

REGIȘTII. UNITATEA ARITMETICO-LOGICĂ. INSTRUCȚIUNI / PSEUDOINSTRUCȚIUNI ARITMETICE ȘI LOGICE

I. Regiștrii

Regiștrii reprezintă locații de memorie situate pe procesor. Aceștia se găsesc în vârful ierarhiei de memorie, fiind accesați cu viteză maximă de către procesor, însă prezintă o capacitate de stocare redusă.

Nu toți regiștrii procesorului sunt accesibili prin limbajul de asamblare, unii dintre aceștia fiind folosiți strict pentru anumite operații (regiștrii speciali). Regiștrii accesibili din limbaj de asamblare sunt de 2 tipuri: regiștri generali (general registers) și regiștri în virgulă mobilă (floating point registers).

1. Regiștrii generali

Intern, regiștrii generali sunt adresați prin coduri de biți. La nivelul limbajului de asamblare, regiștrii sunt referiți prin cod, precedat de semnul \$: \$1, \$17, \$20, etc. În plus, aceștia pot fi accesați printr-un mnemonic unic, care să evidențieze scopul pentru care sunt folosiți (în general prin convenție, în unele cazuri și din constrângeri hardware): \$t1, \$s2, etc.

În QtSpim, se găsesc grupați în fereastra principală, sub denumirea de General Registers:

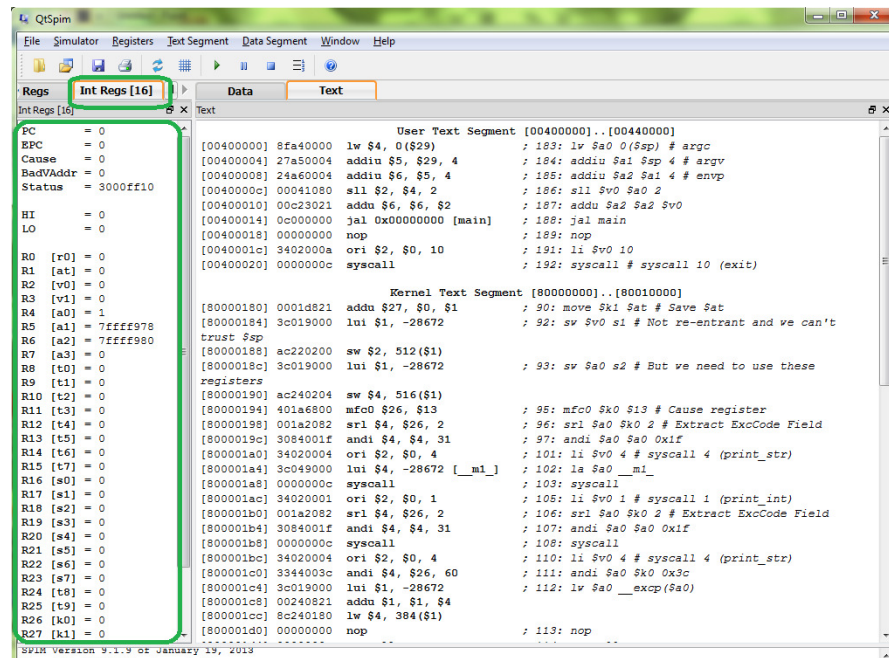


Figura 1. Regiștrii generali

? Întrebări:

- 1) Câți regiștri generali sunt?
- 2) Ce dimensiune are un registru general?
- 3) Câți biți sunt necesari pentru reprezentarea codului unui registru general?
- 4) Cum se reprezintă în binar codul registrul t1? Dar s1?


📖 În tabelul următor sunt prezentați regiștrii generali:

Nume	Mnemonic	Utilizare convențională
\$0	\$zero	Zero register – conține întotdeauna valoarea 0.
\$1	\$at	Assembler Temporary – este rezervat pentru asamblor.
\$2; \$3	\$v0; \$v1	Value registers - folosiți pentru reținerea rezultatelor întregi ale evaluărilor unor expresii sau funcții.
\$4...\$7	\$a0...\$a3	Arguments – folosiți pentru transmiterea argumentelor unor subrutine (parametrii actuali). Valorile acestora nu se păstrează după apelul de procedură.
\$8...\$15	\$t0...\$t7	Temporary registers – folosiți pentru evaluarea expresiilor. Valorile acestora nu se păstrează la apeluri de proceduri.
\$16...\$23	\$s0...\$s7	Saved registers - valorile acestora sunt păstrate la apeluri de proceduri.
\$24; \$25	\$t8; \$t9	Temporary registers – folosiți pentru evaluarea expresiilor. Valorile acestora nu se păstrează la apeluri de proceduri.
\$26; \$27	\$k0; \$k1	Kernel registers - rezervați pentru sistemul de operare.
\$28	\$gp	Global Pointer – indică spre mijlocul unui bloc de memorie care păstrează constante și variabile globale.
\$29	\$sp	Stack Pointer – indică ultima locație utilizată în stivă.
\$30	\$fp	Frame Pointer - pointer spre cadrul curent în stivă.
\$31	\$ra	Return Address – conține adresa de întoarcere, fiind folosit pentru evaluarea expresiilor.

Tabelul 1. Regiștrii generali

📖 Regiștrii \$0 și \$31 sunt singurii diferiți: registrul general \$0 conține întotdeauna valoarea 0, iar registrul general \$31 servește implicit pentru instrucțiunile de salt și de legătură. Pentru toți ceilalți regiștri nu există restricții hardware, însă scopul prezentat este cel destinat utilizării.

2. Regiștrii speciali

 Procesorul MIPS prezintă 3 regiștri speciali, prezenți în partea de sus a ferestrei **Int Regs** a simulatorului QtSpim:

Nume	Descriere
PC	Program Counter
HI	H igher – registru special de înmulțire/împărțire în care se depozitează cei mai semnificativi 32 de biți ai produsului, respectiv restul împărțirii.
LO	L ower - registru special de înmulțire/împărțire în care se depozitează cei mai puțin semnificativi 32 de biți ai produsului, respectiv câtul împărțirii.

Tabelul 2. Regiștrii speciali

 Semnificația EPC, Cause, BadVAddr, Status va fi discutată ulterior, la tratarea excepțiilor.

3. Regiștrii în virgulă mobilă


Regiștrii în virgulă mobilă sunt utilizați de FPU (**F**loating **P**oint **U**nit). Aceștia nu vor fi tratați în cadrul laboratorului.

① Mai multe informații se găsesc la:

http://www.cs.cornell.edu/courses/cs3410/2008fa/mips_voll.pdf

II. Unitatea logico-aritmetică (ALU)

1. Regiștrii și ALU

 Unitatea aritmetico-logică este componenta procesorului care realizează operații aritmetice și logice.

Pentru execuția unei astfel de operații, este nevoie de unul sau mai mulți operanzi (întregi folosiți ca intrare). Aceștia sunt menținuți în regiștri. Ei nu pot fi preluați din memorie și utilizați pentru calcul într-o singură instrucțiune. În cazul în care valorile operanzilor se găsesc în memorie, atunci:

1. Se încarcă datele din memorie în regiștri, utilizând **instrucțiuni de transfer** din memorie în regiștri;
2. Se realizează calculele, utilizând **instrucțiuni aritmetice** sau **logice**;

3. Rezultatul este obținut într-un registru. În cazul în care se dorește salvarea rezultatului în memorie se utilizează o **instrucțiune de transfer** din registru în memorie.

📖 În consecință, pentru execuția unei operații de către ALU este nevoie să se specifice:

- Operația care să se execute;
- 2 operanzi, păstrați de obicei în regiștri;
- 1 registru în care se va stoca rezultatul obținut.

Este posibil ca un operand să nu fie specificat printr-un registru, ci să fie direct indicat de instrucțiunea mașină.

Luând de exemplul instrucțiunea *add \$10,\$8,\$12*

- operația este adunarea;
- locațiile în care se găsesc cei 2 operanzi sunt regiștrii \$8 și \$12;
- registrul în care se va stoca rezultatul este \$10.

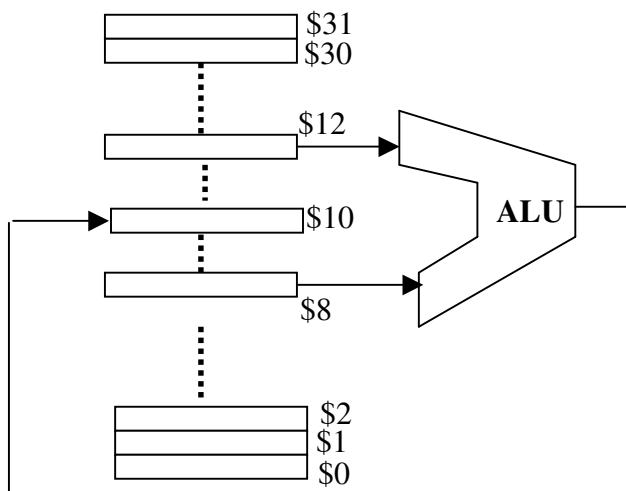


Figura 2. Regiștrii și unitatea aritmetico-logică ALU

2. Formatul instrucțiunilor

📄 **Exercițiu:**

1. Încărcați în QtSpim programul următor:

```
.data
# declaratii date
.text
```

```

# cod
main: # eticheta marcand punctul de start
# cod
add $t1,$t2,$t3
li $v0,10
syscall

```

2. Care este codul mașină corespunzător instrucțiunii *add \$t1,\$t2,\$t3*? Găsiți reprezentarea sa în binar.
3. Adăugați în cod linia următoare: *sub \$t1,\$t2,\$t3*.
4. Care este codul mașină corespunzător acestei instrucțiuni? Găsiți reprezentarea sa în binar.
5. Cum se diferențiază cele 2 operații?
6. Adăugați în cod linia următoare: *add \$s1,\$t2,\$t3*.
7. Care este codul mașină corespunzător acestei instrucțiuni? Găsiți reprezentarea sa în binar
8. Puteți deduce pozițiile biților care indică registrul destinație?
9. Adăugați în cod linia următoare: *add \$t1,\$s2,\$s3*.
10. Care este codul mașină corespunzător acestei instrucțiuni? Găsiți reprezentarea sa în binar.
11. Puteți deduce pozițiile biților care indică regiștrii celor 2 operanzi?

📖 În limbajul de asamblare pentru MIPS32, instrucțiunile ocupă întotdeauna 32 de biți (= 4 octeți = 1 word). Acestea respectă unul dintre următoarele 3 formate:

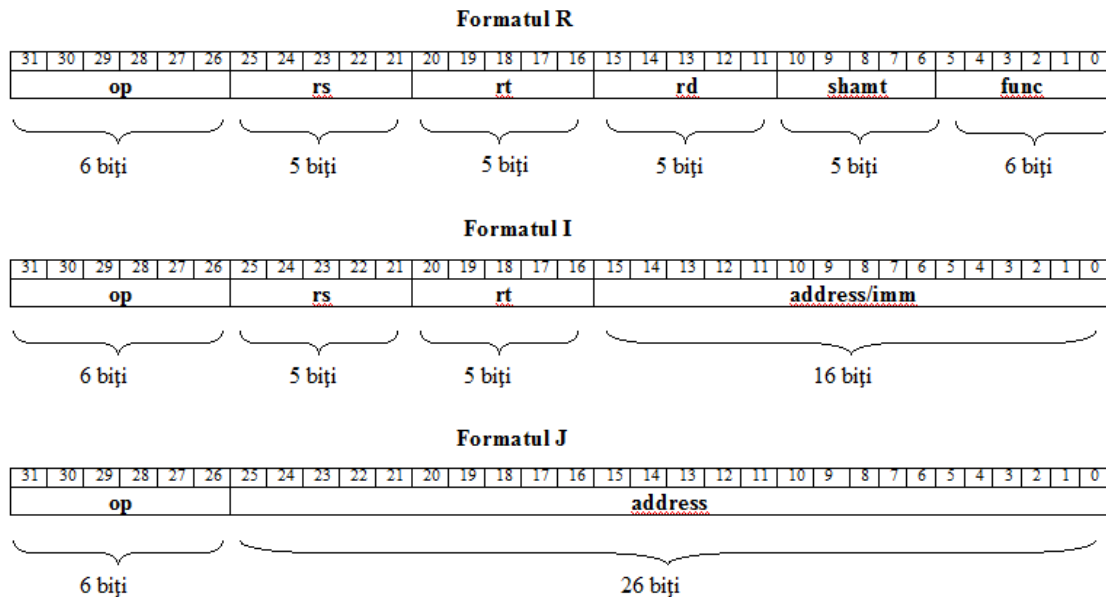



Figura 3. Formatul instrucțiunilor

 Câmpurile au următoarele specificații:


- **op** = operația de bază (opcod)
 - În cazul instrucțiunilor în format R, op este întotdeauna 000000.
 - În cazul instrucțiunilor în format J, op este întotdeauna de forma 00001x, cu x cifră binară;
 - În cazul instrucțiunilor în format I, op diferă, însă nu este niciodată de forma 000000, 00001x sau 0100xx, cu x cifră binară.
- **rs** = registru sursă – registrul care conține primul argument;
- **rt** = registru sursă – registrul care conține al doilea argument (în cazul instrucțiunilor în format R) sau registrul destinație (în cazul instrucțiunilor în format I);
- **rd** = registru destinație – registrul în care se stochează rezultatul obținut în urma operației;
- **shamt** = **shift amount** – folosit la operațiile de deplasare (shiftare);
- **func** = funcția – combinată cu op indică operația/funcția care se aplică;
- **address** = adresă;
- **imm** = valoare imediată.

? Întrebări:

- 1) Ce format respectă instrucțiunile *add* și *sub* folosite în exemplul precedent?
- 2) Completați următorul tabel pentru instrucțiunile folosite în exercițiul precedent:

Instrucțiune	op	rs	rt	rd	shamt	func
<i>add \$t1,\$t2,\$t3</i>	000000	01010	01011	01001	00000	100000
<i>sub \$t1,\$t2,\$t3</i>			01011			
<i>add \$s1,\$t2,\$t3</i>					00000	
<i>add \$t1,\$s2,\$s3</i>		10010				

3. Instrucțiuni cu și fără depășire

 Cum dimensiunea unui registru general este de 32 de biți, există 2^{32} de combinații binare posibile pe care un astfel de registru le poate conține. Operanzii care se păstrează în regiștri se pot considera:

- **Numere naturale, fără semn** : de la 0 la $2^{32}-1$;
- **Numere întregi, cu semn**: de la -2^{31} la $2^{31}-1$.

Instrucțiunile se împart în instrucțiuni care:

- **nu generează niciodată excepție de overflow (fără depășire)**: dacă rezultatul iese din intervalul considerat, se face trunchiere;
- **pot genera excepție de overflow (cu depășire)**: dacă rezultatul iese din intervalul considerat se generează eroare de overflow;

Astfel:

- ***add \$t1,\$t2,\$t3***
 - consideră operanzii din regiștrii \$t2 și \$t3 numere cu semn;
 - întoarce overflow în cazul în care se obține o valoare care nu este în intervalul $[-2^{31}, 2^{31}-1]$ (este o instrucțiune cu depășire);
- ***addu \$t1,\$t2,\$t3***
 - consideră operanzii din regiștrii \$t2 și \$t3 numere fără semn (sufixul **u** provine de la unsigned);
 - nu generează niciodată excepție (overflow), ci trunchiază rezultatul (este instrucțiune fără depășire).
 - rezultatul este corect numai când bitul de transport obținut după efectuarea calculului este 0; în caz contrar s-a realizat trunchiere și rezultatul nu este cel corect aritmetic.



Exercițiu:


1. Încărcați în QtSpim programul de mai jos:

```
.data
# declaratii date
.text
# cod
main: # eticheta marcand punctul de start
# cod
li $t2,0x7fffffff
li $t3,0x00000001
addu $t1,$t2,$t3
add $t0,$t2,$t3
li $v0,10
syscall
```

2. Rulați programul pas cu pas. Observați că:
 - 2.1. Se poate folosi *li* (load immediate) pentru a pune direct într-un registru o valoare.
 - 2.2. În registrul \$t2 se introduce valoarea maximă pozitivă în cazul în care se consideră numere cu semn;
 - 2.3. Instrucțiunea *addu* realizează corect adunarea întrucât nu se depășește valoarea maximă a numerelor fără semn și deci nu se realizează trunchiere.
 - 2.4. Instrucțiunea *add* generează excepție întrucât suma $(2^{31}-1) + 1$ depășește valoarea maximă a numerelor cu semn.
3. Modificați programul de mai sus astfel încât în \$t2 să fie valoarea -2^{31} și în \$t3 valoarea -1 :

```
li $t2,0x80000000
li $t3,0xffffffff
```
4. Rulați programul pas cu pas. Observați că:

- 4.1. Instrucțiunea *addu* nu generează excepție, însă nu calculează corect suma (trunchiază rezultatul);
- 4.2. Instrucțiunea *add* generează excepție, întrucât suma $-2^{31}-1$ iese din intervalul $[-2^{31}, 2^{31}-1]$.


 Instrucțiunile *add* și *addu* (*add unsigned*) utilizează ca operanzi valorile din regiștri. Cei 3 regiștri prezenți în instrucțiune nu trebuie să fie neapărat distincți.

Există operații de adunare care permit adunarea unei valori imediate la o valoare stocată într-un registru. Acestea sunt instrucțiunile:

- ***addi* (*add immediate*): *addi \$t1,\$t2,12***
 - realizează adunare imediată cu semn, cu depășire (generează excepție de overflow când se depășește intervalul $[-2^{31}, 2^{31}-1]$);
 - adună la valoarea dintr-un registru o valoare imediată și stochează rezultatul într-un registru;
- ***addiu* (*add immediate unsigned*): *addiu \$t1,\$t2,12***
 - realizează adunare imediată fără semn, fără depășire (nu generează niciodată excepție de overflow);
 - adună la valoarea dintr-un registru o valoare imediată și stochează rezultatul într-un registru.

Spre deosebire de instrucțiunile *add* și *addu* care sunt în format R, instrucțiunile *addi* și *addiu* sunt în format I.

4. Instrucțiuni aritmetice

 Instrucțiunile aritmetice sunt cuprinse în Tabelul 3. Pentru fiecare dintre ele se specifică dacă poate întoarce sau nu excepție de overflow, modul de realizare al operației, formatul și câte un exemplu.

Instrucțiune	Tip	Operația efectuată	Format	Exemplu
add rd, rs, rt	Cu semn, cu depășire	$rd \leftarrow rs + rt$	R	<i>add \$t1, \$t2, \$t3</i>
addu rd, rs, rt	Fără semn, fără depășire	$rd \leftarrow rs + rt$	R	<i>addu \$t1, \$t2, \$t3</i>
addi rt, rs, imm	Cu semn, cu depășire	$rt \leftarrow rs + imm$	I	<i>addi \$t1, \$t2, 1</i>
addiu rt, rs, imm	Fără semn, fără depășire	$rt \leftarrow rs + imm$	I	<i>addiu \$t1, \$t2, 1</i>
sub rd, rs, rt	Cu semn, cu depășire	$rd \leftarrow rs - rt$	R	<i>sub \$t1, \$t2, \$t3</i>
subu rd, rs, rt	Fără semn, fără depășire	$rd \leftarrow rs - rt$	R	<i>subu \$t1, \$t2, \$t3</i>
mult rs, rt	Cu semn, fără depășire	$HI, LO \leftarrow rs * rt$	R	<i>mult \$t1, \$t2</i>

multu rs, rt	Fără semn, fără depășire	$HI, LO \leftarrow rs * rt$	R	<i>multu \$t1, \$t2</i>
div rs, rt	Cu semn, cu depășire	$LO \leftarrow rs / rt$ $HI \leftarrow rs \% rt$	R	<i>div \$t1, \$t2</i>
divu rs, rt	Fără semn, fără depășire	$LO \leftarrow rs / rt$ $HI \leftarrow rs \% rt$	R	<i>divu \$t1, \$t2</i>

Tabelul 3. Instrucțiuni aritmetice

- ① Formatul exact al instrucțiunilor se găsește la
http://www.cs.sunysb.edu/~cse320/MIPS_Instruction_Coding_With_Hex.pdf

📖 Valorile conținute în regiștri HI și LO nu pot fi direct utilizate în instrucțiuni logice sau aritmetice. Pentru utilizarea informațiilor menținute în HI și LO, acestea trebuie mai întâi mutate în regiștrii generali. Instrucțiunile care permit accesarea informației din cei 2 regiștri speciali sunt următoarele:

Instrucțiune	Abrevieri	Operația efectuată	Format	Exemplu
mthi rs	Move To HI	$HI \leftarrow rs$	R	<i>mthi \$t1</i>
mfhi rd	Move From HI	$rd \leftarrow HI$	R	<i>mfhi \$t1</i>
mtlo rs	Move To LO	$LO \leftarrow rs$	R	<i>mtlo \$t1</i>
mflo rd	Move From LO	$rd \leftarrow LO$	R	<i>mflo \$t1</i>

Tabelul 4. Instrucțiuni pentru utilizarea regiștrilor speciali HI și LO

? Întrebări:

- 1) Observați că spre deosebire de adunare și scădere, înmulțirea utilizează pentru păstrarea rezultatelor 2 regiștri (regiștrii speciali HI și LO). De ce credeți că se întâmplă asta?
- 2) De ce în cazul înmulțirii, nici una dintre instrucțiunile *mult* și *multu* nu întorc excepții de overflow?

📖 Pe lângă instrucțiuni aritmetice, există și pseudoinstrucțiuni. Acestea sunt linii de cod care se convertesc la asamblare în una sau mai multe instrucțiuni. Pseudoinstrucțiunile nu fac parte din ISA. Ele au rolul de a ușura scrierea unui program prin adăugarea unui surplus de claritate.

- ① Pentru mai multe informații despre pseudoinstrucțiuni se poate accesa:
<http://www.cs.umd.edu/class/spring2003/cmsc311/Notes/Mips/pseudo.html>

Pseudoinstrucțiune	Tip	Operația efectuată	Exemplu
abs rd, rs	Cu semn	$rd \leftarrow rs $	<i>abs \$t1, \$t2</i>
neg rd, rs	Cu depășire	$rd \leftarrow -rs$	<i>neg \$t1, \$t2</i>
negu rd, rs	Fără depășire	$rd \leftarrow -rs$	<i>negu \$t1, \$t2</i>
add rd, rs, imm	Cu semn, cu depășire	$rd \leftarrow rs + imm$	<i>add \$t1, \$t2, 1</i>
addu rd, rs, imm	Fără semn, fără depășire	$rd \leftarrow rs + imm$	<i>addu \$t1, \$t2, 1</i>
add rd, imm	Cu semn, cu depășire	$rd \leftarrow rd + imm$	<i>add \$t1, 1</i>
addu rd, imm	Fără semn, fără depășire	$rd \leftarrow rd + imm$	<i>addu \$t1, 1</i>
sub rd, rs, imm	Cu semn, cu depășire	$rd \leftarrow rs - imm$	<i>sub \$t1, \$t2, 1</i>
subu rd, rs, imm	Fără semn, fără depășire	$rd \leftarrow rs - imm$	<i>subu \$t1, \$t2, 1</i>
sub rd, imm	Cu semn, cu depășire	$rd \leftarrow rd - imm$	<i>sub \$t1, 1</i>
subu rd, imm	Fără semn, fără depășire	$rd \leftarrow rd - imm$	<i>subu \$t1, 1</i>
mulo rd, rs, rt	Cu semn, cu depășire	$rd \leftarrow rs * rt$	<i>mulo \$t1, \$t2, \$t3</i>
mulou rd, rs, rt	Fără semn, fără depășire	$rd \leftarrow rs * rt$	<i>mulou \$t1, \$t2, \$t3</i>
div rd, rs, rt	Cu semn, cu depășire	$rd \leftarrow rs / rt$	<i>div \$t1, \$t2, \$t3</i>
divu rd, rs, rt	Fără semn, fără depășire	$rd \leftarrow rs / rt$	<i>div \$t1, \$t2, \$t3</i>
rem rd, rs, rt	Cu semn, cu depășire	$rd \leftarrow rs \% rt$	<i>rem \$t1, \$t2, \$t3</i>
remu rd, rs, rt	Fără semn, fără depășire	$rd \leftarrow rs \% rt$	<i>remu \$t1, \$t2, \$t3</i>

Tabelul 5. Pseudoinstrucțiuni aritmetice



Problemă rezolvată:

Să se calculeze în \$t4 suma valorilor din \$t1 și \$t2 minus valoarea din \$t3.

.data

.text

main:

add \$t4,\$t1,\$t2 # \$t4 = \$t1 + \$t2

sub \$t4,\$t4,\$t3 # \$t4 = \$t4 - \$t3

li \$v0,10

syscall

**Problemă rezolvată:**

Să se obțină în \$t3 expresia $8 \cdot x - [y/16]$, unde x este valoarea din \$t1 și y este valoarea din \$t2.

```
.data
.text
main:
li $t0,8
mulo $t3,$t1,$t0 #se obtine $t3 = 8*$t1
li $t0,16
div $t2, $t2, $t0 #se obtine $t2 = $t2/16
sub $t3, $t3, $t2 #se obtine $t3 = 8*$t1 - [$t2/16]
li $v0,10
syscall
```

**Probleme propuse:**

- 1) Să se obțină în \$t3 valoarea: $[x/y] * \{y/x\}$, unde x este valoare stocată în \$t1 și y este valoarea stocată în \$t2, unde s-a notat cu [] câtul și cu { } restul împărțirii.
- 2) Să se obțină în \$t3 valoarea $|x-y|$, unde x este valoarea din registrul \$t1 și y este valoarea din registrul \$t2.

5. Instrucțiuni logice**? Întrebări:**

- 1) Care sunt operațiile logice pe biți?
- 2) Care sunt tabelele de adevăr corespunzătoare?



Instrucțiunile logice sunt cuprinse în Tabelul 6:

Instrucțiune	Operația efectuată	Format	Exemplu
and rd, rs, rt	$rd \leftarrow rs \text{ AND } rt$	R	<i>and \$t1, \$t2, \$t3</i>
andi rt, rs, imm	$rt \leftarrow rs \text{ AND } imm$	I	<i>andi \$t1, \$t2, 10</i>
or rd, rs, rt	$rd \leftarrow rs \text{ OR } rt$	R	<i>or \$t1, \$t2, \$t3</i>
ori rt, rs, imm	$rt \leftarrow rs \text{ OR } imm$	I	<i>ori \$t1, \$t2, 10</i>
xor rd, rs, rt	$rd \leftarrow rs \text{ XOR } rt$	R	<i>xor \$t1, \$t2, \$t3</i>
xori rt,rs,imm	$rt \leftarrow rs \text{ XOR } imm$	I	<i>xori \$t1, \$t2, 10</i>
nor rd, rs, rt	$rd \leftarrow rs \text{ NOR } rt$	R	<i>nor \$t1, \$t2, \$t3</i>

Tabel 6. Instrucțiuni logice

📖 Întrucât $0 \text{ OR } a = a$, pentru orice a binar, instrucțiunea **ori** se folosește pentru introducerea unei valori într-un registru. De exemplu, pentru introducerea valorii 0x2A în registrul \$t0:

ori \$t0, \$0, 0x2A

Observați utilizarea registrului zero, care conține întotdeauna valoarea 0.

① Formatul exact al instrucțiunilor se găsește la
http://www.cs.sunysb.edu/~cse320/MIPS_Instruction_Coding_With_Hex.pdf

📖 Pseudoinstrucțiunile logice sunt prezentate în tabelul următor:

Pseudoinstrucțiune	Operația efectuată	Exemplu
not rd, rs	$rd \leftarrow \text{NOT } rs$	<i>not \$t1, \$t2</i>
nor rd, rs, imm	$rd \leftarrow rs \text{ NOR } imm$	<i>nor \$t1, \$t2, 10</i>
and rd, rs, imm	$rd \leftarrow rs \text{ AND } imm$	<i>and \$t1, \$t2, 10</i>

Tabelul 7. Pseudoinstrucțiuni logice

📄 Probleme propuse:

- 1) Să se evalueze expresia logică $(x \text{ OR } y) \text{ AND } (\text{NOT } z) \text{ XOR } w$, unde x este valoarea din \$t1, y din \$t2, z din \$t3 și w din \$t4 și să se memoreze rezultatul în \$t5.
- 2) Introduceți în registrul \$t1 valoarea 0x25, folosind numai instrucțiuni logice.

6. Instrucțiuni de shiftare

📖 Instrucțiunile de shiftare sunt prezentate în tabelul următor:

Instrucțiune	Operația efectuată	Format	Exemplu
sll rd, rt, imm	Shift Left Logical $rd \leftarrow rt \ll imm$ - se deplasează biții la stânga cu imm poziții; - biții care ies din word prin stânga se pierd; - locurile goale ramase în dreapta se completează cu 0.	R	<i>sll \$t1, \$t2, 2</i>
srl rd, rt, imm	Shift Right Logical $rd \leftarrow rt \gg imm$	R	<i>srl \$t1, \$t2, 2</i>

	- se deplasează biții la dreapta cu imm poziții; - biții care ies din word prin dreapta se pierd; - locurile goale ramase în stânga se completează cu 0.		
sra rd, rt, imm	Shift Right Arithmetic $rd \leftarrow rt \gg imm$ - se deplasează biții la dreapta cu imm poziții; - biții care ies din word prin dreapta se pierd; - locurile goale ramase în stânga se completează cu 0, cu excepția bitului de semn, care se păstrează.	R	<i>sra \$t1, \$t2, 2</i>
sllv rd, rt, rs	Shift Left Logical Variable $rd \leftarrow rt \ll rs$	R	<i>sllv \$t1, \$t2, \$t3</i>
srlv rd, rt, rs	Shift Right Logical Variable $rd \leftarrow rt \gg rs$	R	<i>srlv \$t1, \$t2, \$t3</i>
srav rd, rt, rs	Shift Right Arithmetic Variable $rd \leftarrow rt \gg rs$	R	<i>srav \$t1, \$t2, \$t3</i>

Tabelul 8. Instrucțiuni de shiftare

① Formatul exact al instrucțiunilor se găsește la
http://www.cs.sunysb.edu/~cse320/MIPS_Instruction_Coding_With_Hex.pdf

? Întrebări:

- 1) Ce operații aritmetice se pot realiza prin operații de shiftare?
- 2) Care este diferența dintre utilizarea unei shiftări logice spre dreapta și a unei shiftări aritmetice spre dreapta în cazul numerelor negative?

7. Pseudoinstrucțiuni de rotație



Pseudoinstrucțiunile de rotație sunt prezentate în tabelul următor:

Pseudoinstrucțiune	Operația efectuată	Exemplu
rol rd, rt, rs	Rotation On Left - se pune în rd configurația de biți din rt deplasată spre stânga cu nr. de biți din rs, a.î. biții care ies din word spre stânga sunt introduși în aceeași ordine în locul gol creat în dreapta.	<i>rol \$t2, \$t1, \$t0</i>

ror rd, rt, rs	Rotation On Right - se pune în rd configurația de biți din rt deplasată spre dreapta cu nr. de biți din rs, a.î. biții care ies din word spre dreapta sunt introduși în aceeași ordine în locul gol creat în stânga.	<i>ror \$t2, \$t1, \$t0</i>
----------------	--	-----------------------------

Tabelul 9. Pseudoinstrucțiuni de rotație

Probleme propuse:

- 1) În registrul t0 se găsește valoarea 0x1234. Obțineți în registrul t2 valoarea 0x91a0 utilizând o instrucțiune de shiftare sau rotație.
- 2) În registrul t0 se găsește valoarea 0x12345678. Obțineți în registrul t1 valoarea 0xc091a2b3 utilizând o instrucțiune de shiftare sau rotație.

① Mai multe informații

Registrii CPU MIPS

<http://www.doc.ic.ac.uk/lab/secondyear/spim/node10.html>

MIPS32™ Architecture For Programmers

Volume I: Introduction to the MIPS32™ Architecture

http://www.cs.cornell.edu/courses/cs3410/2008fa/mips_vol1.pdf

MIPS Assembly Language Programmer's Guide

http://www.cs.unibo.it/~solmi/teaching/arch_2002-2003/AssemblyLanguageProgDoc.pdf

Programmed Introduction to MIPS Assembly Language

<http://chortle.ccsu.edu/AssemblyTutorial/index.html>

MIPS Instruction Coding

<http://www3.cs.stonybrook.edu/~lw/spim/MIPSinstHex.pdf>