INSTRUCŢUNI CONDIŢIONALE, DE RAMIFICARE ŞI SALT. TABLOURI. APELURI SISTEM

1. Instrucţiuni de testare a condiţiei şi mutări condiţionate

🕮 Instrucţiunile de testare a condiţiei şi mutări condiţionate sunt prezentate în tabelul următor:

Instrucţiune	Operaţia efectuată	Format	Exemplu
slt rd, rs, rt	Set On Less Than rd ← (rs < rt) Dacă rs < rt, atunci rd = 1 (true), altfel rd = 0 (false). Valorile sunt considerate cu semn şi comparaţia nu întoarce overflow.	R	*slt $1, $9, $10*
sltu rd, rs, rt	Set On Less Than Unsigned rd ← (rs < rt) Dacă rs < rt, atunci rd = 1 (true), altfel rd = 0 (false). Valorile sunt considerate fără semn şi comparaţia nu întoarce overflow.	R	*sltu $1, $9, $10*
slti rd, rs, imm	Set On Less Than Immediate rd ← (rs < imm) Dacă rs < imm, atunci rd = 1 (true), altfel rd = 0 (false). Valorile sunt considerate cu semn şi comparaţia nu întoarce overflow.	R	*slti $1, $9, 10*
sltiu rd, rs, imm	Set On Less Than Immediate Unsigned rd ← (rs < imm) Dacă rs < imm, atunci rd = 1 (true), altfel rd = 0 (false). Valorile sunt considerate fără semn şi comparaţia nu întoarce overflow.	R	*sltiu $1, $9, 10*
movn rd, rs, rt	Move Conditional on Not Zero Dacă rt ≠ 0, atunci rd ← rs	R	*movn $t1, $t2, $t3*
movz rd, rs, rt	Move Conditional on Zero Dacă rt = 0, atunci rd ← rs	R	*movz $t1, $t2, $t3*

🕮 Pseudiostrucţiunile de ramificare sunt prezentate în tabelul următor:

Pseudoinstrucţiune	**Operaţia efectuată**	**Exemplu**
seq rd, rs, rt	**S**et on **Eq**ual rd ← (rs = rt) Dacă rs = rt, atunci rd = 1 (true), altfel rd = 0 (false).	*seq $t1, $t2, $t3*
sne rd, rs, rt	**S**et **N**ot **E**qual rd ← (rs ≠ rt) Dacă rs ≠ rt, atunci rd = 1 (true), altfel rd = 0 (false).	*sne $t1, $t2, $t3*
sle rd, rs, rt	**S**et on **L**ess Then or **E**qual rd ← (rs ≤ rt) Dacă rs ≤ rt, atunci rd = 1 (true), altfel rd = 0 (false). Valorile sunt considerate cu semn.	*sle $t1, $t2, $t3*
sgt rd, rs, rt	**S**et on **G**reater **T**hen rd ← (rs > rt) Dacă rs > rt, atunci rd = 1 (true), altfel rd = 0 (false). Valorile sunt considerate cu semn.	*sgt $t1, $t2, $t3*
sge rd, rs, rt	**S**et on **G**reater Then or **E**qual rd ← (rs ≥ rt) Dacă rs ≥ rt, atunci rd = 1 (true), altfel rd = 0 (false). Valorile sunt considerate cu semn.	*sge $t1, $t2, $t3*
sleu rd, rs, rt	**S**et on **L**ess Then or **E**qual **U**nsigned rd ← (rs ≤ rt) Dacă rs ≤ rt, atunci rd = 1 (true), altfel rd = 0 (false). Valorile sunt considerate fară semn.	*sleu $t1, $t2, $t3*
sgtu rd, rs, rt	**S**et on **G**reater **T**hen **U**nsigned rd ← (rs > rt) Dacă rs > rt, atunci rd = 1 (true), altfel rd = 0 (false). Valorile sunt considerate fără semn.	*sgtu $t1, $t2, $t3*

sgeu rd, rs, rt	**S**et on **G**reater Then or **E**qual **U**nsigned rd ← (rs ≥ rt) Dacă rs ≥ rt, atunci rd = 1 (true), altfel rd = 0 (false). Valorile sunt considerate fără semn.	*sgeu $t1, $t2, $t3*

2. Instrucţiuni şi pseudoinstrucţiuni de ramificare

📖 Instrucţiunile de ramificare sunt prezentate în tabelul următor:

Instrucţiune	Operaţia efectuată	Format	Exemplu
beq rs, rt, eticheta	**B**ranch on **Eq**ual Dacă rs=rt, atunci salt la etichetă;	I	*beq $t1, $t2, et*
bne rs, rt, eticheta	**B**ranch **N**ot **E**qual Dacă rs ≠ rt, atunci salt la etichetă;	I	*bne $t1, $t2, et*
bgtz rs, eticheta	**B**ranch on **G**reater **T**hen **Z**ero Dacă rs > 0, atunci salt la etichetă	I	*bgtz $t1, et*
bltz rs, eticheta	**B**ranch on **L**ess **T**han **Z**ero Dacă rs < 0, atunci salt la etichetă.	I	*bltz $t1, et*
bgez rs, eticheta	**B**ranch on **G**reater Than or **E**qual to **Z**ero Dacă rs ≥ 0, atunci salt la etichetă.	I	*bgez $t1, et*
blez rs, eticheta	**B**ranch on **L**ess Then or **E**qual to **Z**ero Dacă rs ≤ 0, atunci salt la etichetă.	I	*blez $t1, et*
bltzal rs, eticehta	**B**ranch on **L**ess **T**han **Z**ero and **L**ink Dacă rs < 0, atunci salt la etichetă ca apel_procedură: în registrul $ra ($31) păstrează adresa de întoarcere, adică adresa următoarei instrucţiuni	I	*bltzal rs, eticheta*
bgezal rs, eticheta	**B**ranch on **G**reater Than or **E**qual to **Z**ero **a**nd **L**ink Dacă rs ≥ 0, atunci salt la etichetă ca apel_procedură: în registrul $ra ($31) păstrează adresa de întoarcere, adică adresa următoarei instrucţiuni	I	*bgezal $t0, et*

ⓘ Formatul exact al fiecărei instrucţiuni îl găsiţi la:
http://www.cs.cornell.edu/courses/cs3410/2008fa/MIPS_Vol2.pdf

🕮 Pseudiostrucţiunile de ramificare sunt prezentate în tabelul următor:

Pseudoinstrucţiune	**Operaţia efectuată**	**Exemplu**
b etichetă	**B**ranch Salt la etichetă (salt necondiţionat)	*b et*
beqz rs, eticheta	**B**ranch on **E**qual to **Z**ero Dacă rs = 0, atunci salt	*beqz $t1, et*
bnez rs, etichetă	Branch Not Equal to Zero Dacă rs ≠ 0, atunci salt.	*bnez $t1, et*
blt rs, rt, eticheta	**B**ranch on **L**ess **T**han Dacă rs < rt, atunci salt la etichetă (numerele sunt considerate cu semn)	*blt $t1, $t2, et*
ble rs, rt, eticheta	**B**ranch on **L**ess Then or **E**qual Dacă rs ≤ rt, atunci salt la etichetă (numerele sunt considerate cu semn)	*ble $t1, $t2, et*
bgt rs, rt, eticheta	**B**ranch on **G**reater **T**hen Dacă rs > rt, atunci salt la etichetă (numerele sunt considerate cu semn)	*bgt $t1, $t2, et*
bge rs, rt, eticheta	**B**ranch on **G**reater Then or **E**qual Dacă rs ≥ rt, atunci salt la etichetă (numerele sunt considerate cu semn)	*bge $t1, $t2, et*
bltu rs, rt, eticheta	**B**ranch on **L**ess **T**han **U**nsigned Dacă rs < rt, atunci salt la etichetă (numerele sunt considerate fără semn)	*bltu $t1, $t2, et*
bleu rs, rt, etichetă	**B**ranch on **L**ess Then or **E**qual **U**nsigned Dacă rs ≤ rt, atunci salt la etichetă (numerele sunt considerate fară semn)	*bleu $t1, $t2, et*
bgtu rs, rt, eticheta	**B**ranch on **G**reater **T**hen **U**nsigned Dacă rs > rt, atunci salt la etichetă (numerele sunt considerate fară semn)	*bgtu $t1, $t2, et*

bgeu rs, rt, eticheta	**B**ranch on **G**reater Then or **E**qual **U**nsigned Dacă rs ≥ rt, atunci salt la etichetă (numerele sunt considerate fară semn)	*bgeu $t1, $t2, et*

🖫 **Problemă rezolvată:**

x şi y desemnează 2 numere. Să se păstreze în x valoarea mai mică şi în y valoarea mai mare.

```
.data
    x:.word 5
    y:.word 2
.text
 main:
        lw $t1,x
        lw $t2,y
        blt $t1, $t2, sfarsit
        sw $t2,x
        sw $t1,y
 sfarsit:
    li $v0,10
    syscall
```

a) Rulaţi programul şi observaţi rezultatul.
b) Observaţi că pseudoinstrucţiunea

```
blt $t1, $t2, sfarsit
```

se transformă în 2 instrucţiuni:

```
slt $1, $9, $10
bne $1, $0, 20[sfarsit – 0x00400014]
```

De ce?

3. Instrucţiuni şi pseudoinstrucţiuni de salt

🕮 Instrucţiunile de salt sunt prezentate în tabelul următor:

Instrucţiune	Operaţia efectuată	Format	Exemplu
j eticheta	Jump Salt necondiţionat la etichetă. Nu este PC-relativă, adică în formatul instrucţiunii nu se păstrează deplasamentul	J	*j et*

	faţă de instrucţiunea curentă (ca în cazul instrucţiunilor branch), ci adresa dată de etichetă.		
jal eticheta	Jump and Link Salt la etichetă ca apel_procedură: în registrul $ra ($31) păstrează adresa de întoarcere, adică adresa următoarei instrucţiuni. Nu este PC-relativă, adică în formatul instrucţiunii nu se păstrează deplasamentul faţă de instrucţiunea curentă (ca în cazul instrucţiunilor branch), ci adresa dată de etichetă.	J	*jal et*
jalr <rd,> rs	Jump and Link Register Salt la etichetă cu păstrarea adresei următoarei instrucţiuni în registrul $rd. Dacă lipseşte registrul rd, atunci acesta este considerat implicit registrul ra. Nu este PC-relativă, adică în formatul instrucţiunii nu se păstrează deplasamentul faţă de instrucţiunea curentă (ca în cazul instrucţiunilor branch), ci adresa dată de etichetă.	R	*jalr $t1, $t2*
jr rs	Jump Register Salt la adresa stocată în registrul rs	R	*jr $t1*

🖫 **Probleme propuse:**

1. Determinaţi dacă într-unul din regiştrii $t1 - $t5 se găseşte valoarea 7. Dacă da, introduceţi valoarea 1 în registrul $t0. Dacă nu, introduceţi valoarea 0 în registrul $t0.
2. Verificaţi dacă numărul stocat în registrul $t1 este pozitiv. Dacă da, puneţi 0 în registrul $t2, dacă nu, puneţi 1 în registrul $t2.

4. Tablouri

🕮 Un tablou de numere se defineşte ca o secvenţă succesivă de date stocate în memorie. Matricile se definesc de asemenea ca o secvenţă succesivă.

De exemplu, definirea unui vector de numere întregi se realizează astfel:

```
.data
tablou: .word 1, 2, 3, 4, 5
```

📖 Un şir de caractere se memorează ca un şir de caractere ASCII finalizat cu un byte null (egal cu 0), care reprezintă delimitatorul de şir.

De exemplu, declararea şirului ‚Acesta este un sir de caractere' se realizează astfel:

```
.data
sir: .asciiz "Acesta este un sir de caractere."
```

🖫 **Probleme propuse:**

1. Declaraţi un şir oarecare de caractere. Determinaţi lungimea şirului, prin parcurgerea şi contorizarea locaţiilor de memorie, până când întâlniţi caracterul delimitator de final.
2. Definiţi un vector de numere întregi de lungime cunoscută. Determinaţi suma elementelor pozitive din vector.

5. Apeluri sistem

📖 Apelurile sistem (întreruperile software) reprezintă o interfaţă cu sistemul de operare. În funcţie de anumiţi parametrii setaţi în prealabil, la apelarea unei rutine din sistemul de operare, acesta va executa sarcina indicată.

Un exemplu de apel de sistem pe care l-aţi întâlnit până acum este indicarea sfârşitului de program (oprirea programului):

```
li $v0,10
syscall
```

Valoarea 10 introdusă în registrul v0 este parametrul care indică sistemului de operare cum trebuie să interpreteze apelul sistem invocat de syscall.

📖 Întreruperile pot servi pentru utilizarea unor resurse (citire de la tastatură, afişare pe monitor, etc.), lucru cu fişiere (deschidere, închidere, scriere, etc.), alocare de memorie, ieşire din program, determinarea unor parametrii de sistem (timp), etc. Toate acestea sunt apelate prin syscall, în funcţie de valorile unor regiştrii anterior setaţi. Paşii pentru utilizarea unei întreruperi sistem sunt:

1. Se introduce în $v0 numărul de serviciu corespunzător funcţiei care se doreşte a fi realizată;
2. Se întroduc în regiştrii $a0, $a1, $f12 valorile corespunzătoare parametrilor, dacă acest lucru este necesar;

3. Se apelează instrucţiunea syscall;
4. Se preiau valorile întoarse din regiştrii, dacă este necesar.

Cele mai frecvent utilizate întreruperi sistem se găsesc în tabelul de mai jos:

Serviciul	Codul apelului sistem (introdus în $v0)	Argumente	Rezultate
print integer	1	$a0 = valoare integer care se afişează	-
print float	2	$f12 = valoare float care se afişează	-
print double	3	$f12 = valoare double care se afişează	-
print string	4	$a0 = adresa string-ului care se afişează	-
read integer	5	-	$v0 = valoarea integer citita
read float	6	-	$f0 = valoarea float citita
read double	7	-	$f0 = valoarea double citita
read string	8	$a0 = adresa la care se va memora string-ul $a1 = numărul de caractere care urmează să fie citite + 1	-
memory allocation	9	$a0 = numărul de octeţi de memorie alocaţi	$v0 = adresa blocului de memorie
exit (end of program)	10	-	-

- **Probleme rezolvate:**

1. Să se afişeze la consolă mesajul: „Afisarea unui mesaj”.

```
.data
string1:    .asciiz  "Afisarea unui mesaj.\n"     # declararea sirului de caractere
.text
main:       li $v0, 4              # incarcarea codului apel sistem corespunzator
            la $a0, string1        # incarcarea adresei sirului de caractere
            syscall                # apelul catre sistemul de operare

li $v0,10           #sfarsitul programului; codul apelului sistem de iesire = 10
syscall             #apelul catre sistemul de operare
```

2. Să se păstreze în zona de date, în variabila i1, un număr întreg introdus de la tastatură.

```
.data
i1: .word 0  # definirea unei variabile i1
.text
main:
li      $v0, 5          # incarcarea codului apel sistem corespunzator
syscall                 # apelul catre sistemul de operare
sw      $v0, i1         #valoarea de la tastatura e intoarsa in variabila i1

li $v0,10
syscall
```

3. Să se citească de la tastatură un număr n şi apoi n elemente ale unui vector. Apoi să se afişeze vectorul.

```
.data
string1: .asciiz "Dati numarul de elemente ale vectorului.\n"
string3: .asciiz "Dati elementele vectorului.\n"
string2: .asciiz "Vectorul este:\n"
n: .word 0
elem: .space 64

.text
main:
li      $v0, 4
la $a0,string1
syscall

li      $v0, 5
syscall
sw      $v0, n

lw $t0,n                #memorare nr de elem n in reg t0
li $t1,0
la $t2, elem    #initializare t2 cu adresa la care se memoreaza primul element

li      $v0, 4          #afisare mesaj de introdus elemente
la $a0,string3
syscall

loop:
beq $t0,$t1,end_loop
addi $t1,1              #contorul: cand t1 ajunge la t0 = n atunci iese din bucla
```

```
li      $v0, 5          #in bucla se citeste fiecare element
syscall
sw $v0, ($t2)   #se pune elementul la urmatoarea adresa in zona de date;
addi$t2,4               # adresarea se face din 4 in 4
b loop

end_loop:
li      $v0, 4
la $a0,string2
syscall

li $t1,0
la $t2, elem

loop_afisare:  #afisarea valorilor
beq $t0,$t1,end_loop_afisare
addi $t1,1              #contorul: cand t1 ajunge la t0=n atunci iese din bucla

lw $a0,($t2)    #afisarea valorii de la adresa memorata in t2
li $v0,1
syscall
addi $t2,4
b loop_afisare
end_loop_afisare:

li $v0,10               #oprire executie program
syscall
```

- **Problemă propusă:**

Să se citească şi să se afişeze un vector de n elemente întregi, unde n este introdus de la tastatură. Să se afişeze un meniu cu următoarele opţiuni:

1. Suma elementelor;
2. Ultimul element;
3. Elementele mai mari decât n;
4. Ieşire din program.

La introducerea opţiunii 1, să se afişeze suma elementelor vectorului.
La introducerea opţiunii 2, să se afişeze ultimul element al vectorului;
La introducerea opţiunii 3, să se afişeze elementele mai mari decât n;
La introducerea opţiunii 4, să se oprească execuţia programului.

ⓘ Mai multe informaţii

MIPS32™ Architecture For Programmers -Volume II: The MIPS32™ Instruction Set
http://www.cs.cornell.edu/courses/cs3410/2008fa/MIPS_Vol2.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

MARS - Mips Assembly and Runtime Simulator
http://courses.missouristate.edu/KenVollmar/Mars/Help/SyscallHelp.html