# CA Lab: Lab 8

student: Dimitri Tabatadze

May 19, 2023

## Task Description

Implement ALU in verilog.

## Solution

The main module:

```
module CPU_ALU (
    input [31:0] A_input, B_input,
    input [3:0] Cmd,
    input [4:0] Shamt,
    input ALU_enable, sh,
    output reg [32:0] Results,
    output [3:0] NZCV
);
    always @(Cmd or A_input or B_input or ALU_enable or sh or Shamt)
    begin
        Results = 0;
        if (ALU_enable) begin
            case (Cmd)
                4'h0 : Results[31:0] = A_input | B_input;
                4'h1 : Results[31:0] = A_input & B_input;
                4'h2 : Results[31:0] = A_input ^ B_input;
                4'h3 : Results[31:0] = A_input ^ B_input; // Since
    bitwise add is same as xor
                4'h4 : Results[31:0] = A_input ^ B_input; // Since
    bitwise sub is same as xor
                4'h5 : Results[31:0] = B_input ^ A_input; // Since
    bitwise sub is same as xor
                4'h6 : Results[32:0] = A_input + B_input;
                4'h7 : Results[32:0] = A_input - B_input;
                4'h8 : Results[32:0] = B_input - A_input;
                4'h9 : Results[32:0] = A_input == B_input;
                4'ha : Results[32:0] = A_input != B_input;
                4'hb : Results[32:0] = A_input > B_input;
                4'hc : Results[32:0] = A_input >= B_input;
                4'hd : Results[32:0] = A_input < B_input;
                4'he : Results[32:0] = A_input <= B_input;
                4'hf : if (sh) Results = A_input << Shamt; else Results
     = A_input >> Shamt;
```

```
            endcase
        end
    end

    assign NZCV[0] = Results[32] & (Cmd != 4'hf);
    assign NZCV[1] = Results[32] & (Cmd == 4'hf);
    assign NZCV[2] = (Results[31:0] == 0) & ALU_enable;
    assign NZCV[3] = !Results[32] & (Cmd == 4'h4 | Cmd == 4'h5) | (
    A_input[31] ^ (~B_input[31]) ^ Results[32]);

endmodule
```

The testbench

```
module alu_tb();

reg [31:0] A_input;
  reg [31:0] B_input;
  reg [3:0] Cmd;
  reg ALU_enable;
  reg sh;
  reg [4:0] Shamt;
  wire [32:0] Results;
  wire [3:0] NZCV;

  CPU_ALU UUT (
    .A_input(A_input),
    .B_input(B_input),
    .Cmd(Cmd),
    .Shamt(Shamt),
    .ALU_enable(ALU_enable),
    .sh(sh),
    .Results(Results),
    .NZCV(NZCV)
  );

  initial begin
    $dumpfile("alu-tb.vcd");
    $dumpvars(0, alu_tb);
    ALU_enable = 1;
    A_input = 32'd11;
    B_input = 32'd7;
    Cmd = 4'b0000;
    #10;
    Cmd = 4'b0001;
    #10;
    Cmd = 4'b0010;
    #10;
    Cmd = 4'b0011;
    #10;
    Cmd = 4'b0100;
    #10;
    Cmd = 4'b0101;
    #10;
    Cmd = 4'b0110;
    #10;
    Cmd = 4'b0111;
    #10;
```

```
    Cmd = 4'b1000;
    #10;
    Cmd = 4'b1001;
    #10;
    Cmd = 4'b1010;
    #10;
    Cmd = 4'b1011;
    #10;
    Cmd = 4'b1100;
    #10;
    Cmd = 4'b1101;
    #10;
    Cmd = 4'b1110;
    #10;
    sh = 1;
    Shamt = 3;
    Cmd = 4'b1111;
    #10;
  end
endmodule
```

## Conclusion

Since addition mod 2 (bitwise addition) is the same as xor, and same as subtraction and reversed subtraction, I have written ^ for the commands 3 to 5. Other than that, everything should be clear.

For NZCV, I used the book.

## Reference

- me.
- The book