Assigment-3

by:

Vraj Desai
(110118396)

The controller:

The design of this controller was divided into five sections.

1: The register file:
First, the R0-R7 register transfer was designed in “reg_file.vhd”.

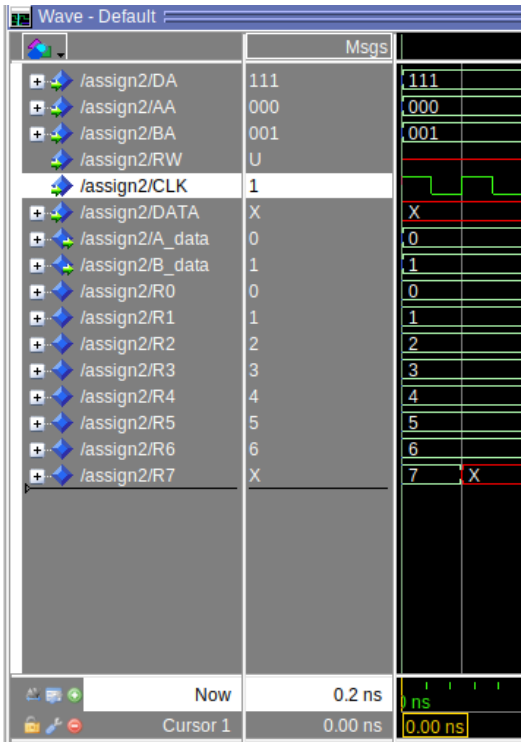

Figure 1: Register transer file.

As seen from Fig. 1, the 0, 1 data provided with address AA and BA is transferred to R0 and R1. And at the next clock, the edge X is transferred to R7 due to DA.

2: Function Unit:
This is implemented in “fun_file.vhd”.

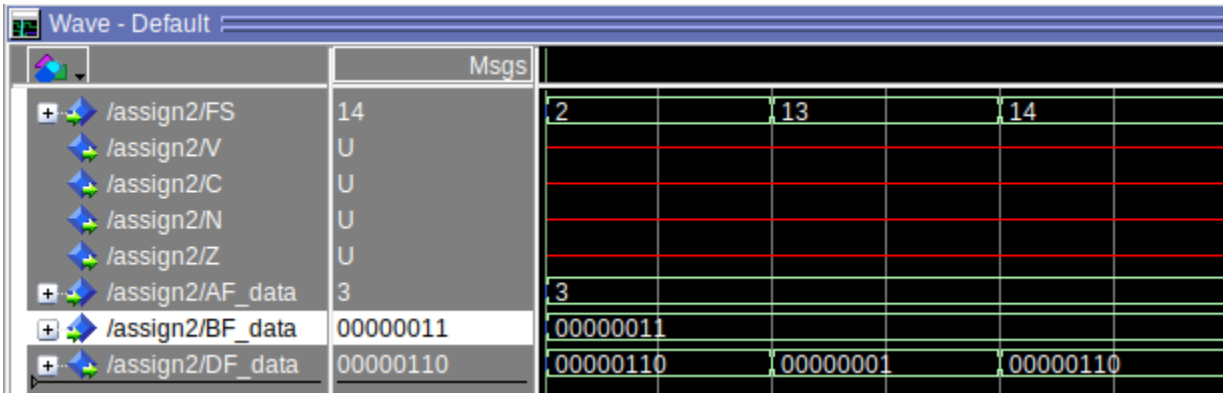

Figure 2: Function Unit

As seen from Fig. 1, the 0, 1 data provided with address AA and BA is transferred to R0 and R1. And at the next clock, the edge X is transferred to R7 due to DA.

3: Datapath:

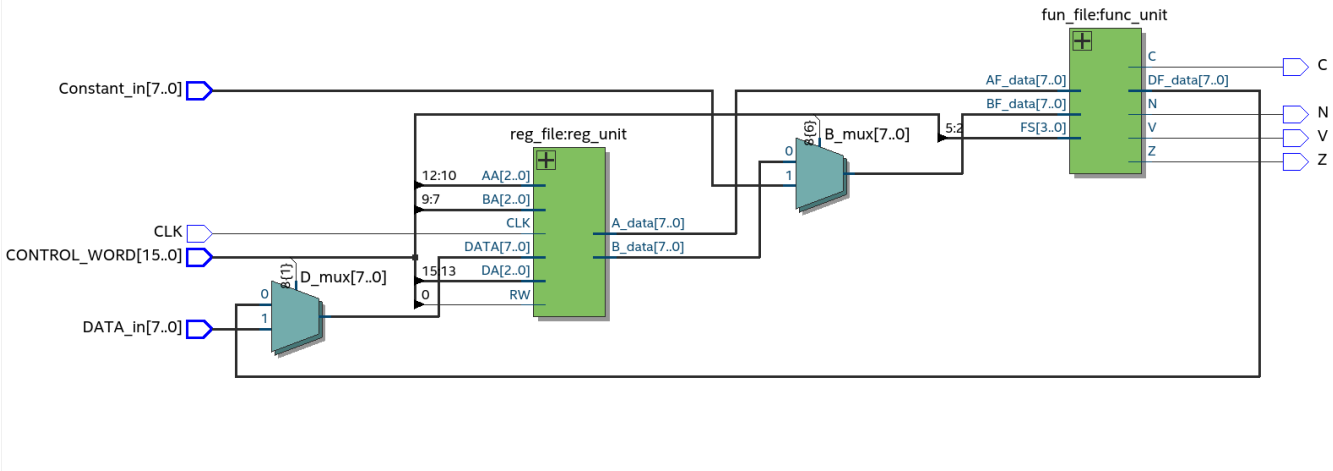

Figure 3: Datapath

The "datapath.vhd" combines "reg_file.vhd" and "fun_file.vhd" together with mux at input driven by the control word itself. The constent is muxed with reg_file output, while data_in is muxed with function unit's output.

The data path gets the control word from the memory and then performs operation from data_in or registers from reg_file. The graph of the datapath can be seen below in Fig. 4.

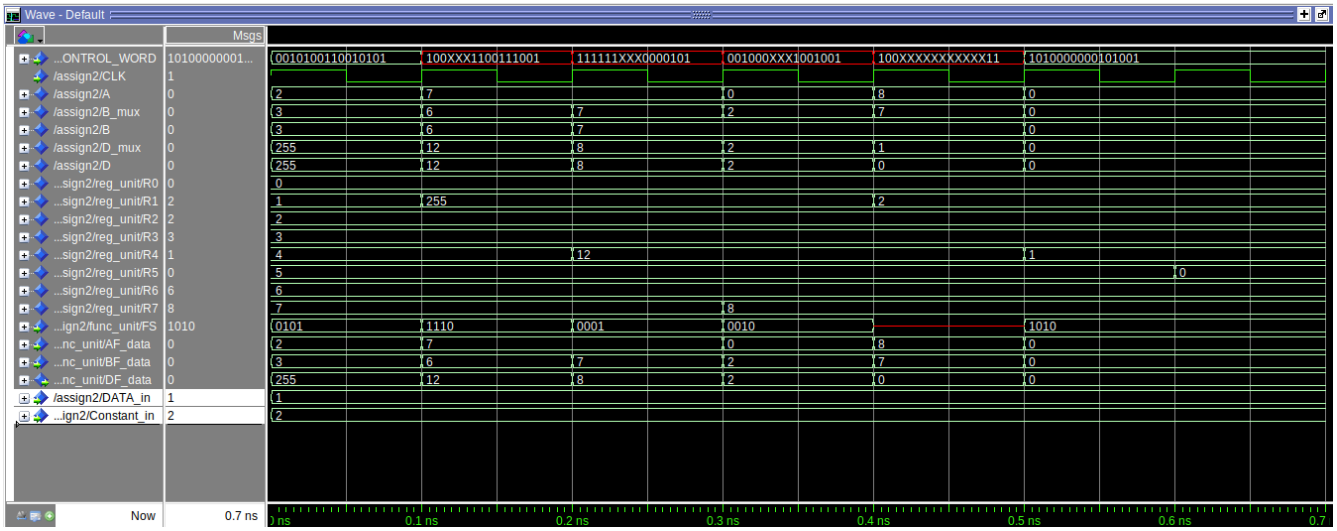

Figure 4: Datapath operation.

4: Control and Memory:
The controller generates an address when go is 1 and provides it to the memory. When the memory gets the address, and RD is high, it provides a control word to the datapath.

5: Assign2:
This file combines all the sub-blocks, as seen in Fig. 5.

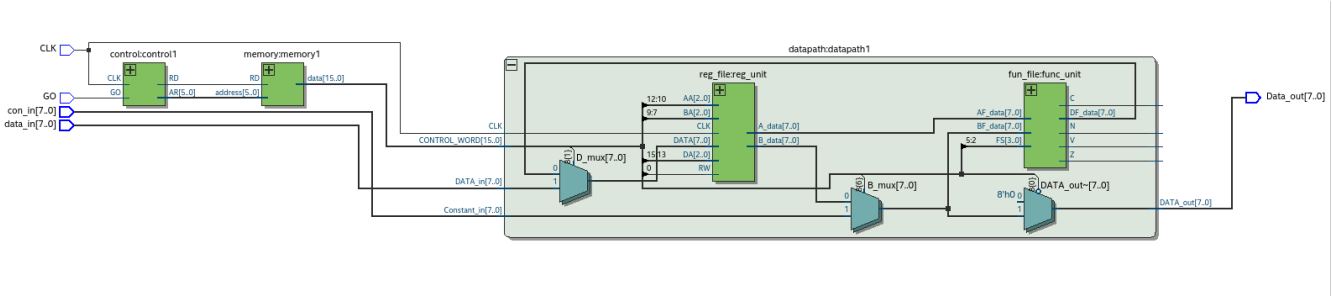

Figure 5: The controller.

As seen in Fig.5, the controller provides an address to memory, and the memory provides a control word to the datapath. And in the datapath, the sub-section reg-file and fun-file can be seen.

The final output of the controller can be seen in Fig.6 and it can be compaerd with fig.7 of the book.

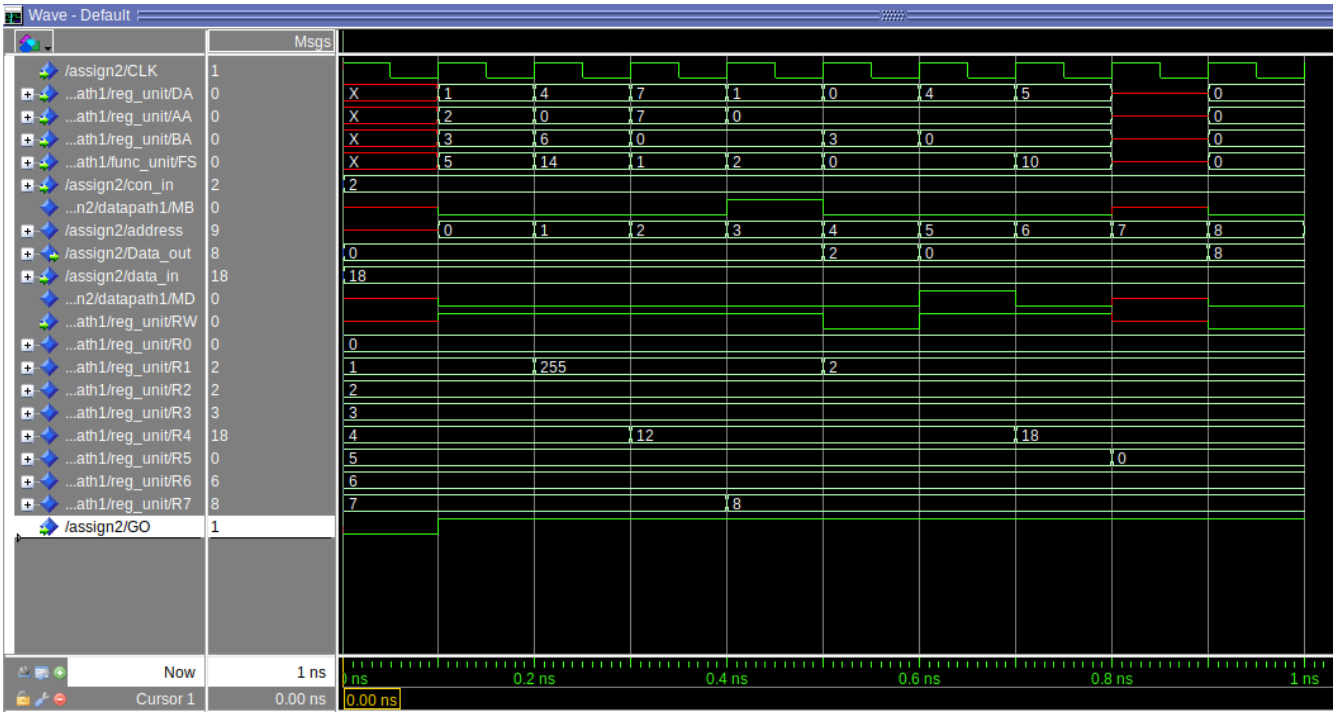

Figure 6: Controller output.

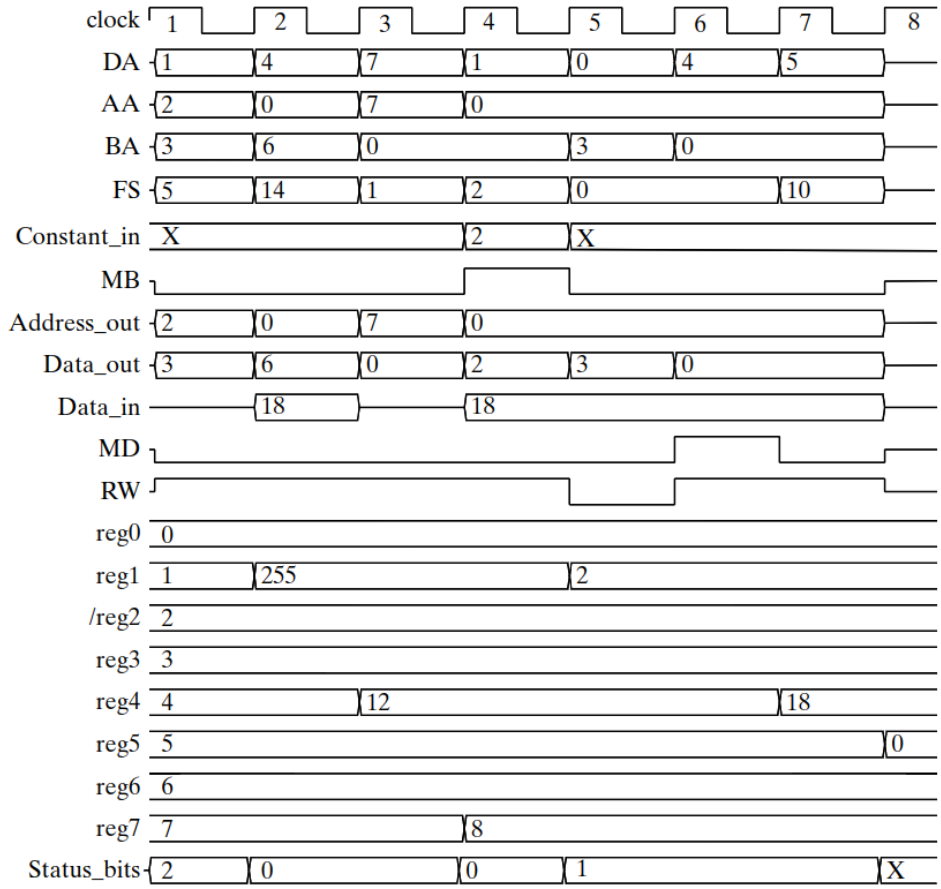

Figure 7: The book's controller.

Apendex:

Assign2.vhd:

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

entity assign2 is
port(
      CLK, GO: in std_logic;
      Data_out: out std_logic_vector(7 downto 0);
      con_in: in std_logic_vector(7 downto 0);
      data_in: in std_logic_vector(7 downto 0)
);
end entity;

architecture BEHAVIORAL of assign2 is
component memory
port (
      address : in std_logic_vector(5 downto 0);
      data : out std_logic_vector(15 downto 0);
      RD : in std_logic
);
end component;

component control
port(
      CLK, GO: in std_logic;
      AR : out std_logic_vector(5 downto 0);
      RD : out std_logic
);
end component;

component datapath is
port(
      CONTROL_WORD: in std_logic_vector(15 downto 0);
      V, C, N, Z: out  std_logic;
      CLK: in std_logic;
      Constant_in : in std_logic_vector(7 downto 0);
      DATA_in: in std_logic_vector(7 downto 0);
      DATA_out: out std_logic_vector(7 downto 0)
);
end component;

signal address: std_logic_vector(5 downto 0);
signal control_word: std_logic_vector( 15 downto 0);
```

```
signal RD_I: std_logic;
signal V,C,N,Z: std_logic;

begin

control1: control port map(CLK => CLK, GO => GO, AR => address, RD => RD_I);
memory1: memory port map(address => address, data => control_word, RD => RD_I);
datapath1: datapath port map(CONTROL_WORD => control_word, V => V, C => C, N => N,
Z => Z,
                                                        CLK => CLK, Constant_in =>
con_in, DATA_IN => data_in, DATA_out => Data_out);

end architecture BEHAVIORAL;
```

---

control.vhd:

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


--Control Word ranges
-- Range DA(15:13)
-- Range AA(12:10)
-- Range BA(9:7)
-- Range MB(6)
-- Range FS(5:2)
-- Range MD(1)
-- Range RW(0)

entity control is
port(
      CLK, GO: in std_logic;
      AR : out std_logic_vector(5 downto 0);
      RD : out std_logic
);
end entity;

architecture BEHAVIORAL of control is

signal counter, counterx: std_logic_vector(5 downto 0) := B"000000";
signal RD_i : std_logic;

begin
      process(CLK, GO)
      begin
            if (rising_edge(CLK)) then
                  if (GO = '1') then
```

```
                        counter <= counter + 1;
                  end if;
            end if;
      end process;
      process(GO,counter)
      begin
            if (GO ='1') then
                  RD_i <= '1';
                  counterx <= counter;
            else
                  RD_i <= '0';
                  counterx <= "XXXXXX";
            end if;
      end process;

RD <= RD_i;
AR <= counterx;
end architecture BEHAVIORAL;
```

---

memory.vdh:

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

entity memory is
  port ( address : in std_logic_vector(5 downto 0);
         data : out std_logic_vector(15 downto 0);
                  RD : in std_logic);
end entity;

architecture behavioral of memory is
  type mem is array ( 0 to 2**3 - 1) of std_logic_vector(15 downto 0);
  constant my_Rom : mem := (
    0  => "0010100110010101",
    1  => "1000001100111001",
    2  => "1111110000000101",
    3  => "0010000001001001",
    4  => "0000000110000000",
    5  => "1000000000000011",
       6  => "1010000000101001",
       7  => "XXXXXXXXXXXXXXXX");
begin
   process (address)
   begin
      if (RD = '1') then
     case address is
       when "000000" => data <= my_rom(0);
       when "000001" => data <= my_rom(1);
       when "000010" => data <= my_rom(2);
```

```
       when "000011" => data <= my_rom(3);
       when "000100" => data <= my_rom(4);
       when "000101" => data <= my_rom(5);
             when "000110" => data <= my_rom(6);
             when "000111" => data <= my_rom(7);
       when others => data <= "0000000000000000";
       end case;
      end if;
  end process;
end architecture behavioral;
```

---

datapath.vhd:

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

--Control Word ranges
-- Range DA(15:13)
-- Range AA(12:10)
-- Range BA(9:7)
-- Range MB(6)
-- Range FS(5:2)
-- Range MD(1)
-- Range RW(0)

entity datapath is
port(
      CONTROL_WORD: in std_logic_vector(15 downto 0);
      V, C, N, Z: out  std_logic;
      CLK: in std_logic;
      Constant_in : in std_logic_vector(7 downto 0);
      DATA_in: in std_logic_vector(7 downto 0);
      DATA_out: out std_logic_vector(7 downto 0)
);
end entity;

architecture BEHAVIORAL of datapath is

signal FS : std_logic_vector(3 downto 0);
signal DA : std_logic_vector(2 downto 0);
signal AA : std_logic_vector(2 downto 0);
signal BA : std_logic_vector(2 downto 0);
signal MB, MD, RW : std_logic;

signal A, B_mux, B, D_mux, D: std_logic_vector(7 downto 0);

component reg_file
      port(
            DA, AA, BA: in std_logic_vector(2 downto 0);
```

```
            RW: in std_logic;
            CLK: in std_logic;
            DATA: in std_logic_vector(7 downto 0);
            A_data, B_data: out std_logic_vector(7 downto 0)
      );
end component;

component fun_file
port(
      FS: in std_logic_vector(3 downto 0); -- Function Select
      V, C, N, Z: out  std_logic;                     -- Signal Out
      AF_data, BF_data: in std_logic_vector(7 downto 0); -- Function unit data
input
      DF_data: out std_logic_Vector(7 downto 0) -- Function unit data output
);
end component;


begin

      DA <= CONTROL_WORD(15 downto 13);
      AA <= CONTROL_WORD(12 downto 10);
      BA <= CONTROL_WORD(9 downto 7);
      MB <= CONTROL_WORD(6);
      FS <= CONTROL_WORD(5 downto 2);
      MD <= CONTROL_WORD(1);
      RW <= CONTROL_WORD(0);

      process(MB,B)
      begin
            case MB is
                  when '1' => B_mux <= Constant_in;
                  when others => B_mux <= B;
            end case;
      end process;

func_unit: fun_file port map (FS => FS, V => V, C => C, N => N, Z => Z, AF_data =>
A, BF_data => B_mux, DF_data => D);

reg_unit: reg_file port map (DA => DA, AA => AA, BA => BA, RW => RW, CLK => CLK,
DATA => D_mux, A_data => A, B_data => B);

process(MD,D)
      begin
            case MD is
                  when '1' => D_mux <= DATA_in;
                  when others => D_mux <= D;
            end case;
      end process;

process(RW)
      begin
```

```
            if (RW = '0') then
                  DATA_OUT <= B_MUX;
            else
                  DATA_OUT <= "00000000";
            end if;
      end process;
end architecture BEHAVIORAL;
```

---

reg_file.vhd:

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

entity reg_file is
port(
      DA, AA, BA: in std_logic_vector(2 downto 0);
      RW: in std_logic;
      CLK: in std_logic;
      DATA: in std_logic_vector(7 downto 0);
      A_data, B_data: out std_logic_vector(7 downto 0)
);
end entity;

architecture BEHAVIORAL of reg_file is
signal R0 : std_logic_vector(7 downto 0) := B"00000000";
signal R1 : std_logic_vector(7 downto 0) := B"00000001";
signal R2 : std_logic_vector(7 downto 0) := B"00000010";
signal R3 : std_logic_vector(7 downto 0) := B"00000011";
signal R4 : std_logic_vector(7 downto 0) := B"00000100";
signal R5 : std_logic_vector(7 downto 0) := B"00000101";
signal R6 : std_logic_vector(7 downto 0) := B"00000110";
signal R7 : std_logic_vector(7 downto 0) := B"00000111";

begin


      process (AA)
      begin
                  case AA is
                        when "000" => A_data <= R0;
                        when "001" => A_data <= R1;
                        when "010" => A_data <= R2;
                        when "011" => A_data <= R3;
                        when "100" => A_data <= R4;
                        when "101" => A_data <= R5;
                        when "110" => A_data <= R6;
                        when others => A_data <= R7;
                  end case;
      end process;
```

```
	process (BA)
	begin
			case BA is
				when "000" => B_data <= R0;
				when "001" => B_data <= R1;
				when "010" => B_data <= R2;
				when "011" => B_data <= R3;
				when "100" => B_data <= R4;
				when "101" => B_data <= R5;
				when "110" => B_data <= R6;
				when others => B_data <= R7;
			end case;
	end process;


	process (DA, CLK)
	begin
		if (rising_edge(CLK) and RW = '1') then
			case DA is
				when "000" => R0 <= DATA;
				when "001" => R1 <= DATA;
				when "010" => R2 <= DATA;
				when "011" => R3 <= DATA;
				when "100" => R4 <= DATA;
				when "101" => R5 <= DATA;
				when "110" => R6 <= DATA;
				when others => R7 <= DATA;
			end case;
		end if;
	end process;
end architecture BEHAVIORAL;
```

---

fun_file.vhd:

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


-- FS is the function select of the arthmatic opeation.
-- V, C, N, Z are the signal output for the ALU to control unit.
entity fun_file is
port(
	FS: in std_logic_vector(3 downto 0); -- Function Select
	V, C, N, Z: out  std_logic;                      -- Signal Out
	AF_data, BF_data: in std_logic_vector(7 downto 0); -- Function unit data
input
```

```
      DF_data: out std_logic_Vector(7 downto 0) -- Function unit data output
);
end entity;

architecture BEHAVIORAL of fun_file is
begin

      process (FS, BF_data, AF_data)
      begin
                  case FS is
                        when "0000" => DF_data <= AF_data;
                        when "0001" => DF_data <= AF_data + 1;
                        when "0010" => DF_data <= AF_data + BF_data;
                        when "0011" => DF_data <= AF_data + BF_data + 1;
                        when "0100" => DF_data <= AF_data + not(BF_data);
                        when "0101" => DF_data <= AF_data + not(BF_data) + 1;
                        when "0110" => DF_data <= AF_data - 1;
                        when "0111" => DF_data <= AF_data;
                        when "1000" => DF_data <= AF_data and BF_data;
                        when "1001" => DF_data <= AF_data or BF_data;
                        when "1010" => DF_data <= AF_data xor BF_data;
                        when "1011" => DF_data <= not(AF_data);
                        when "1100" => DF_data <= BF_data;
                        when "1101" => DF_data <= '0' & BF_data(7 downto 1);
                        when "1110" => DF_data <= BF_data(6 downto 0) & '0';
                        when others => DF_data <= B"00000000";
                  end case;
      end process;

end architecture BEHAVIORAL;
```