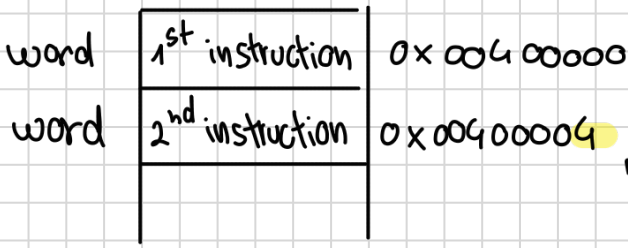


← the program starts at this address

4 bytes increment (1 word = 4 bytes)

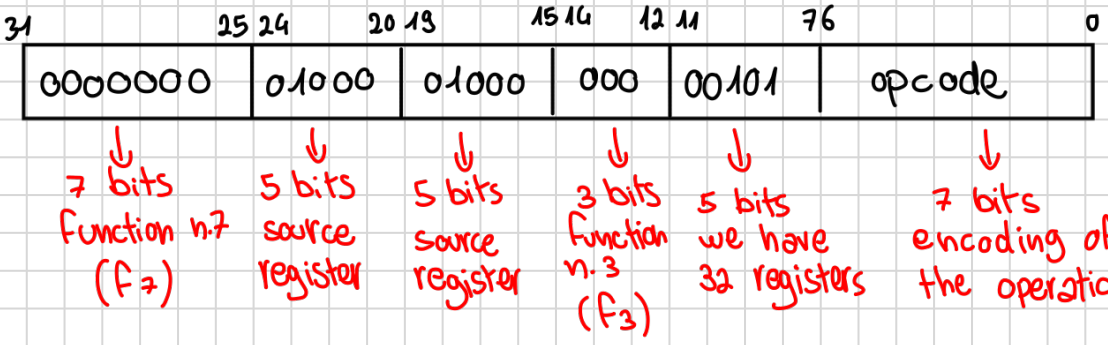
Let's introduce some possible operations:

- add
- addi (add immediate)
- sub
- lw (load word)
- lui (lower upper immediate)
- sw (store word)
- beq (branch on equal)
- bne (branch on not equal)

ASSEMBLY OPERATION (arithmetic):

- the most famous is "add"

How to encode these instructions?



Example: R28 + R29 + R30 → R5
rs rs rs rd

add 6, 28, 29

R6 is the "parking" register

```
add   5, 6, 30
```

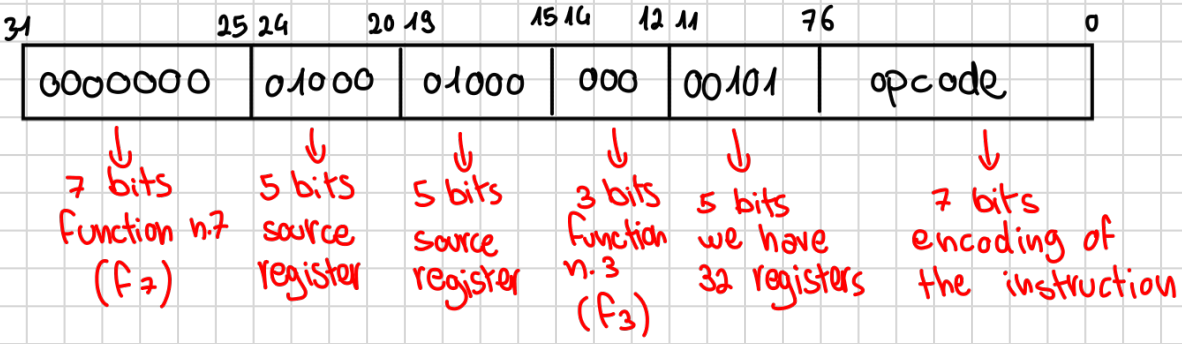
Another way to do this sum would be:

```
add   5, 28, 29
add   5, 5, 30
```

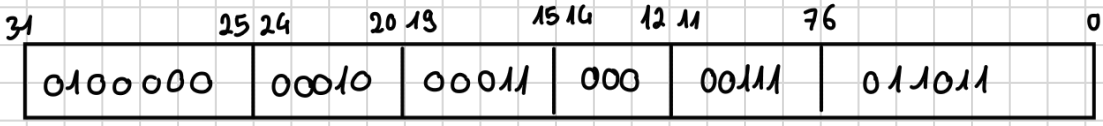
Another way (but in this way I'd loose one of the operands):

~~add 28, 29, 30~~
~~add 5, 28, 0~~

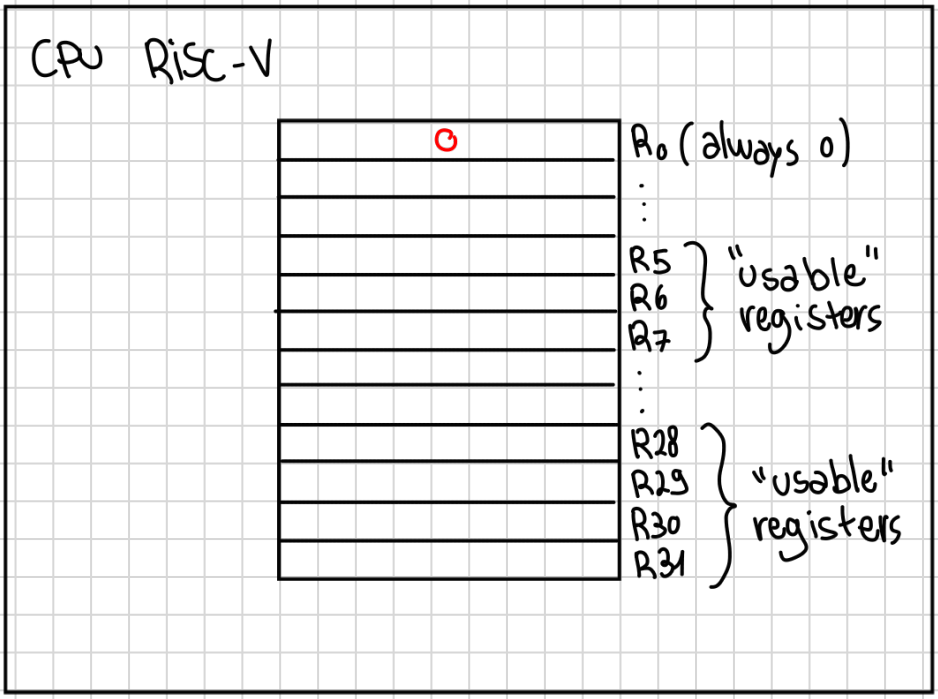
Example: `add 5,8,8`



Example 2: `sub 7,3,2`



f3 and f7 change according to the instruction.



* Some of the registers are used for specific purposes

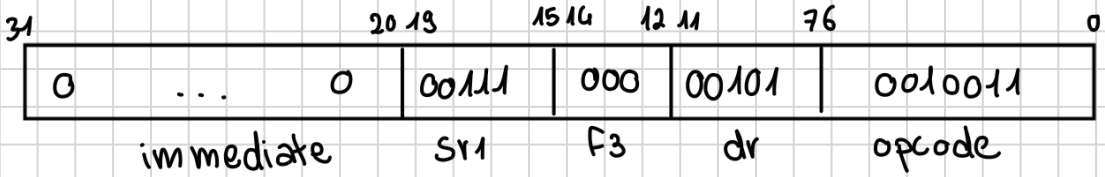
ASSEMBLY OPERATION:

- addi (add immediate)

This operation allows to put small numbers inside the registers.

Example: addi 5, 7, 1 R5 ← R7 + 1

with this we add 1 increment



12 bits

$[-2^{11}; 2^{11}-1]$

~ With the addi operation I can also do the move operation.

Example: addi 5, 7, 0 R5 ← R7 + 0

the subi operation doesn't exist because it is an addi with a negative number.

N.B. If the operands are larger than 12 bits, then the sub operation is required !!!

ASSEMBLY OPERATION:

- lw (load word)

this is an operation that is very important because allows us to take stuff from the memory (RAM); read the memory and load the content into a register (CPU).

Example: I want to take something from a certain address of the memory and move the content inside the R5.

lw 5, ox.....

The problem is: how long long the address is. In the 32-bits arch., the address is made up by 32 bits.

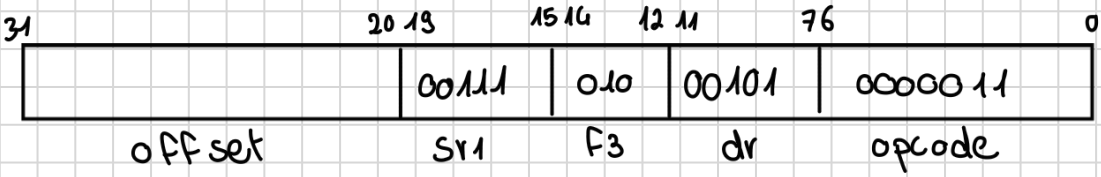
The problem is that I can't encode in 32 bits the opcode, the destination register, the address; it wouldn't fit.

But there is something that we can do, even more clever:

lw 5, offset (7)

I want to load into the register n.5 the content of the register n.7, with an offset (the difference in terms of bits, considering a base address; It's like an absolute value).

R5 ← RAM [R7 + offset]



ASSEMBLY OPERATION:

- lui (lower upper immediate)

This is an operation that allows us to put big numbers inside a register (instead of addi).

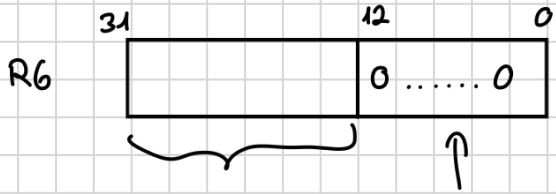
Example: 6, 0x100000

destination register

this is a HEX number and it requires 20 bits because:

100000

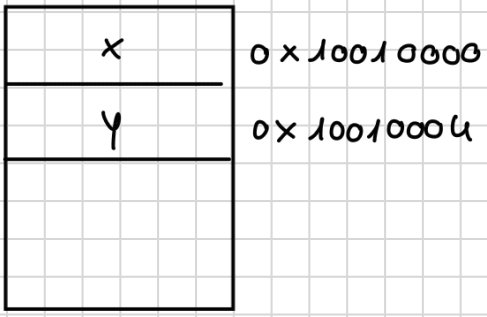
4 x 5 = 20



all zeros (don't ask why) !!

0x100000	00110	0110111

Exercise:



we want to put x+y in R5

" R5 ⟵ x+y "

write the instructions: ↴

Mei's solution (using offsets):

- lui 7, 0x10010
- lw 5, 0(7)
- lw 6, 4(7)
- R6 ⟵ MEM [R7 + 4] =
  = MEM [0x10010000 + 4] =
  = MEM [0x10010004]
- add 5, 5, 6
- sw 5, 8(7)

1st I have to move the address of x and put it into a register, in this case R7 (using lui);

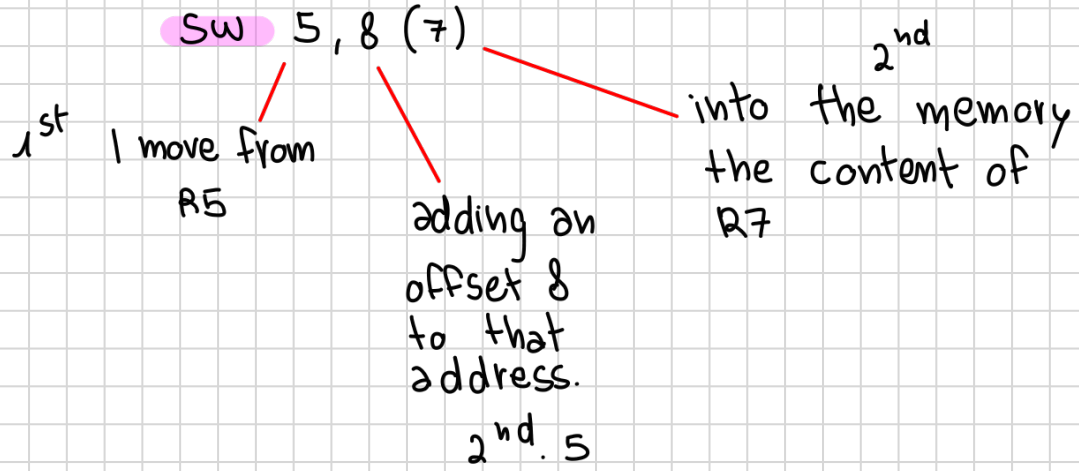
2nd Then, by reading the address inside R7, I can go again inside the memory and pick-up the real content, in order to move it into R5;

3rd Now, I can read the address stored into the R7 that will tell me the position of y

inside the RAM and move the content into R6;

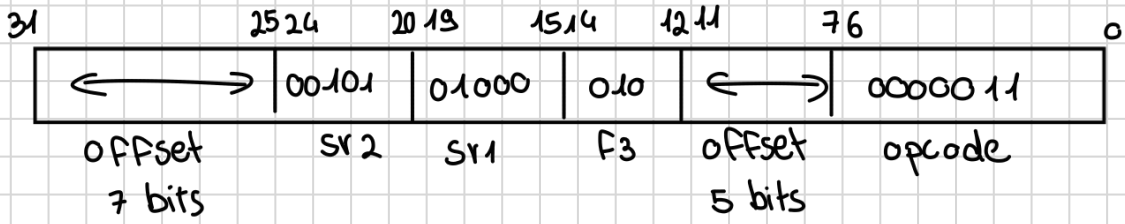
4th Now I can add R5 and R6 and store the result into R5;

5th Finally if I want to move back into the RAM the result obtained, I can do this using the "sw" (store word) operation.



* store word is basically the opposite of load word.

Example: MEM [R8 + offset] ← R5



Rars emulator

R = risc - v

A = assembler

R = and runtime

S = simulator

- data # starting at 0x10010000
  - word 13
  - word 25 — random numbers

0x10010000 — starting address ALWAYS

- text # starting at 0x00400000

```
lui 7, 0x10010
lw 5, 0 (7)
lw 6, 4(7)
add 5,6,5
sw 5, 8 (7)
```

To close the program use the following instruct.:

- addi 17, 0, 10 *
- ecall

17 → R17

10 → exit encoding (0x0a) HEX

(this stuff is a mistery)

ASSEMBLY OPERATION:

- beq (branch is equal) ⟺ "branch = ="

This is like a jump in the program, but only if two registers are equal (same content).
If two registers have the same content, then jump.

```
beq 5, 6, offset
```

↓

if R5 = R6

PC ← PC + offset

this is a register inside the CPU that contains the address of the next instruction.

Example:

Program:

add 5,0,0	0x00400000	1st	instruction address
add 6,0,0	0x00400004	2nd	//
beq 5,6,8	0x00400008	3rd	//
~~addi 5,5,1~~	~~0x0040000c~~	~~4th~~	//
addi 6,6,1	0x00400010	5th	//

When the CPU executes this code, the program counter (pc) starts with the value below:

PC 0x00400000

Another example:

```
addi  5,0,0
addi  6,0,10
add   5,5,6
addi  6,6,-1
bne   6,0,-8
```

* Which is the content of R5?

```
addi  5,0,0
addi  6,0,10
ciclo add   5,5,6
addi  6,6,-1
bne   6,0,ciclo
```

* Which is the content of R5?