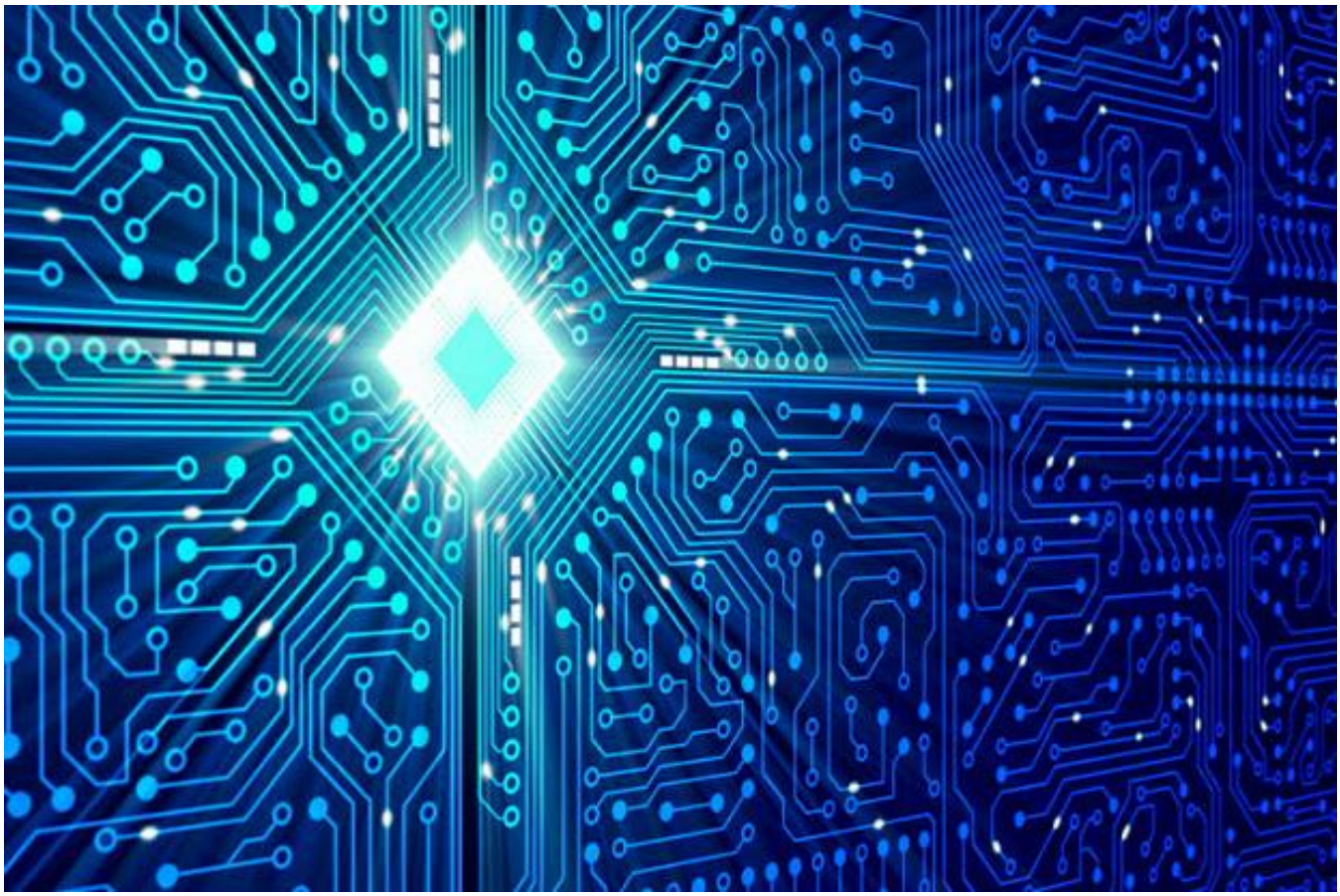


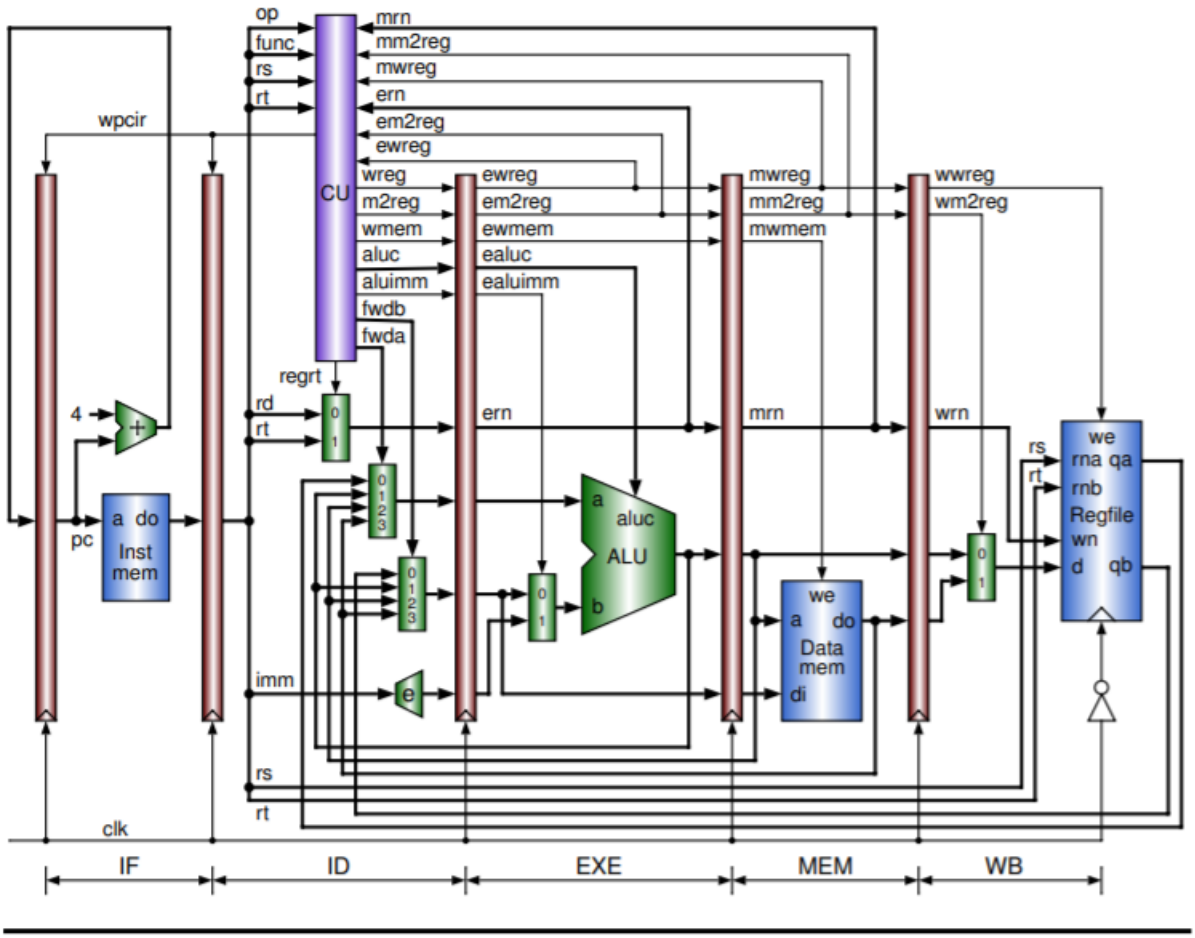
Lab 5 (Honors Option)

COMPUTER ORGANIZATION AND DESIGN

Taylan Unal | CMPEN 331 Section 1 | 12/21/19

Summary:

Throughout the semester, stage by stage, we've worked towards implementing a fully functioning CPU design over on FPGA (XC7Z010--1CLG400C). The design features 5 stages (IF, ID, EXE, MEM, WB) complete with registers (IFID, IDEXE, EXEMEM, MEMWB) and various components needed by the CPU to perform necessary functions. The CPU features forwarding and pipelining to avoid hazards common to instruction sets requiring the same data values. Additionally, the CPU can handle most common instructions like loading, storing data to registers/memory, performing arithmetic operations on register values and memory values, and can handle branches and jumps in instructions. Working through this lab, I learned a great deal about the intricacies of designing a functioning CPU, correctly defining variables and I/O, and ensuring designing an accurate testbench. This report features all the code necessary to run the CPU design, the testbench necessary to run all of the components in the CPU, design schematics, floor planning, and I/O planning.



Top Module (CPU):

```
1.  //Taylan Unal CMPEN 331.001 Lab5 Honors Option Top Module
2.  //Avoiding Data Hazards in CPU
3.  `timescale 1ns / 1ps
4.  module PC(clk, PCin, PCout);
5.      input clk;
6.      input [31:0] PCin;
7.      output reg [31:0] PCout;
8.      initial begin
9.          PCout = 100;
10.     end
11.
12.     always @(posedge clk) begin
13.         PCout <= PCin;
14.     end
15. endmodule
16.
17. module PCAdder(PCin, PCout);
18.     input [31:0] PCin;
19.     output reg [31:0] PCout;
20.
21.     always @(PCin) begin
22.         PCout <= PCin + 4;
23.     end
24. endmodule
25.
26. module InstMem(pcin, do); //a is 'PC input', do is 'IM output'
27.     input [31:0] pcin;
28.     output reg [31:0] do;
29.     reg [31:0] IM[0:127]; //load word from memory, leaving room for more instructions later.
30.     initial begin
31.         IM[32'd100] = 32'h8c220000; //lw $v0 00($at)
32.         IM[32'd104] = 32'h8c230004; //lw $v1 04($at)
33.         IM[32'd108] = 32'h8c240008; //lw $a0 08($at)
34.         IM[32'd112] = 32'h8c25000c; //lw $a1 12($at)
35.         IM[32'd116] = 32'h012E6820; //add $t5,$1,$t6 (adding $6,$2,$10)
36.     end
37.
38.     always @ (pcin) begin //good
39.         do <= IM[pcin];
40.     end
41. endmodule
42.
43. module IFID(clk, do, op, rd, rs, rt, func, imm);
44.     input clk;
45.     input [31:0] do;//instruction input
46.     output reg [5:0] op,func;//I-type values
47.     output reg [5:0] rd,rs,rt;//I-type values
48.     output reg [15:0] imm;//immediate value
49.     reg [31:0] IF; //IFID  temporarymemory
50.     always @ (posedge clk) begin
51.         IF <= do; //save value of input until negedge
52.     end
53.
54.     always @(negedge clk)
55.         begin
56.             op <= IF[31:26];
57.             rs <= IF[25:21];
58.             rt <= IF[20:16];
```

```
            rd <= IF[15:11];
            func <= IF[5:0];
            imm <= IF[15:0];
        end
endmodule

//UPDATED ControlUnit with internal forwarding and pipeline stall.
//Have to extend CU to detect and handle hazards.
module ControlUnit(op,func,rs,rt,mrn,mm2reg,mwreg,ern,em2reg,ewreg, //inputs
                   aluc,wreg,m2reg,wmem,aluimm,fwda,fwdb,wpcir,regrt);//outputs
    input [5:0] op,func;
    input [4:0] rs,rt; //added rs,rt to control unit.
    input [4:0] mrn; // MEM RN
    input mm2reg,mwreg; // MEM m2reg, wreg
    input [4:0] ern; // EXE RN
    input em2reg,ewreg; // EXE m2reg, wreg
    output reg [3:0] aluc;
    output reg wreg, m2reg, wmem, aluimm, regrt;

    output reg [1:0] fwda, fwdb; //Forwarding values to MUX.
    output reg wpcir; //Write enable for PC

    initial begin
        wreg <= 0; //RegWrite
        m2reg <= 0; //Mem2Reg
        wmem <= 0; //Write Memory
        aluimm <= 0; //ALU source
        regrt <= 0; //Reg Destination
    end
    always @ (op,func) begin //adding func
        if(op == 0) begin //R-Type instructions
            case(func)
                32: aluc <= 2; //Register ADD
                34: aluc <= 6; //Register SUB
                36: aluc <= 0; //Register AND
                37: aluc <= 1; //Register OR
                39: aluc <= 12;//Register XOR
             endcase
             wreg <= 1;//RegWrite
             m2reg <= 0;//Mem2Reg
             wmem <= 0;//WriteMem
                    aluimm <= 0; //AluSrc
                    regrt <= 1; //input to ControlMux
              end
              if(op == 6'b100011) begin //Load Word (LW) I-type
                  aluc <= 2;//add
                  wreg <= 1;//RegWrite
                  m2reg <= 1;//Mem2Reg
                  wmem <= 0;//WriteMem
                  aluimm <= 1;//AluSrc
                  regrt <= 0; //input to ControlMux
              end
              if(op == 6'b101011) begin //Store Word (SW) I-type
                  aluc <= 2; //add
                  wreg <= 0;//RegWrite
                  m2reg <= 0;//Mem2Reg
                  wmem <= 1;//WriteMem
                  aluimm <= 1;//AluSrc
                  regrt <= 1; //input to ControlMux
              end
              if (op == 6'b000100) //Branch Equals (BEQ)
```

```
120.                begin
121.                    aluc <= 6; //subtract
122.                    wreg <= 0; //WriteReg
123.                    m2reg <= 0;//Mem2Reg
124.                    wmem <= 0; //Write Memory
125.                    aluimm <= 0; //ALUsrc
126.                    regrt <= 1; //input to ControlMux
127.                end
128.        end
129.
130.            //Start Forwarding Unit. From Textbook pg. 222
131.            //Inputs: rs,rt,mrn,mm2reg,mwreg,ern,em2reg,ewreg. Outputs: fwda,fwdb
132.            always @ (rs or rt) begin //input RS and RT into CU.
133.                //Forward A Register
134.                if (ewreg && (ern != 0) && !em2reg) begin
135.                    assign fwda = 2'b01; //MEM Hazard. Select EXE_ALU (removing assign)
136.                end
137.                else if (mwreg && (mrn != 0) && !mm2reg) begin
138.                    assign fwda = 2'b10; //EXE Hazard. Select MEM_ALU
139.                end
140.                else if (mwreg && (mrn != 0) && mm2reg) begin
141.                    assign fwda = 2'b11; //EXE Hazard. Select MEM_LW
142.                end
143.                else begin
144.                    assign fwda = 2'b00;//default, no hazards
145.                end
146.
147.                //Forward B Register
148.                if (ewreg && (ern != 0) && !em2reg) begin
149.                    assign fwdb = 2'b01; //MEM Hazard. Select maluOut
150.                end
151.                else if (mwreg && (mrn != 0) && !mm2reg) begin
152.                    assign fwdb = 2'b10; //EXE Hazard. Select mqb
153.                end
154.                else if (mwreg && (mrn != 0) && mm2reg) begin
155.                    assign fwdb = 2'b11; //EXE Hazard. Select dmOut
156.                end
157.                else begin
158.                    assign fwdb = 2'b00;//default, no hazards
159.                end
160.            end
161.        endmodule
162.
163.        module ForwardAMux(fwda,qa,maluOut,mqb,dmOut,fwda_out);
164.            //input forwardingA [1:0]
165.            //input 0: qa [31:0], 1: maluOut [31:0], 2: mqb [31:0], 3: dmOut [31:0]
166.            //output: FWDA_out [31:0]. to IDEXE, then is (a) value in ALU
167.            input [1:0] fwda; //fwda signal from CU
168.            input [31:0] qa, maluOut, mqb, dmOut; //WB signals: (qa), MEM:(maluOut,mqb,dmOut)
169.            output fwda_out;
170.            reg [31:0] fwda_out;
171.
172.            always @ (qa or maluOut or mqb or dmOut or fwda)//if selector, any signal enabled
173.            begin
174.                case (fwda)//depending on fwda state, select output
175.                    2'b00: fwda_out <= qa;//means no hazard, go to regfile (qa)
176.                    2'b01: fwda_out <= maluOut;//MEM Hazard. Select maluOut
177.                    2'b10: fwda_out <= mqb;//EXE Hazard. Select mqb
178.                    2'b11: fwda_out <= dmOut;//EXE Hazard. Select dmOut
179.                endcase
180.            end
```

```
181.        endmodule
182.
183.        module ForwardBMux(fwdb,qb,maluOut,mqb,dmOut,fwdb_out);
184.            //input forwardingB [1:0]
185.            //input 0: qb, 1: maluOut [31:0], 2: mqb [31:0], 3: dmOut [31:0]
186.            //output: FWDB_out [31:0]. to IDEXE, goes to eqb in ALUMUX.
187.            input [1:0] fwdb; //fwda signal from CU
188.            input [31:0] qb, maluOut, mqb, dmOut; //WB signals: (qa), MEM:(maluOut,mqb,dmOut)
189.            output fwdb_out;
190.            reg [31:0] fwdb_out;
191.
192.            always @ (qb or maluOut or mqb or dmOut or fwdb)//if selector, any signal enabled
193.            begin
194.                case (fwdb)//depending on fwda state, select output
195.                    2'b00: fwdb_out <= qb;//means no hazard, go to regfile (qa)
196.                    2'b01: fwdb_out <= maluOut;//MEM Hazard. Select maluOut
197.                    2'b10: fwdb_out <= mqb;//EXE Hazard. Select mqb
198.                    2'b11: fwdb_out <= dmOut;//EXE Hazard. Select dmOut
199.                endcase
200.            end
201.        endmodule
202.
203.        module ControlMux(rd,rt,regrt,rn); //selects either rd,rt
204.            input [4:0] rd,rt; //output from IFIDReg
205.            input regrt; //from controlunit, enable write
206.            output reg [4:0] rn; //output
207.
208.            always @(regrt,rt,rd) begin
209.                case (regrt)
210.                    0:
211.                        rn <= rt;
212.                    1:
213.                        rn <= rd;
214.                endcase
215.            end
216.        endmodule
217.
218.        module SignExtend(immIn, immOut);//input IF instruct val, output extend 32 bit val
219.            input [15:0] immIn; //short (nonextended) value
220.            output reg [31:0] immOut; //extended value
221.
222.            always @(immIn) begin
223.                immOut <= {{16{immIn[15]}},immIn[15:0]}; //extends 16bit number to 32bits.
224.            end
225.        endmodule
226.
227.        module IDEXE(clk, wreg, m2reg, wmem, aluc, aluimm, rn, qa, qb, imm,
228.                     ewreg, em2reg, ewmem, ealuc, ealuimm, ern, eqa, eqb, eimm);
229.            input clk;
230.            input wreg, m2reg, wmem, aluimm; //input to IDEXE
231.            input [3:0] aluc; //output from control unit
232.            input [4:0] rn; //output from mux into IDEXE
233.            input [31:0] qa, qb; //output from regfile
234.            input [31:0] imm; //extended immediate value
235.
236.            //Use these to store values for later assignment using posedge, negedge.
237.            reg wreg2, m2reg2, wmem2, aluimm2;
238.            reg [4:0] rn2;
239.            reg [3:0] aluc2;
240.            reg [31:0] qa2, qb2;
241.            reg [31:0] imm2;
```

```
242.            output reg ewreg, em2reg, ewmem, ealuimm; //extended outputs from control unit
243.            output reg [3:0] ealuc; //extended outputs from control unit, into ALU
244.            output reg [4:0] ern; //extended outputs from mux
245.            output reg [31:0] eqa, eqb; //extended outputs from regfile
246.            output reg [31:0] eimm; //extended outputs from sign extender
247.
248.            always@(posedge clk) begin //pass values into middle save values.(save regs<=input)
249.                wreg2 <= wreg;
250.                m2reg2 <= m2reg;
251.                wmem2 <= wmem;
252.                aluimm2 <= aluimm;
253.                rn2 <= rn;
254.                aluc2 <= aluc;
255.                qa2 <= qa;
256.                qb2 <= qb;
257.                imm2 <= imm;
258.            end
259.
260.            always@(negedge clk) begin //output values from saved values. (output <= save regs)

261.                ewreg <= wreg2;
262.                em2reg <= m2reg2;
263.                ewmem <= wmem2;
264.                ealuimm <= aluimm2;
265.                ern <= rn2;
266.                ealuc <= aluc2;
267.                eqa <= qa2;
268.                eqb <= qb2;
269.                eimm <= imm2;
270.            end
271.        endmodule
272.
273.        module ALUMux(eqb, eimmExt, ealuimm, muxImm);
274.            input [31:0] eqb, eimmExt; //qb value and immExt from IDEXE
275.            input ealuimm;//selector from IDEXE
276.            output reg [31:0] muxImm; //output of ALUMux
277.
278.            always @ (eqb, eimmExt) begin
279.                case (ealuimm)
280.                    0: muxImm = eqb;
281.                    1: muxImm = eimmExt;
282.                endcase
283.            end
284.        endmodule
285.
286.        module ALU (eqa, eqb, ealuc, aluOut);
287.            input [31:0] eqa, eqb;
288.            input [3:0] ealuc; //4bit number
289.
290.            output reg [31:0] aluOut;
291.
292.            always @ (eqa, eqb) begin
293.                case (ealuc)
294.                    4'b0010: aluOut <= eqa + eqb;
295.                endcase
296.            end
297.        endmodule
298.
299.        module EXEMEM (clock, ewreg, em2reg, ewmem, ern, aluOut, eqb,
300.                       mwreg, mm2reg, mwmem, mrn, maluOut, mqb);
301.            input clock, ewreg, em2reg, ewmem;
```

```
302.            input [4:0] ern;
303.            input [31:0] aluOut, eqb;
304.
305.            reg ewreg2, em2reg2, ewmem2;
306.            reg [4:0] ern2;
307.            reg [31:0] aluOut2, eqb2;
308.
309.            output reg mwreg, mm2reg, mwmem; //outputs on memory side of register
310.            output reg [4:0] mrn;
311.            output reg [31:0] maluOut, mqb;
312.
313.            always @ (posedge clock) begin //pass values into middle save (save regs <= input)
314.                ewreg2 <= ewreg;
315.                em2reg2 <= em2reg;
316.                ewmem2 <= ewmem;
317.                ern2 <= ern;
318.                aluOut2 <= aluOut;
319.                eqb2 <= eqb;
320.            end
321.
322.            always @ (negedge clock) begin
323.                mwreg <= ewreg2;
324.                mm2reg <= em2reg;
325.                mwmem <= ewmem;
326.                mrn <= ern2;
327.                maluOut <= aluOut2;
328.                mqb <= eqb2;
329.            end
330.        endmodule
331.
332.        module DataMemory (maluOut, mqb, mwmem, dmOut);//wm, aludata_in, qbdata_in, dmdata_out)
333.            input [31:0] maluOut, mqb; //input alu output value, qb value
334.            input mwmem;//write memory
335.            output reg [31:0] dmOut;
336.            reg [31:0] DM [0:36];
337.            initial begin //set first 10 words to data memory
338.                DM[32'd0] = 32'hA00000AA;
339.                DM[32'd4] = 32'h10000011;
340.                DM[32'd8] = 32'h20000022;
341.                DM[32'd12] = 32'h30000033;
342.                DM[32'd16] = 32'h40000044;
343.                DM[32'd20] = 32'h50000055;
344.                DM[32'd24] = 32'h60000066;
345.                DM[32'd28] = 32'h70000077;
346.                DM[32'd32] = 32'h80000088;
347.                DM[32'd36] = 32'h90000099;
348.            end
349.
350.            always @ (maluOut, mqb) begin
351.                case (mwmem)
352.                    1'b0: dmOut <= DM[maluOut];
353.                    1'b1: dmOut <= DM[mqb];
354.                endcase
355.            end
356.        endmodule
357.        module MEMWB(clk, mwreg, mm2reg, mrn, maluOut, dmOut,
358.                     wwreg, wm2reg, wrn, waluOut, wdmOut);//adding WB outputs
359.            input clk, mwreg, mm2reg;
360.            input [31:0] maluOut, dmOut;
361.            input [4:0] mrn;
362.            reg mwreg2, mm2reg2;
```

```
363.            reg [31:0] maluOut2, dmOut2;
364.            reg [4:0] mrn2;
365.            output reg wwreg, wm2reg;
366.            output reg [31:0] waluOut, wdmOut;
367.            output reg [4:0] wrn;
368.
369.            always @(posedge clk) begin //stores into temporary regs
370.                mwreg2 <= mwreg;
371.                mm2reg2 <= mm2reg;
372.                maluOut2 <= maluOut;
373.                dmOut2 <= dmOut;
374.                mrn2 <= mrn;
375.            end
376.            always @(negedge clk) begin
377.                wwreg <= mwreg2;
378.                wm2reg <= mm2reg2;
379.                waluOut <= maluOut2;
380.                wdmOut <= dmOut2;
381.                wrn <= mrn2;
382.            end
383.        endmodule
384.
385.        module WBMux(waluOut, wdmOut, wm2reg, wbMuxOut);
386.            input [31:0] waluOut, wdmOut;//takes in aluresult and data memory address
387.            input wm2reg; //write enable memory to register
388.            output reg [31:0] wbMuxOut; //output to regfile
389.
390.            always @(waluOut, wdmOut, wm2reg) begin
391.                case(wm2reg)
392.                    0: wbMuxOut <= waluOut; //if wm2reg=0, output alu value
393.                    1: wbMuxOut <= wdmOut; //if wm2reg=1, output datamem value.
394.                endcase
395.            end
396.        endmodule
397.
398.        module RegFile(clk, rs, rt, qa, qb, wrn, wbmuxOut); //(clk,rs,rt,qa,qb,wrn,wbmuxOut)
399.            input clk;
400.            input [31:0] wbmuxOut; //data to write to register
401.            input [4:0] rs, rt, wrn; //rs val in, rt val in, rd val in
402.            reg [31:0] regs [0:31]; //32 x 32 register file. Store all the registers.
403.            output reg [31:0] qa, qb; //qa->rt out, qb->rs
404.
405.            initial begin //initialize all 32 registers to 0.
406.                {regs[0], regs[1], regs[2], regs[3], regs[4], regs[5], regs[6], regs[7],
407.                regs[8], regs[9], regs[10], regs[11], regs[12], regs[13], regs[14], regs[15],
408.                regs[16], regs[17], regs[18], regs[19], regs[20], regs[21], regs[22], regs[23],
409.                regs[24], regs[25], regs[26], regs[27], regs[28], regs[29], regs[30], regs[31]}
    = 0;//32'h00000000;
410.            end
411.
412.            always @(posedge clk) begin //write during first half of cycle
413.                if(wrn != 0) begin//write operation
414.                    regs[wrn] <= wbmuxOut;
415.                end
416.            end
417.
418.            always @(rs, rt) begin//read during second half of cycle
419.                qa <= regs[rs]; //register val in register rs
420.                qb <= regs[rt]; //register val in register rt
421.            end
422.        endmodule
```

Testbench Module:

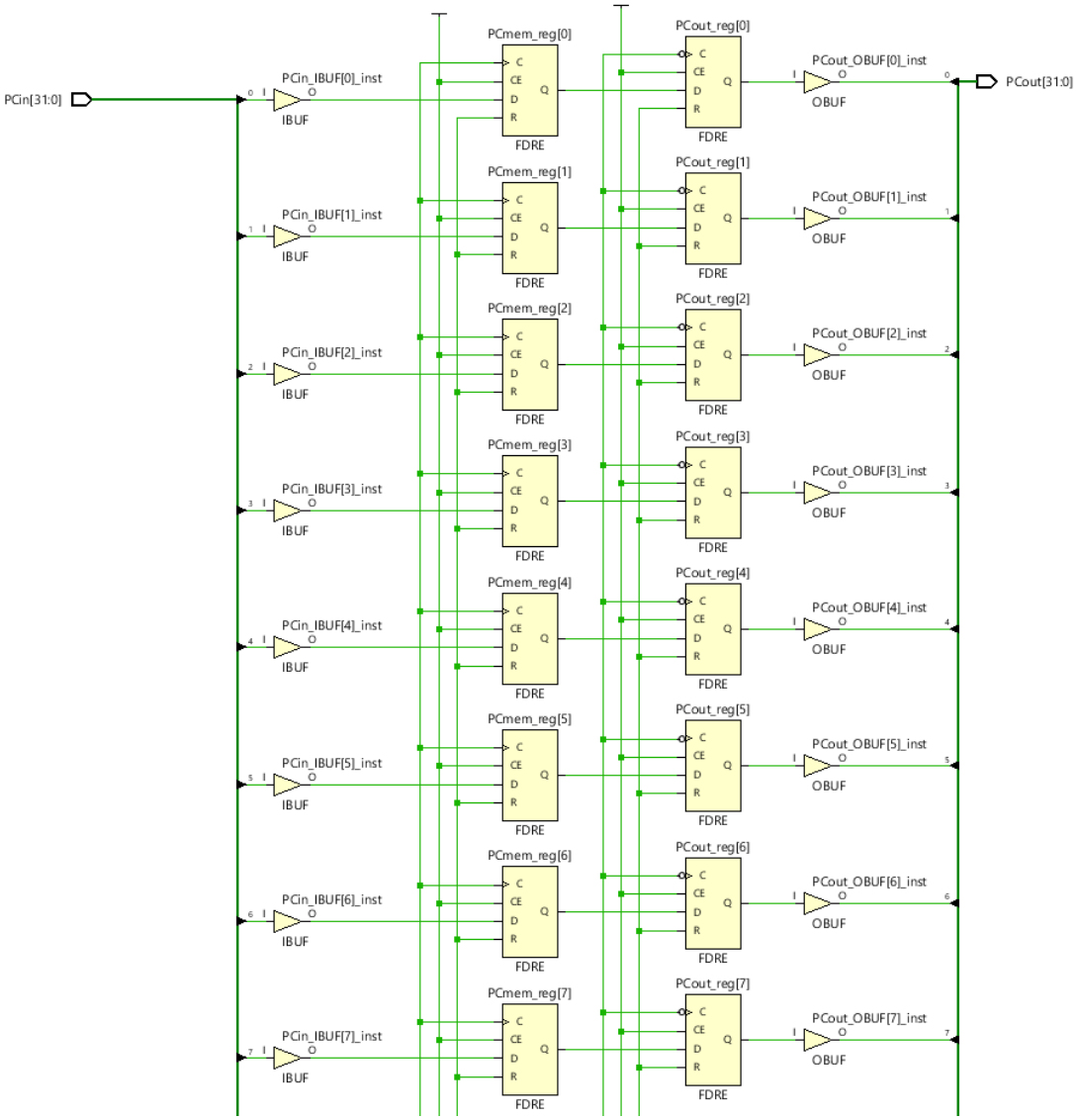
```
1.  //Taylan Unal CMPEN 331.001 Lab5 Honors Option Testbench
2.  `timescale 1ns / 1ps
3.
4.  module CPU_Test();
5.  //WIRES ARE OUTPUTS, REGS ARE INPUTS.
6.      reg clk; //global clock signal
7.      //IF STAGE
8.      wire [31:0] pcin; //PC output to PC register
9.      wire [31:0] pcout; //PC output to adder
10.     wire [31:0] do; //instruction to be executed (im_out)
11.
12.     //ID STAGE
13.     wire [5:0] op, func; //opcode and func fields.
14.     wire [4:0] rd,rs,rt; //source, destination registers of instruction
15.     wire [15:0] imm; //immediate field (for i-type instructions)
16.     wire [3:0] aluc; //alucontrol to input into ALU to determine operation
17.     wire wreg; //(ID) write enable for IDEXE register
18.     wire m2reg; //(ID) write enable for IDEXE register from data memory
19.     wire wmem; //(ID) write enable for data memory
20.     wire aluimm; //(ID) sends either alu control signal or immediate value to ALUMUX
21.     wire regrt; // (ID) input to controlMux, chooses what to write rt/rd to IDEXEregister
22.     wire [4:0] rn; //(ID) ** output of controlMux, chooses to write reg number to IDEXE
23.     wire [31:0] qa,qb;//(ID) ** register file values for instruction. (Note register is now at
    end of WB)
24.     wire [31:0] immExt; // (ID)extended immediate value to 32 bits (I-type)
25.
26.     //EXE STAGE
27.     wire [3:0] ealuc; //(EXE) ALUControl to determine ALU operation
28.     wire ewreg; //(EXE) write enable for EXEMEM register
29.     wire em2reg; //(EXE) write enable for EXEMEM register from data memory
30.     wire ewmem; //(EXE) write enable for data memory
31.     wire ealuimm; //(EXE) sends either ALUcontrol signal or immediate value to ALUMUX
32.     wire [4:0] ern; //(EXE) **output of controlMux, chooses to write reg number to EXEMEM
33.     wire [31:0] eqa, eqb; //(EXE) **register file values for instruction
34.     wire [31:0] eimmExt; //(EXE) extended immediate value after written to IDEXE
35.     wire [31:0] muxImm; //(EXE) output of immediate mux
36.     wire [31:0] aluOut; //(EXE) output of ALU Module
37.
38.     //MEM STAGE
39.     wire mwreg; //(MEM) write enable for MEMWB register
40.     wire mm2reg; //(MEM) write enable for MEMWB register from data memory
41.     wire mwmem; //(MEM) write enable for data memory
42.     wire [4:0] mrn; //(MEM) **output of controlMux, chooses to write reg number to MEMWB
43.     wire [31:0] maluOut; //(MEM) output of ALU Module
44.     wire [31:0] mqb; //(MEM) qb value from regfile->IDEXEReg->EXEMEMReg
45.     wire [31:0] dmOut; //(MEM) data memory output
46.
47.     //WB STAGE
48.     wire wwreg; //(WB) write enable for Regfile. (After all other stages enable values
49.     wire wm2reg; //(WB) write enable for Regfile from datamemory
50.     wire [4:0] wrn; //(WB) **output of controlMux, chooses to write reg number to Regfile
51.     wire [31:0] waluOut; //(WB) output of ALU Module
52.     wire [31:0] wdmOut; //(WB) data memory output
53.     wire [31:0] wbMuxOut;//(WB) output of Mem2Reg Mux for DataMem->Regfile
54.
55.     //FORWARDING UNIT and MUXes (All in ID Stage)
56.     wire [1:0] fwda; //output signal from ControlUnit, input for ForwardAMux
57.     wire [31:0] fwdaOut; //output signal from ForwardAMux
```

```
58.     wire [1:0] fwdb; //output signal from ControlUnit, input for ForwardBMux
59.     wire [31:0] fwdbOut; //output signal from ForwardBMux
60. ////////////////////////////////////////////////////////////////////////////////
61.     //START Testbench
62.     //IF STAGE
63.     PC PC(clk, pcin, pcout); //PCRegister: input clk, PCinput, outputs result from PCAdder
64.     PCAdder adder(pcout, pcin); //PCAdder: input PCOutput, increments by 4
65.     InstMem instmem(pcout, do); //InstMem: has instructions/outputs from PCRegister
66.
67.     //IFID REGISTER
68.     IFID ifid(clk,do,//inputs
69.               op,rd,rs,rt,func,imm);//outputs
70.
71.     //ID STAGE
72.     ControlUnit CU(op,func,rs,rt,mrn,mm2reg,mwreg,ern,em2reg,ewreg, //inputs
73.                    aluc,wreg,m2reg,wmem,aluimm,fwda,fwdb,wpcir,regrt);//outputs
74.     ControlMux ctrmux(rd,rt,regrt,rn); //ControlMux: Selects whether to write rd/rt into RegWr
    ite, outputs to IDEXE
75.     SignExtend extender(imm, immExt); //SignExtend: input is 16bit nonextended from instructio
    n memory, output 32bit extended value.
76.
77.     ForwardAMux fwdamux(fwda,qa,maluOut,mqb,dmOut,fwdaOut); //inputs: fwdb,qb,maluOut,mqb,dmOu
    t outputs: fwda_out
78.     ForwardBMux fwdbmux(fwdb,qb,maluOut,mqb,dmOut,fwdbOut); //inputs: fwdb,qb,maluOut,mqb,dmOu
    t outputs: fwdb_out
79.     //IDEXE REGISTER
80.     IDEXE idexe(clk,wreg,m2reg,wmem,aluc,aluimm,rn,qa,qb,immExt,//inputs
81.                 ewreg,em2reg,ewmem,ealuc,ealuimm,ern,eqa,eqb,eimmExt);//outputs
82.
83.     //EXE STAGE
84.     ALUMux alumux(eqb,eimmExt,ealuimm,muxImm);//ALUMux: selects qb reg output or immediate val
    ue
85.     ALU alu(eqa, muxImm, ealuc, aluOut);//ALU: executes operations for CPU using aluc control,
     a,b inputs
86.
87.     //EXEMEM REGISTER
88.     EXEMEM exemem(clk,ewreg,em2reg,ewmem,ern,aluOut,eqb,//inputs
89.                   mwreg,mm2reg,mwmem,mrn,maluOut,mqb);//outputs
90.
91.     //MEM STAGE
92.     DataMemory datamem(maluOut,mqb,mwmem,dmOut); //DataMemory: data memory handler, inputs fro
    m EXEMEM, out to dmOut
93.
94.     //MEMWB REGISTER
95.     MEMWB memwb(clk,mwreg,mm2reg,mrn,maluOut,dmOut,//inputs
96.                 wwreg,wm2reg,wrn,waluOut,wdmOut); //ouputs
97.
98.     //WB STAGE
99.     WBMux wbmux(waluOut, wdmOut, wm2reg, wbMuxOut); //WBMux: selects what to write back into r
    egfile
100.          RegFile regfile(clk,rs,rt,qa,qb,wrn,wbMuxOut); //RegFile: generates a 32x32 registe
    r file to read/write from
101.
102.          initial begin //Clock signal loop.
103.              clk = 0;
104.          end
105.          always begin
106.              #5 clk = !clk;
107.          end
108.      endmodule
```

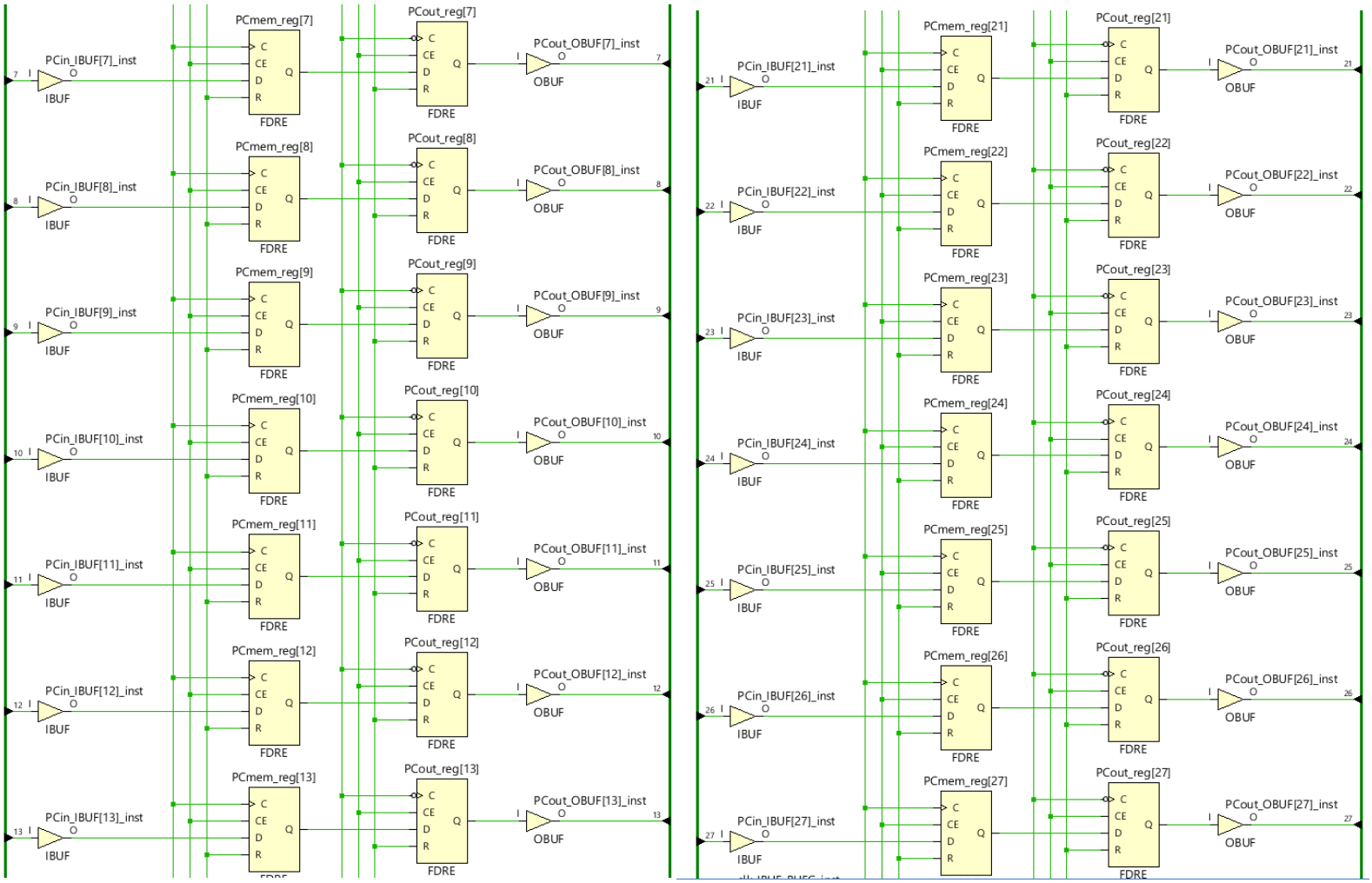
Waveforms:


Name
Value
clk
pcin[31:0]
pcout[31:0]
do[31:0]
op[5:0]
func[5:0]
rd[4:0]
rs[4:0]
rt[4:0]
imm[15:0]
aluc[3:0]
wreg
m2reg
wmem
aluimm
regrt
rn[4:0]
qa[31:0]
qb[31:0]
immExt[31:0]
ealuc[3:0]
ewreg
em2reg
ewmem
ealuimm
ern[4:0]
eqa[31:0]
eqb[31:0]
eimmExt[31:0]
muxImm[31:0]
aluOut[31:0]
mwreg
mm2reg
mwmem
mrn[4:0]
maluOut[31:0]
mqb[31:0]
dmOut[31:0]
wwreg
wm2reg
wrn[4:0]
waluOut[31:0]
wdmOut[31:0]
wbMuxOut[31:0]
fwda[1:0]
fwdaOut[31:0]
fwdb[1:0]
fwdbOut[31:0]
wpcir

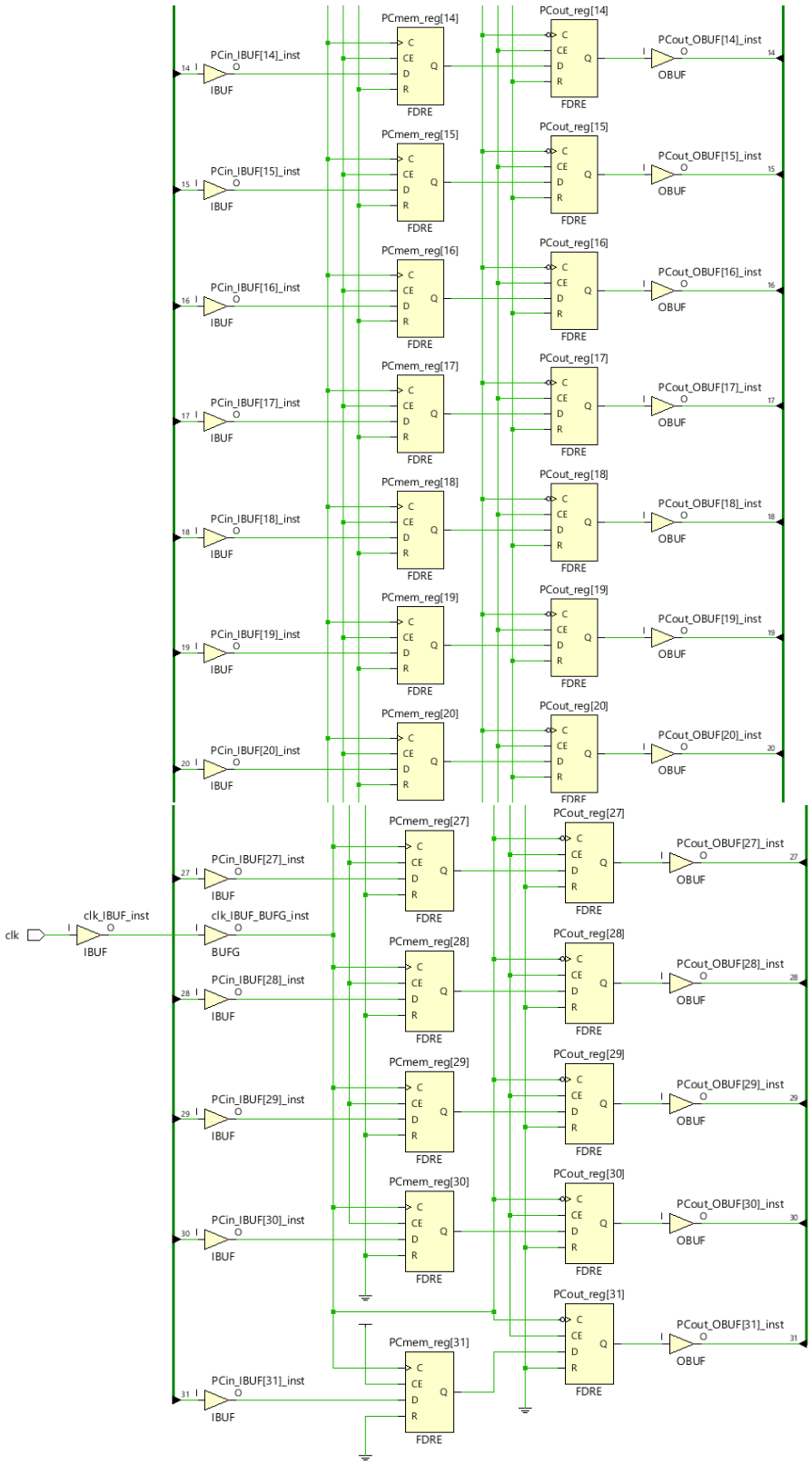
Design Schematic:



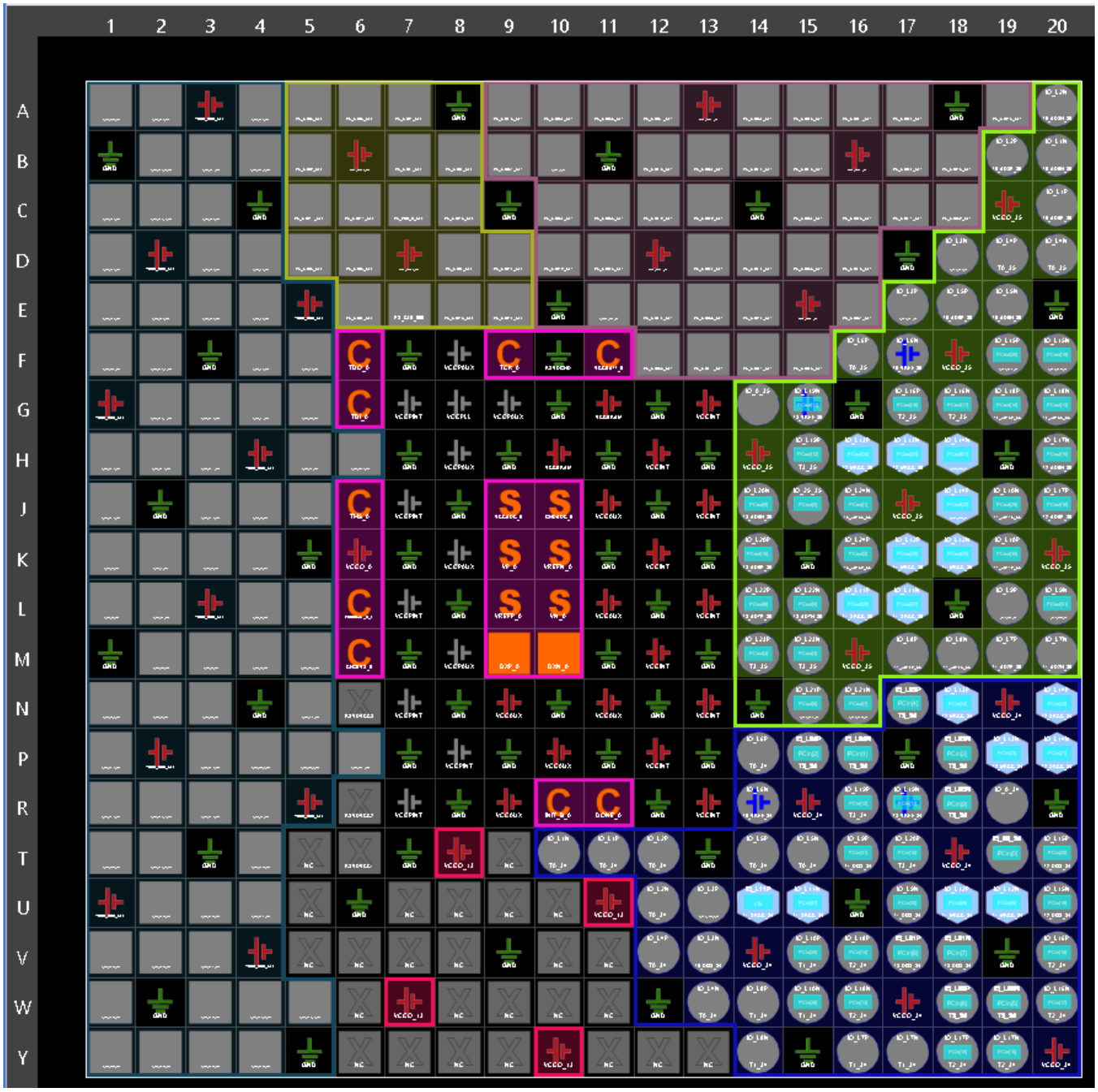
Design Schematic (contd):



Design Schematic (contd):



IO Planning:


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
A
B
C
D
E
F
G
H
J
K
L
M
N
P
R
T
U
V
W
Y

Floor Planning: