```
# riscvtest.s
# Sarah.Harris@unlv.edu
# David_Harris@hmc.edu
# 27 Oct 2020
#
# Test the RISC-V processor:
#   add, sub, and, or, slt, addi, lw, sw, beq, jal
# If successful, it should write the value 25 to address 100
#       RISC-V Assembly         Description              Address   Machine Code
main:   addi x2, x0, 5          # x2 = 5                 0         00500113
        addi x3, x0, 12         # x3 = 12                4         00C00193
        addi x7, x3, -9         # x7 = (12 - 9) = 3      8         FF718393
        or   x4, x7, x2         # x4 = (3 OR 5) = 7      C         0023E233
        and  x5, x3, x4         # x5 = (12 AND 7) = 4    10        0041F2B3
        add  x5, x5, x4         # x5 = 4 + 7 = 11        14        004282B3
        beq  x5, x7, end        # shouldn't be taken     18        02728863
        slt  x4, x3, x4         # x4 = (12 < 7) = 0      1C        0041A233
        beq  x4, x0, around     # should be taken        20        00020463
        addi x5, x0, 0          # shouldn't execute      24        00000293
around: slt  x4, x7, x2         # x4 = (3 < 5) = 1       28        0023A233
        add  x7, x4, x5         # x7 = (1 + 11) = 12     2C        005203B3
        sub  x7, x7, x2         # x7 = (12 - 5) = 7      30        402383B3
        sw   x7, 84(x3)         # [96] = 7               34        0471AA23
        lw   x2, 96(x0)         # x2 = [96] = 7          38        06002103
        add  x9, x2, x5         # x9 = (7 + 11) = 18     3C        005104B3
        jal  x3, end            # jump to end, x3 = 0x44 40        008001EF
        addi x2, x0, 1          # shouldn't execute      44        00100113
end:    add  x2, x2, x9         # x2 = (7 + 18) = 25     48        00910133
        sw   x2, 0x20(x3)       # [100] = 25             4C        0221A023
done:   beq  x2, x2, done       # infinite loop          50        00210063
```

Figure 7.64 riscvtest.s

```
00500113
00C00193
FF718393
0023E233
0041F2B3
004282B3
02728863
0041A233
00020463
00000293
0023A233
005203B3
402383B3
0471AA23
06002103
005104B3
008001EF
00100113
00910133
0221A023
00210063
```

Figure 7.65 riscvtest.txt

7.6.3 Testbench

The testbench loads a program into the memories. The program in Figure 7.64 exercises all of the instructions by performing a computation that should produce the correct result only if all of the instructions are functioning correctly. Specifically, the program will write the value 25 to address 100 if it runs correctly, but it is unlikely to do so if the hardware is buggy. This is an example of *ad hoc* testing.

The machine code is stored in a text file called riscvtest.txt (Figure 7.65) which is loaded by the testbench during simulation. The file consists of the machine code for the instructions written in hexadecimal, one instruction per line.

The testbench, top-level RISC-V module (that instantiates the RISC-V processor and memories), and external memory HDL code are given in the following examples. The testbench instantiates the top-level module being tested and generates a periodic clock and a reset at the start of the simulation. It checks for memory writes and reports success if the correct value (25) is written to address 100. The memories in this example hold 64 32-bit words each.

HDL Example 7.12 TESTBENCH

SystemVerilog

```
module testbench();

  logic        clk;
  logic        reset;
  logic [31:0] WriteData, DataAdr;
  logic        MemWrite;

  // instantiate device to be tested
  top dut(clk, reset, WriteData, DataAdr, MemWrite);

  // initialize test
  initial
    begin
      reset <= 1; # 22; reset <= 0;
  end

  // generate clock to sequence tests
  always
    begin
      clk <= 1; # 5; clk <= 0; # 5;
  end

  // check results
  always @(negedge clk)
    begin
      if(MemWrite) begin
        if(DataAdr === 100 & WriteData === 25) begin
          $display("Simulation succeeded");
          $stop;
        end else if (DataAdr !== 96) begin
          $display("Simulation failed");
          $stop;
        end
      end
    end
endmodule
```

VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD_UNSIGNED.all;

entity testbench is
end;

architecture test of testbench is
  component top
    port(clk, reset:          in  STD_LOGIC;
         WriteData, DataAdr: out STD_LOGIC_VECTOR(31 downto 0);
         MemWrite:           out STD_LOGIC);
  end component;

  signal WriteData, DataAdr:  STD_LOGIC_VECTOR(31 downto 0);
  signal clk, reset, MemWrite: STD_LOGIC;
begin
  -- instantiate device to be tested
  dut: top port map(clk, reset, WriteData, DataAdr, MemWrite);

  -- Generate clock with 10 ns period
  process begin
     clk <= '1';
     wait for 5 ns;
     clk <= '0';
     wait for 5 ns;
  end process;

  -- Generate reset for first two clock cycles
  process begin
    reset <= '1';
    wait for 22 ns;
    reset <= '0';
    wait;
  end process;

  -- check that 25 gets written to address 100 at end of program
  process(clk) begin
    if(clk'event and clk = '0' and MemWrite = '1') then
      if(to_integer(DataAdr) = 100 and
         to_integer(writedata) = 25) then
        report "NO ERRORS: Simulation succeeded" severity
        failure;
      elsif (DataAdr /= 96) then
        report "Simulation failed" severity failure;
      end if;
    end if;
  end process;
end;
```