

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
.word $1, -12, 0, 52, 8      # length array
    "Trovato."
.asciz "Non trovato"
```

• text

```
lw $0, 0($10)      # store 1st address
lw $0, 0($0)        # load 52 in t0
lw $1, 4($0)        # = 5 " t1 (length)
add $2, $0, 8        # address 1st el. of array
```

ciclo :

```
lw $3, 0($2)        # loads 1st el. of array
bne $3, $0, salta   # if value ≠ 52, salta
beq zero, zero, trovato # go to "trovato"
```

salta :

```
add $2, $2, 4          # increment address of 4
add $1, $1, -1         # decrement length array
bne $1, zero, ciclo   # if len(array) ≠ 0, ciclo
add $0, $0, 0x24        # take address of "Non Trovato"
li $7, 4                # print string
ecall
li $7, 10               # exit
ecall
```

HEX

trovato :

```
aodil $0, $0, 0x1c      # take address of "Trovato"
li $7, 4                  # print str
ecall
li $7, 10                 # exit
ecall
```

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: . ascii "Trovato."
s2: . ascii "Non trovato"
```

.text

LOAD ADDRESS la a0, s2 li a7, 4 ecall li a7, 10 ecall	EW t0, x EW t1, n ea t2, v	→	
---	---	----------	--

This is actually divided into 2 instructions:

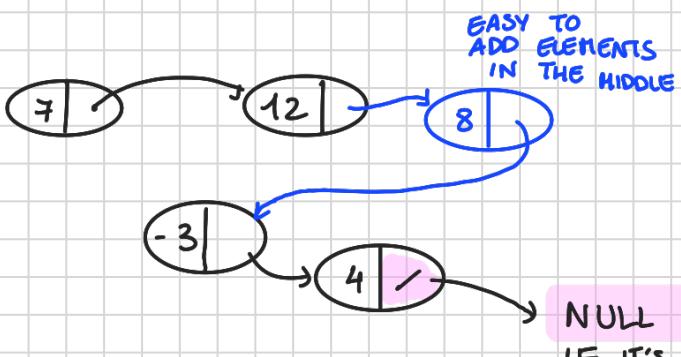
ewi x, 0x10010
ew t0, a(x)

It's just easier for us (visually)

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

MAPS 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

.data

YOU CAN WRITE IN 2 WAYS

+4 +4
 . Word 7, 0x10010010
 . word 2, 0x00000000
 . word -1, 0x10010008

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

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

EXAMPLE



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  
  
add a0, a0, 1      # increment counter  
  
niente: add t1, t1, 4      # == address array  
add t0, t0, -1      # decrement length of array  
bne t0, zero, ciclo # check if len(array) = 0  
  
li a7, 1          # print integer  
ecall             # exit  
li a7, 10         # exit  
ecall
```

SUM ELEMENTS OF A
LINKED LIST

.data

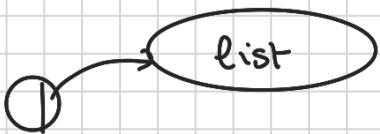
```
lista: word n01      # head of linked list  
n01: .word 6, n02  
n04: .word 5, 0  
n02: .word -2, n03  
n03: .word 0, n04
```

.text

```
li a0, 0  
lw t0, lista  
  
ciclo: lw t1, 0(t0)  
add a0, a0, t1  
lw t0, 4(t0)  
bne t0, zero, ciclo  
li a7, 1  
ecall  
li a7, 10  
ecall
```

or la t0, n01
counter
i'm putting content of "lista" so
the address of the head
load information of node 1
increment counter with list element
loading address of next node
if address/zero, means list didn't end
print int
exit

RECURSIVE VERSION



ALWAYS LOOK
FOR BASE CASE
↳ if list empty

f: bne a0, zero, difficile
jalr zero, ra, 0

difficile : addi sp, sp, -8
sw ra, 0(sp)
sw a0, 4(sp)
lw a0, 4(a0)
jal f
lw t0, 4(sp)
lw t0, 0(t0)
add a0, a0, t0
lw ra, 0(sp)
addi sp, sp, 8
jalr zero, ra, 0

if list empty
go back to main

move space in the stack
save main address
= n01 node
points to next address
f works for smaller cases
pointer at head (address)
load content at head
sum head to rest
take the address ra from stack
clean your stack
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 n <= 3
li a0, 0 # initialize counter

ciclo:

lw t3, 0(t0) # load element of array
bgt t3, t2, saleta # if element ≥ 3 , skip
addi a0, a0, 1 # otherwise, increment counter

saleta: addi t0, t0, 4
addi t1, t1, -1
bne t1, zero, ciclo

increment address of array
decrement length of array
if len(array) $\neq 0$, keep looping

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

.data

lista: .word n01
 n01 : .word -5, n02

If we put 0
 means empty
 list

Print the length
 of the list

.text

li a0, 0
 lw t0, lista
 beq t0, zero, vuota

counter
we need the head
if the address is zero,
list empty

ciclo: add, a0, a0, 1
 lw t0, 4(t0)
 bne t0, zero, ciclo
 li a7, 1
 ecall
 li a7, 10
 ecall

increment counter
load next address of the node
if address ≠ 0, keep loop *
print int
exit

RECURSIVE VERSION FOR THE PREVIOUS EXERCISE

f: bne a0, zero, ric
 jalr zero, ra, 0

check if empty
a0 already has 0 so go to main

ric: addi sp, sp, -4
 sw ra, 0(sp)
 lw a0, 4(a0)
 jal f
 addi a0, a0, 1
 lw ra, 0(sp)
 addi sp, sp, 4
 jalr zero, ra, 0

make space
save main address
load next address
recursion
increment counter
take your main address
clean stack
* go to main