Single-Cycle MIPS Processor

Digital Design and Computer Architecture

David M Harris & Sarah L. Harris

HDL Example 7.1 SINGLE-CYCLE MIPS PROCESSOR

Verilog

```
module mips(input          clk, reset,
            output [31:0]  pc,
            input  [31:0]  instr,
            output         memwrite,
            output [31:0]  aluout, writedata,
            input  [31:0]  readdata);

  wire       memtoreg, branch,
             alusrc, regdst, regwrite, jump;
  wire [2:0] alucontrol;

  controller c(instr[31:26], instr[5:0], zero,
               memtoreg, memwrite, pcsrc,
               alusrc, regdst, regwrite, jump,
               alucontrol);
  datapath dp(clk, reset, memtoreg, pcsrc,
              alusrc, regdst, regwrite, jump,
              alucontrol,
              zero, pc, instr,
              aluout, writedata, readdata);
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity mips is -- single cycle MIPS processor
  port(clk, reset:        in  STD_LOGIC;
       pc:                out STD_LOGIC_VECTOR(31 downto 0);
       instr:             in  STD_LOGIC_VECTOR(31 downto 0);
       memwrite:          out STD_LOGIC;
       aluout, writedata: out STD_LOGIC_VECTOR(31 downto 0);
       readdata:          in  STD_LOGIC_VECTOR(31 downto 0));
end;

architecture struct of mips is
  component controller
    port(op, funct:         in  STD_LOGIC_VECTOR(5 downto 0);
         zero:              in  STD_LOGIC;
         memtoreg, memwrite: out STD_LOGIC;
         pcsrc, alusrc:     out STD_LOGIC;
         regdst, regwrite:  out STD_LOGIC;
         jump:              out STD_LOGIC;
         alucontrol:        out STD_LOGIC_VECTOR(2 downto 0));
 end component;
 component datapath
   port(clk, reset:        in  STD_LOGIC;
        memtoreg, pcsrc:   in  STD_LOGIC;
        alusrc, regdst:    in  STD_LOGIC;
        regwrite, jump:    in  STD_LOGIC;
        alucontrol:        in  STD_LOGIC_VECTOR(2 downto 0);
        zero:              out STD_LOGIC;
        pc:                buffer STD_LOGIC_VECTOR(31 downto 0);
        instr:             in  STD_LOGIC_VECTOR(31 downto 0);
        aluout, writedata: buffer STD_LOGIC_VECTOR(31 downto 0);
        readdata:          in  STD_LOGIC_VECTOR(31 downto 0));
 end component;
 signal memtoreg, alusrc, regdst, regwrite, jump, pcsrc:
  STD_LOGIC;
 signal zero: STD_LOGIC;
 signal alucontrol: STD_LOGIC_VECTOR(2 downto 0);
begin
  cont: controller port map(instr(31 downto 26), instr
                            (5 downto 0), zero, memtoreg,
                            memwrite, pcsrc, alusrc, regdst,
                            regwrite, jump, alucontrol);
  dp: datapath port map(clk, reset, memtoreg, pcsrc, alusrc,
                        regdst, regwrite, jump, alucontrol,
                        zero, pc, instr, aluout, writedata,
                        readdata);
end;
```

HDL Example 7.2 CONTROLLER

Verilog

```
module controller(input  [5:0] op, funct,
                  input        zero,
                  output       memtoreg, memwrite,
                  output       pcsrc, alusrc,
                  output       regdst, regwrite,
                  output       jump,
                  output [2:0] alucontrol);

  wire [1:0] aluop;
  wire       branch;

  maindec md(op, memtoreg, memwrite, branch,
             alusrc, regdst, regwrite, jump,
             aluop);
  aludec ad(funct, aluop, alucontrol);

  assign pcsrc = branch & zero;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity controller is -- single cycle control decoder
 port(op, funct:           in  STD_LOGIC_VECTOR(5 downto 0);
      zero:                in  STD_LOGIC;
      memtoreg, memwrite: out STD_LOGIC;
      pcsrc, alusrc:      out STD_LOGIC;
      regdst, regwrite:   out STD_LOGIC;
      jump:               out STD_LOGIC;
      alucontrol:         out STD_LOGIC_VECTOR(2 downto 0));
end;

architecture struct of controller is
 component maindec
   port(op:                 in  STD_LOGIC_VECTOR(5 downto 0);
        memtoreg, memwrite: out STD_LOGIC;
        branch, alusrc:     out STD_LOGIC;
        regdst, regwrite:   out STD_LOGIC;
        jump:               out STD_LOGIC;
        aluop:              out STD_LOGIC_VECTOR(1 downto 0));
 end component;
 component aludec
   port(funct:      in  STD_LOGIC_VECTOR(5 downto 0);
        aluop:      in  STD_LOGIC_VECTOR(1 downto 0);
        alucontrol: out STD_LOGIC_VECTOR(2 downto 0));
 end component;
 signal aluop:  STD_LOGIC_VECTOR(1 downto 0);
 signal branch: STD_LOGIC;
begin
 md: maindec port map(op, memtoreg, memwrite, branch,
                      alusrc, regdst, regwrite, jump, aluop);
 ad: aludec port map(funct, aluop, alucontrol);

 pcsrc <= branch and zero;
end;
```

HDL Example 7.3 MAIN DECODER

Verilog

```
module maindec(input  [5:0] op,
               output       memtoreg, memwrite,
               output       branch, alusrc,
               output       regdst, regwrite,
               output       jump,
               output [1:0] aluop);

 reg [8:0] controls;

 assign {regwrite, regdst, alusrc,
         branch, memwrite,
         memtoreg, jump, aluop} = controls;

 always @(*)
  case(op)
    6'b000000: controls <= 9'b110000010; //Rtyp
    6'b100011: controls <= 9'b101001000; //LW
    6'b101011: controls <= 9'b001010000; //SW
    6'b000100: controls <= 9'b000100001; //BEQ
    6'b001000: controls <= 9'b101000000; //ADDI
    6'b000010: controls <= 9'b000000100; //J
    default:   controls <= 9'bxxxxxxxxx; //???
  endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity maindec is -- main control decoder
 port(op:                 in  STD_LOGIC_VECTOR(5 downto 0);
      memtoreg, memwrite: out STD_LOGIC;
      branch, alusrc:     out STD_LOGIC;
      regdst, regwrite:   out STD_LOGIC;
      jump:               out STD_LOGIC;
      aluop:              out STD_LOGIC_VECTOR(1 downto 0));
end;

architecture behave of maindec is
 signal controls: STD_LOGIC_VECTOR(8 downto 0);
begin
 process(op) begin
  case op is
     when "000000" => controls <= "110000010"; -- Rtyp
     when "100011" => controls <= "101001000"; -- LW
     when "101011" => controls <= "001010000"; -- SW
     when "000100" => controls <= "000100001"; -- BEQ
     when "001000" => controls <= "101000000"; -- ADDI
     when "000010" => controls <= "000000100"; -- J
     when others   => controls <= "---------"; -- illegal op
  end case;
 end process;

 regwrite <= controls(8);
 regdst   <= controls(7);
 alusrc   <= controls(6);
 branch   <= controls(5);
 memwrite <= controls(4);
 memtoreg <= controls(3);
 jump     <= controls(2);
 aluop    <= controls(1 downto 0);
end;
```

HDL Example 7.4 ALU DECODER

Verilog

```
module aludec (input      [5:0] funct,
               input      [1:0] aluop,
               output reg [2:0] alucontrol);

always @(*)
 case (aluop)
  2'b00: alucontrol <= 3'b010; // add
  2'b01: alucontrol <= 3'b110; // sub
  default: case(funct)         // RTYPE
      6'b100000: alucontrol <= 3'b010; // ADD
      6'b100010: alucontrol <= 3'b110; // SUB
      6'b100100: alucontrol <= 3'b000; // AND
      6'b100101: alucontrol <= 3'b001; // OR
      6'b101010: alucontrol <= 3'b111; // SLT
      default:   alucontrol <= 3'bxxx; // ???
    endcase
  endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity aludec is -- ALU control decoder
  port(funct:      in  STD_LOGIC_VECTOR(5 downto 0);
       aluop:      in  STD_LOGIC_VECTOR(1 downto 0);
       alucontrol: out STD_LOGIC_VECTOR(2 downto 0));
end;

architecture behave of aludec is
begin
 process(aluop, funct) begin
  case aluop is
    when "00" => alucontrol <= "010"; -- add (for lb/sb/addi)
    when "01" => alucontrol <= "110"; -- sub (for beq)
    when others => case funct is          -- R-type instructions
                       when "100000" => alucontrol <=
                         "010"; -- add
                       when "100010" => alucontrol <=
                         "110"; -- sub
                       when "100100" => alucontrol <=
                         "000"; -- and
                       when "100101" => alucontrol <=
                         "001"; -- or
                       when "101010" => alucontrol <=
                         "111"; -- slt
                       when others   => alucontrol <=
                         "---"; -- ???
                   end case;
  end case;
 end process;
end;
```

HDL Example 7.5 DATAPATH

Verilog

```
module datapath(input          clk, reset,
                input          memtoreg, pcsrc,
                input          alusrc, regdst,
                input          regwrite, jump,
                input  [2:0]   alucontrol,
                output         zero,
                output [31:0]  pc,
                input  [31:0]  instr,
                output [31:0]  aluout, writedata,
                input  [31:0]  readdata);

  wire [4:0]  writereg;
  wire [31:0] pcnext, pcnextbr, pcplus4, pcbranch;
  wire [31:0] signimm, signimmsh;
  wire [31:0] srca, srcb;
  wire [31:0] result;

  // next PC logic
  flopr #(32) pcreg(clk, reset, pcnext, pc);
  adder       pcadd1(pc, 32'b100, pcplus4);
  sl2         immsh(signimm, signimmsh);
  adder       pcadd2(pcplus4, signimmsh, pcbranch);
  mux2 #(32)  pcbrmux(pcplus4, pcbranch, pcsrc,
                      pcnextbr);
  mux2 #(32)  pcmux(pcnextbr, {pcplus4[31:28],
                    instr[25:0], 2'b00},
                    jump, pcnext);

  // register file logic
  regfile     rf(clk, regwrite, instr[25:21],
                 instr[20:16], writereg,
                 result, srca, writedata);
  mux2 #(5)   wrmux(instr[20:16], instr[15:11],
                    regdst, writereg);
  mux2 #(32)  resmux(aluout, readdata,
                     memtoreg, result);
  signext     se(instr[15:0], signimm);

  // ALU logic
  mux2 #(32)  srcbmux(writedata, signimm, alusrc,
                      srcb);
  alu         alu(srca, srcb, alucontrol,
                  aluout, zero);
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all; use
IEEE.STD_LOGIC_ARITH.all;
entity datapath is -- MIPS datapath
  port(clk, reset:        in     STD_LOGIC;
       memtoreg, pcsrc:   in     STD_LOGIC;
       alusrc, regdst:    in     STD_LOGIC;
       regwrite, jump:    in     STD_LOGIC;
       alucontrol:        in     STD_LOGIC_VECTOR(2 downto 0);
       zero:              out    STD_LOGIC;
       pc:                buffer STD_LOGIC_VECTOR(31 downto 0);
       instr:             in     STD_LOGIC_VECTOR(31 downto 0);
       aluout, writedata: buffer STD_LOGIC_VECTOR(31 downto 0);
       readdata:          in     STD_LOGIC_VECTOR(31 downto 0));
end;

architecture struct of datapath is
  component alu
    port(a, b:       in  STD_LOGIC_VECTOR(31 downto 0);
         alucontrol: in  STD_LOGIC_VECTOR(2 downto 0);
         result:     buffer STD_LOGIC_VECTOR(31 downto 0);
         zero:       out STD_LOGIC);
  end component;
  component regfile
    port(clk:           in  STD_LOGIC;
         we3:           in  STD_LOGIC;
         ra1, ra2, wa3: in  STD_LOGIC_VECTOR(4 downto 0);
         wd3:           in  STD_LOGIC_VECTOR(31 downto 0);
         rd1, rd2:      out STD_LOGIC_VECTOR(31 downto 0));
  end component;
  component adder
    port(a, b: in  STD_LOGIC_VECTOR(31 downto 0);
         y:    out STD_LOGIC_VECTOR(31 downto 0));
  end component;
  component sl2
    port(a: in  STD_LOGIC_VECTOR(31 downto 0);
         y: out STD_LOGIC_VECTOR(31 downto 0));
  end component;
  component signext
    port(a: in  STD_LOGIC_VECTOR(15 downto 0);
         y: out STD_LOGIC_VECTOR(31 downto 0));
  end component;
  component flopr generic(width: integer);
    port(clk, reset: in  STD_LOGIC;
         d:          in  STD_LOGIC_VECTOR(width-1 downto 0);
         q:          out STD_LOGIC_VECTOR(width-1 downto 0));
  end component;
```

Continuação do VHDL

```
  component mux2 generic(width: integer);
    port(d0, d1: in  STD_LOGIC_VECTOR(width-1 downto 0);
         s:      in  STD_LOGIC;
         y:      out STD_LOGIC_VECTOR(width-1 downto 0));
  end component;
  signal writereg: STD_LOGIC_VECTOR(4 downto 0);
  signal pcjump, pcnext, pcnextbr,
         pcplus4, pcbranch: STD_LOGIC_VECTOR(31 downto 0);
  signal signimm, signimmsh: STD_LOGIC_VECTOR(31 downto 0);
  signal srca, srcb, result: STD_LOGIC_VECTOR(31 downto 0);
begin
  -- next PC logic
  pcjump <= pcplus4(31 downto 28) & instr(25 downto 0) & "00";
  pcreg: flopr generic map(32) port map(clk, reset, pcnext, pc);
  pcadd1: adder port map(pc, X"00000004", pcplus4);
  immsh: sl2 port map(signimm, signimmsh);
  pcadd2: adder port map(pcplus4, signimmsh, pcbranch);
  pcbrmux: mux2 generic map(32) port map(pcplus4, pcbranch,
                                         pcsrc, pcnextbr);
  pcmux: mux2 generic map(32) port map(pcnextbr, pcjump, jump,
                                       pcnext);

  -- register file logic
  rf: regfile port map(clk, regwrite, instr(25 downto 21),
                       instr(20 downto 16), writereg, result, srca,
                       writedata);
  wrmux: mux2 generic map(5) port map(instr(20 downto 16),
  instr(15 downto 11), regdst, writereg);
  resmux: mux2 generic map(32) port map(aluout, readdata,
                                        memtoreg, result);
  se: signext port map(instr(15 downto 0), signimm);

  -- ALU logic
  srcbmux: mux2 generic map (32) port map(writedata, signimm,
                                          alusrc, srcb);
  mainalu: alu port map(srca, srcb, alucontrol, aluout, zero);
end;
```

HDL Example 7.6 REGISTER FILE

Verilog

```
module regfile(input         clk,
               input         we3,
               input  [4:0]  ra1, ra2, wa3,
               input  [31:0] wd3,
               output [31:0] rd1, rd2);

  reg [31:0] rf[31:0];

  // three ported register file
  // read two ports combinationally
  // write third port on rising edge of clock
  // register 0 hardwired to 0

  always @(posedge clk)
    if(we3) rf[wa3] <= wd3;

  assign rd1 = (ra1 != 0) ? rf[ra1] : 0;
  assign rd2 = (ra2 != 0) ? rf[ra2] : 0;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity regfile is -- three-port register file
  port(clk:           in  STD_LOGIC;
       we3:           in  STD_LOGIC;
       ra1, ra2, wa3: in  STD_LOGIC_VECTOR(4 downto 0);
       wd3:           in  STD_LOGIC_VECTOR(31 downto 0);
       rd1, rd2:      out STD_LOGIC_VECTOR(31 downto 0));
end;

architecture behave of regfile is
  type ramtype is array(31 downto 0) of STD_LOGIC_VECTOR(31
                        downto 0);
  signal mem: ramtype;
begin
  -- three-ported register file
  -- read two ports combinationally
  -- write third port on rising edge of clock
  process(clk) begin
   if clk'event and clk = '1' then
    if we3 = '1' then mem(CONV_INTEGER(wa3)) <= wd3;
    end if;
   end if;
  end process;
  process(ra1, ra2) begin
    if(conv_integer(ra1) = 0) then rd1 <= X"00000000";
       -- register 0 holds 0
    else rd1 <= mem(CONV_INTEGER(ra1));
    end if;
    if(conv_integer(ra2) = 0) then rd2 <= X"00000000";
    else rd2 <= mem(CONV_INTEGER(ra2));
    end if;
  end process;
end;
```

HDL Example 7.7 ADDER

Verilog

```
module adder(input  [31:0] a, b,
             output [31:0] y);

  assign y = a + b;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity adder is -- adder
  port(a, b: in  STD_LOGIC_VECTOR(31 downto 0);
       y:    out STD_LOGIC_VECTOR(31 downto 0));
end;

architecture behave of adder is
begin
  y <= a + b;
end;
```

HDL Example 7.8 LEFT SHIFT (MULTIPLY BY 4)

Verilog

```
module sl2(input  [31:0] a,
           output [31:0] y);

  // shift left by 2
  assign y = {a[29:0], 2'b00};
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity sl2 is -- shift left by 2
  port(a: in  STD_LOGIC_VECTOR(31 downto 0);
       y: out STD_LOGIC_VECTOR(31 downto 0));
end;

architecture behave of sl2 is
begin
  y <= a(29 downto 0) & "00";
end;
```

HDL Example 7.9 SIGN EXTENSION

Verilog

```
module signext (input  [15:0] a,
                output [31:0] y);

  assign y = {{16{a[15]}}, a};
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity signext is -- sign extender
port(a: in  STD_LOGIC_VECTOR (15 downto 0);
     y: out STD_LOGIC_VECTOR (31 downto 0));
end;

architecture behave of signext is
begin
  y <= X"0000" & a when a(15) = '0' else X"ffff" & a;
end;
```

HDL Example 7.10 RESETTABLE FLIP-FLOP

Verilog

```
module flopr #(parameter WIDTH = 8)
              (input                  clk, reset,
               input      [WIDTH-1:0] d,
               output reg [WIDTH-1:0] q);

  always @(posedge clk, posedge reset)
    if(reset) q <= 0;
    else      q <= d;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all; use
IEEE.STD_LOGIC_ARITH.all;
entity flopr is -- flip-flop with synchronous reset
  generic(width: integer);
  port(clk, reset: in  STD_LOGIC;
       d:          in  STD_LOGIC_VECTOR(width-1 downto 0);
       q:          out STD_LOGIC_VECTOR(width-1 downto 0));
end;

architecture asynchronous of flopr is
begin
  process(clk, reset) begin
    if reset = '1' then q <= CONV_STD_LOGIC_VECTOR(0, width);
    elsif clk'event and clk = '1' then
      q <= d;
    end if;
  end process;
end;
```

HDL Example 7.11 2:1 MULTIPLEXER

Verilog

```
module mux2 #(parameter WIDTH = 8)
             (input  [WIDTH-1:0] d0, d1,
              input              s,
              output [WIDTH-1:0] y);

  assign y = s ? d1 : d0;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity mux2 is -- two-input multiplexer
  generic(width: integer);
  port(d0, d1: in  STD_LOGIC_VECTOR(width-1 downto 0);
       s:      in  STD_LOGIC;
       y:      out STD_LOGIC_VECTOR(width-1 downto 0));
end;

architecture behave of mux2 is
begin
  y <= d0 when s = '0' else d1;
end;
```

```
# mipstest.asm
# David_Harris@hmc.edu 9 November 2005
#
# Test the MIPS processor.
# add, sub, and, or, slt, addi, lw, sw, beq, j
# If successful, it should write the value 7 to address 84

#            Assembly              Description              Address     Machine
main:        addi $2, $0, 5        # initialize $2 = 5       0           20020005
             addi $3, $0, 12       # initialize $3 = 12      4           2003000c
             addi $7, $3, -9       # initialize $7 = 3       8           2067fff7
             or   $4, $7, $2       # $4 <= 3 or 5 = 7        c           00e22025
             and  $5, $3, $4       # $5 <= 12 and 7 = 4      10          00642824
             add  $5, $5, $4       # $5 = 4 + 7 = 11         14          00a42820
             beq  $5, $7, end      # shouldn't be taken      18          10a7000a
             slt  $4, $3, $4       # $4 = 12 < 7 = 0         1c          0064202a
             beq  $4, $0, around   # should be taken         20          10800001
             addi $5, $0, 0        # shouldn't happen        24          20050000
around:      slt  $4, $7, $2       # $4 = 3 < 5 = 1          28          00e2202a
             add  $7, $4, $5       # $7 = 1 + 11 = 12        2c          00853820
             sub  $7, $7, $2       # $7 = 12 - 5 = 7         30          00e23822
             sw   $7, 68($3)       # [80] = 7                34          ac670044
             lw   $2, 80($0)       # $2 = [80] = 7           38          8c020050
             j    end              # should be taken         3c          08000011
             addi $2, $0, 1        # shouldn't happen        40          20020001
end:         sw   $2, 84($0)       # write adr 84 = 7        44          ac020054
```

Figure 7.60 Assembly and machine code for MIPS test program

```
20020005
2003000c
2067fff7
00e22025
00642824
00a42820
10a7000a
0064202a
10800001
20050000
00e2202a
00853820
00e23822
ac670044
8c020050
08000011
20020001
ac020054
```

Figure 7.61 Contents of `memfile.dat`

HDL Example 7.12 MIPS TESTBENCH

Verilog

```
module testbench();

  reg     clk;
  reg     reset;

  wire [31:0] writedata, dataadr;
  wire        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 === 84 & writedata === 7) begin
          $display("Simulation succeeded");
          $stop;
        end else if(dataadr !== 80) begin
          $display("Simulation failed");
          $stop;
        end
      end
    end
endmodule
```

VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all; use IEEE.STD_LOGIC_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 7 gets written to address 84
  -- at end of program
  process (clk) begin
    if (clk'event and clk = '0' and memwrite = '1') then
     if (conv_integer(dataadr) = 84 and conv_integer
         (writedata) = 7) then
        report "Simulation succeeded";
      elsif (dataadr /= 80) then
        report "Simulation failed";
      end if;
    end if;
  end process;
end;
```

HDL Example 7.13 MIPS TOP-LEVEL MODULE

Verilog

```
module top(input         clk, reset,
           output [31:0] writedata, dataadr,
           output        memwrite);

  wire [31:0] pc, instr, readdata;

  // instantiate processor and memories
  mips mips(clk, reset, pc, instr, memwrite, dataadr,
            writedata, readdata);
  imem imem(pc[7:2], instr);
  dmem dmem(clk, memwrite, dataadr, writedata,
            readdata);
endmodule
```

VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all; use IEEE.STD_LOGIC_UNSIGNED.all;
entity top is -- top-level design for testing
  port(clk, reset:          in     STD_LOGIC;
       writedata, dataadr:  buffer STD_LOGIC_VECTOR(31 downto
                                                    0);
       memwrite:            buffer STD_LOGIC);
end;

architecture test of top is
  component mips
    port(clk, reset:        in  STD_LOGIC;
         pc:                out STD_LOGIC_VECTOR(31 downto 0);
         instr:             in  STD_LOGIC_VECTOR(31 downto 0);
         memwrite:          out STD_LOGIC;
         aluout, writedata: out STD_LOGIC_VECTOR(31 downto 0);
         readdata:          in  STD_LOGIC_VECTOR(31 downto 0));
  end component;
  component imem
    port(a:  in  STD_LOGIC_VECTOR(5 downto 0)
         rd: out STD_LOGIC_VECTOR(31 downto 0));
  end component;
  component dmem
    port(clk, we: in  STD_LOGIC;
         a, wd:   in  STD_LOGIC_VECTOR(31 downto 0);
         rd:      out STD_LOGIC_VECTOR(31 downto 0));
  end component;
  signal pc, instr,
         readdata: STD_LOGIC_VECTOR(31 downto 0);
begin
  -- instantiate processor and memories
  mips1: mips port map(clk, reset, pc, instr, memwrite,
                       dataadr, writedata, readdata);
  imem1: imem port map(pc(7 downto 2), instr);
  dmem1: dmem port map(clk, memwrite, dataadr, writedata,
                       readdata);
end;
```

HDL Example 7.14 MIPS DATA MEMORY

```
module dmem (input          clk, we,
             input  [31:0] a, wd,
             output [31:0] rd);

  reg [31:0] RAM[63:0];

  assign rd = RAM[a[31:2]]; // word aligned

  always @(posedge clk)
    if (we)
      RAM[a[31:2]] <= wd;
endmodule
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all; use STD.TEXTIO.all;
use IEEE.STD_LOGIC_UNSIGNED.all; use IEEE.STD_LOGIC_ARITH.all;
entity dmem is -- data memory
 port(clk, we: in  STD_LOGIC;
      a, wd:   in  STD_LOGIC_VECTOR(31 downto 0);
      rd:      out STD_LOGIC_VECTOR(31 downto 0));
end;

architecture behave of dmem is
begin
  process is
    type ramtype is array(63 downto 0) of STD_LOGIC_VECTOR
                          (31 downto 0);
    variable mem: ramtype;
  begin
    -- read or write memory
    loop
      if clk'event and clk = '1' then
          if (we = '1') then mem(CONV_INTEGER(a(7 downto
                                              2))): = wd;
          end if;
      end if;
      rd <= mem(CONV_INTEGER(a(7 downto 2)));
      wait on clk, a;
    end loop;

  end process;
end;
```

HDL Example 7.15 MIPS INSTRUCTION MEMORY

Verilog

```
module imem(input  [5:0]  a,
            output [31:0] rd);

  reg [31:0] RAM[63:0];

  initial
    begin
      $readmemh("memfile.dat",RAM);
    end

  assign rd = RAM[a]; // word aligned
endmodule
```

VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all; use STD.TEXTIO.all;
use IEEE.STD_LOGIC_UNSIGNED.all; use IEEE.STD_LOGIC_ARITH.all;

entity imem is -- instruction memory
  port(a:   in  STD_LOGIC_VECTOR(5 downto 0);
       rd:  out STD_LOGIC_VECTOR(31 downto 0));
end;

architecture behave of imem is
begin
  process is
    file mem_file: TEXT;
    variable L: line;
    variable ch: character;
    variable index, result: integer;
    type ramtype is array(63 downto 0) of STD_LOGIC_VECTOR
                          (31 downto 0);
    variable mem: ramtype;
  begin
    -- initialize memory from file
    for i in 0 to 63 loop -- set all contents low
      mem(conv_integer(i)) := CONV_STD_LOGIC_VECTOR(0, 32);
    end loop;
    index := 0;
    FILE_OPEN(mem_file, "C:/mips/memfile.dat", READ_MODE);
    while not endfile(mem_file) loop
      readline(mem_file, L);
      result := 0;
      for i in 1 to 8 loop
        read(L, ch);
        if '0' <= ch and ch <= '9' then
            result := result*16 + character'pos(ch) -
                      character'pos('0');
        elsif 'a' <= ch and ch <= 'f' then
          result := result*16 + character'pos(ch) -
                    character'pos('a') + 10;
        else report "Format error on line" & integer'image
             (index) severity error;
        end if;
      end loop;
      mem(index) := CONV_STD_LOGIC_VECTOR(result, 32);
      index := index + 1;
    end loop;

    -- read memory
    loop
      rd <= mem(CONV_INTEGER(a));
      wait on a;
    end loop;
  end process;
end;
```