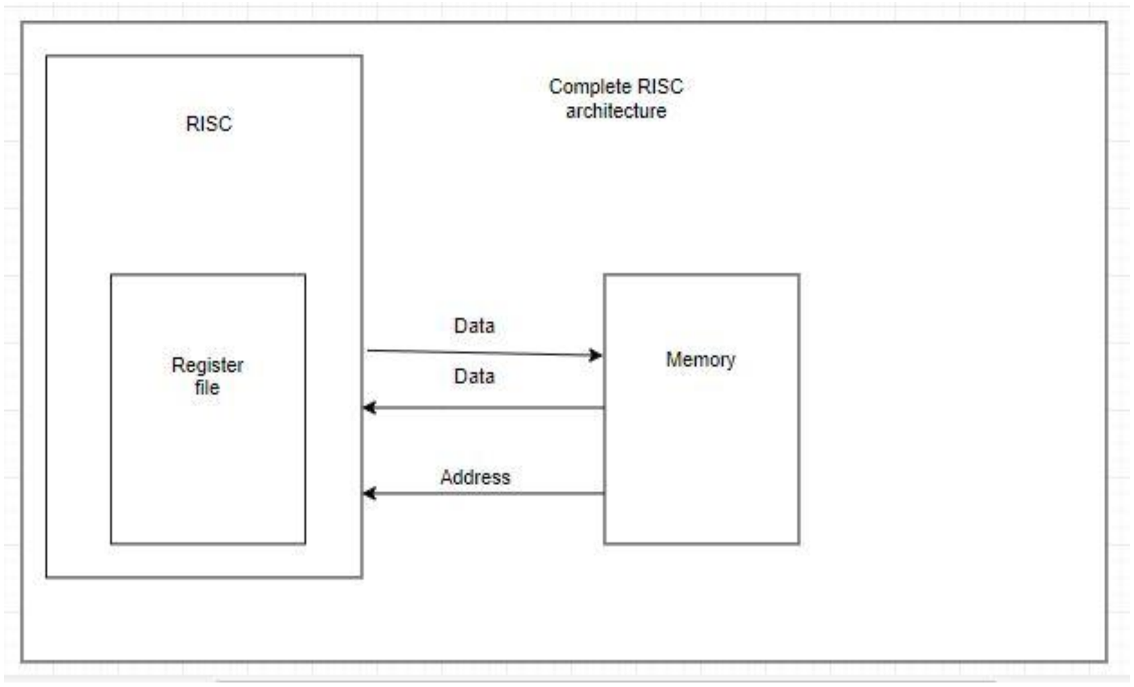
Appendix A: VHDL Code for building the processor

1. RISC Processor in Nutshell



2. Code for memory

```
--------------------------------------------------------------------------------
-- Project 520
-- Author_1: Kushan Parikh
-- Author_2: Aditya Tumsare
-- Module Name: memory - Behavioral
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity memory is
   Port ( Address : in unsigned (6 downto 0);
            Read_instr : out unsigned (31 downto 0);
            Write_data : in unsigned (31 downto 0);
            Write_enable : in STD_LOGIC;
            clk, cs: in STD_LOGIC);
end memory;

architecture Behavioral of memory is
type RAMtype1 is array (0 to 127) of unsigned (31 downto 0);
signal mem_type : RAMtype1 := (others => (others => '0'));
begin
    Read_instr <= (others => 'Z') when cs = '0' or Write_enable = '1' else Read_instr;
 -- if chip not selected or write enable is 1 then data in 'Z'
    process(CLK)
    begin
        if CLK = '0' and CLK'event then
            if data_write_enable = '1' then
                        mem_type(to_integer(Adr)) <= write_data;
            end if;
                        Read_instr <= mem_type(to_integer(Adr));
        end if;
    end process;
end Behavioral;
```

3. Code for Register file

```
-------------------------------------------------------------------------------
-- Project 520
-- Author_1: Kushan Parikh
-- Author_2: Aditya Tumsare
-- Module Name: register_file - Behavioral
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity register_file is
  Port ( SR1 : in unsigned (4 downto 0);
       SR2 : in unsigned (4 downto 0);
       DR : in unsigned (4 downto 0);
       Read_port1 : out unsigned (31 downto 0);
       Read_port2 : out unsigned (31 downto 0);
       Write_port : in unsigned (31 downto 0);
       Write_enable3 : in STD_LOGIC;
       CLK : in STD_LOGIC);
end register_file;

architecture Behavioral of register_file is
type RAMtype1 is array (0 to 31) of unsigned (31 downto 0);
signal RAM1 : RAMtype1 := (others => (others => '1'));
begin
  process(clk)
  begin
    if CLK = '1' and CLK'event then
      if Write_Enable3 = '1' then
        RAM1(to_integer(DR)) <= Write_Data3;    --writing the data into desination register
      end if;
    end if;
  end process;
```

```
Read_Data1 <= RAM1(to_integer(SR1));    --reading data from source register
Read_Data2 <= RAM1(to_integer(SR2));    --reading data from source register
end Behavioral;
```

4. Code for processor

```
--------------------------------------------------------------------------------
-- Project 520
-- Author_1: Kushan Parikh
-- Author_2: Aditya Tumsare
-- Module Name: risc_proc - Behavioral
--------------------------------------------------------------------------------s
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity riscv_proc is
 Port (clk, rst : in  bit;
       cs, wr   : out bit := '0';
       addr     : out unsigned(31 downto 0) := x"00000000";
       inst_r   : in unsigned(31 downto 0)  := x"00000000";
       inst_w   : out unsigned(31 downto 0) := x"00000000" ;
       addr_ram : out unsigned(31 downto 0) := x"00000FFF";
       mode     : out unsigned(1 downto 0);
       cs_dm    : out bit := '0';                   -- chip select data memory signal
       wr_dm    : out bit := '0';                   -- write data memory signal
       d_r      : in  unsigned(31 downto 0) := x"00000000";       -- data memory read bus
       d_w      : out unsigned(31 downto 0) := x"00000000");      -- data memory write bus for
simulation purpose
end riscv_proc;
```

```
architecture Behavioral of riscv_proc is


            Component register_file is
               Port ( SR1 : in unsigned (31 downto 0);
           SR2 : in unsigned (31 downto 0);
           DR : in unsigned (31 downto 0);
           Read_port1 : out unsigned (31 downto 0);
           Read_port2 : out unsigned (31 downto 0);
           Write_port : in unsigned (31 downto 0);
           Write_enable3 : in STD_LOGIC;
           clk, rst : in STD_LOGIC);
            end component;


--Format of instructions coverd
            type operation is (add, sub,          -- Arithmatic
                  lw,                  -- Load
                  xor_r, or_r, and_r    -- Logical
                  );
signal instr : unsigned(31 downto 0) := x"00000000";                        -- Instruction
signal pc, npc, tmp_cur_pc : unsigned(31 downto 0) := x"00000000";          -- program counter
and next pc


--source, register_file and ALU signals
signal rs1_data, rs2_data, alu_out : unsigned(31 downto 0) := x"00000000";
signal op : operation;                                          -- stores ALU control signal
type i_fmt is (R, I, U);                                        -- Instruction Basic types
signal format : i_fmt := R;                                             -- Instruction Format
signal wr_data : unsigned(31 downto 0) := x"00000000";                      -- write data to destination
register
```

```
signal wr_en : bit := '0';                                  -- Enable register file to write data

-- sign immediate extension signals for I and U type instructions
signal sig_imm_i, sig_imm_u : unsigned(31 downto 0) := x"00000000";

alias opcode1: unsigned(1 downto 0) is instr(1 downto 0);          -- opcodes
alias opcode2: unsigned(4 downto 0) is instr(6 downto 2);          -- opcodes
alias rd:      unsigned(4 downto 0) is instr(11 downto 7);         -- destination register in reg_file
alias func3:   unsigned(2 downto 0) is instr(14 downto 12);        -- opcode
alias rs1:     unsigned(4 downto 0) is instr(19 downto 15);        -- source regiseter1 in reg_file
alias rs2:     unsigned(4 downto 0) is instr(24 downto 20);        -- opcode
alias func7:   unsigned(6 downto 0) is instr(31 downto 25);        -- source register2 in reg_file
alias imm_i:   unsigned(11 downto 0) is instr(31 downto 20);       -- Immediate for I type
instruction
alias imm_u:   unsigned(19 downto 0) is instr(31 downto 12);       -- Immediate for U type
instruction

-- State Machine Variables
type cpu_state is (reset, fetch, decode, execute, memory_access, write_back);
signal pstate, nstate : cpu_state := (fetch);              -- Default in fetch state



begin

   -- register file is part of CPU
   reg_file0: reg_file port map(clk, rst, rs1, rs2, rd, wr_data, wr_en, rs1_data, rs2_data);

   -- connect program counter to address bus of Instruction Memory
   addr <= pc;
```

```
-- Read Instruction From Instruction memory
instr <=  inst_r when pstate = fetch else instr;

 process(clk, rst)
     begin
         if (rst = '1') then
             pstate <= reset;
         else if (clk = '1' and clk'event) then
             pstate <= nstate;
             tmp_cur_pc <= npc;
         end if;
         end if;
    end process;

 process(pstate)
 begin

    -- I type instruction Immediate is sign extended
      sig_imm_i(11 downto 0) <= imm_i(11 downto 0);
      sig_imm_i(31 downto 12) <= (others => imm_i(11));

      -- U type instruction Immediate is sign extended
      sig_imm_u(19 downto 0) <= imm_u(19 downto 0);
      sig_imm_u(31 downto 12) <= (others => imm_u(19));

      cs <= '0';                          -- Instruction memory disable by default
      wr <= '0';                          -- Instruction memory write disable
```

```
cs_dm <= '0';                      -- data memory disable by default
wr_dm <= '0';                      -- data memory write disable
wr_en <= '0';                      -- register file write disable
npc <= tmp_cur_pc;                 -- Hold npc

case pstate is
   when reset =>                   -- CPU in reset state
      pc <= x"00000000";           -- default program counter
      npc <= pc;                   -- next program counter
      cs <= '1';                   -- enable flash to read
      wr <= '0';                   -- write disable in flash memory
      addr_ram <= x"00000fff";     -- Last address of RAM, as stack down growing
      nstate <= fetch;             -- next state is fetch

   when fetch =>                                   -- Stage FETCH Instruction
      pc <= tmp_cur_pc;
      cs <= '1';                   -- select instruction memory chip
      wr <= '0';                   -- read memory
      npc <= tmp_cur_pc+1;         -- increment program counter
      nstate <= decode;            -- next state

   when decode =>                                  -- Stage 2 DECODE Opcode and Register
Fetch
        if (opcode1 = "11") then
           case opcode2 is
              when "01100" =>
                 format <= R;                      -- R type Instruction
                 case func3 is
                    when "000" =>
```

```
            if (func7 = "0000000") then
                op <= add;
            else if (func7 = "0100000") then
                op <= sub;
            end if;
            end if;
        when "001" => op <= sll_r;
        when "010" => op <= slt;
        when "011" => op <= sltu;
        when "101" =>
            if(func7 = "0000000") then
                op <= srl_r;
            else if (func7 = "0100000") then
                op <= sra_r;
            end if;
            end if;
        when "100" => op <= xor_r;
        when "110" => op <= or_r;
        when "111" => op <= and_r;
        when others =>
   end case;
when "00000" =>                                    -- load instructions
    format <= I;                                -- I type Instruction
    case func3 is
        when "010" => op <= lw;
        when others =>
    end case;
when "01101" => op <= lui; format <= U;            -- U type Instruction
```

```
            when "00101" => op <= auipc; format <= U;
            when "00100" =>
                format <= I;                         -- I type Instruction
                case func3 is
                    when "000" => op <= addi;
                    when "001" => op <= slli;
                    when "010" => op <= slti;
                    when "011" => op <= sltiu;
                    when "100" => op <= xori;
                    when "101" =>
                        case func7 is
                            when "0000000" => op <= srli;       -- shift right logical Immediate
                            when "0100000" => op <= srai;       -- shift right arithmatic Immediate
                            when others =>
                        end case;
                    when "110" => op <= ori;
                    when "111" => op <= andi;
                    when others =>
                end case;
        nstate <= execute;

    when execute =>                              -- Stage 3 EXECUTE INSTRUCTION
        case op is
            -- Arithmetic instructions
            when add => alu_out <= rs1_data + rs2_data;
            when sub => alu_out <= rs1_data - rs2_data;
            when or_r  => alu_out <= rs1_data or rs2_data;
            when and_r => alu_out <= rs1_data and rs2_data;
```

```
          when xor_r => alu_out <= rs1_data xor rs2_data;
                alu_out(31) <=  rs1_data(31);

          when others =>
       end case;
nstate <= memory_access;

when memory_access =>                          -- Stage 4 MEMORY_ACCESS
    case op is
        when lw=>
            addr_ram <= alu_out;
            --lw_data <= d_r;
            mode <= "01";
            wr_dm <= '0';
            cs_dm <= '1';
    when others =>
    end case;
nstate <= write_back;

when write_back =>                          -- stage 5 WRITE_BACK
    if format = R or format = U then
        wr_data <= alu_out;
        wr_en <= '1';
    else if format = I then
        if (op = lw) then
            wr_data <= d_r;
            cs_dm <= '0';
            wr_en <= '1';
```

```
                        else
                            wr_data <= alu_out;
                            wr_en <= '1';
                        end if;
                    end if;
                nstate <= fetch;
            end case;
     end process;

end Behavioral;
```

5. Code for testing processor

```
----------------------------------------------------------------------------------
-- Project 520
-- Author_1: Kushan Parikh
-- Author_2: Aditya Tumsare
-- Module Name: riscv_cmt_proc - Behavioral
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity riscv_cmt_proc is
    Port ( CLK : in STD_LOGIC;
           ALU_out : in STD_LOGIC);
end riscv_cmt_proc;

architecture Behavioral of riscv_cmt_proc is
```

```
begin


component riscv_proc

Port (clk, rst : in  bit;

      cs, wr   : out bit := '0';

      addr     : out unsigned(31 downto 0) := x"00000000";

      inst_r   : in unsigned(31 downto 0)  := x"00000000";

      inst_w   : out unsigned(31 downto 0) := x"00000000" ;

      addr_ram : out unsigned(31 downto 0) := x"00000FFF";

      mode     : out unsigned(1 downto 0);

      cs_dm    : out bit := '0';              -- chip select data memory signal

      wr_dm    : out bit := '0';              -- write data memory signal

      d_r      : in  unsigned(31 downto 0) := x"00000000";     -- data memory read bus

      d_w      : out unsigned(31 downto 0) := x"00000000");

end componenent;


component memory

      Port ( Address : in unsigned (6 downto 0);

          Read_instr : out unsigned (31 downto 0);

          Write_data : in unsigned (31 downto 0);

          Write_enable : in STD_LOGIC;

          clk, cs: in STD_LOGIC);

end componenent;



type instructions is array (0 to code_len) of unsigned(31 downto 0);  -- Memory data type to
store instructions

signal clk, rst : bit;
```

```
--Memory Variables
signal cs_mem, wr_mem, cs_flash, wr_flash : bit;
signal data_in, data_out : unsigned(31 downto 0);
signal data_r, data_w : unsigned(31 downto 0);
signal addr_flash, addr_mem, addr_tb : unsigned(31 downto 0) := x"00000000";



begin


    cpu0  : cpu port map (clk, rst, cs_flash, wr_flash, addr_flash, data_r, data_w, addr_ram,
mode_ram, cs_ram, wr_ram, d_r_cpu, d_w_cpu);


    flash : memory port map (addr_mem, data_in, data_out, wr_mem, clk, cs_mem);


    clk <= not clk after 10 ns;


    type Iarr is array(1 to W) of unsigned(31 downto 0);
    constant Instr_List: Iarr : = (
        x"30000000", -- Assume this is an hex code for instruction 1 (i1)
        x"20010006" -- Assume this is an hex code for instruction 1 (i2)
        );


    --memory
    wr_mem <= wr_flash;
    cs_mem <= cs_flash;
    addr_mem <= addr_tb when rst = '1' else addr_flash;
    data_r <= data_out;
    data_in <= d_w_cpu;
    d_r_cpu <= Instr_List;
    process
    begin
        rst <= '1';
        wait until clk='1' and clk'event;   -- 1    reset on

        rst <= '0';                          -- reset off

        for i in 0 to code_len loop

            wait until clk = '1' and clk'event; -- n      fetch

            wait until clk = '1' and clk'event; -- n+1    decode

            wait until clk = '1' and clk'event; -- n+2    execute

            wait until clk = '1' and clk'event; -- n+3    Memory access

            wait until clk = '1' and clk'event; -- n+4    Write Back

        end loop;
    end process;

end Behavioral;
```