Computer Architecture (Unit 2)

DATA: 9/03/2022

REGISTERS

In RISC-V, we have 32 registers that go from 0 to 31. Those Registers are encoded in the RARS simulator. The most relevant ones are listed below.

REGISTER	RARS ENC.
0	x0 or zero
5 to 7	x5, x7 or t0 - t2
10, 11	x10, x11 or a0, a1
17	x17 or a7

SYSTEM CALLS

You call a system call with the instruction "ecall".

a7=10 } exit()
ecall }

a7 = 1 → print_int(a0)

a7 = 4 → print_string(a0)

NOTE: put "ecall" after every system call

NOTE: you have to put the address inside a0

INSTRUCTIONS: lui, add, addi, lw, sw

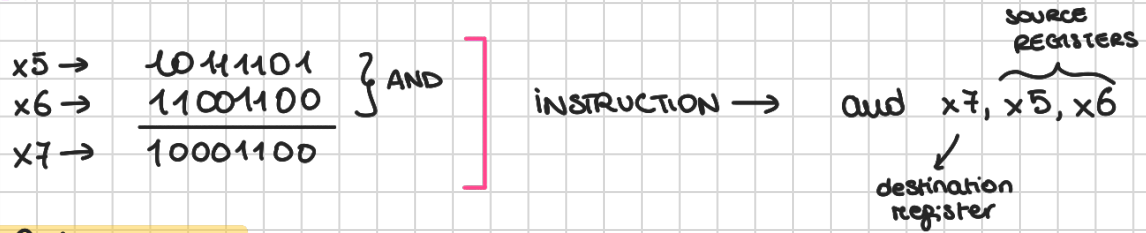
TO WRITE A PROGRAM

```
.data                # data segment starts at 0x10010000
.asciz "Ciao Mondo!"   # to put a string
.text                  # 0x00400000
    lui a0, 0x10010    # putting address inside a0
    addi a7, zero, 4   # print string
    ecall              --> use ecall to say the instruction before "print"
    addi a7, zero, 10  } exit
    ecall
```

AND instruction

Boolean function AND. Performs the AND function bit by bit between the content of two registers and stores in another reg.

EX:



ANDi instr.

Performs the AND function between the content of a register and a number

Example : andi x7, x5, 0x01

OR instr.

Perform the OR function between the content of two registers and stores in another

Example : or x8, x4, x5

ORi instr

Performs the OR function between the content of a register and a number

Example : ori x8, x4, 0x04

XOR instr.

Perform the XOR function between the content of two registers and stores in another

Example : xor x5, x6, x7

XORi instr.

Performs the XOR function between the content of a register and a number

Example : xori x8, x6, 0x02

PROGRAM #1 (Print "pari" if 7 is even, otherwise "dispari")

- data

```
.word 7
.asciz "Pari"
.asciz "Dispari"
```

→ 5 bytes (4 bytes "pari" + 1 byte for putting zero to specify that the string ended)

- text

* you can use HEX or DEC

```
lui t0, 0x10010
lw t1, 0(t0)
andi t1, t1, 0x01*
beq t1, zero, epari
addi a0, t0, 9
addi a7, zero, 4
ecall
beq zero, zero, uscita
```

puts address of 7 in t0
7 goes in t1
to determine if its even or odd

DISPARI BRANCH

addi a0, t0, 9 → it's better to use "ori" but it does the same thing (ori a0,t0,9)

addi a7, zero, 4 → print string

epari:

```
ori a0, t0, 4
addi a7, zero, 4
ecall
```

(we could also use addi)
putting address of string "pari"
print string

uscita:

```
addi a7, zero, 10
ecall
```

} EXIT

PROGRAM #2 (sum up elements of the array and store in a register)

A vector is organized one word/number after the other.

- data

```
.word 5
.word 5, -2, 7, 9, 8
```

→ length array
→ array

- text

```
li a0, 0
lui t0, 0x10010
lw t1, 0(t0)
addi t2, t0, 4
ciclo: lw t0, 0(t2)
add a0, a0, t0
addi t2, t2, 4
addi t1, t1, -1
bne t1, zero, ciclo
```

we could use ori

(we could use add, addi etc.)
variable to store sum
address length
loads 5 in t1 (the length value 5)
address of the 1st element of array
loads 5 in t0 (we overwrote)
sum element
incrementing address next element
decrement of length

```
li  a7, 1          # print integer
ecall
li  a7, 10         } EXIT
ecall
```

LOAD IMMEDIATE (Li)

This is a pseudo-instruction. In assembly language, this "instruction" chooses another instruction out of the several ones present and performs that.
It's useful simply because it's easier to read.