```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity top_level_tb is
end entity;

architecture behavior of top_level_tb is
  constant TIME_DELAY : time := 10 ns;
  constant NUM_VALS : integer := 26;

  type RegWr_array is array   (0 to (NUM_VALS - 1)) of std_logic;
  type Rd_array is array      (0 to (NUM_VALS - 1)) of std_logic_vector(4 downto 0);
  type Rs_array is array      (0 to (NUM_VALS - 1)) of std_logic_vector(4 downto 0);
  type Rt_array is array      (0 to (NUM_VALS - 1)) of std_logic_vector(4 downto 0);
  type ALUctr_array is array  (0 to (NUM_VALS - 1)) of std_logic_vector(2 downto 0);
  type Zero_array is array    (0 to (NUM_VALS - 1)) of std_logic;
  type Overflow_array is array(0 to (NUM_VALS - 1)) of std_logic;
  type Carryout_array is array(0 to (NUM_VALS - 1)) of std_logic;
  type Result_array is array  (0 to (NUM_VALS - 1)) of std_logic_vector(31 downto 0);

  -- Expected input and output data.
  constant RegWr_vals : RegWr_array := ('0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1');
  constant Rd_vals : Rd_array := ("00011","00011","00011","00011","00011","00011","00111","00111","00100","00100","00101","00101","00110","00110","00111","00111","01000","01000","01001","01001","01010","01010","01011","01011","01100","01100");
  constant Rs_vals : Rs_array := ("00001","00001","00001","00001","00001","00001","00111","00111","00010","00010","00011","00011","00101","00101","00110","00110","00011","00011","01000","01000","00110","00110","01000","01000","01000","01000");
  constant Rt_vals : Rt_array := ("00010","00010","00010","00010","00010","00010","00111","00111","00011","00011","00100","00100","00101","00101","00101","00101","00101","00101","00101","00101","00111","00111","01000","01000","01000","01000");
  constant ALUctr_vals : ALUctr_array := ("011", "011","000", "000","001", "001","000","000","010","010","000","000","100","100","110","110","111","111","101","101","000","000","000","000","001","001");
  constant Zero_vals : Zero_array := ('0','0','0','0','0','0','1','1','1','1','0','0','0','0','0','0','0','0','0','0','1','1','1','1','1','1');
  constant Overflow_vals : Overflow_array := ('0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0');
  constant Carryout_vals : Carryout_array := ('0','0','0','0','0','0','0','0','0','0','1','1','1','1','1','1','1','1','1','1','0','0','1','1','1','1');
  constant Result_vals : Result_array := ("00000000000000000000000000000011",
                                          "00000000000000000000000000000011",
                                          "00000000000000000000000000000011",
                                          "00000000000000000000000000000011",
                                          "11111111111111111111111111111111",
                                          "11111111111111111111111111111111",
                                          "00000000000000000000000000000000",
                                          "00000000000000000000000000000000",
                                          "00000000000000000000000000000010",
                                          "00000000000000000000000000000010",
                                          "00000000000000000000000000000001",
                                          "00000000000000000000000000000001",
                                          "00000000000000000000000000000010",
                                          "00000000000000000000000000000010",
                                          "00000000000000000000000000000100",
                                          "00000000000000000000000000000100",
                                          "11111111111111111111111111111111",
                                          "11111111111111111111111111111111",
                                          "01111111111111111111111111111111",
                                          "01111111111111111111111111111111",
                                                   "00000000000000000000000000000110",
                                                   "00000000000000000000000000000110",
                                                   "00000000000000000000000000000110",
                                                   "00000000000000000000000000000110",
                                                   "00000000000000000000000000000110",
                                                   "00000000000000000000000000000110");

    --signal clk_sig : std_logic := '0';
    signal RegWr_sig : std_logic;
    signal Rd_sig : std_logic_vector(4 downto 0);
    signal Rs_sig : std_logic_vector(4 downto 0);
    signal Rt_sig : std_logic_vector(4 downto 0);
    signal ALUctr_sig : std_logic_vector(2 downto 0);
    signal Zero_sig : std_logic;
    signal Overflow_sig : std_logic;
    signal Carryout_sig : std_logic;
    signal Result_sig : std_logic_vector(31 downto 0):= "00000000000000000000000000000000";

  begin

    DUT : entity work.top_level(simple)
      port map(--clk => clk_sig,
               RegWr => RegWr_sig,
               Rd => Rd_sig,
               Rs => Rs_sig,
               Rt => Rt_sig,
               ALUctr => ALUctr_sig,
               Zero => Zero_sig,
               Overflow => Overflow_sig,
               Carryout => Carryout_sig,
               Result => Result_sig);


    stimulus : process
    begin
      for i in 0 to (NUM_VALS - 1) loop
        RegWr_sig <= RegWr_vals(i);
        Rd_sig <= Rd_vals(i);
        Rs_sig <= Rs_vals(i);
        Rt_sig <= Rt_vals(i);
        ALUctr_sig <= ALUctr_vals(i);
        wait for TIME_DELAY;
      end loop;
      wait;
    end process stimulus;

    monitor : process
      variable i : integer := 0;
    begin
      wait for TIME_DELAY/4;
      while (i < NUM_VALS) loop
        assert RegWr_sig = RegWr_vals(i)
          report "RegWr value is incorrect."
          severity note;

        assert Rd_sig = Rd_vals(i)
          report "Rd value is incorrect."
          severity note;

        assert Rs_sig = Rs_vals(i)
          report "Rs value is incorrect."
          severity note;

        assert Rt_sig = Rt_vals(i)
          report "Rt value is incorrect."
          severity note;

        assert ALUctr_sig = ALUctr_vals(i)
          report "ALUctr value is incorrect."
```

```vhdl
        severity note;

      wait for TIME_DELAY/2;

      assert Zero_sig = Zero_vals(i)
        report "Zero value is incorrect."
        severity note;

      assert Overflow_sig = Overflow_vals(i)
        report "Overflow value is incorrect."
        severity note;

      assert Carryout_sig = Carryout_vals(i)
        report "Carryout value is incorrect."
        severity note;

      assert Result_sig = Result_vals(i)
        report "Results value is incorrect."
        severity note;

      i := i + 1;
      wait for TIME_DELAY/2;
    end loop;
    wait;
  end process monitor;

end behavior;
```