

计算机组成原理课程设计

实验报告

实验参与者：孙嘉祎，黄欣灵，李静

指导教师：周强

2022年7月6日

目 录

[**1.实验平台硬件资源** 4](#_Toc108086997)

[1.1规格概述 4](#_Toc108086998)

[1.2主要功能 4](#_Toc108086999)

[**2.模型计算机指令集** 5](#_Toc108087000)

[2.1指令集概述 5](#_Toc108087001)

[2.2寻址方式，寄存器、IO规定 7](#_Toc108087002)

[2.3微码字段含义 8](#_Toc108087003)

[2.3.1微码格式 8](#_Toc108087004)

[2.3.2P测试 8](#_Toc108087005)

[2.3.3ALU部件功能及开关 9](#_Toc108087006)

[2.3.4存储器控制开关 9](#_Toc108087007)

[2.3.5微码具体含义 10](#_Toc108087008)

[2.4模型机如何执行一条指令 12](#_Toc108087009)

[2.4.1数据通路图 12](#_Toc108087010)

[2.4.2微指令方框图 13](#_Toc108087011)

[2.4.3ADD指令如何进行 14](#_Toc108087012)

[2.5如何对模型机编程 18](#_Toc108087013)

[2.5.1微码写入 18](#_Toc108087014)

[2.5.2应用程序写入 19](#_Toc108087015)

[**3.微码设计** 20](#_Toc108087016)

[3.1新增指令功能及分析 20](#_Toc108087017)

[3.2改进后的微码表 20](#_Toc108087018)

[**4.应用程序设计** 25](#_Toc108087019)

[4.1验证新增指令 25](#_Toc108087020)

[4.2例程——N个数求和 26](#_Toc108087021)

[4.3自主设计——猜数游戏 29](#_Toc108087022)

[**5.试验心得体会** 31](#_Toc108087023)

**1.实验平台硬件资源**

这部分描述试验器材的基本规格和功能。

1.1规格概述

本试验模拟的是一种8位模型计算机，具备输入，运算，显示，编程等基本功能的模拟。

模型机采用微程序控制的设计思路，数据总线宽度8位，地址总线宽度8位，主存储器由SRAM组成，有效容量为2K\*8位，控制存储器由ROM组成，有效容量为64\*24位。输入输出设备与内存独立编址，I/O地址为8位，由2位读写片选控制和6位端口号构成，最多可拖带64个外围设备。

寄存器方面：模型机有4个通用寄存器R0-R3，用2位二进制数依次编号。两个数据缓冲寄存器A，B连接ALU两个输入。固定R2作为变址寄存器，程序状态字实时监测，未设置单独的寄存器。

1.2主要功能

模型机的主要结构如图所示：

Diagram

Description automatically generated

主要功能有：

访存

算术运算和逻辑运算

输入输出

指令跳转

**2.模型计算机指令集**

这部分介绍模型机的机器指令和微指令的详细含义和运行过程。

2.1指令集概述

模型机共16条指令，其中15条已由厂家设计完毕，预留有一条保留指令（命令代码1011）用于进行扩展试验。

模型机有单字长双字长两种，其中所有运算类指令均为单字长，所有访存/IO类指令均为双字长

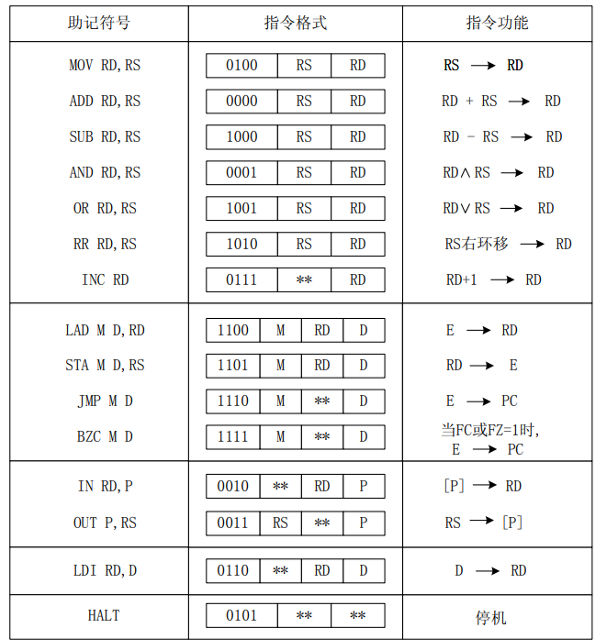
单字长指令格式：

|  |  |  |
| --- | --- | --- |
| 4位操作码 | 2位源寄存器号 | 2位目的寄存器号 |

双字长指令格式：

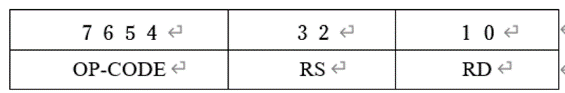
|  |  |  |
| --- | --- | --- |
| 4位操作码 | 2位寻址方式或寄存器号 | 2位寄存器号 |
| 8位二进制数值 | | |

下表是所有指令及其字段含义。

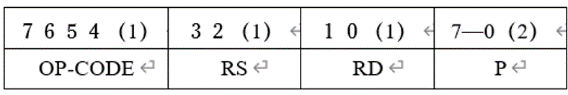


RD循环右移RS位放回RD(右移值取RS低3位)

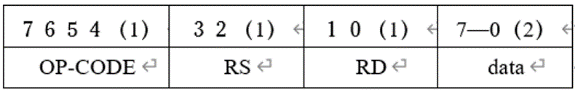
单字节指令：



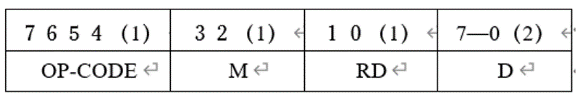
INOUT指令：P为8位IO地址



LDI指令，data为8为立即数



JMP，BZC，D为8位偏移量



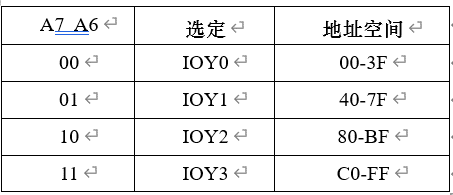
2.2寻址方式，寄存器、IO规定

RS，RD代表源寄存器和目的寄存器，编号规定如表：

Table

Description automatically generated

在IO指令中给出8位IO地址，其含义如下



在输入输出设备各只用一个的情况下，A7A6=00代表输入，01代表输出，后6位地址任意。

访存指令的M字段代表寻址方式，除LDI为取立即数指令外，寻址方式有四种：直接，间接，变址，相对，由2位M字段决定，具体编码如表。

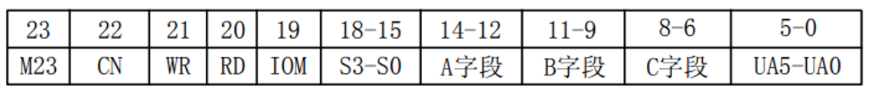
Table

Description automatically generated

2.3微码字段含义

2.3.1微码格式

本模型机控存字长24位，除最高位保留不用外其他各部分功能如图所示：



其中M23不用，ABC为微码编码表示的分组，后6位为断定法的后续微地址。

2.3.2P测试

这套微码有三个P测试，P<1>，P<2>和P<3>。其中P<1>根据机器指令操作码选择微程序入口，P<2>是几个需要访存的指令的具体划分，P<3>为BZC跳转逻辑。图为P测试的具体实现方式

Diagram, engineering drawing

Description automatically generated

P<1>P<2>的综合效果是：微程序的入口地址=0x30（（110000）BIN）+机器指令的4位操作码，P<3>实现了BZC根据状态字条件转移功能

2.3.3ALU部件功能及开关

ALU有2个操作数输入，从数据缓冲寄存器读入，1个进位输入，被微码的CN字段控制，4位操作选择S0-S3，两个程序状态字FC，FZ，其中S0-S3对应微码的S0-S3位，具体编码方式如表：

Table

Description automatically generated

两个状态字含义：FC代表有进位，FZ代表运算结果为0，当FC和FZ任何为1时，都可以触发BZC指令（如有）进行跳转。

2.3.4存储器控制开关

存储和IO由一个开关切换，被微码的IOM字段控制，IOM=1访外，IOM=0访存。读取（输入）写入（输出）被微码的RD，WR控制（二者不能同时有效）图为工作原理图

Diagram

Description automatically generated

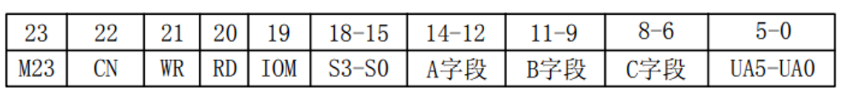
2.3.5微码具体含义

后面一部分采用编码表示，3位一组分三组，编码如表：

A picture containing text, electronics

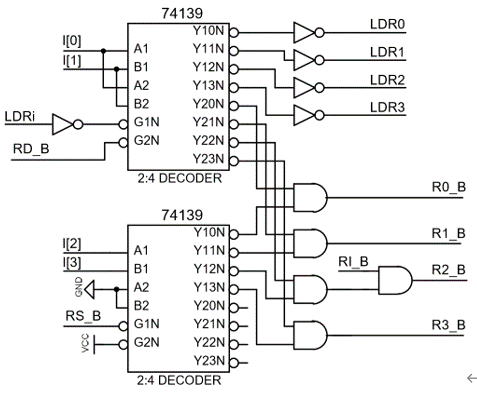
Description automatically generated

至此我们可以归纳出各字段含义，如表所示。



| 微码 | 含义 |
| --- | --- |
| CN | 运算器输入进位 |
| WR | 写入（输出） |
| RD | 读取（输入） |
| IOM | 访外/访存 |
| S3-S0 | 选择ALU执行何种运算 |
| LDA | 数据缓冲寄存器A进料 |
| LDB | 数据缓冲寄存器B进料 |
| LDRi | RD写开关 |
| LOAD | 地址送入PC |
| LDPC | 程序计数器自增 |
| LDAR | 地址寄存器进料 |
| LDIR | 指令寄存器进料 |
| ALU\_B | ALU出料 |
| RS\_B | RS读开关 |
| PC\_B | 程序计数器内容读出 |
| RD\_B | RD读开关 |
| RI\_B | 变址寄存器读开关 |
| P<X> | P测试 |
| NOP | 什么也不做 |

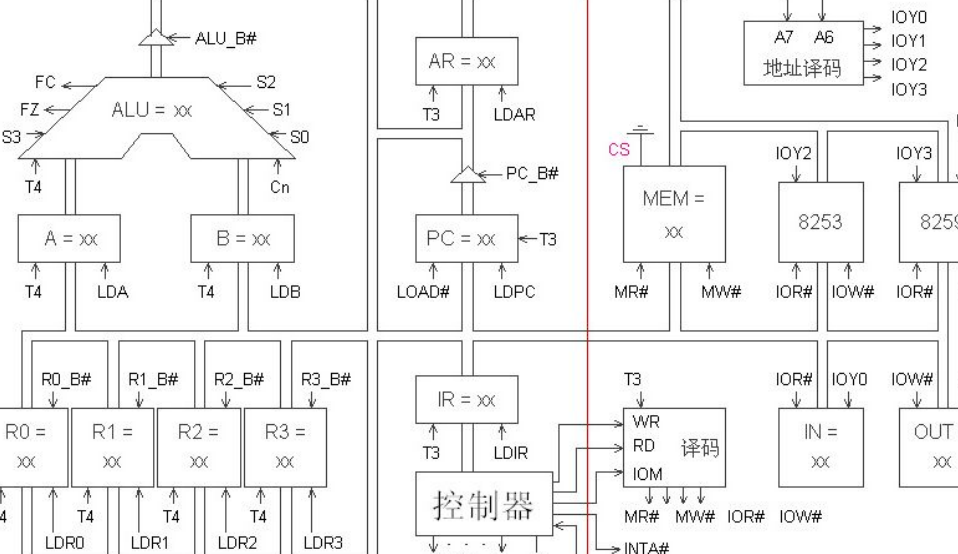
需要注意的是，RS与RD由机器指令直接决定，但需要微码开关启机，这个过程并不是P测试的管辖范围而是由单独的判断过程，下面是这个判断的工作原理图：



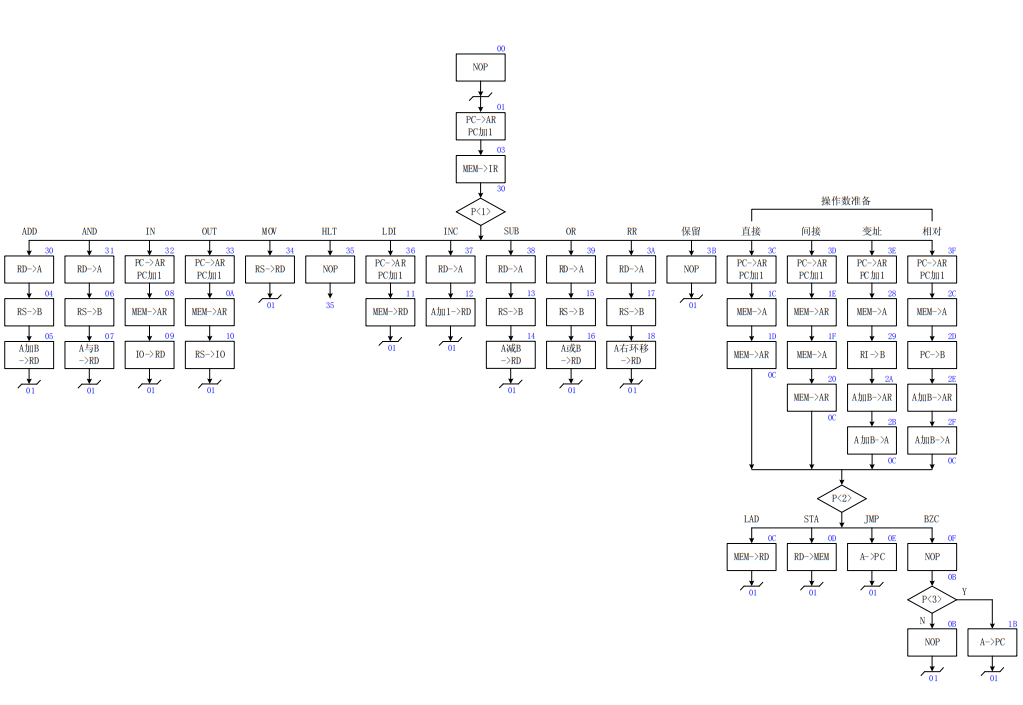
2.4模型机如何执行一条指令

2.4.1数据通路图

如图所示：



2.4.2微指令方框图

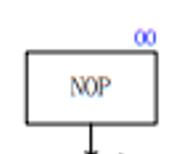


2.4.3ADD指令如何进行

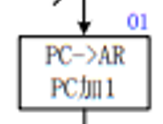
我们以ADD指令为例分析这个过程，不考虑程序中断。

假定PC当中的值为0x00，内存中的内容为ADD R1 R0，即0000 0001，现分析这条指令运行的全过程

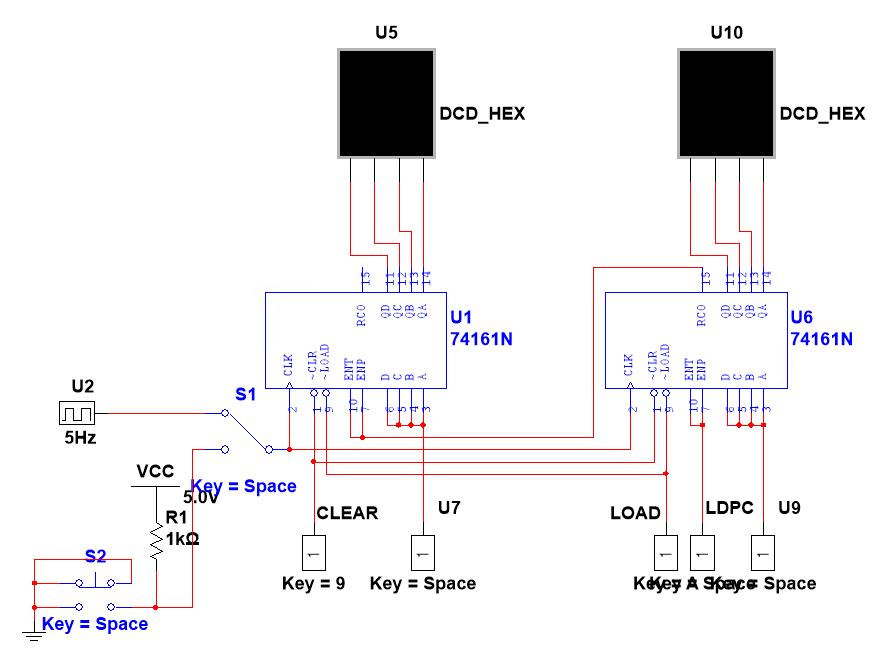
* **S0：**控存当中后续微地址为000000，转到第一条微指令，这条微指令什么也不做：



* **S1：**后续微地址为000001，转到取指的过程：

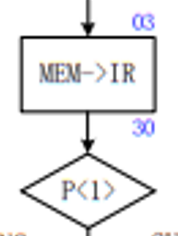


这条微指令有三个有效字段，首先A=110打开地址寄存器写开关，B=110从PC中取数，C=101让PC+1，PC的变化过程如图所示，图为程序计数器的内部构造：



当LDPC有效时74LS161开启计数功能，计数值+1

* **S2：**后续微地址为000010，转到取出指令并进行指令译码（P<1>测试）

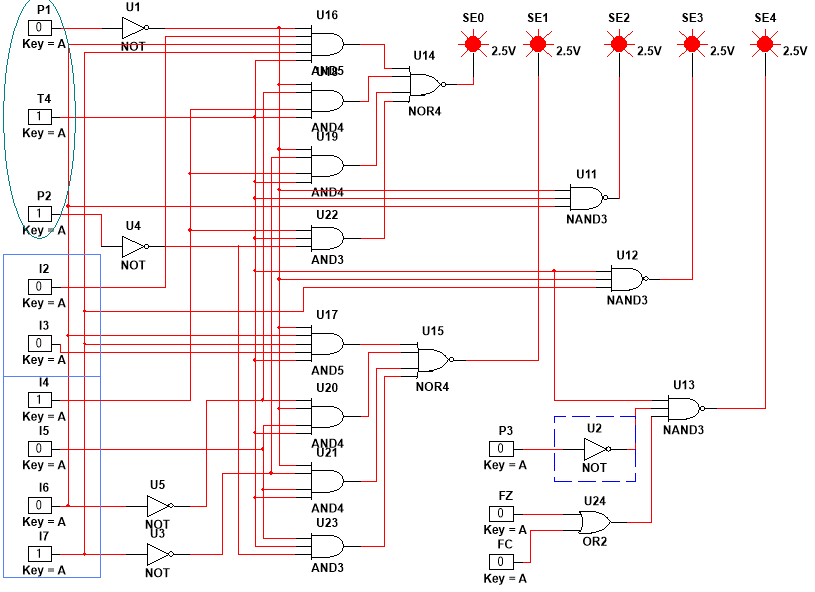


下面是从内存取指令的过程：这种情况下RD为1，IOM为0，代表访存。访存部件的工作原理如图所示，这种情况下只有XMRD=0，代表读内存有效。A=111将数据送入IR

Diagram, schematic

Description automatically generated

接下来是P测试，根据初始微地址30和操作码（这里是0000），得到SE5-SE0=110000，P测试的工作原理如图所示。SE5-SE0连接触发器的置数端，因此经过触发器后这就是修改后的微地址110000：

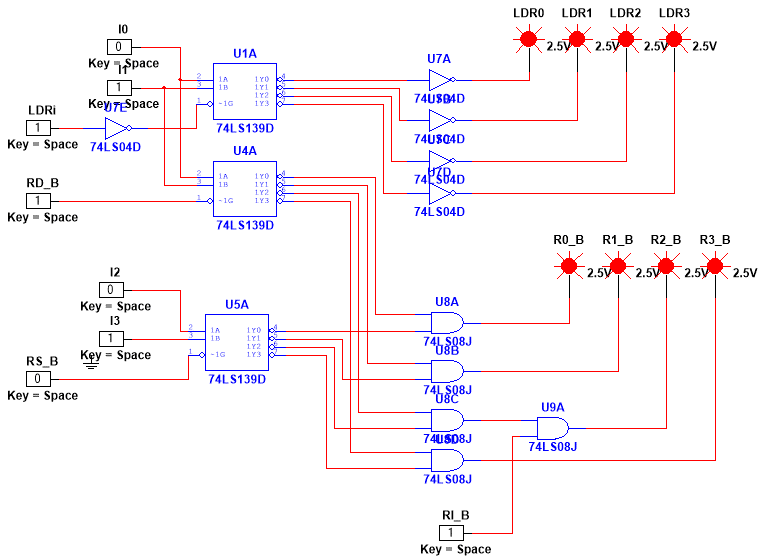


* **S3：**根据微地址110000，进入30区间，执行下面的微指令：

A picture containing diagram

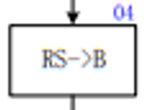
Description automatically generated

RD和RS由下面的逻辑根据机器指令对应字段确定，如图所示：图为根据指令选择寄存器的逻辑图。



A=001打开数据缓冲寄存器A，B=011，RD\_B有效，如上图所示，U4A的74LS139被选中，将I0，I1送入对应的寄存器开关。这里I1，I0=01，因此R1\_B选中，读R1的内容。

* **S4：**后续微地址000100，执行这条微指令：



同理，B=010，RS\_B有效，根据寄存器译码逻辑图，U5A的74LS139被选中，I1，I0=00，R0\_B有效，读R0的数据。

* **S5：**后续微地址000101，执行这条微指令：



S4-S0=1001，让ALU做加法，LDRi代表写目的寄存器，根据寄存器译码逻辑图，选中U1A的74LS139，由于R1，R0=01，因此经过2-4译码后LDR1有效，数据写入RD=R1。

* **S6：**后续微地址000001，回到取指，本次任务结束。

2.5如何对模型机编程

2.5.1微码写入

(1) 手动写入微程序

① 将时序与操作台单元的开关 KK1 置为‘停止’档，KK3 置为‘编程’档，KK4 置为‘控存’档，KK5 置为‘置数’档。

② 使用 CON 单元的 SD05——SD00 给出微地址，IN 单元给出低 8 位应写入的数据，连续两次按动时序与操作台的开关 ST，将 IN 单元的数据写到该单元的低 8 位。

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

④ IN 单元给出中 8 位应写入的数据，连续两次按动时序与操作台的开关 ST，将 IN 单元的数据写到该单元的中 8 位。IN 单元给出高 8 位应写入的数据，连续两次按动时序与操作台的开关 ST，将 IN 单元的数据写到该单元的高 8 位。

⑤ 重复①、②、③、④四步，将所有的微代码写入 2816 芯片中。

(2) 手动校验微程序

① 将时序与操作台单元的开关 KK1 置为‘停止’档，KK3 置为‘校验’档，KK4 置为‘控存’档，KK5 置为‘置数’档。

② 使用 CON 单元的 SD05——SD00 给出微地址，连续两次按动时序与操作台的开关 ST， MC 单元的指数据指示灯 M7——M0 显示该单元的低 8 位。

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

④ 连续两次按动时序与操作台的开关 ST，MC 单元的指数据指示灯 M15——M8 显示该单元的中 8 位，MC 单元的指数据指示灯 M23——M16 显示该单元的高 8 位。

⑤ 重复①、②、③、④四步，完成对微代码的校验。如果校验出微代码写入错误，重新写入、校验，直至确认微指令的输入无误为止。

2.5.2应用程序写入

(1) 手动写入机器程序

① 将时序与操作台单元的开关 KK1 置为‘停止’档，KK3 置为‘编程’档，KK4 置为‘主存’档，KK5 置为‘置数’档。

② 使用 CON 单元的 SD07——SD00 给出地址，IN 单元给出该单元应写入的数据，连续两次按动时序与操作台的开关 ST，将 IN 单元的数据写到该存储器单元。

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

④ IN 单元给出下一地址（地址自动加 1）应写入的数据，连续两次按动时序与操作台的开关 ST，将 IN 单元的数据写到该单元中。然后地址会又自加 1，只需在 IN 单元输入后续地址的数据，连续两次按动时序与操作台的开关 ST，即可完成对该单元的写入。

⑤ 亦可重复①、②两步，将所有机器指令写入主存芯片中。

(2) 手动校验机器程序

①将时序与操作台单元的开关 KK1 置为‘停止’档，KK3 置为‘校验’档，KK4 置为‘主存’档，KK5 置为‘置数’档。

② 使用 CON 单元的 SD07——SD00 给出地址，连续两次按动时序与操作台的开关 ST，CPU 内总线的指数据指示灯 D7——D0 显示该单元的数据。

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

④ 连续两次按动时序与操作台的开关 ST，地址自动加 1，CPU 内总线的指数据指示灯 D7 ——D0 显示该单元的数据。此后每两次按动时序与操作台的开关 ST，地址自动加 1，CPU 内总线的指数据指示灯 D7——D0 显示该单元的数据，继续进行该操作，直至完成校验，如发现错误，则返回写入，然后校验，直至确认输入的所有指令准确无误。

⑤ 亦可重复①、②两步，完成对指令码的校验。如果校验出指令码写入错误，重新写入、 校验，直至确认指令码的输入无误为止。

**3.微码设计**

这部分介绍如何扩展指令以及微码表的具体内容。

3.1新增指令功能及分析

我们决定新增一条异或指令，命令代码1011，助记符是XOR RD RS，含义是(RS) XOR (RD) -> RD。

由于ALU没有直接异或运算，因此必须使用传统的与或非模拟，根据

我们可以得到以下思路：

* S0：RS->A
* S1：A/->A
* S2：RD->B
* S3：AB->RD
* S4：B->A
* S5：A/->A
* S6：RS->B
* S7：AB->A
* S8：RD->B
* S9：A+B->RD

上述过程翻译为微码，就是新增微指令设计

3.2改进后的微码表

我们充分利用可用空间，如表所示，红色为新增内容

| 地址 | 十六进制表示 | 二进制表示 | X CN WR RD IOM | S3-S0 | A 字段 | B 字段 | C 字段 | UA5-UA0 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 00 | 00 00 01 | 00000000 00000000 00000001 | 00000 | 0000 | 000 | 000 | 000 | 000001 |
| 01 | 00 6D 43 | 00000000 01101101 01000011 | 00000 | 0000 | 110 | 110 | 101 | 000011 |
| 02 | 02 12 19 | 00000010 00010010 00011001 | 00000 | 0100 | 001 | 001 | 000 | 011001 |
| 03 | 10 70 70 | 00010000 01110000 01110000 | 00010 | 0000 | 111 | 000 | 001 | 110000 |
| 04 | 00 24 05 | 00000000 00100100 00000101 | 00000 | 0000 | 010 | 010 | 000 | 000101 |
| 05 | 04 B2 01 | 00000100 10110010 00000001 | 00000 | 1001 | 011 | 001 | 000 | 000001 |
| 06 | 00 24 07 | 00000000 00100100 00000111 | 00000 | 0000 | 010 | 010 | 000 | 000111 |
| 07 | 01 32 01 | 00000001 00110010 00000001 | 00000 | 0010 | 011 | 001 | 000 | 000001 |
| 08 | 10 60 09 | 00010000 01100000 00001001 | 00010 | 0000 | 110 | 000 | 000 | 001001 |
| 09 | 18 30 01 | 00011000 00110000 00000001 | 00011 | 0000 | 011 | 000 | 000 | 000001 |
| 0A | 10 60 10 | 00010000 01100000 00010000 | 00010 | 0000 | 110 | 000 | 000 | 010000 |
| 0B | 00 00 01 | 00000000 00000000 00000001 | 00000 | 0000 | 000 | 000 | 000 | 000001 |
| 0C | 10 30 01 | 00010000 00110000 00000001 | 00010 | 0000 | 011 | 000 | 000 | 000001 |
| 0D | 20 06 01 | 00100000 00000110 00000001 | 00100 | 0000 | 000 | 011 | 000 | 000001 |
| 0E | 00 53 41 | 00000000 01010011 01000001 | 00000 | 0000 | 101 | 001 | 101 | 000001 |
| 0F | 00 00 CB | 00000000 00000000 11001011 | 00000 | 0000 | 000 | 000 | 011 | 001011 |
| 10 | 28 04 01 | 00101000 00000100 00000001 | 00101 | 0000 | 000 | 010 | 000 | 000001 |
| 11 | 10 30 01 | 00010000 00110000 00000001 | 00010 | 0000 | 011 | 000 | 000 | 000001 |
| 12 | 06 B2 01 | 00000110 10110010 00000001 | 00000 | 1101 | 011 | 001 | 000 | 000001 |
| 13 | 00 24 14 | 00000000 00100100 00010100 | 00000 | 0000 | 010 | 010 | 000 | 010100 |
| 14 | 05 B2 01 | 00000101 10110010 00000001 | 00000 | 1011 | 011 | 001 | 000 | 000001 |
| 15 | 00 24 16 | 00000000 00100100 00010110 | 00000 | 0000 | 010 | 010 | 000 | 010110 |
| 16 | 01 B2 01 | 00000001 10110010 00000001 | 00000 | 0011 | 011 | 001 | 000 | 000001 |
| 17 | 00 24 18 | 00000000 00100100 00011000 | 00000 | 0000 | 010 | 010 | 000 | 011000 |
| 18 | 02 B2 01 | 00000010 10110010 00000001 | 00000 | 0101 | 011 | 001 | 000 | 000001 |
| 19 | 00 26 1A | 00000000 00100110 00011010 | 00000 | 0000 | 010 | 011 | 000 | 011010 |
| 1A | 01 32 21 | 00000001 00110010 00100001 | 00000 | 0010 | 011 | 001 | 000 | 100001 |
| 1B | 00 53 41 | 00000000 01010011 01000001 | 00000 | 0000 | 101 | 001 | 101 | 000001 |
| 1C | 10 10 1D | 00010000 00010000 00011101 | 00010 | 0000 | 001 | 000 | 000 | 011101 |
| 1D | 10 60 8C | 00010000 01100000 10001100 | 00010 | 0000 | 110 | 000 | 010 | 001100 |
| 1E | 10 60 1F | 00010000 01100000 00011111 | 00010 | 0000 | 110 | 000 | 000 | 011111 |
| 1F | 10 10 20 | 00010000 00010000 00100000 | 00010 | 0000 | 001 | 000 | 000 | 100000 |
| 20 | 10 60 8C | 00010000 01100000 10001100 | 00010 | 0000 | 110 | 000 | 010 | 001100 |
| 21 | 00 92 22 | 00000000 10010010 00100010 | 00000 | 0001 | 001 | 001 | 000 | 100010 |
| 22 | 02 12 23 | 00000010 00010010 00100011 | 00000 | 0100 | 001 | 001 | 000 | 100011 |
| 23 | 00 24 24 | 00000000 00100100 00100100 | 00000 | 0000 | 010 | 010 | 000 | 100100 |
| 24 | 01 12 25 | 00000001 00010010 00100101 | 00000 | 0010 | 001 | 001 | 000 | 100101 |
| 25 | 00 26 26 | 00000000 00100110 00100110 | 00000 | 0000 | 010 | 011 | 000 | 100110 |
| 26 | 01 B2 01 | 00000001 10110010 00000001 | 00000 | 0011 | 011 | 001 | 000 | 000001 |
| 27 |  |  |  |  |  |  |  |  |
| 28 | 10 10 29 | 00010000 00010000 00101001 | 00010 | 0000 | 001 | 000 | 000 | 101001 |
| 29 | 00 28 2A | 00000000 00101000 00101010 | 00000 | 0000 | 010 | 100 | 000 | 101010 |
| 2A | 04 E2 2B | 00000100 11100010 00101011 | 00000 | 1001 | 110 | 001 | 000 | 101011 |
| 2B | 04 92 8C | 00000100 10010010 10001100 | 00000 | 1001 | 001 | 001 | 010 | 001100 |
| 2C | 10 10 2D | 00010000 00010000 00101101 | 00010 | 0000 | 001 | 000 | 000 | 101101 |
| 2D | 00 2C 2E | 00000000 00101100 00101110 | 00000 | 0000 | 010 | 110 | 000 | 101110 |
| 2E | 04 E2 2F | 00000100 11100010 00101111 | 00000 | 1001 | 110 | 001 | 000 | 101111 |
| 2F | 04 92 8C | 00000100 10010010 10001100 | 00000 | 1001 | 001 | 001 | 010 | 001100 |
| 30 | 00 16 04 | 00000000 00010110 00000100 | 00000 | 0000 | 001 | 011 | 000 | 000100 |
| 31 | 00 16 06 | 00000000 00010110 00000110 | 00000 | 0000 | 001 | 011 | 000 | 000110 |
| 32 | 00 6D 48 | 00000000 01101101 01001000 | 00000 | 0000 | 110 | 110 | 101 | 001000 |
| 33 | 00 6D 4A | 00000000 01101101 01001010 | 00000 | 0000 | 110 | 110 | 101 | 001010 |
| 34 | 00 34 01 | 00000000 00110100 00000001 | 00000 | 0000 | 011 | 010 | 000 | 000001 |
| 35 | 00 00 35 | 00000000 00000000 00110101 | 00000 | 0000 | 000 | 000 | 000 | 110101 |
| 36 | 00 6D 51 | 00000000 01101101 01010001 | 00000 | 0000 | 110 | 110 | 101 | 010001 |
| 37 | 00 16 12 | 00000000 00010110 00010010 | 00000 | 0000 | 001 | 011 | 000 | 010010 |
| 38 | 00 16 13 | 00000000 00010110 00010011 | 00000 | 0000 | 001 | 011 | 000 | 010011 |
| 39 | 00 16 15 | 00000000 00010110 00010101 | 00000 | 0000 | 001 | 011 | 000 | 010101 |
| 3A | 00 16 17 | 00000000 00010110 00010111 | 00000 | 0000 | 001 | 011 | 000 | 010111 |
| 3B | 00 14 02 | 00000000 00010100 00000010 | 00000 | 0000 | 001 | 010 | 000 | 000010 |
| 3C | 00 6D 5C | 00000000 01101101 01011100 | 00000 | 0000 | 110 | 110 | 101 | 011100 |
| 3D | 00 6D 5E | 00000000 01101101 01011110 | 00000 | 0000 | 110 | 110 | 101 | 011110 |
| 3E | 00 6D 68 | 00000000 01101101 01101000 | 00000 | 0000 | 110 | 110 | 101 | 101000 |
| 3F | 00 6D 6C | 00000000 01101101 01101100 | 00000 | 0000 | 110 | 110 | 101 | 101100 |

为了联机操作的方便性，这里同时给出用上位机烧写的固件代码

|  |  |
| --- | --- |
| Code Snippet 1 | ASM |
| ; //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  ; // //  ; //文件1 微码表 //  ; // //  ; //By Steve Workshop //  ; // 版权所有，侵权不究 //  ; // //  ; //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  ; //\*\* Start Of MicroController Data \*\*//  $M 00 000001 ; NOP  $M 01 006D43 ; PC->AR, PC 加 1  $M 02 021219  $M 03 107070 ; MEM->IR, P<1>  $M 04 002405 ; RS->B  $M 05 04B201 ; A 加 B->RD  $M 06 002407 ; RS->B  $M 07 013201 ; A 与 B->RD  $M 08 106009 ; MEM->AR  $M 09 183001 ; IO->RD  $M 0A 106010 ; MEM->AR  $M 0B 000001 ; NOP  $M 0C 103001 ; MEM->RD  $M 0D 200601 ; RD->MEM  $M 0E 005341 ; A->PC  $M 0F 0000CB ; NOP, P<3>  $M 10 280401 ; RS->IO  $M 11 103001 ; MEM->RD  $M 12 06B201 ; A 加 1->RD  $M 13 002414 ; RS->B  $M 14 05B201 ; A 减 B->RD  $M 15 002416 ; RS->B  $M 16 01B201 ; A 或 B->RD  $M 17 002418 ; RS->B  $M 18 02B201 ; A 右环移->RD  $M 19 00261A  $M 1A 013221  $M 1B 005341 ; A->PC  $M 1C 10101D ; MEM->A  $M 1D 10608C ; MEM->AR, P<2>  $M 1E 10601F ; MEM->AR  $M 1F 101020 ; MEM->A  $M 20 10608C ; MEM->AR, P<2>  $M 21 009222  $M 22 021223  $M 23 002424  $M 24 011225  $M 25 002626  $M 26 01B201  $M 28 101029 ; MEM->A  $M 29 00282A ; RI->B  $M 2A 04E22B ; A 加 B->AR  $M 2B 04928C ; A 加 B->A, P<2>  $M 2C 10102D ; MEM->A  $M 2D 002C2E ; PC->B  $M 2E 04E22F ; A 加 B->AR  $M 2F 04928C ; A 加 B->A, P<2>  $M 30 001604 ; RD->A  $M 31 001606 ; RD->A  $M 32 006D48 ; PC->AR, PC 加 1  $M 33 006D4A ; PC->AR, PC 加 1  $M 34 003401 ; RS->RD  $M 35 000035 ; NOP  $M 36 006D51 ; PC->AR, PC 加 1  $M 37 001612 ; RD->A  $M 38 001613 ; RD->A  $M 39 001615 ; RD->A  $M 3A 001617 ; RD->A  $M 3B 000001 ; NOP  $M 3C 006D5C ; PC->AR, PC 加 1  $M 3D 006D5E ; PC->AR, PC 加 1  $M 3E 006D68 ; PC->AR, PC 加 1  $M 3F 006D6C ; PC->AR, PC 加 1  ; //\*\* End Of MicroController Data \*\*// | |

**4.应用程序设计**

这部分介绍如何根据机器指令设计应用程序

4.1验证新增指令

为了验证XOR指令是否有效，我们设计了一段应用程序，这个程序的功能是从内存地址60H和61H读入两个数，求异或并显示出来。这段程序的二进制代码和汇编代码分别如下：

二进制代码（前面是地址后面是数据，下同）：

|  |  |
| --- | --- |
| Code Snippet 2 | Raw Binary Data |
| 0000 0000 1100 0000  0000 0001 0110 0000 ; // (60H)  0000 0010 1100 0001  0000 0011 0110 0001 ; //(61H)  0000 0100 1011 0100  0000 0101 0011 0000  0000 0110 0100 0000  0000 0111 0101 0000  0110 0000 XXXX XXXX ; //XXXX YYYY任意  0110 0001 YYYY YYYY | |

汇编代码：

|  |  |
| --- | --- |
| Code Snippet 3 | ASM |
| ; //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  ; // //  ; //文件2 验证异或指令 //  ; // //  ; //By Steve Workshop //  ; // 版权所有，侵权不究 //  ; // //  ; //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  ; //\*\*\*\*\*\* Start Of Main Memory Data \*\*\*\*\*\* //  $P 00 C0 ;  $P 01 60; // LAD M R0 60H  $P 02 C1  $P 03 61; // LAD M R1 61H  $P 04 B4; // XOR R0 R1  $P 05 30  $P 06 40; //OUT R0 40H  $P 07 50; //HLT  $P 60 01;  $P 61 00;  ; //\*\*\*\*\*\* End Of Main Memory Data \*\*\*\*\*\* // | |

4.2例程——N个数求和

这个程序的功能是：从 IN 单元读入一个数据，根据读入数据的低 4位值 X，求 1+2+…+X 的累加和，01H 到 0FH 共 15 个数据存于 60H 到 6EH 单元。

二进制代码：

|  |  |
| --- | --- |
| Code Snippet 4 | Raw Binary Data |
| 0000 0000 0010 0000  0000 0001 0000 0000  0000 0010 0110 0001  0000 0011 0000 1111  0000 0100 0001 0100  0000 0101 0110 0001  0000 0110 0000 0000  0000 0111 1111 0000  0000 1000 0001 0110  0000 1001 0110 0010  0000 1010 0110 0000  0000 1011 1100 1011  0000 1100 0000 0000  0000 1101 0000 1101  0000 1110 0111 0010  0000 1111 0110 0011  0001 0000 0000 0001  0001 0001 1000 1100  0001 0010 1111 0000  0001 0011 0001 0110  0001 0100 1110 0000  0001 0101 0000 1011  0001 0110 1101 0001  0001 0111 0111 0000  0001 1000 0011 0100  0001 1001 0100 0000  0001 1010 1110 0000  0001 1011 0000 0000  0001 1100 0101 0000  0110 0000 0000 0001  0110 0001 0000 0010  0110 0010 0000 0011  0110 0010 0000 0100  0110 0100 0000 0101  0110 0101 0000 0110  0110 0110 0000 0111  0110 0111 0000 1000  0110 1000 0000 1001  0110 1001 0000 1010  0110 1010 0000 1011  0110 1011 0000 1100  0110 1100 0000 1101  0110 1101 0000 1110  0110 1110 0000 1111 | |

汇编代码：

|  |  |
| --- | --- |
| Code Snippet 5 | ASM |
| ; //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  ; // //  ; //文件3 常数求和 //  ; // //  ; //By Steve Workshop //  ; // 版权所有，侵权不究 //  ; // //  ; //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  ; //\*\*\*\*\*\* Start Of Main Memory Data \*\*\*\*\*\* //  $P 00 20 ; START: IN R0,00H  $P 01 00 ; 从 IN 单元读入计数初值  $P 02 61 ; LDI R1,0FH  $P 03 0F ; 立即数 0FH 送 R1  $P 04 14 ; AND R0,R1 得到 R0 低四位  $P 05 61 ; LDI R1,00H  $P 06 00 ; 装入和初值 00H  $P 07 F0 ; BZC RESULT  $P 08 16 ; 计数值为 0 则跳转  $P 09 62 ; LDI R2,60H 读入数据始地址  $P 0A 60  $P 0B CB ; LOOP: LAD R3,[RI],00H 从 MEM 读入数据送 R3，变址寻址，偏移量为 00H  $P 0C 00  $P 0D 0D ; ADD R1,R3 累加求和  $P 0E 72 ; INC RI 变址寄存加 1，指向下一数据  $P 0F 63 ; LDI R3,01H  $P 10 01  $P 11 8C ; SUB R0,R3 装入比较值  $P 12 F0 ; BZC RESULT  $P 13 16 ; 相减为 0，表示求和完毕  $P 14 E0 ; JMP LOOP 未完则继续  $P 15 0B  $P 16 D1 ; RESULT: STA 70H,R1 和存于 MEM 的 70H 单元  $P 17 70  $P 18 34 ; OUT 40H,R1  $P 19 40 ; 和在 OUT 单元显示  $P 1A E0 ; JMP START  $P 1B 00 ; 跳转至 START  $P 1C 50 ; HLT 停机    $P 60 01 ; 数据  $P 61 02  $P 62 03  $P 63 04  $P 64 05  $P 65 06  $P 66 07  $P 67 08  $P 68 09  $P 69 0A  $P 6A 0B  $P 6B 0C  $P 6C 0D  $P 6D 0E  $P 6E 0F  ; //\*\*\*\*\* End Of Main Memory Data \*\*\*\*\*// | |

4.3自主设计——猜数游戏

内置一个8位二进制数（确定的立即数），与用户输入的数据比较，给出大小的提示，直至用户输入正确数值。下面给出二进制代码和汇编代码。

二进制代码：

|  |  |
| --- | --- |
| Code Snippet 6 | Raw Binary Data |
| 0000 0000 0110 0001  0000 0001 XXXX XXXX ; XXXX XXXX 是任意立即数  0000 0010 0010 0000  0000 0011 0000 0000  0000 0100 1011 0100  0000 0101 1111 0000  0000 0110 0000 1101  0000 0111 0110 0011  0000 1000 0000 0000  0000 1001 0011 1100  0000 1010 0100 0000  0000 1011 1110 0000  0000 1100 0000 0010  0000 1101 0110 0011  0000 1110 0000 0001  0000 1111 0011 1100  0001 0000 0100 0000  0001 0001 0101 0000 | |

汇编代码：

|  |  |
| --- | --- |
| Code Snippet 7 | ASM |
| ; //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  ; // //  ; // 文件4 猜数游戏 //  ; // //  ; //By Steve Workshop //  ; // 版权所有，侵权不究 //  ; // //  ; //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  $P 00 61  $P 01 XX; // LDI R1 XX XX是设计好的一个八位二进制数  $P 02 20$P 03 00 ; // (addr0=02) IN R0 00H  $P 04 B4; // XOR R0 R1 这里就用到了新增的指令  $P 05 F0; // BZC addr1  $P 06 0D  $P 07 63  $P 08 00; // LDI R3 00H 0代表失败  $P 09 3C  $P 0A 40; // OUT R3 40H  $P 0B E0  $P 0C 02; //JMP addr0  $P 0D 63  $P 0E 01; //(addr1=0D) LDI R3 01H  $P 0F 3C  $P 10 40; //OUT R3 40H  $P 11 50; //HLT | |

**5.试验心得体会**

通过这次试验，使得我们对计算机的工作原理有了进一步的理解。我们深刻体会到一段程序在计算机内部到底经历了什么过程。设计计算机结构的方法让我们理解了自底向上，先部件后整机，先微观后宏观的思想理念。同时也是对电子技术等基础知识的一次复习提高。对计算机系学生是一次难能可贵的锻炼机会。