# Laborator 0x0B

1. Recap

2. Instrucțiuni : LOAD
   STORE
   MOVE

3. Instrucțiuni aritmetice

4. Apeluri de sistem

5. Salturi cond ^ record

6. Tablouri unidimensionale

7. Suma el dintr-un array

9. Proceduri

Recap

Registrii arhitecturii RISC V

→ zero - are mereu val. 0

→ gp - global pointer. Retine adresa de date

→ tp - thread pointer

→ sp - stack pointer

→ s0 - frame pointer (/ alt)

→ ra - return address

→ pc - program counter (/ aip)

→ t0 → t6 reg temp

→ s1 → s11 reg. salvati

→ a0 → a7 reg argumente

a0, a1 => pt return

a7 => pt codurile apelurilor de sistem

Tipuri de date

. byte : 1 B = 8 biti

. halfword : 2 B (16 b)

. word : 4 B (32 b)

. double word : 8 B (64 b)

+ .asciz, .ascii, .space

# Declarări

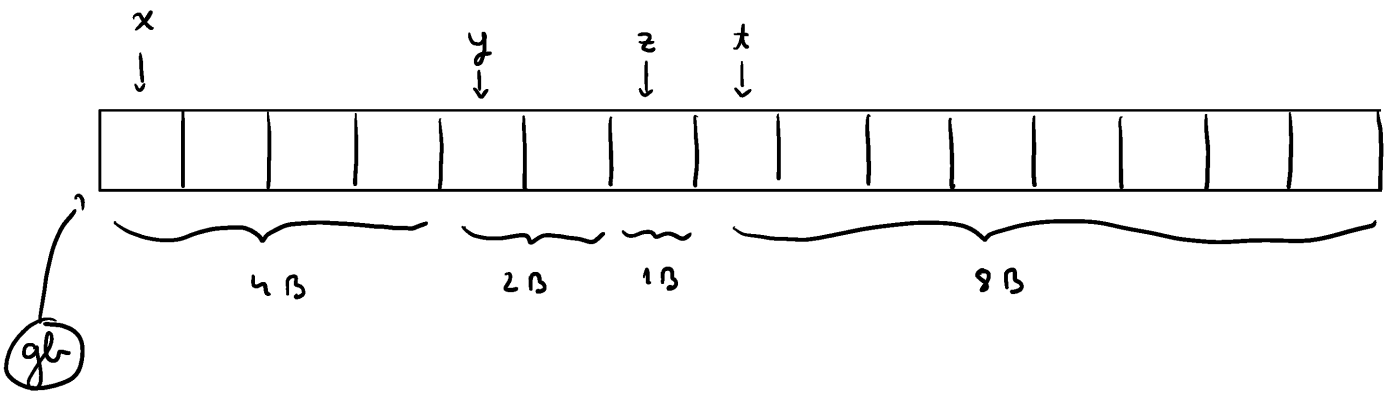
. data

x : . word 15

y : . halfword 255

z : . byte 65

t : . doubleword 2000




x ← 0(gp)

y ← 4(gp)

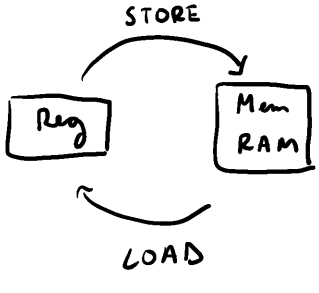
z ← 6(gp)

t ← 7(gp)

## Instrucțiuni:

1) **LOAD** → încarcă o val din RAM în registru

2) **STORE** → încarcă val din reg. în RAM



## 1) LOAD — notat `l` și trebuie sufixat

`lw` , `lb` , `ldw` , `lhw`

`li` = load immediate value
↳ pt constante

`la` = load address
↳ practic lea

Exemple

```
lw   t0, x      // în t0 pune x
lw   t1, y      // în t1 pune y

li   t2, 2
li   t3, 'a'

la   t4, v      // <=> lea v, %edi
```

## 2) STORE

`sw` , `sb` , ~~`sh`~~ , ~~`sd`~~ NU

```
sw   t0, x
```

## 3) MOVE ( ! exclusiv pt regiștii )

`move` registru_dest , reg_sursă

```
move   t2, t1    <=>   addi   t2, t1, 0
```

# Instrucțiuni aritmetice

```
add   rd , rs1 , rs2      =>  rd = rs1 + rs2
sub   rd , rs1 , rs2
mul   rd , rs1 , rs2
div   rd , rs1 , rs2      =>  rd = rs1 / rs2
rem   rd , rs1 , rs2      =>  rd = rs1 % rs2
```

```
addi  rd , rs , imm   ( ultimul op. const. numerică)
addi  t0 , t1 , 1
addi  t0 , t0 , 1
```

# Apeluri de sistem

- codul apelului se retine în a7
- argumentele sunt a0 → a6
- instr. ecall

| | cod | arg |
|---|---|---|
| EXIT : | 93 | 0 |
| PRINT STRING | 4 | $ str |
| PRINT INT | 1 | x |

exit (0) ← arg

```
.data
str : .asciz "RISC V !\n"
x : word 7
```

**exit (0) :**

```
li a7, 93
li a0, 0
ecall
```

**print string :**

```
li a7, 4
li a0, str
ecall
```

**print int :**

```
li a7, 1
lw a0, x
ecall
```

# Salturi

## Saltul necondiționat :

j etichetă  sau (=)  b et  (b = branch)

## Salturi condiționate :

| | | |
|---|---|---|
| < | blt | op1, op2, etichetă |
| ≤ | ble | op1, op2, et |
| > | bgt | op1, op2, et |
| ≥ | bge | op1, op2, et |
| == | beq | op1, op2, et |
| != | bne | op1, op2, et |

Obs Nu mai e pe invers

# Tablouri unidimensionale de date

Array-uri de word-uri

Declarare și inițializare

. data

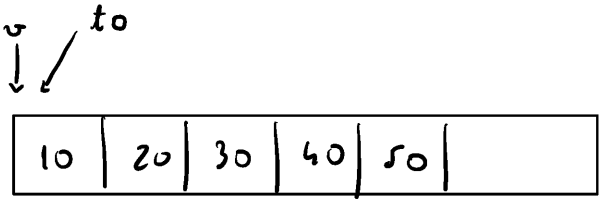
v: . word 10, 20, 30, 40, 50

n: . word 5

Obs In RISC V nu avem op. de diferentiere de forma a(b, c, d) !!!

In schimb, avem a ( b )
- a: constantă numerică
- b: registru

Exemplu:




10 ← 0(to)

20 ← 4(to)

30 ← 8(to)

40 ← 12(to)

50 ← 16(to)

la to, v

Problemă: Dat un array în .data să se calculeze și afișeze suma el.

```
.data
    v: .word 10, 20, 30, 40, 50
    n: .word 5
    rez: .space 4
    str: .asciz "Suma este "
.text
    main:
        la t0, v
        li t1, 0    // index
        li t2, 0    // suma
        lw t3, n
```

```
et_loop

    beq   t1, t3, et_exit      # jump equal

    lw    t4, 0(t0)

    add   t2, t2, t4           # t2 = t2 + t4  <=> sum = sum + v[i]

    addi  t1, t1, 1            // t1 = t1 + 1   <=> index ++

    addi  t0, t0, 4            // t0 = t0 + 4

    j     et_loop

et_exit :

    sw    t2, res              // store dim reg in variável

    li    a7, 4
    la    a0, str              ] ofrecer string
    ecall

    li    a7, 1
    lw    a0, res              ] oferecer int
    ecall

    li    a7, 93
    li    a0, 0                ] exit (0)
    ecall
```

# Proceduri în RISC-V

## Convenții de apel

1. Arg. unei proceduri se încarcă în ordine inversă pe stivă

| Instrucțiunea stivei | x86 | RISC-V |
|---|---|---|
| push op | sub $4, %esp<br>mov op, 0(%esp) | addi sp, sp, -4<br>sw op, 0(sp) |
| pop op | mov 0(%esp), op<br>add $4, %esp | lw op, 0(sp)<br>addi sp, sp, 4 |
| pop | add $4, %esp | addi sp, sp, 4 |

2. Apelul / revenirea din apel sunt controlate prin call / ret

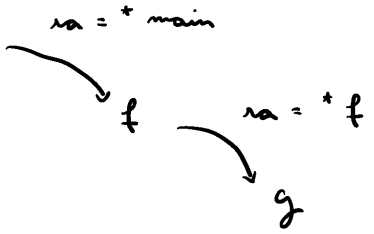
call function => jal funct ≡ jal ra

jal = jump and link → sare la funcție
→ reține adresa de întoarcere în ra

ret => jr ra

jr = jump register → sare la adresa din ra

Obs




Soluția: ra se pune pe stivă

3. Registrii t NU trb restaurați

4. Registrii s (+ ra) trebuie restaurați

5. Se folosește s0 ca frame pointer pt. accesare fixă în cadrul de apel (+ dx)

Problemă: Suma a două nr. prin procedură

- data

  x: .word 15

  y: .word 13

  res: .space 4

- text

main:

lw t0, x

lw t1, y

addi sp, sp, -4
sw t1, 0(sp)
} push t1 (y)

```
addi  sp, sp, -4
sw    t0, 0(sp)        ] push t0 (x)

jal   sum   # call sum

addi  sp, sp, 8    # add $8, to sp

sum :

addi  sp, sp, -4
sw    ra, 0(sp)

addi  sp, sp, -4
sw    s0, 0(sp)
move  s0, sp

[ addi  sp, sp, -4
  sw    s1, 0(sp)
[ addi  sp, sp, -4
  sw    s2, 0(sp)

lw    s1, 8(s0)
lw    s2, 12(s0)
add   a0, s1, s2

lw    s2, -8(s0)
lw    s1, -4(s0)
lw    ra, 4(s0)
lw    s0, 0(s0)

jr    ra    # ret
```

s0 ⟶

| | |
|---|---|
| s2 | -8(s0) |
| s1 | -4(s0) |
| s0 | 0(s0) |
| ra | 4(s0) |
| x | 8(s0) |
| y | 12(s0) |

restaurare codice di ogni