

-- **** STUDENT: 64000225.....	3
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni rezultatov simulacije zaradi napak sintetizatorja:.....	3
ERROR:HDLCompiler:806 - "cpu_datapath.vhd" Line 128: Syntax error near "end".....	3
Zaključna vrstica "end ideal" je podvojena. Če eno odstranim, koda deluje pravilno.	3
-- **** STUDENT: 64200100.....	7
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni rezultatov simulacije zaradi napak sintetizatorja:.....	7
ERROR:HDLCompiler:1314 - "cpu_datapath.vhd" Line 78: Formal port/generic <desS> is not declared in <reg_file>	7
ERROR:HDLCompiler:432 - "cpu_datapath.vhd" Line 75: Formal <dest_select> has no actual or default value.	7
Pri povezovanju komponente REG_FILE niste pravilno dali imena izbire cilja (destination select). Namesto (dest_select) imate (desS). Ko to popravim, koda deluje pravilno.	7
-- **** STUDENT: 64200112.....	11
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	11
-- **** STUDENT: 64200238.....	15
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	15
-- **** STUDENT: 64200288.....	19
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	19
-- **** STUDENT: 64200296.....	23
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	23
-- **** STUDENT: 64200385.....	26
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	26
-- **** STUDENT: 64210113.....	30
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	30
-- **** STUDENT: 64210290.....	33
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	33
-- **** STUDENT: 64210382.....	38
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	38
-- **** STUDENT: 64210384.....	41
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	41
-- **** STUDENT: 64210386.....	45
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	45
-- **** STUDENT: 64210445.....	49
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	49
-- **** STUDENT: 64210455.....	52
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni rezultatov simulacije zaradi napak sintetizatorja:.....	52

ERROR:HDLCompiler:374 - "cpu_datapath.vhd" Line 1: Entity <cpu_datapath> is not yet compiled.	52
ERROR:HDLCompiler:24 - "cpu_datapath.vhd" Line 2: "***" expects 2 arguments.....	52
ERROR:HDLCompiler:845 - "cpu_datapath.vhd" Line 3: Type of aggregate cannot be determined without context ; 0 visible types match here ...	52
Manjka vam deklaracija entitete. Če dodam deklaracijo, dobim rezultate simulacije, vendar sta Address_out in Data_out vedno nič, ker vam manjkajo for zanke za preslikavo 2d bitnih polj to 1d polje vektorjev std_logic_vector za multiplekserja B (muxb) in D (muxd). Če to dodam, so C,V,Z biti nedefinirani ('U').	52
Manjka povezava komponente polja registrov (REG_FILE) in povezave bitov v MB_vector.....	52
-- **** STUDENT: 64210457.....	54
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	54
-- **** STUDENT: 64240430.....	58
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	58
-- **** PREDLOGA VAJE	62
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	62

```

-- *****
-- **** STUDENT: 64000225
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni rezultatov simulacije zaradi napak sintetizatorja:
ERROR:HDLCompiler:806 - "cpu_datapath.vhd" Line 128: Syntax error near "end".
Zaključna vrstica "end ideal" je podvojena. Če eno odstranim, koda deluje pravilno.
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
use ieee.math_real.all;

entity cpu_datapath is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk,      -- clock input
           RW        : in std_logic;    -- register write input  ( active '1' )
           nRST      : in std_logic;    -- reset input          ( active '0' )
           DA,       -- destination register number select input
           AA,       -- A, B bus register number select input
           BA        : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
           MB        : in std_logic;    -- constant/operand bus B bus multiplexer control signal
           MD        : in 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
    ALU_mode           : in std_logic;      -- mode of alu operation ( M in upper table )
    ALU_function : in std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )
    ALU_N_bit          : out std_logic;      -- Negative bit of alu operation
    ALU_C_bit          : out std_logic;      -- Carry bit of alu operation
    ALU_V_bit          : out std_logic;      -- Overflow bit of alu operation
    ALU_Z_bit          : out std_logic;      -- Zero bit of alu operation
    Const_in           : in  std_logic_vector( reg_width - 1 downto 0 ); -- constant input
bus
    Data_in            : in  std_logic_vector( reg_width - 1 downto 0 ); -- data input bus
input
    Address_out        : out std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
    Data_out           : out std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
    );
end cpu_datapath;

architecture ideal of cpu_datapath is

    component ndn_alu is
        generic( n: natural := 8 );
        port( M : in std_logic;
-- naèin delovanja ( '0' => aritmetièni, '1' => logièni )
            F : in std_logic_vector( 2 downto 0 );
-- funkcijski vhod za operacije
            X, Y : in std_logic_vector( n-1 downto 0 );
            S : out std_logic_vector( n-1 downto 0 );
            Negative, Cout, Overflow, Zero, Gout, Pout :
                out std_logic );
    end component;

-- 2/1 bus multiplexer input type definitions
type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal b_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );

signal b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNTO 0, reg_width - 1 DOWNTO 0 ) := ( others => ( others =>
'0' ) );
signal d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNTO 0, reg_width - 1 DOWNTO 0 ) := ( others => ( others =>
'0' ) );

```

```

signal      A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
signal      ALU_result      : std_logic_vector( reg_width - 1 downto 0 );

signal      MB_vector : std_logic_vector( 0 downto 0 );  -- MB constant/operand bus B bus multiplexer control signal
signal      MD_vector : std_logic_vector( 0 downto 0 );  -- external data/alu result multiplexer control signal

begin

    rfile: reg_file
    generic map ( nr_regs => nr_regs, reg_width => reg_width )
    port map (
        clk => clk, LE => RW, nRST => nRST, dest_select => DA,      A_select => AA, B_select => BA, D => D_data,
        A => A_data, B => B_data
    );
    Address_out  <= A_data;
    Data_out     <= B_muxed;

    muxb: muxnto1_bus
    generic map ( n_addr => 1, bus_width => reg_width )
    port map (
        s => MB_vector, w => b_mux_2d_bit_array, f => B_muxed
    );
    MB_vector( 0 )      <= MB;
    b_mux_vector_array( 0 ) <= B_data;
    b_mux_vector_array( 1 ) <= Const_in;

    muxd: muxnto1_bus
    generic map ( n_addr => 1, bus_width => reg_width )
    port map (
        s => MD_vector, w => d_mux_2d_bit_array, f => D_data
    );
    MD_vector( 0 )      <= MD;
    d_mux_vector_array( 0 ) <= ALU_result;
    d_mux_vector_array( 1 ) <= Data_in;

    arrays_to_2d: for i in 0 to reg_width-1 generate
        b_mux_2d_bit_array( 0, i )<= b_mux_vector_array( 0 )( i );
        b_mux_2d_bit_array( 1, i )<= b_mux_vector_array( 1 )( i );
        d_mux_2d_bit_array( 0, i )<= d_mux_vector_array( 0 )( i );
        d_mux_2d_bit_array( 1, i )<= d_mux_vector_array( 1 )( i );
    end generate

```

```
end generate;

alu: ndn_alu
generic map ( n => reg_width )
port map (
    M => ALU_mode, F => ALU_function,      X => A_data, Y => B_muxed, S => ALU_result,   Negative =>
    ALU_N_bit, Cout => ALU_C_bit,   Overflow => ALU_V_bit, Zero => ALU_Z_bit
);

end ideal;

end ideal;
```

```

-- *****
-- **** STUDENT: 64200100
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni rezultatov simulacije zaradi napak sintetizatorja:
ERROR:HDLCompiler:1314 - "cpu_datapath.vhd" Line 78: Formal port/generic <dess> is not declared in <reg_file>
ERROR:HDLCompiler:432 - "cpu_datapath.vhd" Line 75: Formal <dest_select> has no actual or default value.
Pri povezovanju komponente REG_FILE niste pravilno dali imena izbire cilja (destination select). Namesto (dest_select)
imate (desS). Ko to popravim, koda deluje pravilno.
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
use ieee.math_real.all;

entity cpu_datapath is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk,      -- clock input
           RW        : in std_logic;    -- register write input  ( active '1' )
           nRST      : in std_logic;    -- reset input          ( active '0' )
           DA,       -- destination register number select input
           AA,       -- A, B bus register number select input
           BA        : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
           MB        : in std_logic;    -- constant/operand bus B bus multiplexer control signal
           MD        : in 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
    ALU_mode           : in std_logic;      -- mode of alu operation ( M in upper table )
    ALU_function : in std_logic_vector( 2 downto 0 ); -- function of ALU ( F in upper table )
    ALU_N_bit          : out std_logic;      -- Negative bit of alu operation
    ALU_C_bit           : out std_logic;      -- Carry bit of alu operation
    ALU_V_bit           : out std_logic;      -- Overflow bit of alu operation
    ALU_Z_bit           : out std_logic;      -- Zero bit of alu operation
    Const_in            : in std_logic_vector( reg_width - 1 downto 0 ); -- constant input
bus
    Data_in             : in std_logic_vector( reg_width - 1 downto 0 ); -- data input bus
input
    Address_out         : out std_logic_vector( reg_width - 1 downto 0 ); -- address bus output
    Data_out            : out std_logic_vector( reg_width - 1 downto 0 ); -- data bus output
    );
end cpu_datapath;

architecture ideal of cpu_datapath is
-- 2/1 bus multiplexer input type definitions

component ndn_alu is
generic( n:natural:=8 );
port( M:in std_logic;
F:in std_logic_vector( 2 downto 0 );
X,Y:in std_logic_vector( n-1 downto 0 );
S:out std_logic_vector( n-1 downto 0 );
Negative,Cout,Overflow,Zero,Gout,Pout:out std_logic );
end component;

type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal b_mux_vector_array : mux_vector_array_type:= ( others=>( others => '0' ) );
signal d_mux_vector_array : mux_vector_array_type:= ( others=>( others => '0' ) );

signal b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0,reg_width-1 DOWNT0 0 ):= ( others =>( others =>'0' ) );
signal d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0,reg_width-1 DOWNT0 0 ):= ( others =>( others =>'0' ) );

signal A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
signal ALU_result : std_logic_vector( reg_width - 1 downto 0 );

```



```

signal      MB_vector : std_logic_vector( 0 downto 0 );  -- MB constant/operand bus B bus multiplexer control signal
signal      MD_vector : std_logic_vector( 0 downto 0 );  -- external data/alu result multiplexer control signal

begin
register_file: reg_file
generic map ( nr_regs => nr_regs, reg_width => reg_width )
port map (
clk => clk, LE => RW, nRST => nRST, desS => DA,
A_select => AA, B_select => BA, D => D_data,
A => A_data, B => B_data );
Address_out  <= A_data;
Data_out     <= B_muxed;

muxB: muxnto1_bus
generic map ( n_addr => 1, bus_width => reg_width )
port map ( s => MB_vector, w => b_mux_2d_bit_array, f => B_muxed );
MB_vector( 0 )      <= MB;
b_mux_vector_array( 0 )  <= B_data;
b_mux_vector_array( 1 )  <= Const_in;

muxD: muxnto1_bus
generic map ( n_addr => 1, bus_width => reg_width )
port map ( s => MD_vector, w => d_mux_2d_bit_array, f => D_data );
MD_vector( 0 )      <= MD;
d_mux_vector_array( 0 )  <= ALU_result;
d_mux_vector_array( 1 )  <= Data_in;

arrays_to_2d: for i in 0 to reg_width-1 generate
b_mux_2d_bit_array( 0, i )<= b_mux_vector_array( 0 )( i );
b_mux_2d_bit_array( 1, i )<= b_mux_vector_array( 1 )( i );
d_mux_2d_bit_array( 0, i )<= d_mux_vector_array( 0 )( i );
d_mux_2d_bit_array( 1, i )<= d_mux_vector_array( 1 )( i );
end generate;

alu: ndn_alu
generic map ( n => reg_width )
port map (
M => ALU_mode, F => ALU_function,
X => A_data, Y => B_muxed, S => ALU_result,
Negative => ALU_N_bit, Cout => ALU_C_bit,

```

```
Overflow => ALU_V_bit, Zero => ALU_Z_bit );  
end ideal;
```

```

-- *****
-- **** STUDENT: 64200112
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
use ieee.math_real.all;

entity cpu_datapath is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk,      -- clock input
           RW        : in std_logic;    -- register write input  ( active '1' )
           nRST      : in std_logic;    -- reset input          ( active '0' )
           DA,       -- destination register number select input
           AA,       -- A, B bus register number select input
           BA        : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
           MB        : in std_logic;    -- constant/operand bus B bus multiplexer control signal
           MD        : in 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 ( dvojiki 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

           ALU_mode          : in std_logic;    -- mode of alu operation ( M in upper table )

```

```

        ALU_function : in std_logic_vector( 2 downto 0 );    -- function of ALU ( F in upper table )
        ALU_N_bit    : out std_logic;    -- Negative bit of alu operation
        ALU_C_bit    : out std_logic;    -- Carry bit of alu operation
        ALU_V_bit    : out std_logic;    -- Overflow bit of alu operation
        ALU_Z_bit    : out std_logic;    -- Zero bit of alu operation
        Const_in     : in  std_logic_vector( reg_width - 1 downto 0 );    -- constant input
bus
        Data_in      : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus
input
        Address_out  : out std_logic_vector( reg_width - 1 downto 0 );    -- address bus output
        Data_out     : out std_logic_vector( reg_width - 1 downto 0 )    -- data bus output
    );
end cpu_datapath;

architecture ideal of cpu_datapath is
    -- 2/1 bus multiplexer input type definitions
    type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal b_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
    signal d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );

    signal b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );
    signal d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );

    signal A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
    signal ALU_result : std_logic_vector( reg_width - 1 downto 0 );

    signal MB_vector : std_logic_vector( 0 downto 0 );    -- MB constant/operand bus B bus multiplexer control signal
    signal MD_vector : std_logic_vector( 0 downto 0 );    -- external data/alu result multiplexer control signal

COMPONENT ndn_alu is
    generic( n: natural := 8 );
    port( M      : in  std_logic;
        -- nain delovanja ( '0' => aritmetini, '1' => logini )
        F      : in  std_logic_vector( 2 downto 0 );
        -- funkcijski vhod za operacije
        X, Y   : in  std_logic_vector( n-1 downto 0 );
        S      : out std_logic_vector( n-1 downto 0 );
    );
end COMPONENT ndn_alu is

```

```

        Negative,      Cout,      Overflow,      Zero,      Gout,      Pout      :
    out std_logic );
END COMPONENT;

```

```

begin

```

```

    U0 : muxnto1_bus
        generic map( n_addr => 1,      bus_width => reg_width )
        port map( s => MD_vector,      w => d_mux_2d_bit_array,      f => D_data
        );
    MD_vector( 0 )      <= MD;
    d_mux_vector_array( 0 )      <= ALU_result;
    d_mux_vector_array( 1 )      <= Data_in;

    U1 : reg_file
        generic map( nr_regs => nr_regs,      reg_width => reg_width )
        port map( clk => clk,      LE => RW,      nRST => nRST,      dest_select => DA,
        A_select => AA,      B_select => BA,      D => D_data,      A => A_data,      B => B_data
        );
    Address_out <= A_data;
    Data_out    <= B_muxed;

    U2 : for i in 0 to reg_width-1 generate
        b_mux_2d_bit_array( 0, i )<= b_mux_vector_array( 0 )( i );
        b_mux_2d_bit_array( 1, i )<= b_mux_vector_array( 1 )( i );
        d_mux_2d_bit_array( 0, i )<= d_mux_vector_array( 0 )( i );
        d_mux_2d_bit_array( 1, i )<= d_mux_vector_array( 1 )( i );
    end generate;

    U3 : muxnto1_bus
        generic map( n_addr => 1,      bus_width => reg_width )
        port map( s => MB_vector,      w => b_mux_2d_bit_array,      f => B_muxed
        );
    MB_vector( 0 )      <= MB;
    b_mux_vector_array( 0 )      <= B_data;
    b_mux_vector_array( 1 )      <= Const_in;

    U4 : ndn_alu
        generic map( n => reg_width )

```

```
    port map( M => ALU_mode,          F => ALU_function,      X => A_data,      Y => B_muxed,  
              S => ALU_result,        Negative => ALU_N_bit,    Cout => ALU_C_bit,    Overflow => ALU_V_bit,  
              Zero => ALU_Z_bit  
            );
```

```
end ideal;
```

```

-- *****
-- **** STUDENT: 64200238
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
use ieee.math_real.all;

entity cpu_datapath is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk,      -- clock input
           RW        : in std_logic;    -- register write input  ( active '1' )
           nRST      : in std_logic;    -- reset input          ( active '0' )
           DA,       -- destination register number select input
           AA,       -- A, B bus register number select input
           BA        : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
           MB        : in std_logic;    -- constant/operand bus B bus multiplexer control signal
           MD        : in 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

           ALU_mode          : in std_logic;    -- mode of alu operation ( M in upper table )

```

```

        ALU_function : in std_logic_vector( 2 downto 0 );    -- function of ALU ( F in upper table )
        ALU_N_bit    : out std_logic;    -- Negative bit of alu operation
        ALU_C_bit    : out std_logic;    -- Carry bit of alu operation
        ALU_V_bit    : out std_logic;    -- Overflow bit of alu operation
        ALU_Z_bit    : out std_logic;    -- Zero bit of alu operation
        Const_in     : in  std_logic_vector( reg_width - 1 downto 0 );    -- constant input
bus
        Data_in      : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus
input
        Address_out  : out std_logic_vector( reg_width - 1 downto 0 );    -- address bus output
        Data_out     : out std_logic_vector( reg_width - 1 downto 0 )    -- data bus output
    );
end cpu_datapath;

architecture ideal of cpu_datapath is
    -- 2/1 bus multiplexer input type definitions
    type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal b_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
    signal d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );

    signal b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );
    signal d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );

    signal A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
    signal ALU_result : std_logic_vector( reg_width - 1 downto 0 );

    signal MB_vector : std_logic_vector( 0 downto 0 );    -- MB constant/operand bus B bus multiplexer control signal
    signal MD_vector : std_logic_vector( 0 downto 0 );    -- external data/alu result multiplexer control signal

    component ndn_alu is
        generic( n: natural := 8 );
        port( M      : in  std_logic;
            -- naèin delovanja ( '0' => aritmetični, '1' => logični )
            F      : in  std_logic_vector( 2 downto 0 );
            -- funkcijski vhod za operacije
            X, Y    : in  std_logic_vector( n-1 downto 0 );
            S      : out std_logic_vector( n-1 downto 0 );
        );
    end component ndn_alu;

```



```

        Negative,      Cout,      Overflow,      Zero,      Gout,      Pout      :
    out std_logic );
end component;

component reg_file is
    generic( nr_regs      : natural := 4;
             reg_width    : natural := 8 );
    PORT ( clk,  -- clock input
          LE      : in std_logic;    -- Load enable input      ( active '1' )
          nRST    : in std_logic;    -- reset input          ( active '0' )
          dest_select, -- register number destination select input
          A_select,  -- A, B bus destination select input
          B_select   : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
          D          : in std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
          A, B       : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
        );
end component;

component muxnto1_bus IS
    generic( n_addr      : INTEGER := 2;
             bus_width   : INTEGER := 8 );
    PORT (
        s      : IN std_logic_vector( n_addr - 1 downto 0 );
        w      : IN muxnto1_bus_type( 2**n_addr - 1 DOWNT0 0, bus_width - 1
DOWNT0 0 );
        f      : OUT std_logic_vector( bus_width - 1 downto 0 )
    );
END component;

begin

U1: ndn_alu
    generic map ( n => reg_width )
    port map (
        M => ALU_mode, F => ALU_function,      X => A_data, Y => B_muxed,      S => ALU_result,      Negative =>
        ALU_N_bit, Cout => ALU_C_bit, Overflow => ALU_V_bit,      Zero => ALU_Z_bit
    );

U2: reg_file
    generic map ( nr_regs      => nr_regs, reg_width => reg_width )

```

```

    port map(
        clk=>clk, LE => RW, nRST=>nRST, dest_select=> DA, A_select => AA, B_select => BA, D => D_data, A => A_data, B
=> B_data
    );

    Address_out  <= A_data;
    Data_out     <= B_muxed;

U3: muxnto1_bus
    generic map ( n_addr => 1, bus_width => reg_width )
    port map (
        s => MB_vector,      w => b_mux_2d_bit_array,  f => B_muxed
    );
    MB_vector( 0 )          <= MB;
    b_mux_vector_array( 0 ) <= B_data;
    b_mux_vector_array( 1 ) <= Const_in;

U4: muxnto1_bus
    generic map ( n_addr => 1, bus_width => reg_width )
    port map (
        s => MD_vector,      w => d_mux_2d_bit_array,  f => D_data
    );
    MD_vector( 0 )          <= MD;
    d_mux_vector_array( 0 ) <= ALU_result;
    d_mux_vector_array( 1 ) <= Data_in;

u5 : for i in 0 to reg_width-1 generate
    b_mux_2d_bit_array( 0, i ) <= b_mux_vector_array( 0 )( i );
    b_mux_2d_bit_array( 1, i ) <= b_mux_vector_array( 1 )( i );
    d_mux_2d_bit_array( 0, i ) <= d_mux_vector_array( 0 )( i );
    d_mux_2d_bit_array( 1, i ) <= d_mux_vector_array( 1 )( i );
end generate;

end ideal;

```

```

-- *****
-- **** STUDENT: 64200288
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
use ieee.math_real.all;

entity cpu_datapath is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk,      -- clock input
           RW        : in std_logic;    -- register write input  ( active '1' )
           nRST      : in std_logic;    -- reset input          ( active '0' )
           DA,       -- destination register number select input
           AA,       -- A, B bus register number select input
           BA        : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
           MB        : in std_logic;    -- constant/operand bus B bus multiplexer control signal
           MD        : in 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
           ALU_mode    : in std_logic;    -- mode of alu operation ( M in upper table )

```

```

        ALU_function : in std_logic_vector( 2 downto 0 );    -- function of ALU ( F in upper table )
        ALU_N_bit    : out std_logic;    -- Negative bit of alu operation
        ALU_C_bit    : out std_logic;    -- Carry bit of alu operation
        ALU_V_bit    : out std_logic;    -- Overflow bit of alu operation
        ALU_Z_bit    : out std_logic;    -- Zero bit of alu operation
        Const_in     : in  std_logic_vector( reg_width - 1 downto 0 );    -- constant input
bus
        Data_in      : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus
input
        Address_out  : out std_logic_vector( reg_width - 1 downto 0 );    -- address bus output
        Data_out     : out std_logic_vector( reg_width - 1 downto 0 )    -- data bus output
    );
end cpu_datapath;

architecture ideal of cpu_datapath is

component ndn_alu is
    generic( n: natural := 8 );
    port( M      : in  std_logic;
          -- naèin delovanja ( '0' => aritmetični, '1' => logični )
          F      : in  std_logic_vector( 2 downto 0 );
          -- funkcijski vhod za operacije
          X, Y    : in  std_logic_vector( n-1 downto 0 );
          S      : out std_logic_vector( n-1 downto 0 );
          Negative, Cout, Overflow, Zero, Gout, Pout    :
            out std_logic );
end component;

-- 2/1 bus multiplexer input type definitions
type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal b_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );

signal b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
'0' ) );
signal d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
'0' ) );

signal A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
signal ALU_result : std_logic_vector( reg_width - 1 downto 0 );

```

```

signal      MB_vector : std_logic_vector( 0 downto 0 );  -- MB constant/operand bus B bus multiplexer control signal
signal      MD_vector : std_logic_vector( 0 downto 0 );  -- external data/alu result multiplexer control signal

begin
  rfile: reg_file
  generic map ( nr_regs => nr_regs, reg_width => reg_width )
  port map (
    clk => clk,  LE => RW,    nRST => nRST,          dest_select => DA,  A_select => AA,    B_select => BA,
    B => B_data, A => A_data, D => D_data
  );

  Data_out      <= B_muxed;
  Address_out   <= A_data;

  muxb: muxnto1_bus
  generic map ( n_addr => 1, bus_width => reg_width )
  port map (
    s => MB_vector,    w => b_mux_2d_bit_array,  f => B_muxed
  );

  MB_vector( 0 )      <= MB;
  b_mux_vector_array( 0 ) <= B_data;
  b_mux_vector_array( 1 ) <= Const_in;

  muxd: muxnto1_bus
  generic map ( n_addr => 1, bus_width => reg_width )
  port map (
    s => MD_vector,    w => d_mux_2d_bit_array,  f => D_data
  );

  MD_vector( 0 )      <= MD;
  d_mux_vector_array( 0 ) <= ALU_result;
  d_mux_vector_array( 1 ) <= Data_in;

  arrays_to_2d: for i in 0 to reg_width-1 generate
    b_mux_2d_bit_array( 0, i ) <= b_mux_vector_array( 0 )( i );
    d_mux_2d_bit_array( 0, i ) <= d_mux_vector_array( 0 )( i );
    d_mux_2d_bit_array( 1, i ) <= d_mux_vector_array( 1 )( i );
    b_mux_2d_bit_array( 1, i ) <= b_mux_vector_array( 1 )( i );
  end generate

```

```

end generate;

alu: ndn_alu
generic map ( n => reg_width )
port map (
    M => ALU_mode,      F => ALU_function, X => A_data, Y => B_muxed,      S => ALU_result,      Negative =>
    ALU_N_bit, Cout => ALU_C_bit, Overflow => ALU_V_bit,      Zero => ALU_Z_bit
);

end ideal;

```

```

-- *****
-- **** STUDENT: 64200296
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
use ieee.math_real.all;

entity cpu_datapath is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk,      -- clock input
           RW        : in std_logic;    -- register write input  ( active '1' )
           nRST      : in std_logic;    -- reset input          ( active '0' )
           DA,       -- destination register number select input
           AA,       -- A, B bus register number select input
           BA        : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
           MB        : in std_logic;    -- constant/operand bus B bus multiplexer control signal
           MD        : in 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

           ALU_mode          : in std_logic;    -- mode of alu operation ( M in upper table )

```

```

        ALU_function : in std_logic_vector( 2 downto 0 );    -- function of ALU ( F in upper table )
        ALU_N_bit    : out std_logic;    -- Negative bit of alu operation
        ALU_C_bit    : out std_logic;    -- Carry bit of alu operation
        ALU_V_bit    : out std_logic;    -- Overflow bit of alu operation
        ALU_Z_bit    : out std_logic;    -- Zero bit of alu operation
        Const_in     : in  std_logic_vector( reg_width - 1 downto 0 );    -- constant input
bus
        Data_in      : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus
input
        Address_out  : out std_logic_vector( reg_width - 1 downto 0 );    -- address bus output
        Data_out     : out std_logic_vector( reg_width - 1 downto 0 )    -- data bus output
    );
end cpu_datapath;

architecture ideal of cpu_datapath is
    -- 2/1 bus multiplexer input type definitions
    type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal b_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
    signal d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );

    signal b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );
    signal d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );

    signal A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
    signal ALU_result : std_logic_vector( reg_width - 1 downto 0 );

    signal MB_vector : std_logic_vector( 0 downto 0 );    -- MB constant/operand bus B bus multiplexer control signal
    signal MD_vector : std_logic_vector( 0 downto 0 );    -- external data/alu result multiplexer control signal

begin

    mux_bus_b: muxnto1_bus
        generic map ( n_addr => 1,                bus_width => reg_width )
        port map (
            s => MB_vector,    w => b_mux_2d_bit_array,    f => B_muxed
        );

    mux_bus_d: muxnto1_bus

```



```

generic map ( n_addr => 1,                bus_width => reg_width )
port map (
    s => MD_vector,    w => d_mux_2d_bit_array,  f => D_data
);

regfile: reg_file
generic map ( nr_regs => nr_regs,          reg_width => reg_width )
port map (
    clk => clk,  LE => RW,    nRST => nRST,    dest_select => DA,  A_select => AA,    B_select => BA,
    D => D_data, A => A_data, B => B_data
);

alu: ndn_alu
generic map ( n => reg_width )
port map (
    M => ALU_mode,    F => ALU_function, X => A_data, Y => B_muxed,    S => ALU_result,    Negative =>
    ALU_N_bit,    Cout => ALU_C_bit, Overflow => ALU_V_bit,    Zero => ALU_Z_bit
);

process( b_mux_vector_array, d_mux_vector_array )
begin
    for i in 1 downto 0 loop
        for j in reg_width-1 downto 0 loop
            b_mux_2d_bit_array( i,j ) <= b_mux_vector_array( i )( j );
            d_mux_2d_bit_array( i,j ) <= d_mux_vector_array( i )( j );
        end loop;
    end loop;
end process;

MB_vector( 0 )    <= MB;
MD_vector( 0 )    <= MD;
b_mux_vector_array( 0 )    <= B_data;
b_mux_vector_array( 1 )    <= Const_in;
d_mux_vector_array( 0 )    <= ALU_result;
d_mux_vector_array( 1 )    <= Data_in;
Address_out <= A_data;
Data_out    <= B_muxed;

end ideal;

```

```

-- *****
-- **** STUDENT: 64200385
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
use ieee.math_real.all;

entity cpu_datapath is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk,      -- clock input
           RW        : in std_logic;    -- register write input  ( active '1' )
           nRST      : in std_logic;    -- reset input          ( active '0' )
           DA,       -- destination register number select input
           AA,       -- A, B bus register number select input
           BA        : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
           MB        : in std_logic;    -- constant/operand bus B bus multiplexer control signal
           MD        : in 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

           ALU_mode          : in std_logic;    -- mode of alu operation ( M in upper table )

```

```

        ALU_function : in std_logic_vector( 2 downto 0 );    -- function of ALU ( F in upper table )
        ALU_N_bit    : out std_logic;    -- Negative bit of alu operation
        ALU_C_bit    : out std_logic;    -- Carry bit of alu operation
        ALU_V_bit    : out std_logic;    -- Overflow bit of alu operation
        ALU_Z_bit    : out std_logic;    -- Zero bit of alu operation
        Const_in     : in  std_logic_vector( reg_width - 1 downto 0 );    -- constant input
bus
        Data_in      : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus
input
        Address_out  : out std_logic_vector( reg_width - 1 downto 0 );    -- address bus output
        Data_out     : out std_logic_vector( reg_width - 1 downto 0 )    -- data bus output
    );
end cpu_datapath;

architecture ideal of cpu_datapath is
    -- 2/1 bus multiplexer input type definitions
    type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal b_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
    signal d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );

    signal b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );
    signal d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );

    signal A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
    signal ALU_result : std_logic_vector( reg_width - 1 downto 0 );

    signal MB_vector : std_logic_vector( 0 downto 0 );    -- MB constant/operand bus B bus multiplexer control signal
    signal MD_vector : std_logic_vector( 0 downto 0 );    -- external data/alu result multiplexer control signal
begin

    U0: reg_file
        generic map ( nr_regs=>nr_regs,
            reg_width=>reg_width )
        port map( clk => clk,
            LE => RW,
            nRST => nRST,
            dest_select => DA,

```

```

A_select => AA,
B_select => BA,
D => D_data,
A => A_data,
B => B_data );

U1: muxnto1_bus
generic map ( n_addr => 1,
bus_width => reg_width )
port map( s => MB_vector,
w => b_mux_2d_bit_array,
f => B_muxed );

b_mux_vector_array( 0 ) <= B_data;
b_mux_vector_array( 1 ) <= Const_in;
MB_vector( 0 ) <= MB;

Address_out <= A_data;
Data_out <= B_muxed;

U2: ndn_alu
generic map ( n => reg_width )
port map( M => ALU_mode,
F => ALU_function,
X => A_data,
Y => B_muxed,
S => ALU_result,
Negative => ALU_N_bit,
Cout => ALU_C_bit,
Overflow => ALU_V_bit,
Zero => ALU_Z_bit,
Gout => open,
Pout => open );

U3: muxnto1_bus
generic map ( n_addr => 1,
bus_width => reg_width )
port map( s => MD_vector,
w => d_mux_2d_bit_array,
f => D_data );

```

```

d_mux_vector_array( 0 )  <= ALU_result;
d_mux_vector_array( 1 )  <= Data_in;
MD_vector( 0 )          <= MD;

vt2: for i in 0 to reg_width - 1 generate
b_mux_2d_bit_array( 0,i )<= b_mux_vector_array( 0 )( i );
b_mux_2d_bit_array( 1,i )<= b_mux_vector_array( 1 )( i );
d_mux_2d_bit_array( 0,i )<= d_mux_vector_array( 0 )( i );
d_mux_2d_bit_array( 1,i )<= d_mux_vector_array( 1 )( i );
end generate;

end ideal;

```

```

-- *****
-- **** STUDENT: 64210113
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
use ieee.math_real.all;

entity cpu_datapath is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk,      -- clock input
           RW        : in std_logic;    -- register write input  ( active '1' )
           nRST      : in std_logic;    -- reset input          ( active '0' )
           DA,       -- destination register number select input
           AA,       -- A, B bus register number select input
           BA        : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
           MB        : in std_logic;    -- constant/operand bus B bus multiplexer control signal
           MD        : in 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

           ALU_mode          : in std_logic;    -- mode of alu operation ( M in upper table )

```

```

        ALU_function : in std_logic_vector( 2 downto 0 );    -- function of ALU ( F in upper table )
        ALU_N_bit    : out std_logic;    -- Negative bit of alu operation
        ALU_C_bit    : out std_logic;    -- Carry bit of alu operation
        ALU_V_bit    : out std_logic;    -- Overflow bit of alu operation
        ALU_Z_bit    : out std_logic;    -- Zero bit of alu operation
        Const_in     : in  std_logic_vector( reg_width - 1 downto 0 );    -- constant input
bus
        Data_in      : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus
input
        Address_out  : out std_logic_vector( reg_width - 1 downto 0 );    -- address bus output
        Data_out     : out std_logic_vector( reg_width - 1 downto 0 );    -- data bus output
    );
end cpu_datapath;

architecture ideal of cpu_datapath is
    -- 2/1 bus multiplexer input type definitions
    type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal b_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
    signal d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );

    signal b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );
    signal d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );

    signal A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
    signal ALU_result : std_logic_vector( reg_width - 1 downto 0 );

    signal MB_vector : std_logic_vector( 0 downto 0 );    -- MB constant/operand bus B bus multiplexer control signal
    signal MD_vector : std_logic_vector( 0 downto 0 );    -- external data/alu result multiplexer control signal
begin

    registerFile: reg_file
    generic map ( nr_regs => nr_regs, reg_width => reg_width )
    port map (
        clk => clk,          LE => RW,          nRST => nRST,          dest_select => DA,
        A_select => AA,      B_select => BA,      D => D_data,          A => A_data,          B => B_data
    );

```

```
Data_out    <= B_muxed;
Address_out <= A_data;
```

```
muxb: muxnto1_bus
generic map ( n_addr => 1, bus_width => reg_width )
port map (
    s => MB_vector,          w => b_mux_2d_bit_array,          f => B_muxed );
```

```
b_mux_vector_array( 0 ) <= B_data;
b_mux_vector_array( 1 ) <= Const_in;
MB_vector( 0 ) <= MB;
```

```
muxd: muxnto1_bus
generic map ( n_addr => 1, bus_width => reg_width )
port map (
    s => MD_vector,          w => d_mux_2d_bit_array,          f => D_data );
```

```
d_mux_vector_array( 0 ) <= ALU_result;
d_mux_vector_array( 1 ) <= Data_in;
MD_vector( 0 ) <= MD;
```

```
arrays_to_2d: for i in 0 to reg_width-1 generate
    b_mux_2d_bit_array( 0, i ) <= b_mux_vector_array( 0 )( i );
    d_mux_2d_bit_array( 0, i ) <= d_mux_vector_array( 0 )( i );
    b_mux_2d_bit_array( 1, i ) <= b_mux_vector_array( 1 )( i );
    d_mux_2d_bit_array( 1, i ) <= d_mux_vector_array( 1 )( i );
```

```
end generate;
```

```
alu: ndn_alu
generic map( n => reg_width )
port map (
    M => ALU_mode,          F => ALU_function,          S => ALU_result,          X => A_data,
    Y => B_muxed,          Cout => ALU_C_bit,          Overflow => ALU_V_bit,          Negative => ALU_N_bit,
    Zero => ALU_Z_bit );
```

```
end ideal;
```



```

-- *****
-- **** STUDENT: 64210290
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE ieee.math_real.all;

ENTITY cpu_datapath IS
  GENERIC(
    nr_regs : NATURAL := 4;
    reg_width : NATURAL := 8
  );
  PORT(
    clk,      -- clock input
    RW        : IN STD_LOGIC;      -- register write input ( active '1' )
    nRST      : IN STD_LOGIC;      -- reset input ( active '0' )
    DA,       -- destination register number select input
    AA,       -- A, B bus register number select input
    BA        : IN STD_LOGIC_VECTOR( sizeof( nr_regs - 1 ) - 1 DOWNTO 0 );
    MB        : IN STD_LOGIC;      -- constant/operand bus B bus multiplexer control signal
    MD        : IN 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
    ALU_mode          : IN STD_LOGIC;      -- mode of alu operation ( M in upper table )
    ALU_function : IN STD_LOGIC_VECTOR( 2 DOWNTO 0 ); -- function of ALU ( F in upper table )
    ALU_N_bit         : OUT STD_LOGIC;      -- Negative bit of alu operation
    ALU_C_bit         : OUT STD_LOGIC;      -- Carry bit of alu operation
    ALU_V_bit         : OUT STD_LOGIC;      -- Overflow bit of alu operation
    ALU_Z_bit         : OUT STD_LOGIC;      -- Zero bit of alu operation
    Const_in          : IN STD_LOGIC_VECTOR( reg_width - 1 DOWNTO 0 ); -- constant input bus
    Data_in           : IN STD_LOGIC_VECTOR( reg_width - 1 DOWNTO 0 ); -- data input bus input
    Address_out       : OUT STD_LOGIC_VECTOR( reg_width - 1 DOWNTO 0 ); -- address bus output
    Data_out          : OUT STD_LOGIC_VECTOR( reg_width - 1 DOWNTO 0 ); -- data bus output
);
END cpu_datapath;

ARCHITECTURE ideal OF cpu_datapath IS

    component reg_file is
        generic(
            nr_regs          : natural;
            reg_width        : natural
        );
        port(
            clk, -- clock input
            LE   : in std_logic; -- Load enable input ( active '1' )
            nRST : in std_logic; -- reset input ( active '0' )
            dest_select, -- register number destination select input
            A_select, -- A, B bus destination select input
            B_select  : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
            D          : in std_logic_vector( reg_width - 1 downto 0 ); -- data input bus input
            A, B       : out std_logic_vector( reg_width - 1 downto 0 ) -- A, B bus output
        );
    end component;

    component muxnto1_bus is
        generic(
            n_addr      : integer;
            bus_width    : integer
        );
        port(

```

```

        s : in      std_logic_vector( n_addr - 1 downto 0 );
        w : in muxnto1_bus_type( 2**n_addr - 1 downto 0, bus_width - 1 downto 0 );
        f : out     std_logic_vector( bus_width - 1 downto 0 )
    );
end component;

component ndn_alu is
    generic(
        n: natural
    );
    port(
        M          : in      std_logic;
        -- naèin delovanja ( '0' => aritmetični, '1' => logični )
        F          : in      std_logic_vector( 2 downto 0 );
        -- funkcijski vhod za operacije
        X, Y       : in      std_logic_vector( n-1 downto 0 );
        S          : out     std_logic_vector( n-1 downto 0 );
        Negative, Cout, Overflow, Zero, Gout, Pout      :
    out std_logic
    );
end component;

-- 2/1 bus multiplexer input type definitions
type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal b_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );

signal b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => (
others => '0' ) );
signal d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => (
others => '0' ) );

signal A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
signal ALU_result : std_logic_vector( reg_width - 1 downto 0 );

signal MB_vector : std_logic_vector( 0 downto 0 ); -- MB constant/operand bus B bus multiplexer control
signal MD_vector : std_logic_vector( 0 downto 0 ); -- external data/alu result multiplexer control
signal

```

BEGIN

```
MB_vector( 0 )      <= MB;
MD_vector( 0 )      <= MD;

mux_wires : process( b_mux_vector_array, d_mux_vector_array )
begin
    for hor in 0 to reg_width-1 loop
        b_mux_2d_bit_array( 0,hor )    <= b_mux_vector_array( 0 )( hor );
        b_mux_2d_bit_array( 1,hor )    <= b_mux_vector_array( 1 )( hor );
        d_mux_2d_bit_array( 0,hor )    <= d_mux_vector_array( 0 )( hor );
        d_mux_2d_bit_array( 1,hor )    <= d_mux_vector_array( 1 )( hor );
    end loop;
end process;

regs : reg_file
    generic map(
        nr_regs => nr_regs,          reg_width => reg_width
    )
    port map(
        clk => clk,          LE => RW,          nRST => nRST,          dest_select => DA,
A_select => AA,          B_select => BA,          D => D_data,          A => A_data,          B => B_data
    );

b_mux_vector_array( 0 )    <= B_data;
b_mux_vector_array( 1 )    <= Const_in;

muxb : muxnto1_bus
    generic map(
        n_addr => 1,          bus_width => reg_width
    )
    port map(
        s => MB_vector,          w => b_mux_2d_bit_array,          f => B_muxed
    );

alu : ndn_alu
    generic map(
        n => reg_width
    )
    port map(
```

```

        M => ALU_mode, -- način delovanja ( '0' => aritmetični, '1' => Logični )
        F => ALU_function, -- funkcijski vhod za operacije
        X => A_data,      Y => B_muxed,      S => ALU_result,      Negative => ALU_N_bit,
Cout => ALU_C_bit,      Overflow => ALU_V_bit,      Zero => ALU_Z_bit,      Gout => open,
Pout => open
    );

```

```

d_mux_vector_array( 0 )    <= ALU_result;
d_mux_vector_array( 1 )    <= Data_in;

```

```

muxd : muxnto1_bus
    generic map(
        n_addr => 1,      bus_width => reg_width
    )
    port map(
        s => MD_vector,      w => d_mux_2d_bit_array,      f => D_data
    );

```

```

Address_out    <= A_data;
Data_out       <= B_muxed;

```

```

END ideal;

```

```

-- *****
-- **** STUDENT: 64210382
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
use ieee.math_real.all;

entity cpu_datapath is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk,      -- clock input
          RW         : in std_logic;    -- register write input  ( active '1' )
          nRST       : in std_logic;    -- reset input          ( active '0' )
          DA,        -- destination register number select input
          AA,        -- A, B bus register number select input
          BA         : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
          MB         : in std_logic;    -- constant/operand bus B bus multiplexer control signal
          MD         : in 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
          ALU_mode     : in std_logic;    -- mode of alu operation ( M in upper table )

```

```

        ALU_function : in std_logic_vector( 2 downto 0 );    -- function of ALU ( F in upper table )
        ALU_N_bit    : out std_logic;    -- Negative bit of alu operation
        ALU_C_bit    : out std_logic;    -- Carry bit of alu operation
        ALU_V_bit    : out std_logic;    -- Overflow bit of alu operation
        ALU_Z_bit    : out std_logic;    -- Zero bit of alu operation
        Const_in     : in  std_logic_vector( reg_width - 1 downto 0 );    -- constant input
bus
        Data_in      : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus
input
        Address_out  : out std_logic_vector( reg_width - 1 downto 0 );    -- address bus output
        Data_out     : out std_logic_vector( reg_width - 1 downto 0 )    -- data bus output
    );
end cpu_datapath;

architecture ideal of cpu_datapath is
    -- 2/1 bus multiplexer input type definitions
    type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal b_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
    signal d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );

    signal b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );
    signal d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );

    signal A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
    signal ALU_result : std_logic_vector( reg_width - 1 downto 0 );

    signal MB_vector : std_logic_vector( 0 downto 0 );    -- MB constant/operand bus B bus multiplexer control signal
    signal MD_vector : std_logic_vector( 0 downto 0 );    -- external data/alu result multiplexer control signal
begin
    rfile: reg_file
    generic map ( nr_regs => nr_regs, reg_width => reg_width )
    port map (
        clk => clk, LE => RW, nRST => nRST, dest_select => DA,    A_select => AA, B_select => BA, D => D_data,
        A => A_data, B => B_data
    );
    Address_out <= A_data;
    Data_out    <= B_muxed;

```

```

muxb: muxnto1_bus
generic map ( n_addr => 1, bus_width => reg_width )
port map (
    s => MB_vector, w => b_mux_2d_bit_array, f => B_muxed
);
MB_vector( 0 )      <= MB;
b_mux_vector_array( 0 )  <= B_data;
b_mux_vector_array( 1 )  <= Const_in;

muxd: muxnto1_bus
generic map ( n_addr => 1, bus_width => reg_width )
port map (
    s => MD_vector, w => d_mux_2d_bit_array, f => D_data
);
MD_vector( 0 )      <= MD;
d_mux_vector_array( 0 )  <= ALU_result;
d_mux_vector_array( 1 )  <= Data_in;

arrays_to_2d: for i in 0 to reg_width-1 generate
    b_mux_2d_bit_array( 0, i ) <= b_mux_vector_array( 0 )( i );
    b_mux_2d_bit_array( 1, i ) <= b_mux_vector_array( 1 )( i );
    d_mux_2d_bit_array( 0, i ) <= d_mux_vector_array( 0 )( i );
    d_mux_2d_bit_array( 1, i ) <= d_mux_vector_array( 1 )( i );
end generate;

alu: ndn_alu
generic map ( n => reg_width )
port map (
    M => ALU_mode, F => ALU_function,      X => A_data, Y => B_muxed, S => ALU_result,   Negative =>
    ALU_N_bit, Cout => ALU_C_bit,   Overflow => ALU_V_bit, Zero => ALU_Z_bit
);
end ideal;

```



```

-- *****
-- **** STUDENT: 64210384
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
use ieee.math_real.all;

entity cpu_datapath is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk, -- clock input
          RW      : in std_logic; -- register write input ( active '1' )
          nRST    : in std_logic; -- reset input ( active '0' )
          DA, -- destination register number select input
          AA, -- A, B bus register number select input
          BA      : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
          MB      : in std_logic; -- constant/operand bus B bus multiplexer control signal
          MD      : in 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
          ALU_mode          : in std_logic; -- mode of alu operation ( M in upper table )

```

```

        ALU_function : in std_logic_vector( 2 downto 0 );    -- function of ALU ( F in upper table )
        ALU_N_bit    : out std_logic;    -- Negative bit of alu operation
        ALU_C_bit    : out std_logic;    -- Carry bit of alu operation
        ALU_V_bit    : out std_logic;    -- Overflow bit of alu operation
        ALU_Z_bit    : out std_logic;    -- Zero bit of alu operation
        Const_in     : in  std_logic_vector( reg_width - 1 downto 0 );    -- constant input
bus
        Data_in      : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus
input
        Address_out  : out std_logic_vector( reg_width - 1 downto 0 );    -- address bus output
        Data_out     : out std_logic_vector( reg_width - 1 downto 0 );    -- data bus output
    );
end cpu_datapath;

architecture ideal of cpu_datapath is
    -- 2/1 bus multiplexer input type definitions
    type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal    b_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
    signal    d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );

    signal    b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );
    signal    d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );

    signal    A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
    signal    ALU_result : std_logic_vector( reg_width - 1 downto 0 );

    signal    MB_vector : std_logic_vector( 0 downto 0 );    -- MB constant/operand bus B bus multiplexer control signal
    signal    MD_vector : std_logic_vector( 0 downto 0 );    -- external data/alu result multiplexer control signal

begin

register_block: reg_file
    generic map(
        nr_regs => nr_regs,                reg_width => reg_width
    )
    port map(
        clk => clk,                LE => RW,                nRST => nRST,                dest_select => DA,
        A_select => AA,                B_select => BA,                D => D_data,                A => A_data,                B => B_data
    );
end architecture ideal;

```

```

);

mux_B: muxnto1_bus
    generic map(
        n_addr => MB_vector'length,          bus_width => reg_width
    )
    port map(
        s => MB_vector,          w => b_mux_2d_bit_array,          f => B_muxed
    );

alu: ndn_alu
    generic map( n => reg_width )
    port map(
        M => ALU_mode,          F => ALU_function,          X => A_data,          Y => B_muxed,
        S => ALU_result,          Negative => ALU_N_bit,          Zero => ALU_Z_bit,          Cout => ALU_C_bit,
        Overflow => ALU_V_bit,          Gout => open,          Pout => open
    );

mux_D: muxnto1_bus
    generic map(
        n_addr => MD_vector'length,          bus_width => reg_width
    )
    port map(
        s => MD_vector,          w => d_mux_2d_bit_array,          f => D_data
    );

MB_vector( 0 )      <= MB;
b_mux_vector_array( 0 )  <= B_data;
b_mux_vector_array( 1 )  <= Const_in;

MD_vector( 0 )      <= MD;
d_mux_vector_array( 0 )  <= ALU_result;
d_mux_vector_array( 1 )  <= Data_in;

prilagajanje_2d:
for i in 0 to reg_width - 1 generate
    b_mux_2d_bit_array( 0, i )<= b_mux_vector_array( 0 )( i );
    b_mux_2d_bit_array( 1, i )<= b_mux_vector_array( 1 )( i );

    d_mux_2d_bit_array( 0, i )<= d_mux_vector_array( 0 )( i );

```

```
        d_mux_2d_bit_array( 1, i )<= d_mux_vector_array( 1 )( i );  
end generate;  
  
Address_out  <= A_data;  
Data_out     <= B_muxed;  
  
end ideal;
```

```

-- *****
-- **** STUDENT: 64210386
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
use ieee.math_real.all;

entity cpu_datapath is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk,      -- clock input
           RW        : in std_logic;    -- register write input  ( active '1' )
           nRST      : in std_logic;    -- reset input          ( active '0' )
           DA,       -- destination register number select input
           AA,       -- A, B bus register number select input
           BA        : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
           MB        : in std_logic;    -- constant/operand bus B bus multiplexer control signal
           MD        : in 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 ( dvojiki 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
           ALU_mode          : in std_logic;    -- mode of alu operation ( M in upper table )

```

```

        ALU_function : in std_logic_vector( 2 downto 0 );    -- function of ALU ( F in upper table )
        ALU_N_bit    : out std_logic;    -- Negative bit of alu operation
        ALU_C_bit    : out std_logic;    -- Carry bit of alu operation
        ALU_V_bit    : out std_logic;    -- Overflow bit of alu operation
        ALU_Z_bit    : out std_logic;    -- Zero bit of alu operation
        Const_in     : in  std_logic_vector( reg_width - 1 downto 0 );    -- constant input
bus
        Data_in      : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus
input
        Address_out  : out std_logic_vector( reg_width - 1 downto 0 );    -- address bus output
        Data_out     : out std_logic_vector( reg_width - 1 downto 0 )    -- data bus output
    );
end cpu_datapath;

architecture ideal of cpu_datapath is

component ndn_alu is
    generic( n: natural := 8 );
    port( M          : in  std_logic;
          -- delovanja ( '0' => aritmetični, '1' => logični )
          F          : in  std_logic_vector( 2 downto 0 );
          -- funkcijski vhod za operacije
          X, Y       : in  std_logic_vector( n-1 downto 0 );
          S          : out std_logic_vector( n-1 downto 0 );
          Negative, Cout, Overflow, Zero, Gout, Pout      :
            out std_logic );
end component;

-- 2/1 bus multiplexer input type definitions
type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal b_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );

signal b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
'0' ) );
signal d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
'0' ) );

signal A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
signal ALU_result : std_logic_vector( reg_width - 1 downto 0 );

```

```

signal      MB_vector : std_logic_vector( 0 downto 0 );  -- MB constant/operand bus B bus multiplexer control signal
signal      MD_vector : std_logic_vector( 0 downto 0 );  -- external data/alu result multiplexer control signal

begin

    reg: reg_file
    generic map ( nr_regs => nr_regs, reg_width => reg_width )
    port map (
        clk => clk, LE => RW, nRST => nRST, dest_select => DA,      A_select => AA, B_select => BA, D => D_data,
        A => A_data, B => B_data
    );
    Address_out  <= A_data;
    Data_out     <= B_muxed;

    mux: muxnto1_bus
    generic map ( n_addr => 1, bus_width => reg_width )
    port map (
        s => MB_vector, w => b_mux_2d_bit_array, f => B_muxed
    );
    MB_vector( 0 )      <= MB;
    b_mux_vector_array( 0 ) <= B_data;
    b_mux_vector_array( 1 ) <= Const_in;

    mux_1: muxnto1_bus
    generic map ( n_addr => 1, bus_width => reg_width )
    port map (
        s => MD_vector, w => d_mux_2d_bit_array, f => D_data
    );
    MD_vector( 0 )      <= MD;
    d_mux_vector_array( 0 ) <= ALU_result;
    d_mux_vector_array( 1 ) <= Data_in;

    arrays_to_2d: for i in 0 to reg_width-1 generate
        b_mux_2d_bit_array( 0, i ) <= b_mux_vector_array( 0 )( i );
        b_mux_2d_bit_array( 1, i ) <= b_mux_vector_array( 1 )( i );
        d_mux_2d_bit_array( 0, i ) <= d_mux_vector_array( 0 )( i );
        d_mux_2d_bit_array( 1, i ) <= d_mux_vector_array( 1 )( i );
    end generate;

```

```
alu: ndn_alu
generic map ( n => reg_width )
port map (
    M => ALU_mode, F => ALU_function,      X => A_data, Y => B_muxed, S => ALU_result,   Negative =>
    ALU_N_bit, Cout => ALU_C_bit,   Overflow => ALU_V_bit, Zero => ALU_Z_bit
);

end ideal;
```



```

-- *****
-- **** STUDENT: 64210445
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
use ieee.math_real.all;

entity cpu_datapath is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk,      -- clock input
           RW        : in std_logic;    -- register write input ( active '1' )
           nRST      : in std_logic;    -- reset input ( active '0' )
           DA,       -- destination register number select input
           AA,       -- A, B bus register number select input
           BA        : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
           MB        : in std_logic;    -- constant/operand bus B bus multiplexer control signal
           MD        : in 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

           ALU_mode          : in std_logic;    -- mode of alu operation ( M in upper table )

```

```

        ALU_function : in std_logic_vector( 2 downto 0 );    -- function of ALU ( F in upper table )
        ALU_N_bit    : out std_logic;    -- Negative bit of alu operation
        ALU_C_bit    : out std_logic;    -- Carry bit of alu operation
        ALU_V_bit    : out std_logic;    -- Overflow bit of alu operation
        ALU_Z_bit    : out std_logic;    -- Zero bit of alu operation
        Const_in     : in  std_logic_vector( reg_width - 1 downto 0 );    -- constant input
bus
        Data_in      : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus
input
        Address_out  : out std_logic_vector( reg_width - 1 downto 0 );    -- address bus output
        Data_out     : out std_logic_vector( reg_width - 1 downto 0 );    -- data bus output
    );
end cpu_datapath;

architecture ideal of cpu_datapath is
    -- 2/1 bus multiplexer input type definitions
    type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal b_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
    signal d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );

    signal b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );
    signal d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );

    signal A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
    signal ALU_result : std_logic_vector( reg_width - 1 downto 0 );

    signal MB_vector : std_logic_vector( 0 downto 0 );    -- MB constant/operand bus B bus multiplexer control signal
    signal MD_vector : std_logic_vector( 0 downto 0 );    -- external data/alu result multiplexer control signal
begin

    rfile: reg_file
    generic map ( nr_regs => nr_regs, reg_width => reg_width )
    port map( clk => clk,          LE => RW,          nRST => nRST,          dest_select => DA,
        A_select => AA,          B_select => BA,          D => D_data,          A => A_data,          B => B_data );
    Address_out <= A_data;
    Data_out    <= B_muxed;

```

```

muxb: muxnto1_bus
generic map ( n_addr => 1, bus_width => reg_width )
port map( s => MB_vector,          w => b_mux_2d_bit_array,          f => B_muxed );

MB_vector( 0 )      <= MB;
b_mux_vector_array( 0 )  <= B_data;
b_mux_vector_array( 1 )  <= Const_in;

muxd: muxnto1_bus
generic map ( n_addr => 1, bus_width => reg_width )
port map( s => MD_vector,          w => d_mux_2d_bit_array,          f => D_data );

MD_vector( 0 )      <= MD;
d_mux_vector_array( 0 )  <= ALU_result;
d_mux_vector_array( 1 )  <= Data_in;

arrays_to_2d: for i in 0 to reg_width-1 generate
    b_mux_2d_bit_array( 0, i )<= b_mux_vector_array( 0 )( i );
    b_mux_2d_bit_array( 1, i )<= b_mux_vector_array( 1 )( i );
    d_mux_2d_bit_array( 0, i )<= d_mux_vector_array( 0 )( i );
    d_mux_2d_bit_array( 1, i )<= d_mux_vector_array( 1 )( i );
end generate;

alu: ndn_alu
generic map ( n => reg_width )
port map( M => ALU_mode,          F => ALU_function,          X => A_data,          Y => B_muxed,          S =>
ALU_result,          Negative => ALU_N_bit,          Cout => ALU_C_bit,          Overflow => ALU_V_bit,          Zero
=> ALU_Z_bit );

end ideal;

```

```

-- *****
-- **** STUDENT: 64210455
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni rezultatov simulacije zaradi napak sintetizatorja:
ERROR:HDLCompiler:374 - "cpu_datapath.vhd" Line 1: Entity <cpu_datapath> is not yet compiled.
ERROR:HDLCompiler:24 - "cpu_datapath.vhd" Line 2: "***" expects 2 arguments
ERROR:HDLCompiler:845 - "cpu_datapath.vhd" Line 3: Type of aggregate cannot be determined without context ; 0 visible
types match here ...
Manjka vam deklaracija entitete. Če dodam deklaracijo, dobim rezultate simulacije, vendar sta Address_out in Data_out
vedno nič, ker vam manjkajo for zanke za preslikavo 2d bitnih polj to 1d polje vektorjev std_logic_vector za
multiplekserja B (muxb) in D (muxd). Če to dodam, so C,V,Z biti nedefinirani ('U').
Manjka povezava komponente polja registrov (REG_FILE) in povezave bitov v MB_vector.
-- *****
architecture ideal of cpu_datapath is
    type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal
        b_mux_vector_array, d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
    signal
        b_mux_2d_bit_array, d_mux_2d_bit_array : muxnto1_bus_type( 1 downto 0, reg_width - 1 downto 0 ) :=
( others => ( others => '0' ) );

    signal
        A_data, B_data, B_muxed, D_data, ALU_result : std_logic_vector( reg_width - 1 downto 0 );
    signal
        MB_vector, MD_vector : std_logic_vector( 0 downto 0 );

begin
    -- MUX za vhod v ALU
    muxb: muxnto1_bus
    generic map ( n_addr => 1, bus_width => reg_width )
    port map (
        s => MB_vector, w => b_mux_2d_bit_array, f => B_muxed
    );

    -- MUX za rezultat ALU ali zunanje podatke
    muxd: muxnto1_bus
    generic map ( n_addr => 1, bus_width => reg_width )
    port map (
        s => MD_vector, w => d_mux_2d_bit_array, f => D_data
    );

    -- ALU

```

```
alu: ndn_alu
generic map ( n => reg_width )
port map (
M => ALU_mode, F => ALU_function, X => A_data, Y => B_muxed, S => ALU_result,
Negative => ALU_N_bit, Cout => ALU_C_bit, Overflow => ALU_V_bit, Zero => ALU_Z_bit
);

    -- Prenos podatkov
Address_out <= A_data;
Data_out    <= B_muxed;

end architecture ideal;
```

```

-- *****
-- **** STUDENT: 64210457
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
use ieee.math_real.all;

entity cpu_datapath is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk,      -- clock input
           RW        : in std_logic;    -- register write input  ( active '1' )
           nRST      : in std_logic;    -- reset input          ( active '0' )
           DA,       -- destination register number select input
           AA,       -- A, B bus register number select input
           BA        : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
           MB        : in std_logic;    -- constant/operand bus B bus multiplexer control signal
           MD        : in 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

           ALU_mode          : in std_logic;    -- mode of alu operation ( M in upper table )

```

```

        ALU_function : in std_logic_vector( 2 downto 0 );    -- function of ALU ( F in upper table )
        ALU_N_bit    : out std_logic;    -- Negative bit of alu operation
        ALU_C_bit    : out std_logic;    -- Carry bit of alu operation
        ALU_V_bit    : out std_logic;    -- Overflow bit of alu operation
        ALU_Z_bit    : out std_logic;    -- Zero bit of alu operation
        Const_in     : in  std_logic_vector( reg_width - 1 downto 0 );    -- constant input
bus
        Data_in      : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus
input
        Address_out  : out std_logic_vector( reg_width - 1 downto 0 );    -- address bus output
        Data_out     : out std_logic_vector( reg_width - 1 downto 0 )    -- data bus output
    );
end cpu_datapath;

architecture ideal of cpu_datapath is

component ndn_alu is
    generic( n: natural := 8 );
    port( M      : in  std_logic;
          -- naèin delovanja ( '0' => aritmetični, '1' => logični )
          F      : in  std_logic_vector( 2 downto 0 );
          -- funkcijski vhod za operacije
          X, Y    : in  std_logic_vector( n-1 downto 0 );
          S      : out std_logic_vector( n-1 downto 0 );
          Negative, Cout, Overflow, Zero, Gout, Pout    :
            out std_logic );
end component;

-- 2/1 bus multiplexer input type definitions
type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal b_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );

signal b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNTO 0, reg_width - 1 DOWNTO 0 ) := ( others => ( others =>
'0' ) );
signal d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNTO 0, reg_width - 1 DOWNTO 0 ) := ( others => ( others =>
'0' ) );

signal A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
signal ALU_result : std_logic_vector( reg_width - 1 downto 0 );

```

```

signal      MB_vector : std_logic_vector( 0 downto 0 );  -- MB constant/operand bus B bus multiplexer control signal
signal      MD_vector : std_logic_vector( 0 downto 0 );  -- external data/alu result multiplexer control signal

begin
  rfile: reg_file
  generic map ( nr_regs => nr_regs, reg_width => reg_width )
  port map (
    clk => clk, LE => RW, nRST => nRST, dest_select => DA,      A_select => AA, B_select => BA, D => D_data,
    A => A_data, B => B_data
  );
  Address_out  <= A_data;
  Data_out     <= B_muxed;

  muxb: muxnto1_bus
  generic map ( n_addr => 1, bus_width => reg_width )
  port map (
    s => MB_vector, w => b_mux_2d_bit_array, f => B_muxed
  );
  MB_vector( 0 )      <= MB;
  b_mux_vector_array( 0 ) <= B_data;
  b_mux_vector_array( 1 ) <= Const_in;

  muxd: muxnto1_bus
  generic map ( n_addr => 1, bus_width => reg_width )
  port map (
    s => MD_vector, w => d_mux_2d_bit_array, f => D_data
  );
  MD_vector( 0 )      <= MD;
  d_mux_vector_array( 0 ) <= ALU_result;
  d_mux_vector_array( 1 ) <= Data_in;

  arrays_to_2d: for i in 0 to reg_width-1 generate
    b_mux_2d_bit_array( 0, i )<= b_mux_vector_array( 0 )( i );
    b_mux_2d_bit_array( 1, i )<= b_mux_vector_array( 1 )( i );
    d_mux_2d_bit_array( 0, i )<= d_mux_vector_array( 0 )( i );
    d_mux_2d_bit_array( 1, i )<= d_mux_vector_array( 1 )( i );
  end generate;

  alu: ndn_alu

```



```
generic map ( n => reg_width )
port map (
    M => ALU_mode, F => ALU_function,      X => A_data, Y => B_muxed, S => ALU_result,  Negative =>
    ALU_N_bit, Cout => ALU_C_bit,  Overflow => ALU_V_bit, Zero => ALU_Z_bit
);

end ideal;
```

```

-- *****
-- **** STUDENT: 64240430
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.math_real.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;

entity cpu_datapath is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk,      -- clock input
           RW        : in std_logic;    -- register write input  ( active '1' )
           nRST      : in std_logic;    -- reset input          ( active '0' )
           DA,       -- destination register number select input
           AA,       -- A, B bus register number select input
           BA        : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
           MB        : in std_logic;    -- constant/operand bus B bus multiplexer control signal
           MD        : in 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

           ALU_mode          : in std_logic;    -- mode of alu operation ( M in upper table )

```

```

        ALU_function : in std_logic_vector( 2 downto 0 );    -- function of ALU ( F in upper table )
        ALU_N_bit    : out std_logic;    -- Negative bit of alu operation
        ALU_C_bit    : out std_logic;    -- Carry bit of alu operation
        ALU_V_bit    : out std_logic;    -- Overflow bit of alu operation
        ALU_Z_bit    : out std_logic;    -- Zero bit of alu operation
        Const_in     : in  std_logic_vector( reg_width - 1 downto 0 );    -- constant input
bus
        Data_in      : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus
input
        Address_out  : out std_logic_vector( reg_width - 1 downto 0 );    -- address bus output
        Data_out     : out std_logic_vector( reg_width - 1 downto 0 )    -- data bus output
    );
end cpu_datapath;

architecture ideal of cpu_datapath is
    -- 2/1 bus multiplexer input type definitions
    type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal b_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
    signal d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );

    signal b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );
    signal d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );

    signal A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
    signal ALU_result : std_logic_vector( reg_width - 1 downto 0 );

    signal MB_vector : std_logic_vector( 0 downto 0 );    -- MB constant/operand bus B bus multiplexer control signal
    signal MD_vector : std_logic_vector( 0 downto 0 );    -- external data/alu result multiplexer control signal

begin

    reg: reg_file
        generic map ( nr_regs => nr_regs,                reg_width => reg_width
        )
        port map ( clk => clk,                LE => RW,                nRST => nRST,                dest_select => DA,
        A_select => AA,                B_select => BA,                D => D_data,                A => A_data,
B => B_data
        );

```

```

muxnto1_bus_B: muxnto1_bus
    generic map( n_addr => 1,          bus_width => reg_width
    )
    port map ( s => MB_vector,          w => b_mux_2d_bit_array,          f => B_muxed
    );

alu: ndn_alu
    generic map ( n => reg_width )
    port map ( M => ALU_mode,          F => ALU_function,          X => A_data,          Y =>
    B_muxed,          S => ALU_result,          Negative => ALU_N_bit,          Zero => ALU_Z_bit,          Cout
    => ALU_C_bit,          Overflow => ALU_V_bit,          Gout => open,          Pout => open
    );

muxnto1_bus_D: muxnto1_bus
    generic map ( n_addr => 1,          bus_width => reg_width
    )
    port map ( s => MD_vector,          w => d_mux_2d_bit_array,          f => D_data
    );

Address_out <= A_data;
Data_out      <= B_muxed;

MB_vector( 0 )      <= MB;
b_mux_vector_array( 0 )      <= B_data;
b_mux_vector_array( 1 )      <= Const_in;

MD_vector( 0 )      <= MD;
d_mux_vector_array( 0 )      <= ALU_result;
d_mux_vector_array( 1 )      <= Data_in;

dva_d:
for i in 0 to reg_width - 1 generate
    b_mux_2d_bit_array( 0,i ) <= b_mux_vector_array( 0 )( i );
    b_mux_2d_bit_array( 1,i ) <= b_mux_vector_array( 1 )( i );

    d_mux_2d_bit_array( 0,i ) <= d_mux_vector_array( 0 )( i );
    d_mux_2d_bit_array( 1,i ) <= d_mux_vector_array( 1 )( i );

end generate;

```

```
end ideal;
```

```

-- *****
-- **** PREDLOGA VAJE
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
use ieee.math_real.all;

entity cpu_datapath is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk,      -- clock input
           RW        : in std_logic;    -- register write input  ( active '1' )
           nRST      : in std_logic;    -- reset input          ( active '0' )
           DA,       -- destination register number select input
           AA,       -- A, B bus register number select input
           BA        : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
           MB        : in std_logic;    -- constant/operand bus B bus multiplexer control signal
           MD        : in 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

           ALU_mode          : in std_logic;    -- mode of alu operation ( M in upper table )

```

```

        ALU_function : in std_logic_vector( 2 downto 0 );    -- function of ALU ( F in upper table )
        ALU_N_bit    : out std_logic;    -- Negative bit of alu operation
        ALU_C_bit    : out std_logic;    -- Carry bit of alu operation
        ALU_V_bit    : out std_logic;    -- Overflow bit of alu operation
        ALU_Z_bit    : out std_logic;    -- Zero bit of alu operation
        Const_in     : in  std_logic_vector( reg_width - 1 downto 0 );    -- constant input
bus
        Data_in      : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus
input
        Address_out  : out std_logic_vector( reg_width - 1 downto 0 );    -- address bus output
        Data_out     : out std_logic_vector( reg_width - 1 downto 0 )    -- data bus output
    );
end cpu_datapath;

architecture ideal of cpu_datapath is
    -- 2/1 bus multiplexer input type definitions
    type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal b_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
    signal d_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );

    signal b_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );
    signal d_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others => ( others =>
    '0' ) );

    signal A_data, B_data, B_muxed, D_data : std_logic_vector( reg_width - 1 downto 0 );
    signal ALU_result : std_logic_vector( reg_width - 1 downto 0 );

    signal MB_vector : std_logic_vector( 0 downto 0 );    -- MB constant/operand bus B bus multiplexer control signal
    signal MD_vector : std_logic_vector( 0 downto 0 );    -- external data/alu result multiplexer control signal

begin

REG_FILE1 : reg_file
    generic map( nr_regs    => nr_regs,          reg_width    => reg_width )
    PORT map (   clk => clk,    -- clock input
        LE    => RW, -- Load enable input      ( active '1' )
        nRST => nRST, -- reset input          ( active '0' )
        dest_select => DA, -- register number destination select input
        A_select => AA,    -- A, B bus destination select input
    );

```

```

        B_select => BA,          D => D_data,          A => A_data,          B => B_data  -- A,
B bus output
    );

    -- b bus multiplexer switches between constant/operand bus B
    b_mux_vector_array( 0 )    <= B_data;
    b_mux_vector_array( 1 )    <= Const_in;
    MB_vector( 0 )            <= MB; -- convert std_logic signal          to std_logic_vector( 0 )

    -- 2d bit array to 1d array of std_logic_vector elements conversion process
    b_mux_in_array_process: PROCESS( b_mux_vector_array, b_mux_2d_bit_array )
    variable b_mux_vector_col : std_logic_vector( reg_width - 1 downto 0 );
    BEGIN
        FOR i IN 0 TO 1 LOOP
            b_mux_vector_col := b_mux_vector_array( i );
            FOR j IN 0 TO reg_width - 1 LOOP
                b_mux_2d_bit_array( i, j ) <= b_mux_vector_col( j );
            END LOOP;
        END LOOP;
    END PROCESS;
    B_MUX: muxnto1_bus
        generic map (          n_addr=> 1,          bus_width => reg_width )
        PORT MAP (             s => MB_vector,          w => b_mux_2d_bit_array,          f => B_muxed
    );

    Address_out <= A_data;  -- external bus output
    Data_out    <= B_muxed;

    ALU: ndn_alu
        generic map (          n => reg_width )
        port map (             M => ALU_mode,          F => ALU_function,          X => A_data,          Y => B_muxed,
            S => ALU_result,          Negative => ALU_N_bit,          Cout => ALU_C_bit,          Overflow => ALU_V_bit,
            Zero => ALU_Z_bit
        );

    -- d bus multiplexer switches between external data/alu result
    d_mux_vector_array( 0 )    <= ALU_result;
    d_mux_vector_array( 1 )    <= Data_in;
    MD_vector( 0 )            <= MD; -- convert std_logic signal          to std_logic_vector( 0 )

```


-- 2d bit array to 1d array of std_logic_vector elements conversion process

d_mux_in_array_process: **PROCESS**(d_mux_vector_array, d_mux_2d_bit_array)

variable d_mux_vector_col : **std_logic_vector**(reg_width - 1 **downto** 0);

BEGIN

FOR i **IN** 0 **TO** 1 **LOOP**

 d_mux_vector_col := d_mux_vector_array(i);

FOR j **IN** 0 **TO** reg_width - 1 **LOOP**

 d_mux_2d_bit_array(i, j)<= d_mux_vector_col(j);

END LOOP;

END LOOP;

END PROCESS;

D_BUS_MUX: muxnto1_bus

generic map (

 n_addr=> 1,

 bus_width => reg_width)

PORT MAP (

 s => MD_vector,

 w => d_mux_2d_bit_array,

 f => D_data

);

end ideal;

