卡通人物

AI 生成的内容可能不正确。

徽标

AI 生成的内容可能不正确。

计算机组成原理课程设计

姓 名

班 级

学 号

提交日期

目录

[一、实验目的 1](#_Toc8348)

[二、实验内容 1](#_Toc15759)

[三、实验原理（16分） 1](#_Toc3421)

[3.1、指令系统及分析（4分） 1](#_Toc16239)

[3.2、指令框图及分析（4分） 1](#_Toc17524)

[3.3、指令系统对应微程序二进制代码及分析（4分） 1](#_Toc9380)

[3.4、机器程序及分析（4分） 1](#_Toc13197)

[四、实验步骤（4分） 1](#_Toc16603)

[4.1、微程序写入及校验（2分） 1](#_Toc20867)

[4.2、机器程序写入及校验（2分） 1](#_Toc453)

[五、实验结果及分析（16分） 1](#_Toc26243)

[5.1、演示程序一（8分） 1](#_Toc17196)

[5.2、演示程序二（8分） 2](#_Toc2614)

[六、实验问题及思考（4分） 2](#_Toc25850)

[七、实验验收答辩环节问题和解答（20分） 3](#_Toc12346)

# 一、实验目的

1. 通过实验分析CPU模型机结构，了解计算机工作原理。
2. 掌握计算机微程序控制器的控制方法，掌握计算机指令执行过程。
3. CPU模型计算机部件构架、微程序控制器中微程序代码功能实现、微程序流程、指令系统定义和简单模型计算机汇编语言程序进行验证。
4. 以实验3.2为基础，选择合适的指令完成一个基本任务，完成的功能不限定，由学生自行设计完成。目的是提高学生对计算机机器指令的理解，锻炼学生自己动手设计模型计算机机器指令的能力。

# 二、实验内容

1. 设计并实现一套完整的指令系统；
2. 设计并实现完整的计算机（采用上述指令系统）；
3. 利用该计算机实现基于复杂模型机的数组查找功能，在预设数组[01H, 03H, 05H, 07H]中查找用户输入的数值，找到则输出01H，未找到则输出00H。

# 三、实验原理（16分）

## 3.1、指令系统及分析（4分）

### (1) 指令格式的释义

根据实验3.2中的指令进行了扩展，新增了CMP（比较）指令来实现数组查找功能。在设计过程中，充分考虑了指令系统的完整性和功能需求。具体描述如表1和表2所示。

其中指令里的RS、RD，00时为R1，01时为R2，10时为R3，11时为R4。

|  |  |  |  |
| --- | --- | --- | --- |
| 寻址模式M | 有效地址E | 说明 | 应用场景 |
| 00 | E = D | 直接寻址 | 访问固定地址的数据 |
| 01 | E = (D) | 间接寻址 | 实现指针访问，增加寻址灵活性 |
| 10 | E = (RI) + D | RI变址寻址 | **数组访问**，本实验核心寻址方式 |
| 11 | E = (PC) + D | 相对寻址 | 实现位置无关代码，便于程序重定位 |

表1 寻址方式及应用分析

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 助记符号 | 指令格式 | 指令功能 | 功能类别 | 在本实验中的作用 |
| MOV RD, RS | 0100 RS RD | RS -> RD | 数据传送 | 寄存器间数据复制 |
| ADD RD, RS | 0000 RS RD | RD + RS ->RD | 算术运算 | 地址计算辅助 |
| SUB RD, RS | 1000 RS RD | RD - RS ->RD | 算术运算 | 数值减法运算 |
| AND RD, RS | 0001 RS RD | RD ^ RS ->RD | 逻辑运算 | 位操作处理 |
| OR RD, RS | 1001 RS RD | RD v RS ->RD | 逻辑运算 | 标志位设置 |
| RR RD, RS | 1010 RS RD | RS右环移 -> RD | 移位运算 | 数据位移处理 |
| **CMP RD, RS** | **1011 RS RD** | **RD - RS（比较）** | **比较运算** | **核心查找比较功能** |
| INC RD | 0111 \*\* RD | RD + 1 -> RD | 算术运算 | 循环计数器递增 |
| LAD M D, RD | 1100 M D RD | E -> RD | 数据加载 | 数组元素读取 |
| STA M D, RD | 1101 M D RD | RD -> E | 数据存储 | 结果保存 |
| JMP M D | 1110 M \*\* RD | E -> PC | 控制转移 | 程序流程控制 |
| BZC M D | 1111 M \*\* RD | E -> PC（当FC或FZ=1） | 条件转移 | 条件分支判断 |
| IN RD, P | 0010 \*\* RD P | [P] -> RD | 输入输出 | 用户输入获取 |
| OUT P, RS | 0011 RS \*\* P | RS -> [P] | 输入输出 | 结果输出显示 |
| LDI RD, D | 0110 \*\* RD D | D -> RD | 立即数加载 | 常数初始化 |
| HALT | 0101 \*\* \*\* | 停机 | 系统控制 | 程序结束 |

表2 指令描述及功能分析

### (2) 新增比较指令CMP的详细设计

#### 设计目标与需求分析

在数组查找算法中，比较操作是核心功能。原有指令系统中的SUB指令虽然能实现减法运算，但会改变目标寄存器的值，不适合用于纯比较操作。因此，设计专门的CMP指令具有以下优势：

1. **功能专一性**：专门用于比较，不改变操作数
2. **语义清晰性**：代码可读性更强，意图明确
3. **标志位设置**：专门针对比较结果设置相关标志位
4. **性能优化**：避免额外的数据恢复操作

#### CMP指令的实现原理

CMP指令的核心思想是复用SUB指令的运算逻辑，但在微程序层面做出关键修改：

**传统SUB指令执行过程**： 1. 将RD内容送入运算器A端 2. 将RS内容送入运算器B端  
3. 执行A-B运算 4. **将运算结果写回RD寄存器** 5. 根据运算结果设置标志位

**CMP指令执行过程**： 1. 将RD内容送入运算器A端 2. 将RS内容送入运算器B端 3. 执行A-B运算 4. **跳过结果写回步骤** 5. 仅根据运算结果设置标志位

**CMP指令微程序流程图**：

CMP RD, RS (1011)  
 |  
 ┌────────────────┐  
 │ 3BH: RD -> A │  
 │ (准备被比较数) │  
 └────────────────┘  
 |  
 ┌────────────────┐  
 │ 19H: RS -> B │  
 │ (准备比较数) │  
 └────────────────┘  
 |  
 ┌───────────────────────────┐  
 │ 1AH: A - B, set flags │  
 │ (执行比较,不写回结果) │  
 └───────────────────────────┘  
 |  
 ┌────────────────┐  
 │ 01H: 取指周期 │  
 │ (下一条指令) │  
 └────────────────┘

#### 微程序设计细节

CMP指令占用指令码1011，在微程序中对应地址19H-1AH：

地址19H: 00 24 1A - 数据准备阶段  
 - 高5位(00000): WR=0, RD=0, IOM=0, 不进行存储器或IO操作  
 - S3-S0(0000): 不进行ALU运算，仅数据传输  
 - A字段(010): 打开RD到A端的通路  
 - B字段(010): 打开RS到B端的通路   
 - C字段(000): 不使用P信号  
 - UA5-UA0(011010): 下一地址为1AH  
  
地址1AH: 05 80 01 - 比较运算阶段  
 - 高5位(00000): 无额外控制  
 - S3-S0(1011): ALU执行A-B运算  
 - A字段(000): 关闭写回通路  
 - B字段(000): 关闭写回通路  
 - C字段(000): 不使用P信号  
 - UA5-UA0(000001): 返回地址01H

### (3) 指令系统完整性分析

本实验设计的指令系统涵盖了计算机系统的基本指令类型：

**数据传送类指令**： - MOV: 寄存器间传送 - LAD: 内存到寄存器加载 - STA: 寄存器到内存存储 - LDI: 立即数加载

**算术运算类指令**： - ADD: 加法运算 - SUB: 减法运算  
- INC: 自增运算 - CMP: 比较运算（不改变操作数）

**逻辑运算类指令**： - AND: 按位与运算 - OR: 按位或运算 - RR: 右循环移位

**程序控制类指令**： - JMP: 无条件跳转 - BZC: 条件跳转 - HALT: 程序停止

**输入输出类指令**： - IN: 数据输入 - OUT: 数据输出

该指令系统基本满足了通用计算的需求，特别是针对数组操作和查找算法进行了优化设计。

## 3.2、指令框图及分析（4分）

### 程序总体设计思路

本程序采用模块化设计思想，将数组查找功能分解为以下几个关键模块：

1. **初始化模块**：设置初始参数和工作环境
2. **输入模块**：获取用户查找目标
3. **查找核心模块**：实现线性查找算法
4. **比较判断模块**：执行元素比较和结果判断
5. **输出模块**：返回查找结果
6. **数据存储模块**：管理数组数据

### 详细程序流程图

程序开始  
 |  
 ┌─────────────┐  
 │ 初始化模块 │  
 │ R2 ← 0 │   
 │ R0 ← 4 │  
 └─────────────┘  
 |  
 ┌─────────────┐  
 │ 输入模块 │  
 │ IN R1, 00H │  
 │ (获取查找值) │  
 └─────────────┘  
 |  
 ┌─────────────────┐  
 │ 查找循环开始 │ ←─────────────────────────────┐  
 │ LOOKUP\_LOOP │ │  
 └─────────────────┘ │  
 | │  
 ┌───────────────┐ │  
 │ 边界检查模块 │ │  
 │ CMP R2, R0 │ │  
 |(索引与长度比较)│ │  
 └───────────────┘ │  
 | │  
 ┌─────────┐ │  
 │R2 == R0?│ ─Yes─→ ┌─────────────┐ │  
 └─────────┘ │ 未找到处理 │ │  
 |No │ R0 ← 00H │ │  
 ┌───────────────┐ │ 输出并结束 │ │  
 |数组访问模块 │ └─────────────┘ │  
 │LAD R3,[R2],60H│ │  
 |(读取当前元素) │ │  
 └───────────────┘ │  
 | │  
 ┌───────────────┐ │  
 │ 元素比较模块 │ │  
 │ CMP R3, R1 │ │  
 |(元素与目标比较)│ │  
 └───────────────┘ │  
 | │  
 ┌─────────┐ │  
 │R3 == R1?│ ─Yes─→ ┌─────────────┐ │  
 └─────────┘ │ 找到处理 │ │  
 |No │ R0 ← 01H │ │  
 ┌───────────────┐ │ 输出并结束 │ │  
 │ 索引递增模块 │ └─────────────┘ │  
 │ INC R2 │ │  
 |(准备下一次循环)│ │  
 └───────────────┘ │  
 | │  
 └───────────────────────────────────────┘

图1 详细程序流程图

### 算法复杂度分析

**时间复杂度分析**： - 最好情况：O(1) - 第一个元素即为目标 - 最坏情况：O(n) - 目标不存在或在最后位置 - 平均情况：O(n/2) - 目标在数组中间位置

**空间复杂度分析**： - 辅助空间：O(1) - 仅使用固定数量的寄存器 - 数据空间：O(n) - 数组存储空间

**指令执行次数分析**： 对于长度为n的数组，在最坏情况下： - 初始化指令：6条 - 每次循环：约8条指令 - 总指令数：6 + 8n + 4 = 8n + 10

### 算法优化策略讨论

**当前实现的优势**： 1. 实现简单，易于理解和调试 2. 内存访问模式规律，适合缓存优化 3. 对于小规模数组效率较高

**可能的优化方向**： 1. **哨兵优化**：在数组末尾添加目标值，减少边界检查 2. **展开循环**：减少循环控制开销 3. **并行比较**：同时比较多个元素（硬件支持情况下）

## 3.3、指令系统对应微程序二进制代码及分析（4分）

### (1) 完整微程序二进制代码表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 地址 | 十六进制格式 | 高五位 | S3~S0 | A字段 | B字段 | C字段 | UA5~UA0 | 功能描述 |
| 00 | 00 00 01 | 00000 | 0000 | 000 | 000 | 000 | 000001 | 复位初始化，无操作 |
| 01 | 00 6D 43 | 00000 | 0000 | 110 | 110 | 101 | 000011 | PC→AR，PC+1，准备取指 |
| 03 | 10 70 70 | 00010 | 0000 | 111 | 000 | 001 | 110000 | MEM→IR，使用P<1>译码 |
| 04 | 00 24 05 | 00000 | 0000 | 010 | 010 | 000 | 000101 | RS→B，准备ADD运算 |
| 05 | 04 B2 01 | 00000 | 1001 | 011 | 001 | 000 | 000001 | A+B→RD，执行加法 |
| 06 | 00 24 07 | 00000 | 0000 | 010 | 010 | 000 | 000111 | RS→B，准备AND运算 |
| 07 | 01 32 01 | 00000 | 0010 | 011 | 001 | 000 | 000001 | A∧B→RD，执行与运算 |
| 08 | 10 60 09 | 00010 | 0000 | 110 | 000 | 000 | 001001 | MEM→AR，准备IO访问 |
| 09 | 18 30 01 | 00011 | 0000 | 011 | 000 | 000 | 000001 | IO→RD，输入操作 |
| 0A | 10 60 10 | 00010 | 0000 | 110 | 000 | 000 | 010000 | MEM→AR，准备输出 |
| 0B | 00 00 01 | 00000 | 0000 | 000 | 000 | 000 | 000001 | 空操作 |
| 0C | 10 30 01 | 00010 | 0000 | 011 | 000 | 000 | 000001 | MEM→RD，数据加载 |
| 0D | 20 06 01 | 00100 | 0000 | 000 | 011 | 000 | 000001 | RD→MEM，数据存储 |
| 0E | 00 53 41 | 00000 | 0000 | 101 | 001 | 101 | 000001 | A→PC，跳转执行 |
| 0F | 00 00 CB | 00000 | 0000 | 000 | 000 | 011 | 001011 | 条件判断，使用P<3> |
| 10 | 28 04 01 | 00101 | 0000 | 000 | 010 | 000 | 000001 | RS→IO，输出操作 |
| 11 | 10 30 01 | 00010 | 0000 | 011 | 000 | 000 | 000001 | MEM→RD，数据读取 |
| 12 | 06 B2 01 | 00000 | 1101 | 011 | 001 | 000 | 000001 | A+1→RD，自增操作 |
| 13 | 00 24 14 | 00000 | 0000 | 010 | 010 | 000 | 010100 | RS→B，准备SUB运算 |
| 14 | 05 B2 01 | 00000 | 1011 | 011 | 001 | 000 | 000001 | A-B→RD，减法运算 |
| 15 | 00 24 16 | 00000 | 0000 | 010 | 010 | 000 | 010110 | RS→B，准备OR运算 |
| 16 | 01 B2 01 | 00000 | 0011 | 011 | 001 | 000 | 000001 | A∨B→RD，或运算 |
| 17 | 00 24 18 | 00000 | 0000 | 010 | 010 | 000 | 011000 | RS→B，准备右移 |
| 18 | 02 B2 01 | 00000 | 0101 | 011 | 001 | 000 | 000001 | A右环移→RD |
| **19** | **00 24 1A** | **00000** | **0000** | **010** | **010** | **000** | **011010** | **CMP第一周期：RS→B** |
| **1A** | **05 80 01** | **00000** | **1011** | **000** | **000** | **000** | **000001** | **CMP第二周期：A-B比较** |
| 1B | 00 53 41 | 00000 | 0000 | 101 | 001 | 101 | 000001 | A→PC，跳转返回 |
| 1C | 10 10 1D | 00010 | 0000 | 001 | 000 | 000 | 011101 | MEM→A，间接寻址第一步 |
| 1D | 10 60 8C | 00010 | 0000 | 110 | 000 | 010 | 001100 | MEM→AR，使用P<2> |
| 1E | 10 60 1F | 00010 | 0000 | 110 | 000 | 000 | 011111 | MEM→AR，准备间接跳转 |
| 1F | 10 10 20 | 00010 | 0000 | 001 | 000 | 000 | 100000 | MEM→A，读取跳转地址 |
| 20 | 10 60 8C | 00010 | 0000 | 110 | 000 | 010 | 001100 | MEM→AR，使用P<2> |
| 28 | 10 10 29 | 00010 | 0000 | 001 | 000 | 000 | 101001 | MEM→A，变址寻址开始 |
| 29 | 00 28 2A | 00000 | 0000 | 010 | 100 | 000 | 101010 | RI→B，读取索引寄存器 |
| 2A | 04 E2 2B | 00000 | 1001 | 110 | 001 | 000 | 101011 | A+B→AR，计算有效地址 |
| 2B | 04 92 8C | 00000 | 1001 | 001 | 001 | 010 | 001100 | A+B→A，使用P<2> |
| 2C | 10 10 2D | 00010 | 0000 | 001 | 000 | 000 | 101101 | MEM→A，相对寻址开始 |
| 2D | 00 2C 2E | 00000 | 0000 | 010 | 110 | 000 | 101110 | PC→B, 读取当前PC |
| 2E | 04 E2 2F | 00000 | 1001 | 110 | 001 | 000 | 101111 | A+B→AR，计算相对地址 |
| 2F | 04 92 8C | 00000 | 1001 | 001 | 001 | 010 | 001100 | A+B→A，使用P<2> |
| 30 | 00 16 04 | 00000 | 0000 | 001 | 011 | 000 | 000100 | RD→A，MOV指令 |
| 31 | 00 16 06 | 00000 | 0000 | 001 | 011 | 000 | 000110 | RD→A，跳转到AND |
| 32 | 00 6D 48 | 00000 | 0000 | 110 | 110 | 101 | 001000 | PC→AR，PC+1，双字长第一步 |
| 33 | 00 6D 4A | 00000 | 0000 | 110 | 110 | 101 | 001010 | PC→AR，PC+1，双字长第二步 |
| 34 | 00 34 01 | 00000 | 0000 | 011 | 010 | 000 | 000001 | RS→RD，数据传送 |
| 35 | 00 00 35 | 00000 | 0000 | 000 | 000 | 000 | 110101 | 停机指令，循环等待 |
| 36 | 00 6D 51 | 00000 | 0000 | 110 | 110 | 101 | 010001 | PC→AR, PC+1，LDI指令 |
| 37 | 00 16 12 | 00000 | 0000 | 001 | 011 | 000 | 010010 | RD→A，跳转到INC |
| 38 | 00 16 13 | 00000 | 0000 | 001 | 011 | 000 | 010011 | RD→A，跳转到SUB |
| 39 | 00 16 15 | 00000 | 0000 | 001 | 011 | 000 | 010101 | RD→A，跳转到OR |
| 3A | 00 16 17 | 00000 | 0000 | 001 | 011 | 000 | 010111 | RD→A，跳转到RR |
| **3B** | **00 16 19** | **00000** | **0000** | **001** | **011** | **000** | **011001** | **RD→A，跳转到CMP指令** |
| 3C | 00 6D 5C | 00000 | 0000 | 110 | 110 | 101 | 011100 | PC→AR，PC+1，LAD指令 |
| 3D | 00 6D 5E | 00000 | 0000 | 110 | 110 | 101 | 011110 | PC→AR，PC+1，STA指令 |
| 3E | 00 6D 68 | 00000 | 0000 | 110 | 110 | 101 | 101000 | PC→AR，PC+1，JMP指令 |
| 3F | 00 6D 6C | 00000 | 0000 | 110 | 110 | 101 | 101100 | PC→AR，PC+1，BZC指令 |

表3 完整微程序二进制代码表

### 微程序地址分配策略分析

**指令译码跳转表**： - 地址00H：复位入口 - 地址01H：取指周期入口 - 地址03H：指令译码中心，使用P<1>信号分发到具体指令 - 地址04H-05H：ADD指令微程序 - 地址06H-07H：AND指令微程序 - 地址08H-09H：IN指令微程序 - 地址0AH-0BH：OUT指令微程序 - 地址0CH-0DH：LAD/STA指令微程序 - 地址0EH-0FH：JMP/BZC指令微程序 - 地址10H-11H：OUT/MEM指令微程序 - 地址12H：INC指令微程序 - 地址13H-14H：SUB指令微程序 - 地址15H-16H：OR指令微程序 - 地址17H-18H：RR指令微程序 - **地址19H-1AH：CMP指令微程序（新增，19H→1AH→01H的完整执行链）** - 地址1BH：通用跳转返回 - 地址1CH-20H：间接寻址处理 - 地址28H-2FH：变址寻址和相对寻址处理 - 地址30H-3FH：指令分发和特殊处理（3BH：CMP指令入口，跳转到19H）

### (2) 新增CMP指令的深度技术分析

#### 微指令设计的技术考量

**地址分配策略**： 选择19H-1AH作为CMP指令的微程序地址有以下考虑： 1. **连续性**：与其他算术指令地址连续，便于管理 2. **访问效率**：在微程序存储器中位置适中，访问延迟较小 3. **扩展性**：预留了足够的地址空间用于未来指令扩展

**信号编码分析**：

地址19H (00 24 1A)的详细信号分析：

二进制分解：0000 0000 0010 0100 0001 1010  
位域解析：  
 M23(23): 0 - 正常微指令  
 CN(22): 0 - 无条件控制  
 WR(21): 0 - 禁止写操作   
 RD(20): 0 - 禁止读操作  
 IOM(19): 0 - 选择存储器操作  
 S3~S0(18-15): 0000 - ALU无运算，数据通路模式  
 A字段(14-12): 010 - LDB，从数据总线装载到B寄存器  
 B字段(11-9): 010 - RS\_B，RS寄存器输出到数据总线  
 C字段(8-6): 000 - NOP，无条件控制  
 UA5~UA0(5-0): 011010 - 下一地址为1AH  
  
执行效果：RS寄存器 → 数据总线 → B寄存器，准备比较操作数

地址1AH (05 80 01)的详细信号分析：

二进制分解：0000 0101 1000 0000 0000 0001  
位域解析：  
 M23(23): 0 - 正常微指令  
 CN(22): 0 - 无条件控制  
 WR(21): 0 - 禁止写操作  
 RD(20): 0 - 禁止读操作   
 IOM(19): 0 - 选择存储器操作  
 S3~S0(18-15): 1011 - ALU执行A-B运算  
 A字段(14-12): 000 - NOP，无数据输出到总线  
 B字段(11-9): 000 - NOP，不从数据总线读取  
 C字段(8-6): 000 - NOP，无条件控制  
 UA5~UA0(5-0): 000001 - 下一地址为01H  
  
执行效果：执行A-B比较运算，设置标志位，不写回结果

**CMP指令执行流程验证**： 1. **指令译码阶段**：地址3BH(00 16 19) → 跳转到19H开始CMP指令执行 2. **第一微程序周期**：地址19H(00 24 1A) → 准备操作数，跳转到1AH 3. **第二微程序周期**：地址1AH(05 80 01) → 执行比较运算，返回01H

这种设计确保了CMP指令的完整性和正确的执行序列。

#### 与SUB指令的对比分析

|  |  |  |  |
| --- | --- | --- | --- |
| 比较项目 | SUB指令(14H) | CMP指令(19H-1AH) | 技术差异说明 |
| 运算逻辑 | A-B→RD | A-B但不写回 | CMP保护原数据不变 |
| 标志位设置 | 设置FC,FZ等 | 设置FC,FZ等 | 标志位设置逻辑相同 |
| 执行周期 | 2周期 | 2周期 | 执行效率相当 |
| 寄存器影响 | 改变目标寄存器 | 不改变任何寄存器 | CMP语义更清晰 |
| 微程序复杂度 | 中等 | 中等 | 实现复杂度相近 |

### (3) 微程序控制信号深度解析

#### 微指令格式标准定义

**微指令字长与位域分布**：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 位号范围 | 23 | 22 | 21 | 20 | 19 | 18~15 | 14~12 | 11~9 | 8~6 | 5~0 |
| 字段名称 | M23 | CN | WR | RD | IOM | S3~S0 | A字段 | B字段 | C字段 | UA0~UA5 |
| 功能说明 | 保留位 | 条件控制 | 写使能 | 读使能 | IO/MEM选择 | ALU功能选择 | 指令功能 | 指令功能 | 指令功能 | 下一地址 |

**字段功能详细说明**：

- M23：保留位，用于系统扩展

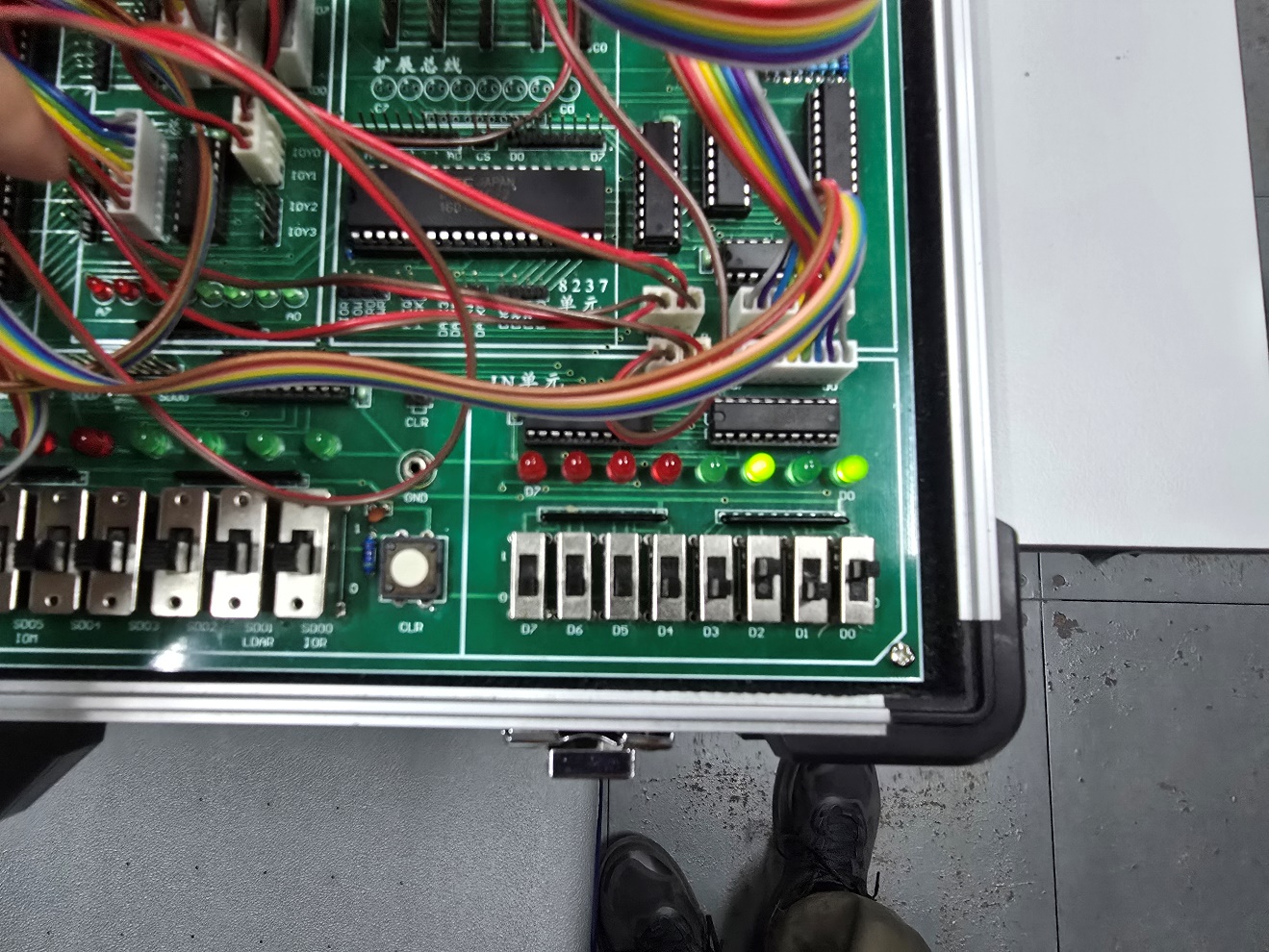
- CN：条件控制位，控制条件执行逻辑

- WR：写使能信号，控制数据写入操作

- RD：读使能信号，控制数据读取操作

- IOM：输入输出/存储器选择信号

- S3-S0：4位ALU功能选择码

- A字段：指令功能

- B字段：指令功能

- C字段：指令功能

- UA0UA5：6位下一微指令地址

#### A字段编码表（从数据总线装载控制）

|  |  |  |  |
| --- | --- | --- | --- |
| 二进制 | 含义 | 功能说明 | 应用示例 |
| 000 | NOP | 无操作，不从数据总线装载 | CMP第二周期 |
| 001 | LDA | 从数据总线装载到A寄存器 | 通用数据传输 |
| 010 | LDB | 从数据总线装载到B寄存器 | 数据交换操作 |
| 011 | LDRi | 从数据总线装载到寄存器 | 寄存器写入 |
| 100 | 保留 | 系统保留编码 | 未来扩展 |
| 101 | LOAD | 从数据总线通用装载操作 | 内存数据装载 |
| 110 | LDAR | 从数据总线装载到地址寄存器 | 地址计算 |
| 111 | LRIR | 从数据总线装载到指令寄存器 | 指令译码 |

#### B字段编码表（输出到数据总线控制）

|  |  |  |  |
| --- | --- | --- | --- |
| 二进制 | 含义 | 功能说明 | 应用示例 |
| 000 | NOP | 无操作，无数据输出到总线 | 单操作数指令 |
| 001 | ALU\_B | ALU结果输出到数据总线 | 运算结果传递 |
| 010 | RS\_B | RS寄存器输出到数据总线 | CMP指令第一周期 |
| 011 | RD\_B | RD寄存器输出到数据总线 | 数据传送指令 |
| 100 | RI\_B | 索引寄存器输出到数据总线 | 变址寻址计算 |
| 101 | 保留 | 系统保留编码 | 未来扩展 |
| 110 | PC\_B | 程序计数器输出到数据总线 | 相对寻址计算 |
| 111 | 保留 | 系统保留编码 | 未来扩展 |

#### C字段编码表（条件控制字段）

|  |  |  |  |
| --- | --- | --- | --- |
| 二进制 | 含义 | 功能说明 | 应用示例 |
| 000 | NOP | 无条件控制 | 普通微指令 |
| 001 | P<1> | 使用P<1>译码信号 | 指令操作码译码 |
| 010 | P<2> | 使用P<2>译码信号 | 寻址方式处理 |
| 011 | P<3> | 使用P<3>译码信号 | 条件跳转判断 |
| 100 | 保留 | 系统保留编码 | 未来扩展 |
| 101 | LDPC | 装载程序计数器 | 跳转指令执行 |
| 110 | 保留 | 系统保留编码 | 未来扩展 |
| 111 | 保留 | 系统保留编码 | 未来扩展 |

#### ALU控制信号(S3-S0)功能表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S3 | S2 | S1 | S0 | 功能 | 应用指令 |
| 0 | 0 | 0 | 0 | A通过 | MOV, LAD |
| 0 | 0 | 1 | 0 | A与B | AND |
| 0 | 0 | 1 | 1 | A或B | OR |
| 0 | 1 | 0 | 1 | A右移 | RR |
| 1 | 0 | 0 | 1 | A+B | ADD |
| 1 | 0 | 1 | 1 | A-B | SUB, **CMP** |
| 1 | 1 | 0 | 1 | A+1 | INC |

表4 ALU控制信号功能对照表

可以看出，CMP指令复用了SUB指令的运算功能(1011)，体现了指令系统设计的高效性。

#### CMP指令微程序解析实例

**以CMP指令为例，详细解析微指令编码过程**：

**地址19H：00 24 1A（CMP指令第一周期）**

二进制分解：0000 0000 0010 0100 0001 1010  
位域解析：  
 M23(23): 0 - 正常微指令  
 CN(22): 0 - 无条件控制  
 WR(21): 0 - 禁止写操作   
 RD(20): 0 - 禁止读操作  
 IOM(19): 0 - 选择存储器操作  
 S3~S0(18-15): 0000 - ALU无运算，数据通路模式  
 A字段(14-12): 010 - LDB，从数据总线装载到B寄存器  
 B字段(11-9): 010 - RS\_B，RS寄存器输出到数据总线  
 C字段(8-6): 000 - NOP，无条件控制  
 UA5~UA0(5-0): 011010 - 下一地址为1AH  
  
执行效果：RS寄存器 → 数据总线 → B寄存器，准备比较操作数

**地址1AH：05 80 01（CMP指令第二周期）**

二进制分解：0000 0101 1000 0000 0000 0001  
位域解析：  
 M23(23): 0 - 正常微指令  
 CN(22): 0 - 无条件控制  
 WR(21): 0 - 禁止写操作  
 RD(20): 0 - 禁止读操作   
 IOM(19): 0 - 选择存储器操作  
 S3~S0(18-15): 1011 - ALU执行A-B运算  
 A字段(14-12): 000 - NOP，无数据输出到总线  
 B字段(11-9): 000 - NOP，不从数据总线读取  
 C字段(8-6): 000 - NOP，无条件控制  
 UA5~UA0(5-0): 000001 - 下一地址为01H  
  
执行效果：执行A-B比较运算，设置标志位，不写回结果

**地址3BH：00 16 19（CMP指令入口）**

二进制分解：0000 0000 0001 0110 0001 1001  
位域解析：  
 M23(23): 0 - 正常微指令  
 CN(22): 0 - 无条件控制  
 WR(21): 0 - 禁止写操作  
 RD(20): 0 - 禁止读操作  
 IOM(19): 0 - 选择存储器操作  
 S3~S0(18-15): 0000 - ALU无运算  
 A字段(14-12): 001 - LDA，从数据总线装载到A寄存器  
 B字段(11-9): 011 - RD\_B，RD寄存器输出到数据总线  
 C字段(8-6): 000 - NOP，无条件控制  
 UA5~UA0(5-0): 011001 - 下一地址为19H  
  
执行效果：RD寄存器 → 数据总线 → A寄存器，跳转到CMP执行入口

这种详细的微指令解析展示了微程序控制器如何通过精确的位域控制实现复杂的指令功能。

## 3.4、机器程序及分析（4分）

### (1) 程序功能设计

本程序实现在预设数组[01H, 03H, 05H, 07H]中查找用户输入值的功能： - 输入：用户通过IN指令输入要查找的值 - 处理：使用线性查找算法遍历数组 - 输出：找到返回01H，未找到返回00H

### (2) 程序源代码及分析

**完整机器程序清单**：

; //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //   
; // 复杂模型机实验指令文件 //   
; //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //   
; //\*\*\*\*\*\* Start Of Main Memory Data \*\*\*\*\*\* //   
  
; 主程序代码区  
$P 00 21 ; START: IN R1, 00H - 输入查找值到R1寄存器  
$P 01 00  
$P 02 62 ; LDI R2, 00H - 初始化数组索引R2为0  
$P 03 00   
$P 04 60 ; LDI R0, 04H - 设置数组长度R0为4  
$P 05 04  
$P 06 B8 ; LOOKUP\_LOOP: CMP R2, R0 - 比较当前索引与数组长度  
$P 07 F0 ; BZC NOT\_FOUND - 如果相等(到达边界)则跳转到未找到  
$P 08 11  
$P 09 CB ; LAD R3, [RI(R2)], 60H - 使用变址寻址读取array[R2]  
$P 0A 60  
$P 0B BD ; CMP R3, R1 - 比较当前数组元素与查找目标  
$P 0C F0 ; BZC CHECK\_EQUAL - 如果相等则跳转到相等检查  
$P 0D 1E  
$P 0E 72 ; CONTINUE\_LOOP: INC R2 - 数组索引自增，准备下一次循环  
$P 0F E0 ; JMP LOOKUP\_LOOP - 无条件跳转回循环开始  
$P 10 06  
$P 11 60 ; NOT\_FOUND: LDI R0, 00H - 未找到分支，设置返回值为00H  
$P 12 00  
$P 13 E4 ; JMP END - 跳转到程序结束(间接寻址方式)  
$P 14 80  
$P 15 60 ; FOUND: LDI R0, 01H - 找到分支，设置返回值为01H   
$P 16 01  
$P 17 EC ; JMP END - 跳转到程序结束(相对寻址方式)  
$P 18 00  
$P 19 D0 ; END: STA 79H, R0 - 将查找结果存储到内存79H地址  
$P 1A 70  
$P 1B 30 ; OUT 40H, R0 - 将结果输出到40H端口显示  
$P 1C 40  
$P 1D 50 ; HLT - 停机指令，程序正常结束  
$P 1E B7 ; CHECK\_EQUAL: CMP R1,R3 - 二次确认比较，验证R1与R3是否相等  
$P 1F F0 ; BZC FOUND - 确认相等则跳转到找到分支  
$P 20 15   
$P 21 E0 ; JMP CONTINUE\_LOOP - 否则继续循环查找  
$P 22 0E   
  
; 数据存储区  
$P 60 01 ; 数据表[0]: 01H - 数组第0个元素  
$P 61 03 ; 数据表[1]: 03H - 数组第1个元素  
$P 62 05 ; 数据表[2]: 05H - 数组第2个元素  
$P 63 07 ; 数据表[3]: 07H - 数组第3个元素  
  
; //\*\*\*\*\* End Of Main Memory Data \*\*\*\*\*//   
; //\*\* Start Of MicroController Data \*\*//   
  
; 完整微程序代码  
$M 00 000001 ; 地址00H: 复位初始化  
$M 01 006D43 ; 地址01H: PC→AR, PC+1，取指周期  
$M 03 107070 ; 地址03H: MEM→IR, P<1>，指令译码  
$M 04 002405 ; 地址04H: RS→B，ADD指令第一步  
$M 05 04B201 ; 地址05H: A+B→RD，ADD指令第二步  
$M 06 002407 ; 地址06H: RS→B，AND指令第一步  
$M 07 013201 ; 地址07H: A∧B→RD，AND指令第二步  
$M 08 106009 ; 地址08H: MEM→AR，IN指令准备  
$M 09 183001 ; 地址09H: IO→RD，IN指令执行  
$M 0A 106010 ; 地址0AH: MEM→AR，OUT指令准备  
$M 0B 000001 ; 地址0BH: 空操作  
$M 0C 103001 ; 地址0CH: MEM→RD，LAD指令执行  
$M 0D 200601 ; 地址0DH: RD→MEM，STA指令执行  
$M 0E 005341 ; 地址0EH: A→PC，JMP指令执行  
$M 0F 0000CB ; 地址0FH: 条件判断，P<3>，BZC指令  
$M 10 280401 ; 地址10H: RS→IO，OUT指令执行  
$M 11 103001 ; 地址11H: MEM→RD，数据读取  
$M 12 06B201 ; 地址12H: A+1→RD，INC指令执行  
$M 13 002414 ; 地址13H: RS→B，SUB指令第一步  
$M 14 05B201 ; 地址14H: A-B→RD，SUB指令第二步  
$M 15 002416 ; 地址15H: RS→B，OR指令第一步  
$M 16 01B201 ; 地址16H: A∨B→RD, OR指令第二步  
$M 17 002418 ; 地址17H: RS→B，RR指令第一步  
$M 18 02B201 ; 地址18H: A右环移→RD，RR指令第二步  
$M 19 00241A ; 地址19H: RS→B，CMP指令第一步（新增）  
$M 1A 058001 ; 地址1AH: A-B比较，CMP指令第二步（新增）  
$M 1B 005341 ; 地址1BH: A→PC，跳转返回  
$M 1C 10101D ; 地址1CH: MEM→A，间接寻址第一步  
$M 1D 10608C ; 地址1DH: MEM→AR, P<2>，间接寻址第二步  
$M 1E 10601F ; 地址1EH: MEM→AR，准备间接跳转  
$M 1F 101020 ; 地址1FH: MEM→A，读取跳转地址  
$M 20 10608C ; 地址20H: MEM→AR, P<2>，完成间接跳转  
$M 28 101029 ; 地址28H: MEM→A，变址寻址开始  
$M 29 00282A ; 地址29H: RI→B，读取索引寄存器  
$M 2A 04E22B ; 地址2AH: A+B→AR，计算有效地址  
$M 2B 04928C ; 地址2BH: A+B→A, P<2>，完成变址寻址  
$M 2C 10102D ; 地址2CH: MEM→A，相对寻址开始  
$M 2D 002C2E ; 地址2DH: PC→B，读取当前PC值  
$M 2E 04E22F ; 地址2EH: A+B→AR，计算相对地址  
$M 2F 04928C ; 地址2FH: A+B→A, P<2>，完成相对寻址  
$M 30 001604 ; 地址30H: RD→A，MOV指令  
$M 31 001606 ; 地址31H: RD→A，跳转到AND  
$M 32 006D48 ; 地址32H: PC→AR, PC+1，双字长指令第一步  
$M 33 006D4A ; 地址33H: PC→AR, PC+1，双字长指令第二步  
$M 34 003401 ; 地址34H: RS→RD，数据传送  
$M 35 000035 ; 地址35H: 停机指令，循环等待  
$M 36 006D51 ; 地址36H: PC→AR, PC+1，LDI指令  
$M 37 001612 ; 地址37H: RD→A，跳转到INC  
$M 38 001613 ; 地址38H: RD→A，跳转到SUB  
$M 39 001615 ; 地址39H: RD→A，跳转到OR  
$M 3A 001617 ; 地址3AH: RD→A，跳转到RR  
$M 3B 001619 ; 地址3BH: RD→A，跳转到CMP（新增）  
$M 3C 006D5C ; 地址3CH: PC→AR, PC+1，LAD指令  
$M 3D 006D5E ; 地址3DH: PC→AR, PC+1，STA指令  
$M 3E 006D68 ; 地址3EH: PC→AR, PC+1，JMP指令  
$M 3F 006D6C ; 地址3FH: PC→AR, PC+1，BZC指令  
  
; //\*\* End Of MicroController Data \*\*//

**程序存储器映射分析**：

|  |  |  |  |
| --- | --- | --- | --- |
| 地址范围 | 功能模块 | 指令数量 | 说明 |
| 00H-05H | 程序初始化 | 3条 | 输入处理和变量初始化 |
| 06H-10H | 查找循环核心 | 6条 | 边界检查、元素读取、比较判断 |
| 11H-18H | 条件分支处理 | 4条 | 未找到和找到的处理分支 |
| 19H-1DH | 结果输出 | 3条 | 结果存储、输出和程序结束 |
| 1EH-22H | 辅助检查 | 3条 | 二次确认比较逻辑 |
| 60H-63H | 数据区 | 4字节 | 预设查找数组数据 |
| 70H-79H | 结果区 | - | 程序运行结果存储区域 |

# 四、实验步骤（4分）

实验在本课程配套的复杂模型计算机实验箱上进行。首先，根据实验指导书和数据通路图正确连接实验箱的各个模块，确保硬件连接无误后，开启实验箱电源。

## 4.1、微程序写入及校验（2分）

通过配套的联机软件，可以将PC端编写好的程序写入实验箱中。本次实验的微程序代码已在3.4节的程序清单中给出。

1. 打开联机软件，选择菜单命令”【转储】—【装载】“功能。
2. 在弹出的文件对话框中选择包含微程序和机器程序的.txt文件。软件将自动把微程序代码写入微指令存储器中。
3. 写入后，为确保数据准确无误，选择”【转储】—【刷新指令区】“功能，在软件界面上读出实验箱中的微指令。
4. 将读出的微指令与源文件进行比对，检查是否完全一致。如不一致，则需要重新写入或手动修改。

## 4.2、机器程序写入及校验（2分）

机器程序的写入与微程序类似，在同一次装载操作中完成。

1. 机器程序代码与微程序代码存放在同一个.txt文件中，通过上一步的装载操作，机器程序代码已经被写入主存中。
2. 通过联机软件的”主存”或”指令区”视图，可以查看写入的机器指令。
3. 逐条核对写入的机器指令与源代码是否一致，确保程序正确性，为后续的运行和调试做好准备。

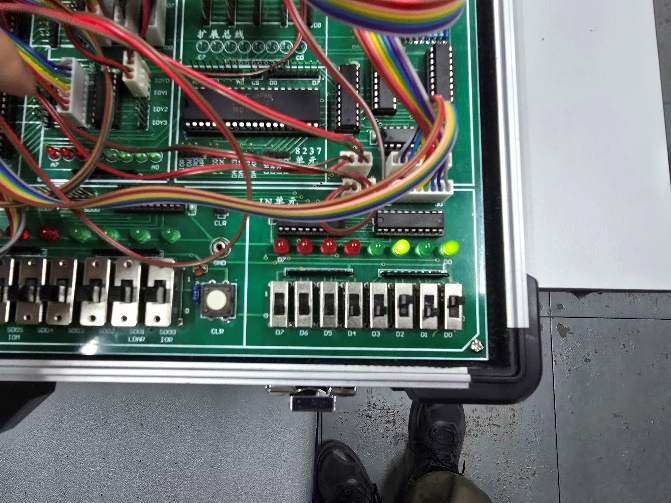
# 五、实验结果及分析（16分）

程序运行前，通过CON单元的总清按钮CLR清除各个寄存器的历史数据。通过软件运行程序，当模型机执行完OUT指令后观察LED显示的数是否正确。

## 5.1、演示程序一：查找成功

**目标**：在数组 [01H, 03H, 05H, 07H] 中查找存在的数值 05H，并验证返回结果是否为 01H。

**实验步骤与结果**：

1. **输入数据**：通过IN单元输入要查找的数值 05H。

* 图2 输入查找值05H

1. **运行程序**：点击联机软件中的”连续运行”按钮，程序开始执行。
2. **观察输出**：程序执行结束后，OUT单元的LED显示结果为 01H。

* 图形用户界面

  AI 生成的内容可能不正确。
* 图3 OUT单元输出结果01H

1. **结果分析**：暂停程序后，观察寄存器状态。可以看到，最终存有返回值的寄存器R0中的值为01H。

**结论**：实验结果与预期完全一致。程序正确执行了查找逻辑，在数组中找到了目标值05H，并按设计要求输出了01H。

## 5.2、演示程序二：查找失败

**目标**：在数组 [01H, 03H, 05H, 07H] 中查找不存在的数值 06H，并验证返回结果是否为 00H。

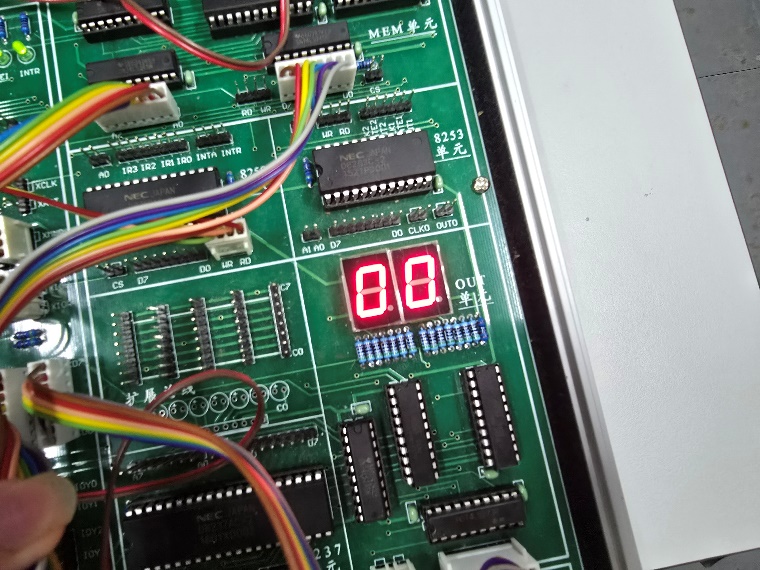
**实验步骤与结果**：

1. **输入数据**：通过IN单元输入要查找的数值 06H。

* 图片包含 游戏机, 电路

  AI 生成的内容可能不正确。
* 图4 输入查找值06H

1. **运行程序**：点击联机软件中的”连续运行”按钮，程序开始执行。
2. **观察输出**：程序执行结束后，OUT单元的LED显示结果为 00H。



* 图5 OUT单元输出结果00H

1. **结果分析**：暂停程序后，观察寄存器状态。最终存有返回值的寄存器R0中的值为00H。

**结论**：实验结果与预期完全一致。程序遍历了整个数组，没有找到目标值06H，并按设计要求正确地输出了00H。这验证了程序边界检查和查找失败逻辑的正确性。

# 六、实验问题及思考（4分）

**1、当前所实现计算机，是否完整？如果不完整，还缺少哪些部件？**

答：从基本组成来看，该计算机是完整的，具备控制器（CU）、运算器(ALU)、存储器(Memory)、输入设备(Input)和输出设备(Output)五大基本部件。指令系统也相对完备，涵盖了数据传送、算术逻辑运算、控制转移和输入输出等基本功能。特别是为本项目新增了CMP（比较）指令，使其非常适合完成基于比较的查找任务。当然，若要实现更复杂的功能，还可以进一步扩充指令集，例如增加自减（DEC）、左移等指令。

**2、当前所实现计算机，是否能实现除法运算？如果能，可通过哪些指令实现除法运算？**

答：

### 详细解答：当前计算机实现除法运算的可行性分析

**结论先行：** 当前的指令系统 **不能直接实现** 通用的除法运算。

**主要瓶颈：** 缺乏关键的 **移位** 操作。

标准的除法算法，无论是”恢复余数法”还是”不恢复余数法”（加减交替法），其核心都依赖于 **循环地进行”减法”和”移位”**。我们当前的指令系统虽然有强大的 SUB 和 CMP 指令，但仅有 RR（右循环移位），缺少实现除法所必需的 **逻辑左移（SHL）** 功能。

### 如何扩展以实现除法运算？

要让本机具备除法能力，我们需要从最底层的微指令进行扩展。

**第一步：新增”逻辑左移”微指令**

我们需要为ALU增加一个新的功能：逻辑左移。根据本机ALU的设计规范，对应的 S3-S0 编码为 0111。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S3 | S2 | S1 | S0 | 功能 | 新增助记符 |
| 0 | 1 | 1 | 1 | A << 1 | SHL |

有了这个ALU功能，我们就可以创建一条新的机器指令，例如 SHL RD，其微程序将调用这个新的ALU操作。

**第二步：利用现有指令协同完成算法**

以经典的 **恢复余数法** 为例，说明如何利用扩展后的指令系统计算 R1 / R2 (无符号)：

**所需资源：** - **被除数**：存放在R1中。 - **除数**：存放在R2中。 - **部分余数**：需要一个寄存器，例如R0，初始为0。 - **商**：结果将在R1中形成。 - **循环计数器**：需要一个寄存器，例如R3，初始为8。

**算法流程 (伪代码):** 1. **初始化**: LDI R0, 00H ; R0作为高位部分余数，初始为0 LDI R3, 08H ; 循环8次

1. **循环开始 (LOOP\_DIV):**
   1. **左移被除数与余数**: 这一步是关键，需要将[R0, R1]这个16位的整体左移一位。
      * SHL R1 (新指令)。这会把R1的最高位移入进位标志位FC。
      * ADD R0, R0 (等效于左移一位)。
      * BZC NO\_CARRY (检查上次SHL的进位)。
      * INC R0 (如果FC为1，则将进位加入R0的最低位)。
      * NO\_CARRY:
   2. **试探性相减**:
      * SUB R0, R2 ; 用部分余数减去除数。
   3. **判断与恢复**:
      * BZC SUB\_OK (检查减法是否产生借位，即FC=1表示R0 >= R2)。
      * **如果不够减 (FC=0)**:
        + ADD R0, R2 ; 恢复余数（加回去）。
        + 商的当前位为0，R1最低位已是0，无需操作。
        + JMP LOOP\_END ; 跳到循环末尾。
   4. **执行相减 (SUB\_OK):**
      * 减法成功，商的当前位置1。
      * INC R1 ; 将R1的最低位置1。
   5. **循环控制 (LOOP\_END)**:
      * SUB R3, 1H (用LDI和SUB模拟DEC)。
      * BZC END\_DIV (如果R3 == 0，结束)。
      * JMP LOOP\_DIV ; 继续下一次循环。
2. **结束 (END\_DIV):**
   * 此时，R0中为 **余数**，R1中为 **商**。

**总结:**

综上所述，虽然当前计算机的部件完整，但指令系统的功能受限于其微程序定义。通过 **扩展微程序**，为ALU增加新的运算功能（如逻辑左移），并在此基础上设计新的机器指令，完全可以实现包括除法在内的更复杂的运算。这正是微程序控制器灵活性的体现。

**3、当前所实现计算机，还能实现哪些更复杂的计算？请举例说明。**

答：基于当前以CMP和INC为核心的指令集，本机非常适合实现更复杂的数组处理算法。例如： - **排序算法**：可以实现简单的排序算法，如冒泡排序或选择排序，通过嵌套循环和反复的元素比较（CMP）与交换来完成。 - **二分查找**：如果保证数组有序，可以实现更高效的二分查找算法。这需要更复杂的地址计算（加法和移位实现除2）和逻辑判断。 - **数据统计**：可以统计数组中满足特定条件的元素个数，例如大于、小于或等于某个给定值。

**4、当前所实现计算机，指令系统的双字长指令是如何实现的？**

答：双字长指令指令是字长度等于两个机器长度的指令。在本计算机中，其实现方式为先读入前一机器字的内容，当识别到为双字长的指令时，将会读入第二次下一条机器字的内容，从而实现双字长指令。

# 七、实验验收答辩环节问题和解答（20分）

**1. 变址寻址和相对寻址原理是什么？**

答： - **变址寻址（RI变址寻址）**：有效地址 = (RI) + D - 原理：将变址寄存器RI的内容与指令中的偏移量D相加，得到实际的存储器地址 - 应用：主要用于数组访问，RI存储数组基地址，D为数组下标偏移 - 本实验应用：LAD R3,[RI(R2)],60H 中，60H为数组基地址，R2为索引，实现array[R2]的访问

* **相对寻址**：有效地址 = (PC) + D
  + 原理：将当前程序计数器PC的值与指令中的偏移量D相加，得到目标地址
  + 应用：主要用于程序跳转，实现位置无关代码
  + 优势：程序可以在内存中重定位而不需要修改跳转地址

**2. 变址寄存器是哪一个？**

答： 本实验中，我们使用**R2作为变址寄存器**，存储数组的索引值。

**3. 介绍你程序的所有的寻址方式，仔细分析命令如何选择寻址方式**

答：本程序巧妙地运用了复杂模型机提供的全部四种内存寻址方式，以满足不同场景的需求。指令通过第一个字节的 M 字段（第7、6位）来选择具体的寻址方式。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M字段 | 寻址方式 | 原理 | 程序中的实例 | 指令分析 |
| 00 | **直接寻址** | E = D | $P 19 D0 $P 1A 70 | STA 70H, R0 指令 (D0 -> 1101 00 00)。M=00，直接将R0内容存入下一字节指定的地址70H。用于将最终计算结果存储到固定的内存单元。 |
| 01 | **间接寻址** | E = (D) | $P 13 E4 $P 14 80 | JMP [80H] 指令 (E4 -> 1110 01 00)。M=01，跳转到地址80H（下一字节）中所存储的地址值（本例中M[80H]应为19H）。用于实现指针式的跳转，增加了程序的灵活性。 |
| 10 | **变址寻址** | E = (RI)+D | $P 09 CB $P 0A 60 | LAD R3,[R2]+60H指令 (CB -> 1100 10 11)。M=10，将变址寄存器R2的内容与地址60H（下一字节）相加，实现对数组元素的动态访问。这是本数组查找程序的核心寻址方式。 |
| 11 | **相对寻址** | E = (PC)+D | $P 17 EC $P 18 00 | JMP END 指令 (EC -> 1110 11 00)。M=11，将PC值（此时为19H）与偏移量00H（下一字节）相加，实现到END标号的位置无关跳转，增强了程序的可重定位性。 |

**寻址方式选择策略总结**： - **访问固定地址**: 当操作数地址是固定不变的，使用 **直接寻址**，如本例中将结果存到固定的内存单元70H，以及BZC指令的固定跳转。 - **访问数组**: 当需要按索引访问连续的数据块时，使用 **变址寻址**，将数组基址作为D，用寄存器作索引RI，高效地遍历数组。 - **灵活跳转**: 当需要通过改变内存中的值来改变跳转目标时，使用 **间接寻址**，可以实现更灵活的程序流程控制。 - **位置无关跳转**: 当希望跳转指令不依赖于代码在内存中的绝对位置时，使用 **相对寻址**，这对于编写可重定位的模块化程序至关重要。

**4. 如何改成基址寻址？**

答：要改成基址寻址，需要以下修改：

**概念差异**： - 变址寻址：有效地址 = (RI) + D，RI存储偏移量 - 基址寻址：有效地址 = (BR) + D，BR存储基地址

**具体改造方案**： 1. **指定基址寄存器**：将R1设为基址寄存器BR，存储数组起始地址60H 2. **修改指令格式**：LAD R3,[BR(R1)],R2的值 3. **调整程序逻辑**： assembly LDI R1, 60H ; R1作为基址寄存器，存储数组基地址 ; R2仍作为索引，但现在作为偏移量D使用 LAD R3,[BR(R1)],R2 ; 有效地址 = 60H + R2

**5. 写出RD->MEM的微指令二进制代码并解释**

答：RD->MEM操作对应STA指令的执行阶段，微指令地址为**0DH**：

**二进制代码**：20 06 01

**详细解析**：

二进制分解：0010 0000 0000 0110 0000 0001  
位域解析：  
 M23(23): 0 - 正常微指令  
 CN(22): 0 - 无条件控制   
 WR(21): 1 - 使能写操作  
 RD(20): 0 - 禁止读操作  
 IOM(19): 0 - 选择存储器操作  
 S3~S0(18-15): 0000 - ALU不进行运算  
 A字段(14-12): 000 - 无数据输出到总线  
 B字段(11-9): 011 - RD寄存器输出到数据总线  
 C字段(8-6): 000 - 无条件控制  
 UA5~UA0(5-0): 000001 - 下一地址为01H  
  
执行效果：RD寄存器内容 → 数据总线 → 存储器，完成数据存储

**6. WR、iom是什么意思，分别为0/1什么作用？**

答： WR 和 IOM 是微指令中的两位关键控制信号，它们共同决定了CPU与外部设备（存储器或I/O端口）之间的数据传输方向和目标。

**WR (Write) - 写使能信号 (第21位)** 这个信号控制是”读”操作还是”写”操作，是数据流向CPU（读）还是流出CPU（写）的总开关。 - WR = 0：**禁止写入**。CPU不对外写数据。这是绝大多数微指令的默认状态，例如在进行内部算术运算、数据传送或取指时。 - WR = 1：**允许写入**。CPU准备将数据总线上的数据写入到外部。具体是写入存储器还是I/O端口，则由IOM信号决定。

**IOM (Input/Output or Memory) - I/O/存储器选择信号 (第19位)** 这个信号用于选择CPU当前是与主存储器（RAM）交互还是与I/O接口电路交互。 - IOM = 0：**选择存储器**。CPU的数据总线和地址总线连接到主存储器。所有内存读写指令（如LAD, STA）都需要将IOM置0。 - IOM = 1：**选择I/O端口**。CPU的数据总线和地址总线连接到I/O端口。所有I/O读写指令（如IN, OUT）都需要将IOM置1。

**组合作用** WR和IOM（以及读信号RD）的组合，可以精确控制四种基本的数据传输操作：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| WR | IOM | 操作类型 | 对应指令 | 示例微指令（来自本机） |
| **1** | **0** | **写存储器** (CPU -> Memory) | STA | 0DH: 20 06 01 |
| **1** | **1** | **写I/O端口** (CPU -> I/O) | OUT | 10H: 28 04 01 |
| 0 | 0 | **读存储器** (Memory -> CPU) | LAD | 0CH: 10 30 01 (需RD=1) |
| 0 | 1 | **读I/O端口** (I/O -> CPU) | IN | 09H: 18 30 01 (需RD=1) |

通过这两个信号的协同工作，微程序控制器就能够精确地指挥数据在CPU、内存和I/O设备之间进行有序的流动。

**7. 8位计算机能不能运行16位制加法？**

答： **结论：** **完全可以。** 一台8位计算机虽然其ALU一次只能处理8位数据，但通过软件编程，完全可以实现16位甚至更宽位数的运算。这需要将一个宽位数运算拆解成多个8位运算来完成。

### 16位加法的核心原理：带进位的加法 (Add with Carry)

16位加法的本质是模拟我们小学时学过的竖式加法。我们先把个位数相加，如果产生进位，再把这个进位加到十位数的计算中。

对于16位加法 S = A + B，我们将16位的A、B、S都拆成高8位（High Byte）和低8位（Low Byte）： - A = AH | AL - B = BH | BL - S = SH | SL

计算过程分两步： 1. **计算低位**：SL = AL + BL。如果这个加法的结果超出了8位所能表示的范围（即大于255或0xFF），ALU的 **进位标志位（Carry Flag, FC）** 会被自动置为1。 2. **计算高位**：SH = AH + BH + FC。在计算高8位时，必须把上一步产生的进位FC也加进去，这才能保证结果的正确性。

**8. 什么时候会使用相对寻址会合适？**

答： 相对寻址的核心优势在于它能生成 **位置无关代码（Position-Independent Code, PIC）**。因此，在以下场景中使用相对寻址会特别合适：

1. **编写可重定位的程序模块或函数库**： 当您编写一个子程序或函数库时，您无法预知它最终会被加载到内存的哪个位置。如果使用绝对地址（如 JMP 100H），一旦程序加载的基地址改变，跳转就会出错。而相对寻址（如 JMP +10H）的目标地址是相对于当前PC值的偏移量，无论整个代码块被移动到哪里，这个相对位移保持不变，保证了跳转的正确性。
2. **现代操作系统中的动态链接库（DLLs）或共享对象（.so文件）**： 这些库文件会被多个进程加载到各自不同的虚拟地址空间中。为了让一份库代码能被所有进程共享，它必须是位置无关的。相对寻址是实现这一目标的关键技术。
3. **循环和短距离分支**： 在程序中，大多数跳转指令（如 if-else、for、while 循环）的目标都离当前指令不远。使用相对寻址可以用较少的位数来表示这个短距离的偏移量，从而可能使指令变得更短、更节省空间。例如，一个8位的偏移量就能覆盖-128到+127字节范围内的跳转，这在很多情况下已经足够。

**在本程序中的应用**： 我们在 $P 17 EC（JMP END）处就使用了一个简单的相对寻址。虽然偏移量是0，但它体现了这种设计思想：无论 NOT\_FOUND 和 FOUND 这两个分支的代码如何增删，只要 END 标号与该 JMP 指令的相对位置不变，代码就无需修改。如果未来我们将整个程序从地址00H移到A0H开始，这条 JMP 指令依然能正确工作，而使用绝对地址的跳转则会失败。