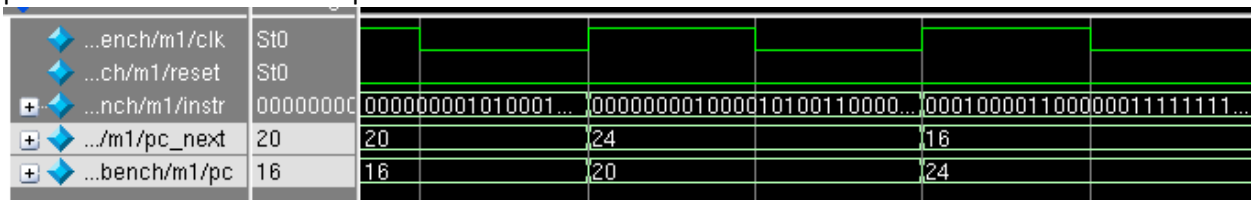
Modeling the Single-Cycle MIPS Processor in Verilog
By Ashtyn Moehlenhoff & Torben Rasmussen
ECE472 – Fall 2010

Part1 – Building and Simulating the Processor Model

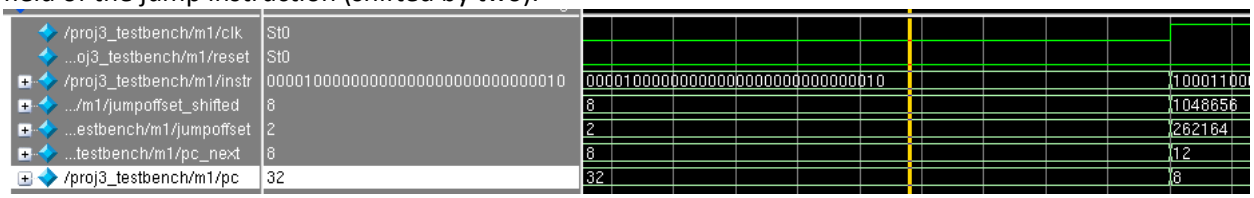
Building the processor is very straight-forward. We created a test bench in order to simulate the processor with a 100ns clock period.



You can see in the above figure that the pc_next = pc+4 for each of these instructions. Also note that these instructions are not jump/branch instructions.

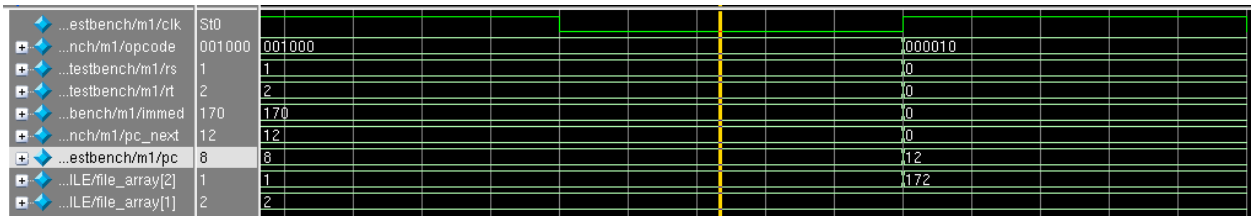
Part2 – Extending the Processor Design

Jump: For jump we created a Jump Mux to choose from the output of the PCMUX and or the immediate field of the jump instruction (shifted by two).



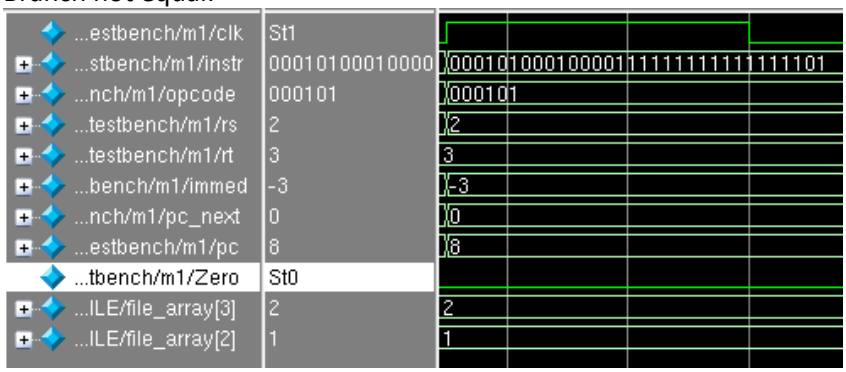
This instruction represents PC = Jump Address = IMM << 2. In our case, pc_next = 2 << 2 = 8. The PC in the cycle following the jump instruction is 8.

Add Immediate:



The opcode 0b001000 is for the add immediate instruction: R[rt] = R[rs] + IMMEDIATE. So in our case, R2 = R1 + IMMEDIATE which equals 2 + 170 = 172. Register 2 is updated in the next cycle to be 172.

Branch not equal:



Our instruction was bne r2, r3, -3. This subtracts r2 and r3, and when the zero flag is NOT set (they are not equal), pc = pc_next + immediate. In our example pc is set to (8 + 4) – (3<<2) = 0 because r3 != r2.

Part3 – Simulating the modified processor

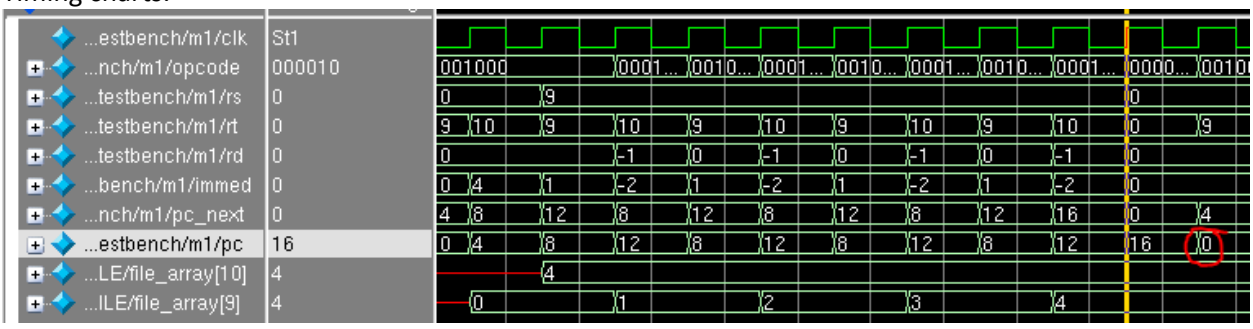
Assembly code (from MARS):

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20090000	addi $9,$0,0x0000	2: addi $t1,$zero,0
	0x00400004	0x200a0004	addi $10,$0,0x0004	3: addi $t2,$zero,4
	0x00400008	0x21290001	addi $9,$9,0x0001	5: addi $t1,$t1,1
	0x0040000c	0x152afffe	bne $9,$10,0xfffe	6: bne $t1, $t2,LOOP
	0x00400010	0x08100000	j 0x00400000	7: j START

Edited Rom32.v:

```
5'd0 : data_out = { 32'h20090000 };
5'd1 : data_out = { 32'h200a0004 };
5'd2 : data_out = { 32'h21290001 };
5'd3 : data_out = { 32'h152afffe };
5'd4 : data_out = { 32'h08000000 };
```

Timing charts:



Above you can see our program being run. First, registers 9 and 10 are initialized. Next, register 9 is incremented and register 9 and 10 are compared. If these registers are equal, it loops back to increment register 9. Finally, when register 9 and 10 are equal, the program jumps to the start (circled in red in the above diagram).

```
//------------------------------------------------------------------------
// Title         : MIPS Single-Cycle Control Unit
// Project       : ECE 313 - Computer Organization
//------------------------------------------------------------------------
// File          : control_single.v
// Author        : John Nestor  <nestorj@lafayette.edu>
// Organization  : Lafayette College
//
// Created       : October 2002
// Last modified : 7 January 2005
//------------------------------------------------------------------------
// Description :
//   Control unit for the MIPS "Single Cycle" processor implementation described
//   Section 5.4 of "Computer Organization and Design, 3rd ed."
//   by David Patterson & John Hennessey, Morgan Kaufmann, 2004 (COD3e).
//
//   It implements the function specified in Figure 5.18 on p. 308 of COD3e.
//
//------------------------------------------------------------------------

module control_single(opcode, RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWr
ite, Branch, ALUOp, Jump, Branch_Sel);
    input [5:0] opcode;
    output RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, Jump,
Branch_Sel;
    output [1:0] ALUOp;
    reg    RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, Jump,
Branch_Sel;
    reg    [1:0] ALUOp;

    parameter R_FORMAT = 6'd0;
    parameter LW = 6'd35;
    parameter SW = 6'd43;
    parameter BEQ = 6'd4;
    parameter BNE = 6'd5;
    parameter J = 6'b000010;
    parameter ADDI = 6'b001000;

    always @(opcode)
    begin
        case (opcode)
          R_FORMAT :
          begin
              RegDst=1'b1; ALUSrc=1'b0; MemtoReg=1'b0; RegWrite=1'b1; MemRead=1'
b0;
              MemWrite=1'b0; Branch=1'b0; ALUOp = 2'b10; Jump = 1'b0; Branch_Sel
=1'bx;
          end
          LW :
          begin
              RegDst=1'b0; ALUSrc=1'b1; MemtoReg=1'b1; RegWrite=1'b1; MemRead=1'
b1;
              MemWrite=1'b0; Branch=1'b0; ALUOp = 2'b00; Jump = 1'b0; Branch_Sel
=1'bx;
          end
          SW :
          begin
              RegDst=1'bx; ALUSrc=1'b1; MemtoReg=1'bx; RegWrite=1'b0; MemRead=1'
b0;
              MemWrite=1'b1; Branch=1'b0; ALUOp = 2'b00; Jump = 1'b0; Branch_Sel
=1'bx;
          end
          BEQ :
          begin
              RegDst=1'bx; ALUSrc=1'b0; MemtoReg=1'bx; RegWrite=1'b0; MemRead=1'
b0;
              MemWrite=1'b0; Branch=1'b1; ALUOp = 2'b01; Jump = 1'b0; Branch_Sel
=1'b0;
          end
```



```
          BNE :
          begin
              RegDst=1'bx; ALUSrc=1'b0; MemtoReg=1'bx; RegWrite=1'b0; MemRead=1'
b0;
              MemWrite=1'b0; Branch=1'b1; ALUOp = 2'b01; Jump = 1'b0; Branch_Sel
=1'b1;
          end
          J :
          begin
              RegDst=1'bx; ALUSrc=1'b0; MemtoReg=1'bx; RegWrite=1'b0; MemRead=1'
b0;
              MemWrite=1'b0; Branch=0'b0; ALUOp = 2'bxx; Jump = 1'b1; Branch_Sel
=1'bx;

          end
          ADDI :
          begin
              RegDst=1'b0; ALUSrc=1'b1; MemtoReg=1'b0; RegWrite=1'b1; MemRead=1'
b0;
              MemWrite=1'b0; Branch=1'b0; ALUOp = 2'b00; Jump = 1'b0; Branch_Sel
=1'bx;
          end
          default
          begin
              $display("control_single unimplemented opcode %d", opcode);
              RegDst=1'bx; ALUSrc=1'bx; MemtoReg=1'bx; RegWrite=1'bx; MemRead=1'
bx;
              MemWrite=1'bx; Branch=1'bx; ALUOp = 2'bxx; Jump = 1'b0; Branch_Sel
=1'bx;
          end

        endcase
    end
endmodule
```

```
//------------------------------------------------------------------------
// Title         : MIPS Single-Cycle Processor
// Project       : ECE 472 - Proj 3
//------------------------------------------------------------------------
// File          : mips_single.v
// Author        : John Nestor  <nestorj@lafayette.edu>
// Students      : Ashtyn Moehlenhoff and Torben Rasmussen
// Organization  : Lafayette College/Oregon State University
//
// Created       : October 2002
// Last modified : 29 October 2010
//------------------------------------------------------------------------
// Description :
//   "Single Cycle" implementation of the MIPS processor subset described in
//   Section 5.4 of "Computer Organization and Design, 3rd ed."
//   by David Patterson & John Hennessey, Morgan Kaufmann, 2004 (COD3e).
//
//   It implements the equivalent of Figure 5.19 on page 309 of COD3e
//
//------------------------------------------------------------------------

module mips_single(clk, reset);
    input clk, reset;

    // instruction bus
    wire [31:0] instr;

    // break out important fields from instruction
    wire [5:0] opcode, funct;
    wire [4:0] rs, rt, rd, shamt;
    wire [15:0] immed;
    wire [31:0] extend_immed, b_offset, jumpoffset_shifted;
    wire [25:0] jumpoffset;

    assign opcode = instr[31:26];
    assign rs = instr[25:21];
    assign rt = instr[20:16];
    assign rd = instr[15:11];
    assign shamt = instr[10:6];
    assign funct = instr[5:0];
    assign immed = instr[15:0];
    assign jumpoffset = instr[25:0];

    // sign-extender
    assign extend_immed = { {16{immed[15]}}, immed };

    // branch offset shifter
    assign b_offset = extend_immed << 2;

    // datapath signals
    wire [4:0] rfile_wn;
    wire [31:0] rfile_rd1, rfile_rd2, rfile_wd, alu_b, alu_out, b_tgt, pc_next,
                pc, pc_incr, br_add_out, dmem_rdata, jump_in;

    // control signals

    wire RegWrite, Branch, PCSrc, PCSrc2, PCSrc_out, RegDst, MemtoReg, MemRead,
MemWrite, ALUSrc, Zero, Jump;
    wire [1:0] ALUOp;
    wire [2:0] Operation;

    // module instantiations

    reg32               PC(clk, reset, pc_next, pc);

    add32               PCADD(pc, 32'd4, pc_incr);

    add32               BRADD(pc_incr, b_offset, b_tgt);
```



```
    reg_file    RFILE(clk, RegWrite, rs, rt, rfile_wn, rfile_rd1, rfile_rd2, rfi
le_wd);

    alu                 ALU(Operation, rfile_rd1, alu_b, alu_out, Zero);

    rom32               IMEM(pc, instr);

    mem32               DMEM(clk, MemRead, MemWrite, alu_out, rfile_rd2, dmem_rd
ata);

    and                 BR_AND(PCSrc, Branch, Zero);

    and    BR_AND2(PCSrc2, Branch, ~Zero);

    mux2 #(5)   RFMUX(RegDst, rt, rd, rfile_wn);

    mux2 #(32)  PCMUX(PCSrc_out, pc_incr, b_tgt, jump_in);

    mux2 #(32) ALUMUX(ALUSrc, rfile_rd2, extend_immed, alu_b);

    mux2 #(32)  WRMUX(MemtoReg, alu_out, dmem_rdata, rfile_wd);

    mux2 #(1) BRMUX(Branch_Sel, PCSrc, PCSrc2, PCSrc_out);

    //****************************** LAB3 Stuff ***************************
*******
    //JUMP instruction:
    assign jumpoffset_shifted = jumpoffset << 2; //get the jumpoffset * 4 to hav
e the correct jump value
    mux2 #(32) JMPMUX(Jump, jump_in, jumpoffset_shifted, pc_next);


    //****************************** END LAB3 Stuff ************************
*******

    control_single CTL(.opcode(opcode), .RegDst(RegDst), .ALUSrc(ALUSrc), .Memto
Reg(MemtoReg),
                       .RegWrite(RegWrite), .MemRead(MemRead), .MemWrite(MemWrit
e), .Branch(Branch),
                       .ALUOp(ALUOp), .Jump(Jump), .Branch_Sel(Branch_Sel));

    alu_ctl     ALUCTL(ALUOp, funct, Operation);
endmodule
```

```
//-----------------------------------------------------------------------------
// Title         : Read-Only Memory (Instruction ROM)
// Project       : ECE 313 - Computer Organization
//-----------------------------------------------------------------------------
// File          : rom32.v
// Author        : John Nestor  <nestorj@lafayette.edu>
// Organization  : Lafayette College
//
// Created       : October 2002
// Last modified : 7 January 2005
//-----------------------------------------------------------------------------
// Description :
//   Behavioral model of a read-only memory used in the implementations of the M
IPS
//   processor subset described in Ch. 5-6 of "Computer Organization and Design,
 3rd ed."
//   by David Patterson & John Hennessey, Morgan Kaufmann, 2004 (COD3e).
//
//   Note the use of the Verilog concatenation operator to specify different
//   instruction fields in each memory element.
//
//-----------------------------------------------------------------------------

module rom32(address, data_out);
  input  [31:0] address;
  output [31:0] data_out;
  reg    [31:0] data_out;

  parameter BASE_ADDRESS = 25'd0; // address that applies to this memory

  wire [4:0] mem_offset;
  wire address_select;

  assign mem_offset = address[6:2];  // drop 2 LSBs to get word offset

  assign address_select = (address[31:7] == BASE_ADDRESS);  // address decoding

  always @(address_select or mem_offset)
  begin
    if ((address % 4) != 0) $display($time, " rom32 error: unaligned address %d", address)
;
    if (address_select == 1)
    begin
      case (mem_offset)
        //5'd0 : data_out = { 6'd35, 5'd0, 5'd1, 16'd4 };              // lw $1,
 4($0)    r1=1
       // 5'd1 : data_out = { 6'd35, 5'd0, 5'd2, 16'd8 };              // lw $2,
 8($0)    r2=2
        5'd0 : data_out = { 32'h20090000 };
        5'd1 : data_out = { 32'h200a0004 };
        5'd2 : data_out = { 32'h21290001 };
        5'd3 : data_out = { 32'h152afffe };
        5'd4 : data_out = { 32'h08000000 };
          //5'd0 : data_out = { 6'd35, 5'd0, 5'd1, 16'd4 };              // lw $
1, 4($0)    r1=1
          //5'd1 : data_out = { 6'd35, 5'd0, 5'd2, 16'd8 };              // lw $
2, 8($0)    r2=2
          //5'd2  : data_out = { 6'b001000, 5'd2, 5'd2, 16'd3 };         // addi
 $2, $2, 4  r2+=3
          //5'd3  : data_out = { 6'b001000, 5'd1, 5'd1, 16'd1 };         // addi
 $1, $1, 1  r1+=1
          //5'd4  : data_out = { 6'd5, 5'd1, 5'd2, -16'd2};              // bne
$1, $2, -2 (if $1 != $2 goto: 5'd3)
          //5'd5  : data_out = { 6'd2, 26'd0 };                          // j 0;
 restart loop
          default data_out = 32'hxxxx;
      endcase
      $display($time, " reading data: rom32[%h] => %h", address, data_out);
    end
```



```
  end
endmodule
```