Laborator 2

Lucrarea prezintă modurile de adresare și setul de instrucțiuni (redus) al microprocesorului MIPS.

2.1 Moduri de adresare MIPS

Prin mod de adresare se înțelege metoda de a specifica un operand sau o adresă de memorie.

- **Adresarea imediată:** Operandul se află în corpul instrucțiunii
  De exemplu:
  ```
  addi $t0, $t1, 3 ;; instrucțiunea conține al doilea operand (3)
  and  $v0, 0x34
  ```

- **Adresarea directă:** Adresa operandului (adresa efectivă, AE) se află în corpul instrucțiunii fie sub forma unui cuvânt adresă la memorie, fie sub forma unui cuvânt număr registru din banca de registre (adresă de registru).
  ```
  move $a0, $t2     ;; instrucțiunea conține adresa operanzilor sub forma unor numere de
                    ;; de regiștrii
  ```
  În cazul arhitecturii MIPS, modul de adresare directă în care operanzii sunt valorile conținute în regiștrii se numește **Register Addressing**.

- **Adresarea indirectă:** În corpul instrucțiunii se specifică o adresă intermediară, AI; apoi, la punctul indicat de adresa intermediară se găsește înscrisă adresa efectivă AE, se extrage adresa efectivă (ca pointer) și se citește, de la locul indicat de AE, operandul necesar în instrucțiune.
  Există două variante de adresare indirectă: prin memorie și prin registru
  ```
  jr $31            ;; adresa de memorie la care se produce saltul este conținută în
                    ;; registrul $31

  lw $t1,4($t0)     ;; valoarea încărcată în $t1 se afla la adresa ($t0+4)
  lw $t1,8($t0)
  lw $t1,16($t0)
  ```
  Pentru MIPS acest mod de adresare se numește **Base addressing** și se referă la situația în care adresa de memorie folosită pentru a aduce operandul se calculează prin însumarea unei valori conținute într-un registru și un ofset specificat în instrucțiune ca valoare imediată.

- **Adresare relativă:** Adresa efectivă, AE, se obține prin sumarea la adresa de referință a instrucțiunii (uzual adresa conținută în PC) a unei valori specificată ca un imediat.
  Acest mod de adresare pentru MIPS se numește **PC-relative addressing**
  ```
  beq $t0,$t1,16  ;; dacă regiștrii $t0 și $t1 au valori egale, noua adresă de  program se va
                  ;; calcula PC=(PC+4)+16. PC+4 reprezintă adresa următoarei  instrucțiuni
  ```

Exerciții și întrebări:

1. Cât de eficient este modul de adresare imediată? Prezentați un avantaj și un dezavantaj

2. În ce situații se utilizează adresarea directă? Explicați atât pentru valori prezente în setul de regiștrii cât și în memorie.

3. Când este utilă adresarea indirectă denumită **Base addressing**?

4. Scrieți instrucțiunile pentru încărcarea registrului $s0 cu o constanta pe 16 biti si apoi stocarea registrului $s0 în memorie.

```
.text
.globl main
main:
   li $s0, 17
   lui $t0, 4096
   sw $s0, 0($t0)
```

5. Scrieți instrucțiunile pentru încărcarea registrului $s0 cu constantă 65540 și apoi stocarea registrului $s0 în memorie.

```
.text
.globl main
main:
lui $s0, 1
ori $s0, 4
lui $t0, 4096
sw $s0, 0($t0)
```

6. Încărcați în registrul $s0 și $s1 valorile din variabilele n = 2147483650 și m = 2147483651. Realizați suma celor doi regiștrii.

```
.data
n: .word 0x80000002
m: .word 0x80000003
.text
.globl main
main:
lw $s0, n
lw $s1, m
add $s2, $s0, $s1
```

   Arithmetic overflow, rezultatul nu încape pe 32 de biți. Cum se poate detecta dacă suma a doi regiștrii da arithmetic overflow ?

7. Realizați înmulțirea a doi regiștrii. Unde este stocat rezultatul ?

```
.data
a: .word 5
n: .word 6

.text
```

```
.globl main
main:
   lw $t0, a
   lw $t1, n
   mul $t1, $t0, $t1
   sw $t1, a
   jr $ra
```

8. Realizati in limbaj de asamblare MIPS următoarele instrucțiuni de nivel înalt:

```
if (a0<a1)
   a2 = a0 + 1
else
   a2 = a1 + 1
```

Studiați mai întâi instrucțiunile **beq** si **slt.**

```
.data
a0: .word 10
a1: .word 20
a2: .space 4

.text
   .globl main
main:
   lw $s0, a0
   lw $s1, a1
   beq $s0, $s1, THENIF
   slt $t0, $s0, $s1    #$s0 < $s1 atunci $t0 = 1
                        #$s0 >= $s1 atunci $t0 = 0
   beq $t0, $0, THENIF
   add $s2, $0, $s0
   j ENDIF
THENIF:
   add $s2, $0, $s1
ENDIF:
   addi $s2, $s2, 1
   sw $s2, a2
   jr $ra
```