รายงาน

เรื่อง 8 bit simple process

จัดทำโดย

นาย ศุภณัฐ โกศัลวิตร

รหัสนักศึกษา 5901012630130

เสนอ

อาจารย์ อรอุมา เทศประสิทธิ์

รายงานนี้เป็นส่วนหนึ่งของรายวิชา Computer Organization รหัสวิชา 010123134

ภาคเรียนที่ 2 ปีการศึกษา 2560 คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

คำนำ

รายงานนี้เป็นส่วนหนึ่งของวิชา Computer Organization ซึ่งจัดทำขึ้นเพื่อศึกษาและออกแบบหน่วยประมวลผลขนาด 8 bit โดยการเขียนภาษาระดับสูง ลงมาถึงภาษาระดับล่าง(MIPS) ซึ่งภายในรายงานนี้ได้มีการออกแบบโครงสร้างลอจิกโดยการเขียนด้วย ภาษา VHDL หากมีข้อผิดพลาดประการใดขออภัยมา ณ ที่นี้ด้วย

ผู้จัดทำ

วัตถุประสงค์

เพื่อเป็นการออกแบบและเรียนรู้โครงสร้างภายในของหน่วยประมวลผล ซึ่งจะเริ่มตั้งแต่ภาษาระดับสูงและแปลงไปเป็น Assembly

คำสั่งที่จากภาษาระดับสูงแปลงเป็นภาษา Assembly(MIPS)

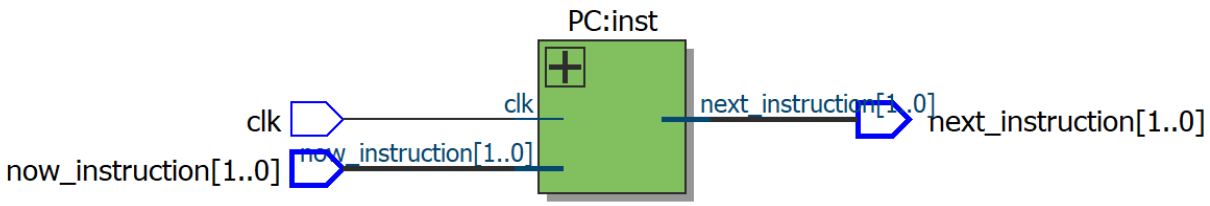
High Level Languange	Assembly(MIPS)
A + B = C	ADD $s3 $s1 $s2
A – C = D	SUB $s4 $s1 $s3
A AND B = C	AND $s3 $s1 $s2
A OR C = D	OR $s4 $s1 $s3

องค์ประกอบมีทั้งหมด 5 ส่วนดังนี้

- Program counter (PC)
- Instruction memory
- Control unit
- Register
- Arithmetic and logic unit (ALU)

รายละเอียดของแต่ละองค์ประกอบ

Program Counter (PC)



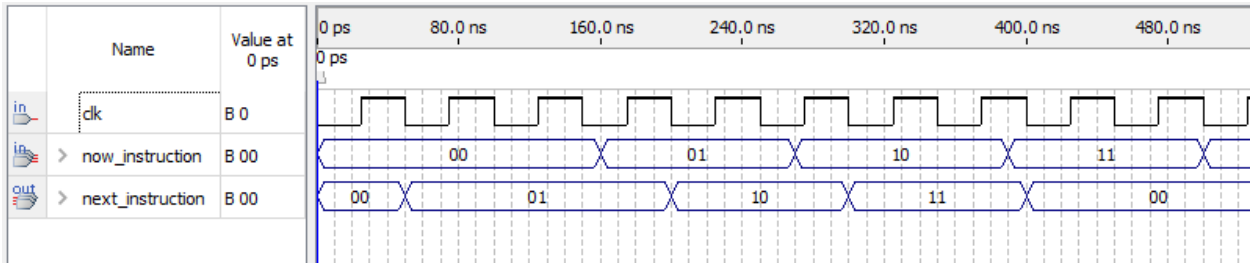
Code VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity PC is
  port(clk : in std_logic;
       now_instruction : in std_logic_vector(1 downto 0);
       next_instruction : out std_logic_vector(1 downto 0)
  );
end PC;

architecture Behavioral of PC is
signal next_signal : std_logic_vector(1 downto 0);
begin
  process(clk)
  begin
    if falling_edge(clk) then
      next_signal <= std_logic_vector(unsigned(now_instruction) + to_unsigned(1,2));
    end if;
  end process;
  next_instruction <= next_signal;
end Behavioral;
```

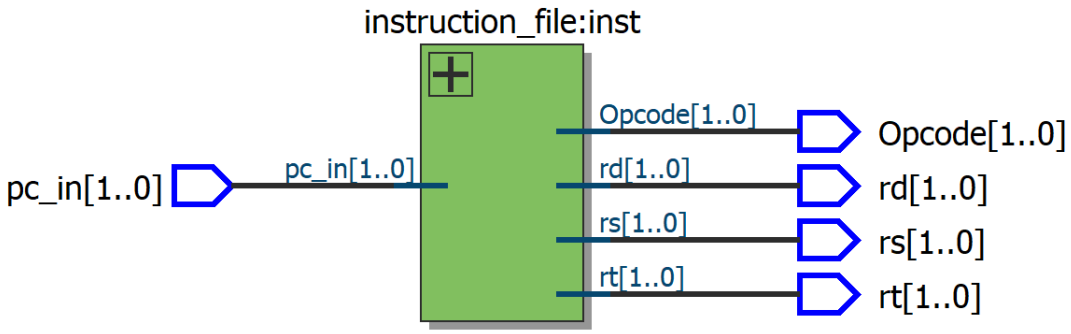
อธิบายการทำงาน

รับค่า input เป็น สัญญาณนาฬิกาโดยพิจารณาจากขอบขาลง และ output(next_instruction) ที่ออกจะมีขนาด 2 bit ส่งไปที่ instruction memory พร้อมทั้งป้อนสัญญาณกลับไปเป็นสัญญาณ input อีกตัว(now_instrucction)

ผลการจำลอง


Name
Value at 0 ps
clk
B 0
now_instruction
B 00
next_instruction
B 00
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
00
01
10
11
00
01
10
11
00

Instruction memory



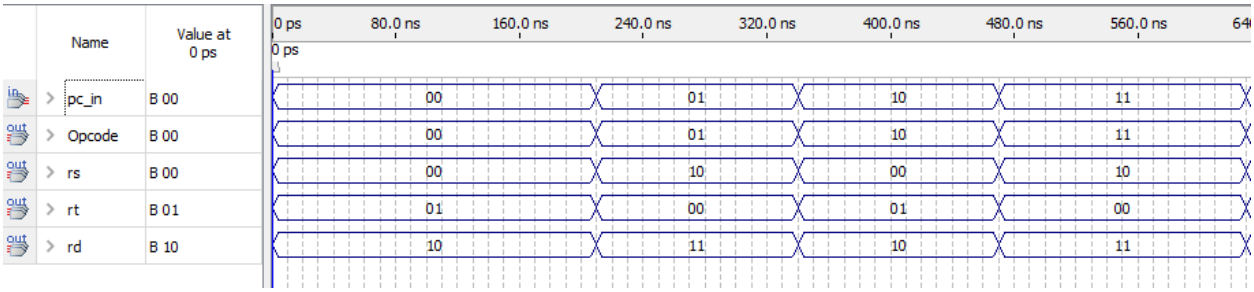
Code VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity instruction_file is
port(
   pc_in : in std_logic_vector(1 downto 0);
   Opcode : out std_logic_vector(1 downto 0);
   rs : out std_logic_vector(1 downto 0);
   rt : out std_logic_vector(1 downto 0);
   rd : out std_logic_vector(1 downto 0)
   --Opcode_check : out std_logic_vector(1 downto 0)
);
end instruction_file;
architecture Behavioral of instruction_file is
--signal pc_addr : std_logic_vector(1 downto 0);
--signal op : std_logic_vector(1 downto 0);
type instruc_type is array (0 to 3) of std_logic_vector(7 downto 0);
constant data_sel : instruc_type:=(
   "00000110", -- add $s0 $s1 $s2
   "01100011", -- sub $s0 $s1 $s2
   "10000110", -- and $s0 $s1 $s2
   "11100011"  -- or  $s0 $s1 $s2
   );
begin
   Opcode <= data_sel(to_integer(unsigned(pc_in)))(7 downto 6);
   rs <= data_sel(to_integer(unsigned(pc_in)))(5 downto 4);
   rt <= data_sel(to_integer(unsigned(pc_in)))(3 downto 2);
   rd <= data_sel(to_integer(unsigned(pc_in)))(1 downto 0);
   --Opcode <= op;
   --Opcode_check <= op;
end Behavioral;
```

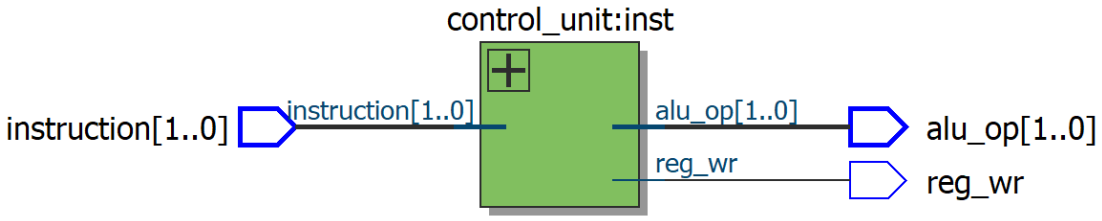
อธิบายการทำงาน

รับค่า input(pc_in) มาจาก PC ซึ่งมีขนาด 2 bit และ output ออกเป็น Opcode , rs ,rt และ rd ซึ่งใน instruction นี้ จะมีการแบ่งคำสั่งออกเป็นอย่างละ 2 bit โดยที่ 2 bit ด้านซ้ายจะเป็น opcode 2 bit ถัดมาจะเป็น rs , rt และ rd ตามลำดับ

ผลการจำลอง



Control unit



Code VHDL

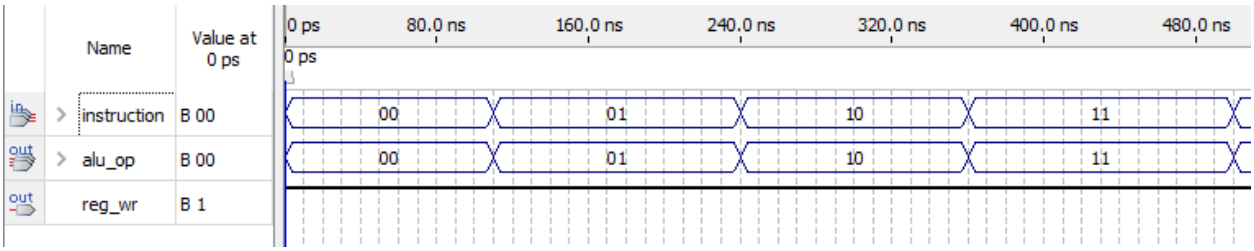
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity control_unit is
port( instruction : in std_logic_vector(1 downto 0);
      reg_wr      : out std_logic; -- if 1 avtive write register
      alu_op      : out std_logic_vector(1 downto 0)
  );
end control_unit;

architecture dataflow of control_unit is
begin
  with instruction select
    reg_wr <= '1' when "00", -- add
              '1' when "01", -- sub
              '1' when "10", -- and
              '1' when "11"; -- or
  with instruction select
    alu_op <= "00" when "00",
              "01" when "01",
              "10" when "10",
              "11" when "11";
end dataflow;
```

อธิบายการทำงาน

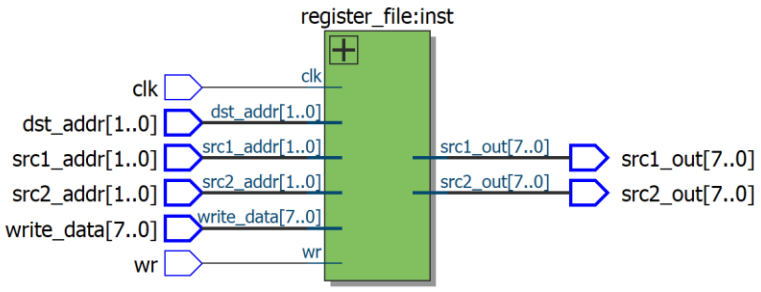
รับค่า input(instruction) มาเป็นขนาด 2 bit และส่ง output ออกไปได้แก่ reg_wr และ alu_op ซึ่ง reg_wr จะมีสถานะลอจิกคือ 0 และ 1 แต่ในที่นี้จะมีสถานะลอจิกเป็น 1 ทั้งหมดซึ่งจะส่งสถานะลอจิก 1 ไปที่ register file เพื่อเป็นการกำหนดให้ active การ write data ลงไปใน register

ส่วน alu_op จะทำหน้าที่ส่ง address ขนาด 2 bit ไปให้ Arithmetic and logical unit (ALU) เพื่อเป็นการกำหนดว่าจะทำคำสั่งใดใน ALU

ผลการจำลอง



Register



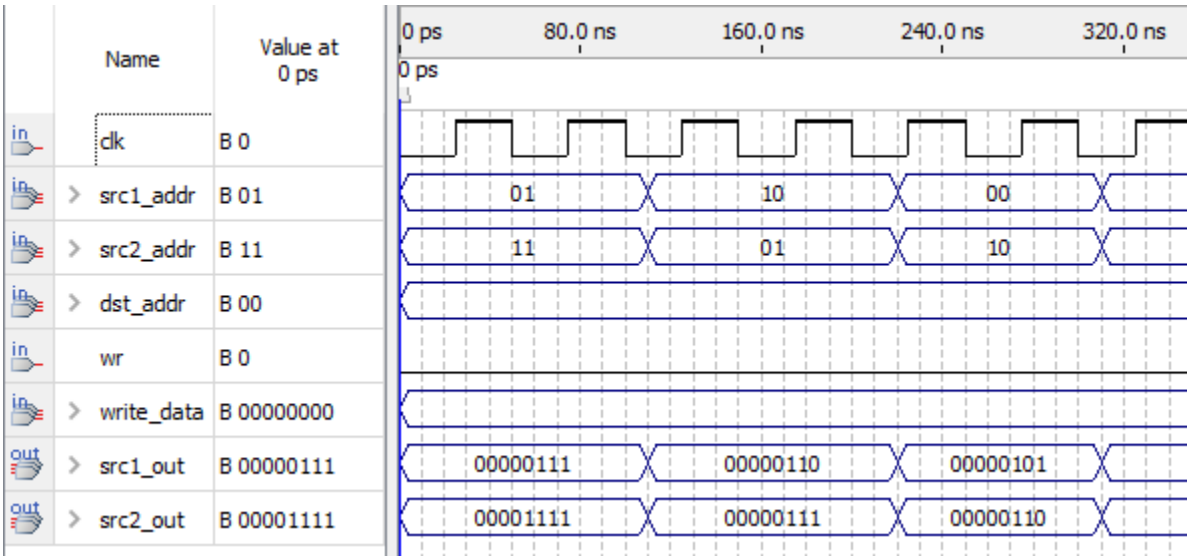
Code VHDL

```
 4  entity register_file is
 5  port(
 6     clk : in std_logic;
 7     wr  : in std_logic; -- write control
 8     write_data : in std_logic_vector(7 downto 0); -- write data to destination
 9
10     src1_addr : in std_logic_vector(1 downto 0); -- source 1 address
11     src2_addr : in std_logic_vector(1 downto 0); -- source 2 address
12     dst_addr : in std_logic_vector(1 downto 0); -- destination address
13
14     src1_out : out std_logic_vector(7 downto 0);-- source 1 address
15     src2_out : out std_logic_vector(7 downto 0) -- source 2 address
16  );
17  end register_file;
18  architecture Behavioral of register_file is
19  type registerFile is array (0 to 3) of  std_logic_vector(7 downto 0);
20  signal reg : registerFile := (
21        "00000101",
22        "00000111",
23        "00000110",
24        "00001111"
25        );
26  begin
27    process(clk)
28    begin
29      if falling_edge(clk) then
30        if(wr = '1') then
31          reg(to_integer(unsigned(src1_addr)))<= write_data;
32        end if;
33      end if;
34    end process;
35    src1_out <= reg(to_integer(unsigned(src1_addr)));
36    src2_out <= reg(to_integer(unsigned(src2_addr)));
37  end Behavioral;
```

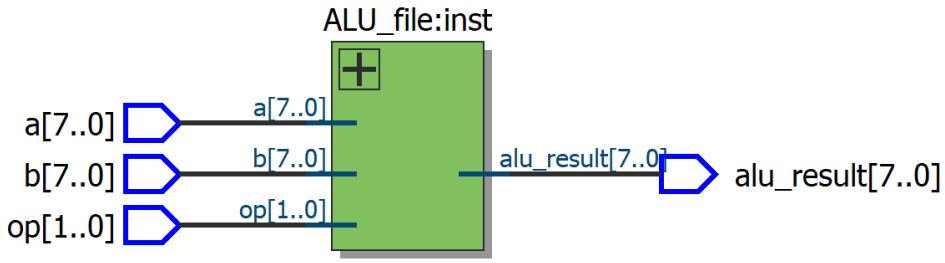
อธิบายการทำงาน

รับค่า input มาเป็นสัญญานาฬิกา (clk) โดยพิจารณาที่ขอบขาลงนอกจากนี้ยังรับ input มาเป็น address ของ rs, rt, rd (จาก code จะใช้เป็น src1_addr, src2_addr และ dst_addr ตามลำดับ) และเมื่อ wr มีสถานะลอจิกเป็น 1 จะกำให้ write_data เขียนไว้ที่ src1_addr และหลังจากนั้นก็ส่ง output ขนาด 8 bit ออกมาคือ src1_out และ src2_out ตามลำดับ ซึ่งน register นี้ได้ทำการกำหนดข้อมูลไว้ภายในแล้ว

ผลการจำลอง



Arithmetic and logical unit (ALU)



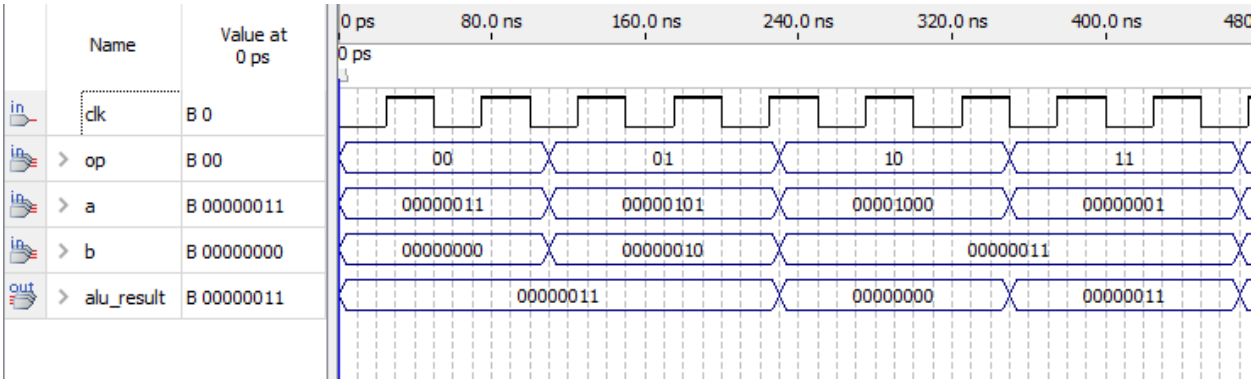
Code VHDL

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity ALU_file is
port(

  a, b : in std_logic_vector(7 downto 0); -- src1, src2
  op : in std_logic_vector(1 downto 0); -- select function
  alu_result : out std_logic_vector(7 downto 0)
);
end ALU_file;

architecture Behavioral of ALU_file is
signal result : std_logic_vector(7 downto 0);
begin
  process(op,a,b)
    begin
      case op is
        when "00" =>
          result <= a + b;
        when "01" =>
          result <= a - b;
        when "10" =>
          result <= a and b;
        when "11" =>
          result <= a or b;
       end case;
  end process;
  alu_result <= result;
end Behavioral;
```

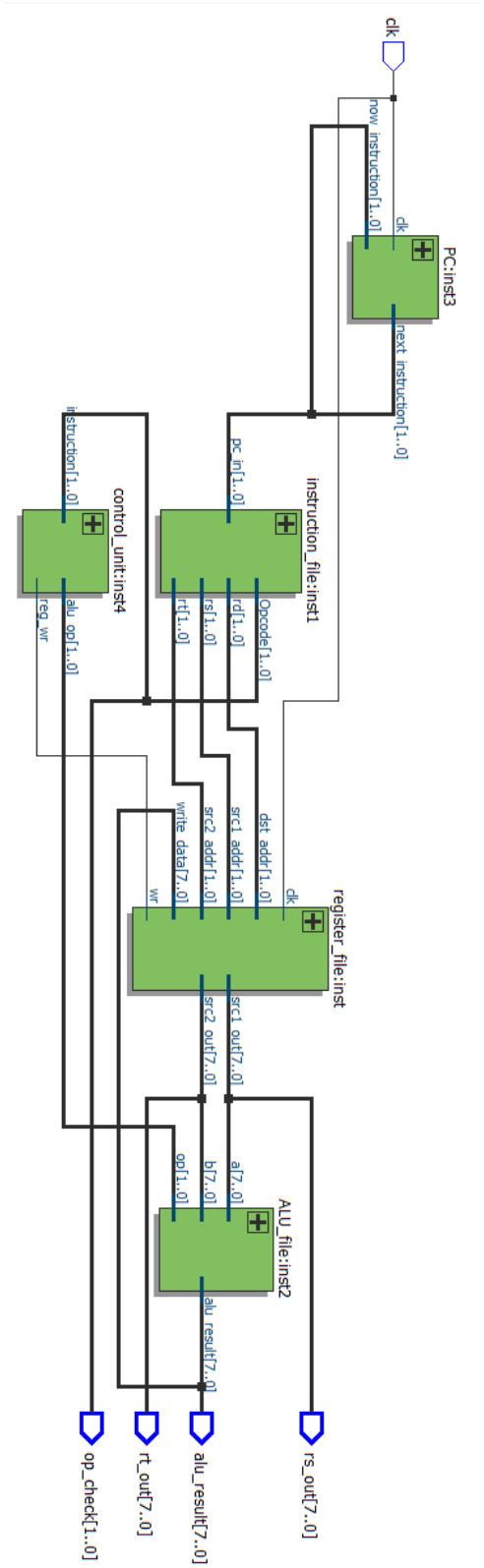
อธิบายการทำงาน

รับค่า input เป็นข้อมูลขนาด 8 bit และ op ขนาด 2 bit ซึ่ง op คือที่รับมาจะเป็น address เพื่อระบุว่าจะทำงานคำสั่งใด ถ้าเป็น “00” จะเป็น add “01” จะเป็น sub “10” จะเป็น or และ “11” จะเป็น and เพื่อคำสั่งเสร็จแล้วจะทำการส่ง output ออกไปเป็น ขนาด 8 bit

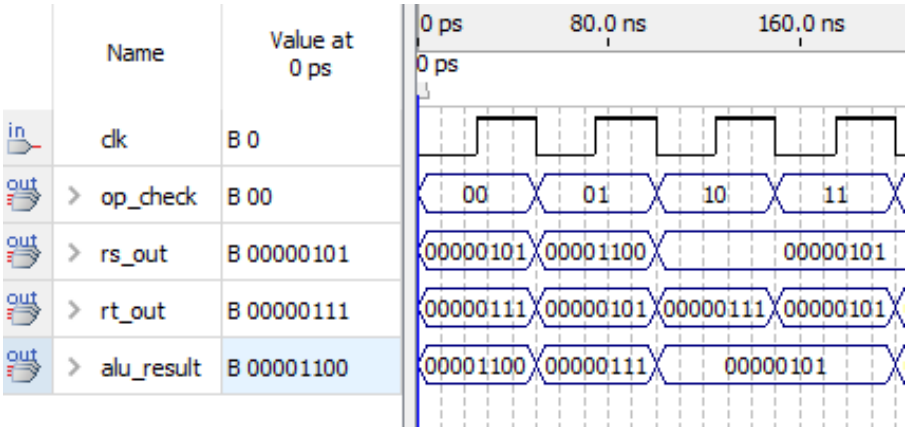
ผลการจำลอง



ภาพจำลอง RTL Viewer



ผลการจำลองรวมทุกองค์ประกอบ



00	01	10	11
ADD	SUB	AND	OR

เมื่อพิจารณาสัญญาณ clk ตมขอบขาลง เมื่อ op_check เป็น “01” จะทำการ ADD กันระหว่าง rs_out กับ rt_out และได้ผลลัพธ์เป็น alu_reult หลังจากนั้น เมื่อ op_check เป็น “01” จะทำการนำผลลัพธ์เมื่อ op_check มีสถานะเป็น “00” มาใช้ต่อโดยเก็บค่าไว้ที่ rs_out ซึ่งสถานะ “01” คือการทำ SUB หลังจากนั้นก็จะทำ SUB ระหว่าง rs_out และ rt_out เมื่อเสร็จแล้ว op_check จะไปที่สถานะ “10” และทำคำสั่ง AND ซึ่ง rs_out ในที่นี้จะไปนำข้อมูลมาจาก register file และหลังจากนั้นเมื่อทำคำสัง AND เสร็จ op_check ก็จะมีสถานะเป็น “11” ซึ่งเป็นคำสั่ง OR หลังจากนั้น rc_out จะเป็นค่าของ alu_result เมื่อตอนทำคำสั่ง AND และหลังจากที่คำสั่ง OR เสร็จแล้ว op_check ก็จะกลับไปที่สถานะ “00” และทำงานตามลำดับที่ได้กล่าวไปในข้างต้น