Samuel Lichtenheld

10/1/2015

Assignment 2

Introduction.

The purpose of this assignment is to get hands on experience with an assembler and disassembler as well as start to make parts of the datapath that will be needed later on.

Setup.

I used C++ for the assembler/diassambler and VHDL and ModelSim for the remainder of the lab.

2.1.1.

	Opcode	Rs	Rt	Rd	Shamt	funct
OR$4,$5,$2	000000	00101	00010	00100	00000	010101
JR $3	00000	00011	00000	00000	00000	001000

	Opcode	Rs	Rt	Imm
BEQ $6, $7, loop	000100	00110	00111	1111111111111110

2.1.2.

0000 0000 0000 00010

Same thing but unsigned

2.1.3.

You can branch 2^18 bytes away, so 2^16 instructions away. This is because there are 16 bits available in the immediate field and this value gets shifted left twice to give a farther range.

You can branch further away with a jump which gives you 25 bits of address space or even further away with jump register, which will jump to the address in a register.

2.2.1

2 separate labels. Thus, 21 instruction + 2 Labels = 23 lines

2.2.2

There are instructions that load only 1 byte, such as load byte. Therefore, it seems to be byte addressable.

Name	Mnemonic	Operation	Type	Opcode (hex)	Funct (hex)	ALU action	ALU function code (bit)	Flags set (**Z**ero, **C**arry (unsigned overflow), **V** (signed overflow), **S**ign(+/-)
Add	add	R[rd] = R[rs] + R[rt]	R	00	20	add	000	CZSV
add imm.	addi	R[rt] = R[rs] + SignExtImm	I	08	-	add	001	CZSV
add imm. uns.	addiu	R[rt] = R[rs] + ZeroExtImm	I	09	-	add	001	CZSV
add uns.	addu	R[rd] = R[rs] + R[rt]	R	00	21	add	000	CZSV
and	and	R[rd] = R[rs] & R[rt]	R	00	24	and	000	SZ
and imm.	andi	R[rt] = R[rs] & ZeroExtImm	I	0C	-	and	011	SZ
branch eq	beq	if (R[rs]==R[rt]) PC=PC+4+BranchAddr	I	04	-	subtract	010	CZSV
branch not eq	bne	if (R[rs]!=R[rt]) PC=PC+4+BranchAddr	I	05	-	subtract	010	CZSV
jump	j	PC=JumpAddr	J	02	-	add	001	CZSV
jump and link	jal	R[31]=PC+8;PC=JumpAddr	J	03	-	add	001	CZSV
jump reg	jr	PC=R[rs]	R	00	08	-	000	
load byte uns.	lbu	R[rt] = {24'b0,M[R[rs]+SignExtImm](7:0)}	I	24	-	add	001	CZSV
load halfword uns.	lhu	R[rt] = {16'b0,M[R[rs]+SignExtImm](15:0)}	I	25	-	add	001	CZSV
load upper imm.	lui	R[rt] = {imm, 16'b0}	I	0F	-	-		

load word	lw	R[rt] = M[R[rs]+SignExtImm]	I	23	-	add	001	CZSV
Nor	nor	R[rd] = ~(R[rs] | R[rt])	R	00	27	NOR	000	SZ
Or	or	R[rd] = R[rs] | R[rt]	R	00	25	OR	000	SZ
Or imm.	ori	R[rt] = R[rs] | ZeroExtImm	I	0D	-	OR	100	SZ
set less than	slt	R[rd] = (R[rs]<R[rt]) ? 1:0	R	00	2A	slt	000	SZ
set less than imm.	slti	R[rt] = (R[rs]<SignExtImm) ? 1:0	I	0A	-	slt	101	SZ
set less than imm. uns.	sltiu	R[rt] = (R[rs]<SignExtImm) ? 1:0	I	0B	-	slt	101	SZ
set less than uns.	sltu	R[rd] = (R[rs]<R[rt]) ? 1:0	R	00	2B	slt	000	SZ
shift left logical	sll	R[rd] = R[rt] << shamt	R	00	00	sll	000	SZ
shift right logical	srl	R[rd] = R[rt] >>> shamt	R	00	02	srl	000	SZ
store byte	sb	M[R[rs]+SignExtImm}(7:0) = R[rt](7:0)	I	28	-	add	001	
store halfword	sh	M[R[rs]+SignExtImm}(15:0) = R[rt](15:0)	I	29	-	add	001	
store word	sw	M[R[rs]+SignExtImm} = R[rt]	I	2B	--	add	001	
subtract	sub	R[rd] = R[rs] -R[rt]	R	00	22	subtract	000	CZSV
subtract uns.	subu	R[rd] = R[rs]= R[rt]	R	00	23	subtract	000	CZSV

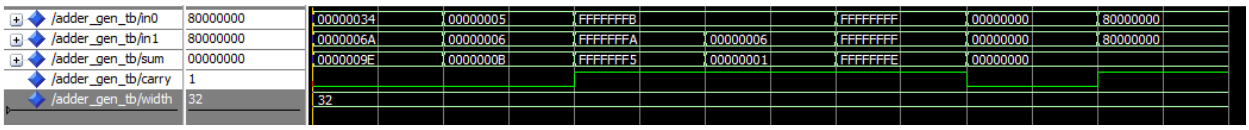

Figure 1: Adder TB

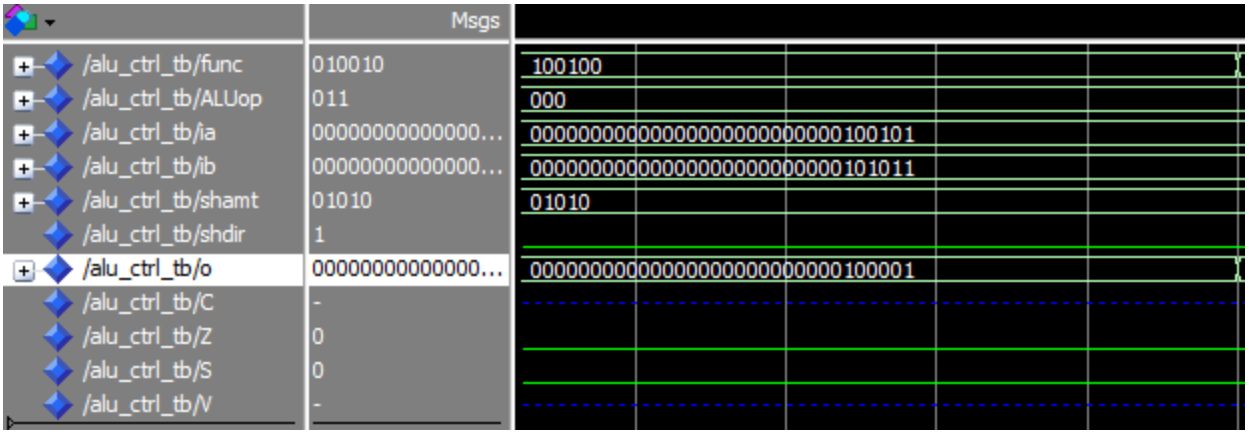

Figure 2: AND ALU + CTRL

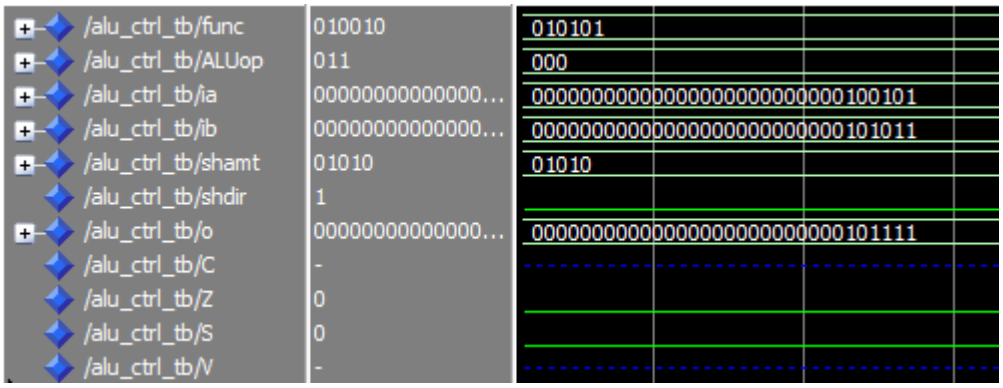

Figure 3: OR ALU + CTRL

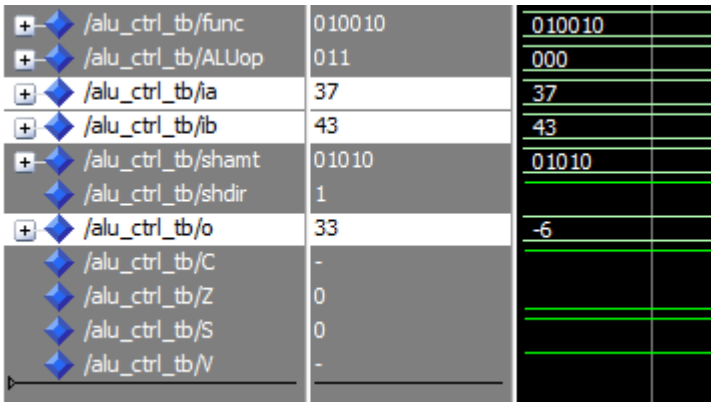

Figure 4: SUB ALU + CTRL

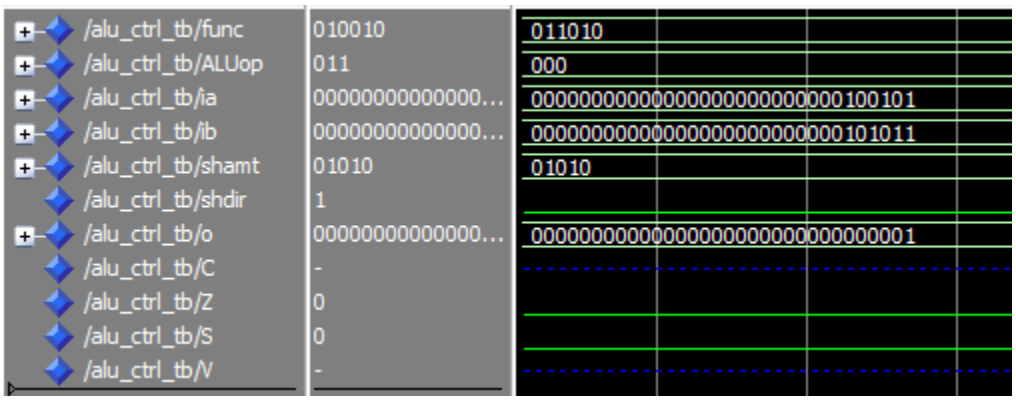

Figure 5: SLT ALU + CTRL

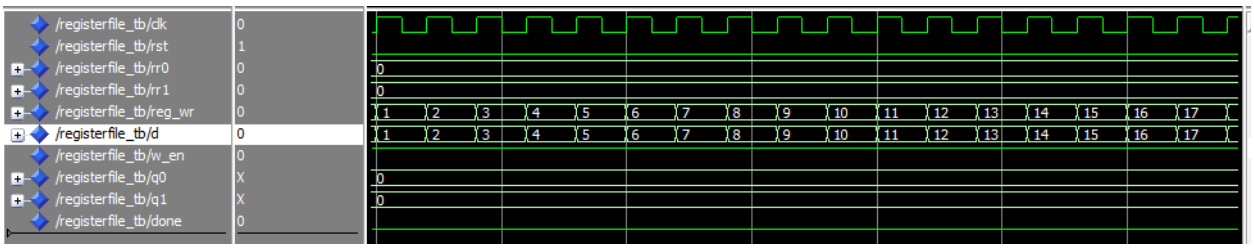

Figure 6: LOAD regfile

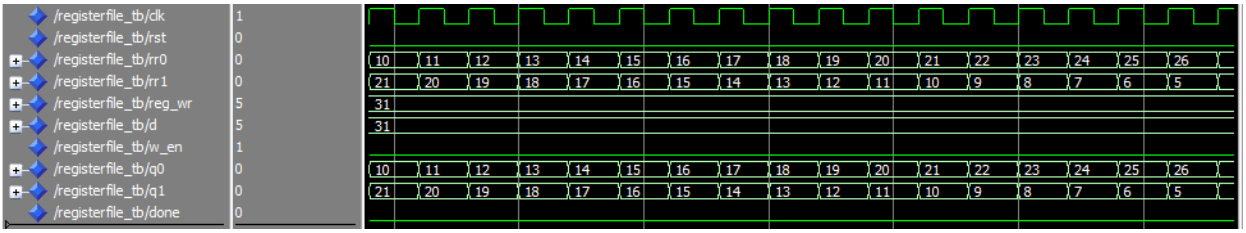

Figure 7: Read from regfile

ADDER

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

entity adder_gen is
	generic (
		width : positive := 32
	);
	port (
		in0			: in std_logic_vector (width-1 downto 0);
		in1		: in std_logic_vector (width-1 downto 0);
		carry		: out std_logic;
		sum		: out std_logic_vector (width-1 downto 0)
	);
end adder_gen;

architecture BHV of adder_gen is
begin
	process(in0, in1)

	variable temp : unsigned (width downto 0);

	begin
		temp := unsigned("0"&in0) + unsigned("0"&in1);
		sum <= std_logic_vector(temp(width-1 downto 0));
		carry <= std_logic(temp(width));
	end process;
end BHV;
```

ADDER TB

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

entity adder_gen_tb is
end adder_gen_tb;

architecture TB of adder_gen_tb is
	constant width : positive := 32;

	signal in0, in1, sum : std_logic_vector(width-1 downto 0);
	signal carry: std_logic;
begin
	UUT: entity work.adder_gen
		generic map (width => width)
		port map (
			in0 => in0,
			in1 => in1,
			carry => carry,
```

```
                        sum => sum
                );

process

        begin
    -- test all input combinations

    in0 <= std_logic_vector(to_signed(52,width));
    in1 <= std_logic_vector(to_signed(106,width));
    wait for 10 ns;

    in0 <= std_logic_vector(to_signed(5,width));
    in1 <= std_logic_vector(to_signed(6,width));
    wait for 10 ns;

    in0 <= std_logic_vector(to_signed(-5,width));
    in1 <= std_logic_vector(to_signed(-6,width));
    wait for 10 ns;

    in0 <= std_logic_vector(to_signed(-5,width));
    in1 <= std_logic_vector(to_signed(6,width));
    wait for 10 ns;

    in0 <= x"FFFFFFFF";
    in1 <= x"FFFFFFFF";
    wait for 10 ns;

    in0 <= x"00000000";
    in1 <= x"00000000";
    wait for 10 ns;

    in0 <= x"80000000";
    in1 <= x"80000000";
    wait for 10 ns;
    wait;



        report "Simulation Finished.";

        wait;
end process;
end TB;
```

ALU32

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

```
entity adder_gen_tb is
end adder_gen_tb;

architecture TB of adder_gen_tb is
	constant width : positive := 32;

	signal in0, in1, sum : std_logic_vector(width-1 downto 0);
	signal carry: std_logic;
begin
	UUT: entity work.adder_gen
		generic map (width => width)
		port map (
			in0 => in0,
			in1 => in1,
			carry => carry,
			sum => sum
		);

process

	begin
   -- test all input combinations

   in0 <= std_logic_vector(to_signed(52,width));
   in1 <= std_logic_vector(to_signed(106,width));
   wait for 10 ns;

   in0 <= std_logic_vector(to_signed(5,width));
   in1 <= std_logic_vector(to_signed(6,width));
   wait for 10 ns;

   in0 <= std_logic_vector(to_signed(-5,width));
   in1 <= std_logic_vector(to_signed(-6,width));
   wait for 10 ns;

   in0 <= std_logic_vector(to_signed(-5,width));
   in1 <= std_logic_vector(to_signed(6,width));
   wait for 10 ns;

   in0 <= x"FFFFFFFF";
   in1 <= x"FFFFFFFF";
   wait for 10 ns;

   in0 <= x"00000000";
   in1 <= x"00000000";
   wait for 10 ns;

   in0 <= x"80000000";
   in1 <= x"80000000";
   wait for 10 ns;
```

```
    wait;



        report "Simulation Finished.";

        wait;
end process;
end TB;
```

ALU32CONTROL

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


--ALUop
--R type       000
--add          001
--sub          010
--and          011
--or           100
--slt          101

entity alu32control is
        port (
                func    : in std_logic_vector(5 downto 0);
                ALUop : in std_logic_vector(2 downto 0);
                control  : out std_logic_vector(3 downto 0)
        );
end alu32control;


architecture BHV of alu32control is

constant C_ADD          : std_logic_vector(3 downto 0) := "0010";
constant C_SUB          : std_logic_vector(3 downto 0) := "0110";
constant C_AND          : std_logic_vector(3 downto 0) := "0000";
constant C_OR  : std_logic_vector(3 downto 0) := "0001";
constant C_NOR          : std_logic_vector(3 downto 0) := "1100";
constant C_SLT: std_logic_vector(3 downto 0) := "0111";
constant C_SLTU         : std_logic_vector(3 downto 0) := "1111";
constant C_SHL          : std_logic_vector(3 downto 0) := "0011";

constant rOp    : std_logic_vector(2 downto 0) := "000";
constant addOp : std_logic_vector(2 downto 0) := "001";
constant subOp : std_logic_vector(2 downto 0) := "010";
constant andOp : std_logic_vector(2 downto 0) := "011";
constant orOp   : std_logic_vector(2 downto 0) := "100";
constant sltOp   : std_logic_vector(2 downto 0) := "101";

constant ADD_F          : std_logic_vector(5 downto 0) := "100000";
```

```
constant ADDU_F        : std_logic_vector(5 downto 0) := "100001";
constant AND_F         : std_logic_vector(5 downto 0) := "100100";
constant JR_F   : std_logic_vector(5 downto 0) := "001000";
constant NOR_F         : std_logic_vector(5 downto 0) := "010111";
constant OR_F  : std_logic_vector(5 downto 0) := "010101";
constant SLT_F: std_logic_vector(5 downto 0) := "011010";
constant SLTU_F        : std_logic_vector(5 downto 0) := "011011";
constant SLL_F: std_logic_vector(5 downto 0) := "000000";
constant SRL_F: std_logic_vector(5 downto 0) := "000010";
constant SUB_F         : std_logic_vector(5 downto 0) := "010010";
constant SUBU_F        : std_logic_vector(5 downto 0) := "010011";

begin


        process(func, ALUop)
        begin
                case ALUop is
                        when rOp =>
                                case func is
                                        when ADD_F =>
                                                control <= C_ADD;
                                        when ADDU_F =>
                                                control <= C_ADD;
                                        when AND_F =>
                                                control <= C_AND;
                                        when JR_F =>
                                                control <= "----";
                                        when NOR_F =>
                                                control <= C_NOR;
                                        when OR_F =>
                                                control <= C_OR;
                                        when SLT_F =>
                                                control <= C_SLT;
                                        when SLTU_F =>
                                                control <= C_SLTU;
                                        when SLL_F =>
                                                control <= C_SHL;
                                        when SRL_F =>
                                                control <= C_SHL;
                                        when SUB_F =>
                                                control <= C_SUB;
                                        when SUBU_F =>
                                                control <= C_SUB;
                                        when others =>
                                                control <= "----";
                                end case;
                        when addOp     =>
                                control <= C_ADD;
                        when subOp     =>
                                control <= C_SUB;
```

```
                        when andOp      =>
                                control <= C_AND;
                        when orOp       =>
                                control <= C_OR;
                        when sltOp      =>
                                control <= C_SLT;
                        when others =>
                                control <= "----";
                end case;
        end process;
end BHV;
```

ALU+CTRL

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

entity alu_ctrl is
        port(
                func     : in std_logic_vector(5 downto 0);
                ALUop : in std_logic_vector(2 downto 0);
                ia, ib     : in std_logic_vector(31 downto 0);
                shamt   : in std_logic_vector(4 downto 0);
                shdir     : in std_logic;
                o                : out std_logic_vector(31 downto 0);
                C,Z,S,V : out std_logic
        );
end alu_ctrl;

architecture STR of alu_ctrl is

        signal ctrlSig     : std_logic_vector(3 downto 0);

begin

        U_ALU32CONTROL : entity work.alu32control
                port map (
                        func => func,
                        ALUop => ALUop,
                        control => ctrlSig
                );

        U_ALU32: entity work.alu32
                port map (
                        ia => ia,
                        ib => ib,
                        ctrl => ctrlSig,
                        shamt => shamt,
                        shdir => shdir,
                        o       => o,
                        C       => C,
```

```
            Z       => Z,
            S       => S,
            V       => V
        );


end STR;
```

ALU_CTRL_TB

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

entity alu_ctrl_tb is
end alu_ctrl_tb;


architecture TB of alu_ctrl_tb is

    constant C_ADD      : std_logic_vector(3 downto 0) := "0010";
    constant C_SUB      : std_logic_vector(3 downto 0) := "0110";
    constant C_AND      : std_logic_vector(3 downto 0) := "0000";
    constant C_OR  : std_logic_vector(3 downto 0) := "0001";
    constant C_NOR      : std_logic_vector(3 downto 0) := "1100";
    constant C_SLT: std_logic_vector(3 downto 0) := "0111";
    constant C_SLTU     : std_logic_vector(3 downto 0) := "1111";
    constant C_SHL      : std_logic_vector(3 downto 0) := "0011";

    constant rOp     : std_logic_vector(2 downto 0) := "000";
    constant addOp : std_logic_vector(2 downto 0) := "001";
    constant subOp : std_logic_vector(2 downto 0) := "010";
    constant andOp : std_logic_vector(2 downto 0) := "011";
    constant orOp    : std_logic_vector(2 downto 0) := "100";
    constant sltOp   : std_logic_vector(2 downto 0) := "101";

    constant ADD_F      : std_logic_vector(5 downto 0) := "100000";
    constant ADDU_F     : std_logic_vector(5 downto 0) := "100001";
    constant AND_F      : std_logic_vector(5 downto 0) := "100100";
    constant JR_F   : std_logic_vector(5 downto 0) := "001000";
    constant NOR_F      : std_logic_vector(5 downto 0) := "010111";
    constant OR_F  : std_logic_vector(5 downto 0) := "010101";
    constant SLT_F: std_logic_vector(5 downto 0) := "011010";
    constant SLTU_F     : std_logic_vector(5 downto 0) := "011011";
    constant SLL_F: std_logic_vector(5 downto 0) := "000000";
    constant SRL_F: std_logic_vector(5 downto 0) := "000010";
    constant SUB_F      : std_logic_vector(5 downto 0) := "010010";
    constant SUBU_F     : std_logic_vector(5 downto 0) := "010011";
```

```
	signal	func	: std_logic_vector(5 downto 0);
	signal	ALUop : std_logic_vector(2 downto 0);
	signal	ia, ib	: std_logic_vector(31 downto 0);
	signal	shamt	: std_logic_vector(4 downto 0);
	signal	shdir	: std_logic;
	signal	o		: std_logic_vector(31 downto 0);
	signal	C,Z,S,V : std_logic;


begin

	UUT: entity work.alu_ctrl
		port map(
			func	=> func,
			ALUop		=> ALUop,
			ia		=> ia,
			ib		=> ib,
			shamt	=> shamt,
			shdir	=> shdir,
			o		=> o,
			C		=> C,
			Z		=> Z,
			S		=> S,
			V		=> V
		);
	process
	begin

		func <= ADD_F;
		ALUop <= rOp;
		ia <= std_logic_vector(to_unsigned(37,32));
		ib <= std_logic_vector(to_unsigned(43,32));
		shamt <= std_logic_vector(to_unsigned(10,5));
		shdir <= '0';
		wait for 10 ns;

		func <= AND_F;
		wait for 10 ns;

		func <= OR_F;
		wait for 10 ns;

		func <= SLT_F;
		wait for 10 ns;

		func <= SLL_F;
		wait for 10 ns;

		shdir <= '1';
```

```
		wait for 10 ns;

		func <= SUB_F;
		wait for 10 ns;

		ALUop <= andOp;
		wait for 10 ns;


		wait;
	end process;
end TB;
```

REGISTERFILE

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

entity registerFile is
	port (
		clk, rst  : in std_logic;
		rr0			: in std_logic_vector(4 downto 0); -- register file reads
		rr1			: in std_logic_vector(4 downto 0);
		reg_wr		: in std_logic_vector(4 downto 0); --which reg should be written to on rising edge of clk
		d			: in std_logic_vector(31 downto 0); -- input word
		w_en		: in std_logic;					-- write register enable
		q0, q1		: out std_logic_vector(31 downto 0)
	);
end registerFile;

architecture STR of registerFile is

	signal ldi		: std_logic_vector(31 downto 0);

	type rr_array is array(31 downto 0) of std_logic_vector(31 downto 0);
	signal q_array: rr_array;

	--type registerarray is array(X downto 0) of std_logic_vector(X downto 0);
	--signal ra: registerarray;
	--The first line defines new a type that is an array of std_logic_vector's; the second line initializes a signal of type “registerarray” with name “ra”. The defined signal can be used in following manner “ra(0) <= 0xDEADBEE7”.
begin

	U_REG_ARRAY: for i in 0 to 31 generate
		U_REG		: entity work.reg_gen
			generic map(32)
			port map(
```

```
                                clk => clk,
                                rst => rst,
                                ld => ldi(i),
                                d => d,
                                q => q_array(i)
                        );
        end generate U_REG_ARRAY;

        process(clk,rst,rr0,rr1,reg_wr,d,w_en)
        variable temp: unsigned (31 downto 0);
        begin

        q0 <= q_array(to_integer(unsigned(rr0)));
        q1 <= q_array(to_integer(unsigned(rr1)));

        temp := x"00000000";
        if (w_en = '1') then
                temp(to_integer(unsigned(reg_wr))) := '1';
        end if;
        ldi <= std_logic_vector(temp);

        end process;
end STR;
```

REGISTERFILE TB

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

entity registerFile_tb is
end registerFile_tb;


architecture TB of registerFile_tb is
        signal clk              : std_logic := '0';
        signal rst              : std_logic;
        signal rr0              : std_logic_vector(4 downto 0);
        signal rr1              : std_logic_vector(4 downto 0);
        signal reg_wr   : std_logic_vector(4 downto 0);
        signal d                : std_logic_vector(31 downto 0);
        signal w_en     : std_logic;
        signal q0               : std_logic_vector(31 downto 0);
        signal q1               : std_logic_vector(31 downto 0);

        signal done     : std_logic := '0';

begin
```

```
	clk <= not clk and not done after 10 ns;

	UUT: entity work.registerFile
		port map(
			clk => clk,
			rst => rst,
			rr0 => rr0,
			rr1 => rr1,
			reg_wr => reg_wr,
			d => d,
			w_en => w_en,
			q0 => q0,
			q1 => q1
		);

	process
	begin

		rst <= '0';
		rr0 <= std_logic_vector(to_unsigned(0,5));
		rr1 <= std_logic_vector(to_unsigned(0,5));
		reg_wr <= std_logic_vector(to_unsigned(0,5));
		d <= std_logic_vector(to_unsigned(0,32));
		w_en <= '0';
		wait for 1 ns;
		rst <= '1';
		wait for 19 ns;
		rst <= '0';
		wait for 10 ns;

		for i in 0 to 31 loop
			w_en <= '1';
			d <= std_logic_vector(to_unsigned(i,32));
			reg_wr <= std_logic_vector(to_unsigned(i,5));
			wait until rising_edge(clk);
		end loop;

		w_en <= '0';
		wait until rising_edge(clk);

		for i in 0 to 31 loop
			rr0 <= std_logic_vector(to_unsigned(i,5));
			rr1 <= std_logic_vector(to_unsigned(31-i,5));
			wait until rising_edge(clk);
		end loop;

		done <='1';
		wait;
	end process;
end TB;
```