Tecnológico de Monterrey

Campus Monterrey

Materia

Arquitectura de computadoras (Gpo 1)

Tema

Práctica #2. La Unidad Aritmético-Lógica

Integrantes

- Jorge Besnier A01039882
- Cinthia Portillo A00811827

Maestro

Diego Valencia

Fecha
01/09/20

Introducción

En esta práctica simularemos un ALU (Arithmetic-logic unit) que es un componente muy importante en la unidad central de proceso ya que permite realizar operaciones aritméticas y lógicas. Para nuestra práctica sólo se realizarán las operaciones de la tabla 1.1. Por lo tanto el documento tiene la siguiente estructura de contenido:

Desarrollo

Módulo ALU

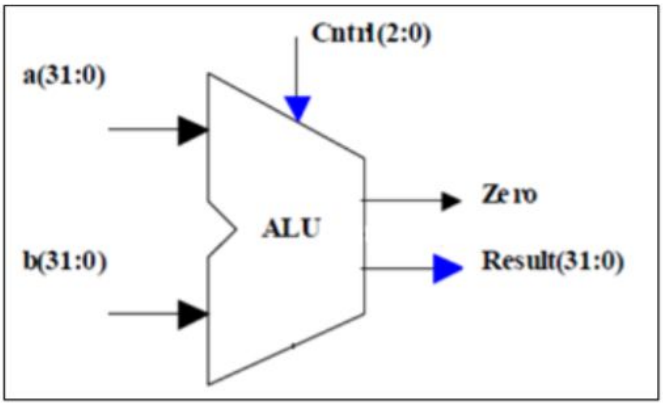

Figura 1.1. Esquema de la Unidad Aritmética Lógica

Tabla 1.1 Operaciones de la Unidad Aritmética Lógica		
ALU Control Lines	Función	Operación
000	And	Result = a and b
001	Or	Result = a or b
010	Add	Result = a + b
011	Mov	Result = a
100	B upper	Result = b[15:0] & x"0000"
110	Substract	Result = a - b
111	Set less than	Result = Consultar más adelante

La bandera de Zero se habilita siempre y cuando ocurra una operación (cualquiera que sea) donde el resultado sea 0, por ejemplo, Result =a – b = 0 , Zero = '1'.

Implementación

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Malu is
    Port ( a : in  STD_LOGIC_VECTOR(31 downto 0);
           b : in  STD_LOGIC_VECTOR(31 downto 0);
           cntrl : in  STD_LOGIC_VECTOR(2 downto 0);
           zero : out  STD_LOGIC;
           result : inout  STD_LOGIC_VECTOR(31 downto 0));
end Malu;

architecture Behavioral of Malu is
signal sub, compare: STD_LOGIC_VECTOR(31 downto 0);

begin

   with cntrl select
      result <=   (a and b) when "000",
                  (a or b) when "001",
                  (a + b) when "010",
                  (a) when "011",
                  (b(15 downto 0) & x"0000") when "100",
                  (a - b) when "110",
                  compare when "111",
                  x"ffffffff" when others;

   compare <= x"00000001" when (a < b) else x"00000000";
   zero <= '1' when (result = x"00000000") else '0';


end Behavioral;
```

Figura 1.3. Código fuente ALU

```vhdl
ARCHITECTURE behavior OF tbMalu IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Malu
    PORT(
         a : IN  std_logic_vector(31 downto 0);
         b : IN  std_logic_vector(31 downto 0);
         cntrl : IN  std_logic_vector(2 downto 0);
         zero : OUT  std_logic;
         result : INOUT  std_logic_vector(31 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal a : std_logic_vector(31 downto 0) := (others => '0');
   signal b : std_logic_vector(31 downto 0) := (others => '0');
   signal cntrl : std_logic_vector(2 downto 0) := (others => '0');

   --Outputs
   signal zero : std_logic;
   signal result : std_logic_vector(31 downto 0);


BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: Malu PORT MAP (
          a => a,
          b => b,
          cntrl => cntrl,
          zero => zero,
          result => result
        );
    -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
         cntrl<= "000";
         a<= x"00000001";
         b<= x"00000101";
      wait for 100 ns;
         cntrl<= "001";
         a<= x"00000001";
         b<= x"00000101";
      wait for 100 ns;
         cntrl<= "010";
      wait for 100 ns;
         cntrl<= "011";
      wait for 100 ns;
         cntrl<= "100";
      wait for 100 ns;
         a<= x"00000001";
         b<= x"00000101";
         cntrl<= "111";
      wait for 100 ns;
         a<= x"00000101";
         b<= x"00000001";
         cntrl<= "111";
      wait for 100 ns;
         a<= x"00000001";
         b<= x"00000101";
         cntrl<= "110";
      wait for 100 ns;
         a<= x"00000101";
         b<= x"00000001";
         cntrl<= "110";

      wait;
   end process;

END;
```

Figura 1.4: Código Fuente de Test Bench

Simulación

Función AND

Para esta operación, AND actúa sobre los operandos a y b, obteniendo su resultado de acuerdo a la siguiente tabla de verdad:

A	B	Result=A and B
0	0	0
0	1	0
1	0	0
1	1	1

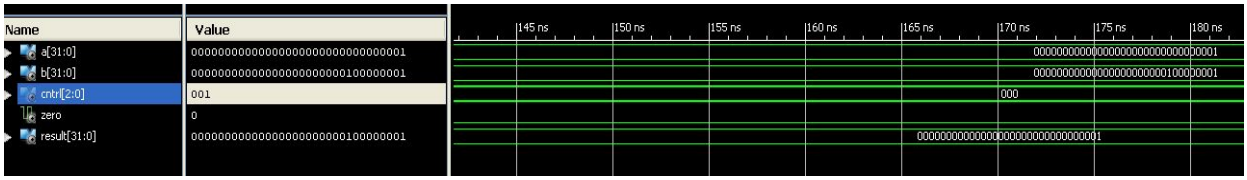

Figura 1.5. Simulación función AND

En la figura 1.5 podemos ver el resultado del "and" aplicado entre los 32 bits de A y los 32 bits de B, para esta simulación se puede ver que como el resultado es diferente de 0, la bandera z=0

Función OR

Para esta operación, OR actúa sobre los operandos a y b, obteniendo su resultado de acuerdo a la siguiente tabla de verdad:

A	B	Result=A OR B
0	0	0
0	1	1
1	0	1
1	1	1

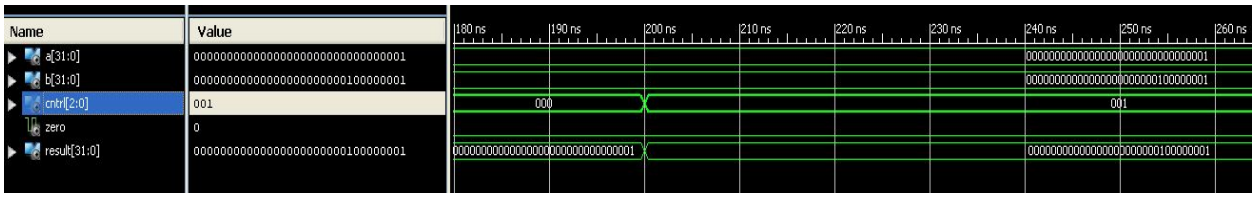

Figura 1.6. Simulación función OR

En la figura 1.6 podemos ver el resultado del “or” aplicado entre los 32 bits de A y los 32 bits de B, para esta simulación se puede ver que como el resultado es diferente de 0, la bandera z=0

Función ADD

Para esta función se tienen como entradas 2 números binarios de 32 bits, y como resultado la suma binaria de estos números.

A	0	1	1	0
B	0	0	1	1
Result A+B	1	0	0	1

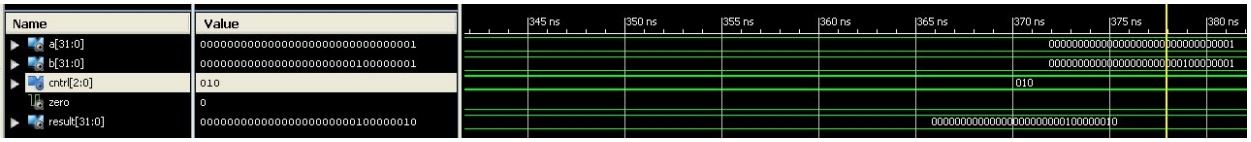

Figura 1.7. Simulación función AND

En la figura 1.7 podemos ver el resultado de la función ADD entre A y B, para esta simulación se puede ver que se suma 257 + 1, obteniendo un 258 en binario, y como el resultado es diferente de 0, la bandera z=0.

Función MOV

Para esta función se tienen como entradas 2 números binarios de 32 bits (a y b), y como resultado siempre obtendremos el valor de entrada de ‘a’.

A	0	1	1	0
B	0	0	1	1
Result A	0	1	1	0

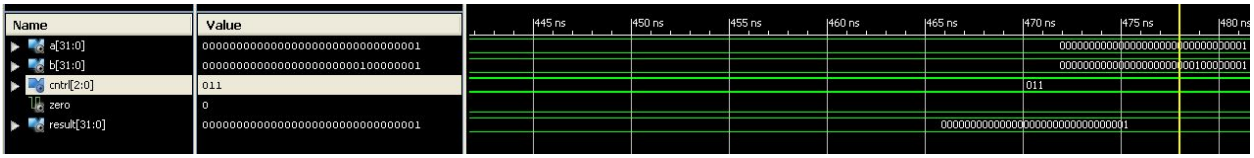

Figura 1.8. Simulación función MOV

En la figura 1.8 podemos ver el resultado de la función MOV entre A y B, para esta simulación se puede ver que se cualquiera que sea el valor de B, el resultado siempre será el valor de A. Para este caso el resultado es diferente de 0 por lo tanto la bandera z=0.

Función B upper

Para esta función se tienen como entradas 2 números binarios de 32 bits (a y b), y como resultado siempre obtendremos el valor de la primera mitad del b de 0 a 15 bits a los cuales después se le concatena 16 bits de ceros a la derecha.

A	0	1	1	0
B	0	0	1	1
Result B[First Half]&0	1	1	0	0

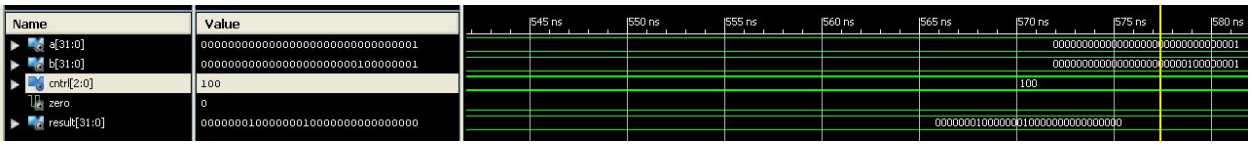

Figura 1.9. Simulación función MOV

En la figura 1.9 podemos ver el resultado de la función B upper entre A y B, para esta simulación se puede ver que se cualquiera que sea el valor de A, el resultado será el valor de los primeros 16 bits de B más los 16 '0's que se concatenan a la derecha. Para este caso el resultado es diferente de 0 por lo tanto la bandera z=0.

Función Substract

Para esta función se tienen como entradas 2 números binarios de 32 bits, y como resultado la resta binaria de estos números en el caso que sea a>b. Por otra parte, en el caso que b > a vhdl realiza una resta en complemento a 2. (Según la librería puede ser otro el resultado.)

A	0	0	1	1
B	0	0	1	0
Result A-B	0	0	0	1

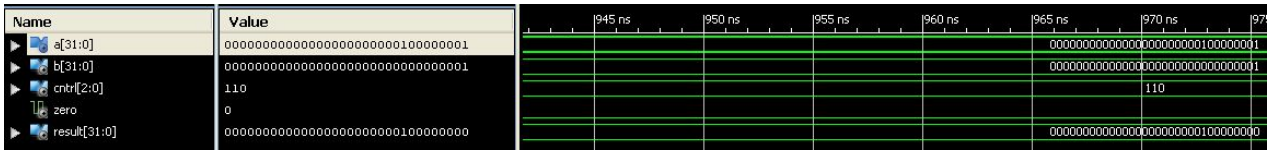

Figura 1.10. Simulación función Substract

En la figura 1.10 podemos ver el resultado de la función substract aplicada entre A y B, para esta simulación se puede ver que A>B. Para este caso el resultado es diferente de 0 por lo tanto la bandera z=0.

Figura 1.11. Simulación función Substract

En la figura 1.11 podemos ver el resultado de la función subtract aplicada entre A y B, para esta simulación se puede ver que A<B. Para este caso el resultado es diferente de 0 por lo tanto la bandera z=0.

Función Set less than

La operación set less than pregunta si a < b, entonces Result = x"00000001"; en caso contrario, Result = x"00000000".

A	0	0	0	1
B	0	0	1	1
Result A<B	0	0	0	1

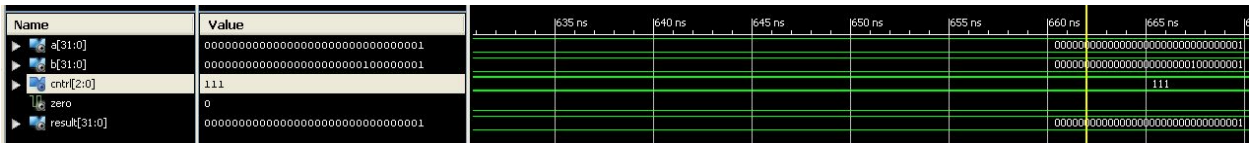

Figura 1.12. Simulación función Set less than

En la figura 1.12 podemos ver el resultado de la función set less than aplicada entre A y B, para esta simulación se puede ver que A<B por lo que el resultado es 1, y por lo tanto la bandera z=0.

A	0	1	1	0
B	0	0	1	1
Result A>B	0	0	0	0

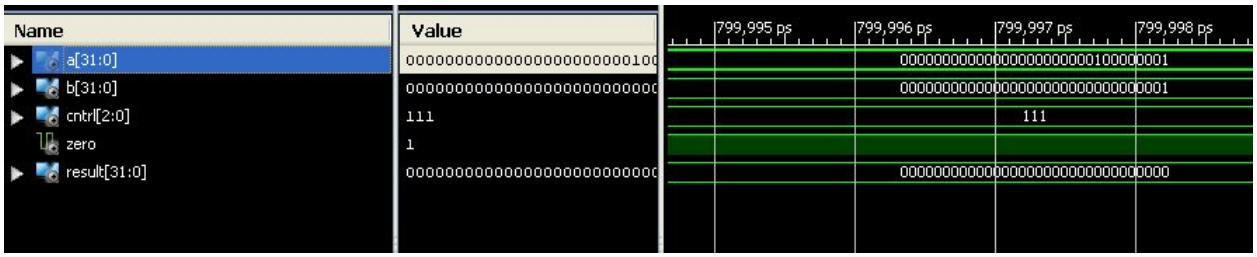

Figura 1.13. Simulación función Set less than

En la figura 1.13 podemos ver el resultado de la función set less than aplicada entre A y B, para esta simulación se puede ver que A>B por lo que el resultado es 0, y por lo tanto la bandera z=1.

Conclusión

Jorge

De esta práctica, me llevó el aprendizaje del funcionamiento básico de una unidad aritmética y especialmente cómo aprovechar la concurrencia que permite el lenguaje de vhdl para crear procesos independientes a una señal de reloj. Finalmente, la reflexión particular de la práctica es leer muy bien las instrucciones, ya que puede llevarnos a desarrollar código que no es necesario.

Cinthia

En esta práctica reforzamos de manera concreta cómo funciona un ALU, como su mismo nombre lo indica, para realizar operaciones tanto lógicas como aritméticas, y verificando el diagrama de MIPS, dichas operaciones se realizan con los datos transferidos de la unidad de control. Además, aprendimos que para una resta cuando el minuendo es mayor que el sustraendo, la resta se hace en complemento a 2. Algo en lo que podemos aplicar lo aprendido, es que nos puede ser más fácil comprender cómo funcionan los múltiples dispositivos a los que están incorporados en las computadoras modernas que tienen múltiples núcleos. El interés que despertó en mí esta actividad es querer conocer más allá los componentes de una computadora, no sólo identificarlos sino conocer con más detalle su comportamiento.