****

**Universidad Autónoma de Guadalajara**

Ingeniería Electrónica Biomédica

Digital Systems

Andrea Alejandra Mondragón Olivos

Jesús Arnoldo Zerecero Núñez

2915351

2885993

*“Practice 1: ALU”*

09-06-17

ALU

**Introduction.**

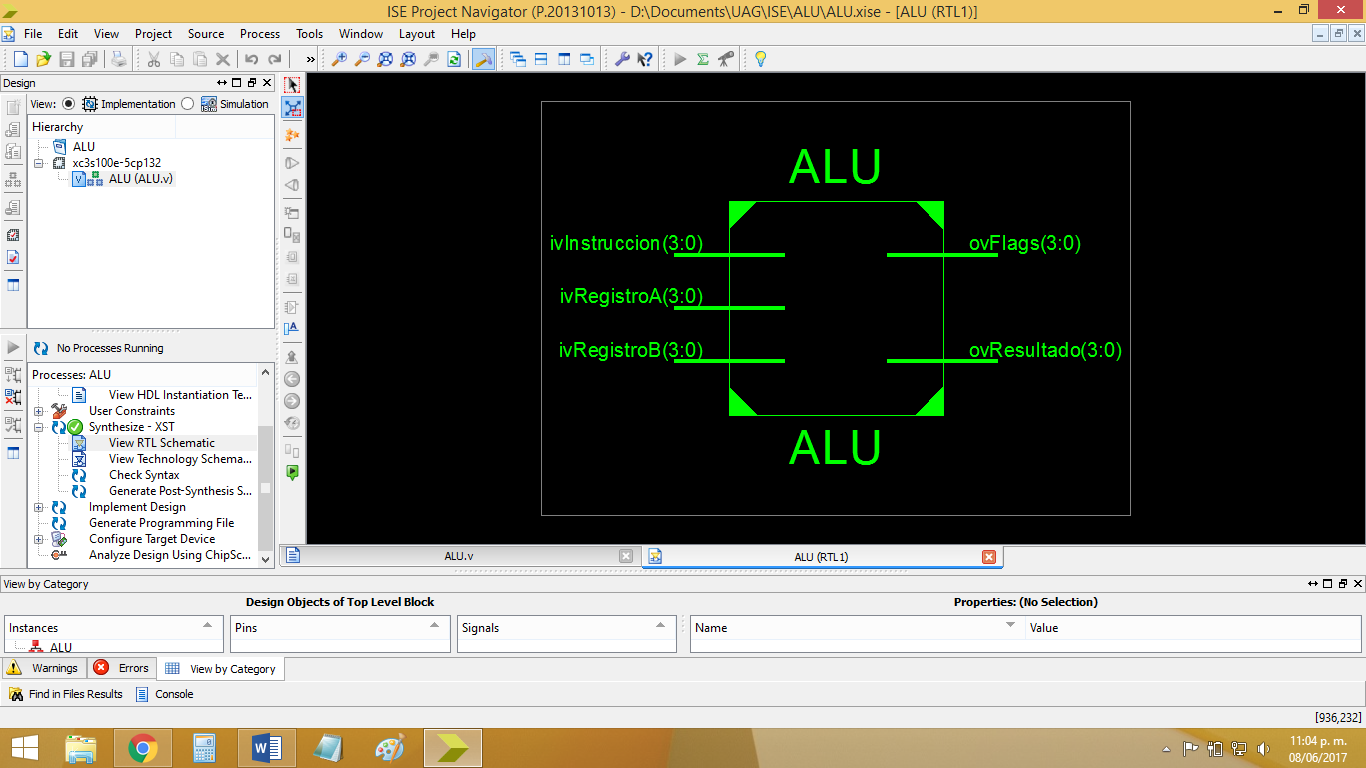
While working with microprocessors, many instructions depend on the usage of arithmetic and logical operations In order to manipulate date bits between registers. Most of these operations can be performed using similar hardware, so, most microprocessors contain a single component where all these operations take place, known as the Arithmetic Logic Unit (ALU).

Apart from arithmetic and logical, the ALU is able to use many different operators, such as relational operators (less than, greater than), bit-wise operators (and, or, not, xor…), reduction operators (nand, nor, xnor…), and shift operators. As for the relational operators, there is an internal component contained within the ALU known as Condition Code Register (CCR) which is able to determine the relation between the inputted data. The CCR is able to make different comparisons between the input registers and outputs 4 different flags: Carry, Overflow, Zero and Negative, depending on the result’s behavior.

* **Carry:** While adding or subtracting, the result may be bigger than the bus size being worked on, leading to data loss as some bits won’t fit and get discarded. This flag tells us whenever that happens.
* **Overflow:** Whenever a result has an unexpected sign (the sign is determined by its MSB).
  + **Addition:** If two negative numbers are added and the result comes out negative, the overflow flag will turn on.
  + **Subtraction:** If negative minus positive yields positive or if positive minus negative yields negative, the overflow flag will turn on.
* **Zero:** Turns on whenever the result is equal to zero.
* **Negative:** Turns on whenever the result is negative.

Depending on the flags turned on, the CCR is able to make to make a comparison between the registers, later used as relational operators to be evaluated by different conditions.

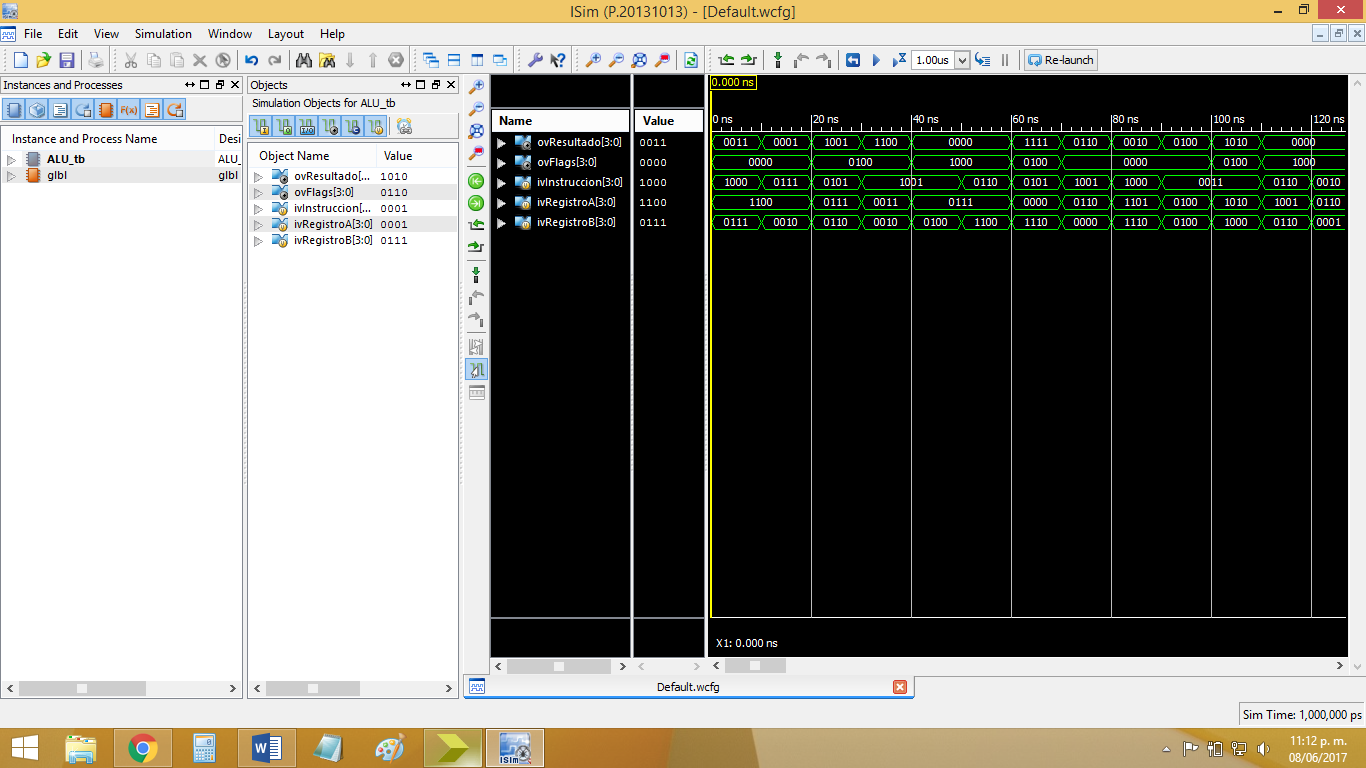
In this practice, an ALU simulation was created using Verilog. This simulation is able to test different operators using 2 4-bit input registers. The CCR flags were also calculated.



*Figure 1: Block diagram for the ALU simulation.*

**Development.**

1. Create a single module for both the ALU and the CCR.
2. Name each instruction using a local parameter corresponding to different 4 bit-values. The instructions simulated were:
   * Addition
   * Subtraction
   * And
   * Or
   * Xor
   * Nand
   * Nor
   * Xnor
   * Not
   * Left shift
   * Right shift.
3. Using cases, perform an operation between the registers depending on the inputted instruction code.
4. Using bit-wise, reduction operators and concatenation, calculate the four CCR flags depending on the input registers and the result of their operation.
   * This is where we had most trouble, as the objective of the CCR is not to use conditions to test each individual bit from the result or input registers in order to know which flags to turn on, but to use as less operators as possible to calculate or concatenate the corresponding flag bit using the same resulting bits.
     1. For example, for the negative flag, it’s best to just use the result’s MSB as this will yield 1 it the result is negative and 0 if it’s positive.
5. Create a test bench module and simulate many possible instruction, input and output outcomes using the $random function.



*Figure 2: Test bench simulation results for the ALU + CCR module.*

**Conclusion.**

**Arnoldo:**

In any microprocessor, the ALU is without a doubt one of its most important components as, without it, most operations and calculations won’t be completed. This is the part of the MCU every instruction depends on, from its result to the comparison between its input registers in order for them to be evaluated in conditionals. Being our first practice of this course, we had very little trouble as we are already familiarized with Verliog coding language, as well as the ISE software. We just needed to investigate a function to be able to simulate lots of outcomes in a single test bench simulation (we opted for the $random function).

**Alejandra:**