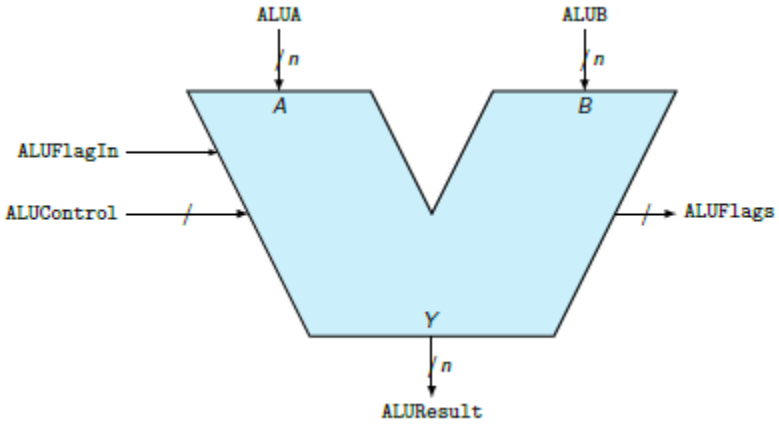
Documentación

Ej. 5: ALU

Marco teórico

La ALU (Arithmetic Logic Unit) es u dispositivo que realiza distintos tipos de operaciones matemáticas o lógicas a las entradas que se le ingresen. A su vez brinda todo tipo de señales sobre el estado de actual de sus cálculos y por su puesto el resultado pertinente.

La ALU sigue el siguiente diseño



El cual consiste en señales de entrada, salida y control que determinan la función que realiza.

Descripción del código a implementar

Módulo TOP

Para la ALU se utiliza como TOP un modulo que reciba todas las señales de la imagen como entradas y salidas. Con esto definido se comienza la lógica del funcionamiento de la ALU. Para esto se utiliza la se de entrada ALUControl. Esta señal se encarga de seleccionar la operación aritmética a realizar por la ALU. Para el funcionamiento de esta se utiliza un case y como condición se utiliza dicha señal de ALUControl. Con esto se asigna la operación correspondiente a la solicitud del usuario.

Módulo 1: And

Compara cada una de las entradas y proporciona una salida binaria comparando cada uno de los bits de las entradas predominando el valor “1” el bit equivalente en cada entrada debe tener valor en alto para que la salida sea 1 de lo contrario la salida será 0.

```
4'h0: begin //and
    assign ALUResult = A & B;
end
```

Módulo 2: Or

Compara cada una de las entradas y proporciona una salida binaria comparando cada uno de los bits de las entradas predominando el valor “si” si al menos un bit de un valor en alto, la salida de ese bit será en alto.

```
4'h1: begin //or
    assign ALUResult = A | B;
end
```

Módulo 3: Suma

Compara cada una de las entradas y proporciona una salida binaria comparando cada uno de los bits de las entradas. Realiza una suma entre los bits y lleva el acarreo en ALUFlags.

```
4'h2: begin //suma
    logic [Ancho:0] temp_sum;
    assign temp_sum = A + B + ALUFlagIn;
    assign ALUResult = temp_sum[Ancho-1:0];
    assign ALUFlags = temp_sum[Ancho];
end
```

Módulo 4: Incremento

En términos generales, añade un 1 lógico a cada una de las entradas.

```
4'h3: begin //incremento
    if (ALUFlagIn) begin
        assign ALUResult = B + 1;
    end
    else begin
        assign ALUResult = A + 1;
    end
end
```

Módulo 5: Decremento

En términos generales, resta un 1 lógico a cada una de las entradas.

```
4'h4: begin //decremento
    if (ALUFlagIn) begin
        assign ALUResult = B - 1;
    end
    else begin
        assign ALUResult = A - 1;
    end
end
```

Módulo 6: Not

Compara cada una de las entradas y proporciona una salida binaria correspondiente al valor inverso de los bits del valor de entrada.

```
4'h5: begin //not
    if (ALUFlagIn) begin
        assign ALUResult = ~B;
    end
    else begin
        assign ALUResult = ~A;
    end
end
```

Módulo 7: Resta

Compara cada una de las entradas y proporciona una salida binaria comparando cada uno de los bits de las entradas. Realiza una resta entre los bits y lleva el acarreo en ALUFlags.

```
4'h6: begin //resta
    assign {ALUFlags, ALUResult} = A - B - ALUFlagIn;
end
```

Módulo 8: Xor

Si el bit equivalente entre ambos valores de entrada es igual, la salida es 0, de lo contrario si son diferentes la salida es 1.

```
4'h7: begin //xor
    assign ALUResult = A ^ B;
end
```

Módulo 9: Corrimiento izquierda

Se implementa un desplazamiento lógico a la izquierda desplazando A por B posiciones (A << B) y almacena el resultado en ALUResult.

El carry_o captura un posible bit de acarreo.

```
4'h8: begin //corriemiento Izquierda
    logic [Ancho - 1:0] resul_ci;
    logic carry_o;

    resul_ci = A << B;
    carry_o = A >> (B - 1);

    ALUResult = resul_ci;
end
```

Módulo 9: Corrimiento derecha

Se realiza un desplazamiento lógico a la derecha, desplazando A por B posiciones (A >> B) y almacena el resultado en ALUResult.

El carry_o guarda el bit más significativo de A.

```
4'h9: begin //corriemiento Derecha
    logic [Ancho - 1:0] resul_cd;
    logic carry_o;

    resul_cd = A >> B;
    carry_o = A[Ancho - 1];

    ALUResult = resul_cd;
end
```