Computer Architectures
Lab Report 2

DATAPATH DESIGN

Y3839090 & Y3840426| Computer Architectures | ELE00009I

ALL DataPaths

ALUParam.vhd (From Lab 1)(Used for All DataPaths)

```
----------------------------------------------------------------------------------
--
-- Uni                     :        University of York
-- Course          :        Electronic Engineering
-- Module                  :        Computer Architectures
-- Engineers       :        Y3839090 & Y3840426
--
-- Create Date     :        13:19:20 02/17/2017
-- Design Name     :        ALU_param - Behavioral
-- Description     :        A paramateriable integer ALU.
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;

entity ALU_param is
        Generic (
                N : natural := 8 -- data size in bits
        );
        Port (
                A : in  STD_LOGIC_VECTOR (N-1 downto 0);
                B : in  STD_LOGIC_VECTOR (N-1 downto 0);
                X : in  STD_LOGIC_VECTOR (log2(N)-1 downto 0); -- shift/rotate amount input
                ctrl : in  STD_LOGIC_VECTOR (3 downto 0);      -- control signals from opcode
                O : out  STD_LOGIC_VECTOR (N-1 downto 0);
                flags :out  STD_LOGIC_VECTOR (7 downto 0)      -- flags
        );
end ALU_param;

architecture Behavioral of ALU_param is

        -- internal signed signal for A and B inpy=uts
        signal A_itrn : SIGNED (N-1 downto 0);
        signal B_itrn : SIGNED (N-1 downto 0);

        -- internal integer for X
        signal X_itrn : integer;

        -- internal signed signal for output
        signal O_itrn : SIGNED (N-1 downto 0);


        -- max positive and negitive N bit signed numbers
        constant max_pos : SIGNED (N-1 downto 0) := to_signed(( 2 ** (N-1) ) - 1, N);
        constant max_neg : SIGNED (N-1 downto 0) := to_signed( -2 ** (N-1) , N);

begin

                A_itrn <= signed(A); -- converts A to signed and maps the result to A_itrn
                B_itrn <= signed(B); -- converts A to signed and maps the result to B_itrn

                -- converts X to integer and maps the result to X_itrn
                X_itrn <= to_integer(unsigned(X));

                -- converts O_itrn to a plain std_logic_vector and maps it to O
                O <= std_logic_vector(O_itrn);

                -- Main ALU multiplexer for each possible command
                O_itrn <=
                        A_itrn                  when ctrl = "0000" else             -- Output A
                        A_itrn and B_itrn       when ctrl = "0100" else             -- Output A & B
                        A_itrn or B_itrn        when ctrl = "0101" else             -- Output A || B
                        A_itrn xor B_itrn       when ctrl = "0110" else             -- Output A xor B
                        not A_itrn              when ctrl = "0111" else             -- Output not A
                        A_itrn + 1              when ctrl = "1000" else             -- Output A + 1
                        A_itrn - 1              when ctrl = "1001" else             -- Output A - 1
                        A_itrn + B_itrn         when ctrl = "1010" else             -- Output A + B
                        A_itrn - B_itrn         when ctrl = "1011" else             -- Output A - B
                        SHIFT_LEFT (A_itrn , X_itrn)        when ctrl = "1100" else -- Output A sla X
                        SHIFT_RIGHT (A_itrn , X_itrn)       when ctrl = "1101" else -- Output A sra X
                        ROTATE_LEFT (A_itrn , X_itrn)       when ctrl = "1110" else -- Output A rotl X
                        ROTATE_RIGHT (A_itrn , X_itrn)      when ctrl = "1111" else -- Output A rotr X
                        (others =>'U');
```

(Continued.)

```
        -- Overflow flag
        flags(7) <=
                -- Will overflow if you add one to the max positive value
                '1' when ctrl = "1000" and A_itrn = max_pos else

                -- Will overflow if you minus one to the max negitive value
                '1' when ctrl = "1001" and A_itrn = max_neg else

                -- Will overflow if two neg values added give a pos result
                '1' when ctrl = "1010" and A_itrn(N-1) = '1' and  B_itrn(N-1) = '1' and O_itrn(N-1) = '0' else
                -- Will overflow if two pos values added give a neg result
                '1' when ctrl = "1010" and A_itrn(N-1) = '0' and  B_itrn(N-1) = '0' and O_itrn(N-1) = '1' else

                -- Will overflow if a pos value is subtracted from a neg value gives a pos result
                '1' when ctrl = "1011" and A_itrn(N-1) = '1' and  B_itrn(N-1) = '0' and O_itrn(N-1) = '0' else
                -- Will overflow if a neg value is subtracted from a pos value gives a neg result
                '1' when ctrl = "1011" and A_itrn(N-1) = '0' and  B_itrn(N-1) = '1' and O_itrn(N-1) = '1'

                -- If none of the above are true then the result hasn't overflown
                else '0';

        -- Other flags
        flags(6) <= '1'         when O_itrn >= 0        else '0'; -- grater than or equal to zero
        flags(5) <= '1'         when O_itrn <= 0        else '0'; -- less than or equal to zero
        flags(4) <= '1'         when O_itrn >  0        else '0'; -- grater than zero
        flags(3) <= '1'         when O_itrn <  0        else '0'; -- less than zero
        flags(2) <= '1'         when O_itrn =  1        else '0'; -- one flag
        flags(1) <= '1'         when O_itrn /= 0        else '0'; -- not zero flag
        flags(0) <= '1'         when O_itrn =  0        else '0'; -- zero flag


end Behavioral;
```

EasyPrint.vhd (Used in testbenches)

A custom package that adds functions that make it easy to print std_logic_vectors in report statements. Two of these function are from stack overflow comments links to the origanl source of the functions are provided.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
USE ieee.numeric_std.ALL;

package easyprint is

function s_tostr(val : std_logic_vector) return string;
function u_tostr(val : std_logic_vector) return string;

function to_bstring(sl : std_logic) return string;
function to_bstring(slv : std_logic_vector) return string;


end easyprint;

package body easyprint is

        -- ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ====
        -- From http://stackoverflow.com/a/24336034 By Morten Zilmer
        -- Allows printing a std_logic_vector as a string that represents it's binary form.
        -- ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ====
        function to_bstring(sl : std_logic) return string is
                variable sl_str_v : string(1 to 3);  -- std_logic image with quotes around
        begin
                sl_str_v := std_logic'image(sl);
                return "" & sl_str_v(2);  -- "" & character to get string
        end function;

        function to_bstring(slv : std_logic_vector) return string is
                alias    slv_norm : std_logic_vector(1 to slv'length) is slv;
                variable sl_str_v : string(1 to 1);  -- String of std_logic
                variable res_v    : string(1 to slv'length);
        begin
                for idx in slv_norm'range loop
                        sl_str_v := to_bstring(slv_norm(idx));
```

(Continued.)

```
                        res_v(idx) := sl_str_v(1);
                end loop;
                return res_v;
        end function;
        -- ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ====


        -- converts an std_logic_vector to a string that represents it's signed value
        function s_tostr(val : std_logic_vector) return string is
        begin
                return integer'image( to_integer(signed(val)) );
        end function;

        -- converts an std_logic_vector to a string that represents it's unsigned value
        function u_tostr(val : std_logic_vector) return string is
        begin
                return integer'image( to_integer(unsigned(val)) );
        end function;

end easyprint;
```

Reg.vhd (Used for all Datapaths)

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

-- A paramateriable synchronous reset D-type registers with enable
entity Reg is
        Generic(
                data_size: natural := 8 -- How many bits will the register deal with
        );
        Port (
                clk : in  STD_LOGIC;
                rst : in STD_LOGIC;     -- synchronus reset
                en : in STD_LOGIC; -- synchronus reset
                data_in : in  STD_LOGIC_VECTOR (data_size-1 downto 0); -- input
                data_out : out  STD_LOGIC_VECTOR (data_size-1 downto 0) -- output
        );
end Reg;

architecture Behavioral of Reg is
begin

        process(clk)
        begin
                -- Synchronise to the clock
                if (rising_edge(clk)) then
                        if (rst = '1') then
                                -- Reset to zero
                                data_out <= (others => '0');
                        elsif (en = '1') then
                                -- Pass the input to the output
                                data_out <= data_in;
                        end if;
                end if;

        end process;


end Behavioral;
```

Architecture B

DataPathB.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.all;

entity DataPath_B is
        GENERIC(
                data_size : natural := 16;
                num_registers : natural := 32
        );
        Port (
                clk : in  STD_LOGIC;

                R_A : in  STD_LOGIC_VECTOR (log2(num_registers)-1 downto 0); -- Read address A
                R_B : in  STD_LOGIC_VECTOR (log2(num_registers)-1 downto 0); -- Read address B
                W_EN : in  STD_LOGIC;                                        -- Register write enable
                W_A : in  STD_LOGIC_VECTOR (log2(num_registers)-1 downto 0); -- Write address
                IMM : in  STD_LOGIC_VECTOR (data_size-1 downto 0);           -- Itermediate value
                AL : in  STD_LOGIC_VECTOR (3 downto 0);                      -- ALU control
                SH : in  STD_LOGIC_VECTOR (log2(data_size)-1 downto 0);      -- Shift amount
                M_A : in  STD_LOGIC_VECTOR (data_size-1 downto 0);           -- Memory address
                S : in  STD_LOGIC_VECTOR (4 downto 1);                       -- Selector control
                flags : out STD_LOGIC_VECTOR(7 downto 0);                    -- ALU flags
                M_B : out STD_LOGIC_VECTOR (data_size-1 downto 0);           -- Memory output (write)
                M_DA : out  STD_LOGIC_VECTOR (data_size-1 downto 0);         -- Memory address
                M_in : in  STD_LOGIC_VECTOR (data_size-1 downto 0)           -- Memory input (read)
        );
end DataPath_B;

architecture Behavioral of DataPath_B is

        signal reg_in : STD_LOGIC_VECTOR (data_size-1 downto 0); -- register write data

        signal A_data : STD_LOGIC_VECTOR (data_size-1 downto 0); -- Data from register at address R_A
        signal B_data : STD_LOGIC_VECTOR (data_size-1 downto 0); -- Data from register at address R_B

        signal B_mux : STD_LOGIC_VECTOR (data_size-1 downto 0); -- B input to the ALU

        signal ALU_out : STD_LOGIC_VECTOR (data_size-1 downto 0); -- Output of the ALU

begin

        -- Multiplexer on the ALU's B input
        B_mux <= B_data when S(1) = '0' else IMM;

        -- Multiplexer for the memory address
        M_DA <= M_A when s(3) = '1' else ALU_out;

        -- Multiplexer for the register write data
        reg_in <= ALU_out when s(4) = '0' else M_in;

        -- Memory write (output) connection
        M_B <= B_data;

        -- The ALU
        ALU: entity work.ALU_param
        GENERIC MAP(
                N => data_size
        )
        PORT MAP(
                A => A_data,
                B => B_mux,
                X => SH,
                ctrl => AL,
                O => ALU_out,
                flags => flags
        );
```

(Continued.)

```
        -- The register bank
        Registers: entity work.regbank
        PORT MAP(
                RSELA => R_A,
                RSELB => R_B,
                WSEL => W_A,
                D => reg_in,
                WEN => W_EN,
                clk => clk,
                A => A_data,
                B => B_data,
                rst => '0'
        );

end Behavioral;
```

Test Data for DataPath B

Here is the control signals and expected outputs that we are using to test datapath B.

Test Command	W_EN	AL[3:0]	R_A[4:0]	R_B[4:0]	W_A[4:0]	IMM[15:0]	SH[3:0]	M_A[3:0]	S[4:1]	M_in[3:0]	flags[7:0]	M_B[3:0]	M_DA[3:0]	OEN
inc R1, R0	1	1000	00000	ØØØØØ	00001	0xØØØØ	ØØØØ	0xØØØØ	0ØØØ	0xØØØØ	01010110	0xØØØØ	0xØØØØ	0
addi R2, R0, 0x0005	1	1010	00000	ØØØØØ	00010	0x0005	ØØØØ	0xØØØØ	0ØØ1	0xØØØØ	01010010	0xØØØØ	0xØØØØ	0
shl R3, R1, 3	1	1100	00001	ØØØØØ	00011	0xØØØØ	0011	0xØØØØ	0ØØØ	0xØØØØ	01010010	0xØØØØ	0xØØØØ	0
storr R2, R3	0	0000	00011	00010	ØØØØØ	0xØØØØ	ØØØØ	0xØØØØ	0ØØØ	0xØØØØ	01010010	0x0005	0x0008	1
loadi R5, 1f1f	1	ØØØØ	ØØØØØ	ØØØØØ	00101	0xØØØØ	ØØØØ	0x1f1f	11ØØ	0xCCCC	ØØØØØØØØ	0xØØØØ	0x1f1f	0

DataPathB-TB.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.DigEng.ALL;
USE work.easyprint.ALL;
USE ieee.numeric_std.ALL;

ENTITY DataPathB_TB IS
END DataPathB_TB;

ARCHITECTURE behavior OF DataPathB_TB IS

        -- Constants
        constant data_size : NATURAL := 16;
        constant num_registers : NATURAL := 32;

        -- Component Declaration for the Unit Under Test (UUT)
        COMPONENT DataPath_B
                GENERIC(
                        data_size : natural;
                        num_registers : natural
                );
                PORT(
                        R_A : IN  std_logic_vector(log2(num_registers)-1 downto 0);
                        R_B : IN  std_logic_vector(log2(num_registers)-1 downto 0);
                        W_EN : IN  std_logic;
                        W_A : IN  std_logic_vector(log2(num_registers)-1 downto 0);
                        clk : IN  std_logic;
                        IMM : IN  std_logic_vector(data_size-1 downto 0);
                        AL : IN  std_logic_vector(3 downto 0);
                        SH : IN  std_logic_vector(log2(data_size)-1 downto 0);
                        M_A : IN  std_logic_vector(data_size-1 downto 0);
                        S : IN  std_logic_vector(4 downto 1);
                        flags : OUT  std_logic_vector(7 downto 0);
                        M_B : OUT  std_logic_vector(data_size-1 downto 0);
                        M_DA : OUT  std_logic_vector(data_size-1 downto 0);
                        M_in : IN  std_logic_vector(data_size-1 downto 0)
                );
        END COMPONENT;


   --Inputs
   signal R_A   : std_logic_vector      (log2(num_registers)-1 downto 0) := (others => '0');
   signal R_B   : std_logic_vector      (log2(num_registers)-1 downto 0) := (others => '0');
   signal W_EN : std_logic                                               := '0';
   signal W_A   : std_logic_vector      (log2(num_registers)-1 downto 0) := (others => '0');
   signal clk   : std_logic                                              := '0';
   signal IMM   : std_logic_vector      (data_size-1 downto 0)           := (others => '0');
   signal AL    : std_logic_vector      (3 downto 0)                     := (others => '0');
   signal SH    : std_logic_vector      (log2(data_size)-1 downto 0)     := (others => '0');
   signal M_A   : std_logic_vector      (data_size-1 downto 0)           := (others => '0');
   signal S     : std_logic_vector      (4 downto 1)                     := (others => '0');
   signal M_in : std_logic_vector       (data_size-1 downto 0)           := (others => '0');
```

(Continued.)

```
--Outputs
signal flags         : std_logic_vector      (7 downto 0);
signal M_B           : std_logic_vector      (data_size-1 downto 0);
signal M_DA  : std_logic_vector      (data_size-1 downto 0);
signal OEN : std_logic := '0';

-- Clock period definitions
constant clk_period  : time  := 10 ns;
constant wait_time   : time  := clk_period;

     -- Test data for self checking test bench
     type TEST_VECTOR is RECORD
              W_EN : std_logic;
              AL : std_logic_vector(3 downto 0);
              R_A : STD_LOGIC_VECTOR(log2(num_registers)-1 downto 0);
              R_B : std_logic_vector(log2(num_registers)-1 downto 0);
              W_A : std_logic_vector(log2(num_registers)-1 downto 0);
              IMM : std_logic_vector(data_size-1 downto 0);
              SH : std_logic_vector(log2(data_size)-1 downto 0) ;
              M_A : std_logic_vector(data_size-1 downto 0);
              S : std_logic_vector(4 downto 1);
              M_in : std_logic_vector(data_size-1 downto 0);

              flags : std_logic_vector(7 downto 0);
              M_B : std_logic_vector(data_size-1 downto 0);
              M_DA : std_logic_vector(data_size-1 downto 0);

              OEN : std_logic;
     end RECORD;

     type TEST_VECTOR_ARRAY is ARRAY(NATURAL RANGE <>) of TEST_VECTOR;

     constant test_vectors : TEST_VECTOR_ARRAY := (
             --W_EN,          AL,              R_A,             R_B,            W_A,             IMM,
               SH,            M_A,                              S,              M_in,                            flags,
               M_B,                            M_DA,                             OEN
             ( '1',           "1000",          "00000",         "-----",        "00001",         "----------------",
               "----",        "----------------",               "0---",         "----------------",              "01010110",
               "----------------",             "----------------",               '0'),

             ('1',            "1010",          "00000",         "-----",        "00010",         X"0005",
               "----",        "----------------",               "0--1",         "----------------",              "01010010",
               "----------------",             "----------------",               '0'),


             ('1',            "1100",          "00001",         "-----",        "00011",         "----------------",
              "0011",         "----------------",               "0---",         "----------------",              "01010010",
              "----------------",             "----------------",               '0'),

             ('0',            "0000",          "00011",         "00010",        "-----",         "----------------",
              "----",         "----------------",               "0---",         "----------------",              "01010010",
              X"0005",                        X"0008",                          '1'),

             ('1',            "----",          "-----",         "-----",        "00101",         "----------------",
              "----",         X"1f1f",                          "11--",         X"CCCC",                         "--------",
              "----------------",             X"1f1f",                          '0')
             );
```

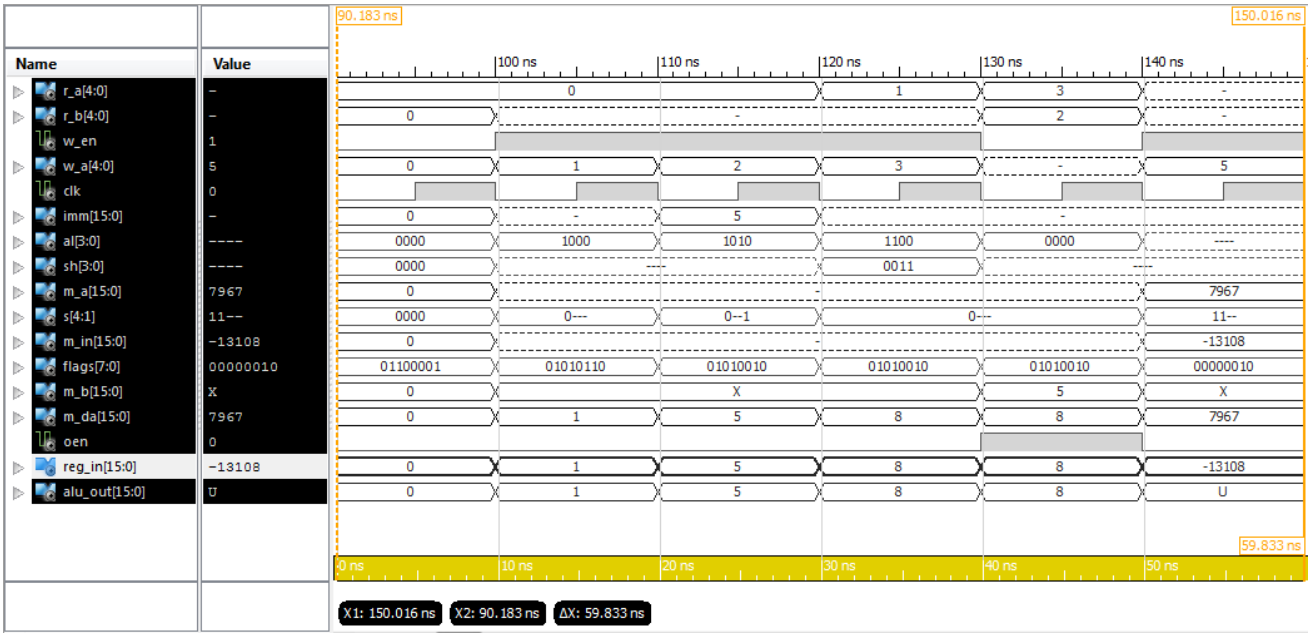
(Continued.)

```
BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: DataPath_B
        Generic Map(
                data_size => data_size,
                num_registers => num_registers
        )
        PORT MAP (
          R_A => R_A,
          R_B => R_B,
          W_EN => W_EN,
          W_A => W_A,
          clk => clk,
          IMM => IMM,
          AL => AL,
          SH => SH,
          M_A => M_A,
          S => S,
          flags => flags,
          M_B => M_B,
          M_DA => M_DA,
          M_in => M_in
        );

   -- Clock process definitions
   clk_process :process
   begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
   end process;


        -- Stimulus process
        stim_proc: process
        begin

                -- hold reset state for 100 ns.
                wait for 100 ns;

                -- run the test for every set of data
                for i in test_vectors'range loop

                        -- assign test inputs
                        R_A <= test_vectors(i).R_A;
                        R_B <= test_vectors(i).R_B;
                        W_EN <= test_vectors(i).W_EN;
                        W_A <= test_vectors(i).W_A;
                        IMM <= test_vectors(i).IMM;
                        AL <= test_vectors(i).AL;
                        SH <= test_vectors(i).SH;
                        M_A <= test_vectors(i).M_A;
                        S <= test_vectors(i).S;
                        M_in <= test_vectors(i).M_in;

                        OEN <= test_vectors(i).OEN;

                        wait until rising_edge(clk);

                        -- Check that the actual outputs are the same as were expecting
                        -- Have to use std_match when comparing meta values like '-'
                        assert std_match(flags, test_vectors(i).flags)
                        report lf & " [ERR!] Test " & integer'image(i)& lf &
                                " Actual flags did not equal expected flags."&
                                " Actual [ " & to_bstring(flags) & " ]" &
                                " Expected [ " & to_bstring(test_vectors(i).flags) & " ]"
                        severity error;

                        assert std_match(test_vectors(i).M_B, M_B)
                        report lf &" [ERR!] Test " & integer'image(i)& lf &
                                " Actual value to memory did not equal expected value to memory."&
                                " Actual [ " & u_tostr(M_B) & " ]" &
                                " Expected [ " & u_tostr(test_vectors(i).M_B) & " ]"
                        severity error;
```

(Continued.)

```
                        assert std_match(M_DA , test_vectors(i).M_DA)
                        report lf &" [ERR!] Test " & integer'image(i)& lf &
                                " Actual memory address did not equal expected memory address."&
                                " Actual [ " & u_tostr(M_DA) & " ]" &
                                " Expected [ " & u_tostr(test_vectors(i).M_DA) & " ]"
                        severity error;

                        -- If there were no isses report that the test was successful
                        assert not (
                                std_match(flags, test_vectors(i).flags) and
                                std_match(M_B, test_vectors(i).M_B) and
                                std_match(M_DA, test_vectors(i).M_DA))
                        report lf &" [ OK ] Test " & integer'image(i)& " was successful!"
                        severity note;

                        wait until falling_edge(clk);

                end loop;

                wait;

        end process;

END;
```

Simulations



This Screenshot shows the entire simulation for DataPath B. The reg_in signal is the data that will be written to the register bank.

iSim Console Output

```
ISim P.28xd (signature 0xa0883be4)
This is a Full version of ISim.
Time resolution is 1 ps
WARNING: Simulation object /datapathb_tb/test_vectors was not traceable in the design for the following reason:
ISim does not yet support tracing of constant and generic multi-dimensional arrays.
Simulator is doing circuit initialization process.
at 0 ps, Instance /datapathb_tb/uut/ALU/ : Warning: NUMERIC_STD.">=": metavalue detected, returning FALSE
at 0 ps, Instance /datapathb_tb/uut/ALU/ : Warning: NUMERIC_STD."<=": metavalue detected, returning FALSE
at 0 ps, Instance /datapathb_tb/uut/ALU/ : Warning: NUMERIC_STD.">": metavalue detected, returning FALSE
at 0 ps, Instance /datapathb_tb/uut/ALU/ : Warning: NUMERIC_STD."<": metavalue detected, returning FALSE
at 0 ps, Instance /datapathb_tb/uut/ALU/ : Warning: NUMERIC_STD."=": metavalue detected, returning FALSE
at 0 ps, Instance /datapathb_tb/uut/ALU/ : Warning: NUMERIC_STD."/=": metavalue detected, returning TRUE
at 0 ps, Instance /datapathb_tb/uut/ALU/ : Warning: NUMERIC_STD."=": metavalue detected, returning FALSE
Finished circuit initialization process.

at 105 ns(1): Note:  [ OK ] Test 0 was successful! (/datapathb_tb/).
at 115 ns(1): Note:  [ OK ] Test 1 was successful! (/datapathb_tb/).
at 125 ns(1): Note:  [ OK ] Test 2 was successful! (/datapathb_tb/).
at 135 ns(1): Note:  [ OK ] Test 3 was successful! (/datapathb_tb/).
at 145 ns(1): Note:  [ OK ] Test 4 was successful! (/datapathb_tb/).

ISim>
```

All but the first meta value warnings have been removed to improve readability.

HDL Synthesis

```
=========================================================================
*                           HDL Synthesis                               *
=========================================================================

Synthesizing Unit <DataPath_B>.
    Related source file is "E:\University\_Second Year\Computer Architectures\assesment\MyMicroProccessor\Lab_2\DataPaths -
RegBankAlt\DataPath_B.vhd".
        data_size = 16
        num_registers = 32
WARNING:Xst:647 - Input <S<2:2>> is never used. This port will be preserved and left unconnected if it belongs to a top-level block or it
belongs to a sub-block and the hierarchy of this sub-block is preserved.
    Summary:
         inferred   3 Multiplexer(s).
```

```
Unit <DataPath_B> synthesized.

Synthesizing Unit <ALU_param>.
    Related source file is "E:\University\_Second Year\Computer Architectures\assesment\MyMicroProccessor\Lab_2\DataPaths -
RegBankAlt\ALU_param.vhd".
        N = 16
    Found 16-bit adder for signal <A_itrn[15]_B_itrn[15]_add_27_OUT> created at line 69.
    Found 16-bit adder for signal <A_itrn[15]_GND_6_o_add_31_OUT> created at line 1253.
    Found 16-bit subtractor for signal <A_itrn[15]_B_itrn[15]_sub_26_OUT<15:0>> created at line 70.
    Found 16-bit subtractor for signal <A_itrn[15]_GND_6_o_sub_30_OUT<15:0>> created at line 1320.
    Found 16-bit shifter rotate right for signal <A_itrn[15]_X_itrn[30]_rotate_right_17_OUT> created at line 3021
    Found 16-bit shifter rotate left for signal <A_itrn[15]_X_itrn[30]_rotate_left_19_OUT> created at line 3012
    Found 16-bit shifter arithmetic right for signal <A_itrn[15]_X_itrn[30]_shift_right_21_OUT> created at line 2982
    Found 16-bit shifter logical left for signal <A_itrn[15]_X_itrn[30]_shift_left_23_OUT> created at line 2973
    Found 16-bit 13-to-1 multiplexer for signal <O_itrn> created at line 42.
    Found 16-bit comparator greater for signal <flags<3>> created at line 99
    Found 16-bit comparator greater for signal <flags<4>> created at line 100
    Summary:
         inferred   1 Adder/Subtractor(s).
         inferred   2 Comparator(s).
         inferred  16 Multiplexer(s).
         inferred   4 Combinational logic shifter(s).
Unit <ALU_param> synthesized.

Synthesizing Unit <regbank>.
    Related source file is "E:\University\_Second Year\Computer Architectures\assesment\MyMicroProccessor\Lab_2\DataPaths -
RegBankAlt\otherRegBank.vhd".
    Found 16-bit register for signal <REG02>.
    Found 16-bit register for signal <REG03>.
    Found 16-bit register for signal <REG04>.
    Found 16-bit register for signal <REG05>.
    Found 16-bit register for signal <REG06>.
    Found 16-bit register for signal <REG07>.
    Found 16-bit register for signal <REG08>.
    Found 16-bit register for signal <REG09>.
    Found 16-bit register for signal <REG10>.
    Found 16-bit register for signal <REG11>.
    Found 16-bit register for signal <REG12>.
    Found 16-bit register for signal <REG13>.
    Found 16-bit register for signal <REG14>.
    Found 16-bit register for signal <REG15>.
    Found 16-bit register for signal <REG16>.
    Found 16-bit register for signal <REG17>.
    Found 16-bit register for signal <REG18>.
    Found 16-bit register for signal <REG19>.
    Found 16-bit register for signal <REG20>.
    Found 16-bit register for signal <REG21>.
    Found 16-bit register for signal <REG22>.
    Found 16-bit register for signal <REG23>.
    Found 16-bit register for signal <REG24>.
    Found 16-bit register for signal <REG25>.
    Found 16-bit register for signal <REG26>.
    Found 16-bit register for signal <REG27>.
    Found 16-bit register for signal <REG28>.
    Found 16-bit register for signal <REG29>.
    Found 16-bit register for signal <REG30>.
    Found 16-bit register for signal <REG31>.
    Found 16-bit register for signal <REG01>.
    Found 16-bit 32-to-1 multiplexer for signal <A> created at line 30.
    Found 16-bit 32-to-1 multiplexer for signal <B> created at line 31.
    Summary:
         inferred 496 D-type flip-flop(s).
         inferred   2 Multiplexer(s).
Unit <regbank> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 1
 16-bit addsub                                        : 1
# Registers                                            : 31
 16-bit register                                      : 31
# Comparators                                          : 2
 16-bit comparator greater                            : 2
# Multiplexers                                         : 21
 1-bit 2-to-1 multiplexer                             : 6
 16-bit 2-to-1 multiplexer                            : 13
 16-bit 32-to-1 multiplexer                           : 2
# Logic shifters                                       : 4
 16-bit shifter arithmetic right                      : 1
 16-bit shifter logical left                          : 1
 16-bit shifter rotate left                           : 1
 16-bit shifter rotate right                          : 1
# Xors                                                 : 1
 16-bit xor2                                          : 1

=========================================================================
```

Architecture C

DataPathC.vhd

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

entity DataPath_C is
	GENERIC(
		data_size : natural := 16;
		num_registers : natural := 32
	);
	Port (
		clk : in  STD_LOGIC;
		rst : in STD_LOGIC;
		en : in STD_LOGIC;

		-- Inputs
		R_A : in  STD_LOGIC_VECTOR (log2(num_registers)-1 downto 0); -- Read address A
		R_B : in  STD_LOGIC_VECTOR (log2(num_registers)-1 downto 0); -- Read address B
		W_EN : in  STD_LOGIC;                                        -- Register write enable
		W_A : in  STD_LOGIC_VECTOR (log2(num_registers)-1 downto 0); -- Write address

		IMM : in  STD_LOGIC_VECTOR (data_size-1 downto 0);           -- Itermediate value
		M_A : in  STD_LOGIC_VECTOR (data_size-1 downto 0);           -- Memory address
		M_in : in  STD_LOGIC_VECTOR (data_size-1 downto 0);          -- Memory input (read)
		PC : in  STD_LOGIC_VECTOR (15 downto 0);                     -- Current Program Counter
		S : in  STD_LOGIC_VECTOR (4 downto 1);                       -- Selector control
		AL : in  STD_LOGIC_VECTOR (3 downto 0);                      -- ALU control
		SH : in  STD_LOGIC_VECTOR (log2(data_size)-1  downto 0);     -- Shift amount

		-- Outputs
		PC_plus : out  STD_LOGIC_VECTOR (15 downto 0);               -- Next Program Counter
		Flags : out  STD_LOGIC_VECTOR (7 downto 0);                  -- ALU flags
		M_DA : out  STD_LOGIC_VECTOR (data_size-1 downto 0);         -- Memory address
		M_out : out  STD_LOGIC_VECTOR (data_size-1 downto 0)         -- Memory output (write)
	);
end DataPath_C;

architecture Behavioral of DataPath_C is

	-- Data to write to the registers
	signal reg_in : STD_LOGIC_VECTOR (data_size-1 downto 0);

	-- Outputs of the registers
	signal A_data : STD_LOGIC_VECTOR (data_size-1 downto 0);
	signal B_data : STD_LOGIC_VECTOR (data_size-1 downto 0);

	-- Output of the register on the A and B buses
	signal A_reg_out : STD_LOGIC_VECTOR (data_size-1 downto 0);
	signal B_reg_out : STD_LOGIC_VECTOR (data_size-1 downto 0);

	-- The Outputs of the two Muxes on the input to the ALU
	signal A_mux : STD_LOGIC_VECTOR (data_size-1 downto 0);
	signal B_mux : STD_LOGIC_VECTOR (data_size-1 downto 0);

	-- The output of the combined ALU and shifter
	signal ALU_out : STD_LOGIC_VECTOR (data_size-1 downto 0);
	signal ALU_reg_out : STD_LOGIC_VECTOR (data_size-1 downto 0);

	-- The output of the memory in register
	signal M_in_reg_out : STD_LOGIC_VECTOR (data_size-1 downto 0);

begin

	-- The two multiplexers on the input to the ALU
	A_mux <= A_reg_out when S(2) = '0' else PC;
	B_mux <= B_reg_out when S(1) = '0' else IMM;

	-- The address multiplexer for the memory
	M_DA <= M_A when s(3) = '1' else ALU_reg_out;

	M_out <= B_reg_out;
	PC_plus <= ALU_reg_out;
```

(Continued.)

```
        -- The register write multiplexer
        reg_in <= ALU_reg_out when s(4) = '0' else M_in_reg_out;

        -- The register on the A bus
        A_reg: entity work.Reg Generic Map (data_size => data_size)
        PORT MAP(
                clk => clk,
                rst => rst,
                en => en,
                data_in => A_data,
                data_out => A_reg_out
        );

        -- The register on the B bus
        B_reg: entity work.Reg Generic Map (data_size => data_size)
        PORT MAP(
                clk => clk,
                rst => rst,
                en => en,
                data_in => B_data,
                data_out => B_reg_out
        );

        -- The Register on the out put of the ALU
        ALU_reg: entity work.Reg Generic Map (data_size => data_size)
        PORT MAP(
                clk => clk,
                rst => rst,
                en => en,
                data_in => ALU_out,
                data_out => ALU_reg_out
        );

        -- The register on the memory read bus
        M_in_reg: entity work.Reg Generic Map (data_size => data_size)
        PORT MAP(
                clk => clk,
                rst => rst,
                en => en,
                data_in => M_in,
                data_out => M_in_reg_out
        );

        -- The ALU and shifter from Lab 1
        ALU: entity work.ALU_param GENERIC MAP( N => data_size )
        PORT MAP(
                A => A_mux,
                B => B_mux,
                X => SH,
                ctrl => AL,
                O => ALU_out,
                flags => flags
        );

        -- The register bank
        Registers: entity work.regbank
        PORT MAP(
                RSELA => R_A,
                RSELB => R_B,
                WSEL => W_A,
                D => reg_in,
                WEN => W_EN,
                clk => clk,
                A => A_data,
                B => B_data,
                rst => '0'
        );

end Behavioral;
```

Test Data for DataPath C

Here is the control signals and test data fro datapath C

Test Command	Stage	R_A [4:0]	R_B [4:0]	W_EN	W_A [4:0]	IMM [15:0]	M_A [15:0]	M_in [15:0]	[15:0]	S [4:1]	AL [3:0]	SH [3:0]	[15:0]	Flags [7:0]	M_DA [15:0]	M_out [15:0]	OEN
inc R1, R0	Fetch	ØØØØØ	ØØØØØ	0	ØØØØØ	0xØØØØ	0xØØØØ	0xØØØØ	00FF	ØØ1Ø	1000	ØØØØ	ØØØØ	01010010	0xØØØØ	0xØØØØ	0
	RegRd	000000	ØØØØØ	0	ØØØØØ	0xØØØØ	0xØØØØ	0xØØØØ	00FF	ØØØØ	ØØØØ	ØØØØ	0100	ØØØØØØ	0xØØØØ	0xØØØØ	0
	ALU	ØØØØØ	ØØØØØ	0	ØØØØØ	0xØØØØ	0xØØØØ	0xØØØØ	00FF	ØØØØ	1000	ØØØØ	ØØØØ	01010110	0xØØØØ	0xØØØØ	0
	RegWr	ØØØØØ	ØØØØØ	1	00001	0xØØØØ	0xØØØØ	0xØØØØ	00FF	0ØØØ	ØØØØ	ØØØØ	ØØØØ	ØØØØØØ	0xØØØØ	0xØØØØ	0
addi R2, R0, 0x0005	Fetch	ØØØØØ	ØØØØØ	0	ØØØØØ	0xØØØØ	0xØØØØ	0xØØØØ	0100	ØØ1Ø	1000	ØØØØ	ØØØØ	01010010	0xØØØØ	0xØØØØ	0
	RegRd	000000	ØØØØØ	0	ØØØØØ	0xØØØØ	0xØØØØ	0xØØØØ	0100	ØØØØ	ØØØØ	ØØØØ	0101	ØØØØØØ	0xØØØØ	0xØØØØ	0
	ALU	ØØØØØ	ØØØØØ	0	ØØØØØ	0x0005	0xØØØØ	0xØØØØ	0100	ØØ01	1010	ØØØØ	ØØØØ	01010010	0xØØØØ	0xØØØØ	0
	RegWr	ØØØØØ	ØØØØØ	1	00010	0xØØØØ	0xØØØØ	0xØØØØ	0100	0ØØØ	ØØØØ	ØØØØ	ØØØØ	ØØØØØØ	0xØØØØ	0xØØØØ	0
shl R3, R1, 3	Fetch	ØØØØØ	ØØØØØ	0	ØØØØØ	0xØØØØ	0xØØØØ	0xØØØØ	0101	ØØ1Ø	1000	ØØØØ	ØØØØ	01010010	0xØØØØ	0xØØØØ	0
	RegRd	00001	ØØØØØ	0	ØØØØØ	0xØØØØ	0xØØØØ	0xØØØØ	0101	ØØØ	ØØØØ	ØØØØ	0102	ØØØØØØ	0xØØØØ	0xØØØØ	0
	ALU	ØØØØ	ØØØØØ	0	ØØØØØ	0xØØØØ	0xØØØØ	0xØØØØ	0101	ØØ0Ø	1100	0011	ØØØØ	01010010	0xØØØØ	0xØØØØ	0
	RegWr	ØØØØ	ØØØØØ	1	00011	0xØØØØ	0xØØØØ	0xØØØØ	0101	0ØØØ	ØØØØ	ØØØØ	ØØØØ	ØØØØØØ	0xØØØØ	0xØØØØ	0
storr R2, R3	Fetch	ØØØØØ	ØØØØØ	0	ØØØØØ	0xØØØØ	0xØØØØ	0xØØØØ	0102	ØØ1Ø	1000	ØØØØ	ØØØØ	01010010	0xØØØØ	0xØØØØ	0
	RegRd	00011	ØØØØØ	0	ØØØØØ	0xØØØØ	0xØØØØ	0xØØØØ	0102	ØØØØ	ØØØØ	ØØØØ	0103	ØØØØØØ	0xØØØØ	0xØØØØ	0
	ALU	ØØØØØ	00010	0	ØØØØØ	0xØØØØ	0xØØØØ	0xØØØØ	0102	ØØ00	0000	ØØØØ	ØØØØ	01010010	0xØØØØ	0xØØØØ	0
	Mem Acc	ØØØØØ	ØØØØØ	0	ØØØØØ	0xØØØØ	0xØØØØ	0xØØØØ	0102	ØØØØ	ØØØØ	ØØØØ	ØØØØ	ØØØØØØ	0x0008	0x0005	1
loadi R5, 1f1f	Fetch	ØØØØØ	ØØØØØ	0	ØØØØØ	0xØØØØ	0xØØØØ	0xØØØØ	0103	ØØ1Ø	1000	ØØØØ	ØØØØ	01010010	0xØØØØ	0xØØØØ	0
	Mem Acc	ØØØØØ	ØØØØØ	0	ØØØØØ	0xØØØØ	1F1F	CCCC	0103	Ø1ØØ	ØØØØ	ØØØØ	0104	ØØØØØØ	0x1f1f	0xØØØØ	0
	RegWrite	ØØØØØ	ØØØØØ	1	00101	0xØØØØ	0xØØØØ	0xØØØØ	0103	1ØØØ	ØØØØ	ØØØØ	ØØØØ	ØØØØØØ	0xØØØØ	0xØØØØ	0
brneq R2, 010F	Fetch	ØØØØØ	ØØØØØ	0	ØØØØØ	0xØØØØ	0xØØØØ	0xØØØØ	0104	ØØ1Ø	1000	ØØØØ	ØØØØ	01010010	0xØØØØ	0xØØØØ	0
	RegRd	00010	00000	0	ØØØØØ	0x010F	0xØØØØ	0xØØØØ	0104	ØØ11	1010	ØØØØ	0105	01010010	0xØØØØ	0xØØØØ	0
	ALU	ØØØØØ	ØØØØØ	0	ØØØØØ	0xØØØØ	0xØØØØ	0xØØØØ	0104	ØØ00	1010	ØØØØ	0213	---------10	0xØØØØ	0xØØØØ	0

DataPathC-TB.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.DigEng.ALL;
USE work.easyprint.ALL;
USE ieee.numeric_std.ALL;

ENTITY DataPath_C_TB IS
END DataPath_C_TB;

ARCHITECTURE behavior OF DataPath_C_TB IS

    -- Constants
        constant data_size : NATURAL := 16;
        constant num_registers : NATURAL := 32;

        -- Clock period definitions
        constant clk_period : time := 10 ns;

        -- Component Declaration for the Unit Under Test (UUT)
        COMPONENT DataPath_C
                GENERIC(
                        data_size : natural;
                        num_registers : natural
                );
                PORT(
                        clk     : IN    std_logic;
                        rst     : IN    std_logic;
                        en      : IN    std_logic;

                        R_A     : IN    std_logic_vector        (log2(num_registers)-1 downto 0);
                        R_B     : IN    std_logic_vector        (log2(num_registers)-1 downto 0);
                        W_EN    : IN    std_logic;
                        W_A     : IN    std_logic_vector        (log2(num_registers)-1 downto 0);
                        IMM     : IN    std_logic_vector        (data_size-1 downto 0);
                        M_A     : IN    std_logic_vector        (data_size-1 downto 0);
                        M_in    : IN    std_logic_vector        (data_size-1 downto 0);
                        PC              : IN    std_logic_vector        (15 downto 0);
                        S               : IN    std_logic_vector        (4 downto 1);
                        AL              : IN    std_logic_vector        (3 downto 0);
                        SH              : IN    std_logic_vector        (log2(data_size)-1 downto 0);

                        PC_plus : OUT   std_logic_vector        (15 downto 0);
                        Flags   : OUT   std_logic_vector        (7 downto 0);
                        M_DA    : OUT   std_logic_vector        (data_size-1 downto 0);
                        M_out   : OUT   std_logic_vector        (data_size-1 downto 0)
                );
    END COMPONENT;


        --Inputs
        signal clk      : std_logic                                                                             := '0';
        signal rst      : std_logic                                                                             := '1';
        signal en       : std_logic                                                                             := '0';

        signal R_A      : std_logic_vector      (log2(num_registers)-1 downto 0)        := (others => '0');
        signal R_B      : std_logic_vector      (log2(num_registers)-1 downto 0)        := (others => '0');
        signal W_EN     : std_logic                                                                             := '0';
        signal W_A      : std_logic_vector      (log2(num_registers)-1 downto 0 )       := (others => '0');
        signal IMM      : std_logic_vector      (data_size-1 downto 0)                          := (others => '0');
        signal M_A      : std_logic_vector      (data_size-1 downto 0)                          := (others => '0');
        signal M_in : std_logic_vector  (data_size-1 downto 0)                          := (others => '0');
        signal PC       : std_logic_vector      (15 downto 0)                                           := (others => '0');
        signal S        : std_logic_vector      (4 downto 1)                                            := (others => '0');
        signal AL       : std_logic_vector      (3 downto 0)                                            := (others => '0');
        signal SH       : std_logic_vector      (log2(data_size)-1 downto 0)            := (others => '0');

        --Outputs
        signal PC_plus  : std_logic_vector      (15 downto 0);
        signal Flags    : std_logic_vector      (7 downto 0);
        signal M_DA     : std_logic_vector      (data_size-1 downto 0);
        signal M_out    : std_logic_vector      (data_size-1 downto 0);
        signal OEN : std_logic := '0';

        -- Test data definitions
        type TEST_VECTOR is RECORD
                 R_A    : STD_LOGIC_VECTOR(log2(num_registers)-1 downto 0);
                 R_B    : std_logic_vector(log2(num_registers)-1 downto 0);
```

(Continued.)

```
            W_EN    : std_logic;
            W_A     : std_logic_vector(log2(num_registers)-1 downto 0);

            IMM     : std_logic_vector(data_size-1 downto 0);
            M_A     : std_logic_vector(data_size-1 downto 0);
            M_in    : std_logic_vector(data_size-1 downto 0);
            PC      : std_logic_vector(15 downto 0);
            S               : std_logic_vector(4 downto 1);
            AL      : std_logic_vector(3 downto 0);
            SH      : std_logic_vector(log2(data_size)-1 downto 0) ;

            PC_plus: std_logic_vector(15 downto 0);
            flags   : std_logic_vector(7 downto 0);
            M_DA    : std_logic_vector(data_size-1 downto 0);
            M_out   : std_logic_vector(data_size-1 downto 0);

            OEN : std_logic;
    end RECORD;
    type TEST_VECTOR_ARRAY is ARRAY(NATURAL RANGE <>) of TEST_VECTOR;

    -- Test Data
    constant test_vectors : TEST_VECTOR_ARRAY := (
            --R_A,          R_B,            W_EN,   W_A,            IMM,                            M_A,
            M_in,                           PC,                     S,              AL,             SH,
            PC_plus,                        flags,                  M_DA,                           M_out,
            OEN --

            -- inc R1, R0
            ( "-----",      "-----",        '0',    "-----",        "----------------",              "----------------",
            "----------------",             X"00FF",                "--1-",         "1000",         "----",
            "----------------",             "01010010",             "----------------",             "----------------",
            '0' ),

            ( "00000",      "-----",        '0',    "-----",        "----------------",              "----------------",
            "----------------",             X"00FF",                "----",         "----",         "----",
            X"0100",                        "--------",             "----------------",             "----------------",
            '0' ),

            ( "-----",      "-----",        '0',    "-----",        "----------------",              "----------------",
            "----------------",             X"00FF",                "--0-",         "1000",         "----",
            "----------------",             "01010110",             "----------------",             "----------------",
            '0' ),

            ( "-----",      "-----",        '1',    "00001",        "----------------",              "----------------",
            "----------------",             X"00FF",                "0---",         "----",         "----",
            "----------------",             "--------",             "----------------",             "----------------",
            '0' ),

            -- addi R2, R0, 0x0005
            ( "-----",      "-----",        '0',    "-----",        "----------------",              "----------------",
            "----------------",             X"0100",                "--1-",         "1000",         "----",
            "----------------",             "01010010",             "----------------",             "----------------",
            '0' ),
            ( "00000",      "-----",        '0',    "-----",        "----------------",              "----------------",
            "----------------",             X"0100",                "----",         "----",         "----",
            X"0101",                        "--------",             "----------------",             "----------------",
            '0' ),

            ( "-----",      "-----",        '0',    "-----",        X"0005",                        "----------------",
            "----------------",             X"0100",                "--01",         "1010",         "----",
            "----------------",             "01010010",             "----------------",             "----------------",
            '0' ),
            ( "-----",      "-----",        '1',    "00010",        "----------------",              "----------------",
            "----------------",             X"0100",                "0---",         "----",         "----",
            "----------------",             "--------",             "----------------",             "----------------",
            '0' ),

            -- shl R3, R1, 3
            ( "-----",      "-----",        '0',    "-----",        "----------------",              "----------------",
            "----------------",             X"0101",                "--1-",         "1000",         "----",
            "----------------",             "01010010",             "----------------",             "----------------",
            '0' ),

            ( "00001",      "-----",        '0',    "-----",        "----------------",              "----------------",
            "----------------",             X"0101",                "----",         "----",         "----",
            X"0102",                        "--------",             "----------------",             "----------------",
            '0' ),
```

(Continued.)

```
        ( "-----",      "-----",      '0',    "-----",      "----------------",      "----------------",
        "----------------",           X"0101",              "--0-",         "1100",         "0011",
        "----------------",           "01010010",           "----------------",      "----------------",
        '0' ),

        ( "-----",      "-----",      '1',    "00011",      "----------------",      "----------------",
        "----------------",           X"0101",              "0---",         "----",         "----",
        "----------------",           "--------",           "----------------",      "----------------",
        '0' ),

        -- storr R2, R3
        ( "-----",      "-----",      '0',    "-----",      "----------------",      "----------------",
        "----------------",           X"0102",              "--1-",         "1000",         "----",
        "----------------",           "01010010",           "----------------",      "----------------",
        '0' ),

        ( "00011",      "-----",      '0',    "-----",      "----------------",      "----------------",
        "----------------",           X"0102",              "----",         "----",         "----",
        X"0103",                      "--------",           "----------------",      "----------------",
        '0' ),

        ( "-----",      "00010",      '0',    "-----",      "----------------",      "----------------",
        "----------------",           X"0102",              "--00",         "0000",         "----",
        "----------------",           "01010010",           "----------------",      "----------------",
        '0' ),

        ( "-----",      "-----",      '0',    "-----",      "----------------",      "----------------",
        "----------------",           X"0102",              "-0--",         "----",         "----",
        "----------------",           "--------",           X"0008",                      X"0005",
        '1' ),

        -- loadi R5, 1f1f
        ( "-----",      "-----",      '0',    "-----",      "----------------",      "----------------",
        "----------------",           X"0103",              "--1-",         "1000",         "----",
        "----------------",           "01010010",           "----------------",      "----------------",
        '0' ),

        ( "-----",      "-----",      '0',    "-----",      "----------------",      X"1F1F",
        X"CCCC",                      X"0103",              "-1--",         "----",         "----",
        X"0104",                      "--------",           X"1f1f",                      "----------------",
        '0' ),

        ( "-----",      "-----",      '1',    "00101",      "----------------",      "----------------",
        "----------------",           X"0103",              "1---",         "----",         "----",
        "----------------",           "--------",           "----------------",      "----------------",
        '0' ),

        -- brneq R2, 010F
        ( "-----",      "-----",      '0',    "-----",      "----------------",      "----------------",
        "----------------",           X"0104",              "--1-",         "1000",         "----",
        "----------------",           "01010010",           "----------------",      "----------------",
        '0' ),

        ( "00010",      "00000",      '0',    "-----",      X"010f",                      "----------------",
        "----------------",           X"0104",              "--11",         "1010",         "----",
        X"0105",                      "01010010",           "----------------",      "----------------",
        '0' ),

        ( "-----",      "-----",      '0',    "-----",      "----------------",      "----------------",
        "----------------",           X"0104",              "--00", "       1010",         "----",
        X"0213",                      "------10",           "----------------",      "----------------",
        '0' )
    );
```

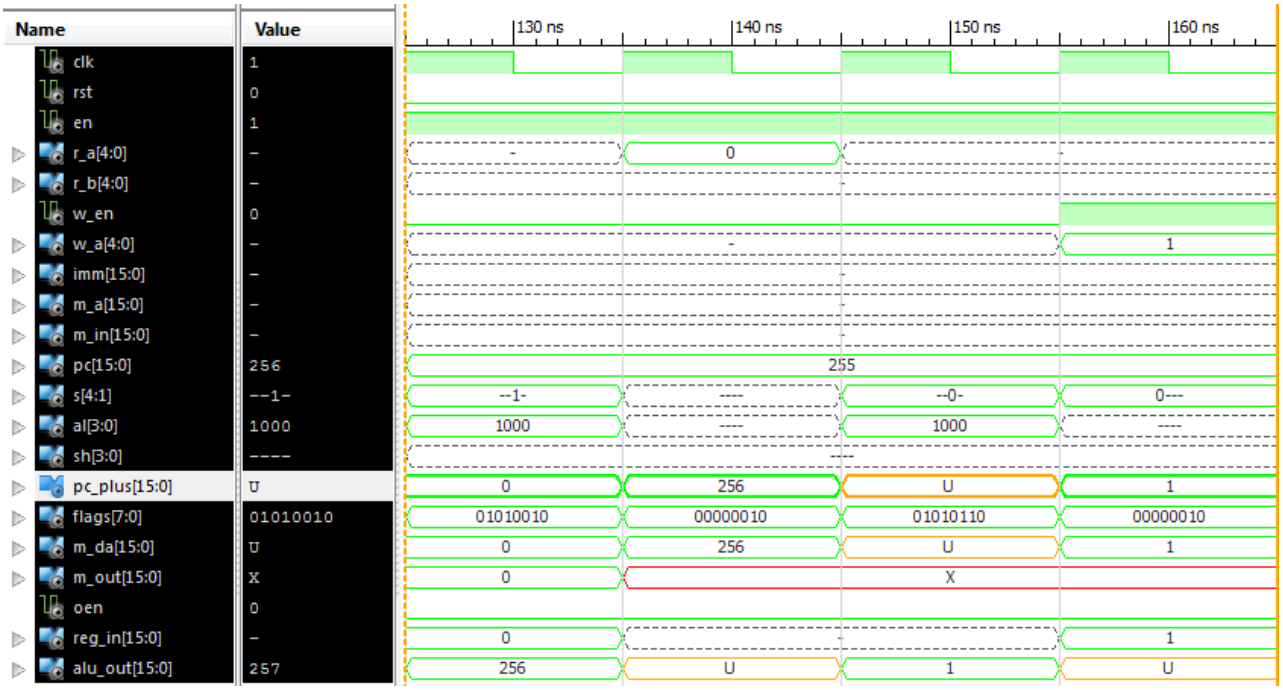
(Continued.)

```
BEGIN

        -- Instantiate the Unit Under Test (UUT)
        uut: DataPath_C
        Generic Map(
                data_size => data_size,
                num_registers => num_registers
        )
        PORT MAP (
                clk => clk,
                rst => rst,
                en => en,

                -- Inputs
                R_A => R_A,
                R_B => R_B,

                W_EN => W_EN,
                W_A => W_A,

                IMM => IMM,
                M_A => M_A,
                M_in => M_in,
                PC => PC,
                S => S,
                AL => AL,
                SH => SH,

                -- Outputs
                PC_plus => PC_plus,
                Flags => Flags,
                M_DA => M_DA,
                M_out => M_out
        );

   -- Clock process definitions
   clk_process :process
   begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin

                -- hold reset state for 100 ns.
                wait for 100 ns;

                -- enable the system
                rst <= '0';
                en <= '1';
                wait for 2*clk_period;

                -- run the test for every set of data
                for i in test_vectors'range loop

                        wait until rising_edge(clk);

                        -- assign test inputs
                        R_A <= test_vectors(i).R_A;
                        R_B <= test_vectors(i).R_B;

                        W_EN <= test_vectors(i).W_EN;
                        W_A <= test_vectors(i).W_A;

                        IMM <= test_vectors(i).IMM;
                        M_A <= test_vectors(i).M_A;
                        M_in <= test_vectors(i).M_in;
                        PC <= test_vectors(i).PC;
                        S <= test_vectors(i).S;
                        AL <= test_vectors(i).AL;
                        SH <= test_vectors(i).SH;

                        OEN <= test_vectors(i).OEN;
```

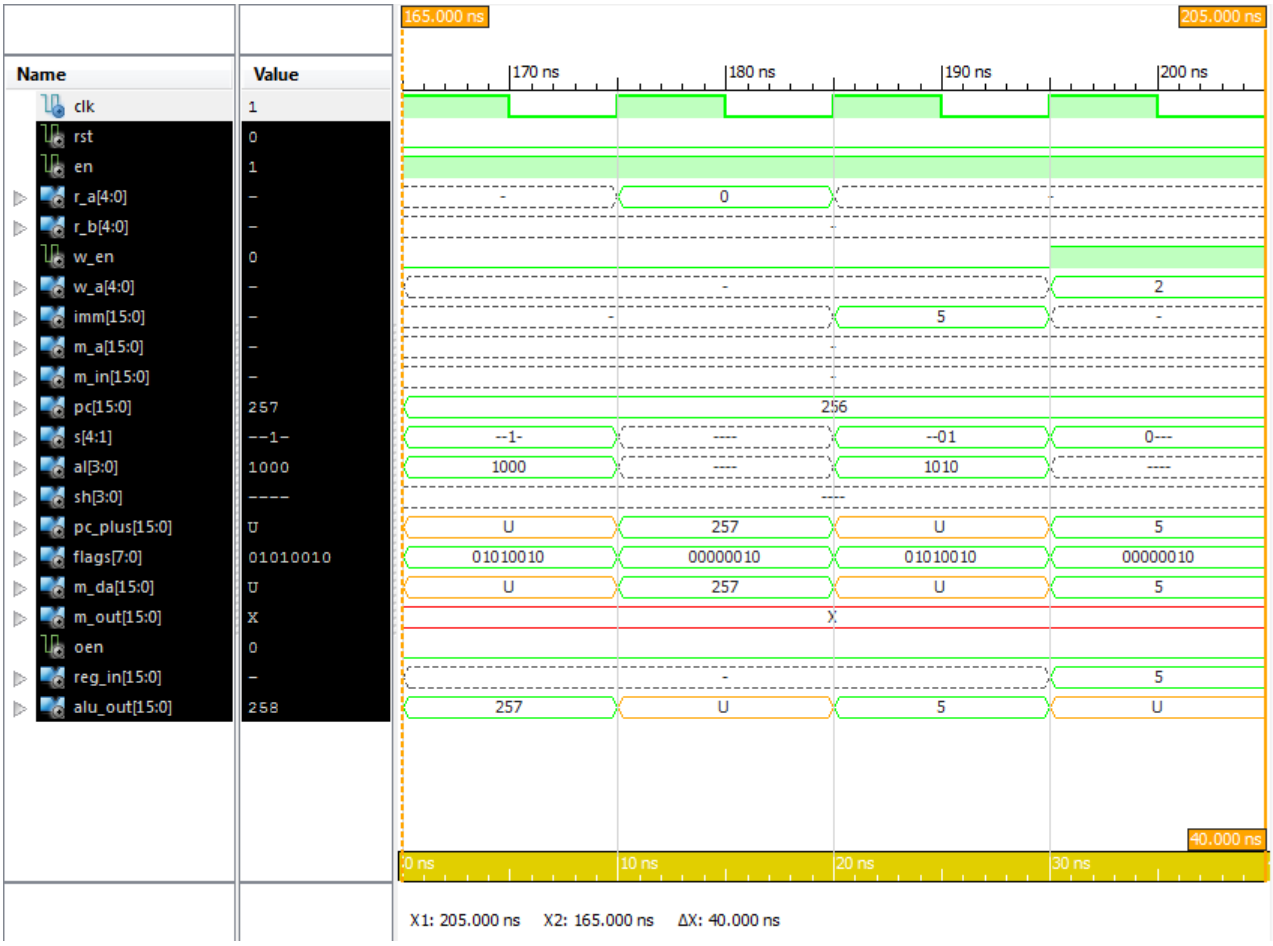
(Continued.)

```
                        wait until falling_edge(clk);

                        -- Check to see if the out put was what we were expecting
                        -- Have to use std_match() instead of = when comparing meta values like '-'
                        assert std_match(test_vectors(i).flags, flags)
                        report lf & " [ERR!] Test " & integer'image(i)&
                                " Actual flags did not equal expected flags."&
                                " Actual [ " & to_bstring(flags) & " ]" &
                                " Expected [ " & to_bstring(test_vectors(i).flags) & " ]" & lf
                        severity error;

                        assert std_match(test_vectors(i).M_out, M_out)
                        report lf & " [ERR!] Test " & integer'image(i)&
                                " Actual value to memory did not equal expected value to memory."&
                                " Actual [ " & u_tostr(M_out) & " ]" &
                                " Expected [ " & u_tostr(test_vectors(i).M_out) & " ]" & lf
                        severity error;

                        assert std_match(M_DA , test_vectors(i).M_DA)
                        report lf & " [ERR!] Test " & integer'image(i)&
                                " Actual memory address did not equal expected memory address."&
                                " Actual [ " & u_tostr(M_DA) & " ]" &
                                " Expected [ " & u_tostr(test_vectors(i).M_DA) & " ]" & lf
                        severity error;

                        assert std_match(PC_plus, test_vectors(i).PC_plus)
                        report lf & " [ERR!] Test " & integer'image(i)&
                                " Actual program counter did not equal expected program counter."&
                                " Actual [ " & u_tostr(PC_plus) & " ]" &
                                " Expected [ " & u_tostr(test_vectors(i).PC_plus) & " ]" & lf
                        severity error;

                        -- if there were no isses report that the test was successful
                        assert not (
                                std_match(flags, test_vectors(i).flags) and
                                std_match(M_out, test_vectors(i).M_out) and
                                std_match(M_DA, test_vectors(i).M_DA) and
                                std_match(PC_plus, test_vectors(i).PC_plus)
                        )
                        report lf & " [ OK ] Test " & integer'image(i)& " was successful!" & lf
                        severity note;

                end loop;

                -- End of test
                wait;
        end process;


END;
```

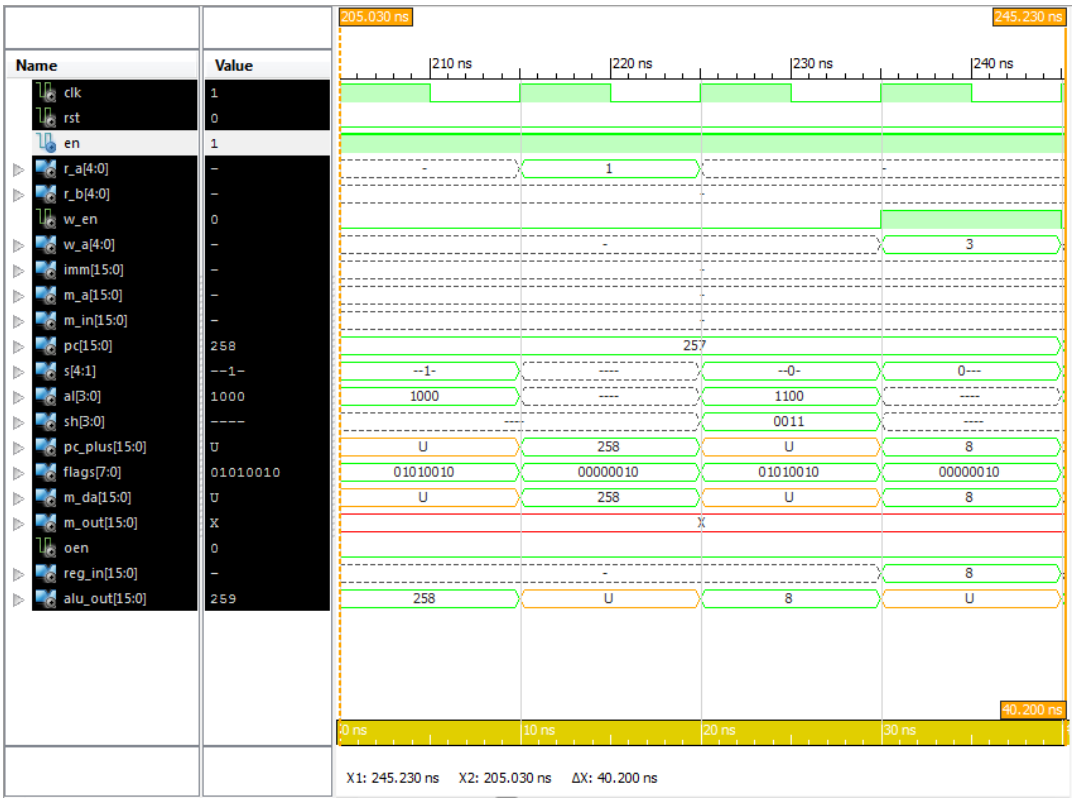
Simulations



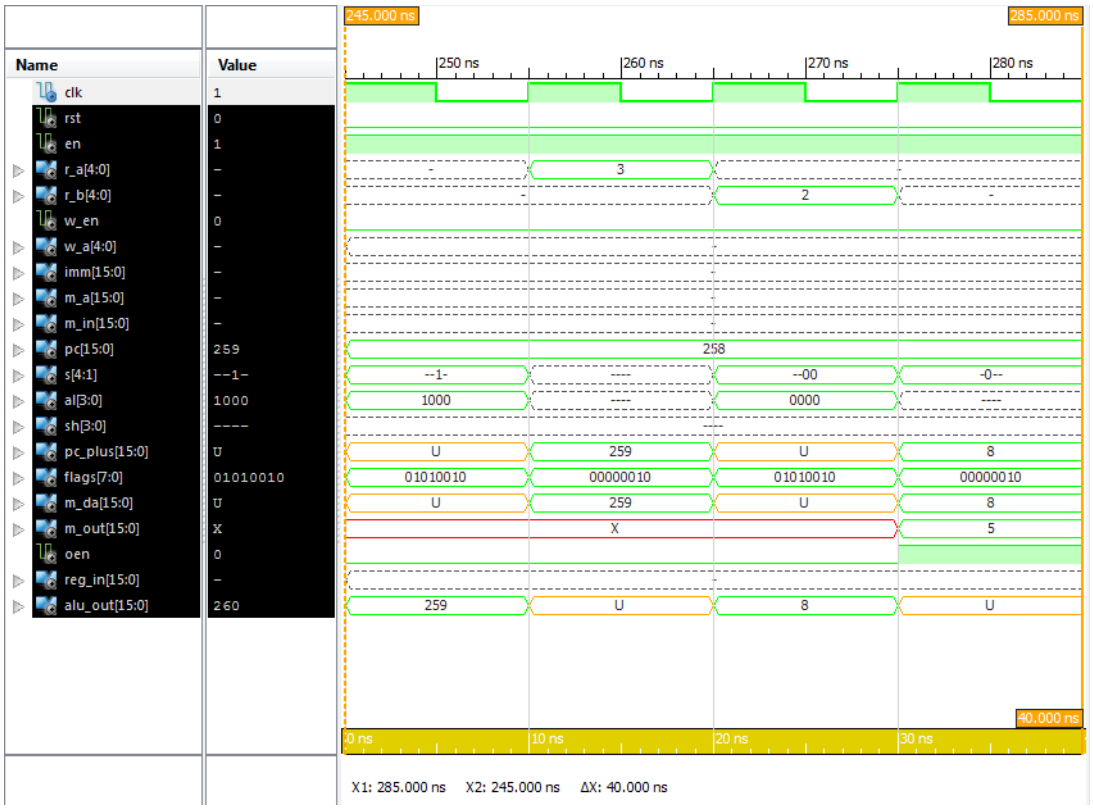
Screenshot showing the 'inc' instruction



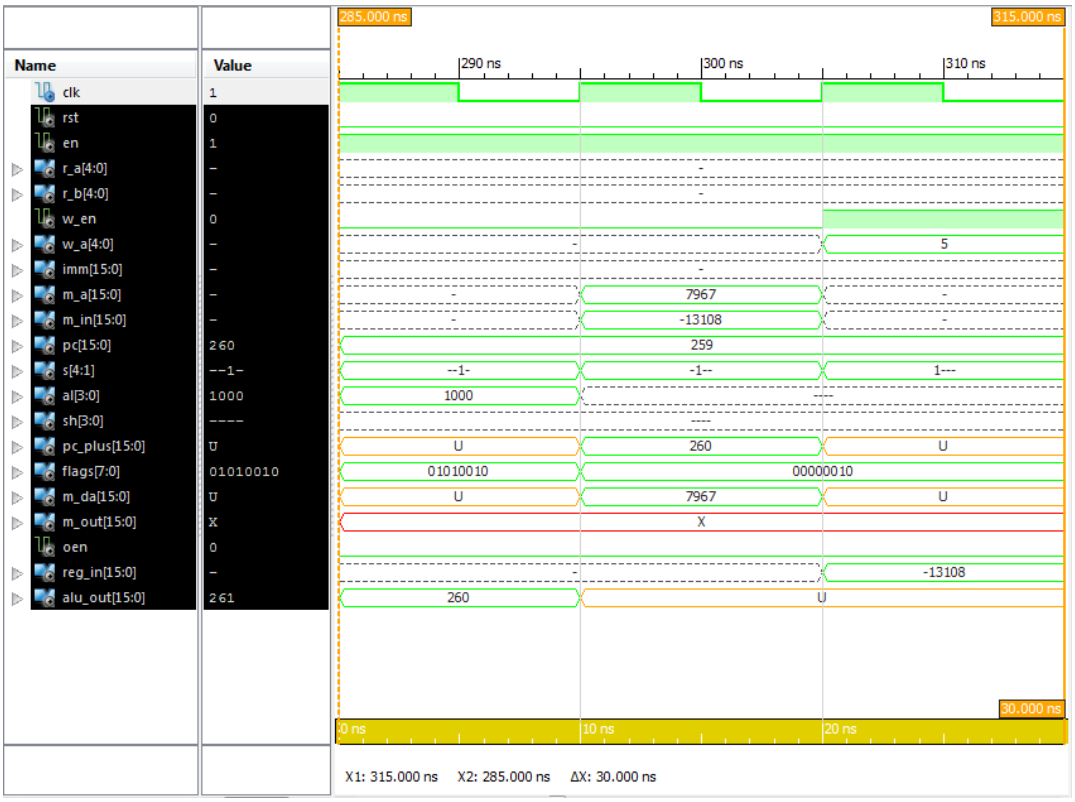
Screenshot showing the 'addi' instruction



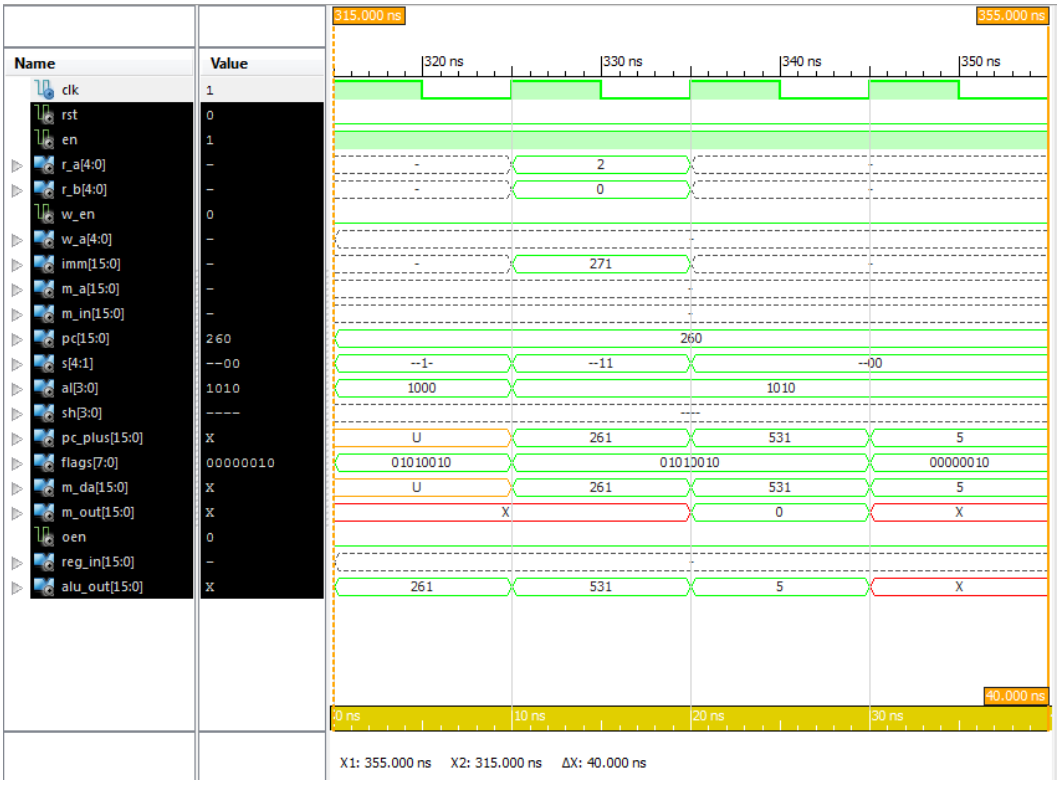
Screenshot showing the 'shl' instruction



Screenshot showing the 'storr' instruction



Screenshot showing the 'loadi' instruction



Screenshot showing the 'brneq' instruction

iSim Console Output

```
ISim P.28xd (signature 0xa0883be4)
This is a Full version of ISim.
Time resolution is 1 ps
WARNING: Simulation object /datapath_c_tb/test_vectors was not traceable in the design for the following reason:
ISim does not yet support tracing of constant and generic multi-dimensional arrays.
Simulator is doing circuit initialization process.
at 0 ps, Instance /datapath_c_tb/uut/ALU/ : Warning: NUMERIC_STD.">=": metavalue detected, returning FALSE
at 0 ps, Instance /datapath_c_tb/uut/ALU/ : Warning: NUMERIC_STD."<=": metavalue detected, returning FALSE
at 0 ps, Instance /datapath_c_tb/uut/ALU/ : Warning: NUMERIC_STD.">": metavalue detected, returning FALSE
at 0 ps, Instance /datapath_c_tb/uut/ALU/ : Warning: NUMERIC_STD."<": metavalue detected, returning FALSE
at 0 ps, Instance /datapath_c_tb/uut/ALU/ : Warning: NUMERIC_STD."=": metavalue detected, returning FALSE
at 0 ps, Instance /datapath_c_tb/uut/ALU/ : Warning: NUMERIC_STD."/=": metavalue detected, returning TRUE
at 0 ps, Instance /datapath_c_tb/uut/ALU/ : Warning: NUMERIC_STD."=": metavalue detected, returning FALSE
Finished circuit initialization process.

at 130 ns(1): Note: [ OK ] Test 0 was successful! (/datapath_c_tb/).
at 140 ns(1): Note: [ OK ] Test 1 was successful! (/datapath_c_tb/).
at 150 ns(1): Note: [ OK ] Test 2 was successful! (/datapath_c_tb/).
at 160 ns(1): Note: [ OK ] Test 3 was successful! (/datapath_c_tb/).
at 170 ns(1): Note: [ OK ] Test 4 was successful! (/datapath_c_tb/).
at 180 ns(1): Note: [ OK ] Test 5 was successful! (/datapath_c_tb/).
at 190 ns(1): Note: [ OK ] Test 6 was successful! (/datapath_c_tb/).
at 200 ns(1): Note: [ OK ] Test 7 was successful!(/datapath_c_tb/).
at 210 ns(1): Note: [ OK ] Test 8 was successful!(/datapath_c_tb/).
at 220 ns(1): Note: [ OK ] Test 9 was successful! (/datapath_c_tb/).
at 230 ns(1): Note: [ OK ] Test 10 was successful! (/datapath_c_tb/).
at 240 ns(1): Note: [ OK ] Test 11 was successful! (/datapath_c_tb/).
at 250 ns(1): Note: [ OK ] Test 12 was successful! (/datapath_c_tb/).
at 260 ns(1): Note: [ OK ] Test 13 was successful! (/datapath_c_tb/).
at 270 ns(1): Note: [ OK ] Test 14 was successful! (/datapath_c_tb/).
at 280 ns(1): Note: [ OK ] Test 15 was successful! (/datapath_c_tb/).
at 290 ns(1): Note: [ OK ] Test 16 was successful! (/datapath_c_tb/).
at 300 ns(1): Note: [ OK ] Test 17 was successful! (/datapath_c_tb/).
at 310 ns(1): Note: [ OK ] Test 18 was successful! (/datapath_c_tb/).
at 320 ns(1): Note: [ OK ] Test 19 was successful! (/datapath_c_tb/).
at 330 ns(1): Note: [ OK ] Test 20 was successful! (/datapath_c_tb/).
at 340 ns(1): Note: [ OK ] Test 21 was successful! (/datapath_c_tb/).

ISim>
```

All but the first meta value warnings have been removed to improve readability.

HDL Synthesis

```
=========================================================================
*                           HDL Synthesis                               *
=========================================================================

Synthesizing Unit <DataPath_C>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Lab_2\DataPaths - RegBankAlt\DataPath_C.vhd".
        data_size = 16
        num_registers = 32
    Summary:
        inferred   4 Multiplexer(s).
Unit <DataPath_C> synthesized.

Synthesizing Unit <Reg>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Lab_2\DataPaths - RegBankAlt\Reg.vhd".
        data_size = 16
    Found 16-bit register for signal <data_out>.
    Summary:
        inferred  16 D-type flip-flop(s).
Unit <Reg> synthesized.

Synthesizing Unit <ALU_param>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Lab_2\DataPaths - RegBankAlt\ALU_param.vhd".
        N = 16
    Found 16-bit adder for signal <A_itrn[15]_B_itrn[15]_add_27_OUT> created at line 69.
    Found 16-bit adder for signal <A_itrn[15]_GND_7_o_add_31_OUT> created at line 1253.
    Found 16-bit subtractor for signal <A_itrn[15]_B_itrn[15]_sub_26_OUT<15:0>> created at line 70.
    Found 16-bit subtractor for signal <A_itrn[15]_GND_7_o_sub_30_OUT<15:0>> created at line 1320.
    Found 16-bit shifter rotate right for signal <A_itrn[15]_X_itrn[30]_rotate_right_17_OUT> created at line 3021
    Found 16-bit shifter rotate left for signal <A_itrn[15]_X_itrn[30]_rotate_left_19_OUT> created at line 3012
    Found 16-bit shifter arithmetic right for signal <A_itrn[15]_X_itrn[30]_shift_right_21_OUT> created at line 2982
```

```
    Found 16-bit shifter logical left for signal <A_itrn[15]_X_itrn[30]_shift_left_23_OUT> created at line 2973
    Found 16-bit 13-to-1 multiplexer for signal <O_itrn> created at line 42.
    Found 16-bit comparator greater for signal <flags<3>> created at line 99
    Found 16-bit comparator greater for signal <flags<4>> created at line 100
    Summary:
        inferred   1 Adder/Subtractor(s).
        inferred   2 Comparator(s).
        inferred  16 Multiplexer(s).
        inferred   4 Combinational logic shifter(s).
Unit <ALU_param> synthesized.

Synthesizing Unit <regbank>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Lab_2\DataPaths - RegBankAlt\otherRegBank.vhd".
    Found 16-bit register for signal <REG02>.
    Found 16-bit register for signal <REG03>.
    Found 16-bit register for signal <REG04>.
    Found 16-bit register for signal <REG05>.
    Found 16-bit register for signal <REG06>.
    Found 16-bit register for signal <REG07>.
    Found 16-bit register for signal <REG08>.
    Found 16-bit register for signal <REG09>.
    Found 16-bit register for signal <REG10>.
    Found 16-bit register for signal <REG11>.
    Found 16-bit register for signal <REG12>.
    Found 16-bit register for signal <REG13>.
    Found 16-bit register for signal <REG14>.
    Found 16-bit register for signal <REG15>.
    Found 16-bit register for signal <REG16>.
    Found 16-bit register for signal <REG17>.
    Found 16-bit register for signal <REG18>.
    Found 16-bit register for signal <REG19>.
    Found 16-bit register for signal <REG20>.
    Found 16-bit register for signal <REG21>.
    Found 16-bit register for signal <REG22>.
    Found 16-bit register for signal <REG23>.
    Found 16-bit register for signal <REG24>.
    Found 16-bit register for signal <REG25>.
    Found 16-bit register for signal <REG26>.
    Found 16-bit register for signal <REG27>.
    Found 16-bit register for signal <REG28>.
    Found 16-bit register for signal <REG29>.
    Found 16-bit register for signal <REG30>.
    Found 16-bit register for signal <REG31>.
    Found 16-bit register for signal <REG01>.
    Found 16-bit 32-to-1 multiplexer for signal <A> created at line 30.
    Found 16-bit 32-to-1 multiplexer for signal <B> created at line 31.
    Summary:
        inferred 496 D-type flip-flop(s).
        inferred   2 Multiplexer(s).
Unit <regbank> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 1
 16-bit addsub                                        : 1
# Registers                                            : 35
 16-bit register                                      : 35
# Comparators                                          : 2
 16-bit comparator greater                            : 2
# Multiplexers                                         : 22
 1-bit 2-to-1 multiplexer                             : 6
 16-bit 2-to-1 multiplexer                            : 14
 16-bit 32-to-1 multiplexer                           : 2
# Logic shifters                                       : 4
 16-bit shifter arithmetic right                      : 1
 16-bit shifter logical left                          : 1
 16-bit shifter rotate left                           : 1
 16-bit shifter rotate right                          : 1
# Xors                                                 : 1
 16-bit xor2                                          : 1


=========================================================================
```

Test Data for Architecture C

Here is our control logic for Arch C as well as the expected outputs.

	Fetch	Reg Read				ALU+						Mem Acc						Reg Write		
Test Command		RA	RB	SEL (6)[7]	SEL (5)[6]	SEL (2)[3:2]	SEL (1)[1:0]	AL	IMM	SH	Flags	MA	M_in	SEL (3)[4]	M_DA	M_out	OEN	WA	W_EN	SEL (4)[5]
inc R1, R0		00000	øøøøø	0	ø	01	øø	1000	0xøøøø	øøøø	01010110	0xøøøø	0xøøøø	ø	0xøøøø	0xøøøø	0	00001	1	1
addi R2, R0, 0x0005		00000	øøøøø	0	ø	01	10	1010	0x0005	øøøø	01010110	0xøøøø	0xøøøø	ø	0xøøøø	0xøøøø	0	00010	1	1
nop																	0		0	
shl R3, R1, 3		øøøøø	øøøøø	1	ø	01	øø	1100	0xøøøø	00011	01010110	0xøøøø	0xøøøø	ø	0xøøøø	0xøøøø	0	00011	1	1
storr R2, R3		øøøøø	øøøøø	ø	1	11	øø	0000	0xøøøø	øøøø	01010110	0xøøøø	0xøøøø	0	0x0008	0x0005	1	ø	0	ø
loadi R5, 1f1f		øøøøø	øøøøø	ø	ø	øø	øø	øøøø	0xøøøø	øøøø	øøøøøøøø	0x1f1f	0xCCCC	1	0x1f1f	0xøøøø	0	00101	1	0

The instruction time line below shows how we used a nop instruction as a pipeline stall between the addi instruction and the shl instruction. We had to use a pipeline stall in this case rather forwarding, as there was no way to use any of the forwarding paths to get the value to be in the correct location at the time when we needed it.

Test command	TimeLine									
inc R1, R0	Fetch	Reg Read	ALU+	Mem Acc	Reg Write					
addi R2, R0, 0x0005		Fetch	Reg Read	ALU+	Mem Acc	Reg Write				
nop										
shl R3, R1, 3				Fetch	Reg Read	ALU+	Mem Acc	Reg Write		
storr R2, R3					Fetch	Reg Read	ALU+	Mem Acc	Reg Write	
loadi R5, 1f1f						Fetch	Reg Read	ALU+	Mem Acc	Reg Write

We also used forwarding in our control logic. In fact, the only time we take a value directly from a register is when the value of R Zero is used for the first two instruction. Both the shift left and the store instruction use forwarding in order to execute sooner than they would of other wise.

You can see in the table below that the only time RA has a definite value is during the first two clock cylces. You can also see use of select 5 and 6 to forward data (that has not yet been writte to the register) into the A and B registers. Select 2 has also been used to forward data from the output of the alu back to the input of the alu.

	Fetch	Reg Read				ALU+						Mem Acc						Reg Write		
Clock cycle		RA	RB	SEL (6)[7]	SEL (5)[6]	SEL (2)[3:2]	SEL (1)[1:0]	AL	IMM	SH	Flags	MA	M_in	SEL (3)[4]	M_DA	M_out	OEN	WA	W_EN	SEL (4)[5]
1		00000	øøøøø	0	ø	øø	øø	øøøø	0xøøøø	øøøø	øøøøøøøø	0xøøøø	0xøøøø	ø	0xøøøø	0xøøøø	0	ø	0	ø
2		00000	øøøøø	0	ø	01	øø	1000	0xøøøø	øøøø	01010110	0xøøøø	0xøøøø	ø	0xøøøø	0xøøøø	0	ø	0	ø
3		øøøøø	øøøøø	ø	ø	01	10	1010	0x0005	øøøø	01010010	0xøøøø	0xøøøø	ø	0xøøøø	0xøøøø	0	ø	0	ø
4		øøøøø	øøøøø	1	ø	øø	øø	øøøø	0xøøøø	øøøø	øøøøøøøø	0xøøøø	0xøøøø	ø	0xøøøø	0xøøøø	0	00001	1	1
5		øøøøø	øøøøø	ø	1	01	øø	1100	0xøøøø	00011	01010010	0xøøøø	0xøøøø	ø	0xøøøø	0xøøøø	0	00010	1	1
6		øøøøø	øøøøø	ø	ø	11	øø	0000	0xøøøø	øøøø	01010010	0xøøøø	0xøøøø	ø	0xøøøø	0xøøøø	0	ø	0	ø
7		øøøøø	øøøøø	ø	ø	øø	øø	øøøø	0xøøøø	øøøø	øøøøøøøø	0xøøøø	0xøøøø	0	0x0008	0x0005	1	00011	1	1
8		øøøøø	øøøøø	ø	ø	øø	øø	øøøø	0xøøøø	øøøø	øøøøøøøø	0x1f1f	0xCCCC	1	0x1f1f	0xøøøø	0	ø	0	ø
9		øøøøø	øøøøø	ø	ø	øø	øø	øøøø	0xøøøø	øøøø	øøøøøøøø	0xøøøø	0xøøøø	ø	0xøøøø	0xøøøø	0	00101	1	0

Architecture D

DataPathD.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.All;

entity DataPath_D is
        Generic(
                data_size : natural := 16;
                num_registers : natural := 32
        );
        Port (
                clk : in  STD_LOGIC;
                rst : in STD_LOGIC;
                en : in STD_LOGIC;

                R_A : in  STD_LOGIC_VECTOR (log2(num_registers)-1 downto 0); -- Read address A
                R_B : in  STD_LOGIC_VECTOR (log2(num_registers)-1 downto 0); -- Read address B
                W_EN : in  STD_LOGIC;                                                                   -- Register
write enable
                W_A : in  STD_LOGIC_VECTOR (log2(num_registers)-1 downto 0); -- Write address
                IMM : in  STD_LOGIC_VECTOR (data_size-1 downto 0);                      -- Itermediate value
                M_A : in  STD_LOGIC_VECTOR (data_size-1 downto 0);                      -- Memory address
                M_in : in  STD_LOGIC_VECTOR (data_size-1 downto 0);                     -- Memory input (read)
                SEL : in  STD_LOGIC_VECTOR (7 downto 0);                                        -- Selector control
                AL : in  STD_LOGIC_VECTOR (3 downto 0);                                         -- ALU control
                SH : in  STD_LOGIC_VECTOR (log2(data_size)-1 downto 0);         -- Shift amount

                Flags : out  STD_LOGIC_VECTOR (7 downto 0);                                     -- ALU flags
                M_DA : out  STD_LOGIC_VECTOR (data_size-1 downto 0);            -- Memory address
                M_out : out  STD_LOGIC_VECTOR (data_size-1 downto 0)            -- Memory output (write)
        );
end DataPath_D;

architecture Behavioral of DataPath_D is

        -- Data to write to the registers
        signal reg_in : STD_LOGIC_VECTOR (data_size-1 downto 0);

        -- Outputs of the registers
        signal A_data : STD_LOGIC_VECTOR (data_size-1 downto 0);
        signal B_data : STD_LOGIC_VECTOR (data_size-1 downto 0);

        -- The two multiplexers for A and B
        signal A_mux : STD_LOGIC_VECTOR (data_size-1 downto 0);
        signal B_mux : STD_LOGIC_VECTOR (data_size-1 downto 0);

        -- Output of the register on the A and B buses
        signal A_reg_out : STD_LOGIC_VECTOR (data_size-1 downto 0);
        signal B_reg_out : STD_LOGIC_VECTOR (data_size-1 downto 0);

        -- The inputs to the ALU
        signal A_ALU : STD_LOGIC_VECTOR (data_size-1 downto 0);
        signal B_ALU : STD_LOGIC_VECTOR (data_size-1 downto 0);

        -- The output of the combined ALU and shifter
        signal ALU_out : STD_LOGIC_VECTOR (data_size-1 downto 0);
        signal ALU_reg_out : STD_LOGIC_VECTOR (data_size-1 downto 0);

        -- The ALU output after it passes through a second register
        signal ALU_reg_out_reg_out : STD_LOGIC_VECTOR (data_size-1 downto 0);

        -- Signals for memory registers
        signal M_in_reg_out : STD_LOGIC_VECTOR (data_size-1 downto 0);

begin

        -- The two multiplexers for A and B
        A_mux <= A_data when SEL(7) = '0' else reg_in;
        B_mux <= B_data when SEL(6) = '0' else reg_in;
```

(Continued.)

```
    -- The multiplexers on the input of the ALU
    A_ALU <=
            ALU_reg_out     when SEL(3 downto 2) = "11" else
            (others => 'U') when SEL(3 downto 2) = "10" else
            A_reg_out               when SEL(3 downto 2) = "01" else
            reg_in                  when SEL(3 downto 2) = "00" else
            (others => 'U');
    B_ALU <=
            ALU_reg_out     when SEL(1 downto 0) = "11" else
            IMM                     when SEL(1 downto 0) = "10" else
            B_reg_out               when SEL(1 downto 0) = "01" else
            reg_in                  when SEL(1 downto 0) = "00" else
            (others => 'U');

    -- The address multiplexer for the memory
    M_DA <= M_A when SEL(4) = '1' else ALU_reg_out;

    -- The register write multiplexer
    reg_in <= ALU_reg_out_reg_out when SEL(5) = '1' else M_in_reg_out;

    -- The register on the A bus
    A_reg: entity work.Reg Generic Map (data_size => data_size)
    PORT MAP(
            clk => clk,
            rst => rst,
            en => en,
            data_in => A_mux,
            data_out => A_reg_out
    );

    -- The register on the B bus
    B_reg: entity work.Reg Generic Map (data_size => data_size)
    PORT MAP(
            clk => clk,
            rst => rst,
            en => en,
            data_in => B_mux,
            data_out => B_reg_out
    );

    -- The Register on the out put of the ALU
    ALU_reg: entity work.Reg Generic Map (data_size => data_size)
    PORT MAP(
            clk => clk,
            rst => rst,
            en => en,
            data_in => ALU_out,
            data_out => ALU_reg_out
    );

    -- The register on the memory read bus
    M_in_reg: entity work.Reg Generic Map (data_size => data_size)
    PORT MAP(
            clk => clk,
            rst => rst,
            en => en,               data_in => M_in,
            data_out => M_in_reg_out
    );

    -- The second register on the ALU out
    ALU_reg_out_reg: entity work.Reg Generic Map (data_size => data_size)
    PORT MAP(
            clk => clk,
            rst => rst,
            en => en,
            data_in => ALU_reg_out,
            data_out => ALU_reg_out_reg_out
    );

    -- The register on the memory read bus
    M_out_reg: entity work.Reg Generic Map (data_size => data_size)
    PORT MAP(
            clk => clk,
            rst => rst,
            en => en,
            data_in => B_reg_out,
            data_out => M_out
    );
```

(Continued.)

```
        -- The ALU
        ALU: entity work.ALU_param GENERIC MAP( N => data_size )
        PORT MAP(
                A => A_ALU,
                B => B_ALU,
                X => SH,
                ctrl => AL,
                O => ALU_out,
                flags => flags
        );

        -- The register bank
        Registers: entity work.regbank
        PORT MAP(
                RSELA => R_A,
                RSELB => R_B,
                WSEL => W_A,
                D => reg_in,
                WEN => W_EN,
                clk => clk,
                A => A_data,
                B => B_data,
                rst => '0'
        );

end Behavioral;
```

DataPathD-TB.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.DigEng.ALL;
USE work.easyprint.ALL;
USE ieee.numeric_std.ALL;

ENTITY DataPathB_TB IS
END DataPathB_TB;

ARCHITECTURE behavior OF DataPathB_TB IS

        -- Constants
        constant data_size : NATURAL := 16;
        constant num_registers : NATURAL := 32;

        -- Component Declaration for the Unit Under Test (UUT)
        COMPONENT DataPath_B
                GENERIC(
                        data_size : natural;
                        num_registers : natural
                );
                PORT(
                        R_A : IN  std_logic_vector(log2(num_registers)-1 downto 0);
                        R_B : IN  std_logic_vector(log2(num_registers)-1 downto 0);
                        W_EN : IN  std_logic;
                        W_A : IN  std_logic_vector(log2(num_registers)-1 downto 0);
                        clk : IN  std_logic;
                        IMM : IN  std_logic_vector(data_size-1 downto 0);
                        AL : IN  std_logic_vector(3 downto 0);
                        SH : IN  std_logic_vector(log2(data_size)-1 downto 0);
                        M_A : IN  std_logic_vector(data_size-1 downto 0);
                        S : IN  std_logic_vector(4 downto 1);
                        flags : OUT  std_logic_vector(7 downto 0);
                        M_B : OUT  std_logic_vector(data_size-1 downto 0);
                        M_DA : OUT  std_logic_vector(data_size-1 downto 0);
                        M_in : IN  std_logic_vector(data_size-1 downto 0)
                );
        END COMPONENT;


   --Inputs
   signal R_A   : std_logic_vector      (log2(num_registers)-1 downto 0) := (others => '0');
   signal R_B   : std_logic_vector      (log2(num_registers)-1 downto 0) := (others => '0');
   signal W_EN : std_logic
        := '0';
   signal W_A   : std_logic_vector      (log2(num_registers)-1 downto 0) := (others => '0');
   signal clk   : std_logic
        := '0';
   signal IMM   : std_logic_vector      (data_size-1 downto 0)                          := (others => '0');
   signal AL    : std_logic_vector      (3 downto 0)                                      := (others => '0');
   signal SH    : std_logic_vector      (log2(data_size)-1 downto 0)          := (others => '0');
   signal M_A   : std_logic_vector      (data_size-1 downto 0)                  := (others => '0');
   signal S     : std_logic_vector      (4 downto 1)                                      := (others => '0');
   signal M_in : std_logic_vector       (data_size-1 downto 0)                  := (others => '0');

        --Outputs
   signal flags         : std_logic_vector      (7 downto 0);
   signal M_B           : std_logic_vector      (data_size-1 downto 0);
   signal M_DA  : std_logic_vector      (data_size-1 downto 0);
   signal OEN : std_logic := '0';

   -- Clock period definitions
   constant clk_period  : time  := 10 ns;
   constant wait_time   : time  := clk_period;

        -- Test data for self checking test bench
        type TEST_VECTOR is RECORD
                 W_EN : std_logic;
                 AL : std_logic_vector(3 downto 0);
                 R_A : STD_LOGIC_VECTOR(log2(num_registers)-1 downto 0);
                 R_B : std_logic_vector(log2(num_registers)-1 downto 0);
                 W_A : std_logic_vector(log2(num_registers)-1 downto 0);
```

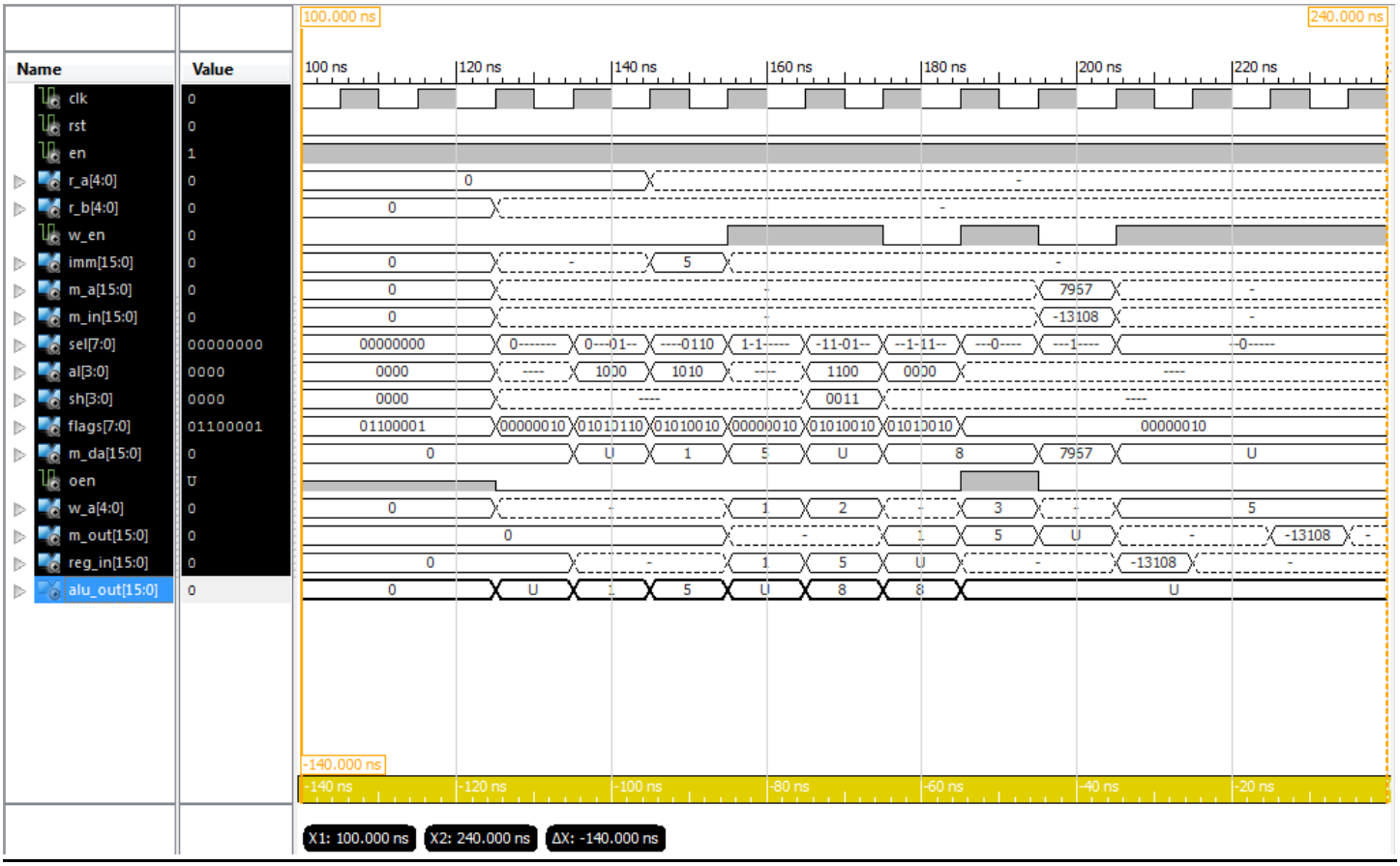
(Continued.)

```
                IMM : std_logic_vector(data_size-1 downto 0);
                SH : std_logic_vector(log2(data_size)-1 downto 0) ;
                M_A : std_logic_vector(data_size-1 downto 0);
                S : std_logic_vector(4 downto 1);
                M_in : std_logic_vector(data_size-1 downto 0);

                flags : std_logic_vector(7 downto 0);
                M_B : std_logic_vector(data_size-1 downto 0);
                M_DA : std_logic_vector(data_size-1 downto 0);

                OEN : std_logic;
        end RECORD;

        type TEST_VECTOR_ARRAY is ARRAY(NATURAL RANGE <>) of TEST_VECTOR;

        constant test_vectors : TEST_VECTOR_ARRAY := (
--              W_EN, AL, R_A, R_B, W_A, IMM, SH, M_A, S, M_in, flags, M_B, M_DA,OEN

                (       '1',    "1000", "00000",        "-----",        "00001",        "----------------",
                "----",         "----------------",     "0---",         "----------------",
                "01010110",                     "----------------",     "----------------",
                '0'),

                (       '1',    "1010", "00000",        "-----",        "00010",        X"0005",
                "----", "----------------",     "0--1",         "----------------",
                "01010010",                     "----------------",     "----------------",
                '0'),

                (       '1',    "1100", "00001",        "-----",        "00011",        "----------------",
                "0011", "----------------",     "0---",         "----------------",
                "01010010",                     "----------------",     "----------------",
                '0'),

                (       '0',    "0000", "00011",        "00010",        "-----",        "----------------",
                "----", "----------------",     "0---",         "----------------",
                "01010010",                     X"0005",                        X"0008",
                '1'),

                (       '1',    "----", "-----",        "-----",        "00101",        "----------------",
                "----", X"1f1f",                        "11--",         X"CCCC",
                "--------",                     "----------------",     X"1f1f",
                '0')
        );

BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: DataPath_B
        Generic Map(
                data_size => data_size,
                num_registers => num_registers
        )
        PORT MAP (
          R_A => R_A,
          R_B => R_B,
          W_EN => W_EN,
          W_A => W_A,
          clk => clk,
          IMM => IMM,
          AL => AL,
          SH => SH,
          M_A => M_A,
          S => S,
          flags => flags,
          M_B => M_B,
          M_DA => M_DA,
          M_in => M_in
        );
```

(Continued.)

```
    -- Clock process definitions
    clk_process :process
    begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
    end process;


         -- Stimulus process
         stim_proc: process
         begin

                -- hold reset state for 100 ns.
                wait for 100 ns;

                -- run the test for every set of data
                for i in test_vectors'range loop

                        -- assign test inputs
                        R_A <= test_vectors(i).R_A;
                        R_B <= test_vectors(i).R_B;
                        W_EN <= test_vectors(i).W_EN;
                        W_A <= test_vectors(i).W_A;
                        IMM <= test_vectors(i).IMM;
                        AL <= test_vectors(i).AL;
                        SH <= test_vectors(i).SH;
                        M_A <= test_vectors(i).M_A;
                        S <= test_vectors(i).S;
                        M_in <= test_vectors(i).M_in;

                        OEN <= test_vectors(i).OEN;

                        wait until rising_edge(clk);

                        -- Check that the actual outputs are the same as were expecting
                        -- Have to use std_match when comparing meta values like '-'
                        assert std_match(flags, test_vectors(i).flags)
                        report lf & " [ERR!] Test " & integer'image(i)& lf &
                                " Actual flags did not equal expected flags."&
                                " Actual [ " & to_bstring(flags) & " ]" &
                                " Expected [ " & to_bstring(test_vectors(i).flags) & " ]"
                        severity error;

                        assert std_match(test_vectors(i).M_B, M_B)
                        report lf &" [ERR!] Test " & integer'image(i)& lf &
                                " Actual value to memory did not equal expected value to memory."&
                                " Actual [ " & u_tostr(M_B) & " ]" &
                                " Expected [ " & u_tostr(test_vectors(i).M_B) & " ]"
                        severity error;

                        assert std_match(M_DA , test_vectors(i).M_DA)
                        report lf &" [ERR!] Test " & integer'image(i)& lf &
                                " Actual memory address did not equal expected memory address."&
                                " Actual [ " & u_tostr(M_DA) & " ]" &
                                " Expected [ " & u_tostr(test_vectors(i).M_DA) & " ]"
                        severity error;

                        -- If there were no isses report that the test was successful
                        assert not (
                                std_match(flags, test_vectors(i).flags) and
                                std_match(M_B, test_vectors(i).M_B) and
                                std_match(M_DA, test_vectors(i).M_DA))
                        report lf &" [ OK ] Test " & integer'image(i)& " was successful!"
                        severity note;

                        wait until falling_edge(clk);

                end loop;

                wait;

         end process;

END;
```

Simulations



Screenshot showing entire simulation of Architecture D. The signal reg_in is the data data that will be written to the register bank.

iSim Console Output

```
ISim P.28xd (signature 0xa0883be4)
This is a Full version of ISim.
Time resolution is 1 ps
WARNING: Simulation object /datapath_d_tb/test_vectors was not traceable in the design for the following reason:
ISim does not yet support tracing of constant and generic multi-dimensional arrays.
Simulator is doing circuit initialization process.
at 0 ps, Instance /datapath_d_tb/uut/ALU/ : Warning: NUMERIC_STD.">=": metavalue detected, returning FALSE
at 0 ps, Instance /datapath_d_tb/uut/ALU/ : Warning: NUMERIC_STD."<=": metavalue detected, returning FALSE
at 0 ps, Instance /datapath_d_tb/uut/ALU/ : Warning: NUMERIC_STD.">": metavalue detected, returning FALSE
at 0 ps, Instance /datapath_d_tb/uut/ALU/ : Warning: NUMERIC_STD."<": metavalue detected, returning FALSE
at 0 ps, Instance /datapath_d_tb/uut/ALU/ : Warning: NUMERIC_STD."=": metavalue detected, returning FALSE
at 0 ps, Instance /datapath_d_tb/uut/ALU/ : Warning: NUMERIC_STD."/=": metavalue detected, returning TRUE
at 0 ps, Instance /datapath_d_tb/uut/ALU/ : Warning: NUMERIC_STD."=": metavalue detected, returning FALSE
Finished circuit initialization process.

at 130 ns(1): Note:  [ OK ] Test 0 was successful! (/datapath_d_tb/).
at 140 ns(1): Note:  [ OK ] Test 1 was successful! (/datapath_d_tb/).
at 150 ns(1): Note:  [ OK ] Test 2 was successful! (/datapath_d_tb/).
at 160 ns(1): Note:  [ OK ] Test 3 was successful! (/datapath_d_tb/).
at 170 ns(1): Note:  [ OK ] Test 4 was successful! (/datapath_d_tb/).
at 180 ns(1): Note:  [ OK ] Test 5 was successful! (/datapath_d_tb/).
at 190 ns(1): Note:  [ OK ] Test 6 was successful! (/datapath_d_tb/).
at 200 ns(1): Note:  [ OK ] Test 7 was successful! (/datapath_d_tb/).
at 210 ns(1): Note:  [ OK ] Test 8 was successful! (/datapath_d_tb/).

ISim>
```

All but the first meta value warnings have been removed to improve readability.

HDL Synthesis

```
=========================================================================
*                           HDL Synthesis                               *
=========================================================================

Synthesizing Unit <DataPath_D>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Lab_2\DataPaths - RegBankAlt\DataPath_D.vhd".
        data_size = 16
        num_registers = 32
    Found 16-bit 3-to-1 multiplexer for signal <A_ALU> created at line 52.
    Found 16-bit 4-to-1 multiplexer for signal <B_ALU> created at line 53.
    Summary:
        inferred   6 Multiplexer(s).
Unit <DataPath_D> synthesized.

Synthesizing Unit <Reg>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Lab_2\DataPaths - RegBankAlt\Reg.vhd".
        data_size = 16
    Found 16-bit register for signal <data_out>.
    Summary:
        inferred  16 D-type flip-flop(s).
Unit <Reg> synthesized.

Synthesizing Unit <ALU_param>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Lab_2\DataPaths - RegBankAlt\ALU_param.vhd".
        N = 16
    Found 16-bit adder for signal <A_itrn[15]_B_itrn[15]_add_27_OUT> created at line 69.
    Found 16-bit adder for signal <A_itrn[15]_GND_7_o_add_31_OUT> created at line 1253.
    Found 16-bit subtractor for signal <A_itrn[15]_B_itrn[15]_sub_26_OUT<15:0>> created at line 70.
    Found 16-bit subtractor for signal <A_itrn[15]_GND_7_o_sub_30_OUT<15:0>> created at line 1320.
    Found 16-bit shifter rotate right for signal <A_itrn[15]_X_itrn[30]_rotate_right_17_OUT> created at line 3021
    Found 16-bit shifter rotate left for signal <A_itrn[15]_X_itrn[30]_rotate_left_19_OUT> created at line 3012
    Found 16-bit shifter arithmetic right for signal <A_itrn[15]_X_itrn[30]_shift_right_21_OUT> created at line 2982
    Found 16-bit shifter logical left for signal <A_itrn[15]_X_itrn[30]_shift_left_23_OUT> created at line 2973
    Found 16-bit 13-to-1 multiplexer for signal <O_itrn> created at line 42.
    Found 16-bit comparator greater for signal <flags<3>> created at line 99
    Found 16-bit comparator greater for signal <flags<4>> created at line 100
    Summary:
        inferred   1 Adder/Subtractor(s).
        inferred   2 Comparator(s).
        inferred  16 Multiplexer(s).
        inferred   4 Combinational logic shifter(s).
Unit <ALU_param> synthesized.

Synthesizing Unit <regbank>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Lab_2\DataPaths - RegBankAlt\otherRegBank.vhd".
    Found 16-bit register for signal <REG02>.
    Found 16-bit register for signal <REG03>.
    Found 16-bit register for signal <REG04>.
    Found 16-bit register for signal <REG05>.
    Found 16-bit register for signal <REG06>.
    Found 16-bit register for signal <REG07>.
    Found 16-bit register for signal <REG08>.
    Found 16-bit register for signal <REG09>.
    Found 16-bit register for signal <REG10>.
    Found 16-bit register for signal <REG11>.
    Found 16-bit register for signal <REG12>.
    Found 16-bit register for signal <REG13>.
    Found 16-bit register for signal <REG14>.
    Found 16-bit register for signal <REG15>.
    Found 16-bit register for signal <REG16>.
    Found 16-bit register for signal <REG17>.
    Found 16-bit register for signal <REG18>.
    Found 16-bit register for signal <REG19>.
    Found 16-bit register for signal <REG20>.
    Found 16-bit register for signal <REG21>.
    Found 16-bit register for signal <REG22>.
    Found 16-bit register for signal <REG23>.
    Found 16-bit register for signal <REG24>.
    Found 16-bit register for signal <REG25>.
    Found 16-bit register for signal <REG26>.
    Found 16-bit register for signal <REG27>.
    Found 16-bit register for signal <REG28>.
    Found 16-bit register for signal <REG29>.
    Found 16-bit register for signal <REG30>.
    Found 16-bit register for signal <REG31>.
```

```
    Found 16-bit register for signal <REG01>.
    Found 16-bit 32-to-1 multiplexer for signal <A> created at line 30.
    Found 16-bit 32-to-1 multiplexer for signal <B> created at line 31.
    Summary:
        inferred 496 D-type flip-flop(s).
        inferred   2 Multiplexer(s).
Unit <regbank> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 1
 16-bit addsub                                        : 1
# Registers                                            : 37
 16-bit register                                      : 37
# Comparators                                          : 2
 16-bit comparator greater                            : 2
# Multiplexers                                         : 24
 1-bit 2-to-1 multiplexer                             : 6
 16-bit 2-to-1 multiplexer                            : 14
 16-bit 3-to-1 multiplexer                            : 1
 16-bit 32-to-1 multiplexer                           : 2
 16-bit 4-to-1 multiplexer                            : 1
# Logic shifters                                       : 4
 16-bit shifter arithmetic right                      : 1
 16-bit shifter logical left                          : 1
 16-bit shifter rotate left                           : 1
 16-bit shifter rotate right                          : 1
# Xors                                                 : 1
 16-bit xor2                                          : 1


=========================================================================
```