

大连海事大学

设计报告

课程名称：计算机组成原理实践

成 员：

设计时间：2022年6月20日至6月30日

目录

**[一、设计目的](#_Toc1648822)** [3](#_Toc1648822)

**[二、设计方案](#_Toc1648823)** [3](#_Toc1648823)

**[三、设计步骤](#_Toc1648828)** [3](#_Toc1648828)

3.**[1 确定任务](#_Toc1648829)** [3](#_Toc1648829)

3.**[2 总体设计](#_Toc1648830)** [3](#_Toc1648830)

**[3.3 软件研制过程](#_Toc1648831)** [4](#_Toc1648831)

**[3.3.1 CPU 指令集](#_Toc1648831)** [4](#_Toc1648831)

**[3.3.2微程序控制器](#_Toc1648831)** [5](#_Toc1648831)

**[3.3.3 取指过程](#_Toc1648831)** [7](#_Toc1648831)

**[3.3.4 寄存器指令](#_Toc1648831)** [9](#_Toc1648831)

**[3.3.5 存储器及堆栈操作指令](#_Toc1648831)** [1](#_Toc1648831)0

**[3.3.6 跳转系列指令](#_Toc1648831)** [1](#_Toc1648831)1

**[3.3.7 算术逻辑运算系列指令](#_Toc1648831)** [1](#_Toc1648831)1

3.**[4 硬件研制过程](#_Toc1648832)** [1](#_Toc1648832)2

3.**[4.1 实验概述](#_Toc1648832)** [1](#_Toc1648832)2

3.**[4.2 微程序 CPU架构](#_Toc1648832)** [1](#_Toc1648832)3

3.**[4.3 时序电路](#_Toc1648832)** [1](#_Toc1648832)4

3.**[4.4 运算器](#_Toc1648832)** [1](#_Toc1648832)5

3.**[4.5 微程序控制器](#_Toc1648832)** [1](#_Toc1648832)9

3.**[4.6 时序发生器](#_Toc1648832)** [2](#_Toc1648832)4

3.**[4.7 存储器](#_Toc1648832)** [2](#_Toc1648832)6

**3.[5 实验总结 3](#_Toc1648832)2**

**四、[参考文献](#_Toc1648838)** [3](#_Toc1648838)**2**

成员分工：

共同完成：整机测试及整合

### 一、设计目的

实现一个简单的 CPU，并且在此 CPU 的基础上，继续构建一个简单的模型计算机。CPU 由运算器（ALU）、微程序控制器（MC）、通用寄存器（R0），指令寄存器（IR）、程序计数器（PC）和地址寄存器（AR）组成,如图所示。这个CPU 在写入相应的微指令后，就具备了执行机器指令的功能，但是机器指令一般存放在主存当中，CPU 必须和主存挂接后，才有实际的意义，所以还需要在该 CPU 的基础上增加一个主存和基本的输入输出部件，以构成一个简单的模型计算机。

### 二、设计方案

硬件设计主要内容包括： ① 基本运算器电路设计； ② 存储器电路设计； ③ 时序产生器电路设计； ④ 微程序控制器电路设计； ⑤ 采用简单的跳转指令和相应的微程序实现，整机验证。

## 三、设计步骤

### 3.1确定任务

小组成员分别设计四个模块：存储器、微程序控制器、运算器和时序发生器，然后分别测试后进行整合做出完整的模型机并测试，小组成员共同学习整个模型机的实现原理，了解程序的运行过程，随后由吴曦源同学设计出了微指令和寻址方式，最后小组成员一起对模型机运行过程中出现的问题进行讨论并解决，最终模型机可以正确地、完整地运行。

### 3.2 总体设计

模型机设计三大类指令共九条，其中包括**运算类指令**、**控制转移类指令**，**数据传送类指令**。

运算类指令设计有 2 条运算类指令，分别为：ANDI/AND、ADDI/ADD，ANDI和ADDI指令都为单字节，寻址方式采用寄存器直接寻址。

控制转移类指令有2条 HLT、JMP，用以控制程序的分支和转移，其中 HLT为单字节指令，JMP为双字节指令。

数据传送类指令有 IN、OUT、MOV、LAD、STA 共 5 条，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 指令为单字节指令外，其余均为双字节指令。

### 3.3 软件设计

3.3.1 CPU 指令集

指令的OP码如表3-1所示。其格式定义如下:指令OP码为指令第一个字节的高4位，即指令寄存器IR的{I7 I6 I5 I4}位。而RA和RB是指由I3 I2和I1 I0定义的逻辑寄存器，Ra或Rb都可以选择4个物理寄存器(R0~R3) 中的任何一个。

CPU指令集

|  |  |  |  |
| --- | --- | --- | --- |
| OP码（I7~I4） | 指令助记符 | OP码（I7~I4） | 指令助记符 |
| 0000 | NOP、HLT | 0100 | LAD、POP |
| 0001 | JMP、JMPR | 0101 | STO |
| 0010 | SET | 1101 | ADD |
| 0011 | MOV | 1110 | AND |

上述指令集总共有11条机器指令，可以分成以下五大类。

1. 系统指令

系统及中断指令包括2条单字节指令:空指令(NOP)、停机指令(HLT), 如表3-2 所示。其中，NOP指令主要用于精准延时(微程序/硬布线CPU延时4个T,流水线CPU延时1个T); HLT指令用于程序末尾CPU停机或设置“断点”，程序自动运行到HLT指令时刻停机，可以观察当时CPU寄存器、运算器标志位等信息。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 汇编语言 | 功能 | I7~I4 | I3I2 | I1I0 |
| NOP; | 无操作（延时4个T） | 0000 | 0/0 | x/0 |
| HLT; | 停机（断点） | 0000 | 0/0 | x/1 |

1. 寄存器指令

寄存器操作指令包括单字节的寄存器间数据传送指令(MOV)和双字节的寄存器赋值指令(SET), 如表所示。SET指令的第二个字节是赋予寄存器RA的立即数IMM。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 汇编语言 | 功能 | I7~I4 | I3I2 | I1I0 |
| MOV RA,RB; | (RB)→RA | 0011 | RA | RB |
| SET RA,IMM; | IMM→RA | 0010 | RA | x/x |
| IMM | | |

3.存储器及堆栈操作指令

存储器操作指令包括2条双字节指令:取数指令(LAD) 和存数指令(STO)。 LAD指令把数据从地址ADDR (指令第二个字节)的存储器单元取出，存入逻辑寄存器R;而STO指令把逻辑寄存器R的数据取出，存人地址ADDR ( 指令第二个字节)的存储器单元。

堆栈操作指令包括一条单字节指令:出栈指令(POP)。此处提到的“堆栈”是基于存储器ROM、RAM的“软堆栈”，其指针就是逻辑寄存器Rb。出栈指令即是把逻辑寄存器Rb存放的内容作为存储器地址，把该地址单元的数据弹出到逻辑寄存器RA。存储器及堆栈操作指令说明如表所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 汇编语言 | 功能 | I7~I4 | I3I2 | I1I0 |
| LAD RA,[ADDR]; | [ADDR]→RA | 0100 | RA | 0/0 |
| ADDR | | |
| POP RA,[RB]; | [RB]→RA | 0100 | RA | RB |
| STO RA,[ADDR]; | RA→[ADDR] | 0101 | RA | 0/0 |
| ADDR | | |

4. 跳转系列指令

无条件跳转指令JMP、JMPR的功能是程序必须跳转到目标地址执行跳转系列指令说明如表所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 汇编语言 | 功能 | I7~I4 | I3I2 | I1I0 |
| JMP ADDR; | ADDR→PC | 0001 | 0/0 | 0/0 |
| ADDR | | |
| JMPR RB; | (RB)→PC | 0001 | 0/0 | RB |

跳转系列指令共用OP码0001，指令的I3 I2位规定是无条件跳转指令(JMP、JMPR)。

5.算术逻辑运算指令

双操作数运算指令可以把两个操作数进行算术运算:加法(ADD)以及逻辑运算与(AND)。指令的I1 I0位指定两个操作数全部来自寄存器或分别来自于寄存器和立即数IMM (指令第二个字节)。前者是单字节指令(ADD、AND),后者是双字节指令( ADDI、ANDI)，如表所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 汇编语言 | 功能 | I7~I4 | I3I2 | I1I0 |
| ADD RA,RB; | （RA）+(RB) →RA | 1101 | RA | RB |
| AND RA,RB; | (RA)^(RB) →RA | 1110 | RA | 0/0 |

**3.3.2微程序控制器**

微程序版CPU的微指令结构图如图所示。微指令字长24位，其中微指令的第1 ~5位是下一条微指令地址，即下址字段[uA4, uA0];微指令的第6~8位是判断字段P1 ~P3;微指令的第9 ~24位则是微命令字段，对应数据通路的所有微操作信号，其中置1的位表示执行相应的微操作;反之，置0的位则是不执行相应的微操作。

图片包含 图示

描述已自动生成

此外，微指令的第15 -13位和12 -10位采用字段编译法(3-8译码)，分别对应源部件输出到总线和总线打人目标部件的微操作信号，如表所示。

表格

描述已自动生成

基于上述微指令结构，本实验设计了如图3-5所示的微程序控制器通路，包括3个8位ROM存储器2764组成的微指令存储器MROM1— MROM3、 3个寄存器 74L5273组成的微指令寄存器MDR1 -MDR3，微指令译码电路、5位微地址寄存器MAO ~ MA及微地址转移电路。

如图所示，24位微指令存储器MROM1~ MROM3共存放了32条微指令，所以微地址是MABUS\_4- MABUS\_0。当信号为ON或在T,节拍上升沿，微地址下址指定的微指令的微命令字段和P字段锁存到微指令寄存器MDR 1~ MDR3,输出微操作信号;而下址字段则送往微地址寄存器。

其中，微指令寄存器MDR2,的输端采用了两个3 -8译码器74LS138进行字段译码，分别实现把数据从源部件输出到总线BUS (译码器U8)和从总线打入目标部件(译码器U7)的微操作。因为，在任何一条微指令中，仅有一条数据路径，该路径中只有一个源部件把数据打人总线，其微操作信号互斥;也只有一个目标部件从总线接收数据，其微操作信号也是互斥。因此，上述两种微操作信号可以分别用译码器实现。

图示, 示意图

中度可信度描述已自动生成

OP码的I4用来指定双操作数运算指令ADD/ADDI和的CN操作信号。

此外，为了节省微指令地址，所有的双操作数和单操作数运算指令都在指令执行序列的第一条微指令就锁存 [S3,S2,S1, S0,M,CN]，待到最后条微指令 再打入运算器74LS181 执行，从而得到运算结果输出到总线BUS。因此，在第一条微指 令的微操作信号DA \_CLK上升沿跳变时刻，把[S3,S2,S1 ,S0,M,CN]锁存到74LS374寄存器U38;在最后-条微指令周期， 微操作信号ALU\_ BUS=0使能74LS374输出[S3,S2,S1,S0,M,CN] 执行，再得到运算结果。

本实验的微地址转移电路如图3-7所示，微地址寄存器5位(MA4 ~ MA0),由触发器74LS74组成。T2 时刻，当前微指令第1 ~5位的下一条微指令地址[ uA4,uA0 ]打人微地址寄存器;此刻，若当前执行的微指令中的判断位P1 ~P3置位，则地址转移逻辑电路将根据判断位P1~P3,置位微地址寄存器MA4 ~MA0。如图3-7所示，P2 位生成信号SET \_MA4，P2生成信号SET\_MA3 ~SET\_MA1, P1和P3,位共同生成信号SET\_MAO。判断位P1~P3的地址跳转逻辑如下。

●PI逻辑:若当前微指令是机器指令取指周期的最后条微指令， 则判断位P1=1,从而根据指令寄存器IR的I7 I6 I5 I4位强制置位微地址寄存器的MA3~MA0，修改微地址[uA3 ,uA0]位，转向该机器指令的执行周期序列的第一条微指令地址[0I7 I6 I5 I4].

●P2逻辑:若当前微指令是机器指令执行周期的最后条微指令， 则判断位P2=1, 此时若无中断发生，则返回取指周期第一条微指令地址[00000];若有中断发生(INT=1)，则强制置位微地址寄存器的MA4,转向中断处理过积第条微指 令地址[10000]。

●P3逻辑:在CPU指令集中部分单字节指令和双字节指令( LAD/POP、STO/PUSH 、ALU系列和JMP系列指令)共用OP码，其执行周期的微指令序列共用第一条微指令(判断位P3=1)，从第二条微指令开始分支，根据指令寄存器IR的I10位来决定不同微指令的分支走向:若I1I0=00，微指令下址的MA0=0,操作数分别来自寄存器和存储器(双字节指令);若I1 I0≠00，则微指令下址的MA0=1,操作数全部来自寄存器(单字节指令)。

**3.3.3取指过程**

除了空指令(NOP)和停机指令(HLT) 以外，所有的CPU指令都包括了取指周期和执行周期。因为NOP指令OP码为“0000”， 所以取指周期末尾P1 (0I7 I6 I5 I4) 译码的时候，直接返回取指周期(取下一条指令)，没有执行周期。而HLT指令与NOP指令完全相同，唯一不同之处是在取指周期后 CPU硬件停机，需要手动RESET (重启)才能跳出停机状态，进人下一条指令。

如图所示是取指周期及NOP、HLT指令的微程序流程图，其中每个方框在时间上表示一个微指令周期， 包括T1和T2两个节拍;在空间上表示数据从某个源部件经过总线(BUS)到达另一个目标部件的路径。每个方框的右上方是该微指令在控制存储器中的地址，右下方则是下一条微指令的地址。表3- 14列出取指周期(即NOP、HLT指令)的微指令代码。

表格

描述已自动生成

图示

描述已自动生成

本实验设计的存储器地址总线8位，地址空间256字节(00H- FFH)。分配其中低半区(00H -7FH) 为ROM存储区(128字节)，高半区(80H~FFH)为RAM存储区(128字节)。如图所示，存储器ROM和RAM共用一个地址寄存器(AR)， 两个存储器共用MEM OE信号作为存储器读信号，由地址最高位A7来作为两个存储器的片选信号。RAM 存储器是可读/写存储器，存放临时的数据。而ROM是只读存储器，存放程序和常量(采用堆栈操作指令访问)。因此，CPU程序和常量的存储容量最大是128字节，若程序和常量的代码量超过了128字节，则会越界出错。同样的，因为只有存储器RAM允许写人，所以当存储器写信号R\_ WE =0的时候，只有地址范围[80H,FFH]是允许写人操作的;对地址范围[00H ,7FH]的存储器单元进行写入操作是非法的。

存储器ROM (存放程序和常量)和RAM (存放数据)共用地址寄存器AR,而程序计数器PC和AR并联挂到总线(BUS)。 因此，如图所示，取指周期需要两条微指令(即两次路径):第一条微指令[00000]的T1时刻，PC输出当前指令地址到总线(PC\_ BUS=0), T2时刻由AR\_ CLK上升沿打人存储器地址寄存器(AR); 第二条微指令T1时刻，程序存储器ROM输出指令(MEM OE =0)到总线(BUS)， 在T2时刻由IR\_CLK上跳沿打人指令寄存器(IR); 并且PC+1(PC\_CLK上升沿)。

**3.3.4寄存器指令**

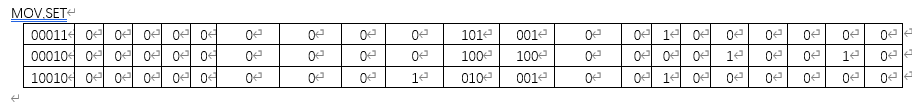
寄存器操作指令包括一条单字 节的寄存器间传送指令(MOV) 和一条双字节的寄存器赋值指令(SET)。

图是MOV、SET指令的微程序流程图。

图示

描述已自动生成

表3-15列出了寄存器操作指令MOV、SET, 以及1/0操作指令IN、OUT/OUTA的微指令代码。



**3.3.5存储器及堆栈操作指令**

双字节存储器操作指令包括了取数指令LAD和存数指令STO,而单字节堆栈操作指令包括出栈指令POP。

如图所示是存储器操作指令LAD、STO和堆栈操作指令POP的微程序流程图。从图中可以看出，POP指令只需要[11011]和[11101]两条微指令就够了，但是为了节省OP码，POP和LAD指令共用OP码[1000]，即共用第条微指令[01000] (即使POP指令其实并不需要微指令[01000]从而可以在第一条微指令的末尾采用P(I1 I0)判断LAD和POP指令的不同路径:若I1I0 =00，执行直接根据第二个字节目标地址ADDR从存储器取数的双字节LAD指令;若I1I0 !=00,则执行根据逻辑寄存器RB(R1 - R3)内容指定的目标地址从存储器取数的单字节POP指令。STO和PUSH指令的关系类似LAD和POP指令。

图示

描述已自动生成

表格

描述已自动生成

**3.3.6跳转系列指令**

图3-17是JMPR JMP指令的微程序流程图。从图中可以看出，JMPR指令只需要[11111]微指令就够了，但是为了节省OP码，两条跳转指令共用OP码“0001”，即共用第一条微指令[00001] ( 即使JMPR指令其实并不需要[00001]微指令)。从而可以在第一条微指令的末尾采用P3(I0 I1)区分两种跳转指令的不同路径:若I0I1=00，执行直接根据第二个字节目标地址ADDR跳转的双字节JMP指令;若I1I0≠00，则执行根据逻辑寄存器Rb (R1- R3)内容指定的目标地址跳转的单字节JMPR指令。

图示

描述已自动生成

表格

描述已自动生成

**3.3.7算术逻辑运算系列指令**

算术逻辑运算指令包括了2条单字节双操作数运算指令ADD、AND和2条双字节双操作数运算指令ADDI、ANDI， 其微程序流程图如图所示。为了节省微指令，所有的运算指令都在第一条微指令期间锁存74181运算器的控制端逻辑[S3,S2,S1,S0,M,CN]。 此外，2种双操作数的运算指令都采取在第一条微指令的末尾采用P3(I1I0)判断双字节和单字节指令的不同路径:若H =00,执行操作数分别来自逻辑寄存器RA和指令第二个字节(立即数IMM)的双字节指令;若I1I0≠00，则执行操作数全部来自寄存器的单字节指令。

图示

描述已自动生成

表格

描述已自动生成

●双操作数运算指令:操作数全部来源于逻辑寄存器RA和RB, DA\_ CLK和DB\_ CLK分别由微操作信号RA\_ BUS和RB\_ BUS驱动。

●双操作数运算指令:操作数之一来自 指令第二个字节(立即数IMM)，则DA\_\_CLK由微操作信号RA\_ BUS驱动，而微指令[11100]期间，DB\_ CLK则由微操作信号MEM\_OE驱动。

### 3.4硬件设计

**3.4.1实验概述**

本实验的主要内容是掌握基于微程序控制器的CPU组成结构，了解CPU的中断工作机制，熟悉CPU微指令设计，掌握机器指令的微程序实现方法。

本实验将设计一个微程序CPU,其中包括微程序控制器、运算器、存储器、寄存器堆及外部I0接口。定义一套较完备的机器指令集，编写每条机器指令对应的微程序，在CPU电路上运行基于上述机器指令集的机器语言程序，并且用汇编助记符(语言)加以注释。

**3.4.2微程序CPU架构**

如图所示，本实验的微程序CPU由微程序控制器通路(CONTROLLER)、时序电路(CLOCK)及数据通路组成。数据通路包括:程序存储器ROM、数据存储器RAM及通用寄存器R0~ R3; I/0接口;算术逻辑运算器(74LS181) 及附带的移位寄存器(74LS194);程序计数器(PC)、 ALU运算结果标志位寄存器(PSW)及其断点寄存器(BP\_ PC、BP\_ PSW)。数据通路的所有部件都共同挂在一条8位系统总线(BUS)上。

图中的微程序CPU架构如图所示，右侧是时序电路(CLOCK)和微程序控制器(CONTROLLER)，左侧则是由8 位系统总线(BUS)串联起来的数据通路。挂在总线BUS上的CPU部件包括:存储器ROM、RAM及其地址寄存器AR;指令寄存器IR;通用寄存器R0~R3;算术逻辑运算器( ALU)及其附属的缓存器DA (兼作移位功能)和DB;外设I/O接口;程序计数器(PC) 及其断点寄存器(BP\_PC);中断向量地址(IVA)。 此外，数据通路还包括了右边的运算结果标志位寄存器(PSW) 及其断点寄存器(BP \_PSW)。

图片包含 日程表

描述已自动生成

图示

描述已自动生成

**3.4.3时序电路**

微程序CPU的时序电路如图所示。图中的CLOCK电路是CPU的基准时钟电路，系统时钟CLK可以由方波信号源AUTO -CLK提供(双击信号源可以自行选择方波信号频率)或者通过开关MANUAL手动步进。当初始化信号ON =0或停机指令信号HLT=0,时钟CLK阻塞(强制CLK=0), CPU停机。

图示, 示意图

描述已自动生成

CLOCK电路右侧是一个JK触发器74LS73实现的微指令状态机。由于微程序控制器和数据通路相互独立，两者操作可以并行执行，如表3-10所示。所以，微程序CPU的微指令周期只需要T1和T2两个状态，时钟信号CLK驱动微指令状态机循环输出节拍序列{T1,T2}，使状态顺序转移: T1→T2→T1......

报纸上的文字

描述已自动生成

CLOCK电路左侧是初始化电路，手动按钮令复位信号RESET上升沿跳变，可以使信号ON=1。CPU启动仿真后，初始化过程十分简单，如下所述。

1)启动仿真后，时钟CLK选择从手动按钮MANUAL输入信号;

2)手动按钮使信号RESET跳变“1→0→1”，令信号ON=1, CLK允许输出，过程

CLOCK电路上方是NOP/HLT指令电路:当指令寄存器IR的OP码I7 I6 I5 I4 =0000的时候，空指令信号NOP=1,送往微指令计数器; OP码I7 I6 I5 I4=0000且I0=1的时候，指令信号HLT=0,时钟CLK阻塞，CPU停机(陷人“断点”)。跳出HLT指令“断点”的复位过程与上述初始化过程完全相同，区别在于初始化过程结束后，CPU进人第一条指令的取指周期T1节拍;而复位过程结束后，CPU进人HLT指令后续下一-条指令的取指周期T1节拍。

如图3-3所示，为了便于观测程序和微程序的运行，时序电路提供了双位的指令计数器I显示当前运行第几条机器指令，以及单位的微指令计数器MICRO- I显示当前运行指令计数器I所示指令中的第几条微指令。微指令抓数器MICRO -I由十进制计数器74LS160构成，基于信号ON (初始化过程)或T,节拍上升沿驱动递增，在指令周期末尾使能加载信号ML=0，在下一个指令周期开始时刻，重置MICRO-I的计数值为“1”， 重新计数。当以下条件之一成立时， 表示当前微指令是指令周期最后一条微指令，令ML=0。

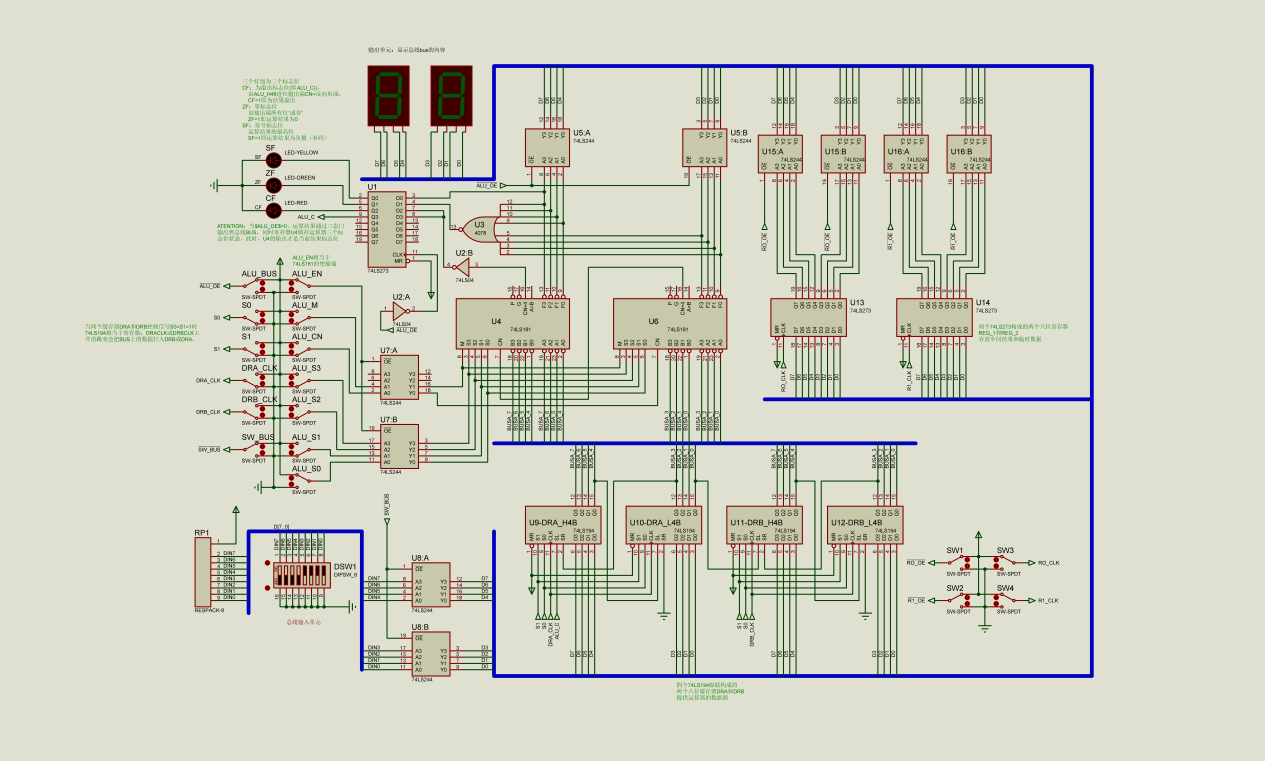
1)当前执行微指令中的判断位P2=1 (即P2非=0)。

2)空指令信号NOP=1且判断位P1=1 (特殊情况: NOP指令末尾)。

3)停机信号HLT=0; (特殊情况: HLT指令末尾)。

如图3-3所示，当加载信号ML=0,下一个指令周期开始的T1节拍上升沿令指令计数信号L\_ CNT=1,驱动指令计数器I (由两个计数器74LS160级联构成)递增。与此同时，指令清除信号IR\_ CLR =0，即指令周期开始之际，清空指令寄存器IR。

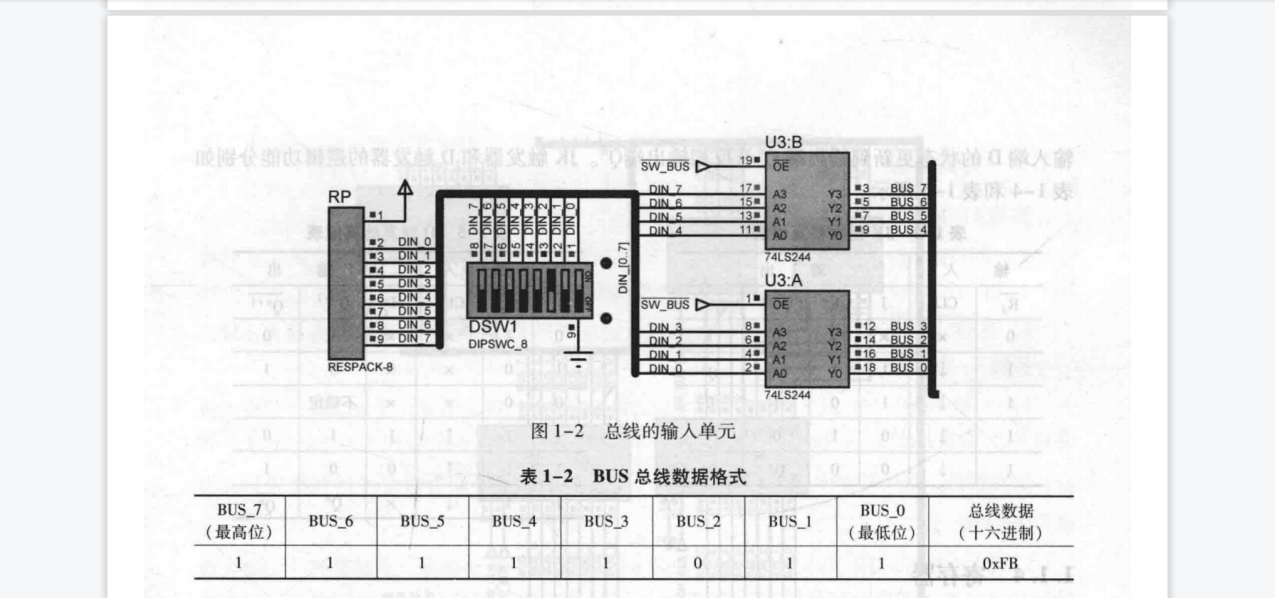
**3.4.4运算器**



运算器ALU通路如图所示。输入单元(拨码开关)通过三态门74LS244向运算器的总线BUS输人参与运算的数据，输出单元(数码管)显示总线BUS的内容;通路右上方则是两个74LS273构成的8位寄存器REG0和REG1，用来存放运算过程的中间结果和临时数据。除了上述电路以外，通路中还有以两个运算器74LS181串行进位形式构成的8位运算器电路(包括运算控制信号电路和运算标志位锁存电路)。其中，参与运算的两个8位数据由总线LINKBUS015]输入，执行的运算类型则由ALU控制端S~So、M和CN决定。

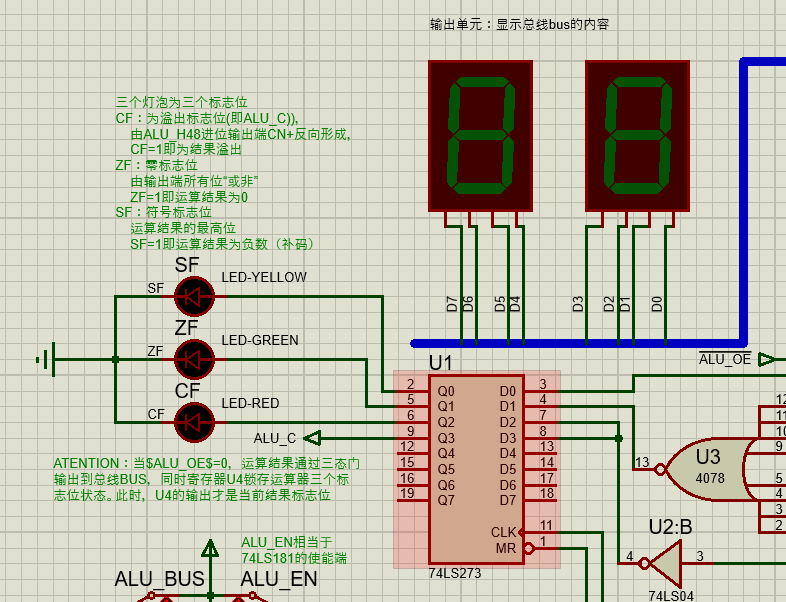
****

总线输人单元如图1-2所示。其中，拨码开关DSW1和上拉电阻RP连接一个8位的输人总线DIN，用来设置输入DIN总线的数据;低电平有效的开关SWBUS控制三态门74LS244(逻辑功能见表1-1)，实现总线DIN与总线BUS的一对一连通(BUS总线数据格式见表1-2)。总线输出单元则是一对数码管，用来显示总线BUS上的当前数据。

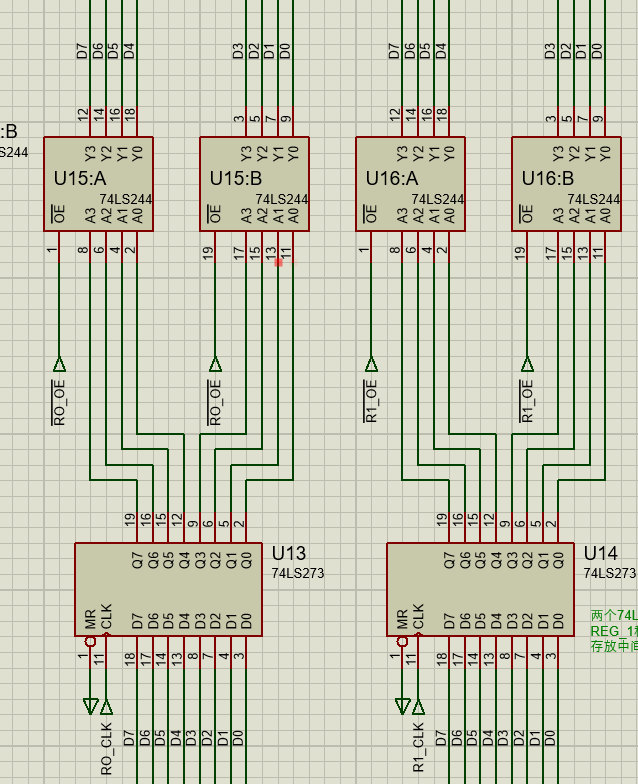
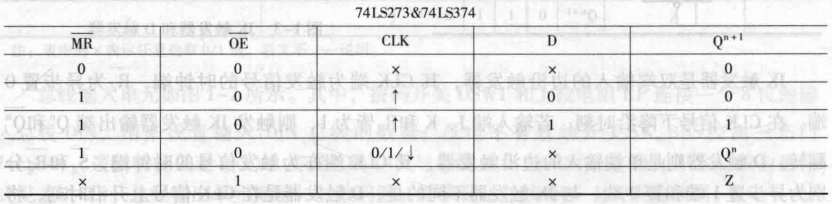




运算器有三个标志位:溢出标志位CF(即ALUC)由ALUH4B进位输出端CN+4反向形成，CF=1即运算结果溢出;零标志位ZF由输出端所有位“或非”形成，ZF=1即运算结果为零:符号标志位SF是运算结果最高位。SF=1即运算结果为负数(补码)。因为74LS181没有使能端，所以运算器通过三态门UA和UB控制上述ALU控制端。

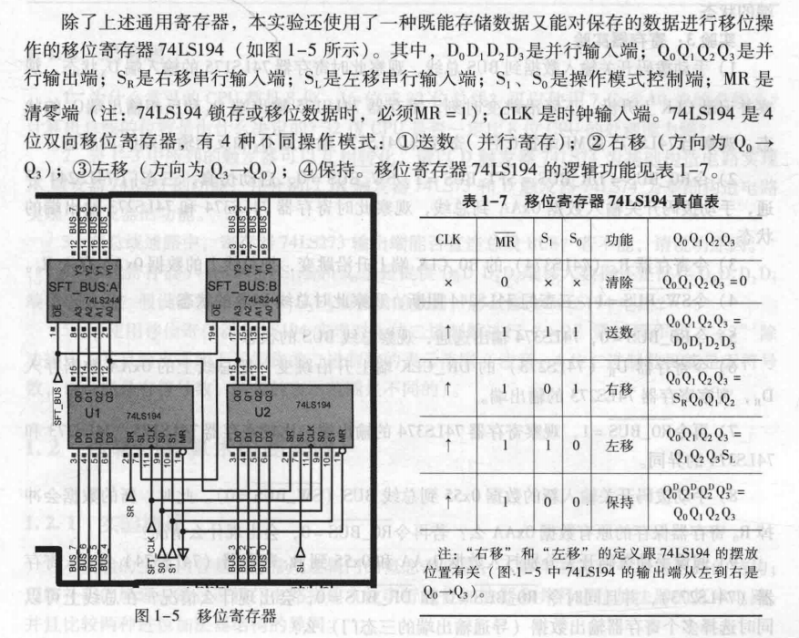
****

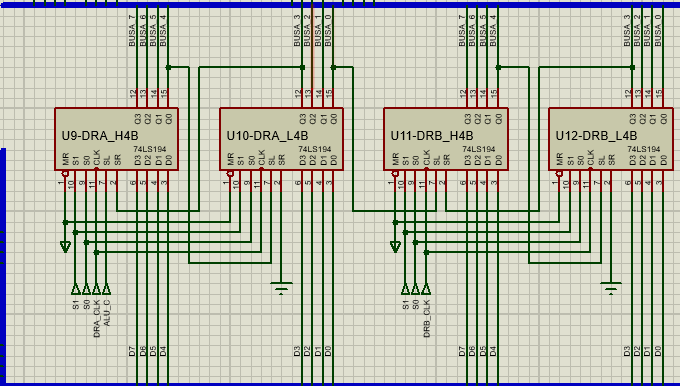
8位寄存器74LS273的逻辑功能则相当于8个并联的D触发器74LS74组合D~D，为并行输入端Q。~Q为并行输出端(省略了反相输出端Qx)，CLK端为时钟脉冲(上升沿触发);输出端 Q的状态只取决于CLK端时钟脉冲到来时刻输人端D的状态，输出状态的更新发生在CLK端脉冲的上升沿。8位寄存器74LS374与74LS273的逻辑功能基本相同，两者的区别是:74LS273省略了置0端MR，其输出端必须经过三态门74LS244连接到数据总线BUS



除了上述通用寄存器，本实验还使用了一种既能存储数据又能对保存的数据进行移位操作的移位寄存器74LS194(如图1-5所示)。其中，DDDD是并行输入端;0000是并行输出端;S，是右移串行输入端;S是左移串行输入端;SS是操作模式控制端:MR是

清零端(注:74LS194锁存或移位数据时，必须MR=1);CLK是时钟输入端。74LS194是4位双向移位寄存器，有4种不同操作模式:①送数(并行寄存);②右移(方向为Q Q;③左移(方向为Q→0);④保持。移位寄存器74LS194的逻辑功能见表1-7

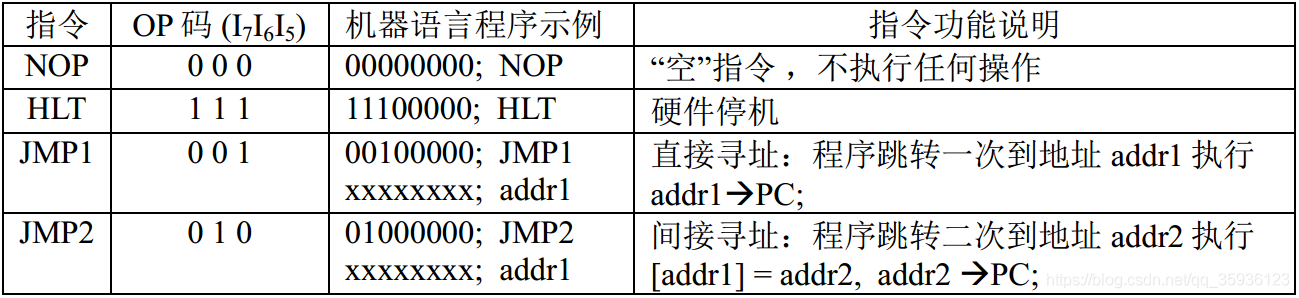




**3.4.5微程序控制器**

1. 设计概述

本CPU目前有4条指令，功能是“程序跳转”。



按照上述指令表的指令格式，用户可以编写一段机器语言程序存放在程序存储器PROGRAM中，如下图所示。其中每一个存储器单元存放一个字节的数据，对应唯一的8位二进制地址（由地址寄存器AR锁存）。若用户需要访问程序存储器的某个单元，须由程序计数器PC提供该单元的地址，才能从程序存储器取出该单元中的数据。因为程序是顺序访问的，所以程序计数器PC是由两个计数器74LSl63级联构成的一个8位递增计数器PC。当前指令从程序存储器PROGRAM取出，并锁存到指令寄存器IR后，PC自动执行PC+1操作，指向相邻下一条指令。

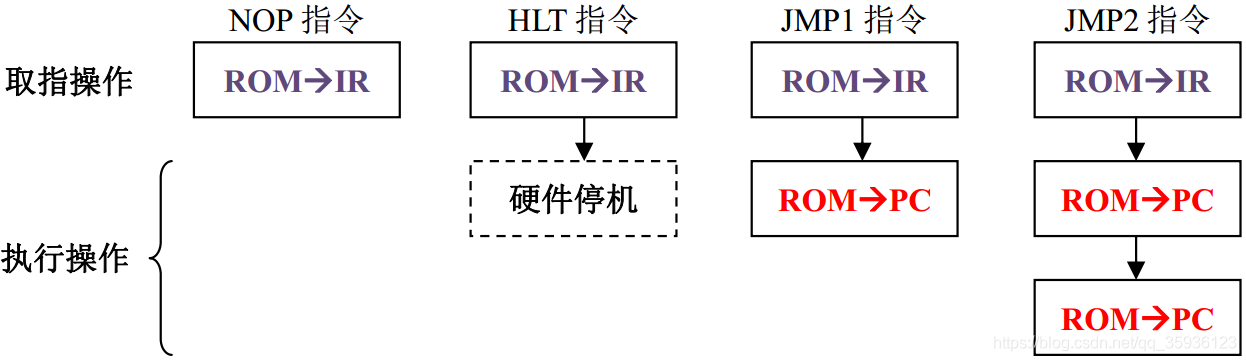
1. 数据通路

图表

中度可信度描述已自动生成

数据通路如图所示，由并联在单条8位总线BUS上的三个部件组成：指令寄存器IR、程序计数器PC和程序存储器PROGRAM。

仔细分析上述图和表可知，数据通路的各条指令状态图如下图所示：所有指令的取指操作都是相同的，即是上图中的指令流ROM→IR：CPU从程序存储器PROGRAM取出指令，经过总线BUS流向指令寄存器IR。NOP和HLT指令只有上述取指操作，没有执行操作（HLT指令取指后硬件停机）；而JMP1和JMP2指令除了上述取指操作外，实际只有一种执行操作，即上图的数据流ROM→PC：CPU从程序存储器PROGRAM取出数据，经过总线BUS流向程序计数器PC。两种跳转指令不同之处在于：JMP1指令的第二字节是目标地址（直接寻址），只要一次数据流（ROM→PC）就把目标地址送入PC；而JMP2指令的第二字节是存放目标地址的存储器单元地址（间接寻址），需要连续两次数据流（ROM→PC）才能把目标地址送入PC。



1. 微程序原理

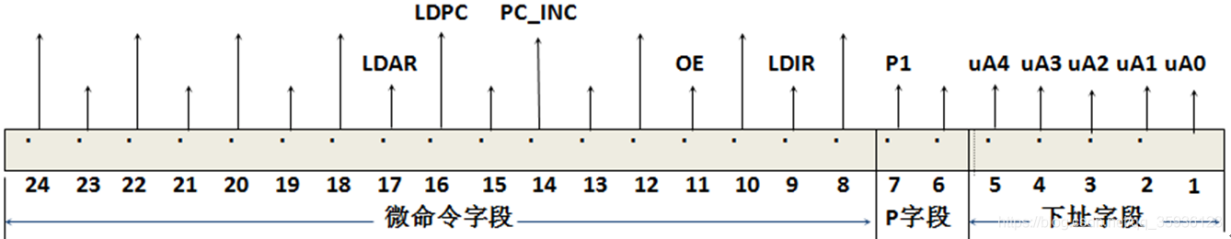
在上图中，虽然微程序控制器指令的状态类型只有两种（指令流和数据流），但是每一条指令拥有的状态数目都不尽相同，其中最关键问题是如何根据不同的指令来判断状态的转移。因此我们可以采用微程序原理来解决这个问题：图2中每一条指令都是一个任务，一个状态则对应一条微指令。若干条微指令组合成一段微程序，解决相对应的任务。

微指令的字长设为24位，结构如下图3所示：

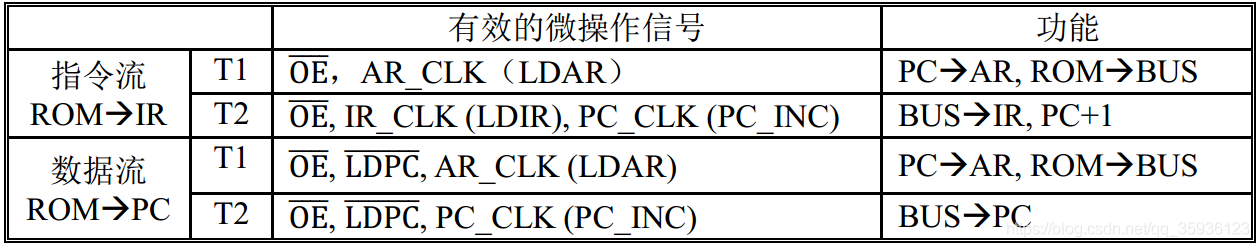
1-5位表示该微指令执行后，下一条微指令的地址 [uA4, uA0]（即下址转移方式）；

6-7位是判断字段Px：P1=1表示该微指令是取指微指令；，P2空缺。

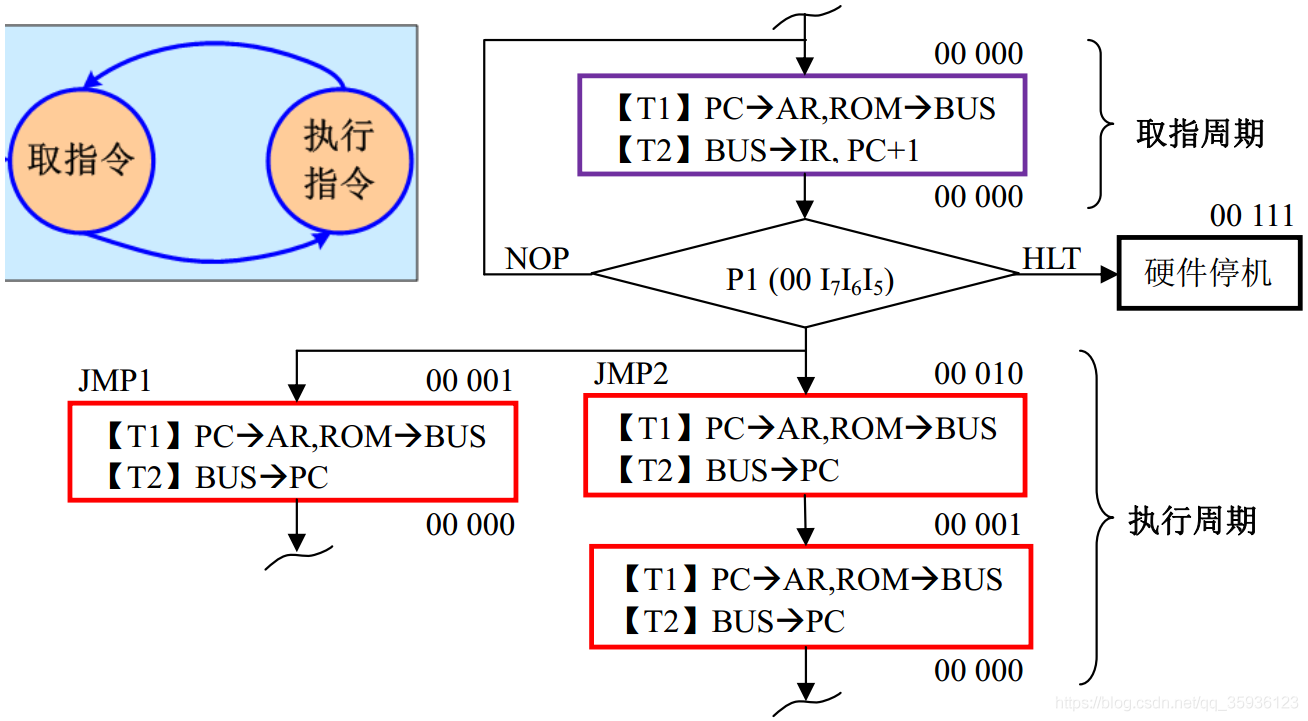
8-24位是微命令字段：微命令即是图1所示数据通路中的微操作信号，其中某位置“1”，表示该位的微命令有效；反之，置“0”则表示该位的微命令无效。



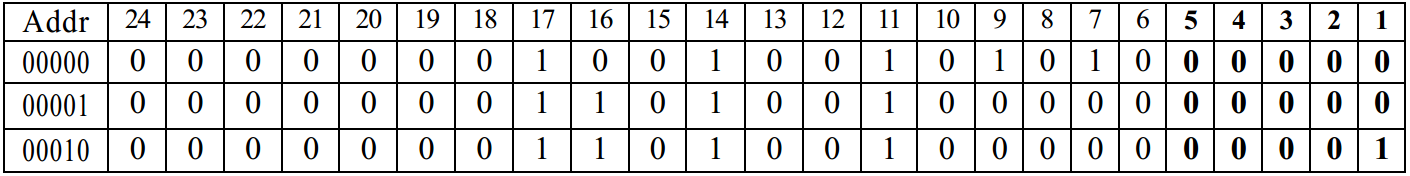
仔细观察数据通路图，可以发现指令的取指或执行过程都是指令或数据从一个部件打入总线BUS，再从总线BUS打入另一个部件的过程。为了保证上述操作先后次序，指令流（ROM→IR）和数据流（ROM→PC）都分为T1和T2两个周期：T1周期，信息从源部件（例如程序存储器PROGRAM）打入总线BUS；T2周期，信息从总线BUS打入目标部件（例如指令寄存器IR或者程序计数器PC），如下表所示。因此，在图所示的微操作信号中，除了信号(存储器输出使能)和(PC加载使能)是全过程有效外，其他信号需要与T1或T2周期节拍信号逻辑“与”，产生新的边沿触发信号，在指定周期开始时刻上升沿跳变，例如AR\_CLK=LDAR•T1，IR\_CLK=LDIR•T2，PC\_CLK=PC\_INC•T2



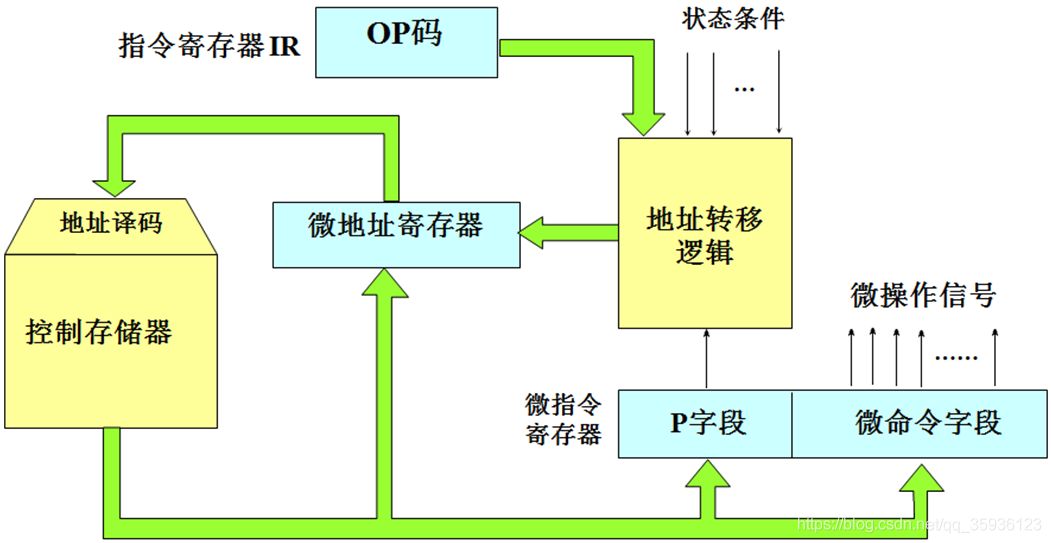
综合上述微指令结构图和微操作信号列表，通过分析微程序控制器指令状态图，可以得到如下图所示的微程序流程图：图中每一个方框在时间上表示一个微指令周期，包括T1（源部件→总线）和T2（总线→目标部件）两个周期；在空间上表示一条微指令，通过一系列微操作信号使得信息从某个源部件经过总线BUS到达目标部件。图中每个方框的右上方是对应微指令的地址，右下方是对应微指令的下一条微指令的地址（简称下址）。



上述微程序流程图中，最上方首先执行的方框是公共的取指微指令，即指令流（ROM→IR）。取出指令后， P1菱形框表示指令译码及地址转移：根据当前指令OP码的[I7,I6,I5]位形成其执行周期第一条微指令地址[0,0,I7,I6,I5]，从而选择该指令的执行周期。菱形框下的四条路径对应指令列表1所述的四条指令的执行周期，其中每个方框是一条执行微指令，即数据流（ROM→PC）。值得注意的是，NOP指令和HLT指令只有取指周期，没有执行周期。NOP指令的OP码是000，取指后译码得到的第一条微指令地址仍为[00000]，即直接返回下一条指令的取指周期。而HLT指令的OP码是111，译码后直接令硬件停机。在所有路径末尾，最后一条微指令的下址[uA4-uA0]都必须是取指微指令地址[00000]，即一条指令结束后必须返回取指微指令，准备取出下一条指令。如图左上方所示，整个数据通路的运行过程就是不断循环的取指令和执行指令。图中总共有三条微指令，其编码如下表所示（具体位置的微命令含义请参考微指令结构图）。



负责执行图所示的微程序流程图的CPU部件是微程序控制器(CONTROLL UNIT)，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，如下图所示。CPU启动或复位后，微地址寄存器清零，控制存储器从地址[00000]开始输出微指令。如前述微指令结构图所示，微指令包括了控制字段、下址字段和判断字段。控制字段即下图中的微命令字段，直接输出微操作信号执行当前微指令；下址字段锁存在微地址寄存器，待当前微指令执行完后，再从控制存储器取出下一条微指令。若当前微指令是取指微指令，则P字段启动地址转移，根据指令寄存器IR中的OP码修改微地址寄存器，转向指令执行周期的第一条微指令。



存放上述微指令列表的存储器电路如下图所示：微指令存储器字长24位，由3个2764芯片MROM1-3组成，其输出端则连接着微指令寄存器MDR1-3（寄存器74LS273和74LS175构成）。在系统启动（信号ON=1）或T1周期开始（信号T1=1）时刻， MROM1-3输出当前微指令的微操作信号，锁存在MDR1-3，送往数据通路执行。部分微操作信号（LDAR、LDIR、PC\_INC）与T1或T2节拍组合，产生边沿触发信号(AR\_CLK、IR\_CLK、PC\_CLK)，在T1或T2周期开始时刻上升沿跳变。

图表

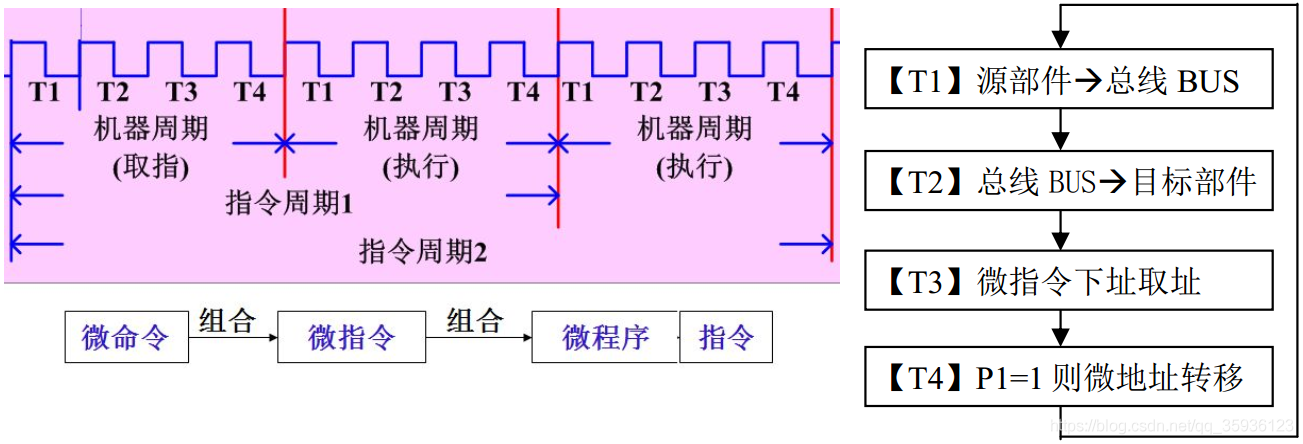
描述已自动生成 图示, 示意图

描述已自动生成

如下图所示，控制器的微地址寄存器字长五位(MA4-MA0)，由触发器74LS74组成，其输入端通过NMABUS总线连接当前微指令的下址字段[uA4-uA0]，其输出端通过控制存储器的地址总线MABUS送到控制存储器的地址端A4~A0。值得注意的是，上述微程序控制器结构图6中的地址转移逻辑（即微程序流程图中的菱形框P1）在图中对应的就是三个三路与非门74LS10。在取指周期末尾，微指令下址字段本来是[00000]；然而判断字段P1=1，启动地址转移逻辑，根据指令寄存器IR的OP码[I7,I6,I5]生成信号#SET\_MAx=0，强制把微地址寄存器MA4-MA0置位为[0,0,I7,I6,I5]，即该指令执行周期的第一条微指令地址。

上述微程序的地址转移过程需要在微指令周期增加T3和T4两个周期：T3周期，当前微指令的下址字段[uA4-uA0]通过NMABUS总线打入微地址寄存器MA4-MA0，进而通过地址总线MABUS送往微指令存储器MROM1-3的地址端，使其输出下一条微指令；T4周期，若当前微指令是执行周期微指令，则P1=0，无任何操作；若当前微指令是取指周期微指令，则P1=1，启动地址转移逻辑，重置微指令地址，跳转到当前指令的执行周期第一条微指令。

综上所述，一条CPU指令就是一段微程序，其中包含若干条微指令（至少有一条是取指微指令）。所以，“微程序”时序如下图9（左）所示：每个指令周期都包含了若干条微指令的机器周期（即微指令周期），其中至少有一个取指微指令周期。而且，每一条微指令的运行过程都可以看成是一个状态机，如下图9（右）所示：状态机有4个状态{T1,T2,T3,T4}，每个状态【Tx】完成从取指、执行到判断下址的相应任务，状态转移T1→T4的一次循环即是一个微指令周期。因此，每个微指令周期内部包含了四个节拍信号Tx，对应“微指令”状态机的4个状态{T1,T2,T3,T4}，状态机周而复始在四个状态【Tx】之间顺序转移。



**3.4.6时序发生器**

CPU时序发生器如下图所示，主要功能是为上述微程序控制器提供时序控制。其中最核心的状态转移电路是由两个D触发器组成的一个2位扭环计数器，输出节拍序列{T1,T2,T3,T4}={00,01,11,10}。CLK为整个CPU电路的时钟信号，可以由手动按键MANUAL\_CLK或方波信号源AUTO\_CLK生成（双击信号源可以自行选择方波信号频率）。

时序发生器还提供了硬件电路实现HLT指令的停机功能（即断点）。当指令寄存器IR的OP码I7I6I5=111的时候，停机信号#HLT=0，阻塞CLK输出，CPU停机在HLT指令的取指周期T2节拍上。跳出HLT指令断点的复位过程与上述初始化过程相同，信号 #RESET=0令指令寄存器IR清零，OP码[I7,I6,I5]=000，则#HLT=1，跳出“断点”；同时，扭环计数器强制为状态T1={00}。#RESET=1，复位成功，CPU进入HLT指令后续下一条指令的取指周期T1节拍。

此外，为了观测微程序的运行，时序发生器电路提供了一个由节拍信号T1上升沿驱动的双位微指令计数器MICRO-I（两个十进制加法计数器74LS160级联构成），通过数码管显示当前运行第几条微指令，显示范围是1~99，如下图右边所示。

图示, 示意图

描述已自动生成

PROGRAM烧写内容

表格

描述已自动生成

MROM1烧写内容

文本, 表格

描述已自动生成

MROM2烧写内容

文本, 表格

描述已自动生成

MROM3烧写内容

表格

描述已自动生成

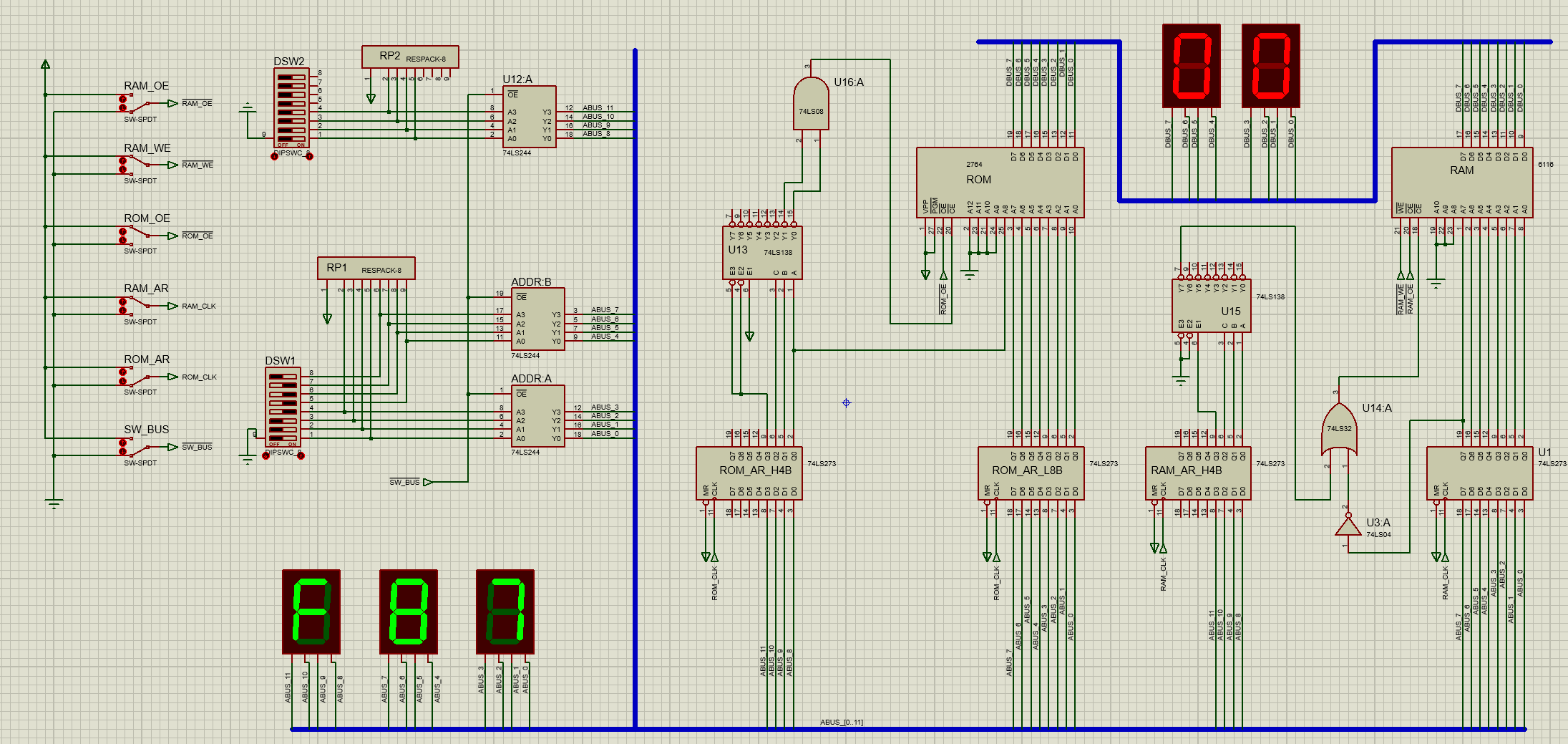
**3.4.7存储器**

ROM存储器一般容量较大，在系统停止供电的时候仍然可以保持数据；ROM只能读出数据，不能写入数据。而RAM存储器一般容量较小，在系统掉电之后就丢失数据；RAM即可读出数据，又可写入数据。

图①所示的存储器电路由ROM、RAM及两条总线构成：ROM存储器是2764(8K×8bit)，RAM存储器是6116(2K×8bit)，两条总线分别是12位地址总线ABUS\_[0..11]和8位数据总线DBUS\_[0..7]，其中数据总线上的8位数值由两个4位的红色数码管来显示。

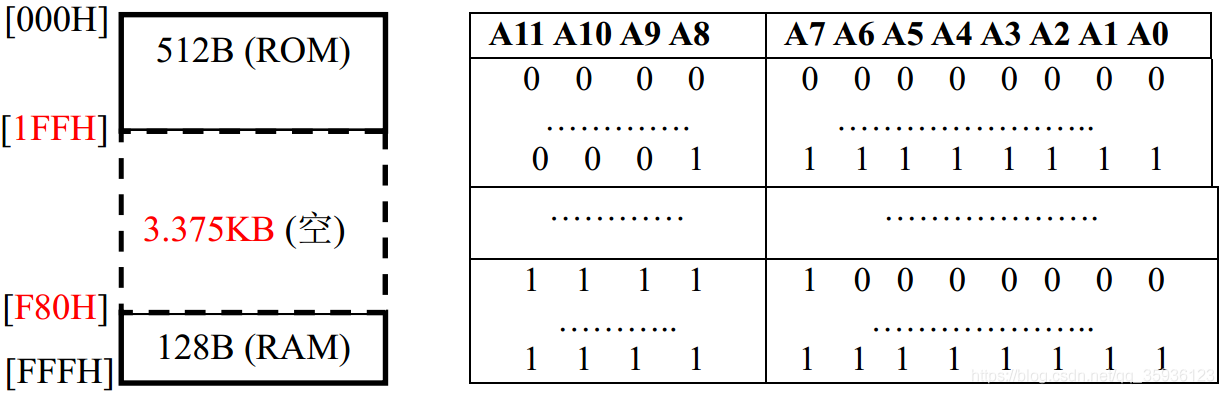
ROM芯片2764的数据线D0-D7接到数据总线，地址线A0-A8由地址锁存器74LS273给出，用来对ROM片内存储单元寻址。其余地址线A9-A12接地；2764有两个控制端：#CE(片选)、#OE(读)；RAM芯片6116的数据线D0-D7接到数据总线，地址线A0-A7由地址锁存器74LS273给出，用来对RAM片内存储单元寻址。其余地址线A8~A10接地； 6116有三个控制端：#CE(片选)、#OE(读)、#WE(写)。

图①.存储器电路原理图



存储器电路设计的最重要环节是存储器与地址总线的连接，因为连接方式决定了存储器地址空间的映射关系，即决定了每个存储器芯片在整个存储空间中的地址范围。12位地址总线的理论地址空间为4K（000H-FFFH），本实验分配其中最低的512地址为ROM区（000H-1FFH），最高的128B地址为RAM区（F80H-FFFH），其余留空，如下表所示：

表A.存储器得地址空间范围



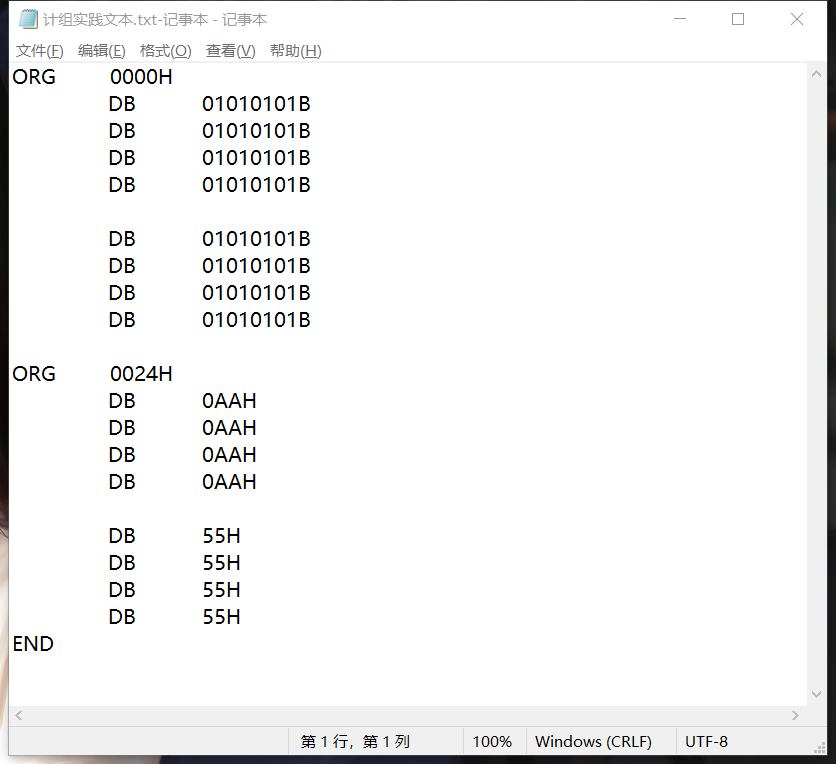
存储器电路的设计一般将地址总线区分为低位地址线和高位地址线两部分：低位地址线直接和存储器芯片的地址信号连接作为片内地址译码，而高位地址线的连接主要用来产生片选信号（称为片间地址译码），以决定每个芯片在整个存储系统中的地址范围。

在上图1的存储器电路中，12位地址总线分为低8位地址线和高4位地址线。低8位地址线ABUS\_0-7分别与ROM和RAM芯片的地址线A0-A7共用；高4位地址线ABUS\_8~11则通过两个3-8译码器进行译码。低位3-8译码器U13最低2位之一输出有效，则片选ROM芯片（“负逻辑”判断，使用与门）；同样，高位3-8译码器U15最高1位输出与地址线A7同时有效，则片选RAM芯片（“负逻辑”判断，A7先反向，再使用或门）。值得注意的是：相同的存储器地址空间映射，可以有不同的片选电路实现方法（例如使用“正逻辑”判断）。

ROM模型得批量导入数据实现方法

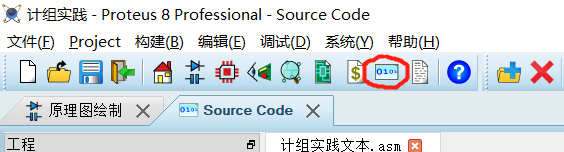
本次操作借用8051单片机的伪汇编指令来实现如何把一组程序或数据一次性批量导入ROM中，使ROM在往后的实验中可以充当程序或数据存储器的角色。具体操作步骤如下：

1. 新建txt文件，把.txt后缀改名为.asm后缀，然后按照下列伪汇编指令格式输入数据：



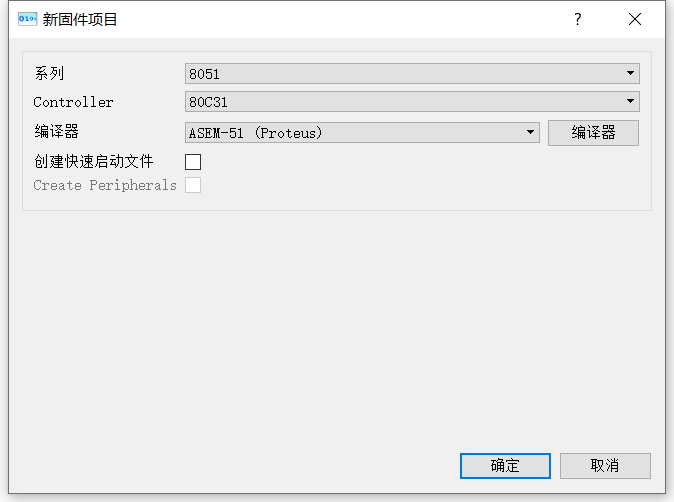
注：上述代码中 的语句“ORG xxxxH”规定该语句后所跟数组存储的首地址，数组末尾必须以其他ORG语句或“END”作为结束。在asm文件中可以使用多个“ORG”语句来规定在存储器的不同位置存放不同长度的数组。在“ORG”语句后的“DB xxxxxxxxB”或“DB xxH”语句都表示一个存储单元存放的8位数据，前者是二进制，x表示0或1；后者是十六进制，x表示09和AF（注：若十六进制数据xxH大于A0H，则要写成0xxH，例如“DB 0A0H”）。

1. 在proteus仿真环境的菜单栏里点击 Source Code图标（红色圈标注的地方），弹出Source Code标签页，如下所示：

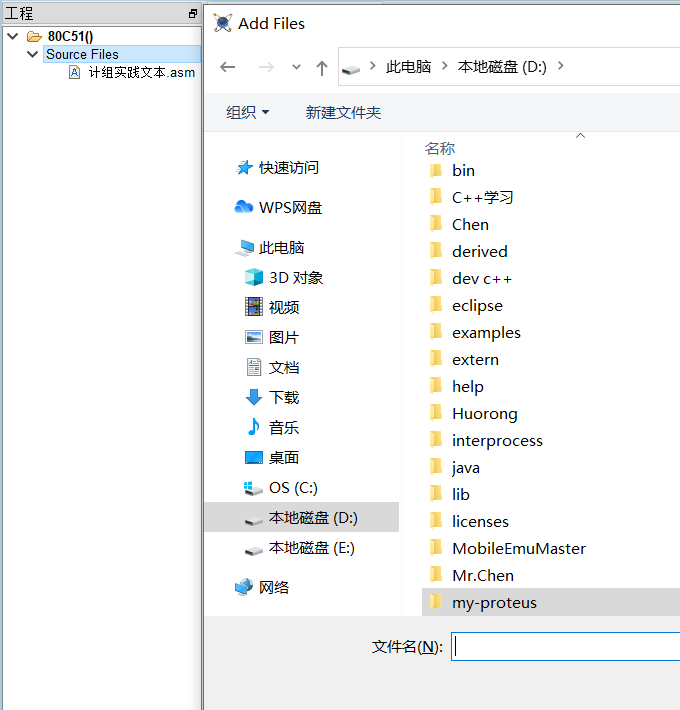
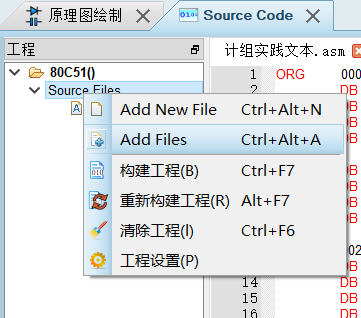


如果该项目没有打开过asm文件，则打开的标签页为空，需要在菜单栏里选择Source Code->Project->新建工程，弹出的对话框如下图所示。此处，在系列选项中选择8051， Contoller选项中选择80C51，则编译器选项会自动选择ASEM-51；注意：下方创建快速启动文件选项不勾选！最后按下“确定”按钮。

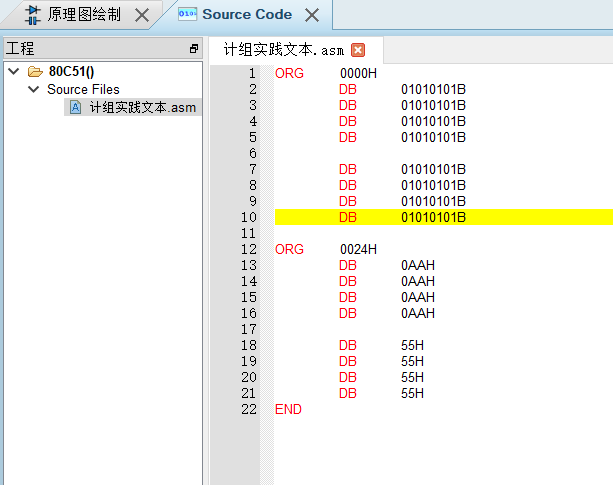
注意：如果使用的proteus版本内没有支持8051内核，则在Source Code的Family和Controller选项中找不到8051/80C51的内容。这意味着在proteus内无法编译asm文件，必须借助支持8051内核的第三方编译软件（例如Keil 软件，1k以内代码有免费版本）把asm文件编译成hex文件，再导入proteus的存储器，生成hex文件后，请直接跳到以下步骤（7）操作。



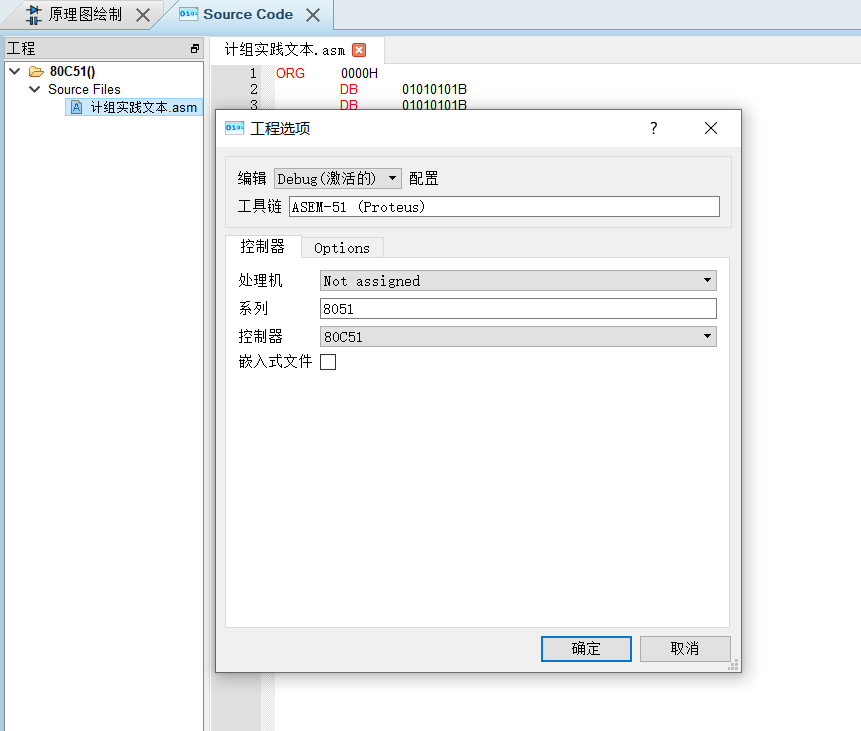
1. 如下图所示，标签页上会出现“”\*80C51()”的新project后，右键单击下方的Source Files，在弹出菜单中选择Add files（或Import Existing File）选项，在弹出的视窗中选择所需的asm文件，按确定就添加到当前的project中。



1. 在project中已经添加的asm文件上双击，右侧会打开asm文本内容，如下图所示。如果该project之前已经添加且编译过asm文件，则弹出的Source Code标签页右侧会直接显示asm文本内容，可以直接对asm文本内容进行修改和保存。如果要更换asm文件，可以右键单击asm文件，菜单列表中选择Remove file删除当前asm文件，然后选择Add files（或Import Existing File）选项，在弹出的视窗中重新选择新的asm文件。

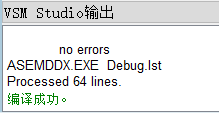


（5）如下图所示，若第一次编译asm文件，则右键单击asm文件，在弹出菜单点击“工程设置”，弹出框的“控制器”选项选择80C51，注：要去掉勾选“嵌入式文件”！确保编译生成的hex文件在project当前目录下，最后点击确定选项。



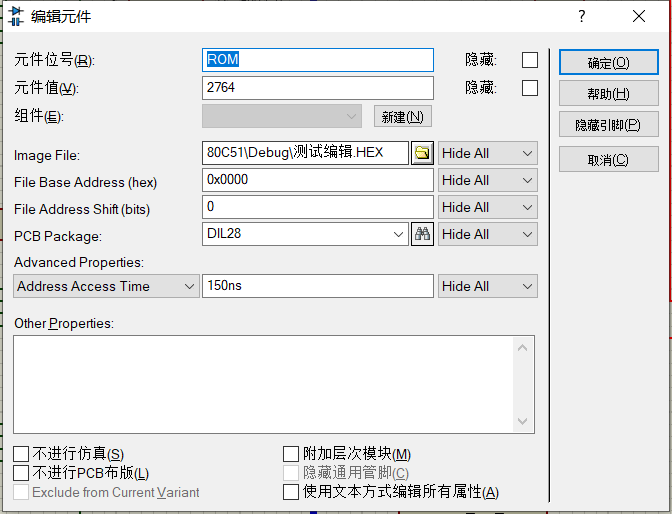
（6）然后，右键单击asm文件，在弹出框内点击Build Project执行编译。编译成功后，在下方的显示栏里会出现“编译成功”字样。此时，在项目当前文件夹的子文件夹80C51的孙文件夹Debug里面，就有编译后生成的hex二进制文件（请注意看文件的修改日期，确认是最近编译的文件）。该hex文件默认编译后的文件名为Debug.hex。

注意：一个良好的编程习惯是每次编译生成hex文件后，就重命名为跟asm源程序一致的文件名，然后与同名asm源程序放在另外的固定目录下。否则下一次编译其他asm源程序的时候，会在相同路径生成重名的Debug.hex，把以前编译的hex文件覆盖。

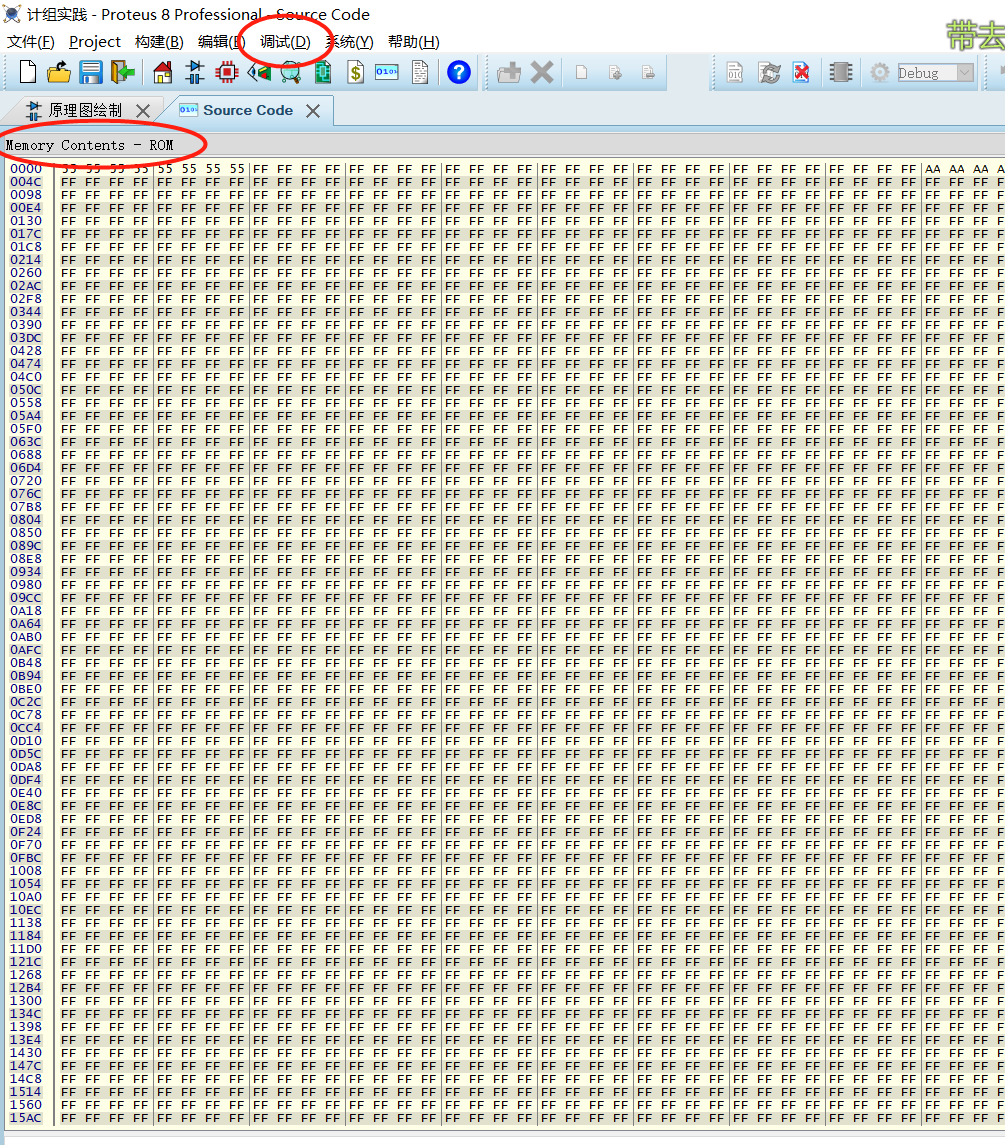


（7）双击ROM 芯片，弹出如下图所示的对话框：在Image File中选择所需烧写的hex文件，点击OK，ROM加载成功，操作结束。

注意：尽量避免直接加载路径“\80C51\Debug”里的Debug.hex文件，因为每一次编译asm源程序，Debug.hex都会自动刷新。但是，相同的文件名很容易混淆，难以确定加载的hex文件与哪个asm源文件对应。最好在每次编译完成后，把Debug.hex文件从路径“\80C51\Debug”里取出，另存在其他文件夹中，并且根据源文件asm的内容重命名编译生成的hex文件。



（8）启动proteus仿真，在仿真过程下按暂停；点击菜单栏里的“调试”，在弹出的菜单列表最下方有“Memory Contents-ROM/RAM”，如下图所示。注意：仿真工程有多少个存储器，就有多少个Memory Contents可供选择。点击需查看的Memory Contents选项，则弹出其对应的存储器内容：其中蓝色为该行的首地址（对应的是左边第一个黑色数据xxH），黑色为存储单元中的数据，按顺序从左到右，从上到下排列显示。可以通过调整显示框的宽度来改变蓝色的行首地址显示，方便查看某一个固定的地址。值得注意的是，图中显示的存储器地址是存储器地址线（A0~A7）所定义的地址。而实际的地址空间不仅要看存储器的地址线，还要参考各存储器片选信号的译码逻辑电路。此外，除非被新的hex文件覆盖，否则通过hex文件烧写定义的数据段会永远存在。所以，上图中显示的二进制数据内容比当前hex文件对应的asm源程序中定义的数据要多.



如果仿真出现下图所示的报错，需要加载的时候，要留意在移动或修改项目的时候不要破坏原有的ROM加载路径。

##### 3.5实验总结

遇到的问题及解决方法：

1.指令和微指令入口的转化方面不太理解，通过查阅资料，将指令的操作码作为地址转移逻辑寻找微指令地址解决问题。

2.对于设计好的微程序如何烧写进入MROM不太了解，通过查阅资料，学习了如何进行ROM的批量导入

3.微指令位数较多比较复杂，烧写过程不够细心导致出现错误，进行再三检查得已解决

4.对P1P2P3判别掌握不熟练，导致微指令下地址的形成出现问题，通过反复巩固得已解决

# 附录4 ：参考文献

[1] 白中英.计算机组成原理（第四版）.北京：科学出版社，2008年1月

[2] 白中英，戴志涛.计算机组成原理（第五版）.北京：科学出版社，2013年3月

[3]《基于Proteus的计算机系统实验教程——逻辑、组成原理、体系结构、微机接口》