

UNIVERSITY OF WATERLOO AQ

Faculty of Engineering
Department of Electrical and Computer Engineering
ECE 621- Computer Organization

Register File and Execute Stage

Group 2

Jordan Ross, Mustafa Faraj
j25ross, mabdulaz
31 October, 2014

ALU Module Code

This section contains the code for our ALU module. It takes in the two source operands as well as an operation control signal (set in the decode stage). It outputs the result of the arithmetic operation as well as set two condition bits (zero and negative).

```
module alu(
      input [31:0] op1, // operand 1 (always from rs)
      input [31:0] op2, // operand 2
      input [5:0] operation, // The arithmatic operation to perform
      input [5:0] shift amount, // The number of bits to shift
      output reg [31:0] result, // The result based on the operation
      output reg zero, // Indicates if the result of the operation is zero.
      output reg neg // Indicates if the result is negative or not.
);
      // The operations that the ALU supports are:
      // 0: Addition
      // 1: Subtraction
      // 2: Multiply
      // 3: Divide
      // 4: Shift logical left
      // 5: Shift logical right
      // 6: Set on less than
      // 7: And
      // 8: Or
      // 9: Xor
      // 10: Nor
      // 11: SRA
      // 12: Load upper
      always @(op1 or op2 or operation or shift amount) begin
            case (operation)
                  0: result = op1+op2;
                  1: result = op1-op2;
                  2: result = op1*op2;
                  3: begin
                        if (op2 != 0) begin
                              result = op1/op2;
                        // TODO: Otherwise we should trap here
                  end
                  4: result = op2 << shift amount;
                  5: result = op2 >> shift amount;
                  6: result = op1 < op2 ? 1 : 0;
                  7: result = op1 & op2;
                  8: result = op1 | op2;
                  9: result = op1 ^ op2;
                  10: result = ~(op1 | op2);
                  11: result = op2 >>> shift amount;
                  12: result = (op2 << 16) &32'hffff0000;</pre>
            endcase
            zero = result == 0 ? 1 : 0;
            neg = result[31];
      end
endmodule
```

Branch Resolve Unit

This section contains our logic for selecting whether a branch should be taken or not based on the two condition bits set by the ALU.

```
module branch resolve (
      input zero,
      input neg,
      input [1:0]branch type,
      input is branch,
      output reg branch taken // Indicates if the branch is taken or not
);
      always @(zero or neg or branch type or is branch) begin
            if (is branch == 1'b1) begin
                  case (branch type)
                        0: branch taken = zero == 0 ? 1 : 0; // BEQ
                        1: branch taken = zero != 0 ? 1 : 0; // BNE
                        2: branch taken = (neg | zero) == 0 ? 1 : 0; // BLEZ
                        3: branch taken = neg != 0 ? 1 : 0; // BGTZ
                  endcase
            end else begin
                  branch taken = 0;
            end
      end
endmodule
```

Processor Module Code

This section contains the code for the processor module. The processor module is our top level module where we instantiate our fetch, decode, memory, register file, and ALU so far. This is also where we have our control signals being set. The control is only being done in the decode stage and the control signals that are needed in later stages are latched in pipeline registers. The processor module currently has the IF/ID and ID/IX pipeline registers implemented.

```
module processor(
   input clk, // The system clock
   );
   // Decoder signals
   wire [4:0]rs;
   wire [4:0]rt;
   wire [4:0]rd;
   wire [4:0]sha;
   wire [5:0]func;
   wire [15:0]immed;
   wire [25:0]target;
   wire [5:0]opcode;
   wire [31:0]pc out;
   wire [31:0]insn out;
   // Control signals
   reg stall;
   wire [1:0]insn access size;
   wire [1:0] fetch access size;
```

```
wire fetch rw;
wire insn rw;
reg reg file write enable;
// Address lines
wire [31:0]pc;
// Data lines
wire [31:0]insn data out;
wire [31:0]insn address;
// SREC registers (for helping the parser write to instruction memory)
reg [31:0]srec address;
reg [31:0] srec data in;
reg srec rw;
reg [1:0]srec access size;
// Decode wires
wire [31:0]decode pc; // the PC for next instruction to be executed
wire [31:0] decode ir; // the current instruction being decoded
wire [31:0]dec A;
wire [31:0]dec B;
wire [4:0]dest reg;
// Decode control signals
reg dest reg sel;
reg dec illegal insn;
reg [5:0] dec alu op;
reg dec is branch;
reg dec op2 sel;
reg [5:0] dec shift amount;
reg [1:0]dec branch type; // 0-BEQ, 1-BNE, 2-BLEZ, 3-BGTZ
reg dec is jump;
// Execute wires
wire [31:0] exe pc; // the PC for next instruction to be executed
wire [31:0]exe_ir; // the current instruction being executied
wire [31:0] exe A;
wire [31:0] exe B;
 wire [31:0] exe op2;
wire [31:0] exe 0;
wire [31:0] exe extended; // The wire the comes for the sign extender
wire exe zero;
wire [5:0] exe alu op;
wire exe_is_branch;
wire exe op2 sel;
wire [5:0] exe shift amount;
wire [31:0] exe shift immed;
wire [31:0] exe shift target;
wire [31:0] exe jump effective address;
wire [31:0] exe branch effective address;
wire exe neg;
wire [1:0] exe branch type;
wire exe branch taken;
wire [31:0] exe next pc;
wire [31:0]exe_branch next pc;
wire exe is jump;
```

```
// Instantiate mux's for each of the SREC registers to aid the SREC
parser.
    mux 2 1 32 bit srec insn_address_mux(
        .line0(pc),
        .line1(srec address),
        .select(srec parse),
        .output line (insn address)
    );
    mux 2 1 1 bit srec insn rw mux(
        .lineO(fetch rw),
        .line1(srec rw),
        .select(srec parse),
        .output line(insn rw)
    mux_2_1_2_bit srec_insn_access_size_mux(
        .lineO(fetch_access_size),
        .line1(srec access size),
        .select(srec parse),
        .output line(insn access size)
    );
    // Instantiate the fetch module
    fetch fetch (
        .clk in(clk),
        .stall in(stall),
        .pc out (pc),
        .rw out (fetch rw),
        .access size out (fetch access size)
    );
    // Instantiate the instruction memory module
    memory insn memory(
        .data out (insn data out),
        .address(insn address),
        .data in(srec data in), // We can tie the srec data in wire to this
port since we should never be writing to instruction memory unless we are
srec parsing
        .write(insn rw),
        .clk(clk),
            .access size(insn access size)
    );
    // Instatiate the IF/ID pipeline register to kep the PC and IR
    if id pipleline reg if id pipleline reg(
        .clk(clk),
        .pc in(pc),
        .ir in(insn data out),
        .pc_out(decode pc),
        .ir out(decode ir)
    );
    // Instantiate a mux for selecting which destination to choose
    mux 2 1 5 bit dest reg mux (
```

```
.line0(rt),
    .line1(rd),
    .select(dest reg sel), // TODO: Implement this control singal
    .output line(dest reg)
);
// Instantiate the register file
reg file reg file (
    .clk(clk),
    .write enable (reg file write enable),
    .source1(rs),
    .source2(rt),
    .dest(dest reg),
    .destVal(exe O), // TODO: Change this to be the output from WB stage
    .slval(dec A),
    .s2val(dec B)
);
// Instantiate the decode module
decode decoder (
    .clk(clk),
    .stall(stall),
    .insn in (decode ir),
    .pc in(pc),
    .rs(rs),
    .rt(rt),
    .rd(rd),
    .sha(sha),
    .func(func),
    .immed(immed),
    .target(target),
    .opcode (opcode),
    .pc out (pc out),
    .insn out(insn out)
);
// Instatiate the ID/IX pipeline register
id ix pipleline reg id ix pipleline reg(
    .clk(clk),
    .pc in (decode pc),
    .ir in (decode ir),
    .A in (dec A),
    .B_in(dec B),
    .alu op in (dec alu op),
    .is_branch_in(dec_is_branch),
    .is jump in (dec is jump),
    .op2 sel in(dec op2 sel),
    .shift amount in (dec shift amount),
    .branch type in (dec branch type),
    .pc out (exe pc),
    .ir out (exe ir),
    .A out (exe A),
    .B out (exe B),
    .alu op out (exe alu op),
    .is branch out (exe is branch),
    .is jump out (exe is jump),
    .op2 sel out(exe op2 sel),
```

```
.shift amount out (exe shift amount),
        .branch type out (exe branch type)
    );
    sign extender sign extender (
        .in data(exe ir[15:0]),
        .out data(exe extended)
    );
    assign exe shift target = exe ir[25:0];
    assign exe jump effective address = (exe pc & 32'hf0000000) |
((exe shift target << 2) & 32'h0fffffff);</pre>
    assign exe shift immed = exe extended << 2;</pre>
    assign exe branch effective address = exe shift immed + exe pc;
    // Instantiate a 32-bit mux for selecting which operand to provide to op2
of the ALU
    mux 2 1 32 bit alu op2 sel mux(
        .line0(exe B),
        .line1(exe extended),
        .select(exe op2 sel),
        .output line (exe op2)
    );
    alu alu(
        .op1(exe A), // operand 1 (always from rs)
        .op2 (exe op2), // operand 2
        .operation(exe alu op), // The arithmatic operation to perform
        .shift amount(exe shift amount), // The number of bits to shift
        .result (exe O), // The arithmatic result based on the operation
        .zero(exe zero), // Indicates if the result of the operation is zero.
        .neg(exe neg)
    );
    branch resolve branch resolve (
      .zero(exe zero),
            .neg(exe neg),
            .branch type (exe branch type),
            .is branch (exe is branch),
            .branch taken(exe branch taken) // Indicates if the branch is
taken or not
    );
    mux 2 1 32 bit branch next pc mux (
        .line0(exe pc),
        .line1 (exe branch effective address),
        .select (exe branch taken),
        .output line (exe branch next pc)
    );
    mux_2_1_32_bit jump_next_pc_mux(
        .line0 (exe branch next pc),
        .line1(exe_jump effective address),
        .select(exe is jump),
        .output line (exe next pc)
    );
```

```
// Control
always @(posedge clk) begin
    reg file write enable = 0;
    // ----- Decode Stage Control Signal Logic ----- //
    dec illegal insn = 0;
    dest req sel = 0;
   dec alu op = 0;
   dec op2 sel = 0;
   dec is branch = 0;
    dec is jump = 0;
   dec branch type = 0;
    if (((opcode & 6'b111000)>> 3) == 3'h0) begin
        if ((opcode & 6'b000111) == 3'h0) begin
              // We are in the SPECIAL Opcode encoding table
              // This is an R-type instruction
              dest_reg_sel = 0;
              if (func == 6'b100000 || func == 6'b100001) // ADD, ADDU
                    dec alu op = 0; // Do an add operation
              else if (func == 6'b100010 || func == 6'b100011)//SUB, SUBU
                   dec alu op = 1;
              else if (func == 6'b011000 || func==6'b011001) //MULT, MULTU
                    dec alu op = 2;
              else if (func == 6'b011010 || func==6'b011011) // DIV, DIVU
                    dec alu op = 3;
              else if (func == 6'b000000) begin // SLL
                   dec alu op = 4;
                    dec shift amount = sha;
              end
              else if (func == 6'b000010) begin // SLL
                    dec alu op = 5;
                    dec shift amount = sha;
              else if (func == 6'b101010 || func==6'b101011) // SLT, SLTU
                   dec alu op = 6;
            else begin
              dec illegal insn = 1;
       end else if (((opcode & 6'b000111) == 3'd2) || ((opcode &
           6'b000111) == 3'd3)) begin
              // J and JAL instructions
              // This is J-type instruction
              dest reg sel = 0;
              dec_{is_jump} = 1;
              if (opcode == 6'b000011) begin
                    // JAL instruction
                    dec illegal insn = 1;
                    // TODO: store the return address into reg[31]
              end
       end else begin
              // BEQ, BNE, BLEZ, BGTZ
              // This is an I-type instruction
              dec is branch = 1;
              if (opcode == 6'b000100) begin
                    // BEQ
                    dec alu op = 1; // We want to test if the result is z
                                      zero from a subtract
```

```
dec branch type = 0;
          end else if (opcode == 6'b000101) begin
                // BNE
                dec alu op = 1; // We want to do a test if the result
               is zero from a subtract
                dec branch type = 1;
          end else if (opcode == 6'b000110) begin
                // BLEZ
                // TODO: Test that rt has zero in it
                dec illegal insn = 1;
                dec alu op = 0;
                dec branch type = 2;
          end else begin
                // BGTZ
                // TODO: Test that rt has zero in it
                dec_illegal_insn = 1;
                dec_alu op = 0;
                dec branch type = 3;
          end
    end
end else if (((opcode & 6'b111000)>> 3) == 3'd1) begin
    // ADDI, ADDIU, SLTI, SLTIU, ANDI, ORI, XORI, LUI
    // This is an I-type instruction
    case (opcode)
          6'b001000: dec alu op = 0; // ADDI
          6'b001001: dec alu op = 0; // ADDIU
          6'b001010: dec alu op = 6; // SLTI
          6'b001011: dec alu op = 6; // SLTUI
          6'b001100: dec alu op = 7; // ANDI
          6'b001101: dec_alu_op = 8; // ORI
6'b001110: dec_alu_op = 9; // XORI
          6'b001111: dec alu op = 12; // LUI
    endcase
    dest reg sel = 1;
    dec op2 sel = 1; // We want op2 to be the immediate in the ALU
end else if (((opcode & 6'b111000)>> 3) == 3'd3) begin
    dest reg sel = 0;
    dec illegal insn = 1;
end else if (((opcode & 6'b111000)>> 3) == 3'd4) begin
    // LB, LH, LWL, LW, LBU, LHU, LWR
    // This is an I-type instruction
    dest reg sel = 1;
    dec op2 sel = 1;
    dec alu op = 0; // add the value of rs to the immediate
end else if (((opcode & 6'b111000)>> 3) == 3'd5) begin
    // SB, SH, SWL, SW, SWR
          // This is an I-type instruction
    dest reg sel = 1;
    dec op2 sel = 1;
    dec alu op = 0; //add the value of rs to the immediate
end else begin
    dec illegal insn = 1;
end
```

end

endmodule

Processor Test Bench

We did not change anything in our test bench actually but the code is shown here again for reference.

```
module processor tb;
   reg clk;
   reg srec parse; // enable signal for srec parser.
   // Registers, writes, and variables for parser
   integer fh = 0; // file handler for input
   integer i = 0; // loop variable
   integer data byte = 0; // variable to keep track of what byte we are on.
   integer data offset = 0; // keep track of the offset from the data
                             address to write the next byte.
   reg [1:0]nibble count = 0; // keep track of which nibble is being written
   reg [7:0]rec type; // record type number
   reg [7:0] byte count; //byte count for address, data, and checksum
   integer record_code; // A record_code is equivalent to 1 ASCII
                      digit/letter in the .srec file
   reg [31:0] rec address = 'b0; // the address given by the record.
   integer highest address = 0; // highest address written by parser
   reg [7:0] rec data; // A single byte of the data from the record.
   reg [7:0]temp; // a temporary byte used for place holding.
   req done = 0; // this will set high when we are done parsing the file.
   reg [7:0] file char = 8'hOA; // Set the initial character from the file
                               read to be the new line character
   // Instantiate the processor as the unit under test.
   processor processor uut(.clk(clk), .srec parse(srec parse));
   initial begin
       //-----
       // Parsing stage of testbench - memory is not valid until
       // after the parser has finished!
       //-----
           //SREC file parsing has been omitted from the report for clarity.
       // Memory is ready to be used after this point!
       $monitor("Beginning the fetch-decode-execute loop!");
       #100;
       // Deassert stall just read out the pc, rw, and access size.
       processor uut.stall = 0;
       // -----
       // This section prints out the decoded instructions
       while (processor uut.pc <= highest address - 4) begin
           @ (posedge clk);
```

```
case (processor uut.opcode)
   6'd0: begin //JR, ADD, ADDU, SUB SUBU, DIV, SLT, SLTU, SLL,
                 SRL, SRA, AND, OR, XOR, NOR, NOP
       case(processor uut.func)
       6'd0: begin //SLL, NOP
           if (processor uut.sha == 5'b0) $strobe("%h:
            NOP", processor_uut.pc_out, processor_uut.insn_out);
           else $strobe("%h: %h sll %h, %h, %h",
               processor uut.pc out, processor uut.insn out,
                processor uut.rd, processor uut.rt,
                processor uut.sha);
       end
       6'd2: begin //SRL
           $strobe("%h:
                           %h
                                 srl %h, %h, %h",
               processor_uut.pc_out, processor_uut.insn_out,
                processor uut.rd, processor uut.rt,
                processor_uut.sha);
       end
       6'd3: begin //SRA
           $strobe("%h:
                           %h
                                sra %h, %h, %h",
               processor uut.pc out, processor uut.insn out,
                processor uut.rd, processor uut.rt,
                processor uut.sha);
       end
       6'd8: begin //JR
           $strobe("%h:
                          %h
                                jr %h",
               processor uut.pc out, processor uut.insn out,
               processor uut.rs);
       end
       6'd26: begin //DIV
           $strobe("%h:
                          %h div %h, %h",
               processor uut.pc out, processor uut.insn out,
               processor uut.rs, processor uut.rt);
       end
       6'd32: begin //ADD
           $strobe("%h:
                           %h
                                add %h, %h, %h",
               processor uut.pc out, processor uut.insn out,
                processor uut.rd, processor uut.rs,
                processor uut.rt);
       end
       6'd33: begin //ADDU
           $strobe("%h:
                          %h
                                addu %h, %h, %h",
               processor uut.pc out, processor uut.insn out,
                processor_uut.rd, processor_uut.rs,
                processor_uut.rt);
       end
       6'd34: begin //SUB
           $strobe("%h:
                                sub %h, %h, %h",
                          %h
               processor uut.pc out, processor uut.insn out,
                processor uut.rd, processor uut.rs,
                processor uut.rt);
       end
       6'd35: begin //SUBU
           $strobe("%h: %h subu %h, %h, %h",
               processor uut.pc out, processor uut.insn out,
               processor uut.rd, processor_uut.rs,
                processor uut.rt);
```

```
end
   6'd36: begin //AND
                      %h and %h, %h, %h",
        $strobe("%h:
           processor uut.pc out, processor uut.insn out,
            processor uut.rd, processor uut.rs,
            processor uut.rt);
    end
    6'd37: begin //OR
       $strobe("%h:
                       %h
                             or %h, %h, %h",
           processor_uut.pc_out, processor_uut.insn_out,
            processor uut.rd, processor uut.rs,
            processor uut.rt);
    end
    6'd38: begin //XOR
       $strobe("%h:
                       %h xor %h, %h, %h",
           processor uut.pc out, processor uut.insn out,
            processor_uut.rd, processor uut.rs,
            processor uut.rt);
    end
   6'd39: begin //NOR
       $strobe("%h:
                       %h
                            nor %h, %h, %h",
           processor uut.pc out, processor uut.insn out,
            processor uut.rd, processor uut.rs,
            processor uut.rt);
    end
    6'd42: begin //SLT
       $strobe("%h:
                       %h
                             slt %h, %h, %h",
           processor uut.pc out, processor uut.insn out,
            processor uut.rd, processor uut.rs,
            processor uut.rt);
    end
   6'd43: begin //SLTU
       $strobe("%h:
                            sltu %h, %h, %h",
                      %h
           processor uut.pc out, processor uut.insn out,
            processor uut.rd, processor uut.rs,
            processor uut.rt);
    end
    default: begin
       $strobe("ERROR! %h: %h", processor uut.pc out,
            processor uut.insn out);
       $stop;
    end
   endcase
6'd1: begin //BLTZ, BGEZ
    case(processor uut.rt)
   5'd0: begin //BLTZ
        $strobe("%h:
                       %h
                             bltz %h, %h",
           processor_uut.pc_out, processor_uut.insn_out,
            processor uut.rs, processor uut.immed);
    end
    5'd1: begin //BGEZ
       $strobe("%h:
                       %h
                             bgez %h, %h",
           processor uut.pc out, processor uut.insn out,
            processor uut.rs, processor uut.immed);
    end
    default: begin
```

```
$strobe("ERROR! %h: %h", processor uut.pc out,
            processor uut.insn out);
        $stop;
    end
    endcase
end
6'd2: begin //J
    //display the destination address of the jump, not the
    offset.
    $strobe("%h:
                   %h
                          j %h",
            processor uut.pc out, processor uut.insn out,
            {processor uut.pc out[31-:4],
            28'b0}+{processor uut.target,2'b0});
end
6'd3: begin //JAL
    //display the destination address of the jump, not the
    offset.
    $strobe("%h:
                    %h
                          jal %h",
           processor uut.pc out, processor uut.insn out,
            {processor uut.pc out[31-:4],
            28'b0}+{processor uut.target,2'b0});
end
6'd4: begin //BEQ
    $strobe("%h:
                          beq %h, %h, %h",
                    %h
           processor uut.pc out, processor uut.insn out,
            processor uut.rs, processor uut.rt,
            processor uut.immed);
end
6'd5: begin //BNE
    $strobe("%h:
                   %h
                          bne %h, %h, %h",
            processor_uut.pc_out, processor_uut.insn_out,
            processor_uut.rs, processor_uut.rt,
            processor uut.immed);
end
6'd6: begin //BLEZ
    $strobe("%h:
                          beq %h, %h",
                    %h
            processor uut.pc out, processor uut.insn out,
            processor uut.rs, processor uut.immed);
end
6'd7: begin //BGTZ
    $strobe("%h:
                          batz %h, %h",
                   응h
           processor uut.pc out, processor uut.insn out,
            processor uut.rs, processor uut.immed);
6'd9: begin //ADDIU
    $strobe("%h:
                    %h
                          addiu %h, %h, %d",
            processor uut.pc out, processor uut.insn out,
            processor uut.rt, processor uut.rs,
            processor uut.immed);
end
6'd10: begin //SLTI
    $strobe("%h:
                          slti %h, %h, %h",
                   %h
            processor uut.pc out, processor uut.insn out,
            processor uut.rt, processor uut.rs,
            processor uut.immed);
end
6'd13: begin //ORI
```

```
ori %h, %h, %h",
                    processor uut.pc out, processor uut.insn out,
                    processor uut.rt, processor uut.rs,
                    processor uut.immed);
        end
        6'd15: begin //LUI
            $strobe("%h:
                            응h
                                  lui %h, %h",
                    processor uut.pc out, processor uut.insn out,
                    processor uut.rt, processor uut.immed);
        end
        6'd28: begin //MUL
            $strobe("%h:
                            %h
                                  mul %h, %h, %h",
                    processor uut.pc out, processor uut.insn out,
                    processor uut.rd, processor_uut.rs,
                    processor uut.rt);
        end
        6'd32: begin //LB
            $strobe("%h:
                            %h
                                  lb %h, %d(%h)",
                    processor uut.pc out, processor uut.insn out,
                    processor uut.rt, processor uut.immed,
                    processor uut.rs);
        end
        6'd35: begin //LW
            $strobe("%h:
                                  lw %h, %d(%h)",
                            %h
                    processor uut.pc out, processor uut.insn out,
                    processor uut.rt, processor uut.immed,
                    processor uut.rs);
        end
        6'd36: begin //LBU
            $strobe("%h:
                                  lbu %h, %d(%h)",
                           %h
                    processor_uut.pc_out, processor_uut.insn_out,
                    processor_uut.rt, processor_uut.immed,
                    processor uut.rs);
        end
        6'd40: begin //SB
            $strobe("%h:
                                  sb %h, %d(%h)",
                           %h
                    processor uut.pc out, processor uut.insn out,
                    processor uut.rt, processor uut.immed,
                    processor uut.rs);
        end
        6'd43: begin //SW
            $strobe("%h:
                                  sw %h, %d(%h)",
                            응h
                    processor uut.pc out, processor uut.insn out,
                    processor uut.rt, processor uut.immed,
                    processor uut.rs);
        end
        default: begin
            $strobe("ERROR! %h:
                                  %h", processor uut.pc out,
                    processor_uut.insn_out);
            $stop;
        end
   endcase
end
#100;
$stop;
```

\$strobe("%h:

%h

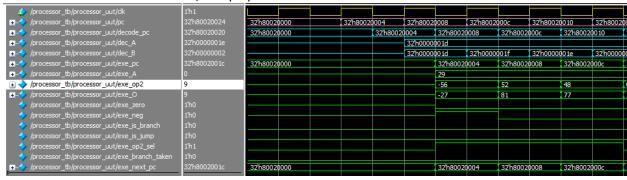
```
end
  always begin
        #50 clk = !clk;
  end
// A function to convert ASCII upper case letters and digits to their
hexadecimal value.
function [7:0] atoh;
    input [7:0]aCode;
    begin
        if (aCode >= 8'h30 && aCode <= 8'h39) begin
            atoh = aCode - 8'h30;
        end else if (aCode >= 8'h41 && aCode <= 8'h5A) begin
            atoh = aCode - 8'h37;
        end
    end
endfunction
```

endmodule

Verification Demonstration

For the verification the setup for us that it is easiest to show is the waveform diagram as the program executes. For readability purposes we have only shown 4 instructions per waveform along with the instruction printout we get from our test bench (this was verified last PD). We show the operands to the ALU, and output from the ALU as decimal to improve readability. The plum colored wave forms are for the fetch stage while the cyan ones are for decode and the green for execute. We executed the bubble sort bench mark in the following diagrams.

80020004: 27bdffc8 addiu 1d, 1d, 65480 # 80020008: afbf0034 sw 1f, 52(1d) # 8002000c: afbe0030 sw 1e, 48(1d)



```
#80020010: 03a0f021 addu 1e, 1d, 00

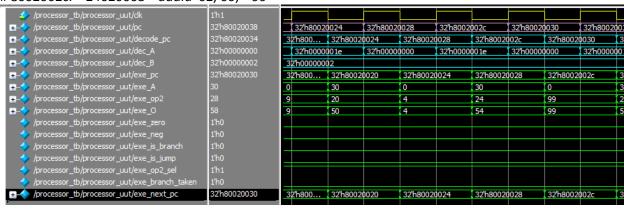
#80020014: 2402000c addiu 02, 00, 12

#80020018: afc20010 sw 02, 16(1e)

#8002001c: 24020009 addiu 02, 00, 9
```

<pre>/processor_tb/processor_uut/clk</pre>	-No Data-										
 /processor_tb/processor_uut/pc	-No Data-		32'h80020	014	32'h80020	018	32'h8002	001c	32'h8002	020	32'h8002
#	-No Data-	32h	18002	32'h80020	014	32'h80020	018	32'h80020	01c	32'h8002	0020
	-No Data-	32h	10000001	d	32'h00000	0000	32'h0000	001e	32'h00000	0000	32'h0000
	-No Data-		32'h00000	000	32'h00000	002					
/processor_tb/processor_uut/exe_pc	-No Data-	32h	18002	32'h80020	0010	32'h80020	014	32'h80020	018	32'h8002	001c
 /processor_tb/processor_uut/exe_A	-No Data-	29				0		30		0	
/processor_tb/processor_uut/exe_op2	-No Data-	48		0		12		16		9	
	-No Data-	77		29		12		46		9	
/processor_tb/processor_uut/exe_zero	-No Data-										
/processor_tb/processor_uut/exe_neg	-No Data-										
/processor_tb/processor_uut/exe_is_branch	-No Data-										
/processor_tb/processor_uut/exe_is_jump	-No Data-										
/processor_tb/processor_uut/exe_op2_sel	-No Data-										
/processor_tb/processor_uut/exe_branch_taken	-No Data-										
→ /processor_tb/processor_uut/exe_next_pc	-No Data-	32	18002	32'h80020	0010	32'h80020	014	32'h80020	018	32'h8002	001c
· ·											

80020020: afc20014 sw 02, 20(1e) # 80020024: 24020004 addiu 02, 00, 4 # 80020028: afc20018 sw 02, 24(1e) # 8002002c: 24020063 addiu 02, 00, 99



80020030: afc2001c sw 02, 28(1e) # 80020034: 24020078 addiu 02, 00, 120 # 80020038: afc20020 sw 02, 32(1e) # 8002003c: 24020001 addiu 02, 00, 1

<pre>/processor_tb/processor_uut/dk</pre>	1'h1									
≖ - <pre>/processor_tb/processor_uut/pc</pre>	32'h80020038	32'h80020	034	32'h8002	038	32'h80020	003c	32'h80020	040	32'h80020
/processor_tb/processor_uut/decode_pc	32'h80020034	32'h800	32'h8002	0034	32'h80020	038	32'h80020	03c	32'h80020	0040
/processor_tb/processor_uut/dec_A	32'h00000000	32'h00000	01e	32'h0000	000	32'h00000	001e	32'h00000	000	32'h00000
/processor_tb/processor_uut/dec_B	32'h00000002	32'h0000000)2							
/processor_tb/processor_uut/exe_pc	32'h80020030	32'h800	32'h8002	0030	32'h80020	034	32'h80020	038	32'h80020	003c
/processor_tb/processor_uut/exe_A	30	0	30		0		30		0	
≖ – ∜ /processor_tb/processor_uut/exe_op2	28	99	28		120		32		1	
/processor_tb/processor_uut/exe_O	58	99	58		120		62		1	
/processor_tb/processor_uut/exe_zero	1'h0									
<pre>/processor_tb/processor_uut/exe_neg</pre>	1'h0									
/processor_tb/processor_uut/exe_is_branch	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
+	32'h80020030	32'h800	32'h8002	0030	32'h80020	034	32'h80020	038	32'h80020	003c

#80020040: afc20024 sw 02, 36(1e)

#80020044: 24020003 addiu 02, 00, 3 #80020048: afc20028 sw 02, 40(1e) #8002004c: 2402000a addiu 02, 00, 10

<pre>/processor_tb/processor_uut/clk</pre>	1'h1										
∓ - <pre>/processor_tb/processor_uut/pc</pre>	32'h80020038	32'h80020	044	32'h80020	048	32'h80020	004c	32'h80020	050	32'h80020	05
/processor_tb/processor_uut/decode_pc	32'h80020034	32'h800	32'h8002	0044	32'h80020	048	32'h80020)04c	32'h80020	0050	3
//processor_tb/processor_uut/dec_A	32'h00000000	32'h00000	01e	32'h00000	0000	32'h00000	001e	32'h00000	0000	32'h00000	01
/processor_tb/processor_uut/dec_B	32'h00000002	32'h0000000)2								
≖ - ∜ /processor_tb/processor_uut/exe_pc	32'h80020030	32'h800	32'h8002	0040	32'h80020	044	32'h80020	048	32'h80020	104c	3
	30	0	30		0		30		0		30
	28	1	36		3		40		10		4
/processor_tb/processor_uut/exe_O	58	1	66		3		70		10		7
<pre>/processor_tb/processor_uut/exe_zero</pre>	1'h0										
<pre>/processor_tb/processor_uut/exe_neg</pre>	1'h0										
/processor_tb/processor_uut/exe_is_branch	1'h0										
<pre>/processor_tb/processor_uut/exe_is_jump</pre>	1'h0										
/processor_tb/processor_uut/exe_op2_sel	1'h1										ı
/processor_tb/processor_uut/exe_branch_taken	1'h0										
processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h8002	0040	32'h80020	044	32'h80020	048	32'h80020	004c	3:

#80020050: afc2002c sw 02, 44(1e) #80020054: 27c20010 addiu 02, 1e, 16 #80020058: 00402021 addu 04, 02, 00 #8002005c: 24050008 addiu 05, 00, 8

· ·										
<pre>/processor_tb/processor_uut/dk</pre>	1'h1									
<pre>#</pre> /processor_tb/processor_uut/pc	32'h80020038	, 32'h80020	054	32'h80020	058	32'h80020	05c	32'h80020	060	32'h800200
<pre>#</pre> /processor_tb/processor_uut/decode_pc	32'h80020034	32'h800	32'h8002	0054	32'h80020	058	32'h80020	05c	32'h80020	0060 3
<pre>#</pre>	32'h00000000	32'h00000	01e			32'h00000	0002	32'h00000	000	
<pre>#=-</pre> /processor_tb/processor_uut/dec_B	32'h00000002	32'h0000000	2			32'h00000	0000	32'h00000	005	32'h000000
<pre>#=-</pre> /processor_tb/processor_uut/exe_pc	32'h80020030	32'h800	32'h8002	0050	32'h80020	054	32'h80020	058	32'h80020	005c 3
<pre>#=-</pre> /processor_tb/processor_uut/exe_A	30	0	30				2		0	
/processor_tb/processor_uut/exe_op2	28	10	44		16		0		8	
<pre>#-</pre> /processor_tb/processor_uut/exe_O	58	10	74		46		2		8	
/processor_tb/processor_uut/exe_zero	1'h0									
/processor_tb/processor_uut/exe_neg	1'h0									
/processor_tb/processor_uut/exe_is_branch	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
# /processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h8002	050	32'h80020	054	32'h80020	058	32'h80020	005c 3

#80020060: 0c008020 jal 80020080

#80020064: 00000000 NOP

#80020068: 00001021 addu 02,00,00 #8002006c: 03c0e821 addu 1d,1e,00

# 0002000C. 03C0E821 addd 1	u, =0, 00									
<pre>/processor_tb/processor_uut/clk</pre>	1'h1									
-/-/ /processor_tb/processor_uut/pc	32'h80020038	32'h80020	064	32'h8002	068	32'h8002	06c	32'h80020	070	32'h8002
	32'h80020034	32'h800	32'h8002	064	32'h80020	0068	32'h80020	06c	32'h80020	070
+-/processor_tb/processor_uut/dec_A	32'h00000000	32'h0000000	0					32'h00000	01e	32'h0000
+	32'h00000002	32'h00000	0000							32'h0000
+	32'h80020030	32'h800	32'h8002	0060	32'h80020	0064	32'h80020	068	32'h80020	06c
+	30	0							30	
+	28	8	0							
+	58	8	0						30	
/processor_tb/processor_uut/exe_zero	1'h0									
/processor_tb/processor_uut/exe_neg	1'h0									
/processor_tb/processor_uut/exe_is_branch	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
+ /processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h8002	080	32'h80020	064	32'h80020	068	32'h8002	06c

#80020070: 8fbf0034 lw 1f, 52(1d) #80020074: 8fbe0030 lw 1e, 48(1d) #80020078: 27bd0038 addiu 1d, 1d, 56

#8002007c: 03e00008 jr 1f

∮ /processor_tb/processor_uut/clk	1'h1									
	32'h80020038	32'h80020	074	32'h80020	078	32'h80020	007c	32'h80020	080	32'h800200
+-/processor_tb/processor_uut/decode_pc	32'h80020034	32'h800	32'h8002	0074	32'h80020	078	32'h80020	07c	32'h80020	080 3
+-/processor_tb/processor_uut/dec_A	32'h00000000	32'h00000	01d					32'h00000	01f	32'h000000
± - /processor_tb/processor_uut/dec_B	32'h00000002	32'h00000	01f	32'h00000	01e	32'h00000	001d	32'h00000	000	
+-/processor_tb/processor_uut/exe_pc	32'h80020030	32'h800	32'h8002	0070	32'h80020	074	32'h80020	078	32'h80020	007c 3
<pre>#</pre> /processor_tb/processor_uut/exe_A	30	30	29						31	0
/processor_tb/processor_uut/exe_op2	28	0	52		48		56		0	
<pre>#</pre> /processor_tb/processor_uut/exe_O	58	30	81		77		85		31	0
/processor_tb/processor_uut/exe_zero	1'h0									
<pre>/processor_tb/processor_uut/exe_neg</pre>	1'h0									
/processor_tb/processor_uut/exe_is_branch	1'h0									
<pre>/processor_tb/processor_uut/exe_is_jump</pre>	1'h0									
<pre>/processor_tb/processor_uut/exe_op2_sel</pre>	1'h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h8002	0070	32'h80020	074	32'h80020	078	32'h80020	007c 3

#80020080: 00000000 NOP

#80020084: 27bdffe8 addiu 1d, 1d, 65512 #80020088: afbe0014 sw 1e, 20(1d) #8002008c: 03a0f021 addu 1e, 1d, 00

<pre>/processor_tb/processor_uut/clk</pre>	1'h1									
├ <mark>-</mark>	32'h80020038	32'h80020	084	32'h80020	088	32'h8002	08c	32'h80020	090	32'h80020
├ <mark>-</mark>	32'h80020034	32'h800	32'h8002	0084	32'h80020	0088	32'h80020	08c	32'h80020	090
├ <mark>-</mark>	32'h00000000	32'h00000	0000	32'h00000	01d					32'h00000
├ <mark>-</mark>	32'h00000002	32'h0000000	00	32'h00000	01d	32'h0000	001e	32'h00000	000	32'h00000
├ <mark>-</mark>	32'h80020030	32'h800	32'h8002	080	32'h80020	084	32'h80020	088	32'h80020	08c
	30	31	0		29					
/processor_tb/processor_uut/exe_op2	28	0			-24		20		0	
/processor_tb/processor_uut/exe_O	58	31	0		5		49		29	
<pre>/processor_tb/processor_uut/exe_zero</pre>	1'h0									
<pre>/processor_tb/processor_uut/exe_neg</pre>	1'h0									
<pre>/processor_tb/processor_uut/exe_is_branch</pre>	1'h0									
<pre>/processor_tb/processor_uut/exe_is_jump</pre>	1'h0									
<pre>/processor_tb/processor_uut/exe_op2_sel</pre>	1'h1									
<pre>/processor_tb/processor_uut/exe_branch_taken</pre>	1'h0									
/processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h8002	0080	32'h80020	084	32'h80020	088	32'h80020	008c

#80020090: afc40018 sw 04, 24(1e) #80020094: afc5001c sw 05, 28(1e) #80020098: afc00000 sw 00, 0(1e) #8002009c: 0800805f j 8002017c

<pre>/processor_tb/processor_uut/dk</pre>	1'h1									
	32'h80020038	32'h80020	094	32'h80020	098	32'h80020	09c	32'h80020	0a0	32'h80020
	32'h80020034	32'h800	32'h8002	0094	32'h80020	098	32'h80020	09c	32'h80020)0a0
	32'h00000000	32'h00000	01e					32'h00000	000	
 /processor_tb/processor_uut/dec_B	32'h00000002	32'h00000	0004	32'h00000	005	32'h00000	0000			
	32'h80020030	32'h800	32'h8002	0090	32'h80020	094	32'h80020	098	32'h80020)09c
	30	29	30						0	
	28	0	24		28		0			
	58	29	54		58		30		0	
/processor_tb/processor_uut/exe_zero	1'h0									
/processor_tb/processor_uut/exe_neg	1'h0									
<pre>/processor_tb/processor_uut/exe_is_branch</pre>	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h8002	0090	32'h80020	094	32'h80020	098	32'h80020)17c

#800200a0: 00000000 NOP

#800200a4: 24020001 addiu 02,00, 1 #800200a8: afc20004 sw 02, 4(1e) #800200ac: 08008055 j 80020154

∮ /processor_tb/processor_uut/clk	1'h1									
+-/-/ /processor_tb/processor_uut/pc	32'h80020038	32'h80020	00a4	32'h80020	00a8	32'h8002	00ac	32'h80020	0b0	32'h80020
<pre>#</pre>	32'h80020034	32'h800	32'h8002	00a4	32'h80020	0a8	32'h80020	0ac	32'h8002)0b0
<pre>#</pre>	32'h00000000	32'h0000000	00			32'h0000	001e	32'h00000	000	
 	32'h00000002	32'h0000000	00	32'h00000	002			32'h00000	000	
<pre>#</pre>	32'h80020030	32'h800	32'h8002	00a0	32'h8002(0a4	32'h80020	00a8	32'h80020)0ac
∓ - <pre>/processor_tb/processor_uut/exe_A</pre>	30	0					30		0	
∓- <pre> /processor_tb/processor_uut/exe_op2</pre>	28	0			1		4		0	
∓- <pre> /processor_tb/processor_uut/exe_O</pre>	58	0			1		34		0	
/processor_tb/processor_uut/exe_zero	1'h0									
/processor_tb/processor_uut/exe_neg	1'h0									
/processor_tb/processor_uut/exe_is_branch	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
+ / /processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h8002(00a0	32'h80020	00a4	32'h80020	00a8	32'h80020	154

#800200b0: 00000000 NOP

800200b4: 8fc20004 lw 02, 4(1e) # 800200b8: 2442ffff addiu 02, 02, 65535 # 800200bc: 00021080 sll 02, 02, 02

<pre>/processor_tb/processor_uut/dk</pre>	1'h1									
-/ /processor_tb/processor_uut/pc	32'h80020038	32'h80020	00b4	32'h80020	00b8	32'h8002	00bc	32'h80020)0c0	32'h80020
-/-/ /processor_tb/processor_uut/decode_pc	32'h80020034	32'h800	32'h8002	00b4	32'h80020	00b8	32'h80020	0bc	32'h80020	00c0
+-/> /processor_tb/processor_uut/dec_A	32'h00000000	32'h0000000	00	32'h00000	01e	32'h0000	0002	32'h00000	0000	32'h00000
+/>/processor_tb/processor_uut/dec_B	32'h00000002	32'h0000000	00	32'h00000	0002					32'h00000
+-/> /processor_tb/processor_uut/exe_pc	32'h80020030	32'h800	32'h8002	0060	32'h80020	00b4	32'h80020	00b8	32'h80020	00bc
+-/>/processor_tb/processor_uut/exe_A	30	0			30		2		0	
/processor_tb/processor_uut/exe_op2	28	0			4		-1		2	
+-/processor_tb/processor_uut/exe_O	58	0			34		1		8	
/processor_tb/processor_uut/exe_zero	1'h0									
/processor_tb/processor_uut/exe_neg	1'h0									
/processor_tb/processor_uut/exe_is_branch	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
+ /processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h8002	00Ь0	32'h80020	0b4	32'h80020	0b8	32'h80020	00bc

#800200c0: 8fc30018 lw 03, 24(1e) #800200c4: 00621021 addu 02, 03, 02 #800200c8: 8c430000 lw 03, 0(02) #800200cc: 8fc20004 lw 02, 4(1e)

<pre>/processor_tb/processor_uut/dk</pre>	1'h1									
+-/processor_tb/processor_uut/pc	32'h80020038	32'h80020	0c4	32'h8002	00c8	32'h8002	Осс	32'h8002	0d0	32'h80020
<pre>#</pre>	32'h80020034	32'h800	32'h8002	00c4	32'h80020	0c8	32'h80020	0сс	32'h80020	00d0
#	32'h00000000	32'h00000	01e	32'h00000	0003	32'h0000	0002	32'h00000	01e	32'h00000
<pre>#</pre>	32'h00000002	32'h00000	0003	32'h00000	0002	32'h0000	0003	32'h00000	002	
-/-/ /processor_tb/processor_uut/exe_pc	32'h80020030	32'h800	32'h8002	00c0	32'h80020	0c4	32'h80020	0с8	32'h8002	Осс
 /processor_tb/processor_uut/exe_A	30	0	30		3		2		30	
/processor_tb/processor_uut/exe_op2	28	2	24		2		0		4	
/processor_tb/processor_uut/exe_O	58	8	54		5		2		34	
/processor_tb/processor_uut/exe_zero	1'h0									
/processor_tb/processor_uut/exe_neg	1'h0									
/processor_tb/processor_uut/exe_is_branch	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
+-> /processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h8002	00c0	32'h80020	0c4	32'h80020	00c8	32'h8002	0сс

#800200d0: 00021080 sll 02, 02, 02 #800200d4: 8fc40018 lw 04, 24(1e) #800200d8: 00821021 addu 02, 04, 02 #800200dc: 8c420000 lw 02, 0(02)

# 000200dc. 00420000 1W 02,	0(02)									
/processor_tb/processor_uut/dk	1'h1									
≖ - <pre>/processor_tb/processor_uut/pc</pre>	32'h80020038	32'h80020	0d4	32'h80020	0d8	32'h8002	00dc	32'h80020	0e0	32'h800200
<pre>#</pre>	32'h80020034	32'h800	32'h80020	00d4	32'h80020	0d8	32'h80020	00dc	32'h80020	0e0 3
	32'h00000000	32'h00000	0000	32'h00000	01e	32'h0000	0004	32'h00000	002	
I → /processor_tb/processor_uut/dec_B	32'h00000002	32'h0000000)2	32'h00000	0004	32'h0000	0002			32'h000000
∓ - <pre> /processor_tb/processor_uut/exe_pc</pre>	32'h80020030	32'h800	32'h8002	00d0	32'h80020	0d4	32'h80020	00d8	32'h80020	0 dc 3
	30	30	0		30		4		2	
/processor_tb/processor_uut/exe_op2	28	4	2		24		2		0	3
+	58	34	8		54		6		2	1
/processor_tb/processor_uut/exe_zero	1'h0									
<pre>/processor_tb/processor_uut/exe_neg</pre>	1'h0									
<pre>/processor_tb/processor_uut/exe_is_branch</pre>	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
<pre>/processor_tb/processor_uut/exe_branch_taken</pre>	1'h0									
+	32'h80020030	32'h800	32'h80020	0d0	32'h80020	0d4	32'h80020	00d8	32'h80020	0 dc 3

#800200e0: 0043102a slt 02, 02, 03 #800200e4: 10400019 beq 02, 00, 0019

#800200e8: 00000000 NOP

#800200ec: 8fc20004 lw 02, 4(1e)

# 800200ec. 81c20004 1W 02,	4(16)									
<pre>/processor_tb/processor_uut/dk</pre>	1'h1									
	32'h80020038	32'h80020	0e4	32'h80020	00e8	32'h8002	00ec	32'h80020	0f0	32'h80020
/processor_tb/processor_uut/decode_pc	32'h80020034	32'h800	32'h8002	00e4	32'h80020	0e8	32'h80020	0ec	32'h80020	00f0
+	32'h00000000	32'h0000000	2			32'h00000	0000	32'h00000	01e	32'h00000
+	32'h00000002	32'h00000	0003	32'h00000	0000			32'h00000	002	
+	32'h80020030	32'h800	32'h8002	00e0	32'h80020	0e4	32'h80020	0e8	32'h80020	J0ec
+	30	2					0		30	
+	28	0	3		0				4	
+	58	2	1		2		0		34	
/processor_tb/processor_uut/exe_zero	1'h0									
/processor_tb/processor_uut/exe_neg	1'h0									
/processor_tb/processor_uut/exe_is_branch	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h8002	00e0	32'h80020	148	32'h80020	0e8	32'h8002	00ec

#800200f0: 2442ffff addiu 02, 02, 65535 #800200f4: 00021080 sll 02, 02, 02 #800200f8: 8fc30018 lw 03, 24(1e) #800200fc: 00621021 addu 02,03,02

<pre>/processor_tb/processor_uut/dk</pre>	1'h1									
/processor_tb/processor_uut/pc	32'h80020038	32'h80020	0f4	32'h80020	0f8	32'h8002	00fc	32'h80020	100	32'h8002
/processor_tb/processor_uut/decode_pc	32'h80020034	32'h800	32'h8002	00f4	32'h80020	0f8	32'h80020	0fc	32'h80020	100
/processor_tb/processor_uut/dec_A	32'h00000000	32'h00000	002	32'h00000	0000	32'h00000	001e	32'h00000	003	32'h0000
/processor_tb/processor_uut/dec_B	32'h00000002	32'h0000000	2			32'h00000	0003	32'h00000	002	
≖ - ∜ /processor_tb/processor_uut/exe_pc	32'h80020030	32'h800	32'h8002	00f0	32'h80020	0f4	32'h80020	0f8	32'h80020	00fc
≖ - ∜ /processor_tb/processor_uut/exe_A	30	30	2		0		30		3	
≖ - ∜ /processor_tb/processor_uut/exe_op2	28	4	-1		2		24		2	
/processor_tb/processor_uut/exe_O	58	34	1		8		54		5	
/processor_tb/processor_uut/exe_zero	1'h0									
/processor_tb/processor_uut/exe_neg	1'h0									
<pre>/processor_tb/processor_uut/exe_is_branch</pre>	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
<pre>/processor_tb/processor_uut/exe_branch_taken</pre>	1'h0									
+	32'h80020030	32'h800	32'h8002	00f0	32'h80020	0f4	32'h80020	0f8	32'h80020	00fc

80020100: 8c420000 lw 02, 0(02) # 80020104: afc20008 sw 02, 8(1e) # 80020108: 8fc20004 lw 02, 4(1e) # 8002010c: 2442ffff addiu 02, 02, 65535

/processor_tb/processor_uut/clk	1'h1									
+	32'h80020038	32'h80020	104	32'h80020	108	32'h80020	10c	32'h80020	110	32'h8002
+	32'h80020034	32'h800	32'h80020	104	32'h80020	108	32'h80020	10c	32'h80020	110
<u>-</u> ∕> /processor_tb/processor_uut/dec_A	32'h00000000	32'h00000	002	32'h00000	01e			32'h00000	002	32'h0000
+	32'h00000002	32'h0000000	2							
+	32'h80020030	32'h800	32'h80020	100	32'h80020	104	32'h80020	108	32'h80020)10c
I− ♦ /processor_tb/processor_uut/exe_A	30	3	2		30				2	
	28	2	0		8		4		-1	
-/	58	5	2		38		34		1	
/processor_tb/processor_uut/exe_zero	1'h0									
/processor_tb/processor_uut/exe_neg	1'h0									
<pre>/processor_tb/processor_uut/exe_is_branch</pre>	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
/processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h80020	100	32'h80020	104	32'h80020	108	32'h80020	10c

#80020110: 00021080 sll 02,02,02 #80020114: 8fc30018 lw 03, 24(1e) #80020118: 00621021 addu 02,03,02 #8002011c: 8fc30004 lw 03, 4(1e)

✓ /processor_tb/processor_uut/dk	1'h1									
//processor_tb/processor_uut/pc	32'h80020038	32'h80020)114	32'h80020	118	32'h80020)11c	32'h80020	120	32'h800
+/>/processor_tb/processor_uut/decode_pc	32'h80020034	32'h800	32'h8002	114	32'h80020	118	32'h80020	11c	32'h80020	120
+	32'h00000000	32'h00000	0000	32'h00000	01e	32'h00000	0003	32'h00000	01e	32'h000
+	32'h00000002	32'h0000000)2	32'h00000	003	32'h00000	0002	32'h00000	003	
+	32'h80020030	32'h800	32'h8002	110	32'h80020	114	32'h80020	118	32'h80020	11c
/processor_tb/processor_uut/exe_A	30	2	0		30		3		30	
+ - √ /processor_tb/processor_uut/exe_op2	28	-1	2		24		2		4	
≖ - √ /processor_tb/processor_uut/exe_O	58	1	8		54		5		34	
<pre>/processor_tb/processor_uut/exe_zero</pre>	1'h0									
<pre>/processor_tb/processor_uut/exe_neg</pre>	1'h0									
<pre>/processor_tb/processor_uut/exe_is_branch</pre>	1'h0									
<pre>/processor_tb/processor_uut/exe_is_jump</pre>	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
+	32'h80020030	32'h800	32'h8002	110	32'h80020	114	32'h80020	118	32'h80020	11c

#80020120: 00031880 sll 03, 03, 02

#80020124: 8fc40018 lw 04, 24(1e) #80020128: 00831821 addu 03, 04, 03 #8002012c: 8c630000 lw 03, 0(03)

/processor_tb/processor_uut/clk	1'h1									
+	32'h80020038	32'h8002	0124	32'h80020	128	32'h80020)12c	32'h80020	130	32'h80020
+	32'h80020034	32'h800	32'h80020	124	32'h80020	128	32'h80020	12c	32'h80020	130
+	32'h00000000	32'h0000	0000	32'h00000	01e	32'h00000	0004	32'h00000	003	32'h00000
+	32'h00000002	32'h000000	03	32'h00000	004	32'h00000	0003			
+-/processor_tb/processor_uut/exe_pc	32'h80020030	32'h800	32'h80020	120	32'h80020	124	32'h80020	128	32'h80020)12c
+-/processor_tb/processor_uut/exe_A	30	30	0		30		4		3	
+-/processor_tb/processor_uut/exe_op2	28	4	3		24		3		0	
<pre>#</pre>	58	34	12		54		7		3	
/processor_tb/processor_uut/exe_zero	1'h0									
/processor_tb/processor_uut/exe_neg	1'h0									
/processor_tb/processor_uut/exe_is_branch	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
+	32'h80020030	32'h800	32'h80020	120	32'h80020	124	32'h80020	128	32'h80020)12c

#80020130: ac430000 sw 03, 0(02) #80020134: 8fc20004 lw 02, 4(1e) #80020138: 00021080 sll 02, 02, 02 #8002013c: 8fc30018 lw 03, 24(1e)

# 6002013C. 61C30016 1W 03,	24(1e)									
<pre>/processor_tb/processor_uut/clk</pre>	1'h1									
+-/processor_tb/processor_uut/pc	32'h80020038	32'h80020	134	32'h80020	138	32'h80020)13c	32'h80020	140	32'h800201
<pre>#-</pre> /processor_tb/processor_uut/decode_pc	32'h80020034	32'h800	32'h80020	134	32'h80020	138	32'h80020	13c	32'h80020	140
<pre>#-</pre> /processor_tb/processor_uut/dec_A	32'h00000000	32'h00000	002	32'h00000	01e	32'h00000	0000	32'h00000	01e	32'h000000
I → /processor_tb/processor_uut/dec_B	32'h00000002	32'h0000000	3	32'h00000	002			32'h00000	003	32'h000000
<pre>#</pre>	32'h80020030	32'h800	32'h80020	130	32'h80020	134	32'h80020	138	32'h80020)13c
	30	3	2		30		0		30	<u> </u>
	28	0			4		2		24	
	58	3	2		34		8		54	
<pre>/processor_tb/processor_uut/exe_zero</pre>	1'h0									
<pre>/processor_tb/processor_uut/exe_neg</pre>	1'h0									
<pre>/processor_tb/processor_uut/exe_is_branch</pre>	1'h0									
<pre>/processor_tb/processor_uut/exe_is_jump</pre>	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
+ / /processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h80020	130	32'h80020	134	32'h80020	138	32'h80020)13c

#80020140: 00621021 addu 02, 03, 02 #80020144: 8fc30008 lw 03, 8(1e) #80020148: ac430000 sw 03, 0(02) #8002014c: 8fc20004 lw 02, 4(1e) #80020150: 24420001 addiu 02, 02, 1

11 00020150: 21120001 udd	14 02, 02,											
<pre>/processor_tb/processor_uut/dk</pre>	1'h1											
+ /processor_tb/processor_uut/pc	32'h80020038	32'h80020	144	32'h80020	148	32'h8002)14c	32'h80020	150	32'h80020	154	32'h800201
<pre>#-</pre> /processor_tb/processor_uut/decode_pc	32'h80020034	32'h800	32'h80020	144	32'h80020	148	32'h80020	14c	32'h80020	150	32'h80020	154 3
//processor_tb/processor_uut/dec_A	32'h00000000	32'h00000	0003	32'h00000	01e	32'h0000	0002	32'h00000	01e	32'h00000	0002	32'h000000
I → /processor_tb/processor_uut/dec_B	32'h00000002	32'h00000	002	32'h00000	0003			32'h00000	002			
I → /processor_tb/processor_uut/exe_pc	32'h80020030	32'h800	32'h80020	140	32'h80020	144	32'h80020	148	32'h80020)14c	32'h80020	150 3
I → /processor_tb/processor_uut/exe_A	30	30	3		30		2		30		2	3
I → /processor_tb/processor_uut/exe_op2	28	24	2		8		0		4		1	- 2
<pre>#-</pre> /processor_tb/processor_uut/exe_O	58	54	5		38		2		34		3	3
/processor_tb/processor_uut/exe_zero	1'h0											
<pre>/processor_tb/processor_uut/exe_neg</pre>	1'h0											
/processor_tb/processor_uut/exe_is_branch	1'h0											
/processor_tb/processor_uut/exe_is_jump	1'h0											
/processor_tb/processor_uut/exe_op2_sel	1'h1											
<pre>/processor_tb/processor_uut/exe_branch_taken</pre>	1'h0											
+-> /processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h80020	140	32'h80020	144	32'h80020	148	32'h80020)14c	32'h80020	150 3
Y	<u> </u>											

#80020150: 24420001 addiu 02, 02, 1 #80020154: afc20004 sw 02, 4(1e) #80020158: 8fc3001c lw 03, 28(1e) #8002015c: 8fc20000 lw 02, 0(1e)

# 0002013C. 01C20000 1W 02,	O(IC)									
<pre>// /processor_tb/processor_uut/dk</pre>	1'h1									
+ /processor_tb/processor_uut/pc	32'h80020038	32'h80020	154	32'h80020	158	32'h80020)15c	32'h80020	160	32'h80020
<pre>#-</pre> /processor_tb/processor_uut/decode_pc	32'h80020034	32'h800	32'h80020	154	32'h80020	158	32'h80020	15c	32'h80020	160
I → /processor_tb/processor_uut/dec_A	32'h00000000	32'h00000	002	32'h00000	01e					32'h00000
<pre>#</pre> /processor_tb/processor_uut/dec_B	32'h00000002	32'h0000000	2			32'h00000	0003	32'h00000	002	
<pre>#</pre>	32'h80020030	32'h800	32'h80020	150	32'h80020	154	32'h80020	158	32'h80020)15c
<pre>#-</pre> /processor_tb/processor_uut/exe_A	30	30	2		30					
/processor_tb/processor_uut/exe_op2	28	4	1		4		28		0	
+	58	34	3		34		58		30	
/processor_tb/processor_uut/exe_zero	1'h0									
/processor_tb/processor_uut/exe_neg	1'h0									
<pre>/processor_tb/processor_uut/exe_is_branch</pre>	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
<pre>/processor_tb/processor_uut/exe_branch_taken</pre>	1'h0									
+	32'h80020030	32'h800	32'h80020	150	32'h80020	154	32'h80020	158	32'h80020)15c

#80020164: 8fc20004 lw 02, 4(1e) #80020168: 0043102a slt 02, 02, 03 #8002016c: 1440ffd1 bne 02, 00, ffd1

/processor_tb/processor_uut/clk	1'h1									
	32'h80020038	32'h80020	164	32'h80020	168	32'h80020)16c	32'h80020	170	32'h80020
/processor_tb/processor_uut/decode_pc	32'h80020034	32'h800	32'h8002	0164	32'h80020	168	32'h80020)16c	32'h8002	170
/processor_tb/processor_uut/dec_A	32'h00000000	32'h00000	0003	32'h00000	01e	32'h00000	0002			32'h00000
≖ - <pre>/processor_tb/processor_uut/dec_B</pre>	32'h00000002	32'h0000000)2			32'h00000	0003	32'h00000	000	
/processor_tb/processor_uut/exe_pc	32'h80020030	32'h800	32'h8002	0160	32'h80020	164	32'h80020	168	32'h8002)16c
/processor_tb/processor_uut/exe_A	30	30	3		30		2			
/processor_tb/processor_uut/exe_op2	28	0	2		4		3		0	
	58	30	1		34		1		2	
/processor_tb/processor_uut/exe_zero	1'h0									
/processor_tb/processor_uut/exe_neg	1'h0									
/processor_tb/processor_uut/exe_is_branch	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
+ /processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h8002	0160	32'h80020	164	32'h80020	168	32'h8002)16c

#80020170: 00000000 NOP

#80020174: 8fc20000 lw 02, 0(1e) #80020178: 24420001 addiu 02, 02, 1 #8002017c: afc20000 sw 02, 0(1e)

/processor_tb/processor_uut/clk	1'h1									
≖ - <pre> /processor_tb/processor_uut/pc</pre>	32'h80020038	32'h80020	174	32'h80020	178	32'h80020	17c	32'h80020	180	32'h80020
<pre></pre>	32'h80020034	32'h800	32'h80020	174	32'h80020	178	32'h80020	17c	32'h80020	180
/processor_tb/processor_uut/dec_A	32'h00000000	32'h00000	0000	32'h00000	01e	32'h00000	0002	32'h00000	01e	
≖ - <pre> // /processor_tb/processor_uut/dec_B</pre>	32'h00000002	32'h0000000	0	32'h00000	002					32'h00000
<pre></pre>	32'h80020030	32'h800	32'h8002(170	32'h80020	174	32'h80020	178	32'h80020)17c
	30	2	0		30		2		30	
/> /processor_tb/processor_uut/exe_op2	28	0					1		0	
<pre># /processor_tb/processor_uut/exe_O</pre>	58	2	0		30		3		30	
/processor_tb/processor_uut/exe_zero	1'h0									
/processor_tb/processor_uut/exe_neg	1'h0									
/processor_tb/processor_uut/exe_is_branch	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1'h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
	32'h80020030	32'h800	32'h80020	170	32'h80020	174	32'h80020	178	32'h80020)17c

#80020180: 8fc30000 lw 03, 0(1e) #80020184: 8fc2001c lw 02, 28(1e) #80020188: 0062102a slt 02, 03, 02 #8002018c: 1440ffc5 bne 02, 00, ffc5

-	30,									
/processor_tb/processor_uut/dk	1'h1									
/processor_tb/processor_uut/pc	32'h80020038	32'h80020	184	32'h80020	188	32'h8002	018c	32'h80020	190	32'h80020
//processor_tb/processor_uut/decode_pc	32'h80020034	32'h800	32'h8002	184	32'h80020	188	32'h80020)18c	32'h80020	190
/processor_tb/processor_uut/dec_A	32'h00000000	32'h0000000	le			32'h0000	0003	32'h00000	002	32'h00000
+	32'h00000002	32'h00000	0003	32'h00000	0002			32'h00000	0000	
+	32'h80020030	32'h800	32'h8002	180	32'h80020	184	32'h80020	188	32'h80020)18c
∓ - <pre> // /processor_tb/processor_uut/exe_A</pre>	30	30					3		2	
I I processor_tb/processor_uut/exe_op2	28	0			28		2		0	
I → /processor_tb/processor_uut/exe_O	58	30			58		0		2	
/processor_tb/processor_uut/exe_zero	1'h0									
/processor_tb/processor_uut/exe_neg	1'h0									
/processor_tb/processor_uut/exe_is_branch	1'h0									
<pre>/processor_tb/processor_uut/exe_is_jump</pre>	1'h0									
<pre>/processor_tb/processor_uut/exe_op2_sel</pre>	1'h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
→ /processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h8002	180	32'h80020	184	32'h80020	188	32'h80020)18c

#80020190: 00000000 NOP

#80020194: 03c0e821 addu 1d, 1e, 00 #80020198: 8fbe0014 lw 1e, 20(1d) #8002019c: 27bd0018 addiu 1d, 1d, 24

•										
/processor_tb/processor_uut/dk	1'h1									
+	32'h80020038	32'h80020	194	32'h80020	198	32'h8002	19c	32'h80020	1a0	32'h80020
/processor_tb/processor_uut/decode_pc	32'h80020034	32'h800	32'h80020	194	32'h80020	198	32'h80020	19c	32'h80020	1a0
/> /processor_tb/processor_uut/dec_A	32'h00000000	32'h00000	000	32'h00000	01e	32'h00000	001d			32'h00000
<pre># /processor_tb/processor_uut/dec_B</pre>	32'h00000002	32'h0000000	0			32'h00000	001e	32'h00000	01d	32'h00000
<pre># /processor_tb/processor_uut/exe_pc</pre>	32'h80020030	32'h800	32'h80020	190	32'h80020	194	32'h80020	198	32'h80020	19c
	30	2	0		30		29			
	28	0					20		24	
	58	2	0		30		49		53	
/processor_tb/processor_uut/exe_zero	1'h0									
/processor_tb/processor_uut/exe_neg	1'h0									
/processor_tb/processor_uut/exe_is_branch	1'h0									
/processor_tb/processor_uut/exe_is_jump	1'h0									
/processor_tb/processor_uut/exe_op2_sel	1h1									
/processor_tb/processor_uut/exe_branch_taken	1'h0									
→ /processor_tb/processor_uut/exe_next_pc	32'h80020030	32'h800	32'h80020	190	32'h80020	194	32'h80020	198	32'h80020	19c