A simple 8-bit microprocessor

A project by,

Aravind Kumar S(2013105004),
Ashvath Nirmal(2013105005),
Bharath A(201315006),
Gautham C K(2013105507)

In fulfillment of VLSI design laboratory,
Submitted on,
1.4.2016.

Synopsis:

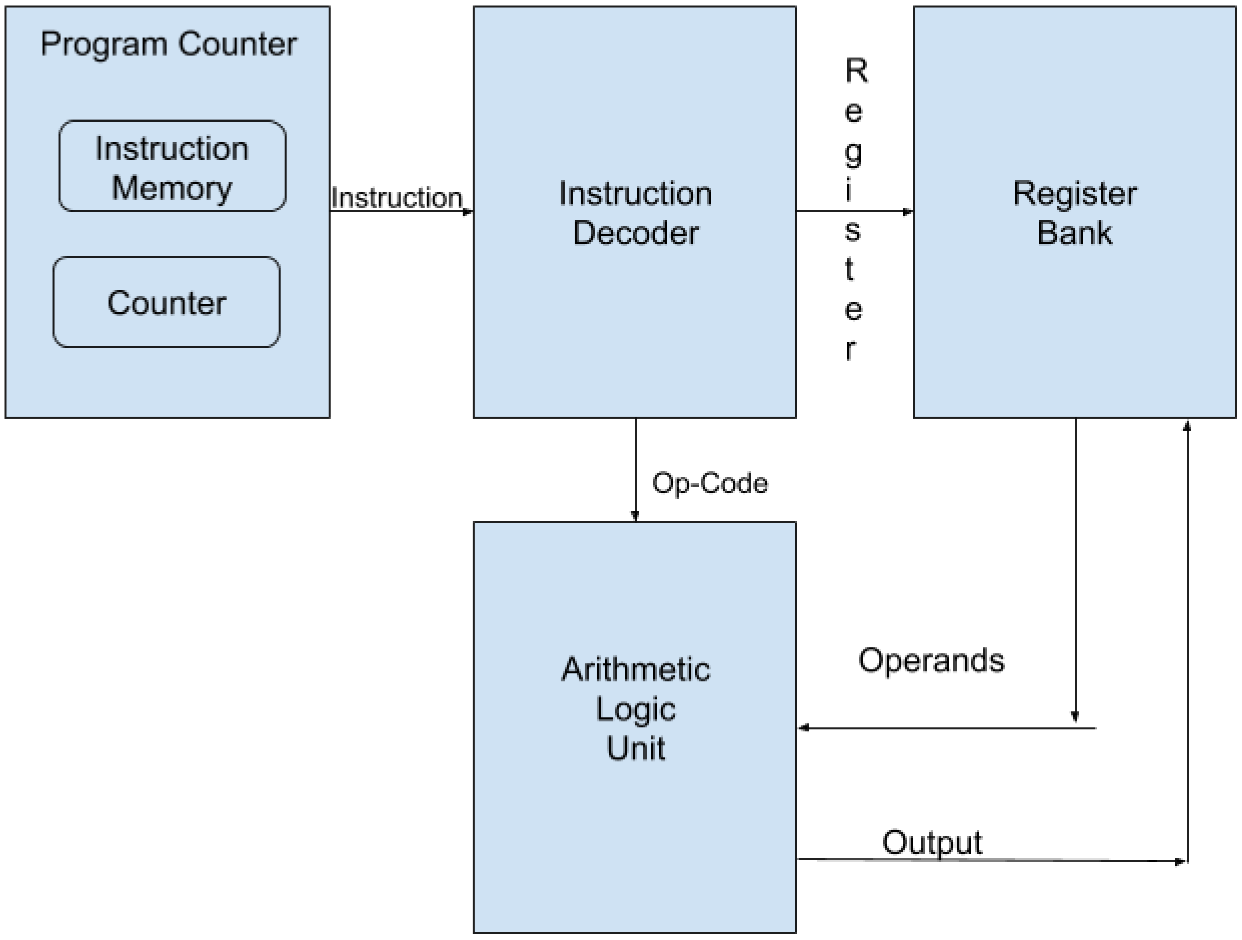
Introduction:

Our project is a small 8-bit microprocessor designed in VHDL.It contains relatively the main components what the early microprocessors had such as the ALU, program counter, instruction decoder, register bank, counter and instruction memory.

Our processor is an 8 bit processor and hence all the registers are 8bits wide and the size of the instruction is 8bits.The processor is capable of performing 4 operations such as,

1. Addition
2.Subtraction
3.Logical OR
4.Load Immediate

Block Diagram:



Instruction Set Architecture:

Each instruction is 8bit wide.The 8bits are explained below for the 4 instructions.The three arithmetic operations share the same instruction set, as they are common and is as below,

7	6	5	4	3	2	1	0

For addition, subtraction and OR :

1. The 0th and 1st bits are used to select the first operand's register in the register bank.

2. The 2nd and 3rd bits are used to select the second operand's register in the register bank.

3. The 4th and 5th bits are used to select the register where the result of the operation will be stored.

4. The 6th and 7th bits is used to denote what ALU operation to perform, 00=Addition, 01=Subtraction, 10=OR,11=Load Immediate(Discussed next).

For Load Immediate :

1. The last four bits(0,1,2,3) are the 4 bit values to be stored in the register.

*Note:** As the register size is 8 and the instruction size is also 8, we use two load immediate operations and to store the last four bits and first four bits in the register selected.The value in the IS is OR-ed with the data in the register after the first load immediate operation.

2. The 5th bit denotes what register to load the immediate value to, here we have used only R0/R1. Choose 0 for R0 and 1 for R1.

3. The 6th bit denotes whether the last 4bits should be loaded as last/first four bits into the register specified by the bit 5.

4. The last two bits are the op code for load immediate and is “11”.

Working:

1. Initially the instruction is loaded from the instruction memory.
2. The fetched instruction is fed into the instruction decoder and it slices the bits and gives it to ALU, when the opcode is not 11(Load Immediate) and the register bank to select the registers.
3. The selected register values are pushed out of the registers in the bank to the ALU.
4. The ALU as per the op code does the operation on the two operands and sends the result 8bit value again to the register bank.
5. In register bank there are 4 registers and hence 2bits are used to select the register from the bank and hence each operand having two bits for the register is sliced from the instruction decoder and sent to the bank to select.
6. The ALU also sends the 2bit selector bits to select the register for storing the resultant value.
7. Finally after one instruction has been performed, the next instruction is fetched from the instruction memory and the same process is repeated.

8. The MUX is used because , at the same time both the ALU and instruction decoder drives the data lines and write enable lines of the register bank at the same time, and hence a mux is used with a selection line from the instruction decoder(1 if opcode is “11” and 0 if opcode is any of the other three).
9. 8bit MUX is used for data lines selection in case of conflict and 1bit MUX is used for the write enable line.
10. As the codes execute in parallel the clock for the instruction fetch from the memory is divided by the amount of 4 , i.e the instruction is fetched only for 4 main clock pulses, within which the operation of one instruction will be performed.
11. In short 1Machin Cycle=4 T states.

Conclusion:

Thus the simple microprocessor is implemented using the VHDL on modelsim.
We are planning to take it further by having an compiler, assembler and our own programming language in the future versions.

Thankyou.

Appendix:

1. Register bank:

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

entity regf is
  port(
        i_clk,i_en,i_we,ird_f: in std_logic;
        ira_en,irb_en,ird_en: in std_logic_vector(1 downto 0);
        ora_d,orb_d: out std_logic_vector(7 downto 0);
        ird_d: in std_logic_vector(7 downto 0)
      );
end entity;

architecture behav of regf is
type store_t is array (0 to 3) of std_logic_vector(7 downto 0);
signal regs: store_t := (others => X"00");
begin
  process(i_clk)
    begin
      if(rising_edge(i_clk) and i_en='1') then
        if(ira_en/="UU") then
          ora_d<=regs(to_integer(unsigned(ira_en)));
        end if;
        if(irb_en/="UU") then
          orb_d<=regs(to_integer(unsigned(irb_en)));
        end if;
        if(i_we='1') then
          if(ird_f='1' and ird_en/="UU") then
            regs(to_integer(unsigned(ird_en)))<=ird_d;
          elsif(ird_f='0' and ird_en/="UU") then
```

```
            regs(to_integer(unsigned(ird_en)))<=regs(to_integer(unsigned(ird_en))) or ird_d;
         end if;
       end if;
      end if;
    end process;
end behav;
```

2. Instruction decoder:

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

entity insdec is
  port(
        i_clk : in  STD_LOGIC;
        i_inst : in  STD_LOGIC_VECTOR (7 downto 0);
        i_en : in  STD_LOGIC;
        ora_en : out  STD_LOGIC_VECTOR (1 downto 0);
        orb_en : out  STD_LOGIC_VECTOR (1 downto 0);
        ord_en : out  STD_LOGIC_VECTOR (1 downto 0);
        ord_d : out  STD_LOGIC_VECTOR (7 downto 0);
        ord_f : out STD_LOGIC;
        o_we : out  STD_LOGIC;
        o_mux : out bit;
        aluo : out  STD_LOGIC_VECTOR (1 downto 0)
      );
end entity;
architecture behav of insdec is
signal rd_d : std_logic_vector(7 downto 0):=X"00";
begin
  process(i_clk)
    begin
      if (rising_edge(i_clk) and i_en='1') then
        if ((i_inst(7 downto 6)) /= B"11") then
          ora_en<=i_inst(1 downto 0);
          orb_en<=i_inst(3 downto 2);
          ord_en<=i_inst(5 downto 4);
          aluo<=i_inst(7 downto 6);
          o_we<='0';
          ord_f<='0';
        else
          aluo<=i_inst(7 downto 6);
          o_we<='1';
          ord_f<='0';
          if(i_inst(4)='0') then
            ord_en<="00";
          else
            ord_en<="01";
          end if;
          if(i_inst(5)='0') then
            ord_d(3 downto 0)<=i_inst(3 downto 0);
            ord_d(7 downto 4)<=B"0000";
          else
            ord_d(7 downto 4)<=i_inst(3 downto 0);
            ord_d(3 downto 0)<=B"0000";
          end if;
        end if;
        if((i_inst(7 downto 6)) = B"11") then
          o_mux<='1';
        else
          o_mux<='0';
        end if;
      end if;
    end process;
  end behav;
```

3. ALU:

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

entity alu is
```

```
  port(i_clk,i_en : in std_logic;
       ira_d,irb_d : in std_logic_vector(7 downto 0);
       aluo : in std_logic_vector(1 downto 0);
       ord_d : out std_logic_vector(7 downto 0);
       ord_f,o_we : out std_logic
       );

end entity;

architecture behav of alu is

  begin

    process(i_clk)

      begin

        if(rising_edge(i_clk) and i_en='1') then

          case aluo is

            when "00" => ord_d <= std_logic_vector(unsigned(ira_d) +
unsigned(irb_d));ord_f <= '1';
            when "01" => ord_d <= std_logic_vector(unsigned(ira_d) -
unsigned(irb_d));ord_f <= '1';
            when "10" => ord_d <= ira_d or irb_d;ord_f <= '1';
            when others => report "Error";

          end case;

          o_we <= '1';

        end if;

    end process;

end architecture;
```

4. Instruction Memory:

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

entity memory is

  port(ip : in std_logic_vector(2 downto 0);
       op : out std_logic_vector(7 downto 0)
       );

end entity;

architecture behav of memory is

  type reg is array (0 to 7) of std_logic_vector(7 downto 0);
  signal regs : reg :=
("11001111","11101010","11010101","11111111","00100100","01110100","11001111","
11101111");
  begin

    process(ip)
      begin
        op <= regs(to_integer(unsigned(ip)));
    end process;

end behav;
```

5. Counter:

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

```
entity counter is

  port(clk : in std_logic;
       op  : inout std_logic_vector(2 downto 0)
       );

end entity;

architecture behav of counter is

  signal t : std_logic_vector(2 downto 0) := "001";
  signal outclk : std_logic:='0';
  signal count : integer:=0;

  begin
    process(clk)
      begin
        if(rising_edge(clk)) then
          if(count=2) then
            outclk<=not(outclk);
            count<=0;
          else
            count<=count+1;
          end if;
        end if;
      end process;

    process(outclk)
      begin

        if(rising_edge(outclk)) then

          op <= std_logic_vector(to_unsigned(to_integer(unsigned(op))+1,3));

        end if;

    end process;

end behav;
```

6. Program Counter:

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

entity pc is

  port(clk,en : in std_logic);

end entity;

architecture behav of pc is

  component counter is

  port(clk : in std_logic;
       op  : inout std_logic_vector(2 downto 0)
       );

  end component;

  component memory is

  port(ip : in std_logic_vector(2 downto 0);
       op : out std_logic_vector(7 downto 0)
       );

  end component;

  component integ is
  port(
        clk : in std_logic;
        en : in std_logic;
        inst : in std_logic_vector(7 downto 0)
      );
  end component;
  signal instruction : std_logic_vector(7 downto 0);
  signal t : std_logic_vector(2 downto 0);

  begin

    g1 : counter port map(clk,t);
    g2 : memory port map(t,instruction);
    g3 : integ port map(clk,en,instruction);

end behav;
```

7. Integrated Module:

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

entity integ is
  port(
        clk : in std_logic;
        en : in std_logic;
        inst : in std_logic_vector(7 downto 0)
      );
end entity;

architecture behav of integ is

component insdec is
  port(
        i_clk : in  STD_LOGIC;
        i_inst : in  STD_LOGIC_VECTOR (7 downto 0);
        i_en : in  STD_LOGIC;
        ora_en : out  STD_LOGIC_VECTOR (1 downto 0);
        orb_en : out  STD_LOGIC_VECTOR (1 downto 0);
        ord_en : out  STD_LOGIC_VECTOR (1 downto 0);
        ord_d : out  STD_LOGIC_VECTOR (7 downto 0);
        ord_f : out STD_LOGIC;
        o_we : out  STD_LOGIC;
        o_mux : out BIT;
        aluo : out  STD_LOGIC_VECTOR (1 downto 0)
      );
end component;

component regf is
  port(
        i_clk,i_en,i_we,ird_f: in std_logic;
        ira_en,irb_en,ird_en: in std_logic_vector(1 downto 0);
        ora_d,orb_d: out std_logic_vector(7 downto 0);
        ird_d: in std_logic_vector(7 downto 0)
      );
end component;

component alu is

  port(i_clk,i_en : in std_logic;
       ira_d,irb_d : in std_logic_vector(7 downto 0);
       aluo : in std_logic_vector(1 downto 0);
       ord_d : out std_logic_vector(7 downto 0);
       ord_f,o_we : out std_logic
       );

end component;

component mux is
port(
                                i0,i1: in std_logic;
                                s:  in bit;
                                op: out std_logic
                                );
end component;


component mux8 is
port(
                                i0,i1: in std_logic_vector(7 downto 0);
                                s: in bit;
                                op: out std_logic_vector(7 downto 0)
                                );
end component;

signal alus,sra_en,srb_en,srd_en : std_logic_vector(1 downto 0);
signal s_we_ins,srd_f_ins,s_we_alu,srd_f_alu,s_we,srd_f : std_logic;
signal s_mux : bit;
signal srd_d_ins,sra_d,srb_d,srd_d_alu,srd_d : std_logic_vector(7 downto 0);
begin
  ins: insdec port map(i_clk=>clk,
                     i_en=>en,
                     i_inst=>inst,
                     o_we=>s_we_ins,
                     ora_en=>sra_en,
                     orb_en=>srb_en,
                     ord_en=>srd_en,
                     ord_d=>srd_d_ins,
                     ord_f=>srd_f_ins,
                     aluo=>alus,
                     o_mux=>s_mux);
  reg: regf port map(
                    i_clk=>clk,
                    i_en=>en,
                    ira_en=>sra_en,
                    irb_en=>srb_en,
                    ird_en=>srd_en,
                    ora_d=>sra_d,
                    orb_d=>srb_d,
                    i_we=>s_we,
                    ird_f=>srd_f,
                    ird_d=>srd_d);
  aluu: alu port map(
                    ira_d=>sra_d,
                    irb_d=>srb_d,
                    i_clk=>clk,
                    i_en=>en,
                    ord_d=>srd_d_alu,
                    ord_f=>srd_f_alu,
                    o_we=>s_we_alu,
                    aluo=>alus);
  mux_rdd: mux8 port map(
                        i0=>srd_d_alu,
                        i1=>srd_d_ins,
                        s=>s_mux,
                        op=>srd_d);
  mux_rdf: mux port map(
                        i0=>srd_f_alu,
                        i1=>srd_f_ins,
                        s=>s_mux,
                        op=>srd_f);
  mux_we: mux port map(
                        i0=>s_we_alu,
                        i1=>s_we_ins,
                        s=>s_mux,
                        op=>s_we);
end behav;
```

8. <u>1bit MUX:</u>

```
                    library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux is
port(
                              i0,i1: in std_logic;
                              s: in bit;
                              op: out std_logic
                              );
end mux;

architecture Behavioral of mux is

begin
```

```
process(s,i0,i1)
begin
        if(s='0') then op<=i0;
        else op<=i1;
        end if;
end process;
end Behavioral;
```

9. 8bit MUX:

```
                library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux8 is
port(
                        i0,i1: in std_logic_vector(7 downto 0);
                        s: in bit;
                        op: out std_logic_vector(7 downto 0)
                        );
end mux8;

architecture Behavioral of mux8 is

begin
process(s,i0,i1)
begin
        if(s='0') then op<=i0;
        else op<=i1;
        end if;
end process;
end Behavioral;
```