

-- **** STUDENT: 64000225.....	2
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	2
-- **** STUDENT: 64200100.....	14
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	14
-- **** STUDENT: 64200112.....	26
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	26
-- **** STUDENT: 64200238.....	38
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	38
-- **** STUDENT: 64200288.....	50
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	50
-- **** STUDENT: 64200296.....	62
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	62
-- **** STUDENT: 64200385.....	74
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	74
-- **** STUDENT: 64210113.....	86
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	86
-- **** STUDENT: 64210290.....	98
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	98
-- **** STUDENT: 64210382.....	110
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	110
-- **** STUDENT: 64210384.....	122
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	122
-- **** STUDENT: 64210386.....	134
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	134
-- **** STUDENT: 64210445.....	146
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	146
-- **** STUDENT: 64210455.....	158
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	158
-- **** STUDENT: 64210457.....	170
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	170
-- **** STUDENT: 64240430.....	182
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	182
-- **** PREDLOGA VAJE.....	194
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	194

```

-- *****
-- **** STUDENT: 64000225
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE      : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB              : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD              : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```

```

constant    ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant    ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant    ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant    ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant    ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant    ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal
-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,      -- A bus register number select input
BA => BA,      -- B bus register number select input
MB => MB,      -- constant/operand bus B bus multiplexer control signal
MD => MD,      -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F          operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode =>      ALU_mode,      -- mode of alu operation ( M in upper table )
ALU_function =>  ALU_function,-- function of ALU ( F in upper table )
ALU_N_bit  =>      ALU_N_bit,   -- Negative bit of alu operation
ALU_C_bit  =>      ALU_C_bit,   -- Carry bit of alu operation
ALU_V_bit  =>      ALU_V_bit,   -- Overflow bit of alu operation
ALU_Z_bit  =>      ALU_Z_bit,   -- Zero bit of alu operation
Const_in   =>      Const_in,   -- constant operand input bus
Data_in     =>      Data_in,    -- data input bus input
Address_out =>      Address_out, -- address bus output
Data_out    =>      Data_out    -- data bus output

);

PROCESS      -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk  <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk  <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(

```

```

        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 );      -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit           : std_logic; -- Carry bit of alu operation
        ALU_V_bit           : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit           : std_logic; -- Zero bit of alu operation
        Const_in           : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in            : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out        : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out           : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment            : string
    ) IS

```

```

    VARIABLE RES_LINE : LINE;

```

```

    BEGIN

```

```

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

```

```

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );

```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA    AA          BA    MB          FS          MD    RW

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word    <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in    <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word    <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word    <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in    <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word    <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word    <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word    <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B      MEM( R( 3 ) ) <= R( 7 )
               ctrl_word    <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST  MEM( R( 3 ) ) <= Const_in
               Const_in     <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word    <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in     <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word    <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR          R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word    <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word    <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--      +--- -- -- -- -- -- -- -- -- -- -- -- -- -- Destination
register address
--      |          +--- -- -- -- -- -- -- -- -- -- -- A bus
register address
--      |  |          +--- -- -- -- -- -- -- -- -- -- B bus register address
--      |  |          |  +--- -- -- -- -- -- -- -- -- constant/B bus operand
multiplexer
--      |          |          |  |          |          ( constant = '1' )
--      |          |          |  |          |          +--- -- -- -- --
ALU function select
--      |          |          |  |          |          +--- -- -- -- --
external data input/ ALU result multiplexer
--      |          |          |  |          |          |          ( ALU result
multiplexer = '0' )
--      |          |          |  |          |          |          +--- -- -- -- -- destination
register result write ( active = '1' )
--      |          |          |  |          |          |          |
--      DA      AA          BA      MB          FS          MD      RW

-- DEC R( 7 )
        Const_in    <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word    <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```

```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** STUDENT: 64200100
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE      : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB          : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD          : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```



```

constant    ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant    ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant    ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant    ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant    ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant    ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,      -- A bus register number select input
BA => BA,      -- B bus register number select input
MB => MB,      -- constant/operand bus B bus multiplexer control signal
MD => MD,      -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F          operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode =>      ALU_mode,      -- mode of alu operation ( M in upper table )
ALU_function =>  ALU_function, -- function of ALU ( F in upper table )
ALU_N_bit  =>      ALU_N_bit,   -- Negative bit of alu operation
ALU_C_bit  =>      ALU_C_bit,   -- Carry bit of alu operation
ALU_V_bit  =>      ALU_V_bit,   -- Overflow bit of alu operation
ALU_Z_bit  =>      ALU_Z_bit,   -- Zero bit of alu operation
Const_in   =>      Const_in,    -- constant operand input bus
Data_in     =>      Data_in,     -- data input bus input
Address_out =>      Address_out, -- address bus output
Data_out    =>      Data_out     -- data bus output

);

PROCESS      -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk  <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk  <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(

```

```

        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 );      -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit          : std_logic; -- Carry bit of alu operation
        ALU_V_bit          : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit          : std_logic; -- Zero bit of alu operation
        Const_in           : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in            : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out        : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out           : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment            : string
    ) IS

```

```

    VARIABLE RES_LINE : LINE;

```

```

    BEGIN

```

```

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

```

```

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );

```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA      AA          BA      MB          FS          MD      RW

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word      <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in          <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word      <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word      <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in          <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word      <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word   <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word   <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word   <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in    <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word   <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in    <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word   <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word   <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word  <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--
--      +--- -- -- -- -- -- -- -- -- -- -- -- -- -- Destination
register address
--      |          +--- -- -- -- -- -- -- -- -- -- -- -- A bus
register address
--      |      |          +--- -- -- -- -- -- -- -- -- -- B bus register address
--      |          |          +--- -- -- -- -- -- -- -- -- constant/B bus operand
multiplexer
--      |          |          |          |          |          |          |          |          |          |
--      |          |          |          |          |          |          |          |          |          |
ALU function select
--      |          |          |          |          |          |          |          |          |          |
external data input/ ALU result multiplexer
--      |          |          |          |          |          |          |          |          |          |
multiplexer = '0' )
--      |          |          |          |          |          |          |          |          |          |
register result write ( active = '1' )
--      |          |          |          |          |          |          |          |          |          |
--      DA      AA          BA      MB          FS          MD      RW

-- DEC R( 7 )
        Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word      <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```



```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** STUDENT: 64200112
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE      : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB          : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD          : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```

```

constant    ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant    ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant    ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant    ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant    ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant    ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal
-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,      -- A bus register number select input
BA => BA,      -- B bus register number select input
MB => MB,      -- constant/operand bus B bus multiplexer control signal
MD => MD,      -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F          operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode =>      ALU_mode,      -- mode of alu operation ( M in upper table )
ALU_function =>  ALU_function, -- function of ALU ( F in upper table )
ALU_N_bit  =>      ALU_N_bit,   -- Negative bit of alu operation
ALU_C_bit  =>      ALU_C_bit,   -- Carry bit of alu operation
ALU_V_bit  =>      ALU_V_bit,   -- Overflow bit of alu operation
ALU_Z_bit  =>      ALU_Z_bit,   -- Zero bit of alu operation
Const_in   =>      Const_in,    -- constant operand input bus
Data_in     =>      Data_in,     -- data input bus input
Address_out =>      Address_out, -- address bus output
Data_out    =>      Data_out     -- data bus output

);

PROCESS      -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk  <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk  <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
    END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(

```

```

        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit          : std_logic; -- Carry bit of alu operation
        ALU_V_bit          : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit          : std_logic; -- Zero bit of alu operation
        Const_in           : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in            : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out        : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out           : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment            : string
    ) IS

```

```

    VARIABLE RES_LINE : LINE;

```

```

    BEGIN

```

```

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

```

```

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );

```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA    AA          BA    MB          FS          MD    RW

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word    <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in    <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word    <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word    <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in    <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word    <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word    <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word    <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word    <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in     <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word    <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in     <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word    <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word    <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word  <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--
--      +--- -- -- -- -- -- -- -- -- -- -- -- -- -- Destination
register address
--      |          +--- -- -- -- -- -- -- -- -- -- -- -- A bus
register address
--      |      |          +--- -- -- -- -- -- -- -- -- -- B bus register address
--      |          |          +--- -- -- -- -- -- -- -- -- constant/B bus operand
multiplexer
--      |          |          |          |          |          |          |          |          |          |
--      |          |          |          |          |          |          |          |          |          |
ALU function select
--      |          |          |          |          |          |          |          |          |          |
external data input/ ALU result multiplexer
--      |          |          |          |          |          |          |          |          |          |
multiplexer = '0' )
--      |          |          |          |          |          |          |          |          |          |
register result write ( active = '1' )
--      |          |          |          |          |          |          |          |          |          |
--      DA      AA          BA      MB          FS          MD          RW

-- DEC R( 7 )
        Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word      <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```

```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** STUDENT: 64200238
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE      : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB              : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD              : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```



```

constant    ALU_X_OR_Y      :          std_logic_vector( 3 downto 0 ) := "1010";
constant    ALU_X_NOR_Y    :          std_logic_vector( 3 downto 0 ) := "1011";
constant    ALU_X_XOR_Y    :          std_logic_vector( 3 downto 0 ) := "1100";
constant    ALU_X_XNOR_Y   :          std_logic_vector( 3 downto 0 ) := "1101";
constant    ALU_X           :          std_logic_vector( 3 downto 0 ) := "1110";
constant    ALU_Y           :          std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal
-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,      -- A bus register number select input
BA => BA,      -- B bus register number select input
MB => MB,      -- constant/operand bus B bus multiplexer control signal
MD => MD,      -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode =>      ALU_mode,      -- mode of alu operation ( M in upper table )
ALU_function =>  ALU_function,-- function of ALU ( F in upper table )
ALU_N_bit  =>      ALU_N_bit,  -- Negative bit of alu operation
ALU_C_bit  =>      ALU_C_bit,  -- Carry bit of alu operation
ALU_V_bit  =>      ALU_V_bit,  -- Overflow bit of alu operation
ALU_Z_bit  =>      ALU_Z_bit,  -- Zero bit of alu operation
Const_in   =>      Const_in,   -- constant operand input bus
Data_in     =>      Data_in,    -- data input bus input
Address_out =>      Address_out, -- address bus output
Data_out    =>      Data_out    -- data bus output

);

PROCESS      -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk  <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk  <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
    END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(

```

```

        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 );      -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit          : std_logic; -- Carry bit of alu operation
        ALU_V_bit          : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit          : std_logic; -- Zero bit of alu operation
        Const_in           : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in            : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out        : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out           : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment            : string
    ) IS

```

```

    VARIABLE RES_LINE : LINE;

```

```

    BEGIN

```

```

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

```

```

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );

```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA    AA          BA    MB          FS          MD    RW

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word    <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in    <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word    <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word    <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in    <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word    <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word   <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word   <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word   <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in    <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word   <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in    <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word   <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word   <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word    <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--
--      +--- -- -- -- -- -- -- -- -- -- -- -- -- -- Destination
register address
--      |          +--- -- -- -- -- -- -- -- -- -- -- -- A bus
register address
--      |      |          +--- -- -- -- -- -- -- -- -- -- B bus register address
--      |          |          +--- -- -- -- -- -- -- -- -- constant/B bus operand
multiplexer
--      |          |          |          |          |          |          |          |          |          |
--      |          |          |          |          |          |          |          |          |          |
ALU function select
--      |          |          |          |          |          |          |          |          |          |
external data input/ ALU result multiplexer
--      |          |          |          |          |          |          |          |          |          |
multiplexer = '0' )
--      |          |          |          |          |          |          |          |          |          |
register result write ( active = '1' )
--      |          |          |          |          |          |          |          |          |          |
--      DA      AA      BA      MB      FS      MD      RW

-- DEC R( 7 )
        Const_in    <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word    <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```



```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** STUDENT: 64200288
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE      : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB              : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD              : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```

```

constant  ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant  ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant  ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant  ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant  ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant  ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal
-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,      -- A bus register number select input
BA => BA,      -- B bus register number select input
MB => MB,      -- constant/operand bus B bus multiplexer control signal
MD => MD,      -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F          operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode =>      ALU_mode,      -- mode of alu operation ( M in upper table )
ALU_function =>  ALU_function,-- function of ALU ( F in upper table )
ALU_N_bit  =>      ALU_N_bit,   -- Negative bit of alu operation
ALU_C_bit  =>      ALU_C_bit,   -- Carry bit of alu operation
ALU_V_bit  =>      ALU_V_bit,   -- Overflow bit of alu operation
ALU_Z_bit  =>      ALU_Z_bit,   -- Zero bit of alu operation
Const_in   =>      Const_in,    -- constant operand input bus
Data_in     =>      Data_in,     -- data input bus input
Address_out =>      Address_out, -- address bus output
Data_out    =>      Data_out     -- data bus output

);

PROCESS      -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk  <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk  <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(

```

```

        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 );      -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit           : std_logic; -- Carry bit of alu operation
        ALU_V_bit           : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit           : std_logic; -- Zero bit of alu operation
        Const_in            : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in             : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out         : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out            : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment             : string
    ) IS

```

```

    VARIABLE RES_LINE : LINE;

```

```

    BEGIN

```

```

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

```

```

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );

```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

        writeline( RESULTS, RES_LINE );
    END;

    variable HDR_line : LINE;

    BEGIN
        write( HDR_line, string'( "clk" & HT & "ctrl_word" & HT & "RW" & HT & "DA" & HT & "AA" & HT & "BA"
    & HT & "MB" & HT & "MD" & HT & "ALU_mode ALU_function" & HT & "N" & HT & "C" & HT & "V" & HT & "Z" & HT & "Const_in" &
    HT & "Data_in" & HT & "Address_out" & HT & "Data_out" & HT & "Comment" ) );
        report HDR_line.all;-- send to report
        writeline( RESULTS, HDR_line );

        Data_in      <= std_logic_vector( to_unsigned( 11, Data_in'length ) ); -- io data in is 11
        Const_in     <= std_logic_vector( to_unsigned( 1, Const_in'length ) ); -- constant operand is 1

        WAIT FOR ( PERIOD );
        nRST <= '1';      -- reset off
        WAIT FOR ( PERIOD );

        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
        MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,      ALU_V_bit,ALU_Z_bit,Const_in,
        Data_in,Address_out,Data_out, "RESET" );

        --      operation code bit positions
        --      +--  --  --  --  --  --  --  --  --  --  --  --  -- - Destination
        register address
        --      /      +--  --  --  --  --  --  --  --  --  --  --  --  -- A bus
        register address
        --      /      /      +--  --  --  --  --  --  --  --  --  -- B bus register address
        --      /      /      /      +--  --  --  --  --  --  --  --  --  -- constant/B bus operand
        multiplexer
        --      /      /      /      /      ( constant = '1' )
        --      /      /      /      /      +--  --  --  --  --  --
        ALU function select
        --      /      /      /      /      /      +--  --  --  --
        external data input/ ALU result multiplexer
        --      /      /      /      /      /      /      ( ALU result
        multiplexer = '0' )
    
```

```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA      AA          BA      MB          FS          MD      RW          /

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word      <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in          <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word      <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word      <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in          <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word      <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word    <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word    <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word    <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in     <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word    <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in     <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word    <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word    <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word  <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--      +--  --  --  --  --  --  --  --  --  --  --  --  --  --  -- Destination
register address
--      |      +--  --  --  --  --  --  --  --  --  --  --  --  --  -- A bus
register address
--      |      |      +--  --  --  --  --  --  --  --  --  --  -- B bus register address
--      |      |      |      +--  --  --  --  --  --  --  --  --  --  -- constant/B bus operand
multiplexer
--      |      |      |      |      |      |      |      |      |      |      |      |      |
--      |      |      |      |      |      |      |      |      |      |      |      |      |
ALU function select
--      |      |      |      |      |      |      |      |      |      |      |      |      |
external data input/ ALU result multiplexer
--      |      |      |      |      |      |      |      |      |      |      |      |      |
multiplexer = '0' )
--      |      |      |      |      |      |      |      |      |      |      |      |      |
register result write ( active = '1' )
--      |      |      |      |      |      |      |      |      |      |      |      |      |
--      DA      AA      BA      MB      FS      MD      RW

-- DEC R( 7 )
        Const_in  <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word  <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```

```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** STUDENT: 64200296
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE      : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB              : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD              : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```



```

constant    ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant    ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant    ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant    ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant    ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant    ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal
-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,      -- A bus register number select input
BA => BA,      -- B bus register number select input
MB => MB,      -- constant/operand bus B bus multiplexer control signal
MD => MD,      -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F          operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode =>      ALU_mode,      -- mode of alu operation ( M in upper table )
ALU_function =>  ALU_function,-- function of ALU ( F in upper table )
ALU_N_bit  =>      ALU_N_bit,   -- Negative bit of alu operation
ALU_C_bit  =>      ALU_C_bit,   -- Carry bit of alu operation
ALU_V_bit  =>      ALU_V_bit,   -- Overflow bit of alu operation
ALU_Z_bit  =>      ALU_Z_bit,   -- Zero bit of alu operation
Const_in   =>      Const_in,    -- constant operand input bus
Data_in     =>      Data_in,     -- data input bus input
Address_out =>      Address_out, -- address bus output
Data_out    =>      Data_out     -- data bus output

);

PROCESS      -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk  <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk  <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(

```

```

        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 );      -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit          : std_logic; -- Carry bit of alu operation
        ALU_V_bit          : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit          : std_logic; -- Zero bit of alu operation
        Const_in           : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in            : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out        : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out           : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment            : string
    ) IS

```

```

    VARIABLE RES_LINE : LINE;

```

```

    BEGIN

```

```

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

```

```

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );

```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA    AA          BA    MB          FS          MD    RW

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word    <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in    <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word    <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word    <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in    <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word    <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word    <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word    <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word    <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in     <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word    <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in     <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word    <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word    <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word    <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--
--      +---  --  --  --  --  --  --  --  --  --  --  --  --  --  -- Destination
register address
--      |
--      |      +---  --  --  --  --  --  --  --  --  --  --  --  --  -- A bus
register address
--      |      |
--      |      |      +---  --  --  --  --  --  --  --  --  --  -- B bus register address
--      |      |      |      +---  --  --  --  --  --  --  --  --  --  -- constant/B bus operand
multiplexer
--      |      |      |      |      |
--      |      |      |      |      |      ( constant = '1' )
--      |      |      |      |      |      +---  --  --  --  --  --
ALU function select
--      |      |      |      |      |      |      +---  --  --  --  --
external data input/ ALU result multiplexer
--      |      |      |      |      |      |      |      ( ALU result
multiplexer = '0' )
--      |      |      |      |      |      |      |      +---  --  --  --  -- destination
register result write ( active = '1' )
--      |      |      |      |      |      |      |      |
--      DA      AA      BA      MB      FS      MD      RW

-- DEC R( 7 )
        Const_in    <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word    <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```



```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** STUDENT: 64200385
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    --   operation code bit positions
    --   bit#
    --   15      DA( 2 )      :      destination register address
    --   14      DA( 1 )
    --   13      DA( 0 )
    --   12      AA( 2 )      :      A bus register address
    --   11      AA( 1 )
    --   10      AA( 0 )
    --   9       BA( 2 )      :      B bus register address
    --   8       BA( 1 )
    --   7       BA( 0 )
    --   6       MB          : constant/B bus operand multiplexer
    --   5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE      : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB          : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD          : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```

```

constant    ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant    ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant    ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant    ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant    ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant    ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal
-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,      -- A bus register number select input
BA => BA,      -- B bus register number select input
MB => MB,      -- constant/operand bus B bus multiplexer control signal
MD => MD,      -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F          operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode =>      ALU_mode,      -- mode of alu operation ( M in upper table )
ALU_function =>  ALU_function, -- function of ALU ( F in upper table )
ALU_N_bit  =>      ALU_N_bit,   -- Negative bit of alu operation
ALU_C_bit  =>      ALU_C_bit,   -- Carry bit of alu operation
ALU_V_bit  =>      ALU_V_bit,   -- Overflow bit of alu operation
ALU_Z_bit  =>      ALU_Z_bit,   -- Zero bit of alu operation
Const_in   =>      Const_in,    -- constant operand input bus
Data_in     =>      Data_in,    -- data input bus input
Address_out =>      Address_out, -- address bus output
Data_out    =>      Data_out,    -- data bus output

);

PROCESS      -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk  <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk  <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(

```

```

        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit           : std_logic; -- Carry bit of alu operation
        ALU_V_bit           : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit           : std_logic; -- Zero bit of alu operation
        Const_in            : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in             : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out          : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out             : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment              : string
    ) IS

```

```

    VARIABLE RES_LINE : LINE;

```

```

    BEGIN

```

```

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

```

```

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );

```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA    AA          BA    MB          FS          MD    RW

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word    <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in    <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word    <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word    <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in    <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word    <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word    <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word    <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word    <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in     <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word    <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in     <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word    <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word    <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word  <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--
--      +---  --  --  --  --  --  --  --  --  --  --  --  --  --  -- Destination
register address
--      |
--      |      +---  --  --  --  --  --  --  --  --  --  --  --  --  -- A bus
register address
--      |      |
--      |      |      +---  --  --  --  --  --  --  --  --  --  -- B bus register address
--      |      |      |      +---  --  --  --  --  --  --  --  --  --  -- constant/B bus operand
multiplexer
--      |      |      |      |      |
--      |      |      |      |      |      +---  --  --  --  --  --  --  --  --  --  --
ALU function select
--      |      |      |      |      |      |      +---  --  --  --  --  --  --  --  --
external data input/ ALU result multiplexer
--      |      |      |      |      |      |      |      ( ALU result
multiplexer = '0' )
--      |      |      |      |      |      |      |      +---  --  --  --  --  --  --  --  --  --  -- destination
register result write ( active = '1' )
--      |      |      |      |      |      |      |      |      |
--      DA      AA      BA      MB      FS      MD      RW

-- DEC R( 7 )
        Const_in  <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word  <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```

```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** STUDENT: 64210113
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE     : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB          : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD          : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode      : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit     : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit     : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit     : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit     : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```



```

constant  ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant  ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant  ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant  ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant  ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant  ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal
-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,      -- A bus register number select input
BA => BA,      -- B bus register number select input
MB => MB,      -- constant/operand bus B bus multiplexer control signal
MD => MD,      -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F          operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode =>      ALU_mode,      -- mode of alu operation ( M in upper table )
ALU_function =>  ALU_function,-- function of ALU ( F in upper table )
ALU_N_bit  =>      ALU_N_bit,   -- Negative bit of alu operation
ALU_C_bit  =>      ALU_C_bit,   -- Carry bit of alu operation
ALU_V_bit  =>      ALU_V_bit,   -- Overflow bit of alu operation
ALU_Z_bit  =>      ALU_Z_bit,   -- Zero bit of alu operation
Const_in   =>      Const_in,    -- constant operand input bus
Data_in     =>      Data_in,     -- data input bus input
Address_out =>      Address_out, -- address bus output
Data_out    =>      Data_out     -- data bus output

);

PROCESS      -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk  <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk  <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(

```

```

        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 );      -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit          : std_logic; -- Carry bit of alu operation
        ALU_V_bit          : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit          : std_logic; -- Zero bit of alu operation
        Const_in           : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in            : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out        : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out           : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment            : string
    ) IS

```

```

    VARIABLE RES_LINE : LINE;

```

```

    BEGIN

```

```

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

```

```

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );

```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA      AA          BA      MB          FS          MD      RW          /

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word      <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in          <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word      <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word      <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in          <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word      <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word    <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word    <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word    <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in     <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word    <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in     <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word    <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word    <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word    <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--      +--- -- -- -- -- -- -- -- -- -- -- -- -- -- Destination
register address
--      |          +--- -- -- -- -- -- -- -- -- -- -- -- A bus
register address
--      |      |          +--- -- -- -- -- -- -- -- -- -- B bus register address
--      |          |          +--- -- -- -- -- -- -- -- -- constant/B bus operand
multiplexer
--      |          |          |          |          |          |          |          |          |          |
--      |          |          |          |          |          |          |          |          |          |
ALU function select
--      |          |          |          |          |          |          |          |          |          |
external data input/ ALU result multiplexer
--      |          |          |          |          |          |          |          |          |          |
multiplexer = '0' )
--      |          |          |          |          |          |          |          |          |          |
register result write ( active = '1' )
--      |          |          |          |          |          |          |          |          |          |
--      DA      AA          BA      MB          FS          MD      RW

-- DEC R( 7 )
        Const_in    <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word    <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```



```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** STUDENT: 64210290
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE      : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB          : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD          : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```

```

constant    ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant    ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant    ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant    ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant    ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant    ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal
-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,    -- A bus register number select input
BA => BA,    -- B bus register number select input
MB => MB,    -- constant/operand bus B bus multiplexer control signal
MD => MD,    -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode => ALU_mode,    -- mode of alu operation ( M in upper table )
ALU_function => ALU_function, -- function of ALU ( F in upper table )
ALU_N_bit => ALU_N_bit,    -- Negative bit of alu operation
ALU_C_bit => ALU_C_bit,    -- Carry bit of alu operation
ALU_V_bit => ALU_V_bit,    -- Overflow bit of alu operation
ALU_Z_bit => ALU_Z_bit,    -- Zero bit of alu operation
Const_in => Const_in,    -- constant operand input bus
Data_in => Data_in,    -- data input bus input
Address_out => Address_out, -- address bus output
Data_out => Data_out    -- data bus output

);

PROCESS    -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(

```

```

        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 );      -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit          : std_logic; -- Carry bit of alu operation
        ALU_V_bit          : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit          : std_logic; -- Zero bit of alu operation
        Const_in           : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in            : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out        : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out           : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment            : string
    ) IS

```

```

    VARIABLE RES_LINE : LINE;

```

```

    BEGIN

```

```

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

```

```

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );

```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA    AA          BA    MB          FS          MD    RW

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word    <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in    <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word    <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word    <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in    <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word    <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word    <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word    <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word    <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in     <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word    <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in     <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word    <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word    <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

WAIT FOR ( PERIOD );
Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
ctrl_word    <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
WAIT FOR ( PERIOD );
Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--      +---  --  --  --  --  --  --  --  --  --  --  --  --  --  -- Destination
register address
--      /      +---  --  --  --  --  --  --  --  --  --  --  --  --  -- A bus
register address
--      /      /      +---  --  --  --  --  --  --  --  --  --  -- B bus register address
--      /      /      /      +---  --  --  --  --  --  --  -- constant/B bus operand
multiplexer
--      /      /      /      /      ( constant = '1' )
--      /      /      /      /      +---  --  --  --  --  --  --
ALU function select
--      /      /      /      /      /      +---  --  --  --  --
external data input/ ALU result multiplexer
--      /      /      /      /      /      /      ( ALU result
multiplexer = '0' )
--      /      /      /      /      /      /      +---  --  destination
register result write ( active = '1' )
--      /      /      /      /      /      /      /
--      DA      AA      BA      MB      FS      MD      RW

-- DEC R( 7 )
Const_in    <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
ctrl_word    <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
WAIT FOR ( PERIOD );
Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```

```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** STUDENT: 64210382
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE      : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB              : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD              : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```



```

constant    ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant    ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant    ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant    ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant    ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant    ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,      -- A bus register number select input
BA => BA,      -- B bus register number select input
MB => MB,      -- constant/operand bus B bus multiplexer control signal
MD => MD,      -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F          operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode =>      ALU_mode,      -- mode of alu operation ( M in upper table )
ALU_function =>  ALU_function, -- function of ALU ( F in upper table )
ALU_N_bit  =>      ALU_N_bit,   -- Negative bit of alu operation
ALU_C_bit  =>      ALU_C_bit,   -- Carry bit of alu operation
ALU_V_bit  =>      ALU_V_bit,   -- Overflow bit of alu operation
ALU_Z_bit  =>      ALU_Z_bit,   -- Zero bit of alu operation
Const_in   =>      Const_in,    -- constant operand input bus
Data_in     =>      Data_in,     -- data input bus input
Address_out =>      Address_out, -- address bus output
Data_out    =>      Data_out     -- data bus output

);

PROCESS      -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk  <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk  <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(

```

```

        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 );      -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit          : std_logic; -- Carry bit of alu operation
        ALU_V_bit          : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit          : std_logic; -- Zero bit of alu operation
        Const_in           : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in            : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out        : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out           : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment            : string
    ) IS

```

```

    VARIABLE RES_LINE : LINE;

```

```

    BEGIN

```

```

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

```

```

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );

```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```

```

        writeline( RESULTS, RES_LINE );
    END;

    variable HDR_line : LINE;

    BEGIN
        write( HDR_line, string'( "clk" & HT & "ctrl_word" & HT & "RW" & HT & "DA" & HT & "AA" & HT & "BA"
    & HT & "MB" & HT & "MD" & HT & "ALU_mode ALU_function" & HT & "N" & HT & "C" & HT & "V" & HT & "Z" & HT & "Const_in" &
    HT & "Data_in" & HT & "Address_out" & HT & "Data_out" & HT & "Comment" ) );
        report HDR_line.all;-- send to report
        writeline( RESULTS, HDR_line );

        Data_in      <= std_logic_vector( to_unsigned( 11, Data_in'length ) ); -- io data in is 11
        Const_in     <= std_logic_vector( to_unsigned( 1, Const_in'length ) ); -- constant operand is 1

        WAIT FOR ( PERIOD );
        nRST  <= '1';      -- reset off
        WAIT FOR ( PERIOD );

        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
        MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,      ALU_V_bit,ALU_Z_bit,Const_in,
        Data_in,Address_out,Data_out, "RESET" );

        --      operation code bit positions
        --      +--  --  --  --  --  --  --  --  --  --  --  --  -- - Destination
        register address
        --      /      +--  --  --  --  --  --  --  --  --  --  --  --  -- A bus
        register address
        --      /      /      +--  --  --  --  --  --  --  --  --  -- B bus register address
        --      /      /      +--  --  --  --  --  --  --  --  --  -- constant/B bus operand
        multiplexer
        --      /      /      /      /      ( constant = '1' )
        --      /      /      /      /      +--  --  --  --  --  --
        ALU function select
        --      /      /      /      /      /      +--  --  --  --
        external data input/ ALU result multiplexer
        --      /      /      /      /      /      /      ( ALU result
        multiplexer = '0' )
    
```

```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA      AA          BA      MB          FS          MD      RW          /

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word      <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in          <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word      <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word      <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in          <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word      <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word    <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word    <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word    <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in     <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word    <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in     <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word    <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word    <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word    <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--
--      +--- -- -- -- -- -- -- -- -- -- -- -- -- -- Destination
register address
--      |
--      |      +--- -- -- -- -- -- -- -- -- -- -- A bus
register address
--      |      |
--      |      |      +--- -- -- -- -- -- -- -- -- B bus register address
--      |      |      |      +--- -- -- -- -- -- -- -- constant/B bus operand
multiplexer
--      |      |      |      |      |      |      |      |      |      |      |      |
--      |      |      |      |      |      |      |      |      |      |      |      |
--      |      |      |      |      |      |      |      |      |      |      |      |
ALU function select
--      |      |      |      |      |      |      |      |      |      |      |      |
--      |      |      |      |      |      |      |      |      |      |      |      |
external data input/ ALU result multiplexer
--      |      |      |      |      |      |      |      |      |      |      |      |
multiplexer = '0' )
--      |      |      |      |      |      |      |      |      |      |      |      |
--      |      |      |      |      |      |      |      |      |      |      |      |
register result write ( active = '1' )
--      |      |      |      |      |      |      |      |      |      |      |      |
--      DA      AA      BA      MB      FS      MD      RW
--
-- DEC R( 7 )
        Const_in    <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word    <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```



```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** STUDENT: 64210384
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE      : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB              : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD              : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```

```

constant    ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant    ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant    ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant    ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant    ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant    ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal
-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,    -- A bus register number select input
BA => BA,    -- B bus register number select input
MB => MB,    -- constant/operand bus B bus multiplexer control signal
MD => MD,    -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode => ALU_mode,    -- mode of alu operation ( M in upper table )
ALU_function => ALU_function, -- function of ALU ( F in upper table )
ALU_N_bit => ALU_N_bit,  -- Negative bit of alu operation
ALU_C_bit => ALU_C_bit,  -- Carry bit of alu operation
ALU_V_bit => ALU_V_bit,  -- Overflow bit of alu operation
ALU_Z_bit => ALU_Z_bit,  -- Zero bit of alu operation
Const_in => Const_in,    -- constant operand input bus
Data_in => Data_in,      -- data input bus input
Address_out => Address_out, -- address bus output
Data_out => Data_out      -- data bus output

);

PROCESS    -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(

```

```

        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 );      -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit          : std_logic; -- Carry bit of alu operation
        ALU_V_bit          : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit          : std_logic; -- Zero bit of alu operation
        Const_in           : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in            : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out        : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out           : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment            : string
    ) IS

```

```

    VARIABLE RES_LINE : LINE;

```

```

    BEGIN

```

```

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

```

```

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );

```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA      AA          BA      MB          FS          MD      RW          /

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word      <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in          <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word      <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word      <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in          <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word      <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word    <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word    <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word    <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in     <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word    <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in     <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word    <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word    <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word  <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--      +--  --  --  --  --  --  --  --  --  --  --  --  --  --  -- Destination
register address
--      |      +--  --  --  --  --  --  --  --  --  --  --  --  --  -- A bus
register address
--      |      |      +--  --  --  --  --  --  --  --  --  --  -- B bus register address
--      |      |      |      +--  --  --  --  --  --  --  --  --  --  -- constant/B bus operand
multiplexer
--      |      |      |      |      |      |      |      |      |      |      |      |      |
--      |      |      |      |      |      |      |      |      |      |      |      |      |
ALU function select
--      |      |      |      |      |      |      |      |      |      |      |      |      |
external data input/ ALU result multiplexer
--      |      |      |      |      |      |      |      |      |      |      |      |      |
multiplexer = '0' )
--      |      |      |      |      |      |      |      |      |      |      |      |      |
register result write ( active = '1' )
--      |      |      |      |      |      |      |      |      |      |      |      |      |
--      DA      AA      BA      MB      FS      MD      RW

-- DEC R( 7 )
        Const_in  <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word  <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```

```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** STUDENT: 64210386
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE     : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB              : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD              : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```



```

constant  ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant  ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant  ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant  ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant  ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant  ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal
-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,      -- A bus register number select input
BA => BA,      -- B bus register number select input
MB => MB,      -- constant/operand bus B bus multiplexer control signal
MD => MD,      -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode =>      ALU_mode,      -- mode of alu operation ( M in upper table )
ALU_function =>  ALU_function,-- function of ALU ( F in upper table )
ALU_N_bit  =>      ALU_N_bit,  -- Negative bit of alu operation
ALU_C_bit  =>      ALU_C_bit,  -- Carry bit of alu operation
ALU_V_bit  =>      ALU_V_bit,  -- Overflow bit of alu operation
ALU_Z_bit  =>      ALU_Z_bit,  -- Zero bit of alu operation
Const_in   =>      Const_in,   -- constant operand input bus
Data_in     =>      Data_in,    -- data input bus input
Address_out =>      Address_out, -- address bus output
Data_out    =>      Data_out    -- data bus output

);

PROCESS      -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk  <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk  <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(

```

```

        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 );      -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit          : std_logic; -- Carry bit of alu operation
        ALU_V_bit          : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit          : std_logic; -- Zero bit of alu operation
        Const_in           : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in            : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out        : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out           : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment            : string
    ) IS

```

```

    VARIABLE RES_LINE : LINE;

```

```

    BEGIN

```

```

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

```

```

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );

```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA      AA          BA      MB          FS          MD      RW

--      --      --      --      --      --      --      --      --      --      --      --
--      --      --      --      --      --      --      --      --      --      --      --
-- LOAD          R( 0 )      <= MEM( R( 0 ) )      -- data_in will be loaded, since there's no external ram

      ctrl_word      <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
      WAIT FOR ( PERIOD );
      Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )      <= MEM( R( 0 ) )" );

      Data_in      <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )      <= MEM( R( 0 ) )      -- data_in will be loaded, because there's no external ram

      ctrl_word      <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1';      -- write Data_in to R4 register
      WAIT FOR ( PERIOD );
      Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )      <= MEM( R( 0 ) )" );

-- ADD          R( 5 )      <= R( 0 ) + R( 4 )
      ctrl_word      <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1';      -- write Data_in to R4 register
      WAIT FOR ( PERIOD );
      Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )      <= R( 0 ) + R( 4 )" );

      Const_in      <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )      <= R( 4 ) + Const_in
      ctrl_word      <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1';      -- write Data_in to R4 register
      WAIT FOR ( PERIOD );
      Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )      <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word   <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word   <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word   <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in    <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word   <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in    <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word   <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word   <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word    <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--      +--- -- -- -- -- -- -- -- -- -- -- -- -- -- Destination
register address
--      |          +--- -- -- -- -- -- -- -- -- -- -- -- A bus
register address
--      |      |          +--- -- -- -- -- -- -- -- -- -- B bus register address
--      |          |          +--- -- -- -- -- -- -- -- -- constant/B bus operand
multiplexer
--      |          |          |          |          |          |          |          |          |          |
--      |          |          |          |          |          |          |          |          |          |
ALU function select
--      |          |          |          |          |          |          |          |          |          |
external data input/ ALU result multiplexer
--      |          |          |          |          |          |          |          |          |          |
multiplexer = '0' )
--      |          |          |          |          |          |          |          |          |          |
register result write ( active = '1' )
--      |          |          |          |          |          |          |          |          |          |
--      DA      AA          BA      MB          FS          MD      RW

-- DEC R( 7 )
        Const_in    <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word    <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```



```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** STUDENT: 64210445
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE      : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB              : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD              : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```

```

constant    ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant    ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant    ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant    ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant    ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant    ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal
-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,      -- A bus register number select input
BA => BA,      -- B bus register number select input
MB => MB,      -- constant/operand bus B bus multiplexer control signal
MD => MD,      -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F          operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode =>      ALU_mode,      -- mode of alu operation ( M in upper table )
ALU_function =>  ALU_function,-- function of ALU ( F in upper table )
ALU_N_bit  =>      ALU_N_bit,  -- Negative bit of alu operation
ALU_C_bit  =>      ALU_C_bit,  -- Carry bit of alu operation
ALU_V_bit  =>      ALU_V_bit,  -- Overflow bit of alu operation
ALU_Z_bit  =>      ALU_Z_bit,  -- Zero bit of alu operation
Const_in   =>      Const_in,   -- constant operand input bus
Data_in     =>      Data_in,    -- data input bus input
Address_out =>      Address_out, -- address bus output
Data_out    =>      Data_out    -- data bus output

);

PROCESS      -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk  <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk  <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
    END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(

```

```

        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 );      -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit          : std_logic; -- Carry bit of alu operation
        ALU_V_bit          : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit          : std_logic; -- Zero bit of alu operation
        Const_in           : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in            : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out        : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out           : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment            : string
    ) IS

```

```

    VARIABLE RES_LINE : LINE;

```

```

    BEGIN

```

```

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

```

```

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );

```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA    AA          BA    MB          FS          MD    RW

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word    <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in    <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word    <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word    <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in    <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word    <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word    <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word    <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word    <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in     <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word    <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in     <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word    <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word    <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word  <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--
--      +---  --  --  --  --  --  --  --  --  --  --  --  --  --  -- Destination
register address
--      |
--      |      +---  --  --  --  --  --  --  --  --  --  --  --  --  -- A bus
register address
--      |      |
--      |      |      +---  --  --  --  --  --  --  --  --  --  -- B bus register address
--      |      |      |      +---  --  --  --  --  --  --  --  --  --  -- constant/B bus operand
multiplexer
--      |      |      |      |      |
--      |      |      |      |      |      ( constant = '1' )
--      |      |      |      |      |      +---  --  --  --  --  --
ALU function select
--      |      |      |      |      |      |      +---  --  --  --  --
external data input/ ALU result multiplexer
--      |      |      |      |      |      |      |      ( ALU result
multiplexer = '0' )
--      |      |      |      |      |      |      |      +---  --  --  --  -- destination
register result write ( active = '1' )
--      |      |      |      |      |      |      |
--      DA      AA      BA      MB      FS      MD      RW

-- DEC R( 7 )
        Const_in  <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word  <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```

```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** STUDENT: 64210455
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE      : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB              : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD              : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```



```

constant    ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant    ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant    ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant    ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant    ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant    ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal
-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,      -- A bus register number select input
BA => BA,      -- B bus register number select input
MB => MB,      -- constant/operand bus B bus multiplexer control signal
MD => MD,      -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F          operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode =>      ALU_mode,      -- mode of alu operation ( M in upper table )
ALU_function =>  ALU_function, -- function of ALU ( F in upper table )
ALU_N_bit  =>      ALU_N_bit,   -- Negative bit of alu operation
ALU_C_bit  =>      ALU_C_bit,   -- Carry bit of alu operation
ALU_V_bit  =>      ALU_V_bit,   -- Overflow bit of alu operation
ALU_Z_bit  =>      ALU_Z_bit,   -- Zero bit of alu operation
Const_in   =>      Const_in,    -- constant operand input bus
Data_in     =>      Data_in,     -- data input bus input
Address_out =>      Address_out, -- address bus output
Data_out    =>      Data_out     -- data bus output

);

PROCESS      -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk  <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk  <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(
        clk                : std_logic;
        ctrl_word          : std_logic_vector( reg_width - 1 downto 0 );
        RW                 : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit          : std_logic; -- Carry bit of alu operation
        ALU_V_bit          : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit          : std_logic; -- Zero bit of alu operation
        Const_in           : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in            : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out        : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out           : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment            : string
    ) IS
    VARIABLE RES_LINE : LINE;
    BEGIN

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );
    
```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA    AA          BA    MB          FS          MD    RW

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word    <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in    <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word    <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word    <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in    <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word    <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word    <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word    <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word    <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in     <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word    <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in     <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word    <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word    <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word    <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--      +--- -- -- -- -- -- -- -- -- -- -- -- -- -- Destination
register address
--      |          +--- -- -- -- -- -- -- -- -- -- -- -- A bus
register address
--      |      |          +--- -- -- -- -- -- -- -- -- -- B bus register address
--      |          |          +--- -- -- -- -- -- -- -- -- constant/B bus operand
multiplexer
--      |          |          |          |          |          |          |          |          |          |
--      |          |          |          |          |          |          |          |          |          |
ALU function select
--      |          |          |          |          |          |          |          |          |          |
external data input/ ALU result multiplexer
--      |          |          |          |          |          |          |          |          |          |
multiplexer = '0' )
--      |          |          |          |          |          |          |          |          |          |
register result write ( active = '1' )
--      |          |          |          |          |          |          |          |          |          |
--      DA      AA          BA      MB          FS          MD      RW

-- DEC R( 7 )
        Const_in    <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word    <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```



```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** STUDENT: 64210457
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE      : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB              : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD              : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```

```

constant    ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant    ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant    ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant    ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant    ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant    ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal
-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,    -- A bus register number select input
BA => BA,    -- B bus register number select input
MB => MB,    -- constant/operand bus B bus multiplexer control signal
MD => MD,    -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

    ALU_mode =>      ALU_mode,    -- mode of alu operation ( M in upper table )
    ALU_function =>  ALU_function,-- function of ALU ( F in upper table )
    ALU_N_bit  =>    ALU_N_bit,   -- Negative bit of alu operation
    ALU_C_bit  =>    ALU_C_bit,   -- Carry bit of alu operation
    ALU_V_bit  =>    ALU_V_bit,   -- Overflow bit of alu operation
    ALU_Z_bit  =>    ALU_Z_bit,   -- Zero bit of alu operation
    Const_in   =>     Const_in,   -- constant operand input bus
    Data_in    =>     Data_in,    -- data input bus input
    Address_out =>    Address_out, -- address bus output
    Data_out   =>     Data_out    -- data bus output
);

PROCESS    -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk  <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk  <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(

```

```

        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 );      -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit          : std_logic; -- Carry bit of alu operation
        ALU_V_bit          : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit          : std_logic; -- Zero bit of alu operation
        Const_in           : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in            : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out        : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out           : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment            : string
    ) IS

```

```

    VARIABLE RES_LINE : LINE;

```

```

    BEGIN

```

```

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

```

```

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );

```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA    AA          BA    MB          FS          MD    RW

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word    <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in    <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word    <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word    <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in    <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word    <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word    <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word    <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word    <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in     <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word    <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in     <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word    <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word    <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word    <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--
--      +--  --  --  --  --  --  --  --  --  --  --  --  --  -- Destination
register address
--      |
--      +--  --  --  --  --  --  --  --  --  --  --  --  -- A bus
register address
--      |  |
--      |  +--  --  --  --  --  --  --  --  --  --  --  --  -- B bus register address
--      |  |
multiplexer      |  |
--      |  |
--      |  |
--      |  |
ALU function select
--      |  |
external data input/ ALU result multiplexer
--      |  |
multiplexer = '0' )
--      |  |
--      |  |
register result write ( active = '1' )
--      |  |
--      DA  AA  BA  MB  FS  MD  RW

-- DEC R( 7 )
        Const_in    <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word    <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```

```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** STUDENT: 64240430
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE      : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB              : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD              : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal    Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal    Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal    Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal    Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant  R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant  R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant  R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant  R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant  R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant  R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant  R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant  R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant  ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant  ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant  ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant  ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant  ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant  ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant  ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant  ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant  ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```



```

constant  ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant  ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant  ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant  ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant  ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant  ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal
-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,      -- A bus register number select input
BA => BA,      -- B bus register number select input
MB => MB,      -- constant/operand bus B bus multiplexer control signal
MD => MD,      -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F          operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode =>      ALU_mode,      -- mode of alu operation ( M in upper table )
ALU_function =>  ALU_function,-- function of ALU ( F in upper table )
ALU_N_bit  =>      ALU_N_bit,   -- Negative bit of alu operation
ALU_C_bit  =>      ALU_C_bit,   -- Carry bit of alu operation
ALU_V_bit  =>      ALU_V_bit,   -- Overflow bit of alu operation
ALU_Z_bit  =>      ALU_Z_bit,   -- Zero bit of alu operation
Const_in   =>      Const_in,    -- constant operand input bus
Data_in    =>      Data_in,     -- data input bus input
Address_out =>      Address_out, -- address bus output
Data_out   =>      Data_out     -- data bus output

);

PROCESS      -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk  <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk  <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(
        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,    -- destination register number select input
        AA,    -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit          : std_logic; -- Carry bit of alu operation
        ALU_V_bit          : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit          : std_logic; -- Zero bit of alu operation
        Const_in           : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in            : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out        : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out           : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment            : string
    ) IS
    VARIABLE RES_LINE : LINE;
    BEGIN

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );
    
```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA    AA          BA    MB          FS          MD    RW

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word    <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in    <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word    <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word    <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in    <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word    <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word    <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word    <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word    <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in     <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word    <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in     <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word    <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word    <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word  <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--
--      +--  --  --  --  --  --  --  --  --  --  --  --  --  -- Destination
register address
--      |
--      +--  --  --  --  --  --  --  --  --  --  --  --  -- A bus
register address
--      |  |
--      |  +--  --  --  --  --  --  --  --  --  --  --  --  -- B bus register address
--      |  |
multiplexer      |  |
--      |  |
--      |  |
--      |  |
ALU function select
--      |  |
external data input/ ALU result multiplexer
--      |  |
multiplexer = '0' )
--      |  |
--      |  |
register result write ( active = '1' )
--      |  |
--      DA  AA  BA  MB  FS  MD  RW

-- DEC R( 7 )
        Const_in  <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word  <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```



```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

```

-- *****
-- **** PREDLOGA VAJE
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE std.textio.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY cpu_datapath_tb IS
    GENERIC (
        nr_regs          : natural := 8;
        reg_width        : natural := 16
    );
END cpu_datapath_tb;

ARCHITECTURE ideal OF cpu_datapath_tb IS
    SIGNAL nRST : std_logic := '0'; -- reset input      ( active '0' )
    SIGNAL clk   : STD_LOGIC := '0';

    FILE RESULTS: TEXT OPEN WRITE_MODE IS "cpu_datapath.csv";
    -- operation code bit positions
    -- bit#
    -- 15      DA( 2 )      : destination register address
    -- 14      DA( 1 )
    -- 13      DA( 0 )
    -- 12      AA( 2 )      : A bus register address
    -- 11      AA( 1 )
    -- 10      AA( 0 )
    -- 9       BA( 2 )      : B bus register address
    -- 8       BA( 1 )
    -- 7       BA( 0 )
    -- 6       MB           : constant/B bus operand multiplexer
    -- 5       FS( 3 )      : ALU function select

```

```

-- 4      FS( 2 )
-- 3      FS( 1 )
-- 2      FS( 0 )
-- 1      MD      : external data input/ ALU result multiplexer
-- 0      RW      : destination register result write ( active = '1' )
signal    ctrl_word      : std_logic_vector( reg_width - 1 downto 0 ); -- ctrl_word

constant  PERIOD          : time := 400 ns;
constant  DUTY_CYCLE     : real := 0.5;
constant  OFFSET          : time := 100 ns;
signal    RW              : std_logic; -- register write input ( active '1' )
signal    DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
signal    MB              : std_logic; -- constant/operand bus B bus multiplexer control signal
signal    MD              : std_logic; -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( dvojiški komplement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
signal    ALU_mode        : std_logic; -- mode of alu operation ( M in upper table )
signal    ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )

signal    ALU_N_bit      : std_logic; -- Negative bit of alu operation
signal    ALU_C_bit      : std_logic; -- Carry bit of alu operation
signal    ALU_V_bit      : std_logic; -- Overflow bit of alu operation
signal    ALU_Z_bit      : std_logic; -- Zero bit of alu operation

```

```

signal      Const_in          :      std_logic_vector( reg_width - 1 downto 0 );  -- constant input bus
signal      Data_in           :      std_logic_vector( reg_width - 1 downto 0 );  -- data input bus input
signal      Address_out       :      std_logic_vector( reg_width - 1 downto 0 );  -- address bus output
signal      Data_out          :      std_logic_vector( reg_width - 1 downto 0 );  -- data bus output

constant    R0                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "000";
constant    R1                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "001";
constant    R2                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "010";
constant    R3                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "011";
constant    R4                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "100";
constant    R5                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "101";
constant    R6                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "110";
constant    R7                :      std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 ) := "111";

-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
constant    ALU_X_PLUS_Y :      std_logic_vector( 3 downto 0 ) := "0000";
constant    ALU_X_MINUS_Y :      std_logic_vector( 3 downto 0 ) := "0001";
constant    ALU_X_PLUS_1 :      std_logic_vector( 3 downto 0 ) := "0010";
constant    ALU_X_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0011";
constant    ALU_X_PLUS_X :      std_logic_vector( 3 downto 0 ) := "0100";
constant    ALU_MINUS_1 :      std_logic_vector( 3 downto 0 ) := "0101";
constant    ALU_NOP      :      std_logic_vector( 3 downto 0 ) := "0110";

constant    ALU_X_AND_Y :      std_logic_vector( 3 downto 0 ) := "1000";
constant    ALU_X_NAND_Y :      std_logic_vector( 3 downto 0 ) := "1001";

```

```

constant    ALU_X_OR_Y      :      std_logic_vector( 3 downto 0 ) := "1010";
constant    ALU_X_NOR_Y    :      std_logic_vector( 3 downto 0 ) := "1011";
constant    ALU_X_XOR_Y    :      std_logic_vector( 3 downto 0 ) := "1100";
constant    ALU_X_XNOR_Y   :      std_logic_vector( 3 downto 0 ) := "1101";
constant    ALU_X          :      std_logic_vector( 3 downto 0 ) := "1110";
constant    ALU_Y          :      std_logic_vector( 3 downto 0 ) := "1111";

```

BEGIN

```

RW    <= ctrl_word( 0 ); -- register write input  ( active '1' )
DA    <= ctrl_word( 15 downto 13 ); -- destination register number select input
AA    <= ctrl_word( 12 downto 10 ); -- A bus register number select input
BA    <= ctrl_word( 9 downto 7 ); -- B bus register number select input
MB    <= ctrl_word( 6 ); -- constant/operand bus B bus multiplexer control signal
MD    <= ctrl_word( 1 ); -- external data/alu result multiplexer control signal
-- ALU operation summary:
-- M F      operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y
ALU_mode    <= ctrl_word( 5 ); -- mode of alu operation ( M in upper table )
ALU_function <= ctrl_word( 4 downto 2 ); -- function of ALU ( F in upper table )

UUT: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
PORT map(
    clk => clk, -- clock input
    RW  => RW, -- register write input  ( active '1' )
    nRST => nRST, -- reset input      ( active '0' )
    DA => DA, -- destination register number select input

```

```

AA => AA,      -- A bus register number select input
BA => BA,      -- B bus register number select input
MB => MB,      -- constant/operand bus B bus multiplexer control signal
MD => MD,      -- external data/alu result multiplexer control signal

-- ALU operation summary:
-- M F          operation
-- 0 0 0 0 S = X plus Y
-- 0 0 0 1 S = X minus Y
-- 0 0 1 0 S = X plus 1
-- 0 0 1 1 S = X minus 1
-- 0 1 0 0 S = X plus X
-- 0 1 0 1 S = minus 1 ( 2' complement )
-- 1 0 0 0 S = X and Y
-- 1 0 0 1 S = X nand Y
-- 1 0 1 0 S = X or Y
-- 1 0 1 1 S = X nor Y
-- 1 1 0 0 S = X xor Y
-- 1 1 0 1 S = X xnor Y
-- 1 1 1 0 S = X
-- 1 1 1 1 S = Y

ALU_mode =>      ALU_mode,      -- mode of alu operation ( M in upper table )
ALU_function =>  ALU_function, -- function of ALU ( F in upper table )
ALU_N_bit  =>      ALU_N_bit,   -- Negative bit of alu operation
ALU_C_bit  =>      ALU_C_bit,   -- Carry bit of alu operation
ALU_V_bit  =>      ALU_V_bit,   -- Overflow bit of alu operation
ALU_Z_bit  =>      ALU_Z_bit,   -- Zero bit of alu operation
Const_in   =>      Const_in,    -- constant operand input bus
Data_in     =>      Data_in,     -- data input bus input
Address_out =>      Address_out, -- address bus output
Data_out    =>      Data_out     -- data bus output

);

PROCESS      -- clock process for clk
BEGIN
    WAIT for OFFSET;
    CLOCK_LOOP : LOOP
        clk  <= '0';
        WAIT FOR ( PERIOD - ( PERIOD * DUTY_CYCLE ) );
        clk  <= '1';
        WAIT FOR ( PERIOD * DUTY_CYCLE );
    END LOOP;
END PROCESS;

```

```

        END LOOP CLOCK_LOOP;
END PROCESS;

```

```

PROCESS

```

```

    PROCEDURE Log_variables(

```

```

        clk                : std_logic;
        ctrl_word           : std_logic_vector( reg_width - 1 downto 0 );
        RW                  : std_logic; -- register write input  ( active '1' )
        DA,  -- destination register number select input
        AA,  -- A, B bus register number select input
        BA                : std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        MB                : std_logic; -- constant/operand bus B bus multiplexer control signal
        MD                : std_logic; -- external data/alu result multiplexer control signal
        ALU_mode           : std_logic; -- mode of alu operation ( M in upper table )
        ALU_function : std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )
        ALU_N_bit          : std_logic; -- Negative bit of alu operation
        ALU_C_bit          : std_logic; -- Carry bit of alu operation
        ALU_V_bit          : std_logic; -- Overflow bit of alu operation
        ALU_Z_bit          : std_logic; -- Zero bit of alu operation
        Const_in           : std_logic_vector( reg_width - 1 downto 0 ); -- constant input bus
        Data_in            : std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
        Address_out        : std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
        Data_out           : std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
        comment            : string
    ) IS

```

```

    VARIABLE RES_LINE : LINE;

```

```

    BEGIN

```

```

        write( RES_LINE, clk, right, 1 );
        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, ctrl_word, right, sizeof( 2**reg_width )/4 );

```

```

        write( RES_LINE, HT );
        write( RES_LINE, RW, right, 1 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & DA, right, sizeof( DA'length )/4 );

```

```

        write( RES_LINE, HT & "0x" );
        hwrite( RES_LINE, "0" & AA, right, sizeof( AA'length )/4 );

```

```

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, "0" & BA, right, sizeof( BA'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, MB, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, MD, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, ALU_mode & ALU_function, right, sizeof( 1 + ALU_function'length )/4 );
write( RES_LINE, HT );

write( RES_LINE, ALU_N_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_C_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_V_bit, right, 1 );
write( RES_LINE, HT );

write( RES_LINE, ALU_Z_bit, right, 1 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Const_in, right, sizeof( 2**reg_width )/4 );
write( RES_LINE, HT & "0x" );

hwrite( RES_LINE, Data_in, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Address_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & "0x" );
hwrite( RES_LINE, Data_out, right, sizeof( 2**reg_width )/4 );

write( RES_LINE, HT & comment );

report      RES_LINE.all; -- send to report

```



```

--          /          /          /          /          /          +--- -- destination
register result write ( active = '1' )
--          /          /          /          /          /          /
--          DA      AA          BA      MB          FS          MD      RW          /

--          --          --          --          --          --          --          --          --          --
--          --          --          --          --          --          --          --          --          --
-- LOAD          R( 0 )          <= MEM( R( 0 ) )          -- data_in will be loaded, since there's no external ram

          ctrl_word      <= R0 & R0 & R0 & '0' & ALU_NOP & '1' & '1';
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 0 )          <= MEM( R( 0 ) )" );

          Data_in          <= std_logic_vector( to_unsigned( 22, Data_in'length ) ); -- io data in is 2

-- LOAD          R( 4 )          <= MEM( R( 0 ) )          -- data_in will be loaded, because there's no external ram

          ctrl_word      <= R4 & R0 & R0 & '1' & ALU_NOP & '1' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "LOAD R( 4 )          <= MEM( R( 0 ) )" );

-- ADD          R( 5 )          <= R( 0 ) + R( 4 )
          ctrl_word      <= R5 & R0 & R4 & '0' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADD R( 5 )          <= R( 0 ) + R( 4 )" );

          Const_in          <= std_logic_vector( to_unsigned( 27, Const_in'length ) ); -- constant operand is 27
-- ADI          R( 6 )          <= R( 4 ) + Const_in
          ctrl_word      <= R6 & R4 & R0 & '1' & ALU_X_PLUS_Y & '0' & '1'; -- write Data_in to R4 register
          WAIT FOR ( PERIOD );
          Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "ADI R( 6 )          <= R( 4 ) + Const_in" );

```

```

-- MOVE A      R( 7 )      <= R( 4 )
               ctrl_word    <= R7 & R4 & R0 & '0' & ALU_X & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE A R( 7 ) <= R( 4 )" );

-- MOVE B      R( 3 )      <= R( 7 )
               ctrl_word    <= R3 & R0 & R7 & '0' & ALU_Y & '0' & '1';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "MOVE B R( 3 ) <= R( 7 )" );

-- STORE B     MEM( R( 3 ) ) <= R( 7 )
               ctrl_word    <= R0 & R3 & R7 & '0' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE B MEM( R( 3 ) ) <= R( 7 )" );

-- STORE CONST MEM( R( 3 ) ) <= Const_in
               Const_in     <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 27
               ctrl_word    <= R0 & R3 & R7 & '1' & ALU_NOP & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "STORE CONST MEM( R( 3 ) ) <= Const_in" );

-- TEST A      set NCVZ bits according to value in R( 1 )
               Const_in     <= std_logic_vector( to_unsigned( 0, Const_in'length ) ); -- constant operand is 0
               ctrl_word    <= R0 & R1 & R0 & '1' & ALU_X_PLUS_Y & '0' & '0';
               WAIT FOR ( PERIOD );
               Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using ADD immediate Y=A + 0 alu command"
);

-- XOR         R( 6 ) <= R( 4 ) xor R( 4 )
               ctrl_word    <= R6 & R4 & R4 & '0' & ALU_X_XOR_Y & '0' & '1';

```

```

        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 6 )  <= R( 4 ) xor R( 4 )" );

-- TEST R( 1 )
        ctrl_word  <= R6 & R1 & R1 & '1' & ALU_X & '0'& '0';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "TEST R( 1 ) using bus A: set NCVZ bits using Y=X alu command" );

--      operation code bit positions
--      +--- -- -- -- -- -- -- -- -- -- -- -- -- -- Destination
register address
--      |          +--- -- -- -- -- -- -- -- -- -- -- -- A bus
register address
--      |      |          +--- -- -- -- -- -- -- -- -- -- B bus register address
--      |          |          +--- -- -- -- -- -- -- -- -- constant/B bus operand
multiplexer
--      |          |          |          |          |          |          |          |          |          |
--      |          |          |          |          |          |          |          |          |          |
ALU function select
--      |          |          |          |          |          |          |          |          |          |
external data input/ ALU result multiplexer
--      |          |          |          |          |          |          |          |          |          |
multiplexer = '0' )
--      |          |          |          |          |          |          |          |          |          |
register result write ( active = '1' )
--      |          |          |          |          |          |          |          |          |          |
--      DA      AA          BA      MB          FS          MD      RW

-- DEC R( 7 )
        Const_in    <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
        ctrl_word  <= R7 & R7 & R7 & '1' & ALU_X_MINUS_1 & '0'& '1';
        WAIT FOR ( PERIOD );
        Log_variables(      clk, ctrl_word,RW,DA,AA,BA,MB,
MD,ALU_mode,ALU_function,ALU_N_bit,ALU_C_bit,ALU_V_bit,ALU_Z_bit,Const_in,
Data_in,Address_out,Data_out, "R( 7 )  <= R( 7 ) - 1" );

```

```

-- INC R( 6 )
    Const_in      <= std_logic_vector( to_unsigned( 95, Const_in'length ) ); -- constant operand is 95
    ctrl_word     <= R6 & R6 & R7 & '1' & ALU_X_PLUS_1 & '0' & '1';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "R( 6 ) <= R( 6 ) + 1" );

    ctrl_word     <= R7 & R7 & R7 & '1' & ALU_NOP & '0' & '0';
    WAIT FOR ( PERIOD );
    Log_variables( clk, ctrl_word, RW, DA, AA, BA, MB,
MD, ALU_mode, ALU_function, ALU_N_bit, ALU_C_bit, ALU_V_bit, ALU_Z_bit, Const_in,
Data_in, Address_out, Data_out, "SET ALU NOP, DA=R( 7 ), AA=R( 7 ), BA=R( 7 ), NO WRITE" );

    WAIT;

END PROCESS;

END ideal;

```

