华中科技大学 计算机学院

课程设计任务书

本课程力图以“培养学生现代计算机系统设计能力”为目标，贯彻“强调软/硬件关联与协同、以 CPU 设计为核心/层次化系统设计的组织思路，有效地增强对学生的计算机系统设计与实现能力的培养”。硬件综合训练课程是完成该计算机组成原理课程并进行了多个单元实验后，综合利用所学的理论知识，并结合在单元实验中所积累的计算机部件设计和调试方法，设计出一台具有一定规模指令系统的简单计算机系统。所设计的系统能在 LOGISIM 仿真平台和 FPGA 实验平台上正确运行，通过检查程序结果的正确性来判断所设计计算机系统正确性。

课程设计属于设计型实验，不仅锻炼学生简单计算机系统的设计能力，而且通过进行中央处理器底层电路的实现、故障分析与定位、系统调试等环节的锻炼，进一步提高学生分析和解决问题的能力。

1．课程设计题目

RISC-V 5 段流水线 CPU 的设计

2、简单计算机系统的设计目标

本课程设计的总体目标是利用 FPGA 以及相关外围器件，设计五段流水 CPU，要求所设计的流水 CPU 系统能支持自动和单步运行方式，能正确地执行存放在主存中的程序的功能，对主要的数据流和控制流通过 LED、数码管等适时的进行显示，方便监控和调试。尽可能利用 EDA 软件或仿真软件对模型机系统中各部件进行仿真分析和功能验证。在学有余力的前提下，可进一步扩展相关功能。

3、主要技术指标

1) 支持规定的 32 位 RISC-V 指令集（指令集任选），具体见表 1；
2) 在 CCAB 扩展指令集中支持 2 条 C 类运算指令，1 条 M 类存储指令，1 条 B 类分支指令，具体任务每位同学不一样，指令编号详见见公文包中的任务分配；
3) 支持多级嵌套中断，利用中断触发扩展指令集测试程序；
4) 支持 5 段流水机制，可处理数据冒险、结构冒险、分支冒险；
5) 能运行由自己所设计的指令系统构成的一段测试程序，测试程序应能涵盖

所有指令，程序执行功能正确。

6) 能运行教师提供的标准测试程序，并**自动统计执行周期数**

7) 能自动统计各类无条件分支指令数目，条件分支成功次数、插入气泡数目、load-use 冲突次数、动态分支预测流水线能自动统计预测成功与失败次数。

表 1 基本指令集

#	RISC-V	指令类型	简单功能描述	备注
1	ADD	R	加法	指令格式与功能 请参考 RISC-V32 指令集英文手册， 或参考 RARS 模拟器
2	ADDI	I	立即数加	
3	AND	R	与	
4	ANDI	I	立即数与	
5	SLLI	I	逻辑左移	
6	SRAI	I	算数右移	
7	SRLI	I	逻辑右移	
8	SUB	R	减	
9	OR	R	或	
10	ORI	I	立即数或	
11	XORI	I	或非/立即数异或	
12	LW	I	加载字	
13	SW	S	存字	
14	BEQ	B	相等跳转	
15	BNE	B	不相等跳转	
16	SLT	R	小于置数	
17	SLTI	I	小于立即数置数	
18	SLTU	R	小于无符号数置数	
19	JAL	J	转移并链接	
20	JALR	I	转移到指定寄存器	
21	ECALL	I	系统调用	if ($a7==34) LED 输出$a0 的值 else 停机等待 Go 按键按下 注意显示逻辑需要考虑如何锁存过

				去的数据，否则数据一闪而过。
22	CSRRSI	I	访问 CSR 寄存器	中断相关，可简化为开中断
23	CSRRCI	I	访问 CSR 寄存器	中断相关，可简化为关中断
24	URET	I	中断返回	清中断，mEPC 送 PC，开中断

表 2 CCAB 扩展指令集

指令分类	编号	RISC-V	指令类型	备注
运算 (C)	1	SLL	R	指令格式参考 RISC-V32 指令集，最终功能以 RARS 模拟器为准。
	2	SRL	R	
	3	SRA	R	
	4	XOR	R	
	5	AUIPC	U	
	6	LUI	U	
	7	SLTIU	I	
	8	MUL	R	
	9	DIVU	R	
	A	REMU	R	
存储访问 (A)	1	LB	I	
	2	LBU	I	
	3	LH	I	
	4	LHU	I	
	5	SB	S	
	6	SH	S	
跳转 (B)	1	BLT	B	
	2	BGE	B	
	3	BLTU	B	
	4	BGEU	B	

4、系统设计要求

1) 根据课程设计指导书的要求，制定出设计方案；
2) 分析指令系统格式，指令系统功能。
3) 根据指令系统构建基本功能部件，主要数据通路。
4) 根据功能部件及数据通路连接，分析所需要的控制信号以及这些控制信号的有效形式；
5) 设计出实现指令功能的硬布线控制器；
6) 调试、数据分析、验收检查；
7) 课程设计报告和总结。

5．课程设计成绩的评定

1) 评定成绩根据考核、课程设计的过程、课程设计的效果、课程设计报告的质量等几部分组成；
2) 评分标准为设计过程占 70%，团队作业 10%，报告和图纸部分占 20%；
3) 课程设计的成绩评定等级为不及格、及格、中、良好、优秀五级，具体的评定标准见评分规则；
4) 对基本功能进行扩展或设计具有非常鲜明的特征和一定程度的创新，可根据实际情况加分。

6、对课程设计报告的要求

1) 课程设计报告是体现和总结课程设计成果的载体，主要内容包括：设计题目、设计目的、设备器材、设计原理及内容、设计步骤、遇到的问题及解决方法、设计总结、参考文献等。
2) 在适当位置配合相应的实验原理图、数据通路图等图表进行说明。应做到文理通顺，内容正确完整，书写工整，装订整齐。
3) 设计总结部分主要写本人工作简介以及设计体会，包括通过课程设计学到了什么，哪里遇到了困难，解决的办法以及今后的目标。
4) 为统一格式和要求，课程设计报告要求采用《硬件综合训练》课程设计报告模板，采用 A4 纸双面打印（教师不要求提交纸质版除外）。为减轻报告撰写工作量，关注报告书写质量，报告总页码不允许超过 40 页。

7．课程设计时间安排

课程设计的总体时间为 2 周，总体安排如下:

1) 第 1 天：到实验室布置任务和集中讲解，领取开发设备。
2) 第 1~3 天：学生查阅资料，开始方案设计。
3) 第 4 天：中期进度检查，单周期 CPU 验收检查，文档检查。
4) 第 6~10 天：阶段性成果随时检查。
5) 第 10 天：最终结果验收。
6) 10~30 天：团队任务。

验收采用分步检查验收方法，以适合各种层次的学生，不断提高学生动手能力。验收辅以答辩的方式，**抄袭被抄袭均按零分处理**。

8. 项目检查

所有阶段性成果能在头歌上完成检查的一定要在头歌上留下记录，完成你认为的最终版本后找指导教师进行最后检查，检查后要在纸质检查表上留下记录。

9. 报告提交

【腾讯文档】https://docs.qq.com/form/page/DYmVrV1ZtZUFSR2xR

参考文献:

[1] DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第 5 版).北京：机械工业出版社.

[2] David Money Harris(美). 数字设计和计算机体系结构（第二版）. 机械工业出版社

[3] 谭志虎，秦磊华，吴非，等.计算机组成原理（微课版 第 2 版）. 北京：人民邮电出版社，2021 年.

[4] 谭志虎，周健，周游.计算机组成原理实验指导（基于 RISC-V 在线实训）. 北京：人民邮电出版社，2024 年.

[5] 曹强，施展.计算机系统结构（微课版）. 北京：人民邮电出版社，2024 年.

课程设计成绩评分标准

课程设计评定成绩根据平时成绩、课程设计的过程、结果、课设报告的质量等几部分组成；设计过程和结果占 70%，团队作业 10，课程设计报告 20%；对基本功能进行扩展或设计具有非常鲜明的特征和一定程度的创新，可根据实际情况加分。

1. 课程设计过程与结果评分标准

- 完成单周期 CPU 设计与实现（LOGISIM 平台）：50 分
- 完成理想流水线的多周期 CPU（LOGISIM 平台）：55 分
- 完成气泡流水线（LOGISIM 平台）：65 分
- 完成重定向流水线（LOGISIM 平台）：75 分
- 实现单级中断（LOGISIM 平台）：+5 分
- 实现多级嵌套中断（LOGISIM 平台）：+5 分
- 完成流水中断机制（LOGISIM/FPGA 平台）：+5 分
- 完成动态预测方式的分支冒险处理（LOGISIM/FPGA 平台）：+10 分
- 实现单周期上开发板（FPGA 平台）：+5 分
- 实现流水线上开发板（FPGA 平台）：+5 分

2. 团队任务（团队互评）

团队开发一个具有展示度的演示系统，有输入输出设备，完整的软硬件系统

3. 课程设计报告评分标准

- 报告规范，清楚表述设计思想、设计思路、设计过程、设计结果，文档资料完整，书写和画图规范：20 分
- 满足课程设计报告格式要求，能较清楚的表述设计思想、设计思路、设计过程、设计结果，文档资料较完整，书写和画图规范性较好：16 分
- 满足课程设计报告格式要求，基本能表述设计思想、设计思路、设计过程、设计结果，文档资料完整一般，书写和画图规范性一般：14 分
- 报告不规范，内容不完整，不能体现设计原理、方法和自己的工作：10 分
- 严重不规范、内容空洞，完全不能体现课程设计的内容：5 分
- 课程设计报告抄袭 0 分

课程设计指南

在本次课程设计，需要使用 LOGISIM 来创建一个 32-位 5 段流水 CPU，该 CPU 运行的是标准 RISC-V 指令集的子集，关于指令功能请仔细查阅 RISC-V 指令手册，常见指令格式如**图 1** 所示：

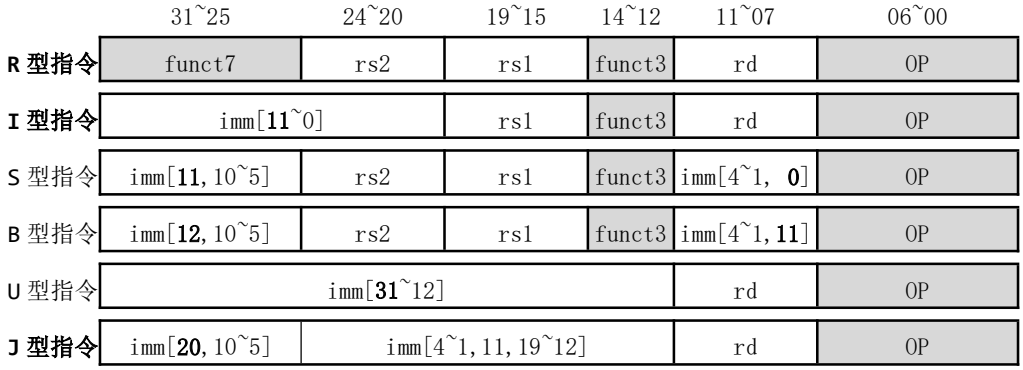


图 1 RISC-V 指令格式

图中 7 位的主操作码 OP 均固定在低位，扩展操作码 funct3，funct7 字段位置也是固定的，另外源寄存器 rs1，rs2 以及目的寄存器 rd 位置在指令字中的位置也是固定不变的。以上字段位置固定后，剩余的位置用于填充立即数字段 imm，这也直接导致 imm 字段看起来比较混乱，不同类型指令立即数字段的长度，甚至顺序都不一致。但 imm 字段的最高位都固定在指令字的最高位，方便立即数的符号扩展，另外立即数字段中部分字段尽量追求位置固定，如 I/S/B/J 型指令的 imm[10~5]字段位置固定，另外 S/B 型指令中的 imm[4~1]字段位置固定。不同指令格式中的立即数字段最终会译码生成 32 位立即数，具体立即数形式如图 2，注意图中 inst[m:n]表示指令字中第 m~n 位。

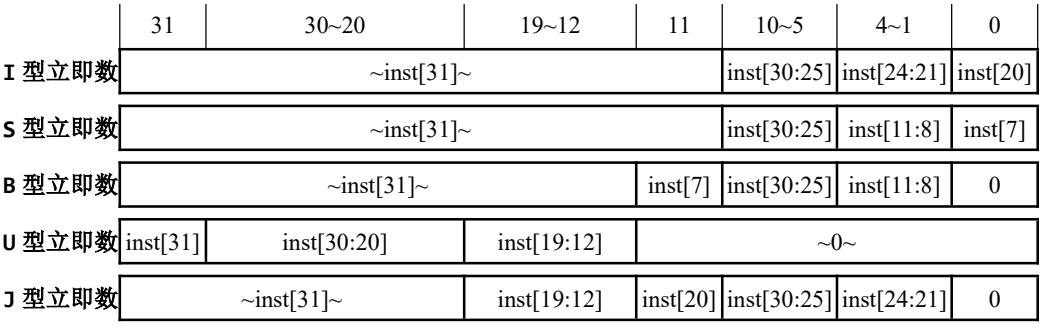


图 2 RISC-V 立即数形式

一、 单周期 CPU 设计流程

单周期处理器是指所有指令均在一个时钟周期内完成的处理器。尽管不同指令执行时间不同，但对单周期处理器而言，时钟周期必须设计成对所有指令都等长。需要特别说明的是，在单周期处理器中，一条指令执行过程中数据通路的任何资源都不能被重复使用，因此任何需要被多次使用的资源(如加法器)都需要设置多个，否则就会发生资源冲突，这是设计单周期处理器数据通路时必需考虑的重要环节，

另外取指令和执行指令阶段均需要使用存储器，所以单周期处理器只能采用指令存储器和数据存储器相分离的结构（哈佛结构）。在这里我们采用简单迭代法实现单周期处理器。

这种方法相对比较直观简单，设计者先完成支持一条固定 R 型指令（如加法指令）的基本数据通路，测试运行通过后再在这个基础上不断增加新的数据通路，支持新的指令，直至所有指令都能正常运行。要支持一条指令的运行必须有取指令逻辑和执行指令的逻辑，所有设计者首先应该完成取指令的数据通路。

1） 设计取指令数据通路

图 3 为取指令的数据通路，涉及到的功能部件包括程序计数器 PC、指令存储器、加法器等。取指令通路应能取出程序计数器 PC 锁存地址对应的的机器指令，并能在时钟上跳沿到来时实现 PC 自动加 4，从而进入下一条指令的指令周期。这里程序计数器 PC 可以采用寄存器实现，其输出作为指令存储器的地址输入，指令存储器经过一个存储周期后一次性取出 32 位的机器指令，故其数据宽度应为 32。需要注意的是，无论是在 Logisim 平台实现还是 FPGA 上实现，设计者均需考虑指令存储器地址位宽的问题，首先在 FPGA 实现中指令存储器地址宽度越大综合速度越慢，另外 PC 锁存地址是 32 位字节地址，指令存储器输入地址是字地址，二者不匹配；为简化电路设计，Logisim 中建议采用 ROM 实现。

在顺序执行方式下，每一个时钟周期内 CPU 取指令后将 PC 寄存器的值加 4，形成下一条指令的地址。这里加“4”是因为 RISC-V32 中指令字长均为 4 字节，每条指令在存储器中占用 4 个字节的存储单元，而 PC 中存放的地址是字节地址。为避免资源冲突，这里加法器应该是独立的器件，和 CPU 中的算术逻辑运算单元分开。

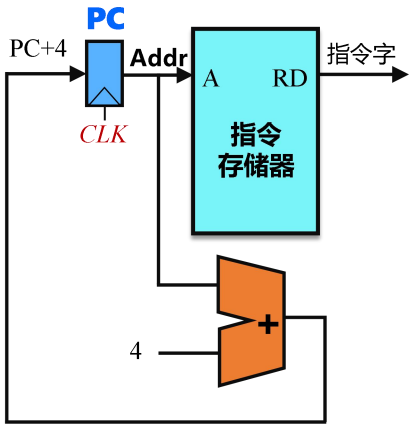


图 3 取指令数据通路

指令取出后即可进入指令的执行周期，接下来我们可以开始设计能支持最为简单的 R 型指令的数据通路，使得能运行一条指令的简单的 CPU 能运行起来。

2） 构建 R 型指令数据通路

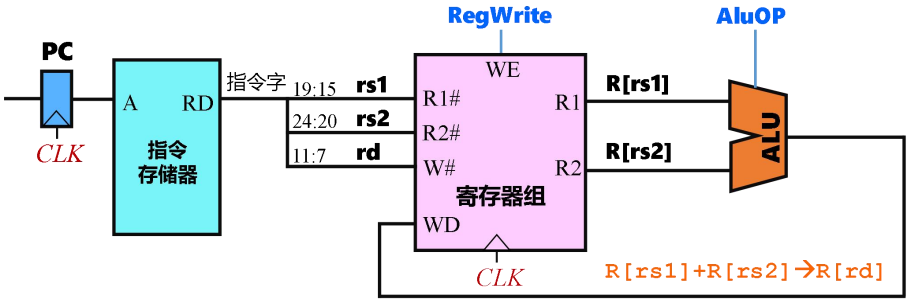


图 4 R 型指令的数据通路

RISC-V32 中算术逻辑运算指令属于 R 型指令，R 指令所有的操作数均是寄存器，执行过程中涉及的功能部件包括寄存器文件和 ALU。R 型指令的数据通路如**图 4** 所示，指令执行过程中的 ALU 的两个源操作数均来自于寄存器文件输出，其中 R[rs1]是寄存器 R1#的值，R[rs2]是寄存器 R2#的值，R1#来自于指令字中的 rs1 字段，R2#来自于指令字中的 rs2 字段，运算结果写入目的寄存器 rd 中，将 AluOp 设置为不同的值，就可以进行不同的运算。

3） 构建 I 型运算指令数据通路

I 型运算类指令另外一个运算操作数来自 12 位立即数，12 位立即数经 32 位符号扩展器扩展后送入 ALU，二者进行运算得到最终的结果，运算结果写入目的寄存器 rd 中，同样将 AluOp 设置为不同的值，就可以进行不同的运算，数据通路如**图 5** 所示。

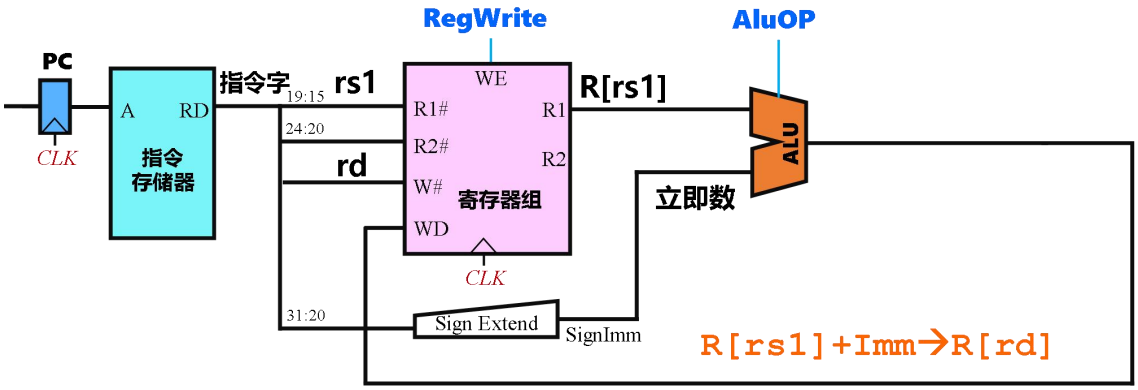


图 5 I 型运算指令的数据通路

4） 构建访存指令数据通路

访存指令属于 I 型指令，访存地址等于变址寄存器 rs1 的值加上 12 位立即数，地址运算通过 ALU 完成，所以需要将 rs1 的值送入 ALU，同时要将 12 位立即数进行 32 位符号扩展后送入 ALU，二者进行加法运算得到最终的访存地址，**图 6** 是访存指令操作的部分数据通路。该数据通路包括指令存储器、寄存器文件、符号扩展、ALU、数据存储器等功能部件。如果是加载指令，则数据存储器的结果将会送回到寄存器文件的 WD 端，写入寄存器编号为 rd。

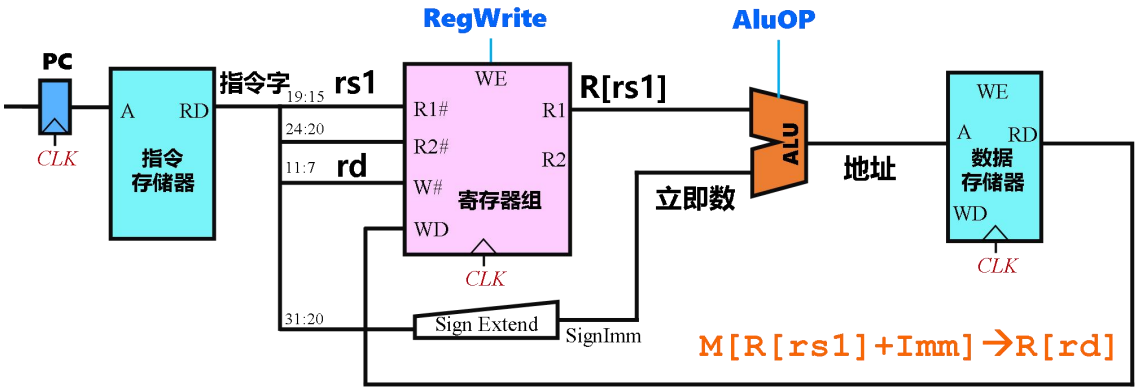


图 6 lw 指令的数据通路

如果是存储指令，则 rs2 寄存器的值会通过 R2 端口输出到数据存储器的写入端口 WD，如**图 7** 所示。

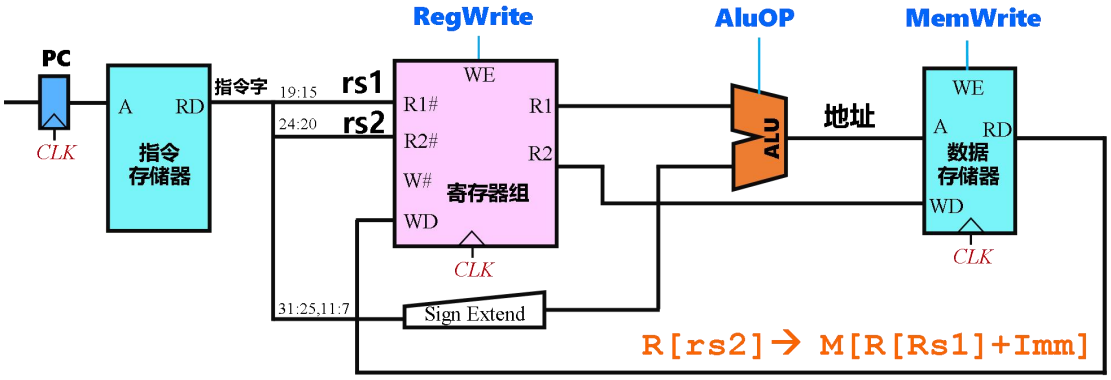


图 7 sw 指令的数据通路

对比不同类型指令的数据通路可发现，寄存器组写入寄存器编号 WD 的输入，ALU 第二个操作数输入，符号扩展器 SignExtend 的输入都包含多个不同的输入来源，为了将不同的数据通路统一到同一个电路中，我们需要在有多个输入来源的端口增加多路选择器，如**图 8** 所示，每增加一个多路选择器就额外引入了一个控制信号，这里分别增加了 S_Type(为 1 表示 S 指令)、AluSrc（为 1 选择立即数送 Src）、MemtoReg（为 1 将内存数据写回寄存器）3 个控制信号。这些控制信号都应该由硬布线控制器根据指令译码自动生成，在完成对应控制逻辑之前，我们可以先手动设置对应的值，验证数据通路的功能是否正常。当数据通路能正常工作时，就可以在控制器中增加相应的组合逻辑自动生成新增加这些控制信号。至此我们设计的 CPU 已经可以支持部分 R 型、I 型、S 型指令。

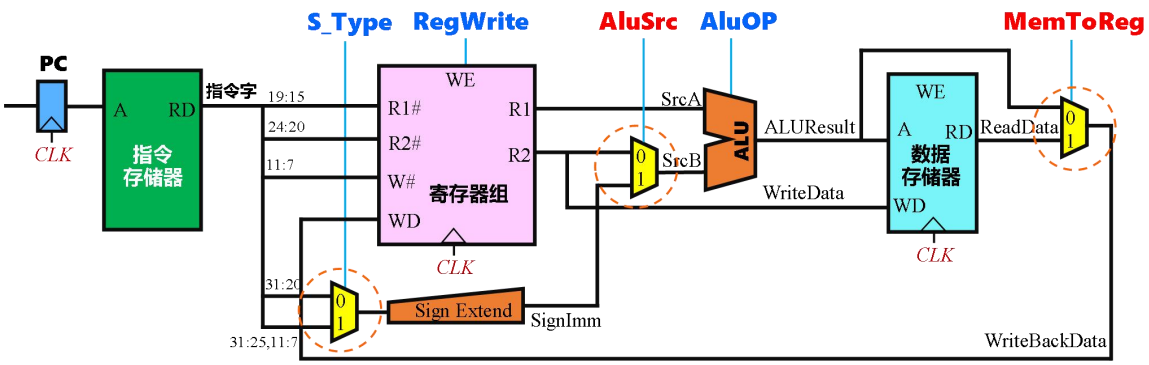


图 8 支持算术逻辑运算指令和访存指令的混合数据通路

继续增加其它的数据通路，功能模块，多路选择器，控制信号，修改硬布线控制器逻辑可以逐步扩展 CPU 的功能，使得最终的 CPU 能支持实验要求的所有指令集，将所有类型的指令的数据通路综合后，再增加必要的控制逻辑，就可以得到支持 MIPS 机器指令系统子集的完整数据通路，**图 9** 是基于单周期方案的 MIPS 处理器数据通路顶层视图，该视图中的数据通路在 PC 寄存器输入增加了一个多路选择器，可以进一步支持 B 型指令中的 beq 指令，读者可以在此基础上进一步扩充完善。

为简化顶层视图逻辑结构，图中引入了 ImmGen 立即数生成器，该部件根据控制器生成的指令类型信号 InstrType(实验中可能包含 S_Type、JAL、JALR 等译码信号）生成正确的立即数，在具体实验时可利用相关功能部件以及多路选择器实现。

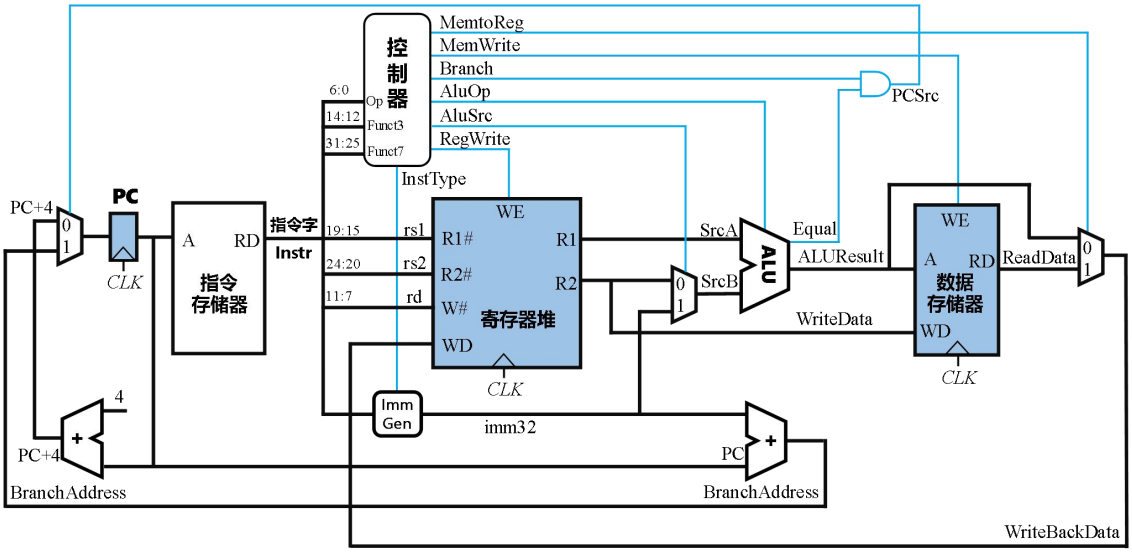


图 9 单周期方案数据通路顶层视图

5） 在 Logisim 中实现单周期数据通路

打开 cpu24-riscv.circ 文件的单周期 RISC-V 子电路，电路框架如**图 10** 所示。利用电路框架已经给出了程序计数器 PC、指令存储器、寄存器文件、运算器 ALU、数据存储器、单周期硬布线控制器（待实现）等电路，参考**图 9** 构建能支持 24 条指令的完整数据通路。

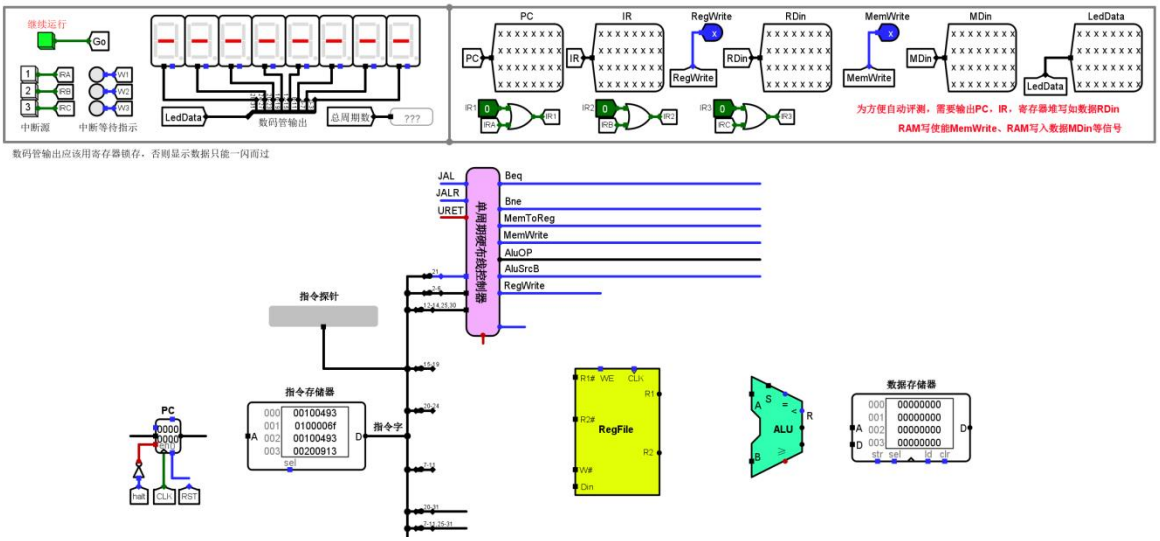
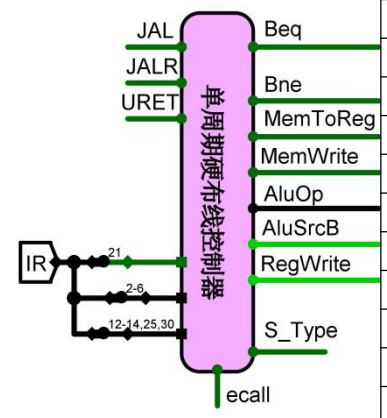


图 10 单周期 RISC-V 电路框架

6） 实现硬布线控制器逻辑

图 10 中单周期硬布线控制器电路封装与引脚功能如**表 1** 所示，对于单周期处理器而言，硬布线控制器是纯组合逻辑，控制器的功能就是根据指令字中操作码 OP 和 Funct 字段的值直接输出所有操作控制信号。

表 1 **硬布线控制器（24 条指令）电路封装**



引脚	类型	位宽	功能说明
OP	输入	5	指令字 Opcode 字段中的 5 位，IR[6:2]
Funct	输入	8	指令字 Funct3/7 字段，IR[30,25,14:12]
IR[21]	输入	1	指令字 21 位，用于区分 ecall 与 uret
JAL	输出	1	JAL 指令译码信号
JALR	输出	1	JAL 指令译码信号
Beq	输出	1	Beq 指令译码信号
Bne	输出	1	Bne 指令译码信号
ecall	输出	1	ecall 指令译码信号
S_Type	输出	1	S 型指令译码信号
MemToReg	输出	1	写回数据选择，为 1 写回存储器数据
MemWrite	输出	1	存储器写使能，高电平有效
AluOp	输出	4	ALU 运算选择
AluSrcB	输出	1	ALU 的第二输入来源选择控制
RegWrite	输出	1	寄存器堆写使能，高电平有效

硬布线控制器电路框架如**图 11** 所示。由于 Logisim 自动生成电路功能输出引脚数目有限，为了方便自动生成电路，这里将 AluOP 输出利用运算器控制器单独输出，其他控制信号则用控制信号生成电路输出，实验需要分别设计实现这两个电路。

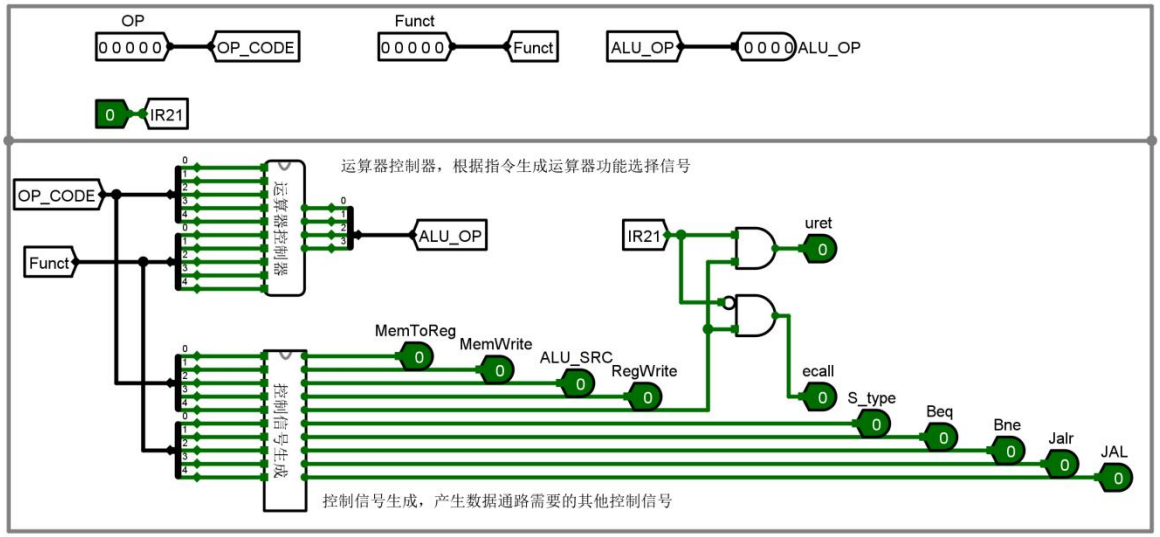


图 11 单周期硬布线控制器（24 条指令）电路框架

利用 EXCEL 打开实验资料包中的“单周期硬布线控制器表达式自动生成.xlsx”文件的真值表，如图 12 所示。查看 RISC-V 指令码表（HUST）.xlsx，逐条填写每条指令 OpCode、Funct7，Funct3 字段以及对应的 AluOp 和其他控制信号值（只填 1 值）。填写完毕后，打开控制信号表达式生成表，可以得到所有控制信号的逻辑表达式，利用逻辑表达式自动生成运算器控制器和控制信号生成电路即可。

#	指令	Funct7 (十进制)	Funct3 (十进制)	OpCode (十六进制)	ALU_OP	MemtoReg	MemWrite	ALU_Src	RegWrite	ecall	S_Type	BEQ	BNE	Jal	jalr
1	add														
2	sub														
3	and														
4	or														
5	slt														
6	sltu														
7	addi														
8	andi														
9	ori														
10	xori														
11	slti														
12	slli														
13	srli														
14	srai														
15	lw														
16	sw														
17	ecall														
18	beq														
19	bne														
20	jal														
21	jalr														
22	CSRRSI														
23	CSRRCI														
24	URET														

真值表 | 控制信号表达式生成 | 运算器规格 | 控制信号产生条件

图 12 硬布线控制器（24 条指令）真值表

7） 系统调试

完成控制器电路实现后，在单周期 RISC-V 子电路中的指令存储器中载入 risc-v-benchmark.hex 程序，利用 Ctrl+k 或⌘+k 快捷键开启时钟自动仿真，测试程序，程序最后一条指令为停机指令，执行完毕后系统总周期数计数应该是 1546，LED 显示区应该显示 00000038，数据存储器中数据分布情况如图 13 所示。

图 13 测试程序运行结果

如程序测试有问题，可以单步调试每条指令，也可利用头歌平台自动评测，根据出错的测试用例信息精准定位出错时钟节拍，然后利用电路框架中的可计数暂停的时钟源运行到到出错时钟节拍前一条指令，单步跟踪，复原错误现场，进一步根据指令功能排查故障即可。

8） 差异化指令支持

在 24 条指令的基础上随机增加扩展指令，应包括 2 条 C 类指令、1 条 A 类指令、1 条 B 类指令，简称 CCAB 系列指令。最终实现的处理器能正确运行标准测试程序 benchmark.asm+4 条指令的测试程序。读者需要将 4 条指令的标准测试程序添加在 benchmark.asm 程序结尾，benchmark 程序运行结束时需要等待 Go 按钮按下，才能继续运行后续测试程序。

9） 注意事项

1. RISC-V 虚存模式设置

由于实验中设计的单周期处理器并不包括内存管理单元 MMU 部件，所以程序运行时的访存地址均是物理地址，设计时采用指令存储器和数据存储器分离的哈佛架构。访问数据时应该访问数据存储器，数据存储器的地址起始地址是 0，实验包中的测试程序均直接使用了 0 号地址开始的数据单元，为了使测试程序在 RARS 和实验设计的处理器中的运行结果保持一致，必须配置 RARS 模拟器中的虚存模式，具体可以选择菜单 Setting→Memory Configration 选项，如**图 14** 所示。这里应将虚存模式设置为 Compact 模式，这样数据段起始位置 0 开始的位置注意，如果采用 Default 模式在 RARS 中运行 sort.asm 排序程序时访存指令会越界访问代码段或内核段，引起保护错。

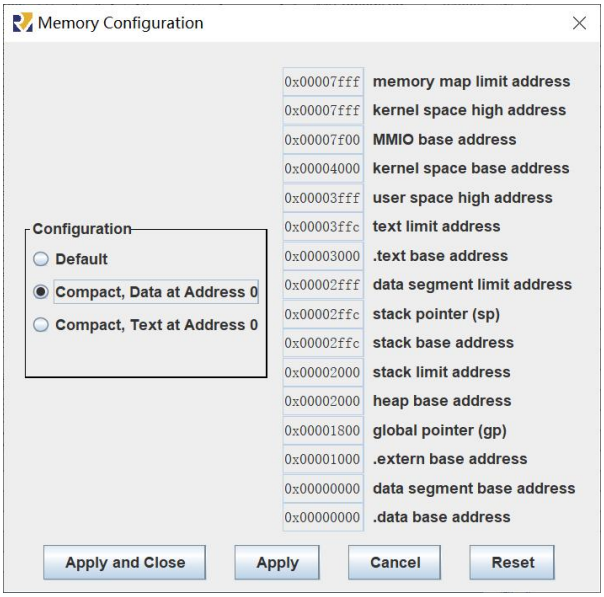


图 14 虚存模式配置

2. RISC-V 机器指令导出

在 RARS 中可以利用菜单 File→Dump Memory 功能将汇编程序的代码段的机器指令和数据段的数据导出，为了能直接在 Logisim 的 RAM 和 ROM 组件中使用，应采用十六进制文本的方式导出，到处成功后还应在文本文件第一行加入“v2.0 raw”，这样导出的文件即可直接加载到 Logisim 平台的 ROM 和 RAM 组件中，如**图 15** 所示。

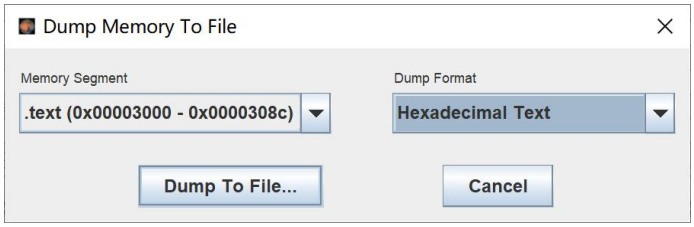


图 15 RISC-V 代码导出

3. 寄存器文件库

实验资料包中的 Logisim 库文件 CS3410.jar 提供了一个标准寄存器文件的库，该库由美国康奈尔大学开发，在 Logisim 平台中可以通过加载 JAR 库的方式加载第三方 JAVA 库，如**图 16** 所示。

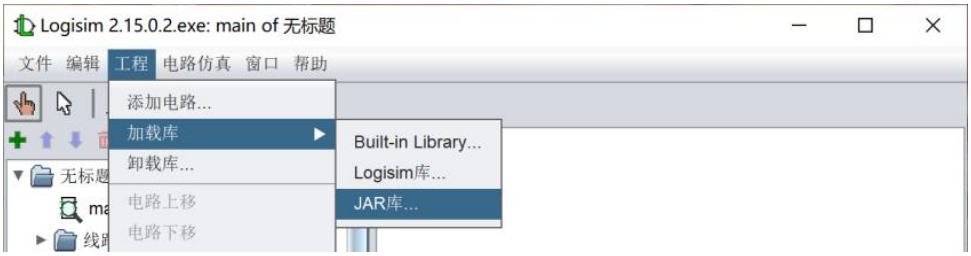


图 16 JAR 库加载

选择实验包中的 CS3410.jar 库，加载成功后 Logisim 左下角的组件库会增加 CS3410 组件的选项，其中第一个组件就是寄存器文件，添加该组件到画布中，如图 17 所示，这个寄存器文件的引脚 rA，rB 为读寄存器编号，rW 为写入寄存器编号，

WE 为写使能，W 为写入数据，A、B 为 rA，rB 寄存器的输出值，该组件直接提供了 32 个寄存器观察窗口，非常直观，用户还可以使用手型的戳工具直接修改寄存器的值，这点对调试比较有用。

注意该寄存器文件组件缩小显示时组件上数字的显示可能不正常，另外如果写入数据位不确定值，则对应寄存器的值会显示为问号。

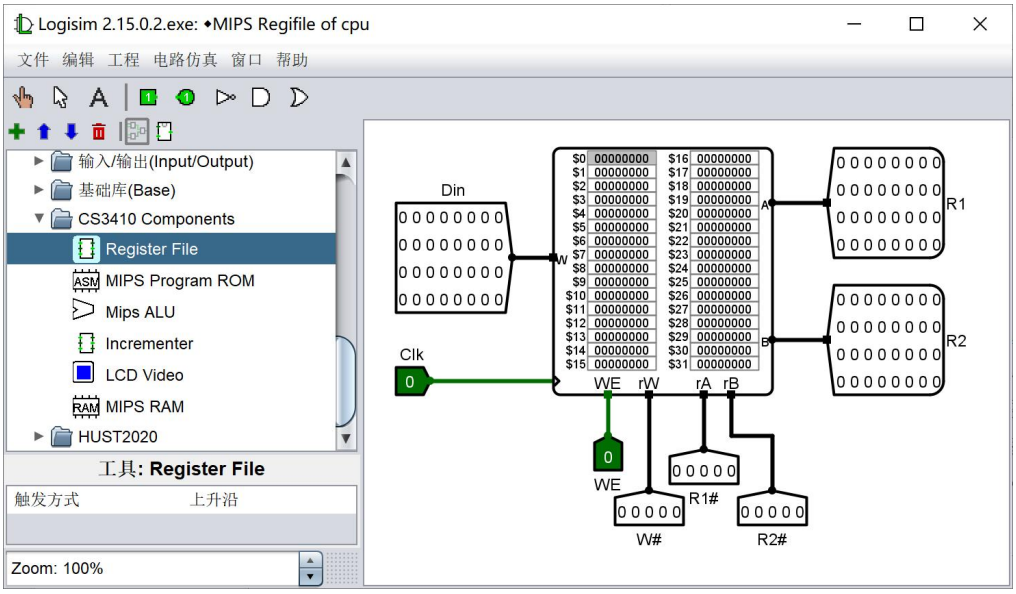


图 17 MIPS 寄存器文件库

4. RAM 库

存储系统实验中设计过 MIPS RAM 电路，CS3410 库中也提供了一个标准的 RAM 组件抽象，该组件也可用于 RISC-V 处理器，在实现 LB、SB、LH、SH 等字节访问或半字访问指令时非常有用。该组件的访问接口如图 18 所示，各引脚功能说明如下：

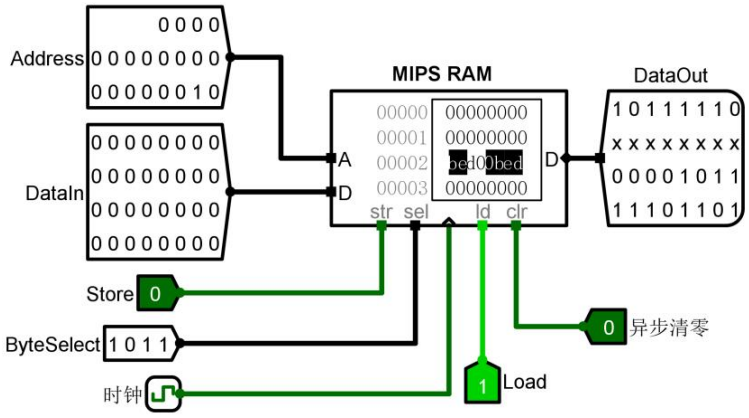


图 18 MIPS RAM 组件

- A 是 20 位字地址输入，MIPS RAM 组件中的每一行代表内存中的 4 个字节，图中 A 端地址为 0x00002，MIPS RAM 中第二行用黑底白字显示，0xbed00bed 为 2 号字单元的值，如果需要访问当前地址中具体某个字节或半字，需要设

置 MIPS RAM 组件底部的 Sel 输入端口。缺省地址为 20 位，可以访问 1MWord=4MB。

- 左侧 D 为数据输入端，位宽 32 位， 包含即将写入 RAM 的数据，具体写入时会根据 sel 端的设置决定数据输入端的哪些字节被写入。
- str 是存储控制位，当 str=1 时，RAM 会根据 sel 的值写入 D 中的某些字节。
- sel 是选择控制字段，线宽 4 位，当 sel=0001 时，D 端口的 0~7 位被选中，当 sel = 0010，D 端口的 8~15 位被选中，当 sel = 0100 时，D 端口 16~23 位被选中，当 Sel=1000 时，D 端口 24~31 位被选中。另外 SEL=0011，表示 0~15 位被选中，SEL=1100，表示 16~31 位被选中，SEL=1111 时，4 个字节同时被选中。
- ld 是载入位，ld 为 1 时组件右侧的数据输出端口 D 输出当前字中被选择的字节，如果 ld=0，输出为高阻态。
- clr 为 1 时会清空 RAM 组件的所有内容。

5. 时钟源设置

实验电路中时钟是一个可计数暂停的时钟源，如**图 19** 所示。其默认状态和普通时钟源一样，如将控制位 D 触发器的值利用手型戳工具设置为 1，时钟变成计数暂停模式，运行到指时钟运行到指定节拍数会自动暂停。

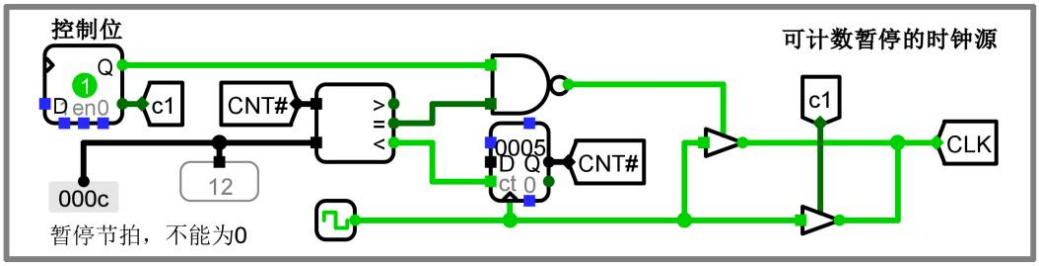


图 19 可计数暂停的时钟源

处理器系统联调时可以灵活利用该时钟源，首先通过头歌平台确定出错时钟节拍数，将时钟源中的暂停节拍常量设置为出错时钟节拍数-1，设置控制位为 1，时钟自动运行并暂停后，关闭时钟自动运行模式，将控制位清零，单步运行时钟即可快速复现出错现场进行精准调试。

6. 常见故障分析

头歌平台可以提供快速精准的测试反馈，在线评测单周期 CPU 时会给出程序运行出错的节拍，以及出错节拍对应的 PC、IR、RegWrite、RDin、MemWrite、MDin 等数据的预期输出和实际输出值。注意 24 条指令的在线测试由于 benchmark 程序运行周期数较长，测试采用静默方式，预期输出为空，当实际输出有问题时会在实际输出窗口输出预期输出和实际输出，并停止测试。通过分析出错节拍（注意是十六进制）和出错信号，很容易发现数据通路和控制信号的逻辑故障。

(1) PC 正确、IR 正确：其他信号不正确，说明当前指令的控制逻辑有问题，

利用可暂停时钟执行到对应节拍，仔细检查相关逻辑排查故障。

(2) PC 正确、IR 错误：说明用户加载的程序不是标准测试程序，请重新加载正确的测试程序再进行测试。

(3) PC 错误、IR 错误：如果出错节拍是 1，表示指令顺序寻址逻辑有问题，否则表示上一个节拍执行的指令是分支指令，分支逻辑存在故障导致程序执行到错误的位置。利用可暂停时钟执行到前一个节拍，检查分支指令执行逻辑排查故障。

二、 五段流水 CPU 设计流程

1) 完成理想流水线。将指令过程分成 5 个阶段 IF ID EX MEM WB(不得简化成 4 段流水线)，**建议分支指令在 EX 段完成，**不同阶段之间设置缓冲接口部件，构建各阶段之间的接口部件，接口定义尽可能简化，流水线应向后续段传递数据信息，控制信息，向前段传递反馈信息，后续部件对数据的加工处理依赖于前阶段传递过来的信息。ID 段译码生成该指令的所有控制信号，控制信号将逐段向后传递（越到最后阶段，信号越少），后续部件控制信号不再单独生成。**注意单周期 CPU 中的控制器是可以在 ID 段复用的。**

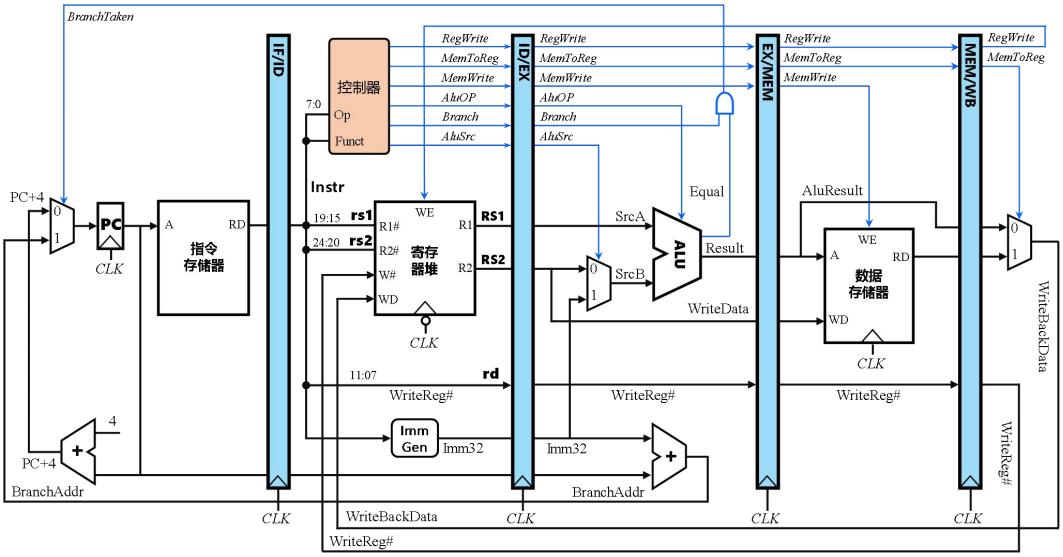


图 20 可计数暂停的时钟源

2) 增加流水冲突检测器。要思考流水冲突检测器在那个阶段？不同类型的指令数据相关性的区别，Load-Use 相关如何检测？相关检测器如何封装，输入输出是什么？**（问题很多，请大家仔细阅读教材第 7 章），设计时请注意增加计数器正确统计冲突，气泡等关键指标。**

3） 增加实现气泡机制的流水线。如何插入气泡，接口是否需要修改，如果需要修改，应该在步骤 2 中就考虑清楚，分支相关和数据相关时 CPU 如何插入气泡，如何保证流水线功能的正确性？

4） 实现采用数据重定向机制的流水线。需要思考在那个阶段完成数据重定向，那个阶段进行检测，答案可能不是唯一的。数据重定向的数据来源来自哪里？采用数据重定向机制后是否还需要插入气泡？Load-Use 冲突如何解决？

三、 单级中断机制设计实验

1、实验内容

为已实现的支持 24 条指令的单周期 CPU 增加单级中断处理机制，要求支持 3 个外部按键中断源，如图 19 左下角所示。

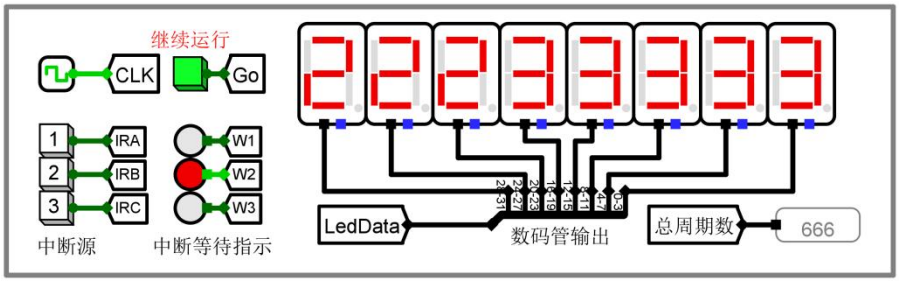


图 21 支持单级中断的单周期 CPU 电路框架

该 CPU 支持 3 个 Logisim 按钮触发的中断源，分别对应编号为 1、2、3 的三个按钮，3 个 LED 指示灯 W1、W2、W3 分别表示对应中断源的存在中断请求，中断处理完成时 LED 熄灭，中断优先级 1＜2＜3，CPU 执行中断服务程序时不能被其他中断请求中断。

实验提供了一个标准的单级中断测试程序，主程序的功能是一个 0~F 的往复循环走马灯 LED，按下某个按键后，对应按键的数字也会在 LED 上循环显示 3 次，表示主程序被中断并进入了对应的中断服务程序，LED 最右侧数字为循环计数值，方便观察中断服务程序执行进度。中断服务程序具体功能请仔细阅读实验包中的“单级中断测试程序.asm”文件。

2、实验步骤

1）**增加中断按键信号采样电路，**具体电路可参考图 22 这里 IR 就是中断请求寄存器，输出与中断屏蔽位进行逻辑与后送中断优先编码器，同步清零信号用于清除中断请求信号，注意中断请求信号必须等待中断服务程序执行到中断返回时才能清除，中断等待指示 LED 用于指示当前中断请求，中断服务程序返回时应熄灭。

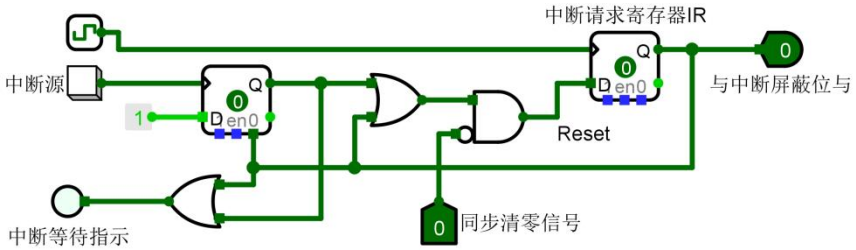


图 22 中断按键采样电路

2) **实现与中断相关的寄存器，**包括中断使能寄存器 IE、异常程序计数器 EPC。IE 用于开关中断，1 表示开中断，0 表是关中断，开关中断建议采用同步置位和复位方式。EPC 用于存放中断程序返回地址，在中断响应阶段硬件会自动将主程序 PC 值送 EPC 保存。

3) **设计中断识别逻辑，**能实现实验要求的中断响应优先级，能正确识别 1~3 号中断源，并设计向量中断机制，可由中断号寻找中断程序入口地址，为简化实现，中断向量表可以直接用硬逻辑实现（中断入口地址固定）。中断识别部分设计可以采用优先编码器实现，由于不需要动态调整中断处理优先级，所以中断屏蔽寄存器部分电路可以省略。

4) **增加中断隐指令数据通路，**中断响应周期需要实现硬件关中断、将主程序断点保存至 EPC 寄存器、将中断识别逻辑产生中断服务程序入口地址送 PC，请逐一实现以上各数据通路。

5) **增加 *uret* 指令数据通路，**单级中断服务程序主体是保护现场，中断服务，恢复现场，开中断，中断返回。保护现场、恢复现场可以采用堆栈方式实现，配合 *sp* 寄存器，利用已实现的 *lw*、*sw* 指令即可实现；中断服务无特殊指令，而开中断在 CPU 中涉及协处理器的操作，相对比较复杂，为简化设计，可以将开中断操作集成到中断返回指令 *uret* 中一并实现，所以本实验只需增加 *uret* 的数据通路即可。这里 *uret* 指令主要功能是将 EPC 寄存器送 PC，开中断，发送中断结束信号熄灭当前中断请求的指示灯。

6) **存储区间规划，**考虑数据存储器存储区间如何划分，数据区和堆栈区如何分配，其中堆栈区主要用于中断服务程序中的保护现场和恢复现场的堆栈操作，需要考虑堆栈指针 *sp* 如何进行初始化，堆栈放在哪个地址合适；另外需要考虑指令存储器的区间规划，中断服务子程序存放在哪里比较合适，堆栈指针在哪里进行初始化？

7) **编写中断服务程序，**实验资料包中已经提供了标准的单级中断测试程序，仔细阅读代码，了解主程序功能以及堆栈初始化逻辑，掌握中断服务程序功能，了解初始化堆栈，保护现场、恢复现场、中断返回的对应指令和代码。

8) **系统联调，功能测试，**完成所有设计后，在指令存储器中加载“**单级中断测试程序.hex**”，启动时钟自动运行，观察主程序走马灯运行状态，在主程序运行时任意点击 1、2、3 号按键中的任何一个，对应中断指示灯应该点亮，CPU 应正常进入中断演示程序执行，中断演示程序执行能正确返回主程序，同时中断指示灯熄灭。如中断服务程序执行期间又有新的按键被按下，中断指示灯点亮，当前中断服务程

序结束时可入新的中断服务程序响应。如存在问题，可以通过头歌平台进行更加精准的测试，利用平台反馈的错误信息快速定位故障。

3、实验思考

(1) 不同的中断请求存储在哪里，何时消失？

不同的中断请求通常保存在 CSR 寄存器组中的中断请求寄存器 IR 中，其寄存器的每 1 位对应一个中断请求。对应中断请求应在中断服务程序结束时清除，这里应该由硬件完成这一清零动作，可以在 *uret* 指令中一并实现。

(2) 硬件响应优先级用什么电路实现，为什么要有处理优先级？

硬件响应优先级一般采用优先编码器实现，同一时刻优先级最高的中断请求被 CPU 响应。只有没有更高优先级的中断请求时，电路才会把较低优先级的中断请求送给 CPU，优先编码器在 Logisim 中有现成的组件。

(3) 中断使能寄存器 IE 的用途是什么？

中断使能寄存器就像是一个开关，用来打开或关闭 CPU 的中断请求，外部设备的中断请求信号会与这个开关进行逻辑与操作。只有中断使能有效时，CPU 才能接收到中断请求，无效时 CPU 无法接收到任何中断请求，也就无法响应中断。

(4) 开中断，关中断如何实现？

x86 指令集中的 sti（开中断）、cli（关中断）指令就是用于控制中断使能寄存器的；RISC-V 处理器中没有专门的开中断、关中断指令，实际实现时是利用控制状态寄存器 CSR 访问指令实现的，可以通过设置或清除 uie 和 ustatus 寄存器中中断使能位实现。在单级中断中可以不实现开关中断指令，在中断响应时硬件关中断，在中断返回时硬件开中断。

(5) CPU 如何判断当前有中断需要响应？

CPU 判断当前有中断响应的依据是连接 CPU 的中断请求信号 IR，当该信号为高电平时，CPU 必须进行中断响应。对具体外部设备而言，其中断请求要被 CPU 响应，必须在该中断请求没有被屏蔽，无更高优先级的中断请求，且中断使能寄存器 IE 有效，CPU 才能收到最终的中断请求信号，当 CPU 正在执行的指令执行完毕进入公操作阶段时，CPU 才会响应该设备的中断请求。

(6) CPU 检测到中断请求需要进行什么操作，哪些是硬件完成，哪些是软件完成，中断响应周期需要多少个时钟周期？

对于单级中断系统，CPU 会进行如下一系列动作：①中断响应：包括关中断、保存断点、中断识别；②保护现场；③中断服务；④恢复现场；⑤关中断；⑥中断

返回。中断响应操作由硬件完成，也就是所谓中断隐指令完成的任务，其他任务由软件实现。中断响应周期在单周期实现中不占用时间，在多周期实现中与具体实现有关。

(7) 单级中断中的断点保存在哪里？

单级中断中的断点通常保存在一个专用的寄存器中，就 RISC-V CPU 而言，是保存在 CSR 中的 EPC 寄存器中。

(8) 中断服务程序入口地址如何查找？

本实验中只有 3 个中断源，可以采用独立式中断请求方式进行中断仲裁，利用优先编码器电路即可实现中断源的识别，再增加中断向量表逻辑实现中断号到中断入口地址的转换，真实的中断向量表涉及内存数据访问，逻辑较为复杂，为简化实验设计，这里可以将中断服务程序入口地址可以固化到电路中。

(9) 中断处理程序中的现场有哪些？

中断响应时 CPU 要转去执行中断服务程序，执行完后再返回被暂停的程序，因此在改变 CPU 状态的任何操作之前，会被修改的状态都需要保存起来，中断返回时再恢复。CPU 的状态都是通过寄存器实现的，所以保护现场就是要备份所有即将修改的寄存器的值，主要有 mEPC、中断屏蔽字、以及其他需要写入的通用寄存器。

(10) 单级中断测试程序的 3 个中断服务程序程序入口地址如何查找？

中断服务程序与主程序一起编译，然后统一导出为存储器镜像文件，加载到指令存储器中，用户可以利用 RARS 对源程序进行汇编后，在 Settings 菜单中打开“Show labels Window”查看对应中断服务程序标签的具体地址。

(11) 数据堆栈放在哪里？RISC-V 如何访问堆栈？

数据堆栈通常放在数据存储器中。RISC-V 中并不存在类似 X86 中的 PUSH，POP 堆栈指令，通常利用内存访问指令配合堆栈指针寄存器 SP 控制实现的，MIPS 处理器的堆栈是向下生长的，每压栈保存一个值，SP 要减 4，出栈时则 SP 需要加 4，具体例子如下：

```
# 设置 SP 指针
li sp, STACK_BASE_ADDR   #堆栈指针初始化
addi sp, sp, -4      # 入栈
sw  ra, 0(sp)
lw  ra, 0(sp)    # 出栈
addi sp, sp, 4
```

(12) 按键中断是电平触发还是跳变触发？连续按键如何处理？实际系统中是如何处理的？

按键一般是边沿触发，但是处理边沿中断触发通常较为困难，因此常用锁存器将边沿触发转换为电平信号，然后用电平中断触发。连续按键最简单的处理是在第一次按键没有处理完之前，忽视后续的按键动作，也就是发生按键时锁存器锁存，不再接受新的按键信息，只有当 CPU 将对应按键中断处理完毕后，再开放这个锁存器。实际系统中这个锁存器是一个缓存，可以保存按键的队列信息，这样 CPU 就可以一个一个的处理，直至保存的按键缓存队列全部被处理。实际系统运行时中断处理响应非常快，操作系统也规定中断服务程序不能太长，连续按键都可以得到及时响应。

(13) 实验中的中断机制什么要使用 CSR 寄存器组？在我们的实验中如何简化？

不使用 CSR 也可以满足实验要求，但使用 CSR 寄存器组实现中断机制的目的是方便程序的汇编。实验中我们采用 RARS 汇编器来汇编程序，因此符合 RISC-V 规范的实现更加方便。但 RISC-V 处理器中 CSR 寄存器的定义较为复杂，具体实现时可以根据需要尽量进行简化，只要能实现中断相关机制即可。

四、 多重中断机制设计实验

1、实验内容

多重中断处理机制，要求支持 3 个外部按键中断源。CPU 支持 3 个 Logisim 按钮触发的中断源，分别对应编号为 1、2、3 的三个按钮，3 个 LED 指示灯 W1、W2、W3 分别表示对应中断源的存在中断请求，中断处理完成时 LED 熄灭，中断优先级 1＜2＜3，高优先级中断应该正确中断低优先级中断服务子程序。

根据 EPC 寄存器是采用硬件堆栈还是内存堆栈保护（2 选 1），实验提供了 2 个标准的多级中断测试程序，两个程序的主程序功能都是一个 0~F 的往复循环走马灯 LED，按下某个按键后，对应按键的数字也会在 LED 上循环显示 3 次，表示主程序被中断并进入了对应的中断服务程序，LED 最右侧数字为循环计数值，方便观察中断服务程序执行进度。中断服务程序具体功能请仔细阅读实验包中的“多级中断测试(EPC 内存堆栈保护).asm”、“多级中断测试(EPC 硬件堆栈保护).asm”文件，注意对比二者在中断服务程序保护现场、开关中断的代码差异。

2、实验步骤

（1） **增加 csrrw、csrrsi、csrrci 指令数据通路**。需要使用这 3 条 CSR 访问指令进行中断控制，实现开中断、关中断、upec 寄存器读写的功能。关于这些 CSR 寄存器的使用规范具体可以参考 RISC-V 指令手册。但真实实现较为复杂，为简化实验，我们可以根据实验的需求进行最大的简化，能实现中断机制即可。如采用硬

件堆栈保护 EPC 寄存器，则只需要将 csrrsi、csrrci 两条指令分别对应开关中断指令即可；如采用内存堆栈保护 EPC 寄存器，则需要完全实现 csrrsi、csrrci、csrrw 三条指令对 CSR 寄存器的读写访问功能。

```
3、# 中断相关指令
4、csrrsi zero,0x4,1    #开中断，设置 0 号 CSR 寄存器 ustatus.MIE=1
5、csrrci zero,0x4,1    #关中断，设置 0 号 CSR 寄存器 ustatus.MIE=0
6、csrrsi s6,0x41,0     #s6=uepc，读 65 号 CSR 寄存器 uepc 寄存器
7、csrrw zero,0x41,s6   #mepc=s6，写 65 号 CSR 寄存器 uepc 寄存器
```

（2） **修改中断服务程序**，相对单级中断，多重中断中 EPC 寄存器也必须作为现场进行保护，如采用内存堆栈方式保护 EPC 寄存器，则必须通过 csrrsi 指令读取 EPC 寄存器的值到通用寄存器后压栈，恢复现场时从堆栈弹出EPC寄存器的值，然后利用 csrrw 指令写入 EPC 寄存器；另外多重中断保护现场结束后应开中断，恢复现场之前也需要关中断。。

（3） **系统联调，功能测试**。完成所有设计后，根据实际设计加载“多重中断测试程序(EPC 内存堆栈保护).hex”或“多重中断测试程序(EPC 硬件堆栈保护).hex”，启动时钟自动运行，观察主程序走马灯运行状态，在主程序运行时任意点击 1、2、3 号按键中的任何一个，对应中断指示灯应该点亮，CPU 应正常进入中断演示程序执行，中断演示程序执行能正确返回主程序，同时中断指示灯熄灭。如中断服务程序执行期间又有新的按键被按下，中断指示灯点亮，高优先级中断服务程序能打断低优先级中断服务程序，中断后应返回低优先中断服务程序直至主程序。如存在问题，可以通过头歌平台进行更精准的测试，利用平台反馈的错误信息快速定位故障。

8、实验思考

(1) 多级嵌套中断的断点如何处理？

单级中断中的断点保存在 EPC 寄存器中，进行多级中断嵌套时，EPC 的值会被后面的中断破坏，所以中断服务程序中应将 EPC 作为现场保存起来，EPC 实际上也是一个被调用者保存寄存器，类似 JAL 嵌套调用需要保护 ra 寄存器一样。

(2) 高优先级中断服务程序执行过程中，有新的按键事件发生，如何处理？

如新的按键中断比当前中断服务程序的优先级更高，就暂停当前正在执行的中断服务程序，转而执行更高优先级中断服务程序；否则需要等待当前中断服务程序执行完毕并返回后，才能够进行中断响应；注意 CPU 响应与否取决与能否收到中断请求信号，如果收不到就不响应。

(3) 中断屏蔽寄存器有什么作用，本实验是否需要中断屏蔽寄存器？

中断屏蔽寄存器用于存放中断屏蔽字，中断屏蔽字用于动态调整中断处理优先级；中断屏蔽字是在中断处理过程中设置的，在真实计算机环境下，通常是由中断服务程序进行设置。由于本实验中并不需要动态调整中断处理优先级，因此可以不

设置中断屏蔽寄存器。

五、 流水中断机制设计实验

1、实验内容

为五段流水 CPU 增加中断处理机制（单级中断、多级中断任选），要求支持 3 个外部按键中断源，如**错误!未找到引用源。**左下角所示。该 CPU 支持 3 个 Logisim 按钮触发的中断源，分别对应编号为 1、2、3 的三个按钮，3 个 LED 指示灯 W1、W2、W3 分别表示对应中断源的存在中断请求，中断处理完成时 LED 熄灭，中断优先级 1＜2＜3。

实验测试程序参考单级中断和多重中断实验，主程序的功能是一个 0~F 的往复循环走马灯 LED，按下某个按键后，对应按键的数字也会在 LED 上循环显示 3 次，表示主程序被中断并进入了对应的中断服务程序，LED 最右侧数字为循环计数值，方便观察中断服务程序执行进度。由于流水中断实现方案差异性较大，所以头歌平台并为提供标准测试，需要读者耐心进行调测。

2、实验思考

(1) **单周期 CPU 中断处理和流水中断处理有何区别？**

在单周期 CPU 中每条指令执行完毕后，如果有中断请求，就进行中断响应，被中断的指令是确切的。但指令流水线中同时有 5 条指令在不同阶段执行，可能有多条指令都进入了结束阶段，比如无条件分支指令可能安排在 ID 段，有条件分支指令可能安排在 EX 段执行，中断时到底中断哪条指令的执行，同时执行的其它指令怎么办都需要考虑。

理论上，任意一个流水段上都可以处理中断，为了实现方便通常会选择一个段来处理中断，可供选择的段常常是 EX 或者是 WB。不管选择那个流水段来处理，都要做下面的事情，以在 EX 段处理中断为例：①MEM、WB 段的 2 条指令要继续执行完；②IF、ID 段的 2 条指令要取消；③EX 段指令如果继续执行完，断点地址应该是该顺序指令地址或分支跳转地址；如果不允许继续执行，断点就是 EX 段指令的 PC；（通常做法是不允许这条指令继续执行）；另外如有指令在 EX 段之前完成，比如无条件分支指令在 ID 段完成，此时逻辑又有区别；④暂停流水线进行中断响应；（根据具体的实现方法也可以不暂停）；⑤重新启动流水线从中断入口地址处取指令，进入中断服务阶段。

六、 注意事项

1) 本课程设计的内容基本无法在一本教科书中弄清楚，请尽可能的仔细阅读资料包中的参考文献。
2) 在实现 CPU 时，你可以使用任何 LOGISIM 内建的电路组件。

3) 指令 ROM 和数据 RAM 必须在 main 电路中可见，不能封装在子电路中。
4) 显示模块应该在主电路中可见。
5) 主要部件之间还是需要适当连线，隧道工具不能过度使用，要能看清楚各部件之间的连接关系。
6) 尽可能的使用标签工具去注释你的电路，包括控制信号，数据通路，显示模块，总线等，这会让你的电路更加容易调试！
7) 注意标签以及注释的命名规范，过长的命名都会对后续的画图连接造成影响。
8) LOGISIM 中可以将不同的模块用不同的颜色区分，建议用颜色区分各接口部件和关键模块。
9) 流水接口部件封装尽可能封装的长一点，各接口部件建议等长，否则控制线多了以后可能不方便布线。
10) 最终的流水 CPU 图相对较为复杂，请注意画出整齐整洁的原理图，各段之间连接尽量用连线表示，有利于结构的清晰，适量的隧道避免线缆的大量交叉。
11) PC，IR 最好一直传递到最后一级，这样方便观测流水线运行的状况。
12) 流水线各级是否产生气泡可以用 LED 指示灯显示，方便观察流水线运行状况。
13) 先在 LOGISIM 仿真平台上将原理跑通后再考虑上 FPGA 开发板。
14) 各里程碑版本经过充分测试后，备份后再开新的分支进行新的开发，以避免新版本无法开发成功，老版本又检查不了的悲剧。
15) **严禁对对时钟信号进行逻辑操作**，否则可能会引发险象和毛刺，导致无法预料的逻辑错误。**所有带状态的部件请尽可能设计统一复位信号，便于系统复位调试。**