RISC-V 32I BASE IMPLEMENTATION in VHDL without JUMPS

Kwame Owusu Ampadu
BRANDENBURG UNIVERSITY OF TECHNOLOGY
ampadkwa@b-tu.de

Content

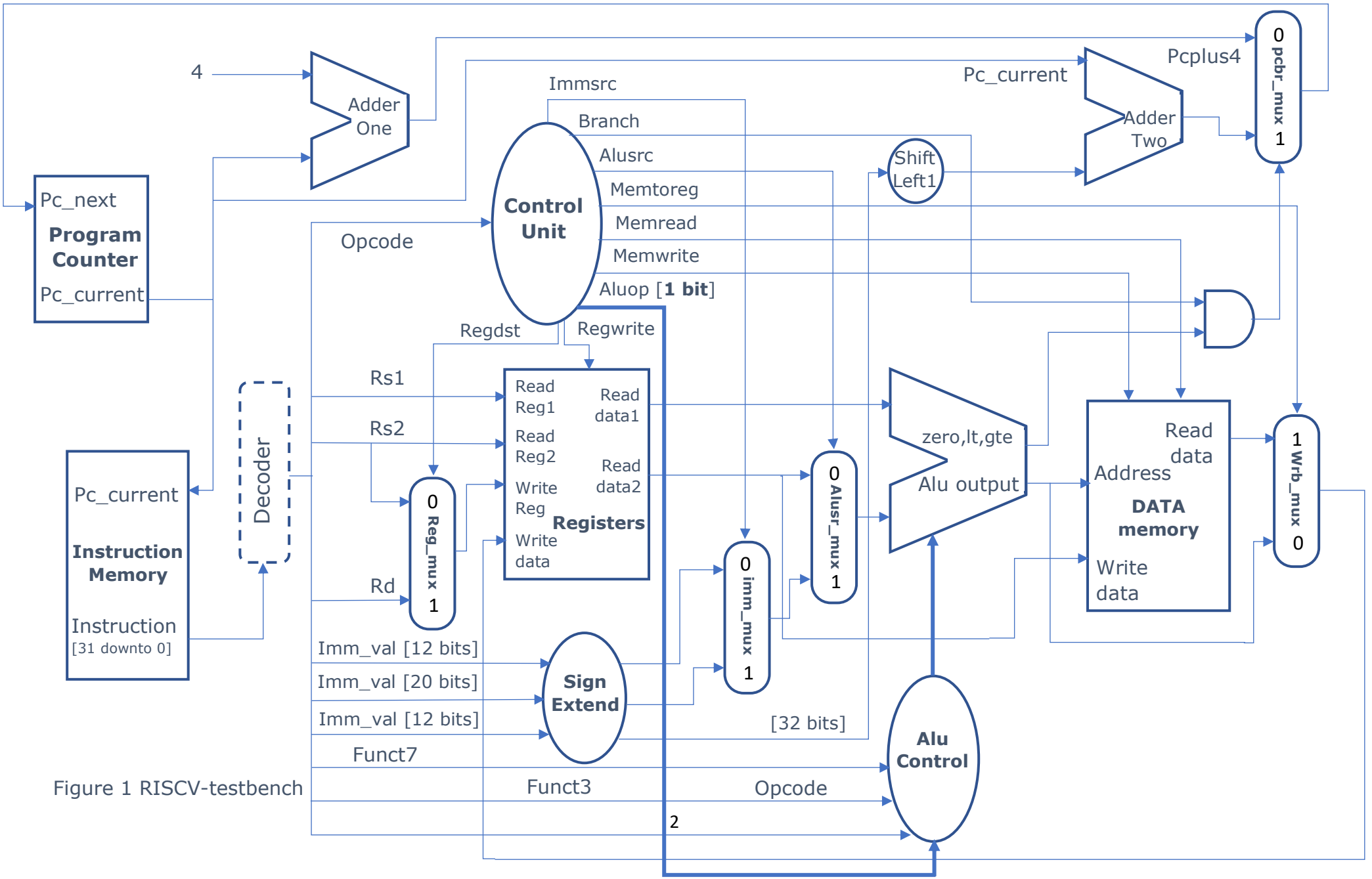

Figure 1 RISCV-testbench

INTRODUCTION

This documentation is to serve as a spring-board for anyone diving into microprocessor architecture implementation for the first time. MIPS is good and RISC-V itself has been described as an offshoot of MIPS – **M**icroprocessor without **I**nterlocked **P**ipeline **S**tages and therefore most training materials will begin with MIPS. The open-source RISC-V32I deviates widely from the MIPS 32 bits implementation as I came to find out. Hence a documentation of my experience in implementing a single cycle RV32I Base Instruction Set processor without Jumps in VHDL using ModelSim PE Student Edition 10.4a to aid students who might find it helpful. I must add that this documentation is based on a microprocessor architecture course project.

MICROPROCESSOR INTERCONNECTIONS

Every microprocessor consists of a **data path** and a **control path** that carry bits - binary digits (or signals) from instruction memory (as instructions) to or from data memory (as data). This means that after building your processor you will need to connect it to an instruction memory and a data memory for simulation.

In general, a **control path** distributes signals that wake-up components on the data path for signals to flow through based on an instruction type. The type of an instruction is specified by its **opcode.** All other parts of an instruction travel on the data path.

Table 1

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	**R - type**
imm[11:0]			rs1	funct3	rd	opcode	**I - type**
imm[11:5]		rs2	rs1	funct3	Imm[4:0]	opcode	**S - type**
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	**B - type**
imm[31:12]					rd	opcode	**U - type**
imm[20|10:1|11|19:12]					rd	opcode	**J - type**

The J-type instruction was not implemented in this introductory project. In figure 1, **the control path** is made up of the Control unit, the Alu-Control and all lines emanating from them.

The **data path** on the other hand consists of the remaining functional units that point to which instruction to fetch from instruction memory for bits to move into data memory or registers or retrieved from them. Data movement begins when a global clock has its logic value set to '1'. Since nothing happens at zero time, often the rising edge of the clock is used. In a processor this clock is a **single wire** that turns on ('1') and off ('0') to offer rhythmic timing. Processes that reference this timing are referred to as synchronous and asynchronous otherwise. In this single cycle implementation, the data path begins with the program counter and ends at the pcbr_mux, excluding the instruction memory and the data memory as afore

mentioned. The twenty-five RISCV-32I Base instructions to implement are stored in the **riscv_lib.vhd** file and can be referenced in the various .vhd files with *use work.riscv_lib.all;*

```
--------------------------------------------------------------------------------
--file: riscv_lib.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
--------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all; -- import std_logic types
use IEEE.std_logic_arith.all; -- import add/sub of std_logic_vector
use IEEE.std_logic_unsigned.all;

package riscv_lib is

	type instruction is (
		-- Load (upper) immediate operation
		-- U-type
		INST_LUI,

		-- Control operations
		-- B-type
		INST_BEQ,
		INST_BLT,
		INST_BGE,

		-- Memory operations
		INST_LW, -- I-type
		INST_SW, -- S-type

		-- Immediate operations
		-- I-type
		INST_ADDI,
		INST_SLTI,
		INST_SLTIU,
		INST_XORI,
		INST_ORI,
		INST_ANDI,

		-- Shifts
		-- R-type
		INST_SLLI,
		INST_SRLI,
		INST_SRAI,

		-- Register-to-Register
		-- R-type
		INST_ADD,
		INST_SUB,
```

```
        INST_SLL,
        INST_SLT,
        INST_SLTU,
        INST_XOR,
        INST_SRL,
        INST_SRA,
        INST_OR,
        INST_AND,

        -- No operation
        INST_NO_OP
    );

end package riscv_lib;
```

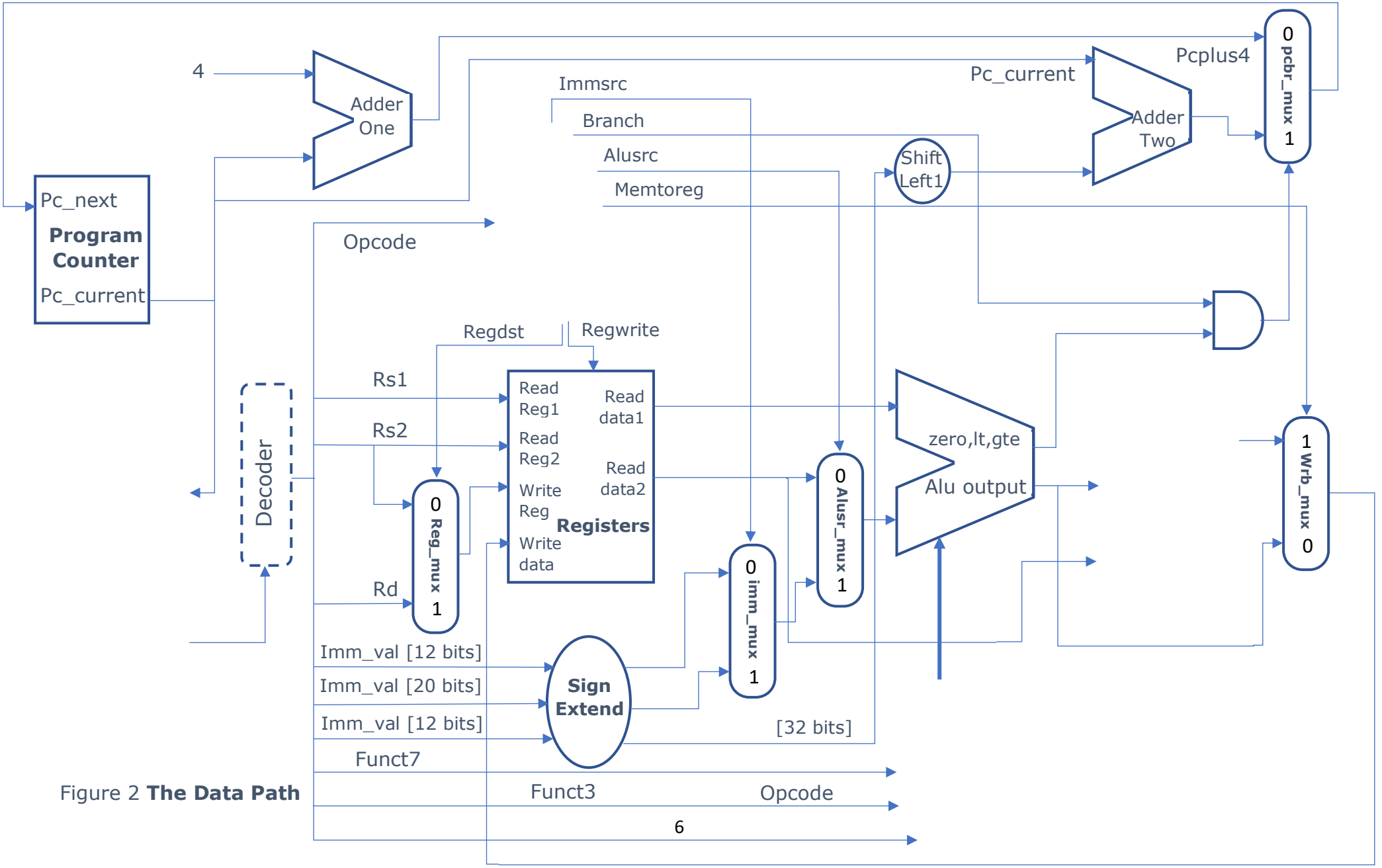

Figure 2 **The Data Path**

In figure 2, you will notice that though the control unit, the alu-control, the instruction memory and the data memory have been removed, arrows connecting to or from the data path have been left intact. These are going to be the input and output ports in the **VHDL data path file**.

Starting with the program counter each unit was implemented in VHDL and tested with its own test bench. This increased the number of files generated as the whole project could have been implemented using six VHDL files. The naming convention adopted used tb_name for testbenches and their signals.

The program counter file

The clocked program counter receives an input in the form of pc_next and generates a pc_current signal that flows to *instruction memory*, *adder one* and *adder two*. The connections are implemented in the **data path** file. The pc_current will introduce an *output port* on the data path for access to the *instruction memory*. The connections to the adder one and adder two will be internal to the data path.

Input signals by convention are assigned to the right side of VHDL statements whereas output signals are placed on the left side of such statements. To get around this, input and output signals can be assigned to internal signals which have the flexibility of appearing on both sides of the VHDL statements and then reassigned after use.

```
--------------------------------------------------------------------------------
--file: programcounter.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity programcounter is
	port (
		--inputs
		clk_i , rset_i	: in std_logic := '0';
		pc_next		: in std_logic_vector (31 downto 0);
		--outputs
		pc_current	: out std_logic_vector (31 downto 0)
	);
end programcounter;

architecture Behavioral of programcounter is
	signal s_pc_current: std_logic_vector (31 downto 0);
begin
	pc_proc: process (clk_i, rset_i)
	begin
		if (rset_i = '1') then
```

```
                s_pc_current <= x"00000000";
            elsif rising_edge(clk_i) then
                s_pc_current <= pc_next;
            end if;
        end process;
        pc_current <= s_pc_current;
end Behavioral;
```

The Adder one file

It takes 4Bytes to store 32 bits instruction addresses. Thus, to point to the next address the current position of the program counter must be incremented by 4. This is implemented in adder one which sends an *output* of pcplus4 through the pcbr_mux for pc_next.

```
----------------------------------------------------------------------------
--file: adderOne.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity adderOne is
        port(
            --inputs
            pc_current : in std_logic_vector (31 downto 0);
            --outputs
            pcplus4_o :   out std_logic_vector (31 downto 0)
        );
end adderOne;

architecture Behavioral of adderOne is

begin
        pcplus4_o <= std_logic_vector(unsigned(pc_current) + x"00000004");

end Behavioral;
```

The instruction decoder file

The decoder is the first point of entry for instructions onto the data path. The decoder thus introduces an *input port* in the **data path** file for instructions to come in. It then splits up the instructions and places the bits and pieces on the appropriate paths. These groups of bits are the opcode, rsi, rs2, rd, imm_val1, imm_val2, imm_val3, funct7 and funct3. The decoder thus has one input port and nine output ports. In addition, the decoder will introduce *output ports* in the data path file for opcode, funct7 and funct3 bits to enter into the **control path**.

The bit patterns are available in the RISC-V Instruction Set Manual.

```
----------------------------------------------------------------------------
--file: decoder.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity decoder is
	port (
		--inputs
		instr_i		: in std_logic_vector (31 downto 0); --receives output from instruction memory
		--outputs
		rs1_o		: out std_logic_vector (4 downto 0);
		rs2_o		: out std_logic_vector (4 downto 0);
		rd_o		: out std_logic_vector (4 downto 0);
		imm_o		: out std_logic_vector (11 downto 0);
		imm_sh		: out std_logic_vector (11 downto 0);
		imm_lui		: out std_logic_vector (19 downto 0);
		opcod_o		: out std_logic_vector (6 downto 0);
		func7_o		: out std_logic_vector (6 downto 0);
		func3_o		: out std_logic_vector (2 downto 0)
	);
end decoder;

architecture Behavioral of decoder is
	signal sig_instr   : std_logic_vector (31 downto 0);
	signal sig_rs1_o   : std_logic_vector (4 downto 0);
	signal sig_rs2_o   : std_logic_vector (4 downto 0);
	signal sig_rd_o    : std_logic_vector (4 downto 0);
	signal sig_imm_o   : std_logic_vector (11 downto 0);
	signal sig_imm_sh  : std_logic_vector (11 downto 0);
	signal sig_imm_lui : std_logic_vector (19 downto 0);
	signal sig_opcod_o : std_logic_vector (6 downto 0);
	signal sig_func7_o : std_logic_vector (6 downto 0);
	signal sig_func3_o : std_logic_vector (2 downto 0);
begin
	process (instr_i)
	begin
		--assign incoming instr_i to internal signal
			sig_instr   <= instr_i;
			sig_func7_o <= instr_i(31 downto 25);
			sig_rs2_o   <= instr_i(24 downto 20);
			sig_rs1_o   <= instr_i(19 downto 15);
			sig_func3_o <= instr_i(14 downto 12);
			sig_rd_o    <= instr_i(11 downto 7);
			sig_opcod_o <= instr_i(6 downto 0);
```

```
    end process;

    process (sig_instr)
    begin
        --decode or split instruction into sub-parts
    case (sig_opcod_o) is

        when "0010011" => --i-type
            sig_imm_o <= std_logic_vector(sig_instr(31 downto 20));
        when "0000011" => --load word (lw)
            sig_imm_o <= std_logic_vector(sig_instr(31 downto 20));

        when "0100011" => --store word (sw)
            sig_imm_sh <= std_logic_vector (sig_instr(31 downto 25) & sig_instr(11 downto 7));
        when "1100011" => --branch (beq, blt, bge)
            sig_imm_o <= std_logic_vector (sig_instr(31) & sig_instr(7) & sig_instr(30 downto 25)
            &   sig_instr(11 downto 8));

        when "0110111" => --LUI
            sig_imm_lui <= std_logic_vector (sig_instr(31 downto 12));

        when others => null; --implement other instructions down here
            sig_imm_o <= std_logic_vector(sig_instr(31 downto 20));
        end case;

    end process;
        rs1_o    <= sig_rs1_o;
        rs2_o    <= sig_rs2_o;
        rd_o     <= sig_rd_o;
        imm_o  <= sig_imm_o;
        imm_sh  <= sig_imm_sh;
        imm_lui         <= sig_imm_lui;
        opcod_o         <= sig_opcod_o;
        func7_o         <= sig_func7_o;
        func3_o         <= sig_func3_o;
end Behavioral;
```

The register multiplexor file

Not all instructions save results in the destination register rd. Some instructions direct computed results into rs2. The two-input one output, multiplexor known as reg_mux directs the register location based on the control signal regdst that flows from the **control path** into the **data path**. The regdst signal will also introduce an input port in the data path file.

```
----------------------------------------------------------------------------
--file: reg_mux.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
----------------------------------------------------------------------------
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity reg_mux is
	port(
		--inputs
		rs2_i0		: in ieee.numeric_std.unsigned(4 downto 0);
		dst_reg_i1	: in ieee.numeric_std.unsigned(4 downto 0);
		rg_dst		: in std_logic;

		--outputs
		wr_dst_o	: out ieee.numeric_std.unsigned(4 downto 0)
	);
end reg_mux;

architecture Behavioral of reg_mux is

begin
	wr_dst_o <= rs2_i0 when rg_dst = '0' else dst_reg_i1;

end Behavioral;
```

The register file

This comprises an array of 32 registers each storing 32-bit values. The register file receives five input signals and outputs data read from rs1 and rs2. The first address in the register file holds the value zero and must not be written to. Since writing takes place on the rising edge of the clock, the inputs to the register file must benefit from the global clock, increasing the number of input signals to seven. The write enable signal known as regwrite will emanate from the control path and must have an input into data path.

```
----------------------------------------------------------------------------
--file: regfile.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
----------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_textio.all;

use std.textio.all;

entity regfile is
	port (
		--inputs
		clk_i	: in std_logic;
```

```
		rst_i		: in std_logic;

		rd_addr1_i	: in ieee.numeric_std.unsigned(4 downto 0);
		rd_addr2_i	: in ieee.numeric_std.unsigned(4 downto 0);

		wr_data_i	: in std_logic_vector(31 downto 0);
		wr_addr_i	: in ieee.numeric_std.unsigned(4 downto 0);
		wr_en_i		: in std_logic;
		--outputs
		reg_data1_o : out std_logic_vector(31 downto 0);
		reg_data2_o : out std_logic_vector(31 downto 0)
	);
end entity regfile;

architecture behavioral of regfile is

	type reg_file_type is array (0 to 31) of std_logic_vector (31 downto 0);
	signal arr_reg : reg_file_type := (others => x"00000000");
	signal rg_data1_o: std_logic_vector (31 downto 0) := (others => '0');
	signal rg_data2_o: std_logic_vector (31 downto 0) := (others => '0');
begin

	write_process: process(clk_i)
	begin
		if (rst_i = '1') then
			for i in 0 to 31 loop
				arr_reg(i) <= x"00000000";
			end loop;
		elsif rising_edge(clk_i) then
			--if wr_en_i, write data to wr_addr
			if (wr_en_i = '1') then
				if (wr_addr_i /= "00000") then --avoid writing to the first location
					arr_reg(to_integer(wr_addr_i)) <= wr_data_i;
				end if;
			end if;
		end if;

	end process;

	--retrieve data from addr1 and addr2
	rg_data1_o <= arr_reg(to_integer(rd_addr1_i));
	rg_data2_o <= arr_reg(to_integer(rd_addr2_i));

	--assign data from addr1 and addr2 to signals
	reg_data1_o <= rg_data1_o; --arr_reg(to_integer(rd_addr1_i));
	reg_data2_o <= rg_data2_o; --arr_reg(to_integer(rd_addr2_i));

end behavioral;
```

The sign extender file

Immediate values from the decoder are either sign extended or padded with zeroes and transferred through output ports. The sign extender then has two 12 bits immediate input ports, one 20 bits immediate input port and three 32 bits output ports. All these signals are internal to the data path.

```
----------------------------------------------------------------------------
--file: signextend.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity signextend is
	port (
		--inputs
		se_i	: in std_logic_vector(11 downto 0);
		se_lui	: in std_logic_vector (19 downto 0);
		se_sh	: in std_logic_vector (11 downto 0);
		--outputs
		se_o	: out std_logic_vector(31 downto 0);
		se_lui_o: out std_logic_vector (31 downto 0);
		se_sh_o : out std_logic_vector (31 downto 0)
	);
end signextend;

architecture Behavioral of signextend is


begin
	--se_o <= "00000000000000000000" & se_i when se_i(11) = '0' else "11111111111111111111" & se_i;

	se_lui_o <= se_lui & x"000";

	se_o <= x"00000" & se_i when se_i(11) = '0' else x"fffff" & se_i;

	se_sh_o <= x"00000" & se_sh when se_sh(11) = '0' else x"fffff" & se_sh;

end Behavioral;
```

The sign extender multiplexor file

This two-input one output multiplexor also receives two immediate signals from the sign extender and directs them into the alu source multiplexor based on the state of the Immsrc signal from the control path. Thus, an input port must be created in the **data path** for the immsrc signal to come in.

```
----------------------------------------------------------------------------
--file: imm_mux.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity imm_mux is
	port (
		--inputs
		frm_seo_imm    : in std_logic_vector(31 downto 0);
		frm_seo_lui    : in std_logic_vector(31 downto 0);
		imux_ctrl_code : in std_logic;
		--outputs
		alu_mux_in2_o : out std_logic_vector(31 downto 0)
	);
end imm_mux;

architecture Behavioral of imm_mux is

begin
	alu_mux_in2_o <= frm_seo_lui when imux_ctrl_code = '1' else frm_seo_imm;

end Behavioral;
```

The shift left one file

This unit receives 32 bits from the sign extender and shifts it by one bit for onward forwarding to adder two.

```
----------------------------------------------------------------------------
--file: shiftLeftOne.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity shiftLeftOne is
	port(
		--inputs
		from_se_sh: in std_logic_vector (31 downto 0);
		--outputs
		to_adder:   out std_logic_vector (31 downto 0)
	);
```

```
end shiftLeftOne;

architecture Behavioral of shiftLeftOne is

begin
	to_adder <= from_se_sh (30 downto 0) & "0";

end Behavioral;
```

The alu source multiplexor file

This multiplexor makes it possible to replace the second input (in2) of the alu with an immediate value. It receives a control signal alusrc from the control path and an input port in the data path file must be reserved for it.

```
----------------------------------------------------------------------------
--file: alusrc_mux.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity alusrc_mux is
	port (
		--inputs
		rd_data2_i0	: in std_logic_vector(31 downto 0);
		from_se_imm	: in std_logic_vector(31 downto 0);
		alusrc_ctrl	: in std_logic;
		--outputs
		mux_in2_o	: out std_logic_vector(31 downto 0)
	);
end alusrc_mux;

architecture Behavioral of alusrc_mux is

begin
	mux_in2_o <= rd_data2_i0 when alusrc_ctrl = '0' else from_se_imm;

end Behavioral;
```

The alu file

ALU, the central execution unit of risc-v takes two inputs and executes one of twenty-five instructions sent by the alu-control unit. For branch instructions the alu generates three additional output signals, namely, gte, lt and zero. The main alu result or output opens an output port in the **data path** file for access into the data memory. The alu result or output may also be written directly into the destination

register via the write back multiplexor – wrb_mux. But this is a data path internal signal handshake.

```
--------------------------------------------------------------------------------
--file: ALU.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
--------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

use work.riscv_lib.all;

entity ALU is
	port (
		--inputs
		in1	: in std_logic_vector(31 downto 0);
		in2	: in std_logic_vector(31 downto 0);
		alu_ctrl : in instruction;
		--outputs
		output	: out std_logic_vector(31 downto 0);
		zero	: out std_logic; --sets when two input numbers are equal
		gte	: out std_logic; --sets when one input number is greater than or equal to the other
		lt	: out std_logic --sets when one input number is less than the other
	);
end ALU;

architecture Behavioral of ALU is
	--Declare internal signals
	signal temp_output : std_logic_vector (31 downto 0); --defined for intermediate case output
	signal temp_zero   : std_logic := '0'; --sets when intermediate case for two input numbers are equal
	signal temp_gte    : std_logic := '0'; --sets when   one input number is greater than or equal to another
	signal temp_lt     : std_logic := '0'; --sets when intermediate case for one input number is less than the
other

begin

	ALU_Output: process (in1, in2, alu_ctrl)
	begin
		case (alu_ctrl) is
			-- Load (upper) immediate operation
			-- U-type
			when INST_LUI =>
				temp_output <= std_logic_vector(unsigned(in2));
			-- Control operations
			-- B-type
			when INST_BEQ =>
				if (signed(in1) /= signed(in2)) then
					temp_zero <= '0';
```

```
                else
                        temp_zero <= '1';
                end if;
        when INST_BLT =>
                if (signed(in1) < signed(in2)) then
                        temp_lt <= '1';
                else
                        temp_lt <= '0';
                end if;
        when INST_BGE =>
                if (signed(in1) >= signed(in2)) then
                        temp_gte <= '1';
                else
                        temp_gte <= '0';
                end if;

        -- Memory operations
        when INST_LW => -- I-type
                temp_output <= std_logic_vector(signed(in1) + signed(in2));
        when INST_SW => -- S-type
                temp_output <= std_logic_vector(unsigned(in1) + unsigned(in2));

        -- Immediate operations
        -- I-type
        when INST_ADDI =>
                temp_output <= std_logic_vector(signed(in1) + signed(in2));
        when INST_SLTI =>
                if (signed(in1) < signed(in2)) then
                        temp_output <= (0 => '1', others => '0'); --x"00000001";
                else
                        temp_output <= (others => '0');  --x"00000000";
                end if;
        when INST_SLTIU =>
                if (unsigned(in1) < unsigned(in2)) then
                        temp_output <= (0 => '1', others => '0'); --x"00000001";
                else
                        temp_output <= (others => '0'); --x"00000000";
                end if;
        when INST_XORI =>
                temp_output <= std_logic_vector(unsigned(in1) xor unsigned(in2));
        when INST_ORI =>
                temp_output <= std_logic_vector(unsigned(in1) or unsigned(in2));
        when INST_ANDI =>
                temp_output <= std_logic_vector(unsigned(in1) and unsigned(in2));

        -- Shifts
        -- R-type
        when INST_SLLI =>
                temp_output <= std_logic_vector(shift_left(unsigned(in1),
to_integer(unsigned(in2(4 downto 0)))));
        when INST_SRLI =>
```

```
                                temp_output <= std_logic_vector(shift_right(unsigned(in1),
to_integer(unsigned(in2(4 downto 0)))));
                        when INST_SRAI =>
                                temp_output <= std_logic_vector(shift_right(signed(in1),
to_integer(unsigned(in2(4 downto 0)))));

                        -- Register-to-Register
                        -- R-type
                        when INST_ADD =>
                        temp_output <= std_logic_vector(signed(in1) + signed(in2));
                        when INST_SUB =>
                                temp_output <= std_logic_vector(unsigned(in1) - unsigned(in2));
                        when INST_SLL =>
                                temp_output <= std_logic_vector(shift_left(unsigned(in1),
to_integer(unsigned(in2(4 downto 0)))));
                        when INST_SLT =>
                                if (signed(in1) < signed(in2)) then
                                        temp_output <= (0 => '1', others => '0'); --x"00000001";
                                else
                                        temp_output <= (others => '0');  --x"00000000";
                                end if;
                        when INST_SLTU =>
                                if (unsigned(in1) < unsigned(in2)) then
                                        temp_output <= (0 => '1', others => '0'); --x"00000001";
                                else
                                        temp_output <= (others => '0'); --x"00000000";
                                end if;
                        when INST_XOR =>
                                temp_output <= std_logic_vector(unsigned(in1) xor unsigned(in2));
                        when INST_SRL =>
                                temp_output <= std_logic_vector(shift_right(unsigned(in1),
to_integer(unsigned(in2(4 downto 0)))));
                        when INST_SRA =>
                                temp_output <= std_logic_vector(shift_right(signed(in1),
to_integer(unsigned(in2(4 downto 0)))));
                        when INST_OR =>
                                temp_output <= std_logic_vector(unsigned(in1) or unsigned(in2));
                        when INST_AND =>
                                temp_output <= std_logic_vector(unsigned(in1) and unsigned(in2));
                        when others =>
                                temp_output <= (others => '0');
                end case;
        end process;

        --Connect internal signals to output signals
                output <= temp_output;
                zero   <= temp_zero;
                gte    <= temp_gte;
                lt     <= temp_lt;

end Behavioral;
```

The And gate for branching file

Three branch-instructions are implemented in the instruction set. This is made possible by comparing two values in the alu and combining the result with a branch signal from the control path at the AND gate. Thus, the branchGate receives three input signals from the alu and a branch input signal from the control path and issues one control signal to the program counter branch multiplexor - pcbr_mux. The branch signal from the control path also introduces an input port in the **data path** file.

```
----------------------------------------------------------------------------
--file: branchGate.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity branchGate is
	port(
		--inputs
		branch_i	:in std_logic;
		zero_i		:in std_logic;
		gt_i		:in std_logic;
		lt_i		:in std_logic;
		--outputs
		gate_o		:out std_logic
	);
end branchGate;

architecture Behavioral of branchGate is

	signal temp_out : std_logic;
begin
	process (branch_i, zero_i, gt_i, lt_i)
	begin
		if (branch_i = '1') then
			if (zero_i = '1') then
				temp_out <= '1';
			elsif (gt_i = '1') then
				temp_out <= '1';
			elsif (lt_i = '1') then
				temp_out <= '1';
			else
				temp_out <= '0';
			end if;
		else
			temp_out <= '0';
```

```
		end if;
	end process;

	gate_o <= temp_out;

end Behavioral;
```

The adder two file

This unit receives the current address from the program counter and adds it to the shifted immediate value and forwards the result to the pcbr_mux. All signals are internal to the data path.

```
----------------------------------------------------------------------------
--file: adderTwo.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity adderTwo is
	port(
		--inputs
		from_se		: in std_logic_vector (31 downto 0);
		from_pcplus4	: in std_logic_vector (31 downto 0);
		--outputs
		adder_o		: out std_logic_vector (31 downto 0)
	);
end adderTwo;

architecture Behavioral of adderTwo is

begin
	adder_o <=std_logic_vector(unsigned(from_se) + unsigned(from_pcplus4));

end Behavioral;
```

The branch multiplexor file

Services the program counter by replacing pc_next with pcplus4 or a branch address. Internal signal assignments are completed in the data path file.

```
----------------------------------------------------------------------------
--file: pcbranch_mux.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
----------------------------------------------------------------------------
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity pcbranch_mux is
	port (
		--inputs
		fr_pcplus4_i0	: in std_logic_vector(31 downto 0);
		fr_adder2_i1	: in std_logic_vector(31 downto 0);
		fr_n_gate_i	: in std_logic;
		--outputs
		pcbr_mux_o	: out std_logic_vector (31 downto 0)
	);
end pcbranch_mux;
architecture Behavioral of pcbranch_mux is

begin
		pcbr_mux_o <= fr_pcplus4_i0 when fr_n_gate_i = '0' else fr_adder2_i1;
end Behavioral;
```

The write back multiplexor file

This multiplexor receives a control signal – memtoreg, data read from memory and alu result. This will introduce two input ports in data path file.

```
--------------------------------------------------------------------------------
--file: wrback_mux.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity wrback_mux is
	port (
		--inputs
		from_alu_i0	: in std_logic_vector (31 downto 0);
		mem_rd_data_i1		: in std_logic_vector (31 downto 0);
		mem_to_reg_i	: in std_logic;
		--outputs
		mux_wr_data_o: out std_logic_vector (31 downto 0)
	);
end wrback_mux;

architecture Behavioral of wrback_mux is

begin
	mux_wr_data_o <= from_alu_i0 when mem_to_reg_i = '0' else mem_rd_data_i1;
end Behavioral;
```

CONNECTING DATA PATH UNITS

The Data Path File

In total eleven input ports and six output ports have been enlisted for the data path file. The reader may rely on the comments for the various internal signal handshakes.

```
----------------------------------------------------------------------------
--file: dataPath.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

use work.riscv_lib.all;

entity dataPath is
	port (
		--inputs
		clk_i		: in std_logic;
		rest_i		: in std_logic;

		memo_to_reg_i : in std_logic; --write back multiplexor control
		branching	: in std_logic; --to branch gate
		reg_dst_i	: in std_logic; --destination register multiplexor control
		alusrc_i	: in std_logic; --alu multiplexor control
		reg_wr_i	: in std_logic; --to write to register file
		immsrc_i	: in std_logic; --immediate multiplexor control

		alucontrol_i	: in instruction; --from alu control
		instru_i	: in std_logic_vector(31 downto 0); --from instruction memory
		dm_rd_data_i	: in std_logic_vector(31 downto 0); --data memory read data
		--outputs
		op_code			: out std_logic_vector (6 downto 0); --to control path
		funct_7		: out std_logic_vector (6 downto 0); --to alu control
		funct_3		: out std_logic_vector (2 downto 0); --to alu control
		prg_counter	: out std_logic_vector (31 downto 0); --to instruction memory
		alu_out		: out std_logic_vector (31 downto 0); --to data memory address
		read_data2	: out std_logic_vector (31 downto 0) --read register 2 data
	);
end dataPath;

architecture Behavioral of dataPath is
		signal sig_clk_i		: std_logic;
		signal sig_rset_i		: std_logic;
		signal sig_pc_next		: std_logic_vector(31 downto 0);
		signal sig_pc_current	: std_logic_vector(31 downto 0);
		------------------------------------------------------------programcounter
		signal sig_pcg_current	: std_logic_vector(31 downto 0);
```

```
signal sig_pcplus4_o  : std_logic_vector(31 downto 0);
-------------------------------------------------------------adder one
signal sig_instr_i          : std_logic_vector (31 downto 0);
signal sig_rs1_o            : std_logic_vector (4 downto 0);
signal sig_rs2_o            : std_logic_vector (4 downto 0);
signal sig_rd_o             : std_logic_vector (4 downto 0);
signal sig_imm_o            : std_logic_vector (11 downto 0);
signal sig_imm_sh           : std_logic_vector (11 downto 0);
signal sig_imm_lui          : std_logic_vector (19 downto 0);
signal sig_opcod_o          : std_logic_vector (6 downto 0);
signal sig_func7_o          : std_logic_vector (6 downto 0);
signal sig_func3_o          : std_logic_vector (2 downto 0);
-------------------------------------------------------------decoder
signal sig_rs2_i0           : ieee.numeric_std.unsigned(4 downto 0);
signal sig_dst_reg_i1       : ieee.numeric_std.unsigned(4 downto 0);
signal sig_rg_dst           : std_logic;
signal sig_wr_dst_o         : ieee.numeric_std.unsigned(4 downto 0);
-------------------------------------------------------------register multiplexor
signal sig_se_i             : std_logic_vector(11 downto 0);
signal sig_se_lui           : std_logic_vector (19 downto 0);
signal sig_se_sh            : std_logic_vector (11 downto 0);
signal sig_se_o             : std_logic_vector(31 downto 0);
signal sig_se_lui_o         : std_logic_vector (31 downto 0);
signal sig_se_sh_o          : std_logic_vector (31 downto 0);
-------------------------------------------------------------sign extender
signal sig_clck_i           : std_logic;
signal sig_rst_i            : std_logic;
signal sig_rd_addr1_i       : ieee.numeric_std.unsigned(4 downto 0);
signal sig_rd_addr2_i       : ieee.numeric_std.unsigned(4 downto 0);
signal sig_wri_data_i       : std_logic_vector(31 downto 0);
signal sig_wr_addr_i        : ieee.numeric_std.unsigned(4 downto 0);
signal sig_wr_en_i          : std_logic;
signal sig_reg_data1_o      : std_logic_vector(31 downto 0);
signal sig_reg_data2_o      : std_logic_vector(31 downto 0);
-------------------------------------------------------------register file
signal sig_rd_data2_i0      : std_logic_vector(31 downto 0);
signal sig_from_se_imm: std_logic_vector(31 downto 0);
signal sig_alusrc_ctrl      : std_logic;
signal sig_mux_in2_o        : std_logic_vector(31 downto 0);
-------------------------------------------------------------alu multiplexor
signal sig_in1              : std_logic_vector(31 downto 0);
signal sig_in2              : std_logic_vector(31 downto 0);
signal sig_alu_ctrl         : instruction;
signal sig_output           : std_logic_vector(31 downto 0);
signal sig_zero             : std_logic;
signal sig_gte              : std_logic;
signal sig_lt               : std_logic;
-------------------------------------------------------------alu
signal sig_from_alu_i0      : std_logic_vector(31 downto 0);
signal sig_mem_rd_data_i1: std_logic_vector(31 downto 0);
signal sig_mem_to_reg_i     : std_logic;
```

```
        signal sig_mux_wr_data_o : std_logic_vector(31 downto 0);
        ------------------------------------------------------------write back multiplexor
        signal sig_from_se_sh   : std_logic_vector(31 downto 0);
        signal sig_to_adder     : std_logic_vector(31 downto 0);
        ------------------------------------------------------------shift left one
        signal sig_from_sh : std_logic_vector (31 downto 0);
        signal sig_from_pc : std_logic_vector (31 downto 0);
        signal sig_adder_o  : std_logic_vector (31 downto 0);
        ------------------------------------------------------------adder two
        signal sig_branch_i : std_logic;
        signal sig_zero_i   : std_logic;
        signal sig_gt_i     : std_logic;
        signal sig_lt_i     : std_logic;
        signal sig_gate_o   : std_logic;
        ------------------------------------------------------------and gate
        signal sig_fr_pcplus4_i0: std_logic_vector(31 downto 0);
        signal sig_fr_adder2_i1 : std_logic_vector(31 downto 0);
        signal sig_fr_n_gate_i  : std_logic;
        signal sig_pcbr_mux_o  : std_logic_vector(31 downto 0);
        ------------------------------------------------------------branch multiplexor
        signal sig_frm_seo_imm   : std_logic_vector(31 downto 0);
        signal sig_frm_seo_lui   : std_logic_vector(31 downto 0);
        signal sig_imux_ctrl_code  : std_logic;
        signal sig_alu_mux_in2_o  : std_logic_vector(31 downto 0);
        ------------------------------------------------------------sign extend multiplexor

begin
    PC_ADD: entity work.programcounter (Behavioral)
    port map (
        clk_i     => sig_clk_i,
        rset_i    => sig_rset_i,
        pc_next   => sig_pc_next,
        pc_current => sig_pc_current
    );
    ADDER1_ADD: entity work.adderOne (Behavioral)
    port map (
        pc_current => sig_pcg_current,
        pcplus4_o => sig_pcplus4_o
    );
    DCode_ADD: entity work.decoder (Behavioral)
    port map (
        instr_i  => sig_instr_i,
        rs1_o    => sig_rs1_o,
        rs2_o    => sig_rs2_o,
        rd_o     => sig_rd_o,
        imm_o  => sig_imm_o,
        imm_sh  => sig_imm_sh,
        imm_lui => sig_Imm_lui,
        opcod_o         => sig_opcod_o,
        func7_o         => sig_func7_o,
        func3_o         => sig_func3_o
```

```
);
REG_MUX_ADD: entity work.reg_mux (Behavioral)
port map (
        rs2_i0          => sig_rs2_i0,
        dst_reg_i1      => sig_dst_reg_i1,
        rg_dst          => sig_rg_dst,
        wr_dst_o        => sig_wr_dst_o
);
SE_ADD: entity work.signextend (Behavioral)
port map (
        se_i      => sig_se_i,
        se_lui    => sig_se_lui,
        se_sh     => sig_se_sh,
        se_o      => sig_se_o,
        se_lui_o => sig_se_lui_o,
        se_sh_o  => sig_se_sh_o
);
REG_FILE_ADD: entity work.regfile (Behavioral)
port map (
        clk_i => sig_clck_i,
        rst_i => sig_rst_i,
        rd_addr1_i => sig_rd_addr1_i,
        rd_addr2_i => sig_rd_addr2_i,
        wr_data_i  => sig_wri_data_i,
        wr_addr_i => sig_wr_addr_i,
        wr_en_i   => sig_wr_en_i,
        reg_data1_o => sig_reg_data1_o,
        reg_data2_o => sig_reg_data2_o
);
ALU_MUX_ADD: entity work.alusrc_mux (Behavioral)
port map (
        rd_data2_i0     => sig_rd_data2_i0,
        from_se_imm     => sig_from_se_imm,
        alusrc_ctrl     => sig_alusrc_ctrl,
        mux_in2_o       => sig_mux_in2_o
);
ALU_ADD: entity work.ALU (Behavioral)
port map (
        in1 => sig_in1,
        in2 => sig_in2,
        alu_ctrl => sig_alu_ctrl,
        output => sig_output,
        zero   => sig_zero,
        gte    => sig_gte,
        lt     => sig_lt
);
WRBACK_MUX_ADD: entity work.wrback_mux (Behavioral)
port map (
        from_alu_i0     => sig_from_alu_i0,
        mem_rd_data_i1=> sig_mem_rd_data_i1,
        mem_to_reg_i    => sig_mem_to_reg_i,
```

```
        mux_wr_data_o=> sig_mux_wr_data_o
);
SHFLFT_ADD: entity work.shiftLeftOne (Behavioral)
port map (
        from_se_sh  => sig_from_se_sh,
        to_adder => sig_to_adder
);
ADDER2_ADD: entity work.adder (Behavioral)
port map (
        from_se => sig_from_sh,
        from_pc => sig_from_pc,
        adder_o => sig_adder_o
);
BRNCH_GATE_ADD: entity work.branchGate (Behavioral)
port map (
        branch_i => sig_branch_i,
        zero_i   => sig_zero_i,
        gt_i     => sig_gt_i,
        lt_i     => sig_lt_i,
        gate_o   => sig_gate_o
);
BRNCH_MUX_ADD: entity work.pcbranch_mux (Behavioral)
port map (
        fr_pcplus4_i0    => sig_fr_pcplus4_i0,
        fr_adder2_i1     => sig_fr_adder2_i1,
        fr_n_gate_i      => sig_fr_n_gate_i,
        pcbr_mux_o       => sig_pcbr_mux_o
);
SE_MUX_ADD: entity work.imm_mux (Behavioral)
port map (
        frm_seo_imm    => sig_frm_seo_imm,
        frm_seo_lui      => sig_frm_seo_lui,
        imux_ctrl_code => sig_imux_ctrl_code,
        alu_mux_in2_o => sig_alu_mux_in2_o
);

--connect input to signals
sig_instr_i <= instru_i; --from instruction memory
sig_rg_dst <= reg_dst_i; --register multiplexor control comes in
sig_se_i <= sig_imm_o;  --sign extender receives immediate address

sig_wr_addr_i   <= sig_wr_dst_o; --set register destination address from reg mux
sig_rd_addr1_i  <= ieee.numeric_std.unsigned(sig_rs1_o);    --set source register
sig_rd_addr2_i  <= ieee.numeric_std.unsigned(sig_rs2_o);    --set transfer register

sig_in1 <= sig_reg_data1_o;          --alu receives source data (rs1)
sig_rd_data2_i0 <= sig_reg_data2_o; --alu source multiplexor receives transfer data (rs2)
sig_alusrc_ctrl <= alusrc_i;          --alu source multiplexor receives control signal
sig_in2 <= sig_mux_in2_o;          --alu receives source multiplexor output

sig_se_sh <= sig_imm_sh; --Connect immediate signals to sign extender
```

```
sig_se_lui <= sig_Imm_lui;
sig_from_alu_i0 <= sig_output;      --write back multiplexor receives alu output
sig_mem_rd_data_i1 <= dm_rd_data_i;  --write back multiplexor receives memory read data
sig_mem_to_reg_i <= memo_to_reg_i;   --write back multiplexor receives control signal
sig_wri_data_i <= sig_mux_wr_data_o ; -- register file receives write back multiplexor output

sig_branch_i <= branching; --and gate receives branch signal from control unit
sig_zero_i  <= sig_zero;   --and gate receives zero signal from alu
sig_gt_i    <= sig_gte;       --and gate receives "greater than" signal from alu
sig_lt_i    <= sig_lt;     --and gate receives "less than" signal from alu

sig_from_se_sh <= sig_se_sh_o ; --shift left one receives sign extended immediate value
sig_pcg_current <= sig_pc_current; --adder one receives current address and generates "pc+4"

--Adder2
sig_from_sh <= sig_to_adder; --adder two receives output of shift left one
sig_from_pc <= sig_pc_current; --adder two receives current address [mips uses pc + 4]

--pcbranch_mux
sig_fr_pcplus4_i0 <= sig_pcplus4_o;   --pc branch multiplexor receives "pc+4" from adder one,
sig_fr_adder2_i1 <= sig_adder_o; --pc branch multiplexor receives output of adder two
sig_fr_n_gate_i  <= sig_gate_o;  --pc branch multiplexor receives control signal from and gate
sig_pc_next <= sig_pcbr_mux_o; --branch mux output goes into program counter

--Connect signals to output
prg_counter <= sig_pc_current; --address to instruction memory
read_data2  <= sig_reg_data2_o; --register data to store in data memory
alu_out <= sig_output; --executed result
op_code <= sig_opcod_o; --opcode from decoder to the control unit
funct_7 <= sig_func7_o; --to alu control
funct_3 <= sig_func3_o;            --to alu control

--Connect signals to global clock
sig_clk_i  <= clk_i;
sig_rset_i <= rest_i;
sig_clck_i <= clk_i;
sig_rst_i  <= rest_i;
sig_alu_ctrl <= alucontrol_i;   --alu receives instruction from alu control
sig_wr_en_i <= reg_wr_i;          --register file receives write signal from control unit

--Connect decoder to register multiplexor
sig_rs2_i0 <= ieee.numeric_std.unsigned(sig_rs2_o);      --receives output of register2 from decoder
sig_dst_reg_i1 <= ieee.numeric_std.unsigned(sig_rd_o);  --receives destination register from decoder

--Sign extender multiplexor
sig_frm_seo_imm    <= sig_se_o;          --receives immediate value from sign extender into alu src
sig_frm_seo_lui    <= sig_se_lui_o;        --receives LUI immediate value from sign extender into alu src
sig_imux_ctrl_code <= immsrc_i;                --receives immediate source from control unit
sig_from_se_imm    <= sig_alu_mux_in2_o;--alu src recieves one immediate value from sign extender

end Behavioral;
```

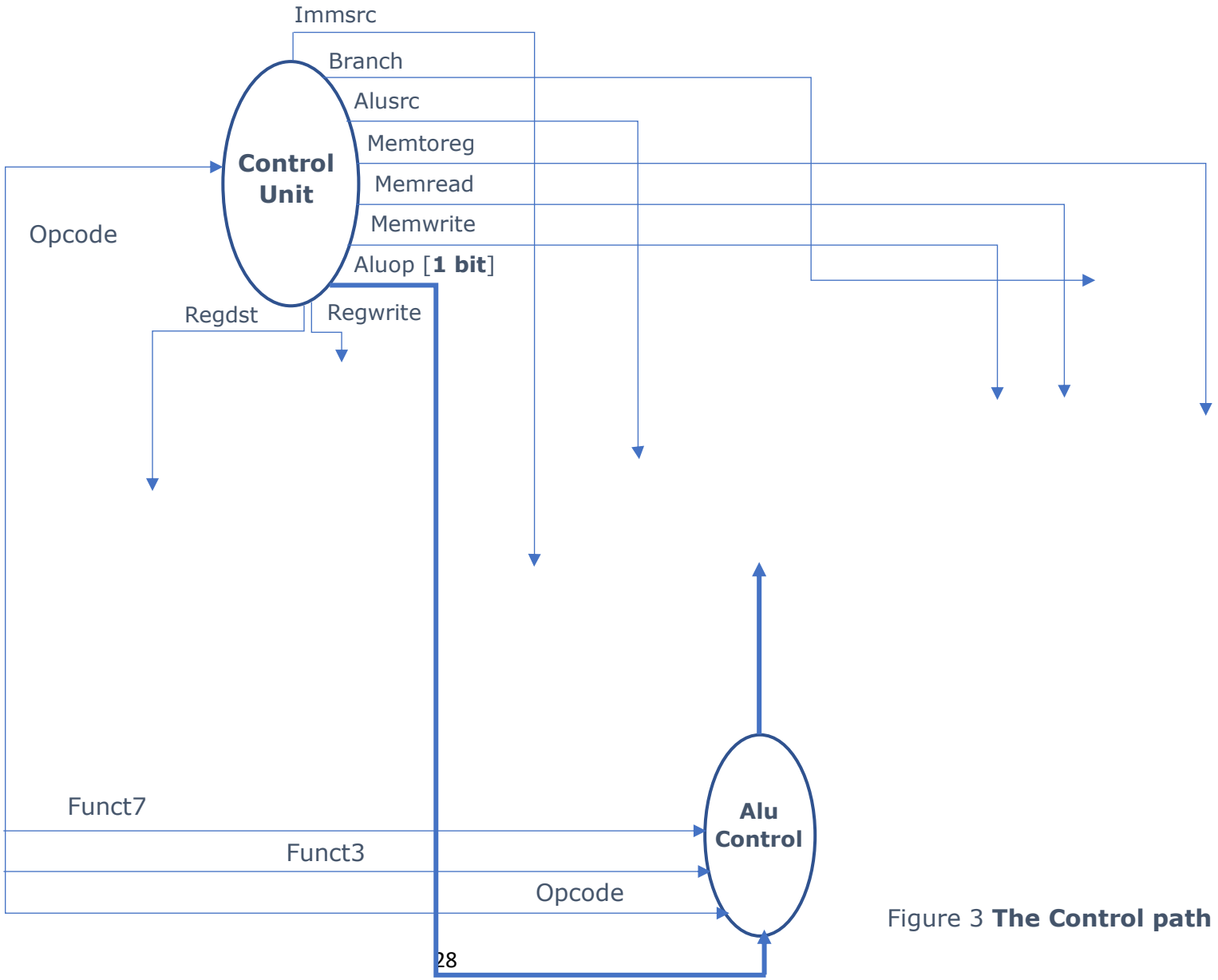

Figure 3 **The Control path**

The Control Unit File

The control unit receives the opcode from the decoder via the data path and returns six signals (regdst, regwrite, memtoreg, alusrc, branch, immsrc) to the data path, sends one signal (alusrc) to the alu-control and two signals (memwrite, memread) to the data memory. The opcode introduces *one input port* into the **control path**, while the six signals introduce *six output ports*. The two signals to data memory also introduce *two output ports* in the control path. The single signal to the alu-control remains an internal signal to the control path. In any case, a signal line may be set to an active ('1'), inactive ('0') or a don't care state ('X') depending on the type of instruction identified by the opcode bits.

```
----------------------------------------------------------------------------
--file: controlUnit.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity controlUnit is
	port (
		--inputs
		opcode : in  std_logic_vector(6 downto 0); --instruction(6 downto 0)
		--outputs
		regdst   : out std_logic;
		immsrc : out std_logic;
		branch  : out std_logic;
		memread : out std_logic;
		memtoreg: out std_logic;
		aluop    : out std_logic;
		memwrite: out std_logic;
		alusrc   : out std_logic;
		regwrite: out std_logic
	);
end controlUnit;

architecture Behavioral of controlUnit is
		signal s_regdst    : std_logic;
		signal s_branch   : std_logic;
		signal s_memread  : std_logic;
		signal s_memtoreg : std_logic;
		signal s_aluop     : std_logic;
		signal s_memwrite : std_logic;
		signal s_alusrc    : std_logic;
		signal s_regwrite : std_logic;
		signal s_immsrc  : std_logic;
begin
```

```
process (opcode)
begin

case opcode is
	when "0110011" => --and, or, add, sub, slt : 0x00
		s_regdst	<= '1';
		s_immsrc	<= '0';
		s_branch	<= '0';
		s_memread	<= '0';
		s_memtoreg	<= '0';
		s_aluop		<= '1';
		s_memwrite	<= '0';
		s_alusrc	<= '0';
		s_regwrite	<= '1';
	when "0010011" => --addi
		s_regdst	<= '0';
		s_immsrc	<= '0';
		s_branch	<= '0';
		s_memread	<= '0';
		s_memtoreg	<= '0';
		s_aluop		<= '1';
		s_memwrite	<= '0';
		s_alusrc	<= '1';
		s_regwrite	<= '1';
	when "0000011" => --load word (lw)
		s_regdst	<= '1';
		s_immsrc	<= '0';
		s_branch	<= '0';
		s_memread	<= '1';
		s_memtoreg	<= '1';
		s_aluop		<= '1';
		s_memwrite	<= '0';
		s_alusrc	<= '1';
		s_regwrite	<= '1';
	when "0100011" => --store word (sw) : 0x2b
		s_regdst	<= 'X'; --don't care
		s_immsrc	<= '0';
		s_branch	<= '0';
		s_memread	<= '0';
		s_memtoreg	<= 'X'; --don't care
		s_aluop		<= '1';
		s_memwrite	<= '1';
		s_alusrc	<= '1';
		s_regwrite	<= '0';
	when "1100011" => --branch equal (beq)
		s_regdst	<= 'X'; --don't care
		s_immsrc	<= '0';
		s_branch	<= '1';
		s_memread	<= '0';
		s_memtoreg	<= 'X'; --don't care
		s_aluop		<= '1';
```

```
                s_memwrite      <= '0';
                s_alusrc        <= '0';
                s_regwrite      <= '0';
            when "0110111" => --load upper immediate
                s_regdst        <= '1';
                s_immsrc        <= '1';
                s_branch        <= '0';
                s_memread       <= '0';
                s_memtoreg      <= '0';
                s_aluop         <= '1';
                s_memwrite      <= '0';
                s_alusrc        <= '1';
                s_regwrite      <= '1';
            when others => null; --implement other instructions down here
                s_regdst        <= '0';
                s_immsrc        <= '0';
                s_branch        <= '0';
                s_memread       <= '0';
                s_memtoreg      <= '0';
                s_aluop         <= '0';
                s_memwrite      <= '0';
                s_alusrc        <= '0';
                s_regwrite      <= '0';
        end case;

        end process;
        --Connect internal signals to output signals
        regdst     <= s_regdst;
        branch     <= s_branch;
        memread    <= s_memread;
        memtoreg   <= s_memtoreg;
        aluop      <= s_aluop;
        memwrite   <= s_memwrite;
        alusrc     <= s_alusrc;
        immsrc     <= s_immsrc;
        regwrite   <= s_regwrite;
end Behavioral;
```

The Alu Control File

The aluop is a dedicated one-bit line that is always set to ‘1’. By admitting the opcode, the funct3 and funct7 signals from the decoder via the data path, the alu-control is able to specify which of the twenty-five instructions the alu executes at any particular time. These instructions are stored in the **riscv_lib.vhd** file and can be linked to, in vhdl with *use work.riscv_lib.all;* Since the opcode has been introduced to the control path via the control unit, we only need to introduce two additional input ports in the **control path** file for the funct7 and funct3 signal lines. The output of the alu control enters the alu via the data path. This signal will exit

the control path through an output port as well. Hence the alu control has four input ports and one output port.

```
--------------------------------------------------------------------------------
--file: aluControl.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

use work.riscv_lib.all;

entity aluControl is
	port (
		--inputs
		aluop    : in std_logic; --from control unit
		op_code: in std_logic_vector(6 downto 0); --from decoder
		funct7   : in std_logic_vector(6 downto 0); --from decoder
		funct3   : in std_logic_vector(2 downto 0); --from decoder
		--ouputs
		alucontrol : out instruction
	);
end aluControl;

architecture Behavioral of aluControl is

	signal sig_aluctrl : instruction;

begin
		process (op_code, funct7, funct3, aluop)
		begin

		if (aluop = '1') then

			case op_code is
				when "0110011" => --and, or, add, sub, slt
					if (funct7 = "0000000") then
						if (funct3 = "000") then
							sig_aluctrl <= INST_ADD;
						elsif (funct3 = "001") then
							sig_aluctrl <= INST_SLL;
						elsif (funct3 = "010") then
							sig_aluctrl <= INST_SLT;
						elsif (funct3 = "011") then
							sig_aluctrl <= INST_SLTU;
						elsif (funct3 = "100") then
							sig_aluctrl <= INST_XOR;
						elsif (funct3 = "101") then
```

```
                                        sig_aluctrl <= INST_SRL;
                                elsif (funct3 = "110") then
                                        sig_aluctrl <= INST_OR;
                                elsif (funct3 = "111") then
                                        sig_aluctrl <= INST_AND;
                                end if;
                        elsif (funct7 = "0100000") then
                                if (funct3 = "000") then
                                        sig_aluctrl <= INST_SUB;
                                elsif(funct3 = "101") then
                                        sig_aluctrl <= INST_SRA;
                                end if;
                        end if;
--                      ----------------------------------------------------------------
                when "0010011" => --addi
                        if (funct7 = "0000000") then
                                if(funct3 = "001") then
                                        sig_aluctrl <= INST_SLLI;
                                elsif(funct3 = "101") then
                                        sig_aluctrl <= INST_SRLI;
                                end if;
                        elsif(funct7 = "0100000") then
                                if (funct3 = "101") then
                                        sig_aluctrl <= INST_SRAI;
                                end if;
                        else
                                if (funct3 = "000") then
                                        sig_aluctrl <= INST_ADDI;
                                elsif (funct3 = "010") then
                                        sig_aluctrl <= INST_SLTI;
                                elsif (funct3 = "011") then
                                        sig_aluctrl <= INST_SLTIU;
                                elsif (funct3 = "100") then
                                        sig_aluctrl <= INST_XORI;
                                elsif (funct3 = "110") then
                                        sig_aluctrl <= INST_ORI;
                                elsif (funct3 = "111") then
                                        sig_aluctrl <= INST_ANDI;
                                end if;

                        end if;
                ----------------------------------------------------------------
                when "0000011" => --load word (lw)
                                if (funct3 = "010") then
                                        sig_aluctrl <= INST_LW;
                                end if;
                ----------------------------------------------------------------
                when "0100011" => --store word (sw)
                                if(funct3 = "010") then
                                        sig_aluctrl <= INST_SW;
                                end if;
```

```
		-----------------------------------------------------------------
			when "1100011" => --branch equal (beq)
					if(funct3 = "000") then
						sig_aluctrl <= INST_BEQ;
					elsif(funct3 = "100") then
						sig_aluctrl <= INST_BLT;
					elsif(funct3 = "101") then
						sig_aluctrl <= INST_BGE;
					end if;
		-----------------------------------------------------------------
			when "0110111" => --LUI
				sig_aluctrl <= INST_LUI;
		-----------------------------------------------------------------
			when others =>
				sig_aluctrl <= INST_NO_OP;
		end case;
		-----------------------------------------------------------------
	end if;

	end process;

	alucontrol <= sig_aluctrl;

end Behavioral;
```

CONNECTING CONTROL PATH UNITS

THE CONTROL PATH FILE

In all three input ports and nine output ports were identified for the control path file. The reader may rely on the lines of comments for the various internal signal line connections.

```
--------------------------------------------------------------------------------
--file: controlPath.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

use work.riscv_lib.all;

entity controlPath is
	port (
		--inputs
		opcode_i	: in std_logic_vector (6 downto 0); --receives opcode from decoder
		funct7_i	: in std_logic_vector (6 downto 0);
		funct3_i	: in std_logic_vector (2 downto 0);
		--outputs
		regdst_o	: out std_logic;
		immsrc_o	: out std_logic;
		branch_o	: out std_logic;
		memread_o	: out std_logic;
		memtoreg_o	: out std_logic;
		memwrite_o	: out std_logic;
		alusrc_o	: out std_logic;
		regwrite_o	: out std_logic;
		alucontrol_o	: out instruction
	);
end controlPath;

architecture Behavioral of controlPath is

		signal sig_opcode	: std_logic_vector (6 downto 0);
		signal sig_regdst	: std_logic;
		signal sig_immsrc	: std_logic;
		signal sig_branch	: std_logic;
		signal sig_memread	: std_logic;
		signal sig_memtoreg	: std_logic;
		signal sig_aluop	: std_logic;
		signal sig_memwrite	: std_logic;
		signal sig_alusrc	: std_logic;
		signal sig_regwrite	: std_logic;
		signal sig_aluopr	: std_logic; ---alu control signals begin here
```

```
        signal sig_op_code      : std_logic_vector(6 downto 0);
        signal sig_funct7       : std_logic_vector(6 downto 0);
        signal sig_funct3       : std_logic_vector(2 downto 0);
        signal sig_alucontrol   : instruction;

begin
    CU_ADD: entity work.controlUnit (Behavioral)
    port map(
        opcode   => sig_opcode,
        regdst   => sig_regdst,
        immsrc   => sig_immsrc,
        branch   => sig_branch,
        memread  => sig_memread,
        memtoreg => sig_memtoreg,
        aluop    => sig_aluop,
        memwrite => sig_memwrite,
        alusrc   => sig_alusrc,
        regwrite => sig_regwrite
    );
    ALU_CTRL_ADD: entity work.aluControl (Behavioral)
    port map(
        aluop      => sig_aluopr,
        op_code    => sig_op_code,
        funct7     => sig_funct7,
        funct3     => sig_funct3,
        alucontrol => sig_alucontrol
    );

    --connect internal signals to input
    sig_opcode <= opcode_i; --picks the opcode from decoder
    sig_op_code <= opcode_i;--opcode goes into alu control
    sig_funct7 <= funct7_i; --funct7 from decoder into alu control
    sig_funct3 <= funct3_i; --funct3 from decoder into alu control
    sig_aluopr <= sig_aluop; --receives aluop from the control unit

    --connect internal signals to output
    regdst_o     <= sig_regdst;
    branch_o     <= sig_branch;
    memread_o    <= sig_memread;
    memtoreg_o   <= sig_memtoreg;
    memwrite_o   <= sig_memwrite;
    alusrc_o     <= sig_alusrc;
    regwrite_o   <= sig_regwrite;
    immsrc_o     <= sig_immsrc;
    alucontrol_o <= sig_alucontrol;


end Behavioral;
```

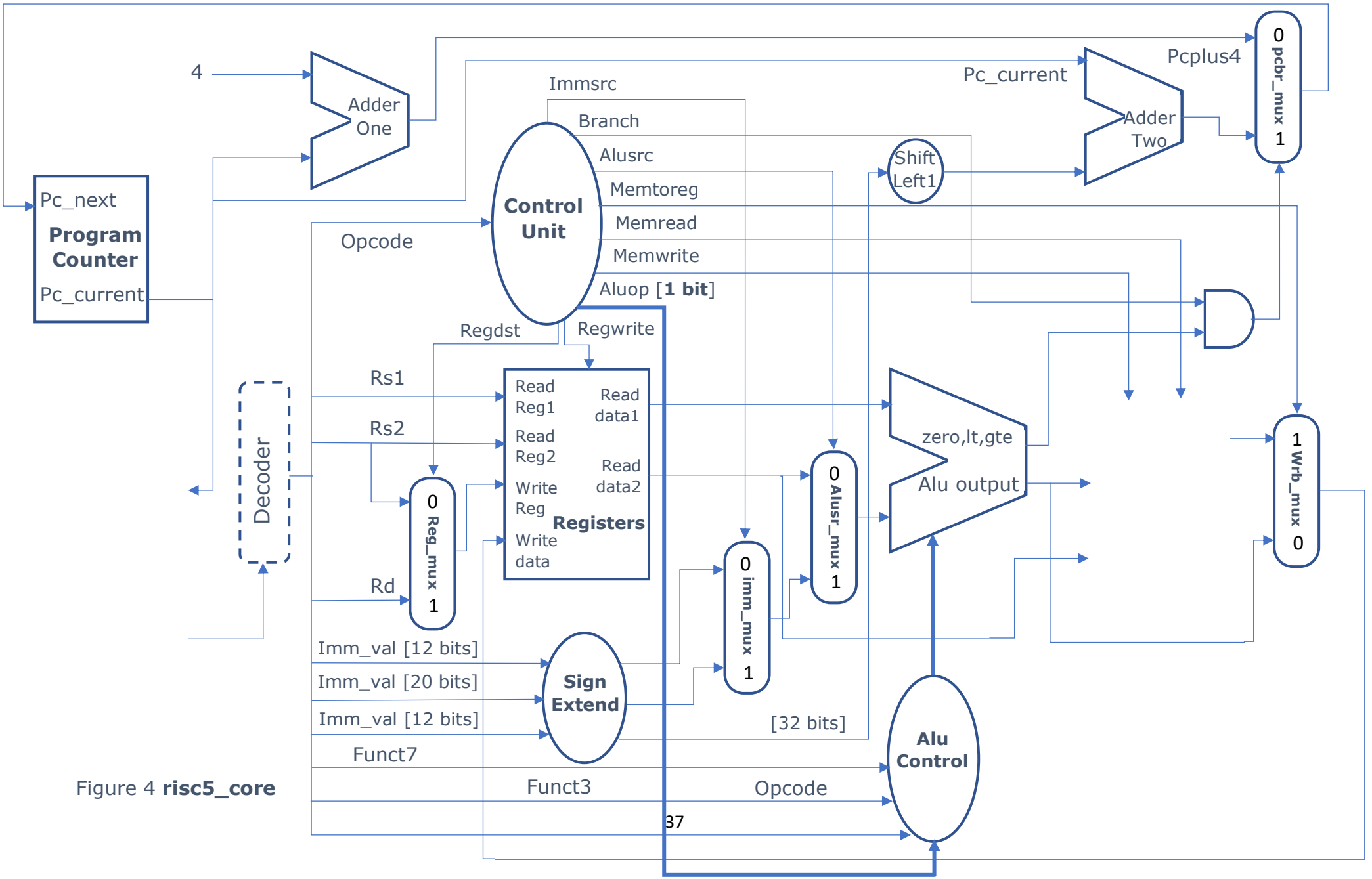

Figure 4 **risc5_core**

CONNECTING CONTROL PATH TO DATA PATH

THE RISCV CORE FILE

In this file the **data** and **control** paths are made to shake hands with each other while providing input ports to admit data from the instruction and data memories. Output ports for pc_current to reach the instruction memory, and memwrite and memread signals to reach the data memory are also provided. Two additional output ports are needed for the alu result or output and the register 2 read data to reach data memory. More comments are added to enhance clarity.

```
--------------------------------------------------------------------------------
--file: risc5_core.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

use work.riscv_lib.all;

entity risc5_core is
	port (
		--inputs
		clk_i		: in std_logic;
		reset_i		: in std_logic;
		instr_i		: in std_logic_vector (31 downto 0); --receives from instruction mememory
		rd_data_i	: in std_logic_vector (31 downto 0); --read from data memory
		--outputs
		prcount_o	: out std_logic_vector (31 downto 0); --into instruction memory
		mem_wri_o	: out std_logic;
		memo_rd_o	: out std_logic;
		alu_out_o	: out std_logic_vector (31 downto 0);
		rd_data2_o	: out std_logic_vector (31 downto 0) --read from register file read data 2
	);
end risc5_core;

architecture Behavioral of risc5_core is

	--Control path signal assignments
		signal sg_funct7_i	: std_logic_vector(6 downto 0);
		signal sg_funct3_i	: std_logic_vector(2 downto 0);
		signal sg_regdst_o	: std_logic;
		signal sg_immsrc_o	: std_logic;
		signal sg_branch_o	: std_logic;
		signal sg_memread_o	: std_logic;
		signal sg_memtoreg_o	: std_logic;
		signal sg_memwrite_o	:  std_logic;
		signal sg_alusrc_o	: std_logic;
		signal sg_regwrite_o	: std_logic;
```

```
        signal sg_alucontrol_o   : instruction;
        signal sg_opcode_i       : std_logic_vector (6 downto 0);
    --Data path signal assignments
        signal sg_clk_i          : std_logic;
        signal sg_rest_i : std_logic;
        signal sg_memo_to_reg_i      : std_logic;
        signal sg_reg_wr_i       : std_logic;
        signal sg_branching      : std_logic;
        signal sg_reg_dst_i      : std_logic;
        signal sg_alusrc_i       : std_logic;
        signal sg_immsrc_i       : std_logic;
        signal sg_alucontrol_i   : instruction;
        signal sg_instru_i       : std_logic_vector(31 downto 0);
        signal sg_dm_rd_data_i : std_logic_vector(31 downto 0);
        signal sg_prg_counter    : std_logic_vector (31 downto 0);
        signal sg_alu_out        : std_logic_vector (31 downto 0);
        signal sg_read_data2     : std_logic_vector (31 downto 0);
        signal sg_op_code        : std_logic_vector (6 downto 0);
        signal sg_funct_7        : std_logic_vector (6 downto 0); --to alu control
        signal sg_funct_3        : std_logic_vector (2 downto 0); --to alu control

begin
    Ctrl_Path_add: entity work.controlPath (Behavioral)
    port map (
        opcode_i     => sg_opcode_i,
        funct7_i     => sg_funct7_i,
        funct3_i     => sg_funct3_i,
        regdst_o     => sg_regdst_o,
        immsrc_o     => sg_immsrc_o,
        branch_o     => sg_branch_o,
        memread_o    => sg_memread_o,
        memtoreg_o   => sg_memtoreg_o,
        memwrite_o   => sg_memwrite_o,
        alusrc_o     => sg_alusrc_o,
        regwrite_o   => sg_regwrite_o,
        alucontrol_o => sg_alucontrol_o
    );
    Data_Path_add: entity work.dataPath (Behavioral)
    port map (
        clk_i          => sg_clk_i,
        rest_i         => sg_rest_i,
        memo_to_reg_i => sg_memo_to_reg_i,
        branching      => sg_branching,
        reg_dst_i      => sg_reg_dst_i,
        alusrc_i => sg_alusrc_i,
        reg_wr_i       => sg_reg_wr_i,
        immsrc_i       => sg_immsrc_i,
        alucontrol_i   => sg_alucontrol_i,
        instru_i => sg_instru_i,
        dm_rd_data_i   => sg_dm_rd_data_i,
        prg_counter    => sg_prg_counter,
```

```
            alu_out         => sg_alu_out,
            read_data2      => sg_read_data2,
            op_code                 => sg_op_code,
            funct_7         => sg_funct_7,
            funct_3         => sg_funct_3
    );
            --Connect input signals
            sg_clk_i <= clk_i;
            sg_rest_i <= reset_i;
            sg_funct7_i <= sg_funct_7; --receives funct7 from decoder into the control path
            sg_funct3_i <=  sg_funct_3; --receives funct3 from decoder into the control path
            sg_dm_rd_data_i  <= rd_data_i; --receives data read from data memory

            --Connect internal signals
            sg_reg_dst_i <= sg_regdst_o; --register multiplexor control
            sg_branching <= sg_branch_o; --and gate branch signal
            sg_memo_to_reg_i <=sg_memtoreg_o; --write back multiplexor control
            sg_alusrc_i <= sg_alusrc_o; --alu multiplexor control
            sg_reg_wr_i <= sg_regwrite_o; --register file write signal
            sg_immsrc_i <= sg_immsrc_o; --immediate multiplexor control signal
            sg_alucontrol_i <= sg_alucontrol_o; --alu control signal

            --Connect output signals
            mem_wri_o  <= sg_memwrite_o; --data memory write signal
            memo_rd_o  <= sg_memread_o; --data memory read signal
            prcount_o  <= sg_prg_counter;
            alu_out_o  <= sg_alu_out;
            rd_data2_o <= sg_read_data2;
            sg_opcode_i<= sg_op_code; --from decoder to control unit
            sg_instru_i<=instr_i;

end Behavioral;
```

TESTING THE RISC-V32I CORE THROUGH SIMULATION

THE TEST BENCH FILE

For the purpose of simulation an inst.mem file is provided with sample instructions. Same file has been renamed as data.mem to satisfy data memory requirements. However, the path to these two files must be specified explicitly in the test bench such as " C:\Users\LENOVO\Desktop\RISCV32I_Compsineer \"; If you have come this far, go ahead and ENJOY your success. Thank you for your patience.

```
----------------------------------------------------------------------------
--file: tb_risc5_core.vhd
--date: 2/2020
--author: Kwame Owusu Ampadu
--email: KwameOwusu.Ampadu@b-tu.de
----------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_textio.all;
use ieee.numeric_std.all;
use STD.textio.all;

use work.riscv_lib.all;

entity tb_risc5_core is
end entity tb_risc5_core;

architecture Behavioral of tb_risc5_core is

	-- set the path to your memory file directory here!
	constant MEM_DIR       : string := " C:\Users\LENOVO\Desktop\RISCV32I_Compsineer \";
	-- instruction memory file is specified here!
	constant INST_MEM_FILE : string := "inst.mem"; --sample instructions kept in the folder
	-- data memory file is specified here!
	constant DATA_MEM_FILE : string := "data.mem"; --sample instructions kept in the folder

	constant PERIOD         : time := 100 ns;
	constant RESET_DELAY    : time := 500 ns;
	constant SIMULATION_TIME : time := 90000 ns;

	-- We assume small memories
	type t_IMEMORY is array (4096-1 downto 0) of std_logic_vector(31 downto 0);
	type t_DMEMORY is array (8192-1 downto 0) of std_logic_vector(31 downto 0);

	signal inst_memory   : t_IMEMORY;
	signal data_memory   : t_DMEMORY;

	signal tb_clk_i       : std_logic;
	signal tb_reset_i     : std_logic;
	signal tb_instr_i     : std_logic_vector(31 downto 0); --receives from instruction mememory
	signal tb_rd_data_i   : std_logic_vector(31 downto 0); --read from data memory
	signal tb_prcount_o   : std_logic_vector(31 downto 0);
	signal tb_mem_wri_o   : std_logic;
	signal tb_memo_rd_o   : std_logic;
	signal tb_alu_out_o   : std_logic_vector(31 downto 0);
	signal tb_rd_data2_o : std_logic_vector (31 downto 0); --read from register file read data 2

begin

	-- instantiate your top level here!
	risc5_core : entity work.risc5_core (Behavioral)
	port map (
		clk_i        => tb_clk_i,
		reset_i     => tb_reset_i,
		instr_i     => tb_instr_i,
		rd_data_i   => tb_rd_data_i,
		prcount_o   => tb_prcount_o,
		mem_wri_o   => tb_mem_wri_o,
		memo_rd_o   => tb_memo_rd_o,
```

```
        alu_out_o  => tb_alu_out_o,
        rd_data2_o => tb_rd_data2_o
);

-- generate clock signal
clk_process : process
begin
        loop
                tb_clk_i <= '0';
                wait for PERIOD / 2;
                tb_clk_i <= '1';
                wait for PERIOD / 2;

                assert now < SIMULATION_TIME
                report "End of Simulation!"
                severity failure;          -- throw failure to break simulation
        end loop;
end process;

tb_reset_i <= '1', '0' after RESET_DELAY;

-- simulates data memory
data_mem : process(tb_reset_i, tb_clk_i, tb_alu_out_o, tb_mem_wri_o, data_memory)
        file dmem_init_file : text open read_mode is MEM_DIR & DATA_MEM_FILE;

        variable line_buf   : line;      -- Line buffers
        variable str_buf    : string(1 to 10); -- string to modify
        variable value_buf  : std_logic_vector(31 downto 0);
        variable address    : integer := 0;
begin
        if tb_reset_i = '1' then

                -- Initialize the instruction memory
                address := 0;
                loop
                        if endfile(dmem_init_file) then
                                exit;
                        else
                                readline(dmem_init_file, line_buf);
                                read(line_buf, str_buf);
                                str_buf := str_buf(str_buf'left + 2 to str_buf'right) &  "  ";
                                write(line_buf, str_buf);

                                hread(line_buf, value_buf);

                                data_memory(address) <= value_buf;
                                address              := address + 1;
                        end if;
                end loop;
        else
                -- assynchronous read
```

```
                tb_rd_data_i <= data_memory(to_integer(shift_right(unsigned(tb_alu_out_o), 2)));

                -- synchronous write
                if rising_edge(tb_clk_i) and tb_mem_wri_o = '1' then
                        data_memory(to_integer(unsigned(tb_alu_out_o))) <= tb_rd_data2_o;
                end if;
        end if;
end process;

-- simulates program memory
inst_mem: process (tb_reset_i, tb_prcount_o, inst_memory)
        file imem_init_file : text open read_mode is MEM_DIR & INST_MEM_FILE;

        variable line_buf   : line;      -- Line buffers
        variable str_buf    : string(1 to 10); -- string to modify
        variable value_buf  : std_logic_vector(31 downto 0);
        variable address    : integer := 0;
begin
        if tb_reset_i = '1' then
                address := 0;
                loop
                        if endfile(imem_init_file) then
                                exit;
                        else
                                readline(imem_init_file, line_buf);
                                read(line_buf, str_buf);
                                assert 0 = 1 report "Read: " & str_buf severity warning;
                                str_buf := str_buf(str_buf'left + 2 to str_buf'right) & "  ";
                                write(line_buf, str_buf);

                                hread(line_buf, value_buf);

                                inst_memory(address) <= value_buf;
                                address              := address + 1;
                        end if;
                end loop;
        else
                tb_instr_i <= inst_memory(to_integer(shift_right(unsigned(tb_prcount_o), 2)));
        end if;
end process;

end Behavioral;
```

Keep in touch for the next update
ampadkwa@b-tu.de
koampadu@st.ug.edu.gh