Laboratorio 5

Victor Ostolaza
Macarena Oyague

Arquitectura de Computadores

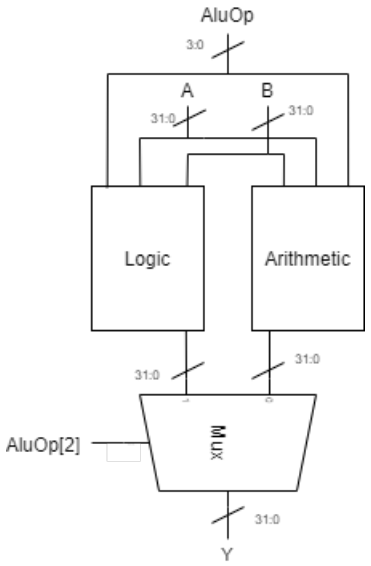
June 15, 2020

Arithmetic Logic Unit

El diseño de un ALU contempla lo siguiente:

- A: input de 32 bits
- B: input de 32 bits
- AluOp: input de 4 bits
- Y (Result): input de 32 bits

El diagrama de un ALU contempla dos módulos, uno para operaciones binarias que son de carácter lógico y otro para aquellas que son de carácter aritmético.



A los módulos lógico y aritmético ingresan A, B y AluOp. Los módulos están diseñados con 'case' para evaluar las diferentes operaciones que corresponden a cada módulo.

Por un lado, el módulo Logic contiene las operaciones:

- and
- or
- xor
- nor

Por otro lado, el módulo Arithmetic contiene las operaciones:

- add
- sub
- slt

Según la tabla expresada en las indicaciones del laboratorio, estas operaciones serán codificadas de la manera siguiente, por lo cual cada módulo arrojará un output:

AluOp	**Mnemonic**	**Result =**	**Description**
0000	add	$A + B$	Addition
0010	sub	$A - B$	Subtraction
0100	and	$AandB$	Logical and
0101	or	$AorB$	Logical or
0110	xor	$AxorB$	Exclusive or
0111	nor	$AnorB$	Logical nor
1010	slt	$(A - B)[31]$	Set less than
Other	n.a.	Don't care	

Para la implementación del módulo Logic en Verilog, el output se genera de manera behavioral en función a lo que representa cada operación parametrizada. Además, si es que se introduce un valor de AluOp que no pertenece a ninguna operación que corresponde al módulo, se arrojará un 'don't care'. Se puede observar el diseño a continuación:

```
module logicUnit (A, B, opcode, out);
    input[31:0] A, B;
    input [3:0] opcode;
    output [31:0] out;
    reg [31:0] out;

    parameter and  = 4'b0100, or  = 4'b0101, xor  = 4'b0110, nor  = 4'b0111;

    always @ (A or B or opcode) begin
        casex (opcode)
            and : out = A & B;
            or : out = A | B;
            xor : out = A ^ B;
            nor : out = ~(A | B);
            default: out = 32'bx;
        endcase
    end

endmodule
```

De igual manera se da la implementación del módulo Arithmetic en Verilog. El diseño es el siguiente:

```
module arithmetic (A, B, opcode, out);
    input[31:0] A, B;
    input [3:0] opcode;
    output [31:0] out;
    reg [31:0] out;

    parameter add = 4'b0000, sub = 4'b0010, slt = 4'b1010;

    always @ (A or B or opcode) begin
        casex (opcode)
            add: out = A + B;
            sub: out = A - B;
            slt: out = (A < B) ? 1 : 0;
            default: out = 32'bx;
        endcase
    end

endmodule
```

Es posible onservar que el bit número 3 del AluOp responde a si es que la operación a realizar será de carácter lógica o aritmética. Si es que AluOp[2] tiene el como valor a 0-lógico, la operación se encuentra dentro del módulo Logic, mientras que si es que es 1-lógico se encuentra en el módulo Arithmetic. Es por ello que se utilizó un mux 2:1 como parte de la implementación, el cuál tiene como selectror al AluOp[2]; en función a esto el ALU arrojará el output al módulo que corresponde.

```
module mux2_1 (logicU, arithU, sel, out);
    input [31:0] logicU, arithU;
    input sel;
    output [31:0] out;
    assign out = sel ? logicU : arithU;
endmodule
```

El resultado final del módulo del ALU contempla un módulo Logic, un módulo Arithmetic y un módulo Mux 2:1. Como output arrojará el resultado de la operación entre los inputs A y B, así como también un output llamado 'Zero' que es 1 si es que todos los bits de Result son 0 El módulo se puede ver a continuación:

```
module alu (A, B, opcode, result, zero);
    input[31:0] A, B;
    input [3:0] opcode;
    output [31:0] result;
    output zero;
    wire [31:0] outLogic, outArithmetic;

    logicUnit log(A, B, opcode, outLogic);
    arithmetic ari(A, B, opcode, outArithmetic);
    mux2_1 mux(outLogic, outArithmetic, opcode[2], result);
    assign zero = (result == 31'b0) ? 1 : 0;
endmodule
```

Se ha realizado un testeo a través de un testbench que pueda probar todas las opciones de opcode facilitadas en las hojas de ejercicios. Se pudo obtener todas las respuestas luego de variar 7 unidades de tiempo, que en este caso son nanosegundos:

```
module alu_tb;

reg[31:0] A, B;
reg[3:0] opcode;
wire[31:0] result;

initial
$monitor ("A = %b, B =%b, opcode = %b, Result = %b, Zero = %b",
A, B, opcode, result, zero);

initial
begin
    A = 15;
    B = 10;

    #1 opcode = 4'b0000;
    #1 opcode = 4'b0010;
    #1 opcode = 4'b0100;
    #1 opcode = 4'b0101;
    #1 opcode = 4'b0110;
    #1 opcode = 4'b0111;
    #1 opcode = 4'b1010;
    #1 opcode = 4'bx;

        #18 $finish;

end

alu inst1 (A, B, opcode, result, zero);

initial begin
        $dumpfile("alu.vcd");
        $dumpvars;
end

endmodule
```

En el WaveForm se puede verificar el funcionamiento correcto de los módulos: