

Lab 4

Computer organization and design

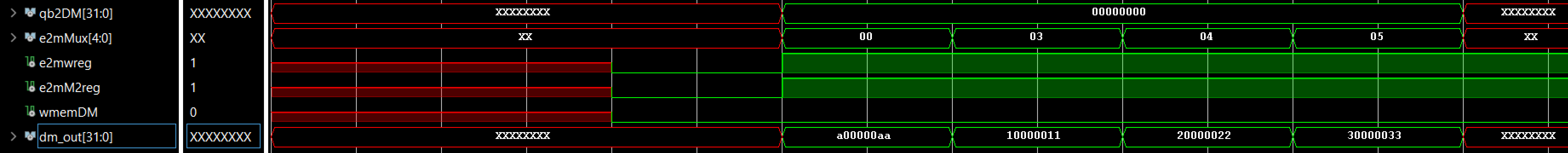
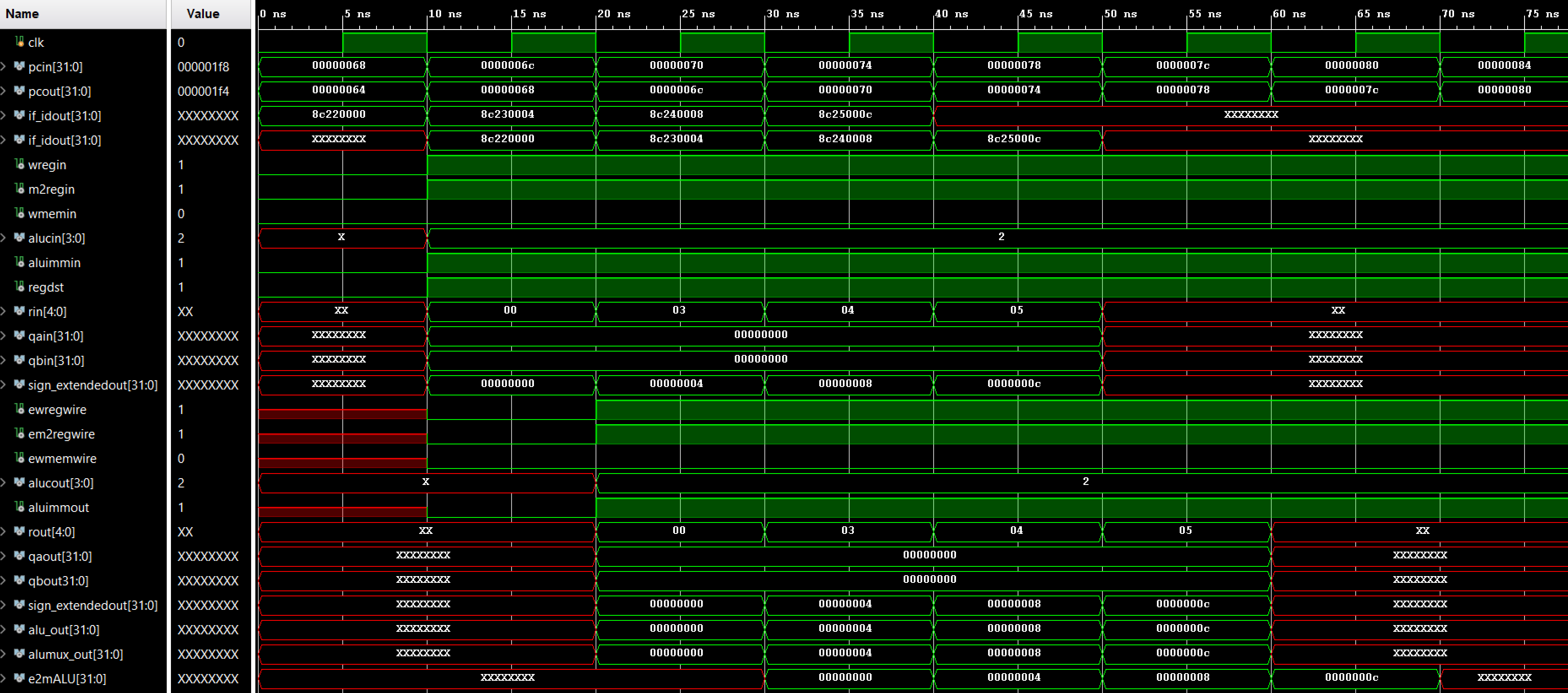
Taylan Unal | CMPEN 331 Section 1 | 10/17/19

**Top Module (CPU):**

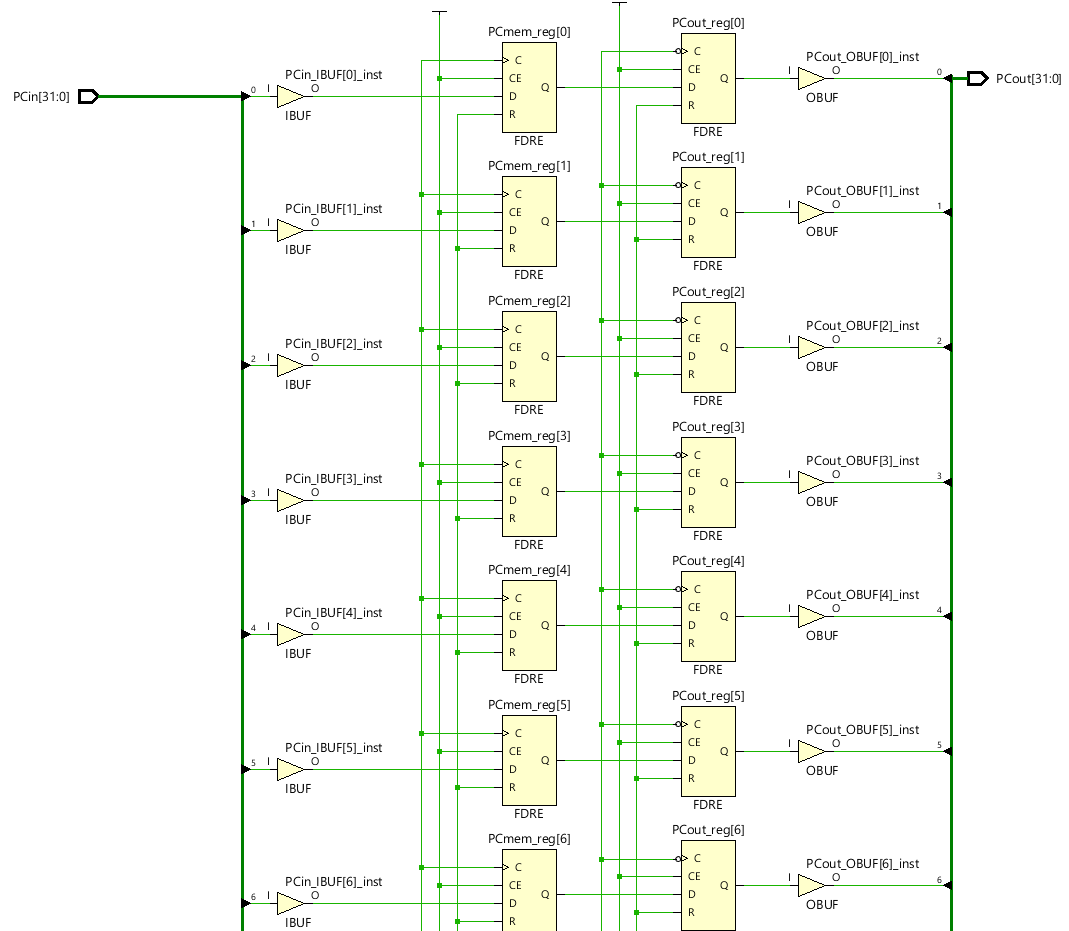
1. //Taylan Unal CMPEN 331.001 LAB4 Top Module
2. `timescale 1ns / 1ps
3. module PC(clk, PCin, PCout);
4. input clk;
5. input [31:0] PCin;
6. reg[31:0] PCmem;
7. output reg [31:0] PCout;
8. initial begin
9. PCout = 0;
10. PCmem = 100;
11. end
13. always @(posedge clk) begin
14. PCmem <= PCin; //register 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 InstructMem(a, **do**); //a is 'PC input', do is 'IM output'
31. input [31:0] a;
32. output reg [31:0] **do**;
33. reg [31:0] IM[0:511];//load word from memory, leaving room forinstructions 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. end
41. always @ (a) begin
42. **do** <= IM[a];
43. end
44. endmodule
46. module IFID(clk, instIn, instOut);//define instruction input and instruction from IM
47. input clk;
48. input [31:0] instIn;
49. reg [31:0] IF; //IF memory
50. output reg [31:0] instOut; //defines the R-type
52. always @ (posedge clk) begin
53. IF <= instIn; //save value of input until negedge.
54. end
55. always @ (negedge clk) begin
56. instOut <= IF; //pull from stored value
57. end
58. endmodule
59. module ControlUnit(inst\_CU, wreg, m2reg, wmem, aluc, aluimm, regrt);
60. input [31:0] inst\_CU; //ifid\_out. Includes op, func.
61. wire [5:0] op, func;
62. output reg wreg, m2reg, wmem, aluimm, regrt;
63. output reg [3:0] aluc;
64. assign op = inst\_CU[31:26];
66. initial begin
67. wreg <= 0; //RegWrite
68. m2reg <= 0; //Mem2Reg
69. wmem <= 0; //Write Memory
70. aluimm <= 0; //ALU source
71. regrt <= 0; //Reg Destination
72. end
74. always @ (op) begin
75. **if**(op == 6'b100011) begin //Load Word (LW)
76. wreg <= 1'b1;
77. m2reg <= 1'b1;
78. wmem <= 1'b0;
79. aluimm <= 1'b1;
80. regrt <= 1'b1; //important part
81. aluc <= 4'b0010;
82. end
83. end
84. endmodule
86. module ControlMux(inst\_Mux, regrt, rd\_rt);
87. input [31:0] inst\_Mux;
88. input regrt;
89. output reg [4:0] rd\_rt;
90. wire [4:0] rd, rt;
92. assign rd = inst\_Mux[15:11];
93. assign rt = inst\_Mux[20:16];
95. always @(rd, rt) begin
96. **case** (regrt)
97. 1'b1:
98. rd\_rt = rt;
99. 1'b0:
100. rd\_rt = rd;
101. endcase
102. end
103. endmodule
105. module RegFile(inst\_RF, qa, qb); //if\_idout is inst\_RF, qa is RS out, qb is RT out
106. input [31:0] inst\_RF;
107. reg [31:0] regs [0:31]; //32 x 32 register file. Store all the registers.
108. wire [4:0] rs, rt;
109. output reg [31:0] qa, qb;
111. assign rs = inst\_RF[25:21];
112. assign rt = inst\_RF[20:16];
114. initial begin //initialize all 32 registers to 0.
115. {regs[0],regs[1],regs[2],regs[3],regs[4],regs[5],regs[6],regs[7],
116. regs[8],regs[9],regs[10],regs[11],regs[12],regs[13],regs[14],regs[15],
117. regs[16],regs[17],regs[18],regs[19],regs[20],regs[21],regs[22],regs[23],
118. regs[24],regs[25],regs[26],regs[27],regs[28],regs[29],regs[30],regs[31]} = 0;
119. end
120. always @(rs, rt) begin
121. qa <= regs[rs]; //register output 1 = val in register rs
122. qb <= regs[rt]; //register output 2 = val in register rt
123. end
124. endmodule
126. module SignExtend(inst\_IF, immOut);//input IF instruct value, output an extended 32 bit value
127. input [31:0] inst\_IF; //short value
128. output reg [31:0] immOut; //extended value
129. wire [15:0] imm; //save wire for main values
131. assign imm = inst\_IF[15:0]; //save first 16 bits.
132. always @(imm) begin
133. immOut = {{16{imm[15]}},imm}; //extends 16bit number to 32bits.
134. end
135. endmodule
137. module IDEXE(clk, wreg, m2reg, wmem, aluc, aluimm, mux, qa, qb, extend,
138. ewreg, em2reg, ewmem, ealuc, ealuimm, emux, eqa, eqb, eextend);
139. input clk;
140. input wreg, m2reg, wmem, aluimm; //input to IDEXE
141. input [4:0] mux; //output from mux into IDEXE
142. input [3:0] aluc; //output from control unit
143. input [31:0] extend, qa, qb; //output from regfile
145. //Use these to store values for later assignment using posedge, negedge.
146. reg wreg2, m2reg2, wmem2, aluimm2;
147. reg [4:0] mux2;
148. reg [3:0] aluc2;
149. reg [31:0] qa2, qb2;
150. reg [31:0] extend2;
152. output reg ewreg, em2reg, ewmem, ealuimm; //extended outputs from control unit
153. output reg [3:0] ealuc; //extended outputs from control unit, into ALU
154. output reg [4:0] emux; //extended outputs from multiplexer
155. output reg [31:0] eqa, eqb; //extended outputs from regfile
156. output reg [31:0] eextend; //extended outputs from sign extender
158. always@(posedge clk) begin //pass values into middle save values. (save regs <= input)
159. wreg2 <= wreg;
160. m2reg2 <= m2reg;
161. wmem2 <= wmem;
162. aluimm2 <= aluimm;
163. mux2 <= mux;
164. aluc2 <= aluc;
165. qa2 <= qa;
166. qb2 <= qb;
167. extend2 <= extend;
168. end
170. always@(negedge clk) begin //output values from saved values. (output <= save regs)
171. ewreg <= wreg2;
172. em2reg <= m2reg2;
173. ewmem <= wmem2;
174. ealuimm <= aluimm2;
175. emux <= mux2;
176. ealuc <= aluc2;
177. eqa <= qa2;
178. eqb <= qb2;
179. eextend <= extend2;
180. end
181. endmodule
183. module ALUMux(ealuimm, eqb, eextend, alumux\_out);
184. input ealuimm;
185. input [31:0] eqb, eextend; //qb value and immExtended
187. output reg [31:0] alumux\_out;
189. always @ (eqb, eextend) begin
190. **case** (ealuimm)
191. 1'b0: alumux\_out = eqb;
192. 1'b1: alumux\_out = eextend;
193. endcase
194. end
195. endmodule
197. module ALU (ALUcontrol, eqa , eqb, alu\_out);
198. input [3:0] ALUcontrol; //4bit number
199. input [31:0] eqa, eqb;
200. output reg [31:0] alu\_out;
202. always @ (eqa, eqb) begin
203. **case** (ALUcontrol)
204. 4'b0010: alu\_out <= eqa + eqb;
205. endcase
206. end
207. endmodule
209. module EXEMEM (clock, eWREG, eM2REG, eWMEM, eMUX, eALU\_OUT, eQB, eWREG\_out,
210. eM2REG\_out, eWMEM\_out, eMUX\_out, eALU\_out, eQB\_out);
211. input clock, eWREG, eM2REG, eWMEM;
212. input [4:0] eMUX;
213. input [31:0] eQB, eALU\_OUT;
215. reg ewreg, em2reg, ewmem;
216. reg [4:0] emux;
217. reg [31:0] ealu\_out, eqb;
219. output reg eWREG\_out, eM2REG\_out, eWMEM\_out;
220. output reg [4:0] eMUX\_out;
221. output reg [31:0] eALU\_out, eQB\_out;
223. always @ (posedge clock) begin
224. ewreg <= eWREG;
225. em2reg <= eM2REG;
226. ewmem <= eWMEM;
227. emux <= eMUX;
228. ealu\_out <= eALU\_OUT;
229. eqb <= eQB;
230. end
232. always @ (negedge clock) begin
233. eWREG\_out <= ewreg;
234. eM2REG\_out <= em2reg;
235. eWMEM\_out <= ewmem;
236. eMUX\_out <= emux;
237. eALU\_out <= ealu\_out;
238. eQB\_out <= eqb;
239. end
240. endmodule
242. module DataMemory (MEMWRITE, ALUDATA\_IN ,QBDATA\_IN, DMDATA\_OUT);
243. input MEMWRITE;
244. input [31:0] ALUDATA\_IN, QBDATA\_IN;
246. reg [31:0] DM [0:36];
248. output reg [31:0] DMDATA\_OUT;
250. initial begin //set first 10 words to data memory
251. DM[32'd0] = 32'hA00000AA;
252. DM[32'd4] = 32'h10000011;
253. DM[32'd8] = 32'h20000022;
254. DM[32'd12] = 32'h30000033;
255. DM[32'd16] = 32'h40000044;
256. DM[32'd20] = 32'h50000055;
257. DM[32'd24] = 32'h60000066;
258. DM[32'd28] = 32'h70000077;
259. DM[32'd32] = 32'h80000088;
260. DM[32'd36] = 32'h90000099;
261. end
263. always @ (ALUDATA\_IN, QBDATA\_IN) begin
264. **case** (MEMWRITE)
265. 1'b0: DMDATA\_OUT <= DM[ALUDATA\_IN];
266. 1'b1: DMDATA\_OUT <= DM[QBDATA\_IN];
267. endcase
268. end
269. endmodule
271. module MemWB(clock, mWREG, mM2REG, mMUX, mALU, mDM);
272. input clock, mWREG, mM2REG;
273. input [4:0] mMUX;
274. input [31:0] mALU, mDM;
276. reg mwreg,mm2reg;
277. reg [4:0] mmux;
278. reg [31:0] malu,mdm;
280. always @(posedge clock) begin
281. mwreg <= mWREG;
282. mm2reg <= mM2REG;
283. mmux <= mMUX;
284. malu <= mALU;
285. mdm <= mDM;
286. end
287. endmodule

**Testbench Module:**

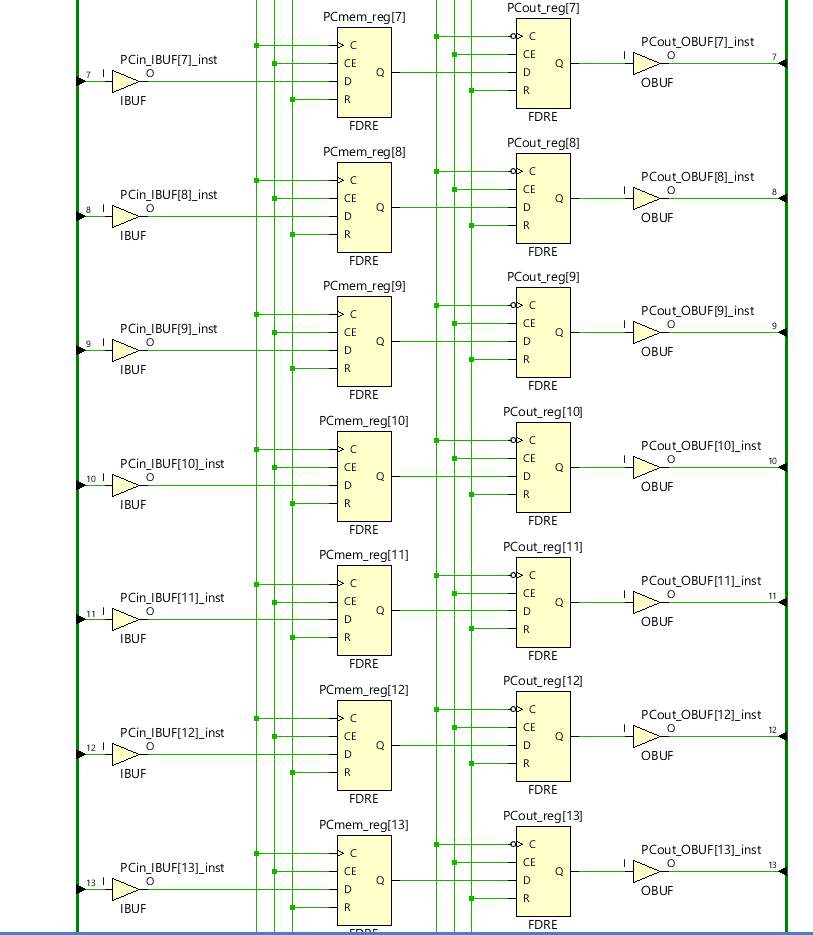
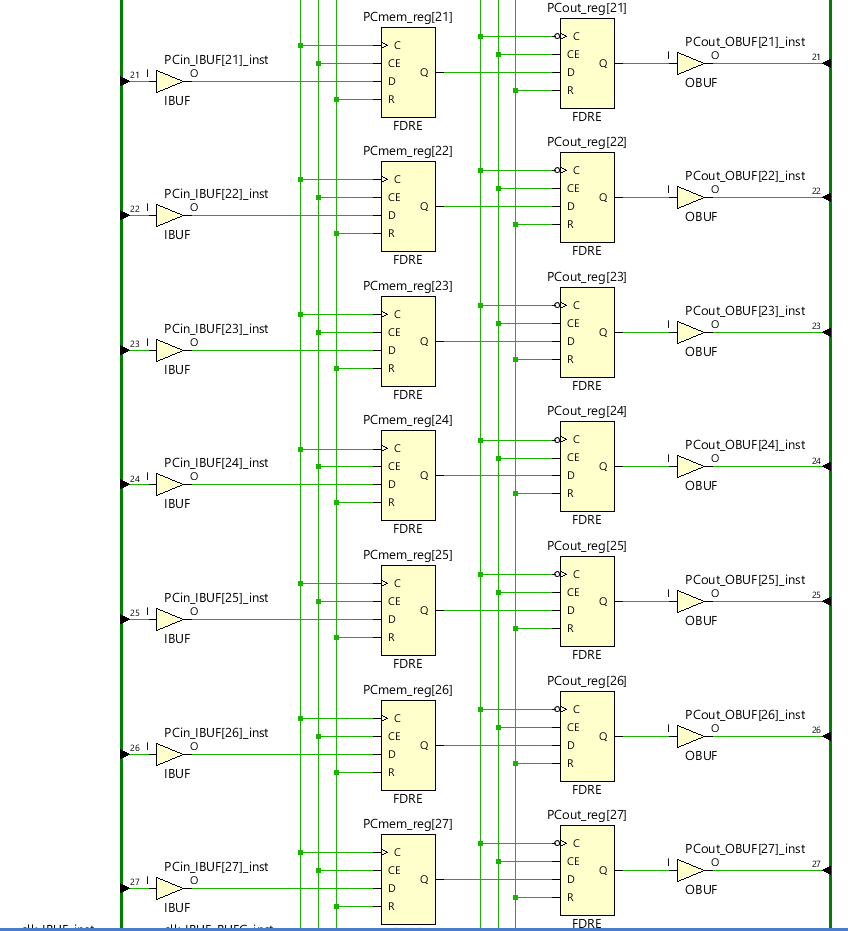
1. //Taylan Unal CMPEN 331.001 LAB4 Testbench
2. `timescale 1ns / 1ps
4. module CPU\_Test();
5. //WIRES ARE OUTPUTS, REGS ARE INPUTS
6. reg clk; //could be clk = 0;
7. wire [31:0] pcin, pcout, im\_out,if\_idout,qawire,qbwire,sign\_extendedout, eqawire
8. eqbwire, eimmnext\_out, alumux\_out, alu\_out, e2mALU, qb2DM, dm\_out;
9. wire [3:0] aluc\_out, ealuc\_out;
10. wire wregwire, m2regwire, wmemwire, aluimmwire, regrtwire, ewregwire,
11. em2regwire, ewmemwire, ealuimmwire, e2mwreg, e2mM2reg, wmemDM;
12. wire [4:0] muxwire, emuxwire, e2mMux; //e2mMux;
13. //////////////////////////////////////////////////////////////////////////////////////
14. PC PC(clk, pcin, pcout); //done
15. PCAdder adder(pcout, pcin); //done
16. InstructMem instmem(pcout, im\_out); //done
17. IFID ifid(clk, im\_out, if\_idout);//needs clk, im\_out is Ifid\_in, if\_idout is output
18. ControlUnit ctrunit(if\_idout, wregwire, m2regwire, wmemwire,
19. aluc\_out, aluimmwire, regrtwire);
20. ControlMux ctrmux(if\_idout, regrtwire, muxwire); // inst\_Mux contains ifid, regrt from control unit, muxwire contains rd\_rt
21. RegFile regfile(if\_idout, qawire, qbwire); //if\_id is inst\_RF, qawire is RS output, qbwire is RT output
22. SignExtend extender(if\_idout, sign\_extendedout); //input 32 bit non sign extended, output 32 bit extended.
23. IDEXE idexe(clk,wregwire,m2regwire,wmemwire, aluc\_out, aluimmwire, muxwire, qawire, qbwire, sign\_extendedout, ewregwire, em2regwire, ewmemwire, ealuc\_out, ealuimmwire,
24. emuxwire, eqawire, eqbwire, eimmnext\_out); //working on it
25. ALUMux alumux(ealuimmwire, eqbwire, eimmnext\_out, alumux\_out);
26. ALU alu(ealuc\_out, eqawire, alumux\_out, alu\_out);
27. EXEMEM exemem(clk, ewregwire, em2regwire, ewmemwire, emuxwire, alu\_out, eqbwire,
28. e2mwreg, e2mM2reg, wmemDM, e2mMux, e2mALU, qb2DM);
29. DataMemory datamem(wmemDM, e2mALU, qb2DM, dm\_out); //memwrite, alu input, qbdata input, DM output
30. MemWB memwb(clk, e2mwreg, e2mM2reg, e2mMux, e2mALU, dm\_out); //clk, memwrite reg, mem Mux, mem ALU, mem DM
32. initial begin
33. clk = 0;
34. end
35. always begin
36. #5 clk = !clk;
37. end
38. endmodule

**Waveforms:**

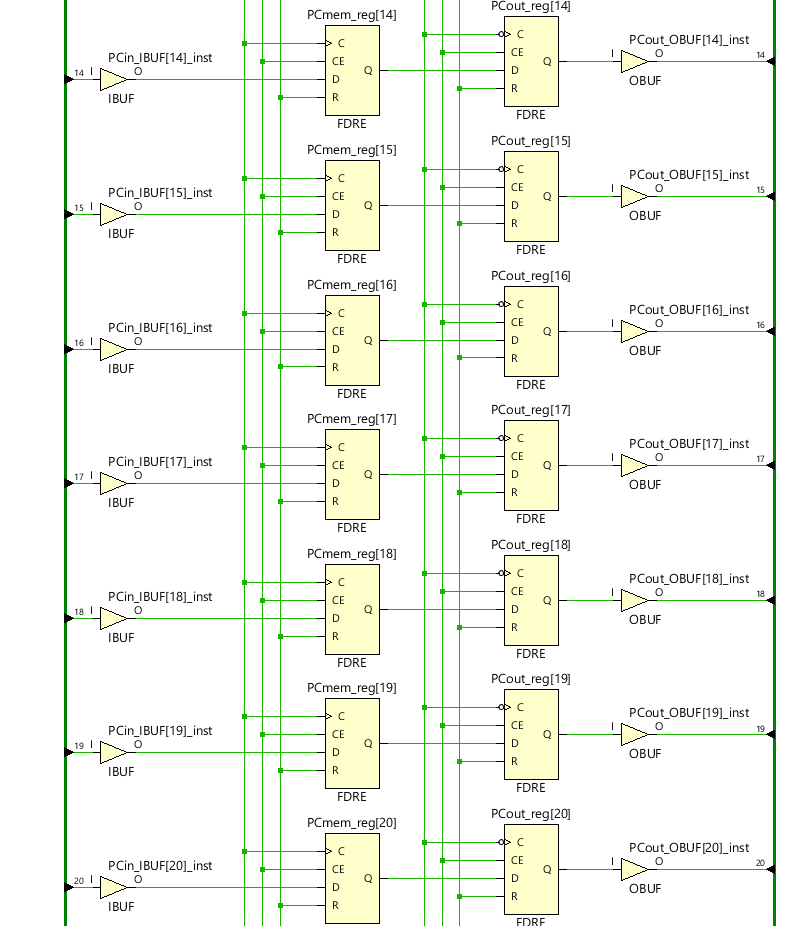
**Design Schematic:**

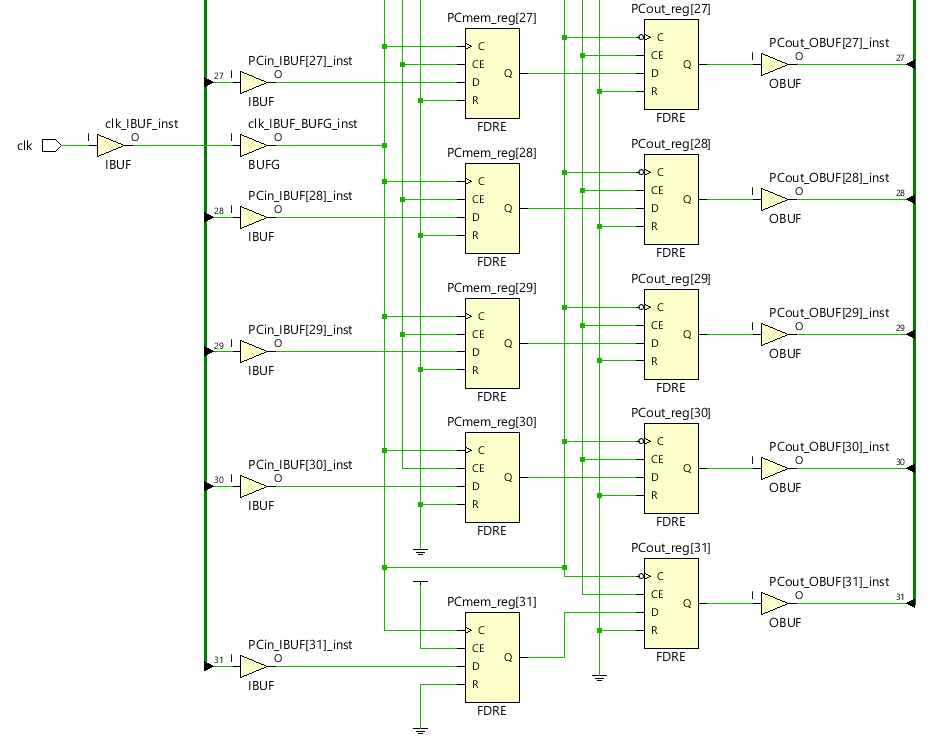
****

**Design Schematic (contd):**

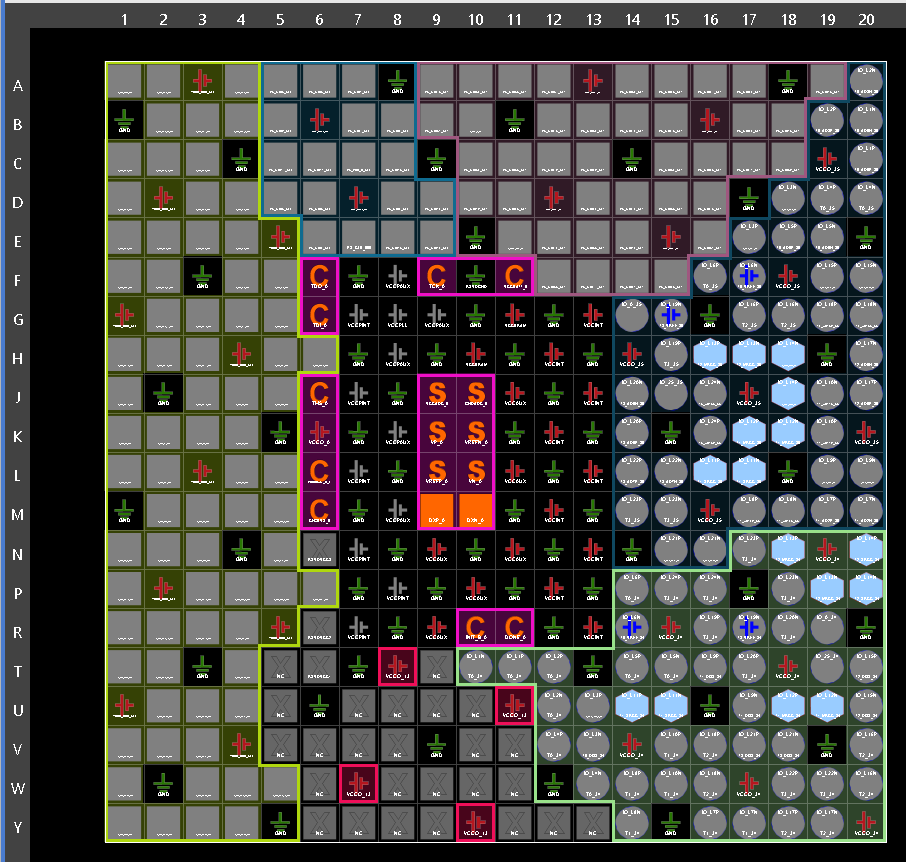
****

**Design Schematic (contd):**

****

****

**IO Planning:**



**Floor Planning:**

