基于 logisim 的单周期单中断 MIPS-CPU 设计

学　　号	E12214052	专　　业	计算机科学与技术	姓　　名	赵宸宇
实验日期	2024 年 11 月 1 日	教师签字		成　　绩	

摘要

本实验设计了一个单周期单中断的 MIPsCPU，该 CPU 设计的特点有：

中断隐指令实现，中断控制部件的实现，中断存储部件的实现，中断逻辑设计。

延续 24+ 可扩展 CPU 设计传统，这次设计同样在硬件电路设计上采用了冗余设计，可以很方便地添加更多中断信号。

同时本次设计也同样大量使用三态门、等组件，简化电路设计。在中断择优电路中使用三态门和替换了多路选择器。

使用原本的带显示窗口的 reg file，以方便后续实验观察运行情况。

大量使用隧道，增强电路的逻辑性、可调试性、可读性，减少长信号线的数量。

实验的主要目的：

熟悉中断软硬协同的机制，能够设计支持单级中断的单周期 MIPS CPU。

附加项目：键盘中断信号添加（仅添加了 INT4 作为键盘中断信号）

实验的主要任务：

1. 构建 MIPSCPU **数据通路**
2. **单周期单中断控制综合逻辑**实现
3. 软硬件**测试联调**
4. 回答灵魂拷问

实验产出：

1. 单周期单中断电路图
2. 头哥网通关记录
3. 实验报告（LaTeX）
4. 支持材料
5. **github | gitee** 异步更新请见`https://gitee.com/cslearnerer/AHU-CSHT`或`https://github.com/YUCHENYUXI/CPUDesign`

目录

一、【实验内容】

1.1 24+ 指令数据通路复用

上次实验中设计所得 24+ 指令单周期 MIPsCPU 的数据通路具有良好的可扩展性、鲁棒性、美观性。故而此次实验的数据通路将基于此 24+ 指令数据通路框架，然后通过分析单中断 CPU 设计的需要，以实现单中断 CPU 为目标进行电路扩充即设计。

首先，将上次 24+ 指令 CPU 的数据通路拷贝到单中断电路中作为模板。然后使用 benchmark 程序检查拷贝的成功性。

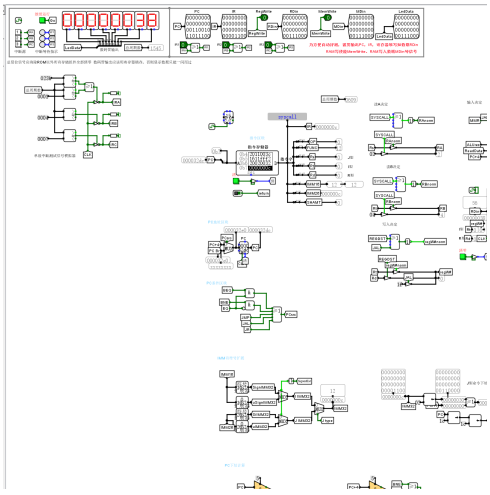


图 1　24+ 指令 CPU 数据通路

通过 benchmark 程序检验发现，仅仅拷贝电路而不加以修改，则无法正常运行。具体来说，异常情况表现为 clk 给上升沿信号，PC 却不会进入下一条指令。经过检查，确定为 CLK 隧道标签的信号异常。经过进一步排查，发现该电路（左上角）自带一个时钟信号，该信号与原本电路的 clk 模块冲突。将 clk 信号通路阻断，拷贝的电路可以正常运行。准备工作完毕。

1.2 中断电路设计

接下来进行单中断电路设计，并同步进行软硬件联调工作。

通过观看学习 Mooc 网课，可以得知：要实现单中断控制，就必须分步实现、联调以下各个功能模块（如下图）：

1.2.1 基本模块分析

1.2.2 中断控制信号的生成与保持与选择性输出

如图 3，中断请求生成逻辑已经被实验课题组预先放在电路中，可以直接使用。为了满足实验要求，本次设计加入了 INT 4，即键盘中断模拟信号。

图 2　MOOC

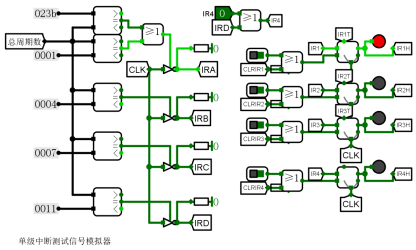


图 3　中断控制信号的生成

如图 3所示，该电路为单级中断测试信号模拟器，其作用是在预先设定好的某个周期的下降沿触发一次同步中断信号。

考虑到：

这种中断信号是一次性信号，一旦周期数递增，则中断信号消失。故而必须对其进行保持，也就是采样。为了能够保持中断信号，这里对信号采样器进行集成，得到采样模块。这个模块可以记录一个中断脉冲，并提供中断结束时清理中断的接口。通过集成，得到了图 4电路，其中右侧方形为中断采样器，是原始电路提供的采样器的封装形式：

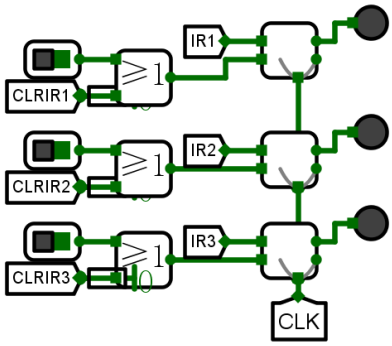


图 4

为了实现中断仲裁，本实验设计采用了一种基于三态门的中断优先级判断方法，并

对其进行封装。如下图所示，优先级信号具有优先、覆盖、互斥关系，故而可以借助三态门控制信号流出，来控制仅仅将最高级中断信号的流出：

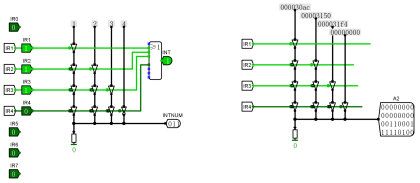


图 **5**

如图 5所示，不同等级信号之间存在互斥、覆盖关系。而三态门原件恰蕴含此逻辑关系，这样就可以让最高级中断信号控制本级、低级信号的三态门的通、断，进而实现了优先级序号、INT 信号、中断向量的输出始终指向最高级信号。

将上述采样电路、输出电路组合，可以得到一个中断控制器（ICU）一部分。

1.2.3 中断响应、中断结束电路分析

为了实现单级中断的开关、PC 切换，还需要一系列寄存器存储状态、地址信息。具体来说，就是还需要一个 IE——中断允许触发器存储中断状态（关中断 IE = 1，开中断 IE = 0）；一个 EPC——中断返回寄存器（保存 PC 指令指针 REG 的断点）。

1.2.4 中断响应与中断隐指令

当任一中断到来时，INT 信号由低打高，说明存在中断需要响应。这时需要由硬件执行一系列预先定义好的操作，这些操作就是所谓的“中断隐指令”。该实验设计包括以下中断隐指令：中断检测：当 IE=0（没有执行中断），INT=1（存在中断）时，说明存在中断没有执行，这时就要响应中断信号。同时保存断点：在下一个上升沿信号到来时，将当前的程序计数器（PC）的值提前送到 EPC 入口，使用上升沿信号更新 EPC 值。中断服务程序的入口跳转：在下一个上升沿到来时，通过判断优先级，得到中断类型码，根据中断类型码找到相应的中断服务程序的入口地址，提前将此地址送往 PC，通过下一个上升沿信号将 PC 设置为这个地址，进入中断服务子程序。为了防止在中断服务程序执行期间再次被新的中断请求打断，将在下一个上升沿信号到来时设置 IE1=1，以阻止其他中断请求。

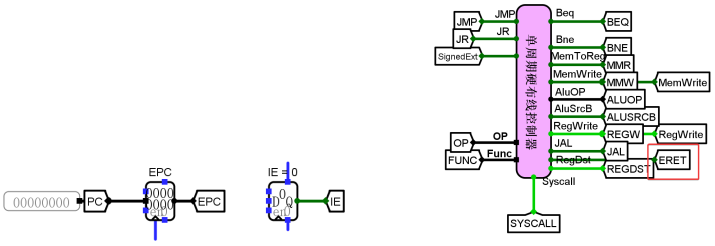
1.2.5 结束中断

当中断服务子程序结束，由子程序执行特定的返回指令 ERET（中断返回指令）。返回检测：ERET 指令经过译码可以得到控制信号 ERET=1。此时若还有 IE=1，则说明需要退出当前中断。这时，执和上面行相反的步骤。即在下一个上升沿：提前将 EPC 断点

送 PC 入口，利用上升沿更新 PC，恢复之前保存的断点地址，使得 PC=EPC。然后，同时将 IE 清零，并将中断服务信号 INT n 清零，实现开中断与中断信号清除。

1.3 中断响应电路设计

本实验采用 1bit IE REG 和 32bit（匹配 PC）的 EPC 寄存器担任状态存储工作核心。并为单周期硬布线控制器增加了 ERET 指令的控制逻辑。按照设计方案，首先需要设计



(a) 标志信号存储　　**(b) ERET** 信号

图 **6**　信号存储 | **ERET** 指令控制器

一个中断检测逻辑：然后分步骤完成以下电路设计：

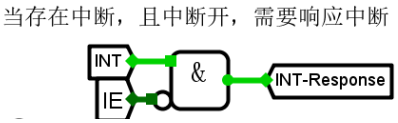


图 **7**

保存断点和关中断：在下一个上升沿信号到来时，将当前的程序计数器（PC）的值提前送到 EPC 入口，使用上升沿信号更新 EPC 值。为了防止在中断服务程序执行期间再次被新的中断请求打断，将在下一个上升沿信号到来时设置 IE1=1，以阻止其他中断请求。

该电路设计如下图所示：在第一周期下降沿，中断信号来到，经过判优，触发 INT

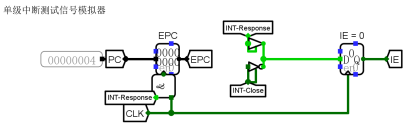


图 **8**

中断信号存在标志。做保存断点、关中断准备。然后通过一个上升沿，成功将 PC 保存至 EPC，同时关中断。

中断服务程序的入口跳转：在下一个上升沿到来时，通过判断优先级，得到中断类型码，根据中断类型码找到相应的中断服务程序的入口地址，提前将此地址送往 PC，通过下一个上升沿信号将 PC 设置为这个地址，进入中断服务子程序。如下图所示，使用

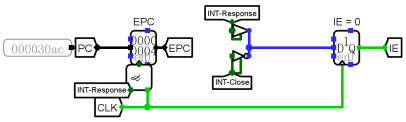
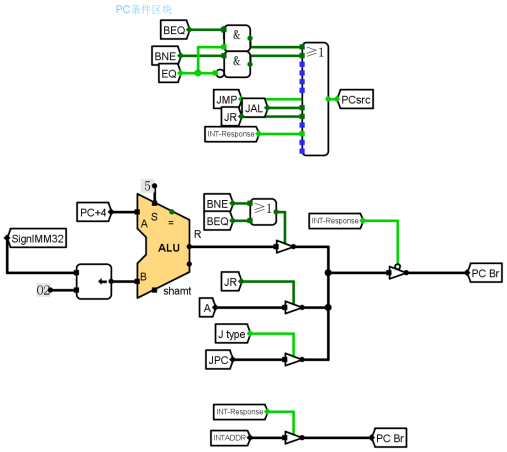
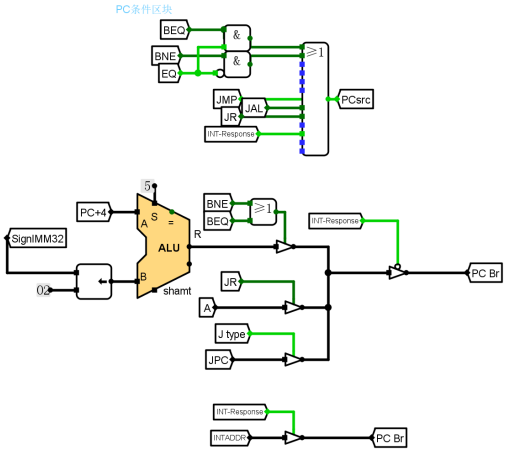


图 9

三态门实现中断服务地址抢占 PC Branch 信号，阻断指令更改 PC 地址，而将中断服务子程序地址送入 PC。如下图所示，当存在 J 型命令时，中断服务子程序地址可以抢先 J



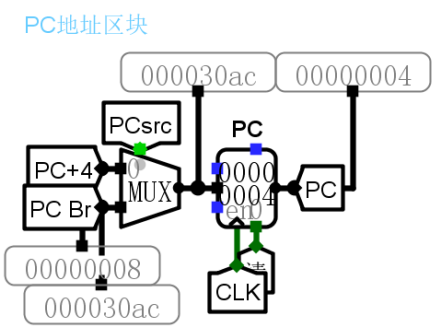
(a) 1



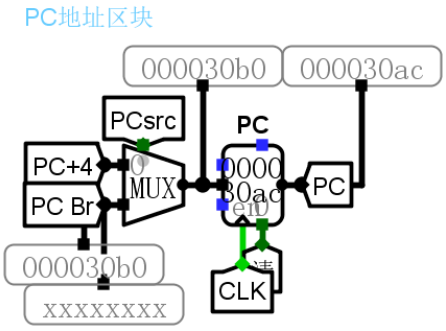
(b) 2

命令下址送入 PC 下址，然后经过一个上升沿，更新 PC，进入中断服务子程序。

经过一个时钟上升沿：



(a) 1



(b) 2

1.4 中断结束电路设计

根据上文分析结果。为了实现中断结束，需要：首先通过 mips 指令集获取 ERET 指令编码。经查，得知其编码为 0100 0010 0000 0000 0000 0000 0001 1000B 即 4200 0018H。将此指令手动载入中断服务子程序 1 入口处，见下图 RAM：然后进行中断结束电路设计：

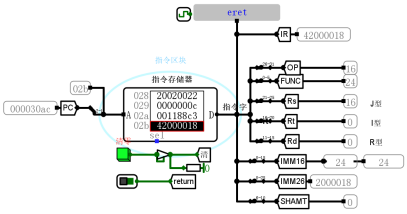
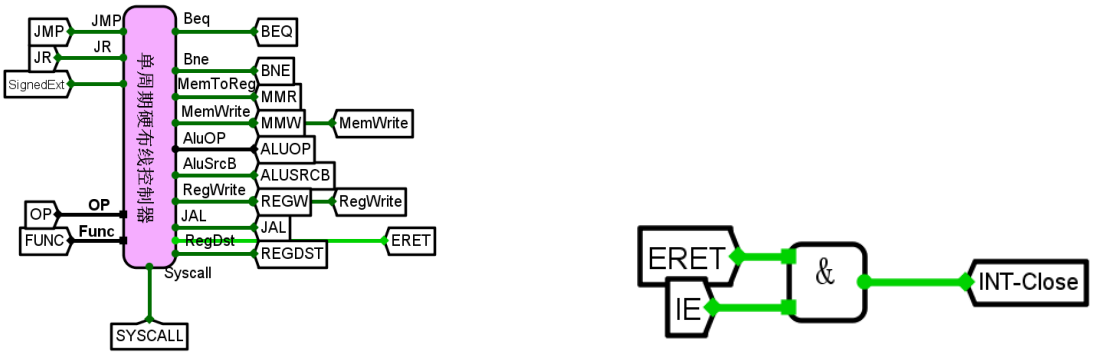


图 **12**

首先检查 ERET 信号是否正常生成。如下图，正常生成。

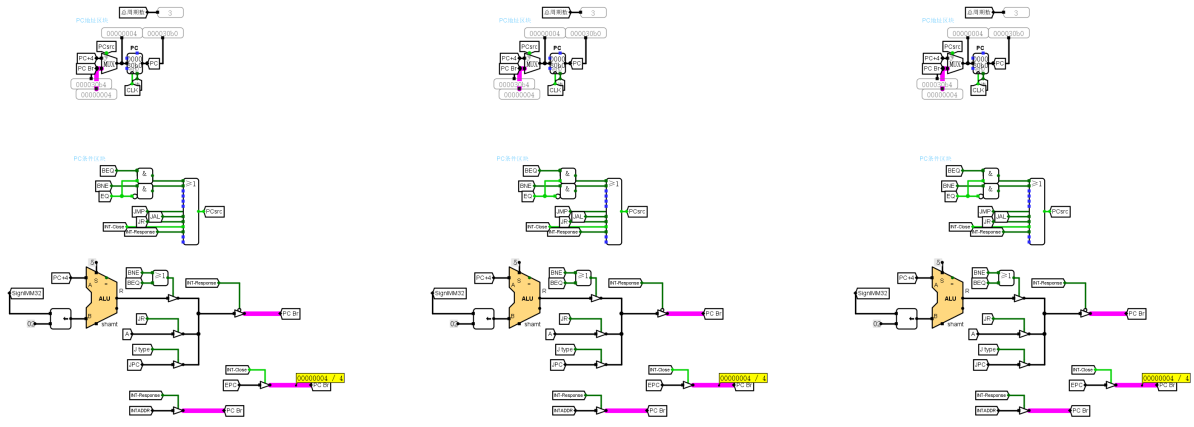
返回检测：ERET 指令经过译码可以得到控制信号 ERET=1。此时若还有 IE=1，则说明需要退出当前中断。这时，执行和上面相反的步骤。即在下一个上升沿：提前将



(a) **(b)**

图 **13**

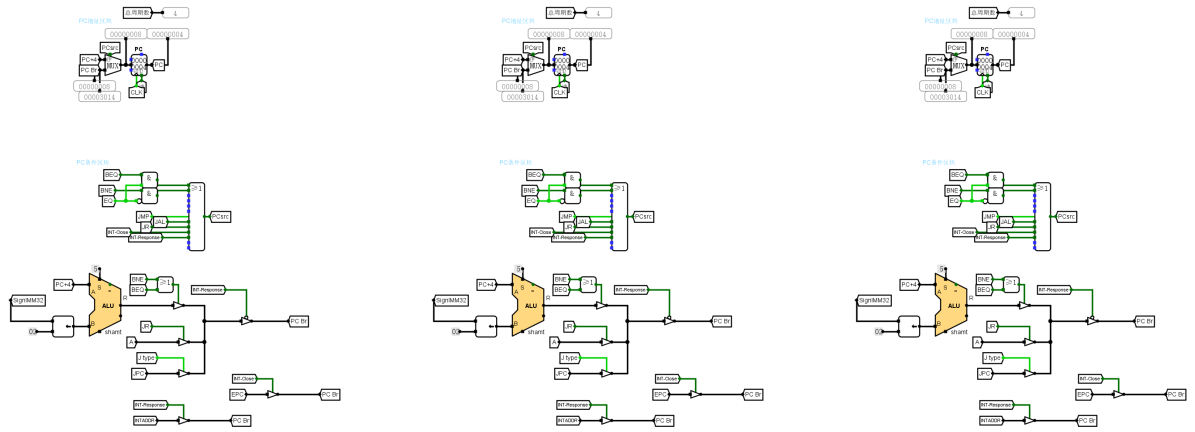
EPC 断点送 PC 入口，利用上升沿更新 PC，恢复之前保存的断点地址，使得 PC=EPC。如下图，在第四周期，因 ERET 指令，PC 回到断点 0x4：然后，将 IE 清零，实现开中



(a) **(b)** **(c)**

图 **14**

断，如下图所示，清零成功（注：此时 INT2 到来，CPU 做响应 INT2 准备）：

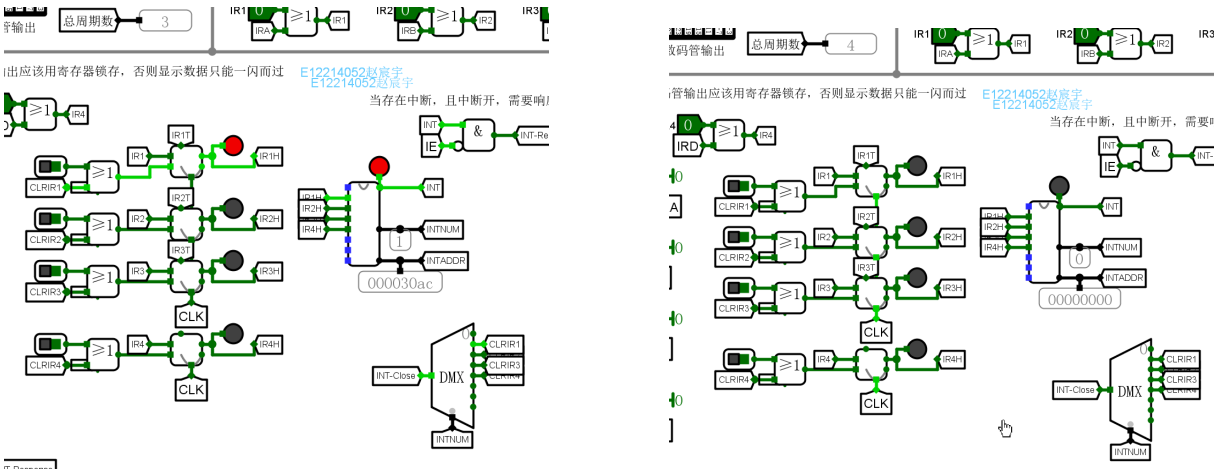


(a) (b) (c)

图 **15**

并将中断服务信号 INT n 清零，与中断信号清除。如下图所示，ERET 指令在第三周期执行，将 clear1 做清零准备：

在第四周期到来的上升沿，INT1 被清除：



(a) (b)

综上，经过中断结束指令，PC 回到断点，INTn 中断指令被清除，IE 被开启等待下一个中断响应。

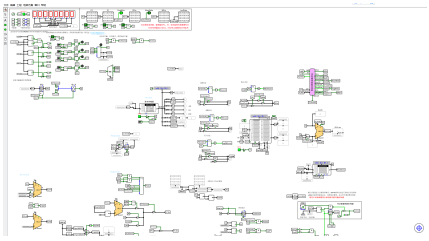


图 **18**

1.5 实验电路图概览

1.5.1 鸟瞰图

1.5.2 核心电路概览

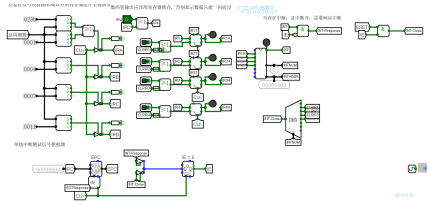


图 **19**

1.6 软硬件测试联调

经过了基础数据通路设计、进阶数据通路优化、硬布线控制器设计、联调控制电路控制信号优化四个设计阶段，并查阅了大量文档、使用了大量辅助工具，本实验才通过了头哥网测试，告示实验终于完成，同时得到了如上文所示的特色电路设计。

1.7 头哥过关记录

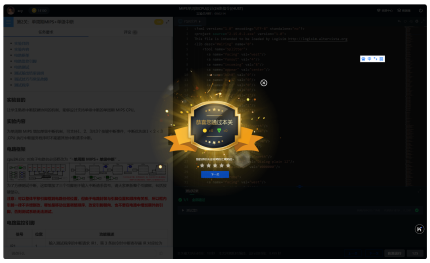


图 **20**

二、【灵魂拷问】

中断周期（隐指令周期）发生时，中断控制器将返回值送入 EPC, 中断入口地址送入 PC,IE=1（关中断）; 请问：此时钟周期的控制器部件在做什么？

此时钟周期的中断控制器完成保存断点、送中断入口、关中断任务。此时钟周期的指令控制器仍负责译码指令产生控制信号。但是在中断发生时，中断操作的优先级会大于当前指令的优先级，指令控制器产生的会影响中断操作的控制信号会被中断操作覆盖。

EG:2000H（假如该指令顺序执行）发生中断按键，中断隐周期控制器执行 2000H 还是执行 2004H?

中断隐指令控制器将执行 2004H。

三、【讨论】

本次实验的目标是实现单中断 CPU 的设计。首先，我将 24+ 指令数据通路复制到单中断电路中，并通过 benchmark 程序验证其正确性。接着，根据 Mooc 网课中的指导，逐步实现了中断请求生成、中断控制信号的生成与保持、状态寄存器设计等关键模块。我引入了 1bit 的 IE 寄存器和 32bit 的 EPC 寄存器，分别用于控制中断的开启与关闭以及保存中断发生时的 PC 值。

在中断响应与处理流程中，我设计了一系列中断隐指令，确保当任一中断请求到来时，能够准确地保存当前 PC 值至 EPC 寄存器，并根据中断类型码跳转到相应的中断服务程序。为了防止中断服务程序执行期间被新的中断请求打断，我在进入中断服务程序后立即设置 IE 寄存器以关闭中断。此外，我还实现了中断返回机制，通过 ERET 指令恢复中断前的状态，包括将 EPC 值重新加载回 PC，清零 IE 寄存器，并清除中断请求信号，确保系统能够平稳地从中断中恢复。

通过本次实验，我不仅掌握了单中断 CPU 的设计方法，还深入理解了中断机制的工作原理。实验过程中遇到的技术难题，如中断电路设计与联调、中断隐指令的实现、中断返回与状态恢复等，都促使我不断探索和学习。最终，我成功完成了单中断 CPU 的设计与调试，提升了硬件设计能力和系统调试技巧，为后续更复杂系统的开发奠定了坚实的基础。