ECE 585 – Computer Organization and Design

Project 2 – MIPS CPU Design and Implementation (Pipeline)

I acknowledge all works including figures, codes and writings belong to me and/or persons who are referenced. I understand if any similarity in the code, comments, customized program behavior, report writings and/or figures are found, both the helper (original work) and the requestor (duplicated/modified work) will be called for academic disciplinary action.

Anand Baraguru Venkateshmurthy

Abstract

The main objective of study is to design and implement 32-bit RISC processor, a stripped-down MIPS processor.

Pipeline datapath 32-bit version of the MIPS processor is implemented using VHDL hardware descriptive language. I have used Xilinx ISE (WebPack) VHDL simulator for implementing this.

In this project I have tried to test the operation of the above said pipeline datapath using R-type, I-type and J-type instructions.

Introduction

The purpose of this project is to design and implement Pipeline datapath 32-bit version of the MIPS processor is implemented using VHDL hardware descriptive language.

I started with implementing the datapath first without the memory and control interface, along with 5-stage MIPS pipeline ability. 5-stage MIPS pipeline has following steps per stage:

1. IF: Instruction fetch from memory
2. ID: Instruction decode & register read
3. EX: Execute operation or calculate address
4. MEM: Access memory operand
5. WB: Write result back to register

R-type, I-type and J-type instruction types are used in implementing the datapath.

Background

The below figure 1 shows the datapath with pipelined control.

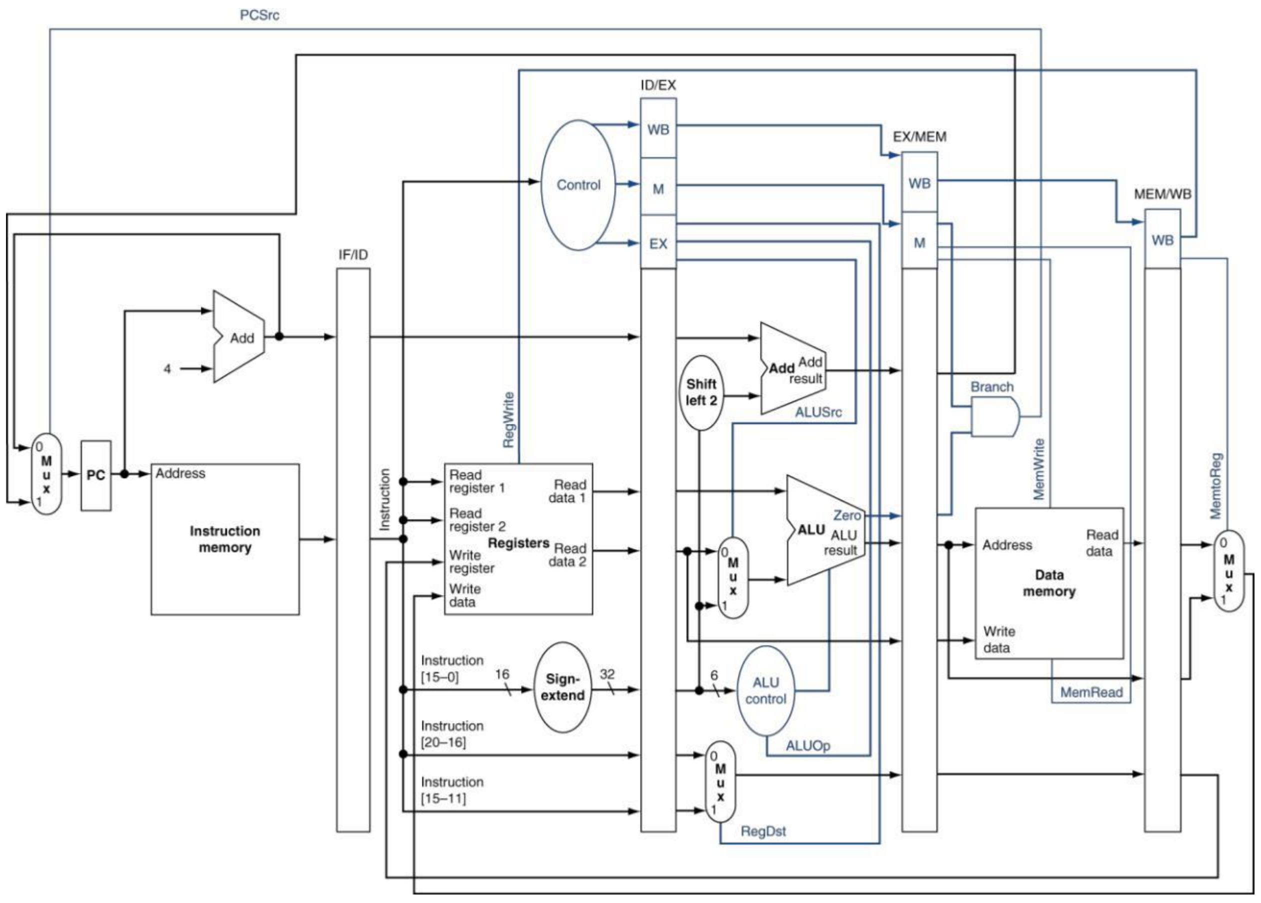

Fig 1. Datapath with Pipelined Control

MIPS instructions classically take five steps:

1. Fetch instruction from memory (IF stage).
2. Read registers while decoding the instruction (ID stage).
3. Execute the operation or calculate an address (EX stage).
4. Access an operand in data memory (MEM stage).
5. Write the result into a register (WB stage).

Table I. Required MIPS Instruction Set

Code Sequence	OpCode [31:26]	Function Field [5:0]	Instruction	Operation (example)
1	000000	100010	sub	sub \$2, \$1, \$3
2	000000	100100	and	and \$12, \$2, \$5
3	000000	100101	or	or \$13, \$6, \$2
4	000000	100000	add1 (RCA)	add1 \$14, \$2, \$2
5	000000	100000	add2 (CLA)	add2 \$14, \$13, \$2
6	100011	-	lw	lw \$15, 100(\$2)
7	001000	-	addi	addi \$3, \$5, 200
8	000010	-	j	j 2500

Table 1 above shows the MIPS instruction set I have implemented in this project.

Key components in the design are as described below:

ALU

Arithmetic Logic unit (ALU) is a digital electronic circuit that performs arithmetic and logical operations. It operates on two inputs called operands and provides output based on the code which defines the operation of ALU.

Memory

Memory module is used for storing data and instruction. To enable read and write operation multiplexor is used.

PC

Program Counter (PC) is a 32-bit register that contains the address of the instruction being executed. After the execution of the current instruction, PC will be incremented to the next instruction.

Register File

Register file is an array holding all registers. To read from the register or write to the register, control signal is used.

Instruction Register

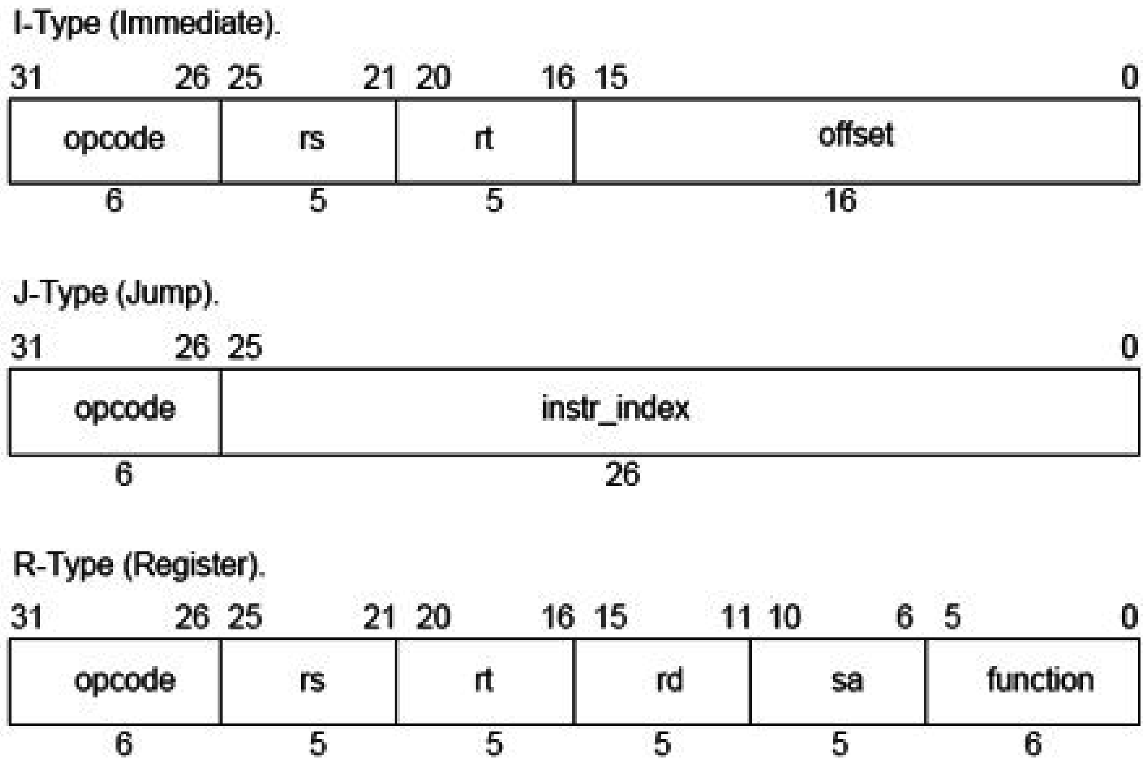

Fig 2. I-type, J-type and R-type instruction format

The above figure 2 shows the I-type, J-type and R-type instruction format.

The hexadecimal representation of the instructions shown in table 1 is as follows:

sub \$2, \$1, \$3	=	0X00231022
and \$12, \$2, \$5	=	0X00456024
or \$13, \$6, \$2	=	0X00C26825
lw \$15, 100(\$2)	=	0X8C4F0064
addi \$3, \$5, 200	=	0X20A300C8
j 2500	=	0X08000271

MUX

Various muxs like 2 to 1 MUX, 3 to 1 MUX and 4 to 1 MUX are used for selecting different inputs for relevant blocks. MUXs select the appropriate inputs based on the control signals.

System Design

First I started with ALU, Full Adder, Register file, Sign-extend block, Mux. Then I started designing memory and control interface along with pipelining capability.

Datapath design was a tedious task. First I designed Full Adders, Mux, Sign-extend blocks and register files were designed first and later ALU.

Control logic and memory was designed after datapath design.

Figure 3 below shows the finite state machine for control logic.

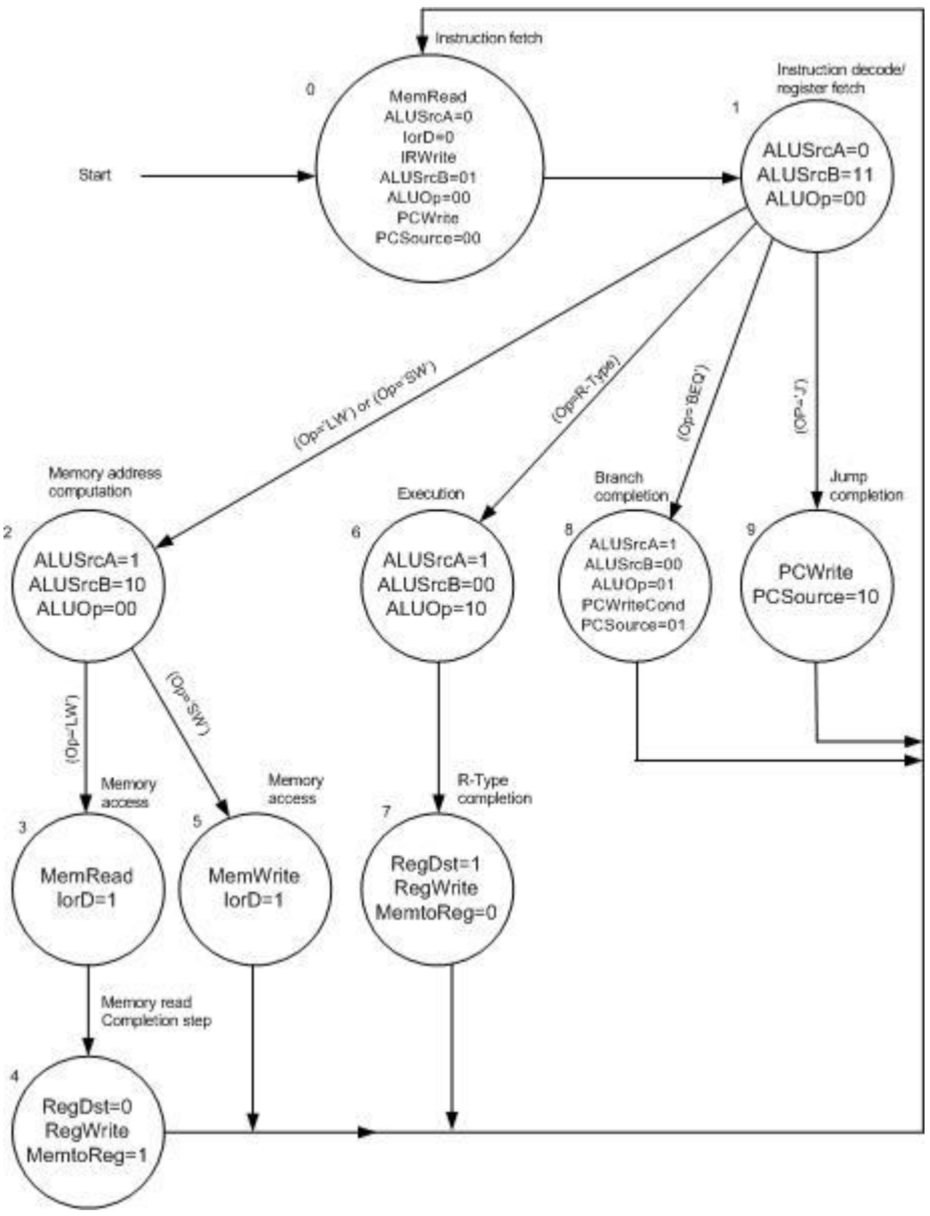

Fig 3. Finite State Machine for Control logic

Simulation Results and discussion

Due to time constraint I was not able to complete the project. I have implemented the datapath, memory and control logic. But due to errors in my code I was not able to simulate and check their operation.

The processor I have tried to implement is 32-bit which is not real implementation because overflow detection circuit which should be used while dealing with large data which leads to overflow of final result. In addition to that hazard detection is not considered while implementing the design. So I would like improve my current design with hazard detection units and overflow detection so that it will be a real time implementation.

Conclusion

In this project, I got to know how the instructions are fetched, decoded and executed in MIPS 5 stage pipeline, how the intermediate results flow between various blocks and how the final output is deduced by combining all the intermediate results.

APPENDIX

ALU:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity alu_32 is
        port( input_A        : in  std_logic_vector(31 downto 0);
              input_B        : in  std_logic_vector(31 downto 0);
              input_ALUOP    : in  std_logic_vector(3  downto 0);
              output_CarryOut : out std_logic;
              output_F       : out std_logic_vector(31 downto 0);
          );
end alu_32;

architecture structure of alu_32 is
    component and2_32 is
        port( input_A : in  std_logic_vector(31 downto 0);
              input_B : in  std_logic_vector(31 downto 0);
              output_F : out std_logic_vector(31 downto 0));
    end component;

    component or2_32bit is
        port( input_A : in  std_logic_vector(31 downto 0);
              input_B : in  std_logic_vector(31 downto 0);
              output_F : out std_logic_vector(31 downto 0));
    end component;

    component addsub_32bit is
        port( input_A         : in std_logic_vector(31 downto 0);
              input_B         : in std_logic_vector(31 downto 0);
              input_nAdd_Sub  : in std_logic;
              output_Cout      : out std_logic;
              output_S         : out std_logic_vector(31 downto 0) );
    end component;

    component slt_32bit is
        port( input_SubF    : in std_logic_vector(31 downto 0);
              input_OVF     : in std_logic;
              output_F      : out std_logic_vector(31 downto 0) );
    end component;



    component nor2_32bit is
        port( input_A : in  std_logic_vector(31 downto 0);
              input_B : in  std_logic_vector(31 downto 0);
              output_F : out std_logic_vector(31 downto 0));
    end component;

    component xor2_32bit is
        port( input_A : in  std_logic_vector(31 downto 0);
              input_B : in  std_logic_vector(31 downto 0);
              output_F : out std_logic_vector(31 downto 0));
    end component;

    component mux7to1_32bit is
        port( input_SEL : in std_logic_vector(3 downto 0);
              input_0   : in std_logic_vector(31 downto 0);
              input_1   : in std_logic_vector(31 downto 0);
              input_2   : in std_logic_vector(31 downto 0);
              input_3   : in std_logic_vector(31 downto 0);
              input_4   : in std_logic_vector(31 downto 0);
```

```
            input_5   : in std_logic_vector(31 downto 0);
            input_6   : in std_logic_vector(31 downto 0);
            output_F   : out std_logic_vector(31 downto 0) );
    end component;



    signal mux0_in, mux1_in, mux2_in, mux3_in, mux4_in, mux5_in, mux6_in
        : std_logic_vector(31 downto 0);

    signal signal_F : std_logic_vector(31 downto 0);

    signal signal_zero, signal_carry : std_logic;

begin

    AND_OP: and2_32bit
        port map (input_A, input_B, mux0_in);

    OR_OP: or2_32bit
        port map (input_A, input_B, mux1_in);

    ARITH_OP: addsub_32bit
        port map (input_A, input_B, input_ALUOP(2), signal_carry, mux2_in);




    SLT_OP: slt_32bit
        port map (mux2_in, signal_ovf, mux3_in);

    NOR_OP: nor2_32bit
```

```
      port map (input_A, input_B, mux5_in);


   XOR_OP: xor2_32bit
      port map (input_A, input_B, mux6_in);


   SELECT_OPERATION: mux7to1_32bit
      port map (input_ALUOP, mux0_in, mux1_in, mux2_in, mux3_in, mux4_in, mux5_in,
mux6_in,
        );
   output_CarryOut <= signal_carry;
end structure;
```

Brach Logic:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity branch_Jump_logic is
   port(input_J              : in std_logic;
input_PCPlus4         : in std_logic_vector(31 downto 0);
input_BNE             : in std_logic;
input_Instruc_25to0    : in std_logic_vector(25 downto 0);
input_JAL             : in std_logic;
input_Zeroutput_Flag       : in std_logic;
input_RD1             : in std_logic_vector(31 downto 0);
input_JR              : in std_logic;
input_IMM             : in std_logic_vector(31 downto 0);
input_BEQ             : in std_logic;
        output_Binput_J_Address      : out std_logic_vector(31 downto 0);
        output_PCSrc           : out std_logic;
        output_BranchTaken      : out std_logic;
        output_Branch          : out std_logic );
```

```
end branch_Jump_logic;

architecture structural of branch_Jump_logic is
    component fulladder_32bit is
        port( input_A    : in std_logic_vector(31 downto 0);
              input_B    : in std_logic_vector(31 downto 0);
              input_Cin  : in std_logic;
              output_Cout : out std_logic;
              output_S    : out std_logic_vector(31 downto 0) );
    end component;

    component mux2to1_32bit is
        port( input_X   : in std_logic_vector(31 downto 0);
              input_Y   : in std_logic_vector(31 downto 0);
              input_SEL : in std_logic;
              output_OUT   : out std_logic_vector(31 downto 0) );
    end component;

    component or2_1bit is
      port(input_A          : in std_logic;
           input_B          : in std_logic;
           output_F          : out std_logic);
    end component;

    component and2_1bit is
      port(input_A          : in std_logic;
           input_B          : in std_logic;
           output_F          : out std_logic);
    end component;
```

```
   component sel_input_BEQ_input_BNE is
      port( input_Zeroutput_Flag : in std_logic;
            input_SELect    : in std_logic_vector(1 downto 0);
            output_F        : out std_logic );
   end component;

   signal signal_Cout_IMM, signal_AND_Out, signal_OR_input_J, signal_OR_input_BEQinput_BNE, signal_sel_br_out : std_logic;
   signal signal_AddIMM_Out, signal_Mux1_Out, signal_Mux2_Out, signal_Mux3_Out : std_logic_vector(31 downto 0);
   signal signal_IMM_Shift, signal_input_J_addr : std_logic_vector(31 downto 0);
   signal signal_i_Instruc_25to0_sl2 : std_logic_vector(27 downto 0);
   signal signal_input_BEQinput_BNE : std_logic_vector(1 downto 0);

begin

   signal_IMM_Shift <= input_IMM(29 downto 0) & "00";
   signal_input_BEQinput_BNE <= input_BEQ & input_BNE;
   signal_i_Instruc_25to0_sl2 <= input_Instruc_25to0 & "00";
   signal_input_J_addr <= input_PCPlus4(31 downto 28) & signal_i_Instruc_25to0_sl2;

   output_Binput_J_Address <= signal_Mux3_Out;
   output_PCSrc <= '1' when (input_BEQ = '1' or input_BNE = '1' or input_JAL = '1' or input_J = '1' or input_JR = '1') else
            '0';

   output_BranchTaken <= signal_AND_Out;
   output_Branch <= signal_OR_input_BEQinput_BNE;

   add_IMM: fulladder_32bit
      port map (input_PCPlus4, signal_IMM_Shift, '0', signal_Cout_IMM, signal_AddIMM_Out);
```

```
  mux1: mux2to1_32bit
    port map (input_PCPlus4, signal_AddIMM_Out, signal_AND_Out, signal_Mux1_Out);


  mux2: mux2to1_32bit
    port map (signal_Mux1_Out, signal_input_J_addr, signal_OR_input_J,
signal_Mux2_Out);


  mux3: mux2to1_32bit
    port map (signal_Mux2_Out, input_RD1, input_JR, signal_Mux3_Out);


  or_input_BEQ_input_BNE: or2_1bit
    port map (input_BEQ, input_BNE, signal_OR_input_BEQinput_BNE);


  and_Z: and2_1bit
    port map (signal_OR_input_BEQinput_BNE, signal_sel_br_out, signal_AND_Out);


  selinput_BEQinput_BNE: sel_input_BEQ_input_BNE
    port map (input_Zeroutput_Flag, signal_input_BEQinput_BNE, signal_sel_br_out);


  or_input_J: or2_1bit
    port map (input_J, input_JAL, signal_OR_input_J);


end structural;
```

Register File:

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


entity register_file is
  port( input_CLK     : in std_logic;
        input_RST     : in std_logic;
```

```
          input_WR        : in std_logic_vector(4 downto 0);
          input_WD        : in std_logic_vector(31 downto 0);
          input_REGWRITE  : in std_logic;
          input_RR1       : in std_logic_vector(4 downto 0);
          input_RR2       : in std_logic_vector(4 downto 0);
          output_RD1       : out std_logic_vector(31 downto 0);
          output_RD2       : out std_logic_vector(31 downto 0) );
end register_file;

architecture structure of register_file is

    component register_32bit_rf
        port( input_CLK  : in std_logic;
              input_RST  : in std_logic;
              input_WD   : in std_logic_vector(31 downto 0);
              input_WE   : in std_logic;
              output_Q    : out std_logic_vector(31 downto 0) );
    end component;

    component decode_5to32bit
        port( input_D : in std_logic_vector(4 downto 0);
              output_F : out std_logic_vector(31 downto 0) );
    end component;

    component mux32to1_32bit is
        port( input_SEL : in std_logic_vector(4 downto 0);
              input_0   : in std_logic_vector(31 downto 0);
              input_1   : in std_logic_vector(31 downto 0);
              input_2   : in std_logic_vector(31 downto 0);
              input_3   : in std_logic_vector(31 downto 0);
              input_4   : in std_logic_vector(31 downto 0);
              input_5   : in std_logic_vector(31 downto 0);
              input_6   : in std_logic_vector(31 downto 0);
              input_7   : in std_logic_vector(31 downto 0);
              input_8   : in std_logic_vector(31 downto 0);
              input_9   : in std_logic_vector(31 downto 0);
              input_10  : in std_logic_vector(31 downto 0);
              input_11  : in std_logic_vector(31 downto 0);
              input_12  : in std_logic_vector(31 downto 0);
              input_13  : in std_logic_vector(31 downto 0);
              input_14  : in std_logic_vector(31 downto 0);
              input_15  : in std_logic_vector(31 downto 0);
              input_16  : in std_logic_vector(31 downto 0);
              input_17  : in std_logic_vector(31 downto 0);
              input_18  : in std_logic_vector(31 downto 0);
              input_19  : in std_logic_vector(31 downto 0);
              input_20  : in std_logic_vector(31 downto 0);
              input_21  : in std_logic_vector(31 downto 0);
              input_22  : in std_logic_vector(31 downto 0);
              input_23  : in std_logic_vector(31 downto 0);
              input_24  : in std_logic_vector(31 downto 0);
              input_25  : in std_logic_vector(31 downto 0);
              input_26  : in std_logic_vector(31 downto 0);
              input_27  : in std_logic_vector(31 downto 0);
              input_28  : in std_logic_vector(31 downto 0);
              input_29  : in std_logic_vector(31 downto 0);
              input_30  : in std_logic_vector(31 downto 0);
              input_31  : in std_logic_vector(31 downto 0);
              output_F   : out std_logic_vector(31 downto 0) );
    end component;
```

```
    component and2_1bit is
      port(input_A          : in std_logic;
           input_B          : in std_logic;
           output_F         : out std_logic);
    end component;

signal signal_decoded : std_logic_vector(31 downto 0);
signal signal_write : std_logic_vector(31 downto 0);

type vector32 is array (natural range<>) of std_logic_vector(31 downto 0);
signal signal_register_data: vector32(31 downto 0);

begin

decode_WR: decode_5to32bit
    port map(input_WR, signal_decoded);

generate_registers: for i in 1 to 31 generate
    do_write: and2_1bit
        port map(signal_decoded(i), input_REGWRITE, signal_write(i));
    reg: register_32bit_rf
        port map(input_CLK, input_RST, input_WD, signal_write(i), signal_register_data(i));
end generate;

reg_0: register_32bit_rf
    port map(input_CLK, '1', (others => '0'), '0', signal_register_data(0));


mux_RD1: mux32to1_32bit
```

```
port map ( input_SEL => input_RR1,
      input_0   => signal_register_data(0),
      input_1   => signal_register_data(1),
      input_2   => signal_register_data(2),
      input_3   => signal_register_data(3),
      input_4   => signal_register_data(4),
      input_5   => signal_register_data(5),
      input_6   => signal_register_data(6),
      input_7   => signal_register_data(7),
      input_8   => signal_register_data(8),
      input_9   => signal_register_data(9),
      input_10  => signal_register_data(10),
      input_11  => signal_register_data(11),
      input_12  => signal_register_data(12),
      input_13  => signal_register_data(13),
      input_14  => signal_register_data(14),
      input_15  => signal_register_data(15),
      input_16  => signal_register_data(16),
      input_17  => signal_register_data(17),
      input_18  => signal_register_data(18),
      input_19  => signal_register_data(19),
      input_20  => signal_register_data(20),
      input_21  => signal_register_data(21),
      input_22  => signal_register_data(22),
      input_23  => signal_register_data(23),
      input_24  => signal_register_data(24),
      input_25  => signal_register_data(25),
      input_26  => signal_register_data(26),
      input_27  => signal_register_data(27),
      input_28  => signal_register_data(28),
        input_29  => signal_register_data(29),
        input_30  => signal_register_data(30),
        input_31  => signal_register_data(31),
        output_F   => output_RD1 );

mux_RD2: mux32to1_32bit
    port map ( input_SEL => input_RR2,
        input_0   => signal_register_data(0),
        input_1   => signal_register_data(1),
        input_2   => signal_register_data(2),
        input_3   => signal_register_data(3),
        input_4   => signal_register_data(4),
        input_5   => signal_register_data(5),
        input_6   => signal_register_data(6),
        input_7   => signal_register_data(7),
        input_8   => signal_register_data(8),
        input_9   => signal_register_data(9),
        input_10  => signal_register_data(10),
        input_11  => signal_register_data(11),
        input_12  => signal_register_data(12),
        input_13  => signal_register_data(13),
        input_14  => signal_register_data(14),
        input_15  => signal_register_data(15),
        input_16  => signal_register_data(16),
        input_17  => signal_register_data(17),
        input_18  => signal_register_data(18),
        input_19  => signal_register_data(19),
        input_20  => signal_register_data(20),
        input_21  => signal_register_data(21),
        input_22  => signal_register_data(22),
        input_23  => signal_register_data(23),
        input_24  => signal_register_data(24),
        input_25  => signal_register_data(25),
        input_26  => signal_register_data(26),
        input_27  => signal_register_data(27),
        input_28  => signal_register_data(28),
        input_29  => signal_register_data(29),
        input_30  => signal_register_data(30),
        input_31  => signal_register_data(31),
        output_F   => output_RD2 );

end structure;
```

Control Logic:

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

entity control is
    port( input_Instruction    : in std_logic_vector(31 downto 0);
          output_ALUOP                : out std_logic_vector(3 downto 0);
           output_ALUSrc         : out std_logic;
                                          output_BEQ           : out std_logic;
          output_RegDst         : out std_logic;
          output_Mem_To_Reg     : out std_logic;
          output_MemWrite       : out std_logic;
          output_output_JR      : out std_logic;
          output_RegWrite       : out std_logic;
          output_BNE            : out std_logic;
          output_J              : out std_logic;
          output_JAL            : out std_logic;
```

```
                                    output_Sel_ALU_A_Mux2 : out std_logic;
     output_MemRead       : out std_logic );
end control;



architecture dataflow of control is
  signal op, funct : std_logic_vector(5 downto 0);
  signal all_outputs : std_logic_vector(8 downto 0);

begin
  process (input_Instruction, op, funct)
  begin
    op <= input_Instruction(31 downto 26);
    funct <= input_Instruction(5 downto 0);
                              output_MemRead <= '0';
    output_Sel_ALU_A_Mux2 <= '0';
    output_BEQ <= '0';
    output_J <= '0';
    output_JAL <= '0';
    output_output_JR <= '0';
                              output_BNE <= '0';



    if op = "000000" then
    -- R-type
      if funct = "101010" then
        all_outputs <= "001110011"; -- slt
      elsif funct = "101011" then
        all_outputs <= "001110011"; -- sltu
      elsif funct = "100000" then
    all_outputs <= "000100011"; -- and
elsif funct = "100001" then
    all_outputs <= "000100011"; -- addu
elsif funct = "100100" then
    all_outputs <= "000000011"; -- and
elsif funct = "100110" then
    all_outputs <= "011010011"; -- xor
elsif funct = "100101" then
    all_outputs <= "000010011"; -- or
elsif funct = "100111" then
    all_outputs <= "011000011"; -- nor
elsif funct = "000000" then
    all_outputs <= "010010011"; -- sll
    output_Sel_ALU_A_Mux2 <= '1';
elsif funct = "000010" then
    all_outputs <= "010000011"; -- srl
    output_Sel_ALU_A_Mux2 <= '1';
elsif funct = "000011" then
    all_outputs <= "010100011"; -- sra
    output_Sel_ALU_A_Mux2 <= '1';
elsif funct = "000100" then
    all_outputs <= "010010011"; -- sllv
elsif funct = "000110" then
    all_outputs <= "010000011"; -- srlv
elsif funct = "000111" then
    all_outputs <= "010100011"; -- srav
elsif funct = "100010" then
    all_outputs <= "001100011"; -- sub
elsif funct = "100011" then
    all_outputs <= "001100011"; -- subu
```

```
        elsif funct = "001000" then
            all_outputs <= "000000000"; -- output_Jr
            output_output_JR <= '1';
        else
            all_outputs <= "111111111";
        end if;
    else
    -- I-or-output_J-type
        if op = "001000" then
            all_outputs <= "000100110"; -- addi
        elsif op = "001001" then
            all_outputs <= "000100110"; -- addiu
        elsif op = "001100" then
            all_outputs <= "000000110"; -- andi
        elsif op = "001111" then
            all_outputs <= "010010110"; -- lui
            output_Sel_ALU_A_Mux2 <= '1';
        elsif op = "100011" then
            all_outputs <= "100100110"; -- lw
            output_MemRead <= '1';
        elsif op = "001110" then
            all_outputs <= "011010110"; -- xori
        elsif op = "001101" then
            all_outputs <= "000010110"; -- ori
        elsif op = "001010" then
            all_outputs <= "001110110"; -- slti
        elsif op = "001011" then
            all_outputs <= "001110110"; -- sltiu
        elsif op = "101011" then
            all_outputs <= "100101100"; -- sw
            elsif op = "000100" then
                all_outputs <= "001100000"; -- BEQ output
                output_BEQ <= '1';
            elsif op = "000101" then
                all_outputs <= "001100000"; -- BNE output
                output_BNE <= '1';
            elsif op = "000010" then
                all_outputs <= "000000000"; -- J outpur
                output_J <= '1';
            elsif op = "000011" then
                all_outputs <= "000000000"; -- Jal output
                output_JAL <= '1';
            else
                all_outputs <= "111111110";
            end if;
        end if;

    end process;



output_Mem_To_Reg <= all_outputs(8);
    output_ALUOP <= all_outputs(7 downto 4);
    output_MemWrite <= all_outputs(3);
  output_ALUSrc <= all_outputs(2);
    output_RegWrite <= '1' when (op = "000011") else
                        all_outputs(1);
    output_RegDst <= all_outputs(0);

end dataflow;
```

Memory:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mem is

            generic
            (
                DATA_WIDTH : natural := 32;
                ADDR_WIDTH : natural := 10
            );

            port
            (
                clk         : in std_logic;
                addr    : in natural range 0 to 2**ADDR_WIDTH - 1;
                data    : in std_logic_vector((DATA_WIDTH-1) downto 0);
                we          : in std_logic := '1';
                q           : out std_logic_vector((DATA_WIDTH -1) downto 0)
            );

end mem;

architecture rtl of mem is


            subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
            type memory_t is array(2**ADDR_WIDTH-1 downto 0) of word_t;
            signal ram : memory_t;
```

```
                attribute ram_init_file : string;
                attribute ram_init_file of ram : signal is "dmem.mif";

begin
--sub $2, $1, $3
-- 0000 0000 0010 0011 0001 0000 0010 0010
ram(addr)<="00000000";
ram(addr)<="00100011";
ram(addr)<="00010000";
ram(addr)<="00100010";

--add $12, $2, $5
-- 0000 0000 0100 0101 0110 0000 0010 0100
ram(addr)<="00000000";
ram(addr)<="01000101";
ram(addr)<="01101000";
ram(addr)<="00100101";

--or $13, $6, $2
-- 0000 0000 1100 0010 0110 1000 0010 0101
ram(addr)<="00000000";
ram(addr)<="11000010";
ram(addr)<="01101000";
ram(addr)<="00100101";

--lw $15, 100($2)
--1000 1100 0100 1111 0000 0000 0110 0100
ram(addr)<="10001100";
ram(addr)<="01001111";
ram(addr)<="00000000";
ram(addr)<="01100100";

--addi $3, $5, 200
--0010 0000 1010 0011 0000 0000 1100 1000
ram(addr)<="00100000";
ram(addr)<="10100011";
ram(addr)<="00000000";
ram(addr)<="11001000";

--j2500
--0000 1000 0000 0000 0000 0010 0111 0001
ram(addr)<="00001000";
ram(addr)<="00000000";
ram(addr)<="00000010";
ram(addr)<="01110001";

                    process(clk)
                    begin
                    if(rising_edge(clk)) then
                       if(we = '1') then
                               ram(addr) <= data;
                       end if;
                    end if;
                    end process;

                    q <= ram(addr);
end rtl;
```

Instruction Fetch:

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

entity instruction_fetch is
   port( input_Reset          : in std_logic;
         input_Clock          : in std_logic;
         input_Stall_PC       : in std_logic;
         input_BranchJ_Addr   : in std_logic_vector(31 downto 0);
         input_Mux_Sel        : in std_logic;
         output_Instruction   : out std_logic_vector(31 downto 0);
         output_PCPlus4       : out std_logic_vector(31 downto 0) );
end instruction_fetch;

architecture structural of instruction_fetch is
   component register_32bit is
      port( input_CLK  : in std_logic;
            input_RST  : in std_logic;
            input_WD   : in std_logic_vector(31 downto 0);
            input_WE   : in std_logic;
            output_Q   : out std_logic_vector(31 downto 0) );
   end component;

   component mem is
                        generic ( DATA_WIDTH : natural := 32; ADDR_WIDTH : natural := 10
);
                        port ( clk  : in std_logic;
                               addr  : in natural range 0 to 2**ADDR_WIDTH - 1;
                               data  : in std_logic_vector((DATA_WIDTH-1) downto 0);
                               we    : in std_logic := '1';
```

```
                          q      : out std_logic_vector((DATA_WIDTH-1) downto 0) );
  end component;


  component fulladder_32bit is
     port( input_A    : in std_logic_vector(31 downto 0);
           input_B    : in std_logic_vector(31 downto 0);
           input_Cin  : in std_logic;
           output_Cout : out std_logic;
           output_S    : out std_logic_vector(31 downto 0) );
  end component;


  component mux2to1_32bit is
     port( input_X   : in std_logic_vector(31 downto 0);
           input_Y   : in std_logic_vector(31 downto 0);
           input_SEL : in std_logic;
           output_OUT   : out std_logic_vector(31 downto 0) );
  end component;


  signal signal_Cout_PC4, signal_write_pc : std_logic;
  signal signal_convert_addr : std_logic_vector(29 downto 0);
  signal signal_PC_Out, signal_AddPC4_Out, signal_Four, signal_MemData_Placehold,
signal_PC_WD : std_logic_vector(31 downto 0) := (others => '0');
  signal signal_convert_to_nat : natural range 0 to 2**10 - 1;


begin
  signal_Four <= (2 => '1', others => '0');
  signal_MemData_Placehold <= (others => '0');
  signal_convert_addr <= "00000000000000000000" & signal_PC_Out(11 downto 2);
  signal_convert_to_nat <= to_integer(unsigned(signal_Convert_Addr));
  signal_write_pc <= not input_Stall_PC;
  output_PCPlus4 <= signal_AddPC4_Out;
```

```
    add_PC4: fulladder_32bit
        port map (signal_PC_Out, signal_Four, '0', signal_Cout_PC4, signal_AddPC4_Out);

    mux: mux2to1_32bit
        port map (signal_AddPC4_Out, input_BranchJ_Addr, input_Mux_Sel, signal_PC_WD);

    pc: register_32bit
        port map (input_Clock, input_Reset, signal_PC_WD, signal_write_pc, signal_PC_Out);

    instruc_mem: mem
        port map (input_Clock, signal_convert_to_nat, signal_MemData_Placehold, '0',
output_Instruction);

end structural;
```

Instruction decode:

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

entity instruction_decode is
    port( input_Reset           : in std_logic;
          input_Clock           : in std_logic;
                                        input_Instruction      : in std_logic_vector(31 downto 0);
          input_Forward_RS_Sel1  : in std_logic;
          input_Forward_RS_Sel2  : in std_logic;
          input_Forward_RT_Sel1  : in std_logic;
          input_Forward_RT_Sel2  : in std_logic;
          input_WB_Data          : in std_logic_vector(31 downto 0);
          input_EXMEM_ALUOut     : in std_logic_vector(31 downto 0);
          input_IDEX_MemRead     : in std_logic;
```

```
input_IDEX_WriteReg    : in std_logic_vector(4 downto 0);
input_EXMEM_WriteReg   : in std_logic_vector(4 downto 0);
input_IFID_RS          : in std_logic_vector(4 downto 0);
input_IFID_RT          : in std_logic_vector(4 downto 0);
input_IDEX_RT          : in std_logic_vector(4 downto 0);
input_WriteReg         : in std_logic_vector(4 downto 0);
input_RegWriteEn       : in std_logic;
input_JAL_WB           : in std_logic;
input_PCPlus4          : in std_logic_vector(31 downto 0);
                          input_WriteData        : in std_logic_vector(31 downto 0);
output_Instruction     : out std_logic_vector(31 downto 0);
output_STALL_IFID      : out std_logic;
output_STALL_PC        : out std_logic;
output_PCPlus4         : out std_logic_vector(31 downto 0);
output_JAL             : out std_logic;
output_SHAMT           : out std_logic_vector(31 downto 0);
output_BJ_Address      : out std_logic_vector(31 downto 0);
output_PCSrc           : out std_logic;
output_Immediate       : out std_logic_vector(31 downto 0);
output_WR              : out std_logic_vector(4 downto 0);
output_RegWriteEn      : out std_logic;
output_ALUOP           : out std_logic_vector(3 downto 0);
output_Sel_Mux2        : out std_logic;
output_Mem_To_Reg      : out std_logic;
output_MemWrite        : out std_logic;
output_ALUSrc          : out std_logic;
output_Branch          : out std_logic;
output_JR              : out std_logic;
                                  output_RD1             : out std_logic_vector(31 downto 0);
output_RD2             : out std_logic_vector(31 downto 0);
```

```
        output_FLUSH_IFID      : out std_logic;
        output_FLUSH_IDEX      : out std_logic;
        output_MemRead         : out std_logic );
end instruction_decode;

architecture structural of instruction_decode is
   component branch_jump_logic is
      port( input_BEQ            : in std_logic;
           input_BNE             : in std_logic;
           input_JR              : in std_logic;
           input_Zero_Flag       : in std_logic;
           input_Instruc_25to0   : in std_logic_vector(25 downto 0);
           input_RD1             : in std_logic_vector(31 downto 0);
           input_IMM             : in std_logic_vector(31 downto 0);
                                                  input_PCPlus4           : in std_logic_vector(31 downto
0);
                                                  input_J              : in std_logic;
           input_JAL             : in std_logic;
           output_BJ_Address     : out std_logic_vector(31 downto 0);
           output_PCSrc          : out std_logic;
           output_BranchTaken    : out std_logic;
           output_Branch         : out std_logic );
   end component;

   component control is
      port( input_Instruction    : in std_logic_vector(31 downto 0);
           output_Sel_ALU_A_Mux2 : out std_logic;
           output_RegDst         : out std_logic;
           output_Mem_To_Reg     : out std_logic;
           output_ALUOP                    : out std_logic_vector(3 downto 0);
           output_MemWrite     : out std_logic;
           output_ALUSrc       : out std_logic;
           output_RegWrite     : out std_logic;
           output_BEQ          : out std_logic;
           output_BNE          : out std_logic;
           output_J            : out std_logic;
           output_JAL          : out std_logic;
           output_JR           : out std_logic;
           output_MemRead      : out std_logic );
   end component;

   component register_file is
      port( input_CLK      : in std_logic;
           input_RST       : in std_logic;
                                                  input_RR1       : in std_logic_vector(4 downto 0);
           input_RR2       : in std_logic_vector(4 downto 0);
           input_WR        : in std_logic_vector(4 downto 0);
           input_WD        : in std_logic_vector(31 downto 0);
           input_REGWRITE  : in std_logic;
           output_RD1      : out std_logic_vector(31 downto 0);
           output_RD2      : out std_logic_vector(31 downto 0) );
   end component;

   component mux2to1_5bit is
      port( input_X   : in std_logic_vector(4 downto 0);
           input_Y   : in std_logic_vector(4 downto 0);
           input_SEL : in std_logic;
           output_OUT   : out std_logic_vector(4 downto 0) );
   end component;
```

```
component extend_16to32bit is
   port( input_input : in std_logic_vector(15 downto 0);
        input_sign    : in std_logic;
        output_output  : out std_logic_vector(31 downto 0) );
end component;

component extend_5to32bit is
   port( input_input   : in std_logic_vector(4 downto 0);
        input_sign    : in std_logic;
        output_output  : out std_logic_vector(31 downto 0) );
end component;

component rd1_rd2_zero_detect is
   port( input_RD1       : in std_logic_vector(31 downto 0);
        input_RD2       : in std_Logic_vector(31 downto 0);
        output_Zero_Flag : out std_logic );
end component;



component mux2to1_32bit is
   port( input_X   : in std_logic_vector(31 downto 0);
        input_Y   : in std_logic_vector(31 downto 0);
        input_SEL : in std_logic;
        output_OUT   : out std_logic_vector(31 downto 0) );
end component;

-- Signal declaration ---
signal signal_Sel_ALU_A_Mux2, signal_RegDst, signal_Mem_To_Reg,
signal_MemWrite,
      signal_ALUSrc, signal_RegWrite, signal_BEQ, signal_BNE, signal_J, signal_JAL,
signal_JR, signal_PCSrc,
        signal_Zero, signal_MemRead, signal_BranchTaken, signal_Branch,
signal_FLUSH_IFID,
        signal_FLUSH_IDEX, signal_STALL_IFID, signal_STALL_PC : std_logic;
   signal signal_ALUOP : std_logic_vector(3 downto 0);
   signal signal_Immediate, signal_RD1, signal_RD2, signal_BJ_Addr, signal_SHAMT,
signal_Forward_RS,
        signal_Forward_RT, signal_RS_Data_Final, signal_RT_Data_Final :
std_logic_vector(31 downto 0);
   signal signal_ThirtyOne, signal_WR_Passthru, signal_WR : std_logic_vector(4 downto 0);


begin


   output_FLUSH_IFID <= signal_FLUSH_IFID;
   output_FLUSH_IDEX <= signal_FLUSH_IDEX;
   output_STALL_IFID <= signal_STALL_IFID;
   output_STALL_PC <= signal_STALL_PC;
   output_PCPlus4 <= input_PCPlus4;
   output_JAL <= signal_JAL;
   output_SHAMT <= signal_SHAMT;
   output_BJ_Address <= signal_BJ_Addr;
   output_PCSrc <= signal_PCSrc;
   output_Immediate <= signal_Immediate;
   output_WR <= signal_WR_Passthru;
   output_RegWriteEn <= signal_RegWrite;
   output_RD1 <= signal_RS_Data_Final;
   output_RD2 <= signal_RT_Data_Final;
   output_ALUOP <= signal_ALUOP;
   output_Sel_Mux2 <= signal_Sel_ALU_A_Mux2;
   output_Mem_To_Reg <= signal_Mem_To_Reg;
   output_MemWrite <= signal_MemWrite;
   output_ALUSrc <= signal_ALUSrc;
```

```
    output_Branch <= signal_Branch;

    output_JR <= signal_JR;

    output_MemRead <= signal_MemRead;

    output_Instruction <= input_Instruction;


    signal_ThirtyOne <= (others => '1');


    control_logic: control
        port map (input_Instruction, signal_Sel_ALU_A_Mux2, signal_RegDst,
signal_Mem_To_Reg, signal_ALUOP,
                signal_MemWrite, signal_ALUSrc, signal_RegWrite, signal_BEQ, signal_BNE,
signal_J, signal_JAL, signal_JR, signal_MemRead);


    bj_logic: branch_jump_logic
        port map (signal_BEQ, signal_BNE, signal_J, signal_JAL, signal_JR, signal_Zero,
input_Instruction(25 downto 0),
                signal_RD1, input_PCPlus4, signal_Immediate, signal_BJ_Addr, signal_PCSrc,
signal_BranchTaken, signal_Branch);


    mux_WR_Pre: mux2to1_5bit
        port map (input_Instruction(20 downto 16), input_Instruction(15 downto 11),
signal_RegDst, signal_WR_Passthru);


    mux_WR_Final: mux2to1_5bit
        port map (input_WriteReg, signal_ThirtyOne, input_JAL_WB, signal_WR);



    rf: register_file
        port map (input_clock, input_reset, signal_WR, input_WriteData, input_RegWriteEn,
                input_Instruction(25 downto 21), input_Instruction(20 downto 16), signal_RD1,
signal_RD2);
```

```
    mux_Forward_RS: mux2to1_32bit
        port map(input_WB_Data, input_EXMEM_ALUOut, input_Forward_RS_Sel2, signal_Forward_RS);


    mux_Forward_RT: mux2to1_32bit
        port map(input_WB_Data, input_EXMEM_ALUOut, input_Forward_RT_Sel2, signal_Forward_RT);


    mux_Final_RS: mux2to1_32bit
        port map(signal_RD1, signal_Forward_RS, input_Forward_RS_Sel1, signal_RS_Data_Final);


    mux_Final_RT: mux2to1_32bit
        port map(signal_RD2, signal_Forward_RT, input_Forward_RT_Sel1, signal_RT_Data_Final);


    extend_imm: extend_16to32bit
        port map (input_Instruction(15 downto 0), '1', signal_Immediate);


    extend_shamt: extend_5to32bit
        port map (input_Instruction(10 downto 6), '1', signal_SHAMT);


    rd1_rd2_zero : rd1_rd2_zero_detect
        port map (signal_RS_Data_Final, signal_RT_Data_Final, signal_Zero);


end structural;
```

Execution:

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


entity execution is
```

```
  port( input_Reset            : in std_logic;
        input_Clock            : in std_logic;
        input_WB_Data           : in std_logic_vector(31 downto 0);
        input_Branch            : in std_logic;
        input_EXMEM_RegWriteEn   : in std_logic;
        input_MEMWB_RegWriteEn   : in std_logic;
        input_EXMEM_WriteReg     : in std_logic_vector(4 downto 0);
        input_MEMWB_WriteReg     : in std_logic_vector(4 downto 0);
        input_IFID_RS           : in std_logic_vector(4 downto 0);
        input_IDEX_RS           : in std_logic_vector(4 downto 0);
        input_IFID_RT           : in std_logic_vector(4 downto 0);
        input_IDEX_RT           : in std_logic_vector(4 downto 0);
        input_EXMEM_RT          : in std_logic_vector(4 downto 0);
        input_PCPlus4           : in std_logic_vector(31 downto 0);
        input_JAL              : in std_logic;
                                        input_JR                : in std_logic;
        input_MemRead           : in std_logic;
                                        input_EXMEM_ALUOut        : in std_logic_vector(31 downto
0);
        input_RD1              : in std_logic_vector(31 downto 0);
        input_RD2              : in std_logic_vector(31 downto 0);
        input_IMM              : in std_logic_vector(31 downto 0);
        input_SHAMT             : in std_Logic_vector(31 downto 0);
        input_WR               : in std_logic_vector(4 downto 0);
        input_RegWriteEn        : in std_logic;
        input_ALUOP             : in std_logic_vector(3 downto 0);
        input_Sel_Mux2          : in std_logic;
        input_Mem_To_Reg        : in std_logic;
        input_MemWrite          : in std_logic;
        input_ALUSrc            : in std_logic;
        input_Instruction       : in std_logic_vector(31 downto 0);
        output_Instruction       : out std_logic_vector(31 downto 0);
        output_PCPlus4           : out std_logic_vector(31 downto 0);
        output_JAL               : out std_logic;
        output_ALUOut            : out std_logic_vector(31 downto 0);
        output_RD2               : out std_logic_vector(31 downto 0);
        output_WR                : out std_logic_vector(4 downto 0);
        output_Mem_To_Reg        : out std_logic;
        output_MemWrite          : out std_logic;
        output_RegWriteEn        : out std_logic;
        output_OVF               : out std_logic;
        output_ZF                : out std_logic;
        output_CF                : out std_logic;
        output_Forward_RS_Sel1   : out std_logic;
        output_Forward_RS_Sel2   : out std_logic;
        output_Forward_RT_Sel1   : out std_logic;
        output_Forward_RT_Sel2   : out std_logic );
end execution;

architecture structural of execution is
   component sel_alu_a is
      port( input_ALUSrc   : in std_logic;
           input_RD1      : in std_logic_vector(31 downto 0);
           input_ALUOP    : in std_logic_vector(3 downto 0);
           input_shamt    : in std_logic_vector(31 downto 0);
           input_mux2_sel : in std_logic;
           output_data    : out std_logic_vector(31 downto 0) );
   end component;

   component alu_32bit is
         port( input_A       : in  std_logic_vector(31 downto 0);
            input_B        : in  std_logic_vector(31 downto 0);
            input_ALUOP    : in  std_logic_vector(3  downto 0);
            output_F       : out std_logic_vector(31 downto 0);
            output_CarryOut : out std_logic;
            output_Overflow : out std_logic;
            output_Zero     : out std_logic );
    end component;

    component mux2to1_32bit is
      port( input_X   : in std_logic_vector(31 downto 0);
          input_Y   : in std_logic_vector(31 downto 0);
          input_SEL : in std_logic;
          output_OUT   : out std_logic_vector(31 downto 0) );
    end component;

    component forwarding_logic is
      port( input_Branch             : in std_logic;
          input_JR                 : in std_logic;
          input_EXMEM_RegWriteEn   : in std_logic;
          input_MEMWB_RegWriteEn   : in std_logic;
          input_EXMEM_WriteReg     : in std_logic_vector(4 downto 0);
          input_MEMWB_WriteReg     : in std_logic_vector(4 downto 0);
          input_IFID_RS            : in std_logic_vector(4 downto 0);
          input_IDEX_RS            : in std_logic_vector(4 downto 0);
          input_IFID_RT            : in std_logic_vector(4 downto 0);
          input_IDEX_RT            : in std_logic_vector(4 downto 0);
          input_EXMEM_RT           : in std_logic_vector(4 downto 0);
          output_Forward_ALU_A_Sel1 : out std_logic;
          output_Forward_ALU_A_Sel2 : out std_logic;
          output_Forward_ALU_B_Sel1 : out std_logic;
```

```
            output_Forward_ALU_B_Sel2 : out std_logic;
            output_Forward_RS_Sel1    : out std_logic;
            output_Forward_RS_Sel2    : out std_logic;
            output_Forward_RT_Sel1    : out std_logic;
            output_Forward_RT_Sel2    : out std_logic );
    end component;

    signal signal_alu_out, signal_Forward_A, signal_Forward_B, signal_Normal_A, signal_Normal_B, signal_Final_A,
        signal_Final_B: std_logic_vector(31 downto 0);
    signal signal_cf, signal_ovf, signal_zero, signal_Forward_ALU_A_Sel1, signal_Forward_ALU_A_Sel2,
        signal_Forward_ALU_B_Sel1, signal_Forward_ALU_B_Sel2, signal_Forward_RS_Sel1,
        signal_Forward_RS_Sel2, signal_Forward_RT_Sel1, signal_Forward_RT_Sel2 : std_logic;

begin



    output_JAL <= input_JAL;
    output_ALUOut <= signal_alu_out;
    output_RD2 <= input_RD2;
    output_MemWrite <= input_MemWrite;
    output_RegWriteEn <= input_RegWriteEn;
    output_OVF <= signal_ovf;
    output_ZF <= signal_zero;
                        output_WR <= input_WR;
                        output_Forward_RT_Sel1 <= signal_Forward_RT_Sel1;
                        output_Mem_To_Reg <= input_Mem_To_Reg;
                        output_PCPlus4 <= input_PCPlus4;
    output_Instruction <= input_Instruction;
    output_Forward_RS_Sel1 <= signal_Forward_RS_Sel1;
    output_Forward_RS_Sel2 <= signal_Forward_RS_Sel2;
    output_Forward_RT_Sel2 <= signal_Forward_RT_Sel2;
                    output_CF <= signal_cf;


    select_normal_a: sel_alu_a
       port map(input_ALUSrc, input_RD1, input_ALUOP, input_SHAMT, input_Sel_Mux2, signal_Normal_A);

    mux_a_fwd: mux2to1_32bit
       port map(input_WB_Data, input_EXMEM_ALUOut, signal_Forward_ALU_A_Sel2, signal_Forward_A);

    final_mux_a: mux2to1_32bit
       port map(signal_Normal_A, signal_Forward_A, signal_Forward_ALU_A_Sel1, signal_Final_A);

    select_normal_b: mux2to1_32bit
       port map(input_RD2, input_IMM, input_ALUSrc, signal_Normal_B);

                    mux_b_fwd: mux2to1_32bit
       port map(input_WB_Data, input_EXMEM_ALUOut, signal_Forward_ALU_B_Sel2, signal_Forward_B);


    final_mux_b: mux2to1_32bit
       port map(signal_Normal_B, signal_Forward_B, signal_Forward_ALU_B_Sel1, signal_Final_B);

    alu: alu_32bit
    port map(signal_Final_A, signal_Final_B, input_ALUOp, signal_alu_out, signal_cf,
signal_ovf, signal_zero);


end structural;
```

Memory:

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


entity memory is
   port( input_Reset       : in std_logic;
                              input_Clock       : in std_logic;
         input_ALUOut      : in std_logic_vector(31 downto 0);
         input_WR          : in std_logic_vector(4 downto 0);
         input_Mem_To_Reg  : in std_logic;
         input_RegWriteEn  : in std_logic;
                                        input_RD2         : in std_logic_vector(31 downto 0);
         input_MemWrite    : in std_logic;
                                        input_PCPlus4     : in std_Logic_vector(31 downto 0);
                                        input_JAL         : in std_logic;
         output_PCPlus4     : out std_logic_vector(31 downto 0);
         output_WR          : out std_logic_vector(4 downto 0);
         output_Mem_To_Reg  : out std_logic;
         output_RegWriteEn  : out std_logic;
                                        output_JAL         : out std_logic;
                                        output_ALUOut      : out std_logic_vector(31 downto 0);
         output_MemOut      : out std_logic_vector(31 downto 0) );
end memory;


architecture structural of memory is
   component mem is
```

```
                        generic ( DATA_WIDTH : natural := 32; ADDR_WIDTH : natural := 10
);
                        port ( clk  : in std_logic;
                              addr  : in natural range 0 to 2**ADDR_WIDTH - 1;
                              data  : in std_logic_vector((DATA_WIDTH-1) downto 0);
                              we    : in std_logic := '1';
                              q     : out std_logic_vector((DATA_WIDTH-1) downto 0) );
   end component;

   signal signal_dmem_out : std_logic_vector(31 downto 0);
   signal signal_dmem_addr : natural range 0 to 2**10 - 1;

begin
   output_PCPlus4 <= input_PCPlus4;
    output_WR <= input_WR;
   output_Mem_To_Reg <= input_Mem_To_Reg;
   output_RegWriteEn <= input_RegWriteEn;
   output_MemOut <= signal_dmem_out;
                        output_JAL <= input_JAL;
                        output_ALUOut <= input_ALUOut;



   signal_dmem_addr <= to_integer(unsigned(input_ALUOut(11 downto 2)));


   data_mem: mem
      port map(input_Clock, signal_dmem_addr, input_RD2, input_MemWrite,
signal_dmem_out);


end structural;
```

Write back:

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

entity writeback is
   port( input_Reset        : in std_logic;
         input_Clock        : in std_logic;
         input_JAL          : in std_logic;
         input_WR           : in std_logic_vector(4 downto 0);
         input_Mem_To_Reg   : in std_logic;
         input_MemOut       : in std_logic_vector(31 downto 0);
                                       input_PCPlus4     : in std_logic_vector(31 downto 0);
                                       input_ALUOut      : in std_logic_vector(31 downto 0);
                                       input_RegWriteEn  : in std_logic;
                                       output_WR         : out std_logic_vector(4 downto 0);
         output_RegWriteEn  : out std_logic;
         output_WD          : out std_logic_vector(31 downto 0);
         output_JAL         : out std_logic );
end writeback;

architecture structural of writeback is

   component fulladder_32bit is
      port( input_A    : in std_logic_vector(31 downto 0);
            input_B    : in std_logic_vector(31 downto 0);
            input_Cin  : in std_logic;
            output_Cout : out std_logic;
            output_S    : out std_logic_vector(31 downto 0) );
   end component;
```

```
    component mux2to1_32bit is
        port( input_X   : in std_logic_vector(31 downto 0);
              input_Y   : in std_logic_vector(31 downto 0);
              input_SEL : in std_logic;
              output_OUT   : out std_logic_vector(31 downto 0) );
    end component;

    signal signal_mux_mem_out, signal_mux_wb_out : std_logic_vector(31 downto 0);

begin
    output_JAL <= input_JAL;
                              output_RegWriteEn <= input_RegWriteEn;
    output_WD <= signal_mux_wb_out;
    output_WR <= input_WR;
    mux_mem: mux2to1_32bit
        port map(input_ALUOut, input_MemOut, input_Mem_To_Reg, signal_mux_mem_out);

    mux_wb_final: mux2to1_32bit
        port map(signal_mux_mem_out, input_PCPlus4, input_JAL, signal_mux_wb_out);

end structural;
```

Temporary register between IF and ID stage

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

entity register_IF_ID is
    port(
                              input_Clock       : in std_logic;
        input_Reset       : in std_logic;
          input_Flush       : in std_logic;
          input_Stall       : in std_logic;
          input_Instruction : in std_logic_vector(31 downto 0);
          input_PCPlus4     : in std_logic_vector(31 downto 0);
          output_PCPlus4     : out std_logic_vector(31 downto 0);
                                                  output_Instruction : out std_logic_vector(31 downto 0));
end register_IF_ID;

architecture structural of register_IF_ID is

    component register_Nbit is
        generic ( N : integer := 64 );
        port ( input_CLK  : in std_logic;
               input_RST  : in std_logic;
               input_WD   : in std_logic_vector(N-1 downto 0);
               input_WE   : in std_logic;
               output_Q   : out std_logic_vector(N-1 downto 0) );
    end component;


    signal signal_WD, signal_RD : std_logic_vector(63 downto 0);
    signal signal_stall_reg : std_logic;

begin

    signal_stall_reg <= not input_Stall;

    with input_Flush select signal_WD <=
        (others => '0') when '1',
        (input_Instruction & input_PCPlus4) when '0',
        (others => '0') when others;

    reg: register_Nbit
        port map (input_Clock, input_Reset, signal_WD, signal_stall_reg, signal_RD);

    output_Instruction <= signal_RD(63 downto 32);
    output_PCPlus4 <= signal_RD(31 downto 0);

end structural;
```

Temporary register between ID and EX stage:

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

entity register_ID_EX is
    port( input_Clock       : in std_logic;
          input_Reset       : in std_logic;
          input_WR          : in std_logic_vector(4 downto 0);
                                      input_MemRead     : in std_logic;
          input_Stall       : in std_logic;
          input_MemWrite    : in std_logic;
                                      input_ALUOP       : in std_logic_vector(3 downto 0);
          input_PCPlus4     : in std_logic_vector(31 downto 0);
          input_JAL         : in std_logic;
          input_SHAMT       : in std_logic_vector(31 downto 0);
          input_RD1         : in std_logic_vector(31 downto 0);
          input_RD2         : in std_logic_vector(31 downto 0);
          input_IMM         : in std_logic_vector(31 downto 0);
```

```
        input_Instruction : in std_logic_vector(31 downto 0);
        input_RegWriteEn  : in std_logic;
        input_Sel_Mux2    : in std_logic;
        input_Mem_To_Reg  : in std_logic;
        input_ALUSrc      : in std_logic;
                                        input_Flush       : in std_logic;
        output_MemRead     : out std_logic;
                                        output_WR          : out std_logic_vector(4 downto 0);
        output_ALUOP       : out std_logic_vector(3 downto 0);
                                        output_MemWrite    : out std_logic;
        output_PCPlus4     : out std_logic_vector(31 downto 0);
        output_SHAMT       : out std_logic_vector(31 downto 0);
        output_RD1         : out std_logic_vector(31 downto 0);
        output_RD2         : out std_logic_vector(31 downto 0);
        output_IMM         : out std_logic_vector(31 downto 0);
        output_RegWriteEn  : out std_logic;
        output_Instruction : out std_logic_vector(31 downto 0);
        output_Sel_Mux2    : out std_logic;
        output_Mem_To_Reg  : out std_logic;
             output_JAL         : out std_logic;
        output_ALUSrc      : out std_logic );
end register_ID_EX;

architecture structural of register_ID_EX is

  component register_Nbit is
    generic ( N : integer := 208 );
    port ( input_CLK  : in std_logic;
         input_RST  : in std_logic;
         input_WD   : in std_logic_vector(N-1 downto 0);
```

```
            input_WE   : in std_logic;
            output_Q    : out std_logic_vector(N-1 downto 0) );
    end component;



    signal signal_WD, signal_RD : std_logic_vector(207 downto 0);
    signal signal_stall_reg : std_logic;

begin

    signal_stall_reg <= not input_Stall;

    with input_Flush select signal_WD <=
        (others => '0') when '1',
        (input_Instruction & input_MemRead & input_PCPlus4 & input_JAL & input_SHAMT
& input_RD1 & input_RD2 & input_IMM & input_WR &
            input_RegWriteEn & input_ALUOP & input_Sel_Mux2 & input_Mem_To_Reg &
            input_MemWrite & input_ALUSrc) when '0',
        (others => '0') when others;

    reg: register_Nbit
        port map (input_Clock, input_Reset, signal_WD, signal_stall_reg, signal_RD);

    output_Instruction <= signal_RD(207 downto 176);
    output_MemRead <= signal_RD(175);
    output_PCPlus4 <= signal_RD(174 downto 143);
    output_JAL <= signal_RD(142);
    output_SHAMT <= signal_RD(141 downto 110);
    output_RD1 <= signal_RD(109 downto 78);
    output_RD2 <= signal_RD(77 downto 46);
    output_IMM <= signal_RD(45 downto 14);
```

```
   output_WR  <= signal_RD(13 downto 9);
   output_RegWriteEn <= signal_RD(8);
   output_ALUOP      <= signal_RD(7 downto 4);
                           output_ALUSrc     <= signal_RD(0);
   output_MemWrite   <= signal_RD(1);
   output_Mem_To_Reg <= signal_RD(2);
   output_Sel_Mux2   <= signal_RD(3);
   end structural;
```

Temporary register between EX and MEM stage:

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

entity register_EX_MEM is
   port( input_Clock       : in std_logic;
                              input_Reset       : in std_logic;
        input_MemWrite    : in std_logic;
        input_JAL         : in std_logic;
                                        input_PCPlus4     : in std_logic_vector(31 downto 0);
        input_Flush       : in std_logic;
        input_Stall       : in std_logic;
        input_ALUOut      : in std_logic_vector(31 downto 0);
        input_RD2         : in std_logic_vector(31 downto 0);
        input_WR          : in std_logic_vector(4 downto 0);
        input_Mem_To_Reg  : in std_logic;
        input_RegWriteEn  : in std_logic;
        input_Instruction : in std_logic_vector(31 downto 0);
        output_Instruction : out std_logic_vector(31 downto 0);
        output_PCPlus4     : out std_logic_vector(31 downto 0);
        output_ALUOut      : out std_logic_vector(31 downto 0);
        output_RD2         : out std_logic_vector(31 downto 0);
        output_Mem_To_Reg  : out std_logic;
        output_JAL         : out std_logic;
        output_MemWrite    : out std_logic;
        output_WR          : out std_logic_vector(4 downto 0);
                                output_RegWriteEn  : out std_logic );
end register_EX_MEM;

architecture structural of register_EX_MEM is

   component register_Nbit is
      generic ( N : integer := 137 );
      port ( input_CLK  : in std_logic;
           input_RST  : in std_logic;
                              input_WD   : in std_logic_vector(N-1 downto 0);
                                          input_WE   : in std_logic;
                              output_Q    : out std_logic_vector(N-1 downto 0) );
   end component;

   signal signal_stall_reg : std_logic;
   signal signal_WD, signal_RD : std_logic_vector(136 downto 0);

                    begin

   signal_stall_reg <= not input_Stall;

   with input_Flush select signal_WD <=
      (others => '0') when '1',
      (input_Instruction & input_PCPlus4 & input_JAL & input_ALUOut & input_RD2 &
input_WR & input_Mem_To_Reg & input_MemWrite & input_RegWriteEn) when '0',
        (others => '0') when others;


    reg: register_Nbit
        port map (input_Clock, input_Reset, signal_WD, signal_stall_reg, signal_RD);


    output_Instruction <= signal_RD(136 downto 105);
    output_JAL <= signal_RD(72);
    output_PCPlus4 <= signal_RD(104 downto 73);
    output_RD2 <= signal_RD(39 downto 8);
                            output_RegWriteEn <= signal_RD(0);
                            output_MemWrite <= signal_RD(1);
    output_Mem_To_Reg <= signal_RD(2);
    output_WR <= signal_RD(7 downto 3);
                                output_ALUOut <= signal_RD(71 downto 40);
end structural;
```

Temporary register between MEM and WB:

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


entity register_MEM_WB is
    port( input_Clock       : in std_logic;
                                  input_Reset       : in std_logic;
          input_Mem_To_Reg  : in std_logic;
          input_JAL         : in std_logic;
          input_Flush       : in std_logic;
          input_WR          : in std_logic_vector(4 downto 0);
          input_Stall       : in std_logic;
          input_PCPlus4     : in std_logic_vector(31 downto 0);
          input_MemOut      : in std_logic_vector(31 downto 0);
```

```
        input_ALUOut      : in std_logic_vector(31 downto 0);
        input_RegWriteEn  : in std_logic;
        output_RegWriteEn  : out std_logic;
        output_ALUOut      : out std_logic_vector(31 downto 0);
        output_JAL         : out std_logic;
        output_PCPlus4     : out std_logic_vector(31 downto 0);
        output_MemOut      : out std_logic_vector(31 downto 0);
        output_WR          : out std_logic_vector(4 downto 0);
        output_Mem_To_Reg  : out std_logic);
        end register_MEM_WB;

architecture structural of register_MEM_WB is
    component register_Nbit is
        generic ( N : integer := 104 );
        port ( input_CLK  : in std_logic;
             input_RST  : in std_logic;
             input_WD   : in std_logic_vector(N-1 downto 0);
             input_WE   : in std_logic;
             output_Q    : out std_logic_vector(N-1 downto 0) );
    end component;



    signal signal_WD, signal_RD : std_logic_vector(103 downto 0);
    signal signal_stall_reg : std_logic;

begin

    signal_stall_reg <= not input_Stall;

    with input_Flush select signal_WD <=
        (others => '0') when '1',
        (input_PCPlus4 & input_JAL & input_ALUOut & input_WR & input_Mem_To_Reg &
input_RegWriteEn & input_MemOut) when '0',          -- updates the register as usual
        (others => '0') when others;


    reg: register_Nbit
        port map (input_Clock, input_Reset, signal_WD, signal_stall_reg, signal_RD);
    output_ALUOut <= signal_RD(70 downto 39);
    output_JAL <= signal_RD(71);
                        output_MemOut <= signal_RD(31 downto 0);
                        output_RegWriteEn <= signal_RD(32);
                        output_Mem_To_Reg <= signal_RD(33);
    output_WR <= signal_RD(38 downto 34);
                        output_PCPlus4 <= signal_RD(103 downto 72);
end structural;
```