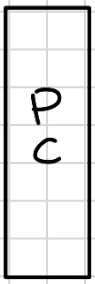
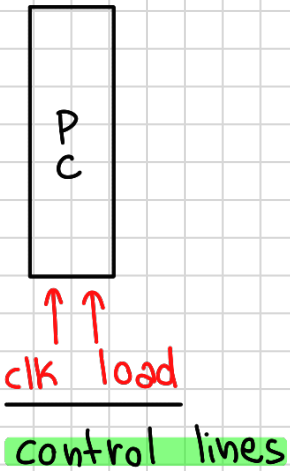
Let's design the real architecture that is able to execute the code that we usually write as instructions.

We start by giving a better definition of PC (program counter).

1st: the PC is a particular register of the CPU.



The PC in the architecture that we're considering stores 1 word (4 bytes / 32 bits).
With a 32 bits program counter, the RAM can be at most 4GB.



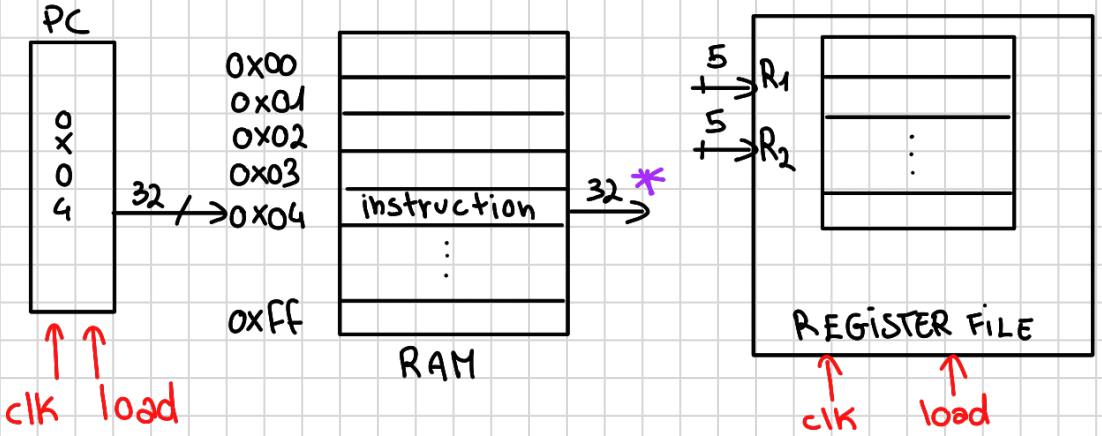
The execution of the instructions by the architecture that we're designing starts at the 1st ascending wavefront of the clock and has to end before the next one.

This kind of architecture is called a single-clock architecture because every instruction happens at a wavefront of the clock signal.

The PC is connected to the RAM memory with 32 wires (because we're working with the 32 bits architecture), and it is controlled by 2 control lines: the clock and the load.

The following drawing shows graphically the interconnection between the PC, the RAM and the REGISTER FILE:

The PC contains the addres of the word into the memory needed to execute an instruction.

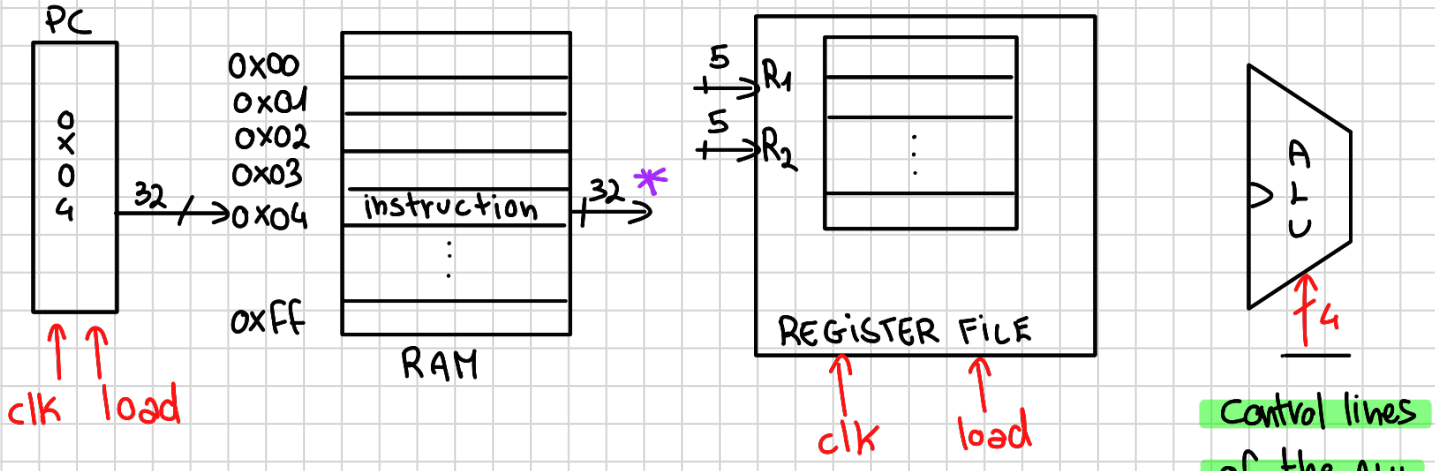


The registers of the CPU are embeded in a circuit called "register file".

P.S. When the load signal is 0, the clk signal has no effect.

* The numbers over the arrows represent the number of bits used.

The next drawing represents another important component: the ALU (arithmetic logic unit).

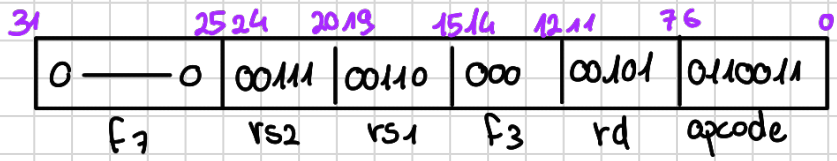


The control lines of the ALU establish if it has to do a sum, a sub, an or, etc...

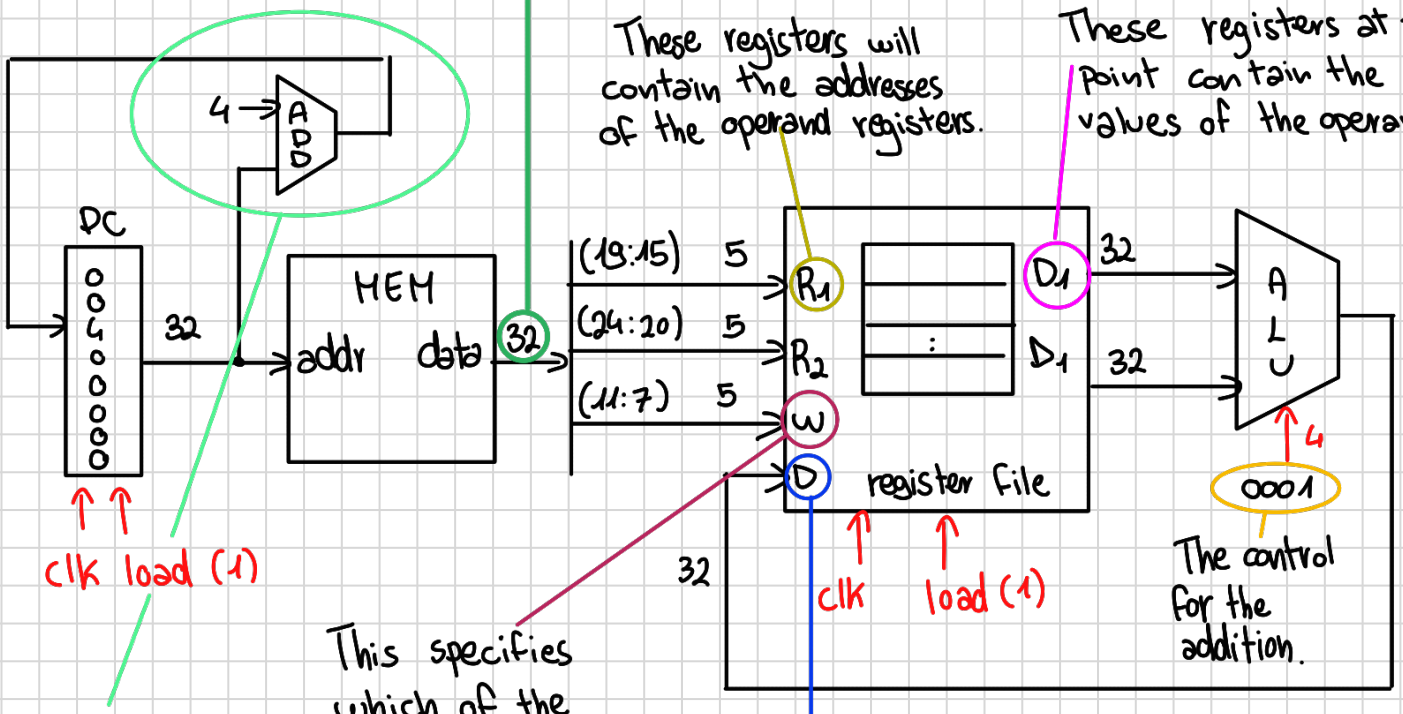
Let's design/draw an architecture that does only the sum:

0x00400000 **add** x_5, x_6, x_7

The add instruction is stored into the memory at this address.



This is the internal format of the instruction.



We do this to prepare the PC to do the next instruction. We add 4 to the PC in order to jump to the next instruction.

This specifies which of the 32 registers we have to write. Basically it stores the address of the final destination register.

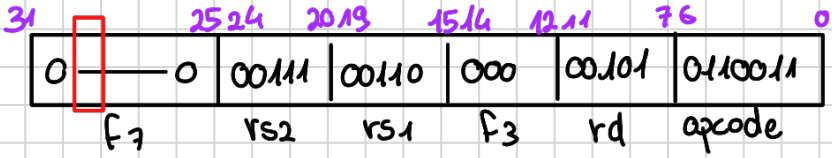
This specifies the final destination register that will contain the sum.

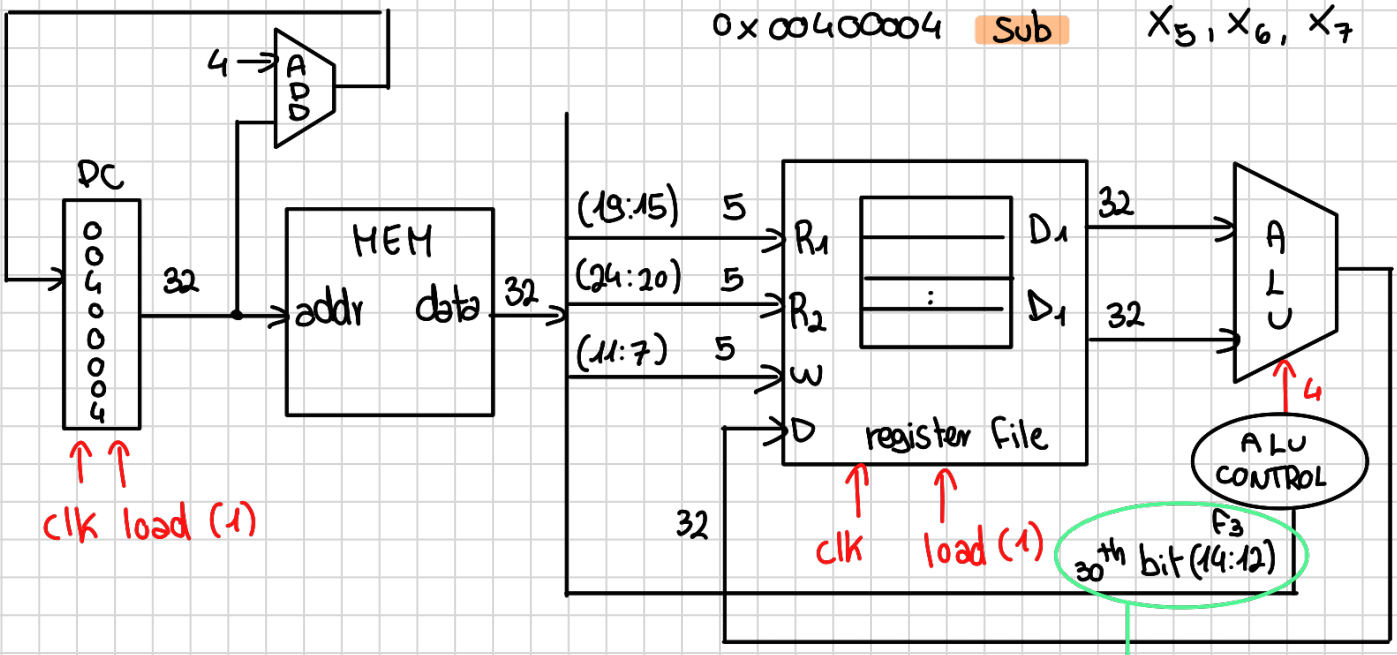
THINGS WORK IN PARALLEL

The architecture that we designed above works only when we want to do sums.

Let's see how the architecture becomes with addi, sub:

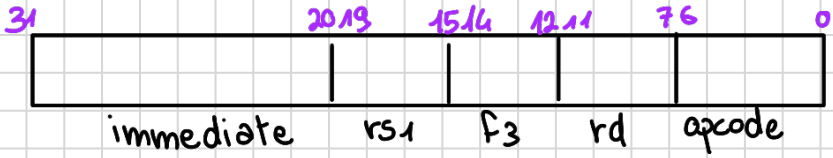
The only bit that changes into the instruction of the subtraction, is the 30th one, so just a little detail.





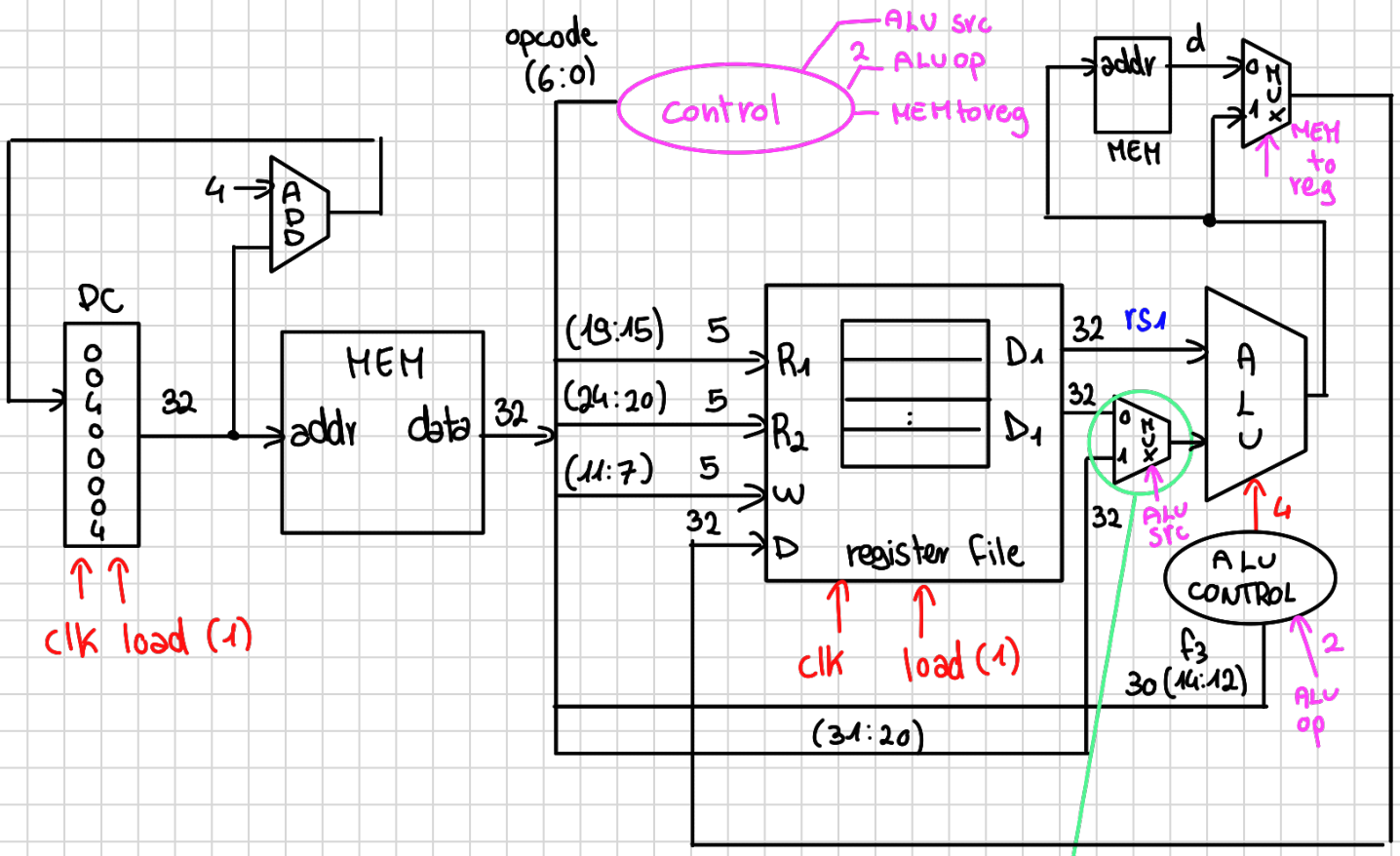
IF the 30th bit of f7 is 0, then the ALU does the sum, otherwise, it does the subtraction.

Let's see how the architecture becomes with add, sub, lw:



These represent wires.

lw rd, offset(rs1)

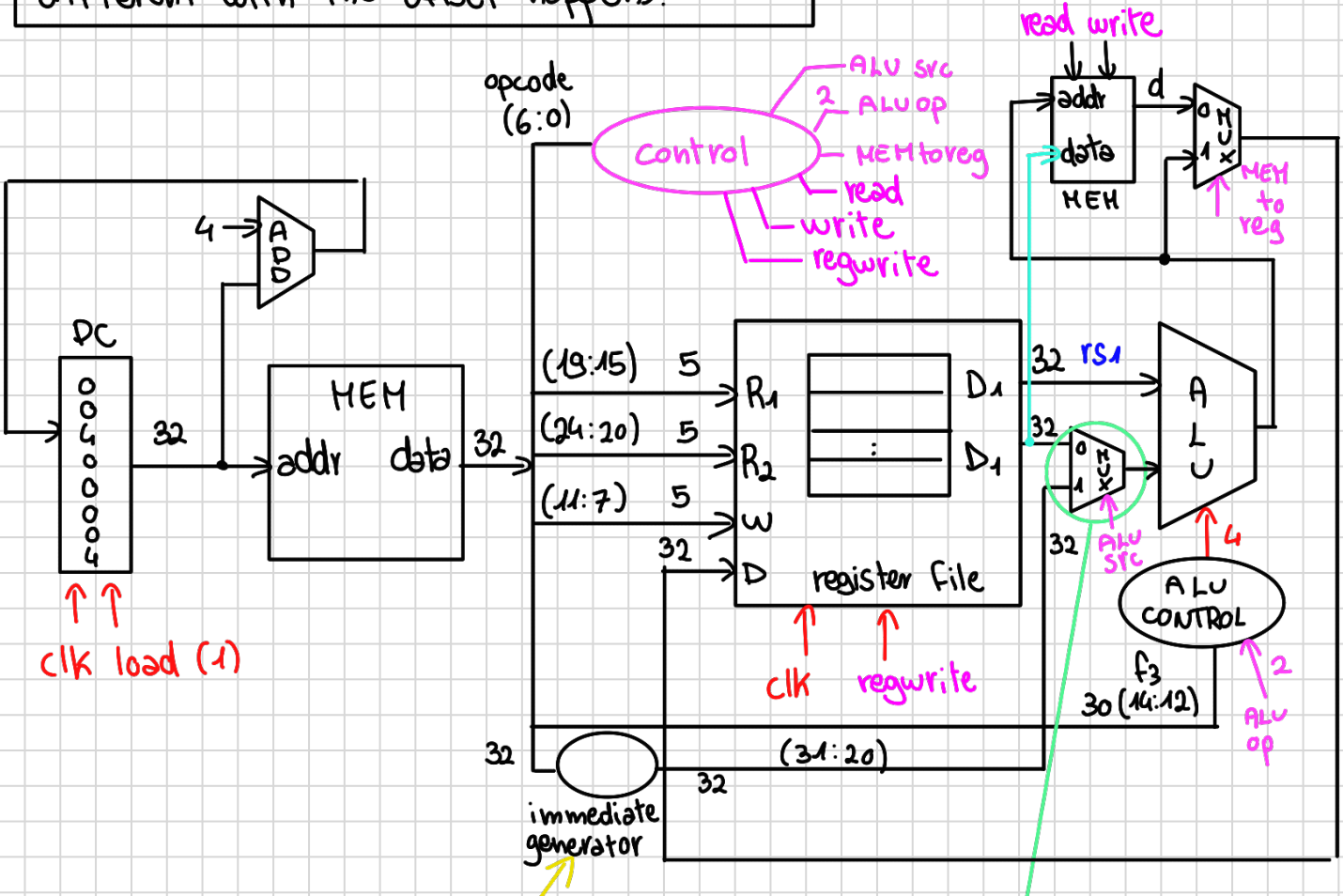


We use the MUX to implement the offset.

Let's see how the architecture becomes with add, sub, lw, sw:

sw rs_2, offset (rs_1)

Actually, with the store word, something different with the offset happens.



The immediate generator allows us to decide if to use lw or sw.

We use the MUX to implement the offset.