Computer Architecture

DATA: 18/03/2022

EXERCISES FOR THE UPCOMING MIDTERM

Print "Trovato." if the first word (52) is in the array otherwise print "Non Trovato."

- data

```
.word 52
.word 5                    # length array
.word 71, -12, 0, 52, 8    # array
.asciz "Trovato."
.asciz "Non trovato"
```

- text

```
lui  s0, 0x10010     # store 1st address
lw   t0, 0(s0)       # load 52 in t0
lw   t1, 4(s0)       # "  5  " t1 (length)
addi t2, s0, 8       # address 1st el. of array
ciclo :
    lw  t3, 0(t2)        # loads 1st el. of array
    bne t3, t0, salta    # if value ≠ 52, salta
    beq zero, zero, trovato   # go to "trovato"
salta : addi t2, t2, 4       # increment address of 4
        addi t1, t1, -1      # decrement length array
        bne  t1, zero, ciclo # if len(array) ≠ 0, ciclo
        addi a0, s0, 0x24    # take address of "Non Trovato"
        li a7, 4             # print string
        ecall
        li a7, 10            # exit
        ecall
trovato : addi a0, s0, 0x1C  # take address of "Trovato"
          li a7, 4           # print str
          ecall
          li a7, 10          # exit
          ecall
```

(0x1C: HEX)

One thing we can do when declaring ".text" in RARS is adding labels.

EXAMPLE:

```
.text
n: .word 52
```

"n" is a label and it will contain the address where 52 is stored

EXAMPLE WITH LABELS

```
.data
x:  .word  52
n:  .word  5
v:  .word  71, -12, 0, 52, 8
s1: .asciz "Trovato."
s2  .asciz "Non trovato"

.text
    lw  t0, x
    lw  t1, n
    la  t2, v
```

This is actually divided into 2 instructions:

```
lui  x, 0x10010
lw   t0, 0(x)
```

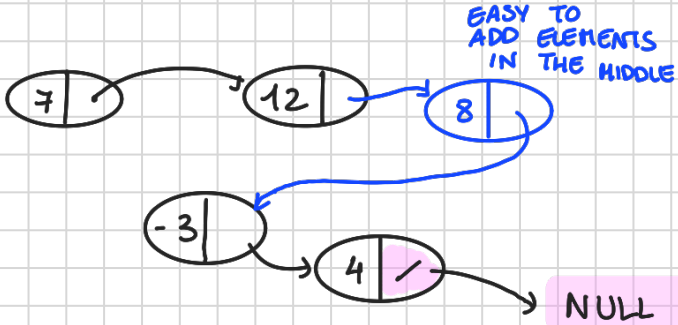
It's just easier for us (visually)

LOAD ADDRESS

```
la a0, s2
li a7, 4
ecall
li a7, 10
ecall
```

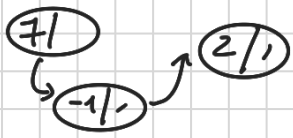
load address is another pseudo-instruction. It loads the address in t2.
(it actually executes 2 instructions but i forgot to write them LOL)

LINKED LISTS → MADE OF NODES. A node contains an information and the address of the next node (POINTER)



NULL POINTER.
IF it's zero, that means the linked list is finished

EXAMPLE




```
.data
```

YOU CAN WRITE IN 2 WAYS

+4 +4

```
. word  7, 0x10010010
. word  2, 0x00000000
. word -1, 0x10010008
```

it's not important to put them in the right order as the drawing

```
n01  . word  7,  n03
n02  .   =   2,  0
n03  .   =   -1, n02
```

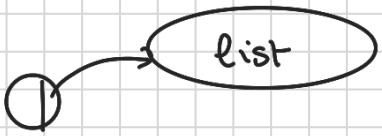
Print the number of zeros in array n

```
.data
n:  .word 4
a:  .word 6,5,0,8

.text
        lw  t0, n          #  array length
        la  t1, a          #  take address of 1st el. of array
        li  a0, 0          #  initialise a counter
ciclo:  lw  t2, 0(t1)      #  load content of array
        bne t2, zero, niente   #  check if its equal to zero

        addi a0, a0, 1     #  increment counter

niente: addi t1, t1, 4     #  //  address array
        addi t0, t0, -1    #  decrement length of array
        bne t0, zero, ciclo    #  check if len(array) = 0

        li a7, 1           # print integer
        ecall
        li a7, 10          #  exit
        ecall
```

SUM ELEMENTS OF A LINKED LIST

```
.data
lista:  word  n01      #  head of linked list
n01 : . word 6, n02
n04 : . word 5, 0
n02 : . word -2, n03
n03 : . word 0, n04

.text
        li  a0, 0
        lw  t0, lista      or la t0, n01
                           # counter
                           # i'm putting content of "lista" so
                             the address of the head
ciclo : lw t1, 0(t0)       # load information of node 1
        add a0, a0, t1     #  increment counter with list element
        lw t0, 4(t0)       #  loading address of next node
        bne t0, zero , ciclo   #  if address ≠ zero , means list didn't end
        li a7, 1           #  print int
        ecall
        li a7, 10          #  exit
        ecall
```

RECURSIVE VERSION



ALWAYS LOOK FOR BASE CASE
↳ if list empty

```
f:          bne   a0, zero, difficile     # if list empty
            jalr  zero, ra, 0             # go back to main

difficile:  addi  sp, sp, -8              #  make space in the stack
            sw    ra, 0(sp)               # save main address
            sw    a0, 4(sp)               #  =  n01 node
            lw    a0, 4(a0)               # points to next address
            jal   f                       # f works for smaller cases
            lw    t0, 4(sp)               #  pointer at head (address)
            lw    t0, 0(t0)               # load content at head
            add   a0, a0, t0              # sum head to rest
            lw    ra, 0(sp)               # take the address ra from stack
            addi  sp, sp, 8               # clean your stack
            jalr  zero, ra, 0             # go back to main
```

. data

```
a:  .word  5, 6, 1, 3, 2, 7     } array
n:  .word  6
x:  .word  3
```

Print the number of item in "a" smaller than "x"

. text

```
            lui   t0, 0x10010     # load in t0 first address (tradition)
            lw    t1, n           #  =  in t1 the length array (6)
            lw    t2, x           #  =  =  t2 the comparison nº 3
            li    a0, 0           # initialize counter
ciclo:
            lw    t3, 0(t0)       # load element of array
            bgt   t3, t2, salta   # if element ≥ 3, skip
            addi  a0, a0, 1       # otherwise, increment counter

salta:      addi  t0, t0, 4       # increment address of array
            addi  t1, t1, -1      # decrement length of array
            bne   t1, zero, ciclo # if len(array) ≠ 0, keep looping

            li a7, 1              # print int
            ecall
            li a7, 10             # exit
            ecall
```

Print the length of the list

```
.data
lista: .word n01          # if we put 0, means empty list
n01 : .word -5, n02

.text
    li   a0, 0            # counter
    lw   t0, lista        # we need the head
    beq  t0, zero, vuota  # if the address is zero, list empty

ciclo: add, a0, a0, 1     # increment counter
    lw t0, 4(t0)          # load next address of the node
    bne t0, zero, ciclo   # if address ≠ 0, keep loop *
    li a7, 1              # print int
    ecall
    li a7, 10             # exit
    ecall
```

* NOTE: when the next address of a node is zero, it means the list ended

---

RECURSIVE VERSION FOR THE PREVIOUS EXERCISE

```
f:   bne  a0, zero, ric   # check if empty
     jalr zero, ra, 0     # a0 already has 0 so go to main

ric: addi sp, sp, -4      # make space
     sw   ra, 0(sp)       # save main address
     lw   a0, 4(a0)       # load next address
     jal f                # recursion
     addi a0, a0, 1       # increment counter
     lw ra, 0(sp)         # take your main address
     addi sp, sp, 4       # clean stack
     jalr zero, ra, 0     # go to main
```