```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity Datapath is
  port(
    rst : in std_logic;
    clk : in std_logic;

    -- Inputs
    ALUSrcA  : in std_logic;
    ALUSrcB  : in std_logic_vector( 1 downto 0 );
    ALUCtrl  : in std_logic_vector( 2 downto 0 );
    RegWrite : in std_logic;
    RegDst   : in std_logic_vector( 1 downto 0 );
    PCSource : in std_logic_vector( 1 downto 0 );
    PCLoad   : in std_logic;
    IorD     : in std_logic;
    MemRead  : in std_logic;
    MemWrite : in std_logic;
    MemtoReg : in std_logic_vector( 1 downto 0 );
    IRWrite  : in std_logic;
    Extend   : in std_logic;
    ReadSrc  : in std_logic;
    PCWriteCond : in std_logic_vector( 2 downto 0 );

    -- Outputs
    OpCode                                    : out std_logic_vector( 5 downto 0 );
    funct                                     : out std_logic_vector( 5 downto 0 );
    BranchType                          : out std_logic);

end Datapath;

-- purpose: put all the components together
architecture bhv of Datapath is

  component Memory
    port(
      rst   : in  std_logic;
      clk   : in  std_logic;
      rd    : in  std_logic;
      wr    : in  std_logic;
      addr  : in  std_logic_vector (31 downto 0);
      d_in  : in  std_logic_vector (31 downto 0);
      d_out : out std_logic_vector (31 downto 0)
      );
  end component;

  component Mux_2to1_xN
    generic (
      WIDTH :     integer := 32);
    port(
      sel   : in  std_logic;
      d_in1 : in  std_logic_vector((WIDTH - 1) downto 0);
      d_in2 : in  std_logic_vector((WIDTH - 1) downto 0);
      d_out : out std_logic_vector((WIDTH - 1) downto 0));
  end component;

  component Mux_4to1_xN
    generic(
      WIDTH :     integer := 32);
    port(
      sel   : in  std_logic_vector(1 downto 0);
      d_in1 : in  std_logic_vector((WIDTH - 1) downto 0);
```

```
    d_in2 : in  std_logic_vector((WIDTH - 1) downto 0);
    d_in3 : in  std_logic_vector((WIDTH - 1) downto 0);
    d_in4 : in  std_logic_vector((WIDTH - 1) downto 0);
    d_out : out std_logic_vector((WIDTH - 1) downto 0));
end component;


component Mux_8to1  -- ********************************
  port(
    sel   : in  std_logic_vector(2 downto 0);
    d_in1 : in  std_logic;
    d_in2 : in  std_logic;
    d_in3 : in  std_logic;
    d_in4 : in  std_logic;
    d_in5 : in  std_logic;
    d_in6 : in  std_logic;
    d_in7 : in  std_logic;
    d_in8 : in  std_logic;
    d_out : out std_logic);
end component;

component Reg_with_load
  generic(
    WIDTH :     integer := 32);
  port(
    rst   : in  std_logic;
    clk   : in  std_logic;
    d_in  : in  std_logic_vector((WIDTH - 1) downto 0);
    load  : in  std_logic;
    d_out : out std_logic_vector((WIDTH - 1) downto 0));
end component;

component Register_File
  port(
    rst       : in  std_logic;
    clk       : in  std_logic;
    RegWrite  : in  std_logic;
    ReadReg1  : in  std_logic_vector(4 downto 0);
    ReadReg2  : in  std_logic_vector(4 downto 0);
    WriteReg  : in  std_logic_vector(4 downto 0);
    WriteData : in  std_logic_vector(31 downto 0);
    ReadData1 : out std_logic_vector(31 downto 0);
    ReadData2 : out std_logic_vector(31 downto 0));
end component;

component RegisterN
  generic(
    WIDTH :     integer := 32);
  port(
    rst   : in  std_logic;
    clk   : in  std_logic;
    d_in  : in  std_logic_vector((WIDTH - 1) downto 0);
    d_out : out std_logic_vector((WIDTH - 1) downto 0));
end component;

component ALU
  port(
    ctrl  : in  std_logic_vector(2 downto 0);
    d_in1 : in  std_logic_vector(31 downto 0);
    d_in2 : in  std_logic_vector(31 downto 0);
    shamt : in  std_logic_vector(4 downto 0);
    d_out : out std_logic_vector(31 downto 0);
    Zero  : out std_logic;
    Positive : out std_logic);
```

```
  end component;

  component Sign_extend
    port(
      d_in  : in  std_logic_vector(15 downto 0);
      d_out : out std_logic_vector(31 downto 0));
  end component;

  component Zero_extend
    port(
      d_in  : in  std_logic_vector(15 downto 0);
      d_out : out std_logic_vector(31 downto 0));
  end component;

  component Shift_left_2
    generic(
      WIDTH :     integer := 32);
    port(
      d_in  : in  std_logic_vector((WIDTH - 1) downto 0);
      d_out : out std_logic_vector((WIDTH - 1) downto 0));
  end component;

-- outputs of ALU

  signal Positive   : std_logic;
  signal Zero       : std_logic;
  signal NotZero    : std_logic;
  signal ZeroNorPositive : std_logic;

-- output of ReadReg2_MUX
  signal ReadReg2 : std_logic_vector (4 downto 0);

-- output of PC
  signal PC_out         : std_logic_vector(31 downto 0);

-- memory Address, connect to output of MUX
  signal Address        : std_logic_vector(31 downto 0);

-- data read from Memory, connect to input of IR and MDR
  signal MemData        : std_logic_vector(31 downto 0);

-- output of IR, gets split A LOT
  signal IR_out         : std_logic_vector(31 downto 0);

-- output of MDR, goes to a mux
  signal MDR_out        : std_logic_vector(31 downto 0);

-- output from mux to write data line on reg32
  signal WriteData      : std_logic_vector(31 downto 0);

-- out from mux, register to write to on big RF
  signal WriteReg       : std_logic_vector(4 downto 0);

-- outputs of the reg32
  signal ReadData1      : std_logic_vector(31 downto 0);
  signal ReadData2      : std_logic_vector(31 downto 0);

-- outputs of delay registers A and B
  signal A_out          : std_logic_vector(31 downto 0);
  signal B_out          : std_logic_vector(31 downto 0);


-- output of sign extender
  signal sign_extended  : std_logic_vector(31 downto 0);
```

```
-- output of zero extender
  signal zero_extended  : std_logic_vector(31 downto 0);

-- output from mux_extender
  signal extender_mux_signal   : std_logic_vector(31 downto 0);

-- output of 32 bit shifter
  signal shifted_left_2 : std_logic_vector(31 downto 0);

-- inputs to ALU
  signal ALU_in1        : std_logic_vector(31 downto 0);
  signal ALU_in2        : std_logic_vector(31 downto 0);

-- out of ALU
  signal ALU_result     : std_logic_vector(31 downto 0);

-- calc the addr to jump to
  signal JumpAddr       : std_logic_vector(31 downto 0);

-- the output of the ALUOut register
  signal ALU_out        : std_logic_vector(31 downto 0);

-- input to PC, out of last mux
  signal PC_in          : std_logic_vector(31 downto 0);

  signal const_0_32bit  : std_logic_vector(31 downto 0);
  signal const_0_5bit            : std_logic_vector(4 downto 0);
  signal const_0        : std_logic;
  signal const_4        : std_logic_vector(31 downto 0);
  signal const_X        : std_logic_vector(31 downto 0);
  signal const_31       : std_logic_vector(4 downto 0);

begin  -- bhv

  -- put everything together:

  OpCode                <= IR_out(31 downto 26);
  funct                 <= IR_out(5 downto 0);
  JumpAddr(31 downto 0) <= PC_out(31 downto 28) & IR_out(25 downto 0) & "00";
  NotZero                       <= not Zero;
  ZeroNorPositive                       <= Zero nor Positive;
  const_4                       <= (2 => '1', others => '0');
  const_31                      <= "11111";
  const_0_32bit                 <= (others => '0');
  const_X                       <= (others => 'X');
  const_0                       <= '0';
  const_0_5bit                  <= (others => '0');


  ReadReg2_MUX : Mux_2to1_xN
    generic map(
      WIDTH => 5)
    port map(
      sel   => ReadSrc,
      d_in1 => IR_out(20 downto 16),
      d_in2 => const_0_5bit,
      d_out => ReadReg2);

  Branch_MUX : Mux_8to1
    port map(
      sel   => PCWriteCond,
      d_in1 => Positive,
      d_in2 => ZeroNorPositive,
```

```
    d_in3 => NotZero,
    d_in4 => Zero,
    d_in5 => const_0,
    d_in6 => const_0,
    d_in7 => const_0,
    d_in8 => const_0,
    d_out => BranchType);

MEM : Memory
  port map (
    rst   => rst,
    clk   => clk,
    rd    => MemRead,
    wr    => MemWrite,
    addr  => Address,
    d_in  => B_out,
    d_out => MemData);

IR : Reg_with_load
  port map(
    rst   => rst,
    clk   => clk,
    d_in  => MemData,
    load  => IRWrite,
    d_out => IR_out );

MDR : RegisterN
  port map(
    rst   => rst,
    clk   => clk,
    d_in  => MemData,
    d_out => MDR_out);

WriteData_MUX : Mux_4to1_xN
  port map(
    sel   => MemtoReg,
    d_in1 => ALU_out,
    d_in2 => MDR_out,
    d_in3 => PC_out,
    d_in4 => const_0_32bit,
    d_out => WriteData);

WriteReg_MUX : Mux_4to1_xN
  generic map(
    WIDTH => 5)
  port map(
    sel   => RegDst(1 downto 0),
    d_in1 => IR_out(20 downto 16),
    d_in2 => IR_out(15 downto 11),
    d_in3 => const_31,
    d_in4 => const_31,
    d_out => WriteReg(4 downto 0));

Extender_MUX : Mux_2to1_xN
  generic map(
    WIDTH => 32)
  port map(
    sel   => Extend,
    d_in1 => zero_extended(31 downto 0),
    d_in2 => sign_extended(31 downto 0),
    d_out => extender_mux_signal);

Register_File_ins : Register_File
  port map(
```

```
    rst       => rst,
    clk       => clk,
    RegWrite  => RegWrite,
    ReadReg1  => IR_out(25 downto 21),
    ReadReg2  => ReadReg2,
    WriteReg  => WriteReg,
    WriteData => WriteData,
    ReadData1 => ReadData1,
    ReadData2 => ReadData2 );

REG_A : RegisterN
  port map(
    rst   => rst,
    clk   => clk,
    d_in  => ReadData1,
    d_out => A_out);

REG_B : RegisterN
  port map(
    rst   => rst,
    clk   => clk,
    d_in  => ReadData2,
    d_out => B_out );

Sign_extend_ins : Sign_extend
  port map(
    d_in  => IR_out(15 downto 0),
    d_out => sign_extended);

Zero_extend_ins : Zero_extend
  port map(
    d_in  => IR_out(15 downto 0),
    d_out => zero_extended);

Shift_left_2_ins : Shift_left_2
  port map(
    d_in  => sign_extended,
    d_out => shifted_left_2);

ALUSrcA_MUX : Mux_2to1_xN
  port map(
    sel   => ALUSrcA,
    d_in1 => PC_out,
    d_in2 => A_out,
    d_out => ALU_in1);

ALUSrcB_MUX : Mux_4to1_xN
  port map(
    sel   => ALUSrcB,
    d_in1 => B_out,
    d_in2 => const_4,
    d_in3 => extender_mux_signal,
    d_in4 => shifted_left_2,
    d_out => ALU_in2);

ALU_ins : ALU
  port map(
    ctrl  => ALUCtrl,
    d_in1 => ALU_in1,
    d_in2 => ALU_in2,
    shamt => IR_out (10 downto 6),
    d_out => ALU_result,
    Zero  => Zero,
    Positive => Positive);
```

```
  ALU_out_REG : RegisterN
    port map(
      rst   => rst,
      clk   => clk,
      d_in  => ALU_result,
      d_out => ALU_out);

  PC_MUX : Mux_4to1_xN
    port map(
      sel   => PCSource,
      d_in1 => ALU_result,
      d_in2 => ALU_out,
      d_in3 => JumpAddr,
      d_in4 => A_out,
      d_out => PC_in);

  PC : Reg_with_load
    port map(
      rst   => rst,
      clk   => clk,
      d_in  => PC_in,
      load  => PCLoad,
      d_out => PC_out );

  MUX_FEEDING_MEM : Mux_2to1_xN
    port map(
      sel   => IorD,
      d_in1 => PC_out,
      d_in2 => ALU_out,
      d_out => Address);

end bhv;
```