University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Science

EECS150, Spring 2010

Homework Assignment 5: More Transistors and Single Cycle Processor Implementations
Due February 26th, 2pm

Homework submission will only be through SVN. Email submissions will not be accepted! Please format your homework as plain text with either PNG or PDF for any necessary figures. Microsoft Visio is installed on the machines in 125 Cory, and is a useful tool for drawing figures of all kinds.

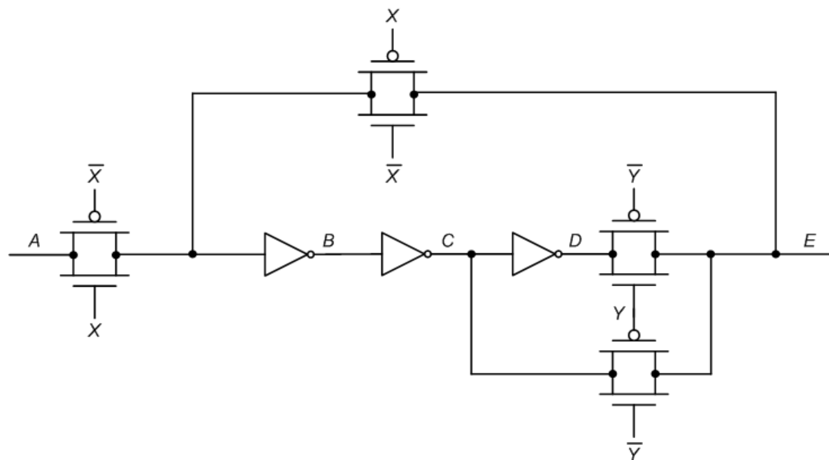
1. Implement a basic 4-input LUT down to the transistor level. Allow the LUT to be programmable with a new function whenever the **prog** signal is asserted (there is no clock). You may implement this in any way you wish, however, the goal is to minimize the number of transistors used. How many configuration bits does your LUT need?
   Hint: You may want to consider latches, which, unlike flip-flops, may be triggered using signals besides **clock**. Transmission gate muxes may also be useful.

2. Using the minimum number of transistors, draw the Static CMOS implementations for the following functions:

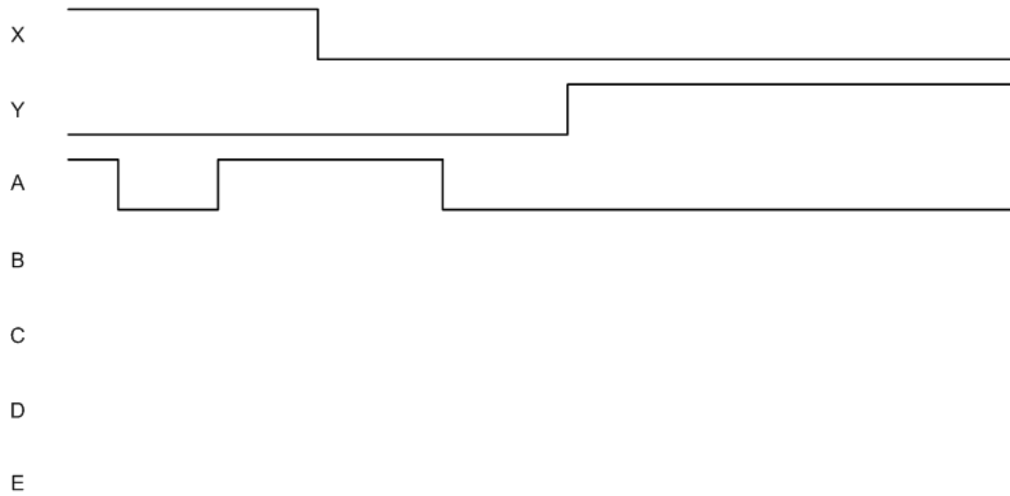
   (a) `y = (abc + d)'`

   (b) `y = ab + cd`

3. Consider the circuit pictured here:

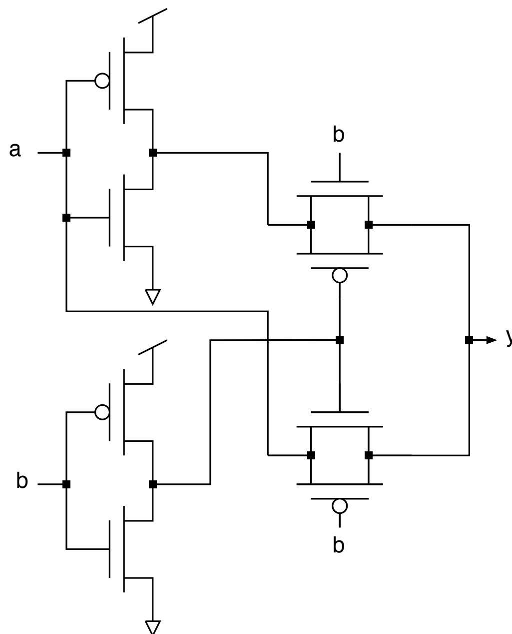


   (a) Complete the timing diagram below for the circuit. Note that it may be helpful for you to think about the inverters as having a little bit of delay. What does this circuit do if **X** is 0 and **Y** is 1? Why?

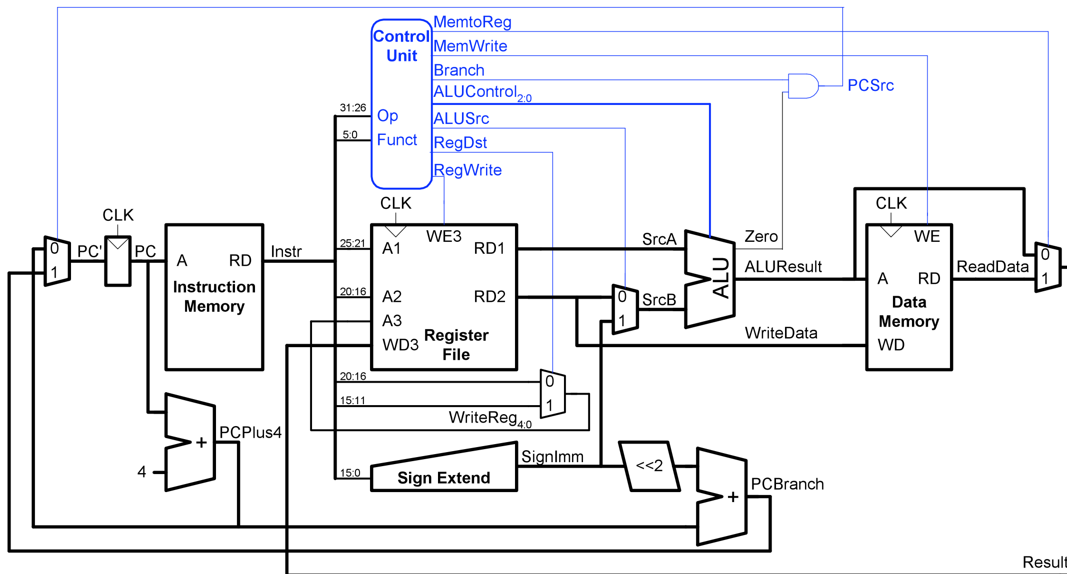


(b) How would the waveform for signal **E** change if the circuit had 5 inverters in series instead of 3? Why?

4. For the CMOS circuit shown in below, write the truth table, give a Boolean expression that corresponds to the circuit, and draw an alternative implementation that doesn't use transmission gates:



5. (based on DDCA 7.3) - Modify the datapath shown below to add functionality for **sll** and **jal**:



6. (based on DDCA 7.10) - Do the following for **sll** and **jal** only.

- For parts **(a)** through **(e)**, make the necessary changes in the Verilog to implement the changes made to the datapath in Problem 5.
- Study parts **(f)** through **(i)** and write a simple MIPS program to test the functionality of the changes you implemented.

(a) Single-cycle MIPS Processor:

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

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

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

(b) Controller:

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

  wire [1:0] aluop;
  wire       branch;

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

  assign pcsrc = branch & zero;
endmodule
```

(c) Main Decoder:

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

  reg [8:0] controls;

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

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

(d) ALU Decoder:

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

  always @(*)
    case(aluop)
      2'b00: alucontrol <= 3'b010;  // add
      2'b01: alucontrol <= 3'b110;  // sub
```

```
      default: case(funct)          // RTYPE
          6'b100000: alucontrol <= 3'b010; // ADD
          6'b100010: alucontrol <= 3'b110; // SUB
          6'b100100: alucontrol <= 3'b000; // AND
          6'b100101: alucontrol <= 3'b001; // OR
          6'b101010: alucontrol <= 3'b111; // SLT
          default:   alucontrol <= 3'bxxx; // ???
        endcase
    endcase
endmodule
```

(e) Datapath:

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

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

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

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

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

```
endmodule
```

(f) MIPS Testbench:

```
module testbench();

  reg         clk;
  reg         reset;

  wire [31:0] writedata, dataadr;
  wire memwrite;

  // instantiate device to be tested
  top dut(clk, reset, writedata, dataadr, memwrite);

  // initialize test
  initial
    begin
      reset <= 1; # 22; reset <= 0;
    end

  // generate clock to sequence tests
  always
    begin
      clk <= 1; # 5; clk <= 0; # 5;
    end

  // check that 7 gets written to address 84
  always@(negedge clk)
    begin
      if(memwrite) begin
        if(dataadr === 84 & writedata === 7) begin
          $display("Simulation succeeded");
          $stop;
        end else if (dataadr !== 80) begin
          $display("Simulation failed");
          $stop;
        end
      end
    end
endmodule
```

(g) MIPS Top-Level Module:

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

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

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

endmodule
```

(h) MIPS Data Memory:

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

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

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

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

(i) MIPS Instruction Memory:

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

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

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

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