Organización y Arquitectura de las Computadoras

Práctica 04: Unidad Aritmético Lógica

Bonilla Reyes Dafne
319089660

Medina Guzmán Sergio
314332428

1. Objetivo

El alumno practicará el proceso de diseño de circuitos combinacionales.

2. Procedimiento

Deberás entregar un archivo de Logisim con las soluciones de los ejercicios, un archivo de código fuente escrito en el lenguaje de programación C con la función que simula el funcionamiento de una ALU y un documento PDF con las respuestas a las preguntas planteadas. Para la simulación de Logisim solo puedes hacer uso de compuertas lógicas **AND, OR, NOT y XOR**, multiplexores, separadores y pines de entrada y salida. Recuerda etiquetar las entradas y salidas de cada uno de los subcircuitos.

3. Ejercicios

1. Da una breve introducción a que son las ALUs en tu reporte.

   Una ALU (por sus siglas en inglés arithmetic logic unit) es un componente principal de la CPU que realiza operaciones aritméticas y lógicas. Tiene la capacidad de realizar todos los procesos relacionados con operaciones aritméticas y lógicas, como operaciones de suma, resta y desplazamiento, incluidas las comparaciones booleanas (operaciones XOR, OR, AND y NOT). Además, los números binarios pueden realizar operaciones matemáticas y bit a bit. La unidad lógica aritmética se divide en AU (unidad aritmética) y LU (unidad lógica). Algunos procesadores contienen más de una AU, por ejemplo, una para operaciones de punto fijo y otra para operaciones de punto flotante.

   Los operandos y el código utilizados por la ALU le indican qué operaciones debe realizar de acuerdo con los datos de entrada. Cuando la ALU completa el procesamiento de la entrada, la información se envía a la memoria de la computadora.

2. Desarrolla un sumador completo de 1 bit.

   Este ejercicio se encuentra dentro del zip en el archivo de logisim `Circuitos-Ejercicio-02`.

3. Desarrolla un sumador de 8 bits descrito en la sección 4.1.2 utilizando el circuito del ejercicio anterior.

   Este ejercicio se encuentra dentro del zip en el archivo de logisim `Circuitos-Ejercicio-03`.

4. Desarrolla un sumador y restador de 8 bits utilizando el circuito del ejercicio anterior y agregando los componentes necesarios para la sustracción descrita en la sección 4.1.3. Se tiene que agregar un multiplexor para seleccionar la operación de salida, 0 para la adición y 1 para sustracción.

   Este ejercicio se encuentra dentro del zip en el archivo de logisim `Circuitos-Ejercicio-04`.

5. En un subcircuito de tu elección, coloca como un componente el sumador y restador final del ejercicio anterior con pines de entradas y salida necesarios con el fin de llevar a cabo la simulación.

   Este ejercicio se encuentra dentro del zip en el archivo de logisim `Circuitos-Ejercicio-05`.

El circuito implementado en este ejercicio corresponde a un multiplicador de dos números de 8 bits. Se incluye la posibilidad de multiplicar números negativos: Si el multiplexor que recibe los números y sus complementos a dos, recibe como operación a ejecutar un 0, ejecuta la multiplicación del número positivo; mientras que si recibe un 1, la ejecuta con su complemento a 2. Esto se logró separando los bits del primer número y recorriendolos a la izquierda con cada separador, luego, el número de 8 bits resultante entra a un transistor, la salida de tipo N, que comparte cable con la salida de un transistor de tipo P, esto porque sólo se suman los bits recorridos que coincidan con los bits que sean 1 en el segundo número de entrada, pues la multiplicación binaria es básicamente sumar el primer número con sus bits recorridos a la izquierda tantos lugares como la posición de cada bit que sea 1 del segundo número. Cuando un bit del segundo número es 0, el transistor de tipo P deja pasar un número de 8 bits donde todos son 0, pues es multiplicar el primer número por 0; cuando un bit del segundo número es 1, el transistor de tipo N deja pasar el número de 8 bits que corresponde al primer número con sus bits recorridos a la izquierda ciertos lugares. Es importante mencionar que si uno de los dos factores es k, entonces el otro número tiene que ser menor o igual a $\lfloor 127/k \rfloor$, pues nuestros 8 bits representan los números del -127 al 128. Finalmente, cada número de 8 bits que sale de cada pareja de transistores es una entrada a una de nuestras ALU's, y la salida de cada ALU va sumándose como entrada a la siguiente ALU, hasta sumar todos los números necesarios y obtener en la salida el resultado.

6. Escribe la función con los requerimientos solicitados en la sección 4.2, la función debe realizar la aritmética y lógica en binario.

   Este ejercicio se encuentra dentro del zip en el archivo de C llamado `practica04`.

4. Preguntas

1. ¿Quién propuso por primera vez el diseño de una ALU? ¿Para qué computador se propuso?

   Tras la creación de la primera computadora electrónica de propósito general en la historia, Presper Eckert, John Mauchly, Herman Goldstine y John von Neumann propusieron una mejor manera de proveer a una computadora del programa a ejecutar. Concluyeron que el programa debía poder almacenarse en la memoria de la máquina, para de allí ser leído e interpretado por la entidad encargada de la ejecución, llamada unidad de control. Las instrucciones serían entonces ejecutadas, con datos tomados de la misma memoria, por la unidad aritmético-lógica (ALU) de la máquina. La idea era usar esta alternativa en la siguiente computadora, la sucesora de ENIAC, a la que se denominó EDVAC (Electronic Discrete Variable Automatic Computer). Así, Von Neumann escribió estos conceptos en un reporte en junio de 1945, llamado First Draft of a Report on the EDVAC. En el reporte se describe, esencialmente, lo que hoy llamamos Arquitectura de Von Neumann. [1]

2. ¿Qué es VHDL? ¿Cómo se ve un full-adder en VHDL?

   VHDL es un lenguaje de programación que ha sido diseñado y optimizado para describir el comportamiento de circuitos y sistemas digitales. Los circuitos digitales capturados con VHDL se pueden simular fácilmente, haciendo que se puedan sintetizar en múltiples tecnologías de destino y se pueden archivar para su posterior modificación y reutilización.[2]

   VHDL es acrónimo proveniente de la combinación de dos acrónimos: VHSIC (Very High Speed Integrated Circuit) y HDL (Hardware Description Language). Aunque puede ser usado de forma general para describir cualquier circuito digital se usa principalmente para programar PLD (Programable Logic Device), FPGA (Field Programmable Gate Array), ASIC y similares.

   Originalmente, el lenguaje VHDL fue desarrollado por el departamento de defensa de los Estados Unidos a inicios de los años 80 basado en el lenguaje de programación ADA con el fin de simular circuitos eléctricos digitales.[3]

   Un full-adder en VHDL se vería de la siguiente manera:[4]

```
library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity adder_full_signed_reg is
generic(
  N             : integer := 8);
port (
  i_clk       : in     std_logic;
  i_add1      : in     std_logic_vector(N-1 downto 0);
  i_add2      : in     std_logic_vector(N-1 downto 0);
  o_sum       : out    std_logic_vector(N downto 0));
end adder_full_signed_reg;
architecture rtl of adder_full_signed_reg is
signal r_add1   : signed(N downto 0);
signal r_add2   : signed(N downto 0);
signal w_sum    : signed(N downto 0);
begin
  w_sum <= r_add1 + r_add2;
r_process : process(i_clk)
begin
  if(rising_edge(i_clk)) then
    r_add1      <=  resize(signed(i_add1),N+1);
    r_add2      <=  resize(signed(i_add2),N+1);
    o_sum       <= std_logic_vector(w_sum);
  end if;

end process r_process;
end rtl;
```

3. ¿Qué operaciones aritméticas y lógicas son básicas para un procesador? Justifica tu respuesta.

   La adición, la resta, la multiplicación y la división son las cuatro operaciones aritméticas básicas. Es posible derivar otras operaciones aritméticas y resolver problemas científicos mediante métodos de análisis numérico. Estas operaciones aritméticas son ejecutadas en la ALU. [5]

   Sin embargo, en otras fuentes se consideran más operaciones aritméticas básicas para el procesador [6], por ejemplo:

   - Suma con acarreo
   - Restar con préstamo (Carry-in y Carry-out)
   - Complemento a dos (negado)
   - Incremento
   - Decremento
   - Paso a través:

   Por otro lado, las operaciones lógicas básicas para un procesador son AND, OR, NOT, XOR, NAND y NOR. Estas operaciones lógicas se consideran fundamentales porque permiten realizar una variedad de operaciones lógicas más complejas mediante combinaciones y modificaciones de estas operaciones básicas.

   Además, estas operaciones lógicas básicas son universales, lo que significa que cualquier otra operación lógica puede ser implementada utilizando solo estas operaciones básicas. Por ejemplo, la operación de comparación "mayor que"puede ser implementada utilizando operaciones AND, OR y NOT.

   Así mismo, estas operaciones lógicas son las más comúnmente utilizadas en la programación y el diseño de sistemas digitales debido a su simplicidad y eficiencia. Los circuitos digitales que implementan

estas operaciones lógicas son muy comunes y se utilizan en la mayoría de los sistemas digitales modernos, desde los procesadores de computadoras hasta los circuitos integrados en los dispositivos electrónicos de consumo.

En resumen, AND, OR, NOT, XOR, NAND y NOR son consideradas las operaciones lógicas básicas de un procesador porque son fundamentales, universales y eficientes, y se utilizan para construir circuitos digitales complejos y realizar cálculos y toma de decisiones en los sistemas informáticos. [7]

4. El diseño utilizado para realizar la adición resulta ser ineficiente, ¿por qué? ¿Qué tipo de sumador resulta ser más eficiente?

   Aunque un sumador completo es muy útil y necesario en ciertas aplicaciones, puede ser menos eficiente que otros tipos de sumadores en algunas situaciones debido a que requiere más compuertas lógicas y, por lo tanto, más tiempo para realizar la suma.

   Por ejemplo, cuando se necesita sumar muchos números de bits muy largos, como en operaciones de multiplicación y división de números grandes, se pueden utilizar otros tipos de sumadores. Algunos ejemplos son:

   - *Sumadores en paralelo:* Los sumadores en paralelo suman todos los bits de los números a la vez y pueden ser muy eficientes para sumar números de muchos bits. El diseño de un sumador en paralelo depende del número de bits a sumar, pero en general son más rápidos que los sumadores completos.
   - *Sumadores de anticipación:* Los sumadores de anticipación son una clase de sumadores que aprovechan la relación entre los bits de los números que se suman para hacer que la suma sea más rápida. Estos sumadores son particularmente útiles cuando se deben sumar números con muchos bits, porque la cantidad de compuertas lógicas que se requiere para implementarlos es menor que en otros tipos de sumadores.
   - *Sumadores de carry-skip:* Los sumadores de carry-skip son una variante de los sumadores completos que dividen la suma en bloques de bits y calculan el acarreo de cada bloque antes de sumar los siguientes bits. Este tipo de sumador puede ser más eficiente que el sumador completo en situaciones en las que se requiere sumar muchos números de muchos bits, porque reduce la cantidad de compuertas lógicas necesarias para realizar la operación.

   En general, la elección del tipo de sumador dependerá del número de bits que se deben sumar, la velocidad requerida para la operación y los recursos disponibles para implementar el circuito lógico.[8]

   Por otro lado, si analizamos el sumador de 8 bits construido con base en sumadores de un bit notaremos que no resulta muy práctico. El retardo necesario para que acarreo se propague hasta la salida implicaría pasar por dos niveles de compuertas, al menos, por cada sumador, por lo que se deben pasar por un total de $2n$ compuertas, haciéndolo poco eficiente.[9]

   Este tipo de sumador requiere una gran cantidad de sumadores completos y, por lo tanto, una gran cantidad de compuertas lógicas. Esto aumenta el tiempo y los recursos necesarios para realizar la suma.

   Similarmente, se pueden utilizar otros tipos de sumadores, como los sumadores en paralelo o los sumadores de anticipación, que son más eficientes en términos de tiempo y recursos. Estos sumadores reducen la cantidad de compuertas lógicas necesarias para realizar la suma de los bits y, por lo tanto, son más eficientes que un sumador de 8 bits construido con sumadores completos de un bit.

5. Bajo este diseño, en la ALU se calculan todas las operaciones de forma simultánea, pero solo se entrega un resultado, ¿se realiza trabajo inútil? ¿Toma tiempo adicional? ¿Cuál es el costo?

   Este diseño es conocido como una ALU paralela. En una ALU paralela, todas las operaciones se calculan simultáneamente, pero solo se entrega un resultado. Si se realiza una sola operación, entonces sí, parte del trabajo realizado por la ALU podría considerarse inútil, ya que se están realizando cálculos para otras operaciones que no se están utilizando. Sin embargo, si se están realizando múltiples operaciones en paralelo, entonces la ALU está realizando todo el trabajo requerido y no hay trabajo inútil.

En cuanto al tiempo adicional, una ALU paralela puede requerir un poco más de tiempo para realizar sus cálculos que una ALU secuencial.

Si bien, es cierto que en un diseño de ALU basado en sumadores completos, la cantidad de cálculos que se realizan para obtener el resultado final puede ser mayor en comparación con otros diseños de ALU, lo que podría aumentar el tiempo necesario para completar el cálculo. Sin embargo, esto dependerá de las especificaciones del circuito y del tipo de operaciones que se estén realizando, así como de la velocidad del circuito.[10]

Por otro lado, el costo de este diseño de ALU depende de varios factores, como la cantidad de bits que se están sumando, la velocidad del circuito, el consumo de energía, entre otros. Algunos de los costos asociados con este tipo de diseño pueden incluir:

- *Consumo de energía:* Un diseño de ALU basado en sumadores completos puede requerir más energía para realizar los cálculos que otros diseños, lo que puede aumentar el consumo de energía total del circuito.
- *Área de circuito:* El número de sumadores completos requeridos para un diseño de ALU aumenta con la cantidad de bits que se están sumando. Esto significa que el área del circuito requerida para implementar una ALU basada en sumadores completos puede ser mayor que la requerida para otros diseños de ALU.
- *Velocidad de cálculo:* Debido a que con este diseño realizan todas las operaciones de suma en paralelo, la velocidad de cálculo puede ser más lenta que en otros diseños de ALU que utilizan técnicas de optimización, como sumadores rápidos o sumadores en árbol.
- *Flexibilidad:* Un diseño de ALU como este puede ser menos flexible que otros diseños, ya que se diseñan específicamente para realizar operaciones de suma. Esto significa que pueden requerirse circuitos adicionales para realizar otras operaciones aritméticas, como restas o multiplicaciones.

Un diseño de ALU basado en sumadores completos puede tener algunos costos asociados, como consumo de energía, área de circuito y velocidad de cálculo. Sin embargo, la elección del diseño adecuado de la ALU dependerá de los requisitos específicos de la aplicación y del costo-beneficio de cada diseño en términos de tiempo y recursos.[11]

6. ¿Cuántas operaciones más podemos agregar al diseño de esta ALU? ¿Qué tendríamos que modificar para realizar más operaciones?

   Un sumador completo es solo una operación aritmética que se puede realizar en una ALU. En teoría, se pueden agregar muchas más operaciones aritméticas y lógicas a una ALU, según las necesidades de la aplicación. Algunas de las operaciones aritméticas adicionales que se pueden realizar en una ALU incluyen restas, multiplicaciones, divisiones, comparaciones, y operaciones lógicas como AND, OR, XOR, y otras.

   Para agregar más operaciones a una ALU, se pueden agregar más circuitos lógicos y aritméticos al diseño existente. Cada operación aritmética o lógica requerirá un circuito diferente y específico. Por ejemplo, para agregar la operación de resta, se puede agregar un circuito complemento a dos y un circuito de suma. Para agregar la operación de multiplicación, se puede agregar un circuito multiplicador, como un multiplicador de Booth o un multiplicador de Wallace.

   En resumen, el número de operaciones que se pueden agregar a una ALU depende del diseño específico de la ALU y de las necesidades de la aplicación. Se pueden agregar muchas operaciones adicionales, pero cada operación requerirá un circuito adicional, lo que aumentará el tamaño y la complejidad del diseño.[12]

Referencias

[1] Galaviz, C. J. (n.d) *Elementos cuantitativos de diseño de computadoras.* p.3

[2] An Introduction to VHDL

[3] VHDL History
[4] How to Implement a Full Adder in VHDL
[5] Morris M. Mano (1992) 3rd Edition *Computer System Architecture*. Prentice Hall India.
[6] Arithmetic Logic Unit
[7] Basic Logical Operations
[8] Mano, M. (2023e). Logic And Computer Design Fundamentals 4E (3.a ed.). Pearson India.
[9] Galaviz, C. J. (n.d) *Lógica digital y diseño de circuitos digitales* p.29
[10] Parallel ALU Design
[11] Arithmetic Logic Unit Design
[12] Full Adder Extension