**中国科学技术大学计算机学院**

**《计算机组成原理实验》报告**



实验题目：\_流水线CPU设计\_

学生姓名：\_\_\_吴毅龙\_\_\_\_\_\_

学生学号：\_\_\_PB19111749\_\_

完成日期：\_\_2021/6/1\_\_\_

计算机实验教学中心制

2020年09月

【实验题目】

流水线CPU设计

【实验目的】

* 理解CPU的功能、结构和工作原理
* 掌握流水线CPU的设计和调试方法，特别是流水线中数据相关和控制相关的处理
* 熟练掌握数据通路和控制器的设计和描述方法

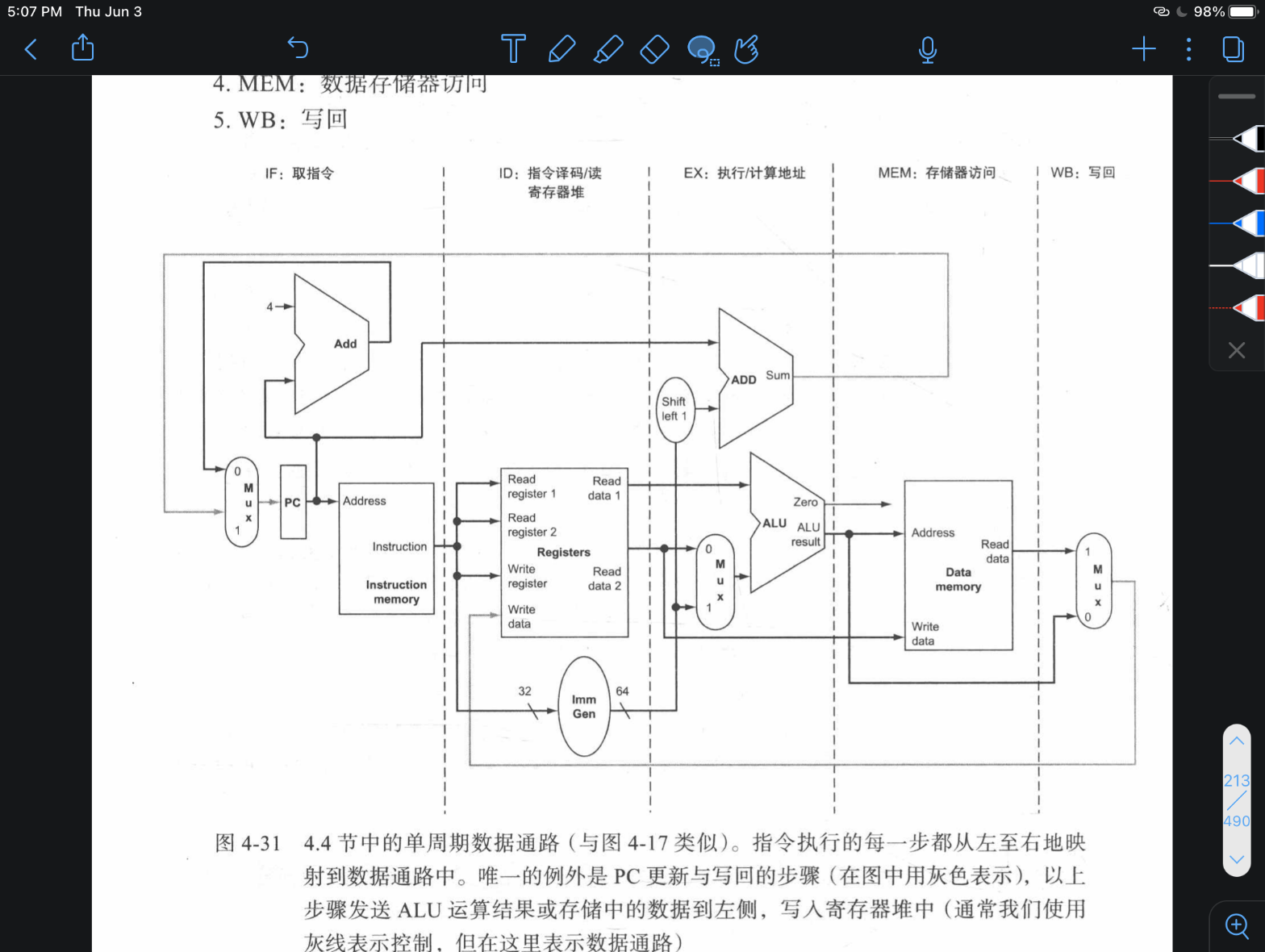
【实验环境】  
FPGAOL: fpgaol.ustc.edu.cn  
Vivado

【实验内容】

1. 修改Lab4寄存器堆模块，使其满足写优先(Write First)，即在对同一寄存器读写时，写数据可立即从读数据输出
2. 设计无数据和控制相关处理的流水线CPU
3. 设计仅有数据相关处理的流水线CPU
4. 设计完整的有数据和控制相关处理的流水线CPU
   * 对CPU进行功能仿真
   * 将CPU和PDU连接并下载至FPGA中测试

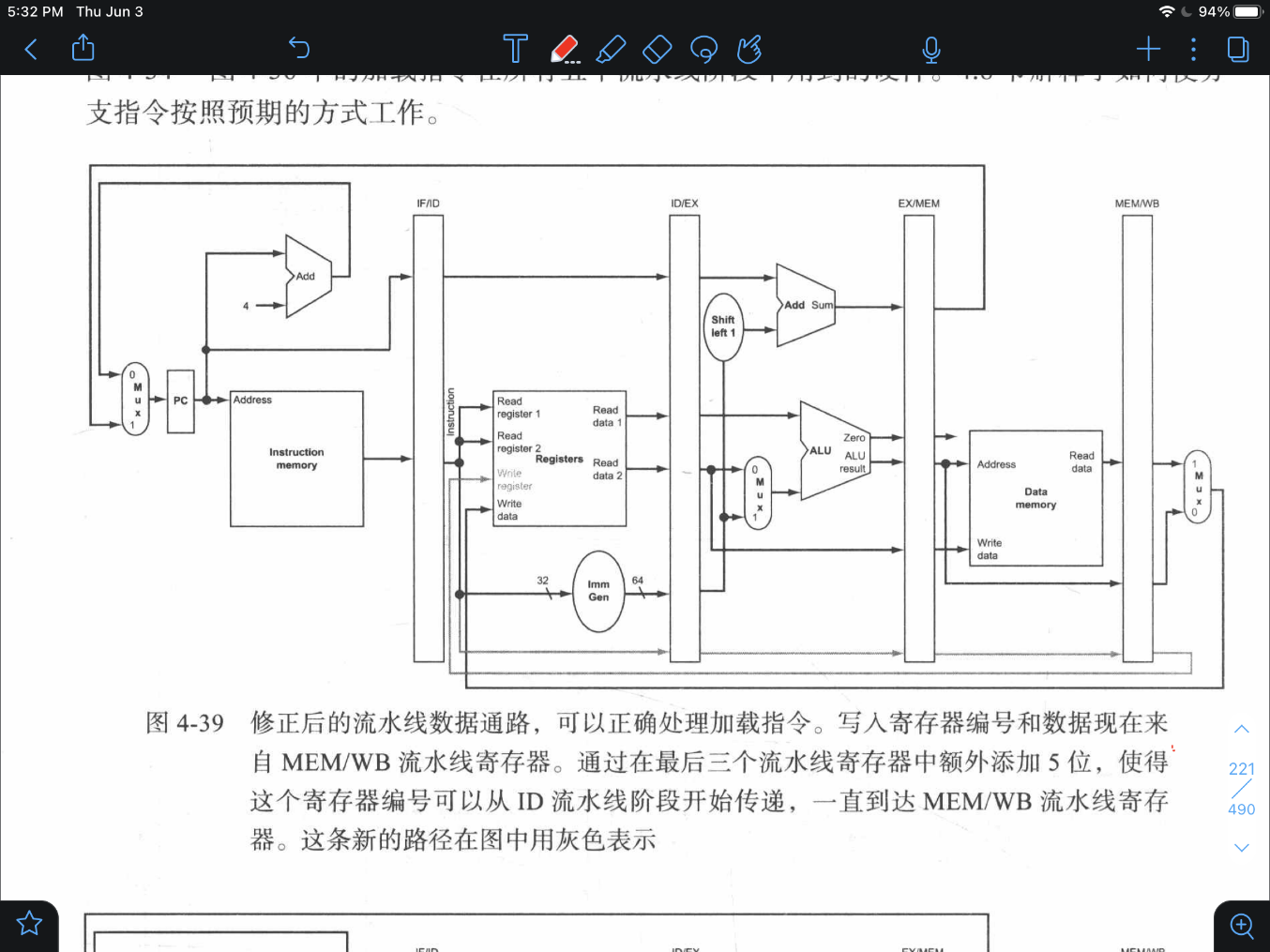
【实验练习】

流水线分级：本次实验实现的流水线CPU包含有五级流水线，意味着在任意单时钟周期里最多执行五条指令，相应的，我们需要把数据通路划分为五个部分，将五个部分用相应的指令执行阶段来命名；



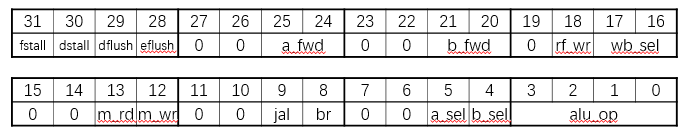
段间寄存器引入：我们可以通过引入寄存器保存数据的方式，是的部分数据通路可以在指令执行的过程中被共享。举例来说，指令存储器只在指令的五个阶段的一个阶段被使用，而在其他四个阶段中允许被其他指令共享。为了保留在其他四个阶段的指令的值，必须把从指令存储器中读取出来的数据保存在寄存器中。类似的理由适用于每个流水线阶段，所以我们必须将寄存器置于上图每个阶段的分割线上。下图显示了流水线寄存器的分布，所有指令都会在每一个时钟周期里从一个流水线寄存器前进到下一个寄存器中。

我们需要在加载指令的流水线寄存器中保留目标寄存器的编号。就像存储指令为了MEM阶段的使用而将寄存器的值从ID/EX中传递到EX/MEM流水线寄存器中那样，加载指令为了WB阶段的使用而将寄存器编号从ID/EX通过EX/MEM传递到MEM/WB流水线寄存器。寄存器编号在WB阶段被使用，指定了要写入的寄存器。



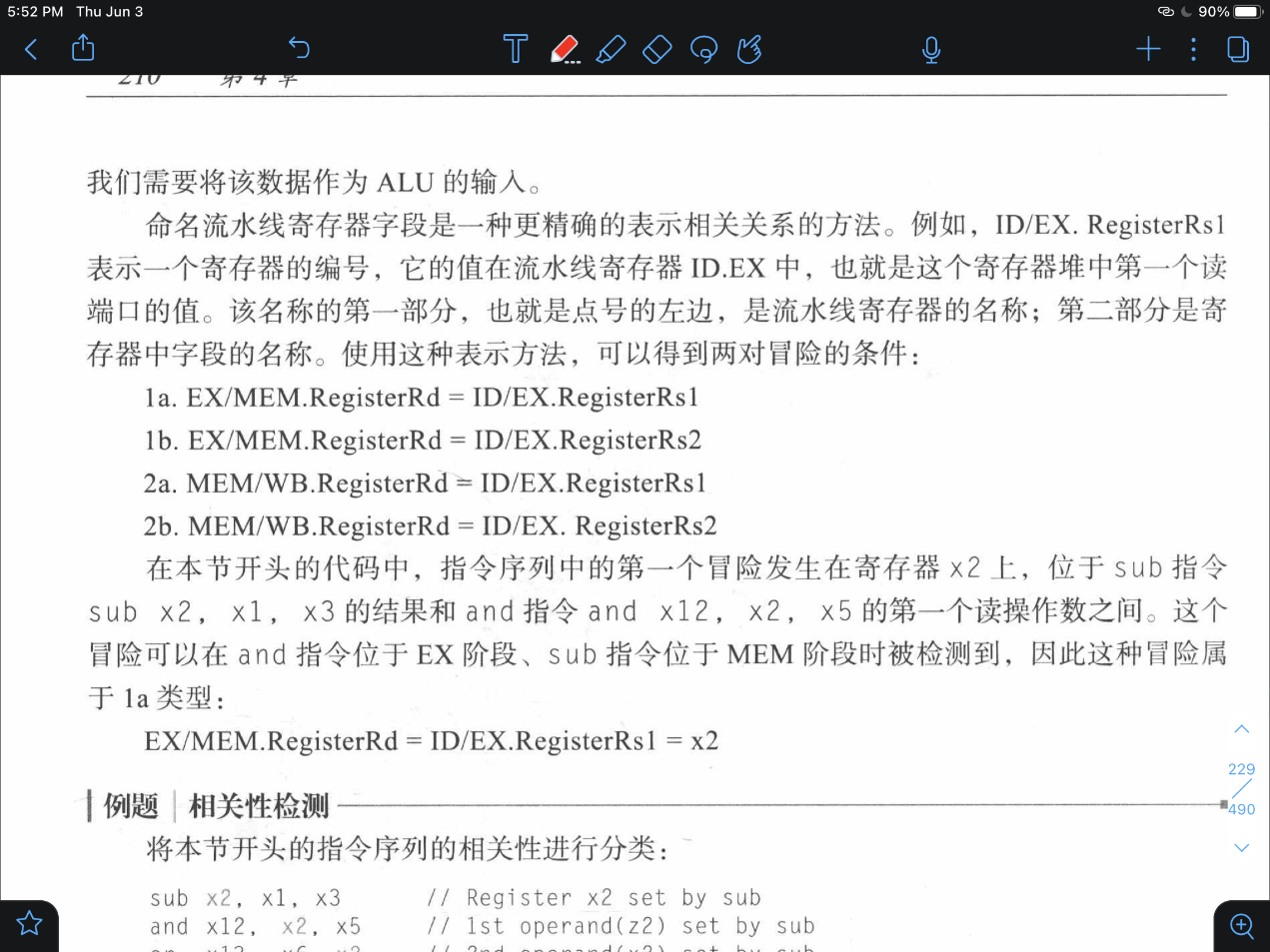
1. module IF\_ID(
2. input clk, IF\_en, IF\_Flush,
3. input [31:0] PC, PCadd4, instruction,
4. output reg [31:0] pcd, pcin, ir
5. );
6. initial begin
7. ir = 0;
8. end
9. always @(posedge clk)
10. begin
11. if (IF\_Flush)    //清除IF阶段的指令
12. begin
13. ir <= 0;
14. pcd <= 0;
15. pcin <= 0;
16. end
17. else if(IF\_en)
18. begin
19. ir <= instruction;
20. pcd <= PC;
21. pcin <= PCadd4;
22. end
23. else
24. begin
25. ir <= ir;
26. pcd <= pcd;
27. pcin <= pcin;
28. end
29. end
30. endmodule
31. module ID\_EX(
32. input clk, ID\_en, ID\_Flush,
33. input [31:0] control,
34. input [31:0] pcin, pcd,
35. input [31:0] rs1, rs2,
36. input [31:0] ir, Imm,
37. output reg[31:0] ctrl,
38. output reg[31:0] pce, pcin2,
39. output reg[31:0] a, b,
40. output reg[31:0] imm,
41. output reg[4:0] rd, ra1, ra2
42. );
43. initial begin
44. ctrl=0;
45. a=0;
46. b=0;
47. end
48. always @(posedge clk)
49. begin
50. if(ID\_Flush)
51. begin
52. ctrl  <= 0;
53. pcin2 <= 0;
54. pce   <= 0;
55. a     <= 0;
56. b     <= 0;
57. imm   <= 0;
58. rd    <= 0;
59. ra1   <= 0;
60. ra2   <= 0;
61. end
62. else if(ID\_en)
63. begin
64. ctrl  <= control;
65. pcin2 <= pcin;
66. pce   <= pcd;
67. a     <= rs1;
68. b     <= rs2;
69. imm   <= Imm;
70. rd    <= ir[11:7];
71. ra1   <= ir[19:15];
72. ra2   <= ir[24:20];
73. end
74. else
75. begin
76. ctrl  <= ctrl ;
77. pcin2 <= pcin2;
78. pce   <= pce  ;
79. a     <= a    ;
80. b     <= b    ;
81. imm   <= imm  ;
82. rd    <= rd   ;
83. ra1   <= ra1  ;
84. ra2   <= ra2  ;
85. end
86. end
87. endmodule
88. module EX\_MEM(
89. input clk,
90. input [31:0] ALUresult, op2, ctrl, pcin2,
91. input [4:0] rd,
92. output reg [31:0] ctrlm, y, bm, pcin3,
93. output reg [4:0] rdm
94. );
95. always @(posedge clk) begin
96. y <= ALUresult;
97. bm <= op2;
98. ctrlm <= ctrl;
99. pcin3 <= pcin2;
100. rdm <=rd;
101. end
102. endmodule
103. module ME­M\_WB(
104. input clk,
105. input [4:0] rdm,
106. input [31:0] y, ReadData, ctrlm, pcin3,
107. output reg [31:0] yw, mdr, ctrlw, pcin4,
108. output reg [4:0] rdw
109. );
110. always @(posedge clk) begin
111. rdw <= rdm;
112. yw <= y;
113. mdr <= ReadData;
114. ctrlw <= ctrlm;
115. pcin4 <= pcin3;
116. end
117. endmodule

流水线控制：由于控制线从EX阶段开始，可以在指令译码阶段为之后的阶段创建控制信号。传递这些控制信号最简单的方式就是扩展流水线寄存器以包含这些控制信息。随着指令沿着流水线向下流动，这些控制信号被用于适当的流水线阶段。在本次实验中，控制信号被规定为如下格式

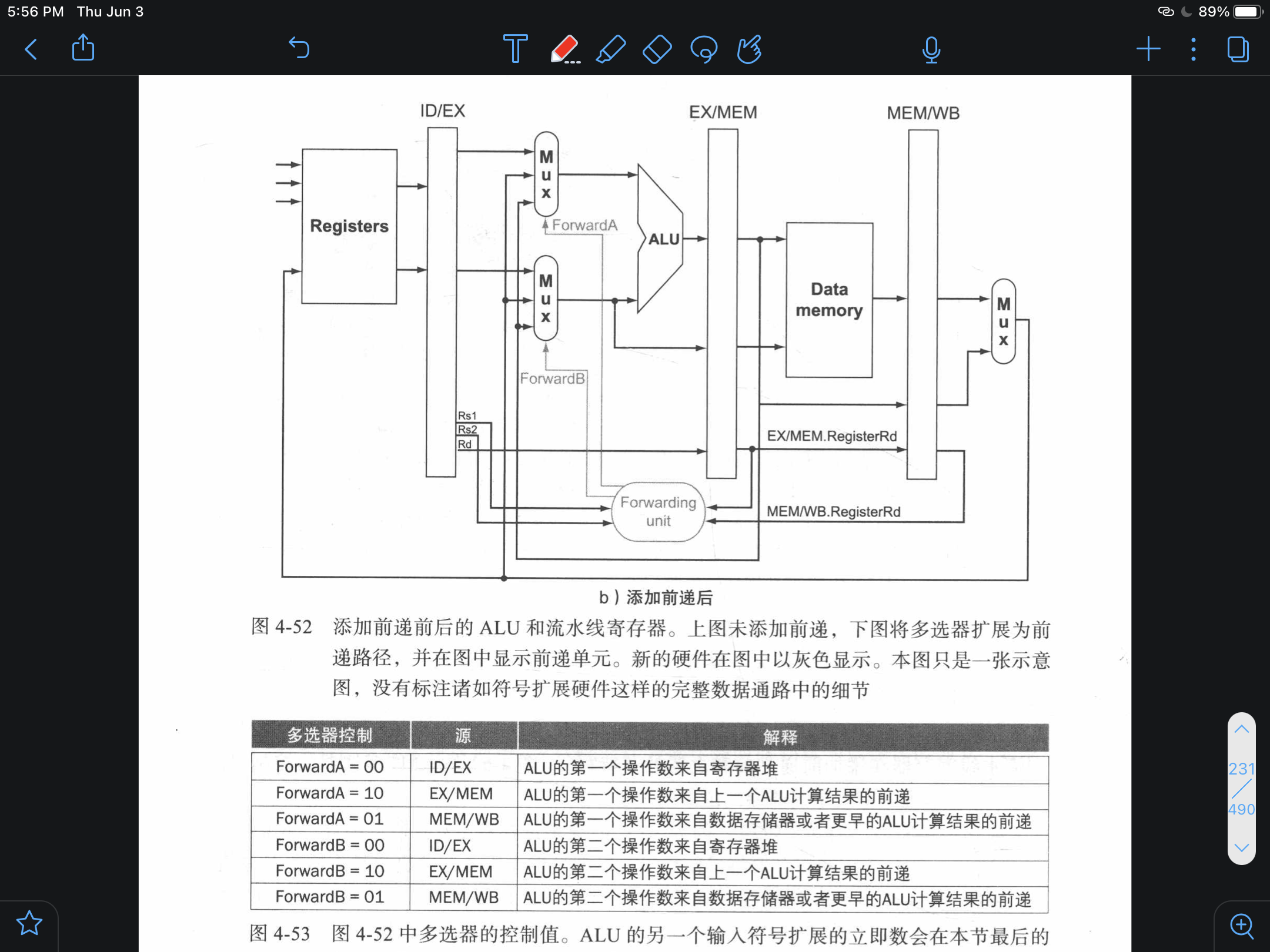
****

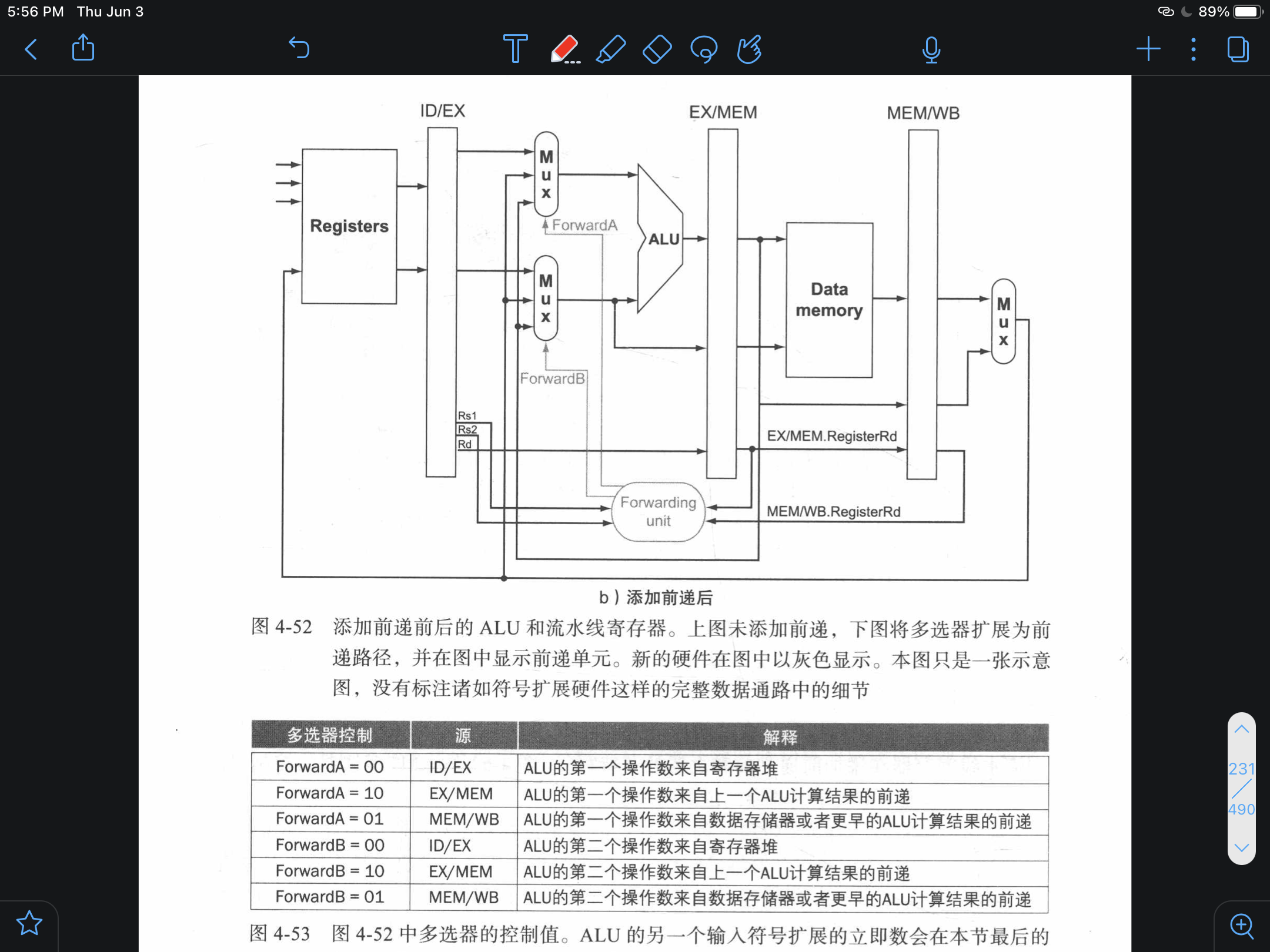
1. module Control(
2. input [6:0] opcode,
3. input [1:0] ForwardA,ForwardB,
4. output reg[31:0] control
5. );
6. localparam ALUresult = 2'b01, DataMem=2'b10, PCadd4=2'b11,IDIE=2'b00;
7. always @(\*) begin
8. control[25:24] = ForwardA;
9. control[21:20] = ForwardB;
10. control[31:26] = 0;
11. control[23:22] = 0;
12. control[19] = 0;
13. control[15:14] = 0;
14. control[11:10] = 0;
15. control[7:6] = 0;
16. case(opcode)
17. 7'b0110011: begin   //add
18. control[3:0] = 4'b0010;     //ALU\_op
19. control[5:4] = 2'b11;       //a\_sel b\_sel
20. control[9:8] = 2'b00;       //jal branch
21. control[13:12] = 2'b00;//memory\_read memory\_write
22. control[18] = 1;       //register file write
23. control[17:16] = ALUresult;    //wb\_sel
24. end
25. 7'b0010011: begin   //addi
26. control[3:0] = 4'b0010;        //ALU\_op
27. control[5:4] = 2'b10;          //a\_sel b\_sel
28. control[9:8] = 2'b00;          //jal branch
29. control[13:12] = 2'b00;//memory\_read memory\_write
30. control[18] = 1;       //register file write
31. control[17:16] = ALUresult;    //wb\_sel
32. end
33. 7'b0000011: begin   //lw
34. control[3:0] = 4'b0010;        //ALU\_op
35. control[5:4] = 2'b10;          //a\_sel b\_sel
36. control[9:8] = 2'b00;          //jal branch
37. control[13:12] = 2'b10;//memory\_read memory\_write
38. control[18] = 1;       //register file write
39. control[17:16] = DataMem;      //wb\_sel
40. end
41. 7'b0100011: begin   //sw
42. control[3:0] = 4'b0010;        //ALU\_op
43. control[5:4] = 2'b10;          //a\_sel b\_sel
44. control[9:8] = 2'b00;          //jal branch
45. control[13:12] = 2'b01;//memory\_read memory\_write
46. control[18] = 0;       //register file write
47. control[17:16] = IDIE;         //wb\_sel
48. end
49. 7'b1100011: begin   //beq
50. control[3:0] = 4'b0110;        //ALU\_op
51. control[5:4] = 2'b11;          //a\_sel b\_sel
52. control[9:8] = 2'b01;          //jal branch
53. control[13:12] = 2'b00;//memory\_read memory\_write
54. control[18] = 0;       //register file write
55. control[17:16] = IDIE;         //wb\_sel
56. end
57. 7'b1101111: begin   //jal
58. control[3:0] = 4'b0010;        //ALU\_op
59. control[5:4] = 2'b00;          //a\_sel b\_sel
60. control[9:8] = 2'b10;          //jal branch
61. control[13:12] = 2'b00;//memory\_read memory\_write
62. control[18] = 1;       //register file write
63. control[17:16] = PCadd4;       //wb\_sel
64. end
65. default: begin
66. control[3:0] = 4'b0000;        //ALU\_op
67. control[5:4] = 2'b00;          //a\_sel b\_sel
68. control[9:8] = 2'b00;          //jal branch
69. control[13:12] = 2'b00;//memory\_read memory\_write
70. control[18] = 0;       //register file write
71. control[17:16] = IDIE;         //wb\_sel
72. end
73. endcase
74. end
75. endmodule

数据冒险（前递）：当一个指令试图在EX阶段使用的寄存器是一个较早的指令在WB阶段要写入的寄存器时，我们需要将该数据作为ALU的输入，所以就可以得到两对冒险的条件：



因为不是所有的指令都会写回寄存器，因此还需要添加必要的判断标准，一种简单的解决方案是检查RegWrite信号是否有效：检查流水线寄存器在EX和MEM阶段的WB控制字段以确定RegWrite是否有效。于是就可以给出检测冒险的条件了。









1. module Forwarding(
2. input [4:0] ra1, ra2, rdm, rdw, rdf,
3. input rf\_wr\_m, rf\_wr\_w, rf\_wr\_f,
4. output reg [1:0] ForwardA, ForwardB
5. );
6. always @(\*)
7. begin
8. if(rf\_wr\_m && ra1 == rdm ) ForwardA = 2'b01;
9. else if(rf\_wr\_w && ra1 == rdw) ForwardA = 2'b10;
10. else if(rf\_wr\_f && ra1 == rdf) ForwardA = 2'b11;
11. else ForwardA = 2'b00;
12. end
13. always @(\*)
14. begin
15. if(rf\_wr\_m && ra2 == rdm) ForwardB = 2'b01;
16. else if(rf\_wr\_w && ra2 == rdw) ForwardB = 2'b10;
17. else if(rf\_wr\_f && ra2 == rdf) ForwardB = 2'b11;
18. else ForwardB = 2'b00;
19. end
20. endmodule

数据冒险（停顿）：冒险控制单元的控制逻辑满足以下条件：



使流水线停顿只需要禁止PC寄存器和IF/ID流水线寄存器的改变就可以阻止这两条指令的执行。如果这些寄存器被保护，在IF阶段的指令就会继续使用相同的PC值来取指令，同时ID段的寄存器会继续使用IF/ID流水线寄存器中相同的字段来读寄存器。另外需要解除EX、MEM和WB阶段的控制信号（将其设置为0）就可以产生一个“没有任何操作”的指令，也就是空指令。

控制冒险：一种提升分支阻塞效率的方法是预测分支条件不发生并持续执行顺序指令流。一旦条件分支发生，已经被读取和译码的指令就将被丢弃，流水线继续从分支目标处开始执行。想要丢弃指令，只需要将初始控制值变为0即可，这与指令停顿以解决加载-使用的数据冒险类似。不同的是，丢弃指令的同时也需要改变当分支指令到达MEM阶段时IF、ID和EX阶段的三条指令；而在加载-使用数据冒险的停顿中，只需要将ID段的控制信号变为0并且精该阶段的指令从流水线中过滤出去即可。

1. module Hazard(
2. input m\_rd\_idex, m\_wr\_ifid,
3. input [4:0]rd, ra1, ra2,
4. input zero, beq\_d, jal\_d, beq,jal,
5. input a\_sel, b\_sel,
6. output reg PC\_en, IF\_en, ID\_en, IF\_Flush, ID\_Flush
7. );
8. always @(\*) begin
9. if(jal || (zero && beq))
10. begin
11. PC\_en = 1;
12. IF\_Flush = 1;
13. IF\_en = 0;
14. ID\_en = 0;
15. ID\_Flush = 1;
16. end
17. else if( m\_rd\_idex && ( (a\_sel && rd == ra1) || ((b\_sel || m\_wr\_ifid ) && rd == ra2) ) )
18. begin
19. PC\_en = 0;
20. IF\_Flush = 0;
21. IF\_en = 0;
22. ID\_en = 0;
23. ID\_Flush = 1;
24. end
25. else
26. begin
27. PC\_en = 1;
28. IF\_Flush = 0;
29. IF\_en = 1;
30. ID\_en = 1;
31. ID\_Flush = 0;
32. end
33. end
34. endmodule

最终连线构成CPU

1. module CPU(
2. input clk, rst,
3. //IO\_BUS
4. output [7:0] io\_addr,
5. output [31:0] io\_dout,
6. output io\_we,
7. input [31:0] io\_din,
8. //Debug\_BUS
9. input [7:0] m\_rf\_addr,
10. output [31:0] rf\_data,
11. output [31:0] m\_data,
12. output [31:0] pcin, pc, pcd, pce,
13. output [31:0] ir, imm, mdr,
14. output [31:0] a, b, y, bm, yw,
15. output [4:0]  rd, rdm, rdw,
16. output [31:0] ctrl, ctrlm, ctrlw
17. );
18. reg [31:0] PC;
19. assign pc = PC;
20. wire [31:0] PCN;
21. wire PC\_en;
22. always @(posedge clk, posedge rst) begin
23. if(rst)PC <= 0;
24. else if(PC\_en) PC <= PCN;
25. else PC <= PC;
26. end
27. wire [31:0] instruction;
28. instruction\_mem Instruction\_mem(
29. .a(PC[9:2]),
30. .spo(instruction)
31. );
32. wire [31:0] PCadd4;
33. assign PCadd4 = PC + 4;
34. wire [31:0] PCjmp;
35. wire isPCjmp;
36. assign PCN = isPCjmp ? PCjmp : PCadd4;
37. wire IF\_en, IF\_Flush;
38. IFID IF\_ID(
39. .clk(clk),
40. .PC(PC),
41. .PCadd4(PCadd4),
42. .instruction(instruction),
43. .pcin(pcin),
44. .pcd(pcd),
45. .ir(ir),
46. .IF\_en(IF\_en),
47. .IF\_Flush(IF\_Flush)
48. );
49. wire [31:0]rs1, rs2;
50. reg [31:0] WriteData;
51. Registers RF(
52. .clk(clk),
53. .we(ctrlw[18]),                //register file write
54. .wa(rdw),
55. .ra0(ir[19:15]),
56. .ra1(ir[24:20]),
57. .ra2(m\_rf\_addr),
58. .rd0(rs1),
59. .rd1(rs2),
60. .rd2(rf\_data),
61. .wd(WriteData)
62. );
63. wire [31:0]Imm;
64. ImmGen ImmGen(ir,Imm);
65. wire [31:0] control,pcin2;
66. wire [4:0] ra1,ra2;
67. wire ID\_en,ID\_Flush;
68. IDEX ID\_EX(
69. .clk(clk),
70. .control(control),.ctrl(ctrl),
71. .rs1(rs1),.a(a),
72. .rs2(rs2),.b(b),
73. .Imm(Imm),.imm(imm),
74. .ir(ir),.rd(rd),
75. .pcd(pcd),.pce(pce),
76. .pcin(pcin),.pcin2(pcin2),
77. .ra1(ra1),.ra2(ra2),
78. .ID\_en(ID\_en),
79. .ID\_Flush(ID\_Flush)
80. );
81. wire [1:0] ForwardB,ForwardB;
82. Control Controler(
83. .opcode(ir[6:0]),
84. .ForwardA(ForwardA),
85. .ForwardB(ForwardB),
86. .control(control)
87. );
88. reg [31:0] op1;
89. wire [31:0] op2;
90. reg [31:0] op2\_temp;
91. wire [31:0] ALUresult;
92. wire zero;
93. wire [31:0] LastWriteData;
95. always @(\*) begin
96. case (control[25:24])        //ForwardA
97. 2'b00:op1 = a;
98. 2'b01:op1 = y;
99. 2'b10:op1 = WriteData;
100. default:op1 = LastWriteData;
101. endcase
102. end
103. always @(\*) begin
104. case (control[21:20])        //ForwardB
105. 2'b00:op2\_temp = b;
106. 2'b01:op2\_temp = y;
107. 2'b10:op2\_temp = WriteData;
108. default:op2\_temp = LastWriteData;
109. endcase
110. end
112. ALU #(32)alu(
113. .a(op1),
114. .b(op2),
115. .s(ctrl[3:0]),
116. .y(ALUresult),
117. .zf(zero)
118. );
119. wire [31:0] pcin3;
120. EXMEM EX\_MEM(
121. .clk(clk),
122. .ALUresult(ALUresult),.y(y),
123. .op2(op2\_temp),.bm(bm),
124. .pcin2(pcin2),.pcin3(pcin3),
125. .ctrl(ctrl),.ctrlm(ctrlm),
126. .rd(rd),.rdm(rdm)
127. );
129. wire [31:0] ReadData\_temp;//map device
130. wire DM\_we;
131. wire [31:0] ReadData;
132. data\_mem Data\_mem(
133. .a(y[9:2]),
134. .d(bm),
135. .dpra(m\_rf\_addr),
136. .clk(clk),
137. .we(DM\_we),                   //ctrlm[12]  memory\_write
138. .spo(ReadData\_temp),
139. .dpo(m\_data)
140. );
141. //memory map device
142. assign DM\_we = ctrlm[12] & (~y[10]);
143. assign io\_addr = y[7:0];
144. assign io\_dout = bm;
145. assign io\_we = ctrlm[12];
146. assign ReadData = y[10] ? io\_din : ReadData\_temp;
147. wire [31:0] pcin4;
148. MEMWB MEM\_WB(
149. .clk(clk),
150. .rdm(rdm),.rdw(rdw),
151. .y(y),.yw(yw),
152. .pcin3(pcin3),.pcin4(pcin4),
153. .ctrlm(ctrlm),.ctrlw(ctrlw),
154. .ReadData(ReadData),.mdr(mdr)
155. );
157. wire [4:0] rdf;
158. wire rf\_wr\_f;
159. Forwarding forwarding\_unit(
160. .ForwardA(ForwardA),
161. .ForwardB(ForwardB),
162. .ra1(ra1),
163. .ra2(ra2),
164. .rdm(rdm),
165. .rdw(rdw),
166. .rdf(rdf),
167. .rf\_wr\_m(ctrlm[18]),
168. .rf\_wr\_w(ctrlw[18]),
169. .rf\_wr\_f(rf\_wr\_f)
170. );
171. always @(\*) begin
172. case(ctrlw[17:16])          //wb\_sel
173. 2'b01:WriteData = yw;     //ALUresult
174. 2'b10:WriteData = mdr;    //DataMem
175. 2'b11:WriteData = pcin4;  //PCadd4
176. default:WriteData = 0;
177. endcase
178. end
180. assign op2 = ctrl[4] ? op2\_temp : imm;  //ctrl[4] b\_sel
181. assign PCjmp = (imm << 1) + pce;
182. assign isPCjmp = ctrl[9] | (zero & ctrl[8]);
184. WBIF WB\_IF(
185. .clk(clk),
186. .WriteData(WriteData),
187. .LastWriteData(LastWriteData),
188. .rdw(rdw),
189. .rdf(rdf),
190. .rf\_wr\_w(ctrlw[18]),
191. .rf\_wr\_f(rf\_wr\_f)
192. );
193. Hazard Hazard\_Unit(
194. .m\_rd\_idex(ctrl[13]),
195. .m\_wr\_ifid(control[12]),
196. .zero(zero),
197. .beq\_d(control[8]),
198. .jal\_d(control[9]),
199. .beq(ctrl[8]),
200. .jal(ctrl[9]),
201. .a\_sel(control[5]),
202. .b\_sel(control[4]),
203. .rd(rd),
204. .ra1(ir[19:15]),
205. .ra2(ir[24:20]),
206. .PC\_en(PC\_en),
207. .IF\_en(IF\_en),
208. .IF\_Flush(IF\_Flush),
209. .ID\_en(ID\_en),
210. .ID\_Flush(ID\_Flush)
211. );
212. endmodule

【总结与思考】

通过本次实验，我利用FPGAOL平台进行实验，并使用IP核，实现了五级流水线的CPU设计。本次实验难度不大，根据实验说明的指导就可以完成实验操作，实验题目的难度层层递进，有基础操作的考核，也有所学知识的综合，难易结合，既有复习又有思考，让所学在实践中得以运用，加深了我对逻辑电路知识的理解。希望今后实验可以保持本次实验中详细实验指导描述的优点，辅助完成每项试验内容。