TD 3 : Construction of a real computer

1 Introduction

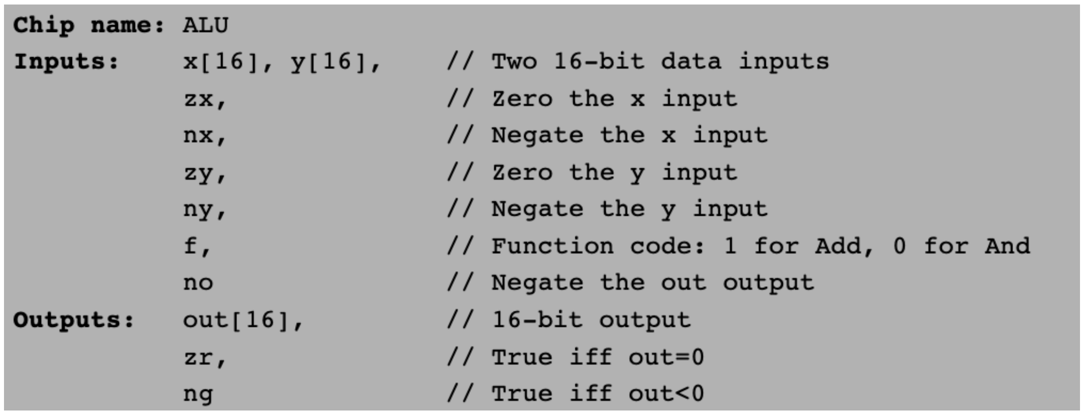
For this TP, you can find the UPDATED zip folder containing all the requisite files here : `https://www.amritasuresh.github.io/teaching/bootstrap.tar.gz`.

Continuing from last week's TP, we will finish our endeavour of building a small computer starting from (almost) scratch.

2 ALU

ALU

Once the previous gates have been implemented, we are now interested in the ALU. The idea of this chip is to take two vectors encoded on 16 bits as inputs x and y and to calculate at output a vector of 16 bits $out(x, y)$. out is a function which is parameterized by 6 flags (as shown in Fig. 2).



```
Chip name: ALU
Inputs:    x[16], y[16],    // Two 16-bit data inputs
           zx,              // Zero the x input
           nx,              // Negate the x input
           zy,              // Zero the y input
           ny,              // Negate the y input
           f,               // Function code: 1 for Add, 0 for And
           no               // Negate the out output
Outputs:   out[16],         // 16-bit output
           zr,              // True iff out=0
           ng               // True iff out<0
```

FIGURE 1 – The Arithmetic Logic Unit

Preliminaires : A few questions before embarking on the implementation (the order of the flags is the one presented above) :

— What is the function $out(x, y)$ when we pass the flags as a vector $(0, 1, 0, 1, 0, 1)$?
— What is the function $out(x, y)$ when we pass the flags as a vector $(0, 0, 1, 1, 0, 1)$?
— What is the function $out(x, y)$ when we pass the flags as a vector $(1, 1, 1, 1, 1, 1)$?
— Is it possible to calculate $x + 1$? If yes, what should be the value of the flags ?
— Is it possible to calculate $x - 1$? If yes, what should be the value of the flags ?
— Is it possible to calculate $x - y$? If yes, what should be the value of the flags ?

Implementation : Now, implement the following chip :

1. `ALU` (15 instructions)

3 Memory

This part is interested in logic gates whose behavior depends on time. In other words, one of the output pins can be connected to one of the input pins. This implies that in practice, this style of chip uses a clock, which will regulate the electrical signal. However, in our case, and to simplify the design of our chips, we are not going to manipulate the clock directly. Instead, we'll assume that we have a *built-in* chip, like the `nand` gate given to us. This chip, called `DFF` copies its input to its output at each clock *tick* (see Fig. 3). Note that using a clock is the same as *discretizing* time.

The Hardware Simulator is a Java program that isn't very smart about memory management. As a result, you should use the built-in versions of lower level RAM devices when constructing larger RAM devices (after you understand how to make the lower-level devices). Otherwise, the Hardware Simulator will recursively generate a ton of memory-resident software objects, each representing one of the parts that make up a typical RAM device. This may cause the simulator program to run slowly or to run out of memory. Hence, the folders have been divided into "a" and "b" to avoid this problem.

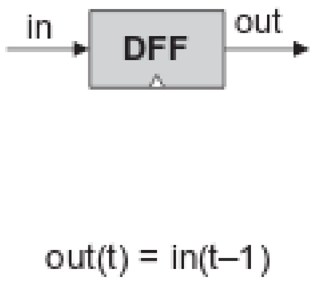


FIGURE 2 – DFF

For this section, program the chips in the following order :

1. `Bit` (2 instructions)
2. `Register` (16 instructions)
3. `PC` (5 instructions)
4. `RAM8` (10 instructions)
5. `RAM64` (10 instructions)
6. `RAM512` (10 instructions)
7. `RAM4K` (10 instructions)
8. `RAM16K` (6 instructions)