

-- **** STUDENT: 64000225.....	3
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	3
-- **** STUDENT: 64200100.....	7
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Rezultatov simulacije ni zaradi napak sintetizatorja: .....	7
ERROR:HDLCompiler:1314 - "cpu.vhd" Line 132: Formal port/generic <datain> is not declared in <cpu_datapath>.....	7
ERROR:HDLCompiler:432 - "cpu.vhd" Line 114: Formal <data_in> has no actual or default value. ....	7
Manjkajo vam vezaji v imenih signalov pri povezovanju cpu_datapath (Data_in, Address_out in Data_out). ....	7
-- **** STUDENT: 64200112.....	11
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Rezultatov simulacije ni zaradi napak sintetizatorja: .....	11
ERROR:HDLCompiler:1314 - "cpu.vhd" Line 83: Formal port/generic <ctrl_width> is not declared in <branch_ctrl> .....	11
ERROR:HDLCompiler:433 - "cpu.vhd" Line 117: Formal <data_in> is already associated. ....	11
Namesto (ctrl_width) bi moralo biti (ctr_width). V povezovalnem stavku imate dvakrat povezan (Data_in) signal: Data_in => Const_in, Data_in => IOBUS_Data_in. Povezava na Const_in je odveč. Manjka tudi izvedba signala IOBUS_address <= A_bus; Brez tega števec naslovov šteje kar naprej, kljub temu da bi moral na naslovu 5 izvesti pogoj vejitve (skok na naslov 9).	11
.....	11
-- **** STUDENT: 64200238.....	15
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	15
-- **** STUDENT: 64200288.....	20
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	20
-- **** STUDENT: 64200296.....	24
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	24
-- **** STUDENT: 64200385.....	28
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	28
-- **** STUDENT: 64210113.....	32
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	32
-- **** STUDENT: 64210290.....	36
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Rezultati simulacije sicer so, samo so vse vrednosti registrov v REG_FILE enaki nič. Podobno z RAM spominom. Vzrok sta napačno napisana signala za JB_address in Const_in: .....	36
--JB_address <= std_logic_vector(resize(signed(IR(8 downto 6)&IR(2 downto 0)), JB_address'length)); je pogojen z JB – če je JB='1' morate postaviti JB_address <= A_bus; (drugi del when-else stavka). Const_in je enak A_bus. ....	36
V povezovalnem stavku imate dvakrat povezan (Data_in) signal: Data_in => Const_in, Data_in => IOBUS_Data_in. Povezava na Const_in je odveč. Manjka tudi izvedba signala IOBUS_address <= A_bus; Brez tega števec naslovov šteje kar naprej, kljub temu da bi moral na naslovu 5 izvesti pogoj vejitve (skok na naslov 9).....	36
-- **** STUDENT: 64210382.....	42

-- KOMENTARJI K OCENI NALOGE	-- Matej Možek: Ni pripomb.....	42
-- **** STUDENT: 64210384.....		46
-- KOMENTARJI K OCENI NALOGE	-- Matej Možek: Ni pripomb.....	46
-- **** STUDENT: 64210386.....		50
-- KOMENTARJI K OCENI NALOGE	-- Matej Možek: Ni pripomb.....	50
-- **** STUDENT: 64210445.....		54
-- KOMENTARJI K OCENI NALOGE	-- Matej Možek: Ni pripomb.....	54
-- **** STUDENT: 64210455.....		58
-- KOMENTARJI K OCENI NALOGE	-- Matej Možek: Naloga ni programirana. ....	58
-- **** STUDENT: 64210457.....		59
-- KOMENTARJI K OCENI NALOGE	-- Matej Možek: Ni pripomb.....	59
-- **** STUDENT: 64240430.....		63
-- KOMENTARJI K OCENI NALOGE	-- Matej Možek: Ni pripomb.....	63
-- **** PREDLOGA VAJE .....		67
-- KOMENTARJI K OCENI NALOGE	-- Matej Mozek: Predloga vaje .....	67

```

-- *****
-- **** STUDENT: 64000225
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

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

-- opcode summary:
-- MB NOT( RW ) MD ALU_M ALU_F
-- 0 0 0 0 000 A plus B
-- 0 0 0 0 001 A minus B
-- 0 0 0 0 010 A plus 1
-- 0 0 0 0 011 A minus 1
-- 0 0 0 0 100 A plus A
-- 0 0 0 0 101 minus 1 ( 2' complement )
-- 0 0 0 1 000 A and B
-- 0 0 0 1 001 A nand B
-- 0 0 0 1 010 A or B
-- 0 0 0 1 011 A nor B
-- 0 0 0 1 100 A xor B
-- 0 0 0 1 101 A xnor B
-- 0 0 0 1 110 TEST A
-- 0 0 0 1 111 NOT IMPLEMENTED
-- 1 0 0 1 111 LDI ( Load immediate )
-- 1 0 0 0 000 ADI ( add immediate )
-- 1 0 0 0 001 SBI ( subtract immediate )
-- 0 0 1 1 110 LOAD A
-- 0 0 0 1 110 MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0 0 1 111 MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1 0 1 111 STORE B
-- JB BC
-- 1 1 0 1 11 0 BRZ R( SA )
-- 1 1 0 1 00 1 BRN R( SA )

```

```
-- 1 1          1 0          00 0 JMP
```

```
entity cpu is
```

```
    generic(
        nr_regs      : natural := 8;
        reg_width    : natural := 16;
        ram_nr_addr  : natural := 4;
        rom_nr_addr  : natural := 4
    );
    PORT (
        clk, -- clock input
        nRST      : in  std_logic; -- reset input      ( active '0' )
        IOBUS_WnR  : out std_logic; -- io bus write input  ( active '1' )
        IOBUS_Data_in  : in  std_logic_vector( reg_width - 1 downto 0 ); -- io bus data input
        IOBUS_Address: out std_logic_vector( reg_width - 1 downto 0 ); -- io address bus
        IOBUS_Data_out : out std_logic_vector( reg_width - 1 downto 0 ); -- io bus data output
        ProgMem_Addr : out std_logic_vector( reg_width - 1 downto 0 ); -- program memory address
        IR           : in  std_logic_vector( reg_width - 1 downto 0 ) -- program memory data
    );
```

```
input
```

```
end cpu;
```

```
architecture ideal of cpu is
```

```
    signal      MW           : std_logic; -- ram write input ( active '1' )

    signal      PL           : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL =
    '1' )

    signal      JB           : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with
    predefined value )

    signal      JB_address   : std_logic_vector( reg_width - 1 downto 0 );
    signal      BC           : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit )

    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

    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      PC              : std_logic_vector( reg_width - 1 downto 0 );
signal      A_bus           : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU

begin

DA    <= IR( 8 downto 6 );
AA    <= IR( 5 downto 3 );
BA    <= IR( 2 downto 0 );
MB    <= IR( 15 );
ALU_mode    <= IR( 12 );
ALU_function <= IR( 11 downto 10 ) & ( IR( 9 ) and ( not PL ) );
MD    <= IR( 13 );
RW    <= not IR( 14 );
MW    <= IR( 14 ) and ( not IR( 15 ) );
PL    <= IR( 14 ) and IR( 15 );
JB    <= IR( 13 );
BC    <= IR( 9 );

JB_address <= STD_LOGIC_VECTOR( resize( signed( IR( 8 downto 6 ) & IR( 2 downto 0 ) ), JB_address'length ) );
Const_in   <= STD_LOGIC_VECTOR( resize( unsigned( IR( 2 downto 0 ) ), Const_in'length ) );

bctrl: branch_ctrl
generic map ( ctr_width => reg_width )
port map (
    clk => clk, nRST => nRST, N => ALU_N_bit, C => ALU_C_bit, V => ALU_V_bit, Z => ALU_Z_bit, PL => PL, JB => JB,
    BC => BC, JB_address => JB_address, PC => PC
);

dpath: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
port map (
    clk => clk, RW => RW, nRST => nRST, DA => DA, AA => AA, BA => BA, MB => MB, MD =>
    MD, ALU_mode => ALU_mode, ALU_function => ALU_function, ALU_N_bit => ALU_N_bit, ALU_C_bit => ALU_C_bit,

```

```
    ALU_V_bit => ALU_V_bit,    ALU_Z_bit => ALU_Z_bit,    Const_in => Const_in,    Data_in => IOBUS_Data_in,  
    Address_out => IOBUS_Address,    Data_out => IOBUS_Data_out  
);
```

```
IOBUS_WnR    <= MW;  
ProgMem_Addr <= PC;
```

```
end ideal;
```

```

-- *****
-- **** STUDENT: 64200100
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Rezultatov simulacije ni zaradi napak sintetizatorja:
ERROR:HDLCompiler:1314 - "cpu.vhd" Line 132: Formal port/generic <datain> is not declared in <cpu_datapath>
ERROR:HDLCompiler:432 - "cpu.vhd" Line 114: Formal <data_in> has no actual or default value.
Manjkajo vam vezaji v imenih signalov pri povezovanju cpu_datapath (Data_in, Address_out in Data_out).
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE work.cpu_functions.all;
use ieee.math_real.all;

-- opcode summary:
-- MB NOT( RW ) MD ALU_M ALU_F
-- 0 0 0 0 000 A plus B
-- 0 0 0 0 001 A minus B
-- 0 0 0 0 010 A plus 1
-- 0 0 0 0 011 A minus 1
-- 0 0 0 0 100 A plus A
-- 0 0 0 0 101 minus 1 ( 2' complement )
-- 0 0 0 1 000 A and B
-- 0 0 0 1 001 A nand B
-- 0 0 0 1 010 A or B
-- 0 0 0 1 011 A nor B
-- 0 0 0 1 100 A xor B
-- 0 0 0 1 101 A xnor B
-- 0 0 0 1 110 TEST A
-- 0 0 0 1 111 NOT IMPLEMENTED
-- 1 0 0 1 111 LDI ( Load immediate )
-- 1 0 0 0 000 ADI ( add immediate )
-- 1 0 0 0 001 SBI ( subtract immediate )
-- 0 0 1 1 110 LOAD A
-- 0 0 0 1 110 MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0 0 1 111 MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1 0 1 111 STORE B

```

```

--
-- 1 1      JB      0      1      BC      11      0      BRZ R( SA )
-- 1 1      0      0      00      1      BRN R( SA )
-- 1 1      1      0      00      0      JMP

```

entity cpu is

```

    generic(
        nr_regs      : natural := 8;
        reg_width    : natural := 16;
        ram_nr_addr  : natural := 4;
        rom_nr_addr  : natural := 4
    );
    PORT (
        clk, -- clock input
        nRST      : in  std_logic; -- reset input      ( active '0' )
        IOBUS_WnR  : out std_logic; -- io bus write input  ( active '1' )
        IOBUS_Data_in  : in  std_logic_vector( reg_width - 1 downto 0 ); -- io bus data input
        IOBUS_Address: out std_logic_vector( reg_width - 1 downto 0 ); -- io address bus
        IOBUS_Data_out : out std_logic_vector( reg_width - 1 downto 0 ); -- io bus data output
        ProgMem_Addr  : out std_logic_vector( reg_width - 1 downto 0 ); -- program memory address
        IR            : in  std_logic_vector( reg_width - 1 downto 0 ) -- program memory data
    );

```

input

end cpu;

architecture ideal of cpu is

```

signal      MW      : std_logic; -- ram write input ( active '1' )

signal      PL      : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL =
'1' )
signal      JB      : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with
predefined value )
signal      JB_address : std_logic_vector( reg_width - 1 downto 0 );
signal      BC      : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit )

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

```



```

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      PC                 : std_logic_vector( reg_width - 1 downto 0 );
signal      A_bus              : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU

begin

DA    <= IR( 8 downto 6 );
AA    <= IR( 5 downto 3 );
BA    <= IR( 2 downto 0 );
MB    <= IR( 15 );
ALU_mode    <= IR( 12 );
ALU_function <= IR( 11 downto 10 ) & ( IR( 9 ) and ( not PL ) );
MD    <= IR( 13 );
RW    <= not IR( 14 );
MW    <= IR( 14 ) and ( not IR( 15 ) );
PL    <= IR( 14 ) and IR( 15 );
JB    <= IR( 13 );
BC    <= IR( 9 );
JB_address <= STD_LOGIC_VECTOR( resize( signed( IR( 8 downto 6 ) & IR( 2 downto 0 ) ), JB_address'length ) );
Const_in   <= STD_LOGIC_VECTOR( resize( unsigned( IR( 2 downto 0 ) ), Const_in'length ) );

bpot: branch_ctrl
generic map ( ctr_width => reg_width )
port map (
clk => clk,
nRST => nRST,
N => ALU_N_bit,
C => ALU_C_bit,
V => ALU_V_bit,
Z => ALU_Z_bit,
PL => PL,
JB => JB,

```

```
BC => BC,  
JB_address => JB_address,  
PC => PC );
```

```
dspot: cpu_datapath  
generic map( nr_regs => nr_regs, reg_width => reg_width )  
port map (  
clk => clk,  
RW => RW,  
nRST => nRST,  
DA => DA,  
AA => AA,  
BA => BA,  
MB => MB,  
MD => MD,  
ALU_mode => ALU_mode,  
ALU_function => ALU_function,  
ALU_N_bit => ALU_N_bit,  
ALU_C_bit => ALU_C_bit,  
ALU_V_bit => ALU_V_bit,  
ALU_Z_bit => ALU_Z_bit,  
Const_in => Const_in,  
DataIn => IOBUS_Data_in,  
AddressOut => IOBUS_Address,  
DataOut => IOBUS_Data_out );
```

```
IOBUS_WnR    <= MW;  
ProgMem_Addr <= PC;  
end ideal;
```

```

-- *****
-- **** STUDENT: 64200112
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Rezultatov simulacije ni zaradi napak sintetizatorja:
ERROR:HDLCompiler:1314 - "cpu.vhd" Line 83: Formal port/generic <ctrl_width> is not declared in <branch_ctrl>
ERROR:HDLCompiler:433 - "cpu.vhd" Line 117: Formal <data_in> is already associated.
Namesto (ctrl_width) bi moralo biti (ctr_width). V povezovalnem stavku imate dvakrat povezan (Data_in) signal: Data_in
=> Const_in, Data_in => IOBUS_Data_in. Povezava na Const_in je odveč. Manjka tudi izvedba signala IOBUS_address <=
A_bus; Brez tega števec naslovov šteje kar naprej, kljub temu da bi moral na naslovu 5 izvesti pogoj vejitve (skok na
naslov 9).
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE work.cpu_functions.all;
use ieee.math_real.all;

-- opcode summary:
-- MB NOT( RW ) MD ALU_M ALU_F
-- 0 0 0 0 000 A plus B
-- 0 0 0 0 001 A minus B
-- 0 0 0 0 010 A plus 1
-- 0 0 0 0 011 A minus 1
-- 0 0 0 0 100 A plus A
-- 0 0 0 0 101 minus 1 ( 2' complement )
-- 0 0 0 1 000 A and B
-- 0 0 0 1 001 A nand B
-- 0 0 0 1 010 A or B
-- 0 0 0 1 011 A nor B
-- 0 0 0 1 100 A xor B
-- 0 0 0 1 101 A xnor B
-- 0 0 0 1 110 TEST A
-- 0 0 0 1 111 NOT IMPLEMENTED
-- 1 0 0 1 111 LDI ( Load immediate )
-- 1 0 0 0 000 ADI ( add immediate )
-- 1 0 0 0 001 SBI ( subtract immediate )
-- 0 0 1 1 110 LOAD A

```

```

-- 0 0      0 1      110  MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0      0 1      111  MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1      0 1      111  STORE B
--          JB          BC
-- 1 1      0 1      11  0    BRZ R( SA )
-- 1 1      0 0      00  1    BRN R( SA )
-- 1 1      1 0      00  0    JMP

```

entity cpu is

```

    generic(
        nr_regs      : natural := 8;
        reg_width    : natural := 16;
        ram_nr_addr  : natural := 4;
        rom_nr_addr  : natural := 4
    );
    PORT (
        clk,          -- clock input
        nRST          : in  std_logic;  -- reset input      ( active '0' )
        IOBUS_WnR      : out std_logic;  -- io bus write input ( active '1' )
        IOBUS_Data_in  : in  std_logic_vector( reg_width - 1 downto 0 ); -- io bus data input
        IOBUS_Address : out std_logic_vector( reg_width - 1 downto 0 ); -- io address bus
        IOBUS_Data_out : out std_logic_vector( reg_width - 1 downto 0 ); -- io bus data output
        ProgMem_Addr  : out std_logic_vector( reg_width - 1 downto 0 ); -- program memory address
        IR            : in  std_logic_vector( reg_width - 1 downto 0 )  -- program memory data
    );

```

input

end cpu;

architecture ideal of cpu is

```

signal      MW          : std_logic; -- ram write input ( active '1' )

signal      PL          : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL =
'1' )
signal      JB          : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with
predefined value )
signal      JB_address  : std_logic_vector( reg_width - 1 downto 0 );
signal      BC          : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit )

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

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    PC                              : std_logic_vector( reg_width - 1 downto 0 );
signal    A_bus                            : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU
begin

    U0 : branch_ctrl
        generic map( ctrl_width => reg_width )
        port map( clk => clk, nRST => nRST, N => ALU_N_bit, C => ALU_C_bit,
V => ALU_V_bit, Z => ALU_Z_bit, PL => PL, JB => JB, BC => BC,
JB_address => JB_address, PC => PC
        );

    U1 : cpu_datapath
        generic map( nr_regs => nr_regs, reg_width => reg_width )
        port map( clk => clk, RW => RW, nRST => nRST, DA => DA, AA =>
AA, BA => BA, MB => MB, MD => MD, ALU_mode => ALU_mode, ALU_function
=> ALU_function, ALU_N_bit => ALU_N_bit, ALU_C_bit => ALU_C_bit, ALU_V_bit => ALU_V_bit,
ALU_Z_bit => ALU_Z_bit, Const_in => Const_in, Data_in => Const_in, Data_in =>
IOBUS_Data_in, Address_out => IOBUS_Address, Data_out => IOBUS_Data_out
        );

    MW    <= not IR( 15 ) and IR( 14 );
    PL    <= IR( 15 ) and IR( 14 );
    JB    <= IR( 13 );
    BC    <= IR( 9 );
    RW    <= not IR( 14 );
    DA    <= IR( 8 downto 6 );
    AA    <= IR( 5 downto 3 );
    BA    <= IR( 2 downto 0 );

```

```
MB    <= IR( 15 );  
MD    <= IR( 13 );  
ALU_mode    <= IR( 12 );  
ALU_function <= IR( 11 downto 10 ) & ( IR( 9 ) and not PL );
```

```
IOBUS_WnR    <= MW;
```

```
ProgMem_Addr <= PC;
```

```
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 work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE work.cpu_functions.all;
use ieee.math_real.all;

-- opcode summary:
-- MB NOT( RW ) MD ALU_M ALU_F
-- 0 0 0 0 000 A plus B
-- 0 0 0 0 001 A minus B
-- 0 0 0 0 010 A plus 1
-- 0 0 0 0 011 A minus 1
-- 0 0 0 0 100 A plus A
-- 0 0 0 0 101 minus 1 ( 2' complement )
-- 0 0 0 1 000 A and B
-- 0 0 0 1 001 A nand B
-- 0 0 0 1 010 A or B
-- 0 0 0 1 011 A nor B
-- 0 0 0 1 100 A xor B
-- 0 0 0 1 101 A xnor B
-- 0 0 0 1 110 TEST A
-- 0 0 0 1 111 NOT IMPLEMENTED
-- 1 0 0 1 111 LDI ( Load immediate )
-- 1 0 0 0 000 ADI ( add immediate )
-- 1 0 0 0 001 SBI ( subtract immediate )
-- 0 0 1 1 110 LOAD A
-- 0 0 0 1 110 MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0 0 1 111 MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1 0 1 111 STORE B
-- JB BC
-- 1 1 0 1 11 0 BRZ R( SA )
-- 1 1 0 1 00 1 BRN R( SA )

```

```
-- 1 1          1 0          00 0 JMP
```

```
entity cpu is
```

```
    generic(
        nr_regs      : natural := 8;
        reg_width    : natural := 16;
        ram_nr_addr  : natural := 4;
        rom_nr_addr  : natural := 4
    );
    PORT (
        clk, -- clock input
        nRST      : in  std_logic; -- reset input      ( active '0' )
        IOBUS_WnR  : out std_logic; -- io bus write input  ( active '1' )
        IOBUS_Data_in  : in  std_logic_vector( reg_width - 1 downto 0 ); -- io bus data input
        IOBUS_Address: out std_logic_vector( reg_width - 1 downto 0 ); -- io address bus
        IOBUS_Data_out : out std_logic_vector( reg_width - 1 downto 0 ); -- io bus data output
        ProgMem_Addr  : out std_logic_vector( reg_width - 1 downto 0 ); -- program memory address
        IR            : in  std_logic_vector( reg_width - 1 downto 0 ) -- program memory data
    );
```

```
input
```

```
end cpu;
```

```
architecture ideal of cpu is
```

```
    signal      MW          : std_logic; -- ram write input ( active '1' )

    signal      PL          : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL =
    '1' )

    signal      JB          : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with
    predefined value )

    signal      JB_address  : std_logic_vector( reg_width - 1 downto 0 );
    signal      BC          : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit )

    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

    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      PC              : std_logic_vector( reg_width - 1 downto 0 );
signal      A_bus           : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU

component 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 component;

component branch_ctrl is
    generic( ctr_width : natural := 16 );
    PORT ( clk,        -- clock input
          nRST,       -- reset input      ( active '0' )
          N,          -- negative bit from ALU operation
          C,          -- carry bit from ALU operation
          V,          -- overflow bit from ALU operation
          Z,          -- zero bit from ALU operation
          PL,         -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL = '1' )
          JB,         -- jump/branch when PL='1' ( jump => PL = '1', Load counter with predefined value )
          BC          : in  std_logic;    -- branch control ( when '0' -> check Z bit, when '1' check
N bit )

          JB_address   : in  std_logic_vector( ctr_width - 1 downto 0 );
          PC           : out std_logic_vector( ctr_width - 1 downto 0 )
    );
end component;

begin

DA    <= IR( 8 downto 6 );
AA    <= IR( 5 downto 3 );
BA    <= IR( 2 downto 0 );
MB    <= IR( 15 );
ALU_mode    <= IR( 12 );
ALU_function <= IR( 11 downto 10 ) & ( IR( 9 ) and ( not PL ) );
MD    <= IR( 13 );
RW    <= not IR( 14 );
MW    <= IR( 14 ) and ( not IR( 15 ) );
PL    <= IR( 14 ) and IR( 15 );

```

```
JB    <= IR( 13 );
BC    <= IR( 9 );
```

```
JB_address    <= STD_LOGIC_VECTOR( resize( signed( IR( 8 downto 6 ) & IR( 2 downto 0 ) ), JB_address'length ) );
Const_in      <= STD_LOGIC_VECTOR( resize( unsigned( IR( 2 downto 0 ) ), Const_in'length ) );
```

```
U1: branch_ctrl
generic map ( ctr_width => reg_width )
port map (
    clk => clk, nRST => nRST, N => ALU_N_bit, C => ALU_C_bit, V => ALU_V_bit, Z => ALU_Z_bit, PL => PL, JB => JB,
    BC => BC, JB_address => JB_address, PC => PC
);
```

```
U2: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
port map (
    clk => clk,  RW => RW,    nRST => nRST,    DA => DA,    AA => AA,    BA => BA,    MB => MB,    MD =>
MD,  ALU_mode => ALU_mode,    ALU_function => ALU_function,    ALU_N_bit => ALU_N_bit,    ALU_C_bit => ALU_C_bit,
    ALU_V_bit => ALU_V_bit,    ALU_Z_bit => ALU_Z_bit,    Const_in => Const_in,    Data_in => IOBUS_Data_in,
    Address_out => IOBUS_Address,    Data_out => IOBUS_Data_out
);
```

```
IOBUS_WnR    <= MW;
ProgMem_Addr <= PC;
```

```
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 work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE work.cpu_functions.all;
use ieee.math_real.all;

-- opcode summary:
-- MB NOT( RW ) MD ALU_M ALU_F
-- 0 0 0 0 000 A plus B
-- 0 0 0 0 001 A minus B
-- 0 0 0 0 010 A plus 1
-- 0 0 0 0 011 A minus 1
-- 0 0 0 0 100 A plus A
-- 0 0 0 0 101 minus 1 ( 2' complement )
-- 0 0 0 1 000 A and B
-- 0 0 0 1 001 A nand B
-- 0 0 0 1 010 A or B
-- 0 0 0 1 011 A nor B
-- 0 0 0 1 100 A xor B
-- 0 0 0 1 101 A xnor B
-- 0 0 0 1 110 TEST A
-- 0 0 0 1 111 NOT IMPLEMENTED
-- 1 0 0 1 111 LDI ( Load immediate )
-- 1 0 0 0 000 ADI ( add immediate )
-- 1 0 0 0 001 SBI ( subtract immediate )
-- 0 0 1 1 110 LOAD A
-- 0 0 0 1 110 MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0 0 1 111 MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1 0 1 111 STORE B
-- JB BC
-- 1 1 0 1 11 0 BRZ R( SA )
-- 1 1 0 1 00 1 BRN R( SA )

```

```
-- 1 1          1 0          00 0 JMP
```

```
entity cpu is
```

```
    generic(
        nr_regs      : natural := 8;
        reg_width    : natural := 16;
        ram_nr_addr  : natural := 4;
        rom_nr_addr  : natural := 4
    );
    PORT (
        clk, -- clock input
        nRST      : in  std_logic; -- reset input      ( active '0' )
        IOBUS_WnR  : out std_logic; -- io bus write input  ( active '1' )
        IOBUS_Data_in  : in  std_logic_vector( reg_width - 1 downto 0 ); -- io bus data input
        IOBUS_Address: out std_logic_vector( reg_width - 1 downto 0 ); -- io address bus
        IOBUS_Data_out : out std_logic_vector( reg_width - 1 downto 0 ); -- io bus data output
        ProgMem_Addr : out std_logic_vector( reg_width - 1 downto 0 ); -- program memory address
        IR           : in  std_logic_vector( reg_width - 1 downto 0 ) -- program memory data
    );
```

```
input
```

```
end cpu;
```

```
architecture ideal of cpu is
```

```
    signal      MW           : std_logic; -- ram write input ( active '1' )

    signal      PL           : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL =
    '1' )

    signal      JB           : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with
    predefined value )

    signal      JB_address   : std_logic_vector( reg_width - 1 downto 0 );
    signal      BC           : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit )

    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

    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      PC              : std_logic_vector( reg_width - 1 downto 0 );
signal      A_bus           : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU

begin

Const_in    <= STD_LOGIC_VECTOR( resize( unsigned( IR( 2 downto 0 ) ), Const_in'length ) );
JB_address  <= STD_LOGIC_VECTOR( resize( signed( IR( 8 downto 6 ) & IR( 2 downto 0 ) ), JB_address'length ) );

MB    <= IR( 15 );
AA    <= IR( 5 downto 3 );
BA    <= IR( 2 downto 0 );
DA    <= IR( 8 downto 6 );
ALU_mode    <= IR( 12 );
ALU_function <= IR( 11 downto 10 ) & ( IR( 9 ) and ( not PL ) );
PL    <= IR( 14 ) and IR( 15 );
JB    <= IR( 13 );
BC    <= IR( 9 );
MD    <= IR( 13 );
RW    <= not IR( 14 );
MW    <= IR( 14 ) and ( not IR( 15 ) );

prtmp1: branch_ctrl
generic map ( ctr_width => reg_width )
port map (
    clk => clk, nRST => nRST, N => ALU_N_bit, C => ALU_C_bit, V => ALU_V_bit, Z => ALU_Z_bit, PL => PL, JB => JB,
    BC => BC, JB_address => JB_address, PC => PC
);

prtmp2: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
port map (
    clk => clk, RW => RW, nRST => nRST, DA => DA, AA => AA, BA => BA, MB => MB, MD =>
MD, ALU_mode => ALU_mode, ALU_function => ALU_function, ALU_N_bit => ALU_N_bit, ALU_C_bit => ALU_C_bit,

```

```
    ALU_V_bit => ALU_V_bit,    ALU_Z_bit => ALU_Z_bit,    Const_in => Const_in,    Data_in => IOBUS_Data_in,  
    Address_out => IOBUS_Address,    Data_out => IOBUS_Data_out  
);
```

```
IOBUS_WnR    <= MW;  
ProgMem_Addr <= PC;
```

```
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 work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE work.cpu_functions.all;
use ieee.math_real.all;

-- opcode summary:
-- MB NOT( RW ) MD ALU_M ALU_F
-- 0 0 0 0 000 A plus B
-- 0 0 0 0 001 A minus B
-- 0 0 0 0 010 A plus 1
-- 0 0 0 0 011 A minus 1
-- 0 0 0 0 100 A plus A
-- 0 0 0 0 101 minus 1 ( 2' complement )
-- 0 0 0 1 000 A and B
-- 0 0 0 1 001 A nand B
-- 0 0 0 1 010 A or B
-- 0 0 0 1 011 A nor B
-- 0 0 0 1 100 A xor B
-- 0 0 0 1 101 A xnor B
-- 0 0 0 1 110 TEST A
-- 0 0 0 1 111 NOT IMPLEMENTED
-- 1 0 0 1 111 LDI ( Load immediate )
-- 1 0 0 0 000 ADI ( add immediate )
-- 1 0 0 0 001 SBI ( subtract immediate )
-- 0 0 1 1 110 LOAD A
-- 0 0 0 1 110 MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0 0 1 111 MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1 0 1 111 STORE B
-- JB BC
-- 1 1 0 1 11 0 BRZ R( SA )
-- 1 1 0 1 00 1 BRN R( SA )

```



```
-- 1 1          1 0          00 0 JMP
```

```
entity cpu is
```

```
    generic(
        nr_regs      : natural := 8;
        reg_width     : natural := 16;
        ram_nr_addr   : natural := 4;
        rom_nr_addr   : natural := 4
    );
    PORT (
        clk, -- clock input
        nRST      : in  std_logic; -- reset input      ( active '0' )
        IOBUS_WnR  : out std_logic; -- io bus write input  ( active '1' )
        IOBUS_Data_in  : in  std_logic_vector( reg_width - 1 downto 0 ); -- io bus data input
        IOBUS_Address: out std_logic_vector( reg_width - 1 downto 0 ); -- io address bus
        IOBUS_Data_out : out std_logic_vector( reg_width - 1 downto 0 ); -- io bus data output
        ProgMem_Addr : out std_logic_vector( reg_width - 1 downto 0 ); -- program memory address
        IR           : in  std_logic_vector( reg_width - 1 downto 0 ) -- program memory data
    );
```

```
input
```

```
end cpu;
```

```
architecture ideal of cpu is
```

```
    signal      MW           : std_logic; -- ram write input ( active '1' )

    signal      PL           : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL =
    '1' )

    signal      JB           : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with
    predefined value )

    signal      JB_address   : std_logic_vector( reg_width - 1 downto 0 );
    signal      BC           : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit )

    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

    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      PC                  : std_logic_vector( reg_width - 1 downto 0 );
signal      A_bus               : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU

begin

branchctrl: branch_ctrl
generic map ( ctr_width => reg_width )
port map (
    clk => clk, nRST => nRST, N => ALU_N_bit, C => ALU_C_bit, V => ALU_V_bit, Z => ALU_Z_bit, PL => PL, JB => JB,
    BC => BC, JB_address => JB_address, PC => PC
);

datapath: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
port map (
    clk => clk, RW => RW, nRST => nRST, DA => DA, AA => AA, BA => BA, MB => MB, MD =>
MD, ALU_mode => ALU_mode, ALU_function => ALU_function, ALU_N_bit => ALU_N_bit, ALU_C_bit => ALU_C_bit,
    ALU_V_bit => ALU_V_bit, ALU_Z_bit => ALU_Z_bit, Const_in => Const_in, Data_in => IOBUS_Data_in,
    Address_out => IOBUS_Address, Data_out => IOBUS_Data_out
);

BA    <= IR( 2 downto 0 );
AA    <= IR( 5 downto 3 );
DA    <= IR( 8 downto 6 );
BC    <= IR( 9 );
ALU_function <= IR( 11 downto 10 ) & ( IR( 9 ) and ( not PL ) );
ALU_mode    <= IR( 12 );
MD    <= IR( 13 );
JB    <= IR( 13 );
RW    <= not IR( 14 );
MW    <= IR( 14 ) and ( not IR( 15 ) );
PL    <= IR( 14 ) and IR( 15 );
MB    <= IR( 15 );
JB_address <= STD_LOGIC_VECTOR( resize( signed( IR( 8 downto 6 ) & IR( 2 downto 0 ) ), JB_address'length ) );

```

```
Const_in      <= STD_LOGIC_VECTOR( resize( unsigned( IR( 2 downto 0 ) ), Const_in'length ) );
IOBUS_WnR     <= MW;
ProgMem_Addr  <= PC;
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 work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE work.cpu_functions.all;
use ieee.math_real.all;

-- opcode summary:
-- MB NOT( RW ) MD ALU_M ALU_F
-- 0 0 0 0 000 A plus B
-- 0 0 0 0 001 A minus B
-- 0 0 0 0 010 A plus 1
-- 0 0 0 0 011 A minus 1
-- 0 0 0 0 100 A plus A
-- 0 0 0 0 101 minus 1 ( 2' complement )
-- 0 0 0 1 000 A and B
-- 0 0 0 1 001 A nand B
-- 0 0 0 1 010 A or B
-- 0 0 0 1 011 A nor B
-- 0 0 0 1 100 A xor B
-- 0 0 0 1 101 A xnor B
-- 0 0 0 1 110 TEST A
-- 0 0 0 1 111 NOT IMPLEMENTED
-- 1 0 0 1 111 LDI ( Load immediate )
-- 1 0 0 0 000 ADI ( add immediate )
-- 1 0 0 0 001 SBI ( subtract immediate )
-- 0 0 1 1 110 LOAD A
-- 0 0 0 1 110 MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0 0 1 111 MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1 0 1 111 STORE B
-- JB BC
-- 1 1 0 1 11 0 BRZ R( SA )
-- 1 1 0 1 00 1 BRN R( SA )

```

```
-- 1 1          1 0          00 0 JMP
```

```
entity cpu is
```

```
    generic(
        nr_regs      : natural := 8;
        reg_width     : natural := 16;
        ram_nr_addr   : natural := 4;
        rom_nr_addr   : natural := 4
    );
    PORT (
        clk, -- clock input
        nRST      : in  std_logic; -- reset input      ( active '0' )
        IOBUS_WnR  : out std_logic; -- io bus write input  ( active '1' )
        IOBUS_Data_in  : in  std_logic_vector( reg_width - 1 downto 0 ); -- io bus data input
        IOBUS_Address: out std_logic_vector( reg_width - 1 downto 0 ); -- io address bus
        IOBUS_Data_out : out std_logic_vector( reg_width - 1 downto 0 ); -- io bus data output
        ProgMem_Addr : out std_logic_vector( reg_width - 1 downto 0 ); -- program memory address
        IR           : in  std_logic_vector( reg_width - 1 downto 0 ) -- program memory data
    );
```

```
input
```

```
end cpu;
```

```
architecture ideal of cpu is
```

```
    signal      MW           : std_logic; -- ram write input ( active '1' )

    signal      PL           : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL =
    '1' )

    signal      JB           : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with
    predefined value )

    signal      JB_address   : std_logic_vector( reg_width - 1 downto 0 );
    signal      BC           : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit )

    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

    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      PC              : std_logic_vector( reg_width - 1 downto 0 );
signal      A_bus           : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU

begin

DA    <= IR( 8 downto 6 );
AA    <= IR( 5 downto 3 );
BA    <= IR( 2 downto 0 );
MB    <= IR( 15 );
ALU_mode    <= IR( 12 );
ALU_function <= IR( 11 downto 10 ) & ( IR( 9 ) and ( not PL ) );
MD    <= IR( 13 );
RW    <= not IR( 14 );
MW    <= IR( 14 ) and ( not IR( 15 ) );
PL    <= IR( 14 ) and IR( 15 );
JB    <= IR( 13 );
BC    <= IR( 9 );

JB_address <= std_logic_vector( resize( signed( IR( 8 downto 6 ) & IR( 2 downto 0 ) ), JB_address'length ) );
Const_in   <= std_logic_vector( resize( unsigned( IR( 2 downto 0 ) ), Const_in'length ) );

ProgMem_Addr <= PC;
IOBUS_WnR    <= MW;

U0: branch_ctrl
generic map( ctr_width => reg_width )
port map( clk => clk,
nRST => nRST,
N => ALU_N_bit,
C => ALU_C_bit,
V => ALU_V_bit,
Z => ALU_Z_bit,
PL => PL,
JB => JB,

```

```
BC => BC,  
JB_address => JB_address,  
PC => PC );
```

```
U1: cpu_datapath  
generic map( nr_regs => nr_regs,  
  reg_width => reg_width )  
port map( clk => clk,  
  RW => RW,  
  nRST => nRST,  
  DA => DA,  
  AA => AA,  
  BA => BA,  
  MB => MB,  
  MD => MD,  
  ALU_mode => ALU_mode,  
  ALU_function => ALU_function,  
  ALU_N_bit => ALU_N_bit,  
  ALU_C_bit => ALU_C_bit,  
  ALU_V_bit => ALU_V_bit,  
  ALU_Z_bit => ALU_Z_bit,  
  Const_in => Const_in,  
  Data_in => IOBUS_Data_in,  
  Address_out => IOBUS_Address,  
  Data_out => IOBUS_Data_out );  
  
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 work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE work.cpu_functions.all;
use ieee.math_real.all;

-- opcode summary:
-- MB NOT( RW ) MD ALU_M ALU_F
-- 0 0 0 0 000 A plus B
-- 0 0 0 0 001 A minus B
-- 0 0 0 0 010 A plus 1
-- 0 0 0 0 011 A minus 1
-- 0 0 0 0 100 A plus A
-- 0 0 0 0 101 minus 1 ( 2' complement )
-- 0 0 0 1 000 A and B
-- 0 0 0 1 001 A nand B
-- 0 0 0 1 010 A or B
-- 0 0 0 1 011 A nor B
-- 0 0 0 1 100 A xor B
-- 0 0 0 1 101 A xnor B
-- 0 0 0 1 110 TEST A
-- 0 0 0 1 111 NOT IMPLEMENTED
-- 1 0 0 1 111 LDI ( Load immediate )
-- 1 0 0 0 000 ADI ( add immediate )
-- 1 0 0 0 001 SBI ( subtract immediate )
-- 0 0 1 1 110 LOAD A
-- 0 0 0 1 110 MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0 0 1 111 MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1 0 1 111 STORE B
-- JB BC
-- 1 1 0 1 11 0 BRZ R( SA )
-- 1 1 0 1 00 1 BRN R( SA )

```



```
-- 1 1          1 0          00 0 JMP
```

```
entity cpu is
```

```
    generic(
        nr_regs      : natural := 8;
        reg_width    : natural := 16;
        ram_nr_addr   : natural := 4;
        rom_nr_addr   : natural := 4
    );
    PORT (
        clk, -- clock input
        nRST      : in  std_logic; -- reset input ( active '0' )
        IOBUS_WnR  : out std_logic; -- io bus write input ( active '1' )
        IOBUS_Data_in : in  std_logic_vector( reg_width - 1 downto 0 ); -- io bus data input
        IOBUS_Address: out std_logic_vector( reg_width - 1 downto 0 ); -- io address bus
        IOBUS_Data_out : out std_logic_vector( reg_width - 1 downto 0 ); -- io bus data output
        ProgMem_Addr : out std_logic_vector( reg_width - 1 downto 0 ); -- program memory address
        IR           : in  std_logic_vector( reg_width - 1 downto 0 ) -- program memory data
    );
```

```
input
```

```
end cpu;
```

```
architecture ideal of cpu is
```

```
    signal      MW           : std_logic; -- ram write input ( active '1' )

    signal      PL           : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL =
    '1' )

    signal      JB           : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with
    predefined value )

    signal      JB_address   : std_logic_vector( reg_width - 1 downto 0 );
    signal      BC           : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit )

    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

    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      PC              : std_logic_vector( reg_width - 1 downto 0 );
signal      A_bus           : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU

begin

    MW      <= IR( 14 ) and ( not IR( 15 ) );
    PL      <= IR( 14 ) and IR( 15 );
    JB      <= IR( 13 );
    BC      <= IR( 9 );
    RW      <= not IR( 14 );
    DA      <= IR( 8 downto 6 );
    AA      <= IR( 5 downto 3 );
    BA      <= IR( 2 downto 0 );
    MB      <= IR( 15 );
    MD      <= IR( 13 );
    ALU_mode <= IR( 12 );
    ALU_function <= IR( 11 downto 10 ) & ( ( not PL ) and IR( 9 ) );

    JB_address <= STD_LOGIC_VECTOR( resize( signed( IR( 8 downto 6 ) & IR( 2 downto 0 ) ), JB_address'length ) );
    Const_in   <= STD_LOGIC_VECTOR( resize( unsigned( IR( 2 downto 0 ) ), Const_in'length ) );

    brnchctrl: branch_ctrl
    generic map ( ctr_width => reg_width )
    port map (
        clk => clk,          nRST => nRST,          PL => PL,          JB => JB,          BC => BC,
        JB_address => JB_address, PC => PC,          N => ALU_N_bit,    C => ALU_C_bit,    V =>
        ALU_V_bit,          Z => ALU_Z_bit
    );

    Path: cpu_datapath
    generic map( nr_regs => nr_regs, reg_width => reg_width )
    port map (
        clk => clk,          nRST => nRST,          RW => RW,          DA => DA,          AA => AA,
        BA => BA,          MB => MB,          MD => MD,          Const_in => Const_in,          Data_in =>

```

```
IOBUS_Data_in,          Address_out => IOBUS_Address,          ALU_N_bit => ALU_N_bit,          ALU_C_bit =>
ALU_C_bit,          ALU_V_bit => ALU_V_bit,          ALU_Z_bit => ALU_Z_bit,          ALU_mode => ALU_mode,
          ALU_function => ALU_function,          Data_out => IOBUS_Data_out );

          IOBUS_WnR      <= MW;
          ProgMem_Addr <= PC;

end ideal;
```

```

-- *****
-- **** STUDENT: 64210290
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Rezultati simulacije sicer so, samo so vse vrednosti registrov v REG_FILE enaki nič. Podobno z RAM
spominom. Vzrok sta napačno napisana signala za JB_address in Const_in:
--JB_address <= std_logic_vector(resize(signed(IR(8 downto 6)&IR(2 downto 0)), JB_address'length)); je pogojen z JB –
če je JB='1' morate postaviti JB_address <= A_bus; (drugi del when-else stavka). Const_in je enak A_bus.
V povezovalnem stavku imate dvakrat povezan (Data_in) signal: Data_in => Const_in, Data_in => IOBUS_Data_in. Povezava
na Const_in je odveč. Manjka tudi izvedba signala IOBUS_address <= A_bus; Brez tega števec naslovov šteje kar naprej,
kljub temu da bi moral na naslovu 5 izvesti pogoj vejitve (skok na naslov 9).
-- *****
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.all;
USE work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE work.cpu_functions.all;
USE ieee.math_real.all;

-- opcode summary:
-- MB NOT( RW ) MD ALU_M  ALU_F
-- 0  0          0      0      000  A plus B
-- 0  0          0      0      001  A minus B
-- 0  0          0      0      010  A plus 1
-- 0  0          0      0      011  A minus 1
-- 0  0          0      0      100  A plus A
-- 0  0          0      0      101  minus 1 ( 2' complement )
-- 0  0          0      1      000  A and B
-- 0  0          0      1      001  A nand B
-- 0  0          0      1      010  A or B
-- 0  0          0      1      011  A nor B
-- 0  0          0      1      100  A xor B
-- 0  0          0      1      101  A xnor B
-- 0  0          0      1      110  TEST A
-- 0  0          0      1      111  NOT IMPLEMENTED
-- 1  0          0      1      111  LDI ( Load immediate )
-- 1  0          0      0      000  ADI ( add immediate )
-- 1  0          0      0      001  SBI ( subtract immediate )

```

```

-- 0 0          1 1          110 LOAD A
-- 0 0          0 1          110 MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0          0 1          111 MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1          0 1          111 STORE B
--                JB                BC
-- 1 1          0 1          11 0 BRZ R( SA )
-- 1 1          0 0          00 1 BRN R( SA )
-- 1 1          1 0          00 0 JMP

```

```

ENTITY cpu IS
  GENERIC(
    nr_regs      : NATURAL := 8;
    reg_width    : NATURAL := 16;
    ram_nr_addr  : NATURAL := 4;
    rom_nr_addr  : NATURAL := 4
  );
  PORT(
    clk,  -- clock input
    nRST  : IN  STD_LOGIC;  -- reset input      ( active '0' )
    IOBUS_WnR : OUT STD_LOGIC;  -- io bus write input    ( active '1' )
    IOBUS_Data_in : IN  STD_LOGIC_VECTOR( reg_width - 1 DOWNT0 0 );  -- io bus data input
    IOBUS_Address: OUT STD_LOGIC_VECTOR( reg_width - 1 DOWNT0 0 );  -- io address bus
    IOBUS_Data_out : OUT STD_LOGIC_VECTOR( reg_width - 1 DOWNT0 0 );  -- io bus data output
    ProgMem_Addr : OUT STD_LOGIC_VECTOR( reg_width - 1 DOWNT0 0 );  -- program memory address
    IR          : IN  STD_LOGIC_VECTOR( reg_width - 1 DOWNT0 0 )  -- program memory data input
  );
END cpu;

```

```

ARCHITECTURE ideal OF cpu IS

```

```

  component cpu_datapath is
    generic(
      nr_regs : natural;
      reg_width : natural
    );
    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 component;

component branch_ctrl is
    generic(
        ctr_width : natural
    );
    port(
        clk,    -- clock input
        nRST,   -- reset input      ( active '0' )

```

```

N,      -- negative bit from ALU operation
C,      -- carry bit from ALU operation
V,      -- overflow bit from ALU operation
Z,      -- zero bit from ALU operation
PL,     -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL = '1' )
JB,     -- jump/branch when PL='1' ( jump => PL = '1', Load counter with predefined value )
BC      :      in      std_logic;  -- branch control ( when '0' -> check Z bit,
when '1' check N bit )
JB_address :      in      std_logic_vector( ctr_width - 1 downto 0 );
PC         :      out     std_logic_vector( ctr_width - 1 downto 0 )
    );
end component;

signal      MW      : std_logic; -- ram write input ( active '1' )

signal      PL      : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active (
PL = '1' )
signal      JB      : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with
predefined value )
signal      JB_address : std_logic_vector( reg_width - 1 downto 0 );
signal      BC      : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit
)

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

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      PC            : std_logic_vector( reg_width - 1 downto 0 );
signal      A_bus         : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU

```

BEGIN

```
IOBUS_WnR    <= MW;

DA    <= IR( 8 downto 6 );
AA    <= IR( 5 downto 3 );
BA    <= IR( 2 downto 0 );
MB    <= IR( 15 );
ALU_mode    <= IR( 12 );
ALU_function <= IR( 11 downto 10 ) & ( IR( 9 ) and not( PL ) );
MD    <= IR( 13 );
RW    <= not IR( 14 );
MW    <= IR( 14 ) and not( IR( 15 ) );
PL    <= IR( 15 ) and IR( 14 );
JB    <= IR( 13 );
BC    <= IR( 9 );

JB_address    <= std_logic_vector( resize( signed( IR( 8 downto 6 ) & IR( 2 downto 0 ) ), JB_address'length ) );
Const_in      <= std_logic_vector( resize( unsigned( IR( 2 downto 0 ) ), Const_in'length ) );

datapath: cpu_datapath
    generic map(
        nr_regs => nr_regs,          reg_width => reg_width
    )
    port map(
        clk => clk,          RW => RW,          nRST => nRST,          DA => DA,          AA => AA,
        BA => BA,          MB => MB,          MD => MD,          ALU_mode => ALU_mode,          ALU_function =>
        ALU_function,          ALU_N_bit => ALU_N_bit,          ALU_C_bit => ALU_C_bit,          ALU_V_bit => ALU_V_bit,
        ALU_Z_bit => ALU_Z_bit,          Const_in => Const_in,          Data_in => A_bus,          Address_out =>
        IOBUS_Address,          Data_out => IOBUS_Data_out
    );

Const_in    <= A_bus;
IOBUS_Address <= A_bus;

branch_controller : branch_ctrl
    generic map(
        ctr_width => reg_width
    )
```



```
port map(  
    clk => clk,          nRST => nRST,          N => ALU_N_bit,          C => ALU_C_bit,  
    V => ALU_V_bit,      Z => ALU_Z_bit,      PL => PL,          JB => JB,          BC => BC,  
    JB_address => JB_address,      PC => PC  
);
```

```
ProgMem_Addr <= PC;
```

```
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 work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE work.cpu_functions.all;
use ieee.math_real.all;

-- opcode summary:
-- MB NOT( RW ) MD ALU_M ALU_F
-- 0 0 0 0 000 A plus B
-- 0 0 0 0 001 A minus B
-- 0 0 0 0 010 A plus 1
-- 0 0 0 0 011 A minus 1
-- 0 0 0 0 100 A plus A
-- 0 0 0 0 101 minus 1 ( 2' complement )
-- 0 0 0 1 000 A and B
-- 0 0 0 1 001 A nand B
-- 0 0 0 1 010 A or B
-- 0 0 0 1 011 A nor B
-- 0 0 0 1 100 A xor B
-- 0 0 0 1 101 A xnor B
-- 0 0 0 1 110 TEST A
-- 0 0 0 1 111 NOT IMPLEMENTED
-- 1 0 0 1 111 LDI ( Load immediate )
-- 1 0 0 0 000 ADI ( add immediate )
-- 1 0 0 0 001 SBI ( subtract immediate )
-- 0 0 1 1 110 LOAD A
-- 0 0 0 1 110 MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0 0 1 111 MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1 0 1 111 STORE B
-- JB BC
-- 1 1 0 1 11 0 BRZ R( SA )
-- 1 1 0 1 00 1 BRN R( SA )

```

```
-- 1 1          1 0          00 0 JMP
```

```
entity cpu is
```

```
    generic(
        nr_regs      : natural := 8;
        reg_width    : natural := 16;
        ram_nr_addr  : natural := 4;
        rom_nr_addr  : natural := 4
    );
    PORT (
        clk, -- clock input
        nRST      : in  std_logic; -- reset input      ( active '0' )
        IOBUS_WnR  : out std_logic; -- io bus write input  ( active '1' )
        IOBUS_Data_in  : in  std_logic_vector( reg_width - 1 downto 0 ); -- io bus data input
        IOBUS_Address: out std_logic_vector( reg_width - 1 downto 0 ); -- io address bus
        IOBUS_Data_out : out std_logic_vector( reg_width - 1 downto 0 ); -- io bus data output
        ProgMem_Addr : out std_logic_vector( reg_width - 1 downto 0 ); -- program memory address
        IR           : in  std_logic_vector( reg_width - 1 downto 0 ) -- program memory data
    );
```

```
input
```

```
end cpu;
```

```
architecture ideal of cpu is
```

```
    signal      MW           : std_logic; -- ram write input ( active '1' )

    signal      PL           : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL =
    '1' )

    signal      JB           : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with
    predefined value )

    signal      JB_address   : std_logic_vector( reg_width - 1 downto 0 );
    signal      BC           : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit )

    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

    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      PC              : std_logic_vector( reg_width - 1 downto 0 );
signal      A_bus           : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU
begin
DA    <= IR( 8 downto 6 );
AA    <= IR( 5 downto 3 );
BA    <= IR( 2 downto 0 );
MB    <= IR( 15 );
ALU_mode    <= IR( 12 );
ALU_function <= IR( 11 downto 10 ) & ( IR( 9 ) and ( not PL ) );
MD    <= IR( 13 );
RW    <= not IR( 14 );
MW    <= IR( 14 ) and ( not IR( 15 ) );
PL    <= IR( 14 ) and IR( 15 );
JB    <= IR( 13 );
BC    <= IR( 9 );

JB_address <= STD_LOGIC_VECTOR( resize( signed( IR( 8 downto 6 ) & IR( 2 downto 0 ) ), JB_address'length ) );
Const_in   <= STD_LOGIC_VECTOR( resize( unsigned( IR( 2 downto 0 ) ), Const_in'length ) );

bctrl: branch_ctrl
generic map ( ctr_width => reg_width )
port map (
    clk => clk, nRST => nRST, N => ALU_N_bit, C => ALU_C_bit, V => ALU_V_bit, Z => ALU_Z_bit, PL => PL, JB => JB,
    BC => BC, JB_address => JB_address, PC => PC
);

dpath: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
port map (
    clk => clk, RW => RW, nRST => nRST, DA => DA, AA => AA, BA => BA, MB => MB, MD =>
MD, ALU_mode => ALU_mode, ALU_function => ALU_function, ALU_N_bit => ALU_N_bit, ALU_C_bit => ALU_C_bit,
    ALU_V_bit => ALU_V_bit, ALU_Z_bit => ALU_Z_bit, Const_in => Const_in, Data_in => IOBUS_Data_in,
    Address_out => IOBUS_Address, Data_out => IOBUS_Data_out
);

```

```
IOBUS_WnR      <= MW;  
ProgMem_Addr <= PC;  
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 work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE work.cpu_functions.all;
use ieee.math_real.all;

-- opcode summary:
-- MB NOT( RW ) MD ALU_M ALU_F
-- 0 0 0 0 000 A plus B
-- 0 0 0 0 001 A minus B
-- 0 0 0 0 010 A plus 1
-- 0 0 0 0 011 A minus 1
-- 0 0 0 0 100 A plus A
-- 0 0 0 0 101 minus 1 ( 2' complement )
-- 0 0 0 1 000 A and B
-- 0 0 0 1 001 A nand B
-- 0 0 0 1 010 A or B
-- 0 0 0 1 011 A nor B
-- 0 0 0 1 100 A xor B
-- 0 0 0 1 101 A xnor B
-- 0 0 0 1 110 TEST A
-- 0 0 0 1 111 NOT IMPLEMENTED
-- 1 0 0 1 111 LDI ( Load immediate )
-- 1 0 0 0 000 ADI ( add immediate )
-- 1 0 0 0 001 SBI ( subtract immediate )
-- 0 0 1 1 110 LOAD A
-- 0 0 0 1 110 MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0 0 1 111 MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1 0 1 111 STORE B
-- JB BC
-- 1 1 0 1 11 0 BRZ R( SA )
-- 1 1 0 1 00 1 BRN R( SA )

```

```
-- 1 1          1 0          00 0 JMP
```

```
entity cpu is
```

```
    generic(
        nr_regs      : natural := 8;
        reg_width     : natural := 16;
        ram_nr_addr   : natural := 4;
        rom_nr_addr   : natural := 4
    );
```

```
    PORT ( clk, -- clock input
        nRST      : in  std_logic; -- reset input      ( active '0' )
        IOBUS_WnR  : out std_logic; -- io bus write input  ( active '1' )
        IOBUS_Data_in : in  std_logic_vector( reg_width - 1 downto 0 ); -- io bus data input
        IOBUS_Address: out std_logic_vector( reg_width - 1 downto 0 ); -- io address bus
        IOBUS_Data_out : out std_logic_vector( reg_width - 1 downto 0 ); -- io bus data output
        ProgMem_Addr : out std_logic_vector( reg_width - 1 downto 0 ); -- program memory address
        IR           : in  std_logic_vector( reg_width - 1 downto 0 ) -- program memory data
    );
```

```
input
```

```
end cpu;
```

```
architecture ideal of cpu is
```

```
    signal      MW           : std_logic; -- ram write input ( active '1' )
```

```
    signal      PL           : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL = '1' )
```

```
    signal      JB           : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with predefined value )
```

```
    signal      JB_address   : std_logic_vector( reg_width - 1 downto 0 );
```

```
    signal      BC           : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit )
```

```
    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
```

```
    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      PC                 : std_logic_vector( reg_width - 1 downto 0 );
signal      A_bus              : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU
begin
    nadzor_vejitve: branch_ctrl
        generic map ( ctr_width => reg_width )
        port map(
            clk => clk,          nRST => nRST,          N => ALU_N_bit,          Z => ALU_Z_bit,
            C => ALU_C_bit,      V => ALU_V_bit,          PL => PL,          JB => JB,          BC => BC,
            JB_address => JB_address,      PC => PC
        );

    cpu_podatkovna_pot: cpu_datapath
        generic map(
            nr_regs => nr_regs,          reg_width => reg_width
        )
        port map(
            clk => clk,          RW => RW,          nRST => nRST,          ALU_mode => ALU_mode,
            MB => MB,          MD => MD,          ALU_function => ALU_function,          Const_in => Const_in,
            Data_in => IOBUS_Data_in,      AA => AA,          BA => BA,          DA => DA,          ALU_N_bit =>
            ALU_N_bit,          ALU_Z_bit => ALU_Z_bit,          ALU_C_bit => ALU_C_bit,          ALU_V_bit => ALU_V_bit,
            Address_out => IOBUS_Address,      Data_out => IOBUS_Data_out
        );

    MB      <= IR( 15 );
    PL      <= IR( 15 ) and IR( 14 );
    MW      <= not IR( 15 ) and IR( 14 );
    RW      <= not IR( 14 );
    MD      <= IR( 13 );
    JB      <= IR( 13 );
    ALU_mode    <= IR( 12 );
    ALU_function <= IR( 11 downto 10 ) & ( IR( 9 ) and not PL );
    BC      <= IR( 9 );
    DA      <= IR( 8 downto 6 );
    AA      <= IR( 5 downto 3 );

```



```
BA    <= IR( 2 downto 0 );
```

```
JB_Address    <= std_logic_vector( resize( signed( DA & BA ), JB_Address'length ) );
```

```
Const_in      <= std_logic_vector( resize( unsigned( BA ), Const_in'length ) );
```

```
IOBUS_WnR     <= MW;
```

```
ProgMem_Addr  <= PC;
```

```
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 work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE work.cpu_functions.all;
use ieee.math_real.all;

-- opcode summary:
-- MB NOT( RW ) MD ALU_M ALU_F
-- 0 0 0 0 000 A plus B
-- 0 0 0 0 001 A minus B
-- 0 0 0 0 010 A plus 1
-- 0 0 0 0 011 A minus 1
-- 0 0 0 0 100 A plus A
-- 0 0 0 0 101 minus 1 ( 2' complement )
-- 0 0 0 1 000 A and B
-- 0 0 0 1 001 A nand B
-- 0 0 0 1 010 A or B
-- 0 0 0 1 011 A nor B
-- 0 0 0 1 100 A xor B
-- 0 0 0 1 101 A xnor B
-- 0 0 0 1 110 TEST A
-- 0 0 0 1 111 NOT IMPLEMENTED
-- 1 0 0 1 111 LDI ( Load immediate )
-- 1 0 0 0 000 ADI ( add immediate )
-- 1 0 0 0 001 SBI ( subtract immediate )
-- 0 0 1 1 110 LOAD A
-- 0 0 0 1 110 MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0 0 1 111 MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1 0 1 111 STORE B
-- JB BC
-- 1 1 0 1 11 0 BRZ R( SA )
-- 1 1 0 1 00 1 BRN R( SA )

```

```
-- 1 1          1 0          00 0 JMP
```

```
entity cpu is
```

```
    generic(
        nr_regs      : natural := 8;
        reg_width    : natural := 16;
        ram_nr_addr   : natural := 4;
        rom_nr_addr   : natural := 4
    );
    PORT (
        clk, -- clock input
        nRST      : in  std_logic; -- reset input ( active '0' )
        IOBUS_WnR  : out std_logic; -- io bus write input ( active '1' )
        IOBUS_Data_in : in  std_logic_vector( reg_width - 1 downto 0 ); -- io bus data input
        IOBUS_Address: out std_logic_vector( reg_width - 1 downto 0 ); -- io address bus
        IOBUS_Data_out : out std_logic_vector( reg_width - 1 downto 0 ); -- io bus data output
        ProgMem_Addr : out std_logic_vector( reg_width - 1 downto 0 ); -- program memory address
        IR           : in  std_logic_vector( reg_width - 1 downto 0 ) -- program memory data
    );
```

```
input
```

```
end cpu;
```

```
architecture ideal of cpu is
```

```
    signal      MW           : std_logic; -- ram write input ( active '1' )

    signal      PL           : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL =
    '1' )

    signal      JB           : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with
    predefined value )

    signal      JB_address   : std_logic_vector( reg_width - 1 downto 0 );
    signal      BC           : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit )

    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

    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      PC              : std_logic_vector( reg_width - 1 downto 0 );
signal      A_bus           : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU
begin

    DA      <= IR( 8 downto 6 );
    AA      <= IR( 5 downto 3 );
    BA      <= IR( 2 downto 0 );
    MB      <= IR( 15 );
    ALU_mode <= IR( 12 );
    ALU_function <= IR( 11 downto 10 ) & ( IR( 9 ) and ( not PL ) );
    MD      <= IR( 13 );
    RW      <= not IR( 14 );
    MW      <= IR( 14 ) and ( not IR( 15 ) );
    PL      <= IR( 14 ) and IR( 15 );
    JB      <= IR( 13 );
    BC      <= IR( 9 );

    JB_address <= STD_LOGIC_VECTOR( resize( signed( IR( 8 downto 6 ) & IR( 2 downto 0 ) ), JB_address'length ) );
    Const_in   <= STD_LOGIC_VECTOR( resize( unsigned( IR( 2 downto 0 ) ), Const_in'length ) );

    br_ctr: branch_ctrl
    generic map ( ctr_width => reg_width )
    port map (
        clk => clk,  nRST => nRST,      N => ALU_N_bit,      C => ALU_C_bit,      V => ALU_V_bit,      Z =>
        ALU_Z_bit,  PL => PL,      JB => JB,      BC => BC,      JB_address => JB_address, PC => PC
    );

    dpath: cpu_datapath
    generic map( nr_regs => nr_regs, reg_width => reg_width )
    port map (
        clk => clk,  RW => RW,      nRST => nRST,      DA => DA,      AA => AA,      BA => BA,      MB => MB,      MD =>
        MD,  ALU_mode => ALU_mode,      ALU_function => ALU_function,      ALU_N_bit => ALU_N_bit,      ALU_C_bit => ALU_C_bit,
        ALU_V_bit => ALU_V_bit,      ALU_Z_bit => ALU_Z_bit,      Const_in => Const_in,      Data_in => IOBUS_Data_in,
        Address_out => IOBUS_Address,      Data_out => IOBUS_Data_out
    );

```

```
);
```

```
IOBUS_WnR    <= MW;
```

```
ProgMem_Addr <= PC;
```

```
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 work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE work.cpu_functions.all;
use ieee.math_real.all;

-- opcode summary:
-- MB NOT( RW ) MD ALU_M ALU_F
-- 0 0 0 0 000 A plus B
-- 0 0 0 0 001 A minus B
-- 0 0 0 0 010 A plus 1
-- 0 0 0 0 011 A minus 1
-- 0 0 0 0 100 A plus A
-- 0 0 0 0 101 minus 1 ( 2' complement )
-- 0 0 0 1 000 A and B
-- 0 0 0 1 001 A nand B
-- 0 0 0 1 010 A or B
-- 0 0 0 1 011 A nor B
-- 0 0 0 1 100 A xor B
-- 0 0 0 1 101 A xnor B
-- 0 0 0 1 110 TEST A
-- 0 0 0 1 111 NOT IMPLEMENTED
-- 1 0 0 1 111 LDI ( Load immediate )
-- 1 0 0 0 000 ADI ( add immediate )
-- 1 0 0 0 001 SBI ( subtract immediate )
-- 0 0 1 1 110 LOAD A
-- 0 0 0 1 110 MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0 0 1 111 MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1 0 1 111 STORE B
-- JB BC
-- 1 1 0 1 11 0 BRZ R( SA )
-- 1 1 0 1 00 1 BRN R( SA )

```

```
-- 1 1          1 0          00 0 JMP
```

```
entity cpu is
```

```
    generic(
        nr_regs      : natural := 8;
        reg_width    : natural := 16;
        ram_nr_addr  : natural := 4;
        rom_nr_addr  : natural := 4
    );
    PORT (
        clk, -- clock input
        nRST      : in  std_logic; -- reset input      ( active '0' )
        IOBUS_WnR  : out std_logic; -- io bus write input  ( active '1' )
        IOBUS_Data_in  : in  std_logic_vector( reg_width - 1 downto 0 ); -- io bus data input
        IOBUS_Address: out std_logic_vector( reg_width - 1 downto 0 ); -- io address bus
        IOBUS_Data_out : out std_logic_vector( reg_width - 1 downto 0 ); -- io bus data output
        ProgMem_Addr  : out std_logic_vector( reg_width - 1 downto 0 ); -- program memory address
        IR            : in  std_logic_vector( reg_width - 1 downto 0 ) -- program memory data
    );
```

```
input
```

```
end cpu;
```

```
architecture ideal of cpu is
```

```
    signal      MW          : std_logic; -- ram write input ( active '1' )

    signal      PL          : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL =
    '1' )

    signal      JB          : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with
    predefined value )

    signal      JB_address  : std_logic_vector( reg_width - 1 downto 0 );
    signal      BC          : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit )

    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

    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      PC              : std_logic_vector( reg_width - 1 downto 0 );
signal      A_bus           : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU
begin

    DA      <= IR( 8 downto 6 );
    AA      <= IR( 5 downto 3 );
    BA      <= IR( 2 downto 0 );
    MB      <= IR( 15 );
    ALU_mode <= IR( 12 );
    ALU_function <= IR( 11 downto 10 ) & ( IR( 9 ) and ( not PL ) );
    MD      <= IR( 13 );
    RW      <= not IR( 14 );
    MW      <= IR( 14 ) and ( not IR( 15 ) );
    PL      <= IR( 14 ) and IR( 15 );
    JB      <= IR( 13 );
    BC      <= IR( 9 );

    JB_address <= STD_LOGIC_VECTOR( resize( signed( IR( 8 downto 6 ) & IR( 2 downto 0 ) ), JB_address'length ) );
    Const_in   <= STD_LOGIC_VECTOR( resize( unsigned( IR( 2 downto 0 ) ), Const_in'length ) );

    bctrl: branch_ctrl
    generic map ( ctr_width => reg_width )
    port map( clk => clk,          nRST => nRST,          N => ALU_N_bit,          C => ALU_C_bit,
              V => ALU_V_bit,      Z => ALU_Z_bit,      PL => PL,          JB => JB,          BC => BC,
              JB_address => JB_address,          PC => PC );

    dpath: cpu_datapath
    generic map( nr_regs => nr_regs, reg_width => reg_width )
    port map( clk => clk,          RW => RW,          nRST => nRST,          DA => DA,          AA => AA,
              BA => BA,          MB => MB,          MD => MD,          ALU_mode => ALU_mode,          ALU_function =>
    ALU_function,          ALU_N_bit => ALU_N_bit,          ALU_C_bit => ALU_C_bit,          ALU_V_bit => ALU_V_bit,
              ALU_Z_bit => ALU_Z_bit,          Const_in => Const_in,          Data_in => IOBUS_Data_in,          Address_out
=> IOBUS_Address,          Data_out => IOBUS_Data_out );

```



```
IOBUS_WnR    <= MW;  
ProgMem_Addr <= PC;
```

```
end ideal;
```

```

-- *****
-- **** STUDENT: 64210455
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Naloga ni programirana.
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.ALL;

entity cpu is
  generic (
    nr_regs : natural := 8;
    reg_width : natural := 16;
    ram_nr_addr : natural := 4;
    rom_nr_addr : natural := 4
  );
  port (
    clk : in std_logic;
    nRST : in std_logic;
    IOBUS_WnR : out std_logic;
    IOBUS_Data_in : in std_logic_vector( reg_width - 1 downto 0 );
    IOBUS_Address : out std_logic_vector( reg_width - 1 downto 0 );
    IOBUS_Data_out: out std_logic_vector( reg_width - 1 downto 0 );
    ProgMem_Addr : out std_logic_vector( reg_width - 1 downto 0 );
    IR : in std_logic_vector( reg_width - 1 downto 0 )
  );
end cpu;

architecture Behavioral of cpu is
  -- Deklaracija komponent in signalov
begin
  -- Povezovanje komponent
end Behavioral;

```

```

-- *****
-- **** STUDENT: 64210457
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

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

-- opcode summary:
-- MB NOT( RW ) MD ALU_M ALU_F
-- 0 0 0 0 000 A plus B
-- 0 0 0 0 001 A minus B
-- 0 0 0 0 010 A plus 1
-- 0 0 0 0 011 A minus 1
-- 0 0 0 0 100 A plus A
-- 0 0 0 0 101 minus 1 ( 2' complement )
-- 0 0 0 1 000 A and B
-- 0 0 0 1 001 A nand B
-- 0 0 0 1 010 A or B
-- 0 0 0 1 011 A nor B
-- 0 0 0 1 100 A xor B
-- 0 0 0 1 101 A xnor B
-- 0 0 0 1 110 TEST A
-- 0 0 0 1 111 NOT IMPLEMENTED
-- 1 0 0 1 111 LDI ( Load immediate )
-- 1 0 0 0 000 ADI ( add immediate )
-- 1 0 0 0 001 SBI ( subtract immediate )
-- 0 0 1 1 110 LOAD A
-- 0 0 0 1 110 MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0 0 1 111 MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1 0 1 111 STORE B
-- JB BC
-- 1 1 0 1 11 0 BRZ R( SA )
-- 1 1 0 1 00 1 BRN R( SA )

```

```
-- 1 1          1 0          00 0 JMP
```

```
entity cpu is
```

```
    generic(
        nr_regs      : natural := 8;
        reg_width     : natural := 16;
        ram_nr_addr   : natural := 4;
        rom_nr_addr   : natural := 4
    );
    PORT (
        clk, -- clock input
        nRST      : in  std_logic; -- reset input      ( active '0' )
        IOBUS_WnR  : out std_logic; -- io bus write input  ( active '1' )
        IOBUS_Data_in  : in  std_logic_vector( reg_width - 1 downto 0 ); -- io bus data input
        IOBUS_Address: out std_logic_vector( reg_width - 1 downto 0 ); -- io address bus
        IOBUS_Data_out : out std_logic_vector( reg_width - 1 downto 0 ); -- io bus data output
        ProgMem_Addr : out std_logic_vector( reg_width - 1 downto 0 ); -- program memory address
        IR           : in  std_logic_vector( reg_width - 1 downto 0 ) -- program memory data
    );
```

```
input
```

```
end cpu;
```

```
architecture ideal of cpu is
```

```
    signal      MW           : std_logic; -- ram write input ( active '1' )

    signal      PL           : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL =
    '1' )

    signal      JB           : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with
    predefined value )

    signal      JB_address   : std_logic_vector( reg_width - 1 downto 0 );
    signal      BC           : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit )

    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

    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      PC              : std_logic_vector( reg_width - 1 downto 0 );
signal      A_bus           : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU

begin

DA    <= IR( 8 downto 6 );
AA    <= IR( 5 downto 3 );
BA    <= IR( 2 downto 0 );
MB    <= IR( 15 );
ALU_mode    <= IR( 12 );
ALU_function <= IR( 11 downto 10 ) & ( IR( 9 ) and ( not PL ) );
MD    <= IR( 13 );
RW    <= not IR( 14 );
MW    <= IR( 14 ) and ( not IR( 15 ) );
PL    <= IR( 14 ) and IR( 15 );
JB    <= IR( 13 );
BC    <= IR( 9 );

JB_address <= STD_LOGIC_VECTOR( resize( signed( IR( 8 downto 6 ) & IR( 2 downto 0 ) ), JB_address'length ) );
Const_in   <= STD_LOGIC_VECTOR( resize( unsigned( IR( 2 downto 0 ) ), Const_in'length ) );

bctrl: branch_ctrl
generic map ( ctr_width => reg_width )
port map (
    clk => clk, nRST => nRST, N => ALU_N_bit, C => ALU_C_bit, V => ALU_V_bit, Z => ALU_Z_bit, PL => PL, JB => JB,
    BC => BC, JB_address => JB_address, PC => PC
);

dpath: cpu_datapath
generic map( nr_regs => nr_regs, reg_width => reg_width )
port map (
    clk => clk, RW => RW, nRST => nRST, DA => DA, AA => AA, BA => BA, MB => MB, MD =>
    MD, ALU_mode => ALU_mode, ALU_function => ALU_function, ALU_N_bit => ALU_N_bit, ALU_C_bit => ALU_C_bit,

```

```
    ALU_V_bit => ALU_V_bit,    ALU_Z_bit => ALU_Z_bit,    Const_in => Const_in,    Data_in => IOBUS_Data_in,  
    Address_out => IOBUS_Address,    Data_out => IOBUS_Data_out  
);
```

```
IOBUS_WnR    <= MW;  
ProgMem_Addr <= PC;
```

```
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 work.reg_file_functions.all;
USE work.cpu_datapath_functions.all;
USE work.cpu_functions.all;
use ieee.math_real.all;

-- opcode summary:
-- MB NOT( RW ) MD ALU_M ALU_F
-- 0 0 0 0 000 A plus B
-- 0 0 0 0 001 A minus B
-- 0 0 0 0 010 A plus 1
-- 0 0 0 0 011 A minus 1
-- 0 0 0 0 100 A plus A
-- 0 0 0 0 101 minus 1 ( 2' complement )
-- 0 0 0 1 000 A and B
-- 0 0 0 1 001 A nand B
-- 0 0 0 1 010 A or B
-- 0 0 0 1 011 A nor B
-- 0 0 0 1 100 A xor B
-- 0 0 0 1 101 A xnor B
-- 0 0 0 1 110 TEST A
-- 0 0 0 1 111 NOT IMPLEMENTED
-- 1 0 0 1 111 LDI ( Load immediate )
-- 1 0 0 0 000 ADI ( add immediate )
-- 1 0 0 0 001 SBI ( subtract immediate )
-- 0 0 1 1 110 LOAD A
-- 0 0 0 1 110 MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0 0 1 111 MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1 0 1 111 STORE B
-- JB BC
-- 1 1 0 1 11 0 BRZ R( SA )
-- 1 1 0 1 00 1 BRN R( SA )

```

```
-- 1 1          1 0          00 0 JMP
```

```
entity cpu is
```

```
    generic(
        nr_regs      : natural := 8;
        reg_width     : natural := 16;
        ram_nr_addr   : natural := 4;
        rom_nr_addr   : natural := 4
    );
    PORT (
        clk, -- clock input
        nRST      : in  std_logic; -- reset input ( active '0' )
        IOBUS_WnR  : out std_logic; -- io bus write input ( active '1' )
        IOBUS_Data_in : in  std_logic_vector( reg_width - 1 downto 0 ); -- io bus data input
        IOBUS_Address: out std_logic_vector( reg_width - 1 downto 0 ); -- io address bus
        IOBUS_Data_out : out std_logic_vector( reg_width - 1 downto 0 ); -- io bus data output
        ProgMem_Addr : out std_logic_vector( reg_width - 1 downto 0 ); -- program memory address
        IR           : in  std_logic_vector( reg_width - 1 downto 0 ) -- program memory data
    );
```

```
end cpu;
```

```
architecture ideal of cpu is
```

```
    signal MW      : std_logic; -- ram write input ( active '1' )

    signal PL      : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL =
    '1' )
    signal JB      : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with
    predefined value )
    signal JB_address : std_logic_vector( reg_width - 1 downto 0 );
    signal BC      : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit )

    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

    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      PC              : std_logic_vector( reg_width - 1 downto 0 );
signal      A_bus           : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU
begin

    brn_ctrl: branch_ctrl
        generic map ( ctr_width => reg_width )
        port map ( clk => clk, nRST => nRST, N => ALU_N_bit, Z =>
ALU_Z_bit, C => ALU_C_bit, V => ALU_V_bit, PL => PL, JB => JB, BC =>
BC, JB_address => JB_address, PC => PC
        );
    data_path: cpu_datapath
        generic map ( nr_regs => nr_regs, reg_width => reg_width
        )
        port map ( clk => clk, RW => RW, nRST => nRST, MB => MB,
MD => MD, AA => AA, BA => BA, DA => DA, ALU_mode => ALU_mode,
ALU_function => ALU_function, ALU_N_bit => ALU_N_bit, ALU_Z_bit => ALU_Z_bit, ALU_C_bit =>
ALU_C_bit, ALU_V_bit => ALU_V_bit, Const_in => Const_in, Data_in => IOBUS_Data_in,
Address_out => IOBUS_Address, Data_out => IOBUS_Data_out
        );
    AA    <= IR( 5 downto 3 );
    BA    <= IR( 2 downto 0 );
    DA    <= IR( 8 downto 6 );
    BC    <= IR( 9 );
    ALU_mode    <= IR( 12 );
    JB    <= IR( 13 );
    MD    <= IR( 13 );
    RW    <= not IR( 14 );
    MB    <= IR( 15 );
    PL    <= IR( 15 ) and IR( 14 );
    MW    <= not IR( 15 ) and IR( 14 );
    ALU_function <= IR( 11 downto 10 ) & ( IR( 9 ) and not PL );

    JB_Address    <= std_logic_vector( resize( signed( DA & BA ), JB_Address'length ) );
    Const_in      <= std_logic_vector( resize( unsigned( BA ), Const_in'length ) );

```

```
IOBUS_WnR    <= MW;  
ProgMem_Addr <= PC;
```

```
end ideal;
```

```

-- *****
-- **** PREDLOGA VAJE
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Mozek: Predloga vaje
-- *****

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

-- opcode summary:
-- MB NOT( RW ) MD ALU_M ALU_F
-- 0 0 0 0 000 A plus B
-- 0 0 0 0 001 A minus B
-- 0 0 0 0 010 A plus 1
-- 0 0 0 0 011 A minus 1
-- 0 0 0 0 100 A plus A
-- 0 0 0 0 101 minus 1 ( 2' complement )
-- 0 0 0 1 000 A and B
-- 0 0 0 1 001 A nand B
-- 0 0 0 1 010 A or B
-- 0 0 0 1 011 A nor B
-- 0 0 0 1 100 A xor B
-- 0 0 0 1 101 A xnor B
-- 0 0 0 1 110 TEST A
-- 0 0 0 1 111 MOVE B
-- 1 0 0 1 111 LDI ( Load immediate )
-- 1 0 0 0 000 ADI ( add immediate )
-- 1 0 0 0 001 SBI ( subtract immediate )
-- 0 0 1 1 110 LOAD A
-- 0 0 0 1 110 MOVE A ( MOVE R( DR )<-R( SA ) )
-- 0 0 0 1 111 MOVE B ( MOVE R( DR )<-R( SB ) )
-- 0 1 0 1 111 STORE B
-- JB BC
-- 1 1 0 1 11 0 BRZ R( SA )
-- 1 1 0 1 00 1 BRN R( SA )

```

```
-- 1 1          1 0          00 0      JMP
```

```
entity cpu is
```

```
    generic( nr_regs      : natural := 8;
              reg_width   : natural := 16;
              ram_nr_addr  : natural := 4;
              rom_nr_addr  : natural := 4
            );
```

```
    PORT ( clk,  -- clock input
           nRST      : in  std_logic;  -- reset input      ( active '0' )
           IOBUS_WnR  : out std_logic;  -- io bus write input ( active '1' )
           IOBUS_Data_in : in  std_logic_vector( reg_width - 1 downto 0 ); -- io bus data input
           IOBUS_Address: out std_logic_vector( reg_width - 1 downto 0 ); -- io address bus
           IOBUS_Data_out : out std_logic_vector( reg_width - 1 downto 0 ); -- io bus data output
           ProgMem_Addr : out std_logic_vector( reg_width - 1 downto 0 ); -- program memory address
           IR           : in  std_logic_vector( reg_width - 1 downto 0 ) -- program memory data
    );
```

```
input
```

```
end cpu;
```

```
architecture ideal of cpu is
```

```
    signal      MW           : std_logic; -- ram write input ( active '1' )
```

```
    signal      PL           : std_logic; -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL = '1' )
```

```
    signal      JB           : std_logic; -- jump/branch when PL='1' ( jump => PL = '1', Load counter with predefined value )
```

```
    signal      JB_address   : std_logic_vector( reg_width - 1 downto 0 );
```

```
    signal      BC           : std_logic; -- branch control ( when '0' -> check Z bit, when '1' check N bit )
```

```
    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
```

```
    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    PC              : std_logic_vector( reg_width - 1 downto 0 );
signal    A_bus           : std_logic_vector( reg_width - 1 downto 0 ); -- A bus from CPU
begin
    -- instruction decoder
    RW    <= not ir( 14 ); -- register write input ( active '1' )
    DA    <= ir( 8 downto 6 ); -- destination register number select input
    AA    <= ir( 5 downto 3 ); -- A bus register number select input
    BA    <= ir( 2 downto 0 ); -- B bus register number select input
    MB    <= ir( 15 ); -- constant/operand bus B bus multiplexer control signal
    MD    <= ir( 13 ); -- 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    <= ir( 12 ); -- mode of alu operation ( M in upper table )
    ALU_function <= ir( 11 downto 10 ) & ( ir( 9 ) and ( not PL ) ); -- function of ALU ( F in upper table )

    MW    <= ir( 14 ) and ( not ir( 15 ) );
    PL    <= ir( 14 ) and ir( 15 ); -- normal op or jump or branch
    JB    <= ir( 13 ); -- jump or branch
    BC    <= ir( 9 ); -- branch control

```

```

-- zero pad const_in operand
Const_in    <= STD_LOGIC_VECTOR( resize( unsigned( ir( 2 downto 0 ) ), Const_in'length ) );

-- sign extension of JB_address when JB='0' ( branch ) or bus A value when jump
JB_address  <= STD_LOGIC_VECTOR( resize( signed( ir( 8 downto 6 ) & ir( 2 downto 0 ) ), JB_address'length ) )
when JB = '0'
    else A_bus;

DATAPATH : 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_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 input bus
          Data_in       => IOBUS_data_in, -- data input bus input
          Address_out   => A_bus, -- address bus output
          Data_out      => IOBUS_data_out -- data bus output
        );

IOBUS_address<= A_bus; -- assign A bus to IO address bus output

--
-- branch controller operation summary
--
-- PL      JB      BC      Z      N      PC
-- 0      X      X      X      X      PC + 1
-- 1      0      0      1      X      branch taken ( Z='1' )
-- 1      0      0      0      X      branch not taken ( Z='0' )=> PC + 1
-- 1      0      1      X      1      branch taken ( N='1' )
-- 1      0      1      X      0      branch not taken ( N='0' )=> PC + 1
-- 1      1      X      X      X      jump

```

```

JB_CTRL : branch_ctrl
    generic map ( ctr_width => reg_width )
    PORT map ( clk          => clk,          -- clock input
               nRST        => nRST,         -- reset input    ( active '0' )
               N            => ALU_N_bit, -- negative bit from ALU operation
               C            => ALU_C_bit, -- carry bit from ALU operation
               V            => ALU_V_bit, -- overflow bit from ALU operation
               Z            => ALU_Z_bit, -- zero bit from ALU operation
               PL           => PL, -- Increment ( PC = PC + 1 ) when inactive, or Load when active ( PL = '1' )
               JB           => JB, -- jump/branch when PL='1' ( jump => PL = '1', Load counter with predefined
value )
               BC           => BC, -- branch control ( when '0' -> check Z bit, when '1' check N bit )
               JB_address   => JB_address, PC => PC
    );

ProgMem_addr <= PC;
IOBUS_WnR    <= MW;
end ideal;

```

