

-- **** STUDENT: 64000225.....	3
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	3
-- **** STUDENT: 64200100.....	6
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni rezultatov simulacije zaradi napak sintetizatorja:.....	6
ERROR:HDLCompiler:1314 - "branch_ctrl.vhd" Line 81: Formal port/generic <buss> is not declared in <muxnto1_bus>.....	6
Spet se vam je od nekod prikladla beseda bus v deklaraciji izbiralnika vodil (busS). Namesto tega bi moralo biti (bus_width). .	6
-- **** STUDENT: 64200112.....	9
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	9
-- **** STUDENT: 64200288.....	12
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	12
-- **** STUDENT: 64200296.....	15
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	15
-- **** STUDENT: 64200385.....	18
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	18
-- **** STUDENT: 64210113.....	21
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni rezultatov simulacije zaradi napak sintetizatorja:.....	21
ERROR:HDLCompiler:432 - "branch_ctrl.vhd" Line 71: Formal <x> has no actual or default value. Pozabili ste definirati povezavo vhoda za vzporedno nalaganje števca. Če v povezovanlni stavek vstavim (x => PC_load_addr, -- counter load value), koda deluje pravilno.	21
-- **** STUDENT: 64210290.....	24
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	24
-- **** STUDENT: 64210382.....	29
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	29
-- **** STUDENT: 64210384.....	32
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	32
-- **** STUDENT: 64210386.....	35
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	35
-- **** STUDENT: 64210445.....	38
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	38
-- **** STUDENT: 64210455.....	41
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	41
-- **** STUDENT: 64210457.....	43
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	43
-- **** STUDENT: 64240430.....	46

-- KOMENTARJI K OCENI NALOGE	-- Matej Možek: Ni pripomb.....	46
-- **** PREDLOGA VAJE		49
-- KOMENTARJI K OCENI NALOGE	-- Matej Možek: Ni pripomb.....	49

```

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

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

--          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

entity 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 branch_ctrl;

architecture ideal of branch_ctrl is

```

-- 2/1 bus multiplexer input type definitions

```
type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( ctr_width - 1 downto 0 );
signal      branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal      branch_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, ctr_width - 1 DOWNT0 0 ) := ( others => ( others
=> '0' ) );
signal      branch_mux_vector : std_logic_vector( 0 downto 0 ); -- PC mode multiplexer control signal
```

```
signal      PC_sig      :      std_logic_vector( ctr_width - 1 downto 0 );
signal      PC_load_addr :      std_logic_vector( ctr_width - 1 downto 0 ); -- absolute address load when JUMP
operation
signal      Adder_result :      std_logic_vector( ctr_width - 1 downto 0 );
signal      CE           :      std_logic; -- count enable
signal      nBRANCH      :      std_logic; -- branching control signal ( active '0' )
begin
```

```
--      ba - always
--      bneg - on negative      ( N='1' )
--      bpos - on positive      ( N='0' )
--      beq - on zero      ( Z='1' )
--      bne - on not zero      ( Z='0' )
--      bvs - on overflow set      ( V='1' )
--      bvc - on overflow clear      ( V='0' )
--      bcs - on carry set      ( C='1' )
--      bcc - on carry clear      ( C='0' )
--      bl - on less than ( ( N xor V )='1' )
--      ble - on less than or equal ( ( Z or ( N xor V ) )='1' )
--      be - on equal ( Z='1' )
--      bne - on not equal ( Z='0' )
--      bge - on greater than or equal ( ( N xor V )='0' )
--      bg - on greater than ( ( Z or ( N xor V ) )='0' )
```

-- 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

```

ctr: counter
generic map ( ctr_size => ctr_width )
port map (
    clk => clk, nCLR => nRST, nLOAD => nBRANCH,    ENP => CE, ENT => CE,    x => PC_load_addr, Q => PC_sig
);

adder: cla_add_n_bit
generic map ( n => ctr_width )
port map (
    Cin => '0', X => JB_address, Y => PC_sig,    S => Adder_result
);

mux: muxnto1_bus
generic map ( n_addr => 1, bus_width => ctr_width )
port map (
    s => branch_mux_vector,    w => branch_mux_2d_bit_array,    f => PC_load_addr
);

branch_mux_vector( 0 )    <= JB;
arrayto2d: for i in 0 to ctr_width-1 generate
    branch_mux_2d_bit_array( 0, i ) <= branch_mux_vector_array( 0 )( i );
    branch_mux_2d_bit_array( 1, i ) <= branch_mux_vector_array( 1 )( i );
end generate;
branch_mux_vector_array( 0 )    <= Adder_result;
branch_mux_vector_array( 1 )    <= JB_address;

nBRANCH    <= not ( PL and ( JB or ( BC and N ) or ( ( not BC ) and Z ) ) );
CE    <= nBRANCH;

PC    <= PC_sig;

end ideal;

```

```

-- *****
-- **** STUDENT: 64200100
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni rezultatov simulacije zaradi napak sintetizatorja:
ERROR:HDLCompiler:1314 - "branch_ctrl.vhd" Line 81: Formal port/generic <buss> is not declared in <muxnto1_bus>.
Spet se vam je od nekod prikradla beseda bus v deklaraciji izbiralnika vodil (busS). Namesto tega bi moralo biti
(bus_width).
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.branch_ctrl_functions.all;
use ieee.math_real.all;

--          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

entity 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 branch_ctrl;

```

```

architecture ideal of branch_ctrl is
-- 2/1 bus multiplexer input type definitions
type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( ctr_width - 1 downto 0 );
signal      branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal      branch_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, ctr_width - 1 DOWNT0 0 ) := ( others => ( others
=> '0' ) );
signal      branch_mux_vector : std_logic_vector( 0 downto 0 ); -- PC mode multiplexer control signal

signal      PC_sig      :      std_logic_vector( ctr_width - 1 downto 0 );
signal      PC_load_addr :      std_logic_vector( ctr_width - 1 downto 0 ); -- absolute address load when JUMP
operation
signal      Adder_result :      std_logic_vector( ctr_width - 1 downto 0 );
signal      CE           :      std_logic; -- count enable
signal      nBRANCH      :      std_logic; -- branching control signal ( active '0' )
begin

--      ba - always
--      bneg - on negative          ( N='1' )
--      bpos - on positive          ( N='0' )
--      beq - on zero              ( Z='1' )
--      bne - on not zero          ( Z='0' )
--      bvs - on overflow set      ( V='1' )
--      bvc - on overflow clear    ( V='0' )
--      bcs - on carry set         ( C='1' )
--      bcc - on carry clear       ( C='0' )
--      bl  - on less than ( ( N xor V )='1' )
--      ble - on less than or equal ( ( Z or ( N xor V ) )='1' )
--      be  - on equal ( Z='1' )
--      bne - on not equal ( Z='0' )
--      bge - on greater than or equal ( ( N xor V )='0' )
--      bg  - on greater than ( ( Z or ( N xor V ) )='0' )

--      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

```

ctronter: counter
generic map ( ctr_size => ctr_width )
port map (
  clk => clk, nCLR => nRST, nLOAD => nBRANCH,
  ENP=> CE, ENT => CE,
  x => PC_load_addr, Q => PC_sig );
adder: cla_add_n_bit
generic map ( n => ctr_width )
port map ( Cin => '0', X => JB_address, Y => PC_sig, S => Adder_result );
muxPLE: muxnto1_bus
generic map ( n_addr => 1, busS => ctr_width )
port map ( S => branch_mux_vector, W => branch_mux_2d_bit_array, F => PC_load_addr );
branch_mux_vector( 0 )    <= JB;
arrayv2D: for i in 0 to ctr_width-1 generate
branch_mux_2d_bit_array( 0, i ) <= branch_mux_vector_array( 0 )( i );
branch_mux_2d_bit_array( 1, i ) <= branch_mux_vector_array( 1 )( i );
end generate;
branch_mux_vector_array( 0 )    <= Adder_result;
branch_mux_vector_array( 1 )    <= JB_address;
nBRANCH    <= not ( PL and ( JB or ( BC and N ) or ( ( not BC ) and Z ) ) );
CE    <= nBRANCH;
PC    <= PC_sig;
end ideal;

```



```

-- *****
-- **** STUDENT: 64200112
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

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

--          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

entity 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 branch_ctrl;

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

```

```

type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( ctr_width - 1 downto 0 );
signal      branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal      branch_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, ctr_width - 1 DOWNT0 0 ) := ( others => ( others
=> '0' ) );
signal      branch_mux_vector : std_logic_vector( 0 downto 0 ); -- PC mode multiplexer control signal

signal      PC_sig      :      std_logic_vector( ctr_width - 1 downto 0 );
signal      PC_load_addr :      std_logic_vector( ctr_width - 1 downto 0 ); -- absolute address load when JUMP
operation
signal      Adder_result :      std_logic_vector( ctr_width - 1 downto 0 );
signal      CE            :      std_logic; -- count enable
signal      nBRANCH      :      std_logic; -- branching control signal ( active '0' )
begin

--      ba      - always
--      bneg - on negative      ( N='1' )
--      bpos - on positive      ( N='0' )
--      beq  - on zero      ( Z='1' )
--      bne  - on not zero      ( Z='0' )
--      bvs  - on overflow set      ( V='1' )
--      bvc  - on overflow clear      ( V='0' )
--      bcs  - on carry set      ( C='1' )
--      bcc  - on carry clear      ( C='0' )
--      bl   - on less than ( ( N xor V )='1' )
--      ble  - on less than or equal ( ( Z or ( N xor V ) )='1' )
--      be   - on equal ( Z='1' )
--      bne  - on not equal ( Z='0' )
--      bge  - on greater than or equal ( ( N xor V )='0' )
--      bg   - on greater than ( ( Z or ( N xor V ) )='0' )

--      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

```

U0 : counter

```

        generic map( ctr_size => ctr_width )
        port map( clk => clk,          nCLR => nRST,          nLOAD => nBRANCH,          ENP => CE,
ENT => CE,          x => PC_load_addr,          Q => PC_sig
        );

U1 : cla_add_n_bit
    generic map( n => ctr_width )
    port map( Cin => '0',          X => JB_address,          Y => PC_sig,          S => Adder_result
    );

U2 : muxnto1_bus
    generic map( n_addr => 1,          bus_width => ctr_width )
    port map( s => branch_mux_vector,          w => branch_mux_2d_bit_array,          f => PC_load_addr
    );

branch_mux_vector( 0 )    <= JB;

U3 : for i in 0 to ctr_width - 1 generate
    branch_mux_2d_bit_array( 0, i ) <= branch_mux_vector_array( 0 )( i );
    branch_mux_2d_bit_array( 1, i ) <= branch_mux_vector_array( 1 )( i );
end generate;

branch_mux_vector_array( 1 )    <= JB_address;
branch_mux_vector_array( 0 )    <= Adder_result;

nBRANCH    <= not( PL and ( JB or ( BC and N ) or ( ( not BC ) and Z ) ) );
CE    <= nBRANCH;

PC    <= PC_sig;

end ideal;

```

```

-- *****
-- **** STUDENT: 64200288
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

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

--          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

entity 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 branch_ctrl;

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

```

```

type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( ctr_width - 1 downto 0 );
signal      branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal      branch_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, ctr_width - 1 DOWNT0 0 ) := ( others => ( others
=> '0' ) );
signal      branch_mux_vector : std_logic_vector( 0 downto 0 ); -- PC mode multiplexer control signal

signal      PC_sig      :      std_logic_vector( ctr_width - 1 downto 0 );
signal      PC_load_addr :      std_logic_vector( ctr_width - 1 downto 0 ); -- absolute address load when JUMP
operation
signal      Adder_result :      std_logic_vector( ctr_width - 1 downto 0 );
signal      CE            :      std_logic; -- count enable
signal      nBRANCH       :      std_logic; -- branching control signal ( active '0' )
begin

--      ba      - always
--      bneg - on negative      ( N='1' )
--      bpos - on positive      ( N='0' )
--      beq  - on zero          ( Z='1' )
--      bne  - on not zero      ( Z='0' )
--      bvs  - on overflow set   ( V='1' )
--      bvc  - on overflow clear ( V='0' )
--      bcs  - on carry set      ( C='1' )
--      bcc  - on carry clear    ( C='0' )
--      bl   - on less than ( ( N xor V )='1' )
--      ble  - on less than or equal ( ( Z or ( N xor V ) )='1' )
--      be   - on equal ( Z='1' )
--      bne  - on not equal ( Z='0' )
--      bge  - on greater than or equal ( ( N xor V )='0' )
--      bg   - on greater than ( ( Z or ( N xor V ) )='0' )

--      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

```

ctr: counter

```

generic map ( ctr_size => ctr_width )
port map (
    clk => clk, nCLR => nRST, nLOAD => nBRANCH, ENP => CE, ENT => CE, x => PC_load_addr, Q => PC_sig
);

adder: cla_add_n_bit
generic map ( n => ctr_width )
port map (
    Cin => '0', X => JB_address, Y => PC_sig, S => Adder_result
);

mux: muxnto1_bus
generic map ( n_addr => 1, bus_width => ctr_width )
port map (
    s => branch_mux_vector, w => branch_mux_2d_bit_array, f => PC_load_addr
);

ArrayTo2D: for i in 0 to ctr_width-1 generate
    branch_mux_2d_bit_array( 0, i ) <= branch_mux_vector_array( 0 )( i );
    branch_mux_2d_bit_array( 1, i ) <= branch_mux_vector_array( 1 )( i );
end generate;

branch_mux_vector( 0 )    <= JB;
branch_mux_vector_array( 0 )    <= Adder_result;
branch_mux_vector_array( 1 )    <= JB_address;

nBRANCH    <= not ( PL and ( JB or ( BC and N ) or ( ( not BC ) and Z ) ) );
CE         <= nBRANCH;

PC         <= PC_sig;

end ideal;

```

```

-- *****
-- **** STUDENT: 64200296
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

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

--          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

entity 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 branch_ctrl;

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

```

```

type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( ctr_width - 1 downto 0 );
signal      branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal      branch_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, ctr_width - 1 DOWNT0 0 ) := ( others => ( others
=> '0' ) );
signal      branch_mux_vector : std_logic_vector( 0 downto 0 ); -- PC mode multiplexer control signal

signal      PC_sig      :      std_logic_vector( ctr_width - 1 downto 0 );
signal      PC_load_addr :      std_logic_vector( ctr_width - 1 downto 0 ); -- absolute address load when JUMP
operation
signal      Adder_result :      std_logic_vector( ctr_width - 1 downto 0 );
signal      CE            :      std_logic; -- count enable
signal      nBRANCH       :      std_logic; -- branching control signal ( active '0' )
begin

--      ba      - always
--      bneg - on negative      ( N='1' )
--      bpos - on positive      ( N='0' )
--      beq  - on zero          ( Z='1' )
--      bne  - on not zero      ( Z='0' )
--      bvs  - on overflow set   ( V='1' )
--      bvc  - on overflow clear ( V='0' )
--      bcs  - on carry set      ( C='1' )
--      bcc  - on carry clear    ( C='0' )
--      bl   - on less than ( ( N xor V )='1' )
--      ble  - on less than or equal ( ( Z or ( N xor V ) )='1' )
--      be   - on equal ( Z='1' )
--      bne  - on not equal ( Z='0' )
--      bge  - on greater than or equal ( ( N xor V )='0' )
--      bg   - on greater than ( ( Z or ( N xor V ) )='0' )

--      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

```

count: counter


```

generic map ( ctr_size => ctr_width )
port map (
    clk => clk, nCLR => nRST, nLOAD => nBRANCH, ENP => CE, ENT => CE, x => PC_load_addr, Q => PC_sig
);

muxto_bus: muxnto1_bus
generic map ( n_addr => 1, bus_width => ctr_width )
port map (
    s => branch_mux_vector, w => branch_mux_2d_bit_array, f => PC_load_addr
);

cla_add: cla_add_n_bit
generic map ( n => ctr_width )
port map (
    Cin => '0', X => JB_address, Y => PC_sig, S => Adder_result
);

nBRANCH      <= '0' when ( PL='1' AND ( ( BC='0' AND Z='1' ) OR ( BC='1' AND N='1' ) OR JB='1' ) ) else '1';
PC           <= PC_sig;
CE           <= nBRANCH;
branch_mux_vector_array( 0 )    <= Adder_result;
branch_mux_vector_array( 1 )    <= JB_address;
branch_mux_vector( 0 )         <= JB;

branch_array: process( branch_mux_vector_array )
begin
    for i in 0 to 1 loop
        for j in 0 to ctr_width-1 loop
            branch_mux_2d_bit_array( i, j ) <= branch_mux_vector_array( i )( j );
        end loop;
    end loop;
end process;

end ideal;

```

```

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

--          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

entity 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 branch_ctrl;

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

```

```

type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( ctr_width - 1 downto 0 );
signal      branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal      branch_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, ctr_width - 1 DOWNT0 0 ) := ( others => ( others
=> '0' ) );
signal      branch_mux_vector : std_logic_vector( 0 downto 0 ); -- PC mode multiplexer control signal

signal      PC_sig      :      std_logic_vector( ctr_width - 1 downto 0 );
signal      PC_load_addr :      std_logic_vector( ctr_width - 1 downto 0 ); -- absolute address load when JUMP
operation
signal      Adder_result :      std_logic_vector( ctr_width - 1 downto 0 );
signal      CE            :      std_logic; -- count enable
signal      nBRANCH      :      std_logic; -- branching control signal ( active '0' )
begin

--      ba      - always
--      bneg - on negative      ( N='1' )
--      bpos - on positive      ( N='0' )
--      beq  - on zero      ( Z='1' )
--      bne  - on not zero      ( Z='0' )
--      bvs  - on overflow set      ( V='1' )
--      bvc  - on overflow clear      ( V='0' )
--      bcs  - on carry set      ( C='1' )
--      bcc  - on carry clear      ( C='0' )
--      bl   - on less than ( ( N xor V )='1' )
--      ble  - on less than or equal ( ( Z or ( N xor V ) )='1' )
--      be   - on equal ( Z='1' )
--      bne  - on not equal ( Z='0' )
--      bge  - on greater than or equal ( ( N xor V )='0' )
--      bg   - on greater than ( ( Z or ( N xor V ) )='0' )

--      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

nBRANCH      <= not ( PL and ( JB or ( BC and N ) or ( ( not BC ) and Z ) ) );

```

```

CE      <= nBRANCH;

U0: counter
generic map( ctr_size => ctr_width )
port map( clk => clk,
  nCLR => nRST,
  nLOAD => nBRANCH,
  ENP => CE,
  ENT => CE,
  RCO => open,
  x => PC_load_addr,
  Q => PC_sig );

U1: cla_add_n_bit
generic map( n => ctr_width )
port map( Cin => '0',
  X => JB_address,
  Y => PC_sig,
  S => Adder_result );

U2: muxnto1_bus
generic map( n_addr => 1,
  bus_width => ctr_width )
port map ( s => branch_mux_vector,
  w => branch_mux_2d_bit_array,
  f => PC_load_addr );

branch_mux_vector( 0 )      <= JB;
branch_mux_vector_array( 0 ) <= Adder_result;
branch_mux_vector_array( 1 ) <= JB_address;

vt2: for i in 0 to ctr_width - 1 generate
  branch_mux_2d_bit_array( 0,i ) <= branch_mux_vector_array( 0 )( i );
  branch_mux_2d_bit_array( 1,i ) <= branch_mux_vector_array( 1 )( i );
end generate;

PC      <= PC_sig;

end ideal;

```

```

-- *****
-- **** STUDENT: 64210113
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni rezultatov simulacije zaradi napak sintetizatorja:
ERROR:HDLCompiler:432 - "branch_ctrl.vhd" Line 71: Formal <x> has no actual or default value.
Pozabili ste definirati povezavo vhoda za vzporedno nalaganje števca. Če v povezovalni stavek vstavim (x =>
PC_load_addr,      -- counter load value), koda deluje pravilno.
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.branch_ctrl_functions.all;
use ieee.math_real.all;

--          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

entity 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 branch_ctrl;

```

```

architecture ideal of branch_ctrl is
-- 2/1 bus multiplexer input type definitions
type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( ctr_width - 1 downto 0 );
signal      branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal      branch_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, ctr_width - 1 DOWNT0 0 ) := ( others => ( others
=> '0' ) );
signal      branch_mux_vector : std_logic_vector( 0 downto 0 ); -- PC mode multiplexer control signal

signal      PC_sig      :      std_logic_vector( ctr_width - 1 downto 0 );
signal      PC_load_addr :      std_logic_vector( ctr_width - 1 downto 0 ); -- absolute address load when JUMP
operation
signal      Adder_result :      std_logic_vector( ctr_width - 1 downto 0 );
signal      CE           :      std_logic; -- count enable
signal      nBRANCH      :      std_logic; -- branching control signal ( active '0' )
begin

--      ba - always
--      bneg - on negative          ( N='1' )
--      bpos - on positive          ( N='0' )
--      beq - on zero              ( Z='1' )
--      bne - on not zero          ( Z='0' )
--      bvs - on overflow set      ( V='1' )
--      bvc - on overflow clear    ( V='0' )
--      bcs - on carry set         ( C='1' )
--      bcc - on carry clear       ( C='0' )
--      bl  - on less than ( ( N xor V )='1' )
--      ble - on less than or equal ( ( Z or ( N xor V ) )='1' )
--      be  - on equal ( Z='1' )
--      bne - on not equal ( Z='0' )
--      bge - on greater than or equal ( ( N xor V )='0' )
--      bg  - on greater than ( ( Z or ( N xor V ) )='0' )

--      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

contr: counter
generic map ( ctr_size => ctr_width )
port map (
    clk => clk,  nCLR => nRST,      nLOAD => nBRANCH,  ENP => CE,  ENT => CE,  Q => PC_sig
);

aderr: cla_add_n_bit
generic map ( n => ctr_width )
port map (
    Cin => '0',  X => JB_address,  Y => PC_sig,  S => Adder_result );

mux: muxnto1_bus
generic map ( n_addr => 1, bus_width => ctr_width )
port map (
    s => branch_mux_vector,  w => branch_mux_2d_bit_array,  f => PC_load_addr
);

branch_mux_vector( 0 )    <= JB;

arrayto2d: for i in 0 to ctr_width-1 generate
    branch_mux_2d_bit_array( 0, i ) <= branch_mux_vector_array( 0 )( i );
    branch_mux_2d_bit_array( 1, i ) <= branch_mux_vector_array( 1 )( i );

end generate;

branch_mux_vector_array( 0 )    <= Adder_result;
branch_mux_vector_array( 1 )    <= JB_address;

nBRANCH    <= not( PL and( JB or( BC and N ) or( Z and( not BC ) ) ) );

PC    <= PC_sig;
CE    <= nBRANCH;

end ideal;

```

```
-- *****
-- **** STUDENT: 64210290
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
```

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE work.branch_ctrl_functions.all;
USE ieee.math_real.all;
```

```
--          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
```

```
ENTITY 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 branch_ctrl;
```


ARCHITECTURE ideal OF branch_ctrl IS

```
    component counter is
        generic(
            ctr_size: natural
        );
        port(
            clk,    -- signal      ure
            nCLR,   -- signal      za brisanje števca ( aktiven '0' )
            nLOAD,  -- signal      za nalaganje števca ( aktiven '0' )
            ENP, ENT : in         std_logic;  -- signala za omogočanje štetja ( aktiven '1' )
            RCO      : out        std_logic;  -- izhodni prenos na naslednjo stopnjo ( rco )
            x         : in         std_logic_vector( ctr_size - 1 downto 0 );  -- vhod za
vzporedno nalaganje
            Q         : out        std_logic_vector( ctr_size - 1 downto 0 )  -- izhodno štetje
        );
    end component;

    component cla_add_n_bit is
        generic(
            n : natural
        );
        port(
            Cin      : in         std_logic ;
            X, Y     : in         std_logic_vector( n-1 downto 0 );
            S         : out        std_logic_vector( n-1 downto 0 );
            Gout,     Pout, Cout  : out        std_logic
        );
    end component;

    component muxnto1_bus is
        generic(
            n_addr      : integer;
            bus_width    : integer
        );
        port(
            s           : in         std_logic_vector( n_addr - 1 downto 0 );
            w           : in         muxnto1_bus_type( 2**n_addr - 1 downto 0, bus_width - 1 downto 0 );
            f           : out        std_logic_vector( bus_width - 1 downto 0 )
        );
    end component;
```

```

    );
end component;

-- 2/1 bus multiplexer input type definitions
type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( ctr_width - 1 downto 0 );
signal branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal branch_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, ctr_width - 1 DOWNT0 0 ) := ( others => (
others => '0' ) );
signal branch_mux_vector : std_logic_vector( 0 downto 0 ); -- PC mode multiplexer control signal

signal PC_sig : std_logic_vector( ctr_width - 1 downto 0 );
signal PC_load_addr : std_logic_vector( ctr_width - 1 downto 0 ); -- absolute address load when
JUMP operation
signal Adder_result : std_logic_vector( ctr_width - 1 downto 0 );
signal CE : std_logic; -- count enable
signal nBRANCH : std_logic; -- branching control signal ( active '0'
)

BEGIN

--      ba    - always
--      bneg - on negative          ( N='1' )
--      bpos - on positive          ( N='0' )
--      beq  - on zero              ( Z='1' )
--      bne  - on not zero          ( Z='0' )
--      bvs  - on overflow set      ( V='1' )
--      bvc  - on overflow clear    ( V='0' )
--      bcs  - on carry set         ( C='1' )
--      bcc  - on carry clear       ( C='0' )
--      bl   - on less than ( ( N xor V )='1' )
--      ble  - on less than or equal ( ( Z or ( N xor V ) )='1' )
--      be   - on equal ( Z='1' )
--      bne  - on not equal ( Z='0' )
--      bge  - on greater than or equal ( ( N xor V )='0' )
--      bg   - on greater than ( ( Z or ( N xor V ) )='0' )

--      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	<i>branch not taken (Z='0')=> PC</i>
+ 1	--	1	0	1	X	1	<i>branch taken (N='1')</i>
	--	1	0	1	X	0	<i>branch not taken (N='0')=> PC</i>
+ 1	--	1	1	X	X	X	<i>jump</i>

```

nBRANCH    <=
    not(
        PL
        and(
            ( not( JB ) and not( BC ) and Z )
            or( not( JB ) and BC and N )
            or JB
        )
    );

pc_counter : counter
    generic map(
        ctr_size => ctr_width
    )
    port map(
        clk => clk,          nCLR => nRST,          nLOAD => nBRANCH,          ENP => '1',          ENT =>
'1',          RCO => open,          x => PC_load_addr,          Q => PC_sig
    );

PC    <= PC_sig;

adder : cla_add_n_bit
    generic map(
        n => ctr_width
    )
    port map(
        Cin => '0',          X => PC_sig,          Y => JB_address,          S => Adder_result,          Gout
=> open,          Pout => open,          Cout => open
    );

mux_wires : process( Adder_result, JB_address )
begin
    for hor in 0 to ctr_width-1 loop

```

```

        branch_mux_2d_bit_array( 0,hor ) <= Adder_result( hor );
        branch_mux_2d_bit_array( 1,hor ) <= JB_address( hor );
    end loop;
end process;

branch_mux_vector( 0 )    <= JB;

mux_branch : muxnto1_bus
    generic map(
        n_addr => 1,          bus_width => ctr_width
    )
    port map(
        s => branch_mux_vector,      w => branch_mux_2d_bit_array,      f => PC_load_addr
    );

END ideal;

```

```

-- *****
-- **** STUDENT: 64210382
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

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

--          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

entity 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 branch_ctrl;

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

```

```

type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( ctr_width - 1 downto 0 );
signal      branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal      branch_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, ctr_width - 1 DOWNT0 0 ) := ( others => ( others
=> '0' ) );
signal      branch_mux_vector : std_logic_vector( 0 downto 0 ); -- PC mode multiplexer control signal

signal      PC_sig      :      std_logic_vector( ctr_width - 1 downto 0 );
signal      PC_load_addr :      std_logic_vector( ctr_width - 1 downto 0 ); -- absolute address load when JUMP
operation
signal      Adder_result :      std_logic_vector( ctr_width - 1 downto 0 );
signal      CE            :      std_logic; -- count enable
signal      nBRANCH       :      std_logic; -- branching control signal ( active '0' )
begin

--      ba      - always
--      bneg - on negative          ( N='1' )
--      bpos - on positive          ( N='0' )
--      beq  - on zero              ( Z='1' )
--      bne  - on not zero          ( Z='0' )
--      bvs  - on overflow set      ( V='1' )
--      bvc  - on overflow clear    ( V='0' )
--      bcs  - on carry set         ( C='1' )
--      bcc  - on carry clear       ( C='0' )
--      bl   - on less than ( ( N xor V )='1' )
--      ble  - on less than or equal ( ( Z or ( N xor V ) )='1' )
--      be   - on equal ( Z='1' )
--      bne  - on not equal ( Z='0' )
--      bge  - on greater than or equal ( ( N xor V )='0' )
--      bg   - on greater than ( ( Z or ( N xor V ) )='0' )

--      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
ctr: counter
generic map ( ctr_size => ctr_width )

```

```

port map (
    clk => clk, nCLR => nRST, nLOAD => nBRANCH, ENP => CE, ENT => CE, x => PC_load_addr, Q => PC_sig
);

adder: cla_add_n_bit
generic map ( n => ctr_width )
port map (
    Cin => '0', X => JB_address, Y => PC_sig, S => Adder_result
);

mux: muxnto1_bus
generic map ( n_addr => 1, bus_width => ctr_width )
port map (
    s => branch_mux_vector, w => branch_mux_2d_bit_array, f => PC_load_addr
);

branch_mux_vector( 0 )    <= JB;
arrayto2d: for i in 0 to ctr_width-1 generate
    branch_mux_2d_bit_array( 0, i ) <= branch_mux_vector_array( 0 )( i );
    branch_mux_2d_bit_array( 1, i ) <= branch_mux_vector_array( 1 )( i );
end generate;
branch_mux_vector_array( 0 )    <= Adder_result;
branch_mux_vector_array( 1 )    <= JB_address;

nBRANCH    <= not ( PL and ( JB or ( BC and N ) or ( ( not BC ) and Z ) ) );
CE         <= nBRANCH;

PC         <= PC_sig;
end ideal;

```

```

-- *****
-- **** STUDENT: 64210384
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

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

--          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

entity 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 branch_ctrl;

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

```



```

type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( ctr_width - 1 downto 0 );
signal      branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal      branch_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, ctr_width - 1 DOWNT0 0 ) := ( others => ( others
=> '0' ) );
signal      branch_mux_vector : std_logic_vector( 0 downto 0 ); -- PC mode multiplexer control signal

signal      PC_sig      :      std_logic_vector( ctr_width - 1 downto 0 );
signal      PC_load_addr :      std_logic_vector( ctr_width - 1 downto 0 ); -- absolute address load when JUMP
operation
signal      Adder_result :      std_logic_vector( ctr_width - 1 downto 0 );
signal      CE            :      std_logic; -- count enable
signal      nBRANCH       :      std_logic; -- branching control signal ( active '0' )

begin
--      ba      - always
--      bneg - on negative      ( N='1' )
--      bpos - on positive      ( N='0' )
--      beq  - on zero          ( Z='1' )
--      bne  - on not zero      ( Z='0' )
--      bvs  - on overflow set   ( V='1' )
--      bvc  - on overflow clear ( V='0' )
--      bcs  - on carry set      ( C='1' )
--      bcc  - on carry clear    ( C='0' )
--      bl   - on less than ( ( N xor V )='1' )
--      ble  - on less than or equal ( ( Z or ( N xor V ) )='1' )
--      be   - on equal ( Z='1' )
--      bne  - on not equal ( Z='0' )
--      bge  - on greater than or equal ( ( N xor V )='0' )
--      bg   - on greater than ( ( Z or ( N xor V ) )='0' )

--      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

```

COUNTRR: counter

```

        generic map ( ctr_size => ctr_width )
        port map(
            clk => clk,          nCLR => nRST,          nLOAD => nBRANCH,          ENT => CE,          ENP =>
CE,          RCO => open,          x => PC_load_addr,          Q => PC_sig
        );

    ADDER: cla_add_n_bit
    generic map ( n => ctr_width )
    port map(
        Cin => '0', X => JB_address, Y => PC_sig, S => Adder_result, Gout => open, Pout => open,
    Cout => open
    );

    MUX_PC: muxnto1_bus
    generic map(
        n_addr => branch_mux_vector'length,          bus_width => ctr_width
    )
    port map(
        s => branch_mux_vector,          w => branch_mux_2d_bit_array,          f => PC_load_addr
    );

    nBRANCH    <= not ( PL and ( JB or ( not BC and Z ) or ( BC and N ) ) );
    CE         <= nBRANCH;

    branch_mux_vector( 0 )    <= JB;
    branch_mux_vector_array( 0 )    <= Adder_result;
    branch_mux_vector_array( 1 )    <= JB_address;

    prilagajanje:
    for i in 0 to ctr_width - 1 generate
        branch_mux_2d_bit_array( 0, i ) <= branch_mux_vector_array( 0 )( i );
        branch_mux_2d_bit_array( 1, i ) <= branch_mux_vector_array( 1 )( i );
    end generate;

    PC         <= PC_sig;

end ideal;

```

```

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

--          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

entity 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 branch_ctrl;

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

```

```

type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( ctr_width - 1 downto 0 );
signal      branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal      branch_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, ctr_width - 1 DOWNT0 0 ) := ( others => ( others
=> '0' ) );
signal      branch_mux_vector : std_logic_vector( 0 downto 0 ); -- PC mode multiplexer control signal

signal      PC_sig      :      std_logic_vector( ctr_width - 1 downto 0 );
signal      PC_load_addr :      std_logic_vector( ctr_width - 1 downto 0 ); -- absolute address load when JUMP
operation
signal      Adder_result :      std_logic_vector( ctr_width - 1 downto 0 );
signal      CE            :      std_logic; -- count enable
signal      nBRANCH      :      std_logic; -- branching control signal ( active '0' )
begin

--      ba - always
--      bneg - on negative          ( N='1' )
--      bpos - on positive          ( N='0' )
--      beq - on zero              ( Z='1' )
--      bne - on not zero          ( Z='0' )
--      bvs - on overflow set      ( V='1' )
--      bvc - on overflow clear    ( V='0' )
--      bcs - on carry set         ( C='1' )
--      bcc - on carry clear       ( C='0' )
--      bl - on less than ( ( N xor V )='1' )
--      ble - on less than or equal ( ( Z or ( N xor V ) )='1' )
--      be - on equal ( Z='1' )
--      bne - on not equal ( Z='0' )
--      bge - on greater than or equal ( ( N xor V )='0' )
--      bg - on greater than ( ( Z or ( N xor V ) )='0' )

--      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

```

cr: counter

```

generic map ( ctr_size => ctr_width )
port map (
    clk => clk, nCLR => nRST, nLOAD => nBRANCH,    ENP => CE, ENT => CE,    x => PC_load_addr, Q => PC_sig
);

add: cla_add_n_bit
generic map ( n => ctr_width )
port map (
    Cin => '0', X => JB_address, Y => PC_sig,    S => Adder_result
);

mux: muxnto1_bus
generic map ( n_addr => 1, bus_width => ctr_width )
port map (
    s => branch_mux_vector, w => branch_mux_2d_bit_array, f => PC_load_addr
);

branch_mux_vector( 0 )    <= JB;
arrayto2d: for i in 0 to ctr_width-1 generate
branch_mux_2d_bit_array( 0, i ) <= branch_mux_vector_array( 0 )( i );
branch_mux_2d_bit_array( 1, i ) <= branch_mux_vector_array( 1 )( i );
end generate;
branch_mux_vector_array( 0 )    <= Adder_result;
branch_mux_vector_array( 1 )    <= JB_address;

nBRANCH    <= not ( PL and ( JB or ( BC and N ) or ( ( not BC ) and Z ) ) );
CE    <= nBRANCH;

PC    <= PC_sig;
end ideal;

```

```

-- *****
-- **** STUDENT: 64210445
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

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

--          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

entity 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 branch_ctrl;

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

```

```

type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( ctr_width - 1 downto 0 );
signal      branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal      branch_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, ctr_width - 1 DOWNT0 0 ) := ( others => ( others
=> '0' ) );
signal      branch_mux_vector : std_logic_vector( 0 downto 0 ); -- PC mode multiplexer control signal

signal      PC_sig      :      std_logic_vector( ctr_width - 1 downto 0 );
signal      PC_load_addr :      std_logic_vector( ctr_width - 1 downto 0 ); -- absolute address load when JUMP
operation
signal      Adder_result :      std_logic_vector( ctr_width - 1 downto 0 );
signal      CE            :      std_logic; -- count enable
signal      nBRANCH       :      std_logic; -- branching control signal ( active '0' )
begin

--      ba - always
--      bneg - on negative          ( N='1' )
--      bpos - on positive          ( N='0' )
--      beq - on zero              ( Z='1' )
--      bne - on not zero          ( Z='0' )
--      bvs - on overflow set      ( V='1' )
--      bvc - on overflow clear    ( V='0' )
--      bcs - on carry set         ( C='1' )
--      bcc - on carry clear       ( C='0' )
--      bl - on less than ( ( N xor V )='1' )
--      ble - on less than or equal ( ( Z or ( N xor V ) )='1' )
--      be - on equal ( Z='1' )
--      bne - on not equal ( Z='0' )
--      bge - on greater than or equal ( ( N xor V )='0' )
--      bg - on greater than ( ( Z or ( N xor V ) )='0' )

--      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

```

ctr: counter

```

generic map ( ctr_size => ctr_width )
port map( clk => clk,          nCLR => nRST,          nLOAD => nBRANCH,          ENP => CE,          ENT =>
CE,          x => PC_load_addr,          Q => PC_sig );

adder: cla_add_n_bit
generic map ( n => ctr_width )
port map( Cin => '0',          X => JB_address,          Y => PC_sig,          S => Adder_result );

mux: muxnto1_bus
generic map ( n_addr => 1, bus_width => ctr_width )
port map( s => branch_mux_vector,          w => branch_mux_2d_bit_array,          f => PC_load_addr );

branch_mux_vector( 0 )    <= JB;
arrayto2d: for i in 0 to ctr_width-1 generate

    branch_mux_2d_bit_array( 0, i ) <= branch_mux_vector_array( 0 )( i );
    branch_mux_2d_bit_array( 1, i ) <= branch_mux_vector_array( 1 )( i );

end generate;

branch_mux_vector_array( 0 )    <= Adder_result;
branch_mux_vector_array( 1 )    <= JB_address;

nBRANCH    <= not ( PL and ( JB or ( BC and N ) or ( ( not BC ) and Z ) ) );
CE         <= nBRANCH;
PC         <= PC_sig;

end ideal;

```



```

-- *****
-- **** STUDENT: 64210455
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
architecture ideal of branch_ctrl is
    signal          branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
    signal          branch_mux_2d_bit_array : muxnto1_bus_type( 1 downto 0, ctr_width - 1 downto 0 ) := ( others => (
others => '0' ) );
    signal          branch_mux_vector : std_logic_vector( 0 downto 0 );
    signal          PC_sig, PC_load_addr, Adder_result : std_logic_vector( ctr_width - 1 downto 0 );
    signal          CE, nBRANCH : std_logic;

begin
    -- MUX za izbiro relativnega ali absolutnega naslova
    mux: muxnto1_bus
    generic map ( n_addr => 1, bus_width => ctr_width )
    port map (
    s => branch_mux_vector, w => branch_mux_2d_bit_array, f => PC_load_addr
    );

    -- Nadzor skokov in vejitev
    nBRANCH    <= not ( PL and ( JB or ( BC and N ) or ( ( not BC ) and Z ) ) );
    CE         <= nBRANCH;

    -- Števec
    ctr: counter
    generic map ( ctr_size => ctr_width )
    port map ( clk => clk, nCLR => nRST, nLOAD => nBRANCH, ENP => CE, ENT => CE, x => PC_load_addr, Q => PC_sig );

    -- Seštevalnik za relativni naslov
    adder: cla_add_n_bit
    generic map ( n => ctr_width )
    port map ( Cin => '0', X => JB_address, Y => PC_sig, S => Adder_result );

    -- Izbira med relativnim in absolutnim naslovom
    branch_mux_vector_array( 0 )    <= Adder_result;
    branch_mux_vector_array( 1 )    <= JB_address;
    branch_mux_vector( 0 )          <= JB;

```

```
PC    <= PC_sig;
```

```
end architecture ideal;
```

```

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

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

--          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

entity 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 branch_ctrl;

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

```

```

type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( ctr_width - 1 downto 0 );
signal      branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal      branch_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, ctr_width - 1 DOWNT0 0 ) := ( others => ( others
=> '0' ) );
signal      branch_mux_vector : std_logic_vector( 0 downto 0 ); -- PC mode multiplexer control signal

signal      PC_sig      :      std_logic_vector( ctr_width - 1 downto 0 );
signal      PC_load_addr :      std_logic_vector( ctr_width - 1 downto 0 ); -- absolute address load when JUMP
operation
signal      Adder_result :      std_logic_vector( ctr_width - 1 downto 0 );
signal      CE            :      std_logic; -- count enable
signal      nBRANCH       :      std_logic; -- branching control signal ( active '0' )
begin

--      ba      - always
--      bneg - on negative          ( N='1' )
--      bpos - on positive          ( N='0' )
--      beq  - on zero              ( Z='1' )
--      bne  - on not zero          ( Z='0' )
--      bvs  - on overflow set      ( V='1' )
--      bvc  - on overflow clear    ( V='0' )
--      bcs  - on carry set         ( C='1' )
--      bcc  - on carry clear       ( C='0' )
--      bl   - on less than ( ( N xor V )='1' )
--      ble  - on less than or equal ( ( Z or ( N xor V ) )='1' )
--      be   - on equal ( Z='1' )
--      bne  - on not equal ( Z='0' )
--      bge  - on greater than or equal ( ( N xor V )='0' )
--      bg   - on greater than ( ( Z or ( N xor V ) )='0' )

--      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

```

ctr: counter

```

generic map ( ctr_size => ctr_width )
port map (
    clk => clk, nCLR => nRST, nLOAD => nBRANCH, ENP => CE, ENT => CE, x => PC_load_addr, Q => PC_sig
);

adder: cla_add_n_bit
generic map ( n => ctr_width )
port map (
    Cin => '0', X => JB_address, Y => PC_sig, S => Adder_result
);

mux: muxnto1_bus
generic map ( n_addr => 1, bus_width => ctr_width )
port map (
    s => branch_mux_vector, w => branch_mux_2d_bit_array, f => PC_load_addr
);

branch_mux_vector( 0 )    <= JB;
arrayto2d: for i in 0 to ctr_width-1 generate
    branch_mux_2d_bit_array( 0, i ) <= branch_mux_vector_array( 0 )( i );
    branch_mux_2d_bit_array( 1, i ) <= branch_mux_vector_array( 1 )( i );
end generate;
branch_mux_vector_array( 0 )    <= Adder_result;
branch_mux_vector_array( 1 )    <= JB_address;

nBRANCH    <= not ( PL and ( JB or ( BC and N ) or ( ( not BC ) and Z ) ) );
CE         <= nBRANCH;

PC         <= PC_sig;

end ideal;

```

```

-- *****
-- **** STUDENT: 64240430
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

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

--          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

entity 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 branch_ctrl;

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

```

```

type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( ctr_width - 1 downto 0 );
signal      branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal      branch_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, ctr_width - 1 DOWNT0 0 ) := ( others => ( others
=> '0' ) );
signal      branch_mux_vector : std_logic_vector( 0 downto 0 ); -- PC mode multiplexer control signal

signal      PC_sig      :      std_logic_vector( ctr_width - 1 downto 0 );
signal      PC_load_addr :      std_logic_vector( ctr_width - 1 downto 0 ); -- absolute address load when JUMP
operation
signal      Adder_result :      std_logic_vector( ctr_width - 1 downto 0 );
signal      CE            :      std_logic; -- count enable
signal      nBRANCH       :      std_logic; -- branching control signal ( active '0' )
begin

--      ba - always
--      bneg - on negative          ( N='1' )
--      bpos - on positive          ( N='0' )
--      beq - on zero              ( Z='1' )
--      bne - on not zero          ( Z='0' )
--      bvs - on overflow set      ( V='1' )
--      bvc - on overflow clear    ( V='0' )
--      bcs - on carry set         ( C='1' )
--      bcc - on carry clear       ( C='0' )
--      bl - on less than ( ( N xor V )='1' )
--      ble - on less than or equal ( ( Z or ( N xor V ) )='1' )
--      be - on equal ( Z='1' )
--      bne - on not equal ( Z='0' )
--      bge - on greater than or equal ( ( N xor V )='0' )
--      bg - on greater than ( ( Z or ( N xor V ) )='0' )

--      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

```

cnt: counter

```

        generic map ( ctr_size => ctr_width )
        port map(
            clk => clk,          nCLR => nRST,          nLOAD => nBRANCH,          ENT => CE,          ENP =>
CE,          RCO => open,          x => PC_load_addr,          Q => PC_sig
        );

    adder: cla_add_n_bit
        generic map ( n => ctr_width )
        port map(
            Cin => '0',          X => JB_address,          Y => PC_sig,          S => Adder_result,          Gout
=> open,          Pout => open,          Cout => open
        );

    mux: muxnto1_bus
        generic map( n_addr => 1,          bus_width => ctr_width
        )
        port map(
            s => branch_mux_vector,          w => branch_mux_2d_bit_array,          f => PC_load_addr
        );

    nBRANCH      <= not ( PL and ( JB or ( not BC and Z ) or ( BC and N ) ) );
    CE           <= nBRANCH;

    branch_mux_vector( 0 )      <= JB;
    branch_mux_vector_array( 0 ) <= Adder_result;
    branch_mux_vector_array( 1 ) <= JB_address;

    dve_d:
    for i in 0 to ctr_width - 1 generate
        branch_mux_2d_bit_array( 0, i ) <= branch_mux_vector_array( 0 )( i );
        branch_mux_2d_bit_array( 1, i ) <= branch_mux_vector_array( 1 )( i );
    end generate;

    PC           <= PC_sig;
end ideal;

```



```

-- *****
-- **** PREDLOGA VAJE
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

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

--          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

entity 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 branch_ctrl;

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

```

```

type mux_vector_array_type is array ( 1 downto 0 ) of std_logic_vector( ctr_width - 1 downto 0 );
signal      branch_mux_vector_array : mux_vector_array_type := ( others => ( others => '0' ) );
signal      branch_mux_2d_bit_array : muxnto1_bus_type( 1 DOWNT0 0, ctr_width - 1 DOWNT0 0 ) := ( others => ( others
=> '0' ) );
signal      branch_mux_vector : std_logic_vector( 0 downto 0 ); -- PC mode multiplexer control signal

signal      PC_sig      :      std_logic_vector( ctr_width - 1 downto 0 );
signal      PC_load_addr :      std_logic_vector( ctr_width - 1 downto 0 ); -- absolute address load when JUMP
operation
signal      Adder_result :      std_logic_vector( ctr_width - 1 downto 0 );
signal      CE            :      std_logic; -- count enable
signal      nBRANCH       :      std_logic; -- branching control signal ( active '0' )
begin

--      ba - always
--      bneg - on negative          ( N='1' )
--      bpos - on positive          ( N='0' )
--      beq - on zero              ( Z='1' )
--      bne - on not zero          ( Z='0' )
--      bvs - on overflow set      ( V='1' )
--      bvc - on overflow clear    ( V='0' )
--      bcs - on carry set         ( C='1' )
--      bcc - on carry clear       ( C='0' )
--      bl - on less than ( ( N xor V )='1' )
--      ble - on less than or equal ( ( Z or ( N xor V ) )='1' )
--      be - on equal ( Z='1' )
--      bne - on not equal ( Z='0' )
--      bge - on greater than or equal ( ( N xor V )='0' )
--      bg - on greater than ( ( Z or ( N xor V ) )='0' )

--      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

-- nBRANCH = '0' upon successful branch or jump condition

```

```
nBRANCH      <= '0' when ( ( ( BC = '0' and Z = '1' ) or ( BC = '1' and N = '1' ) or JB = '1' ) and PL = '1' ) else
'1';
```

```
-- CE = '1' upon normal operation or unsuccessful branch condition
```

```
CE      <= '1' when ( PL = '0' ) or ( nBRANCH = '1' ) else '0';
```

```
PC_ctr : counter
```

```
    generic map ( ctr_size => ctr_width )
    PORT map (   clk => clk,  -- clock signal
                nCLR => nRST,  -- asynchronous clear ( active = '0' )
                nLOAD => nBRANCH, -- Load counter when nBRANCH = '0'
                ENP => CE,     ENT => CE,  -- PC = PC + 1 when PL = '0'
                x => PC_load_addr, -- counter load value
                Q => PC_sig
    );
```

```
-- branch/jump mode multiplexer control signal ( JB )
```

```
branch_mux_vector_array( 0 ) <= Adder_result;
```

```
branch_mux_vector_array( 1 ) <= JB_address;
```

```
branch_mux_vector( 0 ) <= JB; -- convert std_logic signal to std_logic_vector( 0 )
```

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

```
branch_mux_in_array_process: PROCESS( branch_mux_vector_array, branch_mux_2d_bit_array )
```

```
variable branch_mux_vector_col : std_logic_vector( ctr_width - 1 downto 0 );
```

```
BEGIN
```

```
    FOR i IN 0 TO 1 LOOP
```

```
        branch_mux_vector_col := branch_mux_vector_array( i );
```

```
        FOR j IN 0 TO ctr_width - 1 LOOP
```

```
            branch_mux_2d_bit_array( i, j ) <= branch_mux_vector_col( j );
```

```
        END LOOP;
```

```
    END LOOP;
```

```
END PROCESS;
```

```
BRANCH_MUX: muxnto1_bus
```

```
    generic map (   n_addr=> 1,          bus_width => ctr_width )
    PORT MAP (     s => branch_mux_vector,      w => branch_mux_2d_bit_array,      f =>
PC_load_addr
    );
```

```
ctr_adder : cla_add_n_bit
  generic map ( n => ctr_width )
  PORT map (   Cin => '0',           X => JB_address,           Y => PC_sig,           S => Adder_result
              );

PC      <= PC_sig;

end ideal;
```

