Department of Electrical Engineering and Computer Science

Howard University

Washington, DC 20059

EECE 406 Advanced Digital System Design

Fall 2017

Final Project: LC-3

By

Pawan Kumar Gaire

@02774258

Instructor: Michaela Amoo

Date: 16th November, 2017

Introduction

LC-3 (Little computer- 3) is an example of von Neumann machine consisting of memory, input/outupt, processing unit and control unit. It is a Reduced Instruction Set Computer (RSIC). The data and addresses are 16 bits each and has few general purpose registers. It also has some special propose resistors as program counter, instruction register.

Schematic

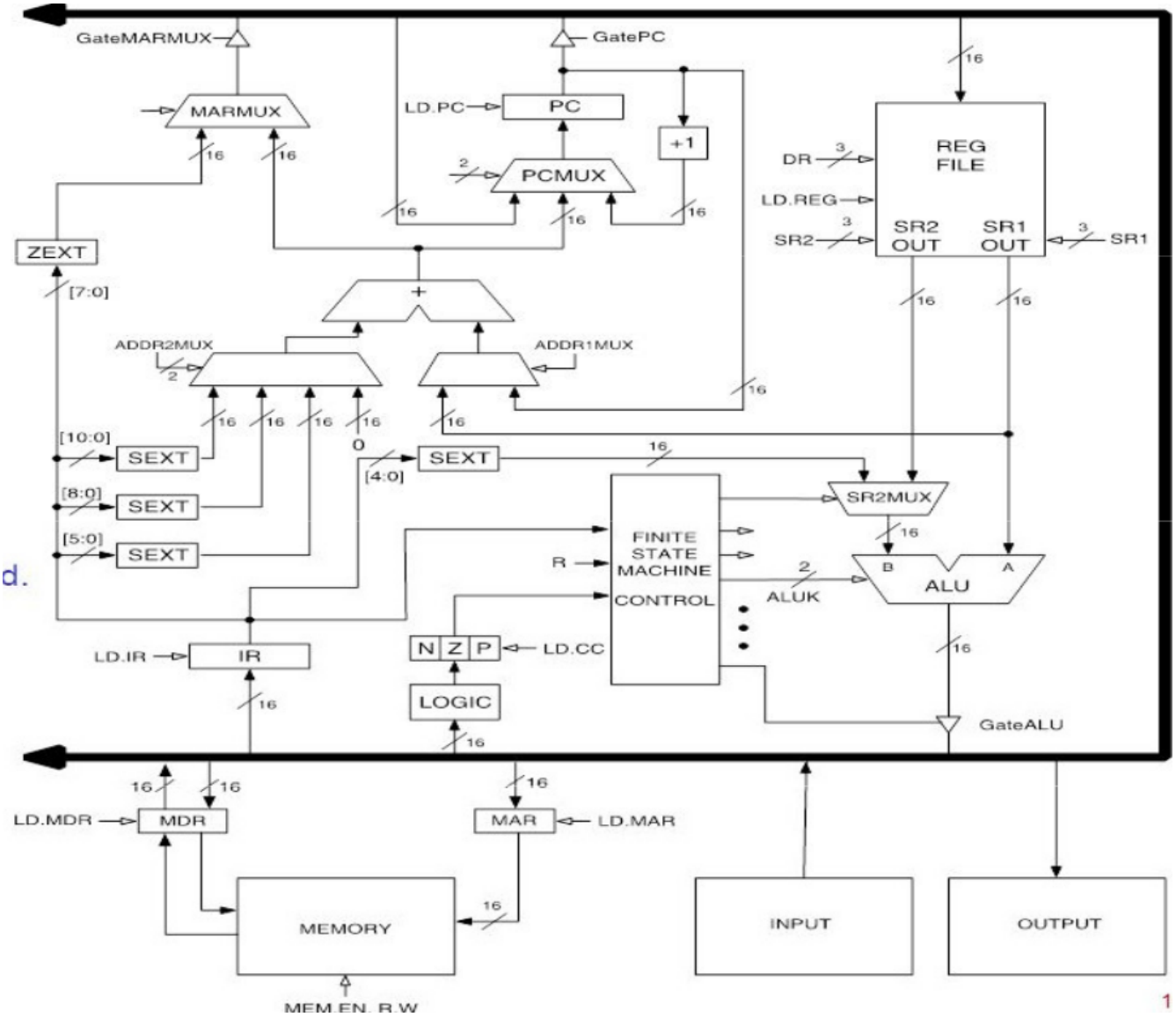

Illustration 1: datapath of LC-3

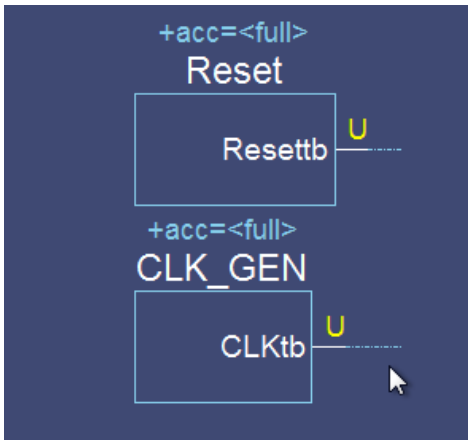

Illustration 2: Schematic as shown by dataflow

Analysis

We started with datapath. We put all the entities into a component file:

Instruction register
Sign extenders
Multiplexers
Tri-state gates
Register array
ALU
Program counter
NZP logic
Register
RAM (random access memory)
FSM (finite state machine)

The memory of LC3 can store 16 bits of data. It can do basic operations like read (load value from memory location and transfer to the processor) and write (store the value transferred from processor to memory location). To load data from any location, it writes the address into MAR, sends signal to memory and read data from MDR. Similarly, to store value to location, it writes data to MDR, write address into MDR and send write signal to memory.

The LC3 evaluates arithmetic and logical functions to determine values that are to be assigned to variable, determines the execution order of statements in any program. The LC3 consists of ALU, arithmetic logical unit which can have many functional units, some of which are for special purpose like floating point addition, square root. In the LC3 that I designed, ALU has 4 functions which are addition, not, and.

The number of bits normally processed by ALU in 1 instruction (word size) is 16 bits.

In our design, Instruction register contains the current instruction and program counter gives address of next instruction that is to be executed.

The instruction of LC-3 specifies 2 things- opcode and operands. The 4 most significant bits is an opcode that determines the type of operation to be done. And the meaning of other bits changes according to the type of instruction and operation. The LC3 that we designed can handle following instructions: load, add, store, and.

To write the structural code, we started listing all components in component file. Then we did the top level entity deceleration. Furthermore, in the architecture we had component declaration followed by internal signals and component instantiation.

Troubleshooting

I got different errors while run to debug the program. After completing writing the structural design code for lc-3, I compiled the code but transcript showed several errors. Some of the error were to due usage of signals that were not defined. Removing those signals solved the error. Secondly, generic map and port map in instantiation of components also showed some error. When reading through the error list, the problem was caused because some signals defined as standard logic were taking standard logic vector values and vice versa. It was because the order of parameters on port map did not map exactly to entity description. I changed the order to match the entity description. Also, to solve the error due to generic map, I adjusted the values in generic map to exactly match the values of generics in entity description. These steps solved the errors that I was having in my code.

Conclusion

In this lab, we designed an LC-3- a von Neumann machine consisting of memory, input/output, processing and control units. To verify that the design is working properly, the design took the input data from memory and provided the output after performing the given operation.

Appendix

Component file

```
--------------1 bit register-------------------


library ieee;
```

```
use ieee.std_logic_1164.all;

Entity gaire_1_bit_register is
    port (   clk : in std_logic;
             Reset : IN STD_LOGIC;
             En : IN STD_LOGIC;
             OP_A : IN STD_LOGIC;
             OP_Q : OUT STD_LOGIC
        );
End gaire_1_bit_register;

ARCHITECTURE arch_1_bit_register of gaire_1_bit_register is
    signal my_sig : STD_LOGIC;

        begin
            process(clk)
            begin
            if(clk'evEnt and clk = '1') then
                if(Reset='1') then
                    my_sig <= '0';
                elsif (En='1') then
                    my_sig <= OP_A;
                else
                    my_sig <= my_sig;
                End if; -- Reset
            End if; -- clk
     End process;
```

```
     OP_Q <= my_sig;
End arch_1_bit_register;


--------- n bit Register -----------

library ieee;
use ieee.std_logic_1164.all;

Entity gaire_n_bit_register is
    gEneric(
            w: Integer := 16
    );
    port (  CLK : in std_logic;
            Reset : IN STD_LOGIC;
            En : IN STD_LOGIC;
            OP_A : IN STD_LOGIC_VECTOR(w-1 DOWNTO 0);
            OP_Q : OUT STD_LOGIC_vector(w-1 DOWNTO 0)
        );
End gaire_n_bit_register;

ARCHITECTURE arch_n_bit_register of gaire_n_bit_register is
    signal my_sig : STD_LOGIC_vector(w-1 DOWNTO 0);

        begin
            process(CLK)
            begin
```

```
            if(CLK'evEnt and CLK = '1') then
                if(Reset='1') then
                    my_sig <= (others => '0');
                elsif (En='1') then
                    my_sig <= OP_A;
                else
                    my_sig <= my_sig;
                End if; -- Reset
            End if; -- CLK
        op_q <= my_sig;
    End process;
End arch_n_bit_register;

------------------ register array ----------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

Entity gaire_register_array is
gEneric(w: integer:= 16;
        p: integer := 3;
        E: natural := 8
);
port(CLK: in std_logic;
    Reset: in std_logic;
```

```
    LD_REG: in std_logic; -- Enable to tell us which register in the
array is active for input
    DR: in std_logic_vector(p-1 downto 0);
    OP_A: in std_logic_vector(w-1 downto 0);
    SR1: in std_logic_vector(p-1 downto 0);
    SR2: in std_logic_vector(p-1 downto 0);
    OP_Q1: out std_logic_vector(w-1 downto 0);
    OP_Q2: out std_logic_vector(w-1 downto 0)
);
End gaire_register_array;

architecture arch_genreg of gaire_register_array is
type regaray is array(E-1 downto 0) of std_logic_vector(w-1 downto
0);
signal sEn: std_logic_vector(E-1 downto 0);
signal sFF: regaray;

component gaire_n_bit_register
generic(w: integer:= 16);
port(CLK: in std_logic;
    Reset: in std_logic;
    En: in std_logic;
    OP_A: in std_logic_vector(w-1 downto 0);
    OP_Q: out std_logic_vector(w-1 downto 0)
);
End component;
```

```
begin
    p0: process(DR, LD_REG)
    begin
        sEn <= (sEn'range => '0');
        sEn(to_integer(unsigned(DR))) <= LD_REG;
    End process;
    g0: for j in 0 to (E-1) gEnerate
        regh0: gaire_n_bit_register port map (CLK, Reset, sEn(j),
OP_A, sFF(j));
    End gEnerate;
OP_Q1 <= sFF(to_integer(unsigned(SR1)));
OP_Q2 <= sFF(to_integer(unsigned(SR2)));

End arch_genreg;



----- 5 to 16 sign extender --------

library ieee;
use work.CLOCKS.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;

-- this is for 5 to 16 bit sign extender
Entity gaire_5_to_16_bit_sign_extender is
generic (p: integer:= 16;
         w: integer:= 5);
```

```
port (CLK: in std_logic;
      Reset: in std_logic;
      OP_A: in std_logic_vector(w-1 downto 0);
      OP_Q: out std_logic_vector(p-1 downto 0)
      );
End gaire_5_to_16_bit_sign_extender;

architecture bhv of gaire_5_to_16_bit_sign_extender is

     signal my_sig: std_logic_vector(p-1 downto 0);
     begin
     process(CLK)
         begin
            if(CLK'evEnt and CLK ='1') then
               if(Reset = '1') then
                    my_sig <= (others => '0');
               elsif(OP_A(w-1) = '0') then
                     my_sig(p-1 downto w) <= (others => '0');
                    my_sig(w-1 downto 0) <= op_A;
               else
                     my_sig(p-1 downto w) <= (others => '1');
                    my_sig(w-1 downto 0) <= OP_A;
               End if;
             End if;
       End process;
     OP_Q <= my_sig;
End bhv;
```

```
----- 6 to 16 sign extender --------
library ieee;
use work.CLOCKS.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;

-- this is for 6 to 16 bit sign extender
Entity gaire_6_to_16_bit_sign_extender is
gEneric (p: integer:= 16;
         w: integer:= 6);
port (CLK: in std_logic;
      Reset: in std_logic;
      OP_A: in std_logic_vector(w-1 downto 0);
      OP_Q: out std_logic_vector(p-1 downto 0)
      );
End gaire_6_to_16_bit_sign_extender;

architecture bhv of gaire_6_to_16_bit_sign_extender is

     signal my_sig: std_logic_vector(p-1 downto 0);
     begin
     process(CLK)
         begin
            if(CLK'evEnt and CLK ='1') then
               if(Reset = '1') then
```

```
                    my_sig <= (others => '0');
                elsif(OP_A(w-1) = '0') then
                      my_sig(p-1 downto w) <= (others => '0');
                    my_sig(w-1 downto 0) <= op_A;
                else
                      my_sig(p-1 downto w) <= (others => '1');
                    my_sig(w-1 downto 0) <= OP_A;
                End if;
             End if;
       End process;
     OP_Q <= my_sig;
End bhv;



----- 8 to 16 zero extender --------
library ieee;
use work.CLOCKS.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;

-- this is for 8 to 16 bit zero extender
Entity gaire_8_to_16_bit_zero_extender is
gEneric (p: integer:= 16;
         w: integer:= 8);
port (CLK: in std_logic;
      Reset: in std_logic;
      OP_A: in std_logic_vector(w-1 downto 0);
```

```
      OP_Q: out std_logic_vector(p-1 downto 0)
      );
End gaire_8_to_16_bit_zero_extender;

architecture bhv of gaire_8_to_16_bit_zero_extender is

     signal my_sig: std_logic_vector(p-1 downto 0);
     begin
     process(CLK)
         begin
            if(CLK'evEnt and CLK ='1') then
               if(Reset = '1') then
                    my_sig <= (others => '0');
               else
                     my_sig(p-1 downto w) <= (others => '0');
                    my_sig(w-1 downto 0) <= op_A;
               End if;
           End if;
       End process;
     OP_Q <= my_sig;
End bhv;


----- 9 to 16 sign extender --------
library ieee;
use work.CLOCKS.all;
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_textio.all;

-- this is for 9 to 16 bit sign extender
Entity gaire_9_to_16_bit_sign_extender is
generic (p: integer:= 16;
         w: integer:= 9);
port (CLK: in std_logic;
      Reset: in std_logic;
      OP_A: in std_logic_vector(w-1 downto 0);
      OP_Q: out std_logic_vector(p-1 downto 0)
      );
End gaire_9_to_16_bit_sign_extender;

architecture bhv of gaire_9_to_16_bit_sign_extender is

     signal my_sig: std_logic_vector(p-1 downto 0);
     begin
     process(CLK)
         begin
            if(CLK'evEnt and CLK ='1') then
               if(Reset = '1') then
                    my_sig <= (others => '0');
               elsif(OP_A(w-1) = '0') then
                     my_sig(p-1 downto w) <= (others => '0');
                    my_sig(w-1 downto 0) <= op_A;
               else
                     my_sig(p-1 downto w) <= (others => '1');
```

```
                    my_sig(w-1 downto 0) <= OP_A;
                End if;
              End if;
       End process;
     OP_Q <= my_sig;
End bhv;



----- 11 to 16 sign extender -------
library ieee;
use work.CLOCKS.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;

-- this is for 11 to 16 bit sign extender
Entity gaire_11_to_16_bit_sign_extender is
gEneric (p: integer:= 16;
         w: integer:= 11);
port (CLK: in std_logic;
      Reset: in std_logic;
      OP_A: in std_logic_vector(w-1 downto 0);
      OP_Q: out std_logic_vector(p-1 downto 0)
      );
End gaire_11_to_16_bit_sign_extender;

architecture bhv of gaire_11_to_16_bit_sign_extender is
```

```
     signal my_sig: std_logic_vector(p-1 downto 0);
     begin
     process(CLK)
         begin
            if(CLK'evEnt and CLK ='1') then
               if(Reset = '1') then
                    my_sig <= (others => '0');
               elsif(OP_A(w-1) = '0') then
                     my_sig(p-1 downto w) <= (others => '0');
                    my_sig(w-1 downto 0) <= op_A;
               else
                     my_sig(p-1 downto w) <= (others => '1');
                    my_sig(w-1 downto 0) <= OP_A;
               End if;
             End if;
       End process;
     OP_Q <= my_sig;
End bhv;



---------- 2 to 1 MUX ------------
library ieee;
use ieee.std_logic_1164.all;
Entity gaire_2_to_1_multiplexer is
     port(
          a_0: in std_logic_vector(15 downto 0);
          a_1: in std_logic_vector(15 downto 0);
```

```
          y: out std_logic_vector(15 downto 0);
          sel: in std_logic);
End gaire_2_to_1_multiplexer;

architecture arch_2_to_1_multiplexer of gaire_2_to_1_multiplexer is
begin
     with sel select
          y <= a_0 when '0',
               a_1 when others;
End arch_2_to_1_multiplexer;



---------- 3 to 1 MUX ------------
library ieee;
use ieee.std_logic_1164.all;
Entity gaire_3_to_1_multiplexer is
     port(
          a_0: in std_logic_vector(15 downto 0);
          a_1: in std_logic_vector(15 downto 0);
          a_2: in std_logic_vector(15 downto 0);
          y: out std_logic_vector(15 downto 0);
          sel: in std_logic_vector(1 downto 0));
End gaire_3_to_1_multiplexer;

architecture arch_3_to_1_multiplexer of gaire_3_to_1_multiplexer is
begin
```

```
     with sel select
          y <= a_0 when "00",
               a_1 when "01",
               a_2 when others;
End arch_3_to_1_multiplexer;



---------- 4 to 1 MUX ------------
library ieee;
use ieee.std_logic_1164.all;
Entity gaire_4_to_1_multiplexer is
     port(
          a_0: in std_logic_vector(15 downto 0);
          a_1: in std_logic_vector(15 downto 0);
          a_2: in std_logic_vector(15 downto 0);
          a_3: in std_logic_vector(15 downto 0);
          y: out std_logic_vector(15 downto 0);
          sel: in std_logic_vector(1 downto 0));
End gaire_4_to_1_multiplexer;

architecture arch_4_to_1_multiplexer of gaire_4_to_1_multiplexer is
begin
     with sel select
          y <= a_0 when "00",
               a_1 when "01",
               a_2 when "10",
```

```
               a_3 when others;
End arch_4_to_1_multiplexer;



------------------ ADDER -----------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

entity gaire_adder IS              -- In/out Ports
port(     OP_A: in std_logic_vector(15 DOWNTO 0);
     OP_B: in std_logic_vector(15 DOWNTO 0);
     OP_Q: out std_logic_vector(15 DOWNTO 0)
);
end gaire_adder;

architecture arch_adder of gaire_adder is
begin
     OP_Q <= (OP_A + OP_B);
end arch_adder;



--------- TRi-state gate -----------
library ieee;
use ieee.std_logic_1164.all;

Entity gaire_tri_state_gate is
```

```
gEneric(w:integer:= 16);
port( En: in std_logic;
     OP_A: in std_logic_vector(w-1 downto 0);
     OP_Q: out std_logic_vector(w-1 downto 0)
);
End gaire_tri_state_gate;

architecture arch_tri_state_gate of gaire_tri_state_gate is
begin
     process(En, OP_A)
     begin
          if(En = '1') then
               OP_Q <= OP_A;
          elsif(En = '0') then
               OP_Q <= (others => 'Z');
          End if;
     End process;
End arch_tri_state_gate;



--------- ALU ----------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

Entity gaire_alu is
```

```
     gEneric(w: integer:= 16);
     port(
     OP_A: in std_logic_vector(w-1 downto 0);
     OP_B: in std_logic_vector(w-1 downto 0);
     sel: in std_logic_vector(1 downto 0);
     OP_Q: out std_logic_vector(w-1 downto 0)
);
End gaire_alu;

architecture arch_alu of gaire_alu is
signal my_sig: std_logic_vector(w-1 downto 0); -- internal signal
begin
     with sel select
          my_sig <= (OP_A + OP_B) when "00",
                    (OP_A - OP_B) when "01",
                    NOT(OP_A) when "10",
                    (OP_A AND OP_B) when others;
     OP_Q <= my_sig(w-1 downto 0);
End arch_alu;


--------- Program counter ----------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

Entity gaire_program_counter is
```

```
     gEneric(w: integer := 16);
     port(CLK: in std_logic;
     Reset: in std_logic;
     En: in std_logic;
     OP_A: in std_logic_vector(w-1 downto 0);
     OP_Q: out std_logic_vector(w-1 downto 0);
     OP_Z: out std_logic_vector(w-1 downto 0)
);
End gaire_program_counter;

architecture arch_program_counter of gaire_program_counter is
signal your_sig: std_logic_vector(w-1 downto 0):=(others => '0');
signal my_sig: std_logic_vector(w-1 downto 0);
begin
     process(CLK)
     begin
          if(CLK'evEnt and CLK = '1') then
               if (Reset = '1') then
                    my_sig <= (others => '0');
                    your_sig <= your_sig + 1;
               elsif(En = '1') then
                    my_sig <= OP_A;
                    your_sig <= OP_A + 1;
               else
                    my_sig <= my_sig;
                    your_sig <= your_sig;
               End if;
```

```
          End if;
     End process;
OP_Q <= my_sig;
OP_Z <= your_sig;
End arch_program_counter;


--------- NZP ----------------------
library ieee;
use work.CLOCKS.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;

Entity NZP is
port(
     CLK: in std_logic;
     Reset: in std_logic;
     En: in std_logic;
     OP_A: in std_logic_vector(15 downto 0);
     OP_Q: out std_logic_vector(2 downto 0)
     );
End NZP;

architecture behavioral of NZP is
signal sOP_Q: std_logic_vector(2 downto 0);
begin
process(CLK)
```

```
begin
   if (CLK'evEnt and CLK ='1') then
      if (Reset = '1') then
         sOP_Q <= "010";
      elsif (En = '1') then
         if (OP_A(15) = '1') then
            sOP_Q <= "100";
         elsif (OP_A = "0000000000000000")then
            sOP_Q <= "010";
         else
            sOP_Q <= "001";
      End if;
   End if;
End if;
End process;
OP_Q <= sOP_Q;
End behavioral;



--------- RAM ----------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

Entity gaire_RAM is
gEneric(ADD_WIDTH: integer:= 9;  -- address bit size 2**9=512 words
        WIDTH: integer:= 16);  -- data size
```

```
port(
    CLK: in std_logic;  --clock
    mem_wr_rd_add: in std_logic_vector(ADD_WIDTH-1 downto 0);
--write address
    data_in: in std_logic_vector(WIDTH-1 downto 0);  --input data
    data_out: out std_logic_vector(WIDTH-1 downto 0);  --output data
    read_write_En: in std_logic;  --read(0)/write(1) Enable
    mem_En: in std_logic);
End gaire_RAM;

architecture RAM_1kb of gaire_RAM is
    type data_array is array (integer range <>) of
std_logic_vector(WIDTH-1 downto 0);
    signal sdata_out : std_logic_vector(WIDTH-1 downto 0);
    signal data: data_array(0 to (2**ADD_WIDTH)) := (
    "0010000000010011", "0010001000010011", "0001010000000001",
    "0001011000000010", "0001100011111111", "0000000000000000",
    "0000000000000000", "0000000000000000", "0000000000000000",
    "0000000000000000", "0000000000000000", "0000000000000000",
    "0000000000000000", "0000000000000000", "0000000000000000",
    "0000000000000000", "0000000000000000", "0000000000000000",
    "0000000000000000", "0000000000000000", "0000000000000000",
    "0000000000000000", "0000000000000000", others =>
"0000000000000000"); --array of 512 word of lEngth 16 bits
begin

MEMORY: process(CLK, read_write_En)
```

```
begin

    if mem_En = '0' then
        sdata_out <= (others => 'Z');

    elsif mem_En = '1' and (CLK'evEnt and CLK = '1') then --if clock
is 1 on the rising edge, and read
        if read_write_En = '1' then                        --is
Enabled(0), then read data from mem array
            data(conv_integer(mem_wr_rd_add)) <= data_in;  --write
mem array by decimal in mem_write_add
            sdata_out <= (others => 'Z');
        elsif read_write_En = '0' then
            sdata_out <= data(conv_integer(mem_wr_rd_add));
--addressed by decimal in mem_read_add
        End if;
    End if;
End process MEMORY;
data_out <= sdata_out;
End RAM_1kb;



------------ MAR -------------------
library IEEE;
use work.all;
use IEEE.STD_LOGIC_1164.all;

Entity gaire_MAR_REG is
```

```
port (
CLK: std_logic;
En: in std_logic;
Reset: in std_logic;
BUS_IN: in std_logic_vector(15 downto 0);
MAR_OUT: out std_logic_vector(8 downto 0)
);
End gaire_MAR_REG;

architecture BEH of gaire_MAR_REG is

signal MAR_DATA_OUT: std_logic_vector(8 downto 0);

begin

gaire_MAR_REG:process (CLK,En,Reset,BUS_IN)

begin

     if (Reset = '1') then -- act like a tri-state

         MAR_DATA_OUT <= (others=>'Z');

     elsif (CLK'evEnt and CLK = '1') then

            if (En='1') then
```

```
                        MAR_DATA_OUT <= BUS_IN(8 downto 0); --grab
the 9-bit address
            else

            MAR_DATA_OUT <=(others=>'Z'); -- act like a tri-state

            End if;
      End if;

End process;
MAR_OUT <= MAR_DATA_OUT;
End BEH;



------------ MDR -------------------
library IEEE;
use work.all;
use IEEE.STD_LOGIC_1164.all;

Entity gaire_MDR_REG is
port (
          CLK: in std_logic;
          En: in std_logic;
          Reset: in std_logic;
          BUS_IN: in std_logic_vector(15 downto 0); --FROM BUS
(WRITE)
          MEM_IN: in std_logic_vector(15 downto 0); --FROM MEM (READ)
```

```
      MDR_OUT: out std_logic_vector(15 downto 0)-- To Bus
           );
End gaire_MDR_REG;

architecture BEH of gaire_MDR_REG is
signal MDR_DATA: std_logic_vector(15 downto 0):=(others =>'0');

begin

gaire_MDR_REG:process (CLK,En,Reset,BUS_IN,MEM_IN)

begin

     if (Reset = '1') then    --act like a tri-state

         MDR_DATA <= (others=>'Z');

     elsif (CLK'evEnt and CLK = '1') then

            if (En='1') then  -- accept data from Bus (WRITE 16-bits)

                        MDR_DATA <= BUS_IN;
            elsif (En='0')then
                        MDR_DATA <= MEM_IN; -- accept data from MEM
(READ 16-bits)
            else
                        MDR_DATA <= MDR_DATA; -- Latch
```

```
            End if;
      End if;

End process;
MDR_OUT <= MDR_DATA;
End BEH;



---------------------------------------------- FSM
---------------------------------------
Library ieee;
use ieee.std_logic_1164.all;
use work.clock;

Entity LC3_FSM IS
port(
     CLK: IN STD_LOGIC;                 --clk
     Reset:IN STD_LOGIC;                --reset
     OUT_FROM_IR: IN STD_LOGIC_VECTOR(15 DOWNTO 0);  --instruction
     FROM_NZP: IN STD_LOGIC_VECTOR(2 DOWNTO 0);       --nzp flag
     LD_IR: OUT STD_LOGIC;                   --IR EnABLE
     LD_MARMUX: out std_logic; --sel Z(1), ADDER(0);
     LD_REG1: out std_logic; --Enable for reg file
     LD_PC: OUT STD_LOGIC;                   --En PC(1) TO WRITE
     LD_CC: OUT STD_LOGIC;
     LD_MAR: OUT STD_LOGIC;                  --En MAR(1) TO WRITE
```

```
     LD_MDR: OUT STD_LOGIC;                  --En MDR(1) FROM BUS,
(0) FROM MEM

     --LD_NZP: OUT STD_LOGIC;                --En NZP(1) TO WRITE

     LD_MEM: OUT STD_LOGIC;                  --En MEM91) TO WRITE,
(0) OTHERWISE

     GATE_PC1: OUT STD_LOGIC;           --(1) TO BUS

     GATE_MAR: OUT STD_LOGIC;           --(1) TO BUS

     GATE_ALU1: OUT STD_LOGIC;               --(1) TO BUS

     GATE_MDR1: OUT STD_LOGIC;               --(1) TO BUS

     ADDR2_MUX_SEL: OUT STD_LOGIC_VECTOR(1 DOWNTO 0);--(00)Z (01)S5
(10)S8 (11)S10

     ADDR1_MUX_SEL: OUT STD_LOGIC;           --(1) SR1, (0) PCIN

     SR1_SEL: OUT STD_LOGIC_VECTOR(2 DOWNTO 0);   -- OUTPUT REG

     SR2_SEL: OUT STD_LOGIC_VECTOR(2 DOWNTO 0);   --OUTPUT REG

     DR_SEL: OUT STD_LOGIC_VECTOR(2 DOWNTO 0);    --INPUT REG

     SR2MUX_SEL: OUT STD_LOGIC;              --(1)S4, (0)SR2IN

     PCMUX_SEL: OUT STD_LOGIC_VECTOR(1 DOWNTO 0);      --(00)PCIN,
(01)ADDERIN, (10)BUS IN

     ALU_SEL: OUT STD_LOGIC_VECTOR(1 DOWNTO 0);   --(00)SR11+2,
(01)SR1&SR2, (10)SR1, (11)~SR1

     MEM_En: OUT STD_LOGIC                   --(1) TO En

);


End LC3_FSM;



ARCHITECTURE BEH OF LC3_FSM IS
```

```
TYPE LC3_STATE_TYPE IS (s0, s1, s1A, S1B, S1C, S1D, S2, S4, S4A, S4B,
S4C, S4D, s4E, s4F, s4G, s5, s6, s6B, s6C, s6D,s7);

signal cpu_state: LC3_State_TYPE;
signal next_state: LC3_State_TYPE;

constant ADD: std_logic_vector(3 downto 0) :=  "0001";
constant ANDi: std_logic_vector(3 downto 0) := "0101";
constant LD: std_logic_vector(3 downto 0) := "0010";
constant ST: std_logic_vector(3 downto 0) := "0011";

begin

FSM: process(OUT_FROM_IR, FROM_NZP, cpu_state, next_state)

variable OPCODE: std_logic_vector(3 downto 0);
variable PC_OFFSET: std_logic_vector(8 downto 0);
variable SR1IN: std_logic_vector(2 downto 0);
variable SR2IN: std_logic_vector(2 downto 0);
variable DRIN: std_logic_vector(2 downto 0);
variable IR_5: std_logic;
variable IMMEDIATE: std_logic_vector(4 downto 0);
variable BASEREG: std_logic_vector(2 downto 0);

begin
```

```
-------------------------------------------------------------------------
--
------------------------- INITIALIZE
---------------------------------
-------------------------------------------------------------------------
--
case cpu_state is
          whEn s0 =>
               LD_IR <= '0';
               LD_MAR <= '0';
               LD_REG1 <= '0';
               LD_PC <= '0';
               LD_MAR <= '0';
               LD_MDR <= '0';
               LD_CC <= '0';
               LD_MEM <= '0';
               GATE_PC1 <= '0';
               GATE_MAR <= '0';
               GATE_ALU1 <= '0';
               GATE_MDR1 <= '0';
               ADDR2_MUX_SEL <= (OTHERS => '0');
               ADDR1_MUX_SEL <= '0';
               SR1_SEL <= (OTHERS => '0');
               SR2_SEL <= (OTHERS => '0');
               DR_SEL <= (OTHERS => '0');
               SR2MUX_SEL <= '0';
               PCMUX_SEL <= (OTHERS => '0');
               ALU_SEL <= (OTHERS => '0');
```

```
                MEM_En <= '0';
                next_state <= S1;

-------------------------------------------------------------------------
-
------------------------------ FETCH INSTRUCTION
---------------------------
-------------------------------------------------------------------------
-
          when S1 =>
                GATE_MAR <= '0';
                GATE_ALU1 <= '0';
                GATE_MDR1 <= '0';
                LD_CC <='0';
                LD_PC <= '1';
                LD_MAR <= '1';
                GATE_PC1 <= '1';
                PCMUX_SEL <= "10";
                MEM_EN <= '0';
                LD_MDR <= '0';
                next_state <= S1A;

          when S1A =>
                LD_PC <= '0';
                LD_IR <= '0';
                LD_CC <= '0';
                LD_MEM <= '0';
                GATE_PC1 <= '0';
```

```
        GATE_MDR1 <= '0';
        MEM_EN <= '0';
        next_state <= S1B;

    when S1B =>
        LD_PC <= '0';
        LD_IR <= '0';
        LD_CC <= '0';
        LD_MEM <= '0';
        GATE_PC1 <= '0';
        GATE_MDR1 <= '0';
        MEM_EN <= '1';
        next_state <= S1C;

    -- to turn off the bus
    when S1C =>
        LD_MDR <= '0';
        LD_IR <= '0';
        LD_CC <= '0';
        LD_MEM <= '0';
        GATE_PC1 <= '0';
        GATE_MDR1 <= '0';
        MEM_EN <= '1';
        LD_PC <= '0';
        next_state <= S1D;

    -- to turn off the bus
```

```
          when S1D =>
               LD_MDR <= '0';
               LD_IR <= '1';
               LD_CC <= '0';
               LD_MEM <= '0';
               GATE_PC1 <= '0';
               GATE_MDR1 <= '1';
               MEM_EN <= '0';
               LD_PC <= '0';
               next_state <= s2;

-------------------------------------------------------------------------
--
------------------------ DECODE INSTRUCTION
------------------------------
-------------------------------------------------------------------------
--
          when S2 =>
               OPCODE := OUT_FROM_IR(15 DOWNTO 12);
               IR_5 := OUT_FROM_IR(5);
               DRIN := OUT_FROM_IR(11 DOWNTO 9);
               SR1IN := OUT_FROM_IR(8 DOWNTO 6);
               SR2IN := OUT_FROM_IR(2 DOWNTO 0);
               IMMEDIATE := OUT_FROM_IR(4 DOWNTO 0);
               BASEREG := OUT_FROM_IR(8 DOWNTO 6);
               PC_OFFSET := OUT_FROM_IR(8 DOWNTO 0);

               case OPCODE is
```

```
                    when "0010" =>                          --0010 is
opcode for load instruction
                         ADDR2_MUX_SEL <= "01";
                         ADDR1_MUX_SEL <=  '1';
                         -- TO TURN THE BUS OFF
                         LD_MDR <= '0';
                         LD_MAR <= '0';
                         LD_IR <= '0';
                         LD_CC <= '0';
                         LD_MEM <= '0';
                         GATE_PC1 <= '0';
                         GATE_MDR1 <= '0';
                         GATE_MAR <= '0';               --(1) TO BUS
                         GATE_ALU1 <= '0';
                         MEM_EN <= '0';
                         LD_PC <= '0';
                         next_state <= s4;
                    when otherS =>
                         next_state <= s0;
               end case;


          when s4 =>
               ADDR2_MUX_SEL <= "01";
               ADDR1_MUX_SEL <=  '1';
               -- TO TURN THE BUS OFF
               LD_MDR <= '0';
               LD_MAR <= '0';
```

```
        LD_IR <= '0';
        LD_CC <= '0';
        LD_MEM <= '0';
        GATE_PC1 <= '0';
        GATE_MDR1 <= '0';
        MEM_EN <= '0';
        LD_PC <= '0';
        LD_MARMUX <= '1';
        GATE_MAR <= '1';
        next_state <= S4A;

    when s4A =>
        next_state <= s4B;

    when s4B =>
        next_state <= s4C;

    when s4C =>
        next_state <= s4D;

    when s4D =>
        MEM_EN <= '1';
        LD_MAR <= '0';
        LD_REG1 <= '0';
        LD_PC <= '1';
        next_state <= s4E;
```

```
            when s4E =>
                 LD_PC <= '0';
                 GATE_PC1 <= '0';
                 LD_MAR <= '0';
                 next_state <= s4F;

            when s4F =>
                 --MEM_EN <= '1';
                 GATE_MDR1 <= '1';
                 LD_IR <= '1';
                 next_state <= s4G;

            when s4G =>
                 GATE_MDR1 <= '0';
                 LD_IR <= '0';
                 next_state <= s5;

            when s5 =>
                 next_state <= s2;

--------------------------- ADD ------------------------
            when s6 =>
                 MEM_EN <= '1';
                 GATE_ALU1 <= '1';
                 LD_REG1 <= '1';
                 DR_SEL <= DRIN;
                 next_state <= s6B;
```

```
            when s6B =>
                 MEM_EN <= '0';
                 GATE_ALU1 <= '0';
                 LD_REG1 <= '0';
                 next_state <= s6C;
            when s6C =>
                 LD_PC <= '1';
                 LD_MAR <= '1';
                 GATE_PC1 <= '1';
                 PCMUX_SEL <= "10";
                 next_state <= s6D;
            when s6D =>
                 MEM_EN <= '1';
                 LD_MAR <= '0';
                 LD_PC <= '0';
                 LD_IR <= '0';
                 LD_CC <= '0';
                 LD_MEM <= '0';
                 GATE_PC1 <= '0';
                 GATE_MDR1 <= '0';
                 next_state <= s2;
-------------------------------------------------

            when s7 =>
                 next_state <= s2;
            when others =>
                 next_state <= s0;
```

```
     end case;
end process;
-------------------------------------------------------------------
--

nextstatelogic: process
begin

     wait until CLK'evEnt and CLK = '1';
          if (Reset ='1') then
               cpu_state <= s0;
          else
               cpu_state <= next_state;
          End if;
End process;
End BEH;
```

datapath

```
--=================================================================

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

```
entity Data_Path_ALL is
generic(
     w: integer := 16;
     p: integer := 4
);
port(
     CLK: in std_logic;
     Reset: in std_logic
);
end Data_Path_ALL;

architecture structural of Data_Path_ALL is

------------------------------------------------------------
-------------------- Component decleration -----------------
------------------------------------------------------------

-------------- 16 bit register ----------------------------

component gaire_n_bit_register is
    generic(
            w: Integer := 16
    );
    port (  CLK : in std_logic;
            Reset : IN STD_LOGIC;
            En : IN STD_LOGIC;
            OP_A : IN STD_LOGIC_VECTOR(w-1 DOWNTO 0);
```

```
            OP_Q : OUT STD_LOGIC_vector(w-1 DOWNTO 0)
        );
end component;

------------- register array -----------------------------

component gaire_register_array is
generic(w: integer:= 16;
        p: integer := 3;
        E: natural := 8
);
port(CLK: in std_logic;
    reset: in std_logic;
    LD_REG: in std_logic; -- enable to tell us which register in the
array is active for input
    DR: in std_logic_vector(p-1 downto 0);
    OP_A: in std_logic_vector(w-1 downto 0);
    SR1: in std_logic_vector(p-1 downto 0);
    SR2: in std_logic_vector(p-1 downto 0);
    OP_Q1: out std_logic_vector(w-1 downto 0);
    OP_Q2: out std_logic_vector(w-1 downto 0)
);
end component;

------------- sign extenders -----------------------------

component gaire_5_to_16_bit_sign_extender is
```

```
generic (p: integer:= 16;
         w: integer:= 5);
port (CLK: in std_logic;
      reset: in std_logic;
      OP_A: in std_logic_vector(w-1 downto 0);
      OP_Q: out std_logic_vector(p-1 downto 0)
      );
end component;

component gaire_6_to_16_bit_sign_extender is
generic (p: integer:= 16;
         w: integer:= 6);
port (CLK: in std_logic;
      reset: in std_logic;
      OP_A: in std_logic_vector(w-1 downto 0);
      OP_Q: out std_logic_vector(p-1 downto 0)
      );
end component;

component gaire_8_to_16_bit_zero_extender is
generic (p: integer:= 16;
         w: integer:= 8);
port (CLK: in std_logic;
      reset: in std_logic;
      OP_A: in std_logic_vector(w-1 downto 0);
      OP_Q: out std_logic_vector(p-1 downto 0)
      );
```

```
end component;

component gaire_9_to_16_bit_sign_extender is
generic (p: integer:= 16;
         w: integer:= 9);
port (CLK: in std_logic;
      reset: in std_logic;
      OP_A: in std_logic_vector(w-1 downto 0);
      OP_Q: out std_logic_vector(p-1 downto 0)
      );
end component;

component gaire_11_to_16_bit_sign_extender is
generic (p: integer:= 16;
         w: integer:= 11);
port (CLK: in std_logic;
      reset: in std_logic;
      OP_A: in std_logic_vector(w-1 downto 0);
      OP_Q: out std_logic_vector(p-1 downto 0)
      );
end component;

------------------------- MUXs -----------------------------

component gaire_2_to_1_multiplexer is
port(
     a_0: in std_logic_vector(15 downto 0);
```

```
     a_1: in std_logic_vector(15 downto 0);
     y: out std_logic_vector(15 downto 0);
     sel: in std_logic);
end component;

component gaire_3_to_1_multiplexer is
port(
     a_0: in std_logic_vector(15 downto 0);
     a_1: in std_logic_vector(15 downto 0);
     a_2: in std_logic_vector(15 downto 0);
     y: out std_logic_vector(15 downto 0);
     sel: in std_logic_vector(1 downto 0));
end component;

component gaire_4_to_1_multiplexer is
port(
     a_0: in std_logic_vector(15 downto 0);
     a_1: in std_logic_vector(15 downto 0);
     a_2: in std_logic_vector(15 downto 0);
     a_3: in std_logic_vector(15 downto 0);
     y: out std_logic_vector(15 downto 0);
     sel: in std_logic_vector(1 downto 0));
end component;

--------------- Adder ------------------------
component gaire_adder is
port(     OP_A: in std_logic_vector(15 DOWNTO 0);
```

```
     OP_B: in std_logic_vector(15 DOWNTO 0);
     OP_Q: out std_logic_vector(15 DOWNTO 0)
);
end component;

---------------- Tri state gate --------------------------

component gaire_tri_state_gate is
generic(w:integer:= 16);
port( EN: in std_logic;
     OP_A: in std_logic_vector(w-1 downto 0);
     OP_Q: out std_logic_vector(w-1 downto 0)
);
end component;

------------------- ALU ----------------------------------

component gaire_alu is
generic(w: integer:= 16);
port(
     OP_A: in std_logic_vector(w-1 downto 0);
     OP_B: in std_logic_vector(w-1 downto 0);
     sel: in std_logic_vector(1 downto 0);
     OP_Q: out std_logic_vector(w-1 downto 0)
);
end component;
```

```
------------------- PC counter ----------------------------

component gaire_program_counter is
     generic(w: integer := 16);
     port(CLK: in std_logic;
     reset: in std_logic;
     EN: in std_logic;
     OP_A: in std_logic_vector(w-1 downto 0);
     OP_Q: out std_logic_vector(w-1 downto 0);
     OP_Z: out std_logic_vector(w-1 downto 0)
);
end component;


--------------------- NZP ---------------------------------

component NZP is
port(
     CLK: in std_logic;
     reset: in std_logic;
     EN: in std_logic;
     OP_A: in std_logic_vector(15 downto 0);
     OP_Q: out std_logic_vector(2 downto 0)
     );
end component;

---------------------- RAM -------------------------------
```

```
component gaire_RAM is
generic(ADD_WIDTH: integer:= 9;  -- address bit size 2**9=512 words
        WIDTH: integer:= 16);  -- data size
port(
    CLK: in std_logic;  --clock
    mem_wr_rd_add: in std_logic_vector(ADD_WIDTH-1 downto 0);
    data_in: in std_logic_vector(WIDTH-1 downto 0);  --input data
    data_out: out std_logic_vector(WIDTH-1 downto 0);  --output data
    read_write_en: in std_logic;  --read(0)/write(1) enable
    mem_en: in std_logic);
end component;

------------------------- MAR ------------------------------

component gaire_MAR_REG is
port (
     CLK: std_logic;
     EN: in std_logic;
     Reset: in std_logic;
     BUS_IN: in std_logic_vector(15 downto 0);
     MAR_OUT: out std_logic_vector(8 downto 0)
);
end component;

------------------------- MDR ------------------------------
```

```
component gaire_MDR_REG is
port (
     CLK: in std_logic;
     EN: in std_logic;
     Reset: in std_logic;
     BUS_IN: in std_logic_vector(15 downto 0); --FROM BUS (WRITE)
     MEM_IN: in std_logic_vector(15 downto 0); --FROM MEM (READ)
     MDR_OUT: out std_logic_vector(15 downto 0)-- To Bus
);
end component;

------------------------ FSM -----------------------------

component LC3_FSM IS
port(
     CLK: IN STD_LOGIC;                --clk
     Reset:IN STD_LOGIC;               --reset
     OUT_FROM_IR: IN STD_LOGIC_VECTOR(15 DOWNTO 0);  --instruction
     FROM_NZP: IN STD_LOGIC_VECTOR(2 DOWNTO 0);       --nzp flag
     LD_IR: OUT STD_LOGIC;                   --IR ENABLE
     LD_MARMUX: out std_logic; --sel Z(1), ADDER(0);
     LD_REG1: out std_logic; --enable for reg file
     LD_PC: OUT STD_LOGIC;                   --EN PC(1) TO WRITE
     LD_CC: OUT STD_LOGIC;
     LD_MAR: OUT STD_LOGIC;                  --EN MAR(1) TO WRITE
     LD_MDR: OUT STD_LOGIC;                  --EN MDR(1) FROM BUS,
(0) FROM MEM
```

```
     --LD_NZP: OUT STD_LOGIC;                --EN NZP(1) TO WRITE
     LD_MEM: OUT STD_LOGIC;                  --EN MEM91) TO WRITE,
(0) OTHERWISE
     GATE_PC1: OUT STD_LOGIC;           --(1) TO BUS
     GATE_MAR: OUT STD_LOGIC;           --(1) TO BUS
     GATE_ALU1: OUT STD_LOGIC;               --(1) TO BUS
     GATE_MDR1: OUT STD_LOGIC;               --(1) TO BUS
     ADDR2_MUX_SEL: OUT STD_LOGIC_VECTOR(1 DOWNTO 0);--(00)Z (01)S5
(10)S8 (11)S10
     ADDR1_MUX_SEL: OUT STD_LOGIC;           --(1) SR1, (0) PCIN
     SR1_SEL: OUT STD_LOGIC_VECTOR(2 DOWNTO 0);   -- OUTPUT REG
     SR2_SEL: OUT STD_LOGIC_VECTOR(2 DOWNTO 0);   --OUTPUT REG
     DR_SEL: OUT STD_LOGIC_VECTOR(2 DOWNTO 0);    --INPUT REG
     SR2MUX_SEL: OUT STD_LOGIC;              --(1)S4, (0)SR2IN
     PCMUX_SEL: OUT STD_LOGIC_VECTOR(1 DOWNTO 0);      --(00)PCIN,
(01)ADDERIN, (10)BUS IN
     ALU_SEL: OUT STD_LOGIC_VECTOR(1 DOWNTO 0);   --(00)SR11+2,
(01)SR1&SR2, (10)SR1, (11)~SR1
     MEM_EN: OUT STD_LOGIC                   --(1) TO EN
);

end component;

-------------------------------------------------------------

---------------------- SIGNALS ----------------------------

-------------------------------------------------------------

SIGNAL FROM_IR : std_logic_vector (15 downto 0);
```

```
SIGNAL FROM_SEXT5 : std_logic_vector (15 downto 0);
SIGNAL FROM_SEXT8 : std_logic_vector (15 downto 0);
SIGNAL FROM_SEXT10: std_logic_vector (15 downto 0);
SIGNAL FROM_ZEXT7 : std_logic_vector (15 downto 0);
SIGNAL FROM_SEXT4 : std_logic_vector (15 downto 0);


SIGNAL LD_REG1 : std_logic;
SIGNAL LD_IR : std_logic;
SIGNAL DR_SEL : std_logic_vector (2 downto 0);
SIGNAL FROM_BUS : std_logic_vector (15 downto 0);
SIGNAL SR1_SEL: std_logic_vector (2 downto 0);
SIGNAL SR2_SEL : std_logic_vector (2 downto 0);
SIGNAL FROM_SR1_RF : std_logic_vector (15 downto 0);
SIGNAL FROM_SR2_RF : std_logic_vector (15 downto 0);
SIGNAL FROM_SR2_MUX : std_logic_vector (15 downto 0);
SIGNAL ALU_SEL: std_logic_vector (1 downto 0);
SIGNAL FROM_ALU: std_logic_vector (15 downto 0);
SIGNAL GATE_ALU1: std_logic;

SIGNAL ADDR1_MUX_SEL : std_logic;
SIGNAL ADDR2_MUX_SEL : std_logic_vector (1 downto 0);
SIGNAL SR2MUX_SEL : std_logic;
SIGNAL ZERO : std_logic_vector (15 downto 0);
SIGNAL FROM_PCMUX : std_logic_vector (15 downto 0);
SIGNAL FROM_ADDR2MUX : std_logic_vector (15 downto 0);
SIGNAL FROM_ADDR1MUX : std_logic_vector (15 downto 0);
```

```
SIGNAL FROM_PC_CNTRBUS : std_logic_vector (15 downto 0);
SIGNAL FROM_PC_CNTRINC : std_logic_vector (15 downto 0);
SIGNAL FROM_PC1 : std_logic_vector (15 downto 0);
SIGNAL FROM_BUS_NZP : std_logic_vector (15 downto 0);
SIGNAL FROM_NZP: std_logic_vector(2 downto 0);
SIGNAL FROM_ADDER : std_logic_vector (15 downto 0);
SIGNAL FROM_MARMUX : std_logic_vector (15 downto 0);
SIGNAL MEM_EN : std_logic;
SIGNAL FROM_MDR : std_logic_vector (15 downto 0);
SIGNAL FROM_MEM : std_logic_vector (15 downto 0);
SIGNAL FROM_MAR : std_logic_vector (8 downto 0);
SIGNAL LD_PC : std_logic;
SIGNAL LD_MAR : std_logic;
SIGNAL LD_MEM : std_logic;
SIGNAL LD_MDR : std_logic;
SIGNAL LD_MARMUX : std_logic;
SIGNAL LD_CC : std_logic;

SIGNAL PCMUX_SEL : std_logic_vector (1 downto 0);
SIGNAL GATE_PC1 : std_logic;
SIGNAL GATE_MAR : std_logic;
SIGNAL GATE_MDR1 : std_logic;

-------------------------------------------------------------

---------------------- INSTANTIATE ----------------------

-------------------------------------------------------------
```

```
begin

Bus_to_IR : gaire_n_bit_register port map (CLK, Reset, LD_IR,
FROM_BUS, FROM_IR);

IR_TO_SEXT4 : gaire_5_to_16_bit_sign_extender generic map(w, (p+1))
port map ( CLK, Reset, FROM_IR(4 downto 0), FROM_SEXT4);

IR_TO_SEXT5 : gaire_6_to_16_bit_sign_extender generic map(w, (p+2))
port map ( CLK, Reset, FROM_IR(5 downto 0), FROM_SEXT5);

IR_TO_ZEXT7 : gaire_8_to_16_bit_zero_extender generic map(w, (p+4))
port map ( CLK, Reset, FROM_IR(7 downto 0), FROM_ZEXT7);

IR_TO_SEXT8 : gaire_9_to_16_bit_sign_extender generic map(w, (p+5))
port map ( CLK, Reset, FROM_IR(8 downto 0), FROM_SEXT8);

IR_TO_SEXT10 : gaire_11_to_16_bit_sign_extender generic map(w, (p+7))
port map ( CLK, Reset, FROM_IR(10 downto 0), FROM_SEXT10);

FROM_REG_FILE : gaire_register_array generic map (w, p-1, 8) port map
(CLK, Reset, LD_REG1, DR_SEL, FROM_BUS, SR1_SEL, SR2_SEL,
FROM_SR1_RF, FROM_SR2_RF);

SR2MUX_TO_ALU : gaire_2_to_1_multiplexer port map
(FROM_SR2_RF,FROM_SEXT4,FROM_SR2_MUX, SR2MUX_SEL);

FROM_ALU_TO_BUS : gaire_alu port map (FROM_SR1_RF, FROM_SR2_RF,
ALU_SEL, FROM_ALU);

FROM_GALU_TO_BUS : gaire_tri_state_gate generic map(w) port map
(GATE_ALU1, FROM_ALU, FROM_BUS );

FROM_ADDR2_MUX : gaire_4_to_1_multiplexer port map (FROM_SEXT10,
FROM_SEXT8, FROM_SEXT5,ZERO, FROM_ADDR2MUX, ADDR2_MUX_SEL);

FROM_ADDR1_MUX : gaire_2_to_1_multiplexer port map
(FROM_SR1_RF,FROM_PC_CNTRBUS, FROM_ADDR1MUX, ADDR1_MUX_SEL);
```

```
FROM_ADDER_16 : gaire_adder port map (FROM_ADDR2MUX, FROM_ADDR1MUX,
FROM_ADDER);


FROM_PC_MUX : gaire_3_to_1_multiplexer port map (FROM_BUS,
FROM_ADDER, FROM_PC_CNTRINC, FROM_PCMUX, PCMUX_SEL);

FROM_PC_CNT : gaire_program_counter generic map (w) port map (CLK,
Reset, LD_PC, FROM_PCMUX, FROM_PC_CNTRBUS, FROM_PC_CNTRINC);

FROM_PC_TO_BUS : gaire_tri_state_gate generic map(w) port map
(GATE_PC1, FROM_PC_CNTRBUS, FROM_BUS );


MARMUX_TO_TRI : gaire_2_to_1_multiplexer port map (FROM_ZEXT7,
FROM_ADDER, FROM_MARMUX, LD_MARMUX);

FROM_MARM_TO_BUS : gaire_tri_state_gate generic map(w) port map
(GATE_MAR, FROM_MARMUX, FROM_BUS);


FROM_MAR_TO_MEM : gaire_MAR_REG port map (CLK, LD_MAR, Reset,
FROM_BUS, FROM_MAR);

FROM_MEM_TO_MDR : gaire_RAM generic map(((2*p)+1), w) port map (CLK,
FROM_MAR, FROM_MDR, FROM_MEM, LD_MEM, MEM_EN);

FROM_MDR_TO_BORM : gaire_MDR_REG port map (CLK,LD_MDR, Reset,
FROM_BUS, FROM_MEM, FROM_MDR);

FROM_MDR_TO_BUS : gaire_tri_state_gate generic map(w) port map
(GATE_MDR1, FROM_MDR, FROM_BUS);


NZP_TO_FSM : NZP port map (CLK, Reset, LD_CC, FROM_BUS_NZP,
FROM_NZP);


FSM_TO_DP : LC3_FSM port map (CLK, Reset, FROM_IR, FROM_NZP, LD_IR,
LD_MARMUX, LD_REG1, LD_PC, LD_CC, LD_MAR, LD_MDR, LD_MEM, GATE_PC1,
GATE_MAR,
```

```
GATE_ALU1, GATE_MDR1, ADDR2_MUX_SEL, ADDR1_MUX_SEL, SR1_SEL, SR2_SEL,
DR_SEL, SR2MUX_SEL, PCMUX_SEL, ALU_SEL, MEM_EN );

end structural;
```

Testbench

```
----------------------testbench LC-3 ----------------------------
--=================================================================

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.all;

entity DATAPATH_ALL_TB is
end DATAPATH_ALL_TB;


architecture TB1 of DATAPATH_ALL_TB is

component Data_Path_ALL is
port (
     CLK: in std_logic;
     Reset: in std_logic
     );
```

```
end component Data_Path_ALL;

signal    CLKtb          : std_logic;
signal    Resettb   : std_logic;

begin

CLK_GEN: process
begin
while now <= 300 ns loop
CLKtb <='0'; wait for 5 ns; CLKtb <='1'; wait for 5 ns;
end loop;
wait;
end process;

Reset: process
begin
Resettb  <='1','0' after 2 ns;
wait;
end process;


---------------------------------------do not
change-----------------------------------------------
datap: Data_Path_ALL port map ( CLK=>CLKtb, Reset=>Resettb);

end TB1;
```

```
configuration CFG_DATAPATH_ALL_TB of DATAPATH_ALL_TB is
     for TB1
     end for;
end CFG_DATAPATH_ALL_TB;
```