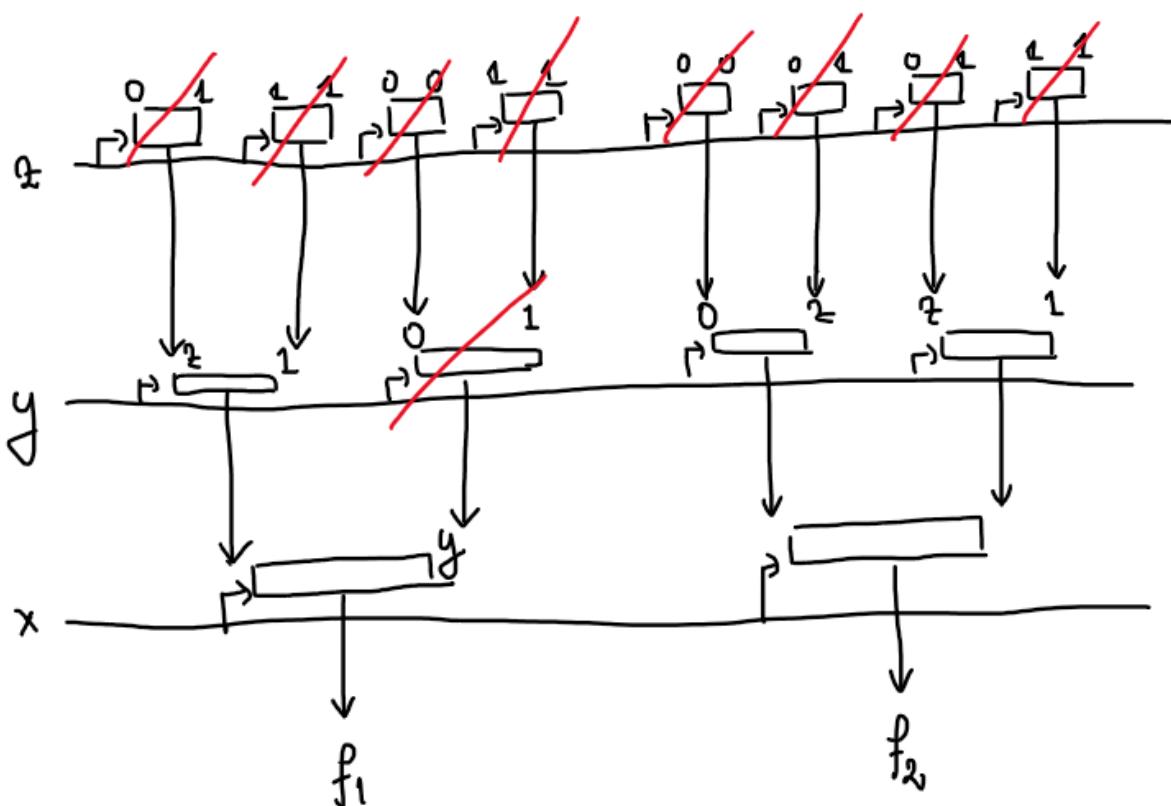
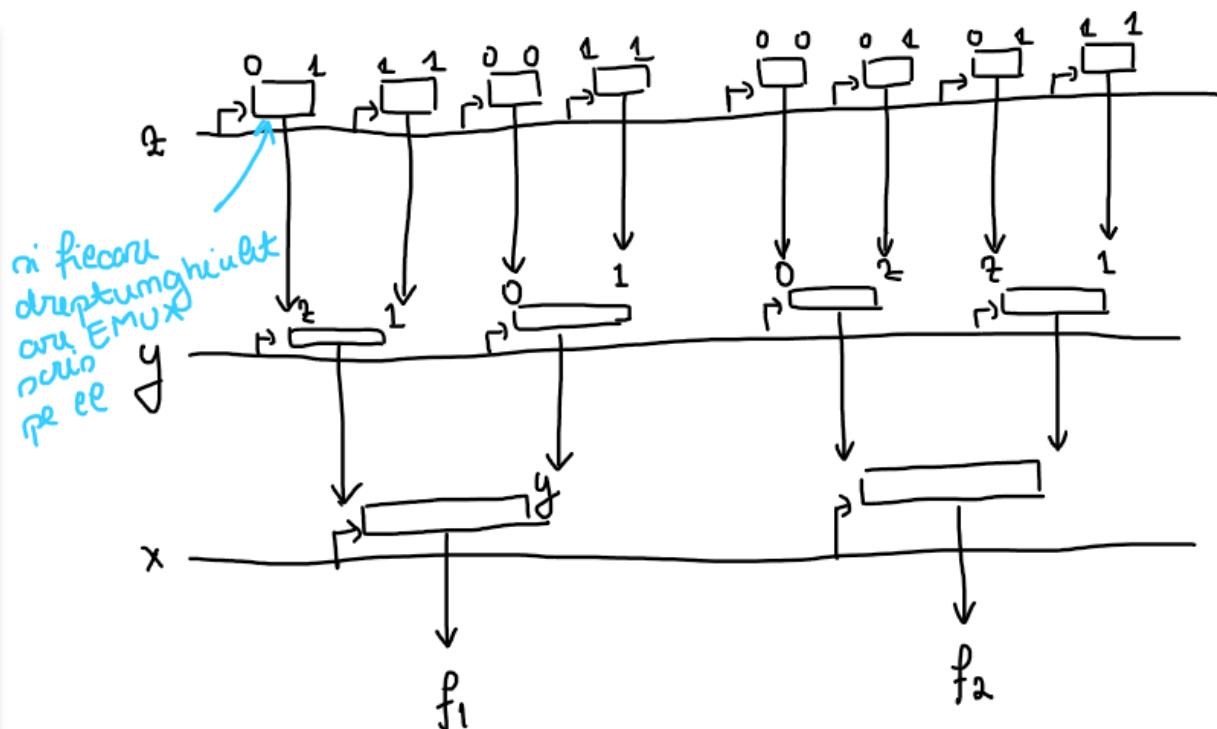
b)



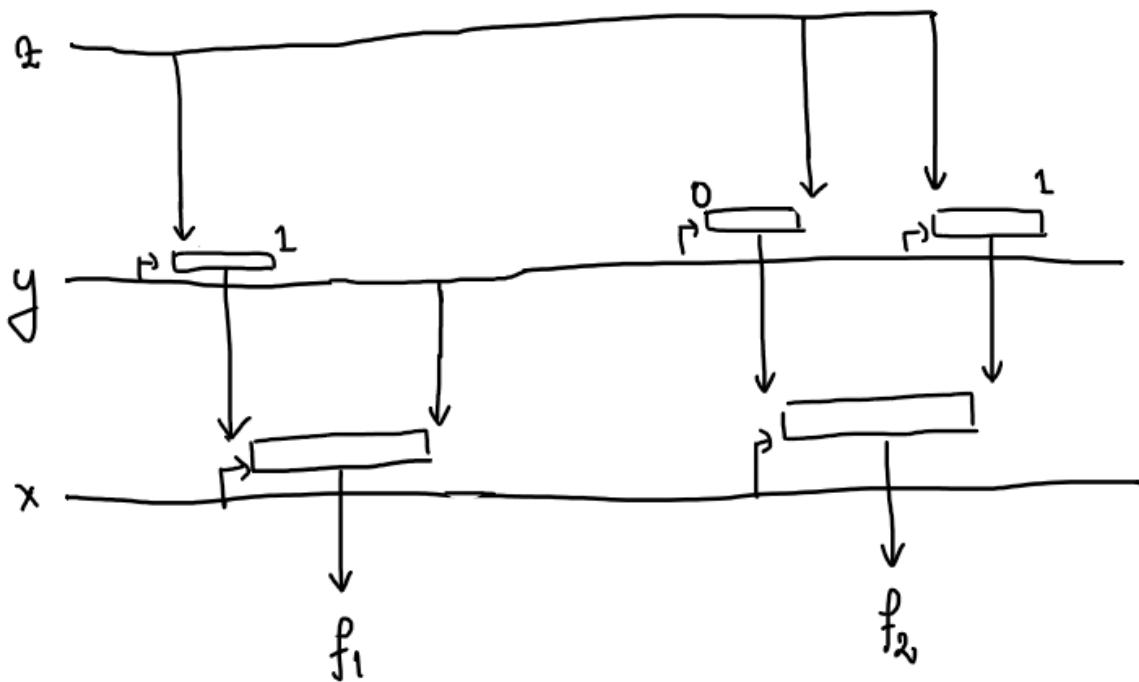
c)



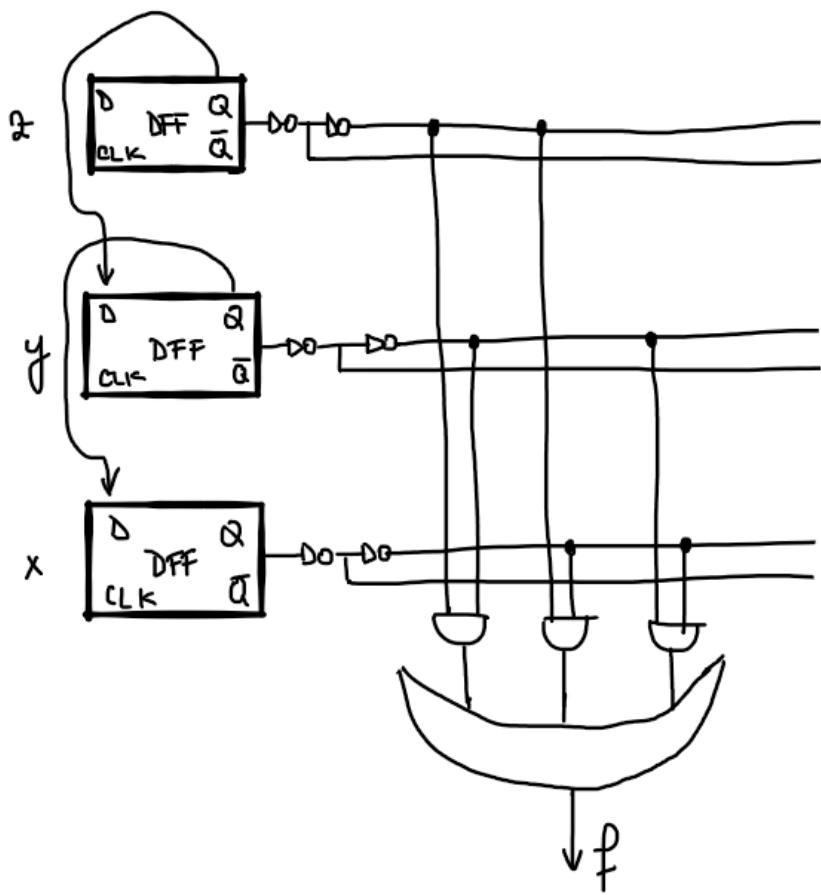
d)



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:

- $\$t3 = (11)_6 = \$t = \overline{B}$  (in hexa)
- $\$t4 = (12)_6 = \$s = \overline{C}$  (in hexa)
- offset =  $(4)_6 = \overline{4}$  (in hexa)

Care e valoarea lui opcode in hexa?

$$\overline{(101011)}_2 = \overline{(2B)}_6$$

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)
- $\$t2 = (11)_6 = \$s = \overline{01011}$  (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.

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

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

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

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

na ignorati complet  
la intamplarea 2  
ce game

Instruction	RegDst	ALUSrc	Memto-Reg	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	

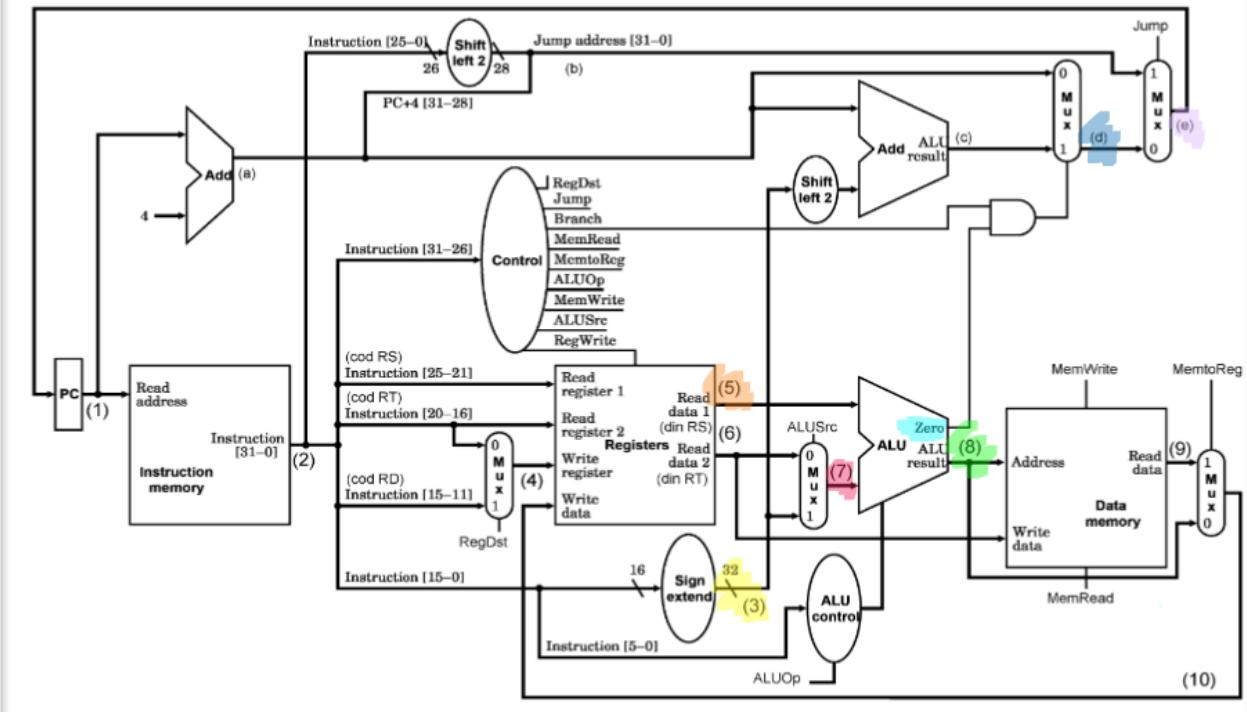
### ALU Control (slide 7.36)

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)	B	M2	MW	ALU Src	ALU Op (2b)	ALU Ctrl (3b)	PC	Mem[4]
Initial	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$\alpha$	5
SW	$\alpha$	4	C	4	10	0	athath	0	0	1	1	00	010	ath	2	
beq	$\alpha+8$	-3	2	2	0	1	$\alpha$	$\alpha$	1	0	0	0	X1	110	$\alpha$	2

$\nwarrow (\alpha+12) + (-3) \cdot 4 = \alpha$



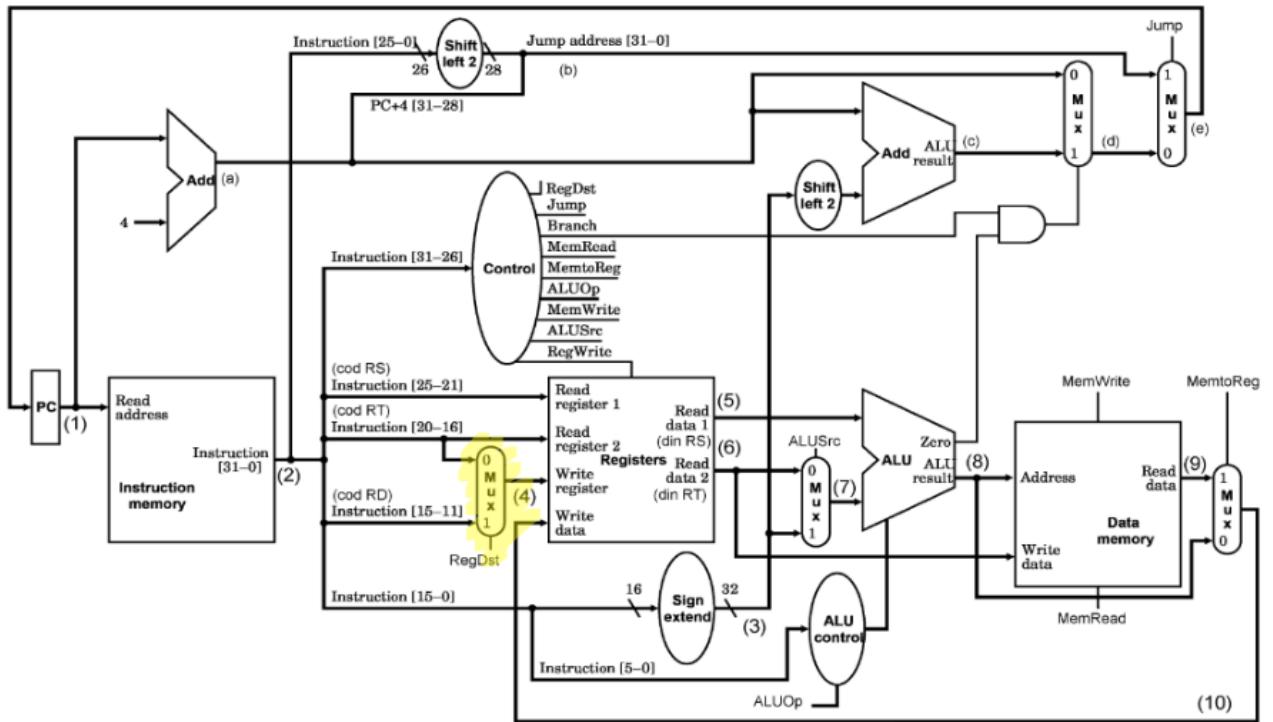
c) Adaugati procesorului implementarea **subp rt, rs** care atribuie registrului rt diferentea valorilor din registrii **rs** si **rt** (adica efectueaza  $rt := rs - rt$ ); se va folosi formatul I, cu  $op=0xFF$  si  $imm=0x0$ .

Pentru implementare, este suficienta adaugarea unei linii in tabelul "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:

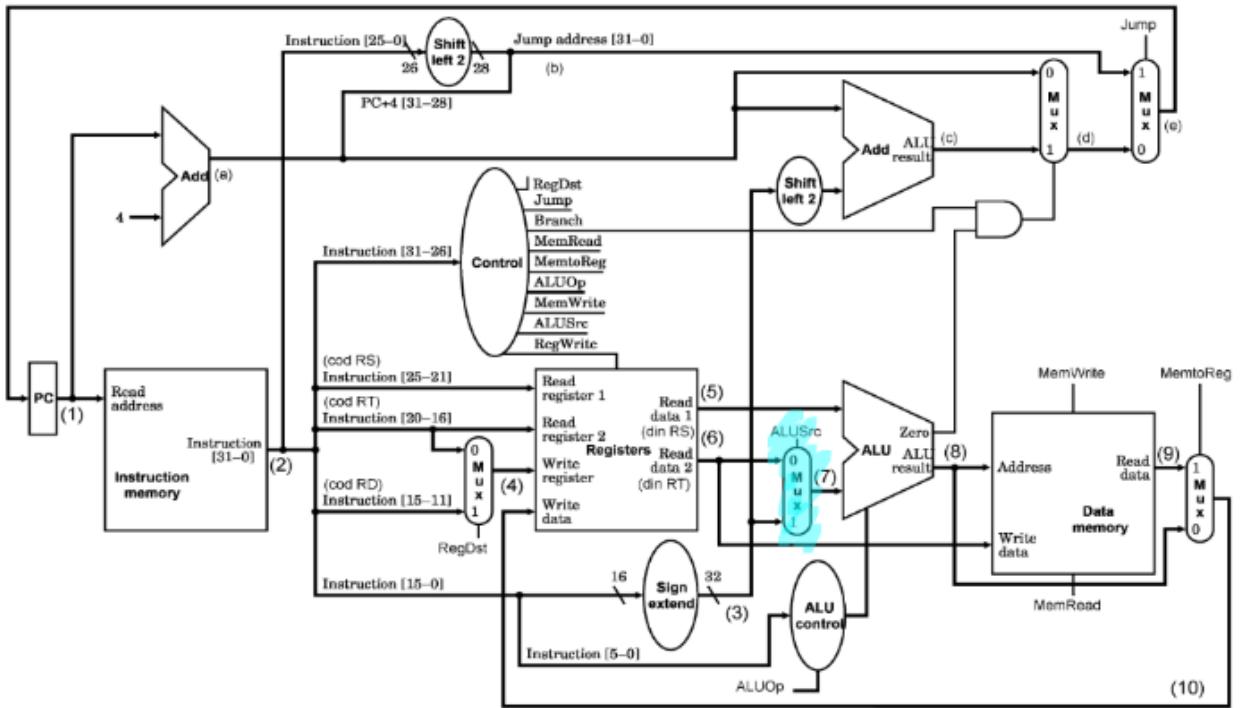


RegDst decide cine este registrul destinatie.

- 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?



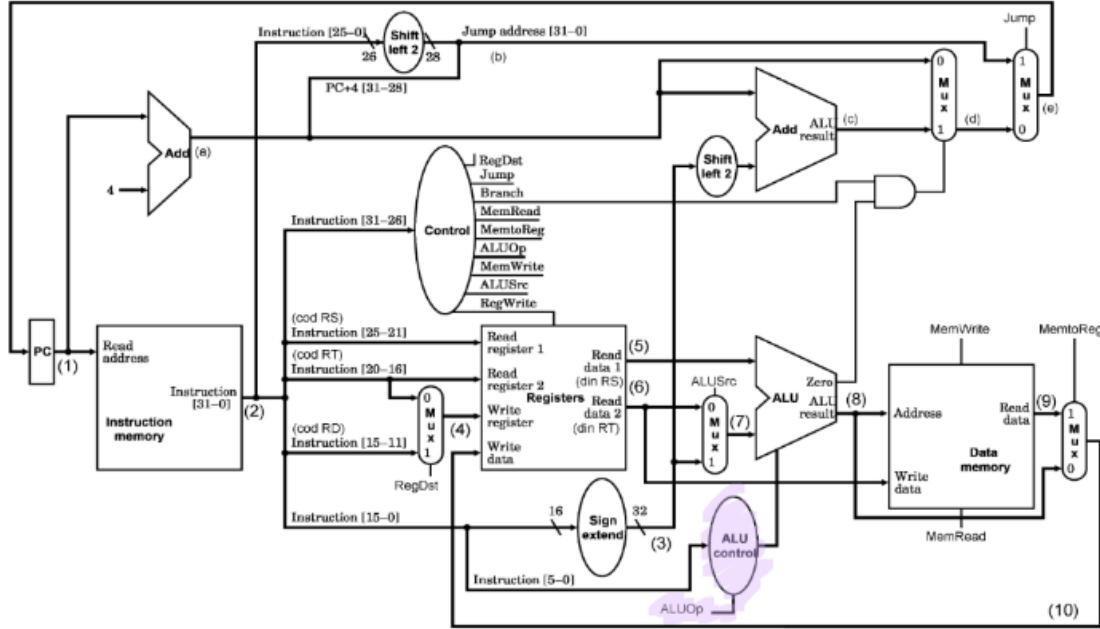
Face diferenta dintre instructiunile care fac operatii pe valori si cele care fac operatii pe adrese de memorie.

- 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:

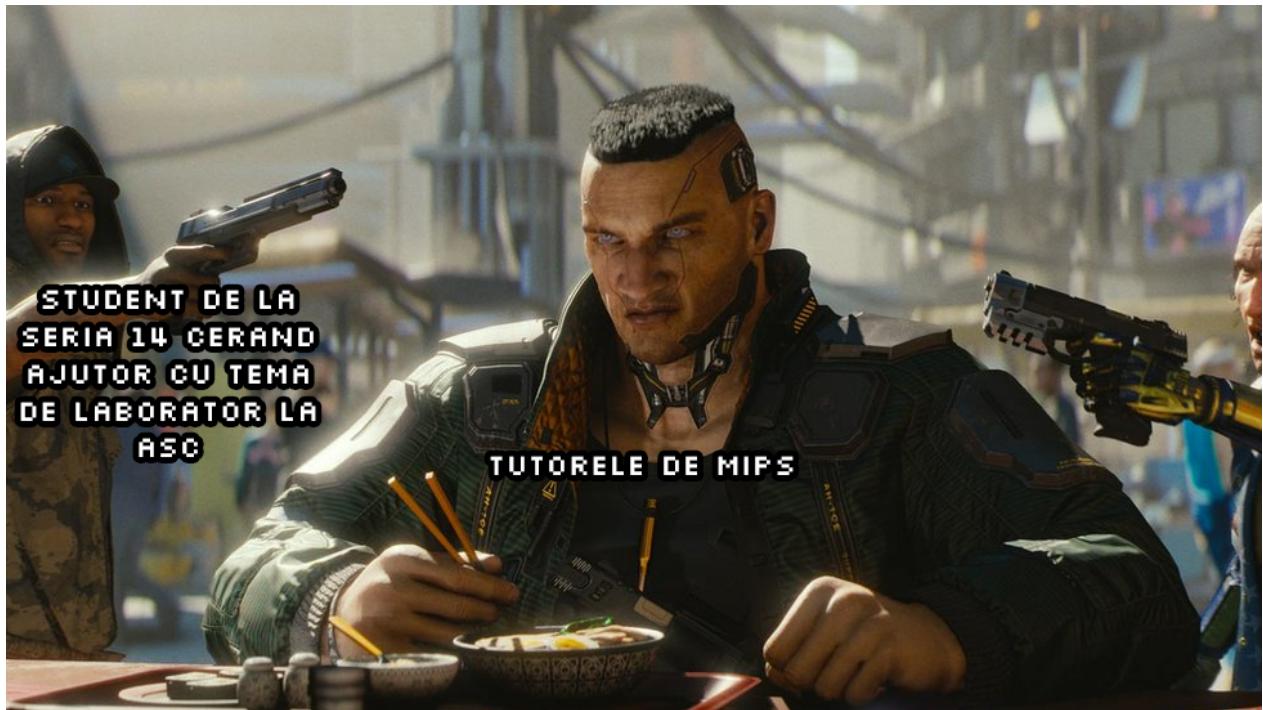
qc	op/RS/RT/RD/0/FCT op/RS/RT/immu	(3)	CTRL RegDst/ALUSrc/MemToReg/ RegWrite/ MemRead/ MemWrite/ Branch/Jump / ALUOp	ALU ctrl	(4)	(5)	(6)	(7)	ALU result (8)	ALU zero	(9)	(10)	(e)
lw	$\alpha$	23/8/9/h	h	010 +	g	p	?	h	p+h	?	A	A	$\alpha+h$
add	$\alpha+h$	0/9/9/A/0/20	5020	010 +	A	A	A	A	h	0	?	1h	$\alpha+h$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

## 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]

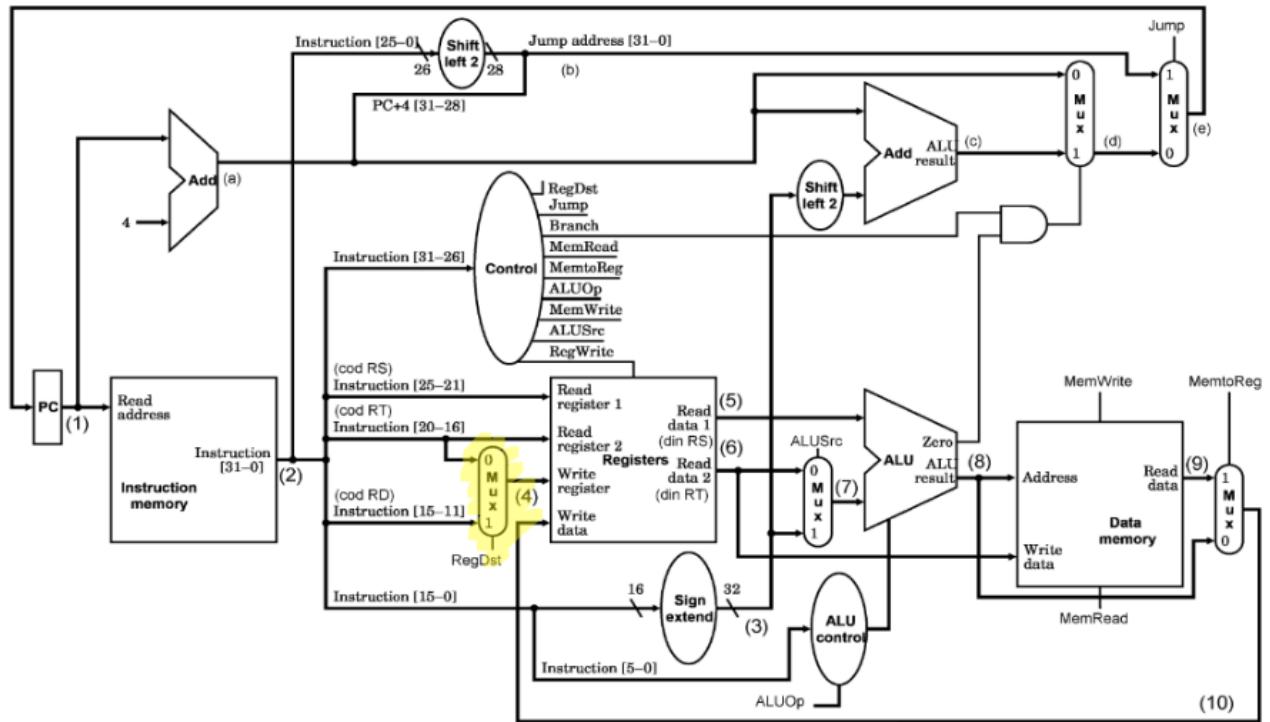
c) Adăugați procesorului implementarea instrucției: dlw rd, rs, rt  
care încarcă în registrul rd valoarea afărată în memorie la adresa care se obține adunând valorile din regiștrii rs și rt (adică efectueză  $rd := \text{mem}[rs + rt]$ ).

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:



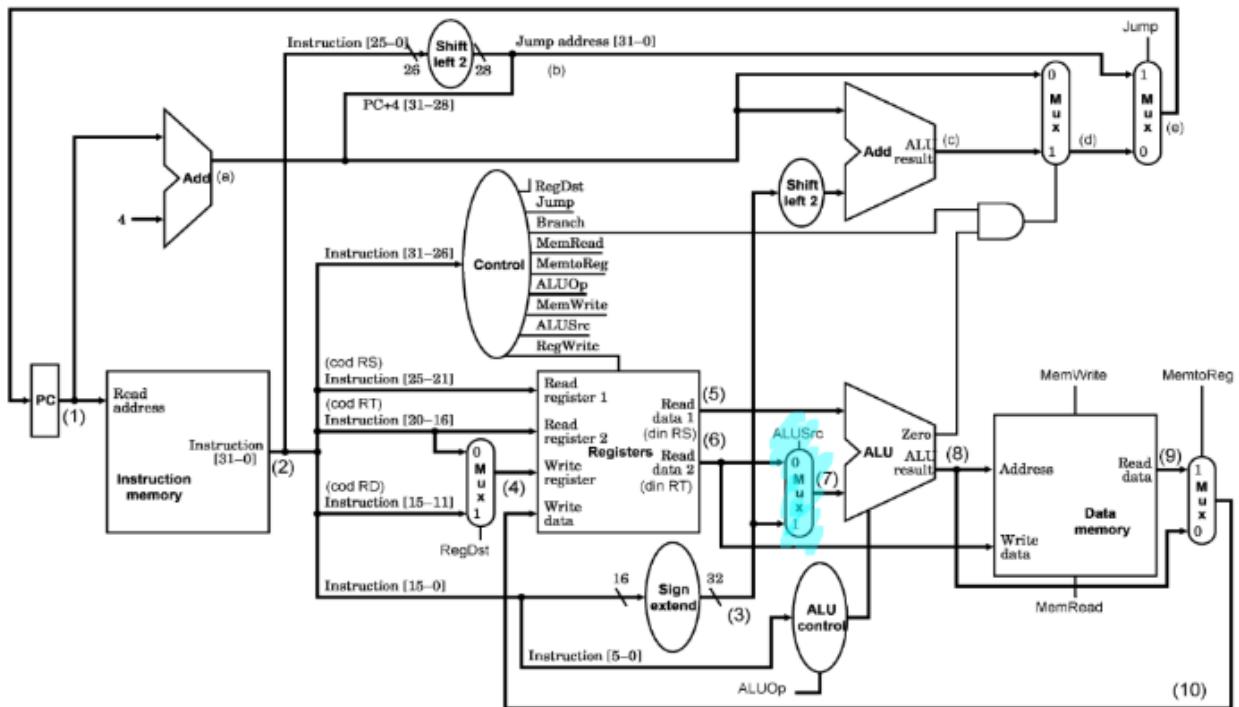
RegDst decide cine este registrul destinație.

- 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.

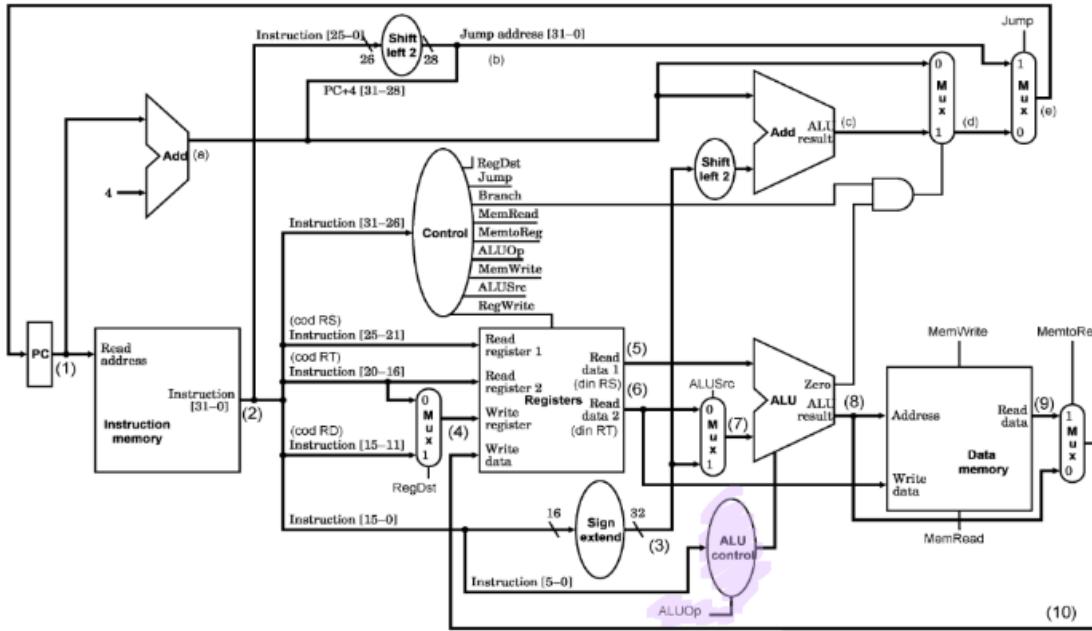
Instruction [31 - 26]	RegDst	ALU Src	Mem To Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
—	1	0	1	1	1	0	0			

- 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 incarc 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)

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

$$\begin{array}{r} 73 \\ 36 \\ 18 \\ 9 \\ 4 \\ 2 \\ 1 \end{array} \left| \begin{array}{l} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{array} \right. \Rightarrow (73)_2 = \overline{1001001}$$
$$0,75 \cdot 2 = 1,5 \quad | \Rightarrow (0,75)_2 = \overline{0,11}$$
$$0,5 \cdot 2 = 1,0 \quad | \Rightarrow (0,25)_2 = \overline{0,01}$$
$$\Rightarrow (73,75) = \overline{1001001,11}$$

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

$$\begin{array}{r} 5 \\ 2 \\ 1 \end{array} \left| \begin{array}{l} 1 \\ 0 \\ 1 \end{array} \right. \Rightarrow (5)_2 = \overline{101}$$
$$0,25 \cdot 2 = 0,5 \quad | \Rightarrow (0,25)_2 = \overline{0,01}$$
$$0,5 \cdot 2 = 1,0 \quad | \Rightarrow (0,25)_2 = \overline{0,01}$$
$$\Rightarrow (5,25)_2 = \overline{101,01}$$

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

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

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

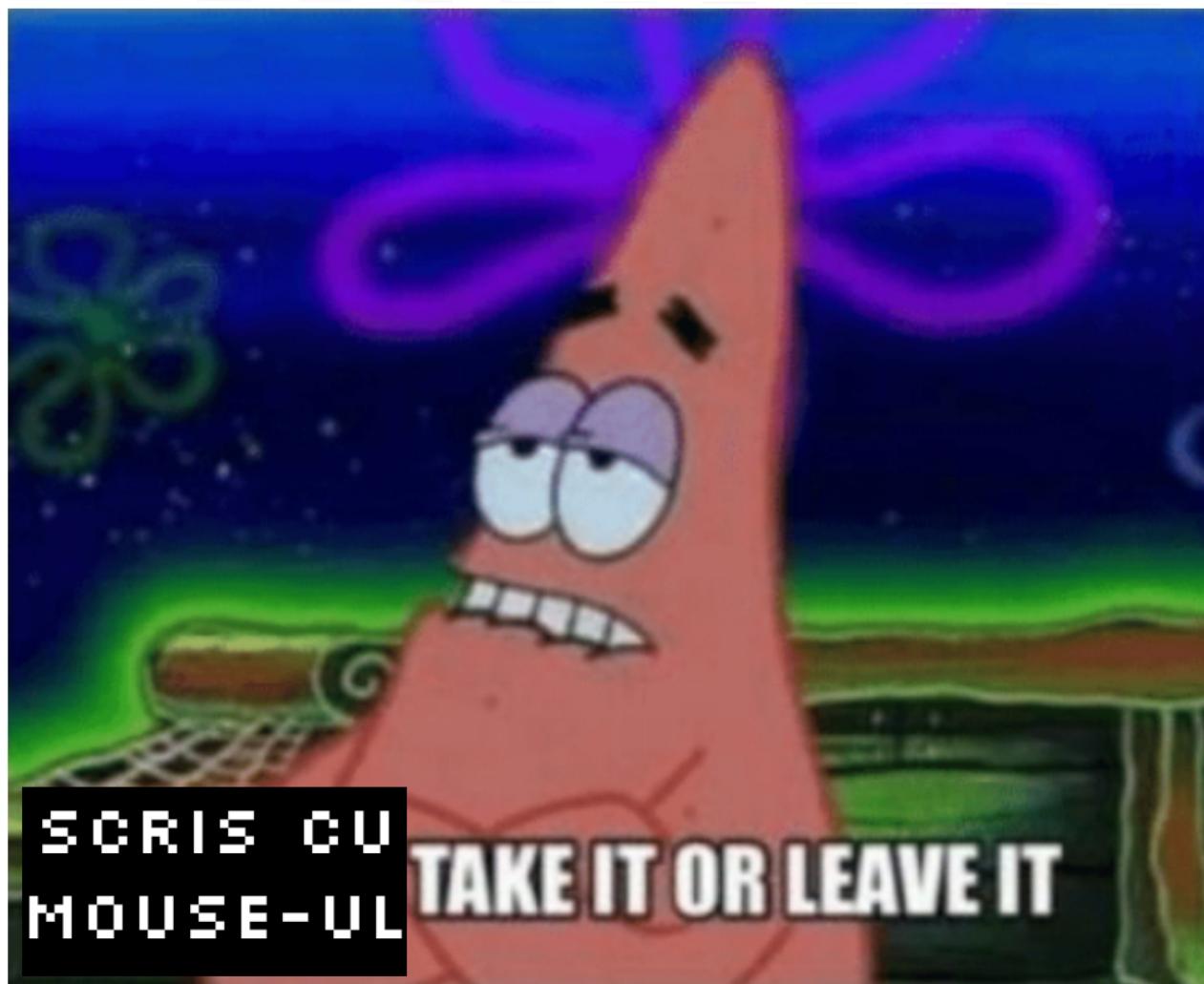
$2 < b \Rightarrow$  add c exp bei y da x

$$\Rightarrow y = -0,00010101 \cdot 2^6$$

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

---

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: 42938000

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

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

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

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

$$(x)_{\frac{1}{2}}^{-1} = -0,375$$

### 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

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

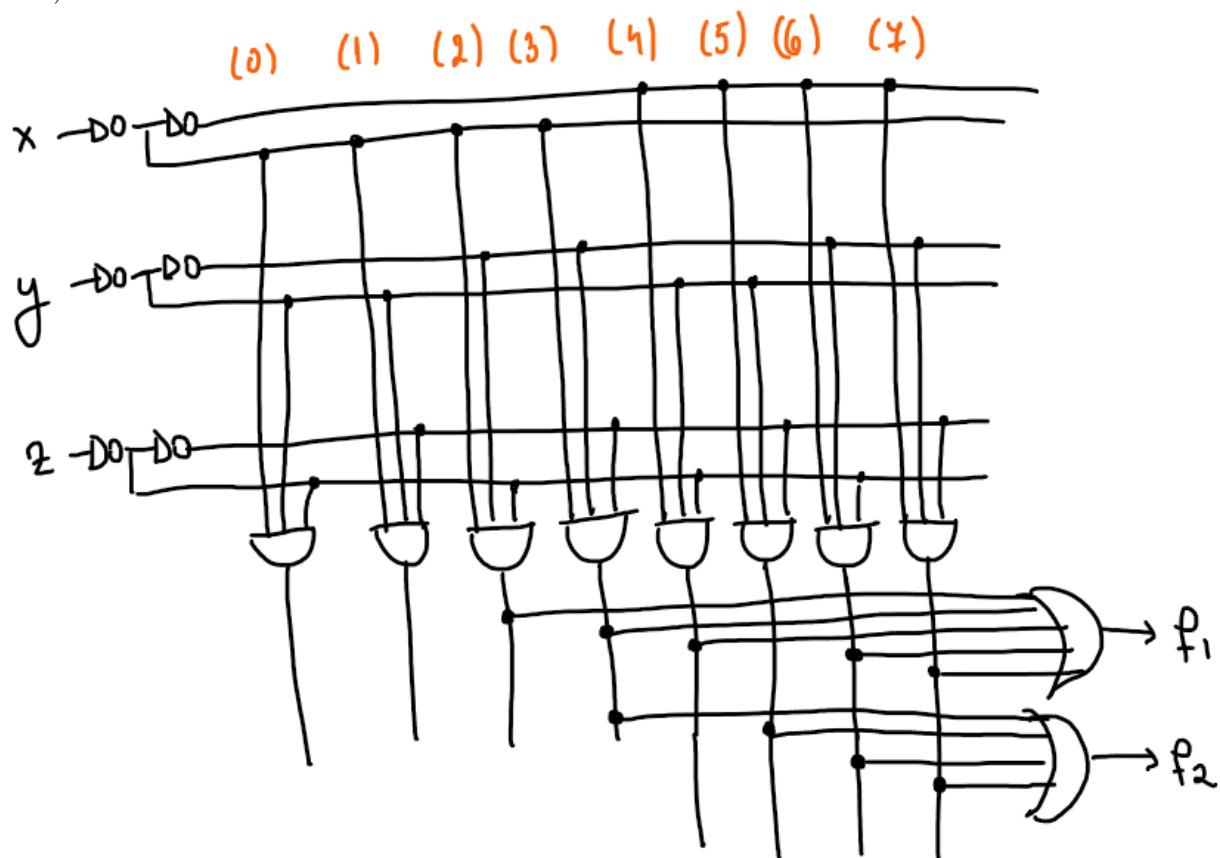
$$f_2 = \underset{(5)}{\bar{x}} \cdot \underset{(6)}{y} \cdot \underset{(7)}{z} + \underset{(6)}{x} \cdot \underset{(5)}{\bar{y}} \cdot \underset{(7)}{z} + \underset{(6)}{x} \cdot \underset{(7)}{y} \cdot \underset{(5)}{\bar{z}} + \underset{(7)}{x} \cdot \underset{(7)}{y} \cdot \underset{(5)}{\bar{z}}$$

FMC

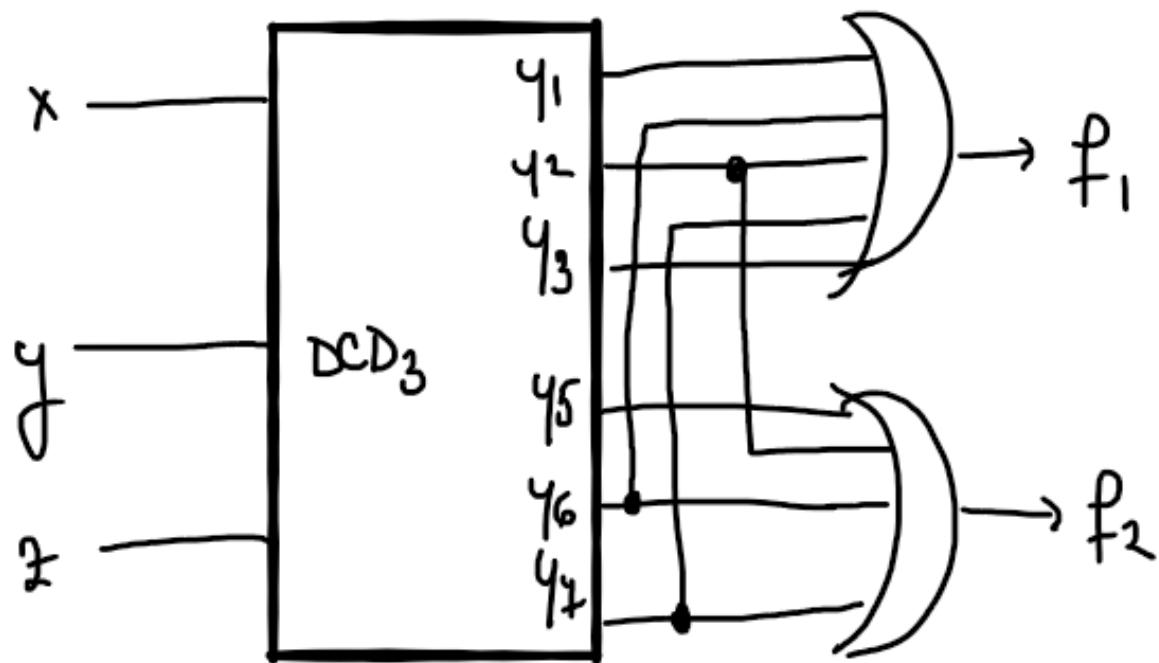
$$f_1 = \underset{(0)}{(x+y+z)} \cdot \underset{(4)}{(\bar{x}+y+z)} \cdot \underset{(5)}{(\bar{x}+\bar{y}+\bar{z})}$$

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

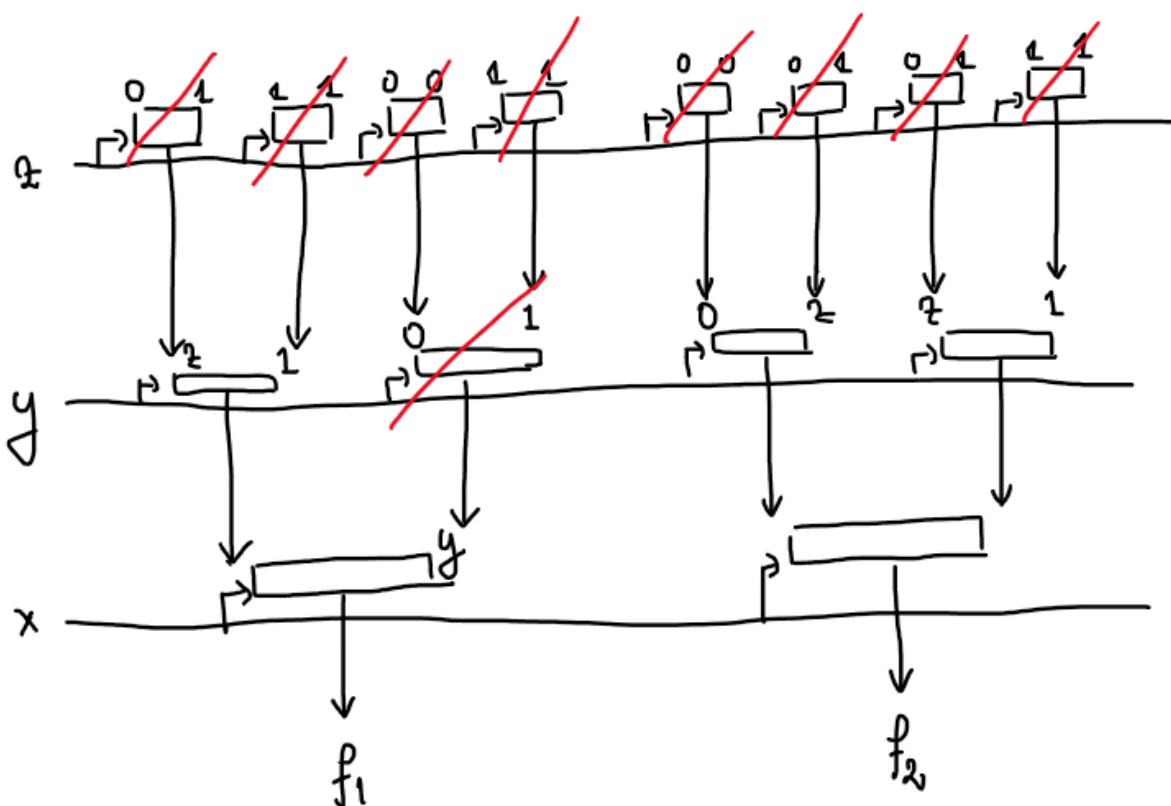
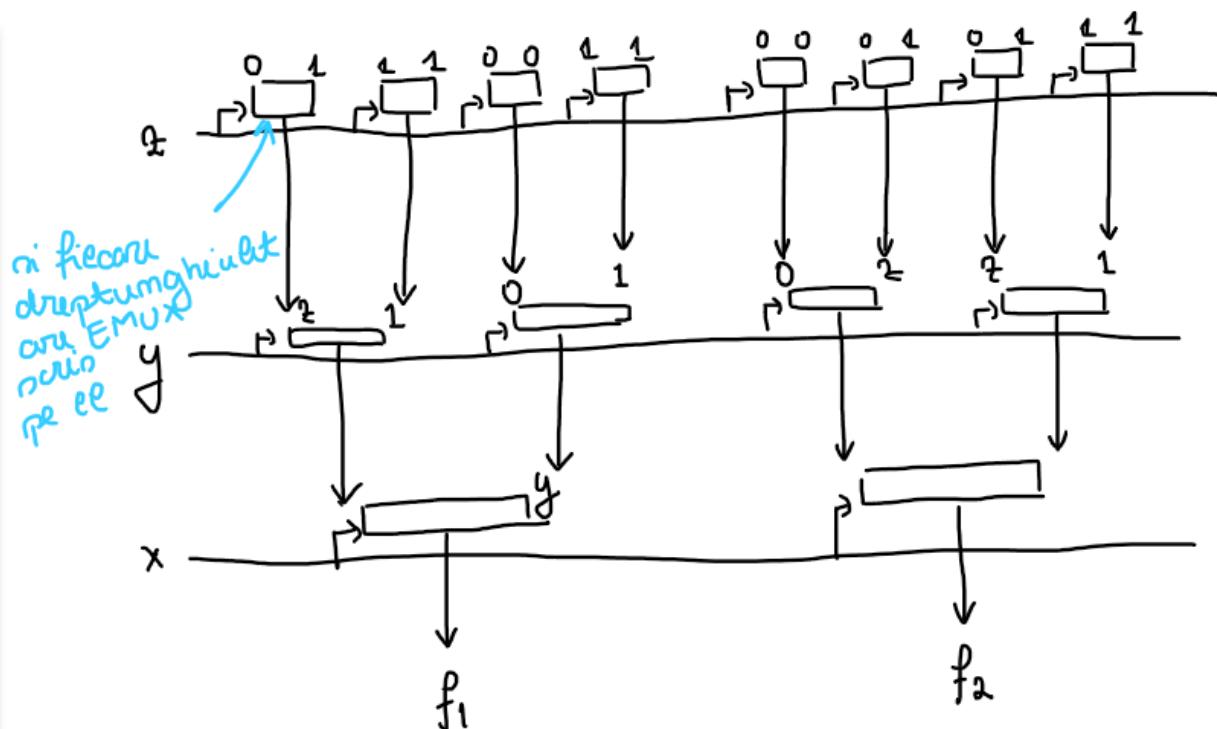
b)



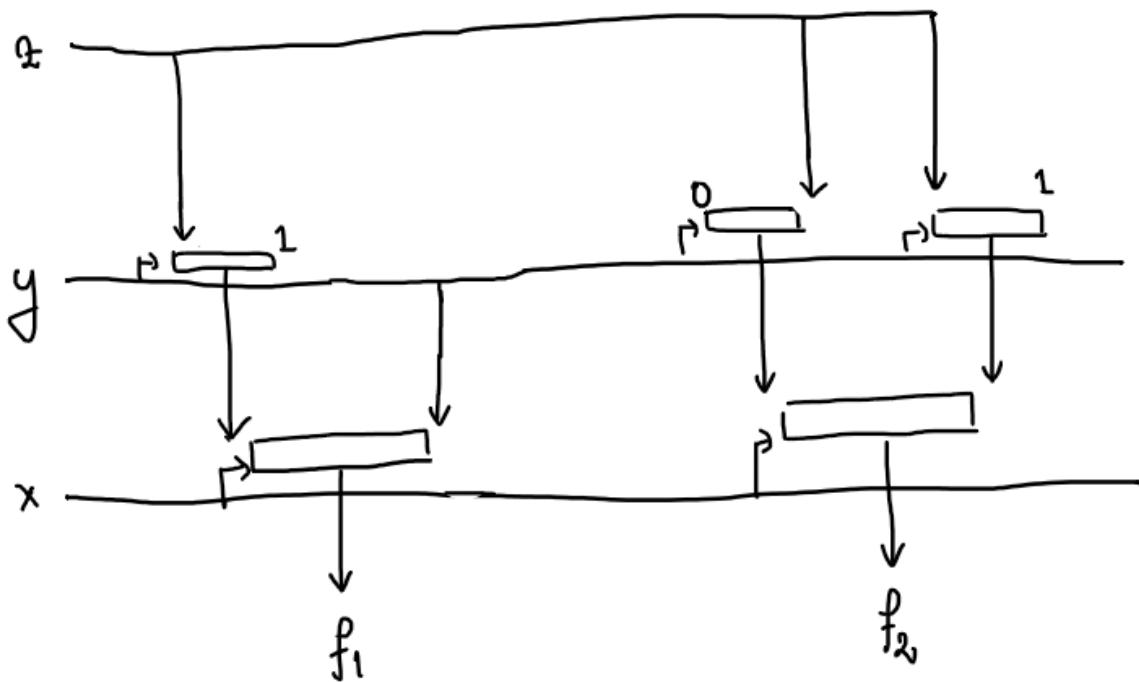
c)



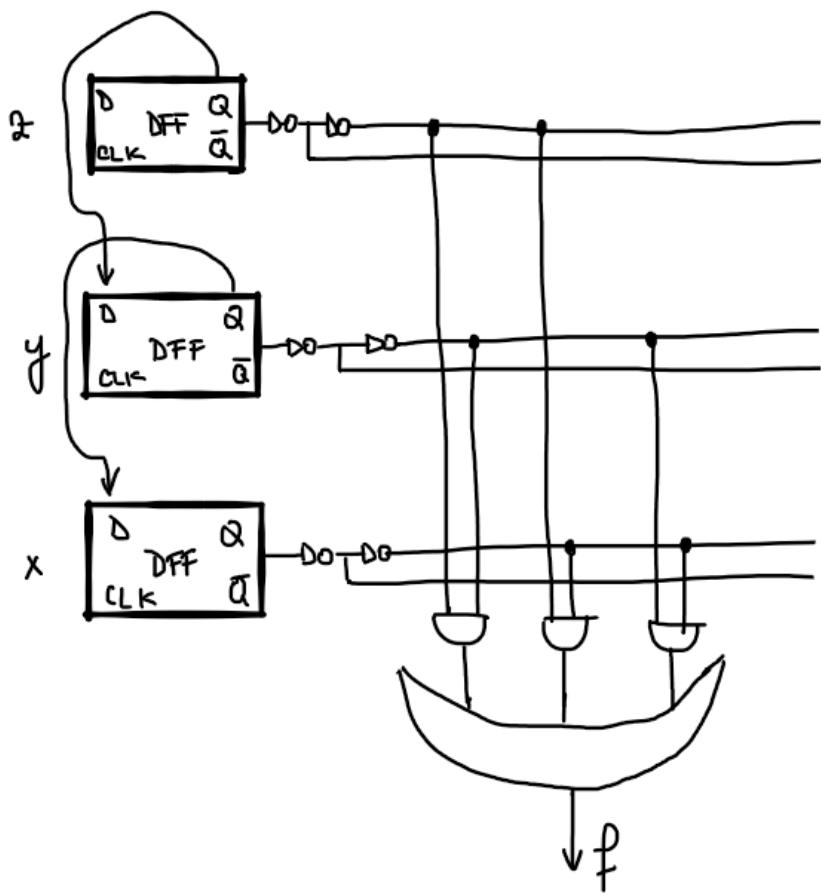
d)



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	

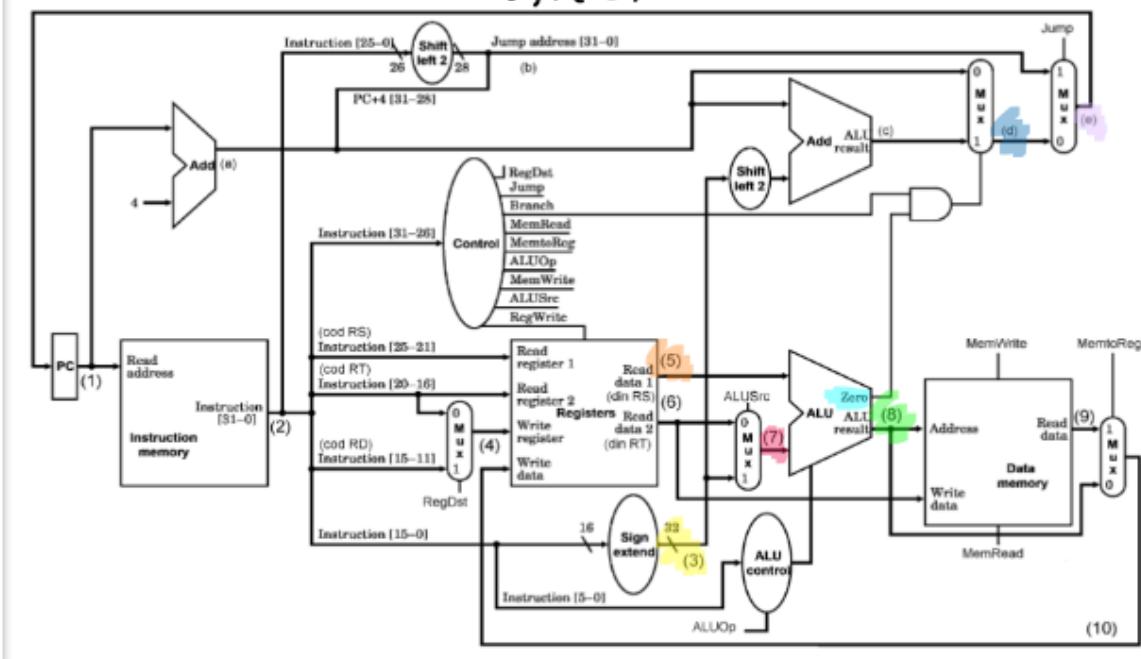
### ALU Control (slide 7.36)

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.

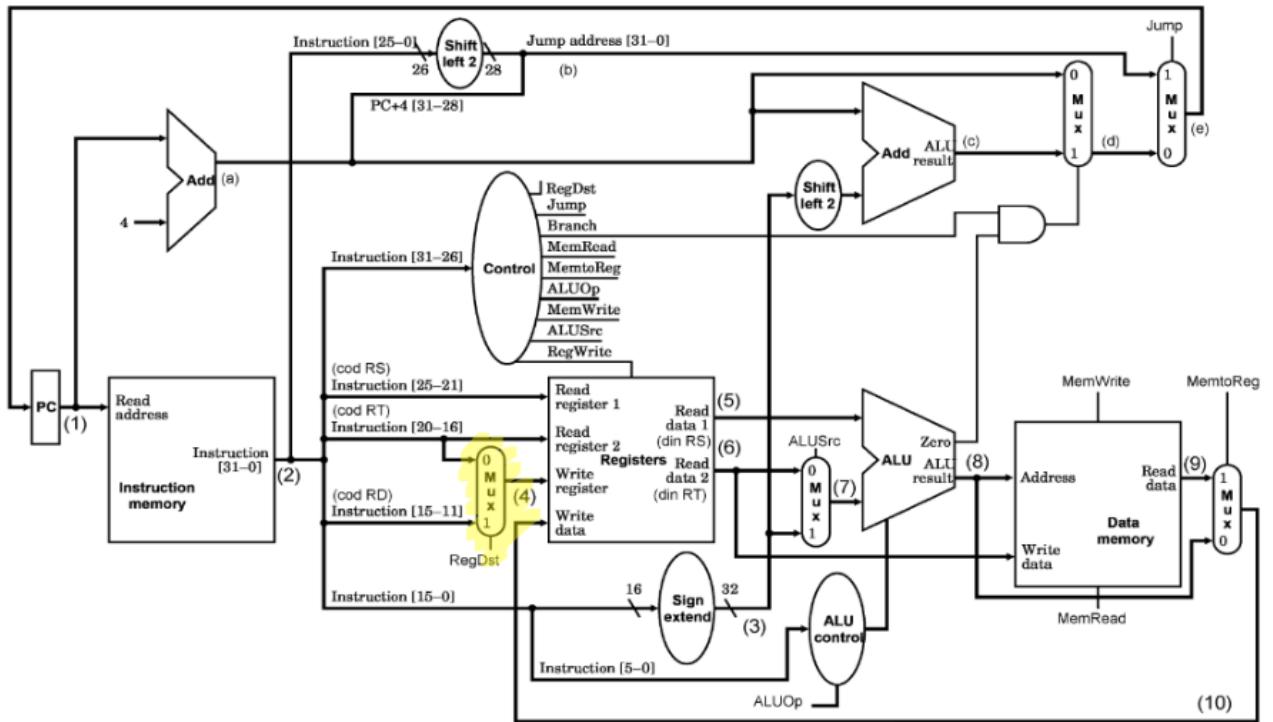
c) Adagati procesorului implementarea **subp rt, rs** care atribuie registrului rt diferența valorilor din registrii **rs** si **rt** (adica efectueaza  $rt := rs - rt$ ); se va folosi formatul I, cu op=0xFF si imm=0x0.

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:

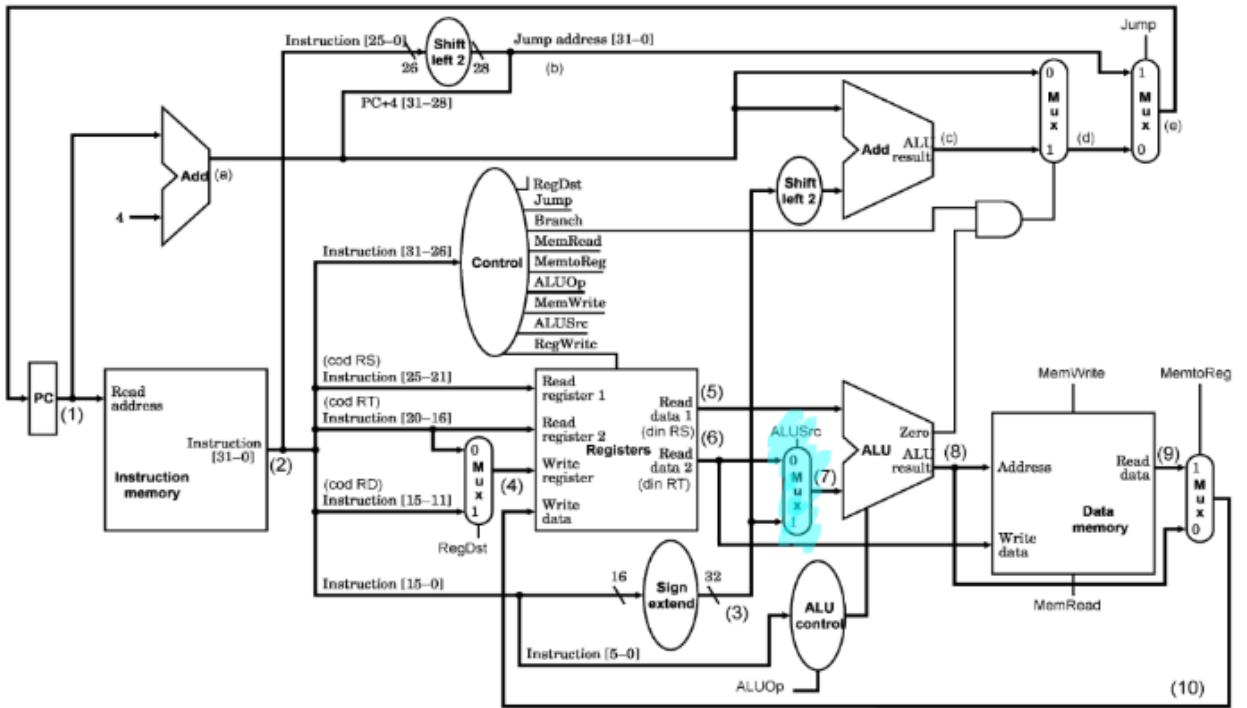


RegDst decide cine este registrul destinatie.

- 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?



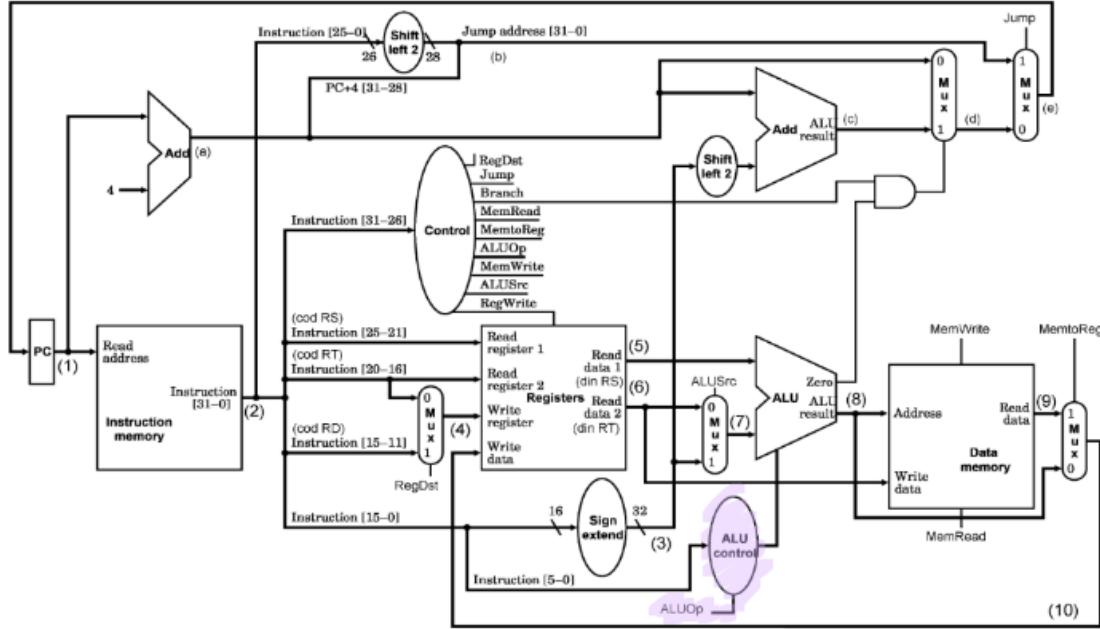
Face diferenta dintre instructiunile care fac operatii pe valori si cele care fac operatii pe adrese de memorie.

- 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:

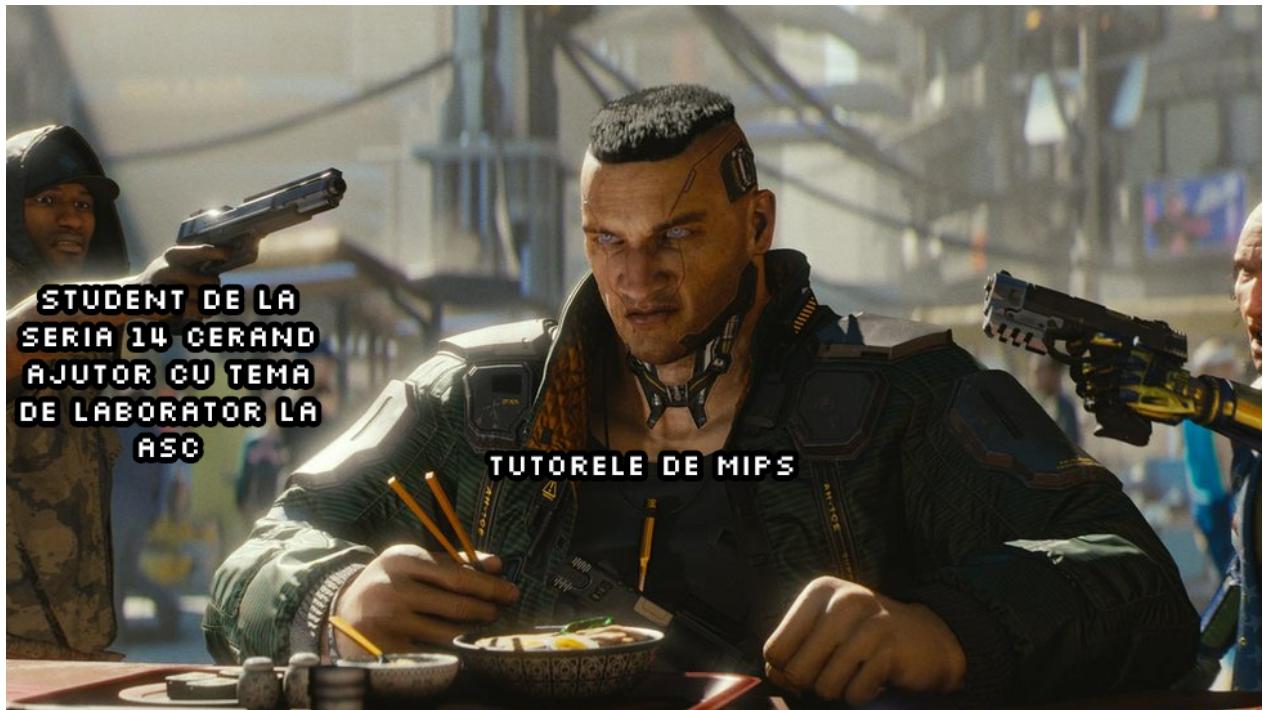
qc	op/RS/RT/RD/0/FCT op/RS/RT/immu	(3)	CTRL RegDst/ALUSrc/MemToReg/ RegWrite/MemRead/ MemWrite/Branch/Jump / ALUOp	ALU ctrl	(4)	(5)	(6)	(7)	ALU result (8)	ALU zero	(9)	(10)	(e)
lw	$\alpha$	23/8/9/h	h	010 +	g	p	?	h	p+h	?	A	A	$\alpha+h$
add	$\alpha+h$	0/9/9/A/0/20	5020	010 +	A	A	A	A	h	0	?	1h	$\alpha+8$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

# Tutoriat 8

Stan Bianca-Mihaela, Stăncioiu Silviu

December 2020

Vreți ca la următorul tutoriat să continuăm cu recapitularea pentru seminar, sau să lucrăm niște probleme de MIPS date la teste de laborator în alți ani?

- Adăugat de tine MIPS is love, MIPS is life  38 voturi
- Adăugat de tine Gândesc ASC-ul în mod metafizic, neavând nevoie de pregătire adițională la acest capitol.  23 voturi
- Adăugat de tine Continuare seminar, lol  5 voturi

**CEI CARE AU ALES OPTIUNEA CU "GÂNDESC ASC-UL"  
IN MOD METAFIZIC":**



# 1 MIPS is love, MIPS is life



## 1.1 Subiecte date de Bogdan Macovei

Exercițiul 1 Să se rezolve următorul model de test de laborator din 2019:

<https://drive.google.com/file/d/1Om0lDKOZjVsMIgSNo-qd5BV9YZ3Y886M/view?usp=sharing>

Rezolvare

Partea I

```
1 .data
2
3     v::space 400                      # vectorul pe care vom lucra
4
5 .text
6
7 suma_cifrelor_para:
8
9     subu $sp, 4                      # punem $fp pe stiva
10    sw $fp, 0($sp)                   # stiva este $sp: ($fp) (z) (
11        celealte_lucruri_care_sunt_deja_pe_stiva)
12
13    addi $fp, $sp, 4                  # face fp sa pointeeze catre cadrul nostru de apel
14    # avem $sp: ($fp) <fp_pointeaza_aici> (z) (
15        celealte_lucruri_care_sunt_deja_pe_stiva)
16
17    subu $sp, 4                      # pune $ra pe stiva
18    sw $ra, 0($sp)                   # stiva este $sp: ($ra) ($fp) <fp_pointeaza_aici> (z) (
19        celealte_lucruri_care_sunt_deja_pe_stiva)
20
21    subu $sp, 4                      # pune $s0 pe stiva
```

```

19    sw $s0, 0($sp)          # stiva este $sp: ($s0) ($ra) ($fp) <fp_pointeaza_aici>
20    (z) (celealte_lucruri_care_sunt_deja_pe_stiva)
21
22    subu $sp, 4             # pune $s1 pe stiva
23    sw $s1, 0($sp)          # stiva este $sp: ($s1) ($s0) ($ra) ($fp) <
24      fp_pointeaza_aici> (z) (celealte_lucruri_care_sunt_deja_pe_stiva)
25
26    lw $s0, 0($fp)          # in $s0 il vom avea pe z si il vom imparti succesiv la
27      10
28    li $v0, 0                # pana cand va avea valoarea 0
29      rezultatul final
30
31    parc_cifre:
32
33      beq $s0, 0, exit_parc_cifre # am ajuns la 0, deci numarul nu mai are cifre
34
35      rem $s1, $s0, 10          # in $s0 vom avea ultima cifra a numarului
36      add $v0, $v0, $s1          # adunam cifra la suma
37
38      div $s0, $s0, 10          # impartim numarul la 10, adica scoatem ultima lui cifra
39      # de la sfarsit
40
41      j parc_cifre            # parcurgem cifrele
42
43    exit_parc_cifre:          # am terminat de parcurs cifrele
44
45      rem $s0, $v0, 2           # in $s0 avem result impartirii sumei cifrelor la 2
46      # pentru a decide daca e par sau nu
47
48      seq $v0, $s0, 0           # daca suma e para, in $v0 pune 1, altfel 0
49
50      lw $s1, -16($fp)          # restauram registrii de pe stiva
51      lw $s0, -12($fp)          # adica (s1, s0, ra, fp)
52      lw $ra, -8($fp)
53      lw $fp, -4($fp)
54
55      addu $sp, 16              # da pop la stiva
56
57      jr $ra                  # ieșe din functie
58
59    evaluateaza:
60
61      subu $sp, 4              # punem fp pe stiva
62      sw $fp, 0($sp)          # stiva este $sp: ($fp) (pointer_catre_v) (n) (x) (y) (z)
63
64      addi $fp, $sp, 4           # face fp sa pointeeze catre cadrul nostru de apel
65      # avem $sp: ($fp) <fp_pointeaza_aici> (pointer_catre_v)
66      # (n) (x) (y) (z)
67
68      subu $sp, 4              # punem $ra pe stiva
69      sw $ra, 0($sp)          # stiva este $sp: ($ra) ($fp) (pointer_catre_v) (n) (x)
70
71      subu $sp, 4              # punem $s0 pe stiva
72      sw $s0, 0($sp)          # stiva este $sp: ($s0) ($ra) ($fp) (pointer_catre_v) (n)
73
74      subu $sp, 4              # punе $s1 pe stiva
75      sw $s1, 0($sp)          # stiva este $sp: ($s1) ($s0) ($ra) ($fp) (
76        pointer_catre_v) (n) (x) (y) (z)
77
78      subu $sp, 4              # punе $s2 pe stiva
79      sw $s2, 0($sp)          # stiva este $sp: ($s2) ($s1) ($s0) ($ra) ($fp) (
80        pointer_catre_v) (n) (x) (y) (z)
81
82      subu $sp, 4              # punе $s3 pe stiva

```

```

77    sw $s3, 0($sp)          # stiva este $sp: ($s3) ($s2) ($s1) ($s0) ($ra) ($fp) (
78      pointer_catre_v) (n) (x) (y) (z)
79
80    subu $sp, 4              # pune $s4 pe stiva
81    sw $s4, 0($sp)          # stiva este $sp: ($s4) ($s3) ($s2) ($s1) ($s0) ($ra) (
82      $fp) (pointer_catre_v) (n) (x) (y) (z)
83
84    subu $sp, 4              # pune $s5 pe stiva
85    sw $s5, 0($sp)          # stiva este $sp: ($s5) ($s4) ($s3) ($s2) ($s1) ($s0) (
86      $ra) ($fp) (pointer_catre_v) (n) (x) (y) (z)
87
88    lw $s0, 4($fp)          # in $s0 il vom avea pe n
89    li $s1, 0                # $s1 va fi counterul (adic i din formula)
90    lw $s2, 0($fp)          # $s2 va fi un pointer catre pozitia curenta din vector
91      # nu vom folosi un registru pentru a tine adresa de
92      # memorie
93      # a vectorului si un alt registru pentru counterul din
94      # vector.
95      # in schimb vom avea un signur registru cu adresa de
96      # memorie
97      # din vector in care ne aflam
98
99    li $s4, 0                # $s3 il vom folosi pentru uz general
100   # in $s4 vom acumula suma
101   # $s5 il vom folosi pentru uz general
102
103  parcurg:                 # parcurge vectorul
104
105  beq $s0, $s1, fin_ev
106
107  lw $s3, 0($s2)          # momentan, in $s3 vom tine valoarea lui v[i]
108
109  subu $sp, 4              # punem v[i] pe stiva pentru a apela procedura
110
111  sw $s3, 0($sp)          # stiva: $sp: (v[i]) (
112    celealte_lucruri_care_sunt_deja_pe_stiva)
113
114  jal suma_cifrelor_para  # apelam procedura suma_cifrelor_para
115
116  addu $sp, 4              # ii dam pop lui v[i] de pe stiva.
117
118  beq $v0, 0, ev_add_cond # daca in v0 am valoarea 1 nu ma intereseaza (pentru ca
119    avem (1-1)*ceva in formula,
120    # adica 0*ceva=0)
121
122  j ev_cont_parc
123
124  ev_add_cond:             # daca $v0 are valoarea 0, adunam a doua parte din
125    formula la suma         # adica $s4 += (v[i] mod x + (y - (z div 3) + i)^3)
126
127  lw $s5, 8($fp)          # in $s5 il punem pe x
128  rem $s5, $s3, $s5        # in $s5 punem v[i] mod x (momentan $s3 are tot valoarea
129    v[i])
130
131  add $s4, $s4, $s5        # adunam la suma pe v[i] mod x
132    # deci ne-a ramas de calculat doar (y - (z div 3) + i)^3
133
134  lw $s3, 12($fp)          # in $s3 il punem pe y
135  lw $s5, 16($fp)          # in $s5 il punem pe z
136  div $s5, $s5, 3           # in $s5 punem (z div 3)
137
138  subu $s3, $s3, $s5        # calculam x - (x div 3) si tinem valoarea in $s3

```

```

133      add $s3, $s3, $s1          # adunam i, deci avem  $x - (x \text{ div } 3) + i$  in $s3
134      move $s5, $s3            # copiem aceeasi valoare si in $s5, adica $s5 =  $x - (x \text{ div } 3) + i$ 
135      div $s3, 3
136      mul $s3, $s3, $s3        # in $s3 avem  $(x - (x \text{ div } 3) + i)^2$ 
137      mul $s3, $s3, $s5        # acum in $s3 avem  $(x - (x \text{ div } 3) + i)^3$ 
138      add $s4, $s4, $s3        # adunam  $(x - (x \text{ div } 3) + i)^3$  la suma
139
140      ev_cont_parc:
141
142      addu $s1, 1               # incrementez counterul
143      addu $s2, 4               # incrementez adresa de memorie curenta din vector
144
145      j parcurg                # continui parcurgerea
146
147      fin_ev:
148
149      move $v0, $s4             # returnam prin $v0 rezultatul
150
151      lw $s5, -32($fp)         # restauram toti registrii de care ne-am folosit
152      lw $s4, -28($fp)         # adica ($s0..$s5, $fp, $ra)
153      lw $s3, -24($fp)
154      lw $s2, -20($fp)
155      lw $s1, -16($fp)
156      lw $s0, -12($fp)
157      lw $ra, -8($fp)
158      lw $fp, -4($fp)
159
160      addu $sp, 32             # dam pop de pe stiva acestor registrii
161
162      jr $ra                  # iesim din functie
163
164 main:
165
166      li $v0, 5                # citim n
167      syscall
168      move $t0, $v0
169
170      li $t1, 0
171      li $t2, 0
172
173      read:                   # citim cele n numere din vector
174
175      beq $t1, $t0, exit_read
176
177      li $v0, 5
178      syscall
179      sw $v0, v($t2)
180
181      addu $t1, 1
182      addu $t2, 4
183
184      j read
185
186 exit_read:
187
188      li $v0, 5                # citim x
189      syscall
190      move $t1, $v0
191
192      li $v0, 5                # citim y
193      syscall
194      move $t2, $v0
195
196      li $v0, 5                # citim z
197      syscall
198      move $t3, $v0
199

```

```

200
201    subu $sp, 4          # punem z pe stiva
202    sw $t3, 0($sp)
203
204    subu $sp, 4          # punem y pe stiva
205    sw $t2, 0($sp)
206
207    subu $sp, 4          # punem x pe stiva
208    sw $t1, 0($sp)
209
210    subu $sp, 4          # punem n pe stiva
211    sw $t0, 0($sp)
212
213    subu $sp, 4          # punem pointer catre v pe stiva
214    la $t0, v
215    sw $t0, 0($sp)
216
217
218    # stiva noastra este:
219    jal evaluateaza      # $sp: (pointer_catre_v) (n) (x) (y) (z)
220
221    addu $sp, 20         # dam pop la partea din stiva unde sunt argumentele
222                                # 5 (nr_de argumente) * 4 (sizeof word) bytes = 20
223
224    move $a0, $v0         # afiseaza pe ecran rezolutatul
225    li $v0, 1
226    syscall
227
228    li $v0, 10
229    syscall

```

problema1.macos

## Partea II

1

Apelul de sistem READ STRING afisează un sir de caractere pe ecran. Argumentele sunt: adresa de memorie la care vrem să reținem sirul de caractere, respectiv lungimea maximă a sirului de caractere. Adresa de memorie se pune în registrul \$a0, iar dimensiunea maximă se pune în registrul \$a1. Codul pentru apelul de sistem este 8 și se pune în registrul \$v0. Rezultatul îl vom primi la adresa de memorie pe care i-am dat-o apelului de sistem.

Exemplu:

```

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     # citeste sirul de caractere
10    la $a0, str        # pune in $a0 adresa de memorie la
11        # care vreau sa retin sirul de caractere
12    li $a1, 99          # dimensiunea maxim a sirului
13    li $v0, 8            # codul pentru apelul de sistem este 8
14    syscall
15
16    # afiseaza sirul de caractere

```

```

17      la $a0, str
18      li $v0, 4
19      syscall
20          # exit
21      li $v0, 10
22      syscall

```

readstring.s

La test nu va fi nevoie să dați un exemplu aşa de mare, puteți doar să declarați un sir de caractere și să faceți apelul de sistem, nefiind nevoie să îl și afișați pe ecran. Exemplul acesta este luat cu copy-pasta din Tutoriatul 4.

## 2

Trebuie să aflăm reprezentarea internă a instrucțiunii add \$t0, \$t0, \$s3.

Avem:

op → 000000 (deoarece este instrucțiune din R)  
 rs → \$t0 = \$8 = 01000  
 rt → \$s3 = \$19 = 10011  
 rd → \$t0 = \$8 = 01000  
 shamt → 00000 (nu avem shiftare)  
 func → 100000

Reprezentarea binară este:

0000 0001 0001 0011 0100 0000 0010 0000

Reprezentarea în hexa este:

0x01134020

## 3

Pentru a parcurge un sir de caractere putem avea un contor cu care să iterăm prin caracterele sirului de caractere. Pentru a extrage un caracter se poate folosi lb \$t0, sir(\$t1), unde \$t0 este destinația, \$t1 este contorul, iar *sir* este sirul de caractere sotcat la nivel de memorie.

O altă metodă este să ținem adresa de memorie din sir la poziția curentă, iar pe aceasta să o tot incrementăm, în loc să avem un contor separat. Vom avea la \$t0, sir. Asta ne va da adresa de memorie a sirului de caractere. La fiecare pas putem incrementa această adresă cu 1, iar pentru a extrage caracterul curent putem face lb \$t1 ,0(\$t0), unde \$t1 este destinația.

Cele două metode sunt echivalente întrucât amândouă se rezumă la a salva într-un registru valoarea de la o adresă de memorie. În prima metodă avem un contor relativ la adresa de memorie a sirului, iar în a doua metodă avem adresa directă din memorie către poziția din sir.

Mai multe explicații pe tema asta se găsesc în Tutoriatul 3.

ACEL MOMENT CAND SINGURELE  
MATERIALE PE CARE LE AI LA  
EXAMENUL DE ASC SUNT  
MEME-URILE DIN TUTORIATE



## 1.2 Subiecte date de domnul Drăgulici



**ACEL MOMENT CAND CINEVA TE  
INTREABA PE STRADA DE UNDE SA  
CUMPERE UN COVRIG, IAR TU DIN  
REFLEX CHIAR II SPUI DE UNDE SA  
CUMPERE, CI NU IL PUI SA CAUTE  
BRUTARII**



Exercițiu 1 Procedură ce primește ca parametrii prin stivă adresa unui string și lungimea lui și-i inversează ordinea caracterelor; în acest scop va parcurge stringul cu \$s0 de la început către sfârșit și cu \$s1 de la sfârșit către început și cât timp  $\$s0 < \$s1$  va interschimba caracterele pointate de  $\$s0,\$s1$ . Procedura își va accesa parametrii cu \$fp iar apelurile vor respecta convențiile MIPS și C (privind cadrul de apel, \$fp, regiștrii salvați de apelant și apelat, etc.).

Program care aplică procedura unui string ".asciiz" declarat cu inițializare (se va inversa partea până la

caracterul nul exclusiv) și apoi îl afișează (cu ”syscall”).

Rezolvare:

```
1 .data
2
3     sir:.asciiiz "Va salut , dragi studenti." # sirul declarat in memorie
4
5 .text
6
7 invert:
8
9     subu $sp , 4          # punem $fp pe stiva
10    sw $fp , 0($sp)       # acum stiva este $sp: ($fp) (adresa_sir) (
11        lungime)
12
13    addi $fp , $sp , 4      # face $fp sa pointeeze catre cadrul nostru de
14        apel
15
16    subu $sp , 4          # pune $ra pe stiva
17    sw $ra , 0($sp)       # stiva este: $sp: ($ra) ($fp) (adresa_sir) (
18        lungime)
19
20    subu $sp , 4          # pune $s0 pe stiva
21    sw $s0 , 0($sp)       # stiva este: $sp: ($s0) ($ra) ($fp) (adresa_sir)
22        ) (lungime)
23
24    subu $sp , 4          # pune $s1 pe stiva
25    sw $s1 , 0($sp)       # stiva este: $sp: ($s1) ($s0) ($ra) ($fp) (
26        adresasir) (lungime)
27
28    lw $s0 , 0($fp)       # in $s0 avem adresa catre inceputul sirului de
29        caractere
30    lw $s1 , 4($fp)       # in $s1 avem momentan lungimea sirului de
31        caractere
32
33    add $s1 , $s0 , $s1      # adunam la $s1 valoarea lui $s0 pentru a avea
34        in el un pointer
35
36    subu $s1 , 1           # catre sfarsitul sirului de caractere
37        ajungem la caracterul \0
38
39    inv_loop:
40
41        bge $s0 , $s1 , sf_inv
42            temp $s0<$s1
43
44        # daca $s0>=$s1 se opreste , adica se executa cat
45        # asa cum scrie in enunt
46
47        lb $t0 , 0($s0)
48            pentru interschimbare
49        lb $t1 , 0($s1)
50
51        sb $t0 , 0($s1)
52        sb $t1 , 0($s0)
53
54        addu $s0 , 1
55            de la inceput spre sfarsit)
56        subu $s1 , 1
57            de la sfarsit spre inceput)
```

```

46      j inv_loop          # continua algoritmul de inversare
47
48 sf_inv:
49
50    lw $s1, -16($fp)      stiva
51    lw $s0, -12($fp)      # restaureaza registrii care au fost pusi pe
52    lw $ra, -8($fp)       # adica ($s1, $s0, $ra, $fp)
53    lw $fp, -4($fp)
54
55    addu $sp, 16          # da pop la aceste registrii de pe stiva
56
57    j $ra                # procedura s-a terminat
58
59 main:
60
61    li $t0, 0              # vom afla lungimea sirului in registrul $t0
62
63    lb $t1, sir($t0)      aflarea
64
65    calc_lung:
66
67      beqz $t1, sf_calc_lung
68
69      addu $t0, 1            # daca am ajuns la \0 ne oprim
70      lb $t1, sir($t0)      # incrementeaza contorul
71
72      j calc_lung          # ia un caracter din sir
73
74
75 sf_calc_lung:
76
77      subu $sp, 4            # continua parcurgerea
78      sw $t0, 0($sp)        # pun adresa sirului de caractere pe stiva
79
80      la $t1, sir           # stiva este: $sp: (adresa_sir) (lungime)
81      subu $sp, 4            # apeleaza procedura care inverseaza sirul
82      sw $t1, 0($sp)
83
84      jal invert
85
86      addu $sp, 8            # da pop la argumentele de pe stiva
87
88      la $a0, sir           # afiseaza sirul cu syscall cum cere problema
89      li $v0, 4
90      syscall
91
92      li $v0, 10
93      syscall

```

problema1\_dragus



## References

- [1] Dumitru Daniel Drăgulici. *Curs+Laborator Arhitectura Sistemelor de Calcul*.
- [2] Stan Bianca-Mihaela, Stăncioiu Silviu *Tutoriat 2020, lol*
- [3] Larisa Dumitrache. *Tutoriat 2019*
- [4] Bogdan Macovei. *Laboratoare ASC 2019/ 2020*