

Lab 5

Computer organization and design

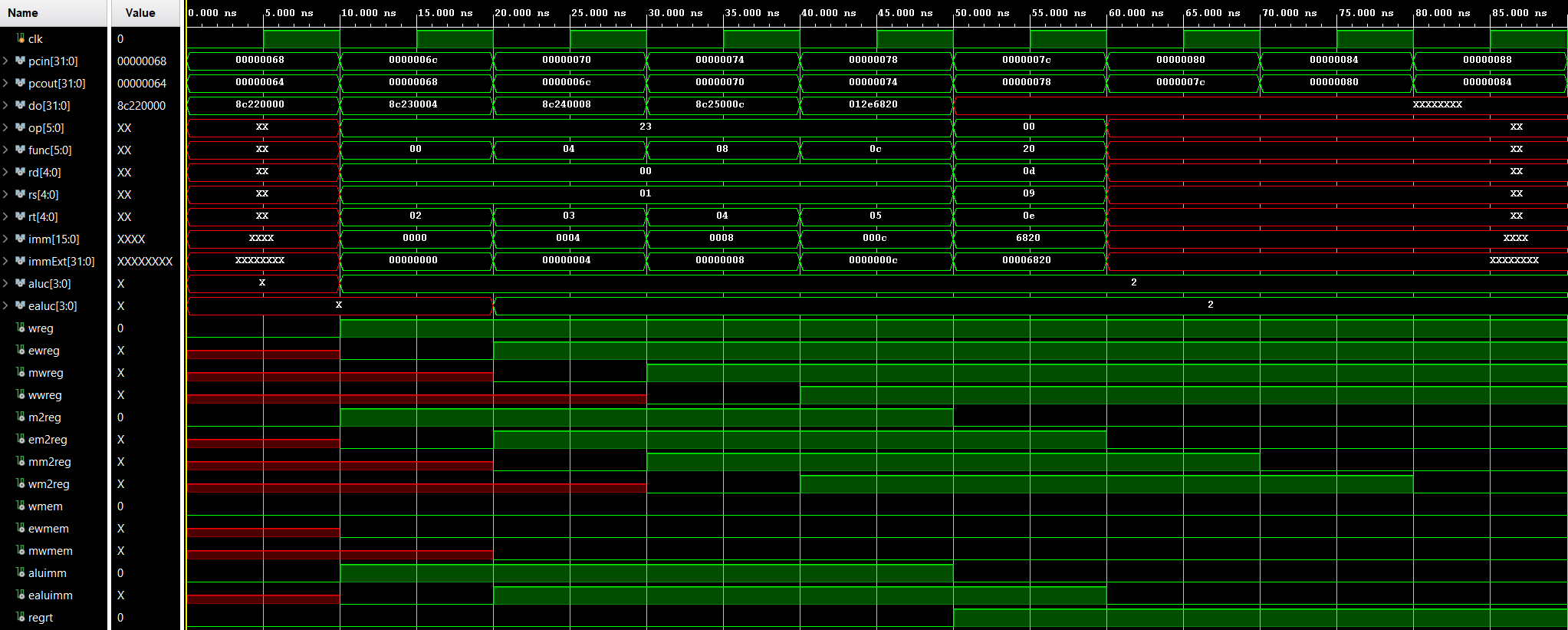
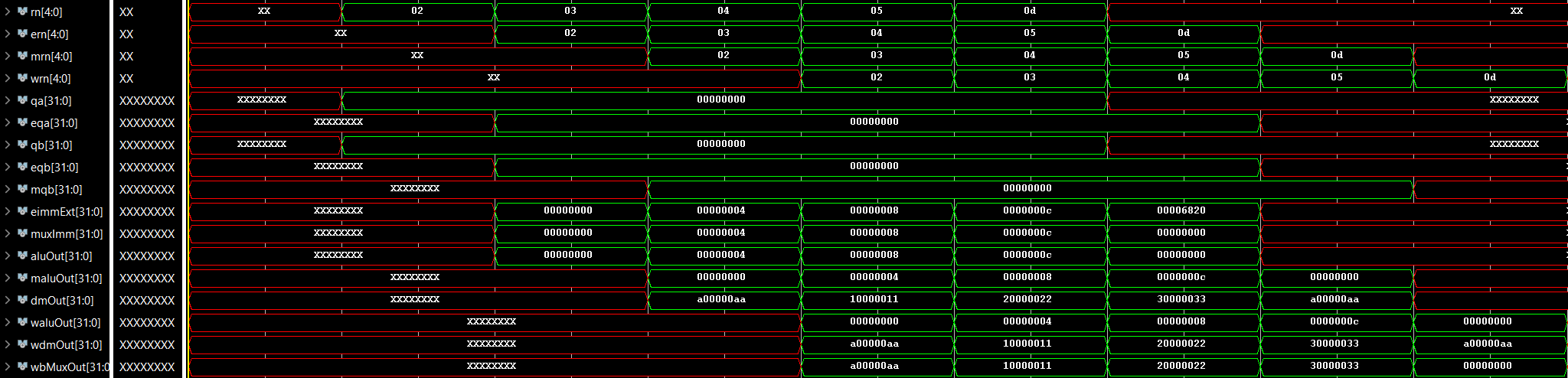
Taylan Unal | CMPEN 331 Section 1 | 10/17/19

**Top Module (CPU):**

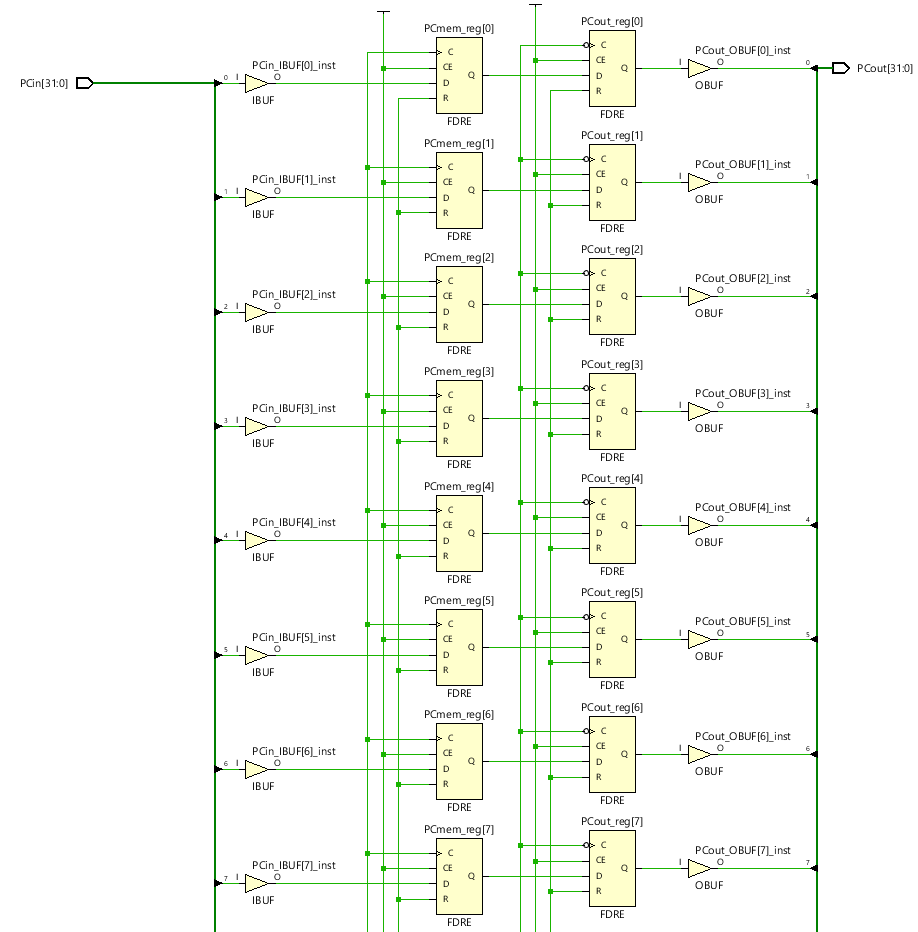
1. //Taylan Unal CMPEN 331.001 LAB5 Top Module
2. `timescale 1ns / 1ps
3. module PC(clk, PCin, PCout);
4. input clk;
5. input [31:0] PCin;
6. reg[31:0] PCmem;
7. output reg [31:0] PCout;
8. initial begin
9. PCout = 0;
10. PCmem = 100;
11. end
13. always @(posedge clk) begin
14. PCmem <= PCin; //register output is saved as its input at clk cycle
15. end
16. always @(negedge clk) begin
17. PCout <= PCmem; //register output is assigned its input at clk cycle
18. end
19. endmodule
21. module PCAdder(PCin, PCout);
22. input [31:0] PCin;
23. output reg [31:0] PCout;
25. always @(PCin) begin
26. PCout <= PCin + 4;
27. end
28. endmodule
30. module InstMem(pcin, **do**); //a is 'PC input', do is 'IM output'
31. input [31:0] pcin;
32. output reg [31:0] **do**;
33. reg [31:0] IM[0:511]; //load word from memory, leaving room for more instructions later.
34. initial begin
35. IM[32'd100] = 32'h8c220000; //lw $v0 00($at)
36. IM[32'd104] = 32'h8c230004; //lw $v1 04($at)
37. IM[32'd108] = 32'h8c240008; //lw $a0 08($at)
38. IM[32'd112] = 32'h8c25000c; //lw $a1 12($at)
39. IM[32'd116] = 32'h012E6820; //add $t5,$1,$t6 (adding $6,$2,$10)
40. //IM[32'd116] = 32'h004a3020; //add $a2,$v0,$t2 (Jerome)
41. end
43. always @ (pcin) begin //good
44. **do** <= IM[pcin];
45. end
46. endmodule
48. module IFID(clk, **do**, op, rd, rs, rt, func, imm);
49. input clk;
50. input [31:0] **do**;//instruction input
51. output reg [5:0] op,func;//I-type values
52. output reg [5:0] rd,rs,rt;//I-type values
53. output reg [15:0] imm;//immediate value
54. reg [31:0] IF; //IFID  temporarymemory
55. always @ (posedge clk) begin
56. IF <= **do**; //save value of input until negedge
57. end
59. always @(negedge clk)
60. begin
61. op <= IF[31:26];
62. rs <= IF[25:21];
63. rt <= IF[20:16];
64. rd <= IF[15:11];
65. func <= IF[5:0];
66. imm <= IF[15:0]; //hmmm
67. end
68. endmodule
70. module ControlUnit(op,func,aluc,wreg,m2reg,wmem,aluimm,regrt);
71. input [5:0] op,func;
72. output reg [3:0] aluc;
73. output reg wreg, m2reg, wmem, aluimm, regrt;
75. initial begin
76. wreg <= 0; //RegWrite
77. m2reg <= 0; //Mem2Reg
78. wmem <= 0; //Write Memory
79. aluimm <= 0; //ALU source
80. regrt <= 0; //Reg Destination
81. end
82. always @ (op,func) begin //adding func
83. **if**(op == 0) begin //R-Type instruction
84. **case**(func)
85. 32: aluc <= 2; //Register ADD
86. 34: aluc <= 6; //Register SUB
87. 36: aluc <= 0; //Register AND
88. 37: aluc <= 1; //Register OR
89. 39: aluc <= 12;//Register XOR
90. endcase
91. wreg <= 1;//RegWrite
92. m2reg <= 0;//Mem2Reg
93. wmem <= 0;//WriteMem
94. aluimm <= 0; //AluSrc
95. regrt <= 1; //input to ControlMux
96. end
97. **if**(op == 6'b100011) begin //Load Word (LW) I-type
98. aluc <= 2;//add
99. wreg <= 1;//RegWrite
100. m2reg <= 1;//Mem2Reg
101. wmem <= 0;//WriteMem
102. aluimm <= 1;//AluSrc
103. regrt <= 0; //input to ControlMux
104. end
105. end
106. endmodule
108. module ControlMux(rd,rt,regrt,rn);
109. input [4:0] rd,rt; //output from IFIDReg
110. input regrt; //from controlunit, enable write
111. output reg [4:0] rn; //output
113. always @(regrt,rt,rd) begin
114. **case** (regrt)
115. 0:
116. rn <= rt;
117. 1:
118. rn <= rd;
119. endcase
120. end
121. endmodule
123. module SignExtend(immIn, immOut);//input IF instruct value, output an extended 32 bit value
124. input [15:0] immIn; //short (nonextended) value
125. output reg [31:0] immOut; //extended value
127. always @(immIn) begin
128. immOut <= {{16{immIn[15]}},immIn[15:0]}; //extends 16bit number to 32bits.
129. end
130. endmodule
132. module IDEXE(clk, wreg, m2reg, wmem, aluc, aluimm, rn, qa, qb, imm,
133. ewreg, em2reg, ewmem, ealuc, ealuimm, ern, eqa, eqb, eimm);
134. input clk;
135. input wreg, m2reg, wmem, aluimm; //input to IDEXE
136. input [3:0] aluc; //output from control unit
137. input [4:0] rn; //output from mux into IDEXE
138. input [31:0] qa, qb; //output from regfile
139. input [31:0] imm; //extended immediate value
141. //Use these to store values for later assignment using posedge, negedge.
142. reg wreg2, m2reg2, wmem2, aluimm2;
143. reg [4:0] rn2;
144. reg [3:0] aluc2;
145. reg [31:0] qa2, qb2;
146. reg [31:0] imm2;
148. output reg ewreg, em2reg, ewmem, ealuimm; //extended outputs from control unit
149. output reg [3:0] ealuc; //extended outputs from control unit, into ALU
150. output reg [4:0] ern; //extended outputs from mux
151. output reg [31:0] eqa, eqb; //extended outputs from regfile
152. output reg [31:0] eimm; //extended outputs from sign extender
154. always@(posedge clk) begin //pass values into middle save values.(save regs<=input)
155. wreg2 <= wreg;
156. m2reg2 <= m2reg;
157. wmem2 <= wmem;
158. aluimm2 <= aluimm;
159. rn2 <= rn;
160. aluc2 <= aluc;
161. qa2 <= qa;
162. qb2 <= qb;
163. imm2 <= imm;
164. end
166. always@(negedge clk) begin //output values from saved values. (output <= save regs)
167. ewreg <= wreg2;
168. em2reg <= m2reg2;
169. ewmem <= wmem2;
170. ealuimm <= aluimm2;
171. ern <= rn2;
172. ealuc <= aluc2;
173. eqa <= qa2;
174. eqb <= qb2;
175. eimm <= imm2;
176. end
177. endmodule
179. module ALUMux(eqb, eimmExt, ealuimm, muxImm);
180. input [31:0] eqb, eimmExt; //qb value and immExt from IDEXE
181. input ealuimm;//selector from IDEXE
182. output reg [31:0] muxImm; //output of ALUMux
184. always @ (eqb, eimmExt) begin
185. **case** (ealuimm)
186. 0: muxImm = eqb;
187. 1: muxImm = eimmExt;
188. endcase
189. end
190. endmodule
192. module ALU (eqa, eqb, ealuc, aluOut);
193. input [31:0] eqa, eqb;
194. input [3:0] ealuc; //4bit number
196. output reg [31:0] aluOut;
198. always @ (eqa, eqb) begin
199. **case** (ealuc)
200. 4'b0010: aluOut <= eqa + eqb;
201. endcase
202. end
203. endmodule
205. module EXEMEM (clock, ewreg, em2reg, ewmem, ern, aluOut, eqb,
206. mwreg, mm2reg, mwmem, mrn, maluOut, mqb);
207. input clock, ewreg, em2reg, ewmem;
208. input [4:0] ern;
209. input [31:0] aluOut, eqb;
211. //Temporary store values for posedge/negedge
212. reg ewreg2, em2reg2, ewmem2;
213. reg [4:0] ern2;
214. reg [31:0] aluOut2, eqb2;
216. output reg mwreg, mm2reg, mwmem; //outputs on memory side of register
217. output reg [4:0] mrn;
218. output reg [31:0] maluOut, mqb;
220. always @ (posedge clock) begin //pass values into middle save (save regs <= input)
221. ewreg2 <= ewreg;
222. em2reg2 <= em2reg;
223. ewmem2 <= ewmem;
224. ern2 <= ern;
225. aluOut2 <= aluOut;
226. eqb2 <= eqb;
227. end
229. always @ (negedge clock) begin
230. mwreg <= ewreg2;
231. mm2reg <= em2reg;
232. mwmem <= ewmem;
233. mrn <= ern2;
234. maluOut <= aluOut2;
235. mqb <= eqb2;
236. end
237. endmodule
239. module DataMemory (maluOut, mqb, mwmem, dmOut);
240. input [31:0] maluOut, mqb; //input alu output value, qb value
241. input mwmem;//write memory
242. output reg [31:0] dmOut;
244. reg [31:0] DM [0:36];
246. initial begin //set first 10 words to data memory
247. DM[32'd0] = 32'hA00000AA;
248. DM[32'd4] = 32'h10000011;
249. DM[32'd8] = 32'h20000022;
250. DM[32'd12] = 32'h30000033;
251. DM[32'd16] = 32'h40000044;
252. DM[32'd20] = 32'h50000055;
253. DM[32'd24] = 32'h60000066;
254. DM[32'd28] = 32'h70000077;
255. DM[32'd32] = 32'h80000088;
256. DM[32'd36] = 32'h90000099;
257. end
259. always @ (maluOut, mqb) begin
260. **case** (mwmem)
261. 1'b0: dmOut <= DM[maluOut];
262. 1'b1: dmOut <= DM[mqb];
263. endcase
264. end
265. endmodule
267. module MEMWB(clk, mwreg, mm2reg, mrn, maluOut, dmOut,
268. wwreg, wm2reg, wrn, waluOut, wdmOut);//adding WB outputs
269. input clk, mwreg, mm2reg;
270. input [31:0] maluOut, dmOut;
271. input [4:0] mrn;
273. //Temporary store values between posedge/negedge
274. reg mwreg2, mm2reg2;
275. reg [31:0] maluOut2, dmOut2;
276. reg [4:0] mrn2;
278. output reg wwreg, wm2reg;
279. output reg [31:0] waluOut, wdmOut;
280. output reg [4:0] wrn;
282. always @(posedge clk) begin //stores into temporary regs
283. mwreg2 <= mwreg;
284. mm2reg2 <= mm2reg;
285. maluOut2 <= maluOut;
286. dmOut2 <= dmOut;
287. mrn2 <= mrn;
288. end
289. always @(negedge clk) begin
290. wwreg <= mwreg2;
291. wm2reg <= mm2reg2;
292. waluOut <= maluOut2;
293. wdmOut <= dmOut2;
294. wrn <= mrn2;
295. end
296. endmodule
298. module WBMux(waluOut, wdmOut, wm2reg, wbMuxOut);
299. input [31:0] waluOut, wdmOut;//takes in aluresult and data memory address
300. input wm2reg; //write enable memory to register
301. output reg [31:0] wbMuxOut; //output to regfile
303. always @(waluOut, wdmOut, wm2reg) begin
304. **case**(wm2reg)
305. 0: wbMuxOut <= waluOut; //if wm2reg=0, output alu value
306. 1: wbMuxOut <= wdmOut; //if wm2reg=1, output datamem value.
307. endcase
308. end
309. endmodule
311. module RegFile(clk, we, rs, rt, qa, qb, wrn, wbmuxOut); //(clk,rs,rt,qa,qb,wrn,wbmuxOut)
312. input clk;
313. input we; //write enable input (wwreg
314. input [31:0] wbmuxOut; //data to write to register
315. input [4:0] rs, rt, wrn; //rs val in, rt val in, rd val in
316. reg [31:0] regs [0:31]; //32 x 32 register file. Store all the registers.
318. output reg [31:0] qa, qb; //qa->rt out, qb->rs
320. initial begin //initialize all 32 registers to 0.
321. {regs[0], regs[1], regs[2], regs[3], regs[4], regs[5], regs[6], regs[7],
322. regs[8], regs[9], regs[10], regs[11], regs[12], regs[13], regs[14], regs[15],
323. regs[16], regs[17], regs[18], regs[19], regs[20], regs[21], regs[22], regs[23],
324. regs[24], regs[25], regs[26], regs[27], regs[28], regs[29], regs[30], regs[31]} = 0;//32'h00000000;
325. end
327. always @(posedge clk) begin //write during first half of cycle
328. **if**(wrn != 0) begin//write operation
329. regs[wrn] <= wbmuxOut;
330. end
331. end
333. always @(rs, rt) begin//read during second half of cycle
334. qa <= regs[rs]; //register val in register rs
335. qb <= regs[rt]; //register val in register rt
336. end
337. endmodule

**Testbench Module:**

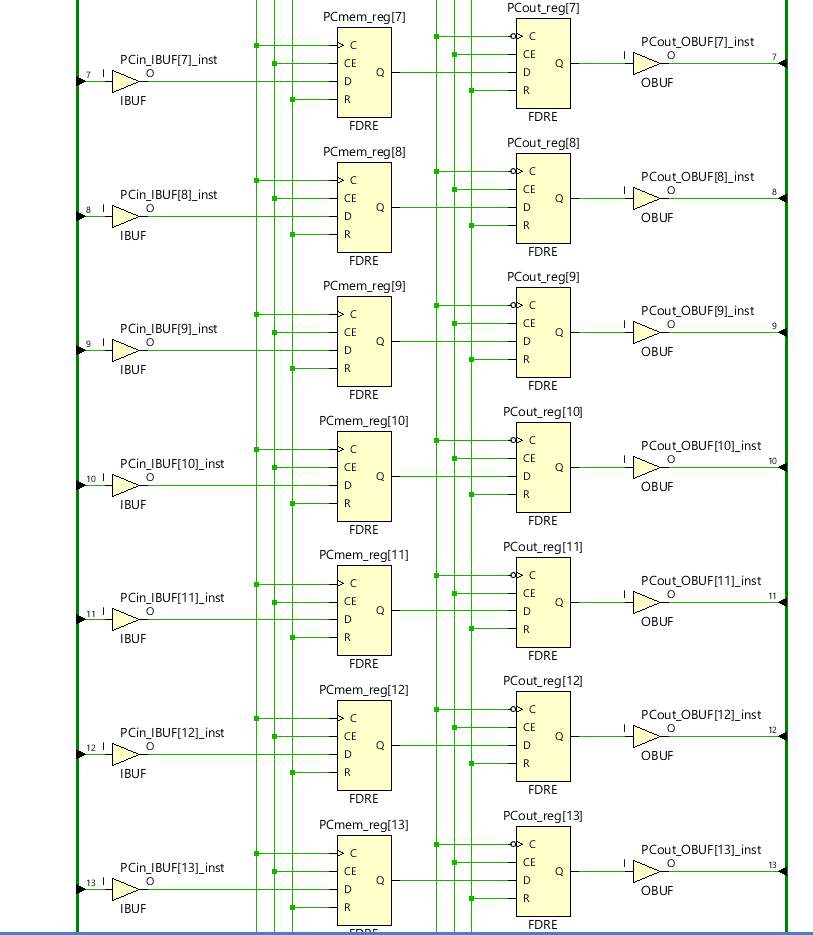
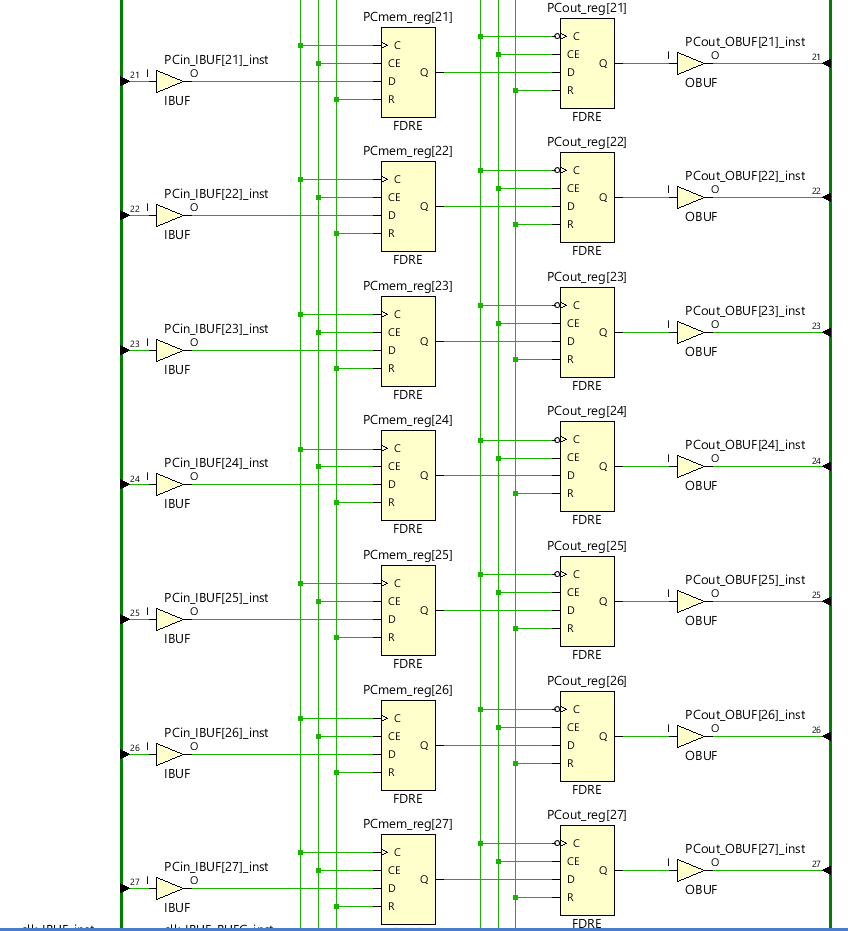
1. //Taylan Unal CMPEN 331.001 LAB5 Testbench
2. `timescale 1ns / 1ps
4. module CPU\_Test();
5. //WIRES ARE OUTPUTS, REGS ARE INPUTS.
6. reg clk; //global clock signal
7. //IF STAGE
8. wire [31:0] pcin; //PC output to PC register
9. wire [31:0] pcout; //PC output to adder
10. wire [31:0] **do**; //instruction to be executed (im\_out)
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)
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
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
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
55. //////////////////////////////////////////////////////////////////////////////////////
56. //START Testbench
57. //IF STAGE
58. PC PC(clk, pcin, pcout);//PCRegister: input clk, PCinput, outputs result from PCAdder
59. PCAdder adder(pcout, pcin);//PCAdder: input PCOutput, increments by 4, outputs to input or
60. InstMem instmem(pcout,**do**);//InstMem: has instructions, outputs instruction from PCRegister
62. //IFID REGISTER
63. IFID ifid(clk,**do**,op,rd,rs,rt,func,imm);
65. //ID STAGE
66. ControlUnit CU(op,func,aluc,wreg,m2reg,wmem,aluimm,regrt); //ControlUnit: inputs op,func fields, outputs aluc,wreg,m2reg,wmem,aluimm into IDEXEReg and regrt into RegisterMux
67. ControlMux ctrmux(rd,rt,regrt,rn); //ControlMux: Selects whether to write rd/rt into RegWrite, outputs to IDEXE
68. SignExtend extender(imm, immExt); //SignExtend: input is 16bit nonextended from instruction memory, output 32bit extended value.
70. //IDEXE REGISTER
71. IDEXE idexe(clk,wreg,m2reg,wmem,aluc,aluimm,rn,qa,qb,immExt,//inputs
72. ewreg,em2reg,ewmem,ealuc,ealuimm,ern,eqa,eqb,eimmExt);//outputs
74. //EXE STAGE
75. ALUMux alumux(eqb,eimmExt,ealuimm,muxImm);//ALUMux: selects qb reg output or immediate value
76. ALU alu(eqa, muxImm, ealuc, aluOut);//ALU: executes operations for CPU using aluc control, a,b inputs
78. //EXEMEM REGISTER
79. EXEMEM exemem(clk,ewreg,em2reg,ewmem,ern,aluOut,eqb,//inputs
80. mwreg,mm2reg,mwmem,mrn,maluOut,mqb);//outputs
82. //MEM STAGE
83. DataMemory datamem(maluOut,mqb,mwmem,dmOut); //DataMemory: data memory handler, inputs from EXEMEM, out to dmOut
85. //MEMWB REGISTER
86. MEMWB memwb(clk,mwreg,mm2reg,mrn,maluOut,dmOut,//inputs
87. wwreg,wm2reg,wrn,waluOut,wdmOut); //ouputs
89. //WB STAGE
90. WBMux wbmux(waluOut, wdmOut, wm2reg, wbMuxOut); //WBMux: selects what to write back into regfile
91. RegFile regfile(clk,rs,rt,qa,qb,wrn,wbMuxOut); //RegFile: generates a 32x32 register file to read/write from
93. initial begin //Clock signal loop.
94. clk = 0;
95. end
96. always begin
97. #5 clk = !clk;
98. end
99. endmodule

**Waveforms:**

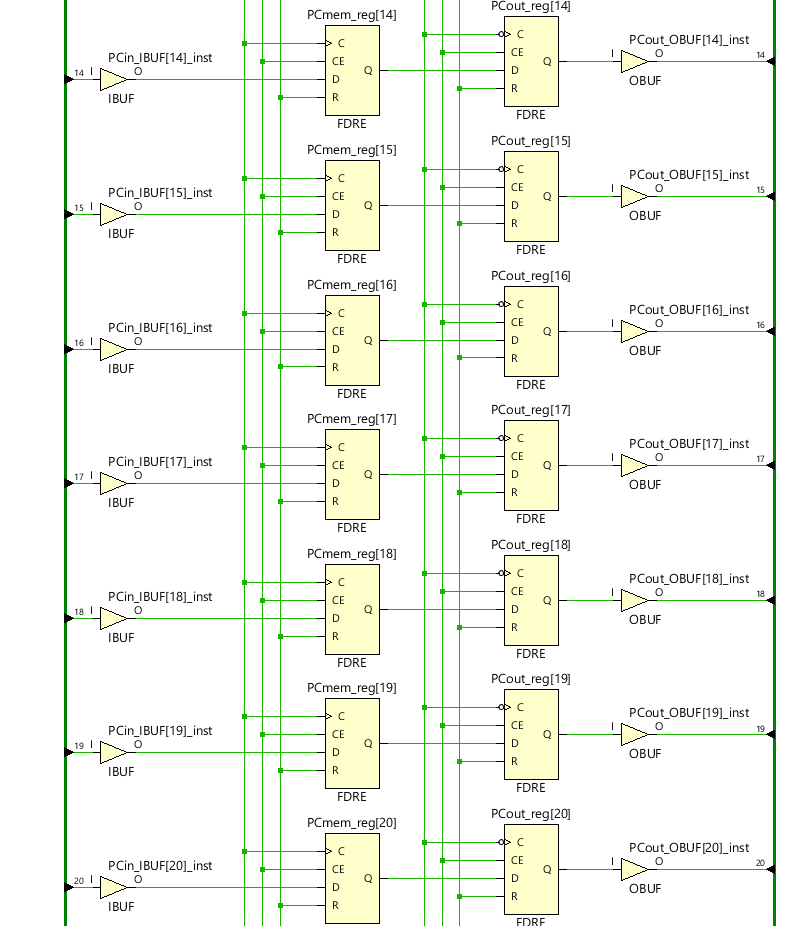
**Design Schematic:**

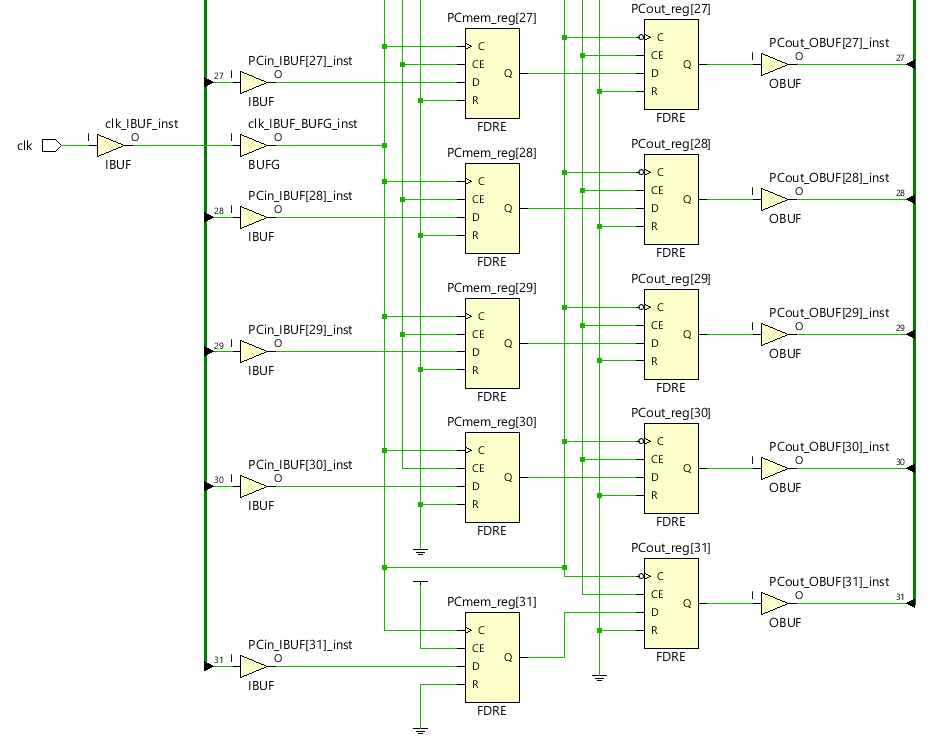
****

**Design Schematic (contd):**

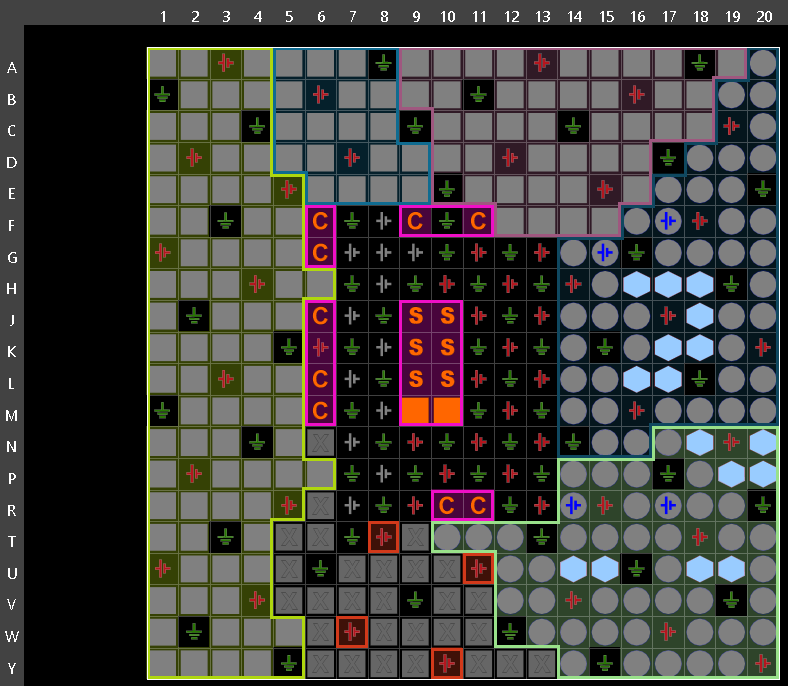
****

**Design Schematic (contd):**

****

****

**IO Planning:**



**Floor Planning:**

