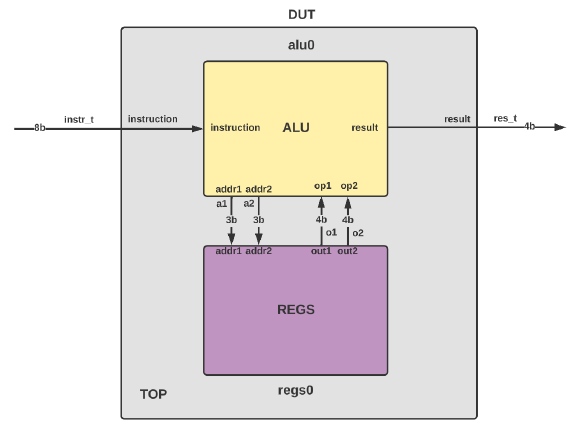
.

The aim is to implement and test an arithmetic logic unit with default registers.

The instruction received by the ALU encodes in itself 3 important parameters: the operation code and the addresses of the two registers from which we extract the information to be processed.

The instruction takes the following form:



Thus, with a single port, we provide all the information needed to select the desired registers and perform the required operation.

ALU

The ALU is a module that receives the instruction code as input, decodes the bits of the two addresses and the opcode, queries the REGS module to obtain the register values ​​from the addresses coded in the instruction, and performs the operation specified in the opcode.

Instructions and their related codes:

• addition - opcode 2'b00 - adds the values ​​found in the two registers whose addresses we receive at the input

• subtract - opcode 2'b01 - decreases the values ​​found in the two registers whose addresses we receive at the input

• AND – opcode 2'b10 – performs the bitwise AND operation

• OR – opcode 2'b11 – performs the bitwise LOGIC OR operation

From the instruction and the diagram we must be able to observe the size of the addresses ourselves.

The output of the ALU module is 4 bits long, as are the registers we read from.

REG

The REGS module contains 8 registers of 4 bits each. For simplicity, each memory location contains a number equal to its address. So location 0 will contain 4'b0000, location 1 will contain 4'b0001 ... and location 7 will contain 4'b0111.

The module will thus contain a vector of 8 elements of 4 bits each. Automating the allocation of values ​​in each memory register is also desired.

Testbench

To test the operation of the circuit, we propose the following testbench structure:

After the validation of the TOP module is done, we start assigning values ​​to the input signals. Each opcode is tried at least twice, with different address values. All possible values ​​of the statement can be attempted.