Registers

we'll be using the following registers in the next lectures.

0	x_0, zero
5-6-7	x_5, x_6, x_7, t_0, t_1, t_2
10-11	x_{10}, x_{11}, a_0, a_1
17	x_{17}, a_7

t = temporary → used to store temporary stuff

System calls

ecall ← and the control goes to the O.S.

if we write:

ecall and $a_7 = 10$ — then the O.S. exit()

$a_7 = 1$ — then // print_int(a_0)

$a_7 = 4$ — then // print_string(a_7)

↑

a_7 doesn't contain a string, but an address.

- lui = lower upper immediate
- add = add
- addi = add immediate
- lw = load word
- sw = store word

assembly instructions

.data # the data segment starts at 0x10010000

.asciiz # "Ciao mondo!" — the "z" means "remember to add a zero"...

.text # 0x00400000

lui a_0, 0x10010000 this instruction stores into the register a_0 an address.

addi a_7, zero, (4) — (10)
ecall

ecall

Now, before doing another program, let's learn other instructions.

and x_7, x_5, x_6

↑ logic operator

x_5 10111101
11001100
10001100

this instruction does the AND between 2 values and puts the result in a destination register.

this should be a 32 bits value, but it's just to make an example.

andi x_7, x_5, 0x01
or x_8, x_4, x_5
ori x_8, x_4, x_5
xor
xori

"The encoding of these instructions is written on the green sheet"

Let's see a program that has to check if a number is even or odd.

⇓

```
.data
.word 7
.asciz "pari"
.asciz "dispari"
```

Possible solution:

question: how can I understand if "7" is even or odd?
answer: Just check the last bit. 0 = even 1 = odd.
By using the andi with the number 1 in binary:

111 andi
001
} nice trick to do the Job

lui t_0, 0x10010000 (the address of 7)
lw t_1, 0(t_0) — we load in t_1 the value of the address contained in t_0.

andi $t_1, t_1, 1$ —— this does the AND between the value contained in t_1 with the value 1.

beq t_1, zero, epari

addi $a_0, t_0, 9$ — instead of addi, ori would be OK.
addi a_7, zero, 4
ecall ~ and we're done
beq zero, zero, uscita — a trick to exit the program

epari: ori $a_0, t_0, 4$
addi a_7, zero, 4
ecall
} with these instructions we print the "epari" string

uscita: addi a_7, zero, t_0
ecall
} with this instruction we close the program

Let's try to do something even more complicated:

the difficulty of the following exercise is similar to the mid-term exam.
The following exercise consists in a loop.

In memory, an array consists in a word (4 bytes), after another.

```
.word 5   # we declare the length of the array.
.word 5, -2, 7, 9, 8   # we declare the elements of the array.
.text   # let's write a loop that sums the elements
        of the array.
```

To write this loop we have to understand when the loop has to finish.

We have to use a register which job is to count the number of iterations, and we need to know the address of each of the array.

1st — we decide to use the register a_0 to store the sum.

2nd — we have to initialize a_0 to zero. We can do that in multiple ways.

```
addi  a0, zero, 0
add   a0, zero, zero
ori   a0, zero, 0      <- let's choose this one
or    a0, zero, zero
and   a0, zero, zero
andi  a0, zero, 0
```

But there exists also another way to do that, but not with an instruction.

li a_0, 0 — li = load immediate / pseudo instruction. When we use li, the system choose automatically an instruction.

lui t_0, 0x10010000 — lui = lower upper immediate. With this instruction we store in t_0 the address of the length of the array.

lw t_1, 0 (t_0) — lw = load word
With this instruction, we read the address stored in t_0, then we go into the RAM at that address, we take the content and we move it into the register t_1. In this, the content stored at that address is 5, so, the value of t_1 is now 5.

addi t_2, t_0, 4 — addi = add immediate
With this instruction we read the address stored in t_0, we add 4 to that address and we store that address in t_2.

lw t_0, 0 (t_2) — lw = load word
ciclo
With this instruction we read the address stored in t_2,

we go into the RAM, we take the first element of the array and we load it into t_0.

`add` a_0, a_0, t_0 — With this instruction we sum the content of a_0 with the content stored in t_0, and we store the result in a_0.

`addi` $t_2, t_2, 4$ — With this instruction we iterate on the 2nd element of the array, by reading the address stored in t_2.

`addi` $t_1, t_1, -1$ — With this instruction we're decreasing the length of the loop.

`bne` t_1, zero, ciclo — With this instruction we check if t_1 is zero. If it's not, then the loop continues.

`li` $a_7, 1$ — With this system call, we print the value stored into a_7.

`ecall`

`li` $a_7, 10$ — With this system call we print the value stored in a_7.

`ecall` — We use the ecalls to specify that the previous instruction is a system call.