

E85: Digital Design and Computer Architecture

Problem Set 9

- 1) Suppose the RegW signal in a single-cycle ARM processor has a *stuck-at-1* fault (e.g. the signal is always 1). Which instructions would malfunction, and why?
- 2) Modify the single-cycle ARM processor to implement the LSR instruction with a *shamt5* immediate shift amount. Mark up copies of the controller, main decoder, ALU decoder, and datapath (attached) to handle the new instruction as simply as possible. Name any control signals you need to add.
- 3) Modify the single-cycle ARM processor to implement the TST instruction. Mark up the Verilog (attached) to implement your changes as simply as possible.
- 4) Alyssa P. Hacker is a crack circuit designer. She offers to speed up one of the functional units in Table 7.5 by 50% to improve the overall performance of the single-cycle processor. Which unit should she optimize, and by what percentage will the execution time improve?
- 5) Impact on Society: Pick a phone or tablet that you or a friend uses that has one or more ARM processors inside. Look up what processors are inside. What company designed the microarchitecture? What company designed the chip? What is the maximum clock frequency? Does the processor implement the 32-bit (ARMv7) or 64-bit (ARMv8) instruction set?

How long did you spend on this problem set? This will not count toward your grade but will help calibrate the workload.

Problem 2: Mark up datapath and controller to implement LSR

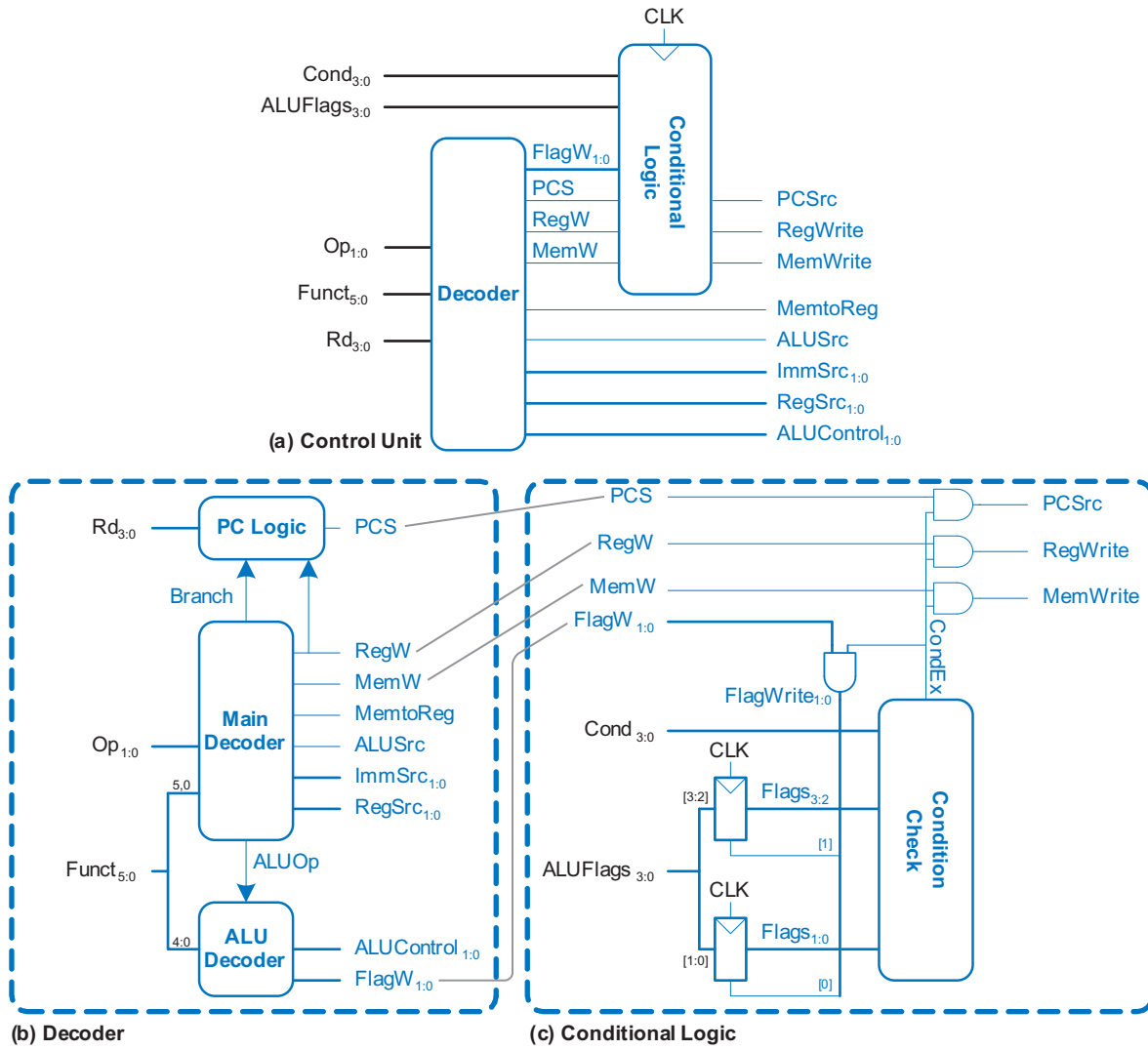


Figure 7.14 Single-cycle control unit

Table 7.2 Main Decoder truth table

| Op | Funct ₅ | Funct ₀ | Type | Branch | MemtoReg | MemW | ALUSrc | ImmSrc | RegW | RegSrc | ALUOp |
|----|--------------------|--------------------|--------|--------|----------|------|--------|--------|------|--------|-------|
| 00 | 0 | X | DP Reg | 0 | 0 | 0 | 0 | XX | 1 | 00 | 1 |
| 00 | 1 | X | DP Imm | 0 | 0 | 0 | 1 | 00 | 1 | X0 | 1 |
| 01 | X | 0 | STR | 0 | X | 1 | 1 | 01 | 0 | 10 | 0 |
| 01 | X | 1 | LDR | 0 | 1 | 0 | 1 | 01 | 1 | X0 | 0 |
| 10 | X | X | B | 1 | 0 | 0 | 1 | 10 | 0 | X1 | 0 |

Table 7.3 ALU Decoder truth table

| ALUOp | Funct _{4:1} (cmd) | Funct ₀ (S) | Type | ALUControl _{1:0} | FlagW _{1:0} |
|-------|----------------------------|------------------------|--------|---------------------------|----------------------|
| 0 | X | X | Not DP | 00 (Add) | 00 |
| 1 | 0100 | 0 | ADD | 00 (Add) | 00 |
| | | 1 | | | 11 |
| | 0010 | 0 | SUB | 01 (Sub) | 00 |
| | | 1 | | | 11 |
| | 0000 | 0 | AND | 10 (And) | 00 |
| | | 1 | | | 10 |
| | 1100 | 0 | ORR | 11 (Or) | 00 |
| | | 1 | | | 10 |

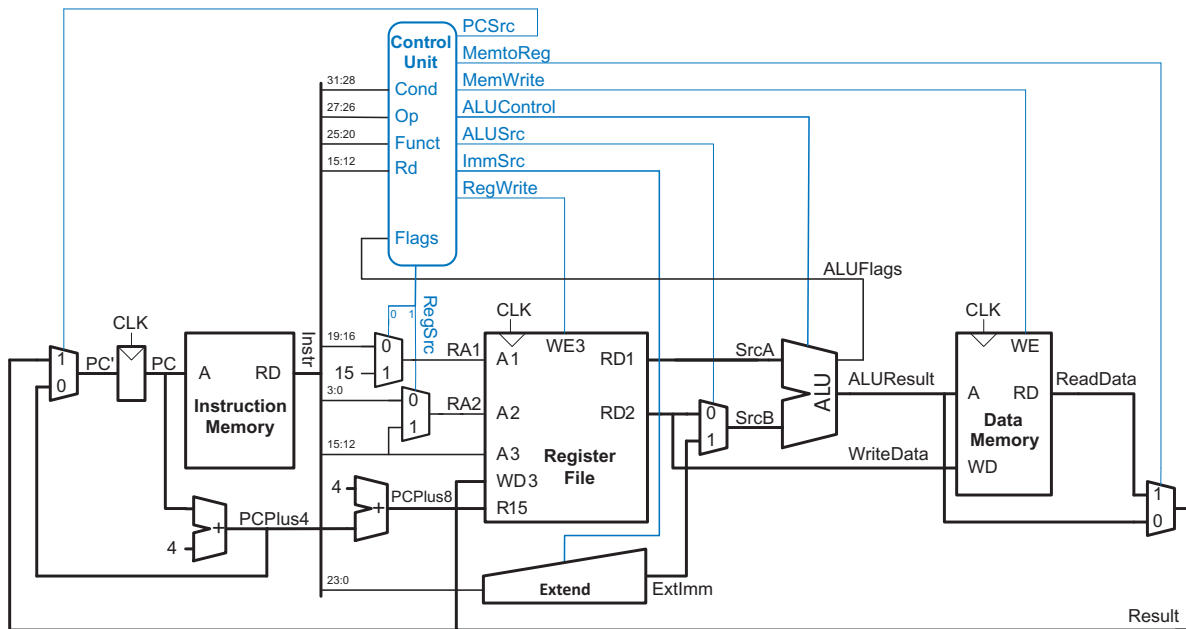


Figure 7.13 Complete single-cycle processor

Problem 3: Mark up Verilog to implement TST

```

module arm(input logic clk, reset,
           output logic [31:0] PC,
           input logic [31:0] Instr,
           output logic MemWrite,
           output logic [31:0] ALUResult, WriteData,
           input logic [31:0] ReadData);

    logic [3:0] ALUFlags;
    logic RegWrite,
          ALUSrc, MemtoReg, PCSrc;
    logic [1:0] RegSrc, ImmSrc, ALUControl;

    controller c(clk, reset, Instr[31:12], ALUFlags,
                RegSrc, RegWrite, ImmSrc,
                ALUSrc, ALUControl,
                MemWrite, MemtoReg, PCSrc);
    datapath dp(clk, reset,
                RegSrc, RegWrite, ImmSrc,
                ALUSrc, ALUControl,
                MemtoReg, PCSrc,
                ALUFlags, PC, Instr,
                ALUResult, WriteData, ReadData);
endmodule

module controller(input logic clk, reset,
                 input logic [31:12] Instr,
                 input logic [3:0] ALUFlags,
                 output logic [1:0] RegSrc,
                 output logic RegWrite,
                 output logic [1:0] ImmSrc,
                 output logic ALUSrc,
                 output logic [1:0] ALUControl,
                 output logic MemWrite, MemtoReg,
                 output logic PCSrc);

    logic [1:0] FlagW;
    logic PCS, RegW, MemW;

    decoder dec(Instr[27:26], Instr[25:20], Instr[15:12],
                FlagW, PCS, RegW, MemW,
                MemtoReg, ALUSrc, ImmSrc, RegSrc, ALUControl);
    condlogic cl(clk, reset, Instr[31:28], ALUFlags,
                FlagW, PCS, RegW, MemW,
                PCSrc, RegWrite, MemWrite);
endmodule

module decoder(input logic [1:0] Op,
              input logic [5:0] Funct,
              input logic [3:0] Rd,
              output logic [1:0] FlagW,
              output logic PCS, RegW, MemW,
              output logic MemtoReg, ALUSrc,
              output logic [1:0] ImmSrc, RegSrc, ALUControl);

    logic [9:0] controls;
    logic Branch, ALUOp;

    // Main Decoder

    always_comb
        case(Op)
            2'b00: if (Funct[5]) // Data processing immediate
                    controls = 10'b0000101001;
                    // Data processing register
                else
                    controls = 10'b0000001001;
                    // LDR
            2'b01: if (Funct[0]) controls = 10'b0001111000;
                    // STR
                else
                    controls = 10'b1001110100;
                    // B
            2'b10: controls = 10'b0110100010;
                    // Unimplemented
            default: controls = 10'bx;
        endcase

    assign {RegSrc, ImmSrc, ALUSrc, MemtoReg,
            RegW, MemW, Branch, ALUOp} = controls;

    // ALU Decoder
    always_comb
        if (ALUOp) begin // which DP Instr?
            case(Funct[4:1])
                4'b0100: ALUControl = 2'b00; // ADD
                4'b0010: ALUControl = 2'b01; // SUB
            endcase
        end

```

```

        4'b0000: ALUControl = 2'b10; // AND
        4'b1100: ALUControl = 2'b11; // ORR
        default: ALUControl = 2'bx; // unimplemented
    endcase
    // update flags if S bit is set
    // (C & V only updated for arith instructions)
    FlagW[1] = Funct[0]; // FlagW[1] = S-bit
    // FlagW[0] = S-bit & (ADD | SUB)
    FlagW[0] = Funct[0] &
        (ALUControl == 2'b00 | ALUControl == 2'b01);
    end else begin
        ALUControl = 2'b00; // add for non-DP instructions
        FlagW = 2'b00; // don't update Flags
    end

    // PC Logic
    assign PCS = ((Rd == 4'b1111) & RegW) | Branch;
endmodule

module condlogic(input logic clk, reset,
                input logic [3:0] Cond,
                input logic [3:0] ALUFlags,
                input logic [1:0] FlagW,
                input logic PCS, RegW, MemW,
                output logic PCSrc, RegWrite, MemWrite);

    logic [1:0] FlagWrite;
    logic [3:0] Flags;
    logic CondEx;

    flopenr #(2)flagreg1(clk, reset, FlagWrite[1],
                        ALUFlags[3:2], Flags[3:2]);
    flopenr #(2)flagreg0(clk, reset, FlagWrite[0],
                        ALUFlags[1:0], Flags[1:0]);

    // write controls are conditional
    condcheck cc(Cond, Flags, CondEx);
    assign FlagWrite = FlagW & {2{CondEx}};
    assign RegWrite = RegW & CondEx;
    assign MemWrite = MemW & CondEx;
    assign PCSrc = PCS & CondEx;
endmodule

module datapath(input logic clk, reset,
                input logic [1:0] RegSrc,
                input logic RegWrite,
                input logic [1:0] ImmSrc,
                input logic ALUSrc,
                input logic [1:0] ALUControl,
                input logic MemtoReg,
                input logic PCSrc,
                output logic [3:0] ALUFlags,
                output logic [31:0] PC,
                input logic [31:0] Instr,
                output logic [31:0] ALUResult, WriteData,
                input logic [31:0] ReadData);

    logic [31:0] PCNext, PCPlus4, PCPlus8;
    logic [31:0] ExtImm, SrcA, SrcB, Result;
    logic [3:0] RA1, RA2;

    // next PC logic
    mux2 #(32) pcmux(PCPlus4, Result, PCSrc, PCNext);
    flopr #(32) pcreg(clk, reset, PCNext, PC);
    adder #(32) pcadd1(PC, 32'b100, PCPlus4);
    adder #(32) pcadd2(PCPlus4, 32'b100, PCPlus8);

    // register file logic
    mux2 #(4) ralmux(Instr[19:16], 4'b1111, RegSrc[0], RA1);
    mux2 #(4) ra2mux(Instr[3:0], Instr[15:12], RegSrc[1], RA2);
    regfile rf(clk, RegWrite, RA1, RA2,
              Instr[15:12], Result, PCPlus8,
              SrcA, WriteData);
    mux2 #(32) resmux(ALUResult, ReadData, MemtoReg, Result);
    extend ext(Instr[23:0], ImmSrc, ExtImm);

    // ALU logic
    mux2 #(32) srcbmux(WriteData, ExtImm, ALUSrc, SrcB);
    alu alu(SrcA, SrcB, ALUControl,
           ALUResult, ALUFlags);
endmodule

```