

Simulating CPU operation using VHDL --FINAL REPORT

Paul Li; Mike Feng,Feifan Li



# Abstract

The purpose of this project is to design, simulate, implement, and verify a simple RISC Computer (Mini SRC) consisting of a simple RISC processor, memory, and I/O. The needed parts are a BusMux, 16 registers, a 32-to-5 encoder, a 2:1 Multiplexer, a MDR, a MAR, a RAM, a select and encoder, a revised register, a Select\_and\_Encoder, a CON\_FF, and a control unit.

Phase 1 of this project consists of the design and Functional Simulation of a part of the Mini SRC Datapath. In Phase 2, the team designed the necessary logic for the ALU and designed Select\_and\_Encoder, a CON\_FF. The Control Unit is designed and tested in Phase 3.

In phase 3, the team also calculated the total time spent and the time spent on each instruction. Because in Phase2, the instructions’ arrangement is different from the guideline on onQ, so the team’s whole process actual spent more time. This problem has not been optimized due to time.

Our team learned a lot and struggled a lot in this lab, but the reward was huge because we finally finished the lab. The team encountered a lot of problems in this lab, including conflicting signals in the encoder due to multiple outputs, register not being able to store the value, signal time too short to connect, etc. With the help of our professor and TA, our team were able to overcome the problems.

Although the team finally finished the lab, we still left some small problems, which can be avoided by using redundant instructions, but there is still room for improvement, and the team will redesign a better signal processing method if there is a chance in the future.

Contents

[Abstract 1](#_Toc69160807)

[Project Specification 3](#_Toc69160808)

[Project Design and Implementation 4](#_Toc69160809)

[Evaluation Result 4](#_Toc69160810)

[Discussion, conclusion, and future work 4](#_Toc69160811)

[Appendices 5](#_Toc69160812)

[Control\_unit.vhd 5](#_Toc69160813)

[Combination.vhd 27](#_Toc69160814)

[ALU.vhd 38](#_Toc69160815)

[busMux.vhd 41](#_Toc69160816)

[encoder.vhd 42](#_Toc69160817)

[MAR.vhd 46](#_Toc69160818)

[MDR.vhd 47](#_Toc69160819)

[Select\_and\_encoder.vhd 48](#_Toc69160820)

[CON\_FF\_Logic.vhd 51](#_Toc69160821)

[Register32.vhd 52](#_Toc69160822)

# Project Specification

The purpose of this project is to design, simulate, implement, and verify a simple RISC Computer (Mini SRC) consisting of a simple RISC processor, memory, and I/O. The following are the needed parts for the overall RISC processor design. A BusMux with fives elect input signals coming from a 32-to-5 encoder. A register that takes the BusMuxOut as the input and BusMuxIn as the output. A 2:1 Multiplexer. A Memory Data Register (MDR) that is different from the other registers in the sense that it has two input sources and two output sources, the inputs to the MDR comes from the memory unit (Mdatain) or from the Bus (BusMuxOut). Data is stored in the MDR using the synchronous Clock signal and the MDRin control signal. The MDR contents can be written into the memory or drive the Bus. Encode and Bus Multiplexer to select which register’s content will be output to the bus.

A “Memory Subsystem” is needed. The “Memory Subsystem” includes the Memory Address Register (MAR), the Memory Data Register (MDR), and the RAM Memory Component. The “Select and Encode” logic accepts the Gra, Grb, Grc, Rin, Rout, and BAout signals as external inputs, these signals will be generated internally by the Control Unit. To support the Load and Store instructions, sometimes the register R0 circuitry will then need to be revised (for example ld r4, $75), the base logic is the BAout signal gates 0’s onto the bus if R0 is selected, and BAout can change the value of R0 to x”00000000”. Gra, Gra and Grc in Select\_and\_Encoder to selected register’s contents if one of the registers R0 –R15 is selected. A “CON FF” is also needed. The “CON FF” logic is used to determine whether the correct condition has been met to cause branching to take place in a Conditional Branch instruction. Input and output ports are needed. As for the input device, the device may have a strobe signal to indicate when the data is available. It is up to you if you want to support such an input device with a strobe signal. Depending on your design, you may (or may not) need the Strobe or the Clock signal.

A Control Unit is also need. The Control Unit is at the heart of the processor. It accepts as inputs those signals that are needed to operate the processor and provides as outputs all the control signals necessary to execute the instructions. The outputs from the Control Unit are the control signals that we use to generate Control Sequences for the instructions of the Mini SRC.

The team had used Quartus II as the software tool for this Mini SRC. The team also choose to VHDL as the language since all team members are more familiar with VHDL then Verilog.

# Project Design and Implementation

Phase 1 of this project consists of the design and Functional Simulation of a part of the Mini SRC Datapath.

In Phase 2, the team designed the necessary logic and simulate the add, sub, mul, div, and, or, shr, shl, ror, rol, neg, and not instructions. The team also designed “Select\_and\_Encoder” logic, “CON\_FF” Logic, “Input/Output” ports, “Memory Subsystem” and load/store instructions, branch and jump instructions, as well as addi, andi, and ori instructions in Phase 2.

The Control Unit is designed and tested in Phase 3. The team used method 1 to design the Control Unit which wrote the VHDL code without worrying about the combinational logic expressions for each control signal. Therefore, the code will come clean, and the instructions will be executed in the most efficient manner. However, it generated more hardware, and made instructions testing a bit harder.

# Evaluation Result

Maximum frequency of operation is 50MHZ, which is 1/20ns.

The time of each state is 40ns(2 clocks). The total time of Phase3 running all the instructions (including not, haut, reset) is 14800000ps = 14800 ns.

In the whole control unit, there are 25 instructions, excluding nop and halt. If the fetch is not included, the average cycle is:

Then in phase 3, each instruction has five fetch cycles, so the average time per instruction is:

Because the team did not follow the guidelines in onQ completely, individual instructions, such as ld, ldi, have one more state than those in onQ, which leads to a slightly larger average instruction time.

# Discussion, conclusion, and future work

The team have encountered many difficulties in this project. With our efforts and the help of from professors and TAs, the team were able to overcome the difficulties and complete the final Lab.

At the beginning of the development, the Testbench according to the Lab Instruction would always have the situation that the busmux was zeroed out or the value that could not have output, i.e., "XXXXXXX" or "UUUUUUUU " situation. After checking the VHDL code repeatedly and observing the output waveform, the team found the problem: for the encoder that controls the delivery of values to the bus, there cannot be more than 2 outputs or be no outputs. Finally, the team completed the lab by restoring the output of the previous step in each phase.

In the subsequent lab, our team also encountered a problem with Register writes. In our experiments, the team successfully read the desired value from the register and prepared to put it into the ALU. However, the team found that the Y Register of the ALU did not write the value of the bus to its own output. After observing the waveform and referring to the TA's advice, the team found that the input and output of a register required one more step than the one described in the instruction. A total of two clock phases are needed to complete the register writing. Therefore, our team temporarily achieved our goal by splitting the T phase.

There are still some problems with our code. The first one is about R0. If R0 is used, then the direct ld value, our Grb is directly grabbing the value of R0, and it will be shifted in a way that should not happen. This problem has something to do with the fact that our team are not clear about the function of BAout values. But this problem can be achieved by adding one more line of instruction. Secondly, the phase of each instruction is not so efficient and there is still a lot of room for optimization. In the future, the team can modify the above problem if there is an opportunity.

# Appendices

## Control\_unit.vhd

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  entity control\_unit is port (                       clock        : in std\_logic;                       reset                   :in std\_LOGIC;                  clear                   : out std\_logic;                       run                     : out std\_logic;                       stop                    : in std\_logic;                       MDR\_read                    : out std\_logic;                       MDR\_write                   : out std\_logic;                  Yin                     : out std\_logic;                  z\_hi\_in                 : out std\_logic;                  z\_lo\_in                 : out std\_logic;                  Hiin                    : out std\_logic;                  Loin                    : out std\_logic;                  Pcin                    : out std\_logic;                  IRin                    : out std\_logic;                  outportin               : out std\_logic;                  c\_sign\_extendedin       : out std\_logic;                  strobe                  : out std\_logic;                  wren\_sig                : out std\_LOGIC;                  Rin                     : out std\_logic;                  Rout                    : out std\_logic;                  Gra                     : out std\_logic;                  Grb                     : out std\_logic;                  Grc                     : out std\_logic;                  BAout                   : out std\_logic;                  Con\_in                  : out std\_logic;                       c\_out                       : out std\_logic;                       incpc                       : out std\_LOGIC;                  hi\_out                  : out std\_logic;                  lo\_out                  : out std\_logic;                  MAR\_in                  : out std\_logic;                  z\_hi\_out                : out std\_logic;                  z\_lo\_out                : out std\_logic;                  PC\_out                  : out std\_logic;                  MDR\_out                 : out std\_logic;                  port\_out                : out std\_logic;                  MDR\_in                  : out std\_logic;                  ALUcontrol\_signals         : out std\_logic\_vector(12 downto 0);                  IRoutput                        :in std\_logic\_vector(31 downto 0);                       con\_out                 :buffer std\_LOGIC;                  inport\_unit             : out std\_logic\_vector(31 downto 0);                  outport\_unit            : out std\_logic\_vector(31 downto 0);      );  end entity control\_unit;  architecture behavior of control\_unit is  --Type state is in the components\_all package  type state is (reset\_state0, reset\_state1, fetch0, fetch1, fetch2, fetch3,fetch4, add3, add4, add5, add6, sub3,  sub4, sub5,sub6, mul3, mul4, mul5, mul6, and3, and4, and5, and6, or3, or4, or5, or6, div3, div4, div5, div6, andi3,  andi4, andi5, ori3, ori4, ori5, branch3, branch4,branch5,branch6, branch7,jr3,jr4, jal3, jal4, in3, mfhi3,   mflo3, shr3, shr4, shr5, shl3, shl4, shl5, ror3, ror4, ror5, rol3, rol4, rol5, neg3, neg4, not3,   not4, ld3, ld4, ld5, ld6, ld7, ldi3, ldi4, ldi5,ldi6, ldr2, ldr3, ldr4, ldr5, ldr6, addi3, addi4, addi5,   out3, st3, st4, st5, st6,st7, str3, str4, str5, str6, nop, halt, shr6,shl6,ror6,rol6, ld8);     signal present\_state:state;  begin  process (clock, reset, stop)  begin      if (reset = '1') then          present\_state <= reset\_state0;      elsif (stop = '1') then          present\_state <= halt;      elsif (clock'event and clock = '1') then          case present\_state is              when reset\_state0 =>                  present\_state <= reset\_state1 after 40ns;              when reset\_state1 =>                  present\_state <= fetch0 after 40ns;              when fetch0 =>                  present\_state <= fetch1 after 40ns;              when fetch1 =>                  present\_state <= fetch2 after 40ns;              when fetch2 =>                  present\_state <= fetch3 after 40ns;              when fetch3 =>                  present\_state <= fetch4 after 40ns;              when fetch4=>                  case IRoutput(31 downto 27) is                      when "00000" =>                          present\_state <= ld3;                      when "00001" =>                          present\_state <= ldi3;                      when "00010" =>                          present\_state <= st3;                      when "00011" =>                          present\_state <= add3;                      when "00100" =>                          present\_state <= sub3;                      when "00101" =>                          present\_state <= shr3;                      when "00110" =>                          present\_state <= shl3;                      when "00111" =>                          present\_state <= ror3;                      when "01000" =>                          present\_state <= rol3;                      when "01001" =>                          present\_state <= and3;                      when "01010" =>                          present\_state <= or3;                      when "01011" =>                          present\_state <= addi3;                      when "01100" =>                          present\_state <= andi3;                      when "01101" =>                          present\_state <= ori3;                      when "01110" =>                          present\_state <= mul3;                      when "01111" =>                          present\_state <= div3;                      when "10000" =>                          present\_state <= neg3;                      when "10001" =>                          present\_state <= not3;                      when "10010" =>                          present\_state <= branch3;                      when "10011" =>                          present\_state <= jr3;                      when "10100" =>                          present\_state <= jal3;                      when "10101" =>                          present\_state <= in3;                      when "10110" =>                          present\_state <= out3;                      when "10111" =>                          present\_state <= mfhi3;                      when "11000" =>                          present\_state <= mflo3;                      when "11001" => -- the nop instruction (do nothing)                          present\_state <= fetch0;                      when "11010" =>                          present\_state <= halt;                      when others=>                  end case;                when add3 =>                  present\_state <= add4 after 40ns;              when add4 =>                  present\_state <= add5 after 40ns;              when add5 =>                  present\_state <= add6 after 40ns;              when add6=>                  present\_state <= fetch0 after 40ns;                when sub3 =>                  present\_state <= sub4 after 40ns;              when sub4 =>                  present\_state <= sub5 after 40ns;              when sub5 =>                  present\_state <= sub6 after 40ns;              when sub6 =>                  present\_state <= fetch0 after 40ns;                when shr3 =>                  present\_state <= shr4 after 40ns;              when shr4 =>                  present\_state <= shr5 after 40ns;              when shr5 =>                  present\_state <= shr6 after 40ns;              when shr6 =>                  present\_state <= fetch0 after 40ns;                when shl3 =>                  present\_state <= shl4 after 40ns;              when shl4 =>                  present\_state <= shl5 after 40ns;              when shl5 =>                  present\_state <= shl6 after 40ns;              when shl6 =>                  present\_state <= fetch0 after 40ns;                when ror3 =>                  present\_state <= ror4 after 40ns;              when ror4 =>                  present\_state <= ror5 after 40ns;              when ror5 =>                  present\_state <= ror6 after 40ns;              when ror6 =>                  present\_state <= fetch0 after 40ns;                when rol3 =>                  present\_state <= rol4 after 40ns;              when rol4 =>                  present\_state <= rol5 after 40ns;              when rol5 =>                  present\_state <= rol6 after 40ns;              when rol6 =>                  present\_state <= fetch0 after 40ns;                when and3 =>                  present\_state <= and4 after 40ns;              when and4 =>                  present\_state <= and5 after 40ns;              when and5 =>                  present\_state <= and6 after 40ns;              when and6 =>                  present\_state <= fetch0 after 40ns;                when ld3 =>                  present\_state <= ld4 after 40ns;              when ld4 =>                  present\_state <= ld5 after 40ns;              when ld5 =>                  present\_state <= ld6 after 40ns;              when ld6 =>                  present\_state <= ld7 after 40ns;              when ld7 =>                  present\_state <= ld8 after 40ns;              when ld8 =>                  present\_state <= fetch0 after 40ns;                when ldi3 =>                  present\_state <= ldi4 after 40ns;              when ldi4 =>                  present\_state <= ldi5 after 40ns;              when ldi5 =>                  present\_state <= ldi6 after 40ns;              when ldi6 =>                  present\_state <= fetch0 after 40ns;                when st3 =>                  present\_state <= st4 after 40ns;              when st4 =>                  present\_state <= st5 after 40ns;              when st5 =>                  present\_state <= st6 after 40ns;              when st6 =>                  present\_state <= st7 after 40ns;              when st7 =>                  present\_state <= fetch0 after 40ns;                when or3 =>                  present\_state <= or4 after 40ns;              when or4 =>                  present\_state <= or5 after 40ns;              when or5 =>                  present\_state <= or6 after 40ns;              when or6 =>                  present\_state <= fetch0 after 40ns;                  when addi3 =>                  present\_state <= addi4 after 40ns;              when addi4 =>                  present\_state <= addi5 after 40ns;              when addi5 =>                  present\_state <= fetch0 after 40ns;                when andi3 =>                  present\_state <= andi4 after 40ns;              when andi4 =>                  present\_state <= andi5 after 40ns;              when andi5 =>                  present\_state <= fetch0 after 40ns;                when ori3 =>                  present\_state <= ori4 after 40ns;              when ori4 =>                  present\_state <= ori5 after 40ns;              when ori5 =>                  present\_state <= fetch0 after 40ns;                when mul3 =>                  present\_state <= mul4 after 40ns;              when mul4 =>                  present\_state <= mul5 after 40ns;              when mul5 =>                  present\_state <= mul6 after 40ns;              when mul6 =>                  present\_state <= fetch0 after 40ns;                when div3 =>                  present\_state <= div4 after 40ns;              when div4 =>                  present\_state <= div5 after 40ns;              when div5 =>                  present\_state <= div6 after 40ns;              when div6 =>                  present\_state <= fetch0 after 40ns;                when neg3 =>                  present\_state <= neg4 after 40ns;              when neg4 =>                  present\_state <= fetch0 after 40ns;                when not3 =>                  present\_state <= not4 after 40ns;              when not4 =>                  present\_state <= fetch0 after 40ns;                when branch3 =>                  present\_state <= branch4 after 40ns;              when branch4 =>                  present\_state <= branch5 after 40ns;              when branch5 =>                  present\_state <= branch6 after 40ns;              when branch6 =>                  present\_state <= branch7 after 40ns;              when branch7=>                  present\_state <= fetch0 after 40ns;                when jr3 =>                  present\_state <= jr4 after 40ns;              when jr4=>                  present\_state <= fetch0 after 40ns;                when jal3 =>                  present\_state <= jal4 after 40ns;              when jal4 =>                  present\_state <= fetch0 after 40ns;                when in3 =>                  present\_state <= fetch0 after 40ns;                when out3 =>                  present\_state <= fetch0 after 40ns;                when mfhi3 =>                  present\_state <= fetch0 after 40ns;                when mflo3 =>                  present\_state <= fetch0 after 40ns;                when halt =>              when others=>          end case;  end if;  end process;      process(present\_state)  begin                incPC         <='0';              run                 <='0';              Yin                 <='0';              z\_hi\_in             <='0';              z\_lo\_in             <='0';              Hiin                <='0';              Loin                <='0';              Pcin                <='0';              IRin                <='0';              outportin           <='0';              c\_sign\_extendedin   <='0';              strobe              <='0';              wren\_sig           <='0';              Rin                 <='0';              Rout                <='0';              Gra                 <='0';              Grb                 <='0';              Grc                 <='0';              BAout               <='0';              Con\_in              <='0';              c\_out               <='0';              hi\_out               <='0';              lo\_out               <='0';              MAR\_in               <='0';              z\_hi\_out             <='0';              z\_lo\_out            <='0';              PC\_out              <='0';              MDR\_out             <='0';              port\_out            <='0';              MDR\_in              <='0';              ALUcontrol\_signals  <="0000000000000";              con\_out             <='0';              inport\_unit         <=x"00000000";              outport\_unit        <=x"00000000";              r14in<='0';      case present\_state is          when reset\_state0 =>              clear <= '0';              run <= '0';          when reset\_state1 =>              clear <= '1';              run <= '1';          when fetch0 =>              clear <= '0';              run <= '0';              PC\_out<= '1';              MAR\_in<= '1'after 10 ns, '0' after 25 ns;          when fetch1 =>              PC\_out<= '0';              ALUcontrol\_signals <= "0111111111111";              z\_lo\_in<='1'after 20 ns, '0' after 35 ns;            when fetch2 =>              ALUcontrol\_signals <= "0000000000000";                  z\_lo\_out <= '1'after 10 ns, '0' after 25 ns;              MDR\_in<='1'after 10 ns, '0' after 25 ns;              MDR\_read <= '1'after 10 ns, '0' after 25 ns;              PCin <= '1'after 10 ns, '0' after 25 ns;          when fetch3 =>              MDR\_out<='1';              IRin <='1';          when fetch4=>            when add3 =>              IRin <='0';              Mdr\_out<='0';              Grb <='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              Rout <='1'after 10 ns, '0' after 25 ns;          when add4 =>              Grc <='1';              Rout <='1'after 10 ns, '0' after 25 ns;              ALUcontrol\_signals <= "1000000000000";          when add5 =>              ALUcontrol\_signals <="0000000000000";              c\_out <='0';              z\_lo\_in <='1';          when add6=>              z\_lo\_in <='0';              Gra <= '1'after 10 ns, '0' after 25 ns;              Rin <= '1'after 10 ns, '0' after 25 ns;              z\_lo\_out <='1'after 10 ns, '0' after 25 ns;              when sub3 =>              IRin <='0';              Mdr\_out<='0';              Grb <='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              Rout <='1'after 10 ns, '0' after 25 ns;          when sub4 =>              Grc <='1';              Rout <='1'after 10 ns, '0' after 25 ns;              ALUcontrol\_signals <= "0100000000000";          when sub5 =>              ALUcontrol\_signals <="0000000000000";              c\_out <='0';              z\_lo\_in <='1';          when sub6=>              z\_lo\_in <='0';              Gra <= '1'after 10 ns, '0' after 25 ns;              Rin <= '1'after 10 ns, '0' after 25 ns;              z\_lo\_out <='1'after 10 ns, '0' after 25 ns;            when mul3 =>              IRin <='0';              Mdr\_out<='0';              Grb <='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              Rout <='1'after 10 ns, '0' after 25 ns;          when mul4 =>              Gra <='1';              Rout <='1'after 10 ns, '0' after 25 ns;              ALUcontrol\_signals <= "0000000010000";          when mul5 =>              ALUcontrol\_signals <="0000000000000";              c\_out <='0';              z\_lo\_in <='1';              z\_hi\_in <='1';              z\_lo\_out <='1';              Loin <= '1';            when mul6=>              Hiin <= '1';              z\_hi\_out <='1';                    when and3 =>              IRin <='0';              Mdr\_out<='0';              Grb <='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              Rout <='1'after 10 ns, '0' after 25 ns;          when and4 =>              Grc <='1';              Rout <='1'after 10 ns, '0' after 25 ns;              ALUcontrol\_signals <= "0000000001000";          when and5 =>              ALUcontrol\_signals <="0000000000000";              c\_out <='0';              z\_lo\_in <='1';          when and6=>              z\_lo\_in <='0';              Gra <= '1'after 10 ns, '0' after 25 ns;              Rin <= '1'after 10 ns, '0' after 25 ns;              z\_lo\_out <='1'after 10 ns, '0' after 25 ns;            when or3 =>              IRin <='0';              Mdr\_out<='0';              Grb <='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              Rout <='1'after 10 ns, '0' after 25 ns;          when or4 =>              Grc <='1';              Rout <='1'after 10 ns, '0' after 25 ns;              ALUcontrol\_signals <= "0000000000100";          when or5 =>              ALUcontrol\_signals <="0000000000000";              c\_out <='0';              z\_lo\_in <='1';          when or6=>              z\_lo\_in <='0';              Gra <= '1'after 10 ns, '0' after 25 ns;              Rin <= '1'after 10 ns, '0' after 25 ns;              z\_lo\_out <='1'after 10 ns, '0' after 25 ns;                  when div3 =>              IRin <='0';              Mdr\_out<='0';              Gra <='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              Rout <='1'after 10 ns, '0' after 25 ns;          when div4 =>              Grb <='1';              Rout <='1'after 10 ns, '0' after 25 ns;              ALUcontrol\_signals <= "0000000000001";          when div5 =>              ALUcontrol\_signals <="0000000000000";              c\_out <='0';              z\_lo\_in <='1';              z\_hi\_in <='1';              z\_lo\_out <='1';              Loin <= '1';            when div6=>              Hiin <= '1';              z\_hi\_out <='1';            when shr3 =>              IRin <='0';              Mdr\_out<='0';              Grb <='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              Rout <='1'after 10 ns, '0' after 25 ns;          when shr4 =>              Grc <='1';              Rout <='1'after 10 ns, '0' after 25 ns;              ALUcontrol\_signals <= "0010000000000";          when shr5 =>              ALUcontrol\_signals <="0000000000000";              c\_out <='0';              z\_lo\_in <='1';          when shr6=>              z\_lo\_in <='0';              Gra <= '1'after 10 ns, '0' after 25 ns;              Rin <= '1'after 10 ns, '0' after 25 ns;              z\_lo\_out <='1'after 10 ns, '0' after 25 ns;            when shl3 =>              IRin <='0';              Mdr\_out<='0';              Grb <='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              Rout <='1'after 10 ns, '0' after 25 ns;          when shl4 =>              Grc <='1';              Rout <='1'after 10 ns, '0' after 25 ns;              ALUcontrol\_signals <= "0001000000000";          when shl5 =>              ALUcontrol\_signals <="0000000000000";              c\_out <='0';              z\_lo\_in <='1';          when shl6=>              z\_lo\_in <='0';              Gra <= '1'after 10 ns, '0' after 25 ns;              Rin <= '1'after 10 ns, '0' after 25 ns;              z\_lo\_out <='1'after 10 ns, '0' after 25 ns;            when ror3 =>              IRin <='0';              Mdr\_out<='0';              Grb <='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              Rout <='1'after 10 ns, '0' after 25 ns;          when ror4 =>              Grc <='1';              Rout <='1'after 10 ns, '0' after 25 ns;              ALUcontrol\_signals <= "0000100000000";          when ror5 =>              ALUcontrol\_signals <="0000000000000";              c\_out <='0';              z\_lo\_in <='1';          when ror6=>              z\_lo\_in <='0';              Gra <= '1'after 10 ns, '0' after 25 ns;              Rin <= '1'after 10 ns, '0' after 25 ns;              z\_lo\_out <='1'after 10 ns, '0' after 25 ns;            when rol3 =>              IRin <='0';              Mdr\_out<='0';              Grb <='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              Rout <='1'after 10 ns, '0' after 25 ns;          when rol4 =>              Grc <='1';              Rout <='1'after 10 ns, '0' after 25 ns;              ALUcontrol\_signals <= "0000010000000";          when rol5 =>              ALUcontrol\_signals <="0000000000000";              c\_out <='0';              z\_lo\_in <='1';          when rol6=>              z\_lo\_in <='0';              Gra <= '1'after 10 ns, '0' after 25 ns;              Rin <= '1'after 10 ns, '0' after 25 ns;              z\_lo\_out <='1'after 10 ns, '0' after 25 ns;            when neg3 =>              IRin <='0';              Mdr\_out<='0';              Rout <= '1';              Grb <= '1';              ALUcontrol\_signals <= "0000001000000";              z\_lo\_in <= '1';          when neg4 =>              Rout <= '0';              Grb <= '0';              ALUcontrol\_signals <= "0000000000000";              z\_lo\_in <= '0';                Z\_lo\_out <= '1';              Gra <= '1';              Rin <= '1';            when not3 =>              IRin <='0';              Mdr\_out<='0';              Rout <= '1';              Grb <= '1';              ALUcontrol\_signals <= "0000000000010";              z\_lo\_in <= '1';          when not4 =>              Rout <= '0';              Grb <= '0';              ALUcontrol\_signals <= "0000000000000";              z\_lo\_in <= '0';              Z\_lo\_out <= '1';              Gra <= '1';              Rin <= '1';            when andi3 =>              IRin <='0';              Mdr\_out<='0';              Grb <='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              Rout <='1'after 10 ns, '0' after 25 ns;          when andi4 =>              C\_out <= '1';              ALUcontrol\_signals <= "0000000001000";              z\_lo\_in <='1'after 35 ns, '0' after 40 ns;          when andi5 =>              ALUcontrol\_signals <="0000000000000";              Gra <='1'after 10 ns, '0' after 25 ns;              Rin <='1'after 10 ns, '0' after 25 ns;              c\_out <='0';              z\_lo\_in <='1'after 0 ns, '0' after 10 ns;              z\_lo\_out <='1'after 10 ns, '0' after 25 ns;            when ori3 =>              IRin <='0';              Mdr\_out<='0';              Grb <='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              Rout <='1'after 10 ns, '0' after 25 ns;          when ori4 =>              C\_out <= '1';              ALUcontrol\_signals <= "0000000000100";              z\_lo\_in <='1'after 35 ns, '0' after 40 ns;          when ori5 =>              ALUcontrol\_signals <="0000000000000";              Gra <='1'after 10 ns, '0' after 25 ns;              Rin <='1'after 10 ns, '0' after 25 ns;              c\_out <='0';              z\_lo\_in <='1'after 0 ns, '0' after 10 ns;              z\_lo\_out <='1'after 10 ns, '0' after 25 ns;            when addi3 =>              IRin <='0';              Mdr\_out<='0';              Grb <='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              Rout <='1'after 10 ns, '0' after 25 ns;          when addi4 =>              C\_out <= '1';              ALUcontrol\_signals <= "1000000000000";              z\_lo\_in <='1'after 35 ns, '0' after 40 ns;          when addi5 =>              ALUcontrol\_signals <="0000000000000";              Gra <='1'after 10 ns, '0' after 25 ns;              Rin <='1'after 10 ns, '0' after 25 ns;              c\_out <='0';              z\_lo\_in <='1'after 0 ns, '0' after 10 ns;              z\_lo\_out <='1'after 10 ns, '0' after 25 ns;              WHEN branch3 =>              MDR\_out<='0';              IRin <='0';                Gra <='1';              Rout<='1';              Con\_in <='1';          WHEN branch4 =>              Gra <='0';              Rout <='0';              Con\_in <='0';              Pc\_out <='1';              Yin<='1';          WHEN branch5 =>              Pc\_out <='0';              Yin<='0';              c\_out <='1';                ALUcontrol\_signals <= "1000000000000";  --add            when branch6 =>              ALUcontrol\_signals <="0000000000000";              c\_out <='0';              z\_lo\_in <='1';          when branch7 =>              z\_lo\_in <='0';              z\_lo\_out <='1';              if(con\_out = '1') then                  PCin <= '1'after 10 ns, '0' after 25 ns;              else                  PCin <='0';              end if;    ------------------          WHEN jr3=>              Gra <='1';          WHEN jr4=>              Gra <='0';              IRin <='0';              Mdr\_out<='0';                Rout<='1';              PCin<='1'after 10 ns, '0' after 25 ns;      when jal3 =>              Rin <= '1';              Grb <= '1';              PC\_out <= '1';              r14in <='1';          when jal4 =>              Rout <= '1';              Gra <= '1';              PCin <= '1';            WHEN in3 =>              IRin <='0';              Mdr\_out<='0';              Gra <='1'after 10 ns, '0' after 25 ns;              Rin <='1'after 10 ns, '0' after 25 ns;              port\_out<='1'after 10 ns, '0' after 25 ns;            WHEN out3 =>              IRin <='0';               Mdr\_out<='0';              Gra <='1'after 10 ns, '0' after 25 ns;              Rout <='1'after 10 ns, '0' after 25 ns;              outportin<='1'after 10 ns, '0' after 25 ns;            when mfhi3 =>              Gra <='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              hi\_out <=  '1';              Rin <='1'after 10 ns, '0' after 25 ns;            when mflo3 =>              IRin <='0';              Mdr\_out<='0';              Gra <='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              lo\_out <=  '1';              Rin <='1'after 10 ns, '0' after 25 ns;            when ld3 =>              IRin <='0';              Mdr\_out<='0';              MAR\_in<= '1';              MDR\_in<='1'after 19 ns, '0' after 40 ns;              MDR\_read <= '1'after 19 ns, '0' after 40 ns;          when ld4 =>              Grb <='1'after 10 ns, '0' after 25 ns;              Rout<='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;          when ld5 =>              C\_out<='1';              ALUcontrol\_signals <= "1000000000000";              z\_lo\_in <='1'after 35 ns, '0' after 40 ns;          when ld6 =>              z\_lo\_in <='1'after 0 ns, '0' after 5 ns;              ALUcontrol\_signals <="0000000000000";              c\_out <='0';                z\_lo\_out <='1';              MAR\_in <='1' after 19 ns, '0' after 40 ns;          when ld7 =>              z\_lo\_out <='1';              MDR\_read <='1'after 10 ns, '0' after 25 ns;              MDR\_in <='1'after 10 ns, '0' after 25 ns;          when ld8 =>              z\_lo\_out <='0';              MDR\_in <='0';                MDR\_out <='1'after 10 ns, '0' after 25 ns;              Gra <='1'after 10 ns, '0' after 25 ns;              Rin <='1'after 19 ns, '0' after 40 ns;            when ldi3 =>              IRin <='0';              Mdr\_out<='0';              MAR\_in<= '1';              MDR\_in<='1'after 19 ns, '0' after 40 ns;              MDR\_read <= '1'after 19 ns, '0' after 40 ns;          when ldi4 =>              Grb <='1'after 10 ns, '0' after 25 ns;              Rout<='1'after 10 ns, '0' after 25 ns;              Yin<='1'after 10 ns, '0' after 25 ns;              --BAout<='1'after 10 ns, '0' after 25 ns;          when ldi5 =>              C\_out<='1' ;              ALUcontrol\_signals <= "1000000000000";              z\_lo\_in <='1'after 20 ns, '0' after 40 ns;          when ldi6=>              z\_lo\_in <='1'after 0 ns, '0' after 4 ns;              ALUcontrol\_signals <="0000000000000";              c\_out <='0';              Gra <= '1'after 10 ns, '0' after 25 ns;              Rin <='1' after 10 ns, '0' after 25 ns;              z\_lo\_out <='1'after 10 ns, '0' after 25 ns;              when st3 =>         IRin <='0';              Mdr\_out<='0';           Grb <='1'after 10 ns, '0' after 25 ns;              Rout <='1';            when st4 =>              Rout <='0';              C\_out<='1';              Yin<='1', '0' after 15 ns;          when st5 =>              Yin<='0';              ALUcontrol\_signals <= "1000000000000";              z\_lo\_in <='1'after 10 ns, '0' after 25 ns;          when st6 =>                  ALUcontrol\_signals <="0000000000000";              c\_out <='0';              MAR\_in <='1';              z\_lo\_out <='1';              when st7 =>                  MAR\_in<='0';              z\_lo\_out <='0';                Gra <='1';              Rout <='1';              wren\_sig<='1';              MDR\_read <='0';              MDR\_in <='1'    ;              -- Gra\_tb <='1'after 10 ns, '0' after 25 ns;              -- Rout\_tb<='1';            when nop =>          when halt =>              run <= '0';          when others=>      end case;  end process;  end behavior; |

## Combination.vhd

|  |
| --- |
| library IEEE;  use ieee.numeric\_std.all;  use ieee.std\_logic\_unsigned.all;  use IEEE.std\_logic\_1164.all;  entity combination is     Port(                                           clock        : in std\_logic;                                           reset                   :in std\_LOGIC;                                           run                     : out std\_logic;                                           stop                    : in std\_logic          );  end combination;  architecture structural of combination is  Component register32 is          generic(          VAL : std\_logic\_vector(31 downto 0) := x"00000000"          );          port(   clr,clk,wr: in std\_logic;              inputD: IN std\_logic\_vector(31 downto 0); --BusMuxOut--              outputQ: out std\_logic\_vector(31 downto 0) --BusMuxIn-Rn--          );  end component register32;  ----------------------------------------------------------------------  Component register14 is          generic(          VAL : std\_logic\_vector(31 downto 0) := x"00000000"          );          port(   clr,clk,wr: in std\_logic;              inputD: IN std\_logic\_vector(31 downto 0); --BusMuxOut--              outputQ: out std\_logic\_vector(31 downto 0) --BusMuxIn-Rn--          );  end component register14;  component control\_unit is  port(                                          clock         : in std\_logic;                                          reset                   :in std\_LOGIC;                  clear                   : out std\_logic;                                           run                     : out std\_logic;                                           stop                    : in std\_logic;                                           MDR\_read                                               : out std\_logic;                                           MDR\_write                                              : out std\_logic;                  Yin                     : out std\_logic;                  z\_hi\_in                 : out std\_logic;                  z\_lo\_in                 : out std\_logic;                  Hiin                    : out std\_logic;                  Loin                    : out std\_logic;                  Pcin                    : out std\_logic;                  IRin                    : out std\_logic;                  outportin               : out std\_logic;                  c\_sign\_extendedin       : out std\_logic;                  strobe                  : out std\_logic;                  wren\_sig                : out std\_LOGIC;                  Rin                     : out std\_logic;                  Rout                    : out std\_logic;                  Gra                     : out std\_logic;                  Grb                     : out std\_logic;                  Grc                     : out std\_logic;                  BAout                   : out std\_logic;                  Con\_in                  : out std\_logic;                                           c\_out                                           : out std\_logic;                                           incpc                                           : out std\_LOGIC;                  hi\_out                  : out std\_logic;                  lo\_out                  : out std\_logic;                  MAR\_in                  : out std\_logic;                  z\_hi\_out                : out std\_logic;                  z\_lo\_out                : out std\_logic;                  PC\_out                  : out std\_logic;                  MDR\_out                 : out std\_logic;                  port\_out                : out std\_logic;                  MDR\_in                  : out std\_logic;                  ALUcontrol\_signals         : out std\_logic\_vector(12 downto 0);                  IRoutput                 : in std\_LOGIC\_VECTOR(31 downto 0);                                           con\_out                 :out std\_LOGIC;                  inport\_unit             : out std\_logic\_vector(31 downto 0);                  outport\_unit            : out std\_logic\_vector(31 downto 0);                                           r14in                   : out std\_logic  );  end component control\_unit;  -------------------------------------------------------------------------------------------------  Component PC is          generic(          VAL : std\_logic\_vector(31 downto 0) := x"00000000"          );          port(   clr,clk,wr: in std\_logic;              inputD: IN std\_logic\_vector(31 downto 0); --BusMuxOut--              outputQ: buffer std\_logic\_vector(31 downto 0); --BusMuxIn-Rn--                                  IncPC : in std\_logic          );  end component PC;  ----------------------------------------------------------------------------------------------------  component busMux is  generic(          VAL : std\_logic\_vector(31 downto 0) := x"00000000"          );  PORT(          r00\_in, r01\_in, r02\_in, r03\_in,          r04\_in, r05\_in, r06\_in, r07\_in,          r08\_in, r09\_in, r10\_in, r11\_in,          r12\_in, r13\_in, r14\_in, r15\_in,          BMIn\_MDR, inhi, inlo ,BMI\_z\_hi,          BMI\_z\_lo, BMI\_inpc, BMI\_portin,          BMIcSignExtended                : IN std\_logic\_vector (31 downto 0);          s\_in                            : IN std\_logic\_vector(4 downto 0);          BusMuxOut                       : out std\_logic\_vector(31 downto 0)  );  end component busMux;  ---------------------------------------------------------------------------------------------------------  component encoder is  PORT(          r00\_out, r01\_out, r02\_out, r03\_out,          r04\_out, r05\_out, r06\_out, r07\_out,          r08\_out, r09\_out, r10\_out, r11\_out,          r12\_out, r13\_out, r14\_out, r15\_out,          hi\_out, lo\_out, z\_hi\_out, z\_lo\_out,          PC\_out, MDR\_out, port\_out, c\_out    : IN std\_logic;          s\_out                               : OUT std\_logic\_vector(4 downto 0)  );  end component encoder;  -------------------------------------------------------------------------------------------------------  component ram          PORT          (                  address         : IN STD\_LOGIC\_VECTOR (8 DOWNTO 0);                  clock           : IN STD\_LOGIC;                  data            : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0);                  wren            : IN STD\_LOGIC ;                  q               : OUT STD\_LOGIC\_VECTOR (31 DOWNTO 0)          );  end component;  component MAR is          generic(          VAL : std\_logic\_vector(31 downto 0) := x"00000000";                    ADDRESS : std\_logic\_vector(8 downto 0):= b"000000000"          );          port(clr, clk,wr: in std\_logic;                          inputD: in std\_logic\_vector(31 downto 0);                          outputQ : buffer std\_logic\_vector(8 downto 0)                          );  end component MAR;  component MDR is  generic(          VAL : std\_logic\_vector(31 downto 0) := x"00000000"          );  port(   busMuxOut, MDataIn          : IN std\_logic\_vector(31 downto 0);          clr,clk,MDRread, mdr\_in     : IN std\_logic;          outputQ                     : Out std\_logic\_vector(31 downto 0)  );  end component MDR;  component ALU is  port(   A,B                 : in std\_logic\_vector(31 downto 0);          ALUcontrol\_signals     : in std\_logic\_vector(12 downto 0);          C\_hi                : out std\_logic\_vector(31 downto 0); --          c\_lo                : out std\_logic\_vector(31 downto 0);          clk                 : in std\_logic);  end component ALU;  component select\_and\_encoder is      generic(          VAL : std\_logic\_vector(15 downto 0):= x"0000"      );  port (          Rin     : in std\_logic;          Rout    : in std\_logic;          BAout   : in std\_logic;          Gra, Grb, Grc : in std\_logic;          R0to15\_out     : out std\_logic\_vector(15 downto 0);          R0to15\_in     : out std\_logic\_vector(15 downto 0);          IR\_in : in std\_logic\_vector(31 downto 0);          c\_sign\_extended : out std\_logic\_vector(31 downto 0)  );  end component select\_and\_encoder;  component revised\_register is   port(          BAout : in std\_logic;          registeroutQ: in std\_logic\_vector(31 downto 0);          BusMuxIn\_R: out std\_logic\_vector(31 downto 0)          );  end component revised\_register;  component CON\_FF\_Logic is          port(              IR\_in\_Conff: in std\_logic\_vector(31 downto 0);              BusMuxout : in std\_logic\_vector(31 downto 0);              Con\_in    : in std\_logic;              outputQ   : out std\_logic          );  end component CON\_FF\_Logic;  signal  s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,          BMIn\_MDR, inhi, inlo ,BMI\_z\_hi, BMI\_z\_lo, BMI\_inpc, BMI\_portin,          BMIcSignExtended,A,BusMuxOut:std\_logic\_vector(31 downto 0) ;--signal C:std\_logic\_vector(6 downto 0);  signal  BMIN\_R0, BMIN\_R1, BMIN\_R2, BMIN\_R3, BMIN\_R4, BMIN\_R5, BMIN\_R6,          BMIN\_R7, BMIN\_R8, BMIN\_R9, BMIN\_R10, BMIN\_R11, BMIN\_R12, BMIN\_R13, BMIN\_R14, BMIN\_R15 :std\_logic\_vector(31 downto 0) ;  signal  C\_hi, C\_lo: std\_logic\_vector(31 downto 0);  signal  sout:std\_logic\_vector(4 downto 0);  signal address\_sig: std\_LOGIC\_VECTOR(8 downto 0);  --phase 2  signal R0to15\_out, R0to15\_in :std\_logic\_vector(15 downto 0);  signal IRoutput, ram\_out, c\_sign\_extended:std\_logic\_vector(31 downto 0);  signal MData\_in :std\_logic\_vector(31 downto 0);  --phase 3  signal Yin, z\_hi\_in ,z\_lo\_in,Hiin,Loin,Pcin,IRin,outportin,c\_sign\_extendedin,strobe,wren\_sig,Rin,Rout,Gra,Grb,Grc,BAout,Con\_in,c\_out,incpc,hi\_out,lo\_out,MAR\_in,z\_hi\_out,z\_lo\_out,PC\_out                          ,MDR\_out,port\_out,MDR\_in, con\_out ,MDR\_read,MDR\_write,clear,r14in: std\_LOGIC;  signal ALUcontrol\_signals: std\_LOGIC\_VECTOR(12 downto 0);  signal inport\_unit,outport\_unit  :  std\_logic\_vector(31 downto 0);  begin    regY: register32  port map(clr=>clear,clk=>clock,wr=>Yin,outputQ=>A,  inputD=>busMuxOut); --BUXMUX related register  reg0: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>R0to15\_in(0),outputQ=>s0,inputD=>busMuxOut);  reg1: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>R0to15\_in(1),outputQ=>s1,inputD=>busMuxOut);  reg2: register32 generic map (VAL => (x"00000000")) port map(clr=>clear,clk=>clock,wr=>R0to15\_in(2),outputQ=>s2,inputD=>busMuxOut);  reg3: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>R0to15\_in(3),outputQ=>s3,inputD=>busMuxOut);  reg4: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>R0to15\_in(4),outputQ=>s4,inputD=>busMuxOut);  reg5: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>R0to15\_in(5),outputQ=>s5,inputD=>busMuxOut);  reg6: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>R0to15\_in(6),outputQ=>s6,inputD=>busMuxOut);  reg7: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>R0to15\_in(7),outputQ=>s7,inputD=>busMuxOut);  reg8: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>R0to15\_in(8),outputQ=>s8,inputD=>busMuxOut);  reg9: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>R0to15\_in(9),outputQ=>s9,inputD=>busMuxOut);  reg10: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>R0to15\_in(10),outputQ=>s10,inputD=>busMuxOut);  reg11: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>R0to15\_in(11),outputQ=>s11,inputD=>busMuxOut);  reg12: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>R0to15\_in(12),outputQ=>s12,inputD=>busMuxOut);  reg13: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>R0to15\_in(13),outputQ=>s13,inputD=>busMuxOut);  reg14: register14 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>r14in,outputQ=>s14,inputD=>busMuxOut);  reg15: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>R0to15\_in(15),outputQ=>s15,inputD=>busMuxOut);  PC1: PC generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>Pcin,outputQ=>BMI\_inpc,inputD=>busMuxOut,incPC=>incPC);  IR1: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>IRin,outputQ=>IRoutput,inputD=>busMuxOut);  regHi: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>Hiin,outputQ=>inhi,inputD=>busMuxOut);  regLo: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>Loin,outputQ=>inlo,inputD=>busMuxOut);  regzhi: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>z\_hi\_in,outputQ=>BMI\_z\_hi,inputD=>c\_Hi);  regzlo: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>z\_lo\_in,outputQ=>BMI\_z\_lo,inputD=>c\_Lo);  inport: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>strobe,outputQ=>BMI\_portin,inputD=>inport\_unit);  outport: register32 generic map (VAL => x"00000000") port map(clr=>clear,clk=>clock,wr=>outportin,outputQ=>outport\_unit,inputD=>busMuxOut);    --Revised\_register  revised\_register0: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R0,registeroutQ=>s0);  revised\_register1: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R1,registeroutQ=>s1);  revised\_register2: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R2,registeroutQ=>s2);  revised\_register3: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R3,registeroutQ=>s3);  revised\_register4: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R4,registeroutQ=>s4);  revised\_register5: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R5,registeroutQ=>s5);  revised\_register6: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R6,registeroutQ=>s6);  revised\_register7: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R7,registeroutQ=>s7);  revised\_register8: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R8,registeroutQ=>s8);  revised\_register9: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R9,registeroutQ=>s9);  revised\_register10: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R10,registeroutQ=>s10);  revised\_register11: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R11,registeroutQ=>S11);  revised\_register12: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R12,registeroutQ=>s12);  revised\_register13: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R13,registeroutQ=>s13);  revised\_register14: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R14,registeroutQ=>s14);  revised\_registerr15: revised\_register port map(BAout =>BAout,BusMuxIn\_R=>BMIN\_R15,registeroutQ=>s15);  --CON\_FF\_Logic  CON\_FF\_Logic1: CON\_FF\_Logic port map(   IR\_in\_Conff=>IRoutput ,                                          BusMuxout=>BusMuxout ,                                          Con\_in=> Con\_in,                                          outputQ=> con\_out                                          );    --MDR  MDR1: MDR generic map (VAL => x"00000000") port map( busMuxOut => busMuxOut,                      MDataIn => MData\_in ,                      clr => clear, clk => clock ,                      mdr\_in => MDR\_in,                      MDRread => MDR\_read,                      outputQ=>BMIn\_MDR);--BusMuxin\_inPort: register32 port map(clr=>clear,clk=>clock,wr=>Pcin,outputQ=>busMuxOut,inputD=>busMuxOut);    --encoder  encoder1: encoder port map( r00\_out=>R0to15\_out(0),                              r01\_out=>R0to15\_out(1),                              r02\_out=>R0to15\_out(2),                              r03\_out=>R0to15\_out(3),                              r04\_out=>R0to15\_out(4),                              r05\_out=>R0to15\_out(5),                              r06\_out=>R0to15\_out(6),                              r07\_out=>R0to15\_out(7),                              r08\_out=>R0to15\_out(8),                              r09\_out=>R0to15\_out(9),                              r10\_out=>R0to15\_out(10),                              r11\_out=>R0to15\_out(11),                              r12\_out=>R0to15\_out(12),                              r13\_out=>R0to15\_out(13),                              r14\_out=>R0to15\_out(14),                              r15\_out=>R0to15\_out(15),                              hi\_out=>hi\_out,                              lo\_out=>lo\_out,                              z\_hi\_out=>z\_hi\_out,                              z\_lo\_out=>z\_lo\_out,                              PC\_out=>PC\_out,                              MDR\_out=>MDR\_out,                              port\_out => port\_out,                              c\_out => c\_out,                              s\_out => sout);    --32:1 Multiplexer BusMux  busMux1: busMux generic map (VAL => x"00000000") port map(   r00\_in => BMIN\_R0, r01\_in => BMIN\_R1, r02\_in => BMIN\_R2,                              r03\_in => BMIN\_R3,r04\_in => BMIN\_R4, r05\_in => BMIN\_R5,                              r06\_in => BMIN\_R6,r07\_in => BMIN\_R7,r08\_in => BMIN\_R8,                              r09\_in => BMIN\_R9, r10\_in => BMIN\_R10,r11\_in => BMIN\_R11,                              r12\_in => BMIN\_R12, r13\_in => BMIN\_R13, r14\_in => BMIN\_R14,                              r15\_in => BMIN\_R15,                              BMIn\_MDR=>BMIn\_MDR, inhi=>inhi,                              inlo=>inlo ,BMI\_z\_hi=>BMI\_z\_hi,                              BMI\_z\_lo=>BMI\_z\_lo, BMI\_inpc=>BMI\_inpc,                              BMI\_portin=>BMI\_portin,BMIcSignExtended=>c\_sign\_extended,                              s\_in => sout, BusMuxOut => BusMuxOut);  --ALU  ALU1: ALU port map(     A => A ,                          B => BusMuxout,                          ALUcontrol\_signals => ALUcontrol\_signals,                          C\_hi=> C\_hi , c\_lo=>c\_lo ,                          clk =>  clock                  );  -- RAM  ram\_inst : ram PORT MAP (                  address  => address\_sig,                  clock    => clock,                  data     => BMIn\_MDR,                  wren     => wren\_sig,                  q        => MData\_in          );  --select\_and\_encoder  select\_and\_encoder1 : select\_and\_encoder generic map (VAL => x"0000") port map(                  Rin             =>      Rin,                  Rout            =>      Rout,                  BAout           =>      BAout,                  R0to15\_out      =>      R0to15\_out,                  R0to15\_in       =>      R0to15\_in,                  c\_sign\_extended =>    c\_sign\_extended,                  Gra             =>      Gra,                  Grb             =>      Grb,                  Grc             =>      Grc,                                           IR\_in           =>      IRoutput          );  --MAR  MAR1:MAR generic map (VAL => x"00000000", ADDRESS=>b"000000000") port map(clr => clear,clk=>clock,                  wr => MAR\_in,                  inputD => busMuxout,                  outputQ => address\_sig                          );    control\_unit1: control\_unit port map(                  clock                    =>  clock,                  reset                   => reset,         clear                  =>  clear,                   run                    =>  run,                   stop                   =>  stop,                   MDR\_read               => MDR\_read,                   MDR\_write              => MDR\_write,                  Yin                     =>  Yin,                  z\_hi\_in                 =>  z\_hi\_in,                  z\_lo\_in                 =>  z\_lo\_in,                  Hiin                    =>  Hiin,                  Loin                    =>  Loin,                  pcin                    =>  pcin,                  IRin                    =>  IRin,                  outportin               =>  outportin,                  c\_sign\_extendedin       =>  c\_sign\_extendedin,                  strobe                  =>  strobe,                  wren\_sig                =>  wren\_sig,                  Rin                     =>  Rin,                  Rout                    =>  Rout,                  Gra                     =>  Gra,                  Grb                     =>  Grb,                  Grc                     =>  Grc,                  BAout                   =>  BAout,                  Con\_in                  =>  Con\_in,                                           c\_out                                           =>  c\_out,                                           incpc                                           =>  incpc,                  hi\_out                  =>  hi\_out,                  lo\_out                  =>  lo\_out,                  MAR\_in                  =>  MAR\_in,                  z\_hi\_out                =>  z\_hi\_out,                  z\_lo\_out                =>  z\_lo\_out,                  PC\_out                  =>  PC\_out,                  MDR\_out                 =>  MDR\_out,                  port\_out                =>  port\_out,                  MDR\_in                  =>  MDR\_in,                  ALUcontrol\_signals      =>  ALUcontrol\_signals,                                           IRoutput                => IRoutput,                  con\_out                 => con\_out,                  inport\_unit             => inport\_unit,                  outport\_unit            => outport\_unit,                                           r14in                  =>r14in                                  );  end architecture structural; |

## ALU.vhd

|  |
| --- |
| library IEEE;  use IEEE.std\_logic\_1164.all;  use ieee.numeric\_std.all;  use ieee.std\_logic\_unsigned.all;  entity ALU is  port( A,B: in std\_logic\_vector(31 downto 0);          ALUcontrol\_signals: in std\_logic\_vector(12 downto 0);--ADD, SUB,MUL,DIV,SHIFT,ROTATE, SHL,SHR, AND\_control, OR\_control: in std\_logic;          C\_hi: out std\_logic\_vector(31 downto 0); --          C\_lo: out std\_logic\_vector(31 downto 0);          clk: in std\_logic    );  end ALU;  architecture behaviour of ALU is    COMPONENT booth      pORT(              Ain :   IN STD\_LOGIC\_VECTOR(31 downto 0);              Bin :  IN  STD\_LOGIC\_VECTOR(31 DOWNTO 0);              output :  OUT  STD\_LOGIC\_VECTOR(63 DOWNTO 0)          );  END COMPONENT;  COMPONENT alu\_div      PORT(denom : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);           numer : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);           quotient : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);           remain : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)      );  END COMPONENT;  SIGNAL  booth\_out : STD\_LOGIC\_VECTOR(63 DOWNTO 0);  SIGNAL  div\_quo,div\_rem:  STD\_LOGIC\_VECTOR(31 DOWNTO 0);    begin      booth\_mul: booth      PORT MAP(              Ain => A,              Bin => B,              output => booth\_out          );        process(clk) is      begin      if rising\_edge(clk) then          case ALUcontrol\_signals is              when "1000000000000" => c\_hi <=  x"0000\_0000" ;  --add                                        c\_lo <= std\_logic\_vector(unsigned(A) + unsigned(B));              when "0100000000000" => c\_hi <=   x"0000\_0000";  -- sub                                      c\_lo <=  std\_logic\_vector(unsigned(A) - unsigned(B));              when "0010000000000" => c\_hi <=   x"0000\_0000"; --srl                                      c\_lo <=  std\_logic\_vector(unsigned(A) srl to\_integer (unsigned(B)));              when "0001000000000" => c\_hi <=   x"0000\_0000";  --sll                                      c\_lo <=  std\_logic\_vector(unsigned(A) sll to\_integer (unsigned(B)));                when "0000100000000" => c\_hi <=   x"0000\_0000";  --ror                                      c\_lo <=  std\_logic\_vector(unsigned(A) ror to\_integer (unsigned(B)));              when "0000010000000" => c\_hi <=   x"0000\_0000";   --rol                                      c\_lo <=  std\_logic\_vector(unsigned(A) rol to\_integer (unsigned(B)));                when "0000001000000" => c\_hi <=  x"0000\_0000";          -- neg                                      c\_lo <= std\_logic\_vector((Not( unsigned(B)))+1);              when "0000000100000" => c\_hi <=   x"0000\_0000";   --equal                                      c\_lo <=  (std\_logic\_vector(unsigned(B)));              when "0000000010000" => c\_hi <= booth\_out(63 downto 32); --Booth algorithm                                      c\_lo <= booth\_out(31 downto 0);              when "0000000001000" => c\_hi <= x"0000\_0000"; --and                                      c\_lo <= (A and B);              when "0000000000100" => c\_hi <= x"0000\_0000"; --or                                      c\_lo <= (A OR B);              when "0000000000010" => c\_hi <=  x"0000\_0000";          -- not                                      c\_lo <= std\_logic\_vector((Not( unsigned(B))));                  when "0000000000001" => c\_hi <= std\_logic\_vector(unsigned(A) mod unsigned(B));  -- divide                                      c\_lo <= std\_lOGIC\_VECTOR((unsigned(A) / unsigned(B)));              when "0000000000011" => c\_hi <=  x"0000\_0000" ;  --addi                                      c\_lo <= std\_logic\_vector(unsigned(A) + unsigned(B));              when "0000000000111" => c\_hi <= x"0000\_0000"; --andi                                      c\_lo <= (A and B);              when "0111111111111" => c\_hi <= x"0000\_0000";     --increment pc value                                      c\_lo <= (B+1);              when others => NULL;          end case;          end if;      end process;  end behaviour; |

## busMux.vhd

|  |
| --- |
| LIBRARY ieee;  USE ieee.std\_logic\_1164.ALL;  ENTITY busMux IS  generic(          VAL : std\_logic\_vector(31 downto 0) := x"00000000"          );  PORT(      r00\_in, r01\_in, r02\_in, r03\_in,      r04\_in, r05\_in, r06\_in, r07\_in,      r08\_in, r09\_in, r10\_in, r11\_in,      r12\_in, r13\_in, r14\_in, r15\_in,      BMIn\_MDR, inhi, inlo ,BMI\_z\_hi,          BMI\_z\_lo, BMI\_inpc, BMI\_portin, BMIcSignExtended    :   IN std\_logic\_vector(31 downto 0);      s\_in        :   IN std\_logic\_vector(4 downto 0);      BusMuxOut   :   OUT std\_logic\_vector(31 downto 0):=VAL  );  END busMux;  ARCHITECTURE behavioural OF busMux IS  BEGIN      BusMux: PROCESS(s\_in, r00\_in, r01\_in, r02\_in, r03\_in,      r04\_in, r05\_in, r06\_in, r07\_in,      r08\_in, r09\_in, r10\_in, r11\_in,      r12\_in, r13\_in, r14\_in, r15\_in,      BMIn\_MDR, inhi, inlo ,BMI\_z\_hi,          BMI\_z\_lo, BMI\_inpc, BMI\_portin,          BMIcSignExtended)      BEGIN          CASE s\_in IS              when "00000"    =>  BusMuxOut <= r00\_in;              when "00001"    =>  BusMuxOut <= r01\_in;              when "00010"    =>  BusMuxOut <= r02\_in;              when "00011"    =>  BusMuxOut <= r03\_in;              when "00100"    =>  BusMuxOut <= r04\_in;              when "00101"    =>  BusMuxOut <= r05\_in;              when "00110"    =>  BusMuxOut <= r06\_in;              when "00111"    =>  BusMuxOut <= r07\_in;              when "01000"    =>  BusMuxOut <= r08\_in;              when "01001"    =>  BusMuxOut <= r09\_in;              when "01010"    =>  BusMuxOut <= r10\_in;              when "01011"    =>  BusMuxOut <= r11\_in;              when "01100"    =>  BusMuxOut <= r12\_in;              when "01101"    =>  BusMuxOut <= r13\_in;              when "01110"    =>  BusMuxOut <= r14\_in;              when "01111"    =>  BusMuxOut <= r15\_in;              when "10000"    =>  BusMuxOut <= inhi;              when "10001"    =>  BusMuxOut <= inlo;              when "10010"    =>  BusMuxOut <= BMI\_z\_hi;              when "10011"    =>  BusMuxOut <= BMI\_z\_lo;              when "10100"    =>  BusMuxOut <= BMI\_inpc;              when "10101"   => BusMuxOut <= BMIn\_MDR;              when "10110"    =>  BusMuxOut <= BMI\_portin;              when "10111"    =>  BusMuxOut <= BMIcSignExtended;              when others     => BusMuxOut <= (others=>'0');          END CASE;      END PROCESS;  END behavioural; |

## encoder.vhd

|  |
| --- |
| LIBRARY ieee;  USE ieee.std\_logic\_1164.ALL;  ENTITY encoder IS  PORT(      r00\_out, r01\_out, r02\_out, r03\_out,      r04\_out, r05\_out, r06\_out, r07\_out,      r08\_out, r09\_out, r10\_out, r11\_out,      r12\_out, r13\_out, r14\_out, r15\_out,      hi\_out, lo\_out, z\_hi\_out, z\_lo\_out,      PC\_out, MDR\_out, port\_out, c\_out    : IN std\_logic;      s\_out   :   OUT std\_logic\_vector(4 downto 0)  );  END encoder;  ARCHITECTURE behavioural OF encoder IS  BEGIN      s\_out <=    "00000" when (r00\_out='1')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "00001" when (r00\_out='0')and(r01\_out='1')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "00010" when (r00\_out='0')and(r01\_out='0')and(r02\_out='1')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "00011" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='1')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "00100" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='1')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "00101" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='1')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "00110" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='1')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "00111" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='1')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "01000" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='1')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "01001" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='1')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "01010" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='1')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "01011" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='1')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "01100" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='1')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "01101" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='1')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "01110" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='1')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "01111" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='1')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "10000" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='1')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "10001" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='1')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "10010" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='1')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "10011" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='1')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "10100" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='1')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      "10101" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='1')and(port\_out='0')and(c\_out='0') else                      "10110" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='1')and(c\_out='0') else                      "10111" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='1') else                      "11110" when (r00\_out='0')and(r01\_out='0')and(r02\_out='0')and(r03\_out='0')and(r04\_out='0')and(r05\_out='0')and(r06\_out='0')and(r07\_out='0')and(r08\_out='0')and(r09\_out='0')and(r10\_out='0')and(r11\_out='0')and(r12\_out='0')and(r13\_out='0')and(r14\_out='0')and(r15\_out='0')and(hi\_out='0')and(lo\_out='0')and(z\_hi\_out='0')and(z\_lo\_out='0')and(PC\_out='0')and(MDR\_out='0')and(port\_out='0')and(c\_out='0') else                      --^ signal for no input                      "11111";  END behavioural; |

## MAR.vhd

|  |
| --- |
| library IEEE;  use IEEE.std\_logic\_1164.all;  entity MAR is      generic(          VAL : std\_logic\_vector(31 downto 0);            ADDRESS : std\_logic\_vector(8 downto 0):= b"000000000"      );      port(clr, clk, wr: in std\_logic;              inputD: in std\_logic\_vector(31 downto 0);              outputQ : buffer std\_logic\_vector(8 downto 0) := ADDRESS              );  end entity MAR;  ARCHITECTURE behaviour OF MAR IS  signal x: std\_logic\_vector(8 downto 0);  BEGIN      process(clk, clr)      begin              x <= inputD(8 downto 0);          if clr = '1' then              outputQ <= b"000000000";          elsif rising\_edge(clk) then              if wr = '1' then                  outputQ <= x;              end if;          end if;      end process;  end architecture behaviour; |

## MDR.vhd

|  |
| --- |
| LIBRARY ieee;  USE ieee.std\_logic\_1164.ALL;  ENTITY MDR IS      generic(          VAL : std\_logic\_vector(31 downto 0):= x"00000000"      );  PORT(      busMuxOut, MDataIn          :   IN std\_logic\_vector(31 downto 0);      clr,clk,mdr\_in,MDRread      :   IN std\_logic;      outputQ                         :   OUT std\_logic\_vector(31 downto 0):=VAL  );  END MDR;  ARCHITECTURE behavioural OF MDR IS      BEGIN      PROCESS(clr, clk, BusMuxOut, MDataIn, MDRRead, mdr\_in)      BEGIN      IF clr ='1' then          outputQ <= x"00000000";      ELSE          IF rising\_edge(clk) then              IF mdr\_in = '1' then                  IF (MDRRead = '0') THEN                      outputQ <= busMuxOut;                  ELSIF (MDRRead = '1') THEN                      outputQ <= MdataIn;                  END IF;              END IF;          END IF;      END IF;      END PROCESS;  END; |

## Select\_and\_encoder.vhd

|  |
| --- |
| LIBRARY IEEE;  USE IEEE.STD\_LOGIC\_1164.ALL;  entity select\_and\_encoder is      generic(          VAL : std\_logic\_vector(15 downto 0):= x"0000"      );  port (              IR\_in        : in std\_logic\_vector(31 downto 0);              Rin     : in std\_logic;              Rout    : in std\_logic;              BAout   : in std\_logic;              Gra, Grb, Grc : in std\_logic;              R0to15\_out     : out std\_logic\_vector(15 downto 0):=VAL;              R0to15\_in     : out std\_logic\_vector(15 downto 0):=VAL;              c\_sign\_extended : out std\_logic\_vector(31 downto 0)    );  end entity select\_and\_encoder;  ARCHITECTURE behavioural OF select\_and\_encoder IS      signal interim : std\_logic\_vector(3 downto 0);      begin      c\_sign\_process : process(IR\_in)          begin          for i in 31 downto 18 loop              c\_sign\_extended(i) <= IR\_in(18);          end loop;          c\_sign\_extended(17 downto 0) <= IR\_in(17 downto 0);      end process;        init : process(IR\_in, Gra, Grb, Grc)      begin          if (Gra ='1') then              interim <= IR\_in(26 downto 23);          elsif (Grb = '1') then              interim <= IR\_in(22 downto 19);          elsif (Grc = '1') then              interim <= IR\_in(18 downto 15);          else              --shouldn't get here!          end if;      end process init;        result : process(interim, Rin, Rout, BAout)          begin          if Rin = '1' then              case interim is                  when "0000" => R0to15\_in <= b"0000\_0000\_0000\_0001";                  when "0001" => R0to15\_in <= b"0000\_0000\_0000\_0010";                  when "0010" => R0to15\_in <= b"0000\_0000\_0000\_0100";                  when "0011" => R0to15\_in <= b"0000\_0000\_0000\_1000";                  when "0100" => R0to15\_in <= b"0000\_0000\_0001\_0000";                  when "0101" => R0to15\_in <= b"0000\_0000\_0010\_0000";                  when "0110" => R0to15\_in <= b"0000\_0000\_0100\_0000";                  when "0111" => R0to15\_in <= b"0000\_0000\_1000\_0000";                  when "1000" => R0to15\_in <= b"0000\_0001\_0000\_0000";                  when "1001" => R0to15\_in <= b"0000\_0010\_0000\_0000";                  when "1010" => R0to15\_in <= b"0000\_0100\_0000\_0000";                  when "1011" => R0to15\_in <= b"0000\_1000\_0000\_0000";                  when "1100" => R0to15\_in <= b"0001\_0000\_0000\_0000";                  when "1101" => R0to15\_in <= b"0010\_0000\_0000\_0000";                  when "1110" => R0to15\_in <= b"0100\_0000\_0000\_0000";                  when "1111" => R0to15\_in <= b"1000\_0000\_0000\_0000";                  when others => R0to15\_in <= b"0000\_0000\_0000\_0000";              end case;          else              R0to15\_in <= b"0000\_0000\_0000\_0000";          end if;          if BAout = '1' then              if interim = "0000" then                  R0to15\_out <= b"0000\_0000\_0000\_0001";              end if;          end if;          if Rout = '1' then              case interim is                  when "0000" =>R0to15\_out <= b"0000\_0000\_0000\_0001";                  when "0001" => R0to15\_out <= b"0000\_0000\_0000\_0010";                  when "0010" => R0to15\_out <= b"0000\_0000\_0000\_0100";                  when "0011" => R0to15\_out <= b"0000\_0000\_0000\_1000";                  when "0100" => R0to15\_out <= b"0000\_0000\_0001\_0000";                  when "0101" => R0to15\_out <= b"0000\_0000\_0010\_0000";                  when "0110" => R0to15\_out <= b"0000\_0000\_0100\_0000";                  when "0111" => R0to15\_out <= b"0000\_0000\_1000\_0000";                  when "1000" => R0to15\_out <= b"0000\_0001\_0000\_0000";                  when "1001" => R0to15\_out <= b"0000\_0010\_0000\_0000";                  when "1010" => R0to15\_out <= b"0000\_0100\_0000\_0000";                  when "1011" => R0to15\_out <= b"0000\_1000\_0000\_0000";                  when "1100" => R0to15\_out <= b"0001\_0000\_0000\_0000";                  when "1101" => R0to15\_out <= b"0010\_0000\_0000\_0000";                  when "1110" => R0to15\_out <= b"0100\_0000\_0000\_0000";                  when "1111" => R0to15\_out <= b"1000\_0000\_0000\_0000";                  when others => R0to15\_out <= b"0000\_0000\_0000\_0000";              end case;          else              --if BAout = '0' then                  R0to15\_out <= b"0000\_0000\_0000\_0000";              --end if;          end if;      end process result;    END; |

## CON\_FF\_Logic.vhd

|  |
| --- |
| LIBRARY ieee;  USE ieee.std\_logic\_1164.ALL;  use ieee.numeric\_std.all;  use ieee.std\_logic\_unsigned.all;  entity CON\_FF\_Logic is      port(          IR\_in\_conff: in std\_logic\_vector(31 downto 0);          BusMuxout : in std\_logic\_vector(31 downto 0);          Con\_in    : in std\_logic;          outputQ    : out std\_logic      );  end entity CON\_FF\_Logic;  architecture behavior of CON\_FF\_LOGIC is      BEGIN      process(BusMuxout,IR\_in\_conff,Con\_in)          begin              if (Con\_in = '1') then                  case IR\_in\_conff(20 downto 19) is                      when "00" =>                          if BusMuxout = x"00000001" then outputQ <= '1'; --branch if equal to 0                          else outputQ <= '0';                          end if;                      when "01" =>                          if BusMuxout = x"00000001" then outputQ <= '1'; --branch if equal to nonzero                          else outputQ <= '0';                          end if;                      when "10" =>                          if BusMuxout = x"00000001" then outputQ <= '1'; --branch if positive                          else outputQ <= '0';                          end if;                      when "11" =>                          if BusMuxout < x"00000000" then outputQ <= '1'; --branch if negative                          else outputQ <= '0';                          end if;                      when others => NULL;                  end case;              else              outputQ <= '1';              end if;      end process;  END; |

## Register32.vhd

|  |
| --- |
| library IEEE;  use IEEE.std\_logic\_1164.all;  entity register32 is      generic(          VAL : std\_logic\_vector(31 downto 0):= x"00000000"      );      port(clr, clk, wr: in std\_logic;              inputD: in std\_logic\_vector(31 downto 0);              outputQ : out std\_logic\_vector(31 downto 0):=VAL              );  end entity register32;  ARCHITECTURE behaviour OF register32 IS  BEGIN        process(clk)        begin          if clr = '1' then              outputQ <= x"00000000";          elsif (clk'event and clk ='1') then              if (wr = '1') then                       outputQ <= inputD;              end if;         end if;      end process;  end architecture behaviour; |