、



计算机组成原理课程设计

题　　目　　 计算组合数

姓　　名　　 何佳莹

班　　级　　 计科2201

学　　号　　 202203150908

提交日期　　 2024-05-27

目录

[一、实验目的 1](#_Toc168476063)

[二、实验内容 1](#_Toc168476064)

[三、实验原理（16分） 1](#_Toc168476065)

[3.1、指令系统及分析（4分） 1](#_Toc168476066)

[3.2、指令框图及分析（4分） 4](#_Toc168476067)

[3.3、指令系统对应微程序二进制代码及分析（4分） 1](#_Toc168476068)

[3.4、机器程序及分析（4分） 3](#_Toc168476069)

[四、实验步骤（4分） 6](#_Toc168476070)

[4.1、微程序写入及校验（2分） 6](#_Toc168476071)

[4.2、机器程序写入及校验（2分） 6](#_Toc168476072)

[五、实验结果及分析（16分） 7](#_Toc168476073)

[5.1、演示程序一（8分） 7](#_Toc168476074)

[5.2、演示程序二（8分） 8](#_Toc168476075)

[六、实验问题及思考（4分） 9](#_Toc168476076)

[七、实验验收答辩环节问题和解答（20分） 10](#_Toc168476077)

# 一、实验目的

综合运用所学计算机组成原理知识，设计并实现较为完整的计算机。

# 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 计算组合数

（题目及简要说明）。

# 三、实验原理（16分）

## 3.1、指令系统及分析（4分）

* + 1. **机器指令系统**

1. 指令设计

在原模型机的基础上修改后的指令仍然分为3大类：运算类指令、控制转移类指令和数据传送类指令，共16条。

运算类指令包含3种运算，算术运算、逻辑运算和移位运算。运算类指令设计有7条，分别为：ADD、CHECK、INC、SUB、SHR、SHL、CMP，所有运算类指令都为单字长，寻址方式采用寄存器直接寻址。

控制转移类指令有3条： HLT、JMP、BZC，用以控制程序的分支和转移，其中 HLT为单字长指令，JMP 和 BZC 为双字长指令。

数据传送类指令有6条： IN、OUT、MOV、LDI、LAD、STA ，用以完成寄存器和寄存器、寄存器和 I/O、寄存器和存储器之间的数据交换，除MOV 指令为单字长指令外，其余均为双字长指令。

1. 指令格式

单字长指令（ADD、CHECK、INC、SUB、SHR、SHL、CMP、HLT和MOV）及部分双字长指令（IN、OUT、LDI）的前一字长格式如表 1所示。

表 1 **单字长指令格式**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| I7 | I6 | I5 | I4 | I3 | I2 | I1 | I0 |
| OP-CODE | | | | RS | | RD | |

上表中，I7～I0分别为指令字长的8位（I7为高位，I0为地位）。其中，OP-CODE为操作码，RS为源寄存器，RD为目的寄存器，RS或RD选定的寄存器如下表 2所示。

表 2 **RS或RD选定的寄存器**

|  |  |
| --- | --- |
| RS或RD | 选定的寄存器 |
| 00 | R0 |
| 01 | R1 |
| 10 | R2 |
| 11 | R3 |

双字长指令LAD、STA、JMP和BZC指令格式如表 2所示。

表 3 **LAD、STA、JMP和BZC指令格式**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 第一字长 | | | | | | | | 第二字长 |
| I7 | I6 | I5 | I4 | I3 | I2 | I1 | I0 | I7~I0 |
| OP-CODE | | | | M | | RD | | D |

上表中，M为寻址模式，对应如表 4所示（以R2做为变址寄存器RI）。

表 4 **寻址模式**

|  |  |  |
| --- | --- | --- |
| 寻址模式M | 有效地址E | 说明 |
| 00 | E=D | 直接寻址 |
| 01 | E=(D) | 间接寻址 |
| 10 | E=(RI)+D | RI变址寻址 |
| 11 | E=(PC)+D | 相对寻址 |

1. 指令系统

表 5中列出了修改后各条指令的汇编符号、指令格式和指令功能。

表 5 **指令描述**

|  |  |  |
| --- | --- | --- |
| 助记符号 | 指令格式 | 指令功能 |
| ADD RD,RS | |  |  |  | | --- | --- | --- | | 0000 | RS | RD | | RD + RS -> RD |
| CHECK RD,RS | |  |  |  | | --- | --- | --- | | **0001** | **RS** | **RD** | | **RD & RS** |
| MOV RD,RS | |  |  |  | | --- | --- | --- | | 0100 | RS | RD | | RS -> RD |
| INC RD | |  |  |  | | --- | --- | --- | | 0111 | \*\* | RD | | RD + 1 -> RD |
| SUB RD,RS | |  |  |  | | --- | --- | --- | | 1000 | RS | RD | | RD - RS -> RD |
| SHR RD,RS | |  |  |  | | --- | --- | --- | | **1001** | **\*\*** | **RD** | | **RD >> 1 -> RD** |
| SHL RD,RS | |  |  |  | | --- | --- | --- | | **1010** | **\*\*** | **RD** | | **RD << 1 -> RD** |
| CMP RD,RS | |  |  |  | | --- | --- | --- | | **1011** | **RS** | **RD** | | **RD - RS** |
| LAD M D,RD | |  |  |  |  | | --- | --- | --- | --- | | 1100 | M | RD | D | | E -> RD |
| STA M D,RS | |  |  |  |  | | --- | --- | --- | --- | | 1101 | M | RD | D | | RD -> E |
| JMP M D | |  |  |  |  | | --- | --- | --- | --- | | 1110 | M | \*\* | D | | E -> PC |
| BZC M D | |  |  |  |  | | --- | --- | --- | --- | | 1111 | M | \*\* | D | | 当FC或FZ=1时,  E -> PC |
| IN RD,P | |  |  |  |  | | --- | --- | --- | --- | | 0010 | \*\* | RD | P | | [P] -> RD |
| OUT P,RS | |  |  |  |  | | --- | --- | --- | --- | | 0011 | RS | \*\* | P | | RS -> [P] |
| LDI RD,D | |  |  |  |  | | --- | --- | --- | --- | | 0110 | \*\* | RD | D | | D -> RD |
| HLT | |  |  |  | | --- | --- | --- | | 0101 | \*\* | \*\* | | 停机 |

* + 1. **计算机微指令**

计算机微指令字长共24位，微指令格式如表 6所示。

表 6 **微指令格式**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 23 | 22 | 21 | 20 | 19 | 18~15 | 14~12 | 11~9 | 8~6 | 5~0 |
| M23 | CN | WR | WD | IOM | S3~S0 | A字段 | B字段 | C字段 | UA5~UA0 |

上表中，UA5~UA0为6位的后续微地址，A，B，C为3个译码字段，其分别由3个控制位译码得到多种指令，具体含义见下表 7。S3--S0选择运算功能，WR,RD控制读写。C字段中的P<1>、P<2>和P<3>为测试字段，其功能是根据机器指令及相应的微代码进行译码，使微程序转入相应的微地址入口，从而完成对指令的识别。

表 7 **3个译码字段指令及对应含义**

A字段 B字段 C字段

|  |  |  |  |
| --- | --- | --- | --- |
| 14 | 13 | 12 | 选择 |
| 0 | 0 | 0 | NOP |
| 0 | 0 | 1 | LDA |
| 0 | 1 | 0 | LDB |
| 0 | 1 | 1 | LDRi |
| 1 | 0 | 0 | 保留 |
| 1 | 0 | 1 | LOAD |
| 1 | 1 | 0 | LDAR |
| 1 | 1 | 1 | LDIR |

|  |  |  |  |
| --- | --- | --- | --- |
| 11 | 10 | 9 | 选择 |
| 0 | 0 | 0 | NOP |
| 0 | 0 | 1 | ALU\_B |
| 0 | 1 | 0 | RO\_B |
| 0 | 1 | 1 | RD\_B |
| 1 | 0 | 0 | RI\_B |
| 1 | 0 | 1 | 保留 |
| 1 | 1 | 0 | PC\_B |
| 1 | 1 | 1 | 保留 |

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | 7 | 6 | 选择 |
| 0 | 0 | 0 | NOP |
| 0 | 0 | 1 | P<1> |
| 0 | 1 | 0 | P<2> |
| 0 | 1 | 1 | P<3> |
| 1 | 0 | 0 | 保留 |
| 1 | 0 | 1 | LDPC |
| 1 | 1 | 0 | 保留 |
| 1 | 1 | 1 | 保留 |

## 3.2、指令框图及分析（4分）



**图 1 指令框图**

参考实验书中复杂机模型的微程序流程图，列出了如上图所示的16条微程序的指令框图。指令框图中一个方框代表一个CPU周期，图中的菱形符号是用于控制程序流程的判别条件，可根据某些状态或标志位来决定程序的下一步操作。其中，p<1>判别可根据操作码确定初始跳转地址或确定寻址方式，p<2>判别用于确定指令功能，P<3>判别基于ALU操作后的标志位进行跳转。

从上图中不难发现，所有指令的执行都从取指令开始，通过PC寄存器（程序计数器）指定要读取的内存地址，然后从内存中取出指令，存储到IR（指令寄存器）中，即取指令周期；然后根据IR中的内容，解析出具体指令，并进行相应的操作，即译码阶段。

## 3.3、指令系统对应微程序二进制代码及分析（4分）

修改后的微程序代码及功能说明如下表 8所示（**下表中绿色内容为书上错误，在此纠正**）。

表 8 **微程序二进制代码及含义表**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 地址 | 十六进制表示 | 高五位 | S3-S0 | A字段 | B字段 | C字段 | UA5-UA0 | 助记符 |
| 00 | 00 00 01 | 00000 | 0000 | 000 | 000 | 000 | 000001 | NOP |
| 01 | 00 6D 43 | 00000 | 0000 | 110 | 110 | 101 | 000011 | PC→AR,PC+1 |
| 03 | 10 70 70 | 00010 | 0000 | 111 | 000 | 001 | 110000 | MEM→IR,P<1> |
| 04 | 00 24 05 | 00000 | 0000 | 010 | **010** | 000 | 000101 | RS→B |
| 05 | 04 B2 01 | 00000 | 1001 | 011 | 001 | 000 | 000001 | A+B→RD |
| 06 | 00 24 07 | 00000 | 0000 | 010 | **010** | 000 | 000111 | RS→B |
| 07 | **01 02 01** | 00000 | 0010 | **000** | 001 | 000 | 000001 | **A&B** |
| 08 | 10 60 09 | 00010 | 0000 | 110 | 000 | 000 | 001001 | MEM→AR |
| 09 | 18 30 01 | 00011 | 0000 | 011 | 000 | 000 | 000001 | IO→RD |
| 0A | 10 60 10 | 00010 | 0000 | 110 | 000 | 000 | 010000 | MEM→AR |
| 0B | 00 00 01 | 00000 | 0000 | 000 | 000 | 000 | 000001 | NOP |
| 0C | 10 30 01 | 00010 | 0000 | 011 | 000 | 000 | 000001 | MEM→RD |
| 0D | 20 06 01 | 00100 | 0000 | 000 | 001 | 100 | 000001 | RD→MEM |
| 0E | 00 53 41 | 00000 | 0000 | 101 | 001 | 101 | 000001 | A→PC |
| 0F | 00 00 CB | 00000 | 0000 | 000 | 000 | 011 | 001011 | NOP,P<3> |
| 10 | 28 04 01 | 00101 | 0000 | 000 | 010 | 000 | 000001 | RS→IO |
| 11 | 10 30 01 | 00010 | 0000 | 011 | 000 | 000 | 000001 | MEM→RD |
| 12 | 06 B2 01 | 00000 | 1101 | 011 | 001 | 000 | 000001 | A+1→RD |
| 13 | 00 24 14 | 00000 | 0000 | 010 | **010** | 000 | 010100 | RS→B |
| 14 | 05 B2 01 | 00000 | 1011 | 011 | 001 | 000 | 000001 | A-B→RD |
| 15 | **03 32 01** | 00000 | **0110** | **011** | **001** | 000 | **000001** | **A>>1->RD** |
| 16 | **00 24 18** | 00000 | **0000** | **010** | **010** | 000 | **011000** | **RS→B** |
| 17 | **03 B2 01** | 00000 | **0111** | **011** | **001** | 000 | **000001** | **A<<1->RD** |
| 18 | **05 82 01** | 00000 | **1011** | **000** | 001 | 000 | 000001 | **A-B** |
| 1B | 00 53 41 | 00000 | 0000 | 101 | 001 | 101 | 000001 | A→PC |
| 1C | 10 10 1D | 00010 | 0000 | 001 | 000 | 000 | 011101 | MEM→A |
| 1D | 10 60 8C | 00010 | 0000 | 110 | 000 | 010 | 001100 | MEM→AR,P<2> |
| 1E | 10 60 1F | 00010 | 0000 | 110 | 000 | 000 | 011111 | MEM→AR |
| 1F | 10 10 20 | 00010 | 0000 | 001 | 000 | 000 | 100000 | MEM→A |
| 20 | 10 60 8C | 00010 | 0000 | 110 | 000 | 010 | 001100 | MEM→AR,P<2> |
| 28 | 10 10 29 | 00010 | 0000 | 001 | 000 | 000 | 101001 | MEM→A |
| 29 | 00 28 2A | 00000 | 0000 | 010 | 100 | 000 | 101010 | RI→B |
| 2A | 04 E2 2B | 00000 | 1001 | 110 | 001 | 000 | 101011 | A+B→AR |
| 2B | 04 92 8C | 00000 | 1001 | 001 | 001 | 010 | 001100 | A+B→A,P<2> |
| 2C | 10 10 2D | 00010 | 0000 | 001 | 000 | 000 | 101101 | MEM→A |
| 2D | 00 2C 2E | 00000 | 0000 | 010 | 110 | 000 | 101110 | PC→B |
| 2E | 04 E2 2F | 00000 | 1001 | 110 | 001 | 000 | 101111 | A+B→AR |
| 2F | 04 92 8C | 00000 | 1001 | 001 | 001 | 010 | 001100 | A+B→A,P<2> |
| 30 | 00 16 04 | 00000 | 0000 | 001 | 011 | 000 | 000100 | RD→A |
| 31 | 00 16 06 | 00000 | 0000 | 001 | 011 | 000 | 000110 | RD→A |
| 32 | 00 6D 48 | 00000 | 0000 | 110 | 110 | 101 | 001000 | PC→AR,PC+1 |
| 33 | 00 6D 4A | 00000 | 0000 | 110 | 110 | 101 | 001010 | PC→AR,PC+1 |
| 34 | 00 34 01 | 00000 | 0000 | 011 | 010 | 000 | 000001 | RS→RD |
| 35 | 00 00 35 | 00000 | 0000 | 000 | 000 | 000 | 110101 | NOP |
| 36 | 00 6D 51 | 00000 | 0000 | 110 | 110 | 101 | 010001 | PC→AR,PC+1 |
| 37 | 00 16 12 | 00000 | 0000 | 001 | 011 | 000 | 010010 | RD→A |
| 38 | 00 16 13 | 00000 | 0000 | 001 | 011 | 000 | 010011 | RD→A |
| 39 | 00 16 15 | 00000 | 0000 | 001 | 011 | 000 | 010101 | RD→A |
| 3A | 00 16 17 | 00000 | 0000 | 001 | 011 | 000 | 010111 | RD→A |
| 3B | **00 16 16** | 00000 | 0000 | **001** | **011** | 000 | **010110** | **RD→A** |
| 3C | 00 6D 5C | 00000 | 0000 | 110 | 110 | 101 | 011100 | PC→AR,PC+1 |
| 3D | 00 6D 5E | 00000 | 0000 | 110 | 110 | 101 | 011110 | PC→AR,PC+1 |
| 3E | 00 6D 68 | 00000 | 0000 | 110 | 110 | 101 | 101000 | PC→AR,PC+1 |
| 3F | 00 6D 6C | 00000 | 0000 | 110 | 110 | 101 | 101100 | PC→AR,PC+1 |

多个微指令组成的序列可以用来实现微程序，参考3.2节中指令框图对应实现（如表 9所示）。

## 3.4、机器程序及分析（4分）

* + 1. **程序功能概述**

组合数是从 个元素中不重复地选取 个元素的所有组合的个数，其公式为

下列步骤通过基本的寄存器操作、条件判断和循环控制来完成组合数的计算。

* + 1. ​**程序主要步骤**

1. 读取数据与初始化：

从IN单元中先后读取两个整数 和 并分别输送到寄存器 R0 和 R1 中；计算 R0 和 R1 的差值，即 ，并将结果存储在寄存器 R2 中；将寄存器 R3 初始化为0，用作计数器；将寄存器 R0 初始化为1，用作存储计算结果的寄存器。

1. 保存输入值

将 、和计数器 R3 的值分别保存到内存地址 60H、61H及62H中。

1. 外层循环 (LOOP)

在外层循环中，程序判断计数器 R3 是否等于 m。如果相等，则跳转到结果输出阶段；否则，程序将计数器 R3 加1，并更新内存中的值。同时，程序将寄存器 R2 加1，并更新内存中的值。

1. 乘法计算 (MULT)

初始化寄存器 R1 用作比较数。如果R2为零，程序跳转到乘法结束部分；否则，通过判断R2的尾数是否为1来判断是否应当对当前结果加上寄存器 R0 的值；然后，程序左移寄存器 R0 的值，并右移寄存器 R2 的值，重复以上步骤，直至R2为零。

1. 除法计算 (DIV)

在乘法结束后，程序从内存地址 62H 读取计数器 到寄存器 R1。程序将 R0 初始化为1，用作存储除法结果的寄存器。进入除法循环，程序根据寄存器 R3 和 R1 的比较结果，进行相应的减法和计数操作，直至 R3 等于 0。

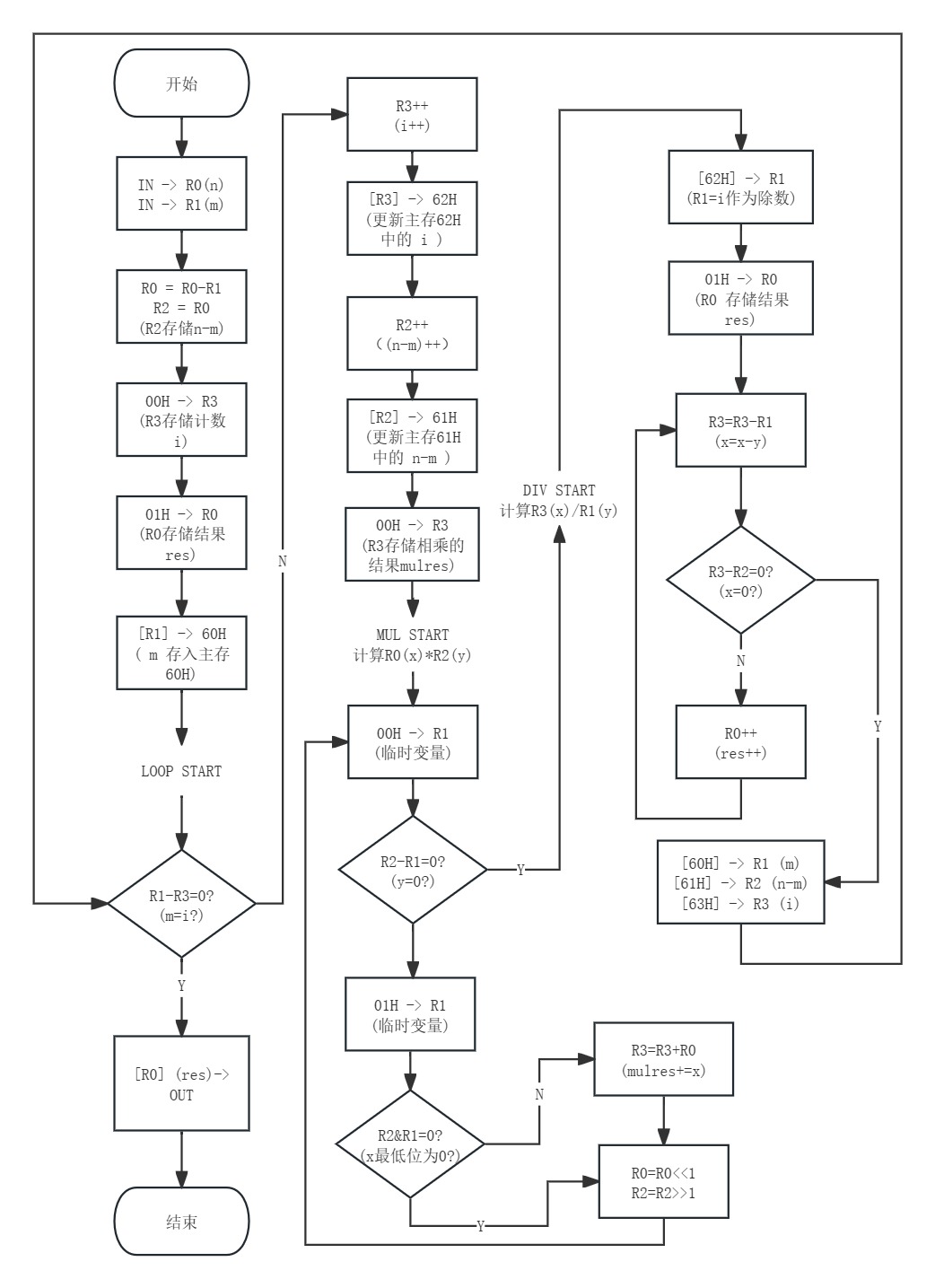
1. 跳转回外层循环

程序从内存中恢复 、 和计数器 的值到相应的寄存器。然后，程序跳转回外层循环，继续执行计算。

1. 输出结果

将寄存器R0中存储的数据输出到OUT单元。

* + 1. **程序流程框图**



**图 2 程序流程图**

* + 1. **程序源代码及说明**

表 9 **程序源代码及说明表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 地址  （十六进制） | 地址  （二进制） | 内容  （十六进制） | 内容  （二进制） | 助记符 | 说明 |
| 00 | 00000000 | 20 | 00100000 | IN R0,00H | 读取 n |
| 01 | 00000001 | 00 | 00000000 |  |  |
| 02 | 00000010 | 21 | 00100001 | IN R1,00H | 读取 m |
| 03 | 00000011 | 00 | 00000000 |  |  |
| 04 | 00000100 | 84 | 10000100 | SUB R0,R1 | R0 = n-m |
| 05 | 00000101 | 42 | 01000010 | MOV R2,R0 | R2 = R0 |
| 06 | 00000110 | 63 | 01100011 | LDI R3,00H | R3 存储计数 ：i |
| 07 | 00000111 | 00 | 00000000 |  |  |
| 08 | 00001000 | 60 | 01100000 | LDI R0,01H | R0 存储结果 ：res |
| 09 | 00001001 | 01 | 00000001 |  |  |
| 0A | 00001010 | D1 | 11010001 | STA R1,60H | 将 m 存入主存60H |
| 0B | 00001011 | 60 | 01100000 |  |  |
| 0C | 00001100 | BD | 10111101 | CMP R1,R3 | i==m |
| 0D | 00001101 | F0 | 11110000 | BZC RESULT | 判断大循环是否结束 |
| 0E | 00001110 | 39 | 00111001 |  |  |
| 0F | 00001111 | 73 | 01110011 | INC R3 | i++ |
| 10 | 00010000 | D3 | 11010011 | STA R3,62H | 更新主存62H中的i |
| 11 | 00010001 | 62 | 01100010 |  |  |
| 12 | 00010010 | 72 | 01110010 | INC R2 | n-m++ |
| 13 | 00010011 | D2 | 11010010 | STA R2,61H | 更新主存61H中的  n-m |
| 14 | 00010100 | 61 | 01100001 |  |  |
| 15 | 00010101 | 63 | 01100011 | LDI R3,00H | R3 存放相乘的结果 |
| 16 | 00010110 | 00 | 00000000 |  |  |
| 17 | 00010111 | 61 | 01100001 | LDI R1,00H | R1 存放比较数 |
| 18 | 00011000 | 00 | 00000000 |  |  |
| 19 | 00011001 | B6 | 10110110 | CMP R2,R1 | n-m==0? |
| 1A | 00011010 | F0 | 11110000 | BZC DIV | 跳转到除法开始 |
| 1B | 00011011 | 26 | 00100110 |  |  |
| 1C | 00011100 | 61 | 01100001 | LDI R1,01H | R1 存放比较数 |
| 1D | 00011101 | 01 | 00000001 |  |  |
| 1E | 00011110 | 19 | 00011001 | CHECK R1,R2 | (n-m)%2==0? |
| 1F | 00011111 | F0 | 11110000 | BZC EVEN | 跳过加操作 |
| 20 | 00100000 | 22 | 00100010 |  |  |
| 21 | 00100001 | 03 | 00000011 | ADD R3,R0 | res+=R0 |
| 22 | 00100010 | A0 | 10100000 | SHL R0 | R0<<=1 |
| 23 | 00100011 | 92 | 10010010 | SHR R2 | R2>>=1 |
| 24 | 00100100 | E0 | 11100000 | JMP MULT | 进入下一次乘法循环 |
| 25 | 00100101 | 17 | 00010111 |  |  |
| 26 | 00100110 | C1 | 11000001 | LAD R1,62H | R1=i 临时作为除数 |
| 27 | 00100111 | 62 | 01100010 |  |  |
| 28 | 00101000 | 60 | 01100000 | LDI R0,01H | R0 仍然作为结果 |
| 29 | 00101001 | 01 | 00000001 |  |  |
| 2A | 00101010 | 87 | 10000111 | SUB R3,R1 | R3-=R1 |
| 2B | 00101011 | BB | 10111011 | CMP R3,R2 | R3==0? |
| 2C | 00101100 | F0 | 11110000 | BZC DIVEND | 跳转除法结束的赋值操作 |
| 2D | 00101101 | 31 | 00110001 |  |  |
| 2E | 00101110 | 70 | 01110000 | INC R0 | R0++ |
| 2F | 00101111 | E0 | 11100000 | JMP DIV | 进入下一次除法循环 |
| 30 | 00110000 | 2A | 00101010 |  |  |
| 31 | 00110001 | C1 | 11000001 | LAD R1, 60H | R1=m |
| 32 | 00110010 | 60 | 01100000 |  |  |
| 33 | 00110011 | C2 | 11000010 | LAD R2, 61H | R2=n-m |
| 34 | 00110100 | 61 | 01100001 |  |  |
| 35 | 00110101 | C3 | 11000011 | LAD R3, 62H | R3=i |
| 36 | 00110110 | 62 | 01100010 |  |  |
| 37 | 00110111 | E0 | 11100000 | JMP LOOP | 进入下一次大循环 |
| 38 | 00111000 | 0C | 00001100 |  |  |
| 39 | 00111001 | 30 | 00110000 | OUT R0, 40H | 输出结果 |
| 3A | 00111010 | 40 | 01000000 |  |  |
| 3B | 00111011 | 50 | 01010000 | HLT | 停机 |
|  |  |  |  |  |  |
| 60 | 01100000 | 00 | 00000000 |  | 存储m |
| 61 | 01100001 | 00 | 00000000 |  | 存储n-m |
| 62 | 01100002 | 00 | 00000000 |  | 存储i |

# 四、实验步骤（4分）

## 4.1、微程序写入及校验（2分）

选择联机软件的“转储”→“装载”命令，将微程序以txt文件的形式（内容如表 8所示）写入实验平台装载入TD-CMA实验系统。装载过程中，在软件的输出区的“结果”栏会显示装载信息。如图 3所示，程序装载成功。

## 4.2、机器程序写入及校验（2分）

将机器程序以txt文件的形式装载入TD-CMA实验系统（内容如表 9所示）。

如图 3所示，微程序和机器程序成功写入实验平台，且通过了验证



图 3 程序装载完成图

# 五、实验结果及分析（16分）

## 5.1、演示程序一（8分）

数据（3分）：通过IN单元分别将6和2输入模型机，计算。

结果（2分）：OUT单元输出0FH



图 4 实验箱计算结果图

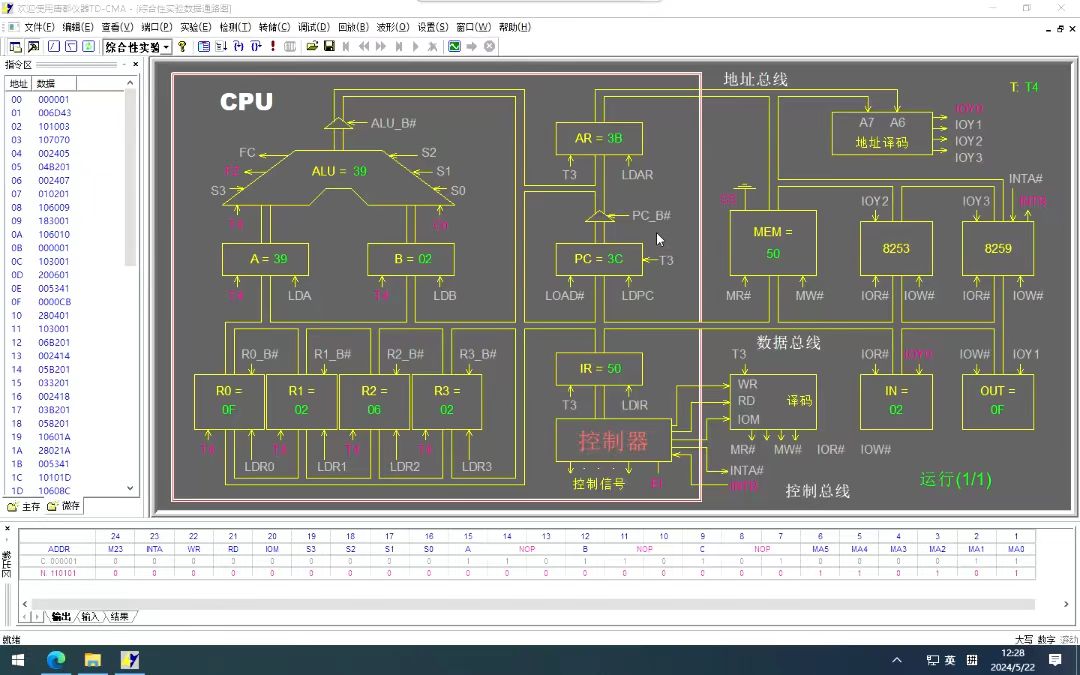


图 5 软件计算结果图

分析（3分）：

根据组合数的计算公式

由图 4和图 5可知软件输出结果和实验箱输出结果都与预期相一致，说明程序能够正确的执行微指令的地址跳转，完成每条微指令的预期操作, 最终实现组合数的计算，并能正确输出对应的结果。

## 5.2、演示程序二（8分）

数据（3分）：通过IN单元分别将9和2输入模型机，计算。

结果（2分）：OUT单元输出24H

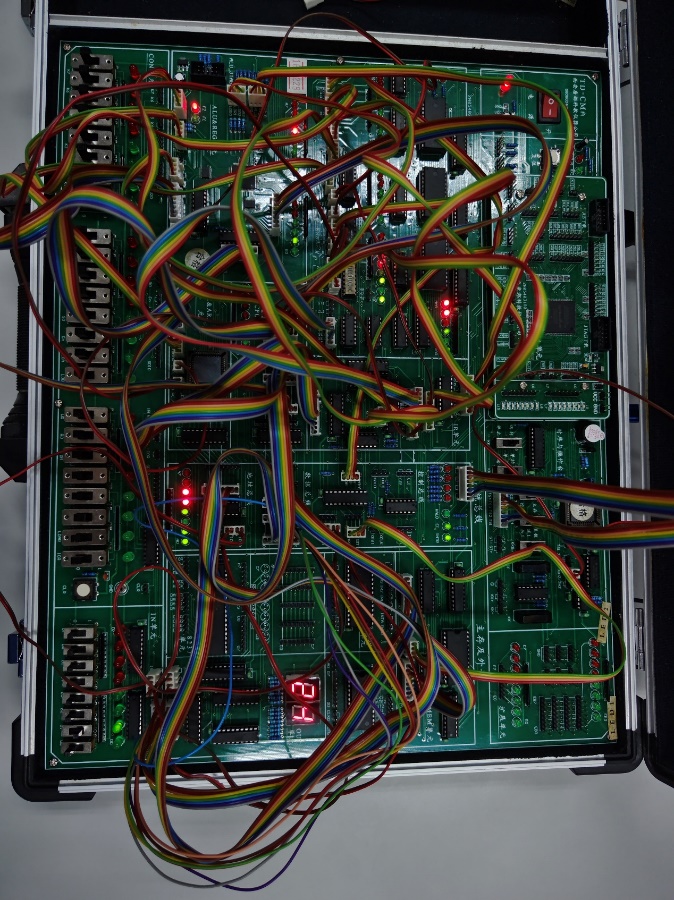


图 6 实验箱计算结果图

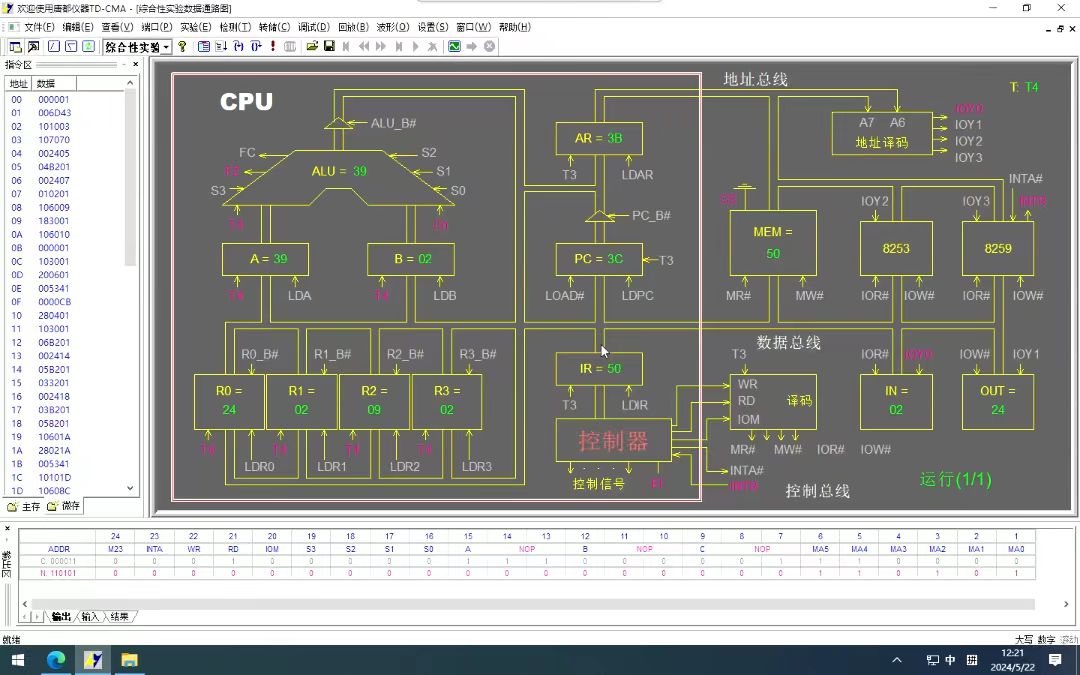


图 7 软件计算结果图

分析（3分）：

根据组合数的计算公式

由图 6和图 7可知软件输出结果和实验箱输出结果都与预期相一致，说明程序能够正确的执行微指令的地址跳转，完成每条微指令的预期操作, 最终实现组合数的计算，并能正确输出对应的结果。

# 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？

**从指令系统构成来看**，其功能是基本完备的，因为它具备基本的运算、数据转移、条件跳转、程序转移控制等指令，对于一般的程序，都可以在当前实现的计算机上运行。但是考虑到寄存器数量和控存空间的限制以及课设中为实现程序所更改的指令（如OR或操作），其就显得不够完备，并不能实现所有功能。

**从冯诺依曼计算机的结构来看，**现代电子计算机是由运算器、存储器、控制器、适配器、总线和输入/输出设备组成的，而实验中实现的计算机缺少了适配器。在模型机中，外围设备、存储器、运算器共用一条数据线路，它们间没有适配器相连，也就是没有Cache，所以程序运行时整体速率往往会向速度最低的设备看齐，所以该计算机运算速度低下。

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？

能。

假设需要计算 R0 / R1，结果存储在 R2，余数存储在 R3。以下是主要步骤：

1. 初始化商 R2 为 0。
2. 初始化余数 R3 为被除数 R0。
3. 使用循环逐步减去除数 R1，并增加商 R2。
4. 当余数 R3 小于除数 R1 时，结束循环，R2 中的值即为商。

对应指令如下：

IN R0, 00H ; 读取被除数 R0

IN R1, 00H ; 读取除数 R1

LDI R2, 00H ; 商 R2 初始化为 0

MOV R3, R0 ; 余数 R3 初始化为被除数 R0

DIV\_LOOP:

CMP R3, R1 ; 比较余数 R3 和除数 R1

BZC DIV\_END ; 如果 R3 < R1，跳转到 DIV\_END

SUB R3, R1 ; R3 = R3 - R1

INC R2 ; 商 R2 加1

JMP DIV\_LOOP; 跳转回 DIV\_LOOP

DIV\_END:

OUT R2, 00H ; 输出商 R2

OUT R3, 00H ; 输出余数 R3

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。

我所实现的组合数的计算，本质上是实现了乘法和除法。那么所有能用乘法、除法和基本运算完成的计算都可以实现。如计算最大公因数(辗转相除法)、最小公倍数(两数相乘之后除以它们的最大公因数)、海伦公式计算三角形的面积、计算两点之间的欧氏距离等等。

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？

经过取指令周期（通过PC寄存器（程序计数器）指定要读取的内存地址，然后从内存中取出一条指令，存储到IR（指令寄存器））和译码（根据IR中的内容，解析出具体指令，并进行相应的操作）后，如果解析到本条指令是指定了寻址方式的双字长指令，那么在进行P<1>判别后，执行阶段会跳转进行PC -> AR, PC + 1操作，从而读取下一个字长中的数据，进一步获得操作数E，最终实现双字长指令的读取。

# 七、实验验收答辩环节问题和解答（20分）

1. 问题：所更改的指令都是什么字长的？寻址方式？

答：都是单字长的指令，都是寄存器寻址。（由于双字长指令LAD、STA、JMP和BZC中可以一个操作数的寻址方式进行选择，而我选择的都是直接寻址，惯性思维就认为所有指令都是直接寻址了……所有单字长指令（无论双操作数还是单操作数）都是寄存器寻址，而IN、OUT、LDI包括LAD、STA中的一个操作数也是寄存器寻址）。

1. 问题：写入控存中的微指令有多少条？

答：54条而不是64条（我的微指令是基于原有微指令上面修改的，并没有注意到微指令之间的地址进行了跳跃…）

1. 问题：LOPC的作用？

答：单独作用可以使得PC+1，与LOAD一起作用可以将数据打入PC（我的原回答中是：LDPC是计数器的计数时钟，LOAD可以控制三态门输入到PC中，二者一起可以将数据打入PC。）

1. 问题：