# 计组实验 5 报告

孙启翔 241220098

May 2025

## 1 只读存储器实验

### 1.1 实验整体方案设计

这道题目的总体逻辑是输入 ASCII 编码值，而后在 LED 点阵组件上显示字符图案。如果输入的编码不是可显示字符，则输出错误标志 CodeErr=1。这里点阵字库文件已经提供好了，是 ascii8-16.zk。

因为需要同时输出 16 个字节的点阵数据，所以需要复制 16 个加载相同点阵字库的只读存储器，读取点阵数据时每个只读存储器的地址数据按点阵行的次序加 1 递增，则可以同时读取第 1 行、第 2 行、…、第 16 行的点阵数据，然后再同步输出到 LED 点阵组件的输入端中。

在点阵字库中 ASCII 码可见字符点阵从地址 0 开始存储，因此在根据 ASCII 码来读取点阵字库时，需要先把 ASCII 码值减去十六进制 0x20，然后根据每个字符的点阵数据是 16 个字节，把得到的差值左移 4 位在低位补 4 个 0，则得到该字符点阵在字库中的起始位置，字符点阵数据为连续存放的 16 个字节。

## 1.2 实验原理图及电路图

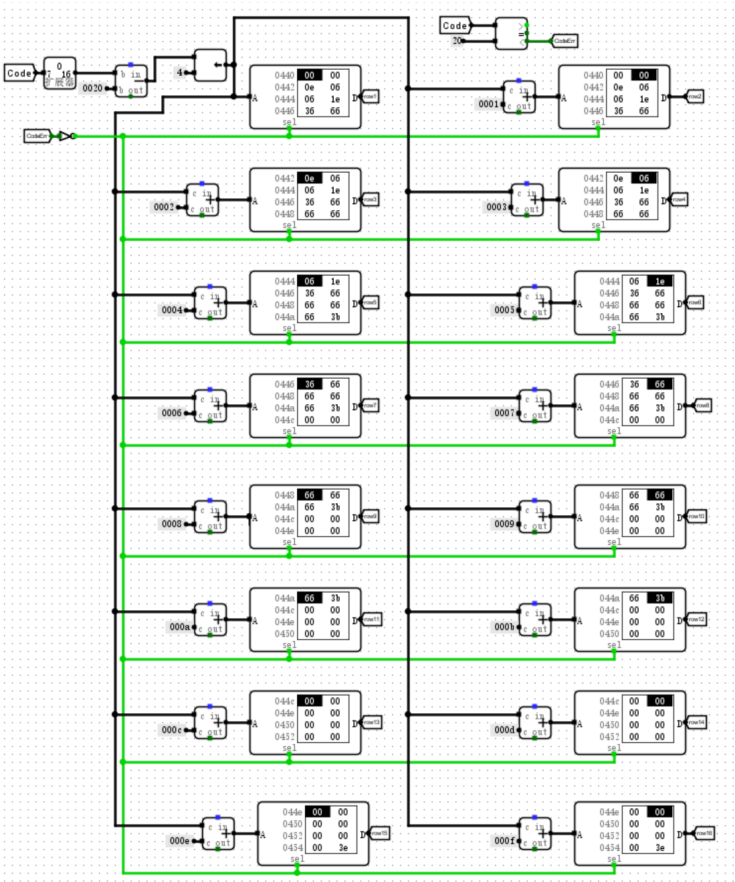


图 1: 5.1 电路图

## 1.3 实验数据仿真测试图

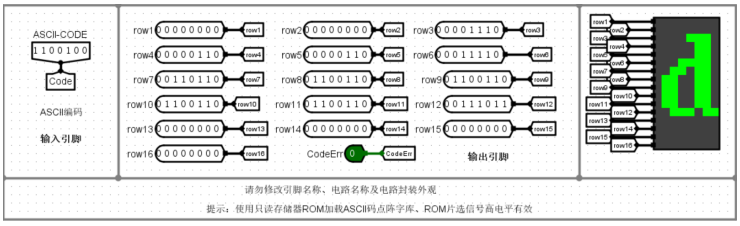



图 2: 5.1 仿真测试 1

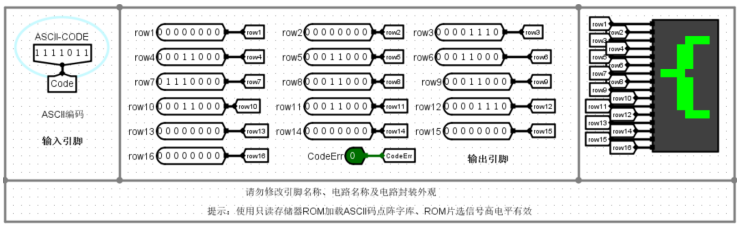



图 3: 5.1 仿真测试 2

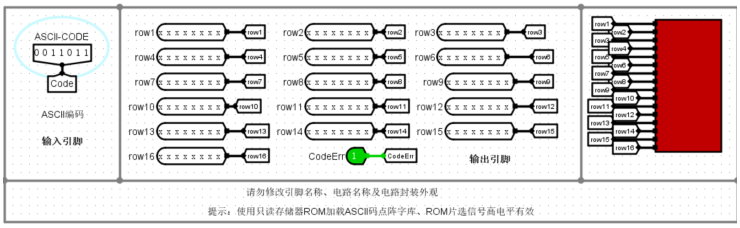



图 4: 5.1 仿真测试 3

### 1.4 错误现象及分析

第一次接触 ROM 部件，需要熟悉相关的操作等

## 2 数据存储器实验

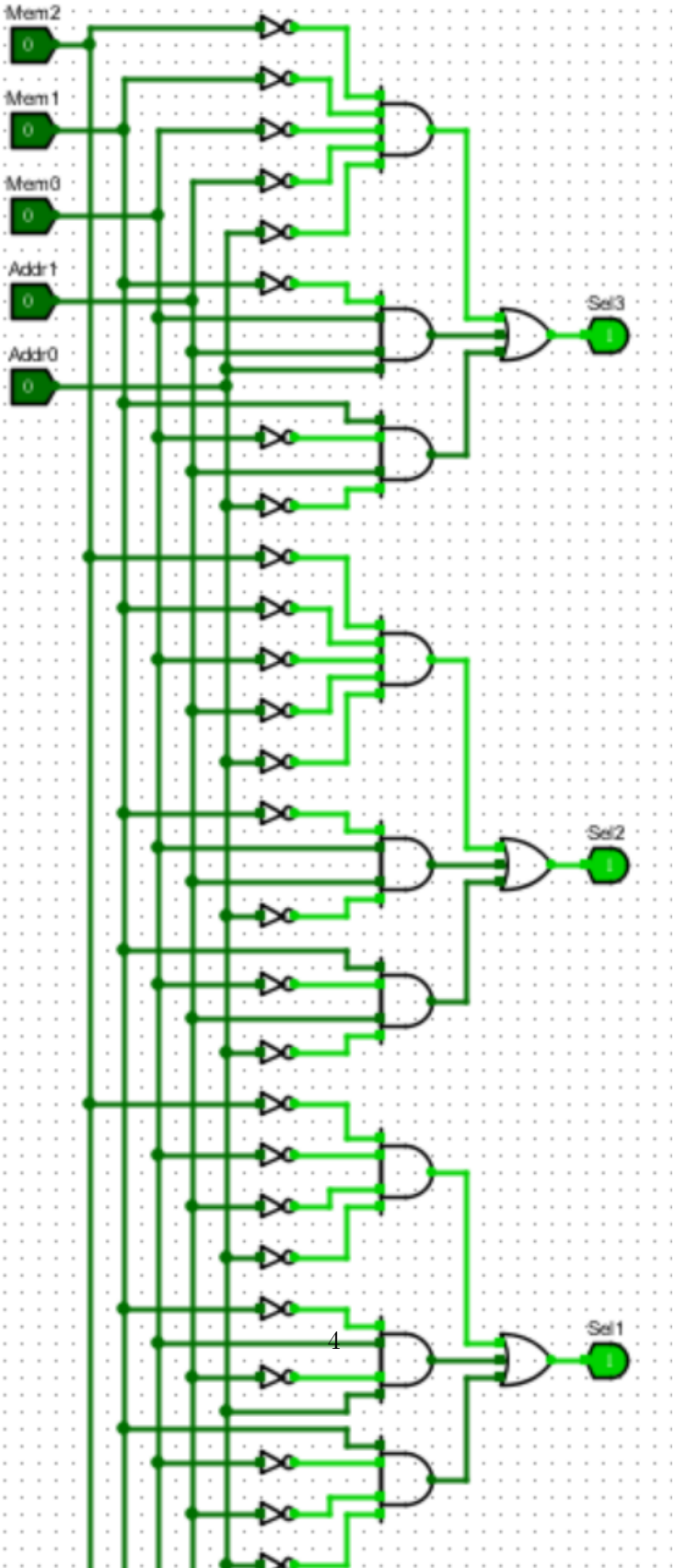
### 2.1 实验整体方案设计

这道题目需要实现一个支持 RV32I 指令集中 lw, sw, lb, lbu, lh, lhu, sb, sh 等指令的按字节寻址的 32 位数据存储器模块，包括字节、半字、字的读写操作控制、对齐检测、扩展处理等。

这个 256KB 数据存储器的基本逻辑为：接收一个地址（Addr）和操作码（MemOp），支持不同的访问类型（按字节、半字、字）进行读写操作，根据地址和访问类型自动选择正确的 RAM 芯片进行操作，对于读取时，支持符号扩展与零扩展，对于写入时，将正确的字节写入对应的 RAM 芯片，如果地址不对齐或 MemOp 编码非法，输出错误标志 AlignErr=1。

片选信号 SEL0-SEL3 的含义为：根据 MemOp[2:0] 与 Addr[1:0] 共同决定哪个字节的 RAM 被选中，用片选信号 SEL0 SEL3 控制哪些 RAM 写入。

## 2.2 实验原理图及电路图



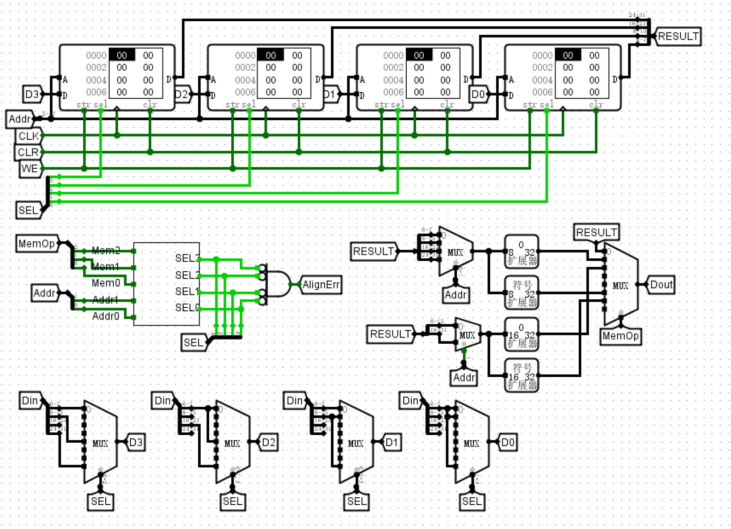



图 6: 5.2 电路图 DataRam

## 2.3 实验数据仿真测试图

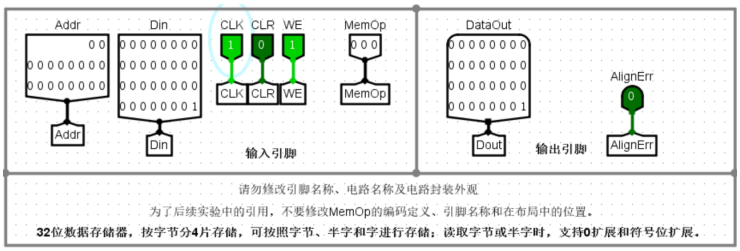



图 7: 5.2 仿真测试 1

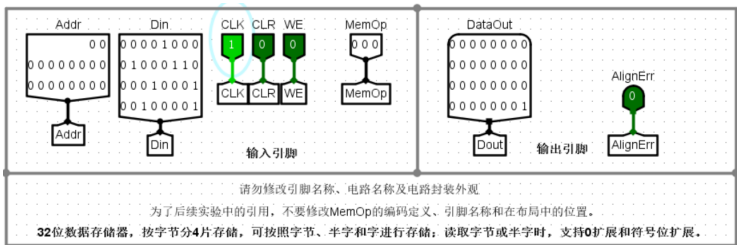



图 8: 5.2 仿真测试 2

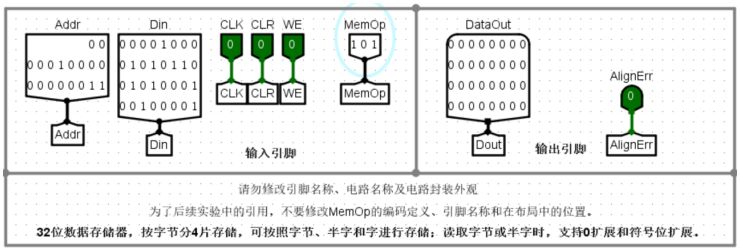



图 9: 5.2 仿真测试 3

### 2.4 错误现象及分析

本题实验手册的叙述并不是十分清楚（，所以需要自己查找一些资料辅助完成这道题目。

## 3 取指令部件实验

### 3.1 实验整体方案设计

这道题要实现一个取指令的部件，主要功能是根据当前的 PC 值，从指令存储器中读取下一条指令，并根据控制信号更新 PC 值。

PC 计算逻辑（下地址逻辑）为：使用两个多路选择器和一个加法器来计算新 PC 值：

多路选择器 A（受 NPCASrc 控制）：0 → 选择 PC 当前值。1 → 选择 BusA（rs1 的值）

多路选择器 B（受 NPCBSrc 控制）：0 → 选择常量 4（用于顺序执行）1 → 选择 Imm（跳转/分支）

加法器：把 MuxA 和 MuxB 的结果相加，得出新 PC 值

总体设计思路为：PC 作为地址源，通过控制信号（NPCASrc/NPCBSrc）选择跳转/顺序执行逻辑。加法器负责计算下一条指令地址。指令 ROM 读取指令（注意地址取 PC[17:2]）。Reset 时初始化 PC，写入 Initial Address 将整个结构封装为 IFU 子电路，主电路中可以测试多种输入组合，观察 PC 和 IR 变化

## 3.2 实验原理图及电路图

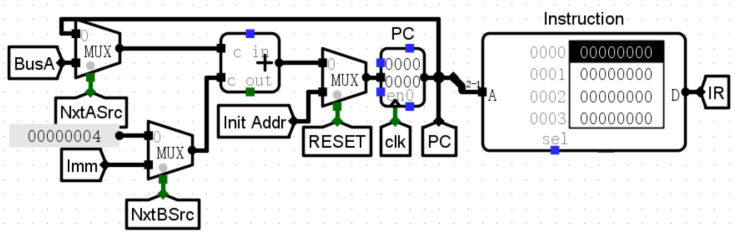



图 10: 5.3 电路图

## 3.3 实验数据仿真测试图

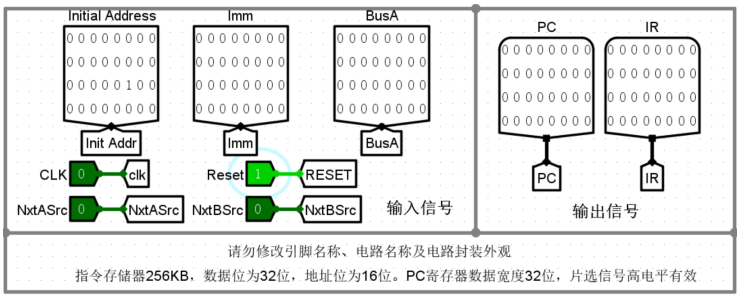



图 11: 5.3 仿真测试 1

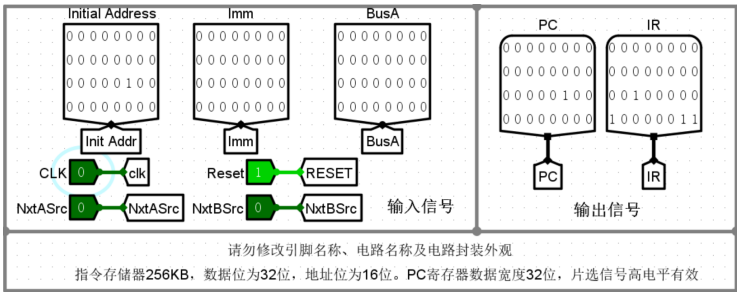



图 12: 5.3 仿真测试 2

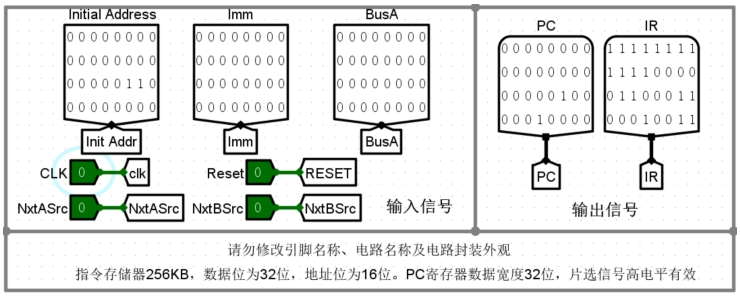



图 13: 5.3 仿真测试 3

### 3.4 错误现象及分析

这道题需要注意 IR 需要取自 ROM 中，ROM 的初始化文件题目中已经提供

## 4 取操作数部件 IDU 实验

### 4.1 实验整体方案设计

本题要求设计取操作数部件（IDU），根据输入的 32 位指令（IR），解析出不同字段（如 rs1、rs2 等），调用你设计的立即数扩展器（根据 ExtOp 控制信号选择不同的指令格式立即数并扩展为 32 位），从寄存器堆 RegFile 中读取对应寄存器的数据（DataA 和 DataB），根据 ALUASrc 和 ALUBSrc 选择 ALU 输入的操作数（来自寄存器、立即数、PC 或常数 4），并输出给后续的数据通路使用。

对于 ALU 输入选择器（ALUASrc/ALUBSrc），这些控制信号用于选择传给 ALU 的两个输入

ALUAsrc（1 位）：

- 0: 选择 BusA
- 1: 选择 PC（通常是 auipc, jalr, jal）

ALUBsrc（2 位）：

- 00: 选择 BusB（R 型用）
- 01: 常量 4（PC+4 的情况）

- 10: Imm（I/S/B/J 型用）
- 11: 暂不常用或保留

## 4.2 实验原理图及电路图

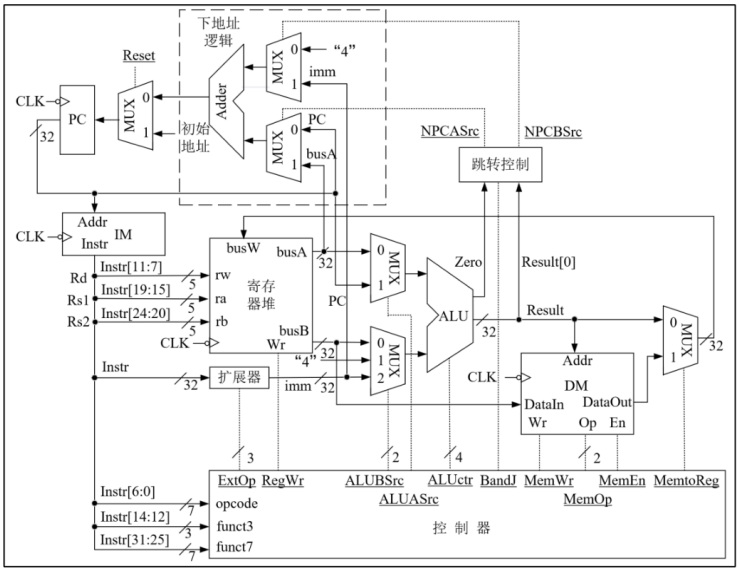



图 14: 5.4 原理图

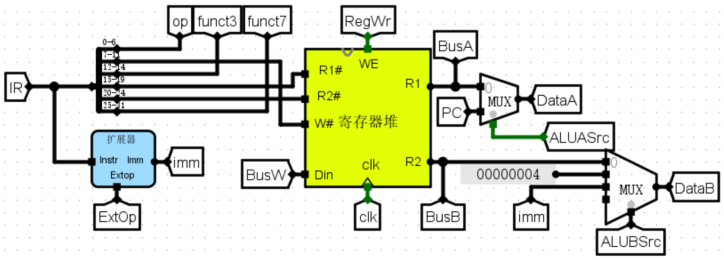



图 15: 5.4 电路图 IDU

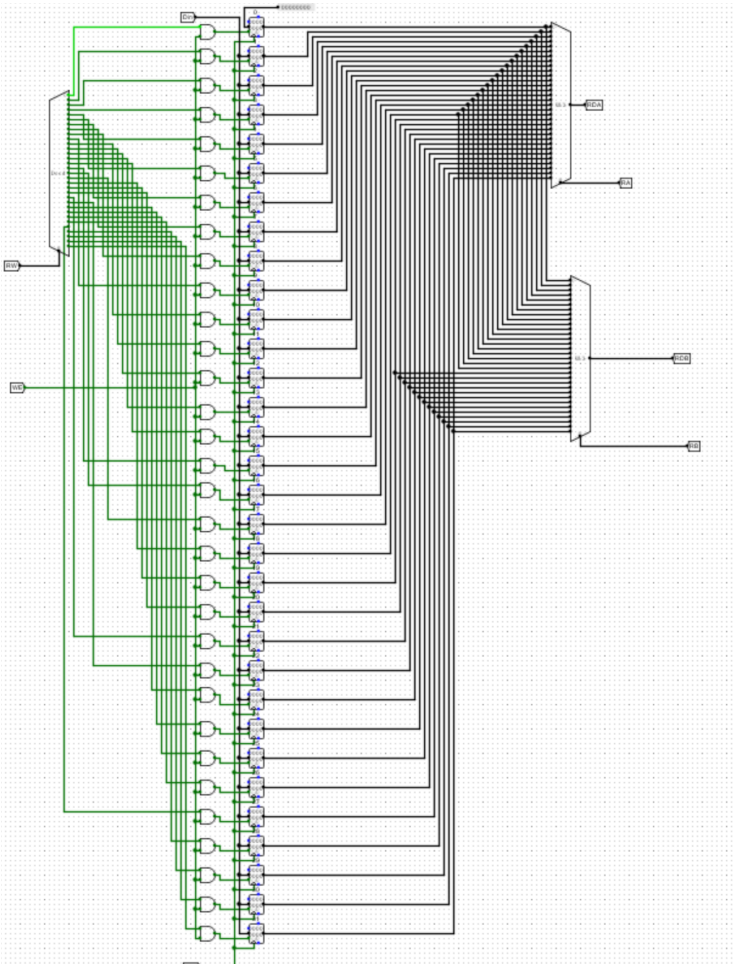


图 16: 5.4 电路图 RegFile

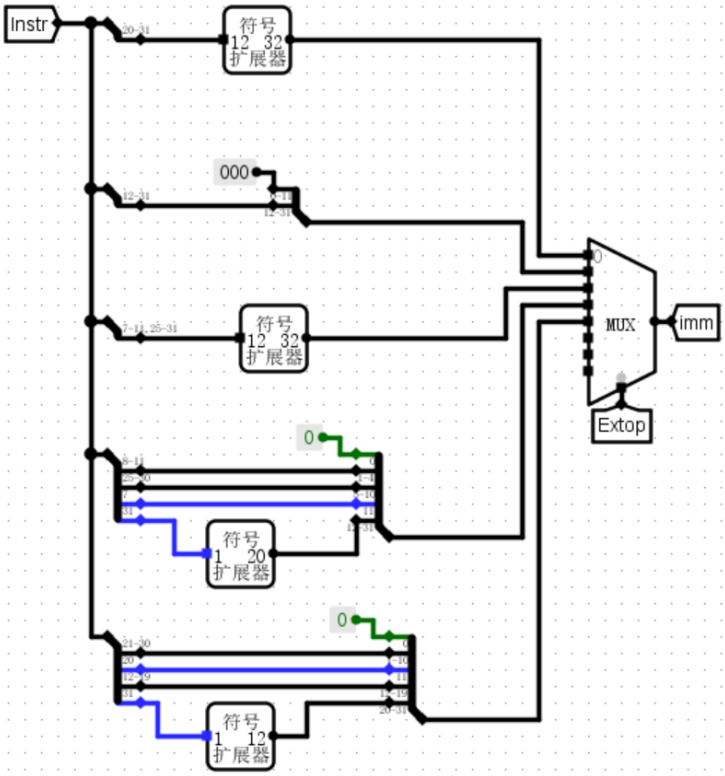



图 17: 5.4 电路图立即数拓展器

## 4.3 实验数据仿真测试图

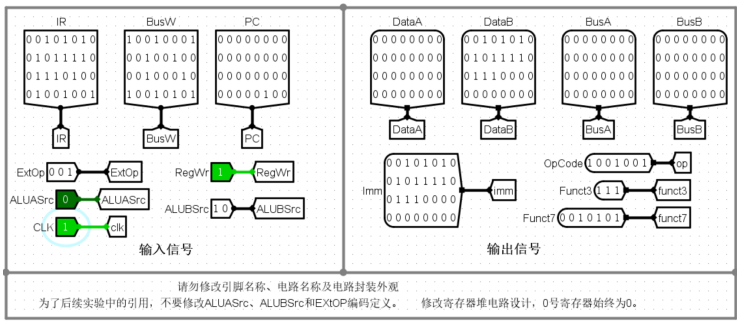



图 18: 5.4 仿真测试 1

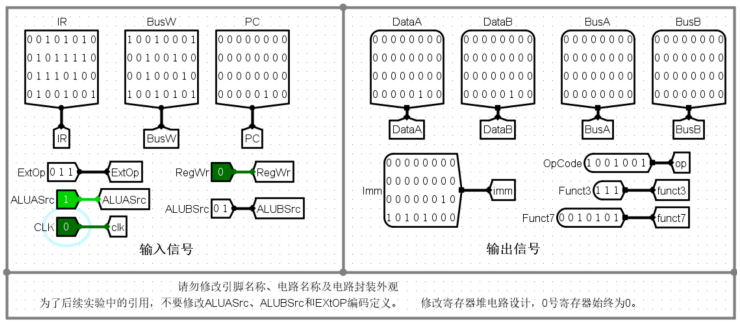



图 19: 5.4 仿真测试 2

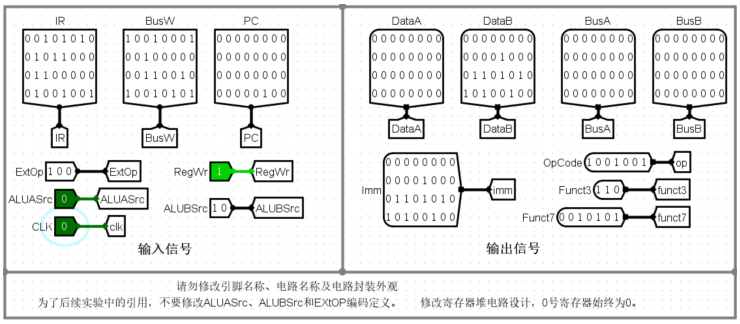



图 20: 5.4 仿真测试 3

### 4.4 错误现象及分析

需要注意 RegFile 的初始地址应赋值为 0x0000000 而不是直接复制之前的实现

## 5 数据通路实验

### 5.1 实验整体方案设计

这道题目需要我们实现：

- 根据 ALU 输出的 Zero、Result[0] 以及控制信号 BandJ，输出两个跳转控制信号 NPCASrc 和 NPCBSrc，用于决定下一条指令的地址是跳转目标还是顺序地址。

- 把原来 lab5.3 中的 IFU 电路提取进当前工程，不再内嵌 ROM，ROM 作为独立模块，IFU 仅负责输出 PC。
- 使用已完成的子电路模块（IFU、IDU、ALU、DataRAM、BandJ），完成一个 RV32I 指令集的单周期数据通路。各模块之间连线实现信息流动。
- 向 ROM 写入提供的 21 条机器码指令，设置控制信号，观察输出信号值和寄存器堆、数据存储器的变化，验证电路是否正确运行。

| 序号 | **PC=0 指令（机器码）** | **RV32 汇编语句** | **输出信号值（32 位，16 进制）** |
|---|---|---|---|
| 1 | 008000ef | jal x1, 0x8 | x1: 0x00000004 |
| 2 | ffdff06f | jal x0, 0xfffffffc | x0: |
| 3 | 00100013 | addi x0, x0, 0x1 | x0: 0x00000000 |
| 4 | deadc2b7 | lui x5, 0x000deadc | x5: 0xdeadc000 |
| 5 | eef28293 | addi x5, x5, 0xffffeef | x5: 0xdeadbeef |
| 6 | 0052d333 | sra x6, x5, x5 | x6: 0x0001bd5b |
| 7 | 00135313 | srli x6, x6, 1 | x6: 0x0000dead |
| 8 | 4052d3b3 | sra x7, x5, x5 | x7: 0xffffbd5b |
| 9 | 00529e33 | sll x28, x5, x5 | x28: 0xdf778000 |
| 10 | 001e1e13 | slli x28, x28, 1 | x28: 0xbeef0000 |
| 11 | 006e0eb3 | add x29, x28, x6 | x29: 0xbeefdead |
| 12 | 01000513 | addi x10, x0, 0x10 | x10: 0x00000010 |
| 13 | 005500a3 | sb x5, 1(x10) | M[0x10]: 0x0000ef00 |
| 14 | 00551223 | sh x5, 4(x10) | M[0x14]: 0x0000beef |
| 15 | 00551323 | sh x5, 6(x10) | M[0x14]: 0xbeefbeef |
| 16 | 00552423 | sw x5, 8(x10) | M[0x18]: 0xdeadbeef |
| 17 | 00150483 | lb x9, 1(x10) | x9: 0xffffffef |
| 18 | 00154903 | lbu x18, 1(x10) | x18: 0x000000ef |
| 19 | 00451983 | lh x19, 4(x10) | x19: 0xffffbeef |
| 20 | 00655a03 | lhu x20, 6(x10) | x20: 0xbeefbeef |
| 21 | 00452a83 | lw x21, 4(x10) | x21: 0xbeefbeef |
| 22 | 00a55b03 | lhu x22, 10(x10) | x22: 0x0000dead |
| 23 | ffff0bb7 | lui x23, 0x000ffff0 | x23: 0xffff0000 |
| 24 | 017afc33 | and x24, x21, x23 | x24: 0xbeef0000 |
| 25 | 016c6cb3 | or x25, x24, x22 | x25: 0xbeefdead |
| 26 | fb9e92e3 | bne x29, x25, 0xffffffa4 | PC: 0x00000068 |
| 27 | 00000073 | ecall | PC: 0x0000006c |

## 5.2 实验原理图及电路图

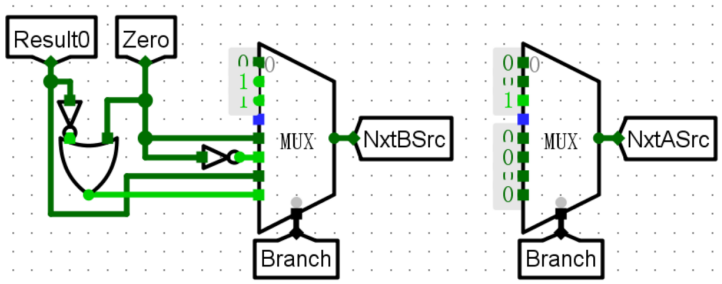



图 21: 5.5 电路图 BandJ

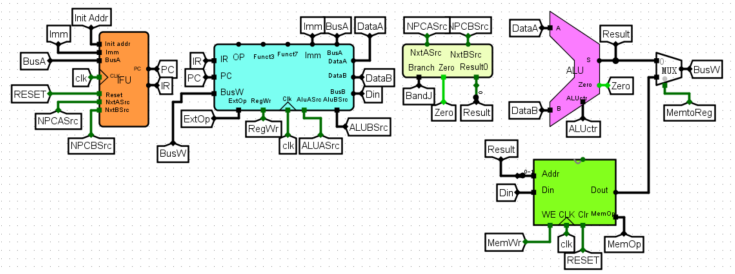



图 22: 5.5 电路图 main

## 5.3 实验数据仿真测试图

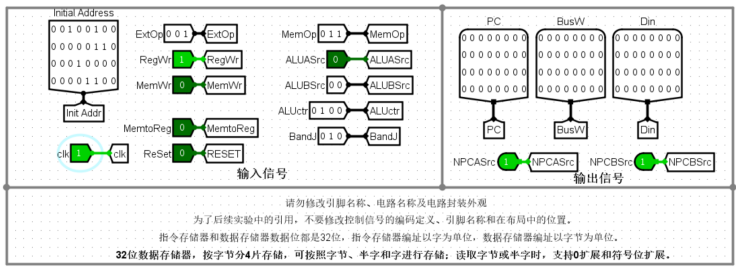



图 23: 5.5 仿真测试 1

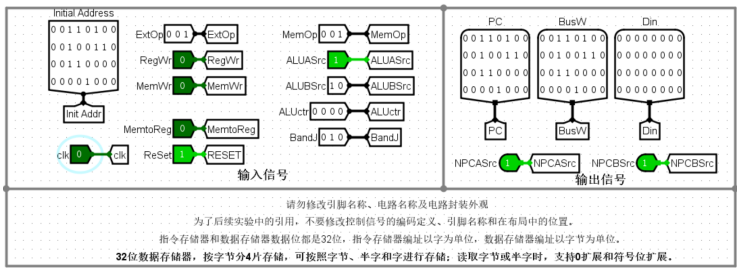



图 24: 5.5 仿真测试 2

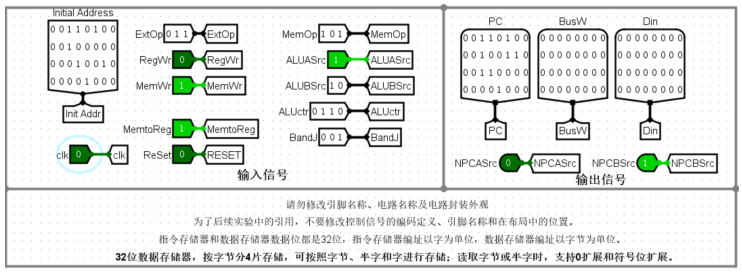



图 25: 5.5 仿真测试 3

### 5.4 错误现象及分析

这道题我的实现使用了特别多的引脚及隧道，需要注意是否符合预期

## 6 思考题

### 6.1 如何利用 ROM 实验实现滚动显示的功能，在 3 个 LED 点阵矩阵中，左右滚动显示 5 个 ASCII 字符，如 “NJUCS”。

首先将每个 ASCII 字符的 8×8 点阵存储在 ROM 中，每个字符使用 8 字节表示，每字节代表一列（或一行）的点阵信息。将 “NJ UCS” 五个字符的点阵依次写入 ROM 中，连续存储。

为了在 3 个 LED 点阵上滚动显示 5 个字符（共 $5\times8=40$ 列），需设置一个缓冲区用于临时存储当前要显示的 24 列数据（3 个点阵）。

设置一个列偏移（Column Offset）寄存器，其值每隔一定时钟周期自增 1，用于实现数据的 “向左滚动”。例如，当偏移为 $i$ 时，显示的是 ROM

中从第 $i$ 列开始的 24 列数据。

利用 8 位计数器扫描行号（行刷新），在每一行时选中对应的 LED 行，并加载当前列的对应位。通过高速刷新使肉眼看到完整字符。

最后控制行刷新和列偏移的时钟周期有较大的差距。

## 6.2 分析说明如果 PC 寄存器、寄存器堆和数据存储器写入数据的时钟信号触发边沿不一致，则对程序执行结果有什么影响？

若 PC 在上升沿更新，而寄存器堆在下降沿写入，那么写寄存器的操作将在下一条指令读取寄存器之后才发生，导致读到的仍然是旧值，形成数据冒险（Data Hazard）。

类似地，如果 RAM 的写入与寄存器写回发生在不同的边沿，也可能导致以下问题：写数据尚未准备好，数据存储器已被触发写入，写入了错误的值，或者写地址尚未更新，旧地址被误写。

整个 CPU 数据通路是按照一个统一节奏工作的。一旦某些模块触发不同步，时序分析与逻辑验证将非常复杂，容易出现隐蔽性 bug。

## 6.3 在 CPU 启动执行后，如何实现在当前程序结束后，CPU 不再继续往下执行？

我们可以利用 RISC-V 中的 **环境调用指令** `ecall`，结合 CPU 控制器的设计，使得当检测到 `ecall` 指令时，输出一个 **Halt 信号**，使 CPU 停止执行。

当 `Opcode = 0x73` 且 `Funct3 = 0x0` 且 `Funct12 = 0x000` 时，输出 `Halt = 1`

若 `Halt = 1`，则停止主时钟信号传递，或通过使 PC 保持不变，停止执行新指令，或在 Logisim 中用 **断点逻辑电路**控制 Cloc k 不再前进。