دانشکده مهندسی کامپیوتر

استاد : خانم دکتر پریا دربانی

سید مهدی رضوی

دی ۱۴۰۱

فهرست مطالب

فهرست تصاویر

۱ مقدمه

در این آزمایش به پیاده‌سازی یک واحد محاسبه و منطق ALU می‌پردازیم.
هر دستورالعمل ما ۴ بیتی می‌باشد که در ابتدا عملگرهای ریاضی(محاسبه) و سپس عملگرهای منطقی را خواهیم‌داشت.
عملکرد یک کامپیوتر پایه بدین شکل است که در یک حلقه :
دستورالعمل را واکشی می‌کند(fetch)
سپس رمزگشایی می‌کند.(decode)
و در نهایت نیز اجرا خواهد کرد.(execute)
سپس ما پیاده‌سازی خود را به ازای دستورالعمل منطقی XOR تست خواهیم‌کرد.

۲ پیاده سازی با زبان VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.NUMERIC_STD.all;

entity ALU is
  generic (
     -- NUMBER OF SHIFTED BITS FOR LOGICAL SHIFT OPERATIONS
     constant N: natural := 1
    );

    Port (
        -- 2 operands 8-bit
        op1 , op2 : in STD_LOGIC_VECTOR(7 downto 0);

        -- 1 input 4-bit (IR or PC Register)
        Instruction_Register : in STD_LOGIC_VECTOR(3 downto 0);

        -- 1 output 8-bit
        ALU_Out  : out STD_LOGIC_VECTOR(7 downto 0);

        -- Carryout flag
        Carryout : out std_logic
    );
end ALU;
```

در ابتدای کار می‌بایستی پورت‌های ورودی و خروجی را تعیین کنیم. پورت آدرس ما با توجه به مطلوب پروژه ۴ بیتی خواهدبود. (Instruction Register)
سپس عملگرها را بر روی عملوندهای ۸ بیتی انجام‌خواهیم داد. به ازای عملگرهای حسابی و یا بعضی از شیفت‌های منطقی ما بیت CarryOut نیز خواهیم داشت.

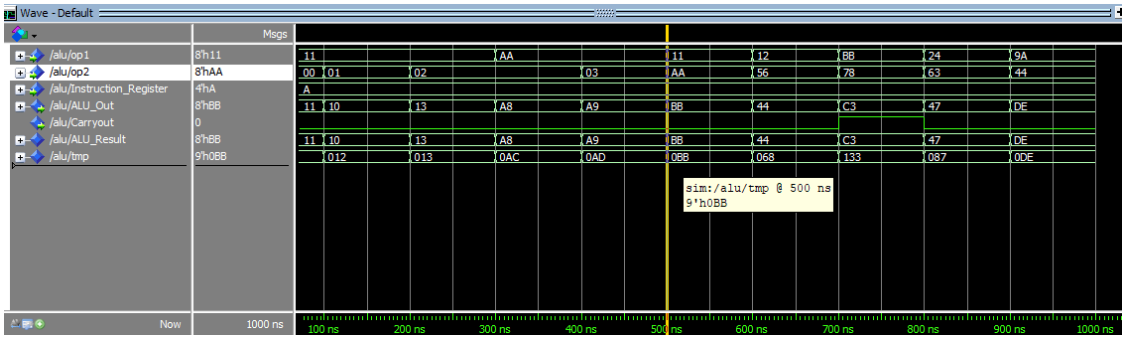
```
architecture Behavioral of ALU is

signal ALU_Result : std_logic_vector (7 downto 0);
signal tmp: std_logic_vector (8 downto 0);

begin
   process(op1 , op2 , Instruction_Register)
 begin
  case(Instruction_Register) is
  -- Addition
  when "0000" =>
   ALU_Result <= op1 + op2 ;

   -- Subtraction
  when "0001" =>
   ALU_Result <= op1 - op2 ;

   -- Multiplication
  when "0010" =>
   ALU_Result <= std_logic_vector(to_unsigned((to_integer(unsigned(op1)) *
       to_integer(unsigned(op2))),8)) ;

  -- Division
  when "0011" =>
   ALU_Result <= std_logic_vector(to_unsigned(to_integer(unsigned(op1)) /
       to_integer(unsigned(op2)),8)) ;
```

در قطعه کد بالا نیز در Process بعضی عملگرهای محاسباتی را مشاهده می‌کنیم.

```
-- Logical shift left
  when "0100" =>
   ALU_Result <= std_logic_vector(unsigned(op1) sll N);

  -- Logical shift right
  when "0101" =>
   ALU_Result <= std_logic_vector(unsigned(op1) srl N);

 -- Rotate left
  when "0110" =>
   ALU_Result <= std_logic_vector(unsigned(op1) rol N);

  -- Rotate right
  when "0111" =>
   ALU_Result <= std_logic_vector(unsigned(op1) ror N);

  -- Logical and
  when "1000" =>
   ALU_Result <= op1 and op2;

  -- Logical or
  when "1001" => -- Logical or
   ALU_Result <= op1 or op2;

  -- Logical xor
  when "1010" => -- Logical xor
   ALU_Result <= op1 xor op2;

  -- Logical nor
  when "1011" =>
   ALU_Result <= op1 nor op2;
```

```
-- Logical nand
  when "1100" =>
   ALU_Result <= op1 nand op2;

   -- Logical xnor
  when "1101" =>
   ALU_Result <= op1 xnor op2;

   -- Greater comparison
  when "1110" =>
   if(op1 > op2) then
    ALU_Result <= x"01" ;
   else
    ALU_Result <= x"00" ;
   end if;

  -- Equal comparison
  when "1111" =>
   if(op1 = op2) then
    ALU_Result <= x"01" ;

   else
    ALU_Result <= x"00" ;
   end if;

  when others => ALU_Result <= op1 + op2 ;
  end case;
 end process;

-- ALU out
 ALU_Out <= ALU_Result;
 tmp <= ('0' & op1) + ('0' & op2);


  -- Carryout flag
 Carryout <= tmp(8);
end Behavioral;
```

در قطعه کد این صفحه و صفحه قبل نیز بعضی از عملگرهای منطقی را مشاهده می‌کنید. با توجه به اینکه می‌خواهیم عملگر XOR را ارزیابی کنیم باید مقدار ثبات دستوالعمل ما مقدار ۱۰۱۰ داشته باشد.



شکل ۱: شکل موج خروجی حاصل از عملیات XOR منطقی

آخرین مقداردهی به دستورالعمل XOR در شکل موج خروجی بالا را مشاهده می‌کنید.

$$(9A)_{Hex} \oplus (44)_{Hex} =$$

$$(1001, 1011)_{Binary} \oplus (0100, 0100)_{Binary} =$$

$$(1101, 1111)_{Binary} =$$

$$(DE)_{Hex}$$

۳ نتیجه‌گیری

در این آزمایش با بررسی چندین دستورالعمل منطقی و حسابی توانستیم یک کامپیوتر پایه را تشکیل دهیم و عملگر XOR منطقی را در آن تست کنیم.