CSC/CPE 142 – Advanced Computer Organization

Term Project Phase III

Michael Colson

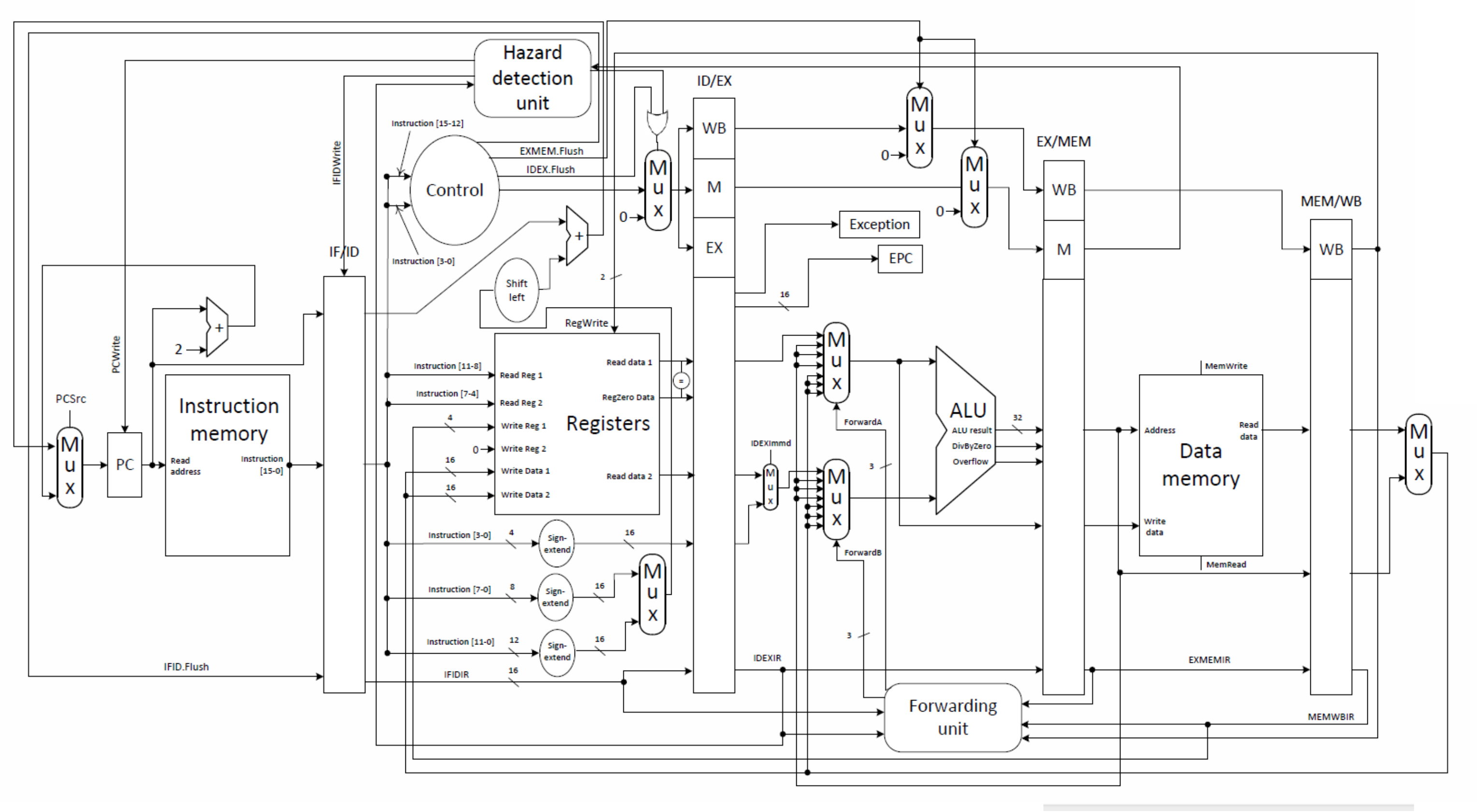
California State University, Sacramento

Professor Behnam S. Arad

5/10/2013

Table of Contents

Datapath Diagram



ALU Module

```
module ALU (OpCode, FuncCode, IDEXop2, stall, A, B, ALUOut1, ALUOut2, Zero, Sign, Overflow, DivByZero);
    input [3:0] OpCode, FuncCode, IDEXop2;
    input stall;
    input signed [15:0] A, B;
    output reg signed [15:0] ALUOut1, ALUOut2;
    output wire Zero, Sign, Overflow, DivByZero;
    reg [15:0] temp;
    wire signed [15:0] Sum, Difference;

    assign Sum = A + B;
    assign Difference = A - B;
    assign Zero = (ALUOut1 == 0);
    assign Sign = ALUOut1 < 0;
    assign Overflow = ~stall & (OpCode == 0 & (FuncCode == 0 & ((A > 0 & B > 0 & Sum < 0) |
        (A < 0 & B < 0 & Sum > 0)))) | (OpCode == 0 &
        (FuncCode == 1 & ((A > 0 & B < 0 & Difference < 0) |
        (A < 0 & B > 0 & Difference > 0))));
    assign DivByZero = OpCode == 0 & FuncCode == 5 & B == 0;

    always @(FuncCode, A, B) begin
        if(OpCode == 0) begin
        case (FuncCode)
            0:ALUOut1 <= A + B;            // addition
            1:ALUOut1 <= A - B;            // subtraction
            2:ALUOut1 <= A & B;            // bitwise AND
            3:ALUOut1 <= A | B;            // bitwise OR
            4:{ALUOut2, ALUOut1} <= A * B; // multiplication
            5:begin
                 if(B != 0)
                 begin
                     ALUOut1 <= A / B;      // division
                     ALUOut2 <= A % B;
                 end
              end
            8:ALUOut1 <= A << IDEXop2;            // shift left
            9:ALUOut1 <= A >> IDEXop2;            // shift right
            10:begin                       // rotate left
               case(IDEXop2)
                   0:ALUOut1 <= A;
                   1:begin
                         temp = A << 1;
                         ALUOut1 <= {temp[15:1], A[15]};
                     end
                   2:begin
                         temp = A << 2;
                         ALUOut1 <= {{temp[15:2], A[15]}, A[14]};
                     end
                   3:begin
                         temp = A << 3;
                         ALUOut1 <= {{{temp[15:3], A[15]}, A[14]}, A[13]};
                     end
                   4:begin
                         temp = A << 4;
                         ALUOut1 <= {{{{temp[15:4], A[15]}, A[14]}, A[13]}, A[12]};
                     end
    5:begin
        temp = A << 5;
        ALUOut1 <= {{{{{temp[15:5], A[15]}, A[14]}, A[13]}, A[12]}, A[11]};
     end
    6:begin
        temp = A << 6;
        ALUOut1 <= {{{{{{temp[15:6], A[15]}, A[14]}, A[13]}, A[12]}, A[11]}, A[10]};
     end
    7:begin
        temp = A << 7;
        ALUOut1 <= {{{{{{{temp[15:7], A[15]}, A[14]}, A[13]}, A[12]}, A[11]}, A[10]},
          A[9]};
     end
    8:begin
        temp = A << 8;
        ALUOut1 <= {{{{{{{{temp[15:8], A[15]}, A[14]}, A[13]}, A[12]}, A[11]}, A[10]},
          A[9]}, A[8]};
     end
    9:begin
        temp = A << 9;
        ALUOut1 <= {{{{{{{{{temp[15:9], A[15]}, A[14]}, A[13]}, A[12]}, A[11]}, A[10]},
          A[9]}, A[8]}, A[7]};
     end
    10:begin
       temp = A << 10;
       ALUOut1 <= {{{{{{{{{{temp[15:10], A[15]}, A[14]}, A[13]}, A[12]}, A[11]}, A[10]},
         A[9]}, A[8]}, A[7]}, A[6]};
      end
    11:begin
       temp = A << 11;
       ALUOut1 <= {{{{{{{{{{{temp[15:11], A[15]}, A[14]}, A[13]}, A[12]}, A[11]}, A[10]},
         A[9]}, A[8]}, A[7]}, A[6]}, A[5]};
      end
    12:begin
       temp = A << 12;
       ALUOut1 <= {{{{{{{{{{{{temp[15:12], A[15]}, A[14]}, A[13]}, A[12]}, A[11]}, A[10]},
         A[9]}, A[8]}, A[7]}, A[6]}, A[5]}, A[4]};
      end
    13:begin
       temp = A << 13;
       ALUOut1 <= {{{{{{{{{{{{{temp[15:13], A[15]}, A[14]}, A[13]}, A[12]}, A[11]}, A[10]},
         A[9]}, A[8]}, A[7]}, A[6]}, A[5]}, A[4]}, A[3]};
      end
    14:begin
       temp = A << 14;
       ALUOut1 <= {{{{{{{{{{{{{{temp[15:14], A[15]}, A[14]}, A[13]}, A[12]}, A[11]},
         A[10]}, A[9]}, A[8]}, A[7]}, A[6]}, A[5]}, A[4]}, A[3]}, A[2]};
      end
    15:begin
       temp = A << 15;
       ALUOut1 <= {{{{{{{{{{{{{{{temp[15], A[15]}, A[14]}, A[13]}, A[12]}, A[11]}, A[10]},
         A[9]}, A[8]}, A[7]}, A[6]}, A[5]}, A[4]}, A[3]}, A[2]}, A[1]};
      end
    endcase
  end
11:begin                    // rotate right
  case(IDEXop2)
```

```
0:ALUOut1 <= A;
1:begin
    temp = A >> 1;
    ALUOut1 <= {A[0], temp[14:0]};
  end
2:begin
    temp = A >> 2;
    ALUOut1 <= {{A[1], A[0]}, temp[13:0]};
  end
3:begin
     temp = A >> 3;
     ALUOut1 <= {{{A[2], A[1]}, A[0]}, temp[12:0]};
  end
4:begin
     temp = A >> 4;
     ALUOut1 <= {{{{A[3], A[2]}, A[1]}, A[0]}, temp[11:0]};
  end
5:begin
     temp = A >> 5;
     ALUOut1 <= {{{{{A[4], A[3]}, A[2]}, A[1]}, A[0]}, temp[10:0]};
  end
6:begin
     temp = A >> 6;
     ALUOut1 <= {{{{{{A[5], A[4]}, A[3]}, A[2]}, A[1]}, A[0]}, temp[9:0]};
  end
7:begin
     temp = A >> 7;
     ALUOut1 <= {{{{{{{A[6], A[5]}, A[4]}, A[3]}, A[2]}, A[1]}, A[0]}, temp[8:0]};
  end
8:begin
     temp = A >> 8;
     ALUOut1 <= {{{{{{{{A[7], A[6]}, A[5]}, A[4]}, A[3]}, A[2]}, A[1]},
         A[0]}, temp[7:0]};
  end
9:begin
     temp = A >> 9;
     ALUOut1 <= {{{{{{{{{A[8], A[7]}, A[6]}, A[5]}, A[4]}, A[3]}, A[2]},
         A[1]}, A[0]}, temp[6:0]};
  end
10:begin
     temp = A >> 10;
     ALUOut1 <= {{{{{{{{{{A[9], A[8]}, A[7]}, A[6]}, A[5]}, A[4]}, A[3]},
         A[2]}, A[1]}, A[0]}, temp[5:0]};
   end
11:begin
     temp = A >> 11;
     ALUOut1 <= {{{{{{{{{{{A[10], A[9]}, A[8]}, A[7]}, A[6]}, A[5]}, A[4]},
         A[3]}, A[2]}, A[1]}, A[0]}, temp[4:0]};
   end
12:begin
     temp = A >> 12;
     ALUOut1 <= {{{{{{{{{{{{A[11], A[10]}, A[9]}, A[8]}, A[7]}, A[6]}, A[5]},
         A[4]}, A[3]}, A[2]}, A[1]}, A[0]}, temp[3:0]};
   end
13:begin
     temp = A >> 13;
     ALUOut1 <= {{{{{{{{{{{{{A[12], A[11]}, A[10]}, A[9]}, A[8]}, A[7]},
               A[6]}, A[5]}, A[4]}, A[3]}, A[2]}, A[1]}, A[0]}, temp[2:0]};
            end
          14:begin
             temp = A >> 14;
             ALUOut1 <= {{{{{{{{{{{{{{A[13], A[12]}, A[11]}, A[10]}, A[9]}, A[8]},
               A[7]}, A[6]}, A[5]}, A[4]}, A[3]}, A[2]}, A[1]}, A[0]}, temp[1:0]};
            end
          15:begin
             temp = A >> 15;
             ALUOut1 <= {{{{{{{{{{{{{{{A[14], A[13]}, A[12]}, A[11]}, A[10]}, A[9]},
               A[8]}, A[7]}, A[6]}, A[5]}, A[4]}, A[3]}, A[2]}, A[1]}, A[0]}, temp[0]};
            end
          endcase
        end
      default:
        begin
          ALUOut1 <= 0;
          ALUOut2 <= 0;
        end
    endcase
    end
  end
endmodule
```

ALU Stimulus

```
`include "alu.v"

module stimulus;

reg [3:0] OpCode = 0, FuncCode = 0, RegOp2 = 3;
reg stall = 0;
reg signed [15:0] A = 'h0F00, B = 'h0900;
wire signed [15:0] ALUOut1, ALUOut2;
wire Zero, Sign, Overflow, DivByZero;

initial
   $vcdpluson;

initial
   $monitor("OpCode = %h FuncCode = %h A = %h B = %h ALUOut1 = %h ALUOut2 = %h Zero = %b Sign =
%b Overflow = %b DivByZero = %b", OpCode, FuncCode, A, B, ALUOut1, ALUOut2, Zero, Sign, Overflow,
DivByZero);

ALU alu(OpCode, FuncCode, RegOp2, stall, A, B, ALUOut1, ALUOut2, Zero, Sign, Overflow, DivByZero);

initial
   begin
      #20 FuncCode = 1;
      #20 FuncCode = 2;
      #20 FuncCode = 3;
      #20 FuncCode = 4;
      #20 FuncCode = 5;
      #20 FuncCode = 8;
      #20 FuncCode = 9;
      #20 FuncCode = 10;
      #20 FuncCode = 11;
   end

endmodule
```

ALU Results

```
Chronologic VCS simulator copyright 1991-2009
Contains Synopsys proprietary information.
Compiler version D-2009.12; Runtime version D-2009.12;  May  6 04:16 2013
VCD+ Writer D-2009.12 Copyright 2009 Synopsys Inc.

OpCode = 0 FuncCode = 0 A = 0f00 B = 0900 ALUOut1 = 1800 ALUOut2 = xxxx Zero = 0 Sign = 0 Overflow = 0
DivByZero = 0

OpCode = 0 FuncCode = 1 A = 0f00 B = 0900 ALUOut1 = 0600 ALUOut2 = xxxx Zero = 0 Sign = 0 Overflow = 0
DivByZero = 0

OpCode = 0 FuncCode = 2 A = 0f00 B = 0900 ALUOut1 = 0900 ALUOut2 = xxxx Zero = 0 Sign = 0 Overflow = 0
DivByZero = 0

OpCode = 0 FuncCode = 3 A = 0f00 B = 0900 ALUOut1 = 0f00 ALUOut2 = xxxx Zero = 0 Sign = 0 Overflow = 0
DivByZero = 0

OpCode = 0 FuncCode = 4 A = 0f00 B = 0900 ALUOut1 = 0000 ALUOut2 = 0087 Zero = 1 Sign = 0 Overflow = 0
DivByZero = 0

OpCode = 0 FuncCode = 5 A = 0f00 B = 0900 ALUOut1 = 0001 ALUOut2 = 0600 Zero = 0 Sign = 0 Overflow = 0
DivByZero = 0

OpCode = 0 FuncCode = 8 A = 0f00 B = 0900 ALUOut1 = 7800 ALUOut2 = 0600 Zero = 0 Sign = 0 Overflow = 0
DivByZero = 0

OpCode = 0 FuncCode = 9 A = 0f00 B = 0900 ALUOut1 = 01e0 ALUOut2 = 0600 Zero = 0 Sign = 0 Overflow = 0
DivByZero = 0

OpCode = 0 FuncCode = a A = 0f00 B = 0900 ALUOut1 = 7800 ALUOut2 = 0600 Zero = 0 Sign = 0 Overflow = 0
DivByZero = 0

OpCode = 0 FuncCode = b A = 0f00 B = 0900 ALUOut1 = 01e0 ALUOut2 = 0600 Zero = 0 Sign = 0 Overflow = 0
DivByZero = 0

           VCS Simulation Report
Time: 180
CPU Time:      0.410 seconds;       Data structure size:   0.0Mb
Mon May  6 04:16:33 2013
```

Control Module

```
module Control(OpCode, FuncCode, IFIDImmd, IFIDMemWrite, IFIDMemRead, IFIDRegWrite);
   input [3:0] OpCode;
   input [3:0] FuncCode;
   output wire IFIDImmd, IFIDMemWrite, IFIDMemRead;
   output wire [1:0] IFIDRegWrite;

   assign IFIDMemWrite = OpCode == 11;
   assign IFIDMemRead = OpCode == 8;

   assign IFIDImmd = (OpCode >= 4 & OpCode <= 6) |
      (OpCode == 0 & (FuncCode >= 8 & FuncCode <= 11));

   assign IFIDRegWrite = (OpCode == 0 &
      (FuncCode == 4 | FuncCode == 5)) ? 2 :
      (OpCode == 0 | OpCode == 8) ? 1 : 0;

endmodule
```

Control Stimulus

```
`include "control.v"

module stimulus;

reg [3:0] OpCode = 0, FuncCode = 0;
wire IFIDImmd, IFIDMemWrite, IFIDMemRead;
wire [1:0] IFIDRegWrite;

initial
    $vcdpluson;

initial
    $monitor("OpCode = %h FuncCode = %h IFIDImmd = %h IFIDMemWrite = %h IFIDMemRead = %h
IFIDRegWrite = %h", OpCode, FuncCode, IFIDImmd, IFIDMemWrite, IFIDMemRead, IFIDRegWrite);

Control Ctrl(OpCode, FuncCode, IFIDImmd, IFIDMemWrite, IFIDMemRead, IFIDRegWrite);

initial
    begin
        #20 FuncCode = 1;
        #20 FuncCode = 2;
        #20 FuncCode = 3;
        #20 FuncCode = 4;
        #20 FuncCode = 5;
        #20 FuncCode = 8;
        #20 FuncCode = 9;
        #20 FuncCode = 10;
        #20 FuncCode = 11;
        #20 OpCode = 8;
        #20 OpCode = 11;
        #20 OpCode = 4;
        #20 OpCode = 5;
        #20 OpCode = 6;
        #20 OpCode = 12;
        #20 OpCode = 15;
        #20 OpCode = 2;
        #20 OpCode = 3;
    end

endmodule
```

Control Results

```
Chronologic VCS simulator copyright 1991-2009
Contains Synopsys proprietary information.
Compiler version D-2009.12; Runtime version D-2009.12;  May  6 04:17 2013
VCD+ Writer D-2009.12 Copyright 2009 Synopsys Inc.

OpCode = 0 FuncCode = 0 IFIDImmd = 0 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 1
OpCode = 0 FuncCode = 1 IFIDImmd = 0 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 1
OpCode = 0 FuncCode = 2 IFIDImmd = 0 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 1
OpCode = 0 FuncCode = 3 IFIDImmd = 0 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 1
OpCode = 0 FuncCode = 4 IFIDImmd = 0 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 2
OpCode = 0 FuncCode = 5 IFIDImmd = 0 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 2
OpCode = 0 FuncCode = 8 IFIDImmd = 1 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 1
OpCode = 0 FuncCode = 9 IFIDImmd = 1 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 1
OpCode = 0 FuncCode = a IFIDImmd = 1 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 1
OpCode = 0 FuncCode = b IFIDImmd = 1 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 1
OpCode = 8 FuncCode = b IFIDImmd = 0 IFIDMemWrite = 0 IFIDMemRead = 1 IFIDRegWrite = 1
OpCode = b FuncCode = b IFIDImmd = 0 IFIDMemWrite = 1 IFIDMemRead = 0 IFIDRegWrite = 0
OpCode = 4 FuncCode = b IFIDImmd = 1 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 0
OpCode = 5 FuncCode = b IFIDImmd = 1 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 0
OpCode = 6 FuncCode = b IFIDImmd = 1 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 0
OpCode = c FuncCode = b IFIDImmd = 0 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 0
OpCode = f FuncCode = b IFIDImmd = 0 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 0
OpCode = 2 FuncCode = b IFIDImmd = 0 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 0
OpCode = 3 FuncCode = b IFIDImmd = 0 IFIDMemWrite = 0 IFIDMemRead = 0 IFIDRegWrite = 0

           V C S   S i m u l a t i o n   R e p o r t
Time: 360
CPU Time:      0.420 seconds;       Data structure size:   0.0Mb
Mon May  6 04:17:15 2013
```

CPU Module

```
`include "alu.v"
`include "control.v"
`include "DMemory.v"
`include "forwarding_unit.v"
`include "hazard_det_unit.v"
`include "IMemory.v"
`include "pc.v"
`include "regfile.v"

module CPU(CLOCK, RESET);
   input CLOCK, RESET;
   reg [15:0] EPC, IFIDIR, IDEXIR, EXMEMIR, MEMWBIR, IFIDPC,
      IDEXPC, EXMEMALUOut1, MEMWBALUOut1, EXMEM_Upper_Half_Product,
      temp, MEMWB_Upper_Half_Product, EXMEM_Remainder,
      MEMWB_Remainder, EXMEMoutForwardAMUX, IDEXReadData1, IDEXReadData2;
   reg [1:0] counter, IDEXRegWrite, EXMEMRegWrite, MEMWBRegWrite;
   reg Exception, IDEXImmd, IDEXMemWrite, IDEXMemRead, EXMEMMemWrite,
      EXMEMMemRead;
   wire signed [15:0] outForwardAMUX, outForwardBMUX, ALUOut1,
      ALUOut2, MEMWBALUOut2;
   wire [15:0] pc_in, pc_out, ReadData1, ReadData2, IMEM_ReadData,
      DMEM_ReadData, DMEM_WriteData, RegZeroData;
   wire invalidOpCode, invalidFuncCode, PCSrc, stall, halt,
      IFIDImmd, IFIDMemWrite, IFIDMemRead, overflow,
      divByZero;
   wire [1:0] IFIDRegWrite;
   wire [2:0] ForwardA, ForwardB;

   assign invalidOpCode = ~(IDEXIR[15:12] == 0 | IDEXIR[15:12] == 2 |
      IDEXIR[15:12] == 8 | IDEXIR[15:12] == 11 | IDEXIR[15:12] == 4 |
      IDEXIR[15:12] == 5 | IDEXIR[15:12] == 6 | IDEXIR[15:12] == 12 |
      IDEXIR[15:12] == 15);

   assign invalidFuncCode = IDEXIR[15:12] == 0 & ~(IDEXIR[3:0] == 0 |
      IDEXIR[3:0] == 1 | IDEXIR[3:0] == 2 | IDEXIR[3:0] == 3 |
      IDEXIR[3:0] == 4 | IDEXIR[3:0] == 5 | IDEXIR[3:0] == 8 |
      IDEXIR[3:0] == 9 | IDEXIR[3:0] == 10 | IDEXIR[3:0] == 11);

   assign PCSrc = ((IFIDIR[15:12] == 6) & (ReadData1 == RegZeroData)) |
      ((IFIDIR[15:12] == 5) & (ReadData1 > RegZeroData)) |
      ((IFIDIR[15:12] == 4) & (ReadData1 < RegZeroData)) |
      (IFIDIR[15:12] == 12);

   initial
   begin
      EXMEMALUOut1 <= 0;
      Exception <= 0;
      counter = 0;

      IFIDPC <= 0;
      IDEXPC <= 0;

      IFIDIR <= 'h2000;   // insert no-ops into pipeline registers
      IDEXIR <= 'h2000;
      EXMEMIR <= 'h2000;
```

```
        MEMWBIR <= 'h2000;

        IDEXReadData1 <= 0;
        IDEXReadData2 <= 0;
        IDEXImmd <= 0;
        IDEXMemRead <= 0;
        IDEXMemWrite <= 0;
        IDEXRegWrite <= 0;
        EXMEMMemRead <= 0;
        EXMEMMemWrite <= 0;
        EXMEMRegWrite <= 0;
        MEMWBRegWrite <= 0;
    end

    always @(negedge CLOCK)
        $display(" PC = %h IFIDIR = %h IDEXIR = %h EXMEMIR = %h MEMWBIR = %h A = %h B = %h ForwardA
= %h ForwardB = %h ALUOut1 = %h ALUOut2 = %h DMEM_ReadData = %h\n", pc_out, IFIDIR, IDEXIR,
EXMEMIR, MEMWBIR, outForwardAMUX, outForwardBMUX, ForwardA, ForwardB, ALUOut1, ALUOut2,
DMEM_ReadData);


    ALU alu(IDEXIR[15:12], IDEXIR[3:0], IDEXIR[7:4], stall, outForwardAMUX, outForwardBMUX, ALUOut1,
ALUOut2, Zero, Sign, overflow, divByZero);

    Control Ctr(IFIDIR[15:12], IFIDIR[3:0], IFIDImmd, IFIDMemWrite, IFIDMemRead, IFIDRegWrite);

    Forwarding_Unit FU(IDEXIR, EXMEMIR, MEMWBIR, ForwardA, ForwardB);

    Hazard_Detection_Unit HDU(EXMEMMemRead, IDEXImmd, IDEXIR[11:8], IDEXIR[7:4],
        EXMEMIR[11:8], stall);

    IMemory IMEM(CLOCK, RESET, pc_out, IMEM_ReadData, 16'b0, 1'b0);

    DMemory DMEM(CLOCK, RESET, EXMEMALUOut1, DMEM_ReadData, EXMEMoutForwardAMUX,
EXMEMMemWrite);

    PC pc(pc_in, pc_out, CLOCK, RESET);

    RegFile RF(CLOCK, RESET, MEMWBRegWrite, IFIDIR[11:8], IFIDIR[7:4],
        MEMWBIR[11:8], 4'b0, MEMWBALUOut1, MEMWBALUOut2, ReadData1,
        ReadData2, RegZeroData);

    assign halt = IDEXIR[15:12] == 'hF;

    assign pc_in = RESET ? 0 : ~PCSrc ? pc_out + 2 : IFIDIR[15:12] == 12 ?
        pc_out + ({{4{IFIDIR[12]}}, IFIDIR[11:0]} << 1) :
        pc_out + ({{9{IFIDIR[7]}}, IFIDIR[6:0]} << 1);

    assign MEMWBALUOut2 = (MEMWBIR[15:12] == 0 & MEMWBIR[3:0] == 4)
        ? MEMWB_Upper_Half_Product :
        (MEMWBIR[15:12] == 0 & MEMWBIR[3:0] == 5)
        ? MEMWB_Remainder : 0;

    assign outForwardAMUX = ForwardA == 0 ? EXMEM_Upper_Half_Product :
        ForwardA == 1 ? EXMEM_Remainder : ForwardA == 2 ? EXMEMALUOut1 :
        ForwardA == 3 ? MEMWBALUOut1 : ForwardA == 4 ?
        MEMWB_Upper_Half_Product : ForwardA == 5 ? MEMWB_Remainder :
```

```
    IDEXReadData1;

assign outForwardBMUX = ForwardB == 0 ? EXMEM_Upper_Half_Product :
    ForwardB == 1 ? EXMEM_Remainder : ForwardB == 2 ? EXMEMALUOut1 :
    ForwardB == 3 ? MEMWBALUOut1 : ForwardB == 4 ?
    MEMWB_Upper_Half_Product : ForwardB == 5 ? MEMWB_Remainder :
    IDEXReadData2;

always@(negedge CLOCK)
begin
if(~RESET) begin
    if(~halt)
    begin
        if(~overflow & ~divByZero & ~invalidOpCode & ~invalidFuncCode)
        begin
            if(~stall)
            begin
                if(~PCSrc)
                    IFIDIR <= IMEM_ReadData;
                else
                    IFIDIR <= 'h2000;  // insert a nop into ID stage

                IFIDPC <= pc_out;
                IDEXPC <= IFIDPC;

                IDEXReadData1 <= ReadData1;
                IDEXReadData2 <= ReadData2;
                IDEXIR <= IFIDIR;
                IDEXImmd <= IFIDImmd;
                IDEXMemWrite <= IFIDMemWrite;
                IDEXMemRead <= IFIDMemRead;
                IDEXRegWrite <= IFIDRegWrite;

                if((IDEXIR[15:12] == 8) | (IDEXIR[15:12] == 11))  // LW or SW
                    EXMEMALUOut1 <= IDEXReadData2 + {{13{IDEXIR[3]}}, IDEXIR[2:0]};
                else if(IDEXIR[15:12] == 0)
                begin
                    EXMEMALUOut1 <= ALUOut1;

                    if(IDEXIR[3:0] == 4)
                        EXMEM_Upper_Half_Product <= ALUOut2;
                    else if(IDEXIR[3:0] == 5)
                        EXMEM_Remainder <= ALUOut2;
                end

                EXMEMRegWrite <= IDEXRegWrite;
                EXMEMMemWrite <= IDEXMemWrite;
                EXMEMMemRead <= IDEXMemRead;
                EXMEMoutForwardAMUX <= outForwardAMUX;
                EXMEMIR <= IDEXIR;
            end
            else
            begin
                EXMEMIR <= 'h2000; // if stall, then insert nop into EX stage
                EXMEMRegWrite <= 0;
                EXMEMMemWrite <= 0;
                EXMEMMemRead <= 0;
            end
        end
        else
        begin                   // exception handler routine
            Exception <= 1;
            EPC = IDEXPC;
            IFIDIR <= 'hF000;   // insert a halt instruction to halt the system
            IDEXIR <= 'h2000;   // insert nops
            IDEXImmd <= 0;
            IDEXRegWrite <= 0;
            IDEXMemWrite <= 0;
            IDEXMemRead <= 0;
            EXMEMIR <= 'h2000;
            EXMEMRegWrite <= 0;
            EXMEMMemWrite <= 0;
            EXMEMMemRead <= 0;

            if(overflow)
                $display("Arithmetic overflow exception: EPC = %h", EPC);
            else if(divByZero)
                $display("Division by zero exception: EPC = %h", EPC);
            else if(invalidOpCode)
                $display("Invalid opcode: EPC = %h", EPC);
            else if(invalidFuncCode)
                $display("Invalid function code: EPC = %h", EPC);
        end

        if(EXMEMIR[15:12] == 0)
        begin
            MEMWBALUOut1 <= EXMEMALUOut1;
            MEMWB_Upper_Half_Product <= EXMEM_Upper_Half_Product;
            MEMWB_Remainder <= EXMEM_Remainder;
        end
        else if(EXMEMMemRead) // LW
            MEMWBALUOut1 <= DMEM_ReadData;

        MEMWBIR <= EXMEMIR;
        MEMWBRegWrite <= EXMEMRegWrite;
    end
    else
    begin                       // halt the system
        $display("system halted");

        if(Exception == 0 & counter < 2)
        begin
            counter <= counter + 1;

            if(counter == 0)
                EXMEMIR <= 'h2000;

            if(EXMEMIR[15:12] == 0)
            begin
                MEMWBALUOut1 <= EXMEMALUOut1;
                MEMWB_Upper_Half_Product <= EXMEM_Upper_Half_Product;
                MEMWB_Remainder <= EXMEM_Remainder;
            end
            else if(EXMEMIR[15:12] == 8) // LW
```

```
                    MEMWBALUOut1 <= DMEM_ReadData;

                MEMWBIR <= EXMEMIR;
                MEMWBRegWrite <= EXMEMRegWrite;
            end
        end
    end
end
endmodule
```

CPU Stimulus

```
`include "CPU3.v"

module stimulus;

reg CLOCK, RESET = 1;

initial
    $vcdpluson;

CPU cpu(CLOCK, RESET);

initial
begin
    #30 RESET = 0;
end

initial
begin
    CLOCK = 0;
    forever #10 CLOCK = ~CLOCK;
end

initial
begin
    #680 $finish;
end
endmodule
```

CPU Results

```
Chronologic VCS simulator copyright 1991-2009
Contains Synopsys proprietary information.
Compiler version D-2009.12; Runtime version D-2009.12;  May  6 04:20 2013
VCD+ Writer D-2009.12 Copyright 2009 Synopsys Inc.

 PC = 0000 IFIDIR = xxxx IDEXIR = xxxx EXMEMIR = xxxx MEMWBIR = xxxx A = xxxx B = xxxx ForwardA = x
ForwardB = x ALUOut1 = xxxx ALUOut2 = xxxx DMEM_ReadData = xxxx

MemoryArray[         0] = xx
MemoryArray[         1] = xx
MemoryArray[         2] = xx
MemoryArray[         3] = xx
MemoryArray[         4] = xx
MemoryArray[         5] = xx
MemoryArray[         6] = xx

RegFileArray[         0] = xxxx
RegFileArray[         1] = xxxx
RegFileArray[         2] = xxxx
RegFileArray[         3] = xxxx
RegFileArray[         4] = xxxx
RegFileArray[         5] = xxxx
RegFileArray[         6] = xxxx
RegFileArray[         7] = xxxx
RegFileArray[         8] = xxxx
RegFileArray[         9] = xxxx
RegFileArray[        10] = xxxx
RegFileArray[        11] = xxxx
RegFileArray[        12] = xxxx
RegFileArray[        13] = xxxx
RegFileArray[        14] = xxxx
RegFileArray[        15] = xxxx

 PC = 0000 IFIDIR = 2000 IDEXIR = 2000 EXMEMIR = 2000 MEMWBIR = 2000 A = 0000 B = 0000 ForwardA =
6 ForwardB = 6 ALUOut1 = xxxx ALUOut2 = xxxx DMEM_ReadData = abcd

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0f00
RegFileArray[         2] = 0050
RegFileArray[         3] = ff0f
RegFileArray[         4] = f0ff
RegFileArray[         5] = 0040
RegFileArray[         6] = 0024
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
```

```
RegFileArray[        11] = 0000
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000

 PC = 0000 IFIDIR = 2000 IDEXIR = 2000 EXMEMIR = 2000 MEMWBIR = 2000 A = 0000 B = 0000 ForwardA =
6 ForwardB = 6 ALUOut1 = xxxx ALUOut2 = xxxx DMEM_ReadData = abcd

MemoryArray[          0] = cd
MemoryArray[          1] = ab
MemoryArray[          2] = 00
MemoryArray[          3] = 00
MemoryArray[          4] = 00
MemoryArray[          5] = 00
MemoryArray[          6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0f00
RegFileArray[         2] = 0050
RegFileArray[         3] = ff0f
RegFileArray[         4] = f0ff
RegFileArray[         5] = 0040
RegFileArray[         6] = 0024
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0000
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000

 PC = 0002 IFIDIR = 0120 IDEXIR = 2000 EXMEMIR = 2000 MEMWBIR = 2000 A = 0000 B = 0000 ForwardA =
6 ForwardB = 6 ALUOut1 = xxxx ALUOut2 = xxxx DMEM_ReadData = abcd

MemoryArray[          0] = cd
MemoryArray[          1] = ab
MemoryArray[          2] = 00
MemoryArray[          3] = 00
MemoryArray[          4] = 00
MemoryArray[          5] = 00
MemoryArray[          6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0f00
RegFileArray[         2] = 0050
RegFileArray[         3] = ff0f
RegFileArray[         4] = f0ff
RegFileArray[         5] = 0040
RegFileArray[         6] = 0024
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0000
```

```
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000

 PC = 0004 IFIDIR = 0121 IDEXIR = 0120 EXMEMIR = 2000 MEMWBIR = 2000 A = 0f00 B = 0050 ForwardA =
6 ForwardB = 6 ALUOut1 = 0f50 ALUOut2 = xxxx DMEM_ReadData = abcd

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0f00
RegFileArray[         2] = 0050
RegFileArray[         3] = ff0f
RegFileArray[         4] = f0ff
RegFileArray[         5] = 0040
RegFileArray[         6] = 0024
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0000
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000

 PC = 0006 IFIDIR = 0343 IDEXIR = 0121 EXMEMIR = 0120 MEMWBIR = 2000 A = 0f50 B = 0050 ForwardA =
2 ForwardB = 6 ALUOut1 = 0f00 ALUOut2 = xxxx DMEM_ReadData = 0000

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0f00
RegFileArray[         2] = 0050
RegFileArray[         3] = ff0f
RegFileArray[         4] = f0ff
RegFileArray[         5] = 0040
RegFileArray[         6] = 0024
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0000
RegFileArray[        12] = ffff
```

```
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000

 PC = 0008 IFIDRegWrite IFIDIR = 0322 IDEXIR = 0343 EXMEMIR = 0121 MEMWBIR = 0120 A = ff0f B = f0ff
ForwardA = 6 ForwardB = 6 ALUOut1 = ffff ALUOut2 = xxxx DMEM_ReadData = 0000

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0f00
RegFileArray[         2] = 0050
RegFileArray[         3] = ff0f
RegFileArray[         4] = f0ff
RegFileArray[         5] = 0040
RegFileArray[         6] = 0024
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0000
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000

 PC = 000a IFIDIR = 0564 IDEXIR = 0322 EXMEMIR = 0343 MEMWBIR = 0121 A = ffff B = 0050 ForwardA = 2
ForwardB = 6 ALUOut1 = 0050 ALUOut2 = xxxx DMEM_ReadData = xx00

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0f50
RegFileArray[         2] = 0050
RegFileArray[         3] = ff0f
RegFileArray[         4] = f0ff
RegFileArray[         5] = 0040
RegFileArray[         6] = 0024
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0000
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
```

```
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000

 PC = 000c IFIDIR = 0155 IDEXIR = 0564 EXMEMIR = 0322 MEMWBIR = 0343 A = 0040 B = 0024 ForwardA =
6 ForwardB = 6 ALUOut1 = 0900 ALUOut2 = 0000 DMEM_ReadData = 0000

MemoryArray[          0] = cd
MemoryArray[          1] = ab
MemoryArray[          2] = 00
MemoryArray[          3] = 00
MemoryArray[          4] = 00
MemoryArray[          5] = 00
MemoryArray[          6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0f00
RegFileArray[         2] = 0050
RegFileArray[         3] = ff0f
RegFileArray[         4] = f0ff
RegFileArray[         5] = 0040
RegFileArray[         6] = 0024
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0000
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000

 PC = 000e IFIDRegWrite IFIDIR = 0001 IDEXIR = 0155 EXMEMIR = 0564 MEMWBIR = 0322 A = 0f00 B =
0900 ForwardA = 6 ForwardB = 2 ALUOut1 = 0001 ALUOut2 = 0600 DMEM_ReadData = 0000

MemoryArray[          0] = cd
MemoryArray[          1] = ab
MemoryArray[          2] = 00
MemoryArray[          3] = 00
MemoryArray[          4] = 00
MemoryArray[          5] = 00
MemoryArray[          6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0f00
RegFileArray[         2] = 0050
RegFileArray[         3] = ffff
RegFileArray[         4] = f0ff
RegFileArray[         5] = 0040
RegFileArray[         6] = 0024
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0000
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
```

```
RegFileArray[        15] = 0000

 PC = 0010 IFIDRegWrite IFIDIR = 0438 IDEXIR = 0001 EXMEMIR = 0155 MEMWBIR = 0564 A = 0600 B =
0600 ForwardA = 1 ForwardB = 1 ALUOut1 = 0000 ALUOut2 = 0600 DMEM_ReadData = 00ab

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0f00
RegFileArray[         2] = 0050
RegFileArray[         3] = 0050
RegFileArray[         4] = f0ff
RegFileArray[         5] = 0040
RegFileArray[         6] = 0024
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0000
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000

 PC = 0012 IFIDRegWrite IFIDIR = 0429 IDEXIR = 0438 EXMEMIR = 0001 MEMWBIR = 0155 A = f0ff B = 0050
ForwardA = 6 ForwardB = 6 ALUOut1 = 87f8 ALUOut2 = 0600 DMEM_ReadData = abcd

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0f00
RegFileArray[         2] = 0050
RegFileArray[         3] = 0050
RegFileArray[         4] = f0ff
RegFileArray[         5] = 0900
RegFileArray[         6] = 0024
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0000
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000
```

```
 PC = 0014 IFIDIR = 063b IDEXIR = 0429 EXMEMIR = 0438 MEMWBIR = 0001 A = 87f8 B = 0050 ForwardA =
2 ForwardB = 6 ALUOut1 = 21fe ALUOut2 = 0600 DMEM_ReadData = 0000

MemoryArray[          0] = cd
MemoryArray[          1] = ab
MemoryArray[          2] = 00
MemoryArray[          3] = 00
MemoryArray[          4] = 00
MemoryArray[          5] = 00
MemoryArray[          6] = 00

RegFileArray[          0] = 0600
RegFileArray[          1] = 0001
RegFileArray[          2] = 0050
RegFileArray[          3] = 0050
RegFileArray[          4] = f0ff
RegFileArray[          5] = 0900
RegFileArray[          6] = 0024
RegFileArray[          7] = 00ff
RegFileArray[          8] = aaaa
RegFileArray[          9] = 0000
RegFileArray[         10] = 0000
RegFileArray[         11] = 0000
RegFileArray[         12] = ffff
RegFileArray[         13] = 0002
RegFileArray[         14] = 0000
RegFileArray[         15] = 0000

 PC = 0016 IFIDIR = 062a IDEXIR = 063b EXMEMIR = 0429 MEMWBIR = 0438 A = 0024 B = 0050 ForwardA =
6 ForwardB = 6 ALUOut1 = 8004 ALUOut2 = 0600 DMEM_ReadData = 0000

MemoryArray[          0] = cd
MemoryArray[          1] = ab
MemoryArray[          2] = 00
MemoryArray[          3] = 00
MemoryArray[          4] = 00
MemoryArray[          5] = 00
MemoryArray[          6] = 00

RegFileArray[          0] = 0000
RegFileArray[          1] = 0001
RegFileArray[          2] = 0050
RegFileArray[          3] = 0050
RegFileArray[          4] = f0ff
RegFileArray[          5] = 0900
RegFileArray[          6] = 0024
RegFileArray[          7] = 00ff
RegFileArray[          8] = aaaa
RegFileArray[          9] = 0000
RegFileArray[         10] = 0000
RegFileArray[         11] = 0000
RegFileArray[         12] = ffff
RegFileArray[         13] = 0002
RegFileArray[         14] = 0000
RegFileArray[         15] = 0000
```

```
 PC = 0018 IFIDIR = 6704 IDEXIR = 062a EXMEMIR = 063b MEMWBIR = 0429 A = 8004 B = 0050 ForwardA =
2 ForwardB = 6 ALUOut1 = 0012 ALUOut2 = 0600 DMEM_ReadData = 0000

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0001
RegFileArray[         2] = 0050
RegFileArray[         3] = 0050
RegFileArray[         4] = 87f8
RegFileArray[         5] = 0900
RegFileArray[         6] = 0024
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0000
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000

 PC = 001a IFIDIR = 0b10 IDEXIR = 6704 EXMEMIR = 062a MEMWBIR = 063b A = 00ff B = 0000 ForwardA = 6
ForwardB = 6 ALUOut1 = 0012 ALUOut2 = 0600 DMEM_ReadData = 0000

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0001
RegFileArray[         2] = 0050
RegFileArray[         3] = 0050
RegFileArray[         4] = 21fe
RegFileArray[         5] = 0900
RegFileArray[         6] = 0024
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0000
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000
```

```
 PC = 001c IFIDIR = 4705 IDEXIR = 0b10 EXMEMIR = 6704 MEMWBIR = 062a A = 0000 B = 0001 ForwardA =
6 ForwardB = 6 ALUOut1 = 0001 ALUOut2 = 0600 DMEM_ReadData = 0000

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0001
RegFileArray[         2] = 0050
RegFileArray[         3] = 0050
RegFileArray[         4] = 21fe
RegFileArray[         5] = 0900
RegFileArray[         6] = 8004
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0000
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000

 PC = 001e IFIDRegWrite IFIDIR = 0b20 IDEXIR = 4705 EXMEMIR = 0b10 MEMWBIR = 6704 A = 00ff B = 0000
ForwardA = 6 ForwardB = 6 ALUOut1 = 0001 ALUOut2 = 0600 DMEM_ReadData = 00ab

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0001
RegFileArray[         2] = 0050
RegFileArray[         3] = 0050
RegFileArray[         4] = 21fe
RegFileArray[         5] = 0900
RegFileArray[         6] = 0012
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0000
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000
```

```
 PC = 0020 IFIDIR = 5702 IDEXIR = 0b20 EXMEMIR = 4705 MEMWBIR = 0b10 A = 0001 B = 0050 ForwardA =
3 ForwardB = 6 ALUOut1 = 0051 ALUOut2 = 0600 DMEM_ReadData = 00ab

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0001
RegFileArray[         2] = 0050
RegFileArray[         3] = 0050
RegFileArray[         4] = 21fe
RegFileArray[         5] = 0900
RegFileArray[         6] = 0012
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0000
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000

 PC = 0024 IFIDIR = 2000 IDEXIR = 5702 EXMEMIR = 0b20 MEMWBIR = 4705 A = 00ff B = 0000 ForwardA = 6
ForwardB = 6 ALUOut1 = 0051 ALUOut2 = 0600 DMEM_ReadData = 0000

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0001
RegFileArray[         2] = 0050
RegFileArray[         3] = 0050
RegFileArray[         4] = 21fe
RegFileArray[         5] = 0900
RegFileArray[         6] = 0012
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0001
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000
```

```
 PC = 0026 IFIDIR = 8890 IDEXIR = 2000 EXMEMIR = 5702 MEMWBIR = 0b20 A = 0000 B = 0000 ForwardA =
6 ForwardB = 6 ALUOut1 = 0051 ALUOut2 = 0600 DMEM_ReadData = 0000

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0001
RegFileArray[         2] = 0050
RegFileArray[         3] = 0050
RegFileArray[         4] = 21fe
RegFileArray[         5] = 0900
RegFileArray[         6] = 0012
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0001
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000

 PC = 0028 IFIDIR = 0880 IDEXIR = 8890 EXMEMIR = 2000 MEMWBIR = 5702 A = aaaa B = 0000 ForwardA =
6 ForwardB = 6 ALUOut1 = 0051 ALUOut2 = 0600 DMEM_ReadData = 0000

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0001
RegFileArray[         2] = 0050
RegFileArray[         3] = 0050
RegFileArray[         4] = 21fe
RegFileArray[         5] = 0900
RegFileArray[         6] = 0012
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0051
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000
```

```
 PC = 002a IFIDIR = b892 IDEXIR = 0880 EXMEMIR = 8890 MEMWBIR = 2000 A = aaaa B = aaaa ForwardA =
6 ForwardB = 6 ALUOut1 = 5554 ALUOut2 = 0600 DMEM_ReadData = abcd

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0001
RegFileArray[         2] = 0050
RegFileArray[         3] = 0050
RegFileArray[         4] = 21fe
RegFileArray[         5] = 0900
RegFileArray[         6] = 0012
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0051
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000
 PC = 002c IFIDIR = b892 IDEXIR = 0880 EXMEMIR = 2000 MEMWBIR = 8890 A = abcd B = abcd ForwardA =
3 ForwardB = 3 ALUOut1 = 579a ALUOut2 = 0600 DMEM_ReadData = abcd

Arithmetic overflow exception: EPC = 0026

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0001
RegFileArray[         2] = 0050
RegFileArray[         3] = 0050
RegFileArray[         4] = 21fe
RegFileArray[         5] = 0900
RegFileArray[         6] = 0012
RegFileArray[         7] = 00ff
RegFileArray[         8] = aaaa
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0051
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000
```

```
 PC = 002e IFIDIR = f000 IDEXIR = 2000 EXMEMIR = 2000 MEMWBIR = 2000 A = aaaa B = aaaa ForwardA =
6 ForwardB = 6 ALUOut1 = 579a ALUOut2 = 0600 DMEM_ReadData = abcd

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0001
RegFileArray[         2] = 0050
RegFileArray[         3] = 0050
RegFileArray[         4] = 21fe
RegFileArray[         5] = 0900
RegFileArray[         6] = 0012
RegFileArray[         7] = 00ff
RegFileArray[         8] = abcd
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0051
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000

 PC = 0030 IFIDIR = 0dd1 IDEXIR = f000 EXMEMIR = 2000 MEMWBIR = 2000 A = 0000 B = 0000 ForwardA =
6 ForwardB = 6 ALUOut1 = 579a ALUOut2 = 0600 DMEM_ReadData = abcd

system halted

MemoryArray[         0] = cd
MemoryArray[         1] = ab
MemoryArray[         2] = 00
MemoryArray[         3] = 00
MemoryArray[         4] = 00
MemoryArray[         5] = 00
MemoryArray[         6] = 00

RegFileArray[         0] = 0000
RegFileArray[         1] = 0001
RegFileArray[         2] = 0050
RegFileArray[         3] = 0050
RegFileArray[         4] = 21fe
RegFileArray[         5] = 0900
RegFileArray[         6] = 0012
RegFileArray[         7] = 00ff
RegFileArray[         8] = abcd
RegFileArray[         9] = 0000
RegFileArray[        10] = 0000
RegFileArray[        11] = 0051
RegFileArray[        12] = ffff
RegFileArray[        13] = 0002
RegFileArray[        14] = 0000
RegFileArray[        15] = 0000
```

Data Memory Module

```
module DMemory(Clock, Reset, Address, ReadData, WriteData, MemWrite);
input Clock, Reset;
input [15:0] Address, WriteData;
input MemWrite;
output wire [15:0] ReadData;

reg [7:0] MemoryArray[0:65535];
integer i;

assign ReadData = {MemoryArray[Address + 1], MemoryArray[Address]};

integer j;

always @(negedge Clock) begin

    /*for(j = 0; j < 7; j = j + 1)
        $display("MemoryArray[%d] = %h", j, MemoryArray[j]); */

    if(Reset)
    begin
        for(i = 0; i < 65536; i = i + 1)
            MemoryArray[i] = 0;

        MemoryArray[0] = 'hCD;
        MemoryArray[1] = 'hAB;
    end

    if(MemWrite)
    begin
        MemoryArray[Address] <= WriteData[7:0];
        MemoryArray[Address + 1] <= WriteData[15:8];
    end
end
endmodule
```

Data Memory Stimulus

```
`include "DMemory.v"

module stimulus;

reg [15:0] Address = 0, WriteData = 'hC435;
reg MemWrite = 0;
wire [15:0] ReadData;

reg CLOCK, RESET = 1;

initial
   $vcdpluson;

initial
   $monitor($time, " Address = %h ReadData = %h WriteData = %h MemWrite = %h",
      Address, ReadData, WriteData, MemWrite);

DMemory(CLOCK, RESET, Address, ReadData, WriteData, MemWrite);

initial
   begin
      #30 RESET = 0;
      #20 Address = 20;
      #20 Address = 0;
      #20 Address = 10;
         MemWrite = 1;
      #20 MemWrite = 0;
         Address = 0;
      #20 Address = 10;
      #20 Address = 55;
         WriteData = 'hC7AE;
         MemWrite = 1;
      #20 MemWrite = 0;
         Address = 0;
      #20 Address = 55;
   end

initial
begin
   CLOCK = 0;
   forever #10 CLOCK = ~CLOCK;
end

initial
begin
   #680 $finish;
end

endmodule
```

Data Memory Results

```
Chronologic VCS simulator copyright 1991-2009
Contains Synopsys proprietary information.
Compiler version D-2009.12; Runtime version D-2009.12;  May  6 04:21 2013
VCD+ Writer D-2009.12 Copyright 2009 Synopsys Inc.

                   0 Address = 0000 ReadData = abcd WriteData = c435 MemWrite = 0
                  50 Address = 0014 ReadData = 0000 WriteData = c435 MemWrite = 0
                  70 Address = 0000 ReadData = abcd WriteData = c435 MemWrite = 0
                  90 Address = 000a ReadData = 0000 WriteData = c435 MemWrite = 1
                 100 Address = 000a ReadData = c435 WriteData = c435 MemWrite = 1
                 110 Address = 0000 ReadData = abcd WriteData = c435 MemWrite = 0
                 130 Address = 000a ReadData = c435 WriteData = c435 MemWrite = 0
                 150 Address = 0037 ReadData = 0000 WriteData = c7ae MemWrite = 1
                 160 Address = 0037 ReadData = c7ae WriteData = c7ae MemWrite = 1
                 170 Address = 0000 ReadData = abcd WriteData = c7ae MemWrite = 0
                 190 Address = 0037 ReadData = c7ae WriteData = c7ae MemWrite = 0

$finish called from file "DMemory_fixture.v", line 46.
$finish at simulation time                  680
           V C S   S i m u l a t i o n   R e p o r t
Time: 680
CPU Time:      0.400 seconds;       Data structure size:   0.1Mb
Mon May  6 04:21:53 2013
```

Forwarding Unit Module

```
module Forwarding_Unit(IDEXIR, EXMEMIR, MEMWBIR, ForwardA, ForwardB);
   input [15:0] IDEXIR, EXMEMIR, MEMWBIR;
   output wire [2:0] ForwardA, ForwardB;

   wire [3:0] IFIDOpCode, IDEXOpCode, EXMEMOpCode, MEMWBOpCode, IFIDRegOp1,
      IFIDRegOp2, IDEXRegOp1, IDEXRegOp2, EXMEMRegOp1, EXMEMRegOp2,
      MEMWBRegOp1, MEMWBRegOp2;

   wire EXMEMMult, EXMEMDiv, MEMWBMult, MEMWBDiv, IDEXImmd, A_MEM_Data_Hazard,
      B_MEM_Data_Hazard, A_WB_Data_Hazard, B_WB_Data_Hazard,
      A_WB_Load_Use_Hazard, B_WB_Load_Use_Hazard, A_MEM_Mult_Data_Hazard,
      A_MEM_Div_Data_Hazard, A_WB_Div_Data_Hazard, A_WB_Mult_Data_Hazard,
      B_MEM_Mult_Data_Hazard, B_MEM_Div_Data_Hazard, B_WB_Div_Data_Hazard,
      B_WB_Mult_Data_Hazard;


   assign IDEXOpCode = IDEXIR[15:12];
   assign EXMEMOpCode = EXMEMIR[15:12];
   assign MEMWBOpCode = MEMWBIR[15:12];
   assign IDEXRegOp1 = IDEXIR[11:8];
   assign IDEXRegOp2 = IDEXIR[7:4];
   assign EXMEMRegOp1 = EXMEMIR[11:8];
   assign EXMEMRegOp2 = EXMEMIR[7:4];
   assign MEMWBRegOp1 = MEMWBIR[11:8];
   assign MEMWBRegOp2 = MEMWBIR[7:4];
   assign EXMEMMult = (EXMEMOpCode == 0) & (EXMEMIR[3:0] == 4);
   assign EXMEMDiv = (EXMEMOpCode == 0) & (EXMEMIR[3:0] == 5);
   assign MEMWBMult = (MEMWBOpCode == 0) & (MEMWBIR[3:0] == 4);
   assign MEMWBDiv = (MEMWBOpCode == 0) & (MEMWBIR[3:0] == 5);
   assign IDEXImmd = (IDEXOpCode >= 4 & IDEXOpCode <= 6) |
      (IDEXOpCode == 0 & (IDEXIR[3:0] >= 8 & IDEXIR[3:0] <= 11));
   assign A_MEM_Data_Hazard = (IDEXRegOp1 == EXMEMRegOp1) & (EXMEMOpCode == 0);
   assign B_MEM_Data_Hazard = (IDEXRegOp2 == EXMEMRegOp1) & (EXMEMOpCode == 0) & ~IDEXImmd;

   assign A_WB_Data_Hazard = (IDEXRegOp1 == MEMWBRegOp1) & (MEMWBOpCode == 0);
   assign B_WB_Data_Hazard = (IDEXRegOp2 == MEMWBRegOp1) & (MEMWBOpCode == 0) & ~IDEXImmd;
   assign A_WB_Load_Use_Hazard = (IDEXRegOp1 == MEMWBRegOp1) & (MEMWBOpCode == 8);
   assign B_WB_Load_Use_Hazard = (IDEXRegOp2 == MEMWBRegOp1) & (MEMWBOpCode == 8) &
         ~IDEXImmd;

   assign A_MEM_Mult_Data_Hazard = (IDEXRegOp1 == 0) & EXMEMMult;
   assign A_MEM_Div_Data_Hazard = (IDEXRegOp1 == 0) & EXMEMDiv;
   assign A_WB_Mult_Data_Hazard = (IDEXRegOp1 == 0) & MEMWBMult;
   assign A_WB_Div_Data_Hazard = (IDEXRegOp1 == 0) & MEMWBDiv;

   assign B_MEM_Mult_Data_Hazard = (IDEXRegOp2 == 0) & EXMEMMult & ~IDEXImmd;
   assign B_MEM_Div_Data_Hazard = (IDEXRegOp2 == 0) & EXMEMDiv & ~IDEXImmd;
   assign B_WB_Mult_Data_Hazard = (IDEXRegOp2 == 0) & MEMWBMult & ~IDEXImmd;
   assign B_WB_Div_Data_Hazard = (IDEXRegOp2 == 0) & MEMWBDiv & ~IDEXImmd;

   assign ForwardA = A_MEM_Mult_Data_Hazard ? 0 : A_MEM_Div_Data_Hazard ? 1
      : A_MEM_Data_Hazard ? 2 : (A_WB_Data_Hazard | A_WB_Load_Use_Hazard) ? 3
      : A_WB_Mult_Data_Hazard ? 4 : A_WB_Div_Data_Hazard ? 5 : 6;

   assign ForwardB = B_MEM_Mult_Data_Hazard ? 0 : B_MEM_Div_Data_Hazard ? 1
```

```
        : B_MEM_Data_Hazard ? 2 : (B_WB_Data_Hazard | B_WB_Load_Use_Hazard) ? 3
        : B_WB_Mult_Data_Hazard ? 4 : B_WB_Div_Data_Hazard ? 5 : 6;
endmodule
```

Forwarding Unit Stimulus

```
`include "forwarding_unit.v"

module stimulus;

reg [15:0] IDEXIR = 'h2000, EXMEMIR = 'h2000, MEMWBIR = 'h2000;
wire [2:0] ForwardA, ForwardB;

initial
   $vcdpluson;

initial
   $monitor("IDEXIR = %h EXMEMIR = %h MEMWBIR = %h ForwardA = %h ForwardB = %h", IDEXIR,
EXMEMIR, MEMWBIR, ForwardA, ForwardB);

Forwarding_Unit FU(IDEXIR, EXMEMIR, MEMWBIR, ForwardA, ForwardB);

initial
   begin
      #20 IDEXIR <= 'h0340;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h0351;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h89a3;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h0be2;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h09b0;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h0214;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h0aa4;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
     #20 IDEXIR <= 'h0ef0
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h0b05;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h0a00;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h8300;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h0ae4;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h0e34;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h0aa0;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'hba90;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h2000;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h0000;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h0101;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
      #20 IDEXIR <= 'h0110;
         EXMEMIR <= IDEXIR;
         MEMWBIR <= EXMEMIR;
   end

endmodule
```

Forwarding Unit Results

```
Chronologic VCS simulator copyright 1991-2009
Contains Synopsys proprietary information.
Compiler version D-2009.12; Runtime version D-2009.12;  May  7 06:41 2013
VCD+ Writer D-2009.12 Copyright 2009 Synopsys Inc.

IDEXIR = 2000 EXMEMIR = 2000 MEMWBIR = 2000 ForwardA = 6 ForwardB = 6
IDEXIR = 0340 EXMEMIR = 2000 MEMWBIR = 2000 ForwardA = 6 ForwardB = 6
IDEXIR = 0351 EXMEMIR = 0340 MEMWBIR = 2000 ForwardA = 2 ForwardB = 6
IDEXIR = 89a3 EXMEMIR = 0351 MEMWBIR = 0340 ForwardA = 6 ForwardB = 6
IDEXIR = 0be2 EXMEMIR = 89a3 MEMWBIR = 0351 ForwardA = 6 ForwardB = 6
IDEXIR = 09b0 EXMEMIR = 0be2 MEMWBIR = 89a3 ForwardA = 3 ForwardB = 2
IDEXIR = 0214 EXMEMIR = 09b0 MEMWBIR = 0be2 ForwardA = 6 ForwardB = 6
IDEXIR = 0aa4 EXMEMIR = 0214 MEMWBIR = 09b0 ForwardA = 6 ForwardB = 6
IDEXIR = 0ef0 EXMEMIR = 0aa4 MEMWBIR = 0214 ForwardA = 6 ForwardB = 6
IDEXIR = 0b05 EXMEMIR = 0ef0 MEMWBIR = 0aa4 ForwardA = 6 ForwardB = 4
IDEXIR = 0a00 EXMEMIR = 0b05 MEMWBIR = 0ef0 ForwardA = 6 ForwardB = 1
IDEXIR = 8300 EXMEMIR = 0a00 MEMWBIR = 0b05 ForwardA = 6 ForwardB = 5
IDEXIR = 0ae4 EXMEMIR = 8300 MEMWBIR = 0a00 ForwardA = 3 ForwardB = 6
IDEXIR = 0e34 EXMEMIR = 0ae4 MEMWBIR = 8300 ForwardA = 6 ForwardB = 3
IDEXIR = 0aa0 EXMEMIR = 0e34 MEMWBIR = 0ae4 ForwardA = 3 ForwardB = 3
IDEXIR = ba90 EXMEMIR = 0aa0 MEMWBIR = 0e34 ForwardA = 2 ForwardB = 6
IDEXIR = 2000 EXMEMIR = ba90 MEMWBIR = 0aa0 ForwardA = 6 ForwardB = 6
IDEXIR = 0000 EXMEMIR = 2000 MEMWBIR = ba90 ForwardA = 6 ForwardB = 6
IDEXIR = 0101 EXMEMIR = 0000 MEMWBIR = 2000 ForwardA = 6 ForwardB = 2
IDEXIR = 0110 EXMEMIR = 0101 MEMWBIR = 0000 ForwardA = 2 ForwardB = 2

           V C S   S i m u l a t i o n   R e p o r t
Time: 380
CPU Time:      0.310 seconds;       Data structure size:   0.0Mb
Tue May  7 06:41:54 2013
```

Hazard Detection Unit Module

```
module Hazard_Detection_Unit(EXMEMMemRead, IDEXImmd, IDEXRegOp1, IDEXRegOp2, EXMEMRegOp1,
stall);
    input EXMEMMemRead, IDEXImmd;
    input [3:0] IDEXRegOp1, IDEXRegOp2, EXMEMRegOp1;
    output wire stall;

    assign stall = EXMEMMemRead & ((EXMEMRegOp1 == IDEXRegOp1) |
             (EXMEMRegOp1 == IDEXRegOp2)) & ~IDEXImmd;

endmodule
```

Hazard Detection Unit Stimulus

```
`include "hazard_det_unit.v"

module stimulus;

reg EXMEMMemRead = 0, IDEXImmd = 0;
reg [3:0] IDEXRegOp1 = 0, IDEXRegOp2 = 0, EXMEMRegOp1 = 0;
wire stall;

initial
   $vcdpluson;

initial
   $monitor("EXMEMMemRead = %h IDEXImmd = %h IDEXRegOp1 = %h IDEXRegOp2 = %h EXMEMRegOp1
= %h stall = %h", EXMEMMemRead, IDEXImmd, IDEXRegOp1, IDEXRegOp2, EXMEMRegOp1, stall);

Hazard_Detection_Unit HDU(EXMEMMemRead, IDEXImmd, IDEXRegOp1, IDEXRegOp2, EXMEMRegOp1,
stall);

initial
   begin
      #20 EXMEMMemRead = 1;
      #20 EXMEMMemRead = 0;
      #20 EXMEMMemRead = 1;
         IDEXImmd = 1;
      #20 IDEXImmd = 0;
         EXMEMMemRead = 1;
         EXMEMRegOp1 = 5;
      #20 EXMEMMemRead = 0;
         IDEXRegOp2 = 5;
   end

endmodule
```

Hazard Detection Unit Results

```
Chronologic VCS simulator copyright 1991-2009
Contains Synopsys proprietary information.
Compiler version D-2009.12; Runtime version D-2009.12;  May  6 04:23 2013
VCD+ Writer D-2009.12 Copyright 2009 Synopsys Inc.

EXMEMMemRead = 0 IDEXImmd = 0 IDEXRegOp1 = 0 IDEXRegOp2 = 0 EXMEMRegOp1 = 0 stall = 0
EXMEMMemRead = 1 IDEXImmd = 0 IDEXRegOp1 = 0 IDEXRegOp2 = 0 EXMEMRegOp1 = 0 stall = 1
EXMEMMemRead = 0 IDEXImmd = 0 IDEXRegOp1 = 0 IDEXRegOp2 = 0 EXMEMRegOp1 = 0 stall = 0
EXMEMMemRead = 1 IDEXImmd = 1 IDEXRegOp1 = 0 IDEXRegOp2 = 0 EXMEMRegOp1 = 0 stall = 0
EXMEMMemRead = 1 IDEXImmd = 0 IDEXRegOp1 = 0 IDEXRegOp2 = 0 EXMEMRegOp1 = 5 stall = 0
EXMEMMemRead = 0 IDEXImmd = 0 IDEXRegOp1 = 0 IDEXRegOp2 = 5 EXMEMRegOp1 = 5 stall = 0

           V C S   S i m u l a t i o n   R e p o r t
Time: 100
CPU Time:      0.400 seconds;       Data structure size:   0.0Mb
Mon May  6 04:23:48 2013
```

Instruction Memory Module

```
module IMemory(Clock, Reset, Address, ReadData, WriteData, MemWrite);
input Clock, Reset;
input [15:0] Address, WriteData;
input MemWrite;
output wire [15:0] ReadData;

reg [7:0] MemoryArray[0:65535];
integer i;

assign ReadData = {MemoryArray[Address + 1], MemoryArray[Address]};

always @(negedge Clock)
begin
    if(Reset)
    begin
        for(i = 0; i < 65536; i = i + 1)
            MemoryArray[i] = 0;

        MemoryArray[0] = 'h20;          // ADD R1, R2
        MemoryArray[1] = 'h01;
        MemoryArray[2] = 'h21;          // SUB R1, R2
        MemoryArray[3] = 'h01;
        MemoryArray[4] = 'h43;          // OR R3, R4
        MemoryArray[5] = 'h03;
        MemoryArray[6] = 'h22;          // AND R3, R2
        MemoryArray[7] = 'h03;
        MemoryArray[8] = 'h64;          // MUL R5, R6
        MemoryArray[9] = 'h05;
        MemoryArray[10] = 'h55;         // DIV R1, R5
        MemoryArray[11] = 'h01;
        MemoryArray[12] = 'h01;         // SUB R0, R0
        MemoryArray[13] = 'h0;
        MemoryArray[14] = 'h38;         // SLL R4, 3
        MemoryArray[15] = 'h04;
        MemoryArray[16] = 'h29;         // SLR R4, 2
        MemoryArray[17] = 'h04;
        MemoryArray[18] = 'h3B;          // ROR R6, 3
        MemoryArray[19] = 'h06;
        MemoryArray[20] = 'h2A;          // ROL R6, 2
        MemoryArray[21] = 'h06;
        MemoryArray[22] = 'h04;         // BEQ R7, 4
        MemoryArray[23] = 'h67;
        MemoryArray[24] = 'h10;         // ADD R11, R1
        MemoryArray[25] = 'h0B;
        MemoryArray[26] = 'h05;         // BLT R7, 2
        MemoryArray[27] = 'h47;
        MemoryArray[28] = 'h20;         // ADD R11, R1
        MemoryArray[29] = 'h0B;
        MemoryArray[30] = 'h02;         // BGT R7, 2
        MemoryArray[31] = 'h57;
        MemoryArray[32] = 'h10;         // ADD R1, R1
        MemoryArray[33] = 'h01;
        MemoryArray[34] = 'h10;         // ADD R1, R1
        MemoryArray[35] = 'h01;
        MemoryArray[36] = 'h90;         // LW R8, 0(R9)
        MemoryArray[37] = 'h88;
        MemoryArray[38] = 'h80;         // ADD R8, R8
        MemoryArray[39] = 'h08;
        MemoryArray[40] = 'h92;         // SW R8, 2(R9)
        MemoryArray[41] = 'hB8;
        MemoryArray[42] = 'h92;         // LW R10, 2(R9)
        MemoryArray[43] = 'h8A;
        MemoryArray[44] = 'hC0;         // ADD R12, R12
        MemoryArray[45] = 'h0C;
        MemoryArray[46] = 'hD1;         // SUB R13, R13
        MemoryArray[47] = 'h0D;
        MemoryArray[48] = 'hD0;         // ADD R12, R13
        MemoryArray[49] = 'h0C;
        MemoryArray[50] = 'hFF;         // INVALID INSTRUCTION
        MemoryArray[51] = 'hEF;
    end

    if(MemWrite)
    begin
        MemoryArray[Address] <= WriteData[7:0];
        MemoryArray[Address + 1] <= WriteData[15:8];
    end
end
endmodule
```

Instruction Memory Stimulus

```
`include "IMemory.v"

module stimulus;

reg [15:0] PC = 0;
reg MemWrite = 0;
wire [15:0] ReadData;

reg CLOCK, RESET = 1;

initial
   $vcdpluson;

initial
   $monitor($time, " PC = %h ReadData = %h", PC, ReadData);

IMemory(CLOCK, RESET, PC, ReadData, 16'b0, 1'b0);

initial
   begin
      #30 RESET = 0;
      #20 PC = 2;
      #20 PC = 4;
      #20 PC = 6;
      #20 PC = 10;
      #20 PC = 12;
      #20 PC = 14;
   end

initial
begin
   CLOCK = 0;
   forever #10 CLOCK = ~CLOCK;
end

initial
begin
   #680 $finish;
end

endmodule
```

Instruction Memory Results

```
Chronologic VCS simulator copyright 1991-2009
Contains Synopsys proprietary information.
Compiler version D-2009.12; Runtime version D-2009.12;  May  6 04:24 2013
VCD+ Writer D-2009.12 Copyright 2009 Synopsys Inc.

                    0 PC = 0000 ReadData = 0120
                   50 PC = 0002 ReadData = 0121
                   70 PC = 0004 ReadData = 0343
                   90 PC = 0006 ReadData = 0322
                  110 PC = 000a ReadData = 0155
                  130 PC = 000c ReadData = 0001
                  150 PC = 000e ReadData = 0438

$finish called from file "IMemory_fixture.v", line 38.
$finish at simulation time                  680
           V C S   S i m u l a t i o n   R e p o r t
Time: 680
CPU Time:      0.410 seconds;       Data structure size:   0.1Mb
Mon May  6 04:24:53 2013
```

Program Counter Module

```
module PC(in, out, clock, reset);
    input [15:0] in;
    input clock, reset;
    output wire[15:0] out;

    reg [15:0] curPC;

    initial
        curPC = 0;

    assign out = curPC;

    always@(negedge clock)
    begin
        if(reset)
            curPC <= 0;
        else
            curPC <= in;
    end
endmodule
```

Program Counter Stimulus

```
`include "pc.v"

module stimulus;

reg CLOCK, RESET = 1;
reg [15:0] in;
wire [15:0] out;

initial
    $vcdpluson;

initial
    $monitor($time, " in = %h out = %h reset = %h", in, out, RESET);

PC pc(in, out, CLOCK, RESET);

initial
    begin
        #30 RESET = 0;
        #20 in = 2;
        #20 in = 4;
        #20 in = 6;
        #20 in = 10;
        #20 in = 12;
        #20 in = 14;
        #20 RESET = 1;
        #100 RESET = 0;
        #20 in = 24;
        #20 in = 26;
    end

initial
begin
    CLOCK = 0;
    forever #10 CLOCK = ~CLOCK;
end

initial
begin
    #680 $finish;
end

endmodule
```

Program Counter Results

```
Chronologic VCS simulator copyright 1991-2009
Contains Synopsys proprietary information.
Compiler version D-2009.12; Runtime version D-2009.12;  May  6 04:25 2013
VCD+ Writer D-2009.12 Copyright 2009 Synopsys Inc.

                    0 in = xxxx out = 0000 reset = 1
                   30 in = xxxx out = 0000 reset = 0
                   40 in = xxxx out = xxxx reset = 0
                   50 in = 0002 out = xxxx reset = 0
                   60 in = 0002 out = 0002 reset = 0
                   70 in = 0004 out = 0002 reset = 0
                   80 in = 0004 out = 0004 reset = 0
                   90 in = 0006 out = 0004 reset = 0
                  100 in = 0006 out = 0006 reset = 0
                  110 in = 000a out = 0006 reset = 0
                  120 in = 000a out = 000a reset = 0
                  130 in = 000c out = 000a reset = 0
                  140 in = 000c out = 000c reset = 0
                  150 in = 000e out = 000c reset = 0
                  160 in = 000e out = 000e reset = 0
                  170 in = 000e out = 000e reset = 1
                  180 in = 000e out = 0000 reset = 1
                  270 in = 000e out = 0000 reset = 0
                  280 in = 000e out = 000e reset = 0
                  290 in = 0018 out = 000e reset = 0
                  300 in = 0018 out = 0018 reset = 0
                  310 in = 001a out = 0018 reset = 0
                  320 in = 001a out = 001a reset = 0

$finish called from file "pc_fixture.v", line 63.
$finish at simulation time                  680
           V C S   S i m u l a t i o n   R e p o r t
Time: 680
CPU Time:      0.400 seconds;       Data structure size:   0.0Mb
Mon May  6 04:25:29 2013
```

Register File Module

```
module RegFile(clock, reset, RegWrite, ReadReg1, ReadReg2, WriteReg1,
   WriteReg2, WriteData1, WriteData2, ReadData1, ReadData2, RegZeroData);
input clock, reset;
input [1:0] RegWrite;
input [3:0] ReadReg1, ReadReg2, WriteReg1, WriteReg2;
input [15:0] WriteData1, WriteData2;
output wire [15:0] ReadData1, ReadData2, RegZeroData;

reg [15:0] RegFileArray[0:15];

assign ReadData1 = RegFileArray[ReadReg1];
assign ReadData2 = RegFileArray[ReadReg2];
assign RegZeroData = RegFileArray[0];

integer i;

always @(negedge clock)
begin
   /*for(i = 0; i < 16; i = i + 1)
      $display("RegFileArray[%d] = %h", i, RegFileArray[i]);  */

   if(reset)
   begin
      RegFileArray[0] <= 0;
      RegFileArray[1] <= 'h0f00;
      RegFileArray[2] <= 'h0050;
      RegFileArray[3] <= 'hff0f;
      RegFileArray[4] <= 'hf0ff;
      RegFileArray[5] <= 'h0040;
      RegFileArray[6] <= 'h0024;
      RegFileArray[7] <= 'h00ff;
      RegFileArray[8] <= 'haaaa;
      RegFileArray[9] <= 0;
      RegFileArray[10] <= 0;
      RegFileArray[11] <= 0;
      RegFileArray[12] <= 'hffff;
      RegFileArray[13] <= 'h0002;
      RegFileArray[14] <= 0;
      RegFileArray[15] <= 0;
   end
   else
   begin
      if(RegWrite == 1)
         RegFileArray[WriteReg1] <= WriteData1;
      else if(RegWrite == 2)
      begin
         RegFileArray[WriteReg1] <= WriteData1;
         RegFileArray[WriteReg2] <= WriteData2;
      end
   end
end

endmodule
```

Register File Stimulus

```
`include "regfile.v"

module stimulus;

reg CLOCK, RESET = 1;
reg [1:0] RegWrite = 0;
reg [3:0] ReadReg1 = 0, ReadReg2 = 0, WriteReg1 = 0, WriteReg2 = 0;
reg [15:0] WriteData1 = 0, WriteData2 = 0;
wire [15:0] ReadData1, ReadData2, RegZeroData;

RegFile RF(CLOCK, RESET, RegWrite, ReadReg1, ReadReg2, WriteReg1,
   WriteReg2, WriteData1, WriteData2, ReadData1, ReadData2, RegZeroData);

initial
   $vcdpluson;

initial
   $monitor($time, " RESET = %h RegWrite = %h ReadReg1 = %h ReadReg2 = %h WriteReg1 = %h WriteReg2
= %h WriteData1 = %h WriteData2 = %h ReadData1 = %h ReadData2 = %h RegZeroData = %h", RESET,
RegWrite, ReadReg1, ReadReg2, WriteReg1, WriteReg2, WriteData1, WriteData2, ReadData1, ReadData2,
RegZeroData);

initial
   begin
      #30 RESET = 0;
      #20 ReadReg1 = 3;
         ReadReg2 = 11;
      #20 ReadReg1 = 5;
         ReadReg2 = 14;
      #20 RegWrite = 1;
         WriteReg1 = 8;
         WriteData1 = 'hc78a;
      #20 RegWrite = 0;
         ReadReg1 = 8;
         ReadReg2 = 2;
      #20 RegWrite = 2;
         WriteReg1 = 3;
         WriteData1 = 'h3251;
         WriteData2 = 'haabb;
      #20 RegWrite = 0;
         ReadReg1 = 3;
         ReadReg2 = 0;
      #20 RESET = 1;
      #100 RESET = 0;
   end

initial
begin
   CLOCK = 0;
   forever #10 CLOCK = ~CLOCK;
end

initial
begin
   #680 $finish;
```

```
end

endmodule
```

Register File Results

```
Chronologic VCS simulator copyright 1991-2009
Contains Synopsys proprietary information.
Compiler version D-2009.12; Runtime version D-2009.12;  May  6 04:26 2013
VCD+ Writer D-2009.12 Copyright 2009 Synopsys Inc.

                    0 RESET = 1 RegWrite = 0 ReadReg1 = 0 ReadReg2 = 0 WriteReg1 = 0 WriteReg2 = 0 WriteData1
= 0000 WriteData2 = 0000 ReadData1 = 0000 ReadData2 = 0000 RegZeroData = 0000

                   30 RESET = 0 RegWrite = 0 ReadReg1 = 0 ReadReg2 = 0 WriteReg1 = 0 WriteReg2 = 0 WriteData1
= 0000 WriteData2 = 0000 ReadData1 = 0000 ReadData2 = 0000 RegZeroData = 0000

                   50 RESET = 0 RegWrite = 0 ReadReg1 = 3 ReadReg2 = b WriteReg1 = 0 WriteReg2 = 0 WriteData1
= 0000 WriteData2 = 0000 ReadData1 = ff0f ReadData2 = 0000 RegZeroData = 0000

                   70 RESET = 0 RegWrite = 0 ReadReg1 = 5 ReadReg2 = e WriteReg1 = 0 WriteReg2 = 0 WriteData1
= 0000 WriteData2 = 0000 ReadData1 = 0040 ReadData2 = 0000 RegZeroData = 0000

                   90 RESET = 0 RegWrite = 1 ReadReg1 = 5 ReadReg2 = e WriteReg1 = 8 WriteReg2 = 0 WriteData1
= c78a WriteData2 = 0000 ReadData1 = 0040 ReadData2 = 0000 RegZeroData = 0000

                  110 RESET = 0 RegWrite = 0 ReadReg1 = 8 ReadReg2 = 2 WriteReg1 = 8 WriteReg2 = 0
WriteData1 = c78a WriteData2 = 0000 ReadData1 = c78a ReadData2 = 0050 RegZeroData = 0000

                  130 RESET = 0 RegWrite = 2 ReadReg1 = 8 ReadReg2 = 2 WriteReg1 = 3 WriteReg2 = 0
WriteData1 = 3251 WriteData2 = aabb ReadData1 = c78a ReadData2 = 0050 RegZeroData = 0000

                  140 RESET = 0 RegWrite = 2 ReadReg1 = 8 ReadReg2 = 2 WriteReg1 = 3 WriteReg2 = 0
WriteData1 = 3251 WriteData2 = aabb ReadData1 = c78a ReadData2 = 0050 RegZeroData = aabb

                  150 RESET = 0 RegWrite = 0 ReadReg1 = 3 ReadReg2 = 0 WriteReg1 = 3 WriteReg2 = 0
WriteData1 = 3251 WriteData2 = aabb ReadData1 = 3251 ReadData2 = aabb RegZeroData = aabb

                  170 RESET = 1 RegWrite = 0 ReadReg1 = 3 ReadReg2 = 0 WriteReg1 = 3 WriteReg2 = 0
WriteData1 = 3251 WriteData2 = aabb ReadData1 = 3251 ReadData2 = aabb RegZeroData = aabb

                  180 RESET = 1 RegWrite = 0 ReadReg1 = 3 ReadReg2 = 0 WriteReg1 = 3 WriteReg2 = 0
WriteData1 = 3251 WriteData2 = aabb ReadData1 = ff0f ReadData2 = 0000 RegZeroData = 0000

                  270 RESET = 0 RegWrite = 0 ReadReg1 = 3 ReadReg2 = 0 WriteReg1 = 3 WriteReg2 = 0
WriteData1 = 3251 WriteData2 = aabb ReadData1 = ff0f ReadData2 = 0000 RegZeroData = 0000

$finish called from file "regfile_fixture.v", line 52.
$finish at simulation time                  680
           V C S   S i m u l a t i o n   R e p o r t
Time: 680
CPU Time:      0.410 seconds;       Data structure size:   0.0Mb
Mon May  6 04:26:49 2013
```

Assembly Instruction Set

Function	Syntax	OpCode	op1	op2	Funct. Code	Type	Operation
addition	add op1, op2	0000	reg	reg	0000	A	op1 = op1 + op2
subtraction	sub op1, op2	0000	reg	reg	0001	A	op1 = op1 - op2
bitwise and	and op1, op2	0000	reg	reg	0010	A	op1 = op1 & op2
bitwise or	or op1, op2	0000	reg	reg	0011	A	op1 = op1 | op2
signed multiplication	mult op1, op2	0000	reg	reg	0100	A	op1 = op1 * op2 op1: Product (lower half) R0: Product (upper half)
signed division	div op1, op2	0000	reg	reg	0101	A	op1 = op1 / op2 op1: 16-bit Quotient R0: 16-bit Remainder
logical shift left	sll op1, op2	0000	reg	immd	1000	A	shift op1 left by op2 bits
logical shift right	slr op1, op2	0000	reg	immd	1001	A	shift op1 right by op2 bits
rotate left	rol op1, op2	0000	reg	immd	1010	A	rotate op1 left by op2 bits
rotate right	ror op1, op2	0000	reg	immd	1011	A	rotate op1 right by op2 bits
load	lw op1, immd(op2)	1000	reg	reg	N/A	B	op1 = Mem[immd + op2] (sign extend immd)
store	sw op1, immd(op2)	1011	reg	reg	N/A	B	Mem[immd + op2] = op1 (sign extend immd)
branch on less than	blt op1, op2	0100	reg	immd	N/A	C	if(op1 < R0) then PC = PC + op2 (sign extend op2 and shift left)
branch on greater than	bgt op1, op2	0101	reg	immd	N/A	C	if(op1 > R0) then PC = PC + op2 (sign extend op2 and shift left)
branch on equal	beq op1, op2	0110	reg	immd	N/A	C	if(op1 = R0) then PC = PC + op2 (sign extend op2 and shift left)
jump	jmp op1	1100	offset	-------	N/A	D	PC = PC + op1 (sign extend op1 and shift left)
halt	Halt	1111	-------	-------	N/A	D	halt program execution

Initial and Expected Final State of the Register File

Initial State of the Register File:

Register	Content
R0	0000
R1	0F00
R2	0050
R3	FF0F
R4	F0FF
R5	0040
R6	0024
R7	00FF
R8	AAAA
R9	0000
R10	0000
R11	0000
R12	FFFF
R13	0002
R14	0000
R15	0000

Expected Final State of the Register File:

Register	Content
R0	0000
R1	0001
R2	0050
R3	0050
R4	21FE
R5	0900
R6	0012
R7	00FF
R8	ABCD
R9	0000
R10	0000
R11	0051
R12	FFFF
R13	0002
R14	0000
R15	0000

Assembly Program Used to Test the Stimulated System

Instruction Memory

Addr	Content	Operation	Expected Result
00	ADD R1, R2	R1 = R1 + R2	R1 = 0F50
02	SUB R1, R2	R1 = R1 – R2	R1 = 0F00
04	OR R3, R4	R3 = R3 | R4	R3 = FFFF
06	AND R3, R2	R3 = R3 & R2	R3 = 0050
08	MUL R5, R6	{R0, R5} = R5 * R6	R5 = 0900; R0 = 0000
0A	DIV R1, R5	R1 = R1 / R5; R0 = R1 % R5	R1 = 0001; R0 = 0600
0C	SUB R0, R0	R0 = R0 – R0	R0 = 0000
0E	SLL R4, 3	R4 = R4 << 3	R4 = 87F8
10	SRL R4, 2	R4 = R4 >> 2	R4 = 21FE
12	ROR R6, 3	R6 = R6 ROR 3	R6 = 8004
14	ROL R6, 2	R6 = R6 ROL 2	R6 = 0012
16	BEQ R7, 4	IF(R7 == R0) PC = PC + 6	Branch Not Taken
18	ADD R11, R1	R11 = R11 + R1	R11 = 0001
1A	BLT R7, 5	IF(R7 < R0) PC = PC + 12	Branch Not Taken
1C	ADD R11, R2	R11 = R11 + R2	R11 = 0051
1E	BGT R7, 2	IF(R7 > R0) PC = PC + 6	Branch Taken; PC = 24
20	ADD R1, R1	R1 = R1 + R1	Should Not Execute
22	ADD R1, R1	R1 = R1 + R1	Should Not Execute
24	LW R8, 0(R9)	R8 = Mem[R9 + 0]	R8 = Mem[0] = ABCD
26	ADD R8, R8	R8 = R8 + R8	Arithmetic Overflow Exception
28	SW R8, 2(R9)	Mem[R9 + 2] = R8	Should Not Execute
2A	LW R10, 2(R9)	R10 = Mem[R9 + 2]	Should Not Execute
2C	ADD R12, R12	R12 = R12 + R12	Should Not Execute
2E	SUB R13, R13	R13 = R13 - R13	Should Not Execute
30	ADD R12, R13	R12 = R12 + R13	Should Not Execute
32	EFFF	Invalid Instruction	Should Not Execute

Term Project Status Report

Name	*% Contribution*	*Grade*
Michael Colson	100	

Project Report/Presentation 20%	/200
Functionality of the individual components 35%	/350
Functionality of the overall design 30%	/300
Design Approach 15%	/150
Total points	/1000

A: List all the instructions that were implemented correctly and verified by the assembly program:

Instructions	**State any issue regarding the instruction.**
addition	Implemented Correctly
subtraction	Implemented Correctly
bitwise and	Implemented Correctly
bitwise or	Implemented Correctly
signed multiplication	Implemented Correctly
signed division	Implemented Correctly
Logical shift left	Implemented Correctly
Logical shift right	Implemented Correctly
rotate left	Implemented Correctly
rotate right	Implemented Correctly
load	Implemented Correctly
store	Implemented Correctly
branch on less than	Implemented Correctly
branch on greater than	Implemented Correctly
branch on equal	Implemented Correctly
jump	Implemented Correctly
halt	Implemented Correctly

B: Individual System Components:

Individual Components	Does your system have this component	Does it work ?	List problems with the component, if any.
ALU	Yes	Yes	None
ALU control unit	No		
Memory Unit	Yes	Yes	None
Register File	Yes	Yes	None
PC	Yes	Yes	None
IR	Yes	Yes	None
Other registers	EPC (Exception Program Counter) Exception (Indicates if an exception has occured or not)	Yes	None
Multiplexors	Yes	Yes	None
Control Units 1. main 2. forwarding 3. lw hazard	Yes	Yes	None

How many stages do you have in your pipeline?

5 stages (IF, ID, EX, MEM and WB stages)

C: State any issue regarding the overall operation of the datapath?

No known issues regarding the overall operation of the datapath.

Control Logic Truth Tables

Control Unit:

OpCode	Funct. Code	IFIDImmd	IFIDMemWrite	IFIDMemRead	IFIDRegWrite
0000	0000	0	0	0	1
0000	0001	0	0	0	1
0000	0010	0	0	0	1
0000	0011	0	0	0	1
0000	0100	0	0	0	2
0000	0101	0	0	0	2
0000	1000	1	0	0	1
0000	1001	1	0	0	1
0000	1010	1	0	0	1
0000	1011	1	0	0	1
1000	xxxx	0	0	1	1
1001	xxxx	0	1	0	0
0100	xxxx	1	0	0	0
0101	xxxx	1	0	0	0
0110	xxxx	1	0	0	0
1100	xxxx	1	0	0	0
1111	xxxx	x	0	0	0

Hazard Detection Unit

EXMEMMemRead	IDEXImmd	IDEXRegOp1	IDEXRegOp2	EXMEMRegOp1	stall
0	0	0000	0000	0000	0
0	0	1011	0011	0000	0
0	0	1101	0101	0101	0
0	0	1000	1010	1000	0
0	1	0000	0000	0000	0
0	1	1011	0011	0000	0
0	1	1101	0101	0101	0
0	1	1000	1010	1000	0
1	0	0000	0000	0000	1
1	0	1011	0011	0000	0
1	0	1101	0101	0101	1
1	0	1000	1010	1000	1
1	1	0000	0000	0000	0
1	1	1011	0011	0000	0
1	1	1101	0101	0101	0
1	1	1000	1010	1000	0

Forwarding Unit

IDEXIR	EXMEMIR	MEMWBIR	ForwardA	ForwardB
‘h0000	‘h0234	‘h0560	‘b0000	‘b0000
‘h0050	‘h0565	‘h07a0	‘b0001	‘b0010
‘h0300	‘h0ab1	‘h0344	‘b0011	‘b0100
‘h0090	‘h0120	‘h05c5	‘b0101	‘b0110
‘h0451	‘h0890	‘h0123	‘b0110	‘b0110