



**计算机组成原理(汇编与接口) 课 程 设 计**

团队（个人）实验报告

|  |  |
| --- | --- |
| 学 院 | 计算机学院 |
| 专 业 | 计算机科学与技术 |
| 指导老师 |  |
| 组 长 | 周振翔 |
| 组 员 | 周振翔 董若扬 穆新宇 张亦驰 |
| 组长联系方式 | 18333667526 |

二O二三年 9 月

目 录

备注：团队与个人报告均使用此模板，红色字体供参考，请根据实际情况填写，黑色框架部分请勿修改。个人报告不需要分工介绍。

# 项目简述

实现经典五级流水线CPU，支持16条riscv-32I指令，包含控制冒险，结合汇编与接口设计实验，设计测试用例，并下板验证成功。

小组分工见表2.1 小组分工。

|  |  |
| --- | --- |
| **组员** | **分工任务** |
| 周振翔 | * 掌握进度，组合汇编和计组任务，设计流水线CPU架构，协同DEBUG，综合上板 |
| 张亦驰 | * 协同设计流水线CPU，实现CPU模块，仿真DEBUG |
| 董若扬 | * 实现VGA汇编操控 |
| 穆新宇 | * 实现蜂鸣器硬联逻辑 |

表2.1 小组分工

# 设计目的

将单周期CPU根据流水线的架构改为流水线CPU，并且实现基本的五级流水线架构，采用分支预测技术。

# 设计环境

|  |  |
| --- | --- |
| 操作系统 | Windows10 |
| 编程语言 | Verilog |
| EDA工具 | Vivado 2019.2 |
| 汇编语言 | Riscv-32I |
| 汇编程序编辑器 | RARS 1.6 |

请标注版本号

# 设计原理及内容

## CPU整体架构

本流水线CPU覆盖X条指令(见表5.1-1)，下面将从数据通路和控制逻辑阐述设计思路。

示例：列出自己设计的CPU支持的指令。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 指令 | opcode | Function3 | Function7 | 功能 |
| 0 | Null |  |  |  | 插入bubble，使用气泡代替执行和译码阶段的指令 |
| 1 | ADD | b'0110011 | b'000 | b'0000000 | rd=rs+rt |
| 2 | SUB | b'0110011 | b'000 | b'0100000 | rd=rs-rt |
| 3 | ORI | b'0110111 | b'110 | \ | rd=rs|im |
| 4 | LUI | b'0100011 | \ | \ | rt=im\*65536 |
| 5 | SW | b'0000011 | b'010 | \ | SW rt, offset(base) |
| 6 | LW | b'1100011 | b'010 | \ | LW rt, offset(base) |
| 7 | BEQ | b'1101111 | b'000 | \ | PC=(rs==rt)?PC+im<<2:PC |
| 8 | JAL | b'0010011 | \ | \ | $31=PC;PC={(PC+4)[31,28],addr,00} |
| 9 | XORI | b'0010011 | b'100 | \ | rd=rs xor im |
| 10 | ANDI | b'0110011 | b'111 | \ | rd=rs&im |
| 11 | XOR | b'0110011 | b'100 | b'0000000 | rd=rs xor rd |
| 12 | OR | b'0110011 | b'110 | b'0000000 | rd=rs|rt |
| 13 | AND | b'1100011 | b'111 | b'0000000 | rd=rs&rt |
| 14 | BNE | b'1100011 | b'001 | \ | PC=(rs!=rt)?PC+im<<2:PC |
| 15 | BGE | b'1100011 | b'101 | \ | PC=(rs>=rt)?PC+im<<2:PC |
| 16 | BLT | b'0110011 | b'100 | \ | PC=(rs<rt)?PC+im<<2:PC |

表5.1-1 16条指令

### 数据通路

数据通路文字描述和图示。

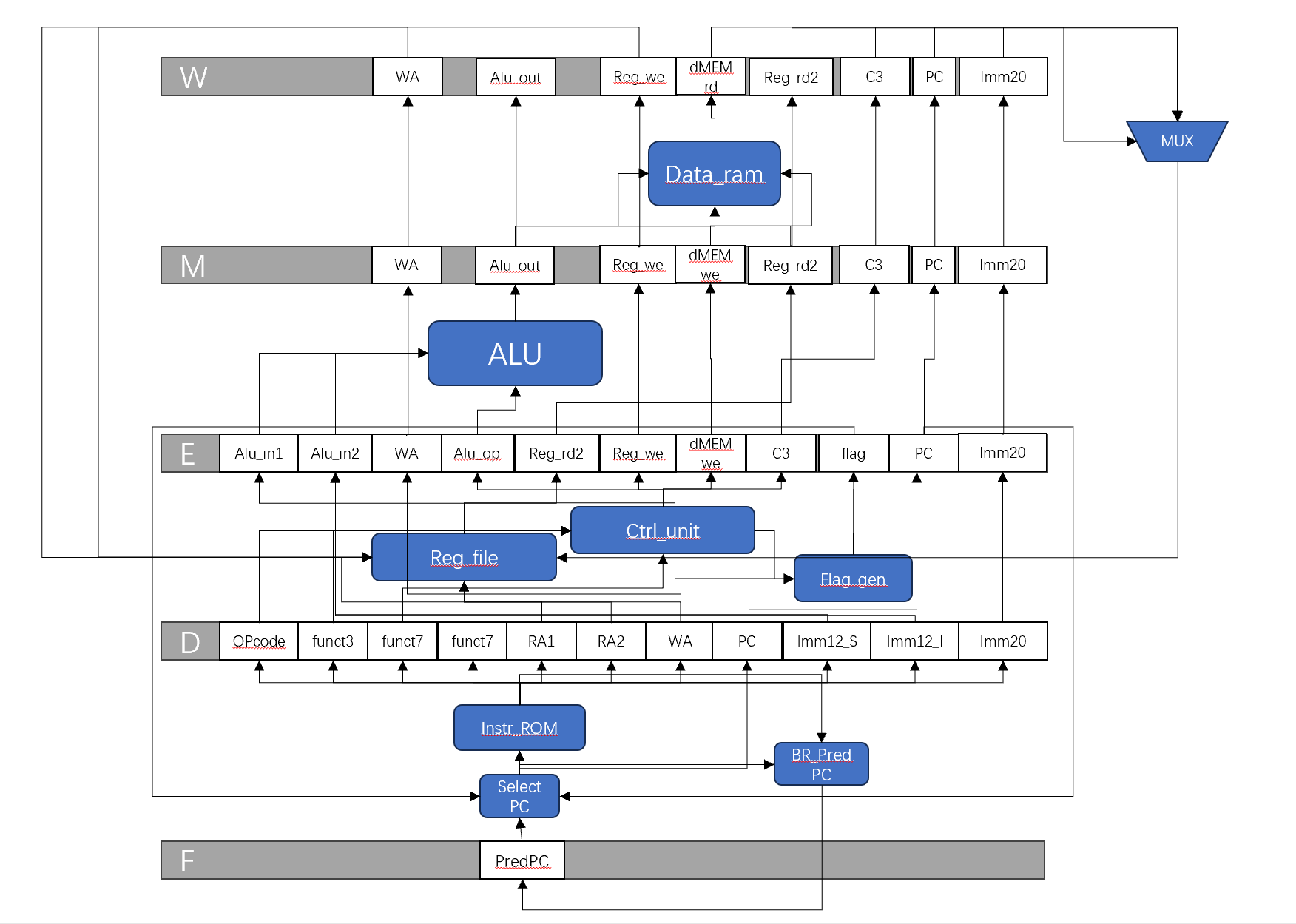


图5.1.1-1 5级流水线CPU数据通路

取指阶段：

*Select PC*完成当前pc为根据上一指令预测的pc还是分支预测错误恢复的pc\_E，取指阶段的select\_pc\_out会流入D流水线寄存器和指令寄存器inst\_rom中。

*Pred\_pc*需要指令的条件码以及当前周期的PC，输出pc\_pred即下一周期即将进入F阶段的指令地址。

*Inst\_rom*负责存储指令和读取下一条指令。

*Cpu\_top*在该阶段完成了基本的指令解码工作（向inst\_rom输入指令地址，从inst\_rom中获取指令信息，并根据指令格式解码获得不同信号值），使得decode阶段的时间消耗更小，提高流水线运行速度。

译码阶段：

*Control*模块在D阶段实现了解码与发送信号，即根据指令中各字段的值（opcode,funct3\_D, funct7\_D）来向后面的流水线寄存器传递信号（c1,c2,c3,alu\_op,branch,dmem\_we,reg\_we）。

*Register file*需要获得写使能(reg\_we)、读数据地址(reg\_ra)、写数据地址(reg\_wa)、写数据(reg\_wd)，读取内容为reg\_rd

单周期cpu中Mux1,mux2的工作在cpu\_top完成，实现了12位符号拓展进和alu\_in2的选择。

*Flag\_generator*需要两个从寄存器文件输出的变量reg\_rd1、reg\_rd2来完成分支预测正确性的判断，输出的flag\_D送入E阶段流水线寄存器。

执行阶段：

*Alu*接受来自E阶段的流水线寄存器，结果alu\_out作为M阶段流水线寄存器的输入。

*Flag*会在执行阶段返回给select PC模块。

访存阶段：

Data\_ram接受来自M阶段流水线寄存器的alu\_out\_M、dmem\_we\_M、reg\_rd2\_M，完成访问数据存储（LW指令）的功能，输出dmem\_rd存入W阶段流水线寄存器。

写回阶段：

在riscv\_top模块中完成mux3的选择过程，从data\_ram中读出的数据写回register file中。

### 控制逻辑

内容不限定以下内容：

示例：分析实现控制逻辑采用的方式：组合逻辑？微程序等。

控制信号列表，控制信号取值的含义，控制信号编码、时序分配、控制存储器内容，微指令编码等。

***控制信号列表：***

|  |  |  |
| --- | --- | --- |
| 控制信号 | 位宽 | 控制对象 |
| dmem\_we | 1 | dmem\_ram |
| Reg\_we | 1 | Regfile |
| Alu\_op | 3 | Alu |
| branch | 2 | Control |
| C1 | 1 | Mux1 |
| C2 | 1 | Mux2 |
| C3 | 2 | Mux3 |
| flag | 1 | Select PC |

***模块输入：***

opcode字段7位（instr[31:26]）

funct3字段3位（instr[14:12]）

funct7字段7位（instr[6:0]）

***模块输出：***

c1——I-type imm（instr[31:20]）和S-type imm（instr[31:25, 11:7]）之间2选1作为12位立即数

c2——12位立即数和rs2的值之间2选1作为ALU的第二个操作数

c3——20位立即数、当前PC+4、ALU运算结果、数据存储器读出数据4选1作为写回数据

alu\_op——ALU运算使能（加/减/逻辑或，同一时间点最多只能某一个运算使能置位）

branch——flag产生选择信号（条件分支/无条件跳转，同一时间点最多只能某一个跳转使能置位）

dmem\_we——数据存储器写使能

reg\_we——寄存器堆写使能

flag­——分支预测恢复信号，flag==1时预测正确，flag==0时分支预测错误

***编码：***

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| inst\_id | 指令 | C1 | C2 | C3 | alu\_op | branch | dmem\_we | reg\_we |
| 0 | null | 0 | 0 |  |  |  |  |  |
| 1 | ADD | 0 | 1 | b'10 | b'00001 | b'000 | 0 | 1 |
| 2 | SUB | 0 | 1 | b'10 | b'00010 | b'000 | 0 | 1 |
| 3 | ORI | 0 | 0 | b'10 | b'00100 | b'000 | 0 | 1 |
| 3 | LUI | 0 | 0 | b'00 | b'00000 | b'000 | 0 | 1 |
| 3 | SW | 1 | 0 | b'00 | b'00001 | b'000 | 1 | 0 |
| 3 | LW | 0 | 0 | b'11 | b'00001 | b'000 | 0 | 1 |
| 3 | BEQ | 0 | 0 | b'00 | b'00000 | b'001 | 0 | 0 |
| 3 | JAL | 0 | 0 | b'01 | b'00000 | b'010 | 0 | 1 |
| 3 | XORI | 0 | 0 | b'10 | b'10000 | b'000 | 0 | 1 |
| 3 | ANDI | 0 | 0 | b'10 | b'01000 | b'000 | 0 | 1 |
| 3 | XOR | 0 | 1 | b'10 | b'10000 | b'000 | 0 | 1 |
| 3 | OR | 0 | 1 | b'10 | b'00100 | b'000 | 0 | 1 |
| 3 | AND | 0 | 1 | b'10 | b'01000 | b'000 | 0 | 1 |
| 3 | BNE | 0 | 0 | b'00 | b'00000 | b'011 | 0 | 0 |
| 3 | BGE | 0 | 0 | b'00 | b'00000 | b'100 | 0 | 0 |
| 3 | BLT | 0 | 0 | b'00 | b'00000 | b'101 | 0 | 0 |

注：flag与分支条件的计算结果有关，并不是可以预设的。

## 流水线冒险问题以及解决方案

控制冒险方面，采取的策略是默认跳转，错误后对分支进行恢复，这样便利多次循环控制流，使用select PC和Pred PC来实现控制冒险。

因为最早在译码阶段才可以得到分支判断条件的正误值，故决定将flag传入E阶段的流水线寄存器中，在下一阶段的上升沿将数据传入select PC模块中，在select PC模块中使用assign select\_pc\_out = flag\_E ? pc\_out : pc\_E +32'h4来完成PC选择的操作。

数据冒险方面，采用插入NOP指令的方式，缓冲流水线，避免RW冒险。

## Cache的设计（可选）

## 中断的设计（可选）

## 其他特色（可选）

# 设计与实现

这里将各模块的代码放入，并且给出模块间的关系。

***代码部分：***

Macro.vh:

1. // opcode
2. `define OPCODE\_R    7'b0110011
3. `define OPCODE\_I    7'b0010011
4. `define OPCODE\_L    7'b0000011
5. `define OPCODE\_S    7'b0100011
6. `define OPCODE\_B    7'b1100011
7. `define OPCODE\_JAL  7'b1101111
8. `define OPCODE\_LUI  7'b0110111
9. // funct3
10. `define FUNCT3\_ADDSUB   3'b000
11. `define FUNCT3\_ORI      3'b110
12. `define FUNCT3\_LW       3'b010
13. `define FUNCT3\_SW       3'b010
14. `define FUNCT3\_BEQ      3'b000
15. `define FUNCT3\_AND      3'b111
16. `define FUNCT3\_XOR      3'b100
17. `define FUNCT3\_BNE      3'b001
18. `define FUNCT3\_BGE      3'b101
19. `define FUNCT3\_BLT      3'b100
20. // funct7
21. `define FUNCT7\_ADD  7'b0000000
22. `define FUNCT7\_SUB  7'b0100000
23. // inst\_id
24. `define ID\_NULL     0
25. `define ID\_ADD      1
26. `define ID\_SUB      2
27. `define ID\_ORI      3
28. `define ID\_LUI      4
29. `define ID\_SW       5
30. `define ID\_LW       6
31. `define ID\_BEQ      7
32. `define ID\_JAL      8
33. // I类
34. `define ID\_XORI     9
35. `define ID\_ANDI     10
36. // R类
37. `define ID\_XOR     11
38. `define ID\_OR       12
39. `define ID\_AND      13
40. // B类
41. `define ID\_BNE      14
42. `define ID\_BGE      15
43. `define ID\_BLT      16
44. // alu\_op
45. `define ALU\_NULL    5'b00000
46. `define ALU\_ADD     5'b00001
47. `define ALU\_SUB     5'b00010
48. `define ALU\_OR      5'b00100
49. `define ALU\_AND     5'b01000
50. `define ALU\_XOR     5'b10000
51. // branch
52. `define BR\_NULL     3'b000
53. `define BR\_BEQ      3'b001
54. `define BR\_JAL      3'b010
55. `define BR\_BNE     3'b011
56. `define BR\_BGE      3'b100
57. `define BR\_BLT      3'b101

Riscv\_top.v

1. `timescale 1ns / 1ps
2. //////////////////////////////////////////////////////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 2023/05/12 01:41:49
7. // Design Name:
8. // Module Name: riscv\_top
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////////////////////////////////////////////////
21. module mycpu(
22. input         rstn,
23. input         clk,
24. output [31:0] inst\_rom\_addr,
25. input  [31:0] inst\_rom\_rdata,
26. output [31:0] data\_ram\_addr,
27. output [31:0] data\_ram\_wdata,
28. output        data\_ram\_wen,
29. input  [31:0] data\_ram\_rdata
30. );
32. wire flag\_D;
33. wire [31:0] pc\_out;
34. wire [31:0] pc\_pred;
35. wire [31:0] select\_pc\_out;
36. wire [31:0] instr;
38. wire [6:0]  opcode;
39. wire [2:0]  funct3;
40. wire [6:0]  funct7;
41. wire        c1;
42. wire        c2;
43. wire [1:0]  c3;
44. wire [4:0]  alu\_op;
45. wire [2:0]  branch;
46. wire        dmem\_we;
47. wire        reg\_we;
49. wire [4:0]  reg\_ra1;
50. wire [4:0]  reg\_ra2;
51. wire [31:0] reg\_rd1;
52. wire [31:0] reg\_rd2;
53. wire [4:0]  reg\_wa;
54. wire [19:0] imm20\_u;
55. wire [31:0] imm20\_e32;
56. wire [31:0] pc\_plus4;
57. wire [31:0] reg\_wd;
59. wire [11:0] imm12\_s;
60. wire [11:0] imm12\_i;
61. wire [31:0] imm12\_e32;
62. wire [31:0] alu\_in1;
63. wire [31:0] alu\_in2;
64. wire [31:0] alu\_out;
65. wire [31:0] dmem\_rd;
67. wire [11:0] imm12\_b;
68. wire [19:0] imm20\_j;
69. wire [31:0] offset;
71. wire [31:0] pc\_D;
72. wire [6:0] opcode\_D;
73. wire [2:0] funct3\_D;
74. wire [6:0] funct7\_D;
75. wire [4:0] reg\_ra1\_D;
76. wire [4:0] reg\_ra2\_D;
77. wire [4:0] reg\_wa\_D;
78. wire [11:0] imm12\_s\_D;
79. wire [11:0] imm12\_i\_D;
80. wire [31:0] imm20\_e32\_D;
82. wire [31:0] pc\_E;
83. wire [31:0] alu\_in1\_E;
84. wire [31:0] alu\_in2\_E;
85. wire flag\_E;
86. wire [4:0] reg\_wa\_E;
87. wire [4:0] alu\_op\_E;
88. wire [31:0] imm20\_e32\_E;
89. wire [1:0] c3\_E;
90. wire dmem\_we\_E;
91. wire reg\_we\_E;
92. wire [31:0] reg\_rd2\_E;
94. wire [31:0] pc\_M;
95. wire [31:0] alu\_out\_M;
96. wire [4:0]  reg\_wa\_M;
97. wire [31:0] imm20\_e32\_M;
98. wire [1:0]  c3\_M;
99. wire        dmem\_we\_M;
100. wire        reg\_we\_M;
101. wire [31:0] reg\_rd2\_M;
103. wire [31:0] pc\_W;
104. wire [31:0] alu\_out\_W;
105. wire [4:0]  reg\_wa\_W;
106. wire [31:0] imm20\_e32\_W;
107. wire [1:0]  c3\_W;
108. wire        reg\_we\_W;
109. wire [31:0] dmem\_rd\_W;
110. //assign pc\_in = pc\_out + (jump ? offset : 32'h4);
112. assign select\_pc\_out = flag\_E ? pc\_out : pc\_E +32'h4;
114. assign inst\_rom\_addr = select\_pc\_out;
115. assign instr = inst\_rom\_rdata;
116. assign funct7 = instr[31:25];
117. assign funct3 = instr[14:12];
118. assign opcode = instr[6:0];
119. assign reg\_ra1 = instr[19:15];
120. assign reg\_ra2 = instr[24:20];
121. assign reg\_wa = instr[11:7];
123. assign imm20\_u = instr[31:12];
124. assign imm20\_e32 = {imm20\_u, 12'b0};
125. assign pc\_plus4 = pc\_W + 32'h4;
126. assign reg\_wd = sel\_2to4(c3\_W, imm20\_e32\_W, pc\_plus4, alu\_out\_W, dmem\_rd\_W);
127. function [31:0] sel\_2to4(input [1:0] sel,
128. input [31:0] in0, input [31:0] in1,
129. input [31:0] in2, input [31:0] in3);
130. begin
131. case (sel)
132. 2'b00 : sel\_2to4 = in0;
133. 2'b01 : sel\_2to4 = in1;
134. 2'b10 : sel\_2to4 = in2;
135. 2'b11 : sel\_2to4 = in3;
136. endcase
137. end
138. endfunction
140. assign alu\_in1 = reg\_rd1;
141. assign imm12\_s = {instr[31:25], instr[11:7]};
142. assign imm12\_i = instr[31:20];
143. assign imm12\_e32 = c1 ? {{20{imm12\_s\_D[11]}}, imm12\_s\_D} : {{20{imm12\_i\_D[11]}}, imm12\_i\_D};
144. assign alu\_in2 = c2 ? reg\_rd2 : imm12\_e32;
146. assign data\_ram\_addr = alu\_out\_M;
147. assign data\_ram\_wen = dmem\_we\_M;
148. assign data\_ram\_wdata = reg\_rd2\_M;
149. assign dmem\_rd = data\_ram\_rdata;
150. assign imm12\_b = {instr[31], instr[7], instr[30:25], instr[11:8]};
151. assign imm20\_j = {instr[31], instr[19:12], instr[20], instr[30:21]};
152. pc \_pc(
153. .clk(clk),
154. .rst(rstn),
155. .pc\_pred(pc\_pred),
156. .pc\_sel\_in(pc\_out)
157. );
159. pipeline\_reg\_D \_pipeline\_reg\_D(
160. .rst(rstn),
161. .clk(clk),
162. .nop(flag\_E),
163. .pc\_D(select\_pc\_out),
164. .opcode(opcode),
165. .funct3(funct3),
166. .funct7(funct7),
167. .reg\_ra1(reg\_ra1),
168. .reg\_ra2(reg\_ra2),
169. .reg\_wa(reg\_wa),
170. .imm12\_s(imm12\_s),
171. .imm12\_i(imm12\_i),
172. .imm20\_e32(imm20\_e32),
173. .\_pc\_D(pc\_D),
174. .\_opcode(opcode\_D),
175. .\_funct3(funct3\_D),
176. .\_funct7(funct7\_D),
177. .\_reg\_ra1(reg\_ra1\_D),
178. .\_reg\_ra2(reg\_ra2\_D),
179. .\_reg\_wa(reg\_wa\_D),
180. .\_imm12\_s(imm12\_s\_D),
181. .\_imm12\_i(imm12\_i\_D),
182. .\_imm20\_e32(imm20\_e32\_D)
183. );
184. control \_cu(
185. .opcode(opcode\_D),
186. .funct3(funct3\_D),
187. .funct7(funct7\_D),
188. .c1(c1),
189. .c2(c2),
190. .c3(c3),
191. .alu\_op(alu\_op),
192. .branch(branch),
193. .dmem\_we(dmem\_we),
194. .reg\_we(reg\_we)
195. );
197. regfile \_regfile(
198. .clk(clk),
199. .rst(rstn),
200. .reg\_we(reg\_we\_W),
201. .reg\_ra1(reg\_ra1\_D),
202. .reg\_ra2(reg\_ra2\_D),
203. .reg\_wa(reg\_wa\_W),
204. .reg\_wd(reg\_wd),
205. .reg\_rd1(reg\_rd1),
206. .reg\_rd2(reg\_rd2)
207. );

210. pipeline\_reg\_E \_pipeline\_reg\_E(
211. .rst(rstn),
212. .clk(clk),
213. .pc\_E(pc\_D),
214. .alu\_in1(alu\_in1),
215. .alu\_in2(alu\_in2),
216. .flag(flag\_D),
217. .reg\_wa(reg\_wa\_D),
218. .alu\_op(alu\_op),
219. .imm20\_e32(imm20\_e32\_D),
220. .c3(c3),
221. .dmem\_we(dmem\_we),
222. .reg\_we(reg\_we),
223. .reg\_rd2(reg\_rd2),
224. .\_pc\_E(pc\_E),
225. .\_alu\_in1(alu\_in1\_E),
226. .\_alu\_in2(alu\_in2\_E),
227. .\_flag(flag\_E),
228. .\_reg\_wa(reg\_wa\_E),
229. .\_alu\_op(alu\_op\_E),
230. .\_imm20\_e32(imm20\_e32\_E),
231. .\_c3(c3\_E),
232. .\_dmem\_we(dmem\_we\_E),
233. .\_reg\_we(reg\_we\_E),
234. .\_reg\_rd2(reg\_rd2\_E)
235. );
236. alu \_alu(
237. .alu\_op(alu\_op\_E),
238. .alu\_in1(alu\_in1\_E),
239. .alu\_in2(alu\_in2\_E),
240. .alu\_out(alu\_out)
241. );
243. pipeline\_reg\_M \_pipeline\_reg\_M(
244. .rst(rstn),
245. .clk(clk),
246. .pc\_M(pc\_E),
247. .alu\_out(alu\_out),
248. .reg\_wa(reg\_wa\_E),
249. .imm20\_e32(imm20\_e32\_E),
250. .c3(c3\_E),
251. .dmem\_we(dmem\_we\_E),
252. .reg\_we(reg\_we\_E),
253. .reg\_rd2(reg\_rd2\_E),
254. .\_pc\_M(pc\_M),
255. .\_alu\_out(alu\_out\_M),
256. .\_reg\_wa(reg\_wa\_M),
257. .\_imm20\_e32(imm20\_e32\_M),
258. .\_c3(c3\_M),
259. .\_dmem\_we(dmem\_we\_M),
260. .\_reg\_we(reg\_we\_M),
261. .\_reg\_rd2(reg\_rd2\_M)
262. );

265. pipeline\_reg\_W \_pipeline\_reg\_W(
266. .rst(rstn),
267. .clk(clk),
268. .pc\_W(pc\_M),
269. .alu\_out(alu\_out\_M),
270. .reg\_wa(reg\_wa\_M),
271. .imm20\_e32(imm20\_e32\_M),
272. .c3(c3\_M),
273. .reg\_we(reg\_we\_M),
274. .dmem\_rd(dmem\_rd),
275. .\_pc\_W(pc\_W),
276. .\_alu\_out(alu\_out\_W),
277. .\_reg\_wa(reg\_wa\_W),
278. .\_imm20\_e32(imm20\_e32\_W),
279. .\_c3(c3\_W),
280. .\_reg\_we(reg\_we\_W),
281. .\_dmem\_rd(dmem\_rd\_W)
282. );
284. //    br \_br(
285. //        .branch(branch),
286. //        .imm12\_b(imm12\_b),
287. //        .imm20\_j(imm20\_j),
288. //        .reg\_rd1(reg\_rd1),
289. //        .reg\_rd2(reg\_rd2),
290. //        .jump(jump),
291. //        .offset(offset)
292. //    );
294. pred\_pc \_pred\_pc(
295. .imm12\_b(imm12\_b),
296. .imm20\_j(imm20\_j),
297. .opcode(opcode),
298. .select\_pc\_out(select\_pc\_out),
299. .pc\_pred(pc\_pred)
300. );
302. flag\_gen \_flag\_gen(
303. .branch(branch),
304. .reg\_rd1(reg\_rd1),
305. .reg\_rd2(reg\_rd2),
306. .flag(flag\_D)
307. );
308. endmodule

***control.v:***

1. `timescale 1ns / 1ps
2. //////////////////////////////////////////////////////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 2020/11/19 21:13:55
7. // Design Name:
8. // Module Name: control
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////////////////////////////////////////////////
21. `include "macro.vh"
22. module control(
23. //    input           clk,
24. //    input           rst,
25. input   [6:0]   opcode,
26. input   [2:0]   funct3,
27. input   [6:0]   funct7,
29. output          c1,
30. output          c2,
31. output  [1:0]   c3,
32. output  [4:0]   alu\_op,
33. output  [2:0]   branch,
34. output          dmem\_we,
35. output          reg\_we
36. );
37. wire [3:0] inst\_id = get\_inst\_id(opcode, funct3, funct7);
38. function [3:0] get\_inst\_id(input [6:0] opcode, input [2:0] funct3, input [6:0] funct7);
39. begin
40. case (opcode)
41. `OPCODE\_R   : begin
42. case (funct3)
43. `FUNCT3\_ADDSUB  : begin
44. case (funct7)
45. `FUNCT7\_ADD : get\_inst\_id = `ID\_ADD;
46. `FUNCT7\_SUB : get\_inst\_id = `ID\_SUB;
47. default     : get\_inst\_id = `ID\_NULL;
48. endcase
49. end
50. `FUNCT3\_ORI : get\_inst\_id = `ID\_OR;
51. `FUNCT3\_XOR : get\_inst\_id = `ID\_XOR;
52. `FUNCT3\_AND : get\_inst\_id = `ID\_AND;
53. default     : get\_inst\_id = `ID\_NULL;
54. endcase
55. end
56. `OPCODE\_I   : begin
57. case (funct3)
58. `FUNCT3\_ORI : get\_inst\_id = `ID\_ORI;
59. `FUNCT3\_XOR : get\_inst\_id = `ID\_XORI;
60. `FUNCT3\_AND : get\_inst\_id = `ID\_ANDI;
61. default     : get\_inst\_id = `ID\_NULL;
62. endcase
63. end
64. `OPCODE\_L   : begin
65. case (funct3)
66. `FUNCT3\_LW  : get\_inst\_id = `ID\_LW;
67. default     : get\_inst\_id = `ID\_NULL;
68. endcase
69. end
70. `OPCODE\_S   : begin
71. case (funct3)
72. `FUNCT3\_SW  : get\_inst\_id = `ID\_SW;
73. default     : get\_inst\_id = `ID\_NULL;
74. endcase
75. end
76. `OPCODE\_B   : begin
77. case (funct3)
78. `FUNCT3\_BEQ : get\_inst\_id = `ID\_BEQ;
79. `FUNCT3\_BNE : get\_inst\_id = `ID\_BNE;
80. `FUNCT3\_BGE : get\_inst\_id = `ID\_BGE;
81. `FUNCT3\_BLT : get\_inst\_id = `ID\_BLT;
82. default     : get\_inst\_id = `ID\_NULL;
83. endcase
84. end
85. `OPCODE\_JAL : get\_inst\_id = `ID\_JAL;
86. `OPCODE\_LUI : get\_inst\_id = `ID\_LUI;
87. default     : get\_inst\_id = `ID\_NULL;
88. endcase
89. end
90. endfunction
91. assign c1 = (inst\_id == `ID\_SW ) ? 1 : 0;
93. reg [16:0] mask\_c2 = 17'b00011100000000110;
94. assign c2 = mask\_c2[inst\_id];
96. assign c3 = get\_c3(inst\_id);
97. function [1:0] get\_c3(input [3:0] inst\_id);
98. begin
99. case (inst\_id)
100. `ID\_LUI, `ID\_SW, `ID\_BEQ, `ID\_BGE, `ID\_BNE, `ID\_BLT  : get\_c3 = 2'b00;
101. `ID\_JAL                   : get\_c3 = 2'b01;
102. `ID\_ADD, `ID\_SUB, `ID\_ORI, `ID\_XORI, `ID\_XOR, `ID\_OR,
103. `ID\_AND, `ID\_ANDI : get\_c3 = 2'b10;
104. `ID\_LW                    : get\_c3 = 2'b11;
105. default                   : get\_c3 = 2'b00;
106. endcase
107. end
108. endfunction
110. assign alu\_op = get\_alu\_op(inst\_id);
111. function [4:0] get\_alu\_op(input [3:0] inst\_id);
112. begin
113. case (inst\_id)
114. `ID\_SW, `ID\_LW, `ID\_ADD : get\_alu\_op = `ALU\_ADD;
115. `ID\_SUB                 : get\_alu\_op = `ALU\_SUB;
116. `ID\_ORI, `ID\_OR                 : get\_alu\_op = `ALU\_OR;
117. `ID\_XOR, `ID\_XORI             : get\_alu\_op = `ALU\_XOR;
118. `ID\_AND, `ID\_ANDI            : get\_alu\_op = `ALU\_AND;
119. default                 : get\_alu\_op = `ALU\_NULL;
120. endcase
121. end
122. endfunction
124. assign branch = get\_branch(inst\_id);
125. function [2:0] get\_branch(input [3:0] inst\_id);
126. begin
127. case (inst\_id)
128. `ID\_BEQ                 : get\_branch = `BR\_BEQ;
129. `ID\_JAL                 : get\_branch = `BR\_JAL;
130. `ID\_BNE                 : get\_branch = `BR\_BNE;
131. `ID\_BGE                 : get\_branch = `BR\_BGE;
132. `ID\_BLT                 : get\_branch = `BR\_BLT;
133. default                 : get\_branch = `BR\_NULL;
134. endcase
135. end
136. endfunction
138. assign dmem\_we = (inst\_id == `ID\_SW ) ? 1 : 0;
140. reg [16:0] mask\_reg\_we = 17'b00011111101011110;
141. assign reg\_we = mask\_reg\_we[inst\_id];
143. endmodule

***alu.v:***

1. `timescale 1ns / 1ps
2. *//////////////////////////////////////////////////////////////////////////////////*
3. *// Company:*
4. *// Engineer:*
5. *//*
6. *// Create Date: 2020/11/19 21:13:55*
7. *// Design Name:*
8. *// Module Name: alu*
9. *// Project Name:*
10. *// Target Devices:*
11. *// Tool Versions:*
12. *// Description:*
13. *//*
14. *// Dependencies:*
15. *//*
16. *// Revision:*
17. *// Revision 0.01 - File Created*
18. *// Additional Comments:*
19. *//*
20. *//////////////////////////////////////////////////////////////////////////////////*
21. `include "macro.vh"
22. module alu(
23. *//    input  wire        clk,*
24. *//    input  wire        rst,*
26. input  wire [4:0]  alu\_op,
27. input  wire [31:0] alu\_in1,
28. input  wire [31:0] alu\_in2,
30. output wire [31:0] alu\_out
31. );
33. reg err;
34. reg zero;
36. wire [31:0] in1   = alu\_in1;
37. wire [31:0] in2   = alu\_in2;
39. wire [32:0] in1\_e = {alu\_in1[31], alu\_in1};
40. wire [32:0] in2\_e = {alu\_in2[31], alu\_in2};
42. wire [32:0] o\_add = in1\_e + in2\_e;
43. wire [32:0] o\_sub = in1\_e - in2\_e;
45. wire [31:0] o\_or  = in1 | in2;
46. wire [31:0] o\_and = in1 & in2;
47. wire [31:0] o\_xor = in1 ^ in2;
48. assign alu\_out = (alu\_op == `ALU\_ADD) ? o\_add[31:0] :
49. (alu\_op == `ALU\_SUB) ? o\_sub[31:0] :
50. (alu\_op == `ALU\_OR ) ? o\_or :
51. (alu\_op == `ALU\_AND) ? o\_and :
52. (alu\_op == `ALU\_XOR) ? o\_xor : 32'b0;
54. endmodule

***flag\_gen.v:***

1. `timescale 1ns / 1ps
2. *//////////////////////////////////////////////////////////////////////////////////*
3. *// Company:*
4. *// Engineer:*
5. *//*
6. *// Create Date: 2020/11/19 21:13:55*
7. *// Design Name:*
8. *// Module Name: alu*
9. *// Project Name:*
10. *// Target Devices:*
11. *// Tool Versions:*
12. *// Description:*
13. *//*
14. *// Dependencies:*
15. *//*
16. *// Revision:*
17. *// Revision 0.01 - File Created*
18. *// Additional Comments:*
19. *//*
20. *//////////////////////////////////////////////////////////////////////////////////*
21. `include "macro.vh"
22. module alu(
23. *//    input  wire        clk,*
24. *//    input  wire        rst,*
26. input  wire [4:0]  alu\_op,
27. input  wire [31:0] alu\_in1,
28. input  wire [31:0] alu\_in2,
30. output wire [31:0] alu\_out
31. );
33. reg err;
34. reg zero;
36. wire [31:0] in1   = alu\_in1;
37. wire [31:0] in2   = alu\_in2;
39. wire [32:0] in1\_e = {alu\_in1[31], alu\_in1};
40. wire [32:0] in2\_e = {alu\_in2[31], alu\_in2};
42. wire [32:0] o\_add = in1\_e + in2\_e;
43. wire [32:0] o\_sub = in1\_e - in2\_e;
45. wire [31:0] o\_or  = in1 | in2;
46. wire [31:0] o\_and = in1 & in2;
47. wire [31:0] o\_xor = in1 ^ in2;
48. assign alu\_out = (alu\_op == `ALU\_ADD) ? o\_add[31:0] :
49. (alu\_op == `ALU\_SUB) ? o\_sub[31:0] :
50. (alu\_op == `ALU\_OR ) ? o\_or :
51. (alu\_op == `ALU\_AND) ? o\_and :
52. (alu\_op == `ALU\_XOR) ? o\_xor : 32'b0;
54. endmodule

***regfile.v:***

1. `timescale 1ns / 1ps
2. *//////////////////////////////////////////////////////////////////////////////////*
3. *// Company:*
4. *// Engineer:*
5. *//*
6. *// Create Date: 2020/11/19 21:13:55*
7. *// Design Name:*
8. *// Module Name: regfile*
9. *// Project Name:*
10. *// Target Devices:*
11. *// Tool Versions:*
12. *// Description:*
13. *//*
14. *// Dependencies:*
15. *//*
16. *// Revision:*
17. *// Revision 0.01 - File Created*
18. *// Additional Comments:*
19. *//*
20. *//////////////////////////////////////////////////////////////////////////////////*
21. module regfile(
22. input  wire       clk,
23. input  wire       rst,
24. input  wire[4:0]  reg\_ra1,
25. input  wire[4:0]  reg\_ra2,
27. input  wire       reg\_we,
28. input  wire[4:0]  reg\_wa,
29. input  wire[31:0] reg\_wd,
31. output wire[31:0] reg\_rd1,
32. output wire[31:0] reg\_rd2
33. );
35. reg [31:0] regs[31:0];
37. assign reg\_rd1 = reg\_ra1 == 5'b0 ? 32'b0 : regs[reg\_ra1];
38. assign reg\_rd2 = reg\_ra2 == 5'b0 ? 32'b0 : regs[reg\_ra2];
40. integer i;
41. always @(posedge clk or negedge rst) begin
42. if (!rst) begin
43. for (i = 0; i <= 31; i = i + 1)
44. regs[i] <= 32'b0;
45. end
46. else if (reg\_we == 1'b1)
47. regs[reg\_wa] <= reg\_wd;
48. end
50. endmodule

流水线寄存器：

***pc.v: (also means pipeline\_F)***

1. `timescale 1ns / 1ps
2. *//////////////////////////////////////////////////////////////////////////////////*
3. *// Company:*
4. *// Engineer:*
5. *//*
6. *// Create Date: 2020/11/20 20:35:35*
7. *// Design Name:*
8. *// Module Name: pc*
9. *// Project Name:*
10. *// Target Devices:*
11. *// Tool Versions:*
12. *// Description:*
13. *//*
14. *// Dependencies:*
15. *//*
16. *// Revision:*
17. *// Revision 0.01 - File Created*
18. *// Additional Comments:*
19. *//*
20. *//////////////////////////////////////////////////////////////////////////////////*
21. module pc(
22. input  wire         clk,
23. input  wire         rst,
25. input  wire [31:0]  pc\_pred,
26. output wire [31:0]  pc\_sel\_in
27. );
29. reg [31:0] pc\_reg;
30. always @(posedge clk or negedge rst) begin
31. if (!rst) pc\_reg <= 32'h00000000;
32. else pc\_reg <= pc\_pred;
33. end
35. assign pc\_sel\_in = pc\_reg;
37. endmodule

***Pipeline\_reg\_D:***

1. `timescale 1ns / 1ps
2. //////////////////////////////////////////////////////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 2023/09/03 14:46:57
7. // Design Name:
8. // Module Name: pipeline\_reg\_D
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////////////////////////////////////////////////
21. module pipeline\_reg\_D(
22. input rst,
23. input clk,
24. input nop,
26. input wire [31:0] pc\_D         ,
27. input wire [6:0]  opcode      ,
28. input wire [2:0]  funct3        ,
29. input wire [6:0]  funct7         ,
30. input wire [4:0]  reg\_ra1       ,
31. input wire [4:0]  reg\_ra2       ,
32. input wire [4:0]  reg\_wa     ,
33. input wire [11:0] imm12\_s   ,
34. input wire [11:0] imm12\_i   ,
35. input wire [31:0] imm20\_e32 ,//写回阶段需要的变量
37. output  wire [31:0] \_pc\_D      ,
38. output  wire [6:0]  \_opcode    ,
39. output  wire [2:0]  \_funct3    ,
40. output  wire [6:0]  \_funct7    ,
41. output  wire [4:0]  \_reg\_ra1   ,
42. output  wire [4:0]  \_reg\_ra2   ,
43. output  wire [4:0]  \_reg\_wa    ,
44. output  wire [11:0] \_imm12\_s   ,
45. output  wire [11:0] \_imm12\_i   ,
46. output  wire [31:0] \_imm20\_e32
47. );
48. reg    [31:0]   pc\_D\_reg     ;
49. reg    [6:0]    opcode\_reg   ;
50. reg    [31:0]   pc\_D\_nop     ;
51. reg    [6:0]    opcode\_nop   ;
52. reg    [2:0]    funct3\_reg   ;
53. reg    [6:0]    funct7\_reg   ;
54. reg    [4:0]    reg\_ra1\_reg  ;
55. reg    [4:0]    reg\_ra2\_reg  ;
56. reg    [4:0]    reg\_wa\_reg   ;
57. reg    [11:0]   imm12\_s\_reg  ;
58. reg    [11:0]   imm12\_i\_reg  ;
59. reg    [31:0]   imm20\_e32\_reg;
61. always @(posedge clk or negedge rst) begin
62. if (!rst) begin
63. pc\_D\_reg <= 32'h00000000;
64. pc\_D\_nop <=  32'h00000000;
65. opcode\_nop <= 7'b0000000;
66. end
67. else begin
68. pc\_D\_reg      <= pc\_D     ;
69. opcode\_reg    <= opcode   ;
70. funct3\_reg    <= funct3   ;
71. funct7\_reg    <= funct7   ;
72. reg\_ra1\_reg   <= reg\_ra1  ;
73. reg\_ra2\_reg   <= reg\_ra2  ;
74. reg\_wa\_reg    <= reg\_wa   ;
75. imm12\_s\_reg   <= imm12\_s  ;
76. imm12\_i\_reg   <= imm12\_i  ;
77. imm20\_e32\_reg <= imm20\_e32;
78. end
79. //pc\_reg <= pc\_in;
80. end


84. assign \_pc\_D      = nop ? pc\_D\_reg : pc\_D\_nop  ;
85. assign \_opcode    = nop ? opcode\_reg : opcode\_nop  ;
86. assign \_funct3    = funct3\_reg   ;
87. assign \_funct7    = funct7\_reg   ;
88. assign \_reg\_ra1   = reg\_ra1\_reg  ;
89. assign \_reg\_ra2   = reg\_ra2\_reg  ;
90. assign \_reg\_wa    = reg\_wa\_reg   ;
91. assign \_imm12\_s   = imm12\_s\_reg  ;
92. assign \_imm12\_i   = imm12\_i\_reg  ;
93. assign \_imm20\_e32 = imm20\_e32\_reg;
94. endmodule

***pipeline\_reg\_E.v:***

1. `timescale 1ns / 1ps
2. //////////////////////////////////////////////////////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 2023/09/03 14:56:42
7. // Design Name:
8. // Module Name: pipeline\_reg\_E
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////////////////////////////////////////////////
21. module pipeline\_reg\_E(
22. input rst,
23. input clk,
25. input wire [31:0] pc\_E          ,
26. input wire [31:0] alu\_in1       ,
27. input wire [31:0] alu\_in2       ,
28. input               flag         ,
29. input wire [4:0]  reg\_wa        ,
30. input wire [4:0]  alu\_op        ,
31. input wire [31:0] imm20\_e32     ,
32. input wire [1:0]  c3            ,
33. input wire        dmem\_we       ,
34. input wire        reg\_we        ,
35. input wire [31:0] reg\_rd2       ,
37. output wire [31:0] \_pc\_E          ,
38. output wire [31:0] \_alu\_in1       ,
39. output wire [31:0] \_alu\_in2       ,
40. output              \_flag         ,
41. output wire [4:0]  \_reg\_wa        ,
42. output wire [4:0]  \_alu\_op        ,
43. output wire [31:0] \_imm20\_e32     ,
44. output wire [1:0]  \_c3            ,
45. output wire        \_dmem\_we       ,
46. output wire        \_reg\_we        ,
47. output wire [31:0] \_reg\_rd2
49. );
50. reg [31:0] pc\_E\_reg     ;
51. reg [31:0] alu\_in1\_reg  ;
52. reg [31:0] alu\_in2\_reg  ;
53. reg          flag\_reg   ;
54. reg [4:0]  reg\_wa\_reg   ;
55. reg [4:0]  alu\_op\_reg   ;
56. reg [31:0] imm20\_e32\_reg;
57. reg [1:0]  c3\_reg       ;
58. reg        dmem\_we\_reg  ;
59. reg        reg\_we\_reg   ;
60. reg [31:0] reg\_rd2\_reg  ;
62. always @(posedge clk or negedge rst) begin
63. if(!rst) begin
64. pc\_E\_reg <= 32'h00000000;
65. flag\_reg  <= 32'b1;
66. end
67. else begin
68. pc\_E\_reg      <= pc\_E     ;
69. alu\_in1\_reg   <= alu\_in1  ;
70. alu\_in2\_reg   <= alu\_in2  ;
71. flag\_reg    <=   flag   ;
72. reg\_wa\_reg    <= reg\_wa   ;
73. alu\_op\_reg    <= alu\_op   ;
74. imm20\_e32\_reg <= imm20\_e32;
75. c3\_reg        <= c3       ;
76. dmem\_we\_reg   <= dmem\_we  ;
77. reg\_we\_reg    <= reg\_we   ;
78. reg\_rd2\_reg   <= reg\_rd2  ;
79. end
80. end
81. assign \_pc\_E     = pc\_E\_reg     ;
82. assign \_alu\_in1  = alu\_in1\_reg  ;
83. assign \_alu\_in2  = alu\_in2\_reg  ;
84. assign  \_flag    =   flag\_reg   ;
85. assign \_reg\_wa   = reg\_wa\_reg   ;
86. assign \_alu\_op   = alu\_op\_reg   ;
87. assign \_imm20\_e32= imm20\_e32\_reg;
88. assign \_c3       = c3\_reg       ;
89. assign \_dmem\_we  = dmem\_we\_reg  ;
90. assign \_reg\_we   = reg\_we\_reg   ;
91. assign \_reg\_rd2  = reg\_rd2\_reg  ;
92. endmodule

***pipeline\_reg\_M.v:***

1. `timescale 1ns / 1ps
2. //////////////////////////////////////////////////////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 2023/09/03 15:13:47
7. // Design Name:
8. // Module Name: pipeline\_reg\_M
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////////////////////////////////////////////////
21. module pipeline\_reg\_M(
22. input rst                   ,
23. input clk                   ,

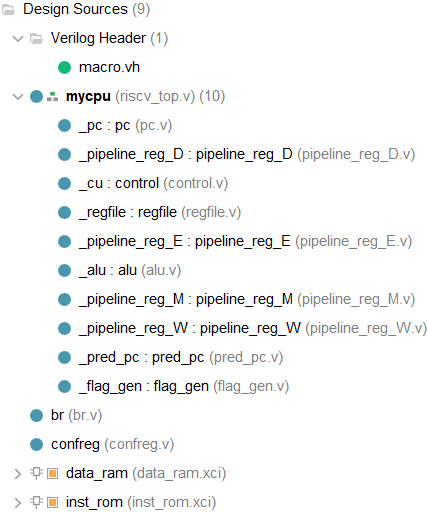
26. input wire [31:0] pc\_M      ,
27. input wire [31:0] alu\_out   ,
28. input wire [4:0]  reg\_wa    ,
29. input wire [31:0] imm20\_e32 ,
30. input wire [1:0]  c3        ,
31. input wire        dmem\_we   ,
32. input wire        reg\_we    ,
33. input wire [31:0] reg\_rd2   ,
35. output wire [31:0] \_pc\_M      ,
36. output wire [31:0] \_alu\_out   ,
37. output wire [4:0]  \_reg\_wa    ,
38. output wire [31:0] \_imm20\_e32 ,
39. output wire [1:0]  \_c3        ,
40. output wire        \_dmem\_we   ,
41. output wire        \_reg\_we    ,
42. output wire [31:0] \_reg\_rd2
43. );
44. reg [31:0] pc\_M\_reg     ;
45. reg [31:0] alu\_out\_reg  ;
46. reg [4:0]  reg\_wa\_reg   ;
47. reg [31:0] imm20\_e32\_reg;
48. reg [1:0]  c3\_reg       ;
49. reg        dmem\_we\_reg  ;
50. reg        reg\_we\_reg   ;
51. reg [31:0] reg\_rd2\_reg  ;
53. always @(posedge clk or negedge rst) begin
54. if(!rst) pc\_M\_reg <= 32'h00000000;
55. else begin
56. pc\_M\_reg      <= pc\_M     ;
57. alu\_out\_reg   <= alu\_out  ;
58. reg\_wa\_reg    <= reg\_wa   ;
59. imm20\_e32\_reg <= imm20\_e32;
60. c3\_reg        <= c3       ;
61. dmem\_we\_reg   <= dmem\_we  ;
62. reg\_we\_reg    <= reg\_we   ;
63. reg\_rd2\_reg   <= reg\_rd2  ;
64. end
65. end
66. assign \_pc\_M      = pc\_M\_reg     ;
67. assign \_alu\_out   = alu\_out\_reg  ;
68. assign \_reg\_wa    = reg\_wa\_reg   ;
69. assign \_imm20\_e32 = imm20\_e32\_reg;
70. assign \_c3        = c3\_reg       ;
71. assign \_dmem\_we   = dmem\_we\_reg  ;
72. assign \_reg\_we    = reg\_we\_reg   ;
73. assign \_reg\_rd2   = reg\_rd2\_reg  ;
74. assign \_reg\_rd2   = reg\_rd2\_reg  ;
76. endmodule

***pipeline\_reg\_W.v:***

1. `timescale 1ns / 1ps
2. //////////////////////////////////////////////////////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 2023/09/03 15:22:02
7. // Design Name:
8. // Module Name: pipeline\_reg\_W
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////////////////////////////////////////////////
21. module pipeline\_reg\_W(
22. input rst                   ,
23. input clk                   ,

26. input wire [31:0] pc\_W      ,
27. input wire [31:0] alu\_out   ,
28. input wire [4:0]  reg\_wa    ,
29. input wire [31:0] imm20\_e32 ,
30. input wire [1:0]  c3        ,
31. input wire        reg\_we    ,
32. input wire [31:0] dmem\_rd   ,
34. output wire [31:0] \_pc\_W      ,
35. output wire [31:0] \_alu\_out   ,
36. output wire [4:0]  \_reg\_wa    ,
37. output wire [31:0] \_imm20\_e32 ,
38. output wire [1:0]  \_c3        ,
39. output wire        \_reg\_we    ,
40. output wire [31:0] \_dmem\_rd
41. );
42. reg [31:0] pc\_W\_reg     ;
43. reg [31:0] alu\_out\_reg  ;
44. reg [4:0]  reg\_wa\_reg   ;
45. reg [31:0] imm20\_e32\_reg;
46. reg [1:0]  c3\_reg       ;
47. reg        reg\_we\_reg   ;
48. reg [31:0] dmem\_rd\_reg  ;
50. always @(posedge clk or negedge rst) begin
51. if(!rst) pc\_W\_reg <= 32'h00000000;
52. else begin
53. pc\_W\_reg      <= pc\_W     ;
54. alu\_out\_reg   <= alu\_out  ;
55. reg\_wa\_reg    <= reg\_wa   ;
56. imm20\_e32\_reg <= imm20\_e32;
57. c3\_reg        <= c3       ;
58. reg\_we\_reg    <= reg\_we   ;
59. dmem\_rd\_reg   <= dmem\_rd  ;
60. end
61. end
62. assign \_pc\_W      = pc\_W\_reg     ;
63. assign \_alu\_out   = alu\_out\_reg  ;
64. assign \_reg\_wa    = reg\_wa\_reg   ;
65. assign \_imm20\_e32 = imm20\_e32\_reg;
66. assign \_c3        = c3\_reg       ;
67. assign \_reg\_we    = reg\_we\_reg   ;
68. assign \_dmem\_rd   = dmem\_rd\_reg  ;
70. endmodule

riscv\_top.v作为顶层设计，负责信号的传递以及各个逻辑组件的整合，其架构图如下：



# 测试

功能性测试主要分为两部分：1. 指令的实现情况 2. 分支预测的实现情况

解释指令与ID便于观察波形图：

`define ID\_ADD 1

`define ID\_SUB 2

`define ID\_ORI 3

`define ID\_LUI 4

`define ID\_SW 5

`define ID\_LW 6

`define ID\_BEQ 7

`define ID\_JAL 8

// I类

`define ID\_XORI 9

`define ID\_ANDI 10

// R类

`define ID\_XOR 11

`define ID\_OR 12

`define ID\_AND 13

// B类

`define ID\_BNE 14

`define ID\_BGE 15

`define ID\_BLT 16

对于指令部分的功能实现情况，汇编代码如下：

RISCV：

.data

.text

ori t1,zero,0x2

lui t2,0x1

nop

nop

nop

add t3,t1,t2

sub t3,t2,t1

sw t1,(zero)

lw t4,(zero)

beq zero,zero,label1

nop

label1:

nop

jal a0,label2

nop

label2:

xori t5,zero,0x2

andi t6,zero,0x2

xor a1,t1,t2

or a2,t1,t2

and a3,t1,t2

bne t1,t2,label3

nop

label3:

nop

bge t1,t2,label4

nop

label4:

nop

bne t1,t2,label5

nop

label5:

nop

ori a1,zero,0x1

ori a1,zero,0x1

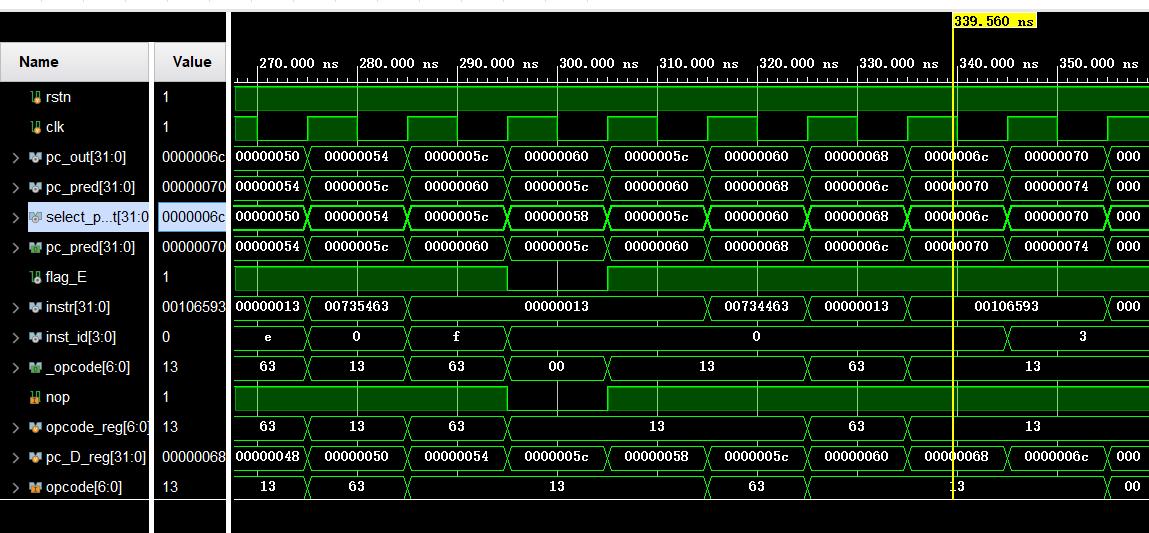
波形图：

日程表

描述已自动生成

图形用户界面

描述已自动生成



分析每条指令影响：

首先利用ori和lui初始化两个寄存器的值，以便测试后续指令。从控制流上看，指令的执行顺序没错，主要关注几个跳转指令，分析跳转结果也可获知其他指令的实现情况，因为跳转条件由其他指令给出。

指令7，即BEQ指令，此时t1=2,t2=1000,如理论所得跳转失败，分支预测恢复，被NOP填充；指令8，JAL直接跳过了后面紧跟的NOP，实现成功；指令e表示不相等则跳转，可以看到只有一条NOP说明跳转成功；指令BGE代表t1>=t2跳转，可以看到后面跟了多个NOP说明跳转失败。最后的ORI代表控制流最后结尾没有问题



可以看到三个周期后数据被存入IP核data\_ram，符合取指阶段和仿存阶段相差三个周期的理论，并且存取数字没有问题（1的位置为1代表二进制10，数值为2和t1相等）。SW LW指令实现无误

其次是分支预测的实现情况，主要测试分支预测失败恢复，编写代码使得条件跳转连续失败：

首先解释用到的指令以及ID：

3：ORI : 或立即数

4：LUI：将立即数存入寄存器高位

5：SW：将寄存器内容存入内存

7：BEQ：若条件相等则跳转

8：JAL：无条件跳转

RISCV测试代码:

.data

.text

ori t4, zero, 0x1

lui a0, 0x40

ori t5, zero, 0x4

sw t6, (a0)

nop

nop

label3:

beq t4, t5, label2

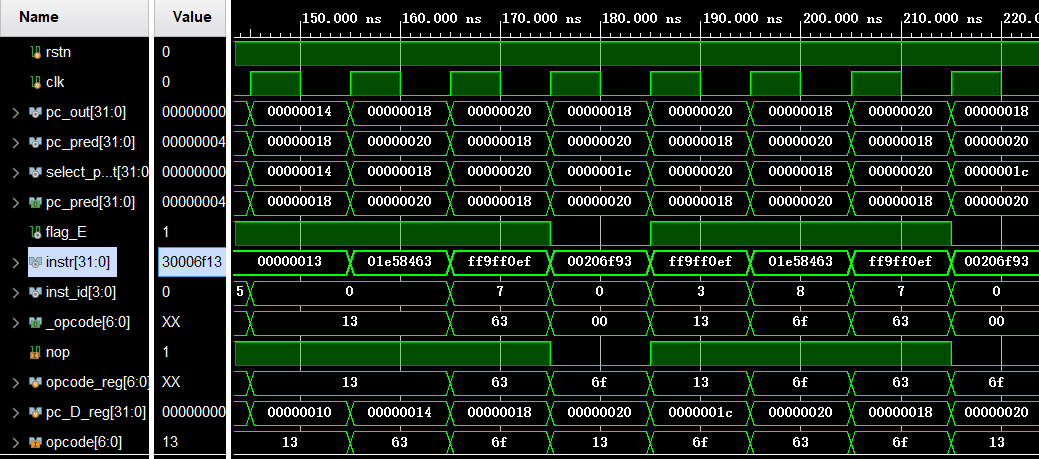
ori t6, zero, 0x2

label2:

jal label3

简要说明：在代码中对t4和t5寄存器进行了赋值以待beq条件跳转比较，插入nop为了防止RW数据冒险，多出的sw指令可以充当一条缓冲指令。在设置上，为了测试分支预测恢复，故意设置所有的条件跳转失败，观察波形图。

波形图：



图形用户界面

描述已自动生成

观察波形图，可以看到连续预测失败的指令顺序为7-0-3-8，也就意味着有条件跳转失败结果已知的情况下，被错误取出的指令会在译码阶段被NOP（0号指令）覆盖，同时取指阶段取出正确位置的ORI指令，此时NOP信号行为同样正确，这与通路设计相符。这里经过了DEBUG，具体看问题解决部分4。

对于性能部分，时钟周期为10ns, CPU主频为100MHz，但理论上可以根据周期最长的流水线周期来确定时钟周期

对于上板部分，流水灯能够正常闪烁，VGA能够正常操作，蜂鸣器能够正常响音。详见视频

# 问题及解决方法

1.



2.初次行为仿真所有信号均为非确定(X)

解决：经过分析，虽然flag\_E为执行阶段信号，但由于flag\_E\_reg未初始化，且其值又影响selectPC模块对于第一个PC的选择，导致PC初值错误

3.部分变量和信号的仿真波形出现高阻态

解决：经过信号通路的溯源，发现在riscv\_top中将变量名写错了，是由不规范的变量名书写造成的错误。

4.仿真波形中，在分支预测失败时会连续地插入两条NOP指令，使得一条应该正确执行的指令被NOP覆盖，NOP冗余

解决：

以下代码均位于模块”PIPELINE\_REG\_D”

原always块触发条件：

always @(posedge clk or negedge rst or negedge nop) begin

错误直觉：由于只有在NOP为1->0的时候才会触发预测恢复，故只需关注下降沿

改正后always块触发条件：

always @(posedge clk or negedge rst or negedge nop or posedge nop) begin

这是因为always在每个条件成立时都会触发一遍，故在下一个时钟跳变时，会出现clk先触发这个模块，还是nop先改变的问题；如果clk先出现，那么此时的nop还是0，那么意味着此时还会执行NOP，而不是进行正常的流水线流水；但是事实上nop是由flag\_E控制的，而flag\_E是由clk控制的，所以此时比如会出现上述的错误情况，但是如果再使用一次posedge就会解决这个问题，因为在同一时钟周期内即便第一次触发Always时nop还没变为1，那么当nop变为1时还会再触发一次always，使得结果正确。

# 心得体会及总结

自己设计一个cpu，需要以下的知识储备：

深入学习计算机体系结构：了解计算机体系结构的基本原理和概念是非常重要的。熟悉了指令集架构、寄存器文件、流水线处理的概念和设计方案。

精通硬件描述语言：Verilog硬件描述语言是实现CPU的关键工具。必须较为深入的理解verilog的基本语法和更为透彻的理解周期信号的含义。

在实现cpu的过程中，发现硬件类的编码会有很多重复的地方，可以使用vivado的ctrlc+ctrlv的便捷操作来粘贴。debug的时候最好逆着数据通路来寻找问题，这样会轻松很多。

# 参考文献有价值的资源推荐

《深入理解计算机系统》, Randal E. Bryant and David R. O'Hallaron, 2015

《计算机组成原理课程设计PPT----单周期RISCV编指与数据通路设计》