**实 验 报 告**

|  |  |
| --- | --- |
| **课程名称：** | **计算机组成与嵌入式系统** |
| **学生姓名：** | **葛子午** |
| **学生学号：** |  |
| **学生专业：** | **软件工程** |
| **开课学期：** | **2024-2025学年第一学期** |

**华南理工大学软件学院**

**2024年10月**

**目 录**

[实验一 进位加法器实验 1](#_Toc16866)

[实验二 运算器实验 10](#_Toc14302)

[实验三 存储器实验 18](#_Toc32626)

[实验四 微程序控制器实验 28](#_Toc20912)

# 进位加法器实验

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **地 点：** | B7楼 | 138房； | **评 分：** |  |
| **实验日期与时间：** | 2024年10月19日 | | **实验教师：** | 王国华 |

**一、实验目的**

（1）了解半加器和全加器的电路结构。

（2）掌握串行进位加法器和并行进位加法器的原理及设计方法。

**二、实验环境**

Proteus 8 Professional软件

**三、实验器件**

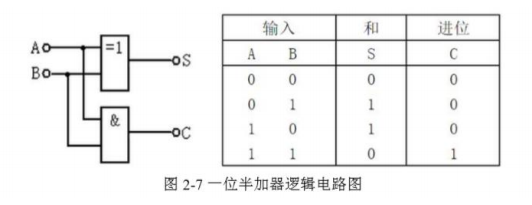
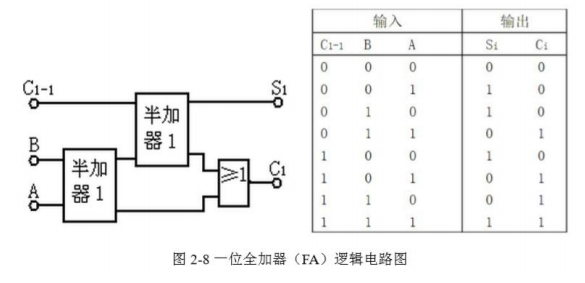
1、2/3/4与门(74LS08/74LS11/74LS21)、非门(74LS04)、或门(74LS32)、异或门(74LS86)等逻辑门。

2、三态门(74LS244)、LED 指示灯及数码显示管。

3、单刀双掷开关(SW SPDT)、拨码开关(DIPSWC 8)。

**四、实验原理**

（1）加法器是最常见的运算逻辑电路，用来执行二进制的加减法运算(减法可以通过补码相加实现)。根据功能的不同，加法器可以分为半加器和全加器(FA)。半加器不考虑低位的进位，只考虑两个二进制数相加的结果和向高位的进位，如图2-7所示;而全加器(FA)是在半加器的基础上又考虑了低位过来的进位信号，如图 2-8所示。



半加器逻辑公式： Si = Ai⊕Bi 且 Ci+1 = Ai·Bi

全加器逻辑公式： Si = Ai⊕Bi ⊕Ci 且 Ci+1=Ai·Bi + (Ai⊕Bi)·Ci

1. 本实验所使用的串行进位加法器，又称为行波进位加法器，其原理图如图 2-10 所示:由若干位全加器 FA串行相连得到，其中低位F的进位输出直接与相邻高位 FA的进位输入相连，加法器的位间进位从低位往高位逐位串行传送。

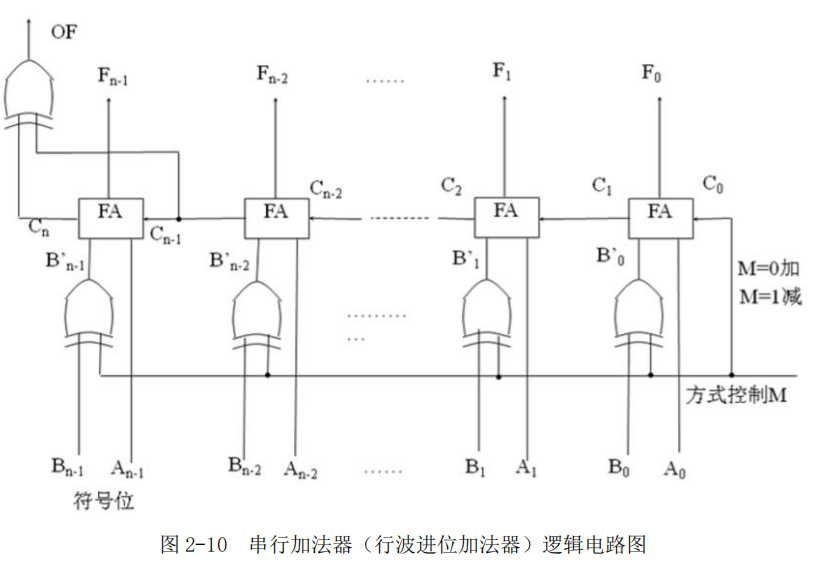
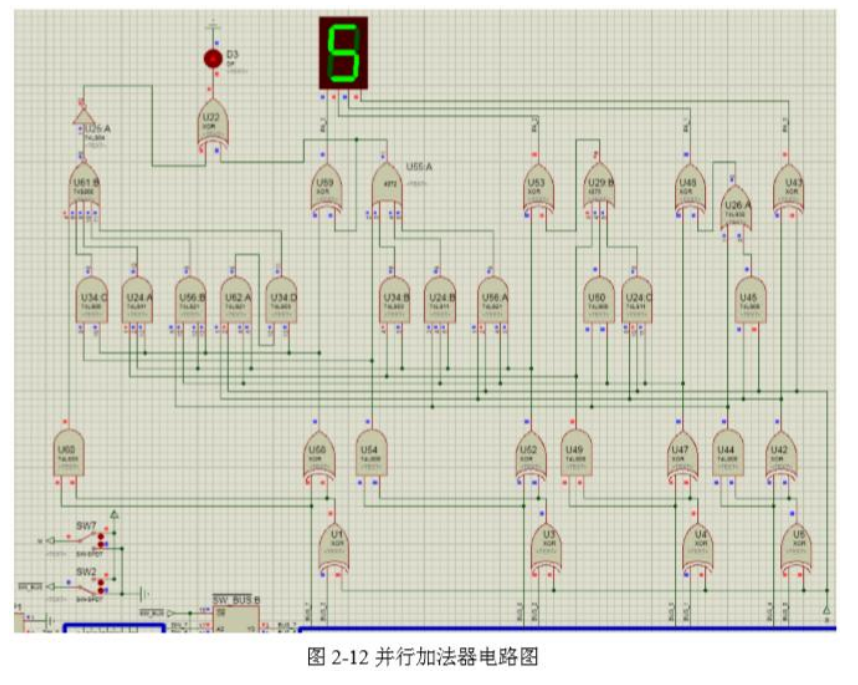


图 2-10 中,n-1 和 Bn-1 是符号位,判溢位 0F=CnФCn-1。FA的运算形式由M决定，当方式控制位 M=0，FA是补码加法器:Bi’=MBi=Bi，C0=0，则[F]补=[A]补+[B]补=[A+B]补;当方式控制位 M=1，FA是补码减法器:Bi’=MBi=Bi，C0 =1，则[F]补=[A]补+[-B]补=[A-B]补; 图 2-11(左)所示是四位串行加法器，由图 2-11(右)所示的一位全加器串行形成，最高位是符号位，数字有效位是三位。M端控制器件做加法(M=0)或减法(M=1)，因为“任意两数之差的补码等于被减数的补码与减数相反数的补码之和”,所以M端的工作原理基于“[Y]补->[-Y]补”求补运算原则:取反再加 1。图 2-11 中执行的运算是:1010+0101=1111.

1. 本实验的并行加法器又称为超前进位加法器,如图 2-12所示:也是由若干位全加器 FA组成，但是每一个FA 所需的低位进位都不依赖其他 FA，而是根据最低进位 C0及各个位的加数、被加数即可同时计算所有进位Cn。因此称之为“并行”加法器

由图 2-12 可以看出，每一个FA所需的低位进位都只跟 Ai、Bi 和 CO 相关，与相邻 FA产生的进位无关。从右到左，随着位数越高，进位 n 生成电路越复杂。图 2-12 中执行的运算是:1010-0101=1010+1011=0101;因为进位导致符号位改变(溢出)，0F指示灯点亮。



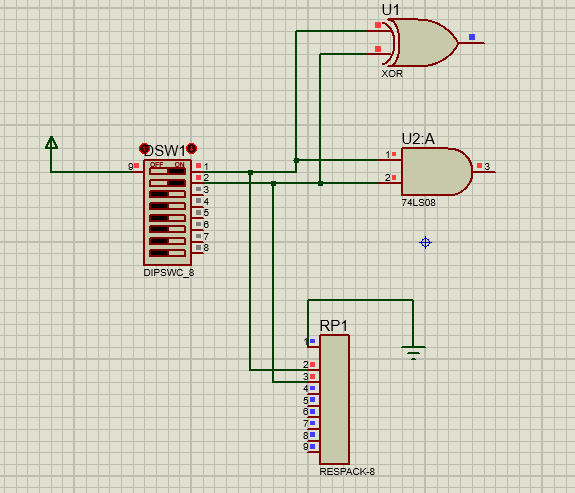
**五、实验步骤（实验电路图，数据记录）**

**（一）根据图2-7和图2-8制作半加器和全加器电路，并验证输入输出与真值表是否一致。**

**1、测试半加器**

1+1，结果0，进位1，正确

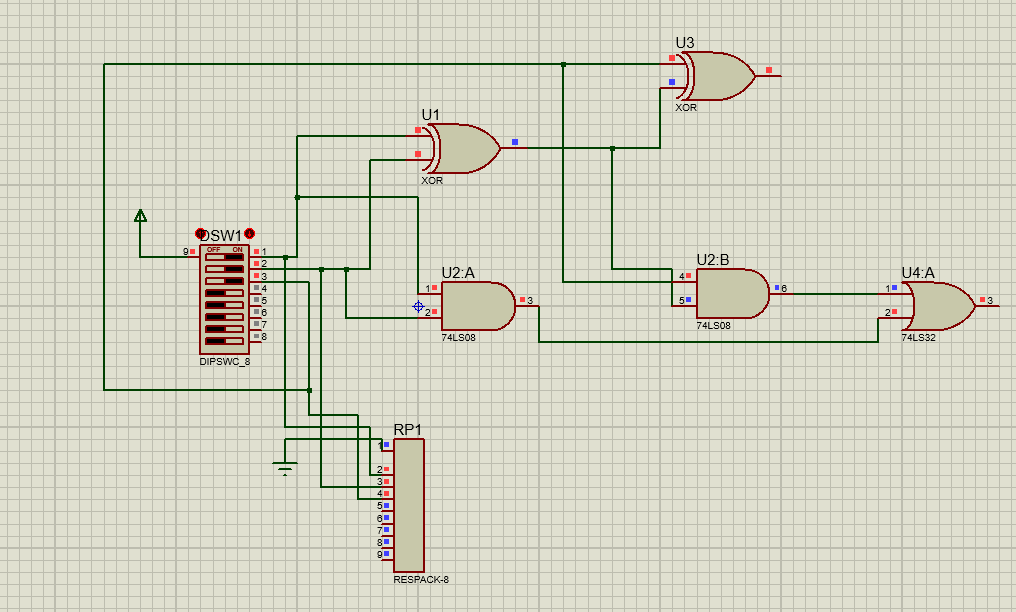
**其余情况经检验均正确**



**2、测试全加器**

1+1 ，上一位进位为1，输出为1，输出进位为1，正确

**其余情况经检验均正确**

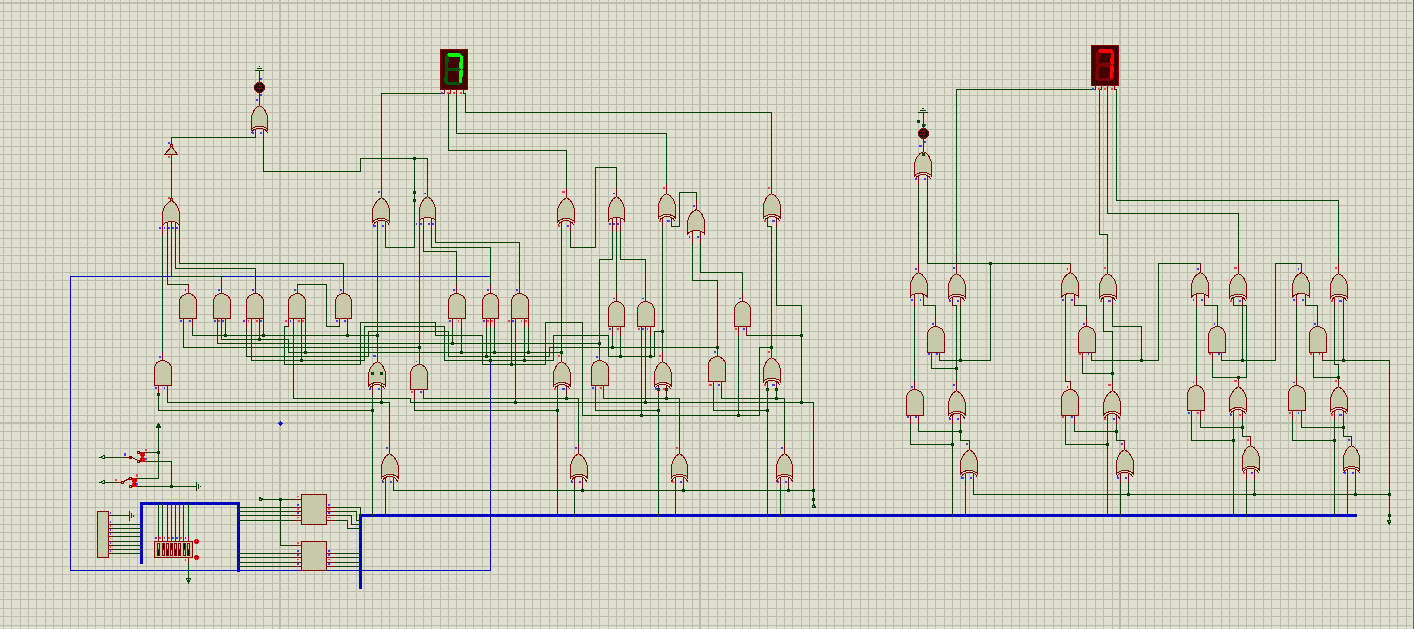
****

**经过与真值表比对，半加器的结果与全加器的结果全部正确。**

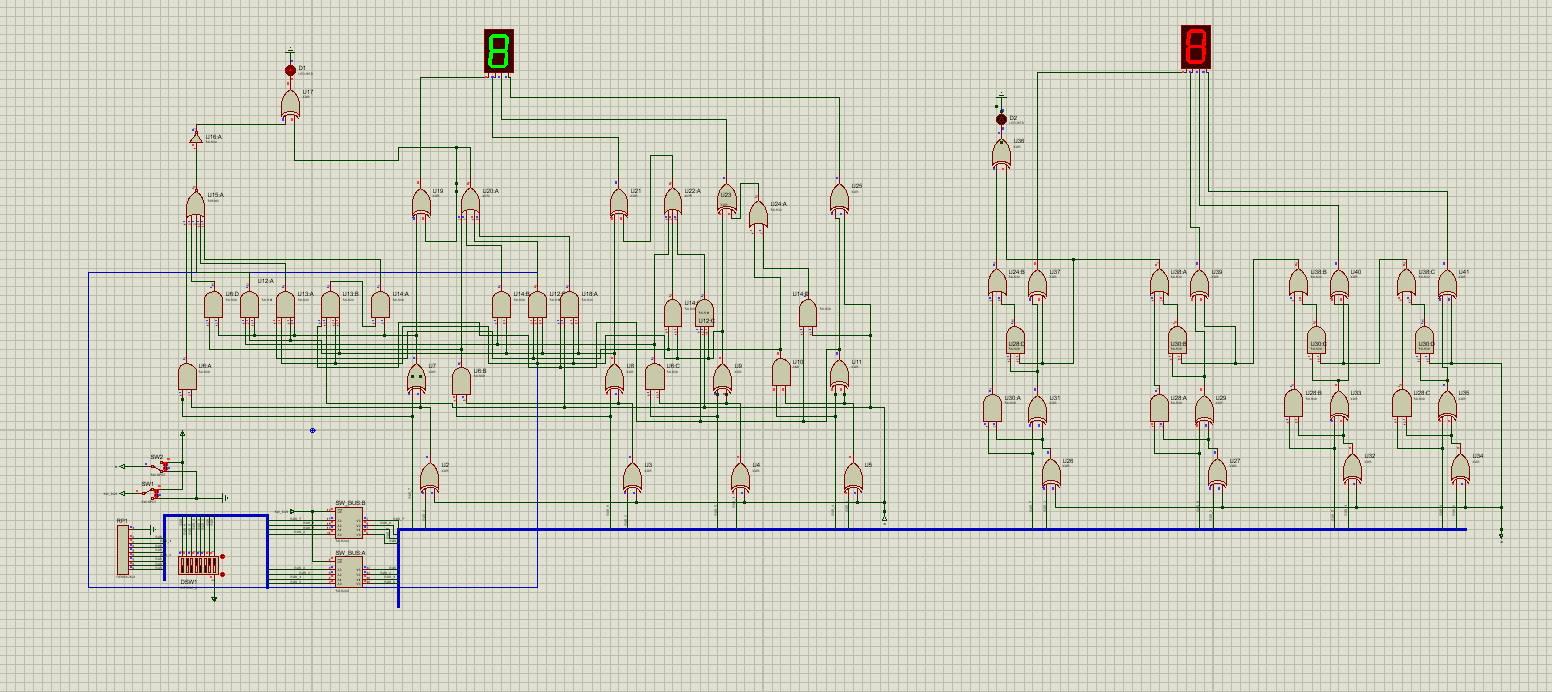
**STEP1：启动仿真，令BUS\_[7..4]=0101，BUS\_[3..0]=0010，M=0，记录并比较串行与并行加法器的运算结果，是否溢出？如果改为 BUS\_[3..0]=0011，结果如何？**

1. 令BUS\_[7..4]=0101，BUS\_[3..0]=0010，M=0

(0101)2+(0010)2=(7)10,，串行与并行加法器的实验结果均为7，正确，且无溢出



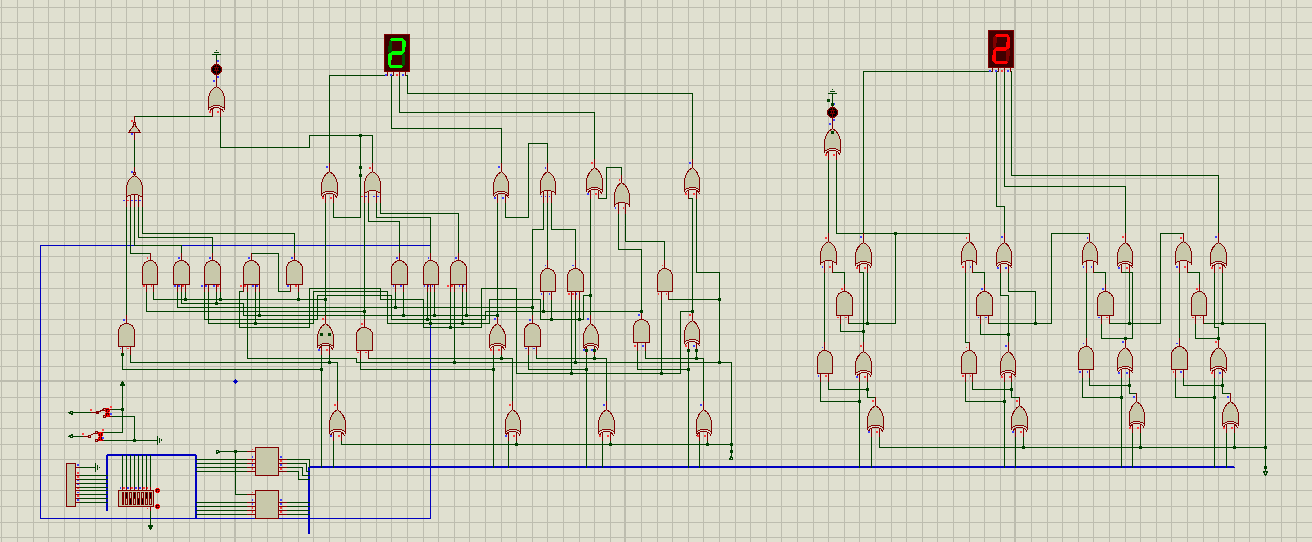
2、改为BUS\_[3..0]=0011



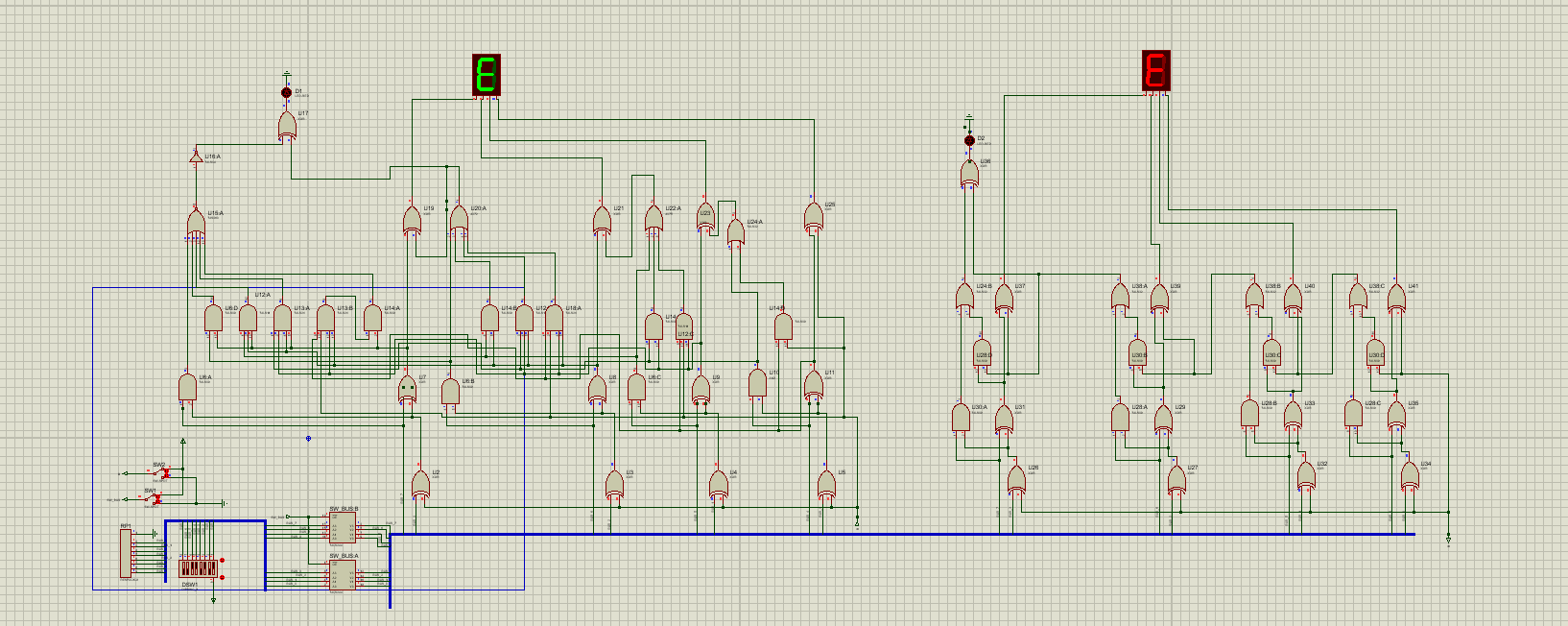
5+3=8，串行与并行加法器的实验结果均为8，正确，且无溢出

**STEP2：启动仿真，令BUS\_[7..4]=0101，BUS\_[3..0]=0011，M=1，记录并比较串行与并行加法器的运算结果，是否溢出？如果运算器的输入改为BUS\_[7..4]=0011，BUS\_[3..0]=0101，M=1不变，结果如何？**

**1、令BUS\_[7..4]=0101，BUS\_[3..0]=0011，M=1**



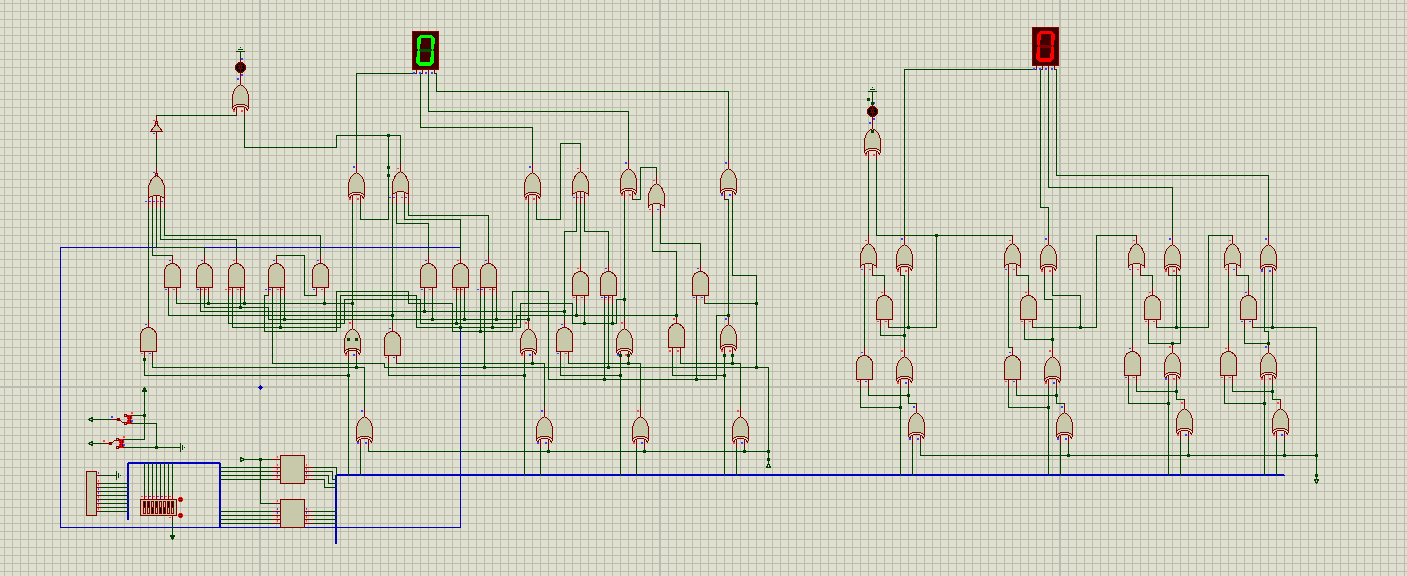
5-3=2，串行和并行的输出结果都为2，无溢出

**2、运算器的输入改为BUS\_[7..4]=0011，BUS\_[3..0]=0101，M=1**

串行和并行输出结果都为E，有溢出，这是因为3-5=-2，而E = 16-2，正是取反加1后的结果，可以视作-2的补码

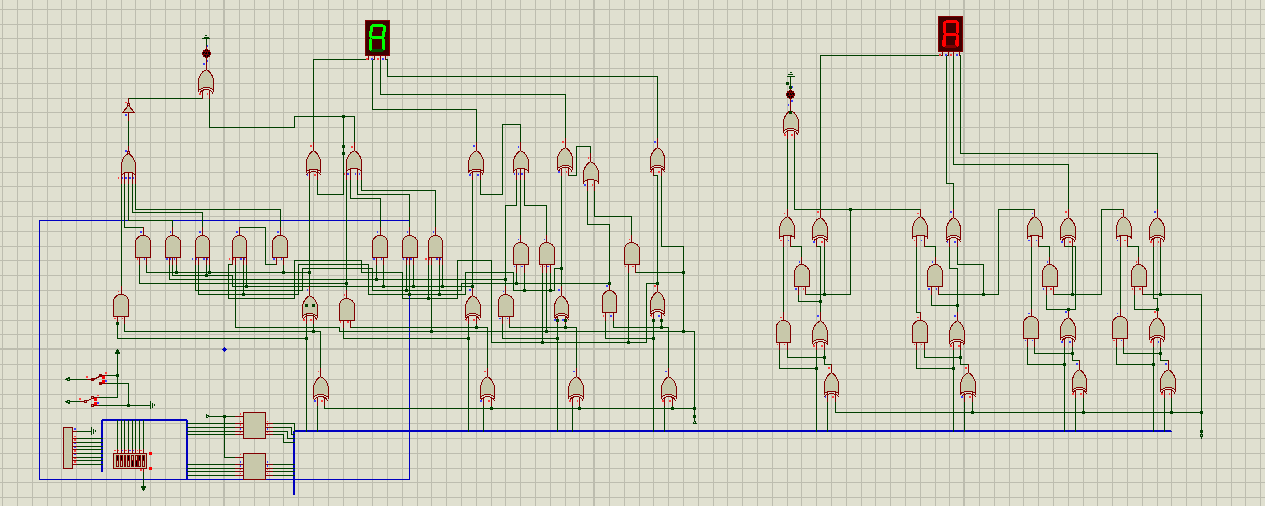
**步骤三：启动仿真，令 BUS\_[7..4]=1101，BUS\_[3..0]=0011，M=0，记录并比较串行与并行加法器的运算结果。是否溢出？如果改为 M=1，结果如何？**

1. 令 BUS\_[7..4]=1101，BUS\_[3..0]=0011，M=0

****

**正数相加，但实验结果为0，有溢出**

1. **改为** M=1

****

**13-3 = 10 = oxA，**串行与并行加法器的实验结果均为A，正确，且无溢出

**五、实验结论、心得体会及思考题**

（一）结论

①通过搭建半加器和全加器电路，并通过仿真验证输入输出结果，可以确认电路的正确性。输出的 Sum 和 Carry 应该与真值表一致，这说明电路的逻辑设计正确，半加器和全加器的运算符合二进制加法的逻辑。

②通过制作半加器和全加器电路，并对串行和并行加法器进行比较，验证了加法器的设计和工作原理。并行加法器表现出更高的速度，但硬件复杂度较高；串行加法器尽管速度较慢，但实现简单。在特定输入条件下可以观察到溢出情况，验证了电路的正确性及溢出处理的必要性。

（二）思考题：

①本实验的运算器是补码运算器和无符号运算器，控制信号M = 1时表示的是减法操作也即补码运算，这说明了该运算器基于补码，但运算器只处理非负整数，不含符号位，所以它是无符号数运算符。

②速度优势：与串行进位加法器相比，并行进位加法器的最大优势在于其运算速度。串行进位加法器需要从最低位开始逐位传递进位，直到最高位，这会导致进位传播延迟。而并行进位加法器通过预先计算进位（使用"先行进位逻辑"等技术），可以同时计算所有位的和及进位，因此速度显著加快。

③所谓“并行”体现在所有位的加法同时进行，而不需要等待进位逐位传递。并行加法器的进位链是同时生成的，不需要依赖于低位的计算结果。这种并行计算使得并行进位加法器适用于对计算速度有高要求的场合。

①实验中运算器是 4 位（没有符号位）的无符号数运算器：

表示的数值范围：0 到 = 15，即 [0, 15]。

②修改为 5 位补码运算器（1 位符号位）

新数值范围：5 位补码数的范围为 ，即 [-16, 15]。

心得体会：

更清楚地理解了二进制加法的逻辑结构。半加器和全加器的设计揭示了如何通过基本逻辑门来实现加法操作，以及进位是如何在电路中传播的。这种动手设计与验证的过程加深了对基本运算电路的理解。

直观地感受到了串行和并行加法器硬件设计上的差异。串行加法器虽然实现简单，但在处理多位加法时速度慢，尤其是随着位数的增加，进位传播的延迟越来越明显。而并行加法器，通过“先行进位”逻辑的实现，显著提升了运算速度，体现出硬件设计的优化对性能的影响。

从基础的电路设计出发，逐步深入到硬件架构优化、数值表示和溢出检测等多个方面，建立了更具象的数字电路基础，为进一步的学习和研究提供了宝贵的经验。

# 运算器实验

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **地 点：** | B7 楼 | 138 房； | **评 分：** |  |
| **实验日期与时间：** | 2024年10月19日 | | **实验教师：** | 王国华 |

**一、实验目的**

1、了解算术逻辑运算器(74LS181)的组成和功能。

2、掌握基本算术和逻辑运算的实现方法

1. **实验内容**

运用算术逻辑运算器 74LS181进行有符号数/无符号数的算术运算和逻辑运算。

**三、实验环境**

Proteus 8 Professional软件

**三、实验器件**

1. 算术逻辑运算器(74LS181)

2. 三态门(74LS244)及寄存器(74LS273、74LS373)。

3. RESPACK-8:排阻

4. DIPSWC 8:拨码开关

5. 7SEG-BCD:显示一位数字

6. 7SEG-BCD-GRN:显示一位数字

7. 4078:8输入端的或门

8. 74LS04:反相器，非门

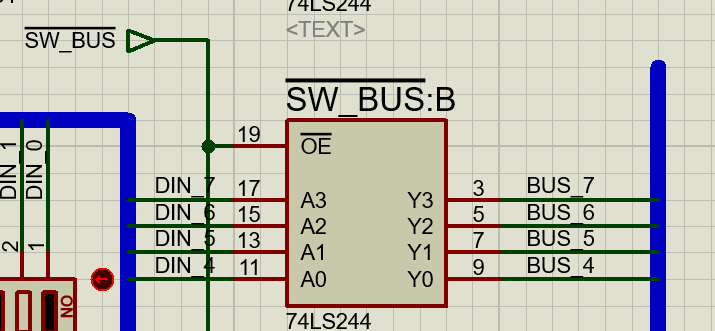
9. BUS:总线

10. SW-SPDT:单刀双掷开关10.

11. LED-GREEN:用来表示 ZF/CF/SF 的值

12. LED-RED:用来表示 ZF/CF/SF 的值

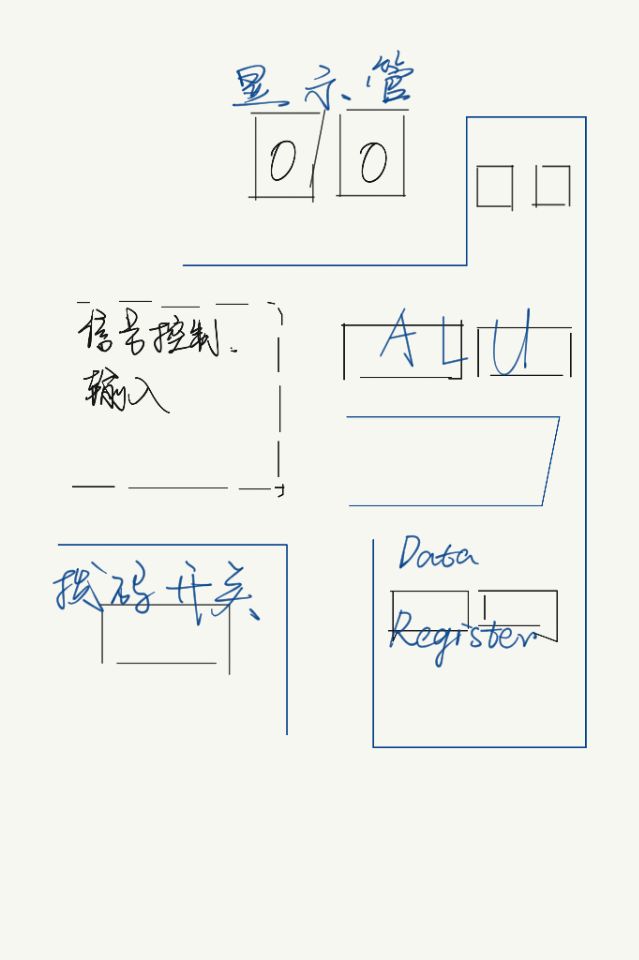
13. LED-YELLOW:用来表示 ZF/CF/SF 的值

1. 74LS181算术逻辑运算器，一个4位的ALU（算术逻辑单元），共有S3,S2,S1,S0和M,CN六个控制信号，其中M信号决定ALU进行算术运算还是逻辑运算，而CN表示运算是否有进位。因此74LS181一共支持进行16种算术运算和16种逻辑运算。
2. 74LS273数据锁存器带有清除端的8D（8输入）触发器，只有在清除端保持高电平时，才具有锁存功能，锁存控制端采用上升沿锁存。在实验一中主要用于作为缓存器保存从拨码开关通过bus传来的数据。
3. 76LS244是一个3态8位缓冲器，一般用作总线驱动器。由于它不具有锁存功能，故只能用作缓冲器，在实验一中用信号控制拨码开关的数据是否能够传入总线上。
4. **实验原理**

本实验的算术逻辑运算器电路:输入单元(拨码开关)用来提供参与运算的数据，输出单元(数码显示管 7SEG-BCD/7SEG-BCD-GRN)显示总线 BUS 的内容。运算器则由两个集成电路芯片 74LS181 以串行进位形式构成8位运算器(ALU):ALU L4B 为低4 位运算芯片， ALU H4B 为高 4 位运算芯片。ALU L4B 的进位输出端 CN+4 与ALU H4B 的进位输入端 CN 相连，使低4位运算产生的进位送进高 4位运算中。ALU L4B 的进位输入端 CN 连接到外来进位端开关 ALU CN，ALU H4B 的进位输出端CN+4 经过反相器 74LS04，通过三态门接到溢出标志位 CF 指示灯(CF=1，即 ALU 运算结果溢出)。

ALU 有三个标志位:溢出标志位 CF(最高位产生进位)，零标志位 ZF(ZF=1,ALU 运算结果为 0)和符号标志位 SF(SF=1，运算结果为负数;SF=0，运算结果为正数或 0)。

ALU 的工作方式可通过设置两个 74LS181 芯片的控制信号(ALU S0、S1、S2、S3、M 及 CN)来实现,其逻辑功能表如表 2-1 所示，表中“A”和“B”分别表示参与运算的两个数，“+”表示逻辑或，“加”表示算术求和。

对实验电路各部分的原理简单描述

实验电路的信号生成区通过手动拨码确定电路的控制信号的01，通过电路信号可以实现时序信号的生成、拨码开关是否能够通过缓冲器打入总线、ALU元件是否使能（）等。

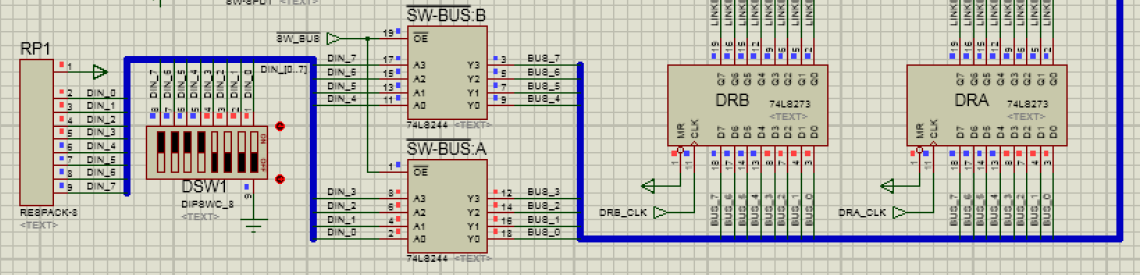
拨码开关分为高四位和低四位，分别传入DRB和DBA，也即我图中Data Register区，在时钟的上升沿会将输入信号传入ALU的输入端中。

显示管显示总线BUS的内容。

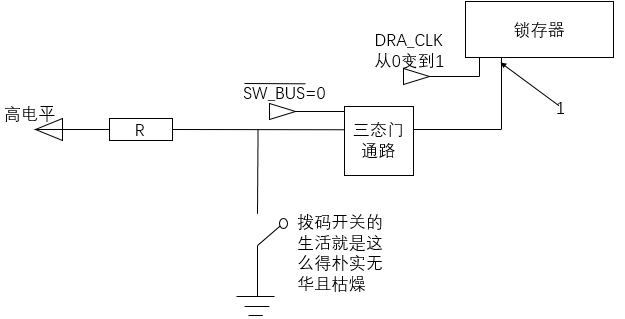
ALU的工作方式是通过以下方式实现的：通过设置两个 74LS181 芯片的控制信号（ALU\_S0、S1、S2、S3、 M 及 CN）来实现, 其逻辑功能表如下表 2-8 所示，表中“A”和“B”分别表示参与运算的两个数，“+”表示逻辑或，“加”表示算术求和。



* 1. 输入的数据是如何产生并锁存的？

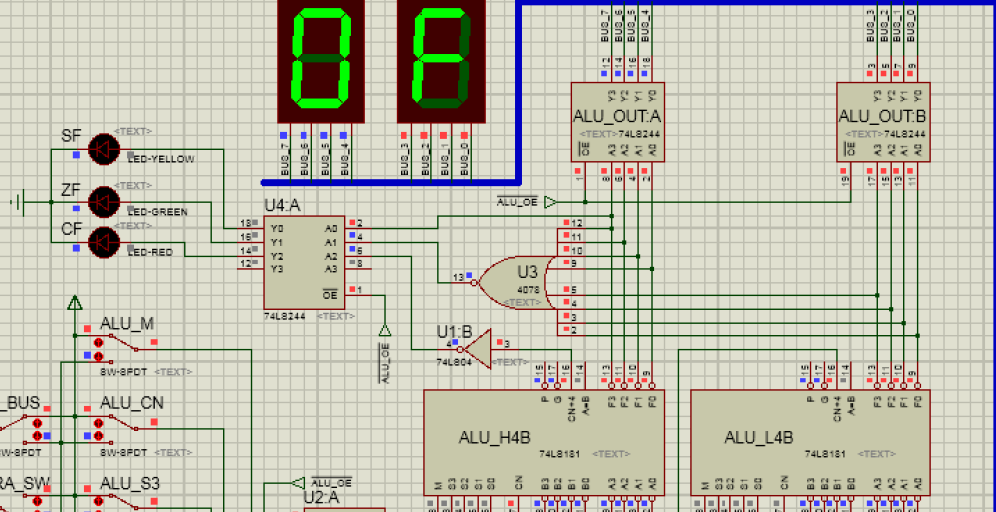


一位二进制数的产生与锁存如下图所示：



正常导通的电路时高电平，三态门导通后锁存器输入端为1，但如果拨码开关将电阻R接地，导致电阻分走绝大部分的电压，这样一来锁存器的输入端就是0输入。

* 1. 输出的数据还有标志位是如何产生的？



输出F=F3H4BF2H4BF1H4BF0H4BF3L4BF2L4BF1L4BF0L4B

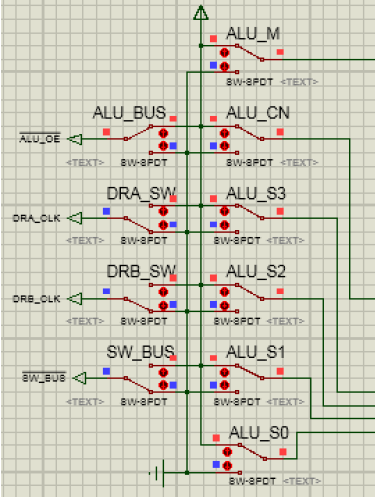
“+”表示逻辑或，“加”表示算术求和

SF=F3H4B

ZF=(F3H4B+F2H4B+F1H4B+F0H4B+F3L4B+F2L4B+F1L4B+F0L4B)取反

CF=CN+4取反

* 1. 各控制信号的作用？



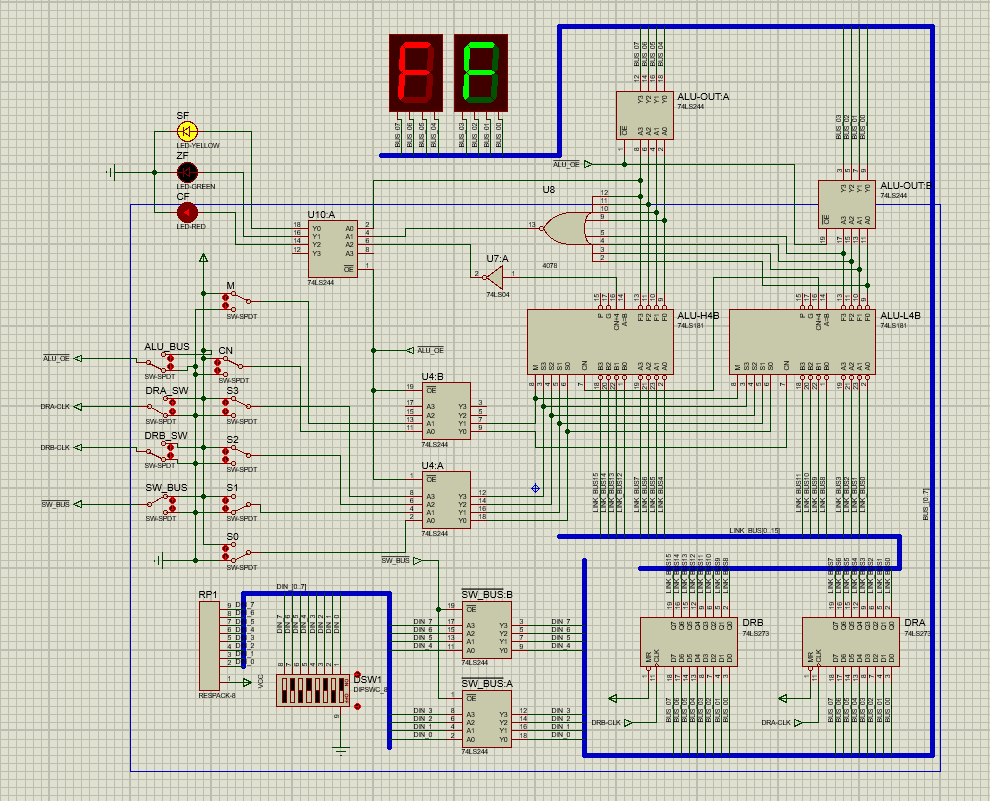
ALU的使能信号（低电平有效）

三态门的使能信号（低电平有效）

  Data Register的时钟信号，上边沿有效

1. **实验步骤（实验电路图，数据记录）**

**实验电路图**

**数据记录**

实验结果记录表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DRA | DRB | S3 | S2 | S1 | S0 | M=0（算术运算） | | 标志位 CF/ZF/SF | | M=1  逻辑运算 |
| CN=1  无进位 | CN=0  有进位 | CN=1  无进位 | CN=0  有进位 |
| 0xAA | 0x55 | 0 | 0 | 0 | 0 | F= AA | F= AB | SF | SF | F= 55 |
|  |  | 0 | 0 | 0 | 1 | F= FF | F= 00 | SF | CF,ZF | F= 00 |
|  |  | 0 | 0 | 1 | 0 | F= AA | F= AB | SF | SF, | F= 55 |
|  |  | 0 | 0 | 1 | 1 | F= FF | F= 00 | SF | ZF,CF | F= 00 |
|  |  | 0 | 1 | 0 | 0 | F= 54 | F= 55 | CF | CF | F= FF |
|  |  | 0 | 1 | 0 | 1 | F= A9 | F= AA | SF,CF | SF,CF | F= AA |
|  |  | 0 | 1 | 1 | 0 | F= 54 | F= 55 | CF | CF | F= FF |
|  |  | 0 | 1 | 1 | 1 | F= A9 | F= AA | SF,CF | SF,CF | F= AA |
|  |  | 1 | 0 | 0 | 0 | F= AA | F= AB | SF | SF | F= 55 |
|  |  | 1 | 0 | 0 | 1 | F= FF | F= 00 | SF | CF,ZF | F= 00 |
|  |  | 1 | 0 | 1 | 0 | F= AA | F= AB | SF | SF | F= 55 |
|  |  | 1 | 0 | 1 | 1 | F= FF | F= 00 | SF | ZF,CF | F= 00 |
|  |  | 1 | 1 | 0 | 0 | F= 54 | F= 55 | CF | CF | F= FF |
|  |  | 1 | 1 | 0 | 1 | F= A9 | F= AA | SF,CF | SF,CF | F= AA |
|  |  | 1 | 1 | 1 | 0 | F= 54 | F= 55 | CF | CF | F= FF |
|  |  | 1 | 1 | 1 | 1 | F= A9 | F= AA | SF,CF | SF,CF, | F= AA |

**五、实验结论、思考题及心得体会**

（一）结论

1、通过实验，可以清楚地观察到74LS181运算器如何在不同的模式下进行有符号数和无符号数的运算。但它无法自动区分有符号数和无符号数运算，用户需要根据数据类型自行解释运算结果。这表明运算器的功能在实际应用中具有灵活性，但也需要我们在操作时对数据的表示方式有清楚的了解。

2、 在进行无符号数运算时，运算结果与标志位的解读方式和有符号数运算有显著差异。无符号数运算的溢出由进位标志位 (CF) 来判断，而有符号数运算需要注意符号标志位 (SF) 和溢出标志位 (OF)。

3、虽然 74LS181 提供了基本的运算能力，但它的 4 位限制需要我们进一步思考如何扩展运算器的能力。通过级联多个 74LS181 单元或添加其他辅助电路，我们可以处理更多位的数据，从而满足更复杂的运算需求。

（二）思考题：

**1、74LS181 运算器可以区分有符号数运算和无符号数运算么？可以执行无符号数的加法和减法运算么？对于有符号数的运算，74LS181 运算器是补码运算器还是原码运算器？**

答：无法区分有符号数运算和无符号数运算；因为它本质上是一个通用的算术逻辑单元 (ALU)，对数值本身的含义没有认知。它只是基于二进制进行加法、减法或逻辑运算。

可以执行无符号数的加法和减法运算；

是补码运算器。

**2、在 74LS181 运算器的通路中，输入锁存器 DRA、DRB 的作用是什么？运算结果输出端连接的 74LS244 缓冲器的作用是什么？假设去掉其中一个输入锁存器，使得 74LS181 的输入直连总线，运算器还能正常工作么？假设去掉输出端 74LS244 缓冲器，使得74LS181 运算器的输出直连总线，运算器还能正常工作么？**

答：**DRA 和 DRB** 的作用是分别暂存输入的两个数值，避免在运算过程中输入信号发生变化。通过锁存器锁定输入，运算器能够稳定地进行计算。若去掉其中一个锁存器，输入直连总线，74LS181 运算器可能无法正常工作，因为输入数据会不稳定，特别是总线数据变化时，可能导致运算过程出错或运算结果不正确。

74LS244可以通过把一个引脚置高后，让输出保持现有的状态，直到把该引脚清0后才能继续变化；因此缓冲器的作用是避免总线上的数据发生冲突，用于控制总线上的数据的输出，

不能，两个输入的信号在总线上会产生冲突；

不能，输出的信号会与输入的信号产生冲突。

**3、当 74LS181 运算器进行无符号数运算的过程中，运算结果的标志位 SF 有无意义？在有符号数运算过程中，标志位 CF 的含义是一致的么？**

答：无符号运算中SF无意义；在无符号数运算中，所有的位都被看作数据位，不存在符号位，因此 SF 在无符号数运算时没有实际意义。

有符号和无符号运算中意义不一致，无符号的含义是表示进位（Carry），用于检测加法中最高位是否有进位，或减法中是否有借位。因此，CF 对判断无符号数溢出至关重要。而在有符号运算中CF并没有有效的含义。因为最高位代表符号，而符号位的进位（或者借位）对结果的数值并没有直接影响。在有符号运算中，溢出的检测更多依赖于其他标志位，例如溢出标志位 (OF)，它反映符号位是否出现不一致的进位。

心得体会：

加深了对 74LS181 算术逻辑单元的理论理解，也通过实际操作掌握了如何进行运算、观察运算结果，并分析标志位对结果的影响，更全面地了解了 74LS181 运算器的功能与应用，掌握了数字电路中基本的运算方式。同时，认识到了运算器设计中的灵活性与局限性，并通过动手实践提高了硬件设计与分析能力。

# 存储器实验

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **地 点：** | B7楼 | 房； | **评 分：** |  |
| **实验日期与时间：** | 2024年11月30日星期六上午 | | **实验教师：** | 王国华 |

**一、实验目的**

1、了解静态随机存储器 RAM 和只读存储器 ROM 的工作特性及读写方法。

2、掌握存储器与总线的连接及存储器地址空间映射的原理。

**二、实验内容**

设计一个 8 位字长的存储器通路，包括 ROM 和 RAM 两个地址相互独立的存储

器，实现对 ROM 和 RAM 存储器的数据读写操作及数据成批导入 ROM 的操作。

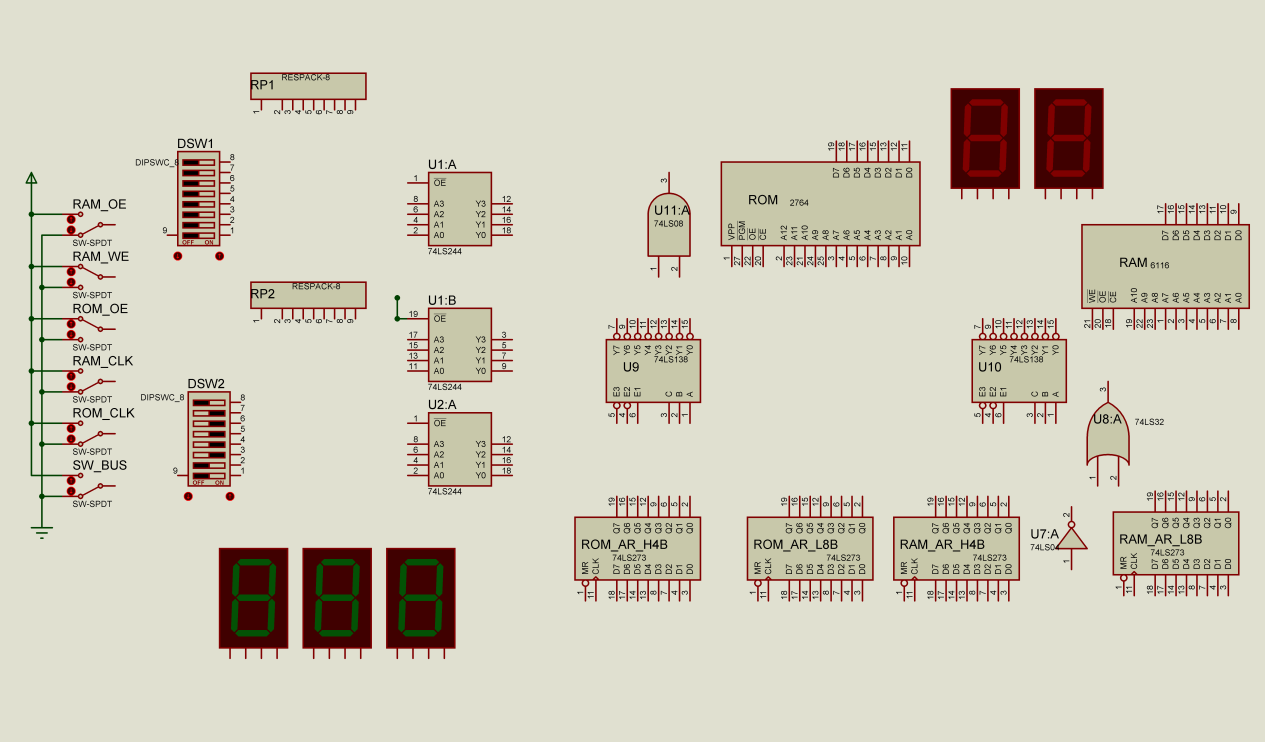
**三、实验器件**

1、只读存储器 2764 及静态随机存储器 6116。

2、三态门（74LS244）、寄存器（74LS273）及 3-8 译码器(74LS138)。

**四、实验原理**

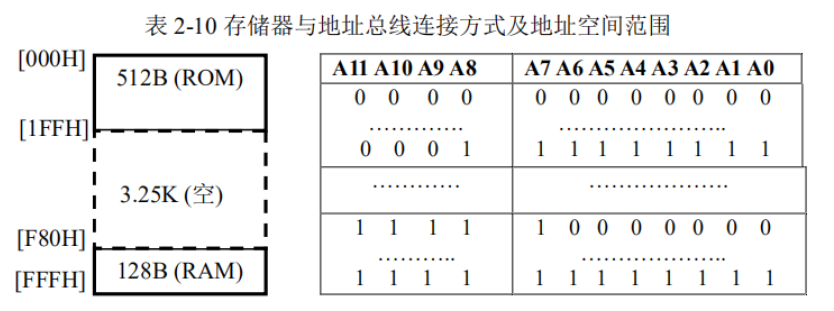
实验器件图如下所示：由地址输入单元、存储器及地址选择电路组成。

存储器通路共有两条总线：12 位地址总线 和 8 位数据总线。下图左边是拨码开关构成的 12 位地址输入端连在地址总线上，通过三个绿色数码管输出显示 12 地址信息。 右边则是存储器 ROM、RAM 及其地址选择电路。ROM 和 RAM 存储器内部有三态 门结构，其数据输出端直接连在数据总线上，通过两个红色数码管显示 8 位数据信息。

1. 存储器类型及特点

ROM（只读存储器）容量较大，掉电数据不丢失，但只能读不能写，本实验使用型号为2764（8K×8位），其数据线D0 - D7接数据总线，地址线A0 - A8由地址锁存器74LS273给出，其余地址线A9 - A12接地，有片选和读两个控制端，均为低电平有效。

RAM（随机存取存储器）容量较小，掉电数据丢失，可读可写，本实验使用型号为6116（2K×8位），数据线D0 - D7接数据总线，地址线A0 - A7由地址锁存器74LS273给出，其余地址线A8 - A10接地，有片选、读、写三个控制端，均为低电平有效。



2. 存储器通路

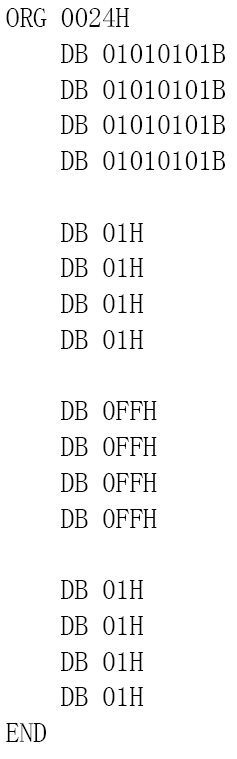
有12位地址总线和8位数据总线。地址输入端由拨码开关构成，通过三个绿色数码管显示12位地址信息；存储器及其地址选择电路在右边，ROM和RAM的数据输出端直接连数据总线，通过两个红色数码管显示8位数据信息。

地址锁存器ROM\_AR和RAM\_AR的输入连接地址总线0 - 11，在CLK端上升沿跳变时锁存地址信号，选中存储单元后可稳定读写。地址总线的连接方式决定存储器地址空间映射关系，本实验将最低512地址分配给ROM（000H - 1FFH），最高128地址给RAM（F80H - FFFH）。



地址总线分为低位和高位地址线，低位与芯片地址线共用，高位产生片选信号（片间地址译码）。本实验中低8位地址线0 - 7分别与ROM和RAM芯片的地址线A0 - A7共用，高4位地址线8 - 11通过两个3 - 8译码器把地址空间分为16个部分，ROM芯片片选由低位3 - 8译码器最低2个部分片选信号的片选电路形成，RAM芯片片选由高位3 - 8译码器74LS138的最高1个部分片选信号与RAM的A7地址线的片选电路形成，片选电路逻辑组合不唯一。



3. ROM批量导入数据技巧

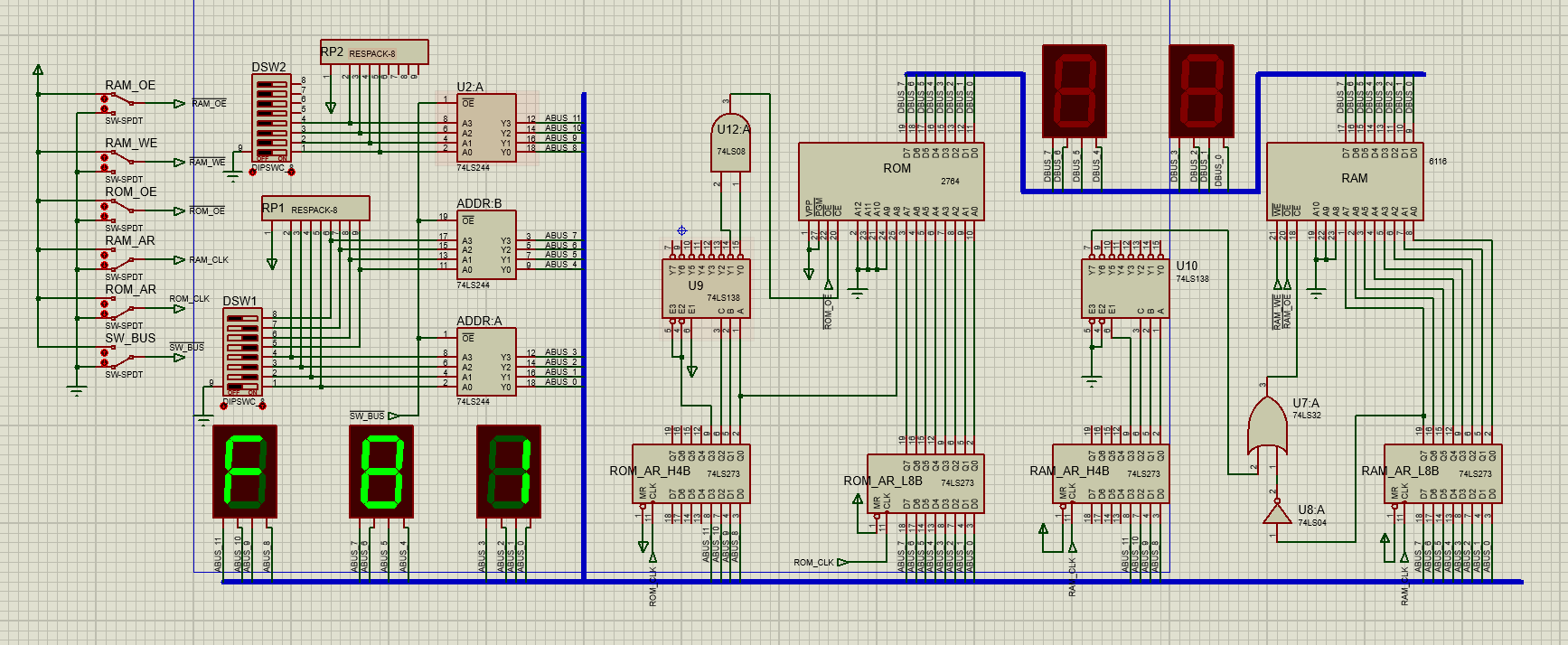
通过Proteus的8051汇编器中的伪汇编指令实现，具体操作包括新建.asm文件并按格式输入数据（用ORG规定首地址，DB表示存储单元数据，注意数据段清零等），在Proteus中进行一系列操作如创建项目、添加文件、编译等，最后将编译生成的hex文件加载到ROM芯片中，启动仿真后可查看存储器内容。

需要烧入的数据如右图：

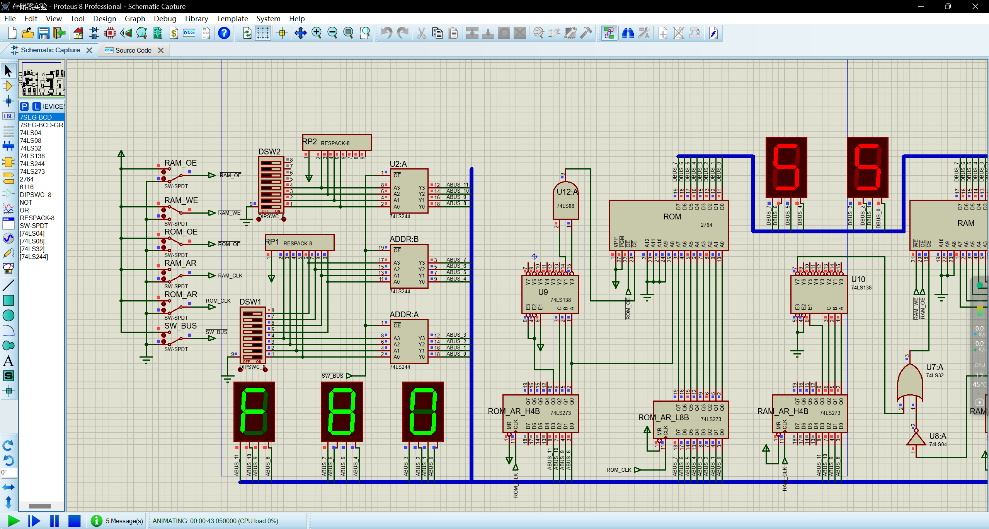
**五．实验步骤**

1. 完成电路图，依照“ROM批量导入数据”方法，加载.asm文件编译的hex二进制文件到ROM芯片2764，并查看ROM烧写的数据段是否正确。
2. 启动仿真前，令；启动仿真后，令，手动拨码开关输入024H到地址总线ABUS\_[0..11]（绿色数码管显示）。
3. 令地址锁存信号ROM\_CLK上升沿跳变“0->1”，将地址总线上的024H打入地址锁存器ROM\_AR；令，使ROM存储器2764输出，在数据总线（红色数码管显示）上查看存储单元[024H]读出的内容。
4. 手动拨码开关输入F80H，令地址锁存信号RAM\_CLK上升沿跳变“0->1”，向地址锁存器RAM\_AR打入地址F80H；令，使RAM存储器6116输入，把存储单元[024H]的内容写入存储单元[F80H]。再令，结束对RAM存储器的写入操作。
5. 令（禁止ROM存储器2764输出）且，（允许RAM存储器6116输出），在数据总线上观察存储单元[F80H]写入内容是否正确。
6. 按照上述操作，把ROM存储器单元[024H]、[028H]、[02CH]、[030H]的内容依次写入RAM存储器单元[F80H]、[F81H]、[F82H]、[F83H]，查看写入RAM的数据是否正确。

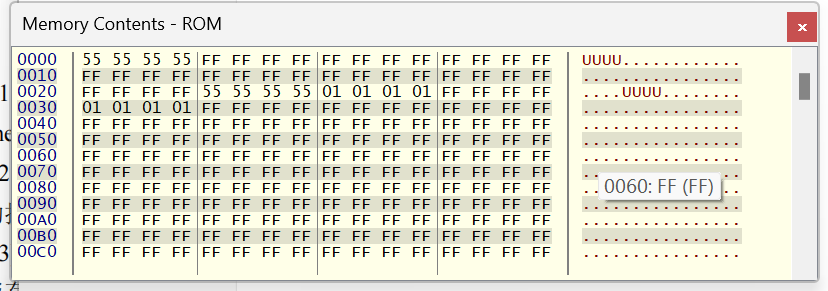
**六．实验结果**

实验图连线如下：

第一轮

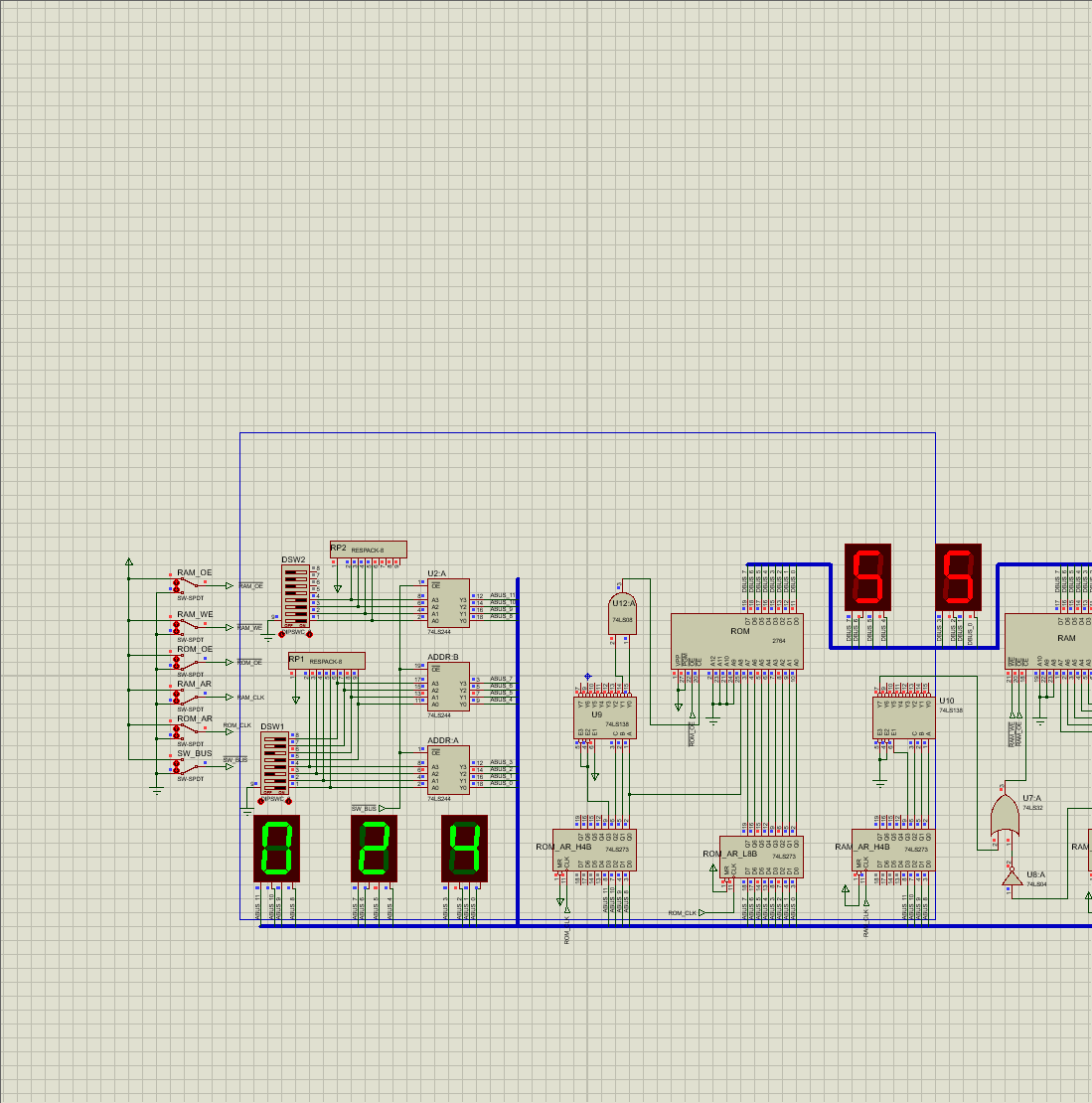
把ROM的[024H]数据写入RAM的[F80H]

烧制后用菜单栏Debug查看0024地址开始的内容，应该是从55开始



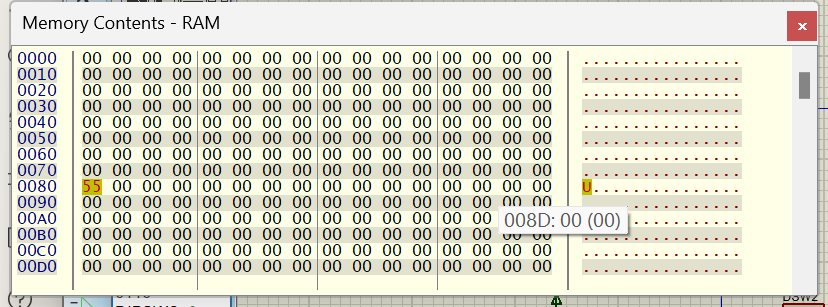
第二步的ABUS\_[0..11]在电路图中间，拨动观察下方调整结果即可

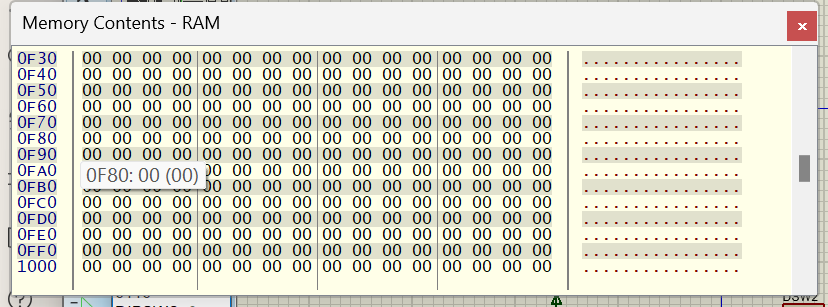
第三步，地址总线上的024H打入地址锁存器，然后再让ROM输出数据，此时在红色数码管上便看到了存储单元[024H]中的内容，与上图ROM中内容一致，55



第四步，再次调整地址线，锁定RAM中的地址并让RAM输入，数据便完成了从[024H]的位置写入到了[F80H]中

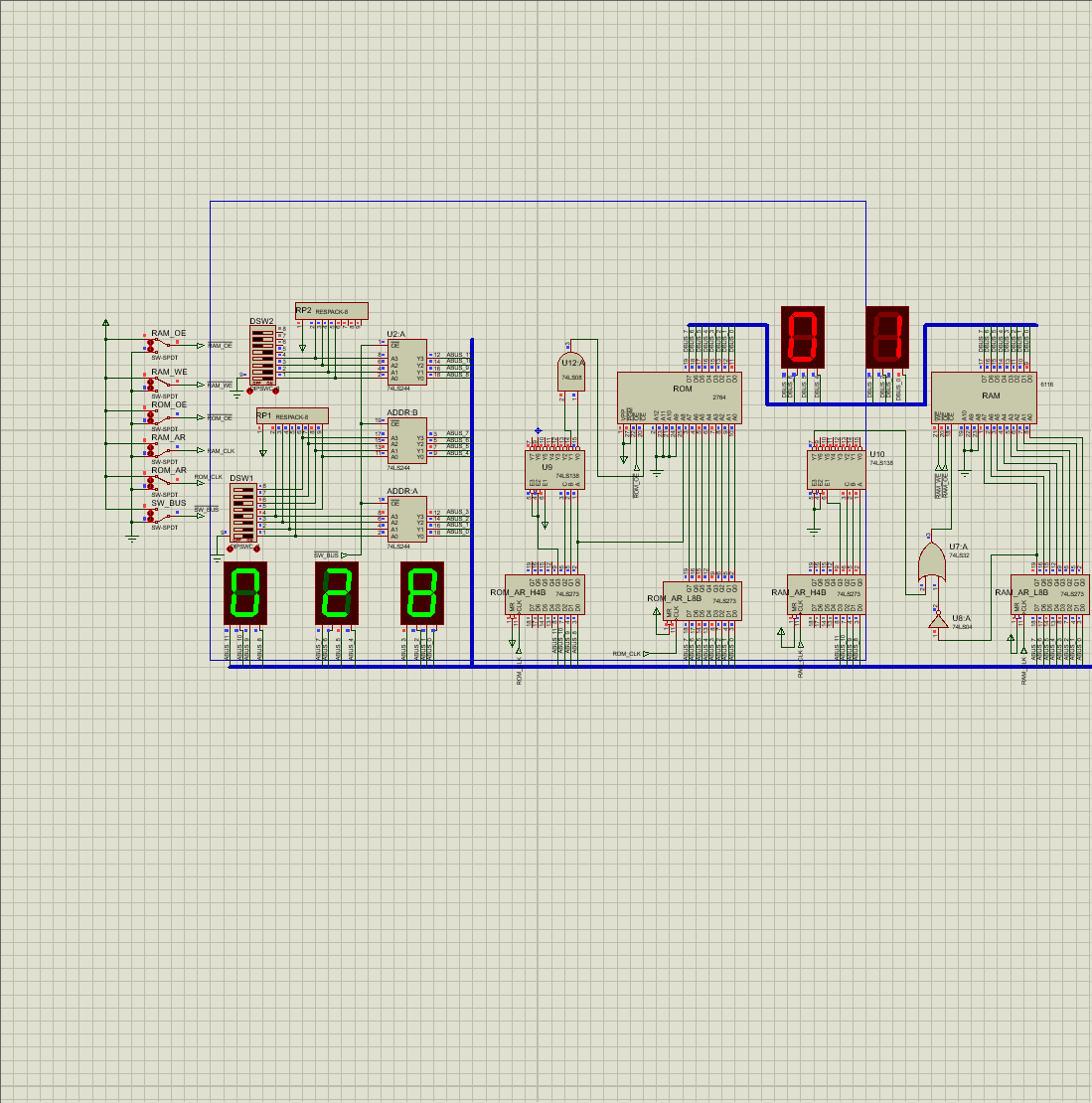
第五步，再次调整电路信号，禁止ROM输出，允许RAM输出，观察存储单元[F80H]位置写入的内容，点击RAM中的信息，看到了[080H]中写入了55





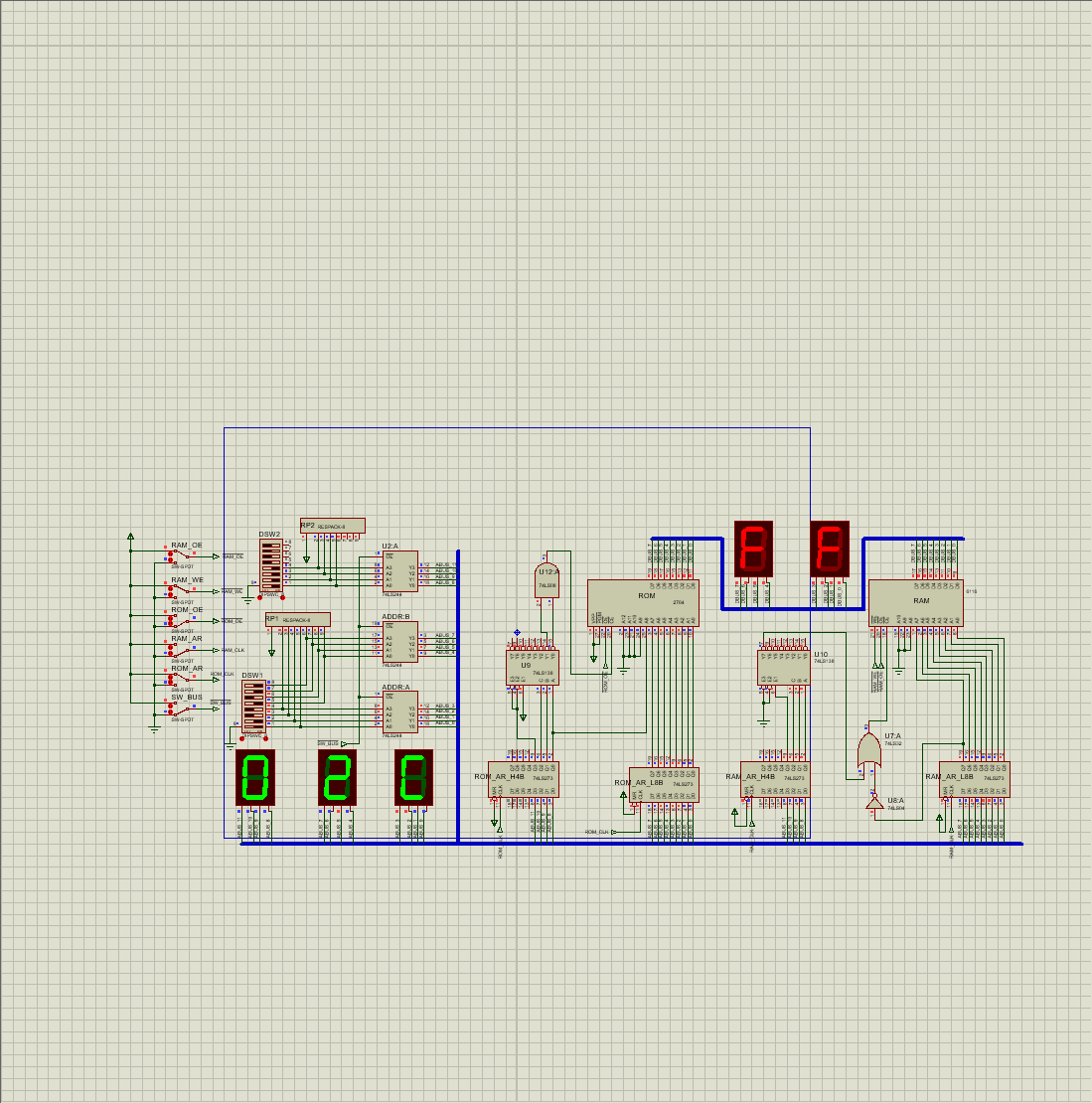
第二轮

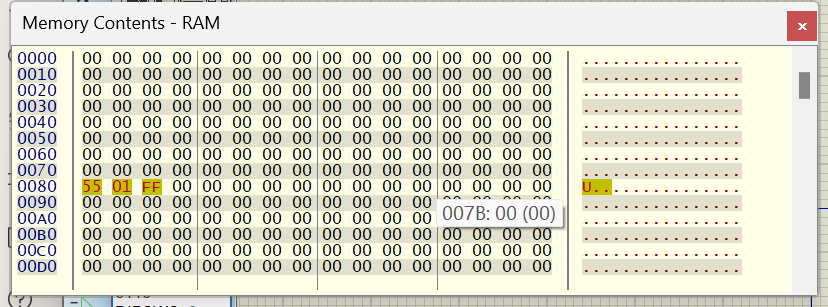
把ROM的[028H]数据写入RAM的[F81H]



第三轮

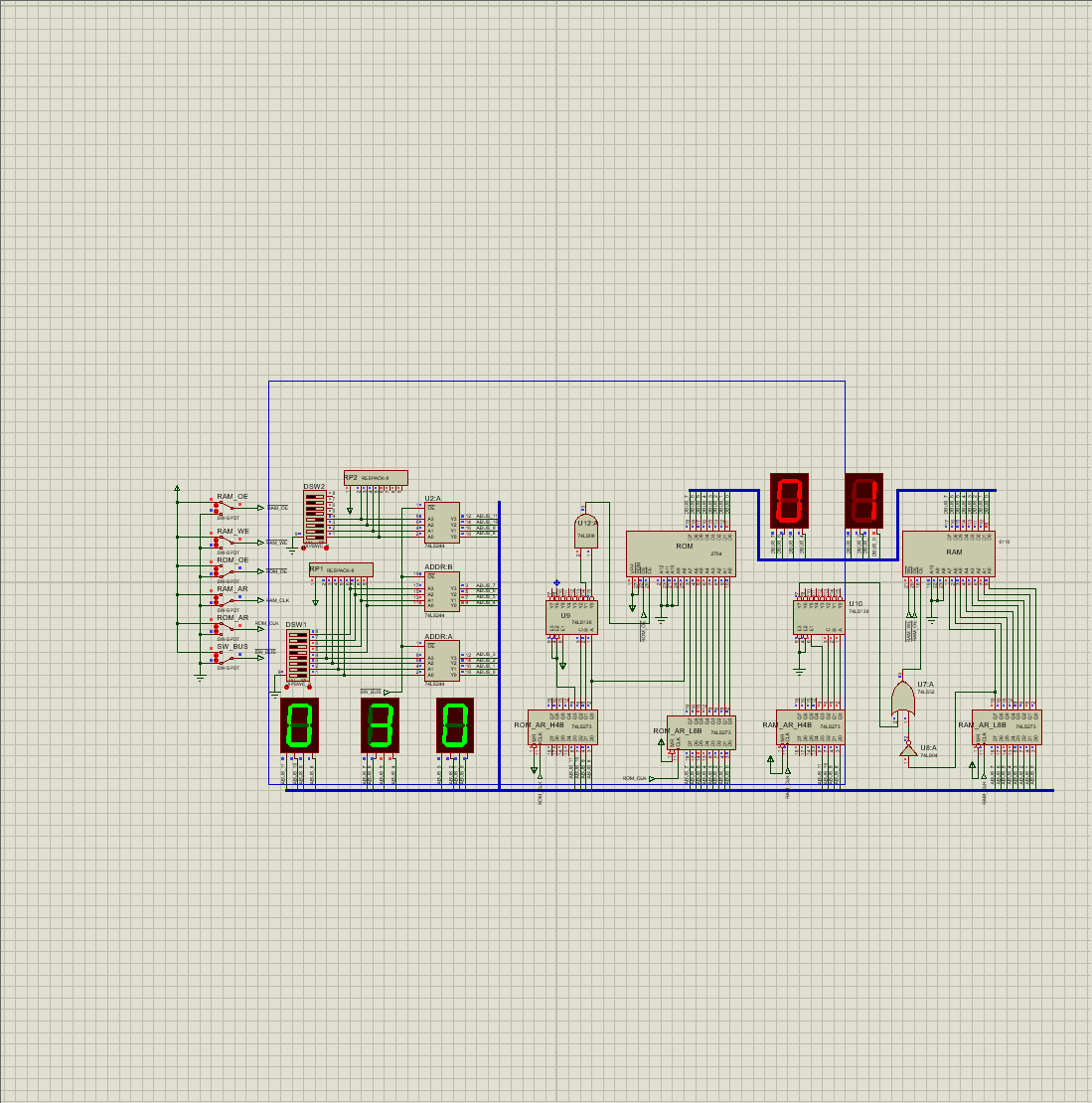
把ROM的[02CH]数据写入RAM的[F82H]

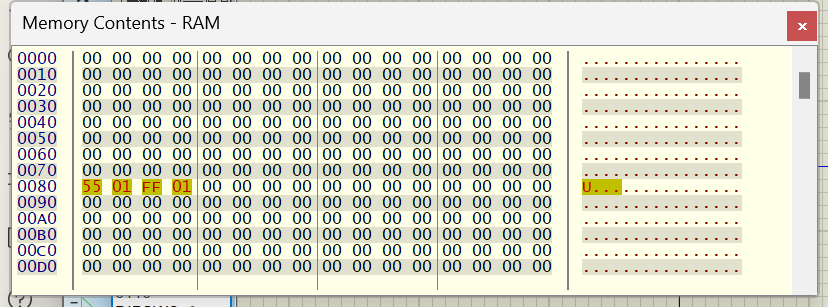




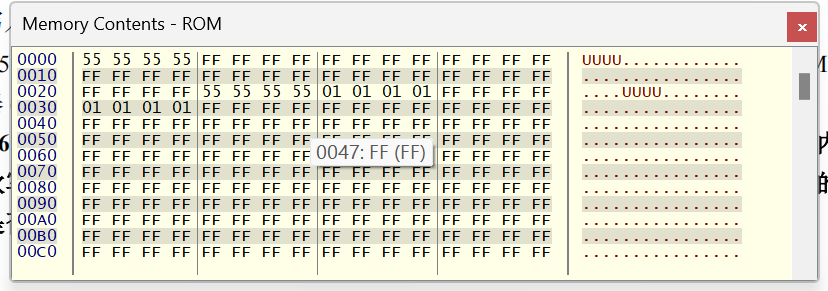
第四轮

把ROM的[030H]数据写入RAM的[F83H]





至此完成所有的数据写入，再次和ROM中内容核对：



55，01，FF，01

完全正确

重新启动，模仿断电后RAM的状态，确实存储的信息都消失了，体现其易失性，而ROM中烧入的数据断电后依旧存在

**七．实验思考**

1.为什么ROM芯片的片选由低位3-8译码器的最低2个部分片选信号的片选电路形成，RAM芯片的片选由高3-8译码器74LS138的最高1个部分片选信号与RAM的A7 地址线的片选电路形成。

**（1）** 地址空间规划匹配：实验中分配给RAM的是最高的128地址（F80H - FFFH），高位3 - 8译码器的最高1个部分片选信号结合RAM的A7地址线可以有效地对应到这一高地址范围，实现对RAM芯片在指定地址空间内的准确片选，确保RAM芯片在整个存储系统中地址范围的正确性和独立性。

**（2）**译码与地址线协同工作：利用高位3 - 8译码器和RAM的A7地址线共同形成片选电路，使得RAM芯片的片选信号不仅依赖于高位地址译码结果，还与A7地址线的状态相关，这种协同工作方式可以更精确地确定RAM芯片的片选时机，保证在对RAM进行读写操作时，地址信号能够准确选中RAM芯片中的相应存储单元，同时避免地址冲突，实现稳定的数据读写操作。

2.假设把.asm 文件中的某个ORG 语句改为“ORG 0224H”，请问该ORG 定义的数据段还能被访问到么？如果不能，是数据批量导入ROM 出错么？请修改ROM 的地址片选电路，保证“ORG 0224H”所定义的数据段能被访问到。

**（1）能否访问及原因：**把.asm文件中的某个ORG语句改为“ORG 0224H”后，该ORG定义的数据段不能被访问到。因为实验中ROM的地址空间范围是000H - 1FFH，0224H超出了这个范围，所以无法访问到。这不是数据批量导入ROM出错，而是地址超出了已设定的ROM地址范围。

**（2）修改片选电路方法：**要保证“ORG 0224H”所定义的数据段能被访问到，需要修改ROM的地址片选电路。由于原ROM地址范围是000H - 1FFH，对应地址线A11A10A9A8为0000 - 0011，现要包含0224H（对应地址线A11A10A9A8为0100），则需要调整译码器的连接，使片选信号能覆盖到新的地址范围。例如，可以重新设计片选逻辑，让低位3 - 8译码器的部分片选信号组合能够对应到新的地址范围，确保0224H所在的数据段能被选中。

3.为何ROM 和RAM 需要使用两个独立的3-8 译码器？假设RAM 的片选电路与ROM的片选电路共用一个3-8 译码器，即ROM 所在3-8 译码器的最低2 个端口给ROM使用，最高1 个端口给RAM 使用。请给出ROM 和RAM 的地址空间范围。

**（1）使用两个独立3 - 8译码器的原因**：ROM和RAM需要使用两个独立的3 - 8译码器是为了实现更灵活和准确的地址空间划分与片选控制。如果共用一个3 - 8译码器，会使地址译码逻辑变得复杂，不利于对ROM和RAM的独立管理和操作。例如，在本实验中ROM和RAM的地址空间范围不同，独立的译码器可以分别根据各自的地址范围需求进行优化设计，使片选信号更精准地对应到相应的存储单元，提高系统的可靠性和读写效率。

**（2）共用一个3 - 8译码器时ROM和RAM的地址空间范围**：若RAM的片选电路与ROM的片选电路共用一个3 - 8译码器，即ROM所在3 - 8译码器的最低2个端口给ROM使用，最高1个端口给RAM使用。对于ROM，其地址范围取决于与最低2个端口相关的地址线组合，假设地址线A11A10A9A8参与片选译码，那么ROM的地址空间范围可能是000H - 3FFH（对应A11A10A9A8为0000 - 0011）。对于RAM，其地址范围与最高1个端口及相关地址线有关，可能是C00H - FFFH（对应A11A10A9A8为1100 - 1111）。但具体范围还需结合其他地址线连接情况确定，这里只是一种可能的范围推导。

4.假设RAM 的地址空间范围改为800H-8FFH，请问存储器地址片选电路如何修改？假设再把ROM 的地址空间范围改为600H-7FFH，请问存储器地址片选电路又如何修改？

**（1）RAM地址空间范围改为800H - 8FFH时存储器地址片选电路的修改**：原RAM地址空间为F80H - FFFH，现改为800H - 8FFH。原RAM片选由高位3 - 8译码器74LS138的最高1个部分片选信号与RAM的A7地址线的片选电路形成，要实现新地址范围，需要调整片选电路逻辑。可能需要重新配置高位3 - 8译码器的输入地址线连接，使对应800H - 8FFH地址范围的信号能够正确产生片选信号。例如，调整与A11、A10、A9等地址线的连接，使译码器输出在地址为800H - 8FFH时能选中RAM芯片，同时要确保与其他存储器（如ROM）的地址范围不冲突。

**（2）ROM地址空间范围改为600H - 7FFH时存储器地址片选电路的修改**：原ROM地址空间为000H - 1FFH，现改为600H - 7FFH。原ROM片选由低位3 - 8译码器的最低2个部分片选信号的片选电路形成，要适应新地址范围，需要对片选电路进行调整。可能要改变低位3 - 8译码器与地址线的连接关系，使在地址为600H - 7FFH时能产生正确的片选信号选中ROM芯片。比如，重新规划与A11、A10、A9等地址线的连接，同时保证整个存储系统中各存储器地址范围不重叠，且片选电路逻辑正确可靠。

# 微程序控制器实验

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **地 点：** | B7楼 | 房； | **评 分：** |  |
| **实验日期与时间：** | 2024年11月30日星期六下午 | | **实验教师：** | 王国华 |

**一．实验目的**

* 理解“微程序”设计思想，了解“指令 - 微指令 - 微命令”的微程序结构。
* 掌握微程序控制器的结构和设计方法。

**二．实验内容**

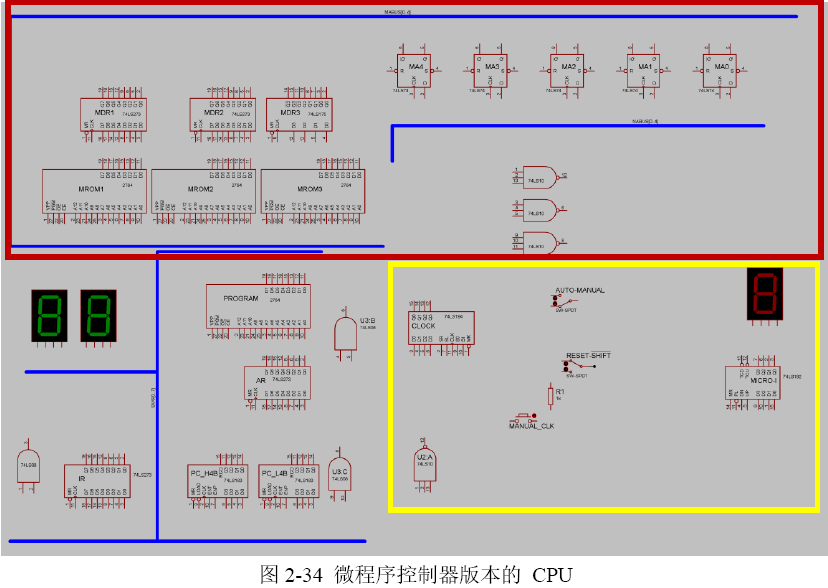
* 设计一个“最简版本”的CPU模型机，利用时序发生器产生CPU预定时序，通过微程序控制器自动控制，在数据通路中完成程序跳转功能。

**三．实验器件**

* ROM存储器（2764）和计数器（74LS163、74LS192）。
* D触发器（74LS74）、寄存器（74LS273）及移位寄存器（74LS194）。

**四．实验原理**

1. CPU构造与数据通路

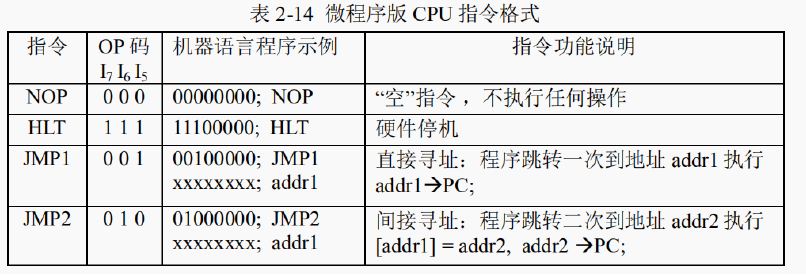
 本实验构造的“最简版本”CPU由时序发生器、微程序控制器和数据通路组成。数据通路由指令寄存器IR（74LS273）、程序计数器PC（74LS163级联）、程序存储器PROGRAM（ROM）及其地址寄存器AR（74LS273）组成，各部件并联挂在单条8位总线BUS上，通过数码管显示总线BUS信息。程序计数器PC是八位递增计数器，由两个同步二进制计数器74LS163级联构成，其加载和自加1功能需满足CLK端上升沿跳变条件。

2. CPU指令格式与功能

CPU使用四条基本指令：空指令NOP（OP码000）、停机指令HLT（OP码111）、直接寻址的跳转指令JMP1（OP码001）和间接寻址的跳转指令JMP2（OP码010）。



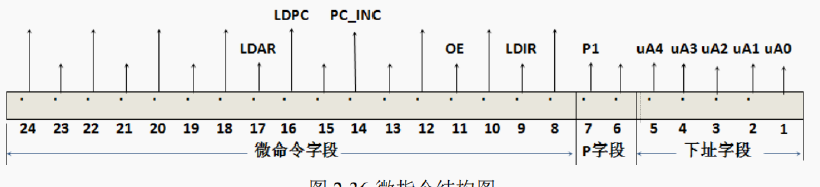
每条指令在指令周期中经历取指周期和执行周期，所有指令取指周期相同，执行周期不同或无（如NOP指令）。指令流（ROM->IR）从存储器取指令到指令寄存器，数据流（ROM->PC）从存储器取数据到程序计数器，两者内部可分为T1（源部件信息打入总线）和T2（总线信息打入目标部件）两个周期。



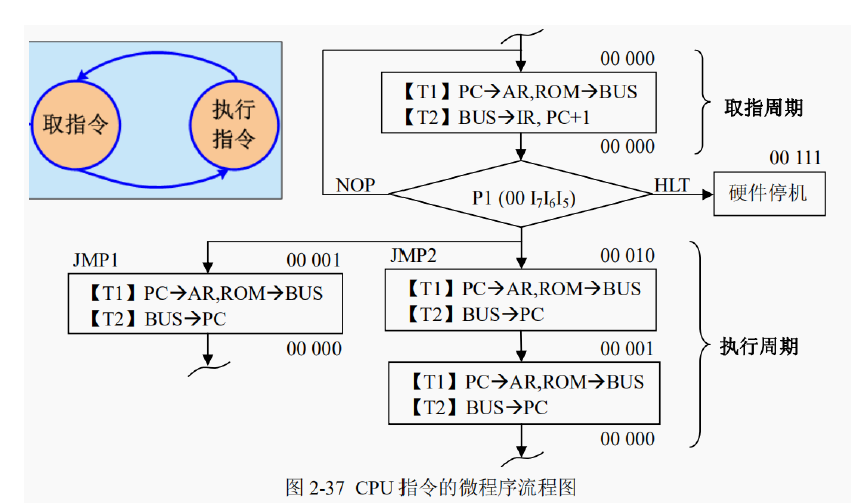
3. 微操作信号与微指令结构

 数据通路上设计一系列微操作信号控制部件，如电平触发信号OE、LDPC、LDAR、LDIR、PC\_INC等，边沿触发信号由电平触发信号与T1或T2周期对应的节拍信号逻辑“与”获得。

微指令字长24位，1 - 5位表示下一条微指令地址[uA4, uA0]，6 - 7位为判断字段Px（P2位空缺，P1 = 1表示取指周期微指令，P1 = 0表示执行周期微指令），8 - 24位是微命令字段（对应电平触发微操作信号）。

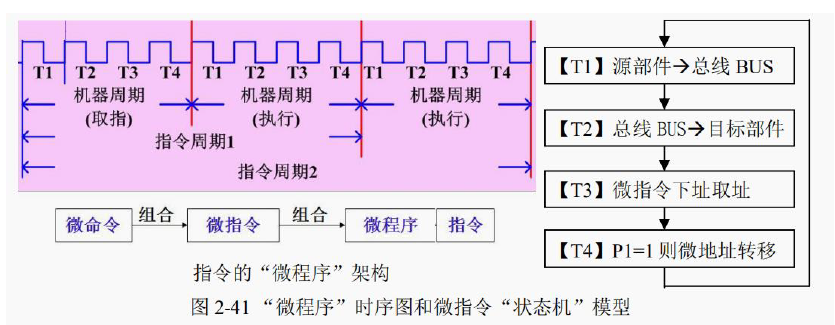


4. 微程序流程图与微程序控制器结构

 CPU指令的微程序流程图中，最上方为取指微指令，取指后根据指令OP码判断选择不同执行路径（入口地址[00I7I6I5]），NOP指令和HLT指令较特殊，NOP指令取指后返回下一条指令取指周期，HLT指令译码后硬件停机，所有路径末尾微指令下址都返回取指微指令地址。微程序控制器由控制存储器、微指令寄存器、微地址寄存器和地址转移逻辑电路组成。控制存储器字长24位，由3个2764芯MROM1 - 3组成，微指令代码表中的四条微指令存储其中，其输出连接微指令寄存器。微地址寄存器字长五位，由触发器74LS74组成，地址转移逻辑由三个三路与非门实现，根据指令寄存器IR的OP码在取指周期末尾修改微地址寄存器，微程序地址转移过程需在微指令周期增加T3（下址字段打入微地址寄存器）和T4（根据P1值决定是否转移微指令地址）两个周期。

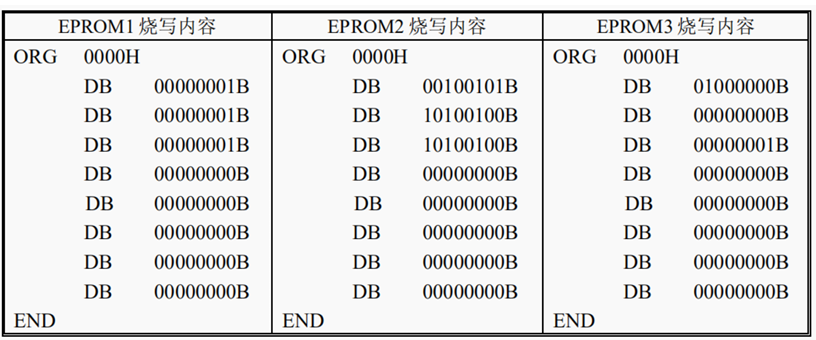
5. 时序发生器

时序发生器电路中CLK为全系统基准时钟信号，可由方波信号AUTO\_CLK提供或通过按键MANUAL\_CLK手动步进。移位寄存器74LS194用作节拍生成器，以CLK为基准时钟发送节拍信号T1 - T4，每个节拍信号对应微指令“状态机”的状态【Tx】，时钟CLK驱动状态转移，T1 - T4的一次循环称为一个CPU周期（微指令周期）。时序发生器还提供HLT指令停机功能，当指令寄存器IR的OP码I7I6I5 = 111时，硬件电路生成停机信号，74LS194状态变为{S0, S1} = {0, 0}，工作模式保持，停机在HLT指令的取指周期T2节拍上。此外，还有由T1上升沿驱动的74LS192计数器MICRO - I显示当前运行微指令条数。CPU初始化和跳出HLT指令“断点”有相应操作步骤。



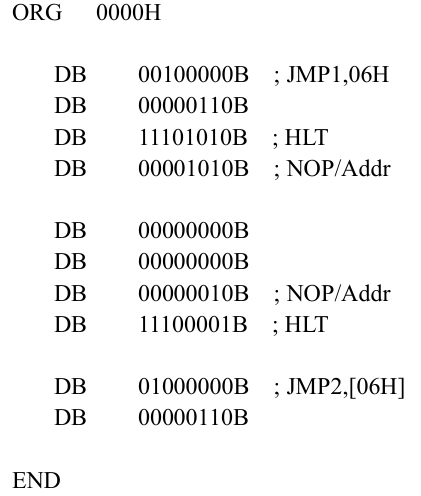
**五．实验步骤**

1. 微程序编写与烧写
   * 根据表2 - 17微指令代码表编写微程序，编译并生成三个HEX文件，分别烧写到控制存储器EPROM1、EPROM2及EPROM3（注意不要写错存储器）。烧写内容如下：



1. 机器语言程序编写与烧写

编译如下机器语言源程序，生成HEX文件烧写到程序存储器PROGRAM中（编译和烧写asm文件的方法参见“存储器实验：ROM批量导入数据”）：



1. 仿真操作

启动仿真前，时钟信号CLK接在MANUAL\_CLK端。启动仿真，使能复位信号RESET = 1，然后手动按钮MANUAL\_CLK，令时钟信号CLK上升沿跳变，初始节拍{T1, T2, T3, T4} = {1, 0, 0, 0}；最后恢复复位信号RESET = 0，初始化过程完成。

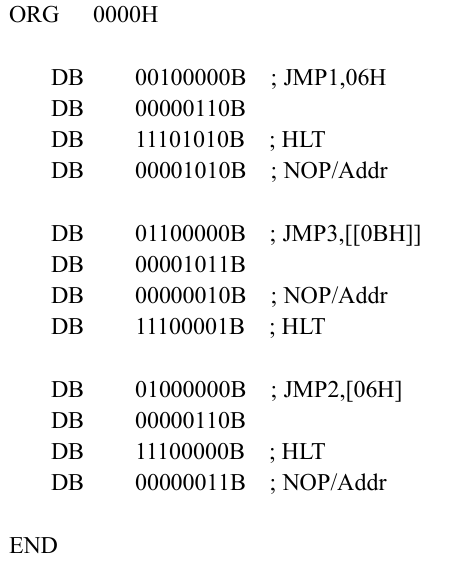
手动按钮MANUAL\_CLK，令时钟信号CLK形成脉冲，单步执行上述机器语言程序。在JMP1或JMP2指令的指令周期中，对照微程序流程图2 - 37，观察每一条微指令的作用，以及单步执行的结果（例如寄存器AR、IR、PC及总线BUS上的数据）。

时钟信号CLK改接在AUTO - CLK信号源（主频10Hz），程序会自动运行到HLT指令“断点”暂停。查看“断点”处的微指令周期数指示，以及寄存器AR、IR、PC及总线BUS上的数据。然后，使能复位信号RESET“0 -> 1 -> 0”，跳出“断点”，进入HLT指令的后续下一条指令继续运行。

1. 新指令JMP3相关操作（实验2部分）

增加二次间接寻址的跳转指令JMP3（OP码011），其功能为程序跳转三次到地址addr3执行[addr1] = addr2，[addr2] = addr3，addr3→PC。补充微程序流程图2 - 37及微指令代码表2 - 17（新增微指令地址[00011]）。

编译如下机器语言源程序，生成HEX文件烧写到程序存储器PROGRAM中（编译和烧写asm文件的方法参见“存储器实验：ROM批量导入数据”）：

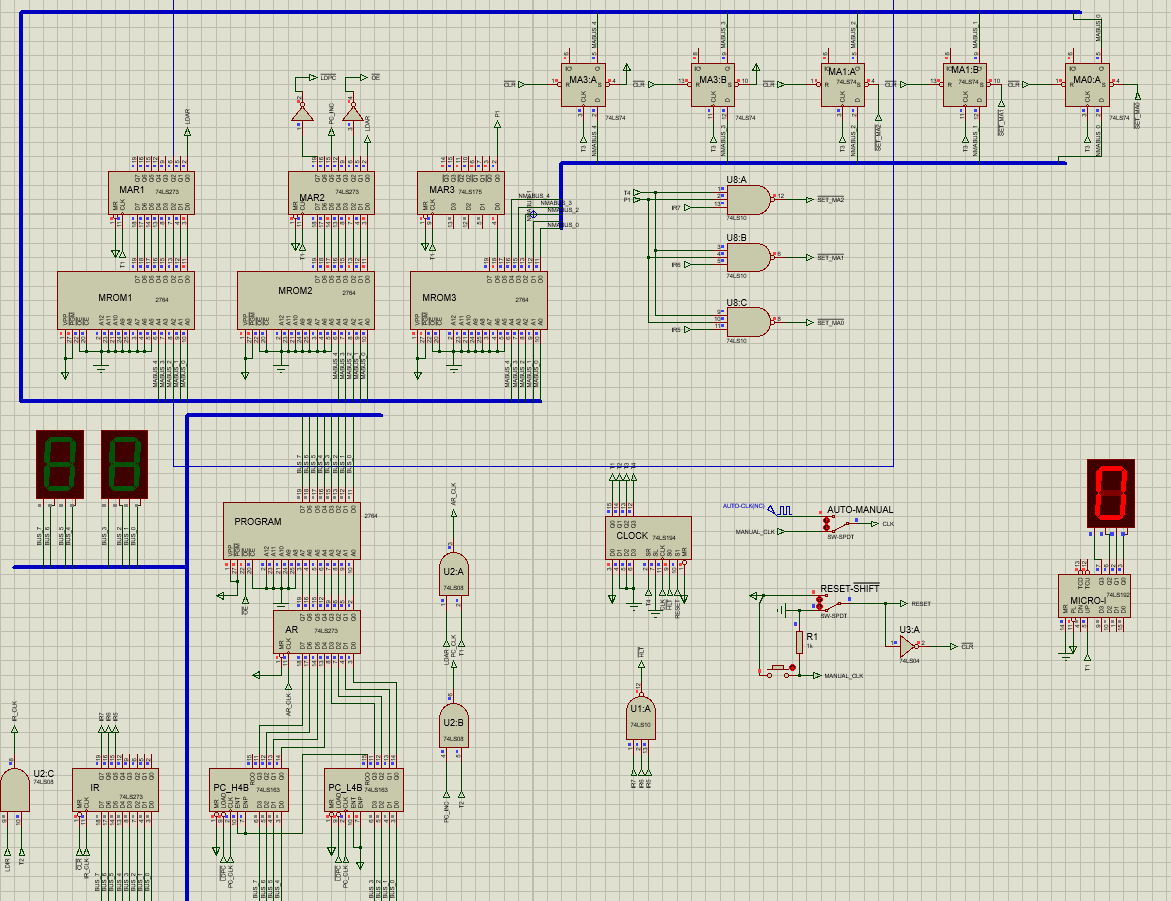


参照上述初始化、手动单步执行的方法，单步执行该机器语言程序。在JMP3指令的指令周期中，对照其微程序流程图，观察每一条微指令的作用，以及单步执行的结果（例如寄存器AR、IR、PC及总线BUS上的数据）

参照上述自动运行以及跳出“断点”的方法，自动运行该机器语言程序到HLT指令“断点”暂停。查看“断点”处的微指令周期数指示，以及寄存器AR、IR、PC及总线BUS上的数据。

**六．实验结果**

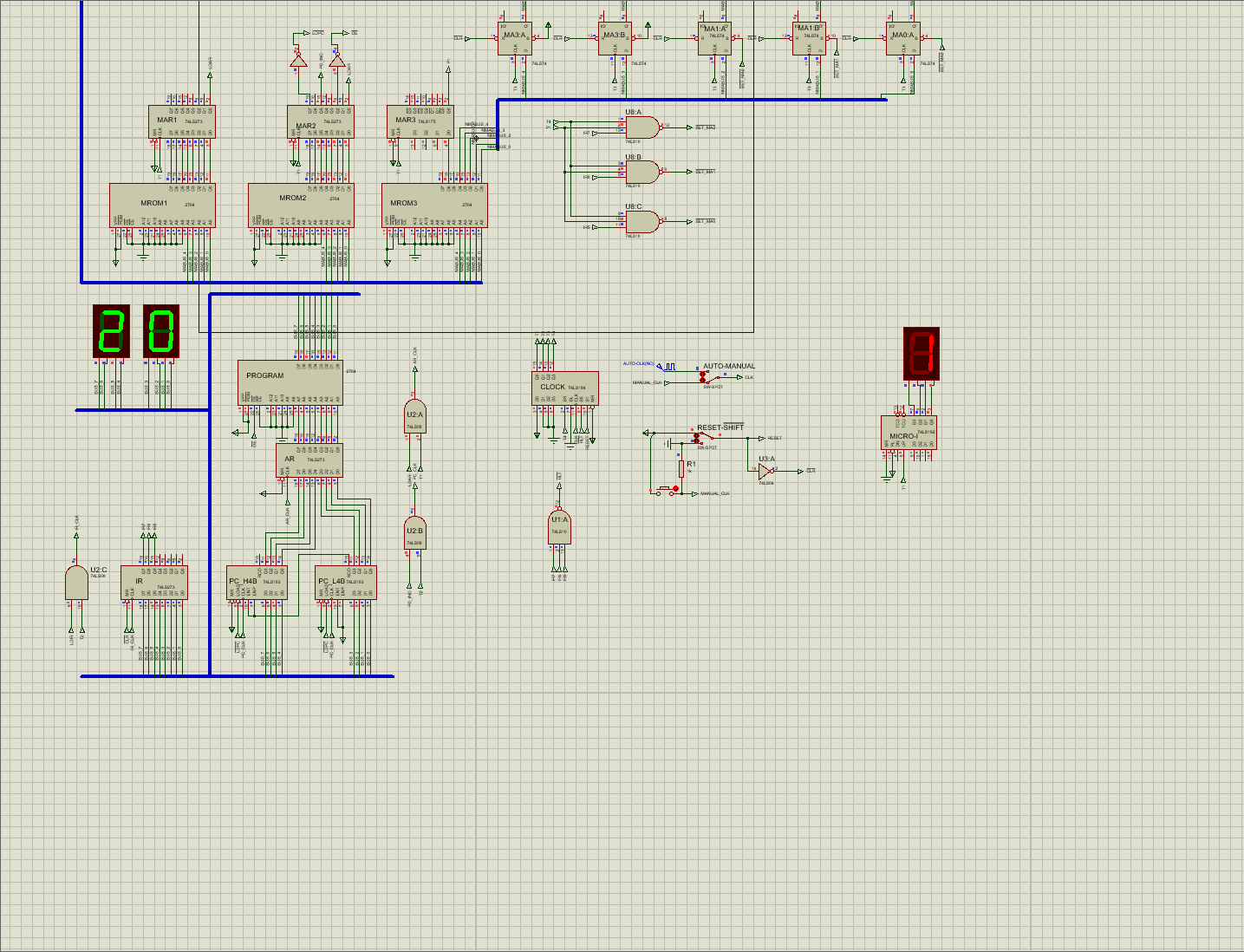
电路如下所示：



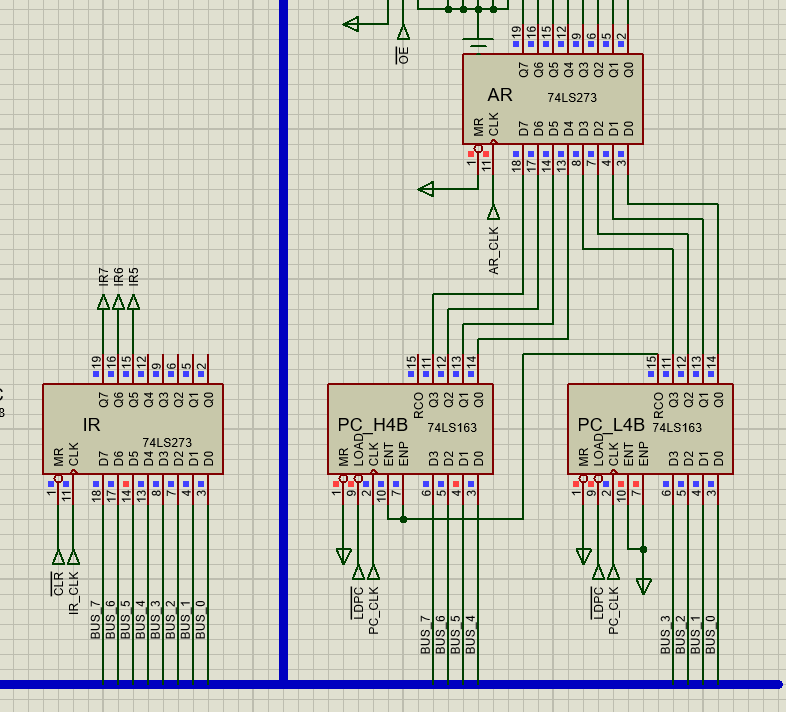
指令仅有NOP，HIT，Jmp1，Jmp2时

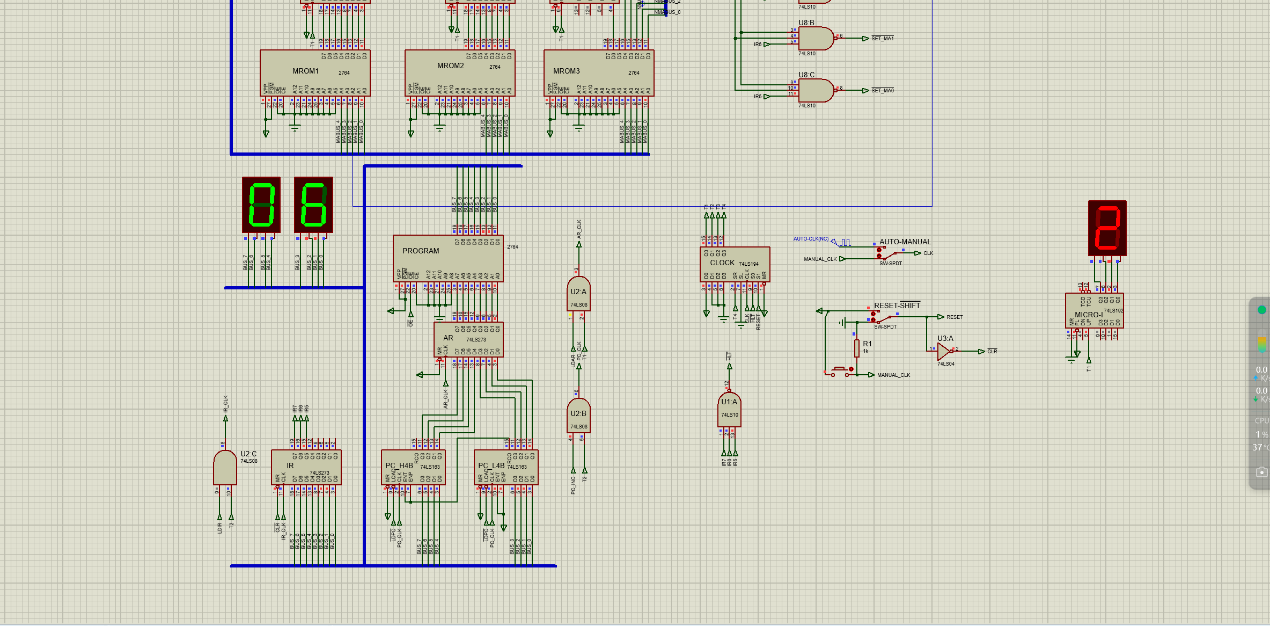
先把时钟信号CLK接在MANUAL\_CLK端，使能复位信号RESET = 1，按钮MANUAL\_CLK，节拍{T1, T2, T3, T4} = {1, 0, 0, 0}；恢复复位信号RESET = 0，完成开机

首先访问指令20，JMP1；

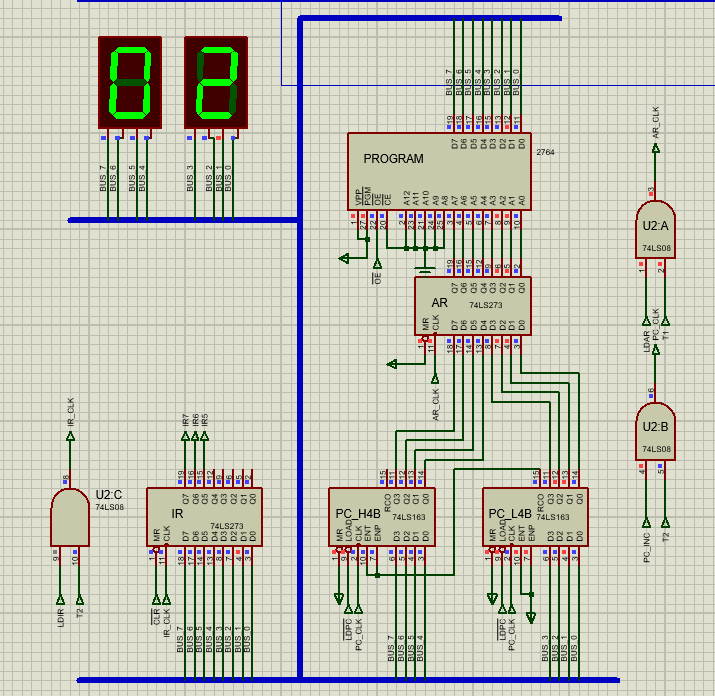


观察结果

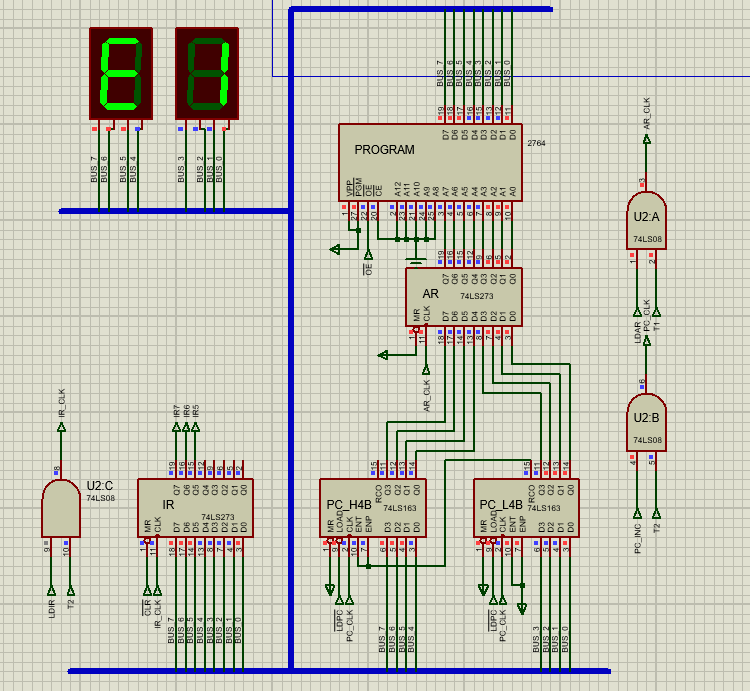


随后PC自增，指向下一条指令，06，空指令

根据指令一的内容，跳转到指令02，NOP

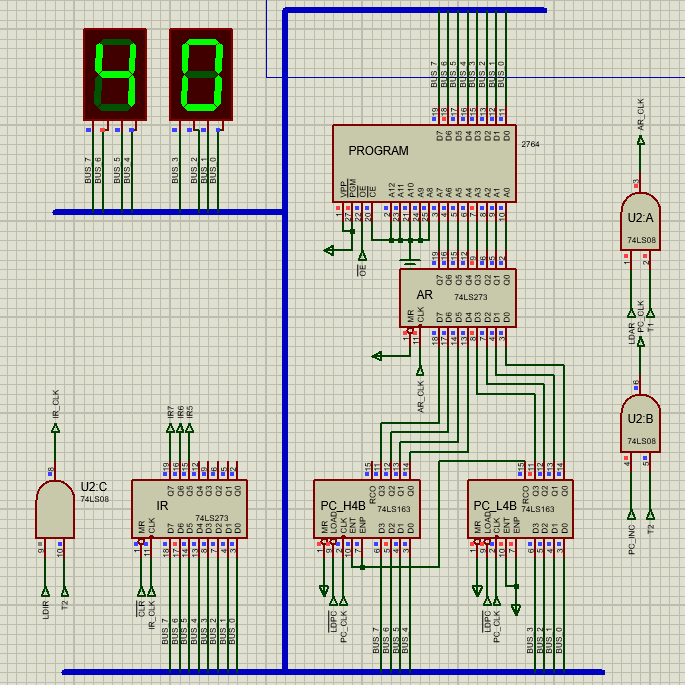
****

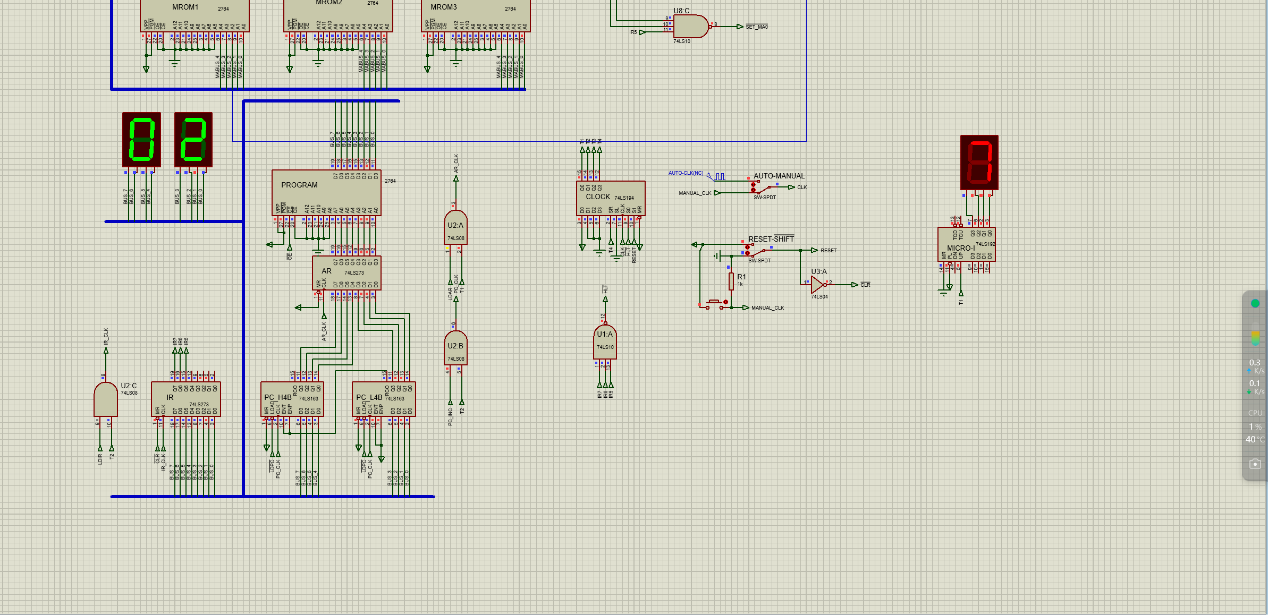
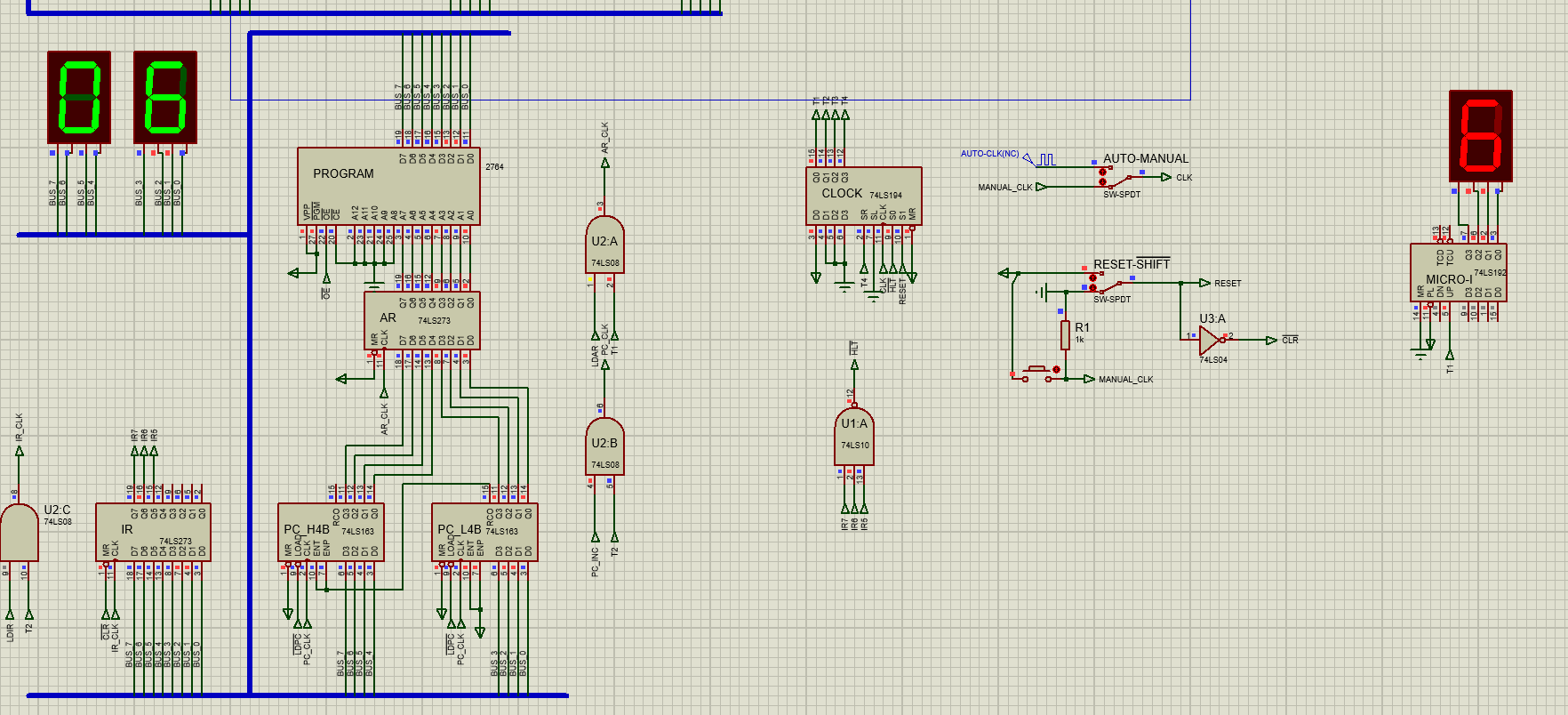
自增，读取下一条指令，E1，HLT

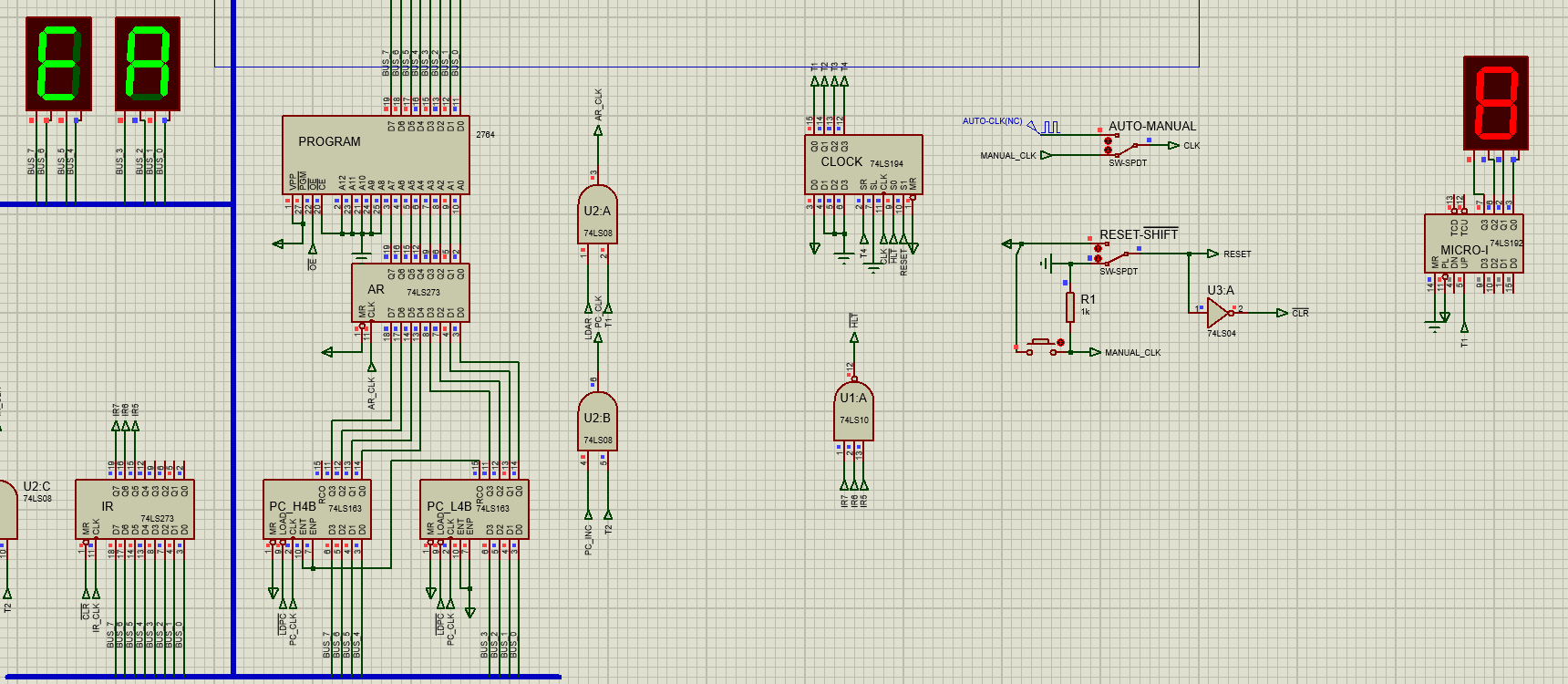


HTL会停止运行，再启动后到下一条指令，JMP2

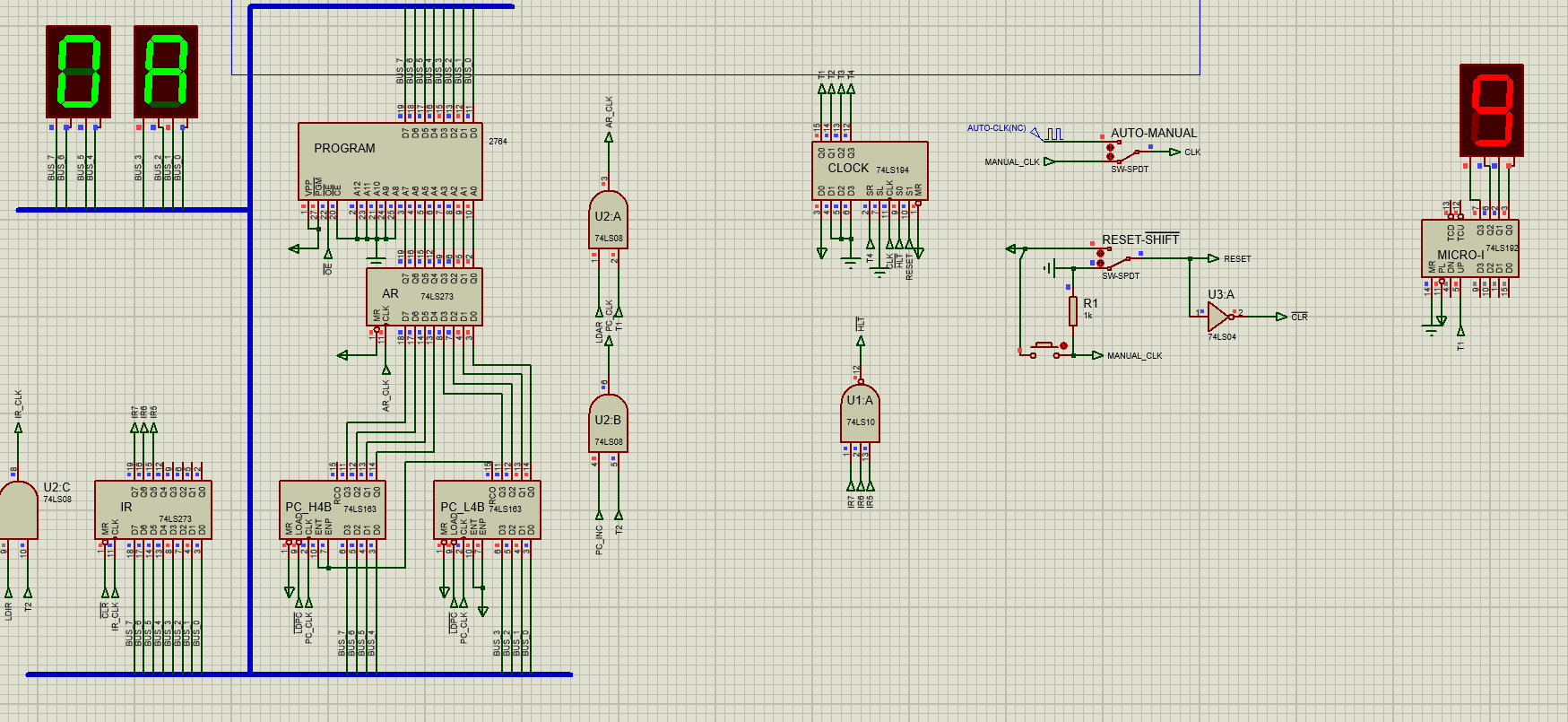
自增到下一条指令



****将06H中的内容作为地址进行访问

访问02H，读取指令，EA，HLT

停机重启后，继续读取下一条指令，NOP



**七．实验思考**

1、微程序版本CPU 最多有多少条微指令？最多有多少条CPU 指令？微指令和CPU 指令的容量分别由什么因素限定？

微指令数量：微程序控制器的控制存储器字长为24位，微地址寄存器字长5位，因此理论上控制存储器最多可存储\(2^5 = 32\)条微指令。但实际使用中，由于部分地址可能被保留或未被充分利用，具体可用微指令数量会小于32条，例如本实验中仅使用了4条微指令（地址[00000] - [00010]中的部分），且新增指令JMP3时需新增微指令地址[00011]，说明可根据需求逐步扩展微指令，但上限受控制存储器地址空间限制。

CPU指令数量：由指令格式可知，OP码占3位（I7I6I5），因此最多可有2^3 = 8条不同的CPU指令。但本实验仅使用了4条基本指令（NOP、HLT、JMP1、JMP2），后续新增JMP3指令，说明可按需扩展CPU指令集，其上限由OP码的位数决定。

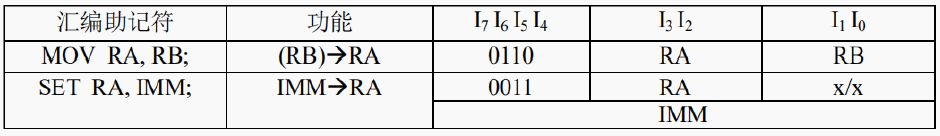
2、请问微程序控制器 “状态机 ”可否提升效率，减少到三个状态 {T1,T2,T3}？即微指令周期可否减少到只用 T1、 T2、 T3 三个节拍即可完成一条微指令从取指到执行的全过程？

目前微指令周期包含T1（源部件->总线）、T2（总线->目标部件）、T3（微指令下址取址）、T4（P1 = 1则微地址转移）四个周期，每个周期都有其特定功能。

T1和T2用于完成指令流和数据流在总线间的数据传输操作，这是保证数据正确流向的关键步骤，难以合并或省略。

T3用于将当前微指令的下址字段打入微地址寄存器，为取下一条微指令做准备，若取消此步骤，微指令地址获取将出现问题。

T4用于根据取指周期末尾微指令的P1信号进行微地址转移，以实现不同指令执行周期的正确跳转，此功能对于指令执行流程的控制至关重要，若去掉T4，指令执行逻辑会混乱。所以在当前架构下，难以仅用三个状态{T1, T2, T3}完成一条微指令从取指到执行的全过程。

3、在本次实验的CPU 模型机上增加两个74LS173 寄存器R1 和R2，以及一个连接总线BUS 的8 位拨码开关，扩展CPU 指令集，增加下述MOV/SET 指令及相应的微指令：

（注：IMM 是由拨码开关输入的8 位立即数；RA 和RB 是在指令“功能”描述中的逻辑寄存器，可以对应R0 或R1 寄存器。）

取指周期：与原实验中的取指过程类似，从程序存储器PROGRAM取出指令，经过总线BUS存入指令寄存器IR，此过程涉及的微操作信号（如OE、AR CLK、LDIR、PC CLK等）及其控制逻辑基本不变。

执行周期

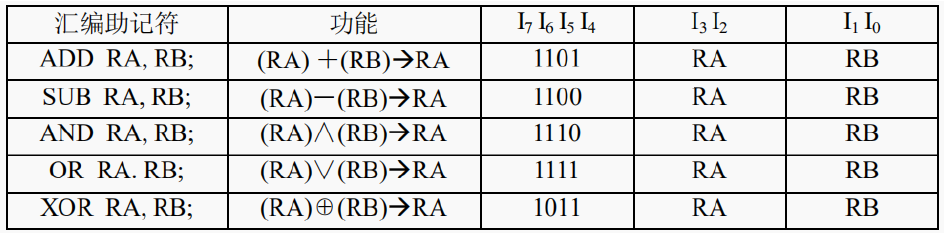
根据指令格式，译码确定是MOV指令后，通过微操作信号控制，将源寄存器RB的数据（通过总线BUS）传输到目标寄存器RA。具体的微操作信号需要根据74LS173寄存器的控制逻辑来设计，例如可能需要设计加载信号（类似LDIR）来控制数据何时存入目标寄存器。

对于SET指令，将拨码开关输入的8位立即数IMM通过总线BUS传输到指定的寄存器RA，同样需要设计合适的微操作信号来实现数据传输和寄存器加载。

微指令设计方面

需要设计新的微指令来实现MOV/SET指令的功能。在微指令结构中（24位），下址转移方式（1 - 5位）可根据微程序流程确定后续微指令地址；判断字段P1（6 - 7位）可用于区分取指和执行周期；微命令字段（8 - 24位）则需定义控制R1、R2寄存器操作以及与拨码开关交互的微操作信号，例如控制寄存器加载、数据传输方向等的信号。具体的微指令编码需要根据详细的硬件连接和操作逻辑来确定，确保能准确实现MOV/SET指令的功能。

4、在思考题3 的电路基础上，参考上述“运算器实验”，增加74LS181 运算器电路，扩展CPU 指令集，增加下述ADD/SUB/AND/OR/XOR 指令及相应的微指令：



**硬件连接方面**

将74LS181运算器电路正确连接到总线BUS上，使其能够接收来自寄存器（R1、R2等）的数据进行运算，并将运算结果输出到总线或指定寄存器。

确保与已有的部件（如寄存器、总线等）连接正确，保证数据通路畅通，以便在执行运算指令时数据能够正确流入和流出运算器。

**指令执行流程方面（以ADD指令为例，其他指令类似）**

**取指周期**：同前面扩展指令集的取指过程，从程序存储器取出指令存入指令寄存器IR。

**执行周期**

译码确定是ADD指令后，通过微操作信号控制，将参与运算的两个寄存器（RA和RB）的数据通过总线BUS传输到74LS181运算器的输入端。

运算器根据指令要求进行加法运算，运算结果通过总线BUS传输回目标寄存器RA（或根据指令指定的其他存储位置）。这需要设计一系列微操作信号来协调运算器与寄存器、总线之间的数据传输和控制运算过程，例如控制运算器的功能选择（加法功能）、数据输入输出使能等信号。

**微指令设计方面**

设计新的微指令来实现这些算术逻辑运算指令的功能。微指令结构中的下址字段、判断字段和微命令字段都需要重新规划。微命令字段需定义控制运算器操作的信号（如选择运算类型、启动运算等）以及与寄存器和总线交互的信号（如数据传输到运算器、从运算器取回结果等）。根据具体的运算指令功能和硬件连接，确定每条微指令的编码，以确保能准确执行相应的算术逻辑运算，并与整个微程序流程和其他指令协调工作。例如，对于ADD指令，微指令需能正确控制数据从寄存器到运算器、执行加法运算并将结果存回寄存器的全过程。