第二章　计算机组成原理实验

2.1 总线与寄存器实验

一、实验目的

1、 掌握总线以及数据通路的概念及传输特性。

2、理解锁存器、通用寄存器及移位寄存器的组成和功能。

二、实验内容

构建一条 8 位单总线的寄存器数据通路，将若干寄存器连接起来。

通过拨码开关输入原始数据到某个寄存器；或者从一个寄存器向另一个寄存器赋值。同时，利用移位寄存器实现数据的置数、左移、右移等功能 。

比较以下各组器件之间的异同：触发器 74LS74 和 74LS175；触发器 74LS175 和寄存器 74LS273；寄存器 74LS273 和 74LS374；寄存器 74LS273 和移位寄存器 74LS194。

三、实验器件

1、D 触发器（74LS74、74LS175）、三态缓冲器（74LS244）。

2、寄存器（74LS273、74LS374 ）和移位寄存器（74LS194）

四、实验原理

总线是多个系统部件之间进行数据传送的公共通路，是构成计算机系统的骨架。借助总线连接，计算机在系统各部件之间实现传送地址、数据和控制信息的操作。因此，所谓总线就是指能为多个功能部件服务的一组公用信息线。

总线示意图如图 2-1 所示：输入单元、输出单元、寄存器、存储器及其地址寄存器等不同的设备挂在同一条总线上。这些设备都需要有三态输出控制，保证任何时刻总线上只有唯一的数据存在。

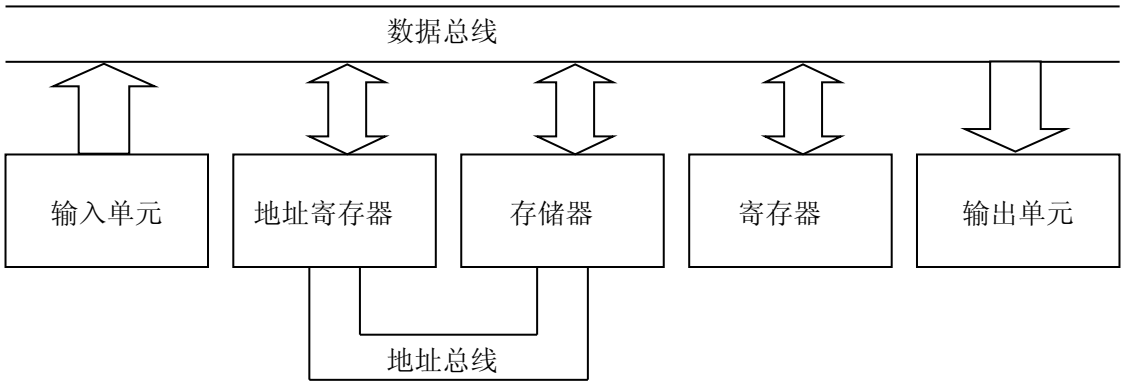


图 2-1 总线示意图

本实验的寄存器数据通路如图 2-2 所示，本通路包含了输入单元（拨码开关）、输出单元（数码显示管）和寄存器堆。输入单元如图 2-3 所示，本身就是一个 8 位的输入总线

DIN_[0..7]（如表 2-1 所示）。DIN 总线上挂靠的拨码开关和上拉电阻用来设置输入数据。低电平有效的开关 SW_BUS\控制三态门 74LS244，实现总线 DIN_[0..7]与数据总线 BUS_[0..7]的连通。三态门 74LS244 逻辑功能如表 2-2 所示。输出单元即是一对数码管，用来显示数据总线 BUS_[0..7]上的当前数据。

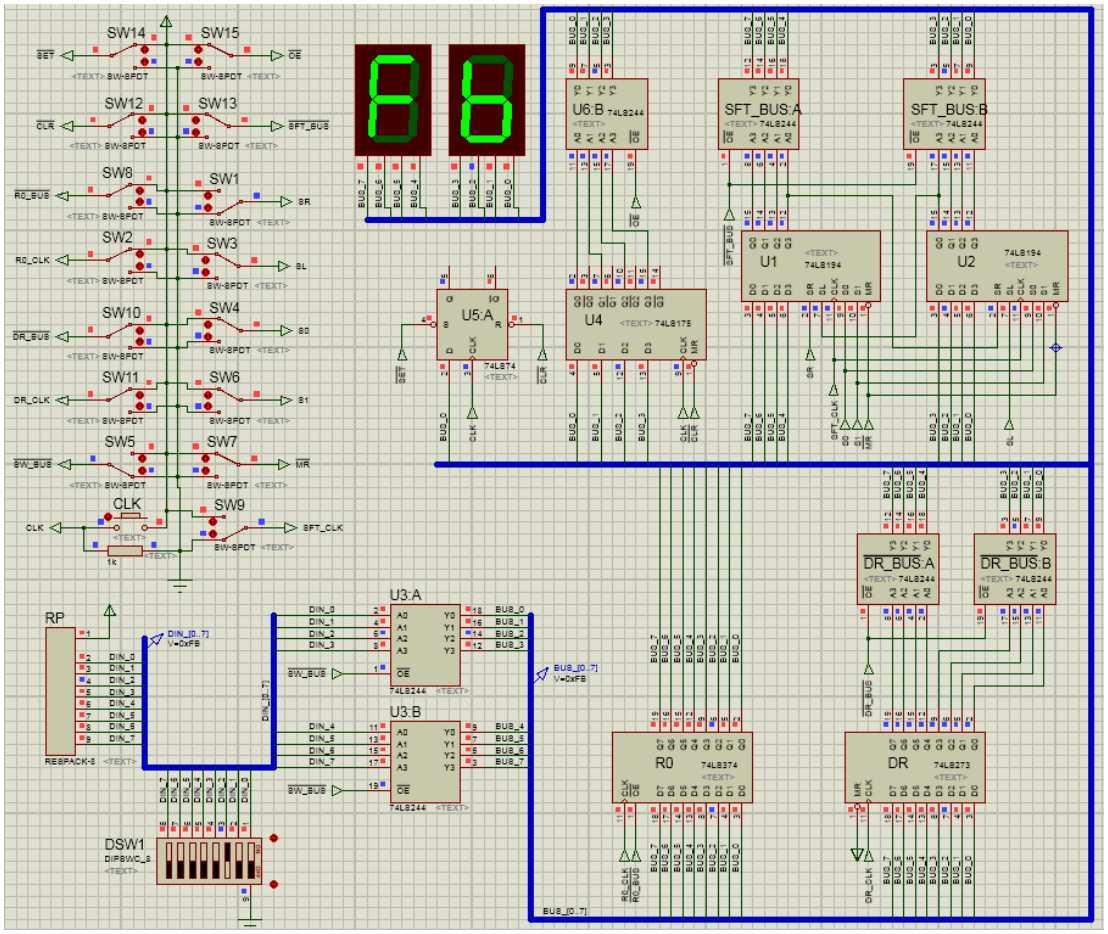


图 2-2 寄存器数据通路图

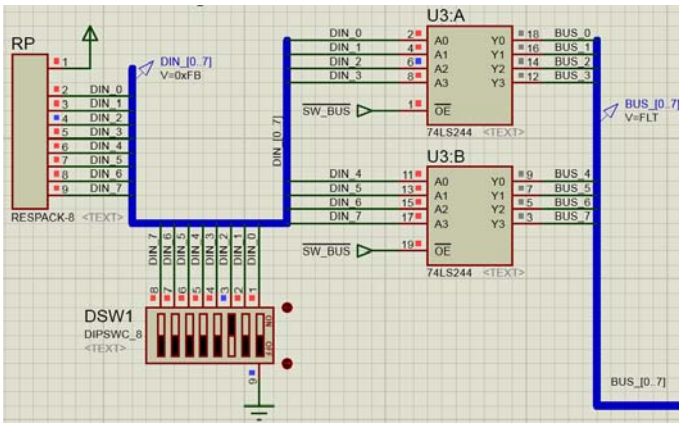
表 2-1 总线逻辑

DIN_7	DIN_6	DIN_5	DIN_4	DIN_3	DIN_2	DIN_1	DIN_0	总线数据 (16 进制)
1	1	1	1	1	0	1	1	0xFB H

表 2-2 74LS244 逻辑功能表

输　入		输　出
A	$\overline{OE}$	Y
0	0	0
1	0	1
×	1	Z

图 2-3 输入单元



寄存器堆由 D 触发器（74LS74、74LS175）构成的锁存器、通用寄存器 R0（74LS374）和 DR（74LS273）、移位寄存器（74LS194）构成。其中除了 74LS374 器件自带三态门结构外，其余锁存器和寄存器的输出都经过三态门（74LS244）和数据总线 BUS_[0..7]相连，以确保按照信息传输的要求恰当有序的控制锁存器和寄存器的数据输出到总线，任何时刻总线数据都不会发生冲突。

图 2-2 中已将本实验需要连接的控制开关标明在左上方：其中除 CLK、R0_CLK、DR_CLK 和 SFT_CLK 为上升沿控制信号，其余开关均为电平控制信号。

上升沿有效的开关 CLK(74LS74 和 74LS175 共用)、R0_CLK(74LS374 使用)、DR_CLK(74LS273 使用)和 SFT_CLK（74LS194 使用），负责把总线数据打入各自的器件；

低电平有效的开关 $\overline{OE}$（74LS74、74LS175 的三态门控制）、$\overline{R0\_BUS}$（74LS374 输出控制）、$\overline{DR\_BUS}$（74LS273 的三态门控制）和 $\overline{SFT\_BUS}$（74LS194 的三态门控制），负责各个器件输出所保存的数据到数据总线。

低电平有效的开关 $\overline{SET}$ 是 74LS74 的置 1 开关，低电平有效的开关 $\overline{CLR}$（74LS74 和 74LS175 共用）、$\overline{MR}$（74LS194 使用）是各个器件相应的清零开关。注意的是：寄存器 74LS374 是没有清零功能的，74LS273 有置 0 端 $\overline{MR}$，可以设置清零功能。但是，在本实验中 $\overline{MR}$ 接高电平，实际上 74LS273 取消了清零功能。

高电平有效的开关 SL、SR、S0、S1 是移位寄存器 74LS194 的专属开关，负责其置数、移位等功能的设置，具体详细见下文移位寄存器 74LS194 的功能列表。

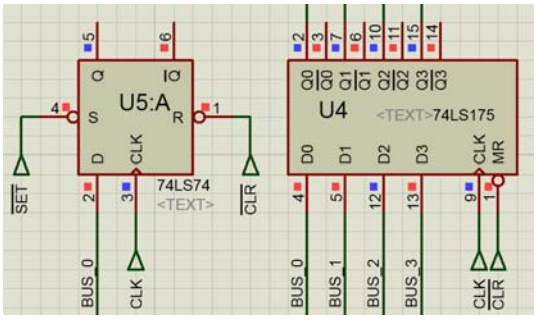
维持-阻塞型 D 触发器：

D 触发器逻辑功能如表 2-3：$\overline{S}_d$ 和 $\overline{R}_d$ 端分别为异步置 1 端和置 0 端，CLK 为时钟脉冲端。CLK 脉冲上升沿触发。触发器的状态只取决于时钟到来前 D 端的状态，其输出状态的更新发生在 CLK 脉冲的上升沿。D 触发器可用作数字信号的寄存，移位寄存，分频和波形发生等。本实验使用的是单位 D 触发器 74LS74 和四位 D 触发器 74LS175，如图 2-3 所示。**74LS175 相当于四个并行的 74LS74 组合，但是省略了置 1 端，保留了置 0 端用以清零。**

表 2-3　D 触发器逻辑功能表

输　入				输　出	
$\overline{S}_d$	$\overline{R}_d$	CLK	D	Q^{n+1}	$\overline{Q}^{n+1}$
0	1	×	×	1	0
1	0	×	×	0	1
0	0	×	×	φ*	φ*
1	1	↑	1	1	0
1	1	↑	0	0	1
1	1	↓	×	Q^n	$\overline{Q}^n$

图 2-4 D 触发器



注：* 这种情况下是不稳定的，即当 $\overline{S}_d$ 和 $\overline{R}_d$ 端回到高电平时，输出状态将不能保持。

通用寄存器：

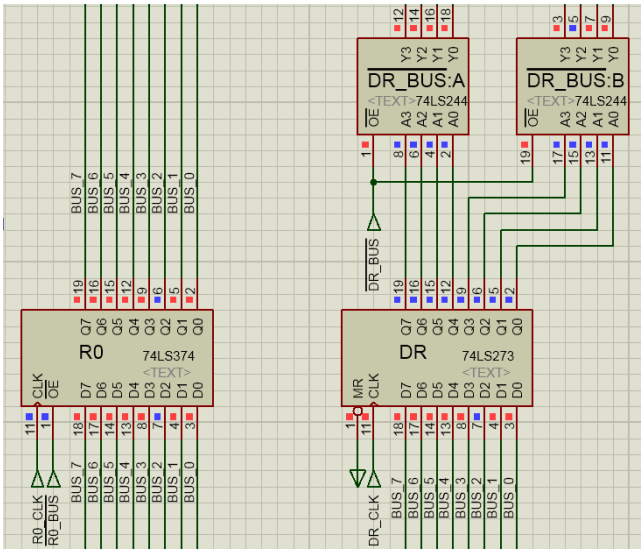
本实验中使用了 74LS273 和 74LS374 两款芯片作为通用寄存器，如图 2-5 所示。两者实现的功能基本相同：D_0-D_7 为并行输入端，Q_0-Q_7 为并行输出端，CLK 端为时钟脉冲（上升沿触发）；输出端 Q_X 的状态只取决于 CLK 端时钟脉冲到来时刻输入端 D_X 的状态，输出状态的更新

发生在 CLK 端脉冲的上升沿。**74LS374 自带三态门输出控制结构，比 74LS273 多了一个 OE 端输出使能，相当于“74LS273 + 74LS244”组合。74LS273 本身没有三态门输出控制，输出端必须经过 74LS244 接到数据总线 BUS。但是，74LS273 比 74LS374 多了一个 MR 端输出清零。**两者的逻辑功能如表 2-4 所示：

表 2-4 74LS273 & 374 逻辑 功能表

$\overline{MR}$	$\overline{OE}$	CLK	D	Q^{n+1}
0	0	×	×	0
1	0	↑	0	0
1	0	↑	1	1
1	0	0	×	Q^n
1	0	1	×	Q^n
×	1	×	×	Z

图 2-5 通用寄存器



移位寄存器：

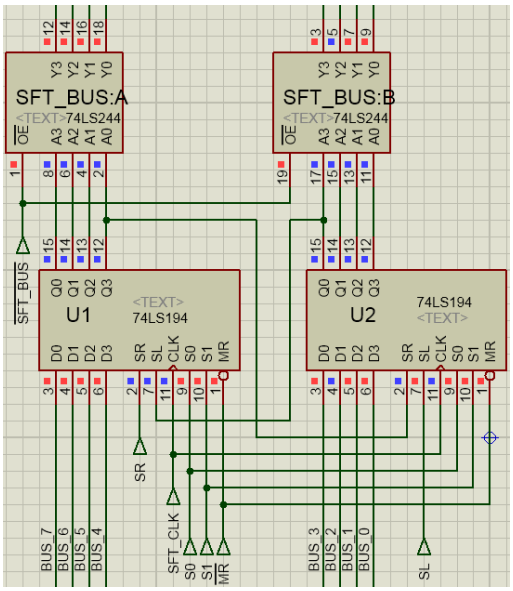
在数字系统中能寄存二进制信息，并进行移位的逻辑部件称为移位寄存器。

本实验采用四位双向通用移位寄存器 74LS194，如图 2-6 所示：D_0、D_1、D_2、D_3为并行输入端；Q_0、Q_1、Q_2、Q_3为并行输出端；S_R为右移串行输入端；S_L为左移串行输入端；S_1、S_0为操作模式控制端；MR 为直接无条件清零端（注：74LS194 寄存或移位数据时，必须 $\overline{MR}$=1）；CLK 为时钟输入端。寄存器有四种不同操作模式：①并行寄存；②右移(方向由 Q_0->Q_3)；③左移(方向由 Q_3->Q_0)；④保持。74LS194 的逻辑功能表如表 3_5 所示。

表 2-5 74LS194 逻辑功能表

CLK	MR\	S_1	S_0	功能	$Q_0\ Q_1\ Q_2\ Q_3$
×	0	×	×	清除	$Q_0\ Q_1\ Q_2\ Q_3$ =0
↑	1	1	1	送数	$Q_0\ Q_1\ Q_2\ Q_3$ = $D_0\ D_1\ D_2\ D_3$
↑	1	0	1	右移	$Q_0\ Q_1\ Q_2\ Q_3$ = $S_R\ Q_0\ Q_1\ Q_2$
↑	1	1	0	左移	$Q_0\ Q_1\ Q_2\ Q_3$ = $Q_1\ Q_2\ Q_3\ S_L$
↑	1	0	0	保持	$Q_0^D Q_1^D Q_2^D Q_3^D$ = $Q_0\ Q_1\ Q_2\ Q_3$

图 2-6 移位寄存器



五、实验步骤

实验 1：拨码开关输入数据至总线

● $\overline{SW\_BUS}$ = $\overline{R0\_BUS}$ = $\overline{DR\_BUS}$ = $\overline{SFT\_BUS}$ = 1；手动操作总线 DIN 上的拨码开关，在总线 DIN 上置位数据 0x55，缓冲器 244 阻断。比较总线 DIN 与 BUS 状态的异同。

● $\overline{SW\_BUS}$=0,，比较总线 DIN 与 BUS 状态的异同，记录 BUS 总线的数据：

BUS_7	BUS_6	BUS_5	BUS_4	BUS_3	BUS_2	BUS_1	BUS_0	BUS 总线

实验 2：D 触发器数据锁存实验

● $\overline{SW\_BUS}$ = 0，$\overline{R0\_BUS}$ = $\overline{DR\_BUS}$ = $\overline{SFT\_BUS}$ =1；通过拨码开关改变 74LS74 的 D 端（即 BUS 总线的 BUS_0）的状态，按照下表置位 74LS74 的$\overline{S}_d$ 端、$\overline{R}_d$ 端，观察并记录 CLK 端上升沿 、下降沿跳变时刻 Q 端、$\overline{Q}$端的状态，填观测结果于表中。

$\overline{S}_d$	$\overline{R}_d$	CLK	D	Q^n	Q^{n+1}	$\overline{Q}^{n+1}$
0	1	X	X	0		
				1		
1	0	X	X	0		
				1		
1	1	↑	0	0		
				1		
1	1	↑	1	0		
				1		
1	1	0（1）	X	0		
				1		

● 74LS175 的三态门 244 阻断（$\overline{OE}$=1），拨码开关置位 BUS 总线数据，使 74LS175 的 D 端分别接高，低电平，观察并记录当 CLK 上升沿 、下降沿跳变时 Q 端、$\overline{Q}$端的状态。

● 观察 74LS175 的 Q 端、$\overline{Q}$端和 74LS74 的 Q 端、$\overline{Q}$端的异同，观察当 74LS175 的 MR 端置 0 后（$\overline{CLR}$ = 0），输出 Q 端、$\overline{Q}$端的变化。

实验 3：通用寄存器实验

● $\overline{SW\_BUS}$ = 0，$\overline{R0\_BUS}$ = $\overline{DR\_BUS}$ = $\overline{SFT\_BUS}$ = 1；操作拨码开关输入数据 0xAA 到总线，观测此时 74LS374 和 74LS273 输出端 Q_X 的各自状态。

● 74LS374 的 CLK 端 R0_CLK 上升沿跳变把总线上的 0xAA 数据存入 R0 寄存器（74LS374）。

● 拨码开关的三态门 244 阻断（$\overline{SW\_BUS}$ = 1），观察此时总线 BUS 上的状态。

● 74LS374 的输出选通（$\overline{R0\_BUS}$ = 0），观测总线 BUS 的状态。

● 74LS273 的 CLK 端 DR_CLK 上升沿跳变把总线上的 0xAA 数据存入 DR 寄存器（74LS273）。观察 74LS374 和 74LS273 输出端 Q_X 的各自状态。

实验 4：移位寄存器实验

● $\overline{SW\_BUS}$ = 0，$\overline{R0\_BUS}$ = $\overline{DR\_BUS}$ = $\overline{SFT\_BUS}$ = 1；通过拨码开关送入总线 BUS 任意八位二进制数，赋值 74LS194 的输入端 $D_0D_1D_2D_3$。按照下表置位 74LS194 的 MR 端、S_1 端、S_0 端、S_L 端、S_R 端，观察并记录 CLK 端上升沿 、下降沿跳变时刻输出端 $Q_0Q_1Q_2Q_3$ 的状态，填观

测结果于表中，并填写功能总结。

清除	模式		时钟	串行		输入	输出	功能总结
MR	S_1	S_0	CLK	S_L	S_R	$D_0D_1D_2D_3$	$Q_0Q_1Q_2Q_3$	
0	×	×	×	×	×	××××		
1	1	1	↑	×	×	a b c d		
1	0	1	↑	×	0	××××		
1	0	1	↑	×	1	××××		
1	0	1	↑	×	0	××××		
1	0	1	↑	×	0	××××		
1	1	0	↑	1	×	××××		
1	1	0	↑	1	×	××××		
1	1	0	↑	1	×	××××		
1	1	0	↑	1	×	××××		
1	0	0	↑	×	×	××××		

六、思考题

1、把 74LS175 的$\overline{Q}$端接三态门 244，拨码开关把数据 0xAA 打入 74LS175。假设 $\overline{OE}$=0，$\overline{SW\_BUS}$=0，会出现什么情况？为什么？

2、拨码开关先打入数据 0xAA 到 R0 寄存器（74LS374），R0_BUS\=1；拨码开关再输入新的数据 0x55 到总线 BUS，会冲掉 R0 寄存器里保存的 0xAA 么？假设此时令 R0_BUS=0，会出现什么情况？为什么？

3、拨码开关分别打入数据 0xAA 和 0x55 到 R0 寄存器（74LS374）和 DR 寄存器（74LS273），假设此时令 $\overline{R0\_BUS}$ = $\overline{DR\_BUS}$ = 0，会出现什么情况？可以同时打开多个寄存器输出到总线的三态门控制么？

4、74LS194 的 S_L 端、S_R 端数据究竟是提供 $D_0D_1D_2D_3$ 端移入数据还是保存 $D_0D_1D_2D_3$ 端移出数据？假设要保存 74LS194 移位时 $D_0D_1D_2D_3$ 端移出的数据，该怎么修改寄存器电路？

2.2 进位加法器实验

一、实验目的

1、了解半加和全加运算器的电路结构。

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、实验内容

设计拥有共同输入端的 4 位带符号位的串行加法器和并行加法器，并比较两者的运算输出结果。

三、实验器件

1、2/3/4 与门(74LS08/11/21)、非门(74LS04)、或门(74LS32)、异或门(XOR)等逻辑门。

2、三态门（74LS244）、LED 指示灯及数码显示管。

四、实验原理

实验中的加法运算器电路如图 2-7 所示。本电路包含了输入单元（拨码开关）、输出单元（数码显示管）和两个四位全加器。输入单元如图 2-3 所示，跟上述实验相同：拨码开关用来设置输入数据，低电平有效的开关 $\overline{\text{SW\_BUS}}$ 控制输入总线 DIN_[0..7]与数据总线 BUS_[0..7]是否连通。数据总线 BUS 同时为两个运算器提供相同的输入端：BUS_[0..3]和 BUS_[4..7]。输出单元是一对数码显示管，绿色显示管是并行进位加法器的输出端，红色显示管是串行进位加法器的输出端

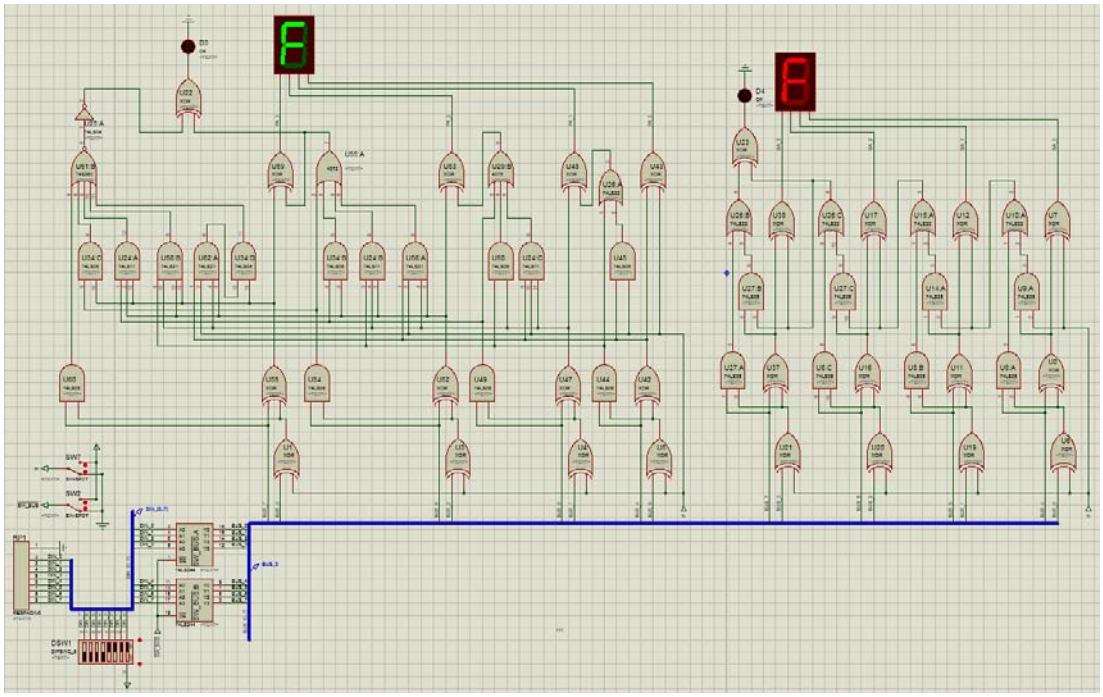
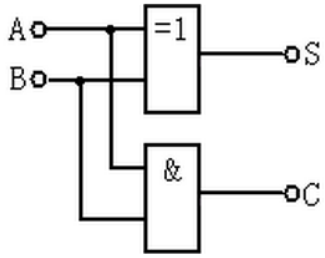


图 2-7 进位加法器电路图

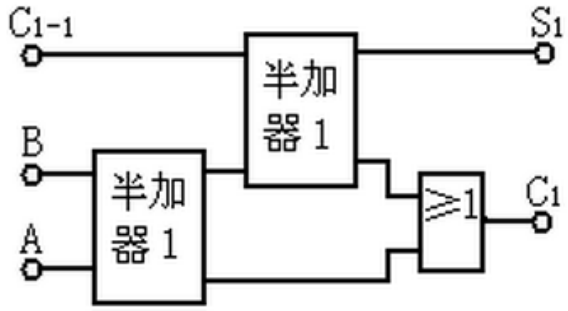
加法器是执行二进制加减法运算的逻辑部件（减法可以通过补码相加实现），分为半加器和全加器（FA），不考虑低位的进位，只考虑两个二进制数相加，得到和以及向高位进位的加法器为半加器，如图 2-8 所示；而全加器(FA)是在半加器的基础上又考虑了低位过来的进位信号，如图 2-9 所示。



输入		和	进位
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

图 2-8 一位半加器逻辑电路图

半加器逻辑公式： $S_i = A_i \oplus B_i$ 且 $C_{i+1} = A_i \cdot B_i$



输入			输出	
C_{1-1}	B	A	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

图 2-9 一位全加器（FA）逻辑电路图

全加器逻辑公式： $S_i = A_i \oplus B_i \oplus C_i$ 且 $C_{i+1}=A_i \cdot B_i + (A_i \oplus B_i) \cdot C_i$

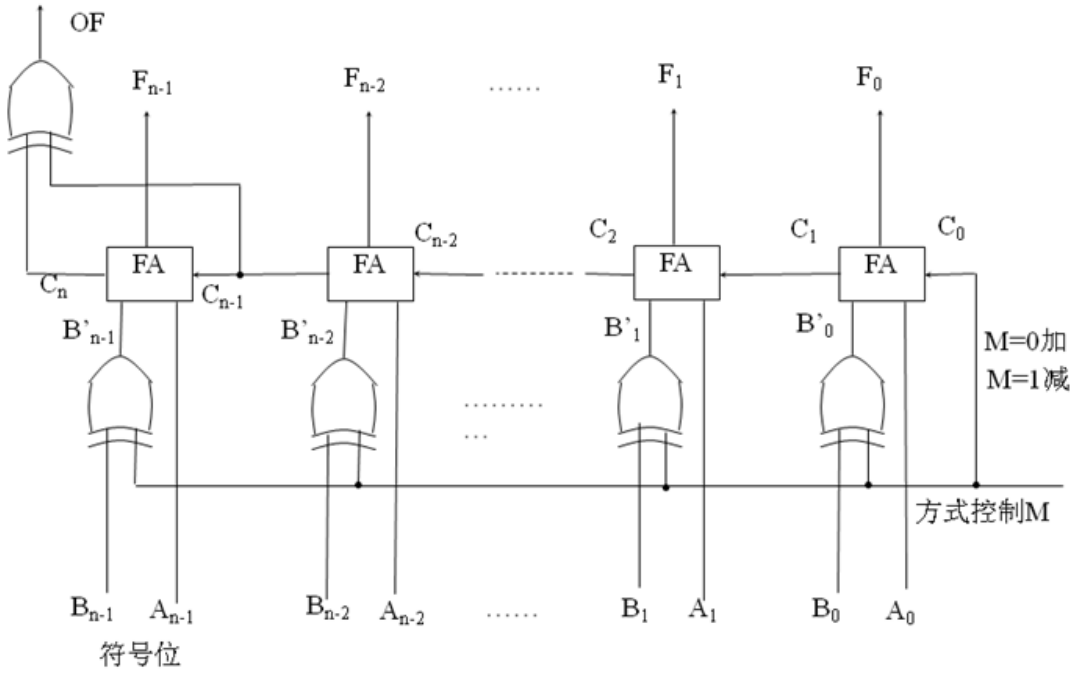


图 2-10 串行加法器（行波进位加法器）逻辑电路图

本实验的串行加法器又称为行波进位加法器，如图 2-10 所示：由若干位全加器 FA 串行相连得到，其中低位 FA 的进位输出直接与相邻的高位 FA 的进位输入相连。因为高位进位产生依赖于相邻低位 FA 的进位，所以各位运算依次完成，延迟较长，效率不高。

图 2-11 所示是四位串行加法器仿真图，由四个图 2-12 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 端工作原理基于“$[Y]_{补}\rightarrow[-Y]_{补}$”求补运算原则：取反再加 1。图 2-11 中执行的运算是：“101+001=110”。

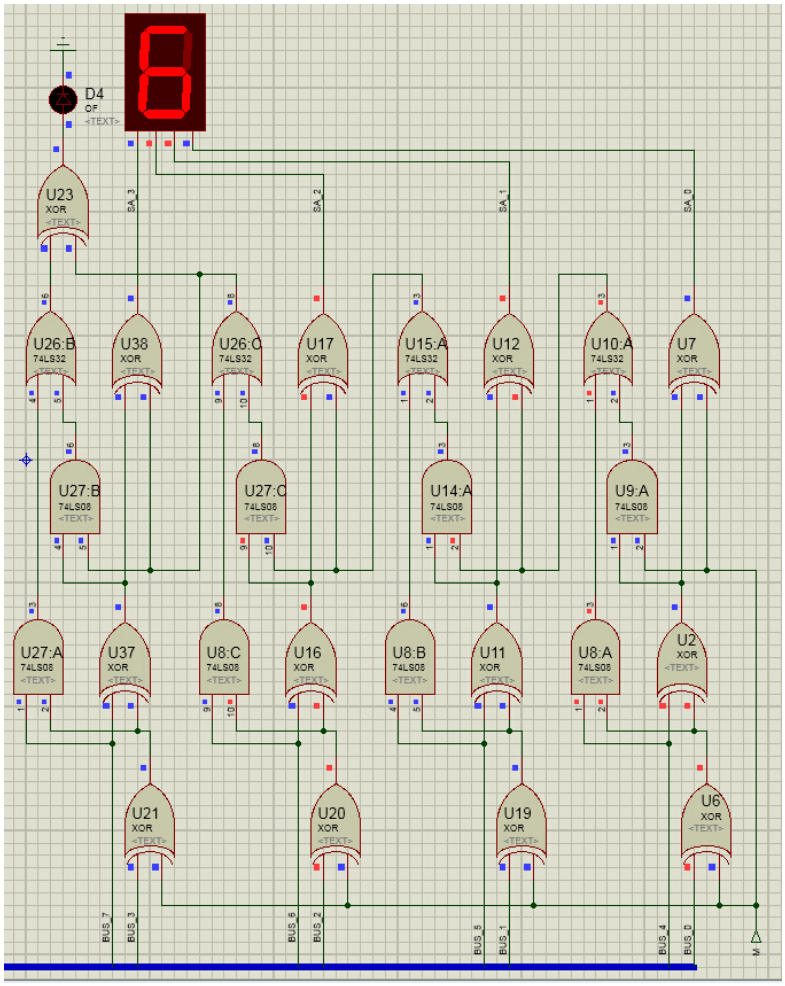


图 2-11 串行加法器

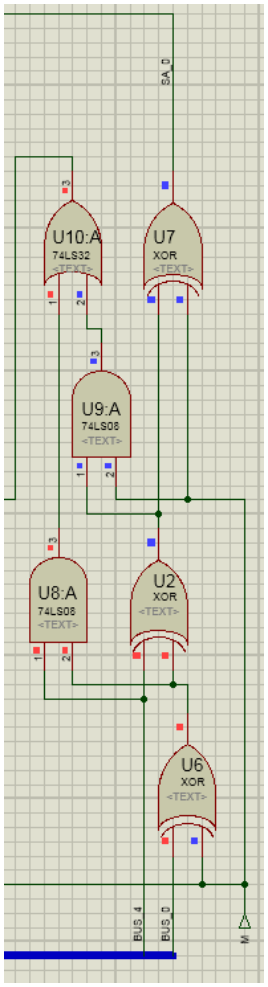


图 2-12 一位全加器

本实验的并行加法器又称为超前进位加法器，如图 2-13 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 C_0 及各个位的加数、被加数即可同时计算所有进位 C_n。因此称之为“并行”加法器，其推导公式如下：

$C_{i+1}=A_i\cdot B_i+(A_i\oplus B_i)\cdot C_i$　令 $Yn=An\cdot Bn$　$Xn=(An\oplus Bn)$，则有：

$C_{n+1}=Y_n+X_n\cdot C_n$　故：

$C_1=Y_0+X_0\cdot C_0$

$C_2=Y_1+X_1\cdot C_1=Y_1+X_1\cdot Y_0+X_1\cdot X_0\cdot C_0$

$C_3=Y_2+X_2\cdot C_2=Y_2+X_2\cdot Y_1+X_2\cdot X_1\cdot Y_0+X_2\cdot X_1\cdot X_0\cdot C_0$

$C_4=Y_3+X_3\cdot C_3=Y_3+X_3\cdot Y_2+X_3\cdot X_2\cdot Y_1+X_3\cdot X_2\cdot X_1\cdot Y_0+X_3\cdot X_2\cdot X_1\cdot X_0\cdot C_0$

由图 2-13 中可以看出，每一个 FA 所需的低位进位都只跟 An、Bn 和 C_0 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 Cn 生成电路越复杂。图 2-13 中执行的运算是：“101+011=1000”；因为进位影响到符号位改变，所以导致溢出，数码管旁边的 OF 指示灯点亮。

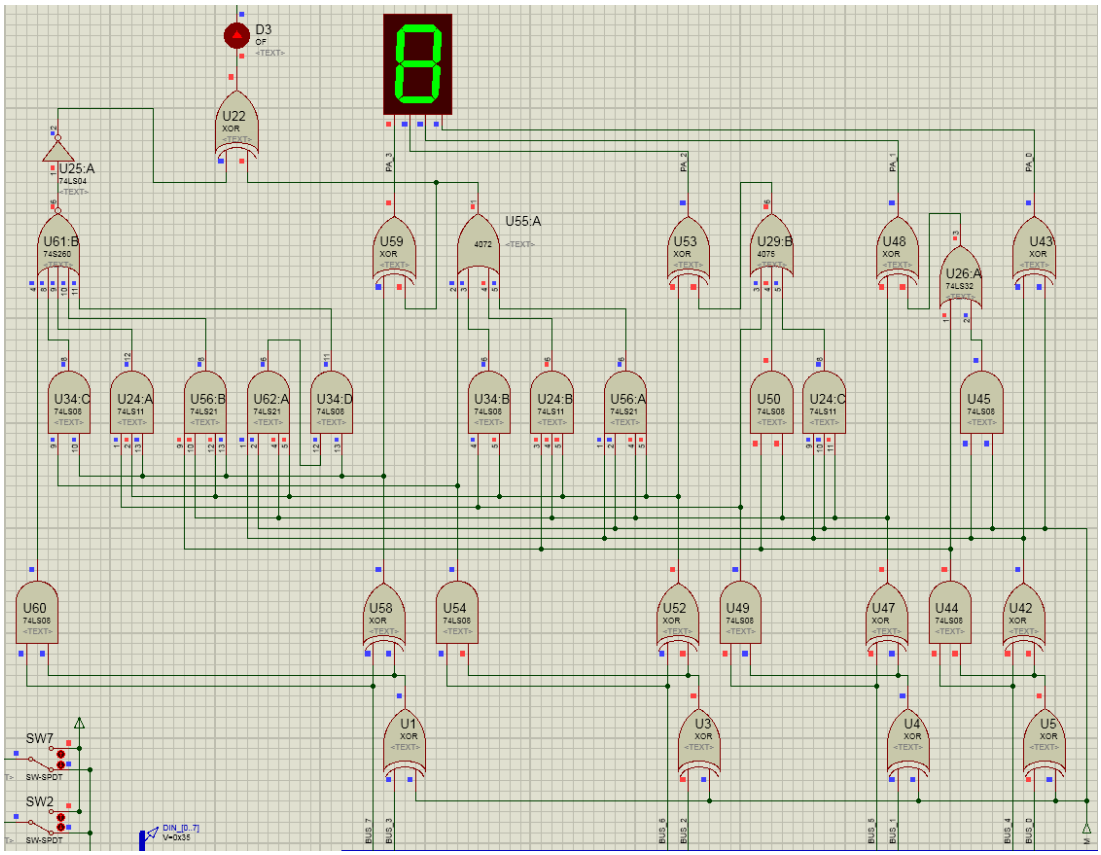


图 2-13 并行加法器

五、实验步骤

● 请制作半加器和全加器的电路，并验证其输入输出是否与图 2-8 和图 2-9 一致。

● 令 BUS_[7..4]=0101，BUS_[3..0]=0010，M=0，写出该运算的真值表达式，记录并比较串行与并行加法器的运算结果，是否有溢出？如果改为 BUS_[3..0]=0011，结果如何呢？

● 令 BUS_[7..4]=0101，BUS_[3..0]=0011，M=1，写出该运算的真值表达式，记录并比较串行与并行加法器的运算结果，是否有溢出？如果运算器的输入改为 BUS_[7..4] =0011，BUS_[3..0]=0101，M=1 不变，结果如何呢？

● 令 BUS_[7..4]=1101，BUS_[3..0]=0011，M=0，写出该运算的真值表达式，记录并比较串行与并行加法器的运算结果。是否有溢出？如果改为 M=1，输入不变，结果如何呢？

六、思考题

1、请问本实验运算器电路是补码运算器还是原码运算器？假设需要把运算器电路改为四位无符号数的原码加法器，请问电路怎么修改？

2、本实验中，四位补码的表数范围是多少？若要修改为五位补码运算器（符号位一位，数值位四位），其表数范围又是多少？请问电路怎么修改？

2.3 运算器实验

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。

二、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、实验器件

1、算术逻辑运算器（74LS181）。

2、三态门（74LS244）及寄存器（74LS273、74LS373）。

四、实验原理

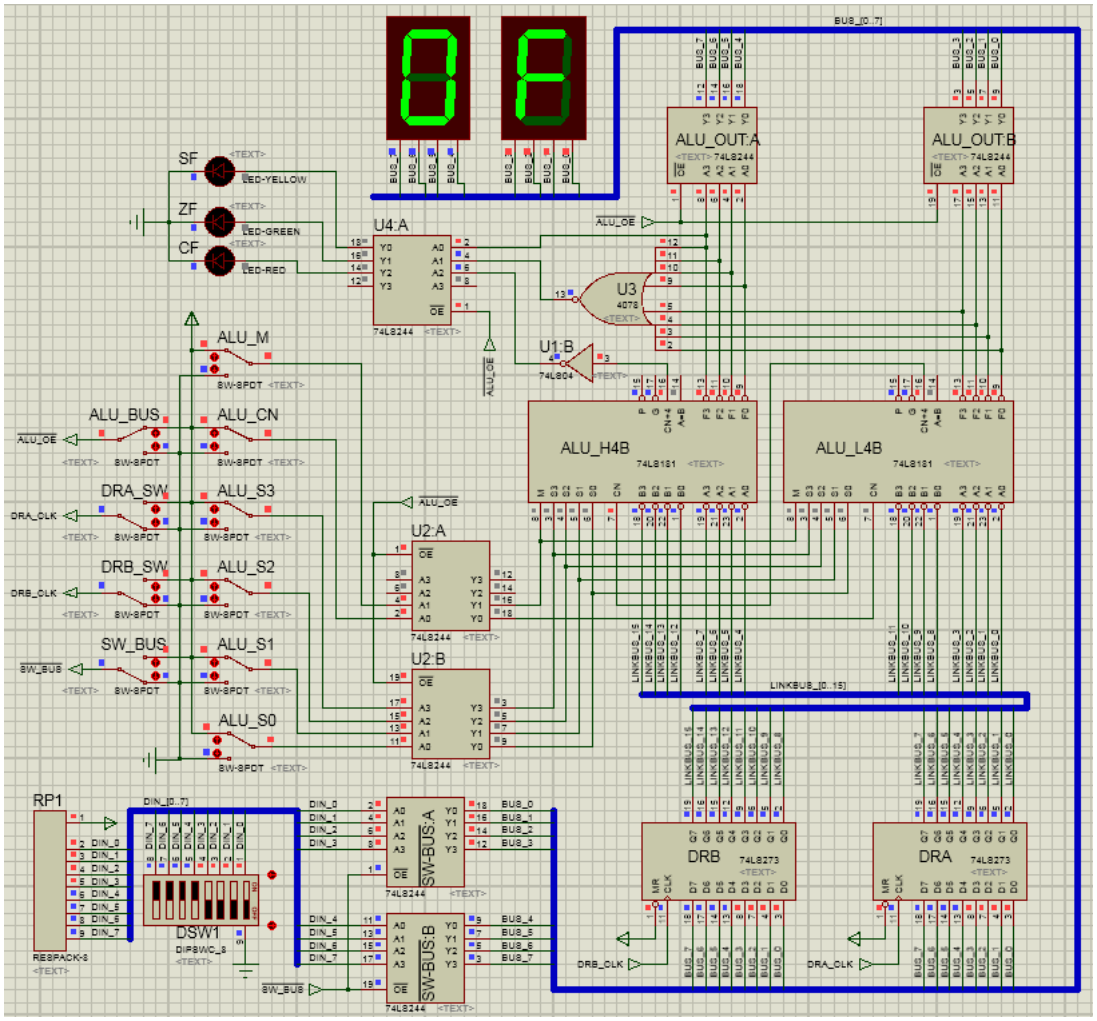


图 2-14 运算器通路

本实验的算术逻辑运算器电路如图 2-14 所示：输入和输出单元跟上述实验相同：拨码

开关用来给出参与运算的数据，并经过三态门 74LS244 和数据总线 BUS 相连；一对数码显示管用来显示数据总线 BUS 的内容。

运算器则由两个 74LS181 以串行进位形式构成 8 位字长的 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 除了溢出标志位 CF 外，还有两个标志位：零标志位 ZF（ZF=1，即 ALU 运算结果为 0）和符号标志位 SF（SF=1，即运算结果为负数；SF=0 即运算结果为正数或 0）。

ALU 的工作方式可通过设置两个 74181 芯片的控制信号（ALU_S0、S1、S2、S3、M 及 CN）来实现, 其逻辑功能表由表 2-6 给出，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。

表 2-6　74LS181 逻辑功能表

S3	S2	S1	S0	M=0（算术运算）		M=1（逻辑运算）
				CN=1 无进位	CN=0 有进位	
0	0	0	0	F=A	F=A 加 1	F=$\overline{A}$
0	0	0	1	F=$A+B$	F=（$A+B$）加 1	F=$\overline{A+B}$
0	0	1	0	F=$A+\overline{B}$	F=（$A+\overline{B}$）加 1	F=$\overline{A}B$
0	0	1	1	F=0 减 1	F=0	F=0
0	1	0	0	F=A 加 $A\overline{B}$	F=A 加 $A\overline{B}$ 加 1	F=$\overline{AB}$
0	1	0	1	F=（$A+B$）加 $A\overline{B}$	F=（$A+B$）加 $A\overline{B}$ 加 1	F=$\overline{B}$
0	1	1	0	F=A 减 B 减 1	F=A 减 B	F=$A\oplus B$
0	1	1	1	F=$A\overline{B}$ 减 1	F=$A\overline{B}$	F=$A\overline{B}$
1	0	0	0	F=A 加 AB	F=A 加 AB 加 1	F=$\overline{A}+B$
1	0	0	1	F=A 加 B	F=A 加 B 加 1	F=$\overline{A\oplus B}$
1	0	1	0	F=（$A+\overline{B}$）加 AB	F=（$A+\overline{B}$）加 AB 加 1	F=B
1	0	1	1	F=AB 减 1	F=AB	F=AB
1	1	0	0	F=A 加 A	F=A 加 A 加 1	F=1
1	1	0	1	F=（$A+B$）加 A	F=（$A+B$）加 A 加 1	F=$A+\overline{B}$
1	1	1	0	F=（$A+\overline{B}$）加 A	F=（$A+\overline{B}$）加 A 加 1	F=$A+B$
1	1	1	1	F=A 减 1	F=A	F=A

运算器 ALU 的输出经过三态门（74LS244）和数据总线 BUS 相连。当运算器使能开关低电平有效（$\overline{\text{ALU\_OE}}$=0）的时候，运算器三个部位的三态门 244 状态为直通：74181 的控制信号（S0~S3、M、CN）全部连通；74181 的运算标志位（CF、ZF 和 SF）的指示灯全部连通；以及 74181 的运算结果输出到数据总线 BUS。当 $\overline{\text{ALU\_OE}}$=1 的时候，74181 停止工作，此时 74181 的输出端数据为无效数据，与数据总线 BUS 隔断。

运算器 ALU 的两个数据输入端分别由两个数据暂存器（74LS273）DRA、DRB 锁存，74LS181 将 DRA、DRB 内的数据作为上述表 2-6 中参与运算的数 A 和 B。由于 DRA、DRB 已经把数据锁存，只要 74LS181 的控制信号不变，那么 74LS181 的输出数据也不会发生改变。数据暂存器 DRA、DRB 的输入连至数据总线 BUS，在 DRA_CLK、DRB_CLK 端开关出现上

升沿跳变的时候，总线 BUS 的数据分别打入 DRA、DRB 锁存。

五、实验步骤

● 令图 2-14 各个开关的初始状态为：DRA_CLK=DRB_CLK=0，$\overline{SW\_BUS}$ = $\overline{ALU\_OE}$ = 1，(S3,S2,S1,S0,M,CN)=(1,1,1,1,1,1)。操作拨码开关，向数据暂存器 DRA 写入 AAH，DRB 写入 55H（即 A=0xAAH，B=0x55H）。改变运算器的控制信号(S3,S2,S1,S0,M,CN)的组合，运算器使能（$\overline{ALU\_OE}$=0），观察运算器的输出和标志位，并填入下表中，与理论值比较，验证 74LS181 的功能。

DRA	DRB	S3	S2	S1	S0	M=0（算术运算）		标志位	M=1
						CN=1 无进位	CN=0 有进位	CF/ZF/SF	逻辑运算
		0	0	0	0	F=	F=		F=
		0	0	0	1	F=	F=		F=
		0	0	1	0	F=	F=		F=
		0	0	1	1	F=	F=		F=
		0	1	0	0	F=	F=		F=
		0	1	0	1	F=	F=		F=
		0	**1**	**1**	**0**	**F=**	**F=**		F=
		0	1	1	1	F=	F=		F=
		1	0	0	0	F=	F=		F=
		1	**0**	**0**	**1**	**F=**	**F=**		F=
		1	0	1	0	F=	F=		F=
		1	0	1	1	F=	F=		F=
		1	1	0	0	F=	F=		F=
		1	1	0	1	F=	F=		F=
		1	1	1	0	F=	F=		F=
		1	1	1	1	F=	F=		F=

● 操作拨码开关，向数据暂存器 DRA、DRB 分别打入有符号数 +7AH，-75H（即 A= + 0x7A H，B= - 0x75 H）。改变运算器的控制信号(S3,S2,S1,S0,M,CN)的组合，运算器使能（$\overline{ALU\_OE}$=0），观察运算器的输出和标志位，并填入上表中，与理论值比较，验证 74LS181 的功能。

六、思考题

1、74181 组成的运算器通路，可以区分有符号数运算和无符号数运算么？两者的运算过程有不同么？两者的 数值表示范围有不同么？

2、当 74181 进行无符号数运算的时候，运算结果的标志位 SF 有无意义？在有符号数运算和无符号数运算过程中，标志位 CF、ZF 和 SF 的含义都是一样的么？

2.4 存储器实验

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 位字长的存储器系统，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

本实验的存储器通路如图 2-14 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 ABUS_[0..11]和 8 位数据总线 DBUS_[0..7]。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 位地址信息。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。

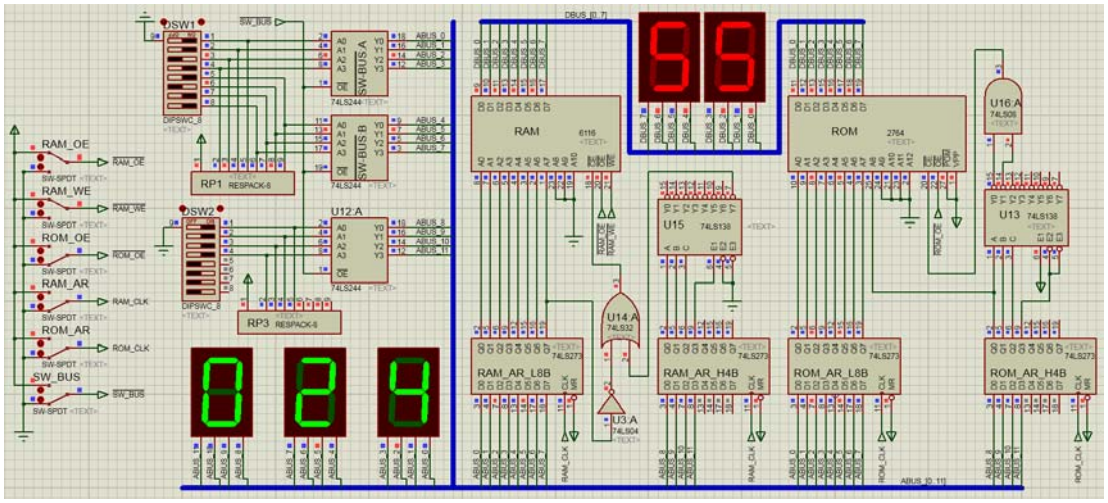


图 2-15 存储器通路

存储器是用来存储信息的部件，是计算机的重要组成部分，常见的半导体存储器类型主要有 ROM 和 RAM：ROM 是 Read Only Memory（只读存储器）的缩写，RAM 是 Random Access Memory（随机存取存储器）的缩写。ROM 价格较低，一般容量较大，在系统停止供电的时候仍然可以保持数据；而 RAM 价格较高，一般容量较小，在掉电之后就丢失数据，典型的 RAM 就是计算机的内存。本实验中使用的 ROM 是 2764（8K×8 位），RAM 是 6116（2K×8 位），如图 2-16 所示。

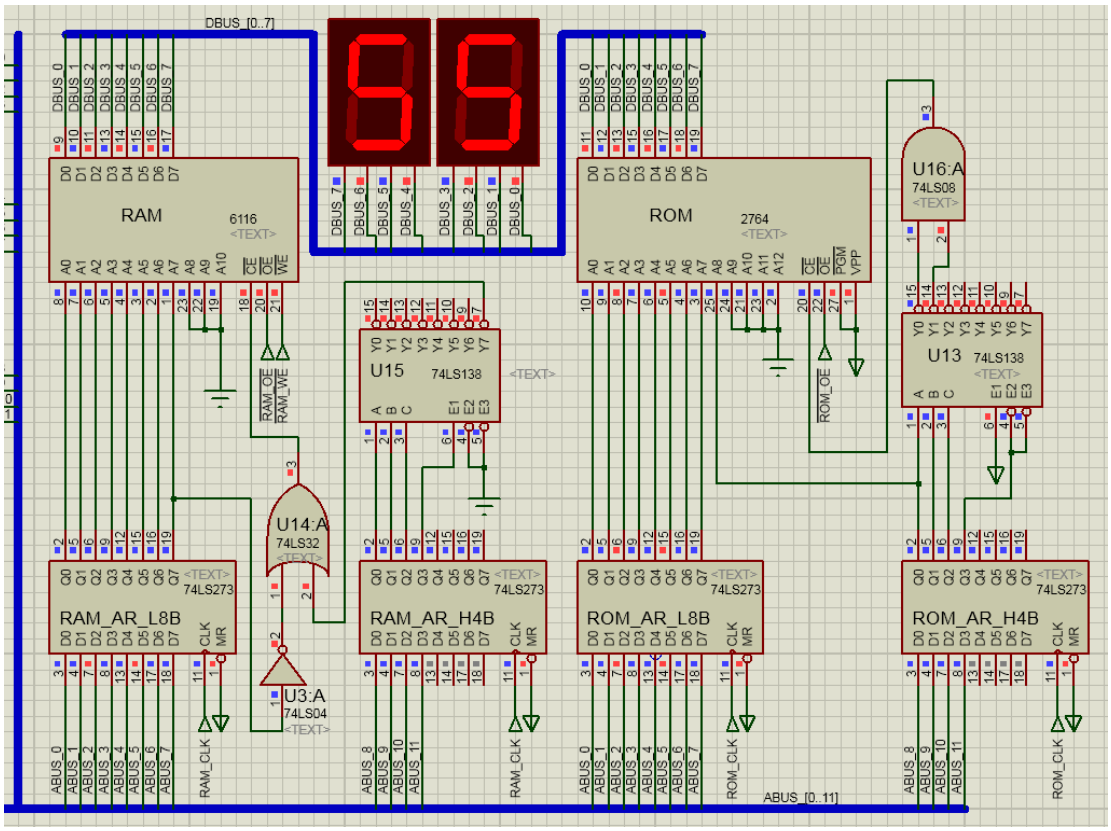


图 2-16 存储器及地址选择电路

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，所以其理论容量为 8K 字节。2764 有两个控制端：$\overline{CE}$（片选）、$\overline{OE}$（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，所以其理论容量为 2K 字节。6116 有三个控制端：$\overline{CE}$（片选）、$\overline{OE}$（读）、$\overline{WE}$（写）。

存储器通路控制信号的逻辑功能见表 2-7。值得注意的是：在对 ROM 或 RAM 读写的时候，首先必须在存储器的片选有效（$\overline{CE}$=0）前提下，才能对相应的存储器读（$\overline{OE}$=0）或写（$\overline{WE}$=0）。例如：对 ROM 芯片 2764 进行读操作，必须使能 $\overline{ROM\_CE}$=0 且 $\overline{ROM\_OE}$=0。存储器片选信号 $\overline{ROM\_CE}$ 和 $\overline{RAM\_CE}$ 是由地址信号的高 4 位 ABUS_8~11 经过片选逻辑电路自动形成，不需要拨码开关控制。

表 2-7 存储器通路控制信号说明

信号名称	作 用	有效电平
ROM_CLK	2764 地址的锁存脉冲信号	上升沿跳变有效（开关手动）
RAM_CLK	6116 地址的锁存脉冲信号	上升沿跳变有效（开关手动）
ROM_CE\	2764 的片选有效信号	低电平有效 （地址驱动）
RAM_CE\	6116 的片选有效信号	低电平有效 （地址驱动）
ROM_OE\	2764 的读允许信号	低电平有效（开关手动）
RAM_OE\	6116 的读允许信号	低电平有效（开关手动）
RAM_WE\	6116 的写允许信号	低电平有效（开关手动）
SW_BUS\	地址总线输入允许信号	低电平有效（开关手动）

其次，必须在地址锁存器（74LS273）ROM_AR、RAM_AR 锁存地址信号，才能选中存储器片内相应的单元。地址锁存器 ROM_AR 和 RAM_AR 的输入都连接至地址总线 ABUS_0~7，在其 CLK 端开关出现上升沿跳变的时候，地址总线 ABUS_0~7 的数据打入 ROM_AR 或 RAM_AR 锁存。锁存后无论地址总线 ABUS 如何变化，选中的存储单元也不会发生改变，可以进行稳定的读写操作（存储器数据端输入或输出）。

存储器通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 128 地址为 RAM 区（F80H~FFFH），其余留空，如表 2-8 所示。

表 2-8 存储器与地址总线连接范围及选片

A11 A10 A9 A8	A7 A6 A5 A4 A3 A2 A1 A0	地址范围	选片
0 0 0 0 …… 0 0 0 1	0 0 0 0 0 0 0 0 …… 1 1 1 1 1 1 1 1	000H~1FFH	ROM 芯片 2764 （实际容量 512 字节）
……	……	200H~F7FH	（空）
1 1 1 1 …… 1 1 1 1	1 0 0 0 0 0 0 0 …… 1 1 1 1 1 1 1 1	F80H~FFFH	RAM 芯片 6116 （实际容量 128 字节）

存储器通路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。

在本实验中，12 位理论地址空间分为低 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 译码器把地址空间分为 16 个部分，如图 2-16 所示。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 芯片的片选由高位 3-8 译码器的最高 1 个部分片选信号与 RAM 的 A7 地址线的片选电路形成。值得注意的是：片选电路的逻辑组合不是唯一的，可以有多种实现形式。

五、实验步骤

实验 1：二进制数据批量导入 ROM

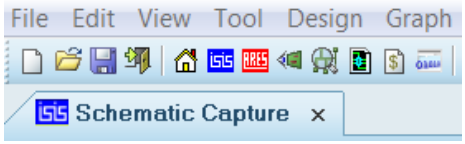
● 新建文本文件（.txt），按照下列格式输入数据。然后，把.txt 后缀改名为.asm 后缀：

```
ORG    0024H
       DB      55H
       DB      55H
       DB      55H
       DB      55H

       DB      01H
       DB      01H
       DB      01H
       DB      01H
```

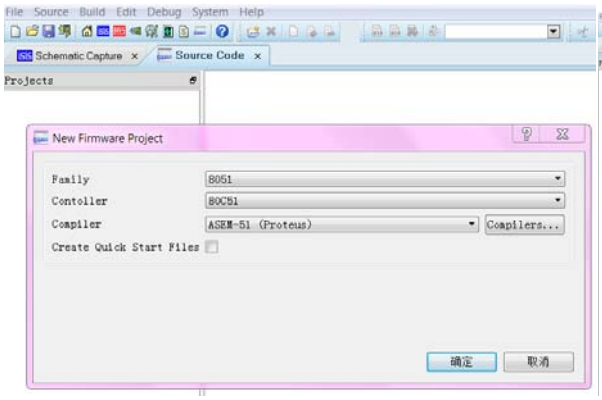
```
        DB      0FFH
        DB      0FFH
        DB      0FFH
        DB      0FFH

        DB      01H
        DB      01H
        DB      01H
        DB      01H
END
```

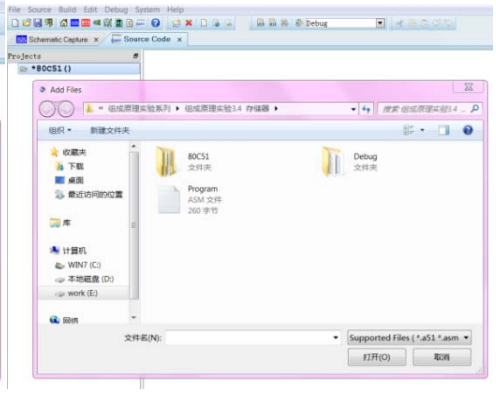
注：上述代码中 “ORG xxxxH”规定存储数据的首地址，可以从 0000H 开始，也可以从任意地址开始。每一行“DB xxH”是一个存储单元存放的数据（注：16 进制 A~F 前面要加 0），从首地址开始，按顺序存放。中间的空行 仅仅为阅读方便，没有实际意义。文件必须以“END”作为结束

● 在 proteus 开发环境中，菜单栏里点击 Source Code 面板（左下图的最右边，Graph 下面那个图标），然后会弹出 Source Code 标签页，如右下图所示：



● 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->Create Project，在弹出的对话框里如左下图所示勾选（注：Creat Quick Start Files 选项不勾），再按“确定”按钮；在标签页上会出现一个“*80C51()”的新 project，右键单击“*80C51()”，在弹出的菜单列表中选择 Add files（或 Import Existing File）选项，在弹出的视窗内选中所需的 asm 文件，按确定就添加进 project。如右下图所示。



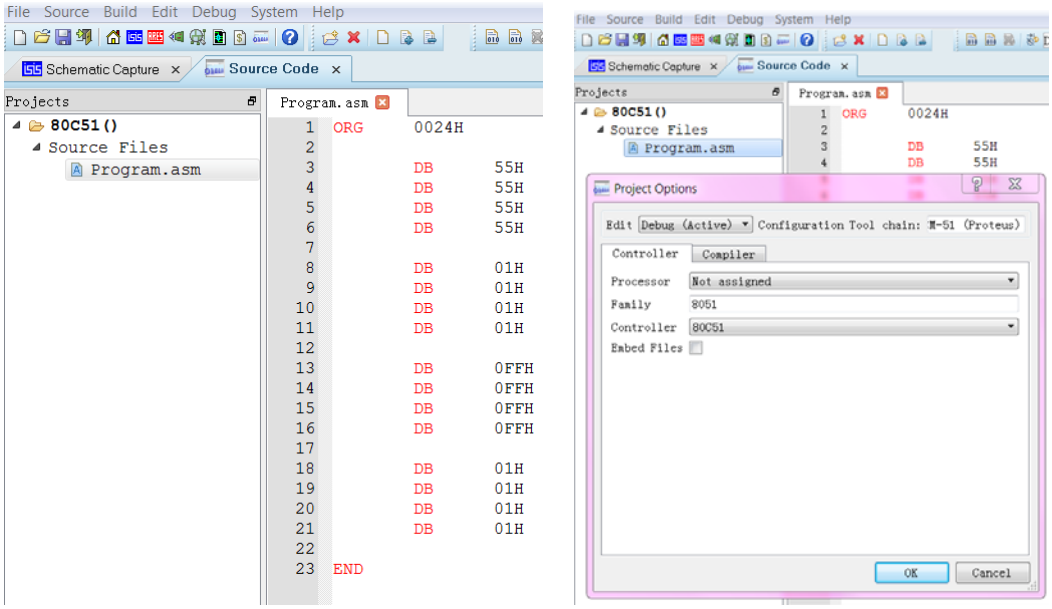


● 在已经添加的 asm 文件上双击，在右侧会打开 asm 文件的文本内容，如左下图所示。如果一个 proteus 的项目中已经建立了 project 且打过 asm 文件，则弹出的 Source Code 标签页会直接显示左下图。在左下图的右侧，可以直接修改 asm 文件的内容，并保存。如果要更换其他 asm 文件，可以右键单击不要的 asm 文件，菜单列表中选择“Remove file”删除当前 asm 文件，然后遵循前述操作，重新添加新的 asm 文件进入 project。

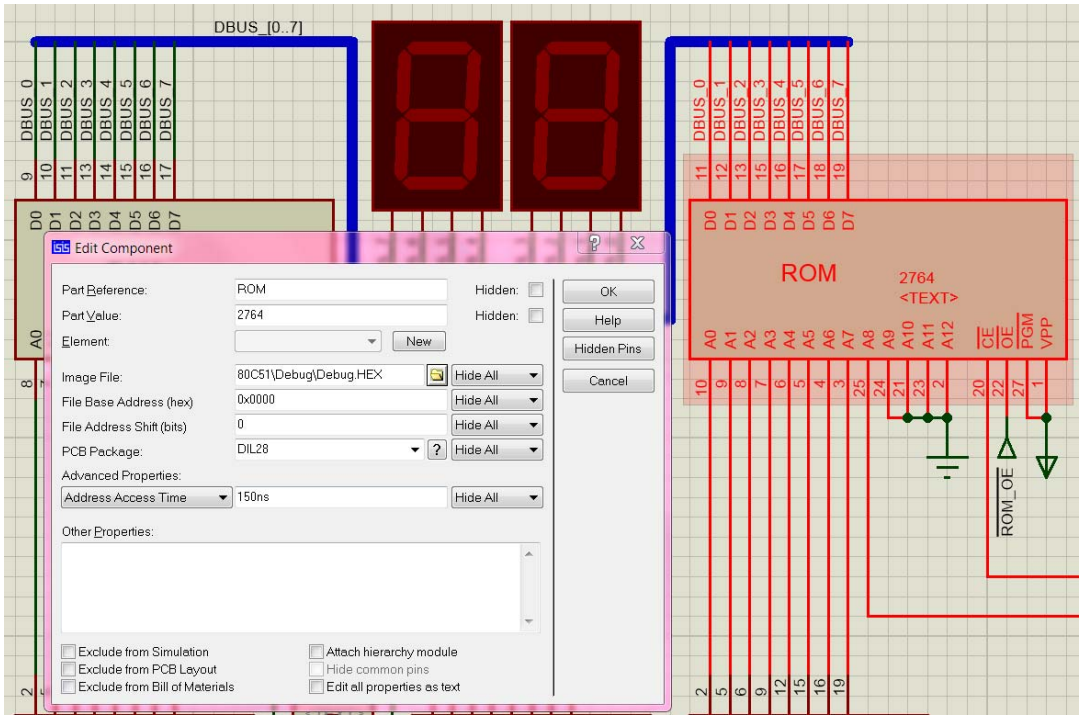
● 如果 asm 文件是第一次编译，则须右键单击 asm 文件，在弹出的菜单列表中选择点击 Project Settings，弹出对话框中 controller 选择”80C51”，注意 Embed files（或 Attach files ）要去掉选中！才能确保编译后生成的二进制 hex 文件在项目当前的目录下。最后，点击“OK”选项，如右下图所示。

● 右键单击 asm 文件，右键单击 asm 文件点击 Build Project 执行编译。编译成功后，在下方的显示栏里会出现“Compiled successfully!”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”。最好把 Debug.hex 文件重新命名，放置在固定的目录里保存。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，把之前编译的 hex 文件冲掉。



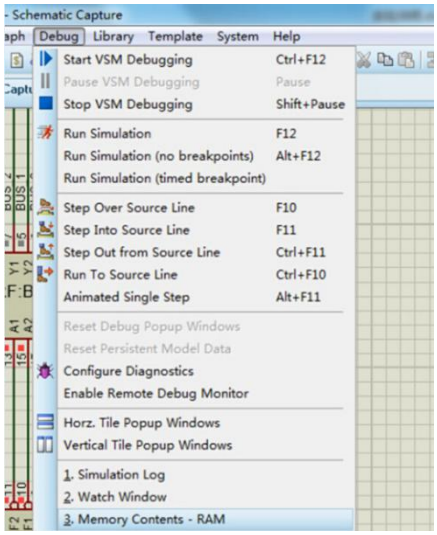
● 双击 ROM 芯片，弹出如下图的对话框：在 Image File 中选择所需烧写的 hex 文件，点击 OK，ROM 加载成功。最好不要直接加载上述直接编译生成的\80C51\Debug 路径里面的 Debug.hex，应加载重命名在固定目录下的 hex 文件。否则，当编译其他 asm 源程序后，Debug.hex 自动刷新，容易造成 ROM 错误加载本应烧写到其他 ROM 里面的 hex 文件。

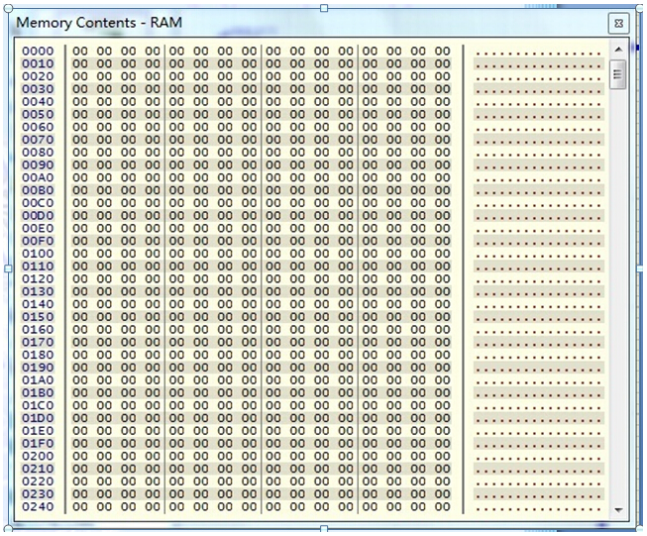


● 假设仿真出现以下报错，可能是项目路径里面原来加载 hex 文件的路径被取消，或是被移动到不同路径文件夹，或是文件名变了。特别是在项目里面有多个 ROM 需要加载的时候，要留意在移动或修改项目的时候不要破坏原有的 ROM 加载路径。

实验 2：查看项目中存储器 ROM 和 RAM 的内容

● 在 proteus 项目的仿真过程下按“暂停”，在点击菜单栏里的 Debug，在弹出的菜单列表最下方有 Memory Contents-ROM/RAM，如左下图所示。项目里有多少个存储器，就有多少个 Memory Contents 可供选择。点击所需查看存储器对应的 Memory Contents-选项，则弹出如右下图所示的当前存储器内容，其中蓝色为地址，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。值得注意的是，右下图显示的存储器地址是存储器芯片地址线 A0~A7 所指定的地址，即实验原理所述的低位“片内地址”。而“片内地址”加上存储器片选信号译码所用的高位“片间地址”，才是地址空间的真实地址。





实验 3：存储器 ROM 和 RAM 的读写

● 依照上述实验 1 步骤，加载 project.asm 文件编译的 hex 二进制文件到 ROM 芯片 2764。依照上述实验 2 方法，查看 ROM 烧写是否正确。

● 令图 2-16 各个开关的初始状态为：$\overline{\text{ROM\_OE}}$=1，$\overline{\text{RAM\_OE}}$=1，$\overline{\text{RAM\_WE}}$=1，$\overline{\text{SW\_BUS}}$=0。允许拨码开关给地址总线 ABUS_[0..11]赋值

● 操作拨码开关，向地址锁存器 ROM_AR 打入地址 024H，然后 2764 输出使能（$\overline{\text{ROM\_OE}}$=0）, 在数据总线 DBUS_[0..7]的红色数码显示管上查看存储单元 024H 读出的内容。再操作拨码开关，向地址锁存器 RAM_AR 打入地址 F80H，6116 输入使能（$\overline{\text{RAM\_WE}}$=0）把存储单元 024H 的内容写入存储单元 F80H。

● 2764 输出失效（$\overline{\text{ROM\_OE}}$=1）, 6116 输出使能（$\overline{\text{RAM\_OE}}$=0）,在数据总线的红色数码显示管上查看存储单元 F80H 写入的内容是否正确。

● 按照上述操作，把 ROM 地址 024H、028H、02CH、030H 的内容依次写入 RAM 地址 F80H、F81H、F82H、F83H。依照上述实验 2 方法，查看 RAM 的写入是否正确。

六、思考题

1、是不是烧写入 ROM 的 ASM 文件里面定义的所有数据都可以被访问到？假设把实验 1 中的 ASM 文件开头改为“ORG 0224H”，请问烧写进去的数据还能被读出么？如果不能，ROM 的片选电路要如何修改？

2、请给出 RAM 片选电路（F80H~FFFH）的第二种逻辑组合实现形式。假如 RAM 的范围改为 800H~874FH，请问片选电路的逻辑组合形式是怎样的？

3、为何 ROM 和 RAM 需要使用两个独立的 3-8 译码器？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。此时 ROM 和 RAM 的地址范围各自为多少？

2.5 微程序控制器实验

一、实验目的

1、了解微程序控制器的组成原理及工作方式。

2、掌握微程序的编制及写入，观察微程序的运行。

二、实验内容

设计一个可自动运行的 CPU 电路，利用时序电路来产生指令执行所需的时间先后次序，通过微程序控制器来代替手动拨码开关的人工操作，在数据通路中完成程序的跳转重定向。

三、实验器件

1、存储器 2764 和计数器（74LS163、74LS192）。

2、D 触发器（74LS74）、寄存器（74LS273）及移位寄存器(74LS194)。

四、实验原理

如图 2-17 所示，指令寄存器 IR（74LS273）、程序计数器 PC（74LS163）、存储器 PROGRAM(2764)及其地址寄存器 AR（74LS273）组成了一个“手动版”的数据通路。该通路是单总线结构（所有部件共用 8 位总线 BUS，通过两个绿色数码管显示 8 位总线信息）和单存储器结构（即程序和数据共享存储器 PROGRAM）。

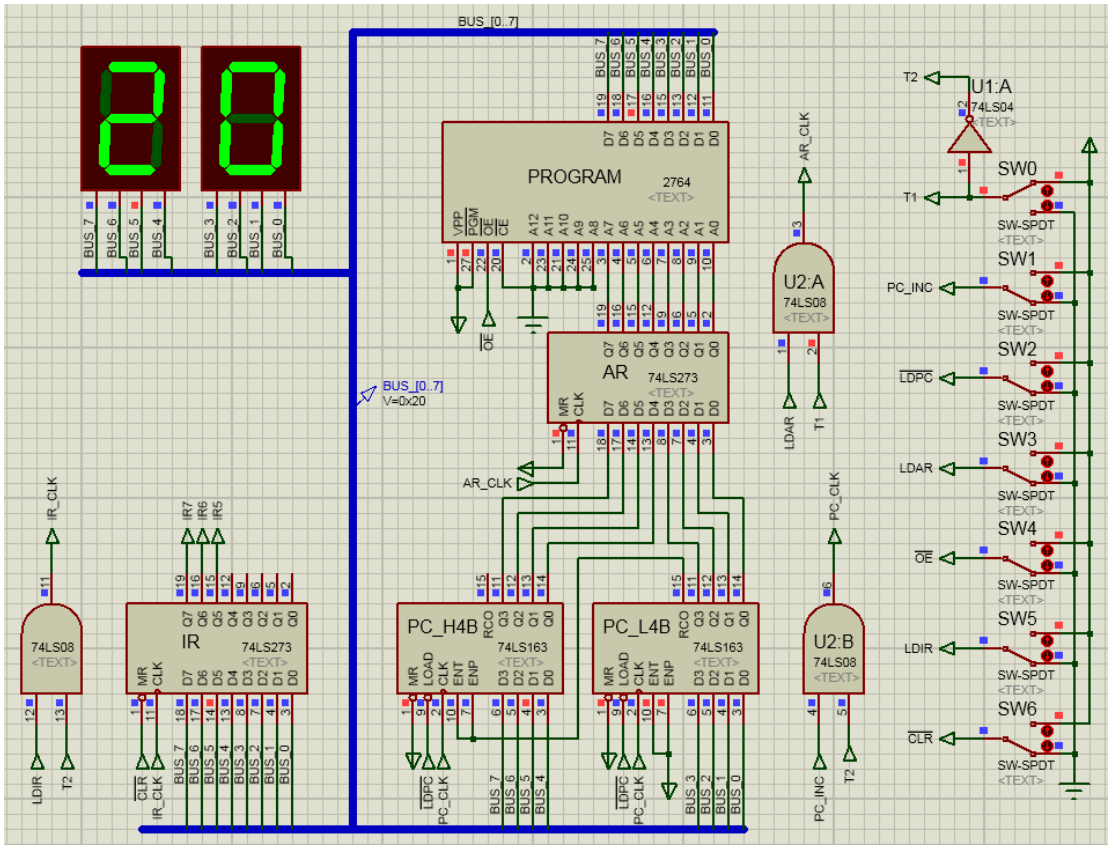


图 2-17　数据通路

在图 2-17 所示的数据通路中，程序计数器 PC 采用两个同步计数器 74LS163 串联形成八位递增计数器，74LS163 的逻辑功能表如表 2-9 所示：D_0、D_1、D_2、D_3为并行输入端；Q_0、Q_1、Q_2、Q_3为并行输出端；ENT、ENP 为递增使能端；LOAD 为置数端；MR 为直接无条件清零端；CLK 为时钟输入端；RCO 为进位输出端（当 Q_3-Q_0输出端递增溢出，则 RCO=1）。值得注意的是：表 2-9 中所列 74LS163 全部功能都必须在 CLK 端上升沿跳变后才能实现。

表 2-9　74LS163 逻辑功能表

MR	L_D	E_T	E_P	功能	Q_0 Q_1 Q_2 Q_3
0	×	×	×	清除	0 0 0 0
1	0	×	×	加载	Q_0 Q_1 Q_2 Q_3 = D_0 D_1 D_2 D_3
1	1	0	×	保持	Q_0 Q_1 Q_2 Q_3
1	1	×	0	保持	Q_0 Q_1 Q_2 Q_3
1	1	1	1	自加 1	{Q_0 Q_1 Q_2 Q_3}状态码+1

数据通路的运行包括了两个阶段：取指周期（取出指令）和执行周期（执行取指周期取出的指令）。而总线 BUS 上的信息流动路径也相应分为指令流和数据流。指令流：取指周期，指令信息从存储器流向指令寄存器（即 ROM->IR），实现指令译码；数据流：执行周期，数据信息从存储器流向唯一的指令执行部件 PC（即 ROM->PC），实现程序的重定向。值得注意的是，若有多个指令执行部件，则数据流还有不同的分支路径

因为数据通路只有 PC 作为唯一的执行部件，用以验证指令跳转的功能。所以如表 2-10 所示，本实验设计了空指令 NOP、停机指令 HLT 和两条不同类型的跳转指令：相对寻址的 JMP1 指令和直接寻址的 JMP2 指令。

表 2-10 微程序控制器实验的指令表

汇编助记符	[OP 码] I7 I6 I5	机器语言程序	指令说明
JMP1,addr	0 0 1	00100000; JMP1 xxxxxxxx; addr	相对寻址：程序最终跳转到地址 xxH 执行 PC->xxH
JMP2,[addr]	0 1 0	01000000; JMP2 xxxxxxxx;[addr]	直接寻址：程序最终跳转到地址 yyH 执行 PC->yyH，yyH=[xxH]
HLT	1 1 1	11100000; HLT	硬件停机
NOP	0 0 0	00000000; NOP	“空”指令 ，不执行任何操作

图 2-18 列出了上述四条指令的逻辑流程图，其中方框代表了从某个源部件经过总线 BUS 到达另一个目标部件的信息路径，菱形代表了指令译码结果（决定不同指令的跳转）。菱形之上的方框属于所有指令都必须经历的取指周期，而菱形之下各个分支代表了不同指令的执行周期。为了信息的稳定可靠传输，每一个方框都包含两个等长的节拍 T1 和 T2。T1 时刻，源部件的信息输出到总线 BUS，所以 AR_CLK 上升沿有效，使得 AR 地址对应的 ROM 存储单元稳定输出信息到总线 BUS；T2 时刻，总线 BUS 上的信息输入到目标部件，所以 PC_CLK 和 IR_CLK 上升沿有效，使得总线 BUS 上的信息稳定打入到 PC（重定向指令地址），或稳定打入到 IR（ 同时 PC 自加 1）。

通过图 2-18 可以发现，所有跳转指令的逻辑流程图实际上仅需要由两种方框组成：在取指周期中出现的指令流｛ROM→IR｝和在执行周期中出现的数据流｛ROM→PC｝。JMP1 和 JMP2 指令的区别仅仅在于 JMP2 指令做了递归操作，执行了两次相同的数据流

{ROM→PC} 操作。而 NOP 指令和 HLT 指令比较特殊，只有取指周期，没有执行周期。指令译码后，NOP 指令直接返回下一个取指周期，而 HLT 指令直接硬件停机。

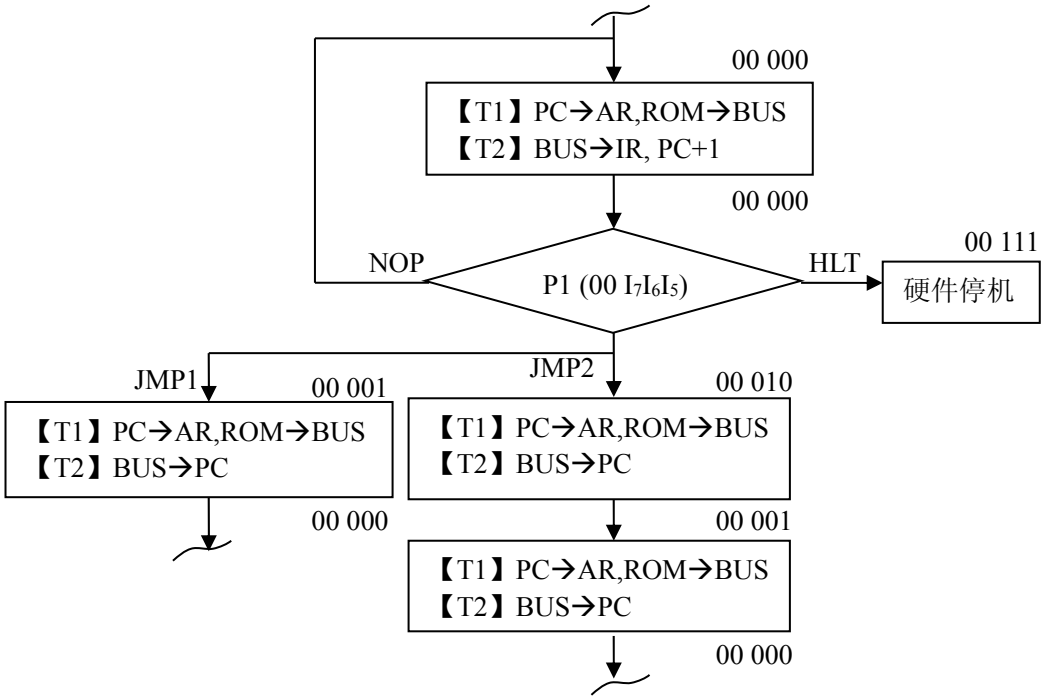


图 2-18 指令逻辑流程图

为了完成指令流{ROM→IR}和数据流{ROM→PC}，数据通路中需要根据一定的时序安排一系列的微操作信号来确保信息从某个部件输出到总线，再从总线输入到另一个部件。数据通路的操作信号说明如表 2-11 所示。值得注意的是：程序计数器 PC、地址寄存器 AR 以及指令寄存器 IR 的各自锁存脉冲信号 PC_CLK、AR_CLK 及 IR_CLK 分别由 PC_INC、LDAR 及 LDIR 信号在 T1 或 T2 节拍形成的。所以，数据通路中实际需要的微操作信号只有：PC_INC、$\overline{\text{LDPC}}$、LDAR、$\overline{\text{OE}}$、LDIR 和 $\overline{\text{CLR}}$ 信号。

表 2-11 数据通路的微操作信号表

信号名称	作 用	有效电平
PC_CLK	程序计数器 PC 的锁存脉冲信号	上升沿跳变有效
PC_INC	T2 节拍，PC_INC 形成 PC_CLK	高电平有效
LDPC\	程序计数器 PC 的加载信号	低电平有效
AR_CLK	地址寄存器 AR 的锁存脉冲信号	上升沿跳变有效
LDAR	T1 节拍，LDAR 形成 AR_CLK	高电平有效
OE\	存储器 PROGRAM 的读允许信号	低电平有效
IR_CLK	指令寄存器 IR 的锁存脉冲信号	上升沿跳变有效
LDIR	T2 节拍，LDIR 形成 IR_CLK	高电平有效
CLR\	指令寄存器 IR 的清零信号	低电平有效

指令流{ROM→IR}和数据流{ROM→PC}各自所需的微操作信号及其信息路径如表 2-12 所示，每个周期内微操作信号再与(T1,T2)节拍逻辑组合。如果把表 2-11 中的微操作信号用人工拨码开关来实现的话，实验者可以根据表 2-12 手动执行微操作信号，以 T 节拍为基准时钟，顺序依次实现指令逻辑流程图 2-18 中的各个方框，完成一条机器指令的执行。其中，实验者在菱形 P1 时刻，可以通过观测指令寄存器 IR 的{IR7,IR6,IR5}是否与某个指令的 OP 码相同，来判断不同指令所需的执行周期（即朝哪个分支继续执行）。

表 2-12 指令流和数据流的微操作信号表

	微操作信号（工作节拍）	功能
指令流 ROM→IR	LDAR(T1), OE (T1&T2), LDIR(T2), PC_INC(T2)	【T1】：PC→AR,ROM→BUS 【T2】：BUS→IR, PC+1
数据流 ROM→PC	LDAR(T1), OE (T1&T2), LDPC (T1&T2), PC_INC(T2)	【T1】：PC→AR,ROM→BUS 【T2】：BUS→PC

进而，我们可以将图 2-17 的“手动版”数据通路升级为“自动版”的 CPU 电路，如下图 2-19 所示。“自动版”CPU 由数据通路、时序电路及微程序控制器三大部分组成：下方左边的数据通路保持不变，而上方红色框内的微程序控制器(CONTROL UNIT)取代了拨码开关及人工操作者；下方右边红色框内的时序电路(CLOCK UNIT) 则负责提供数据通路和微程序控制器依序运行所需的节拍序列｛T1, T2, T3, T4｝。

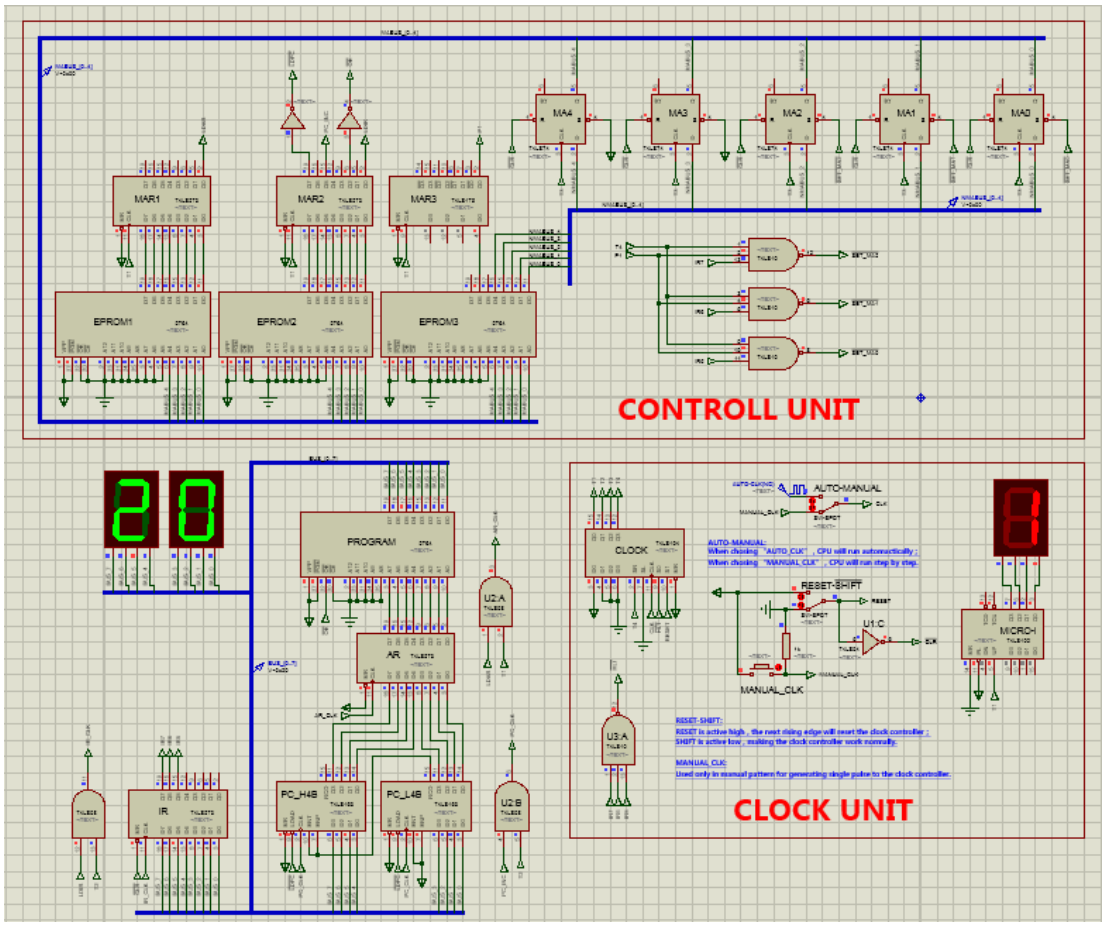


图 2-19 微程序控制器实验电路

微程序控制器(CONTROL UNIT)的主要任务是完成当前指令的翻译和执行，即将当前存储器 PROGRAM 读出的指令转换成硬件逻辑部件工作的微操作信号序列，完成信息传送和各种处理操作。因此，微程序控制器将表 2-11 中的一条微操作信号视为一个微命令，而所有微命令的集合以并行二进制序列的形式构成微指令，如表 2-13 所示。

表 2-13 微指令结构图

微代码	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
微命令	LDAR	LDPC	\	PC_INC	\	\	OE	\	LDIR	\	P1	\	uA4	uA3	uA2	uA1	uA0

上表中的微指令字长 24 位，采用下址转移方式确定后续运行的微指令：微指令的 1-5 位表示顺序下一条微指令地址 [uA4-uA0]；而 6-24 位表示微命令集合（由于微指令 18-24 位没有对应的微命令，所以表 2-13 只显示了微指令 1-17 位）。所有微指令都存储在专用的控制存储器(CM)中，如图 2-20 所示：控制存储器 24 位，由 3 个 ROM 芯片 2764 组成。微命令寄存器 18 位，由两个寄存器 74LS273 和一个 4 位触发器 74LS175 组成。

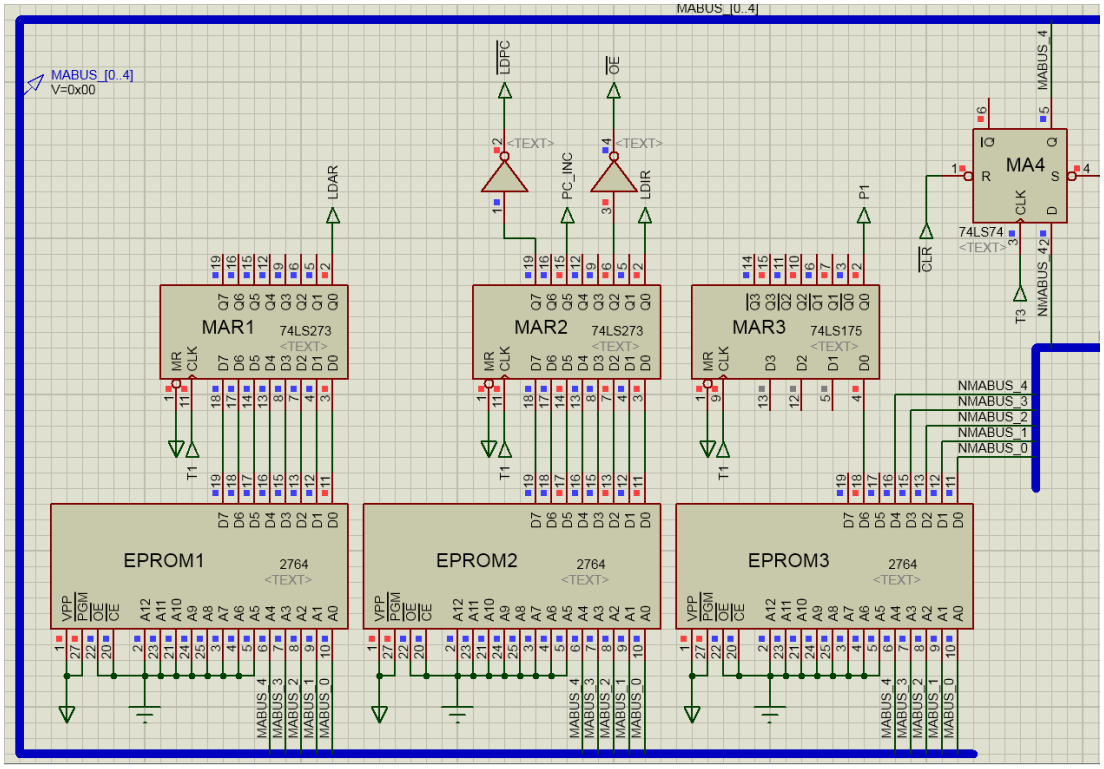


图 2-20 微程序控制器的控制存储器（CM）电路

在指令逻辑流程图 2-18 中，一个方框对应一条特定的微指令，方框右上方是当前微指令在控制存储器 CM 中的地址，右下方是顺序下一条微指令地址[uA4-uA0]。若考虑方框右上方和右下方微指令地址的差异，在图 2-18 中总共有四种不同的方框，所以本实验的微指令代码表 2-14 所示。每条微指令的 1-5 位即下址字段[uA4-uA0]，而 6-24 位是控制字段。

在 T1 节拍，上图 2-20 中的控制存储器把当前微指令打入微命令寄存器，微指令中所有标识为“1”的位表示执行了相应的微操作（且数据通路自动生成 LDAR→AR_CLK），从而实现对应方框内的信息路径；反之，标识为“0”的位则代表没有执行对应的微操作。在 T2 节拍，数据通路自动生成了 LDIR→IR_CLK 和 PC_INC→PC_CLK 的后续操作。

表 2-14 微指令代码表

微地址	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
00000	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0
00001	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
00010	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1
00111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

一条机器指令所经历的所有微指令（方框）组成的序列称为该机器指令的微程序，每条微指令包含了相同的取指周期微指令和各异的执行周期微指令序列。取指周期微指令地址必须是控制存储器 CM 首地址 00000（因为系统启动时从首地址开始运行）。而菱形框 P1 (00 $I_7I_6I_5$)是地址转移逻辑电路，当取指周期结束的时候，根据机器指令 OP 码的 $I_7I_6I_5$ 位进行译

码，转入该机器指令对应的执行周期微指令序列的入口地址[00$I_7I_6I_5$]。指令｛NOP, HLT, JMP1, JMP2｝的执行周期微指令序列的入口地址分别为｛00000, 00111,00001,00010｝。

在所有指令执行周期的微指令序列末尾，最后一条微指令的[uA4-uA0]都必须是取指微指令地址 00000，即一条机器指令结束后必须返回公共的取指微指令，以便于取出下一条指令。而 NOP 指令因为取指后译码的执行指令入口地址仍为 00000，直接指向下一条指令的取指周期微指令；所以 NOP 指令在取指后实际没有执行操作。

本实验微程序控制器的地址转移电路如图 2-21 所示，微指令地址寄存器五位(MA4-MA0)，由触发器 74LS74 组成，均自带清“0”端和预置端。地址转移逻辑需要增加 T3、T4 节拍，T3 时刻，当前微指令的后续微指令地址[uA4-uA0]打入微地址寄存器。T4 时刻，若当前微指令不是取指周期微指令（即 P1=0），则不执行任何操作，当前微地址转为顺序下一条微指令地址[uA4-uA0]；若当前微指令是取指周期微指令（即 P1=1），则图 2-21 中的地址转移逻辑电路根据指令寄存器 IR 的 $I_7I_6I_5$ 位强制置位寄存器 MA2-MA0，即修改当前微地址[uA2-uA0]位，转向该指令的执行周期微指令序列的入口地址[00$I_7I_6I_5$]。

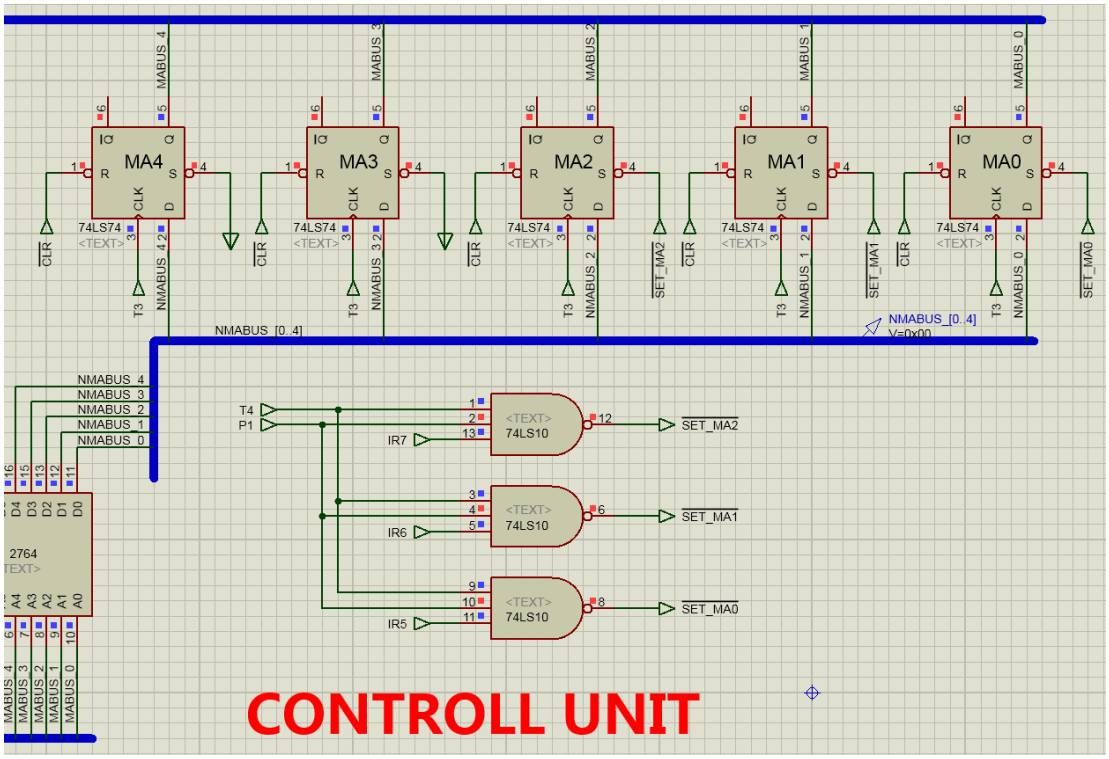


图 2-21 微程序控制器的地址转移电路

综上所述，一条微指令包括了 4 个节拍（T1、T2、T3、T4），其作用分别是：

T1：当前微指令的微操作信号使能；信息从源部件输出到总线 BUS

T2：信息从总线 BUS 打入目的部件；程序计数器 PC+1（取指周期微指令专用）

T3：微指令下址取址

T4：根据指令寄存器 IR 所存的指令 OP 码修改微指令下址 00$I_7I_6I_5$（取指周期微指令专用）

微程序控制器必须严格按照节拍时序来工作，所以时序电路的设计是非常重要的，本实验的时序电路如图 2-23 所示。CLK 为全系统的基准时钟信号，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），或者通过人工操作按键 MANUAL_CLK 手动步

进；移位寄存器 74LS194 用来作为节拍生成器：当 $\overline{HLT}$=1 且 RESET=0，74LS194 的状态{S0, S1}={1, 0}，工作模式为右移。而 S_R=T4 使得 74LS194 处于循环右移模式，以 CLK 为基准时钟，周而复始地发送 4 个基准时钟长度的时序节拍 T1-T4（T1->T4 的一次循环即为一个微指令的运行周期，称为 CPU 周期），如图 2-22 所示：

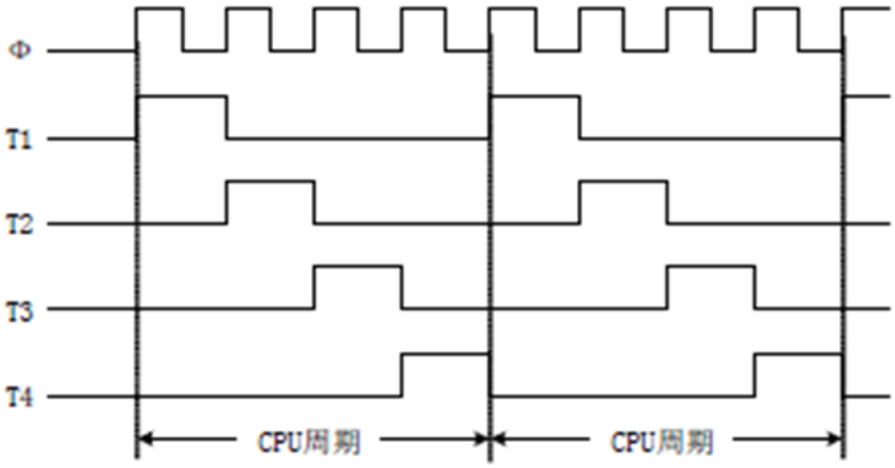


图 2-22 基于 74LS194 的节拍时序状态图

为了便于观测微程序的运行，时序电路提供了微指令显示功能：一个由 T1 上升沿驱动的升序计数器 74LS192，通过数码管显示当前运行第几条微指令。时序电路还提供了软件停机的“断点”功能（便于调试），即 HLT 指令：当指令寄存器 IR 的 OP 码 $I_7I_6I_5$=111 的时候，硬件逻辑生成 $\overline{HLT}$=0，此时 74LS194 的状态{S0, S1}={0, 0}，工作模式为保持，节拍时序状态静止在｛T1, T2, T3, T4｝={0, 1, 0, 0}，即停机在 HLT 指令的取指周期 T2 节拍上。

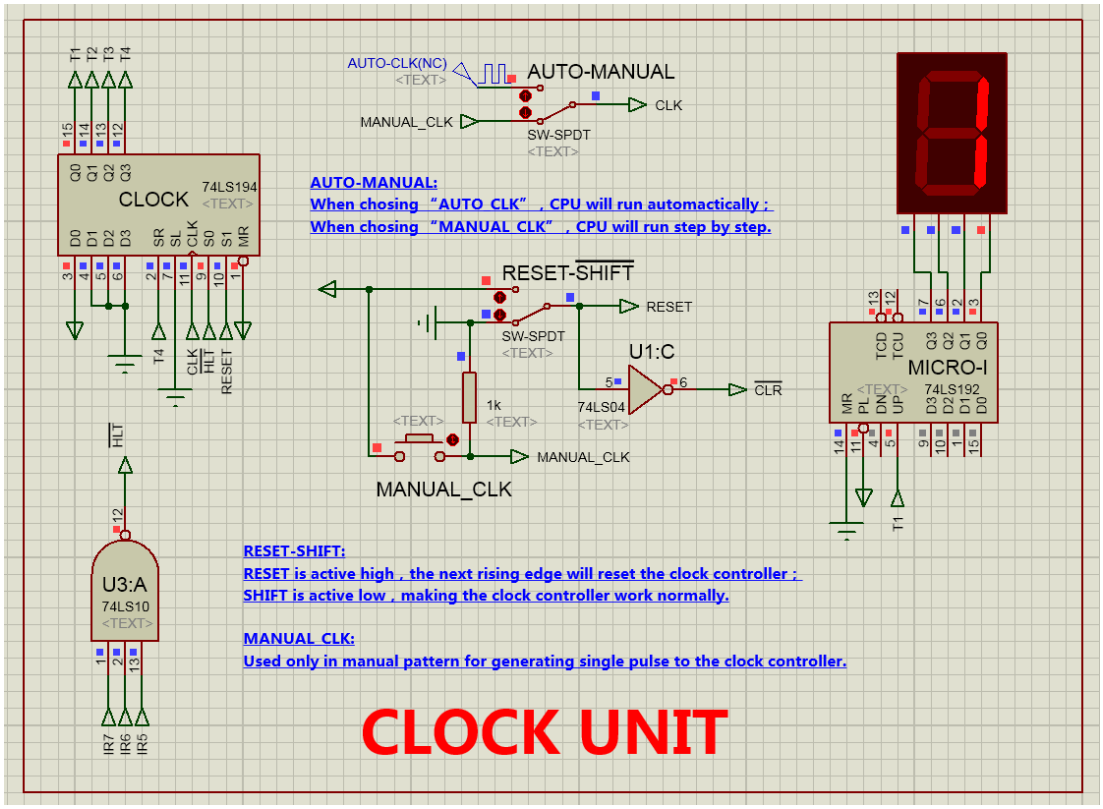


图 2-23 微程序控制器实验的时序电路

此外，时序电路设置了 RESET 电路，用来初始化时序发生器 74LS194，以及 HLT 指令停机后跳出“断点”。其“重启”的操作步骤如下：

1） RESET 置 1，则 $\overline{CLR}$=0，清零指令寄存器 IR（指令 OP 码归零且 $\overline{HLT}$=1）和微地址寄存器 MA4-MA0（微程序控制器从取指周期开始运行）。此时，74LS194 的状态{S0,S1}={1,1}，工作模式为送数。

2） 手动按一次 CLK，将 {T1,T2,T3,T4} 置位初始化值{1,0,0,0}。

3） RESET 置 0，74LS194 恢复循环右移模式，进入第一条指令的取指周期节拍时序。

（注：若是 HLT 指令“断点”后操作，则进入 HLT 指令后续下一条指令的取指周期）

五、实验步骤

实验 1：手动版程序通路运行

●在数据通路（手动版）的存储器 PROGRAM 中，烧写进去如下所示的机器语言程序文本（烧写 ROM 方法参见 2.4 存储器实验）：

```
ORG    0000H
       DB      00100000B; JMP1,06H
       DB      00000110B
       DB      11101010B; HLT
       DB      00001010B;

       DB      00000000B
       DB      00000000B
       DB      00000010B; NOP/[ADDR]
       DB      11100001B; HLT

       DB      01000000B; JMP2,06H->02H
       DB      00000110B
END
```

● 在图 2-17 所示的数据通路（手动版）上，依据指令逻辑流程图 2-18，手动拨码开关执行上述程序，观察每一步手工操作的结果（寄存器 AR、IR、PC 及总线 BUS 信息）。

实验 2：基于 JMP1 和 JMP2 指令的汇编程序

● 根据表 2-14 微指令代码表，将下列微程序分别烧写入 2-20 所示微程序控制器的控制存储器 EPROM1、EPROM2 及 EPROM3（切记勿写错存储器!）。

EPROM1 烧写内容	EPROM2 烧写内容	EPROM3 烧写内容
ORG 0000H	ORG 0000H	ORG 0000H
DB 00000001B	DB 00100101B	DB 01000000B
DB 00000001B	DB 10100100B	DB 00000000B
DB 00000001B	DB 10100100B	DB 00000001B
DB 00000000B	DB 00000000B	DB 00000000B
DB 00000000B	DB 00000000B	DB 00000000B
DB 00000000B	DB 00000000B	DB 00000000B
DB 00000000B	DB 00000000B	DB 00000000B
DB 00000000B	DB 00000000B	DB 00000000B
END	END	END

● 启动仿真图 2-19 所示的微程序控制器实验电路，首先 RESET=1，然后手动 CLK 一次，初始化 74LS194 节拍发生器｛T1,T2,T3,T4｝={1,0,0,0}，最后 RESET=0，恢复正常节拍时序。开始手动 CLK 单步执行上述实验 1 的机器语言程序。观察每一次单步执行的结果，例如寄存器 AR、IR、PC 及总线 BUS 信息。（注意程序进入 HLT 指令“断点”后需要重复上述初始化操作才能跳出“断点”）

● 依照上述启动初始化方法，在 RESET=0 恢复正常运行后，转为 AUTO_CLK 方波信号源自动 CLK。当程序通路陷入“断点”时刻（HLT 指令），查看寄存器 AR、IR、PC、总线 BUS 信息及微指令周期数指示。

实验 3：增加新指令 JMP3

● 增加一条间接寻址的跳转指令 JMP3，如下所示。请补充指令逻辑流程图 2-18 及微指令代码表 2-14（新增的微指令地址定为 00011），以实现 JMP3 指令。

汇编助记符	OP 码	机器语言程序	指令说明
JMP3,[[addr]]	01100000	01100000; JMP3 xxxxxxxx;[[addr]]	间接寻址：程序最终跳转到地址 zzH， PC->zzH, zzH=[yyH], yyH=[xxH]

●在图 2-19 所示微程序控制器实验电路的存储器 PROGRAM 中，烧写进去如下所示的机器语言程序文本（烧写 ROM 方法参见 2.4 存储器实验）：

```
ORG   0000H
      DB    00100000B; JMP1,06H
      DB    00000110B
      DB    11101010B; HLT
      DB    00001010B; NOP/[ADDR]

      DB    01100000B; JMP3,0BH->03H->0AH
      DB    00001011B
      DB    00000010B; NOP/[ADDR]
      DB    11100001B ; HLT

      DB    01000000B; JMP2,06H->02H
      DB    00000110B
      DB    11100000B; HLT
      DB    00000011B; NOP/[ADDR]
END
```

● 根据实验 2 所述方法，分别手动单步执行或 AUTO_CLK 自动执行实验 3 新的机器语言程序文本，观察单步执行或“断点”时刻的寄存器 AR、IR、PC 及总线 BUS 信息。

六、思考题

1、如图 2-19 所示的 微程序控制器实验电路中，如果采用微命令来实现 HLT 指令（例如安排控制存储器中一位作为 $\overline{\text{HLT}}$ 信号）是否可行？请修改电路试验。

2、在实验 2 和实验 3 的机器语言文本中，有部分地址标示“NOP/[ADDR]”，为何相同位置会有不同的执行效果？在什么情况下执行到该处是 NOP 指令（顺序滑过）？什么情况下执行到该处是打入 PC 的地址（跳转）？

3、本实验的控制存储器 CM 最多可容纳多少条微指令？而指令体系最多可容纳多少条机器指令？微指令容量和指令容量各自是由于什么因素限定的？

4、假设只用两个 2764 芯片作为控制存储器，节拍脉冲精简到三个脉冲（T1-T3），请问微程序怎么改？机器语言程序需要改么？请修改电路试验。

2.6 硬布线控制器实验

一、实验目的

1、掌握硬布线控制器的组成原理及设计方法。

2、了解单周期和多周期硬布线控制器的各自特点。

二、实验内容

分别设计单周期和多周期硬布线版本的 CPU 电路，两者在功能上皆完全兼容前述微程序版本的 CPU：数据通路相同，指令体系相同，仅用硬布线逻辑取代微程序控制器来产生各时序阶段的微操作信号。

三、实验器件

1、存储器 2764 和计数器（74LS163、74LS192）。

2、寄存器（74LS273）及移位寄存器(74LS194)。

四、实验原理

硬布线控制器的数据通路和机器指令集都与微程序控制器版本保持一致，不同之处是实现各指令功能所需的微操作信号不是由微指令的方式产生，而是由微操作信号出现的逻辑条件（机器指令）及相应时序的逻辑组合形成。如表 2-15 所示。本实验共定义了空指令 NOP、停机指令 HLT 和三条不同类型的跳转指令 JMPx：相对寻址的 JMP1 指令、直接寻址的 JMP2 指令和间接寻址的 JMP3 指令。

表 2-15 硬布线控制器实验的指令表

汇编助记符	[OP 码] I7 I6 I5	机器语言程序	指令说明
JMP1,addr	0 0 1	00100000; JMP1 xxxxxxxx; addr	相对寻址：程序跳转到地址 xxH 执行 PC->xxH
JMP2,[addr]	0 1 0	01000000; JMP2 xxxxxxxx;[addr]	直接寻址：程序跳转到地址 yyH 执行 PC->yyH，yyH=[xxH]
JMP3,[[addr]]	0 1 1	01100000; JMP3 xxxxxxxx;[[addr]]	间接寻址：程序跳转到地址 zzH 执行 PC->zzH，zzH=[yyH]，yyH=[xxH]
HLT	1 1 1	11100000; HLT	硬件停机
NOP	0 0 0	00000000; NOP	“空”指令 ，不执行任何操作

硬布线控制器有两种设计方法：单周期和多周期硬布线控制器。单周期硬布线控制器的定义是所有指令均在一个等长的时钟周期内完成，这个时钟周期是由执行时间最长的指令决定的。 在微程序控制器实验的指令逻辑流程图 2-18 中的方框代表了一个 CPU 周期，又代表了该周期内执行的一条微指令。在硬布线控制器实验中，虽然没有微指令结构及微程序编程，但是指令的逻辑流程是保持一致的，图 2-18 中的一个方框依旧表示一个 CPU 周期，同时代表了硬布线逻辑在此 CPU 周期的状态。因为微程序控制器实验中，JMP3 指令执行时间最长（4 个 CPU 周期），所以在单周期硬布线控制器实验中，除了 HLT 指令在取指周期 M1 末尾硬件停机。其他指令都必须经历四个 CPU 周期 M1-M4。所有指令的状态机如图 2-24 所

示，与前述微控制器版本的图 2-18 非常类似，方框右上方是当前 CPU 周期标识 Mx。M1 是取指周期，M2、M3 和 M4 是执行周期。

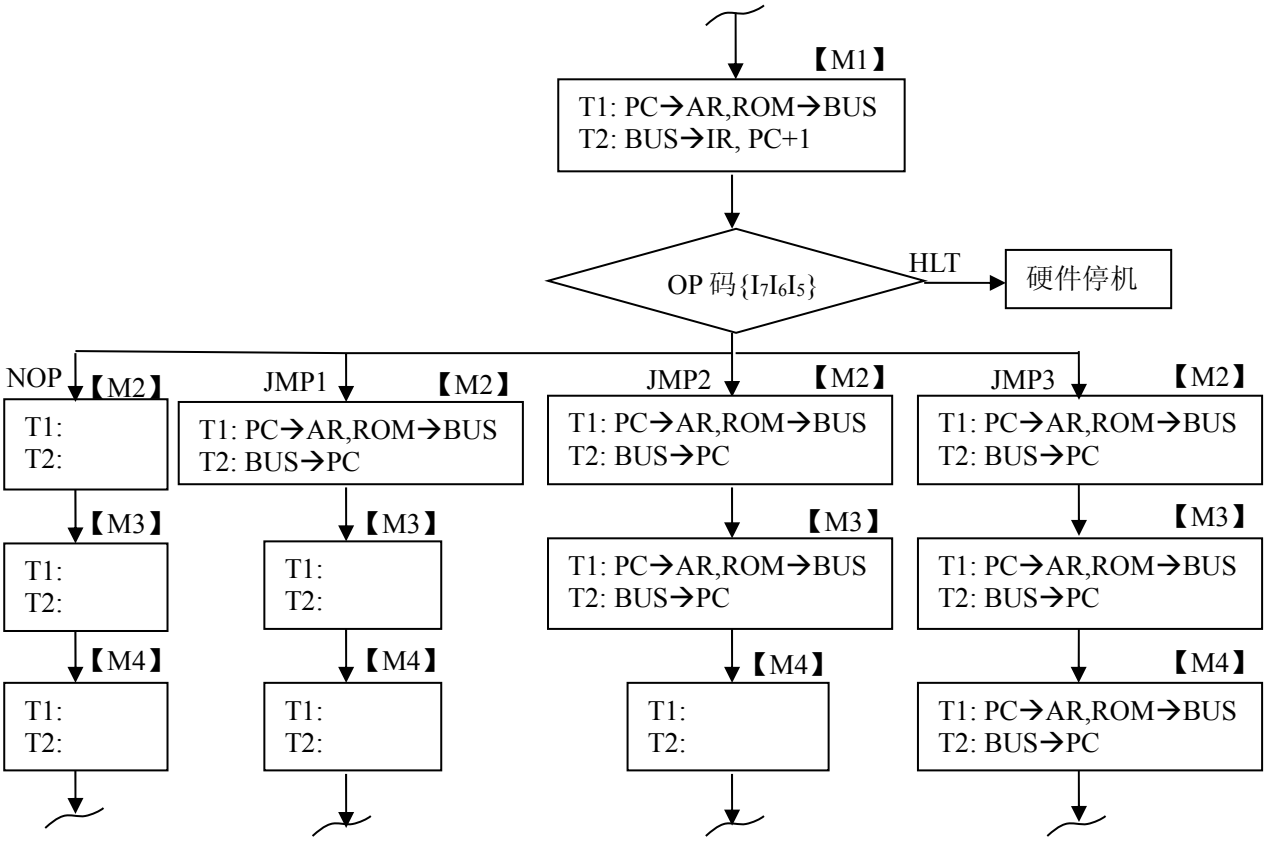


图 2-24 单周期硬布线控制器的指令状态机

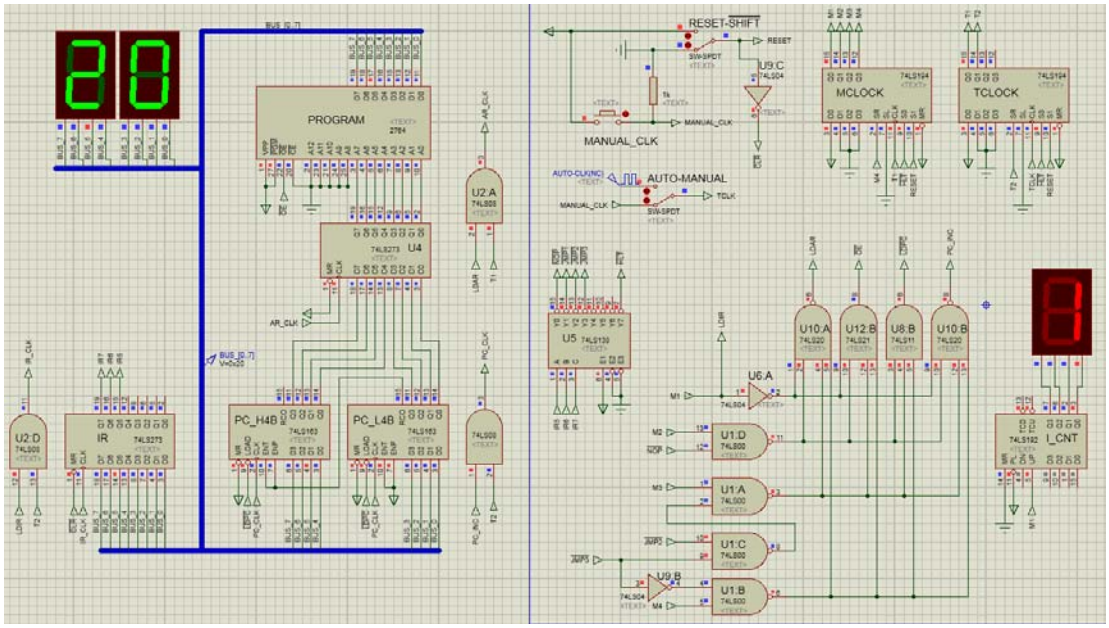


图 2-25 单周期硬布线控制器

单周期硬布线控制器实验的电路图如上图 2-25 所示，由左边的数据通路和右边的时序电路及硬布线控制逻辑组成，左边的数据通路部分与微程序版本完全相同。而右边的时序电路如图 2-26 所示，有两级时序发生器：节拍 T1-T2 和 CPU 周期 M1-M4。基准时钟信号 TCLK

驱动第一级移位寄存器 TCLOCK，循环发送节拍 T1->T2->T1->T2->T1……，每一次{T1,T2}时序循环即为一个 CPU 周期 M；而节拍 T1 驱动第二级移位寄存器 MCLOCK，循环发送 4 个等长的 CPU 周期 M1->M4，每一次循环即为一个机器指令周期。所以，单周期硬布线控制器可以周而复始的产生时序：M1{T1,T2}, M2{T1,T2}, M3{T1,T2}, M4{T1,T2}, M1{T1,T2},……。此外，下图 2-26 中的基准时钟电路，RESET 复位电路，以及用以系统初始化和跳出 HLT 指令“断点”的“重启”操作步骤均与微程序版电路保持一致。

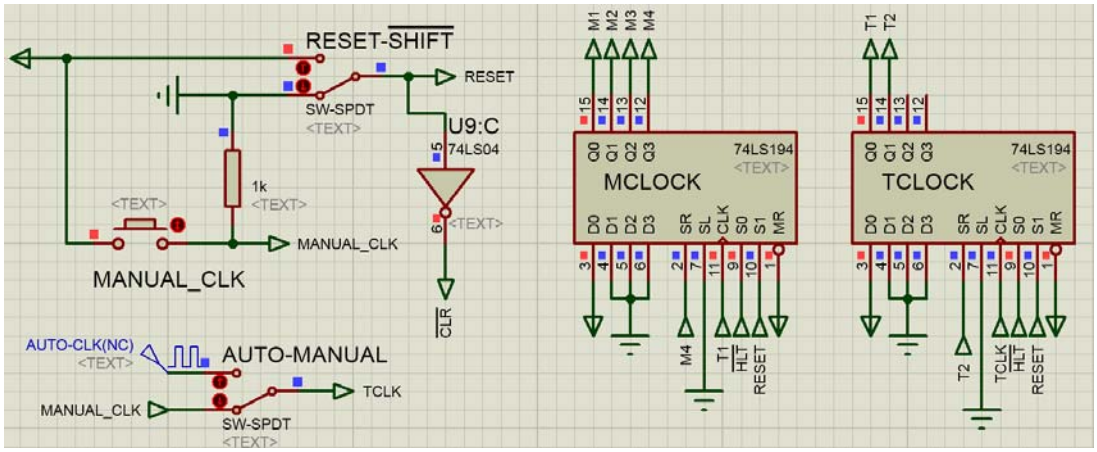


图 2-26 单周期硬布线控制器的时序电路

硬布线控制器的设计思想是：控制器发出的微操作信号是时间因素和空间因素的函数。各条机器指令的 OP 码{$I_7I_6I_5$}经过译码产生的不同指令信号构成了空间特征，而 CPU 周期时序信号 Mx 构成了时间特征，硬布线控制器就是把输入的时间特征和空间特征逻辑组合，输出具有定时特点的微控制信号。而微操作信号的组合逻辑如下表 2-16 所示：

表 2-16 微操作信号的组合逻辑

	M1	M2	M3	M4
LDIR	NOP/HLT/ JMP1/JMP2/JMP3			
LDAR	NOP/HLT/ JMP1/JMP2/JMP3	JMP1/JMP2/JMP3	JMP2/JMP3	JMP3
OE\	NOP/HLT/ JMP1/JMP2/JMP3	JMP1/JMP2/JMP3	JMP2/JMP3	JMP3
LDPC\		JMP1/JMP2/JMP3	JMP2/JMP3	JMP3
PC_INC	NOP/HLT/ JMP1/JMP2/JMP3	JMP1/JMP2/JMP3	JMP2/JMP3	JMP3

根据上表 2-16，可以设计出如图 2-27 中间所示的由指令信号和时序信号 Mx 构成的微操作信号的组合逻辑，其逻辑表达式如下（注：HLT 指令只有 M1 周期）：

LDIR=M1 （注：因为所有指令在 M1 都用到 LDIR，所以简化为 M1）

LDAR=M1+$\overline{NOP}$ • M2+(JMP2+JMP3) • M3+JMP3 • M4（注：$\overline{NOP}$=JMP1+JMP2+JMP3）

OE= M1+$\overline{NOP}$ • M2+(JMP2+JMP3) • M3+JMP3 • M4

LDPC= $\overline{NOP}$ • M2+(JMP2+JMP3) • M3+JMP3 • M4

PC_INC= M1+$\overline{NOP}$ • M2+(JMP2+JMP3) • M3+JMP3 • M4

上述时序信号 Mx 由图 2-26 的时序发生器 MCLK 提供，而每条机器指令唯一对应的指令信号则从指令寄存器 IR 输出的 OP 码{$I_7I_6I_5$}经过下图 2-27 左边的 3-8 译码器 74LS138 转换成。值得注意的是，当指令寄存器 IR 的 OP 码 $I_7I_6I_5$=111 的时候，3-8 译码器产生的指令信号 HLT\把上图 2-26 中的两级时序发生器 MCLK 和 TCLK 的工作模式改为保持，即整个系统停机在 M1 取指周期的 T2 节拍上。

此外，为了便于观测程序的运行，本实验提供了指令显示功能：下图 2-27 右边是一个由 M1 上升沿驱动的升序计数器 74LS192，通过数码管显示当前运行是第几条指令。

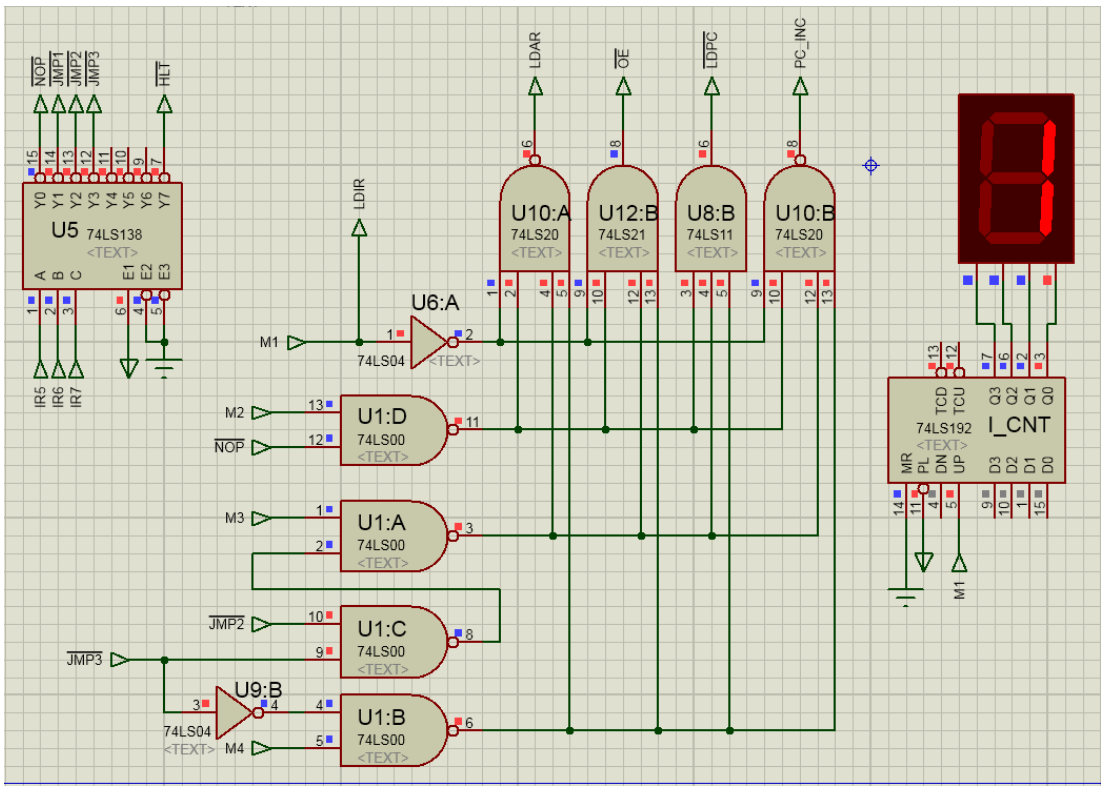


图 2-27 单周期硬布线控制器的硬布线控制逻辑

单周期硬布线控制器虽然直观简单，但是存在运行效率不高的问题： 所有指令的周期都是一致的，由执行时间最长的指令 JMP3 决定。因此，如图 2-24 所示，对于其他执行时间较短的指令来说，存在着一个或多个空闲的 CPU 周期，造成了 CPU 性能的闲置浪费。

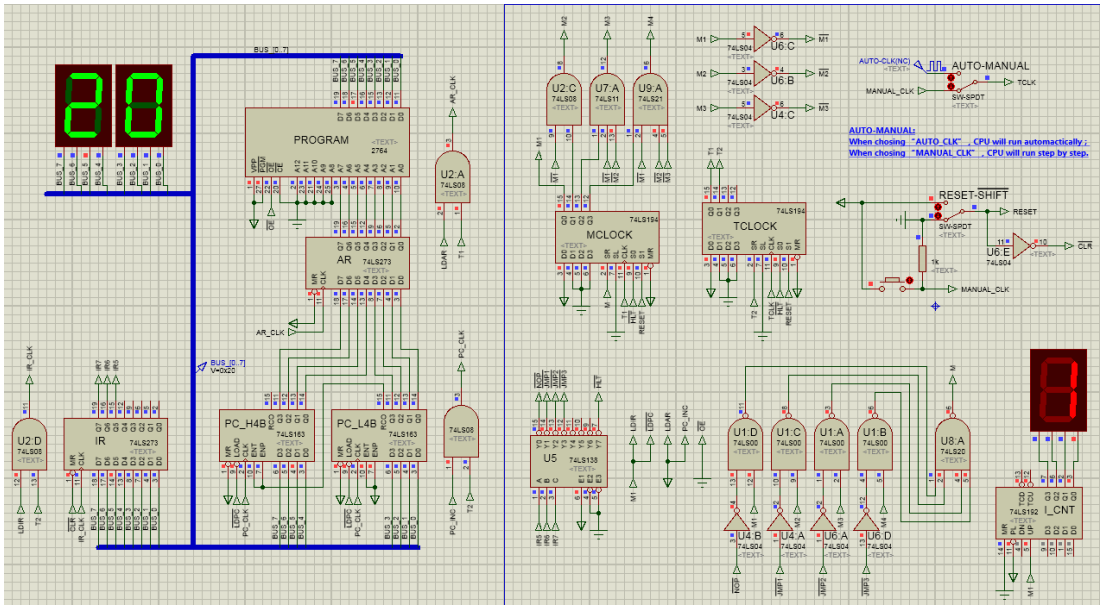


图 2-28 多周期硬布线控制器

硬布线控制器的另一种较为普遍的设计方法是多周期硬布线控制器，其定义是不同指令

依据各自的功能需要，可以有不同数目的 CPU 周期。多周期硬布线控制器的实验电路图如上图 2-28 所示，左边的数据通路部分依旧与微程序版本保持一致，而右边的时序电路和硬布线逻辑则实现了如下图 2-29 所示的多周期硬布线版本的指令状态机。与单周期硬布线版本的指令状态机图 2-24 比较，多周期硬布线版本的指令的 CPU 周期数都是按照指令功能所需定制，没有空闲的 CPU 周期，因此系统的运行效率大大提高。

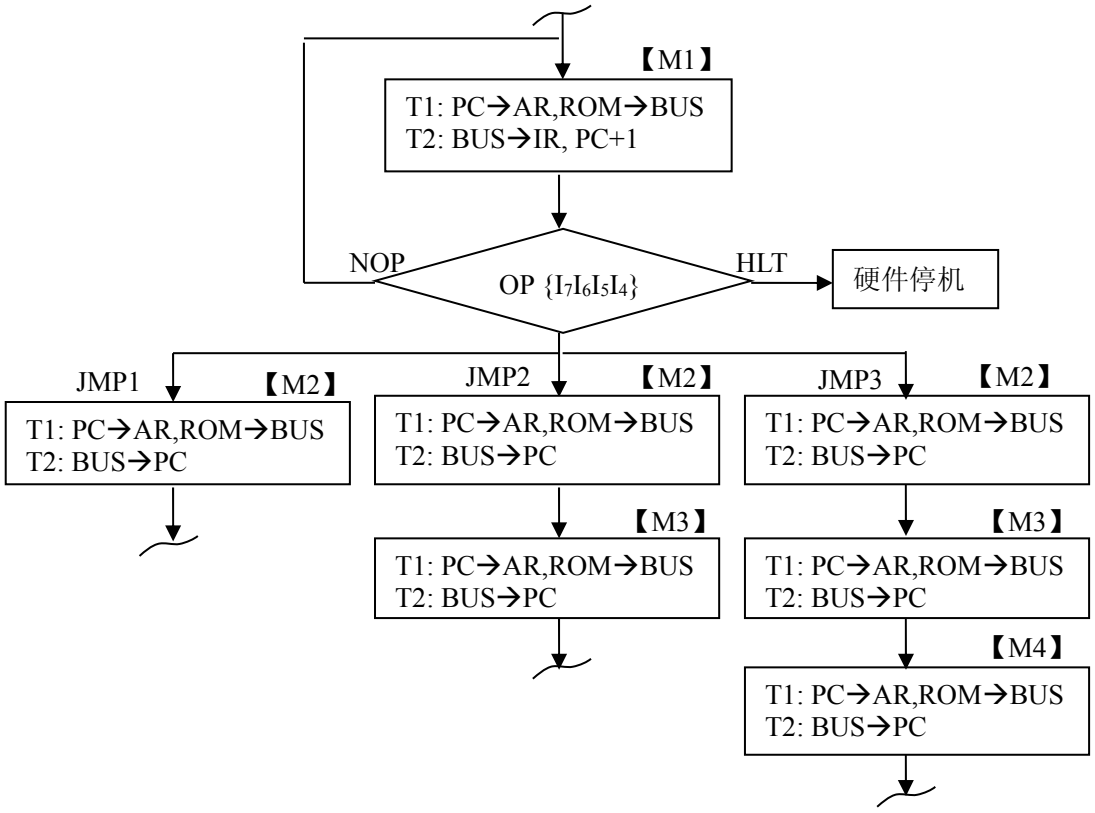


图 2-29 多周期硬布线控制器的状态机描述

如下图 2-30 所示，时序电路依旧有两级时序：节拍 T1-T2 和 CPU 周期 M1-M4。基准时钟信号 TCLK 驱动第一级移位寄存器 TCLOCK，循环发送节拍 T1->T2->T1->T2->T1……，每一次{T1,T2}时序循环即为一个 CPU 周期，与单周期版本保持一致。节拍 T1 同样驱动第二级移位寄存器 MCLOCK，但是 MCLOCK 的循环周期不是统一的 4 个 CPU 周期了，而是由移位寄存器的右移串行输入端 S_R 的 M 值决定。根据上图 2-29 的指令状态机，可以推出如下表 2-17 所示的各个指令所需的 CPU 周期：

表 2-17 多周期硬布线控制器的指令周期组成

指令	NOP/HLT/	JMP1	JMP2	JMP3
CPU 周期	M1	M1+ M2	M1+ M2+M3	M1+ M2+M3+M4

根据上表可以推导出 M 的逻辑表达式如下（具体电路如图 2-31 中间的组合逻辑所示）：

M = NOP • M1+ JMP1 • M2+JMP2 • M3+JMP3 • M4

因为从指令寄存器 IR 输出的 OP 码{$I_7I_6I_5$}经过下图 2-31 左边的 3-8 译码器 74LS138，只可能输出唯一的指令信号，所以任何情况下上述表达式中只有一个分项是有效的。假设当前执行的是指令 JMP1，则根据上式可知 M=M2，MCLOCK 输出的 $Q_0Q_1Q_2Q_3$ 时序：1000、0100、1010、0101、1000…为了使得 MCLOCK 状态机保持“M1→M2→ M1→….”，下图 2-30 中 MCLOCK 的 $Q_1Q_2Q_3$ 端外接了限制项，只有前述移位 Q_{x-1} 都为 0 的状态下，Q_x=1 才成立；

否则，Q_x清零。因此，在 M=M2 的条件下，M1-M4 的时序最终为：1000、0100、1000、0100、1000……即 MCLOCK 状态机保持为“M1→M2→ M1→….”状态。

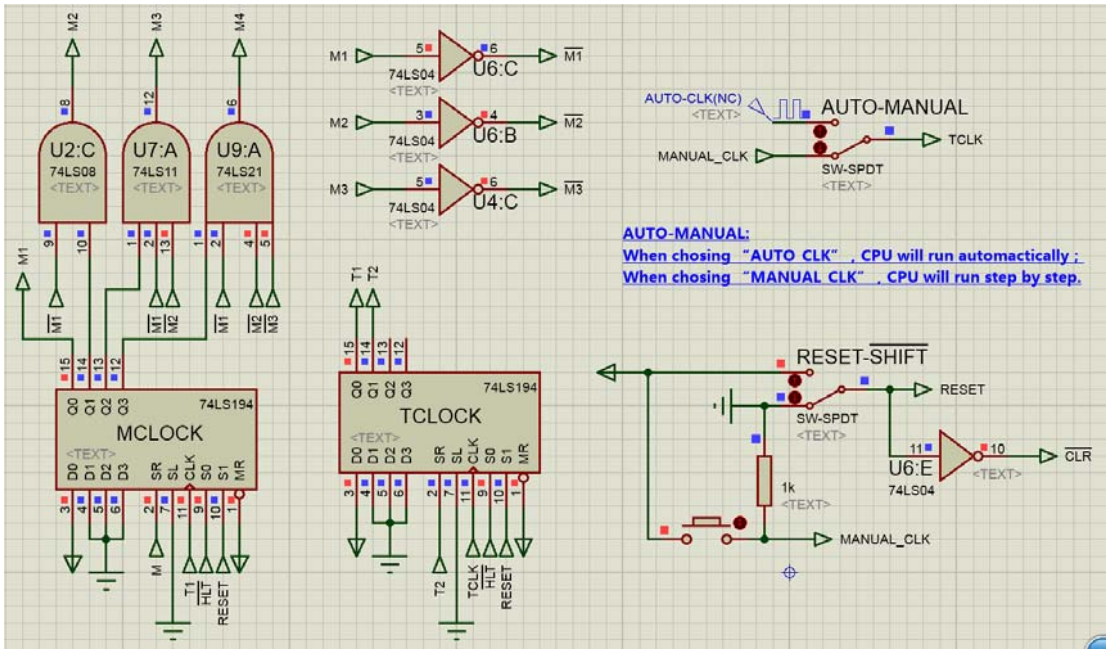


图 2-30 多周期硬布线控制器的时序电路

此外，上图 2-30 中的基准时钟电路、RESET 复位电路，以及用以初始化和跳出 HLT 指令“断点”的“重启”操作步骤都与单周期硬布线版本完全相同。下图 2-31 中的指令译码器 74LS138 电路和指令显示电路亦与单周期硬布线版本保持一致。指令信号 HLT\的出现同样把图 2-30 中的两级时序发生器 MCLK 和 TCLK 的工作模式改为保持，让系统停机在 M1 取指周期的 T2 节拍上。根据表 2-16 和上表 2-17，可以将数据通路的微操作信号的组合逻辑简化如下（电路如下图 2-31 中指令译码器 74LS138 右边所示）：

LDIR = M1

$\overline{\text{LDPC}}$ = M1

LDAR = $\overline{\text{OE}}$ = PC_INC = 1

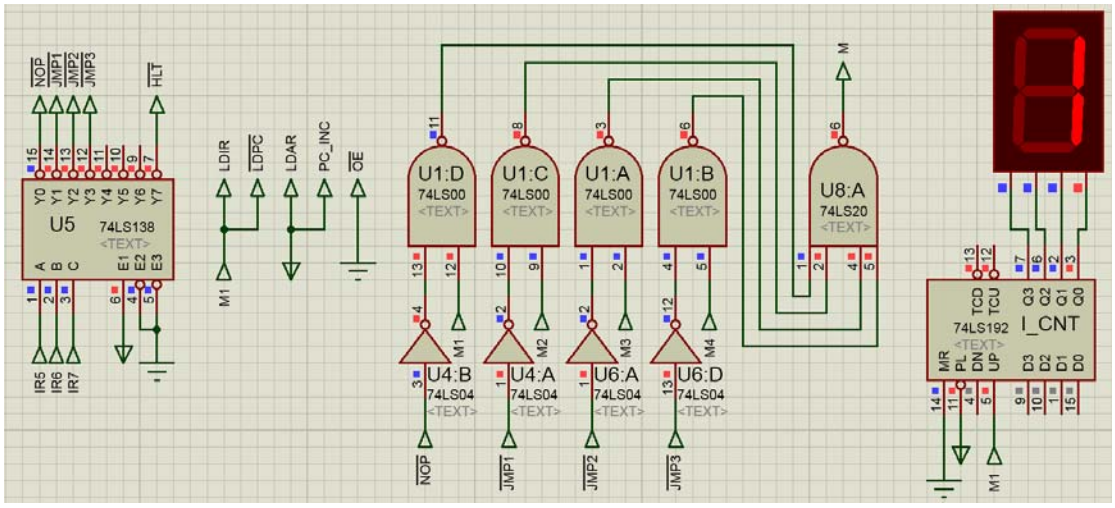


图 2-31 多周期硬布线控制器的微操作信号硬布线逻辑

五、实验步骤

●在图 2-25 和图 2-28 所示的硬布线控制器电路的存储器 PROGRAM 中，烧写进去如下所示的机器语言程序文本（烧写 ROM 方法参见 2.4 存储器实验）：

```
ORG    0000H
       DB      00100000B; JMP1, 06H
       DB      00000110B
       DB      11101010B; HLT
       DB      00001010B; NOP/[ADDR]

       DB      01100000B; JMP3, 0BH->03H->0AH
       DB      00001011B
       DB      00000010B; NOP/[ADDR]
       DB      11100001B ; HLT

       DB      01000000B; JMP2, 06H->02H
       DB      00000110B
       DB      11100000B; HLT
       DB      00000011B; NOP/[ADDR]
END
```

● 分别启动仿真图 2-25 所示的单周期硬布线控制器和图 2-28 所示的多周期硬布线控制器：首先 RESET=1，然后手动 CLK 一次，初始化 74LS194 节拍发生器｛T1,T2,T3,T4｝={1,0,0,0}，最后 RESET=0，恢复正常节拍时序。开始手动 CLK 单步执行上述实验 1 的机器语言程序。观察每一次单步执行的结果，例如寄存器 AR、IR、PC 及总线 BUS 信息。（注意程序进入 HLT 指令“断点”后需要重复上述初始化操作跳出“断点”）

● 依照上述启动初始化方法，在 RESET=0 恢复正常运行后，转为 AUTO_CLK 方波信号源自动 CLK 运行单周期和多周期硬布线控制器。当程序通路陷入“断点”时刻（HLT 指令），查看寄存器 AR、IR、PC、总线 BUS 信息及微指令周期数指示。

六、思考题

1、在上述机器语言文本中，有部分地址标示“NOP/[ADDR]”，为何相同位置会有不同的执行效果？在什么情况下执行到该处是 NOP 指令（顺序滑过）？什么情况下执行到该处是打入 PC 的地址（跳转）？

2、假设上述机器语言文本的倒数第二行不是 HLT 指令，而是修改为 NOP 指令。则系统启动仿真时会出现什么情况？系统会跑飞么？当系统运行到超过机器语言文本最后一行代码后会出现什么情况？会自动停机么？是什么原因导致自动停机？

3、假设只有一级时序发生器 MCLOCK，节拍脉冲 T 简化到只有一个 CLK，请问硬布线电路怎么改？数据通路需要修改么？机器语言程序需要改么？请修改电路试验。