

# 计算机组成原理与系统结构 实验教程



西安唐都科教仪器公司

Copyright Reserved 2008

## 版权声明

本书的版权归西安唐都科教仪器开发有限责任公司所有，保留一切权利。非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本书的部分或全部，并以任何形式传播。

西安唐都科教仪器开发有限责任公司，1999-2008(C)，All right reserved.

计算机组成原理与系统结构实验教程

©版权所有 非经许可 严禁复制

唐都公司网址: <http://www.tangdu.com/>

# 前 言

本书是为西安唐都科教仪器公司研发、生产的 TD - CMX 实验教学系统开展“计算机组成原理”课程的实验教学而配套的实验教程。

第一章到第三章为部件实验，研究组成计算机的每个部件的工作原理及设计方法；第四章为计算机系统总线设计实验，提供了具有基本输入输出功能的总线接口实验和具有中断、DMA 功能的总线接口设计实验。第五章为模型计算机设计实验，通过对几种不同复杂程度的模型计算机的设计，来研究计算机各部件是如何来配合工作的，并掌握设计一个计算机系统的方法。第六章为输入输出系统扩展实验，通过对模型机的扩展设计，使之拥有中断、DMA 等功能，并通过对定时计数器 8253 的扩展及编程，使学生熟悉并掌握典型接口芯片的扩展应用。

书中所含实验项目丰富，内容完备，各学校可以根据自己的教学计划和教学特点选取教学内容。例如，对于书中应用大规模可编程逻辑器件 FPGA 的章节，需要读者具有 FPGA 器件及其设计方法等方面的基础知识，也可以作为 EDA 在模型机设计应用中的参考教程；对于没有学习过“计算机接口技术”课程内容的，也可以通过第四章和第六章对于系统总线的介绍和外围接口芯片的扩展应用部分来学习。

由于编者水平有限，加上计算机技术飞速发展，新的理念和技术层出不穷，在教材中会存在一些问题和错误，恳请广大读者批评指正。

编 者

2008 年 5 月

# 目 录

<b>第 1 章 运算器</b>	<b>1</b>
1.1 基本运算器实验	1
1.2 超前进位加法器设计实验	6
1.3 阵列乘法器设计实验	12
<b>第 2 章 存储系统</b>	<b>14</b>
2.1 静态随机存储器实验	14
2.2 Cache控制器设计实验	18
<b>第 3 章 控制器</b>	<b>24</b>
3.1 时序发生器设计实验	24
3.2 微程序控制器实验	27
<b>第 4 章 系统总线与总线接口</b>	<b>36</b>
4.1 系统总线和具有基本输入输出功能的总线接口实验	36
4.2 具有中断控制功能的总线接口实验	41
4.3 具有DMA控制功能的总线接口实验	43
<b>第 5 章 模型计算机</b>	<b>45</b>
5.1 CPU与简单模型机设计实验	45
5.2 硬布线控制器模型机设计实验	52
5.3 复杂模型机设计实验	56
<b>第 6 章 输入输出系统</b>	<b>70</b>
6.1 带中断处理能力的模型机设计实验	70
6.2 带DMA控制功能的模型机设计实验	84
6.3 典型I/O接口 8253 扩展设计实验	90

## 第 1 章 运算器

计算机的一个最主要的功能就是处理各种算术和逻辑运算，这个功能要由 CPU 中的运算器来完成，运算器也称作算术逻辑部件 ALU。本章首先安排一个基本的运算器实验，了解运算器的基本结构，然后再设计一个加法器和一个乘法器。

### 1.1 基本运算器实验

#### 1.1.1 实验目的

- (1) 了解运算器的组成结构。
- (2) 掌握运算器的工作原理。

#### 1.1.2 实验设备

PC 机一台，TD-CMX 实验系统一套。

#### 1.1.3 实验原理

本实验的原理如图 1-1-1 所示。

运算器内部含有三个独立运算部件，分别为算术、逻辑和移位运算部件，要处理的数据存于暂存器 A 和暂存器 B，三个部件同时接受来自 A 和 B 的数据（有些处理器体系结构把移位运算器放于算术和逻辑运算部件之前，如 ARM），各部件对操作数进行何种运算由控制信号  $S_3 \cdots S_0$  和 CN 来决定，任何时候，多路选择开关只选择三部件中一个部件的结果作为 ALU 的输出。如果是影响进位的运算，还将置进位标志 FC，在运算结果输出前，置 ALU 零标志。ALU 中所有模块集成在一片 FPGA 中。

逻辑运算部件由逻辑门构成，较为简单，而后面又有专门的算术运算部件设计实验，在此对这两个部件不再赘述。移位运算采用的是桶形移位器，一般采用交叉开关矩阵来实现，交叉开关的原理如图 1-1-2 所示。图中显示的是一个 4X4 的矩阵（系统中是一个 8X8 的矩阵）。每一个输入都通过开关与一个输出相连，把沿对角线的开关导通，就可实现移位功能，即：

(1) 对于逻辑左移或逻辑右移功能，将一条对角线的开关导通，这所有的输入位与所使用的输出分别相连，而没有同任何输入相连的则输出连接 0。

(2) 对于循环右移功能，右移对角线同互补的左移对角线一起激活。例如，在 4 位矩阵中使用‘右 1’和‘左 3’对角线来实现右循环 1 位。

(3) 对于未连接的输出位，移位时使用符号扩展或是 0 填充，具体由相应的指令控制。使用另外的逻辑进行移位总量译码和符号判别。

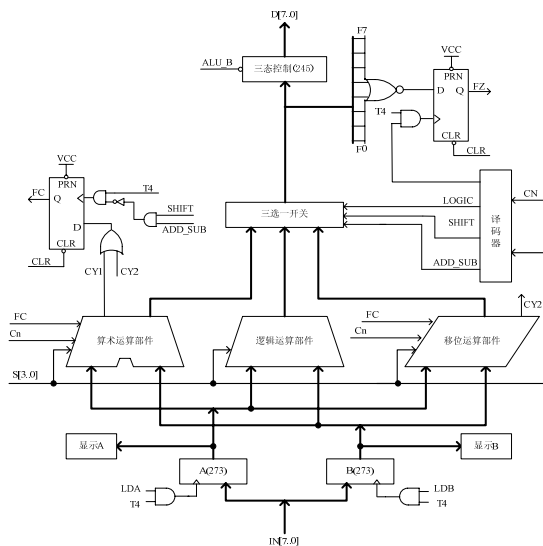


图 1-1-1 运算器原理图

运算器部件由一片 FPGA 实现。ALU 的输入和输出通过三态门 74LS245 连到 CPU 内总线上，另外还有指示灯标明进位标志 FC 和零标志 FZ。请注意：实验箱上凡丝印标注有马蹄形标记 ‘ $\sqcap$ ’，表示这两根排针之间是连通的。图中除 T4 和 CLR，其余信号均来自于 ALU 单元的排线座，实验箱中所有单元的 T1、T2、T3、T4 都连接至控制总线单元的 T1、T2、T3、T4，CLR 都连接至 CON 单元的 CLR 按钮。T4 由时序单元的 TS4 提供（时序单元的介绍见附录二），其余控制信号均由 CON 单元的二进制数据开关模拟给出。控制信号中除 T4 为脉冲信号外，其余均为电平信号，其中 ALU\_B 为低有效，其余为高有效。

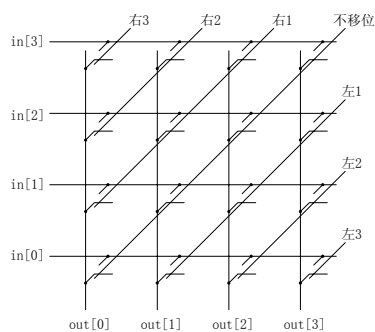


图 1-1-2 交叉开关桶形移位器原理图

暂存器 A 和暂存器 B 的数据能在 LED 灯上实时显示，原理如图 1-1-3 所示（以 A0 为例，其它相同）。进位标志 FC、零标志 FZ 和数据总线 D7...D0 的显示原理也是如此。

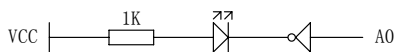


图 1-1-3 A0 显示原理图

ALU 和外围电路的连接如图 1-1-4 所示，图中的小方框代表排针座。

运算器的逻辑功能表如表 1-1-1 所示，其中 S3 S2 S1 S0 CN 为控制信号，FC 为进位标志，FZ 为运算器零标志，表中功能栏内的 FC、FZ 表示当前运算会影响到该标志。

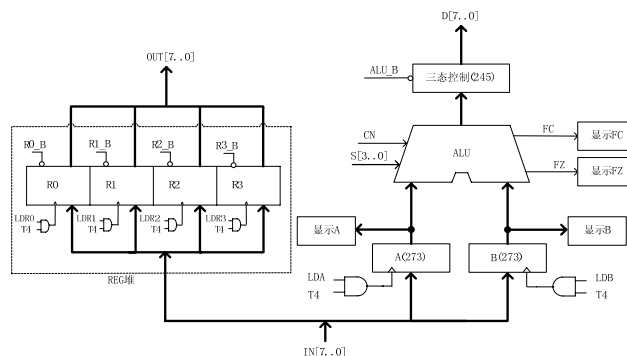


图 1-1-4 ALU 和外围电路连接原理图

表 1-1-1 运算器逻辑功能表

运算类型	S3 S2 S1 S0	CN	功 能
逻辑运算	0000	X	F=A（直通）
	0001	X	F=B（直通）
	0010	X	F=AB (FZ)
	0011	X	F=A+B (FZ)
	0100	X	F=/A (FZ)
移位运算	0101	X	F=A 不带进位循环右移 B（取低 3 位）位 (FZ)
	0110	0	F=A 逻辑右移一位 (FZ)
		1	F=A 带进位循环右移一位 (FC, FZ)
	0111	0	F=A 逻辑左移一位 (FZ)
		1	F=A 带进位循环左移一位 (FC, FZ)
算术运算	1000	X	置 FC=CN (FC)
	1001	X	F=A 加 B (FC, FZ)
	1010	X	F=A 加 B 加 FC (FC, FZ)
	1011	X	F=A 减 B (FC, FZ)
	1100	X	F=A 减 1 (FC, FZ)
	1101	X	F=A 加 1 (FC, FZ)
	1110	X	(保留)
	1111	X	(保留)

\*表中“X”为任意态，下同

### 1.1.4 实验步骤

(1) 按图 1-1-5 连接实验电路，并检查无误。图中将用户需要连接的信号用圆圈标明（其它

实验相同)。

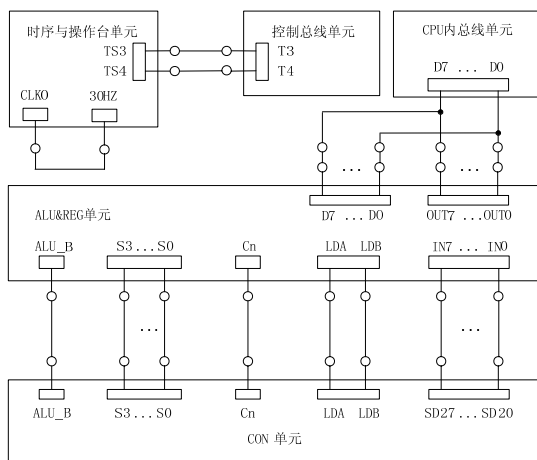


图 1-1-5 实验接线图

(2) 将时序与操作台单元的开关 KK2 置为‘单拍’档,开关 KK1、KK3 置为‘运行’档。(3) 打开电源开关,如果听到有‘嘀’报警声,说明有总线竞争现象,应立即关闭电源,重新检查接线,直到错误排除。然后按动 CON 单元的 CLR 按钮,将运算器的 A、B 和 FC、FZ 清零。

(4) 用输入开关向暂寄存器 A 置数。

① 拨动 CON 单元的 SD27...SD20 数据开关,形成二进制数 01100101 (或其它数值),数据显示亮为‘1’,灭为‘0’。

② 置 LDA=1, LDB=0,连续按动时序单元的 ST 按钮,产生一个 T4 上沿,则将二进制数 01100101 置入暂寄存器 A 中,暂寄存器 A 的值通过 ALU 单元的 A7...A0 八位 LED 灯显示。

(5) 用输入开关向暂寄存器 B 置数。

① 拨动 CON 单元的 SD27...SD20 数据开关,形成二进制数 10100111 (或其它数值)。

② 置 LDA=0, LDB=1,连续按动时序单元的 ST 按钮,产生一个 T4 上沿,则将二进制数 10100111

置入暂寄存器 B 中,暂寄存器 B 的值通过 ALU 单元的 B7...B0 八位 LED 灯显示。

(6) 改变运算器的功能设置,观察运算器的输出。置 ALU\_B=0、LDA=0、LDB=0,然后按表 1-1-1 置 S3、S2、S1、S0 和 Cn 的数值,并观察数据总线 LED 显示灯显示的结果。如置 S3、S2、S1、S0 为 0010,运算器作逻辑与运算,置 S3、S2、S1、S0 为 1001,运算器作加法运算。

如果实验箱和 PC 联机操作,则可通过软件中的数据通路图来观测实验结果(软件使用说明请看附录一),方法是:打开软件,选择联机软件的“【实验】—【运算器实验】”,打开运算器实验的数据通路图,如图 1-1-6 所示。进行上面的手动操作,每按动一次 ST 按钮,数据通路图会有数据的流动,反映当前运算器所做的操作,或在软件中选择“【调试】—【单节拍】”,其作用相当于将时序单元的状态开关 KK2 置为‘单拍’档后按动了一次 ST 按钮,数据通路图也会反映当前运算器所做的操作。

重复上述操作,并完成表 1-1-2。然后改变 A、B 的值,验证 FC、FZ 的锁存功能。



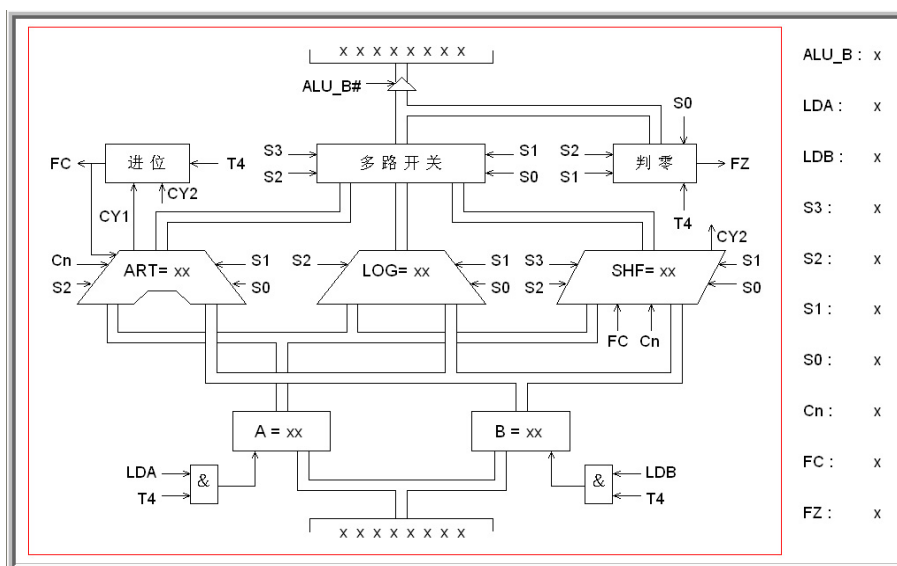


图 1-1-6 数据通路图

表 1-1-2 运算结果表

运算类型	A	B	S3 S2 S1 S0	CN	结果
逻辑运算	65	A7	0 0 0 0	X	F=( 65 ) FC=( ) FZ=( )
	65	A7	0 0 0 1	X	F=( A7 ) FC=( ) FZ=( )
			0 0 1 0	X	F=( ) FC=( ) FZ=( )
			0 0 1 1	X	F=( ) FC=( ) FZ=( )
			0 1 0 0	X	F=( ) FC=( ) FZ=( )
移位运算			0 1 0 1	X	F=( ) FC=( ) FZ=( )
			0 1 1 0	0	F=( ) FC=( ) FZ=( )
				1	F=( ) FC=( ) FZ=( )
			0 1 1 1	0	F=( ) FC=( ) FZ=( )
				1	F=( ) FC=( ) FZ=( )
算术运算			1 0 0 0	X	F=( ) FC=( ) FZ=( )
			1 0 0 1	X	F=( ) FC=( ) FZ=( )
			1 0 1 0 (FC=0)	X	F=( ) FC=( ) FZ=( )
			1 0 1 0 (FC=1)	X	F=( ) FC=( ) FZ=( )
			1 0 1 1	X	F=( ) FC=( ) FZ=( )
			1 1 0 0	X	F=( ) FC=( ) FZ=( )
			1 1 0 1	X	F=( ) FC=( ) FZ=( )

## 1.2 超前进位加法器设计实验

### 1.2.1 实验目的

- (1) 掌握超前进位加法器的原理及其设计方法。
- (2) 熟悉 FPGA 应用设计及 EDA 软件的使用。

### 1.2.2 实验设备

PC 机一台，TD-CMX 实验系统一套。

### 1.2.3 实验原理

加法器是执行二进制加法运算的逻辑部件，也是 CPU 运算器的基本逻辑部件（减法可以通过补码相加来实现）。加法器又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，而全加器是在半加器的基础上又考虑了低位过来的进位信号。

表 1-2-1 1 位全加器真值表

输 入			输 出	
A	B	Ci	S	Co
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

A、B 为 2 个 1 位的加数，Ci 为来自低位的进位，S 为和，Co 为向高位的进位，根据表 1-2-1 所示的真值表，可得到全加器的逻辑表达式为：

$$S = \overline{A}\overline{B}C_i + \overline{A}B\overline{C_i} + A\overline{B}\overline{C_i} + ABC_i$$

$$Co = AB + AC_i + BC_i$$

根据逻辑表达式，可得到如图 1-2-1 所示的逻辑电路图。

有了 1 位全加器，就可以用它来构造多位加法器，加法器根据电路结构的不同，可以分为串行加法器和并行加法器两种。串行加法器低位全加器产生的进位要依次串行地向高位进位，其电路简单，占用资源较少，但是串行加法器每位和以及向高位的进位的产生都依赖于低位的进位，导致完成加法运算的延迟时间较长，效率并不高。

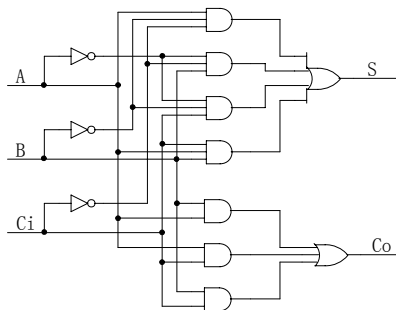


图 1-2-1 1 位全加器 (FA) 逻辑电路图

串行加法器运算速度慢，其根本原因是每一位的结果都要依赖于低位的进位，因而可以通过并行进位的方式来提高效率。只要能设计出专门的电路，使得每一位的进位能够并行地产生而与低位的运算情况无关，就能解决这个问题。可以对加法器进位的逻辑表达式做进一步的推导：

$$C_0 = 0$$

$$C_{i+1} = A_i B_i + A_i C_i + B_i C_i = A_i B_i + (A_i + B_i) C_i$$

设

$$g_i = A_i B_i$$

$$p_i = A_i + B_i$$

则有：

$$\begin{aligned} C_{i+1} &= g_i + p_i C_i \\ &= g_i + p_i (g_{i-1} + p_{i-1} C_{i-1}) \\ &= g_i + p_i (g_{i-1} + p_{i-1} (g_{i-2} + p_{i-2} C_{i-2})) \\ &\quad \dots \\ &= g_i + p_i (g_{i-1} + p_{i-1} (g_{i-2} + p_{i-2} (\dots (g_0 + p_0 C_0) \dots))) \\ &= g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \dots + p_i p_{i-1} \dots p_1 g_0 + p_i p_{i-1} \dots p_1 p_0 C_0 \end{aligned}$$

由于  $g_i$ 、 $p_i$  只和  $A_i$ 、 $B_i$  有关，这样  $C_{i+1}$  就只和  $A_i$ 、 $A_{i-1}$ 、 $\dots$ 、 $A_0$ ， $B_i$ 、 $B_{i-1}$ 、 $\dots$ 、 $B_0$  及  $C_0$  有关。所以各位的进位  $C_i$ 、 $C_{i-1}$ 、 $\dots$ 、 $C_1$  就可以并行地产生，这种进位就叫超前进位。

根据上面的推导，随着加法器位数的增加，越是高位的进位逻辑电路就会越复杂，逻辑器件使用也就越多。事实上我们可以继续推导进位的逻辑表达式，使得某些基本逻辑单元能够复用，且能照顾到进位位的并行产生。

定义

$$G_{i,j} = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \dots + p_i p_{i-1} \dots p_{j+1} g_j$$

$$P_{i,j} = p_i p_{i-1} \dots p_{j+1} p_j$$

则有

$$G_{i,i} = g_i$$

$$P_{i,i} = p_i$$

$$G_{i,j} = G_{i,k} + P_{i,k}G_{k-1,j}$$

$$P_{i,j} = P_{i,k}P_{k-1,j}$$

$$C_{i+1} = G_{i,j} + P_{i,j}C_j$$

从而可以得到表 1-2-2 所示的算法，该算法为超前进位算法的扩展算法，这里实现的是一个 8 位加法器的算法。

表 1-2-2 超前进位扩展算法

$G_{1,0} = g_1 + p_1g_0$	$G_{3,0} = G_{3,2} + P_{3,2}G_{1,0}$	$G_{7,0} = G_{7,4} + P_{7,4}G_{3,0}$ $P_{7,0} = P_{7,4}P_{3,0}$
$P_{1,0} = p_1p_0$		
$G_{3,2} = g_3 + p_3g_2$	$P_{3,0} = P_{3,2}P_{1,0}$	
$P_{3,2} = p_3p_2$		
$G_{5,4} = g_5 + p_5g_4$	$G_{7,4} = G_{7,6} + P_{7,6}G_{5,4}$ $P_{7,4} = P_{7,6}P_{5,4}$	
$P_{5,4} = p_5p_4$		
$G_{7,6} = g_7 + p_7g_6$		
$P_{7,6} = p_7p_6$		
$C_8 = G_{7,0} + P_{7,0}C_0$		

从上表可以看出，本算法的核心思想是把 8 位加法器分成两个 4 位加法器，先求出低 4 位加法器的各个进位，特别是向高 4 位加法器的进位  $C_4$ 。然后，高 4 位加法器把  $C_4$  作为初始进位，使用低 4 位加法器相同的方法来完成计算。每一个 4 位加法器在计算时，又分成了两个 2 位的加法器。如此递归，如图 1-2-2 所示。

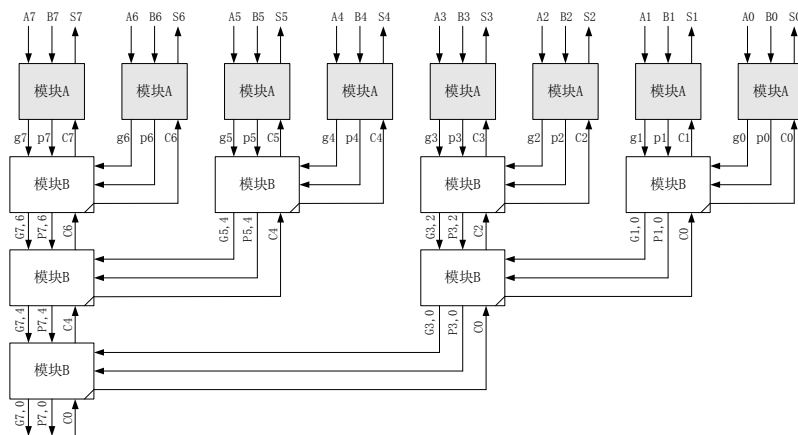


图 1-2-2 超前进位扩展算法示意图

这样，在超前进位扩展算法的逻辑电路实现中，需要设计两种电路。模块 A 逻辑电路需要完成如下计算逻辑，其原理图如图 1-2-3 所示。

$$G_{i,i} = A_iB_i$$

$$P_{i,i} = A_i + B_i$$

$$S_i = \overline{A_iB_i}C_i + \overline{A_i}B_iC_i + A_i\overline{B_i}C_i + A_iB_i\overline{C_i}$$

模块 B 逻辑电路需要完成如下计算逻辑，其原理图如图 1-2-4 所示。

$$G_{i,j} = G_{i,k} + P_{i,k}G_{k-1,j}$$

$$P_{i,j} = P_{i,k}P_{k-1,j}$$

$$C_{i+1} = G_{i,j} + P_{i,j}C_j$$

按图 1-2-2 将这两种电路连接起来，就可以得到一个 8 位的超前进位的加法器。

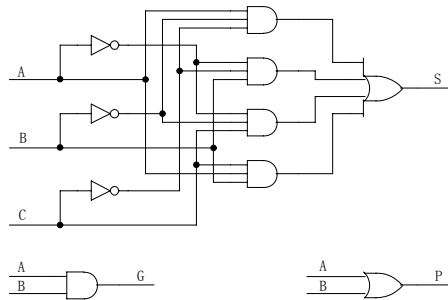


图 1-2-3 模块 A 原理图

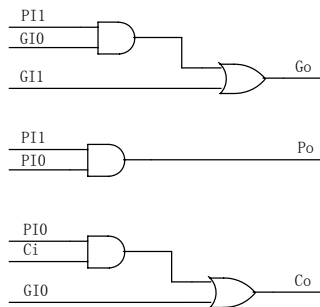


图 1-2-4 模块 B 原理图

从图中可以看到  $G_{i,i}$  和  $P_{i,i}$  既参与了每位上进位的计算，又参与了下一级  $G_{i,i}$  和  $P_{i,i}$  的计算。这样就复用了这些电路，使得需要的总逻辑电路数大大减少。超前进位加法器的运算速度较快，但是，与串行进位加法器相比，逻辑电路比较复杂，使用的逻辑器件较多，这些是为提高运算速度付出的代价。

本实验在 FPGA 单元上进行，FPGA 单元由由两大部分组成，一是 LED 显示灯，两组 16 只，供调试时观测数据，LED 灯为正逻辑，1 时亮，0 时灭。另外是一片 Cyclone II EP2C5T144C8 及其外围电路。

Cyclone II EP2C5T144C8 有 144 个引脚，分成四个块，即 BANK1...BANK4，将每个块的通用 I/O 脚加以编号，就形成 A01...A13、B01...B22 等 I/O 号，如图 1-2-5 所示。



### 1.2.4 实验步骤

	B01	C16...C09
	Co	S7...S0
	EP2C5T144C8	
Ci	A7...A0	B7...B0
C17	A08...A01	D07...D01

图 1-2-7 引脚分配图

(2) 关闭实验系统电源，按图 1-2-8 连接实验电路，图中将用户需要连接的信号用圆圈标明。

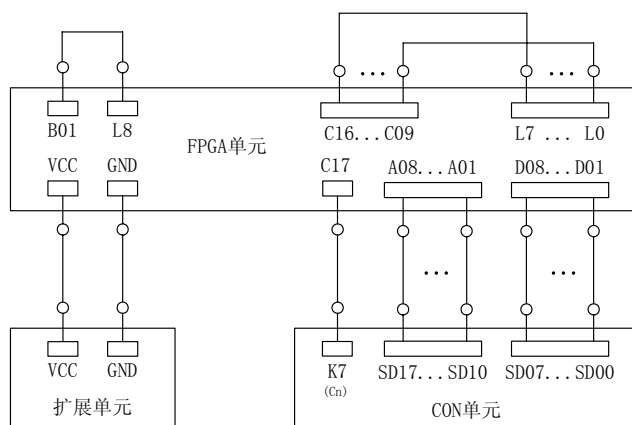


图 1-2-8 实验接线图

(3) 打开实验系统电源，将下载电缆插入 FPGA 单元的 JTAG 口，把生成的 SOF 文件下载到 FPGA 单元中去。

(4) 以 CON 单元中的 SD17...SD10 八个二进制开关为被加数 A，SD07...SD00 八个二进制开关为加数 B，K7 用来模拟来自低位的进位信号，相加的结果在 FPGA 单元的 L7...L0 八个 LED 灯显示，相加后向高位的进位用 FPGA 单元的 L8 灯显示。给 A 和 B 置不同的数，观察相加的结果。

## 1.3 阵列乘法器设计实验

### 1.3.1 实验目的

- (1) 掌握乘法器的原理及其设计方法。
- (2) 熟悉 FPGA 应用设计及 EDA 软件的使用。

### 1.3.2 实验设备

PC 机一台，TD-CMX 实验系统一套。

### 1.3.3 实验原理

硬件乘法器常规的设计是采用“串行移位”和“并行加法”相结合的方法，这种方法并不需要很多的器件，然而“加法-移位”的方法毕竟太慢。随着大规模集成电路的发展，采用高速的单元阵列乘法器，无论从计算机的计算速度，还是从提高计算效率，都是十分必要的。阵列乘法器分带符号和不带符号的阵列乘法器，本节只讨论不带符号阵列乘法。高速组合阵列乘法器，采用标准加法单元构成乘法器，即利用多个一位全加器（FA）实现乘法运算。

对于一个 4 位二进制数相乘，有如下算式：

$$\begin{array}{r}
 \times \qquad \qquad \qquad \begin{array}{cccc} A3 & A2 & A1 & A0 \\ B3 & B2 & B1 & B0 \end{array} \\
 \hline
 \qquad \qquad \qquad A3B0 & A2B0 & A1B0 & A0B0 \\
 \qquad \qquad A3B1 & A2B1 & A1B1 & A0B1 \\
 \qquad A3B2 & A2B2 & A1B2 & A0B2 \\
 + \quad A3B3 & A2B3 & A1B3 & A0B3 \\
 \hline
 P7 & P6 & P5 & P4 & P3 & P2 & P1 & P0
 \end{array}$$

这个  $4 \times 4$  阵列乘法器的原理如图 1-3-1 所示。

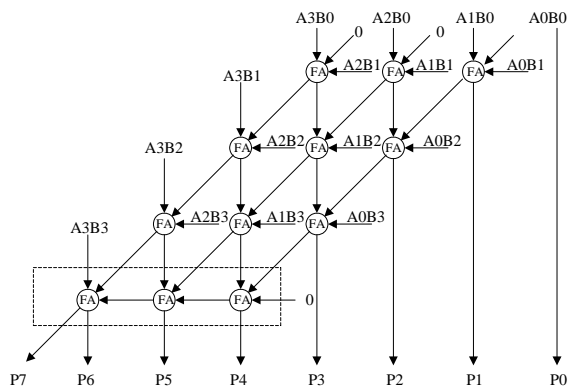


图 1-3-1  $4 \times 4$  阵列乘法器原理图

FA（全加器）的斜线方向为进位输出，竖线方向为和输出。图中阵列的最后一行构成了一个串行进位加法器。由于 FA 一级是无需考虑进位的，它的进位被暂时保留下来不往前传递，因此同一极中任意一位 FA 加法器的进位输出与和输出几乎是同时形成的，与“串行移位”相比可大大减少同级间的进位传递延迟，所以送往最后一行串行加法器的输入延迟仅与 FA 的级数（行数）有关，即与乘数位数有关。本实验用 FPGA 来设计一个  $4 \times 4$  位加法器，且全部采用原理图



方式实现。

### 1.3.4 实验步骤

(1) 根据上述阵列乘法器的原理,使用 Quartus II 软件编辑相应的电路原理图并进行编译,其在 EP2C5T144C8 芯片中对应的引脚如图 1-3-2 所示,框外文字表示 I/O 号,框内文字表示该引脚的含义(本实验例程见‘安装路径\FPGA \Multiply\Multiply.qpf’工程)。

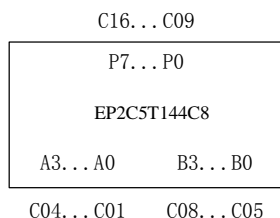


图 1-3-2 引脚分配图

(2) 关闭实验系统电源,按图 1-3-3 连接实验电路,图中将用户需要连接的信号用圆圈标明。

(3) 打开实验系统电源,将下载电缆插入 FPGA 单元的 JTAG 口,把生成的 SOF 文件下载到 FPGA 单元中去,FPGA 单元介绍见实验 1.2。

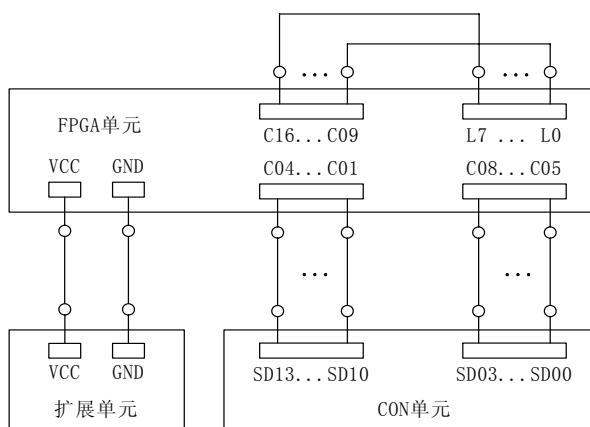


图 1-3-3 阵列乘法器实验接线图

(4) 以 CON 单元中的 SD10...SD13 四个二进制开关为乘数 A, SD14...SD17 四个二进制开关为被乘数 B,而相乘的结果在 FPGA 单元的 L7...L0 八个 LED 灯显示。给 A 和 B 置不同的数,观察相乘的结果。

## 第2章 存储系统

存储器是计算机各种信息存储与交换的中心。在程序执行过程中，所要执行的指令是从存储器中获取，运算器所需要的操作数是通程序中的访问存储器指令从存储器中得到，运算结果在程序执行完之前又必须全部写到存储器中，各种输入输出设备也直接与存储器交换数据。把程序和数据存储在存储器中，是冯·诺依曼型计算机的基本特征，也是计算机能够自动、连续快速工作的基础。

本章安排了两个实验：静态随机存储器实验及 Cache 控制器设计实验。

### 2.1 静态随机存储器实验

#### 2.1.1 实验目的

掌握静态随机存储器 RAM 工作特性及数据的读写方法。

#### 2.1.2 实验设备

PC 机一台，TD-CMX 实验系统一套。

#### 2.1.3 实验原理

实验所用的静态存储器由一片 6116 (2K×8bit) 构成 (位于 MEM 单元)，如图 2-1-1 所示。6116 有三个控制线：CS (片选线)、OE (读线)、WE (写线)，其功能如表 2-1-1 所示，当片选有效 (CS=0) 时，OE=0 时进行读操作，WE=0 时进行写操作，本实验将 CS 常接地。

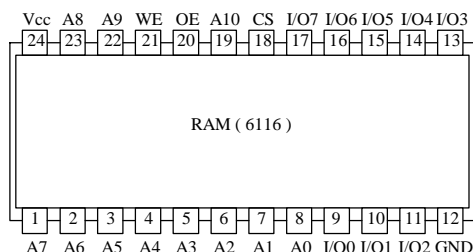


图 2-1-1 SRAM 6116 引脚图

由于存储器 (MEM) 最终是要挂接到 CPU 上，所以其还需要一个读写控制逻辑，使得 CPU 能控制 MEM 的读写，实验中的读写控制逻辑如图 2-1-2 所示，由于 T3 的参与，可以保证 MEM 的写脉宽与 T3 一致，T3 由时序单元的 TS3 给出 (时序单元的介绍见附录 2)。IOM 用来选择是对 I/O 还是对 MEM 进行读写操作，RD=1 时为读，WR=1 时为写。

表 2-1-1 SRAM 6116 功能表

$\overline{CS}$	$\overline{WE}$	$\overline{OE}$	功能
1	×	×	不选择
0	1	0	读
0	0	1	写
0	0	0	写

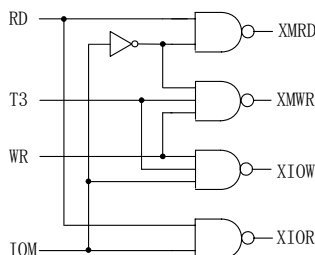


图 2-1-2 读写控制逻辑

实验原理图如图 2-1-3 所示，存储器数据线接至数据总线，数据总线上接有 8 个 LED 灯显示 D7…D0 的内容。地址线接至地址总线，地址总线上接有 8 个 LED 灯显示 A7…A0 的内容，地址由地址锁存器（74LS273，位于 PC&AR 单元）给出。数据开关（位于 IN 单元）经一个三态门（74LS245）连至数据总线，分时给出地址和数据。地址寄存器为 8 位，接入 6116 的地址 A7…A0，6116 的高三位地址 A10…A8 接地，所以其实际容量为 256 字节。

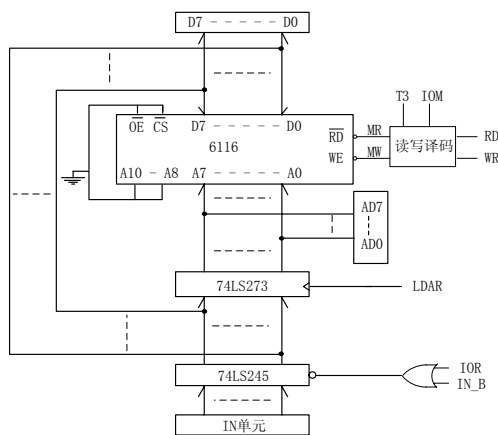


图 2-1-3 存储器实验原理图

实验箱中所有单元的时序都连接至时序与操作台单元，CLR 都连接至 CON 单元的 CLR 按钮。实验时 T3 由时序单元给出，其余信号由 CON 单元的二进制开关模拟给出，其中 IOM 应为低（即 MEM 操作），RD、WR 高有效，MR 和 MW 低有效，LDAR 高有效。

#### 2.1.4 实验步骤

(1) 关闭实验系统电源，按图 2-1-4 连接实验电路，并检查无误，图中将用户需要连接的信号用圆圈标明。

(2) 将时序与操作台单元的开关 KK1、KK3 置为运行档、开关 KK2 置为‘单步’档（时序单元的介绍见附录二）。

(3) 将 CON 单元的 IOR 开关置为 1（使 IN 单元无输出），打开电源开关，如果听到有‘嘀’报警声，说明有总线竞争现象，应立即关闭电源，重新检查接线，直到错误排除。

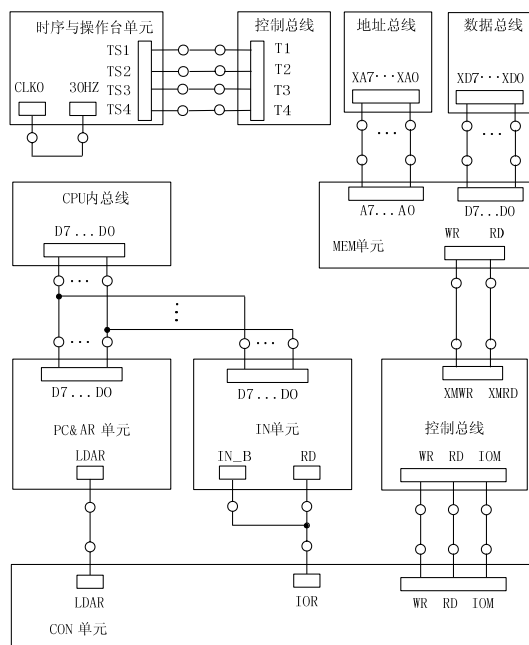


图 2-1-4 实验接线图

(4) 给存储器的 00H、01H、02H、03H、04H 地址单元中分别写入数据 11H、12H、13H、14H、15H。由前面的存储器实验原理图（图 2-1-3）可以看出，由于数据和地址由同一个数据开关给出，因此数据和地址要分时写入，先写地址，具体操作步骤为：先关掉存储器的读写（WR=0，RD=0），数据开关输出地址（IOR=0），然后打开地址寄存器门控信号（LDAR=1），按动 ST 产生 T3 脉冲，即将地址打入到 AR 中。再写数据，具体操作步骤为：先关掉存储器的读写（WR=0，RD=0）和地址寄存器门控信号（LDAR=0），数据开关输出要写入的数据，打开输入三态门（IOR=0），然后使存储器处于写状态（WR=1，RD=0，IOM=0），按动 ST 产生 T3 脉冲，即将数据打入到存储器中。写存储器的流程如图 2-1-5 所示（以向 00 地址单元写入 11H 为例）：

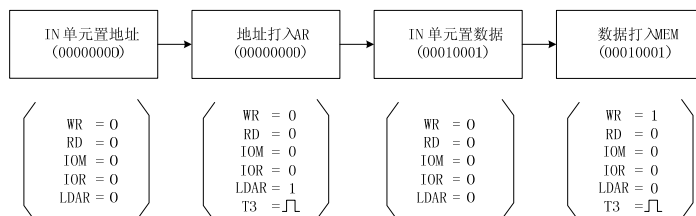


图 2-1-5 写存储器流程图

(5) 依次读出第 00、01、02、03、04 号单元中的内容，观察上述各单元中的内容是否与前面写入的一致。同写操作类似，也要先给出地址，然后进行读，地址的给出和前面一样，而在进行读操作时，应先关闭 IN 单元的输出（IOR=1），然后使存储器处于读状态（WR=0，RD=1，IOM=0），

此时数据总线上的数即为从存储器当前地址中读出的数据内容。读存储器的流程如图 2-1-6 所示（以从 00 地址单元读出 11H 为例）：

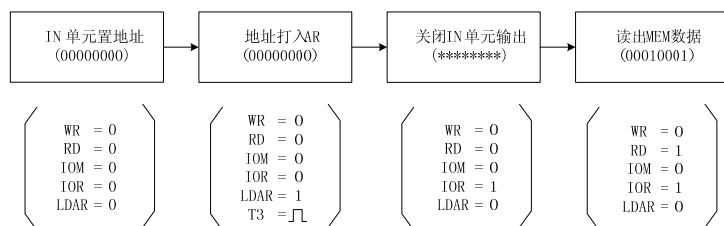


图 2-1-6 读存储器流程图

如果实验箱和 PC 联机操作，则可通过软件中的数据通路图来观测实验结果（软件使用说明请看附录 1），方法是：打开软件，选择联机软件的“【实验】—【存储器实验】”，打开存储器实验的数据通路图，如图 2-1-7 所示。

进行上面的手动操作，每按动一次 ST 按钮，数据通路图会有数据的流动，反映当前存储器所做的操作（即使是对存储器进行读，也应按动一次 ST 按钮，数据通路图才会有数据流动），或在软件中选择“【调试】—【单周期】”，其作用相当于将时序单元的状态开关置为‘单步’档后按动了一次 ST 按钮，数据通路图也会反映当前存储器所做的操作，借助于数据通路图，仔细分析 SRAM 的读写过程。

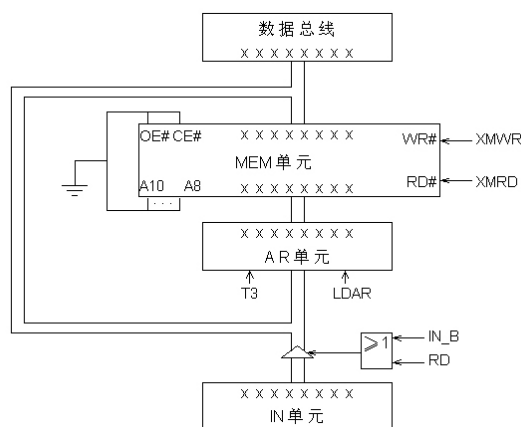


图 2-1-7 数据通路图

## 2.2 Cache 控制器设计实验

### 2.2.1 实验目的

- (1) 掌握 Cache 控制器的原理及其设计方法。
- (2) 熟悉 FPGA 应用设计及 EDA 软件的使用。

### 2.2.2 实验设备

PC 机一台，TD-CMX 实验系统一套。

### 2.2.3 实验原理

本实验采用的地址变换是直接映象方式，这种变换方式简单而直接，硬件实现很简单，访问速度也比较快，但是块的冲突率比较高。其主要原则是：主存中一块只能映象到 Cache 的一个特定的块中。

假设主存的块号为  $B$ ，Cache 的块号为  $b$ ，则它们之间的映象关系可以表示为：

$$b = B \bmod C_b$$

其中， $C_b$  是 Cache 的块容量。设主存的块容量为  $M_b$ ，区容量为  $M_e$ ，则直接映象方法的关系如图 2-2-1 所示。把主存按 Cache 的大小分成区，一般主存容量为 Cache 容量的整数倍，主存每一个分区内的块数与 Cache 的总块数相等。直接映象方式只能把主存各个区中相对块号相同的那些块映象到 Cache 中同一块号的那个特定块中。例如，主存的块 0 只能映象到 Cache 的块 0 中，主存的块 1 只能映象到 Cache 的块 1 中，同样，主存区 1 中的块  $C_b$ （在区 1 中的相对块号是 0），

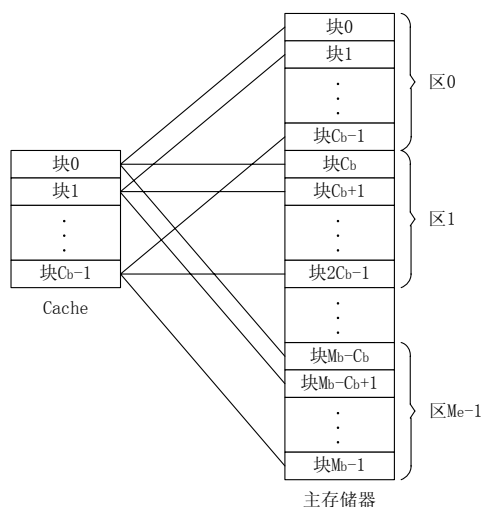


图 2-2-1 直接相联映象方式

也只能映象到 Cache 的块 0 中。根据上面给出的地址映象规则，整个 Cache 地址与主存地址的低位部分是完全相同的。

直接映象方式的地址变换过程如图 2-2-2 所示，主存地址中的块号  $B$  与 Cache 地址中的块号  $b$  是完全相同的。同样，主存地址中的块内地址  $W$  与 Cache 地址中的块内地址  $w$  也是完全相同的，主存地址比 Cache 地址长出来的部分称为区号  $E$ 。

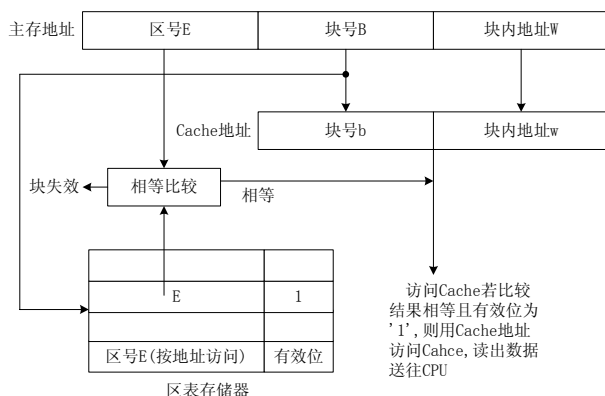


图 2-2-2 直接相联地址变换

在程序执行过程中，当要访问 Cache 时，为了实现主存块号到 Cache 块号的变换，需要有一个存放主存区号的小容量存储器，这个存储器的容量与 Cache 的块数相等，字长为主存地址中区号 E 的长度，另外再加一个有效位。

在主存地址到 Cache 地址的变换过程中，首先用主存地址中的块号去访问区号存储器（按地址访问）。把读出来的区号与主存地址中的区号 E 进行比较，根据比较结果和与区号在同一存储字中的有效位情况作出处理。如果区号比较结果相等，有效位为 '1'，则 Cache 命中，表示要访问的那一块已经装入到 Cache 中了，这时 Cache 地址（与主存地址的低位部分完全相同）是正确的。用这个 Cache 地址去访问 Cache，把读出来的数据送往 CPU。其他情况均为 Cache 没有命中，或称为 Cache 失效，表示要访问的那个块还没有装入到 Cache 中，这时，要用主存地址去访问主存储器，先把该地址所在的块读到 Cache 中，然后 CPU 从 Cache 中读取该地址中的数据。

本实验要在 FPGA 中实现 Cache 及其地址变换逻辑（也叫 Cache 控制器），采用直接相联地址变换，只考虑 CPU 从 Cache 读数据，不考虑 CPU 从主存中读数据和写回数据的情况，Cache 和 CPU 以及存储器的关系如图 2-2-3 所示。

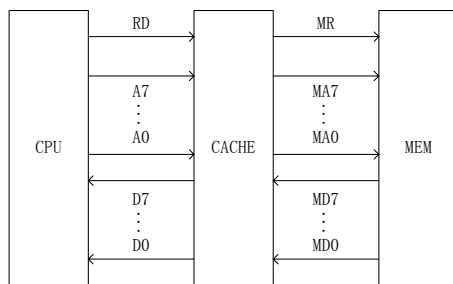


图 2-2-3 Cache 系统图

Cache 控制器顶层模块如图 2-2-4 所示，主存地址为 A7...A0，共 8 位，区号 E 取 3 位，这样 Cache 地址还剩 5 位，所以 Cache 容量为 32 个单元，块号 B 取 3 位，那么 Cache 分为 8 块，块内地址 W 取 2 位，则每块为 4 个单元。图 2-2-4 中，WCT 为写 Cache 块表信号，CLR 为系统总清零信号，A7...A0 为 CPU 访问内存的地址，M 为 Cache 失效信号，CA4...CA0 为 Cache 地址，MD7...MD0 为主存送 Cache 的数据，D7...D0 为 Cache 送 CPU 数据，T2 为系统时钟，RD 为 CPU 访

问内存读信号，LA1 和 LA0 为块内地址。

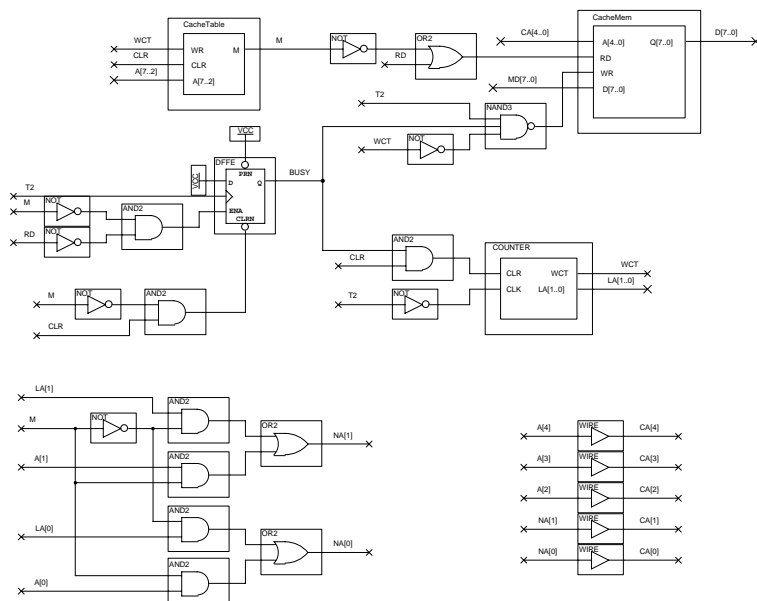


图 2-2-4 Cache 控制器顶层模块图

在 QuartusII 软件中先实现一个 8 位的存储单元（见例程中的 MemCell.bdf），然后用这个 8 位的存储单元来构成一个 32 X 8 位的 Cache（见例程中的 CacheMem.bdf），这样就实现了 Cache 的存储体。

再实现一个 4 位的存储单元（见例程中的 TableCell.bdf），然后用这个 4 位的存储单元来构成一个 8 X 4 位的区表存储器，用来存放区号和有效位（见例程中的 CacheTable.bdf），在这个文件中，还实现了一个区号比较器，如果主存地址的区号 E 和区表中相应单元中的区号相等，且有效位为 1，则 Cache 命中，否则 Cache 失效，标志为 M，M 为 0 时表示 Cache 失效。

当 Cache 命中时，就将 Cache 存储体中相应单元的数据送往 CPU，这个过程比较简单。当 Cache 失效时，就将主存中相应块中的数据读出写入 Cache 中，这样 Cache 控制器就要产生访问主存储器的地址和主存储器的读信号，由于每块占四个单元，所以需要连续访问四次主存，这就需要有一个低地址发生器，即一个 2 位计数器（见例程中的 Counter.vhd），将低 2 位和 CPU 给出的高 6 位地址组合起来，形成访问主存储器的地址。M 就可以做为主存的读信号，这样，在时钟的控制下，就可以将主存中相应的块写入到 Cache 的相应块中，最后再修改区表（见例程中的 CacheCtrl.bdf）。

#### 2.2.4 实验步骤

(1) 使用 Quartus II 软件编辑实现相应的逻辑并进行编译，直到编译通过，Cache 控制器在 EP2C5T144C8 芯片中对应的引脚如图 2-2-5 所示，框外文字表示 I/O 号，框内文字表示该引脚的含义（本实验例程见‘安装路径\FPGA \CacheCtrl\CacheCtrl.qpf’工程）。



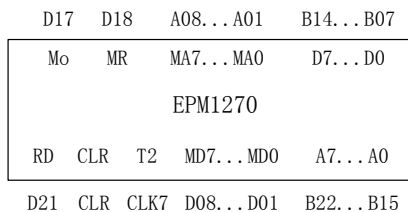


图 2-2-5 引脚分配图

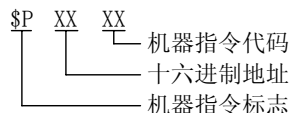
(2) 关闭实验系统电源，按图 2-2-6 连接实验电路，并检查无误，图中将用户需要连接的信号用圆圈标明。

(3) 打开实验系统电源，将下载电缆插入 FPGA 单元的 JTAG 口，把生成的 SOF 文件下载到 FPGA 单元中去，FPGA 单元介绍见实验 1.2。

(4) 将时序与操作台单元的开关 KK3 置为‘运行’档，CLR 信号由 CON 单元的 CLR 模拟给出，按动 CON 单元的 CLR 按钮，清空区表。

(5) 预先往主存写入数据：联机软件提供了机器程序下载功能，以代替手动读写主存，机器程序以指定的格式写入到以 TXT 为后缀的文件中，机器指令的格式如下：

机器指令格式说明：



如\$P 1F 11，表示机器指令的地址为 1FH，指令值为 11H，本次实验只初始化 00-0FH 共 16 个单元，初始数据如下，程序中分号‘;’为注释符，分号后面的内容在下载时将被忽略掉。

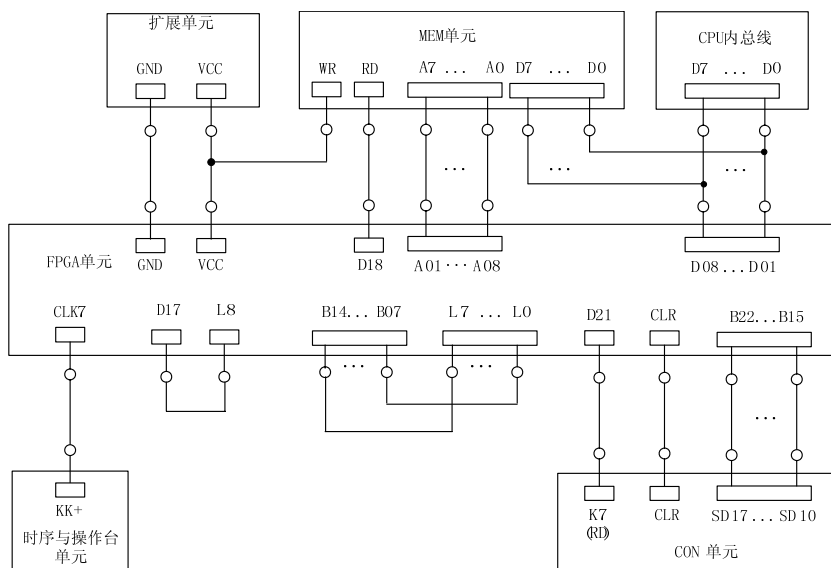


图 2-2-6 实验接线图

; //\*\*\*\*\*

```

; //                                     //
; //      Cache 控制器实验指令文件      //
; //                                     //
; //      By TangDu CO.,LTD              //
; //                                     //
; //*****                               //

; //***** Start Of Main Memory Data ***** //
$P 00 11      ; 数据
$P 01 22
$P 02 33
$P 03 44
$P 04 55
$P 05 66
$P 06 77
$P 07 88
$P 08 99
$P 09 AA
$P 0A BB
$P 0B CC
$P 0C DD
$P 0D EE
$P 0E FF
$P 0F 00
; //***** End Of Main Memory Data ***** //

```

用联机软件的“【转储】—【装载】”功能将该格式 (\*.TXT) 文件装载入实验系统。装入过程中，在软件的输出区的‘结果’栏会显示装载信息，如当前正在装载的是机器指令还是微指令，还剩多少条指令等。

(6) 联机软件在启动时会读取所有机器指令和微指令，在指令区显示，软件启动后，也可以选择菜单命令“【转储】—【刷新指令区】”读取下位机指令，并在指令区显示。点击指令区的‘主存’TAB 按钮，两列数据中显示了主存的所有数据，第一列为主存地址，第二列为该地址中的数据。对上面文件检查机器程序是否正确，如果不正确，则说明写入操作失败，应重新写入，可以通过联机软件单独修改某个单元的指令，单击需修改单元的数据，此时该单元变为编辑框，输入 2 位数据并回车，编辑框消失，写入数据以红色显示。

(7) CPU 访问主存地址由 CON 单元的 SD17…SD10 模拟给出，如 00000001。CPU 访问主存的读信号由 CON 单元的 K7 模拟给出，置 K7 为低，可以观察到 FPGA 单元上的 L8 指示灯亮，L0…L7 指示灯灭，表示 Cache 失效。此时按动 KK 按钮四次，注意 CPU 内总线上指示灯的变化情况，地址会依次加一，数据总线上显示的是当前主存数据，按动四次 KK 按钮后，L8 指示灯变灭，L0…L7 上显示的值即为 Cache 送往 CPU 的数据。

(8) 重新给出主存访问地址，如 00000011，L8 指示灯变灭，表示 Cache 命中，说明第 0 块数据已写入 Cache。

(9) 记住 01H 单元的数据，然后通过联机软件，修改 01H 单元的数据，重新给出主存访问地址 00000001，再次观察 L0-L7 指示灯表示的值是 01H 单元修改前的值，说明送往 CPU 的数据是由 Cache 给出的。

(10) 重新给出大于 03H 地址，体会 Cache 控制器的工作过程。

## 第 3 章 控制器

控制器是计算机的核心部件，计算机的所有硬件都是在控制器的控制下，完成程序规定的操作。控制器的基本功能就是把机器指令转换为按照一定时序控制机器各部件的工作信号，使各部件产生一系列动作，完成指令所规定的任务。

本章安排了两个实验：时序发生器设计实验和微程序控制器实验。

### 3.1 时序发生器设计实验

#### 3.1.1 实验目的

- (1) 掌握时序发生器的原理及其设计方法。
- (2) 熟悉 FPGA 应用设计及 EDA 软件的使用。

#### 3.1.2 实验设备

PC 机一台，TD-CMX 实验系统一套。

#### 3.1.3 实验原理

计算机的工作是按照时序分步地执行。这就需要能产生周期节拍、脉冲等时序信号的部件，称为时序发生器。如图 3-1-1 所示。

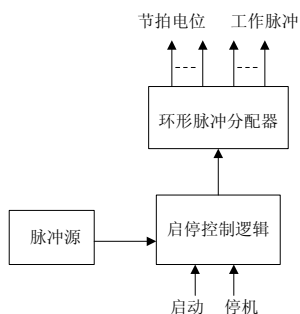


图 3-1-1 时序发生器

时序部件包括：

- (1) 脉冲源：又称主振荡器，为计算机提供基准时钟信号。

(2) 脉冲分配器：对主频脉冲进行分频，产生节拍电位和脉冲信号。时钟脉冲经过脉冲发生器产生时标脉冲、节拍电位及周期状态电位。一个周期状态电位包含多个节拍电位，而一个节拍单位又包含多个时标脉冲。

- (3) 启停控制电路：用来控制主脉冲的启动和停止。

本实验是用 VHDL 语言来实现一个时序发生器，输出如图 3-1-2 所示 T1…T4 四个节拍信号。时序发生器需要一个脉冲源，由时序单元的  $\phi$  提供(时序单元的介绍见附录二)，一个总清零 CLR，为低时，T1…T4 输出低。一个停机信号 STOP，当 T4 的下沿到来时，且 STOP 为低，T1…T4 输出低。一个启动信号 START，当 START、T1…T4 都为低，且 STOP 为高，T1…T4 输出环形脉冲。

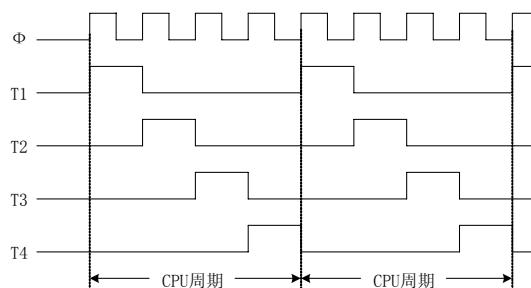


图 3-1-2 时序状态图

可通过 4 位循环移位寄存器来实现  $T4 \cdots T1$ ，CLR 为总清零信号，STOP 为低时在  $T4$  脉冲下沿清零时序，时序发生器启动后，移位寄存器在时钟的上沿循环左移一位，移位寄存器的输出端即为  $T4 \cdots T1$ 。

### 3.1.4 实验步骤

(1) 参照上面的实验原理，用 VHDL 语言来具体设计一个时序发生器。使用 Quartus II 软件编辑 VHDL 文件并进行编译，时序发生器在 EP2C5T144C8 芯片中对应的引脚如图 3-1-3 所示，框外文字表示 I/O 号，框内文字表示该引脚的含义（本实验例程见‘安装路径\FPGA\Timer\Timer.qpf’工程）。

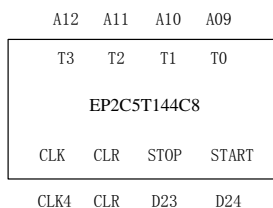


图 3-1-3 实验接线图

(2) 关闭实验系统电源，按图 3-1-4 连接实验电路，并检查无误，图中将用户需要连接的信号用圆圈标明。

(3) 打开实验系统电源，将下载电缆插入 FPGA 单元的 JTAG 口，把生成的 SOF 文件下载到 FPGA 单元中去，FPGA 单元介绍见实验 1.2。

(4) 将 CON 单元的 K7 (START)、K6 (STOP) 开关置 ‘1’，按下 CON 单元的总清按钮 CLR，使  $T1 \cdots T4$  输出低。运行联机软件，选择“【波形】—【打开】”打开逻辑示波器窗口，然后选择“【波形】—【运行】”启动逻辑示波器，逻辑示波器窗口显示  $T1 \cdots T4$  四路时序信号波形。

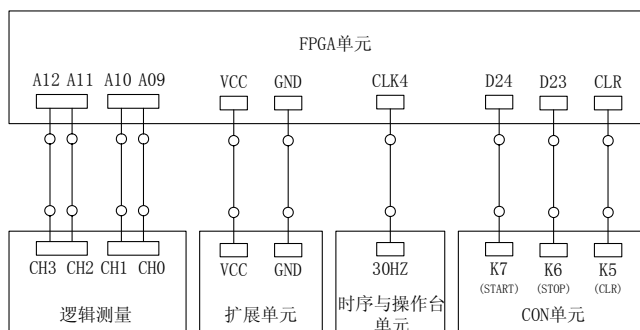


图 3-1-4 实验接线图

(5) 将 CON 单元的 K7 (START) 开关置 ‘1-0-1’, 启动 T1…T4 时序, 示波器窗口显示 T1…T4 波形, 如图 3-1-5 所示。

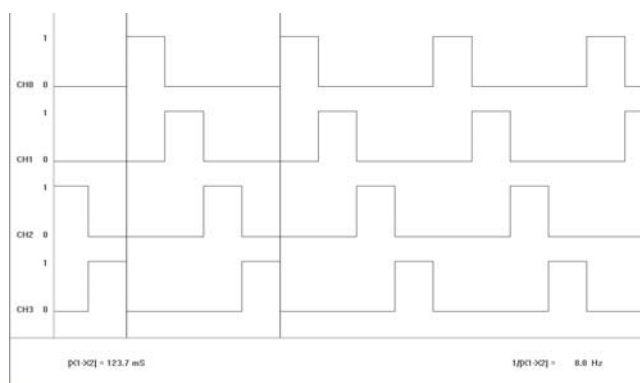


图 3-1-5 时序波形图

(6) 将 CON 单元的 K6 (STOP) 开关置 ‘0’, 停止 T1…T4 时序, 示波器窗口显示 T1…T4 波形均变为低。

## 3.2 微程序控制器实验

### 3.2.1 实验目的

- (1) 掌握微程序控制器的组成原理。
- (2) 掌握微程序的编制、写入，观察微程序的运行过程。

### 3.2.2 实验设备

PC 机一台，TD-CMX 实验系统一套。

### 3.2.3 实验原理

微程序控制器的基本任务是完成当前指令的翻译和执行，即将当前指令的功能转换成可以控制的硬件逻辑部件工作的微命令序列，完成数据传送和各种处理操作。它的执行方法就是将控制各部件动作的微命令的集合进行编码，即将微命令的集合仿照机器指令一样，用数字代码的形式表示，这种表示称为微指令。这样就可以用一个微指令序列表示一条机器指令，这种微指令序列称为微程序。微程序存储在一种专用的存储器中，称为控制存储器，微程序控制器原理框图如图 3-2-1 所示。

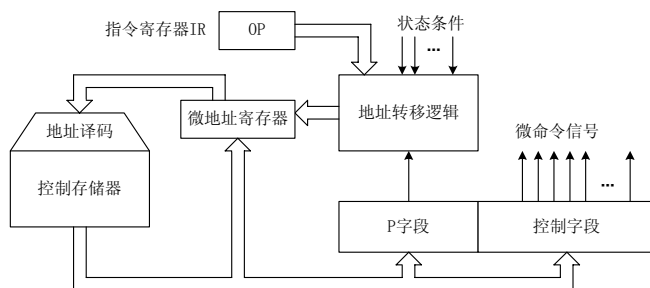
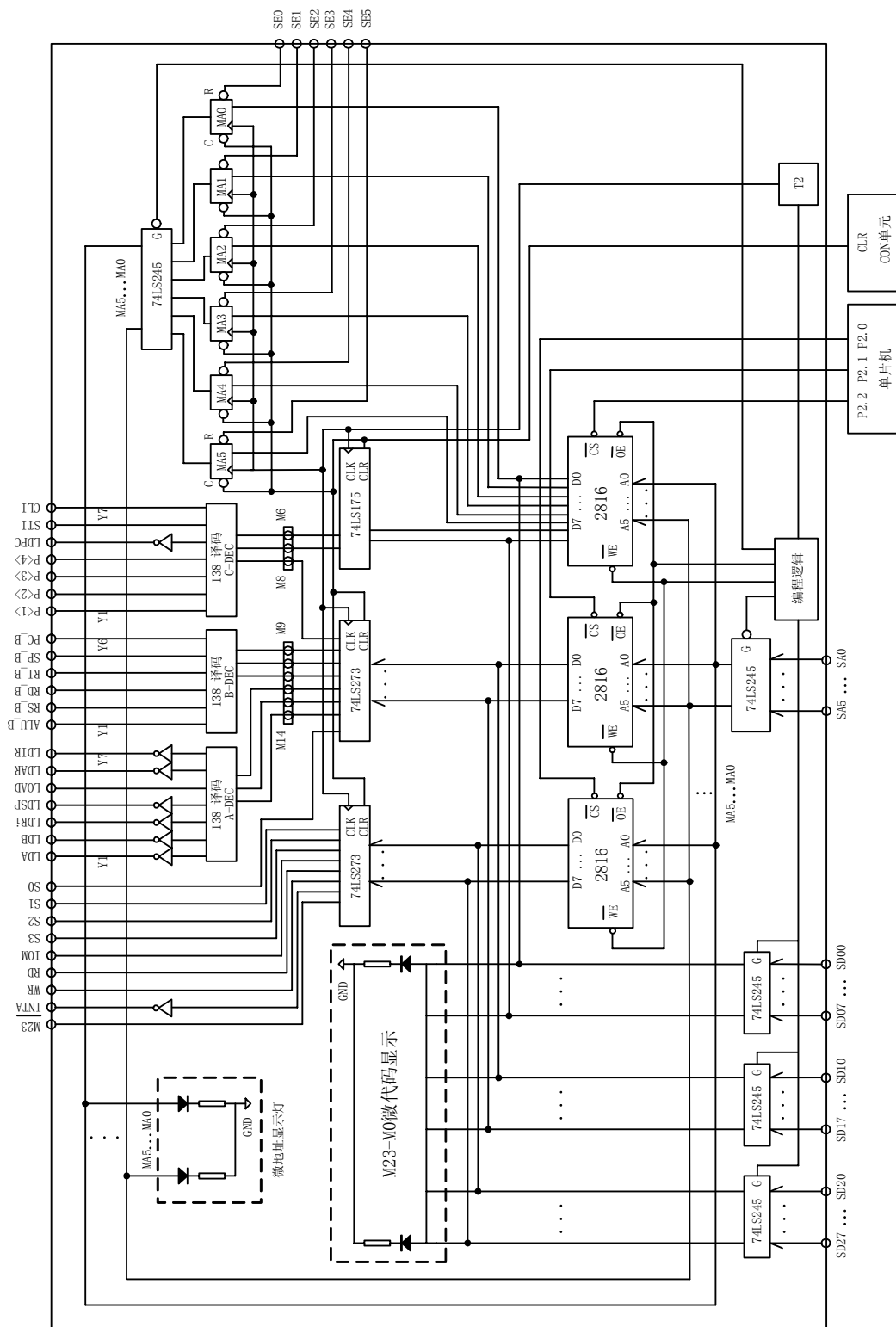


图 3-2-1 微程序控制器组成原理框图

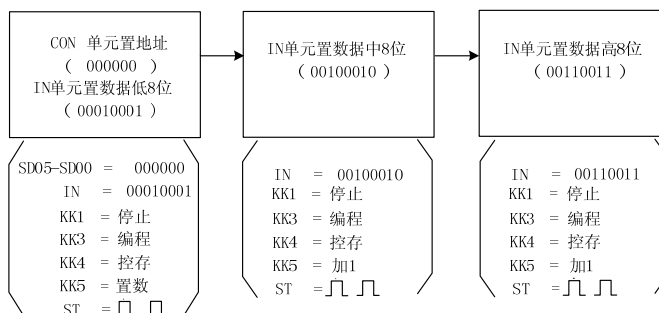
控制器是严格按照系统时序来工作的，因而时序控制对于控制器的设计是非常重要的，从前面的实验可以很清楚地了解时序电路的工作原理，本实验所用的时序由时序单元来提供，分为四拍 TS1、TS2、TS3、TS4，时序单元的介绍见附录 2。

微程序控制器的组成见图 3-2-2，其中控制存储器采用 3 片 2816 的 E<sup>2</sup>PROM，具有掉电保护功能，微命令寄存器 18 位，用两片 8D 触发器（273）和一片 4D（175）触发器组成。微地址寄存器 6 位，用三片正沿触发的双 D 触发器（74）组成，它们带有清“0”端和预置端。在不判别测试的情况下，T2 时刻打入微地址寄存器的内容即为下一条微指令地址。当 T4 时刻进行测试判别时，转移逻辑满足条件后输出的负脉冲通过强置端将某一触发器置为“1”状态，完成地址修改。

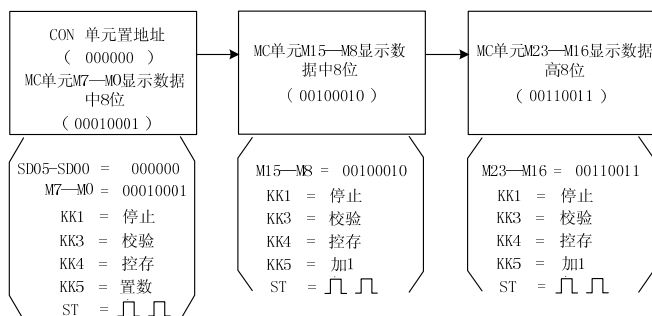




在实验平台中设有一组编程控制开关 KK3、KK4、KK5（位于时序与操作台单元），可实现对存储器（包括存储器和控制存储器）的三种操作：编程、校验、运行。考虑到对于存储器（包括存储器和控制存储器）的操作大多集中在一个地址连续的存储空间中，实验平台提供了便利的手动操作方式。以向 00H 单元中写入 332211 为例，对于控制存储器进行编辑的具体操作步骤如下：首先将 KK1 拨至‘停止’档、KK3 拨至‘编程’档、KK4 拨至‘控存’档、KK5 拨至‘置数’档，由 CON 单元的 SD05——SD00 开关给出需要编辑的控存单元首地址（000000），IN 单元开关给出该控存单元数据的低 8 位（00010001），连续两次按动时序与操作台单元的开关 ST（第一次按动后 MC 单元低 8 位显示该单元以前存储的数据，第二次按动后显示当前改动的数据），此时 MC 单元的指示灯 MA5——MA0 显示当前地址（000000），M7——M0 显示当前数据（00010001）。然后将 KK5 拨至‘加 1’档，IN 单元开关给出该控存单元数据的中 8 位（00100010），连续两次按动开关 ST，完成对该控存单元中 8 位数据的修改，此时 MC 单元的指示灯 MA5——MA0 显示当前地址（000000），M15——M8 显示当前数据（00100010）；再由 IN 单元开关给出该控存单元数据的高 8 位（00110011），连续两次按动开关 ST，完成对该控存单元高 8 位数据的修改此时 MC 单元的指示灯 MA5——MA0 显示当前地址（000000），M23——M16 显示当前数据（00110011）。此时被编辑的控存单元地址会自动加 1（01H），由 IN 单元开关依次给出该控存单元数据的低 8 位、中 8 位和高 8 位配合每次开关 ST 的两次按动，即可完成对后续单元的编辑。



编辑完成后需进行校验，以确保编辑的正确。以校验 00H 单元为例，对于控制存储器进行校验的具体操作步骤如下：首先将 KK1 拨至‘停止’档、KK3 拨至‘校验’档、KK4 拨至‘控存’档、KK5 拨至‘置数’档。由 CON 单元的 SD05——SD00 开关给出需要校验的控存单元地址（000000），连续两次按动开关 ST，MC 单元指示灯 M7——M0 显示该单元低 8 位数据（00010001）；KK5 拨至‘加 1’档，再连续两次按动开关 ST，MC 单元指示灯 M15——M8 显示该单元中 8 位数据（00100010）；再连续两次按动开关 ST，MC 单元指示灯 M23——M16 显示该单元高 8 位数据（00110011）。再连续两次按动开关 ST，地址加 1，MC 单元指示灯 M7——M0 显示 01H 单元低 8 位数据。如校验的微指令出错，则返回输入操作，修改该单元的数据后再进行校验，直至确认输入的微代码全部准确无误为止，完成对微指令的输入。



位于实验平台 MC 单元左上角一列三个指示灯 MC2、MC1、MC0 用来指示当前操作的微程序字  
段，分别对应 M23——M16、M15——M8、M7——M0。实验平台提供了比较灵活的手动操作方式，  
比如在上述操作中在对地址置数后将开关 KK4 拨至‘减 1’档，则每次随着开关 ST 的两次拨动  
操作，字节数依次从高 8 位到低 8 位递减，减至低 8 位后，再按动两次开关 ST，微地址会自动  
减一，继续对下一个单元的操作。

微指令字长共 24 位，控制位顺序如表 3-2-1：

表 3-2-1 微指令格式

23	22	21	20	19	18-15	14-12	11-9	8-6	5-0
M23	M22	WR	RD	IOM	S3-S0	A字段	B字段	C字段	MA5-MA0

#### A字段

14	13	12	选择
0	0	0	NOP
0	0	1	LDA
0	1	0	LDB
0	1	1	LDRO
1	0	0	保留
1	0	1	保留
1	1	0	保留
1	1	1	LDIR

#### B字段

11	10	9	选择
0	0	0	NOP
0	0	1	ALU_B
0	1	0	RO_B
0	1	1	保留
1	0	0	保留
1	0	1	保留
1	1	0	保留
1	1	1	保留

#### C字段

8	7	6	选择
0	0	0	NOP
0	0	1	P<1>
0	1	0	保留
0	1	1	保留
1	0	0	保留
1	0	1	保留
1	1	0	保留
1	1	1	保留

其中 MA5…MA0 为 6 位的后续微地址，A、B、C 为三个译码字段，分别由三个控制位译码出  
多位。C 字段中的 P<1>为测试字位。其功能是根据机器指令及相应微代码进行译码，使微程序  
转入相应的微地址入口，从而实现完成对指令的识别，并实现微程序的分支，本系统上的指令  
译码原理如图 3-2-3 所示，图中 I7…I2 为指令寄存器的第 7…2 位输出，SE5…SE0 为微控器单  
元微地址锁存器的强置端输出，指令译码逻辑在 IR 单元的 INS\_DEC (GAL20V8) 中实现。

从图 3-2-2 中也可以看出，微控器产生的控制信号比表 3-2-1 中的要多，这是因为实验的  
不同，所需的控制信号也不一样，本实验只用了部分的控制信号。

本实验除了用到指令寄存器 (IR) 和通用寄存器 R0 外，还要用到 IN 和 OUT 单元，从微控  
器出来的信号中只有 IOM、WR 和 RD 三个信号，所以对这两个单元的读写信号还应先经过译码，  
其译码原理如图 3-2-4 所示。IR 单元的原理图如图 3-2-5 所示，R0 单元原理图如图 3-2-7 所示，  
IN 单元的原理图见图 2-1-3 所示，OUT 单元的原理图见图 3-2-6 所示。

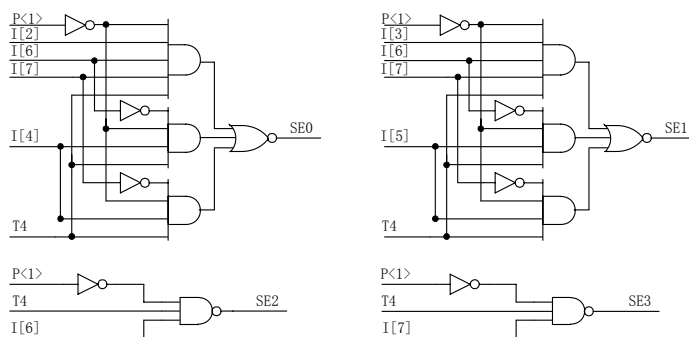


图 3-2-3 指令译码原理图

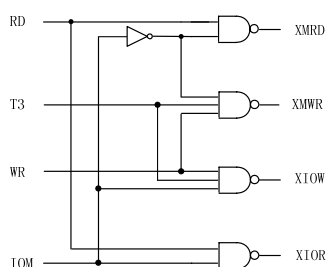


图 3-2-4 读写控制逻辑

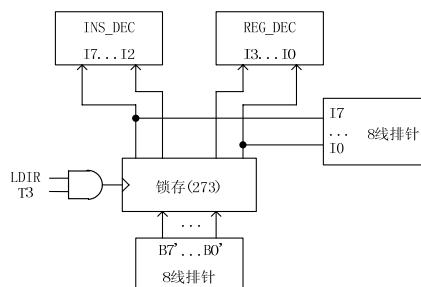


图 3-2-5 IR 单元原理图

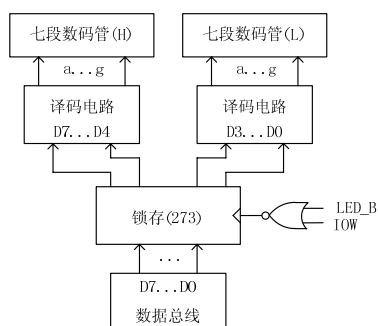


图 3-2-6 OUT 单元原理图

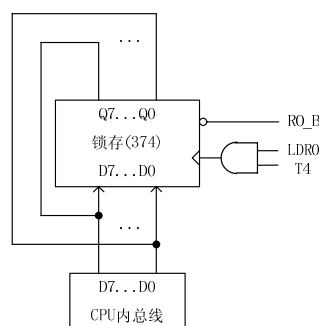


图 3-2-7 R0 原理图

本实验安排了四条机器指令，分别为 ADD (0000 0000)、IN (0010 0000)、OUT (0011 0000) 和 HLT (0101 0000)，括号中为各指令的二进制代码，指令格式如下：

助记符	机器指令码	说明
IN	0010 0000	IN → R0
ADD	0000 0000	R0 + R0 → R0
OUT	0011 0000	R0 → OUT
HLT	0101 0000	停机

实验中机器指令由 CON 单元的二进制开关手动给出，其余单元的控制信号均由微程序控制器自动产生，为此可以设计出相应的数据通路图，见图 3-2-8 所示。

几条机器指令对应的参考微程序流程图如图 3-2-9 所示。图中一个矩形方框表示一条微指

令，方框中的内容为该指令执行的微操作，右上角的数字是该条指令的微地址，右下角的数字是该条指令的后续微地址，所有微地址均用 16 进制表示。向下的箭头指出了下一条要执行的指令。P<1>为测试字，根据条件使微程序产生分支。

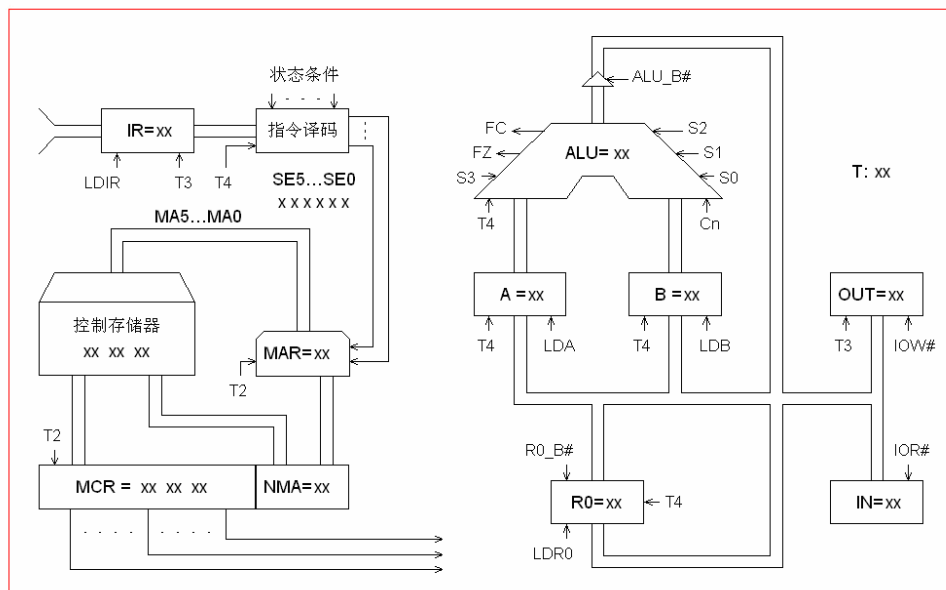


图 3-2-8 数据通路图

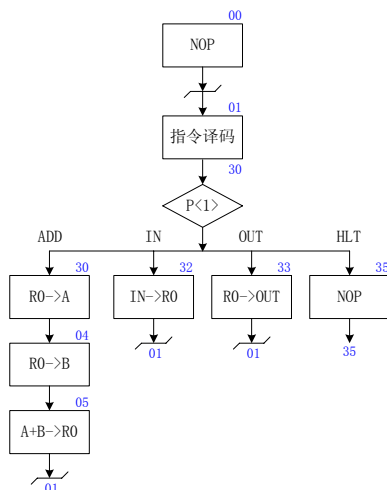


图 3-2-9 微程序流程图

将全部微程序按微指令格式变成二进制微代码, 可得到表 3-2-2 的二进制代码表。

表 3-2-2 二进制微代码表

地址	十六进制	高五位	S3-S0	A 字段	B 字段	C 字段	MA5-MA0
00	00 00 01	00000	0000	000	000	000	000001
01	00 70 70	00000	0000	111	000	001	110000
04	00 24 05	00000	0000	010	010	000	000101
05	04 B2 01	00000	1001	011	001	000	000001
30	00 14 04	00000	0000	001	010	000	000100
32	18 30 01	00011	0000	011	000	000	000001
33	28 04 01	00101	0000	000	010	000	000001
35	00 00 35	00000	0000	000	000	000	110101

### 3.2.4 实验步骤

1. 按图 3-2-10 所示连接实验线路，仔细查线无误后接通电源。如果有‘滴’报警声，说明总线有竞争现象，应关闭电源，检查接线，直到错误排除。

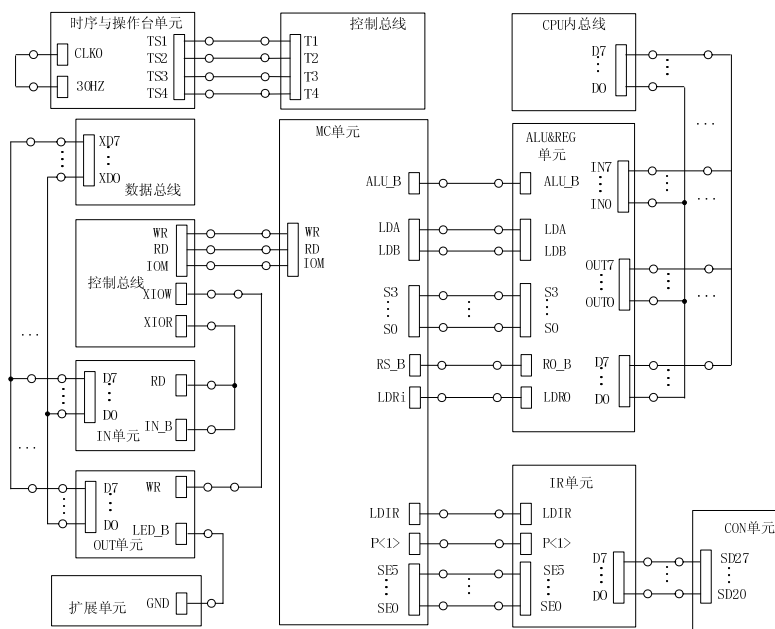


图 3-2-10 实验接线图

2. 对微控器进行读写操作，分两种情况：手动读写和联机读写。

1) 手动读写

(1) 手动对微控器进行编程（写）

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

② 使用 CON 单元的 SD05——SD00 给出微地址，IN 单元给出低 8 位应写入的数据，连续两

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

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

⑤ 重复①、②、③、④四步，将表 3-2-2 的微代码写入 2816 芯片中。

### (2) 手动对微控器进行校验 (读)

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

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

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

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

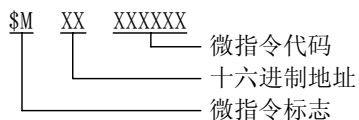
⑤ 重复①、②、③、④四步，完成对微代码的校验。如果校验出微代码写入错误，重新写入、校验，直至确认微指令的输入无误为止。

## 2) 联机读写

### (1) 将微程序写入文件

联机软件提供了微程序下载功能，以代替手动读写微控器，但微程序得以指定的格式写入到以 TXT 为后缀的文件中，微程序的格式如下：

微指令格式说明:



如\$M 1F 112233, 表示微指令的地址为 1FH, 微指令值为 11H (高)、22H (中)、33H (低), 本次实验的微程序如下, 其中分号 ‘;’ 为注释符, 分号后面的内容在下载时将被忽略掉。

## (2) 写入微程序

用联机软件的“【转储】—【装载】”功能将该格式 (\*.TXT) 文件装载入实验系统。装入过程中, 在软件的输出区的‘结果’栏会显示装载信息, 如当前正在装载的是机器指令还是微指令, 还剩多少条指令等。

### (3) 校验微程序

选择联机软件的“【转储】—【刷新指令区】”可以读出下位机所有的机器指令和微指令，并在指令区显示。检查微控器相应地址单元的数据是否和表 3-2-2 中的十六进制数据相同，如果不同，则说明写入操作失败，应重新写入，可以通过联机软件单独修改某个单元的微指令，先用鼠标左键单击指令区的‘微存’TAB 按钮，然后再单击需修改单元的数据，此时该单元变为编辑框，输入 6 位数据并回车，编辑框消失，并以红色显示写入的数据。

```

; //*****
; //
; //
; //

```

```

; //          微控器实验指令文件          //
; //                                           //
; //          By TangDu CO.,LTD             //
; //                                           //
; //*****                               //
; //**** Start Of MicroController Data **** //
$M 00 000001    ; NOP
$M 01 007070    ; CON(INS)->IR, P<1>
$M 04 002405    ; R0->B
$M 05 04B201    ; A 加 B->R0
$M 30 001404    ; R0->A
$M 32 183001    ; IN->R0
$M 33 280401    ; R0->OUT
$M 35 000035    ; NOP
; //**** End Of MicroController Data **** //

```

### 3. 运行微程序

运行时也分两种情况：本机运行和联机运行。

#### 1) 本机运行

① 将时序与操作台单元的开关 KK1、KK3 置为‘运行’档，按动 CON 单元的 CLR 按钮，将微地址寄存器（MAR）清零，同时也将指令寄存器（IR）、ALU 单元的暂存器 A 和暂存器 B 清零。

② 将时序与操作台单元的开关 KK2 置为‘单拍’档，然后按动 ST 按钮，体会系统在 T1、T2、T3、T4 节拍中各做的工作。T2 节拍微控器将后续微地址（下条执行的微指令的地址）打入微地址寄存器，当前微指令打入微指令寄存器，并产生执行部件相应的控制信号；T3、T4 节拍根据 T2 节拍产生的控制信号做出相应的执行动作，如果测试位有效，还要根据机器指令及当前微地址寄存器中的内容进行译码，使微程序转入相应的微地址入口，实现微程序的分支。

③ 按动 CON 单元的 CLR 按钮，清微地址寄存器（MAR）等，并将时序与单元的开关 KK2 置为‘单步’档。

④ 置 IN 单元数据为 00100011，按动 ST 按钮，当 MC 单元后续微地址显示为 000001 时，在 CON 单元的 SD27…SD20 模拟给出 IN 指令 00100000 并继续单步执行，当 MC 单元后续微地址显示为 000001 时，说明当前指令已执行完；在 CON 单元的 SD27…SD20 给出 ADD 指令 00000000，该指令将会在下个 T3 被打入指令寄存器（IR），它将 R0 中的数据和其自身相加后送 R0；接下来在 CON 单元的 SD27…SD20 给出 OUT 指令 00110000 并继续单步执行，在 MC 单元后续微地址显示为 000001 时，观察 OUT 单元的显示值是否为 01000110。

#### 2) 联机运行

联机运行时，进入软件界面，在菜单上选择【实验】—【微控器实验】，打开本实验的数据通路图，也可以通过工具栏上的下拉框打开数据通路图，数据通路图如图 3-2-8 所示。

将时序与操作台单元的开关 KK1、KK3 置为‘运行’档，按动 CON 单元的总清开关后，按动软件中单节拍按钮，当后续微地址（通路图中的 MAR）为 000001 时，置 CON 单元 SD27…SD20，产生相应的机器指令，该指令将会在下个 T3 被打入指令寄存器（IR），在后面的节拍中将执行这条机器指令。仔细观察每条机器指令的执行过程，体会后续微地址被强置转换的过程，这是计算机识别和执行指令的根基。也可以打开微程序流程图，跟踪显示每条机器指令的执行过程。

按本机运行的顺序给出数据和指令，观察最后的运算结果是否正确。

## 第 4 章 系统总线与总线接口

总线是计算机中连接各个功能部件的纽带，是计算机各部件之间进行信息传输的公共通路。总线不只是一组简单的信号传输线，它还是一组协议。分时与共享是总线的两大特征。所谓共享，在总线上可以挂接多个部件，它们都可以使用这一信息通路来和其他部件传送信息。所谓分时，同一总线在同一时刻，只能有一个部件占领总线发送信息，其他部件要发送信息得在该部件发送完释放总线后才能申请使用。总线结构是决定计算机性能、功能、可扩展性和标准化程度的重要因素。

本章安排了三个实验：系统总线和具有基本输入输出功能的总线接口实验、具有中断控制功能的总线接口实验和具有 DMA 控制功能的总线接口实验。

### 4.1 系统总线和具有基本输入输出功能的总线接口实验

#### 4.1.1 实验目的

1. 理解总线的概念及其特性。
2. 掌握控制总线的功能和应用。

#### 4.1.2 实验设备

PC 机一台，TD-CMX 实验系统一套。

#### 4.1.3 实验原理

由于存储器和输入、输出设备最终是要挂接到外部总线上，所以需要外部总线提供数据信号、地址信号以及控制信号。在该实验平台中，外部总线分为数据总线、地址总线、和控制总线，分别为外设提供上述信号。外部总线和 CPU 内总线之间通过三态门连接，同时实现了内外总线的分离和对于数据流向的控制。地址总线可以为外部设备提供地址信号和片选信号。由地址总线的高位进行译码，系统的 I/O 地址译码原理见图 4-1-1（在地址总线单元）。由于使用 A6、A7 进行译码，I/O 地址空间被分为四个区，如表 4-1-1 所示：

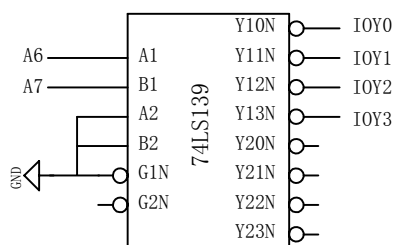


图 4-1-1 I/O 地址译码原理图

表 4-1-1 I/O 地址空间分配

A7	A6	选定	地址空间
----	----	----	------



00	IOY0	00-3F
01	IOY1	40-7F
10	IOY2	80-BF
11	IOY3	C0-FF

为了实现对于 MEM 和外设的读写操作，还需要一个读写控制逻辑，使得 CPU 能控制 MEM 和 I/O 设备的读写，实验中的读写控制逻辑如图 4-1-2 所示，由于 T3 的参与，可以保证写脉宽与 T3 一致，T3 由时序单元的 TS3 给出（时序单元的介绍见附录 2）。IOM 用来选择是对 I/O 设备还是对 MEM 进行读写操作，IOM=1 时对 I/O 设备进行读写操作，IOM=0 时对 MEM 进行读写操作。RD=1 时为读，WR=1 时为写。

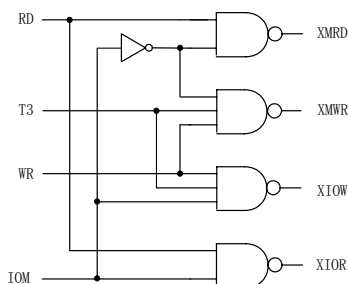


图 4-1-2 读写控制逻辑

在理解读写控制逻辑的基础上我们设计一个总线传输的实验。实验所用总线传输实验框图如图 4-1-3 所示，它将几种不同的设备挂至总线上，有存储器、输入设备、输出设备、寄存器。这些设备都需要有三态输出控制，按照传输要求恰当有序的控制它们，就可实现总线信息传输。

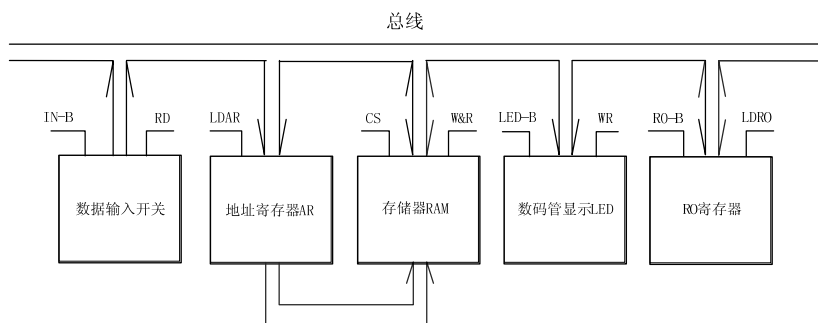


图 4-1-3 总线传输实验框图

#### 4.1.4 实验步骤

##### 1. 读写控制逻辑设计实验。

(1) 按照图 4-1-4 实验接线图进行连线。

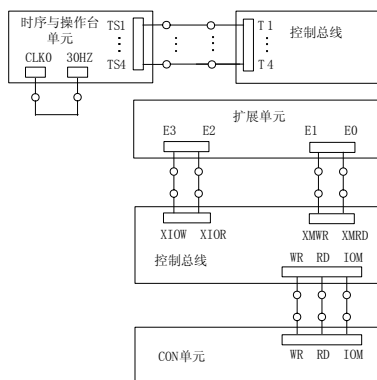


图 4-1-4 实验接线图

(2) 具体操作步骤图示如下：

首先将时序与操作台单元的开关 KK1、KK3 置为‘运行’档，开关 KK2 置为‘单拍’档，按动 CON 单元的总清按钮 CLR，并执行下述操作。

- ① 对 MEM 进行读操作 (WR=0, RD=1, IOM=0)，此时 E0 灭, 表示存储器读功能信号有效。
- ② 对 MEM 进行写操作 (WR=1, RD=0, IOM=0)，连续按动开关 ST，观察扩展单元数据指示灯，指示灯显示为 T3 时刻时，E1 灭, 表示存储器写功能信号有效。
- ③ 对 I/O 进行读操作 (WR=0, RD=1, IOM=1)，此时 E2 灭, 表示 I/O 读功能信号有效。
- ④ 对 I/O 进行写操作 (WR=1, RD=0, IOM=1)，连续按动开关 ST，观察扩展单元数据指示灯，指示灯显示为 T3 时刻时，E3 灭, 表示 I/O 写功能信号有效。

## 2. 基本输入输出功能的总线接口实验。

(1) 根据挂在总线上的几个基本部件，设计一个简单的流程：

- ① 输入设备将一个数打入 R0 寄存器。
- ② 输入设备将另一个数打入地址寄存器。
- ③ 将 R0 寄存器中的数写入到当前地址的存储器中。
- ④ 将当前地址的存储器中的数用 LED 数码管显示。

(2) 按照图 4-1-5 实验接线图进行连线。

(3) 具体操作步骤图示如下：

进入软件界面，选择菜单命令“【实验】—【简单模型机】”，打开简单模型机实验数据通路图。

将时序与操作台单元的开关 KK1、KK3 置为‘运行’档，开关 KK2 置为‘单拍’档，CON 单元所有开关置 0（由于总线有总线竞争报警功能，在操作中应当先关闭应关闭的输出开关，再打开应打开的输出开关，否则可能由于总线竞争导致实验出错），按动 CON 单元的总清按钮 CLR，然后通过运行程序，在数据通路图中观测程序的执行过程。

- ① 输入设备将 11H 打入 R0 寄存器。

将 IN 单元置 00010001，K7 置为 1，关闭 R0 寄存器的输出；K6 置为 1，打开 R0 寄存器的输入；WR、RD、IOM 分别置为 0、1、1，对 IN 单元进行读操作；LDAR 置为 0，不将数据总线的数打入地址寄存器。连续四次点击图形界面中的“单节拍运行”按钮（运行一个机

器周期)，观察图形界面，在 T4 时刻完成对寄存器 R0 的写入操作。

② 将 R0 中的数据 11H 打入存储器 01H 单元。

将 IN 单元置 00000001（或其他数值）。K7 置为 1，关闭 R0 寄存器的输出；K6 置为 0，关闭 R0 寄存器的输入；WR、RD、IOM 分别置为 0、1、1，对 IN 单元进行读操作；LDAR 置为 1，将数据总线的数打入地址寄存器。连续四次点击图形界面上的“单节拍运行”按钮，观察图形界面，在 T3 时刻完成对地址寄存器的写入操作。

先将 WR、RD、IOM 分别置为 1、0、0，对存储器进行写操作；再把 K7 置为 0，打开 R0 寄存器的输出；K6 置为 0，关闭 R0 寄存器的输入；LDAR 置为 0，不将数据总线的数打入地址寄存器。连续四次点击图形界面上的“单节拍运行”按钮，观察图形界面，在 T3 时刻完成对存储器的写入操作。

③ 将当前地址的存储器中的数写入到 R0 寄存器中。

将 IN 单元置 00000001（或其他数值），K7 置为 1，关闭 R0 寄存器的输出；K6 置为 0，关闭 R0 寄存器的输入；WR、RD、IOM 分别置为 0、1、1，对 IN 单元进行读操作；LDAR 置为 1，不将数据总线的数打入地址寄存器。连续四次点击图形界面上的“单节拍运行”按钮，观察图形界面，在 T3 时刻完成对地址寄存器的写入操作。

将 K7 置为 1，关闭 R0 寄存器的输出；K6 置为 1，打开 R0 寄存器的输入；WR、RD、IOM 分别置为 0、1、0，对存储器进行读操作；LDAR 置为 0，不将数据总线的数打入地址寄存器。连续四次点击图形界面上的“单节拍运行”按钮，观察图形界面，在 T3 时刻完成对寄存器 R0 的写入操作。

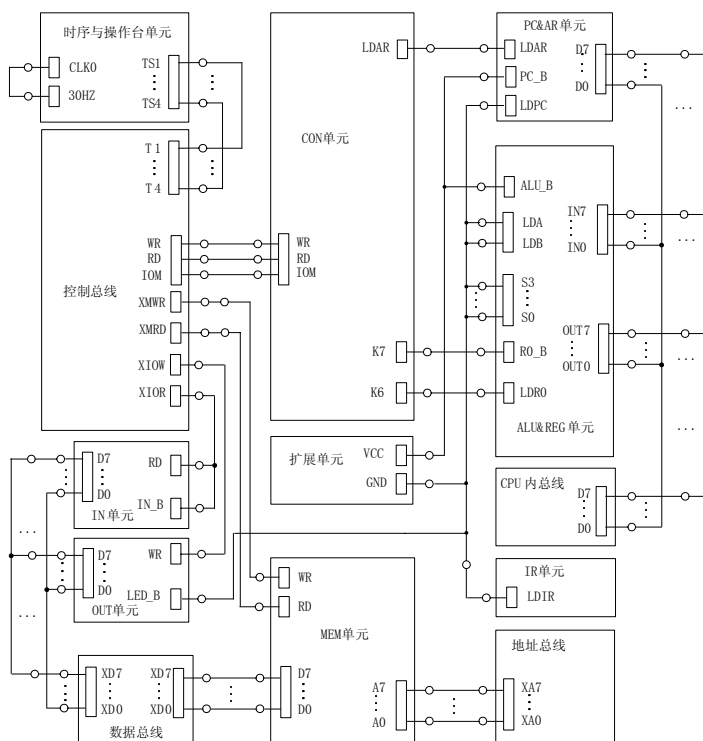


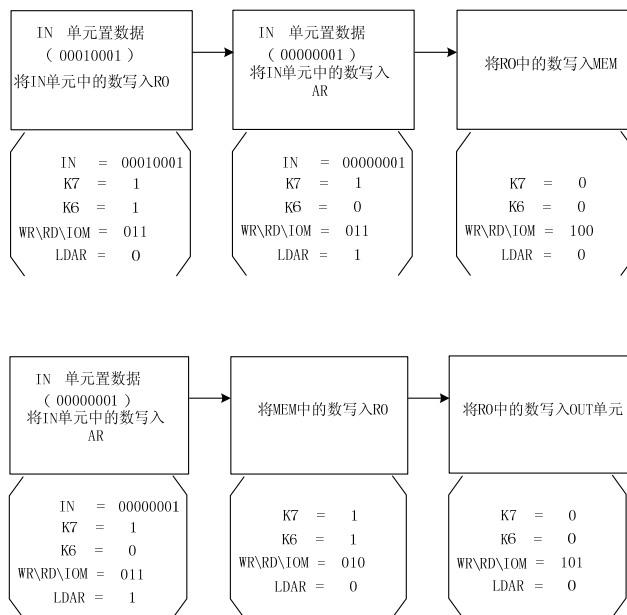
图 4-1-5 实验接线图

注：由于采用简单模型机的数据通路图，为了不让悬空的信号引脚影响通路图的显示结果，将

这些引脚置为无效。在接线时为了方便,可将管脚接到 CON 单元闲置的开关上,若开关打到‘1’,等效于接到‘VCC’;若开关打到‘0’,等效于接到‘GND’。

④ 将 R0 寄存器中的数用 LED 数码管显示。

先将 WR、RD、IOM 分别置为 1、0、1,对 OUT 单元进行写操作;再将 K7 置为 0,打开 R0 寄存器的输出;K6 置为 0,关闭 R0 寄存器的输入; LDAR 置为 0,不将数据总线的数打入地址寄存器。连续四次点击图形界面中的“单节拍运行”按钮,观察图形界面,在 T3 时刻完成对 OUT 单元的写入操作。



## 4.2 具有中断控制功能的总线接口实验

### 4.2.1 实验目的

1. 掌握中断控制信号线的功能和应用。
2. 掌握在系统总线上设计中断控制信号线的方法。

### 4.2.2 实验设备

PC 机一台, TD-CMX 实验系统一套, 电压表一台。

### 4.2.3 实验原理

为了实现中断控制, CPU 必须有一个中断使能寄存器, 并且可以通过指令对该寄存器进行操作。设计下述中断使能寄存器, 其原理如图 4-2-1 所示。其中 EI 为中断允许信号, CPU 开中断指令 STI 对其置 1, 而 CPU 关中断指令 CLI 对其置 0。每条指令执行完时, 若允许中断, CPU 给出开中断使能标志 STI, 打开中断使能寄存器, EI 有效。EI 再和外部给出的中断请求信号一起参与指令译码, 使程序进入中断处理流程。

本实验要求设计的系统总线具备有类 X86 的中断功能, 当外部中断请求有效、CPU 允许响应中断, 在当前指令执行完时, CPU 将响应中断。当 CPU 响应中断时, 将会向 8259 发送两个连续的  $\overline{\text{INTA}}$  信号, 请注意, 8259 是在接收到第一个  $\overline{\text{INTA}}$  信号后锁住向 CPU 的中断请求信号 INTR (高电平有效), 并且在第二个  $\overline{\text{INTA}}$  信号到达后将其变为低电平 (自动 EOI 方式), 所以, 中断请求信号 IR0 应该维持一段时间, 直到 CPU 发送出第一个  $\overline{\text{INTA}}$  信号, 这才是一个有效的中断请求。8259 在收到第二个  $\overline{\text{INTA}}$  信号后, 就会将中断向量号发送到数据总线, CPU 读取中断向量号, 并转入相应的中断处理程序中。在读取中断向量时, 需要从数据总线向 CPU 内总线传送数据。所以需要设计数据缓冲控制逻辑, 在  $\overline{\text{INTA}}$  信号有效时, 允许数据从数据总线流向 CPU 内总线。其原理图如图 4-2-2 所示。其中 RD 为 CPU 从外部读取数据的控制信号。

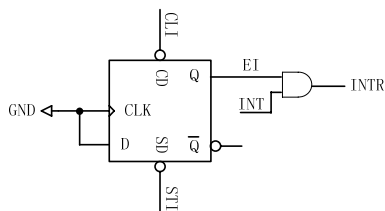


图 4-2-1 中断使能寄存器原理图

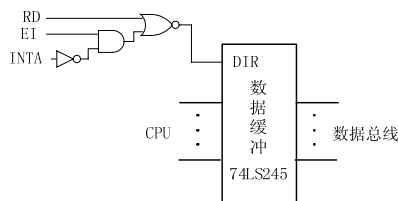


图 4-2-2 数据缓冲控制原理图

在控制总线部分表现为当 CPU 开中断允许信号 STI 有效、关中断允许信号 CLI 无效时, 中断标志 EI 有效, 当 CPU 开中断允许信号 STI 无效、关中断允许信号 CLI 有效时, 中断标志 EI 无效。EI 无效时, 外部的中断请求信号不能发送给 CPU。

### 4.2.4 实验步骤

- (1) 按照图 4-2-3 实验接线图进行连线。

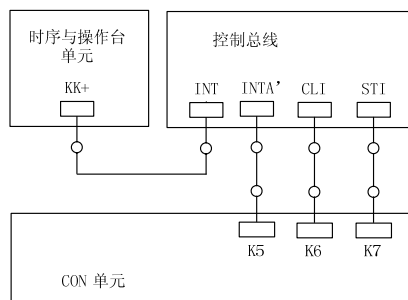


图 4.2-3 实验接线图

(2) 具体操作步骤图示如下:

① 对总线进行置中断操作 ( $K6=1, K7=0$ ), 观察控制总线部分的中断允许指示灯 EI, 此时 EI 亮, 表示允许响应外部中断。按动时序与操作台单元的开关 KK, 观察控制总线单元的指示灯 INTR, 发现当开关 KK 按下时 INTR 变亮, 表示总线将外部的中断请求送到 CPU。

② 对总线进行清中断操作 ( $K6=0, K7=1$ ), 观察控制总线部分的中断允许指示灯 EI, 此时 EI 灭, 表示禁止响应外部中断。按动时序与操作台单元的开关 KK, 观察控制总线单元的指示灯 INTR, 发现当开关 KK 按下时 INTR 不变, 仍然为灭, 表示总线锁死了外部的中断请求。

③ 对总线进行置中断操作 ( $K6=1, K7=0$ ), 当 CPU 给出的中断应答信号  $INTA'$  ( $K5=0$ ) 有效时, 使用电压表测量数据缓冲 74LS245 的 DIR (第 1 脚), 显示为低, 表示 CPU 允许外部送中断向量号。

## 4.3 具有 DMA 控制功能的总线接口实验

### 4.3.1 实验目的

1. 掌握 DMA 控制信号线的功能和应用。
2. 掌握在系统总线上设计 DMA 控制信号线的方法。

### 4.3.2 实验设备

PC 机一台, TD-CMX 实验系统一套, 电压表一台。

### 4.3.3 实验原理

有一类外设在使用时需要占用总线, 其中的典型代表是 DMA 控制机。在使用这类外设时, 总线的控制权要在 CPU 和外设之间进行切换, 这就需要总线具有相应的信号来实现这种切换, 避免总线竞争, 使 CPU 和外设能够正常工作。下面以 DMA 操作为例, 设计相应的总线控制信号线。实验原理图如图 4-3-1 所示。

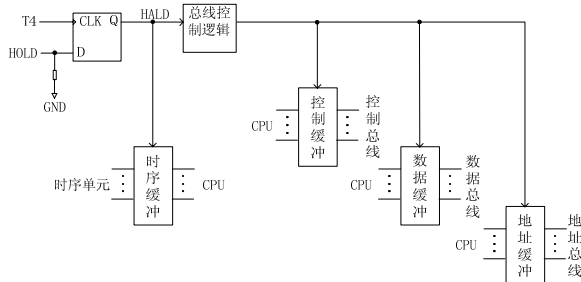


图 4-3-1 实验原理图

进行 DMA 操作时, 外设向 DMAC (DMA 控制机) 发出 DMA 传送请求, DMAC 通过总线上的 HOLD 信号向 CPU 提出 DMA 请求。CPU 在完成当前总线周期后对 DMA 请求做出响应。CPU 的响应包括两个方面, 一方面让出总线控制权, 一方面将有效的 HALD 信号加到 DMAC 上, 通知 DMAC 可以使用总线进行数据传输。此时 DMAC 进行 DMA 传输, 传输完成后, 停止向 CPU 发 HOLD 信号, 撤消总线请求, 交还总线控制权。CPU 在收到无效的 HOLD 信号后, 一方面使 HALD 无效, 另一方面又重新开始控制总线, 实现正常的运行。

如图 4-3-1 所示, 在每个机器周期的 T4 时刻根据 HOLD 信号来判断是否有 DMA 请求, 如果有, 则产生有效的 HALD 信号, HALD 信号一方面锁死 CPU 的时钟信号, 使 CPU 保持当前状态, 等待 DMA 操作的结束。另一方面使控制缓冲、数据缓冲、地址缓冲都处于高阻状态, 隔断 CPU 与外总线的联系, 将外总线交由 DMAC 控制。当 DMA 操作结束后, DMAC 将 HOLD 信号置为无效, DMA 控制逻辑在 T4 时刻将 HALD 信号置为无效, HALD 信号一方面打开 CPU 的时钟信号, 使 CPU 开始正常运行。另一方面把控制缓冲、数据缓冲和地址缓冲交由 CPU 控制, 恢复 CPU 对总线的控制权。

在本实验中, 控制缓冲由写在 16V8 芯片中的组合逻辑实现, 数据缓冲和地址缓冲由数据总线和地址总线左侧的 74LS245 实现。以存储器读信号为例, 体现 HALD 信号对控制总线的控制。首先模拟 CPU 给出存储器读信号 (置 WR、RD、IOM 分别为 0、1、0), 当 HALD 信号无效时, 总线上输出的存储器读信号 XMRD 为有效态“0”, 当 HALD 信号有效时, 总线上输出的存储器读信号 XMRD 为高阻态。可以自行设计其余的控制信号验证实验。

### 4.3.4 实验步骤

(1) 按照图 4-3-2 实验接线图进行连线。

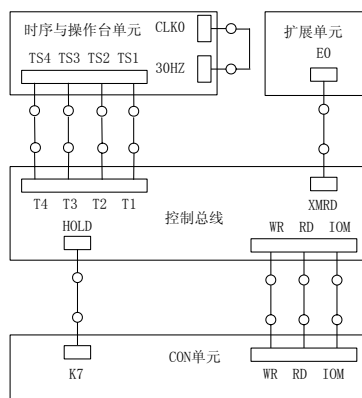


图 4-3-2 实验接线图

(2) 具体操作步骤如下：

① 将时序与操作台单元的开关 KK1、KK3 置为‘运行’档，开关 KK2 置为‘单拍’档，按动 CON 单元的总清按钮 CLR，将 CON 单元的 WR、RD、IOM 分别置为“0”、“1”、“0”，此时 XMRD 为低，相应的指示灯 E0 灭。使用电压表测量数据总线和地址总线左侧的芯片 74LS245 的使能控制信号（第 19 脚），发现电压为低，说明数据总线和地址总线与 CPU 连通。

② 然后将 CON 单元的 K7 置为 1，连续按动时序与操作台单元的开关 ST，T4 时刻控制总线的指示灯 HALD 为亮，继续按动开关 ST，发现控制总线单元的时钟信号指示灯 T1——T4 保持不变，说明 CPU 的时钟被锁死。此时 XMRD 为高阻态，相应的指示灯 E0 亮。使用万用表测量数据总线和地址总线左侧的芯片 74LS245 的使能控制信号（第 19 脚），发现电压为高，说明总线和 CPU 的连接被阻断。

③ 将 CON 单元的 K7 置为 0，按动时序与操作台单元的开关 ST，当时序信号走到 T4 时刻时，控制总线的指示灯 HALD 为灭，继续按动开关 ST，发现控制总线单元的时钟信号指示灯 T1——T4 开始变化，说明 CPU 的时钟被接通。此时 XMRD 受 CPU 控制，恢复有效为低，相应的指示灯 E0 灭。使用万用表测量数据总线和地址总线左侧的芯片 74LS245 的使能控制信号（第 19 脚），发现电压为低，说明总线和 CPU 恢复连通。



## 第 5 章 模型计算机

在前面的章节中，我们重点讨论计算机中每个部件的组成及特性，本节中，我们将重点讨论如何完整设计一台模型计算机，进一步建立整机的概念。

本章安排了三个实验：有 CPU 与简单模型机设计实验、硬布线控制器模型机设计实验和复杂模型机设计实验。

### 5.1 CPU 与简单模型机设计实验

#### 5.1.1 实验目的

- (1) 掌握一个简单 CPU 的组成原理。
- (2) 在掌握部件单元电路的基础上，进一步将其构造一台基本模型计算机。
- (3) 为其定义五条机器指令，编写相应的微程序，并上机调试掌握整机概念。

#### 5.1.2 实验设备

PC 机一台，TD-CMX 实验系统一套。

#### 5.1.3 实验原理

本实验要实现一个简单的 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（RO），指令寄存器（IR）、程序计数器（PC）和地址寄存器（AR）组成，如图 5-1-1 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。

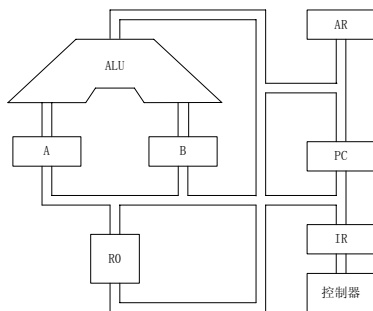


图 5-1-1 基本 CPU 构成原理图

除了程序计数器（PC），其余部件在前面的实验中都已用到，在此不再讨论。系统的程序计数器（PC）和地址寄存器（AR）集成在一片 FPGA 芯片中。CLR 连接至 CON 单元的总清端 CLR，按下 CLR 按钮，将使 PC 清零，LDPC 和 T3 相与后作为计数器的计数时钟，当 LOAD 为低时，计数时钟到来后将 CPU 内总线上的数据打入 PC。

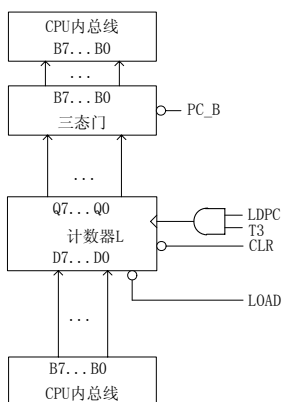


图 5-1-2 程序计数器(PC)原理图

本模型机和前面微程序控制器实验相比，新增加一条跳转指令 JMP，共有五条指令：IN（输入）、ADD（二进制加法）、OUT（输出）、JMP（无条件转移）、HLT（停机），其指令格式如下（高 4 位为操作码）：

助记符	机器指令码	说明
IN	0010 0000	IN $\rightarrow$ R0
ADD	0000 0000	R0 + R0 $\rightarrow$ R0
OUT	0011 0000	R0 $\rightarrow$ OUT
JMP addr	1110 0000 *****	addr $\rightarrow$ PC
HLT	0101 0000	停机

其中 JMP 为双字节指令，其余均为单字节指令，\*\*\*\*\*为 addr 对应的二进制地址码。微程序控制器实验的指令是通过手动给出的，现在要求 CPU 自动从存储器读取指令并执行。根据以上要求，设计数据通路图，如图 5-1-3 所示。

本实验在前一个实验的基础上增加了三个部件，一是 PC（程序计数器），另一个是 AR（地址寄存器），还有就是 MEM（主存）。因而在微指令中应增加相应的控制位，其微指令格式如表 5-1-1 所示。

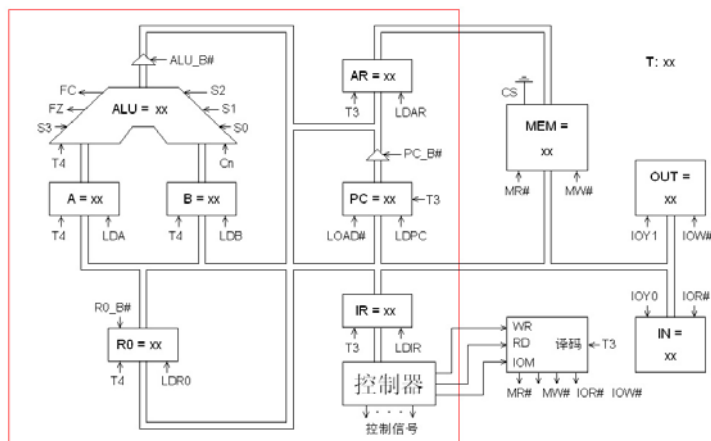


图 5-1-3 数据通路图

表 5-1-1 微指令格式

23	22	21	20	19	18-15	14-12	11-9	8-6	5-0
M23	M22	WR	RD	IOM	S3-S0	A字段	B字段	C字段	MA5-MA0

A字段				B字段				C字段			
14	13	12	选择	11	10	9	选择	8	7	6	选择
0	0	0	NOP	0	0	0	NOP	0	0	0	NOP
0	0	1	LDA	0	0	1	ALU_B	0	0	1	P<1>
0	1	0	LDB	0	1	0	R0_B	0	1	0	保留
0	1	1	LDRO	0	1	1	保留	0	1	1	保留
1	0	0	保留	1	0	0	保留	1	0	0	保留
1	0	1	LOAD	1	0	1	保留	1	0	1	LDPC
1	1	0	LDAR	1	1	0	PC_B	1	1	0	保留
1	1	1	LDIR	1	1	1	保留	1	1	1	保留

系统涉及到的微程序流程见图 5-1-4 所示，当拟定“取指”微指令时，该微指令的判别测试字段为 P<1>测试。指令译码原理见图 3-2-3 所示，由于“取指”微指令是所有微程序都使用的公用微指令，因此 P<1> 的测试结果出现多路分支。本机用指令寄存器的高 6 位（IR7—IR2）作为测试条件，出现 5 路分支，占用 5 个固定微地址单元，剩下的其它地方就可以一条微指令占用控存一个微地址单元随意填写，微程序流程图上的单元地址为 16 进制。

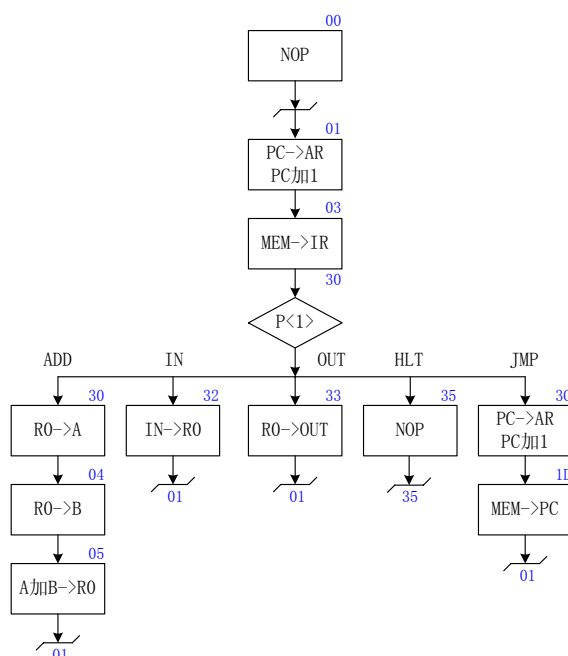


图 5-1-4 简单模型机微程序流程图

当全部微程序设计完毕后，应将每条微指令代码化，表 5-1-2 即为将图 5-1-4 的微程序流程图按微指令格式转化而成的“二进制微代码表”。

表 5-1-2 二进制微代码表

地址	十六进制	高五位	S3-S0	A 字段	B 字段	C 字段	MA5-MA0
00	00 00 01	00000	0000	000	000	000	000001
01	00 6D 43	00000	0000	110	110	101	000011
03	10 70 70	00010	0000	111	000	001	110000
04	00 24 05	00000	0000	010	010	000	000101
05	04 B2 01	00000	1001	011	001	000	000001
1D	10 51 41	00010	0000	101	000	101	000001
30	00 14 04	00000	0000	001	010	000	000100
32	18 30 01	00011	0000	011	000	000	000001
33	28 04 01	00101	0000	000	010	000	000001
35	00 00 35	00000	0000	000	000	000	110101
3C	00 6D 5D	00000	0000	110	110	101	011101

设计一段机器程序，要求从 IN 单元读入一个数据，存于 R0，将 R0 和自身相加，结果存于 R0，再将 R0 的值送 OUT 单元显示。

根据要求可以得到如下程序，地址和内容均为二进制数。

地 址	内 容	助记符	说 明
00000000	00100000 ;	START: IN R0	从 IN 单元读入数据送 R0
00000001	00000000 ;	ADD R0, R0	R0 和自身相加，结果送 R0
00000010	00110000 ;	OUT R0	R0 的值送 OUT 单元显示
00000011	11100000 ;	JMP START	跳转至 00H 地址
00000100	00000000 ;		
00000101	01010000 ;	HLT	停机

#### 5.1.4 实验步骤

1. 按图 5-1-5 连接实验线路。
2. 写入实验程序，并进行校验，分两种方式，手动写入和联机写入。

##### 1) 手动写入和校验

##### (1) 手动写入微程序

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

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

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

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

⑤ 重复①、②、③、④四步，将表 5-1-2 的微代码写入 2816 芯片中。

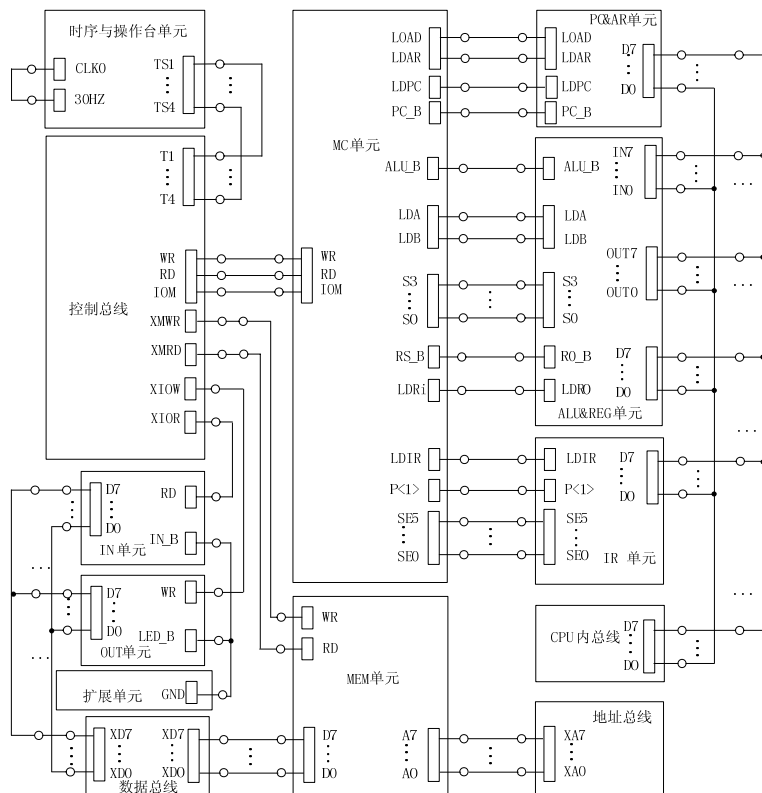


图 5-1-5 实验接线图

## (2) 手动校验微程序

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

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

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

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

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

## (3) 手动写入机器程序

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

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

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

④ IN 单元给出下一地址（地址自动加 1）应写入的数据，连续两次按动时序与操作台的开

关 ST，将 IN 单元的数据写到该单元中。然后地址会又自加 1，只需在 IN 单元输入后续地址的数据，连续两次按动时序与操作台的开关 ST，即可完成对该单元的写入。

⑤ 亦可重复①、②两步，将所有机器指令写入主存芯片中。

#### (4) 手动校验机器程序

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

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

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

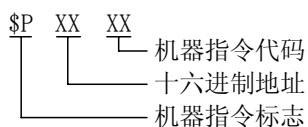
④ 连续两次按动时序与操作台的开关 ST，地址自动加 1，CPU 内总线的指数数据指示灯 D7——D0 显示该单元的数据。此后每两次按动时序与操作台的开关 ST，地址自动加 1，CPU 内总线的指数数据指示灯 D7——D0 显示该单元的数据，继续进行该操作，直至完成校验，如发现错误，则返回写入，然后校验，直至确认输入的所有指令准确无误。

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

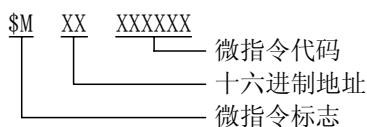
#### 2) 联机写入和校验

联机软件提供了微程序和机器程序下载功能，以代替手动读写微程序和机器程序，但是微程序和机器程序得以指定的格式写入到以 TXT 为后缀的文件中，微程序和机器程序的格式如下：

机器指令格式说明：



微指令格式说明：



本次实验程序如下，程序中分号‘；’为注释符，分号后面的内容在下载时将被忽略掉：

```

; //*****
; //
; //      CPU 与简单模型机实验指令文件      //
; //
; //      By TangDu CO.,LTD                    //
; //
; //*****

; //***** Start Of Main Memory Data ***** //
$P 00 20    ; START: IN  R0      从 IN 单元读入数据送 R0
$P 01 00    ; ADD R0,R0         R0 和自身相加，结果送 R0
$P 02 30    ; OUT R0            R0 的值送 OUT 单元显示
$P 03 E0    ; JMP START        跳转至 00H 地址
$P 04 00    ;
$P 05 50    ; HLT                停机
; //***** End Of Main Memory Data ***** //

; //**** Start Of MicroController Data **** //
$M 00 000001    ; NOP

```

```
$M 01 006D43      ; PC->AR, PC 加 1
$M 03 107070      ; MEM->IR, P<1>
$M 04 002405      ; R0->B
$M 05 04B201      ; A 加 B->R0
$M 1D 105141      ; MEM->PC
$M 30 001404      ; R0->A
$M 32 183001      ; IN->R0
$M 33 280401      ; R0->OUT
$M 35 000035      ; NOP
$M 3C 006D5D      ; PC->AR, PC 加 1
; /** End Of MicroController Data **/
```

选择联机软件的“【转储】—【装载】”功能，在打开文件对话框中选择上面所保存的文件，软件自动将机器程序和微程序写入指定单元。

选择联机软件的“【转储】—【刷新指令区】”可以读出下位机所有的机器指令和微指令，并在指令区显示，对照文件检查微程序和机器程序是否正确，如果不正确，则说明写入操作失败，应重新写入，可以通过联机软件单独修改某个单元的指令，以修改微指令为例，先用鼠标左键单击指令区的‘微存’TAB按钮，然后再单击需修改单元的数据，此时该单元变为编辑框，输入6位数据并回车，编辑框消失，并以红色显示写入的数据。

### 3. 运行程序

#### 方法一：本机运行

将时序与操作台单元的开关KK1、KK3置为‘运行’档，按动CON单元的总清按钮CLR，将使程序计数器PC、地址寄存器AR和微程序地址为00H，程序可以从头开始运行，暂存器A、B，指令寄存器IR和OUT单元也会被清零。

将时序与操作台单元的开关KK2置为‘单步’档，每按动一次ST按钮，即可单步运行一条微指令，对照微程序流程图，观察微地址显示灯是否和流程一致。每运行完一条微指令，观测一次CPU内总线和地址总线，对照数据通路图，分析总线上的数据是否正确。

当模型机执行完JMP指令后，检查OUT单元显示的数是否为IN单元值的2倍，按下CON单元的总清按钮CLR，改变IN单元的值，再次执行机器程序，从OUT单元显示的数判别程序执行是否正确。

#### 方法二：联机运行

将时序与操作台单元的开关KK1和KK3置为‘运行’档，进入软件界面，选择菜单命令“【实验】—【简单模型机】”，打开简单模型机数据通路图。

按动CON单元的总清按钮CLR，然后通过软件运行程序，选择相应的功能命令，即可联机运行、监控、调试程序，当模型机执行完JMP指令后，检查OUT单元显示的数是否为IN单元值的2倍。在数据通路图和微程序流中观测指令的执行过程，并观测软件中地址总线、数据总线以及微指令显示和下位机是否一致。

## 5.2 硬布线控制器模型机设计实验

### 5.2.1 实验目的

- (1) 掌握硬布线控制器的组成原理、设计方法。
- (2) 了解硬布线控制器和微程序控制器的各自优缺点。

### 5.2.2 实验设备

PC 机一台，TD-CMX 实验系统一套。

### 5.2.3 实验原理

硬布线控制器本质上是一种由门电路和触发器构成的复杂树形网络，它将输入逻辑信号转换成一组输出逻辑信号，即控制信号。硬布线控制器的输入信号有：指令寄存器的输出、时序信号和运算结果标志状态信号等，输出的就是所有各部件需要的各种微操作信号。

硬布线控制器的设计思想是：在硬布线控制器中，操作控制器发出的各种控制信号是时间因素和空间因素的函数。各个操作定时的控制构成了操作控制信号的时间特征，而各种不同部件的操作所需要的不同操作信号则构成了操作控制信号的空间特征。硬布线控制器就是把时间信号和操作信号组合，产生具有定时特点的控制信号。

简单模型机的控制器是微程序控制器，本实验中的模型机将用硬布线取代微程序控制器，其余部件和简单模型机的一样，所以其数据通路图也和简单模型机的一样，见图 5-1-3，机器指令也和简单模型机的机器指令一样，如下所示。

助记符	机器指令码	说明
IN	0010 0000	IN $\rightarrow$ R0
ADD	0000 0000	R0 + R0 $\rightarrow$ R0
OUT	0011 0000	R0 $\rightarrow$ OUT
JMP addr	1110 0000 *****	addr $\rightarrow$ PC
HLT	0101 0000	停机

根据指令要求，得出用时钟进行驱动的状态机描述，即得出其有限状态机，如图 5-2-1 所示。

下面分析每个状态中的基本操作：

S0：空操作，系统复位后的状态

S1：PC $\rightarrow$ AR, PC+1

S2：MEM $\rightarrow$ BUS, BUS $\rightarrow$ IR

S3：R0 $\rightarrow$ BUS, BUS $\rightarrow$ A

S4：R0 $\rightarrow$ BUS, BUS $\rightarrow$ B

S5：A 加 B $\rightarrow$ BUS, BUS $\rightarrow$ R0

S6：IN $\rightarrow$ BUS, BUS $\rightarrow$ R0

S7：R0 $\rightarrow$ BUS, BUS $\rightarrow$ OUT

S8：空操作

S9：PC $\rightarrow$ AR, PC+1

S10：MEM $\rightarrow$ BUS, BUS $\rightarrow$ PC



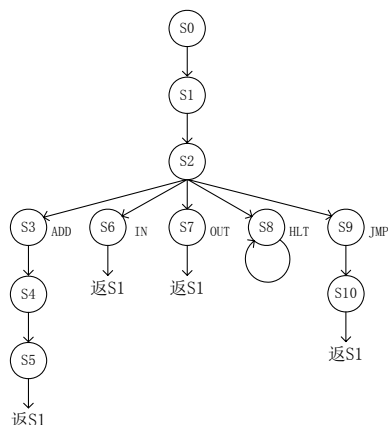


图 5-2-1 状态机描述

设计一段机器程序，要求从 IN 单元读入一个数据，存于 R0，将 R0 和自身相加，结果存于 R0，再将 R0 的值送 OUT 单元显示。

#### 5.2.4 实验步骤

(1) 分析每个状态所需的控制信号，并汇总成表，如表 5-2-1 所示。

表 5-2-1 控制信号表

状态号	控制信号
S0	0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0
S1	0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 1
S2	0 1 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0
S3	0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0
S4	0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0
S5	0 0 0 1 0 0 1 0 0 1 0 0 0 1 1 1 0
S6	0 1 1 0 0 0 0 0 0 1 0 0 1 1 1 1 0
S7	1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 0
S8	0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0
S9	0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 1
S10	0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1

控制信号由左至右，依次为：WR，RD，IOM，S3，S2，S1，S0，LDA，LDB，LOAD，LDAR，LDIR，ALU\_B，R0\_B，LDR0，PC\_B，LDPC。

(2) 用 VHDL 语言来设计本实验的状态机，使用 Quartus II 软件编辑 VHDL 文件并进行编译，硬布线控制器在 EP2C5T144C8 芯片中对应的引脚如图 5-2-2 所示(本实验例程见‘安装路径\FPGA\Controllor\Controllor.qpf’工程)。

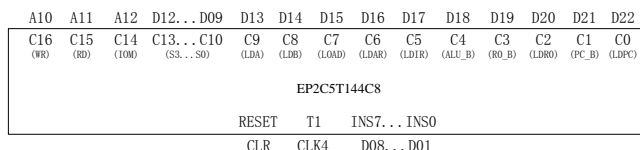


图 5-2-2 引脚分配图

(3) 关闭实验系统电源，按图 5-2-3 连接实验电路。注意：不要将 FPGA 扩展板上的“A09”引脚接至控制总线的“INTA’”否则可能导至实验失败。

(4) 打开实验系统电源，将下载电缆插入 FPGA 单元的 JTAG 口，把生成的 SOF 文件下载到 FPGA 单元中去。

(5) 用本实验定义的机器指令系统，可具体编写多种应用程序，下面给出的是本次实验的例程，其程序的文件名以.TXT 为后缀。程序中分号‘;’为注释符，分号后面的内容在下载时将被忽略掉

(6) 进入软件界面，装载机器指令，选择菜单命令“【实验】—【简单模型机】”，打开简单模型机数据通路图，按动 CON 单元的总清按钮 CLR，使程序计数器 PC 地址清零，控制器状态机回到 S0，程序从头开始运行，选择相应的功能命令，即可联机运行、监控、调试程序。

(7) 当模型机执行完 JMP 指令后，检查 OUT 单元显示的数是否为 IN 单元值的 2 倍，按下 CON 单元的总清按钮 CLR，改变 IN 单元的值，再次执行机器程序，从 OUT 单元显示的数判别程序执行是否正确。

```

; //*****
; //
; // 硬布线控制器模型机实验指令文件
; //
; // By TangDu CO.,LTD
; //
; //*****

; //**** Start Of Main Memory Data **** //
$P 00 20 ; START: IN R0 从 IN 单元读入数据送 R0
$P 01 00 ; ADD R0,R0 R0 和自身相加，结果送 R0
$P 02 30 ; OUT R0 R0 的值送 OUT 单元显示
$P 03 E0 ; JMP START 跳转至 START
$P 04 00 ;
$P 05 50 ; HLT 停机
; //***** End Of Main Memory Data *****//

```

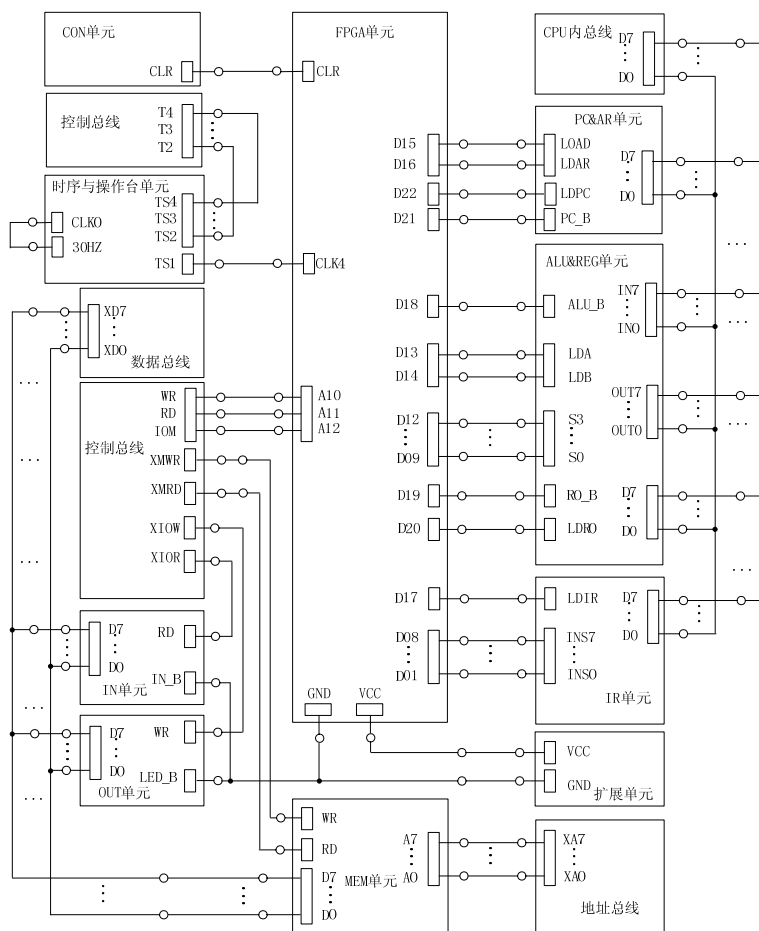


图 5-2-3 实验接线图

## 5.3 复杂模型机设计实验

### 5.3.1 实验目的

综合运用所学计算机组成原理知识，设计并实现较为完整的计算机。

### 5.3.2 实验设备

PC 机一台，TD-CMX 实验系统一套。

### 5.3.3 实验原理

下面讲述一下模型计算机的数据格式及指令系统。

#### 1. 数据格式

模型机规定采用定点补码表示法表示数据，字长为 8 位，8 位全用来表示数据（最高位不表示符号），数值表示范围是： $0 \leq X \leq 2^8 - 1$ 。

#### 2. 指令设计

模型机设计三大类指令共十五条，其中包括运算类指令、控制转移类指令，数据传送类指令。运算类指令包含三种运算，算术运算、逻辑运算和移位运算，设计有 6 条运算类指令，分别为：ADD、AND、INC、SUB、OR、RR，所有运算类指令都为单字节，寻址方式采用寄存器直接寻址。控制转移类指令有三条 HLT、JMP、BZC，用以控制程序的分支和转移，其中 HLT 为单字节指令，JMP 和 BZC 为双字节指令。数据传送类指令有 IN、OUT、MOV、LDI、LAD、STA 共 6 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. 指令格式

所有单字节指令（ADD、AND、INC、SUB、OR、RR、HLT 和 MOV）格式如下：

7 6 5 4	3 2	1 0
OP-CODE	RS	RD

其中，OP-CODE 为操作码，RS 为源寄存器，RD 为目的寄存器，并规定：

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

IN 和 OUT 的指令格式为：

7 6 5 4 (1)	3 2 (1)	1 0 (1)	7—0 (2)
-------------	---------	---------	---------

OP-CODE	RS	RD	P
---------	----	----	---

其中括号中的 1 表示指令的第一字节，2 表示指令的第二字节，OP-CODE 为操作码，RS 为源寄存器，RD 为目的寄存器，P 为 I/O 端口号，占用一个字节，系统的 I/O 地址译码原理见图 5-3-1（在地址总线单元）。

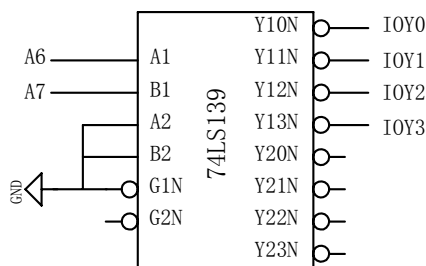


图 5-3-1 I/O 地址译码原理图

由于用的是地址总线的高两位进行译码，I/O 地址空间被分为四个区，如表 5-3-1 所示：

表 5-3-1 I/O 地址空间分配

A7 A6	选定	地址空间
00	IOY0	00-3F
01	IOY1	40-7F
10	IOY2	80-BF
11	IOY3	C0-FF

系统设计五种数据寻址方式，即立即、直接、间接、变址和相对寻址，LDI 指令为立即寻址，LAD、STA、JMP 和 BZC 指令均具备直接、间接、变址和相对寻址能力。

LDI 的指令格式如下，第一字节同前一样，第二字节为立即数。

7 6 5 4 (1)	3 2 (1)	1 0 (1)	7—0 (2)
OP-CODE	RS	RD	data

LAD、STA、JMP 和 BZC 指令格式如下。

7 6 5 4 (1)	3 2 (1)	1 0 (1)	7—0 (2)
OP-CODE	M	RD	D

其中 M 为寻址模式，具体见表 5-3-2，以 R2 做为变址寄存器 RI。

表 5-3-2 寻址方式

寻址模式 M	有效地址 E	说 明
--------	--------	-----

00	$E = D$	直接寻址
01	$E = (D)$	间接寻址
10	$E = (RI) + D$	RI 变址寻址
11	$E = (PC) + D$	相对寻址

#### 4. 指令系统

本模型机共有 15 条基本指令，表 5-3-3 列出了各条指令的格式、汇编符号、指令功能。

表 5-3-3 指令描述

助记符号	指令格式			指令功能	
MOV RD, RS	0100	RS	RD	RS → RD	
ADD RD, RS	0000	RS	RD	RD + RS → RD	
SUB RD, RS	1000	RS	RD	RD - RS → RD	
AND RD, RS	0001	RS	RD	RD ∧ RS → RD	
OR RD, RS	1001	RS	RD	RD ∨ RS → RD	
RR RD, RS	1010	RS	RD	RS右环移 → RD	
INC RD	0111	**	RD	RD+1 → RD	
LAD M D, RD	1100	M	RD	D	E → RD
STA M D, RS	1101	M	RD	D	RD → E
JMP M D	1110	M	**	D	E → PC
BZC M D	1111	M	**	D	当FC或FZ=1时, E → PC
IN RD, P	0010	**	RD	P	[P] → RD
OUT P, RS	0011	RS	**	P	RS → [P]
LDI RD, D	0110	**	RD	D	D → RD
HALT	0101	**	**		停机

#### 5.3.4 总体设计

本模型机的数据通路框图如图 5-3-2 所示。

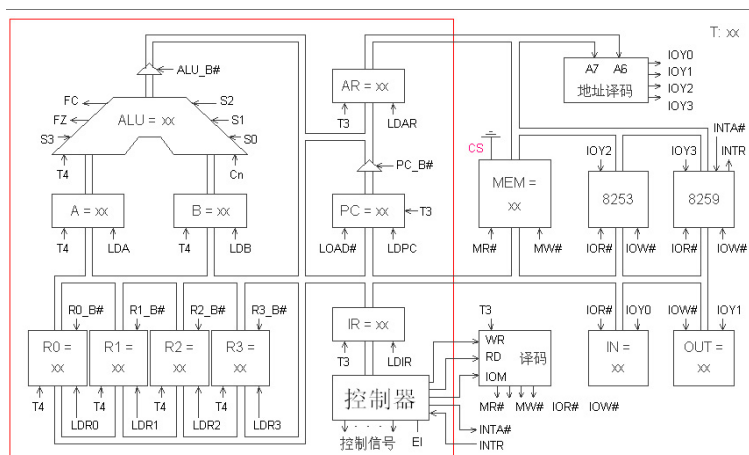


图 5-3-2 数据通路框图

和前面的实验相比，复杂模型机实验指令多，寻址方式多，只用一种测试已不能满足设计要求，为此指令译码电路需要重新设计。如图 5-3-3 所示在 IR 单元的 INS\_DEC 中实现。

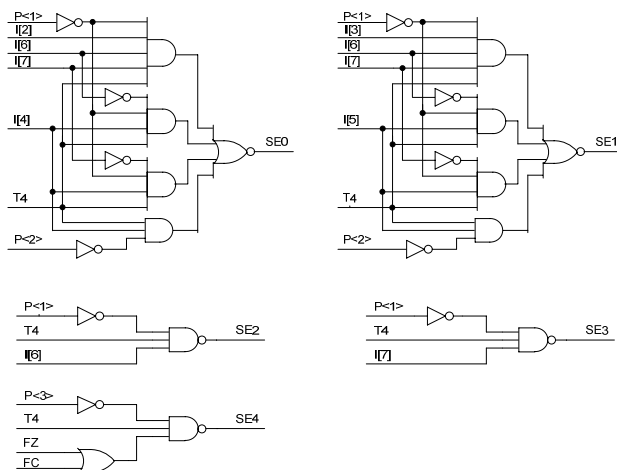


图 5-3-3 指令译码原理图

本实验中要用到四个通用寄存器 R3...R0，而对寄存器的选择是通过指令的低四位，为此还得设计一个寄存器译码电路，在 IR 单元的 REG\_DEC (GAL16V8) 中实现，如图 5-3-4 所示。

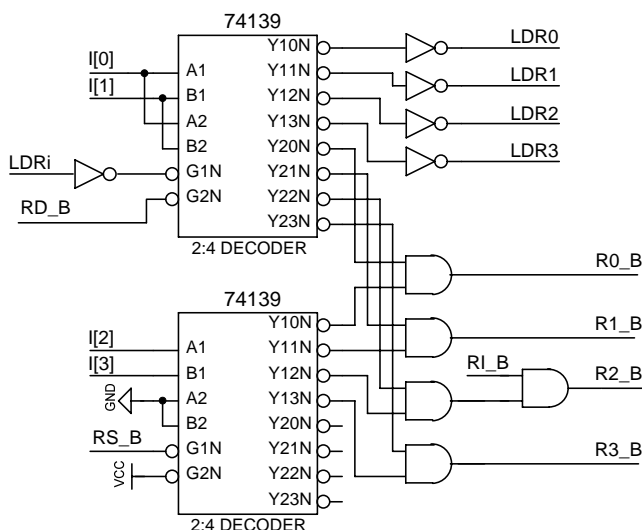


图 5-3-4 寄存器译码原理图

根据机器指令系统要求，设计微程序流程图及确定微地址，如图 5-3-5 所示。

按照系统建议的微指令格式，见表 5-3-4，参照微指令流程图，将每条微指令代码化，译成二进制代码表，见表 5-3-5，并将二进制代码表转换为联机操作时的十六进制格式文件。

表 5-3-4 微指令格式

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

A字段

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

B字段

11	10	9	选择
0	0	0	NOP
0	0	1	ALU_B
0	1	0	RS_B
0	1	1	RD_B
1	0	0	RI_B
1	0	1	保留
1	1	0	PC_B
1	1	1	保留

C字段

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



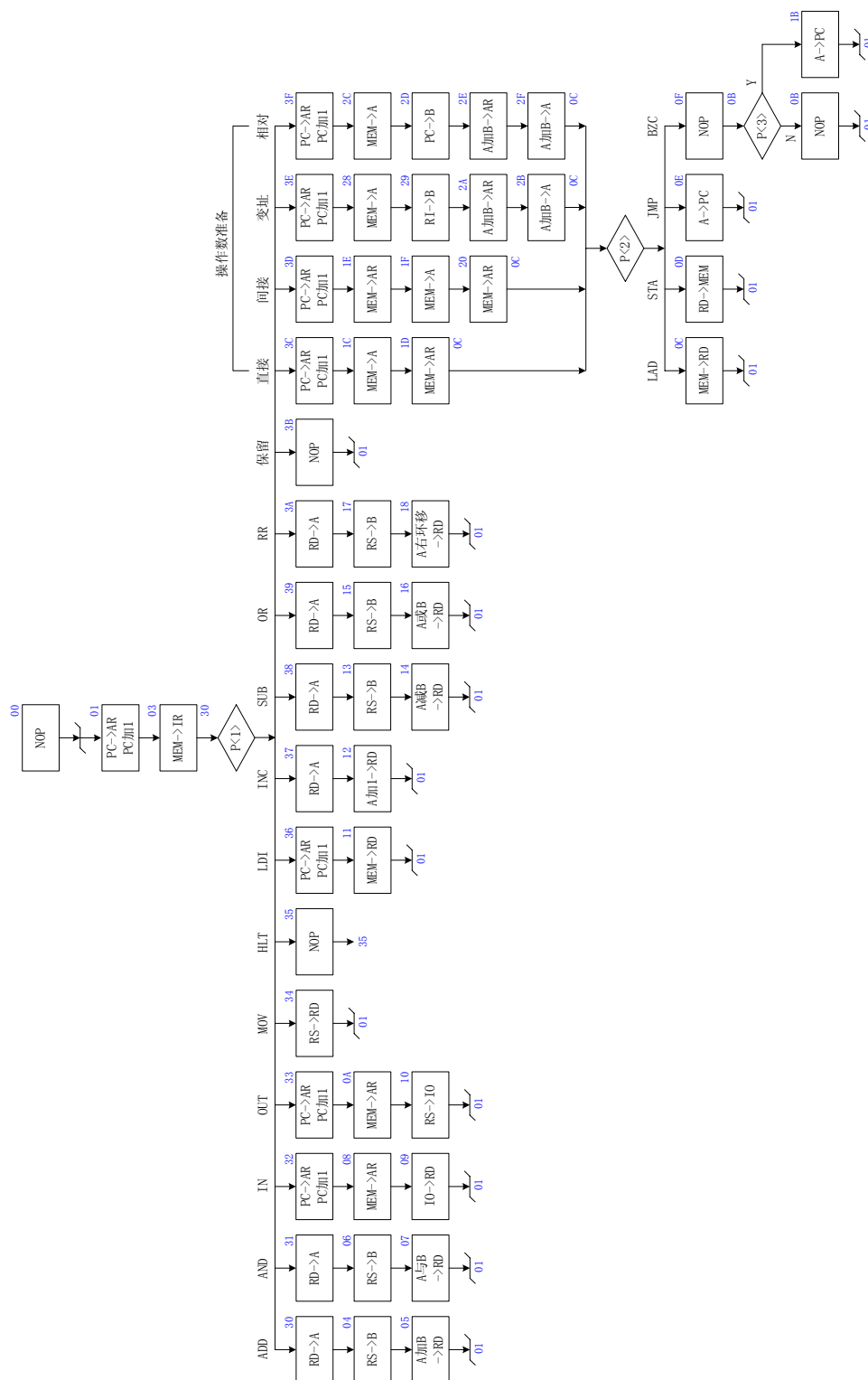


图 5-3-5 微程序流程图

表 5-3-5 二进制代码表

地址	十六进制表示	高五位	S3-S0	A 字段	B 字段	C 字段	UA5-UA0
00	00 00 01	00000	0000	000	000	000	000001
01	00 6D 43	00000	0000	110	110	101	000011
03	10 70 70	00010	0000	111	000	001	110000
04	00 24 05	00000	0000	010	011	000	000101
05	04 B2 01	00000	1001	011	001	000	000001
06	00 24 07	00000	0000	010	011	000	000111
07	01 32 01	00000	0010	011	001	000	000001
08	10 60 09	00010	0000	110	000	000	001001
09	18 30 01	00011	0000	011	000	000	000001
0A	10 60 10	00010	0000	110	000	000	010000
0B	00 00 01	00000	0000	000	000	000	000001
0C	10 30 01	00010	0000	011	000	000	000001
0D	20 06 01	00100	0000	000	001	100	000001
0E	00 53 41	00000	0000	101	001	101	000001
0F	00 00 CB	00000	0000	000	000	011	001011
10	28 04 01	00101	0000	000	010	000	000001
11	10 30 01	00010	0000	011	000	000	000001
12	06 B2 01	00000	1101	011	001	000	000001
13	00 24 14	00000	0000	010	011	000	010100
14	05 B2 01	00000	1011	011	001	000	000001
15	00 24 16	00000	0000	010	011	000	010110
16	01 B2 01	00000	0011	011	001	000	000001
17	00 24 18	00000	0000	010	011	000	011000
18	02 B2 01	00000	0101	011	001	000	000001
1B	00 53 41	00000	0000	101	001	101	000001
1C	10 10 1D	00010	0000	001	000	000	011101
1D	10 60 8C	00010	0000	110	000	010	001100
1E	10 60 1F	00010	0000	110	000	000	011111
1F	10 10 20	00010	0000	001	000	000	100000
20	10 60 8C	00010	0000	110	000	010	001100
28	10 10 29	00010	0000	001	000	000	101001
29	00 28 2A	00000	0000	010	100	000	101010
2A	04 E2 2B	00000	1001	110	001	000	101011
2B	04 92 8C	00000	1001	001	001	010	001100
2C	10 10 2D	00010	0000	001	000	000	101101

2D	00 2C 2E	00000	0000	010	110	000	101110
2E	04 E2 2F	00000	1001	110	001	000	101111
2F	04 92 8C	00000	1001	001	001	010	001100
30	00 16 04	00000	0000	001	011	000	000100
31	00 16 06	00000	0000	001	011	000	000110
32	00 6D 48	00000	0000	110	110	101	001000
33	00 6D 4A	00000	0000	110	110	101	001010
34	00 34 01	00000	0000	011	010	000	000001
35	00 00 35	00000	0000	000	000	000	110101
36	00 6D 51	00000	0000	110	110	101	010001
37	00 16 12	00000	0000	001	011	000	010010
38	00 16 13	00000	0000	001	011	000	010011
39	00 16 15	00000	0000	001	011	000	010101
3A	00 16 17	00000	0000	001	011	000	010111
3B	00 00 01	00000	0000	000	000	000	000001
3C	00 6D 5C	00000	0000	110	110	101	011100
3D	00 6D 5E	00000	0000	110	110	101	011110
3E	00 6D 68	00000	0000	110	110	101	101000
3F	00 6D 6C	00000	0000	110	110	101	101100

根据现有指令，在模型机上实现以下运算：从 IN 单元读入一个数据，根据读入数据的低 4 位值 X，求  $1+2+\dots+X$  的累加和，01H 到 0FH 共 15 个数据存于 60H 到 6EH 单元。

根据要求可以得到如下程序，地址和内容均为二进制数。

地 址	内 容	助记符	说 明
00000000	00100000	; START: IN R0, 00H	从 IN 单元读入计数初值
00000001	00000000		
00000010	01100001	; LDI R1, 0FH	立即数 0FH 送 R1
00000011	00001111		
00000100	00010100	; AND R0, R1	得到 R0 低四位
00000101	01100001	; LDI R1, 00H	装入和初值 00H
00000110	00000000		
00000111	11110000	; BZC RESULT	计数值为 0 则跳转
00001000	00010110		
00001001	01100010	; LDI R2, 60H	读入数据始地址
00001010	01100000		
00001011	11001011	; LOOP: LAD R3, [R1], 00H	从 MEM 读入数据送 R3， 变址寻址，偏移量为 00H
00001100	00000000		
00001101	00001101	; ADD R1, R3	累加求和
00001110	01110010	; INC RI	变址寄存加 1，指向下一数据
00001111	01100011	; LDI R3, 01H	装入比较值
00010000	00000001		
00010001	10001100	; SUB R0, R3	

00010010	11110000	; BZC RESULT	相减为 0, 表示求和完毕
00010011	00010110		
00010100	11100000	; JMP LOOP	未完则继续
00010101	00001011		
00010110	11010001	; RESULT: STA 70H, R1	和存于 MEM 的 70H 单元
00010111	01110000		
00011000	00110100	; OUT 40H, R1	和在 OUT 单元显示
00011001	01000000		
00011010	11100000	; JMP START	跳转至 START
00011011	00000000		
00011100	01010000	; HLT	停机
01100000	00000001	; 数据	
01100001	00000010		
01100010	00000011		
01100011	00000100		
01100100	00000101		
01100101	00000110		
01100110	00000111		
01100111	00001000		
01101000	00001001		
01101001	00001010		
01101010	00001011		
01101011	00001100		
01101100	00001101		
01101101	00001110		
01101110	00001111		

### 5.3.5 实验步骤

1. 按图 5-3-6 连接实验线路, 仔细检查接线后打开实验箱电源。
2. 写入实验程序, 并进行校验, 分两种方式, 手动写入和联机写入。

#### 1) 手动写入和校验

##### (1) 手动写入微程序

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

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

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

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

⑤ 重复①、②、③、④四步, 将表 5-3-5 的微代码写入 2816 芯片中。

##### (2) 手动校验微程序

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

② 使用 CON 单元的 SD05——SD00 给出微地址, 连续两次按动时序与操作台的开关 ST, MC

单元的指数数据指示灯 M7——M0 显示该单元的低 8 位。

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

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

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

(5) 手动写入机器程序

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

② 使用 CON 单元的 SD7——SD0 给出地址，IN 单元给出该单元应写入的数据，连续两次按动时序与操作台的开关 ST，将 IN 单元的数据写到该存储器单元。

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

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

⑤ 亦可重复①、②两步，将所有机器指令写入主存芯片中。

(6) 手动校验机器程序

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

② 使用 CON 单元的 SD7——SD0 给出地址，连续两次按动时序与操作台的开关 ST，CPU 内总线的指数数据指示灯 D7——D0 显示该单元的数据。

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

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

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

2) 联机写入和校验

联机软件提供了微程序和机器程序下载功能，以代替手动读写微程序和机器程序，但是微程序和机器程序得以指定的格式写入到以 TXT 为后缀的文件中，本次实验程序如下，程序中分号‘;’为注释符，分号后面的内容在下载时将被忽略掉。

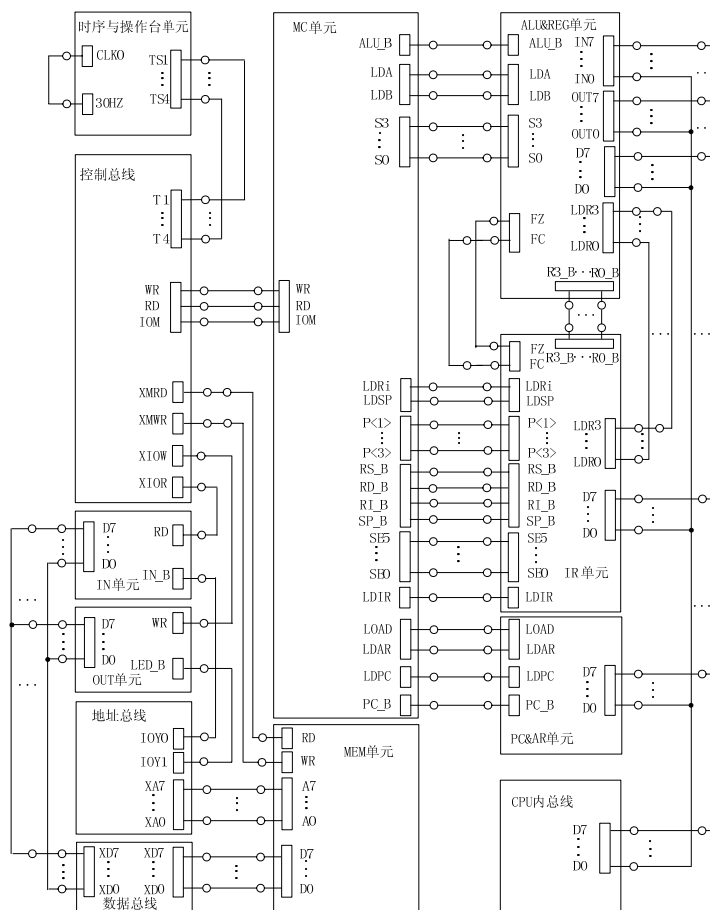


图 5-3-6 实验接线图

```

; //*****
; //
; //      复杂模型机实验指令文件
; //
; //      By TangDu CO.,LTD
; //
; //*****

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

$P 60 01 ; 数据
$P 61 02
$P 62 03
$P 63 04
$P 64 05
$P 65 06
$P 66 07
$P 67 08
$P 68 09
$P 69 0A
$P 6A 0B

```

```

$P 6B 0C
$P 6C 0D
$P 6D 0E
$P 6E 0F
; //***** End Of Main Memory Data *****//

; /** Start Of MicroController Data **//
$M 00 000001 ; NOP
$M 01 006D43 ; PC->AR, PC 加 1
$M 03 107070 ; MEM->IR, P<1>
$M 04 002405 ; RS->B
$M 05 04B201 ; A 加 B->RD
$M 06 002407 ; RS->B
$M 07 013201 ; A 与 B->RD
$M 08 106009 ; MEM->AR
$M 09 183001 ; IO->RD
$M 0A 106010 ; MEM->AR
$M 0B 000001 ; NOP
$M 0C 103001 ; MEM->RD
$M 0D 200601 ; RD->MEM
$M 0E 005341 ; A->PC
$M 0F 0000CB ; NOP, P<3>
$M 10 280401 ; RS->IO
$M 11 103001 ; MEM->RD
$M 12 06B201 ; A 加 1->RD
$M 13 002414 ; RS->B
$M 14 05B201 ; A 减 B->RD
$M 15 002416 ; RS->B
$M 16 01B201 ; A 或 B->RD
$M 17 002418 ; RS->B
$M 18 02B201 ; A 右环移->RD
$M 1B 005341 ; A->PC
$M 1C 10101D ; MEM->A
$M 1D 10608C ; MEM->AR, P<2>
$M 1E 10601F ; MEM->AR
$M 1F 101020 ; MEM->A
$M 20 10608C ; MEM->AR, P<2>
$M 28 101029 ; MEM->A
$M 29 00282A ; RI->B
$M 2A 04E22B ; A 加 B->AR
$M 2B 04928C ; A 加 B->A, P<2>
$M 2C 10102D ; MEM->A
$M 2D 002C2E ; PC->B
$M 2E 04E22F ; A 加 B->AR
$M 2F 04928C ; A 加 B->A, P<2>
$M 30 001604 ; RD->A
$M 31 001606 ; RD->A
$M 32 006D48 ; PC->AR, PC 加 1
$M 33 006D4A ; PC->AR, PC 加 1
$M 34 003401 ; RS->RD
$M 35 000035 ; NOP
$M 36 006D51 ; PC->AR, PC 加 1

```



```
$M 37 001612    ; RD->A
$M 38 001613    ; RD->A
$M 39 001615    ; RD->A
$M 3A 001617    ; RD->A
$M 3B 000001    ; NOP
$M 3C 006D5C    ; PC->AR, PC 加 1
$M 3D 006D5E    ; PC->AR, PC 加 1
$M 3E 006D68    ; PC->AR, PC 加 1
$M 3F 006D6C    ; PC->AR, PC 加 1
; /** End Of MicroController Data **//
```

选择联机软件的“【转储】—【装载】”功能，在打开文件对话框中选择上面所保存的文件，软件自动将机器程序和微程序写入指定单元。

选择联机软件的“【转储】—【刷新指令区】”可以读出下位机所有的机器指令和微指令，并在指令区显示，对照文件检查微程序和机器程序是否正确，如果不正确，则说明写入操作失败，应重新写入，可以通过联机软件单独修改某个单元的指令，以修改微指令为例，先用鼠标左键单击指令区的‘微存’TAB按钮，然后再单击需修改单元的数据，此时该单元变为编辑框，输入6位数据并回车，编辑框消失，并以红色显示写入的数据。

### 3. 运行程序

#### 方法一：本机运行

将时序与操作台单元的开关KK1、KK3置为‘运行’档，按动CON单元的总清按钮CLR，将使程序计数器PC、地址寄存器AR和微程序地址为00H，程序可以从头开始运行，暂存器A、B，指令寄存器IR和OUT单元也会被清零。

将时序与操作台单元的开关KK2置为‘单步’档，每按动一次ST按钮，即可单步运行一条微指令，对照微程序流程图，观察微地址显示灯是否和流程一致。每运行完一条微指令，观测一次数据总线和地址总线，对照数据通路图，分析总线上的数据是否正确。

当模型机执行完OUT指令后，检查OUT单元显示的数是否正确，按下CON单元的总清按钮CLR，改变IN单元的值，再次执行机器程序，从OUT单元显示的数判别程序执行是否正确。

#### 方法二：联机运行（软件使用说明请看附录1）

进入软件界面，选择菜单命令“【实验】—【CISC实验】”，打开相应的数据通路图，选择相应的功能命令，即可联机运行、监控、调试程序。

按动CON单元的总清按钮CLR，然后通过软件运行程序，当模型机执行完OUT指令后，检查OUT单元显示的数是否正确。在数据通路图和微程序流中观测指令的执行过程，并观测软件中地址总线、数据总线以及微指令显示和下位机是否一致。

## 第 6 章 输入输出系统

计算机的输入输出系统也称为 I/O 系统，包括外围设备、设备控制器、I/O 接口以及一些专门为输入输出操作而设计的软件和硬件。

本章要在前面模型机的基础上，对 I/O 接口进行扩展，丰富模型机的功能。为此安排了三个实验：带中断处理能力的模型机设计实验，带 DMA 控制功能的模型机设计实验和典型 I/O 接口 8253 扩展实验。

### 6.1 带中断处理能力的模型机设计实验

#### 6.1.1 实验目的

- (1) 掌握中断原理及其响应流程。
- (2) 掌握 8259 中断控制器原理及其应用编程。

#### 6.1.2 实验设备

PC 机一台，TD-CMX 实验系统一套。

#### 6.1.3 实验原理

8259 的引脚分配图如图 6-1-1 所示。

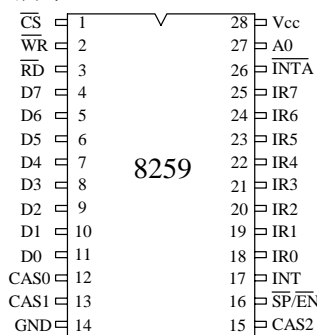


图 6-1-1 8259 芯片引脚说明

8259 芯片引脚说明如下。

- D7~D0 为双向三态数据线
- $\overline{CS}$  片选信号线
- A0 用来选择芯片内部不同的寄存器，通常接至地址总线的 A0。
- $\overline{RD}$  读信号线，低电平有效，其有效时控制信息从 8259 读至 CPU。
- $\overline{WR}$  写信号线，低电平有效，其有效时控制信息从 CPU 写入至 8259。
- $\overline{SP/EN}$  从程序/允许缓冲
- $\overline{INTA}$  中断响应输入
- INT 中断输出
- IR0~IR7 8 条外界中断请求输入线。
- CAS2~CAS0 级连信号线。

$\overline{CS}$ 、A0、 $\overline{RD}$ 、 $\overline{WR}$ 、D4、D3 位的电平与 8259 操作关系如表 6-1-1 所示。

CPU 必须有一个中断使能寄存器，并且可以通过指令对该寄存器进行操作，其原理如图 6-1-2 所示。CPU 开中断指令 STI 对其置 1，而 CPU 关中断指令 CLI 对其置 0。

表 6-1-1 8259A 的读/写操作

A0	D4	D3	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	操 作
0			0	1	0	输入操作(读) IRR, ISR或中断级别 → 数据总线
1			0	1	0	IMR 数据总线
0	0	0	1	0	0	输出操作(写) 数据总线 → OCW2
0	0	1	1	0	0	数据总线 → OCW3
0	1	X	1	0	0	数据总线 → OCW1
1	X	X	1	0	0	数据总线 → ICW1, ICW2, ICW3, ICW4
X	X	X	1	1	0	断开功能 数据总线 → 三态(无操作)
X	X	X	X	X	1	数据总线 → 三态(无操作)

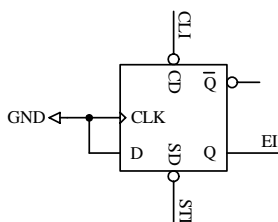


图 6-1-2 中断使能寄存器原理图

8259 的数据线 D7...D0 挂接到数据总线，地址线 A0 挂接到地址总线的 A0 上，片选信号  $\overline{CS}$  接控制总线的 IOY3，IOY3 由地址总线的高 2 位译码产生，其地址分配见表 6-1-2， $\overline{RD}$ 、 $\overline{WR}$ （实验箱上丝印为 XIOR 和 XIOW）接 CPU 给出的读写信号，8259 和系统的连接如图 5-2-3 所示。

表 6-1-2 I/O 地址空间分配

A7 A6	选定	地址空间
00	IOY0	00-3F
01	IOY1	40-7F
10	IOY2	80-BF
11	IOY3	C0-FF

本实验要求设计的模型计算机具备有类 X86 的中断功能，当外部中断请求有效、CPU 允许中断，且在一条指令执行完时，CPU 将响应中断。当 CPU 响应中断时，将会向 8259 发送两个

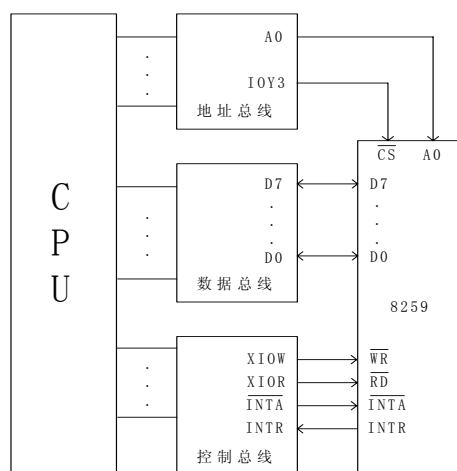


图 6-1-3 8259 和 CPU 挂接图

连续的  $\overline{\text{INTA}}$  信号，请注意，8259 是在接收到第一个  $\overline{\text{INTA}}$  信号后锁住向 CPU 的中断请求信号 INTR（高电平有效），并且在第二个  $\overline{\text{INTA}}$  信号到达后将其变为低电平（自动 EOI 方式），所以，中断请求信号 IR0 应该维持一段时间，直到 CPU 发送出第一个  $\overline{\text{INTA}}$  信号，这才是一个有效的中断请求。8259 在收到第二个  $\overline{\text{INTA}}$  信号后，就会将中断向量号发送到数据总线，CPU 读取中断向量号，并转入相应的中断处理程序中。

本系统的指令译码电路是在 IR 单元的 INS\_DEC (GAL20V8) 中实现，如图 6-1-4 所示。和前面复杂模型机实验指令译码电路相比，主要增加了对中断的支持，当 INTR（有中断请求）和 EI（CPU 允许中断）均为 1，且  $P<4$  测试有效，那么在 T4 节拍时，微程式序就会产生中断响应分支，从而使得 CPU 能响应中断。

在中断过程中需要有现场保护，而且在编程的过程中也需要一些压栈或弹栈操作，所以还需设置一个堆栈，由 R3 做堆栈指针。系统的寄存器译码电路如图 6-1-5 所示，在 IR 单元的 REG\_DEC (GAL16V8) 中实现，和前面复杂模型机实验寄存器译码电路相比，增加一个或门和一个与门，用以支持堆栈操作。

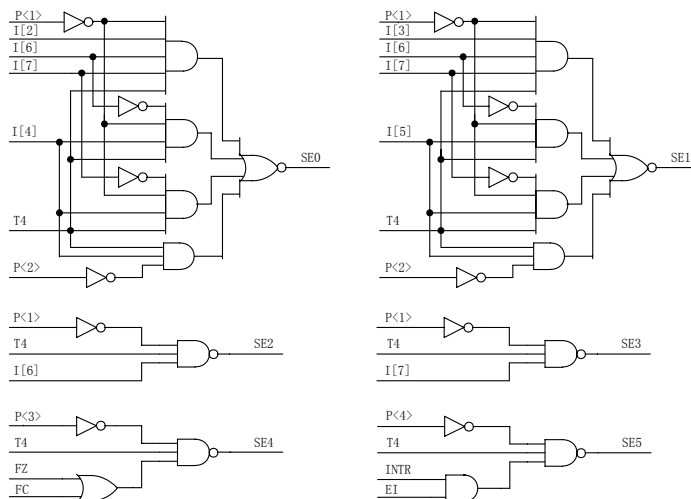


图 6-1-4 指令译码原理图

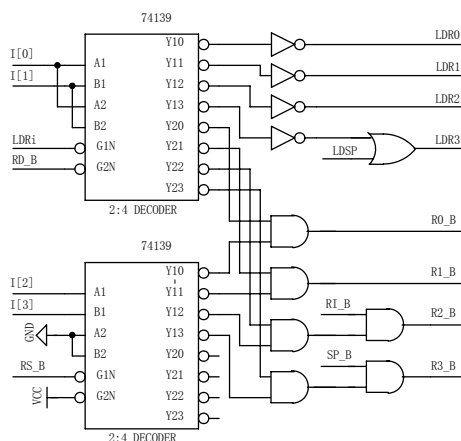


图 6-1-5 寄存器译码原理图

本模型机共设计 16 条指令, 表 6-1-3 列出了基本指令的格式、助记符及其功能。其中, D 为立即数, P 为外设的端口地址, RS 为源寄存器, RD 为目的寄存器, 并规定如下。

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

设定微指令格式，如表 6-1-4 所示。

根据指令系统要求，设计微程序流程及确定微地址，并得到微程序流程图，如图 6-1-6 所示。

表 6-1-3 指令助记符、格式及功能

助记符号	指令格式	指令功能
MOV RD, RS	0100 RS RD	RS $\rightarrow$ RD
ADD RD, RS	0000 RS RD	RD + RS $\rightarrow$ RD
AND RD, RS	0001 RS RD	RD $\wedge$ RS $\rightarrow$ RD
STI	0111 ** **	CPU开中断
CLI	1000 ** **	CPU关中断
PUSH RS	1001 RS **	RS $\rightarrow$ 堆栈
POP RD	1010 ** RD	堆栈 $\rightarrow$ RD
IRET	1011 ** **	中断返回
LAD M D, RD	1100 M RD D	E $\rightarrow$ RD
STA M D, RS	1101 M RD D	RD $\rightarrow$ E
JMP M D	1110 M ** D	E $\rightarrow$ PC
BZC M D	1111 M ** D	当FC或FZ=1时, E $\rightarrow$ PC
IN RD, P	0010 ** RD P	[P] $\rightarrow$ RD
OUT P, RS	0011 RS ** P	RS $\rightarrow$ [P]
LDI RD, D	0110 ** RD D	D $\rightarrow$ RD
HALT	0101 ** **	停机

表 6-1-4 微指令格式

23	22	21	20	19	18-15	14-12	11-9	8-6	5-0
M23	INTA	WR	RD	IOM	S3-S0	A字段	B字段	C字段	MA5-MA0

A字段

14	13	12	选择
0	0	0	NOP
0	0	1	LDA
0	1	0	LDB
0	1	1	LDRi
1	0	0	LDSP
1	0	1	LOAD
1	1	0	LDAR
1	1	1	LDIR

B字段

11	10	9	选择
0	0	0	NOP
0	0	1	ALU_B
0	1	0	RS_B
0	1	1	RD_B
1	0	0	RI_B
1	0	1	SP_B
1	1	0	PC_B
1	1	1	保留

C字段

8	7	6	选择
0	0	0	NOP
0	0	1	P<1>
0	1	0	P<2>
0	1	1	P<3>
1	0	0	P<4>
1	0	1	LDPC
1	1	0	STI
1	1	1	CLI

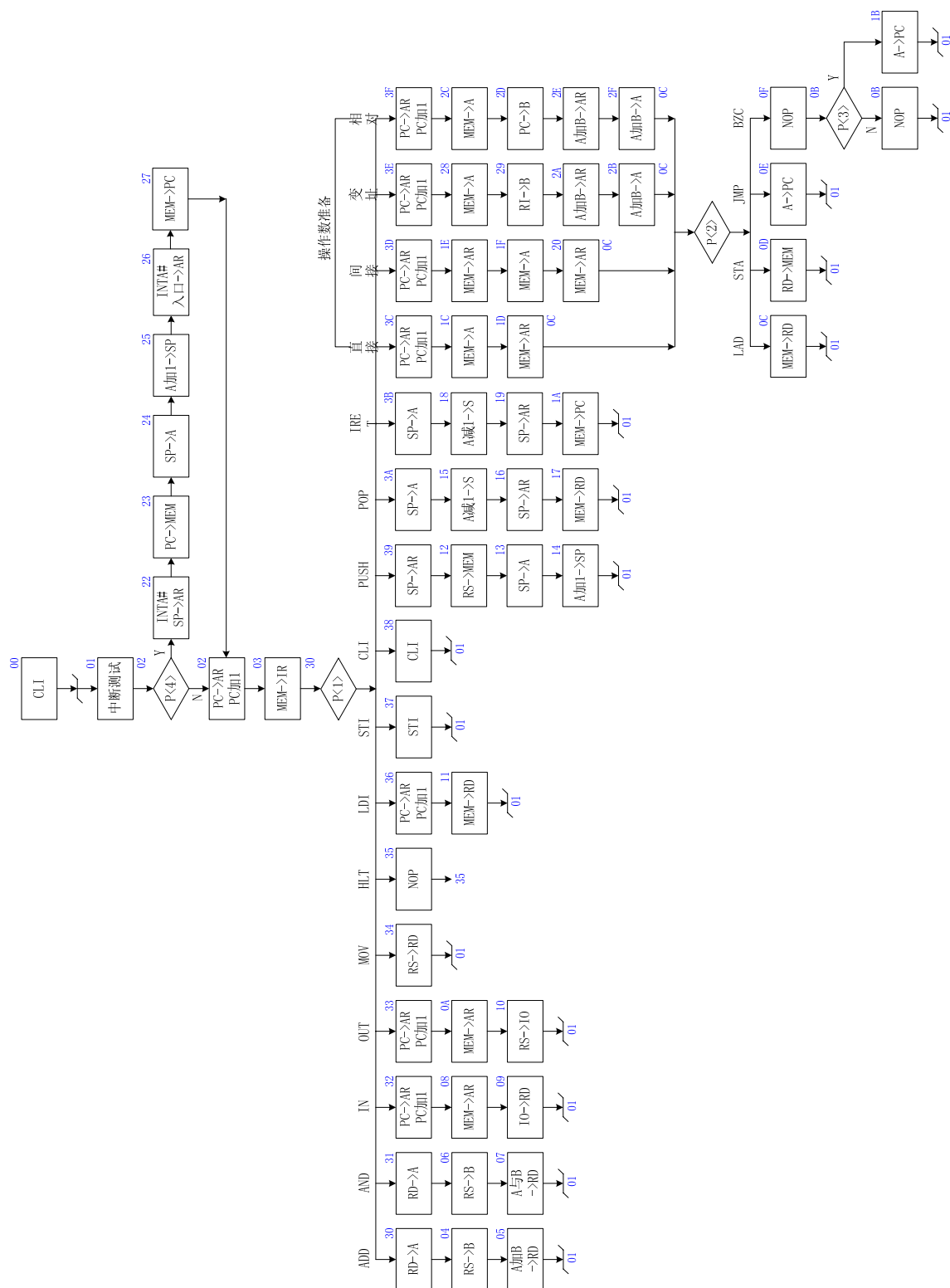


图 6-1-6 微程序流程图

参照微程序流程图，将每条微指令代码化，译成二进制微代码表，如表 6-1-5 所示。

表 6-1-5 二进制微代码表

地址	十六进制表示	高五位	S3-S0	A 字段	B 字段	C 字段	UA5~uA0
00	00 01 C1	00000	0000	000	000	111	000001
01	00 01 02	00000	0000	000	000	100	000010
02	00 6D 43	00000	0000	110	110	101	000011
03	10 70 70	00010	0000	111	000	001	110000
04	00 24 05	00000	0000	010	011	000	000101
05	04 B2 01	00000	1001	011	001	000	000001
06	00 24 07	00000	0000	010	011	000	000111
07	01 32 01	00000	0010	011	001	000	000001
08	10 60 09	00010	0000	110	000	000	001001
09	18 30 01	00011	0000	011	000	000	000001
0A	10 60 10	00010	0000	110	000	000	010000
0B	00 00 01	00000	0000	000	000	000	000001
0C	10 30 01	00010	0000	011	000	000	000001
0D	20 06 01	00100	0000	000	001	100	000001
0E	00 53 41	00000	0000	101	001	101	000001
0F	00 00 CB	00000	0000	000	000	011	001011
10	28 04 01	00101	0000	000	010	000	000001
11	10 30 01	00010	0000	011	000	000	000001
12	20 04 13	00100	0000	000	010	000	010011
13	00 1A 14	00000	0000	001	101	000	010100
14	06 C2 01	00000	1101	100	001	000	000001
15	06 42 16	00000	1100	100	001	000	010110
16	00 6A 17	00000	0000	110	101	000	010111
17	10 30 01	00010	0000	011	000	000	000001
18	06 42 19	00000	1100	100	001	000	011001
19	00 6A 1A	00000	0000	110	101	000	011010
1A	10 51 41	00010	0000	101	000	101	000001
1B	00 53 41	00000	0000	101	001	101	000001
1C	10 10 1D	00010	0000	001	000	000	011101
1D	10 60 8C	00010	0000	110	000	010	001100
1E	10 60 1F	00010	0000	110	000	000	011111
1F	10 10 20	00010	0000	001	000	000	100000
20	10 60 8C	00010	0000	110	000	010	001100
22	40 6A 23	01000	0000	110	101	000	100011



23	20 0C 24	00100	0000	000	110	000	100100
24	00 1A 25	00000	0000	001	101	000	100101
25	06 C2 26	00000	1101	100	001	000	100110
26	40 60 27	01000	0000	110	000	000	100111
27	10 51 42	00010	0000	101	000	101	000010
28	10 10 29	00010	0000	001	000	000	101001
29	00 28 2A	00000	0000	010	100	000	101010
2A	04 E2 2B	00000	1001	110	001	000	101011
2B	04 92 8C	00000	1001	001	001	010	001100
2C	10 10 2D	00010	0000	001	000	000	101101
2D	00 2C 2E	00000	0000	010	110	000	101110
2E	04 E2 2F	00000	1001	110	001	000	101111
2F	04 92 8C	00000	1001	001	001	010	001100
30	00 16 04	00000	0000	001	010	000	000100
31	00 16 06	00000	0000	001	010	000	000110
32	00 6D 48	00000	0000	110	110	101	001000
33	00 6D 4A	00000	0000	110	110	101	001010
34	00 34 01	00000	0000	011	010	000	000001
35	00 00 35	00000	0000	000	000	000	110101
36	00 6D 51	00000	0000	110	110	101	010001
37	00 01 81	00000	0000	000	000	110	000001
38	00 01 C1	00000	0000	000	000	111	000001
39	00 6A 12	00000	0000	110	101	000	010010
3A	00 1A 15	00000	0000	001	101	000	010101
3B	00 1A 18	00000	0000	001	101	000	011000
3C	00 6D 5C	00000	0000	110	110	101	011100
3D	00 6D 5E	00000	0000	110	110	101	011110
3E	00 6D 68	00000	0000	110	110	101	101000
3F	00 6D 6C	00000	0000	110	110	101	101100

根据现有指令，编写一段程序，在模型机上实现以下功能：从 IN 单元读入一个数据 X 存于寄存器 R0，CPU 每响应一次中断，对 R0 中的数据加 1，并输出到 OUT 单元。

根据要求可以得到如下程序，地址和内容均为二进制数。

地 址	内 容	助记符	说 明
00000000	01100000 ; LDI	R0, 13H	将立即数 13 装入 R0
00000001	00010011		
00000010	00110000 ; OUT	COH, R0	将 R0 中的内容写入端口 C0 中，即写
00000011	11000000 ;		ICW1，边沿触发，单片模式，需 ICW4
00000100	01100000 ; LDI	R0, 30H	将立即数 30 装入 R0

```

00000101 00110000
00000110 00110000 ; OUT C1H, R0      将 R0 中的内容写入端口 C1 中, 即写
00000111 11000001 ;                  ICW2, 中断向量为 30-37
00001000 01100000 ; LDI R0, 03H      将立即数 03 装入 R0
00001001 00000011
00001010 00110000 ; OUT C1H, R0      将 R0 中的内容写入端口 C1 中, 即写
00001011 11000001 ;                  ICW4, 非缓冲, 86 模式, 自动 EOI
00001100 01100000 ; LDI R0, FEH      将立即数 FE 装入 R0
00001101 11111110
00001110 00110000 ; OUT C1H, R0      将 R0 中的内容写入端口 C1 中, 即写
00001111 11000001 ;                  OCW1, 只允许 IRO 请求
00010000 01100011 ; LDI SP, A0H      初始化堆栈指针为 A0
00010001 10100000
00010010 01110000 ; STI              CPU 开中断
00010011 00100000 ; IN R0, 00H       从端口 00 (IN 单元) 读入计数初值
00010100 00000000
00010101 01000001 ; LOOP: MOV R1, R0  移动数据, 并等待中断
00010110 11100000 ; JMP LOOP        跳转, 并等待中断
00010111 00010101

```

; 以下为中断服务程序:

```

00100000 0000000080 ; CLI              CPU 关中断
00100001 0000000061 ; LDI R1, 01H      将立即数 01 装入 R1
00100010 0000000001
00100011 0000000004 ; ADD R0, R1      将 R0 和 R1 相加, 即计数值加 1
00100100 0000000030 ; OUT 40H, R0      将计数值输出到端口 40 (OUT 单元)
00100101 0000000040
00100110 0000000070 ; STI              CPU 开中断
00100111 00000000B0 ; IRET            中断返回
00110000 0000000020 ;                  IRO 的中断入口地址 20

```

#### 6.1.4 实验步骤

- 按图 6-1-7 所示连接实验接线, 仔细检查接线后打开实验箱电源。
- 写入实验程序, 并进行校验, 分两种方式, 手动写入和联机写入。

##### 1) 手动写入和校验

##### (1) 手动写入微程序

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

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

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

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

⑤ 重复①、②、③、④四步, 将表 6-1-5 的微代码写入 2816 芯片中。

##### (2) 手动校验微程序

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

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

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

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

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

#### (7) 手动写入机器程序

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

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

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

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

⑤ 亦可重复①、②两步，将所有机器指令写入主存芯片中。

#### (8) 手动校验机器程序

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

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

③ 将时序与操作台单元的开关 KK5 置为‘加 1’档。

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

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

#### 2) 联机写入和校验

联机软件提供了微程序和机器程序下载功能，以代替手动读写微程序和机器程序，但是微程序和机器程序得以指定的格式写入到以 TXT 为后缀的文件中，本次实验程序如下，程序中分号‘；’为注释符，分号后面的内容在下载时将被忽略掉：

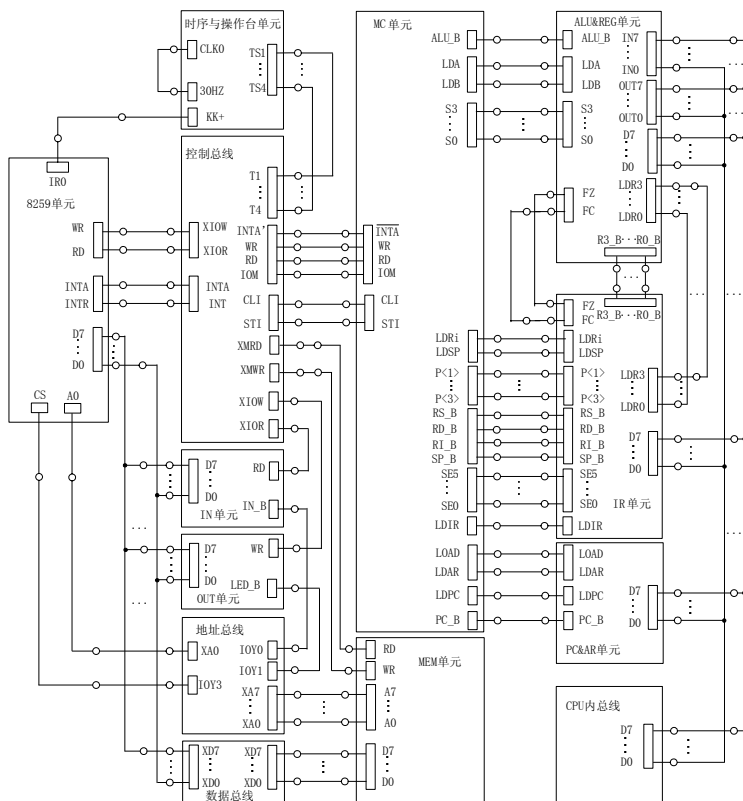


图 6-1-7 实验接线图

```

; //*****//
; //
; // 带中断处理能力的模型机实验指令文件 //
; // //
; // By TangDu CO.,LTD //
; // //
; //*****//

; //**** Start Of Main Memory Data ****//
$P 00 60 ; LDI R0,13H 将立即数 13 装入 R0
$P 01 13
$P 02 30 ; OUT C0H,R0 将 R0 中的内容写入端口 C0 中,即写
$P 03 C0 ; ICW1,边沿触发,单片模式,需要 ICW4
$P 04 60 ; LDI R0,30H 将立即数 30 装入 R0
$P 05 30
$P 06 30 ; OUT C1H,R0 将 R0 中的内容写入端口 C1 中,即写
$P 07 C1 ; ICW2,中断向量为 30-37
$P 08 60 ; LDI R0,03H 将立即数 03 装入 R0
$P 09 03
$P 0A 30 ; OUT C1H,R0 将 R0 中的内容写入端口 C1 中,即写
$P 0B C1 ; ICW4,非缓冲,86 模式,自动 EOI
$P 0C 60 ; LDI R0,FEH 将立即数 FE 装入 R0
    
```

```

$P 0D FE
$P 0E 30 ; OUT C1H, R0      将 R0 中的内容写入端口 C1 中，即写
$P 0F C1 ;                  0CW1，只允许 IRO 请求
$P 10 63 ; LDI SP, A0H      初始化堆栈指针为 A0
$P 11 A0
$P 12 70 ; STI              CPU 开中断
$P 13 20 ; IN R0, 00H       从端口 00 (IN 单元) 读入计数初值
$P 14 00
$P 15 41 ; LOOP: MOV R1, R0  移动数据，并等待中断
$P 16 E0 ; JMP LOOP         跳转，并等待中断
$P 17 15

; 以下为中断服务程序：
$P 20 80 ; CLI              CPU 关中断
$P 21 61 ; LDI R1, 01H      将立即数 01 装入 R1
$P 22 01
$P 23 04 ; ADD R0, R1       将 R0 和 R1 相加，即计数值加 1
$P 24 30 ; OUT 40H, R0      将计数值输出到端口 40 (OUT 单元)
$P 25 40
$P 26 70 ; STI              CPU 开中断
$P 27 B0 ; IRET             中断返回
$P 30 20 ;                  IRO 的中断入口地址 20
; //***** End Of Main Memory Data *****/

; /** Start Of MicroController Data **//
$M 00 0001C1 ; NOP
$M 01 000102 ; 中断测试, P<4>
$M 02 006D43 ; PC->AR, PC 加 1
$M 03 107070 ; MEM->IR, P<1>
$M 04 002405 ; RS->B
$M 05 04B201 ; A 加 B->RD
$M 06 002407 ; RS->B
$M 07 013201 ; A 与 B->RD
$M 08 106009 ; MEM->AR
$M 09 183001 ; IO->RD
$M 0A 106010 ; MEM->AR
$M 0B 000001 ; NOP
$M 0C 103001 ; MEM->RD
$M 0D 200601 ; RD->MEM
$M 0E 005341 ; A->PC
$M 0F 0000CB ; NOP, P<3>
$M 10 280401 ; RS->IO
$M 11 103001 ; MEM->RD
$M 12 200413 ; RS->MEM
$M 13 001A14 ; SP->A
$M 14 06C201 ; A 加 1->SP
$M 15 064216 ; A 减 1->SP
$M 16 006A17 ; SP->AR
$M 17 103001 ; MEM->RD
$M 18 064219 ; A 减 1->SP
$M 19 006A1A ; SP->AR
$M 1A 105141 ; MEM->PC

```

```
$M 1B 005341 ; A->PC
$M 1C 10101D ; MEM->A
$M 1D 10608C ; MEM->AR, P<2>
$M 1E 10601F ; MEM->AR
$M 1F 101020 ; MEM->A
$M 20 10608C ; MEM->AR, P<2>
$M 22 406A23 ; INTA#, SP->AR
$M 23 200C24 ; PC->MEM
$M 24 001A25 ; SP->A
$M 25 06C226 ; A 加 1->SP
$M 26 406027 ; INTA#, 入口->AR
$M 27 105142 ; MEM->PC
$M 28 101029 ; MEM->A
$M 29 00282A ; RI->B
$M 2A 04E22B ; A 加 B->AR
$M 2B 04928C ; A 加 B->A, P<2>
$M 2C 10102D ; MEM->A
$M 2D 002C2E ; PC->B
$M 2E 04E22F ; A 加 B->AR
$M 2F 04928C ; A 加 B->A, P<2>
$M 30 001604 ; RD->A
$M 31 001606 ; RD->A
$M 32 006D48 ; PC->AR, PC 加 1
$M 33 006D4A ; PC->AR, PC 加 1
$M 34 003401 ; RS->RD
$M 35 000035 ; NOP
$M 36 006D51 ; PC->AR, PC 加 1
$M 37 000181 ; STI
$M 38 0001C1 ; CLI
$M 39 006A12 ; SP->AR
$M 3A 001A15 ; SP->A
$M 3B 001A18 ; SP->A
$M 3C 006D5C ; PC->AR, PC 加 1
$M 3D 006D5E ; PC->AR, PC 加 1
$M 3E 006D68 ; PC->AR, PC 加 1
$M 3F 006D6C ; PC->AR, PC 加 1
; /** End Of MicroController Data **//
```

选择联机软件的“【转储】—【装载】”功能，在打开文件对话框中选择上面所保存的文件，软件自动将机器程序和微程序写入指定单元。

选择联机软件的“【转储】—【刷新指令区】”可以读出下位机所有的机器指令和微指令，并在指令区显示，对照文件检查微程序和机器程序是否正确，如果不正确，则说明写入操作失败，应重新写入，可以通过联机软件单独修改某个单元的指令，以修改微指令为例，先用鼠标左键单击指令区的‘微存’TAB 按钮，然后再单击需修改单元的数据，此时该单元变为编辑框，输入 6 位数据并回车，编辑框消失，并以红色显示写入的数据。

### 3. 运行程序。

方法一：本机运行

将时序与操作台单元的开关 KK1、KK3 置为‘运行’档，按动 CON 单元的总清按钮 CLR，将使程序计数器 PC、地址寄存器 AR 和微程序地址为 00H，程序可以从头开始运行，暂存器 A、B，指令寄存器 IR 和 OUT 单元也会被清零。

将时序与操作台单元的开关 KK2 置为‘连续’档，按动一次 ST 按钮，即可连续运行指令，按动 KK 开关，每按动一次，检查 OUT 单元显示的数是否在原有基础加 1（第一次是在 IN 单元值的基础加 1）。

#### 方法二：联机运行

进入软件界面，选择菜单命令“【实验】—【CISC 实验】”，打开相应的数据通路图，选择相应的功能命令，即可联机运行、监控、调试程序。

按动 CON 单元的总清按钮 CLR，然后通过软件运行程序，在数据通路图和微程序流中观测程序的执行过程。在微程序流程图中观测：选择‘单周期’运行程序，当模型机执行完 MOV 指令后，按下 KK 开关，不要松开，可见控制总线 INTR 指示灯亮，继续‘单周期’运行程序，直到模型机的 CPU 向 8259 发送完第一个  $\overline{\text{INTA}}$ ，然后松开 KK 开关，INTR 中断请求被 8259 锁存，CPU 响应中断。仔细分析中断响应时现场保护的过程，中断返回时现场恢复的过程。

每响应一次中断，检查 OUT 单元显示的数是否在原有基础加 1（第一次是在 IN 单元值的基础加 1）。

## 6.2 带 DMA 控制功能的模型机设计实验

### 6.2.1 实验目的

- (1) 掌握 CPU 外扩接口芯片的方法。
- (2) 掌握 8237DMA 控制器原理及其应用编程。

### 6.2.2 实验设备

PC 机一台，TD-CMX 实验系统一套。

### 6.2.3 实验原理

#### 1. 8237 芯片简介

- (1) 8237 的引脚分配图如图 6-2-1 所示。

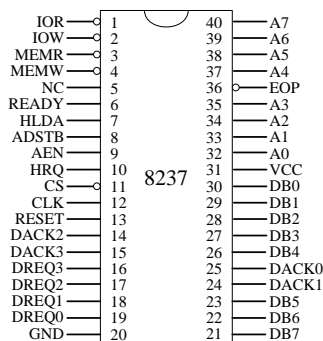


图 6-2-1 8237 引脚图

芯片引脚说明

- A0~A3 为双向地址线。
- A4~A7 为三态输出线。
- DB0~DB7 为双向三态数据线。
- $\overline{IOW}$  为双向三态低电平有效的 I/O 写控制信号。
- $\overline{IOR}$  为双向三态低电平有效的 I/O 读控制信号。
- $\overline{MEMW}$  为双向三态低电平有效的存储器写控制信号。
- $\overline{MEMR}$  为双向三态低电平有效的存储器读控制信号。
- ADSTB 为地址选通信号。
- AEN 为地址允许信号。
- $\overline{CS}$  为片选信号。
- RESET 为复位信号。
- READY 为准备好输入信号。
- HRQ 为保持请求信号。
- HLDA 保持响应信号。
- DREQ0~DREQ3 为 DMA 请求(通道 0~3)信号



- DACK0~DACK3 为 DMA 应答(通道 0~3) 信号
- CLK 为时钟输入
- $\overline{EOP}$  为过程结束命令线

(2) 8237 的内部结构图如图 6-2-2 所示。

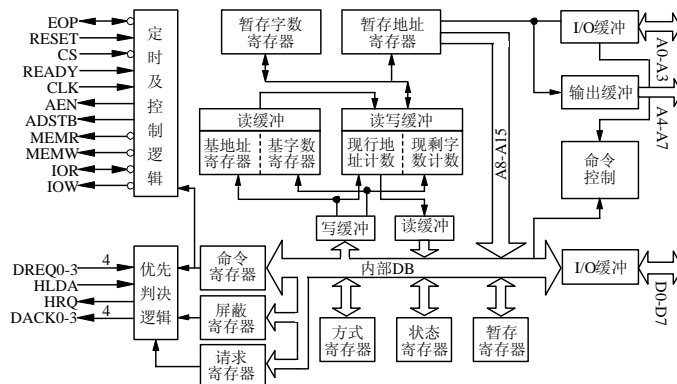
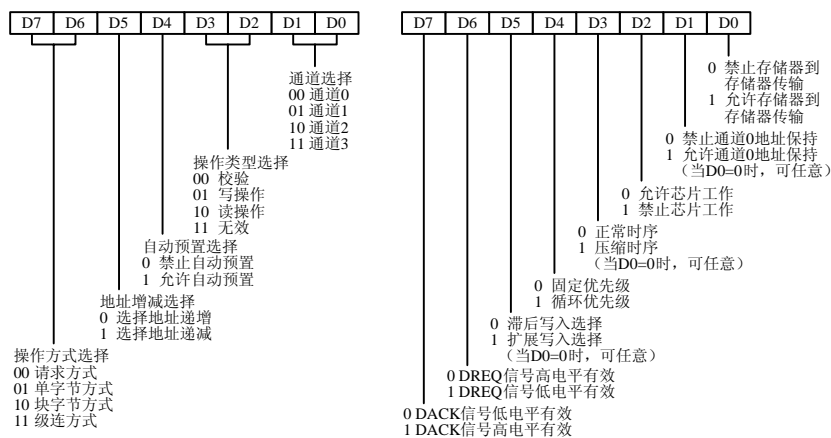


图 6-2-2 8237 内部结构图

(3) 8237 的寄存器定义如图 6-2-3 所示。



(a) 方式寄存器

(b) 命令寄存器

(c) 请求寄存器

(d) 单通道屏蔽寄存器

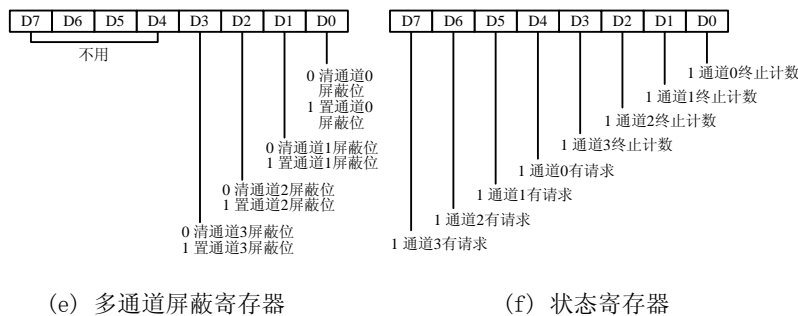


图 6-2-3 8237 的寄存器定义

(4) 8237 的初始化。

使用 DMA 控制器，必须对其进行初始化。8237 的初始化需要按一定的顺序对各寄存器进行写入，初始化顺序如下：

- (1) 写主清除命令。
- (2) 写地址寄存器。
- (3) 写字节计数寄存器。
- (4) 写工作方式寄存器。
- (5) 写命令寄存器。
- (6) 写屏蔽寄存器。
- (7) 写请求寄存器。

## 2. 8237 芯片外部连接

对于 CPU 外扩接口芯片，其重点是要设计接口芯片数据线、地址线和控制线与 CPU 的挂接，图 6-2-5 是 8237 接口芯片的典型扩展接法。这里的模型计算机可以直接应用前面的复杂模型机，其 I/O 地址空间分配情况如表 6-2-4 所示。

表 6-2-4 I/O 地址空间分配

A7 A6	选定	地址空间
00	I0Y0	00-3F
01	I0Y1	40-7F
10	I0Y2	80-BF
11	I0Y3	C0-FF

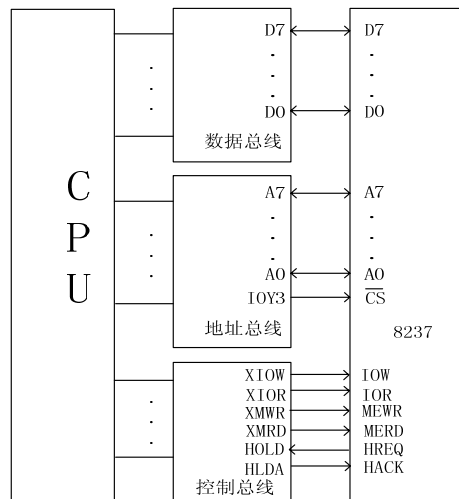


图 6-2-5 8237 和 CPU 挂接图

我们可以应用复杂模型机指令系统来对外扩的 8237 芯片进行初始化操作。实验箱上 8237 的引脚都以排针形式引出。

应用复杂模型机的指令系统，实现以下功能：对 8237 进行初始化，每次给通道 0 发一次请求信号，8237 将存储器中 40H 单元中的数据以字节传输的方式送至输出单元显示。

根据实验要求编写机器程序如下：

\$P 00 60	; LDI R0, 00H	将立即数 00 装入 R0
\$P 01 00		
\$P 02 30	; OUT CDH, R0	将 R0 中的内容写入端口 CD 中，总清
\$P 03 CD	;	
\$P 04 60	; LDI R0, 40H	将立即数 40 装入 R0
\$P 05 40		
\$P 06 30	; OUT C0H, R0	将 R0 中的内容写入端口 C0 中，即写
\$P 07 C0	;	通道 0 地址低 8 位
\$P 08 60	; LDI R0, 00H	将立即数 00 装入 R0
\$P 09 00		
\$P 0A 30	; OUT C0H, R0	将 R0 中的内容写入端口 C0 中，即写
\$P 0B C0	;	通道 0 地址高 8 位
\$P 0C 60	; LDI R0, 00H	将立即数 00 装入 R0
\$P 0D 00		
\$P 0E 30	; OUT C1H, R0	将 R0 中的内容写入端口 C1 中，即写
\$P 0F C1	;	通道 0 传送字节数低 8 位
\$P 10 60	; LDI R0, 00H	将立即数 00 装入 R0
\$P 11 00		
\$P 12 30	; OUT C1H, R0	将 R0 中的内容写入端口 C1 中，即写
\$P 13 C1	;	通道 0 传送字节数高 8 位

```

$P 14 60 ; LDI R0, 18H      将立即数 18 装入 R0
$P 15 18
$P 16 30 ; OUT CBH, R0      将 R0 中的内容写入端口 CB 中，即写
$P 17 CB ;                  通道 0 方式字
$P 18 60 ; LDI R0, 00H      将立即数 00 装入 R0
$P 19 00
$P 1A 30 ; OUT C8H, R0      将 R0 中的内容写入端口 C8 中，即写
$P 1B C8 ;                  命令字
$P 1C 60 ; LDI R0, 0EH      将立即数 0E 装入 R0
$P 1D 0E
$P 1E 30 ; OUT CFH, R0      将 R0 中的内容写入端口 CF 中，即写
$P 1F CF ;                  主屏蔽寄存器
$P 20 60 ; LDI R0, 00H      将立即数 00 装入 R0
$P 21 00
$P 22 30 ; OUT C9H, R0      将 R0 中的内容写入端口 C9 中，即写
$P 23 C9 ;                  请求字
$P 24 60 ; LDI R0, 00H      将立即数 00 装入 R0
$P 25 00
$P 26 61 ; LDI 01H, R1      将立即数 01 装入 R1
$P 27 01 ;
$P 28 04 ; ADD R0, R1       R0+R1->R0
$P 29 D0 ; STA 40H, R0      将 R0 中的内容存入 40H 中
$P 2A 40 ;
$P 2B E0 ; JMP 26H
$P 2C 26 ;
$P 2D 50 ; HLT
; //*****数据*****//
$P 40 00

```

#### 6.2.4 实验步骤

(1) 在复杂模型机实验接线图的基础上，再增加本实验 8237 部分的接线。接线图如图 6-2-6 所示。

(2) 本实验只用了 8237 的 0 通道，将它设置成请求方式。REQ0 接至脉冲信号源 KK+ 上。

(3) 微程序沿用复杂模型机的微代码程序，选择联机软件的“【转储】—【装载】”功能，在打开文件对话框中选择“带 DMA 的模型机设计实验.txt”，软件自动将机器程序和微程序写入指定单元。

(4) 运行上述程序。

将时序与操作台单元的开关 KK1、KK3 置为‘运行’档，按动 CON 单元的总清按钮 CLR，将使程序计数器 PC、地址寄存器 AR 和微程序地址为 00H，程序可以从头开始运行，暂存器 A、B，指令寄存器 IR 和 OUT 单元也会被清零。

将时序与操作台单元的开关 KK2 置为‘连续’档，按动一次 ST 按钮，即可连续运行指令，按动 KK 开关，每按动一次，OUT 单元显示循环程序段（26H——2CH）已执行的次数（由于存储单元 40H 初值为‘0’）。循环程序段（26H——2CH）每执行一次，存储单元 40H 中的数据加 1，因而存储单元 40H 中的值就是循环程序段（26H——2CH）已执行的次数）。

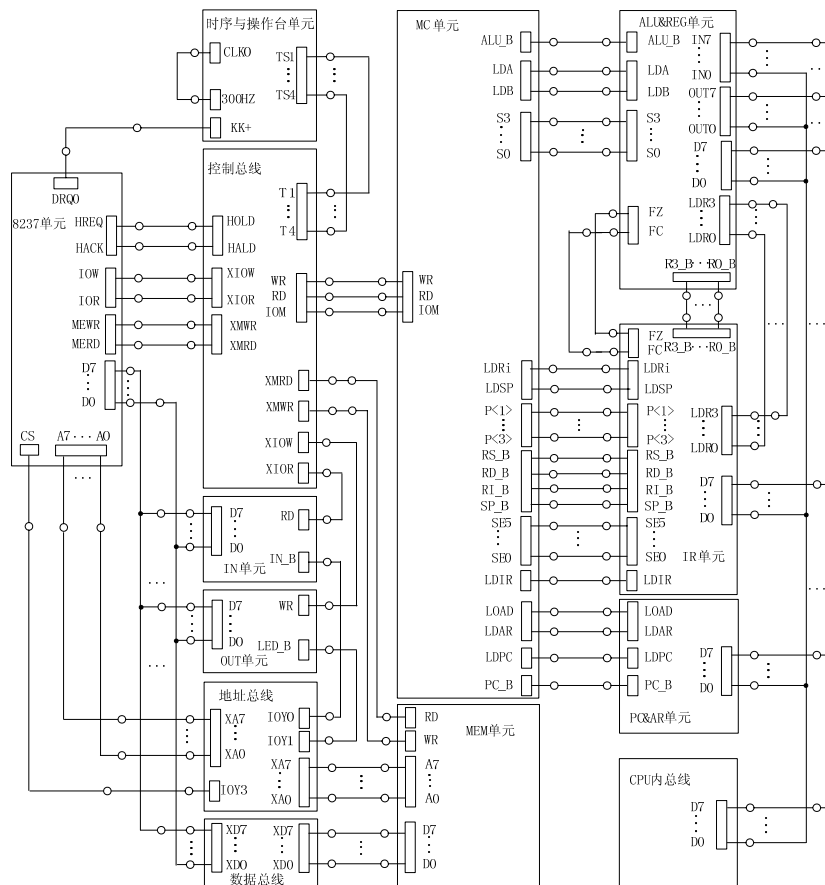


图 6-2-6 实验接线图

## 6.3 典型 I/O 接口 8253 扩展设计实验

### 6.3.1 实验目的

- (1) 掌握 CPU 外扩接口芯片的方法。
- (2) 掌握 8253 定时器/计数器原理及其应用编程。

### 6.3.2 实验设备

PC 机一台，TD-CMX 实验系统一套。

### 6.3.3 实验原理

#### 1. 8253 芯片引脚说明

- (1) 8253 的引脚分配图如图 6-3-1 所示。

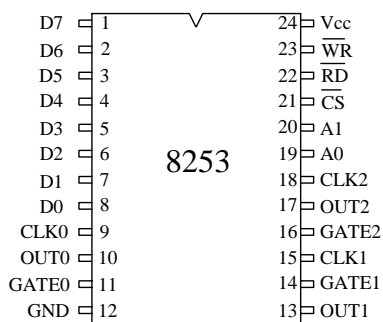


图 6-3-1 8253 芯片引脚说明

- (2) 芯片引脚说明

- D7~D0 为数据线。
- $\overline{CS}$  为片选信号，低电平有效。
- A0、A1 用来选择三个计数器及控制寄存器。
- $\overline{RD}$  为读信号，低电平有效。它控制 8253 送出数据或状态信息至 CPU。
- $\overline{WR}$  为写信号，低电平有效。它控制把 CPU 输出的数据或命令信号写到 8253。
- CLK<sub>n</sub>、GATE<sub>n</sub>、OUT<sub>n</sub> 分别为三个计数器的时钟、门控信号及输出端。

$\overline{CS}$ 、A0、A1、 $\overline{RD}$ 、 $\overline{WR}$  五个引脚的电平与 8253 操作关系如表 5-1-1 所示。

表 6-3-1 引脚电平与 8253 芯片的操作关系

$\overline{\text{CS}}$	$\overline{\text{RD}}$	$\overline{\text{WR}}$	A1	A0	寄存器选择和操作
0	1	0	0	0	写入寄存器#0
0	1	0	0	1	写入寄存器#1
0	1	0	1	0	写入寄存器#2
0	1	0	1	1	写入控制寄存器
0	0	1	0	0	读计数器#0
0	0	1	0	1	读计数器#1
0	0	1	1	0	读计数器#2
0	0	1	1	1	无操作 (3 态)
1	X	X	X	X	禁止 (3 态)
0	0	1	X	X	无操作 (3 态)

## 2. 8253 芯片外部连接

对于 CPU 外扩接口芯片，其重点是要设计接口芯片数据线、地址线和控制线与 CPU 的挂接，图 6-3-2 是 8253 接口芯片的典型扩展接法。这里的模型计算机可以直接应用前面的复杂模型要，其 I/O 地址空间分配情况如表 6-3-2 所示。

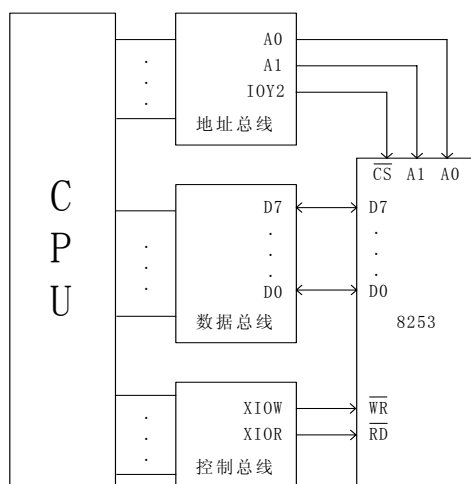


图 6-3-2 8253 和 CPU 挂接图

表 6-3-2 I/O 地址空间分配

A7	A6	选定	地址空间
00		IOY0	00-3F
01		IOY1	40-7F
10		IOY2	80-BF
11		IOY3	C0-FF

我们可以应用复杂模型机指令系统的 IN、OUT 指令来对外扩的 8253 芯片进行操作。实验箱上 8253 的 GATE0 已接为高电平，其余都以排针形式引出。

应用复杂模型机的指令系统，实现以下功能：对 8253 进行初始化，使其以 IN 单元数据 N 为计数初值，在 OUT 端输出方波，8253 的输入时钟为系统总线上的 XCLK。

根据实验要求编写机器程序如下：

```
; //***** Start Of Main Memory Data *****//
$P 00 21    ; IN   R1,00H      IN->R1
$P 01 00
$P 02 C0    ; LAD   R0,30H      30 单元数据送 R0（直接寻址）
$P 03 30
$P 04 30    ; OUT   83H,R0      R0 送 83H 端口（写控制字）
$P 05 83
$P 06 34    ; OUT   80H,R1      R1 送 80H 端口（写 0#通道低字节）
$P 07 80
$P 08 50    ; HLT                      停机
$P 30 16    ;                      控制字
; //***** End Of Main Memory Data *****//
```

#### 6.3.4 实验步骤

(1) 在复杂模型机实验接线图的基础上，再增加本实验 8253 部分的接线。接线图如图 6-3-3 所示。

(2) 本实验只用了计数器#0 通道，将它设置成方波速率发生器（方式 3）。CLK0 接至系统总线的 XCLK 上；GATE0=1（已常接高），计数允许。OUT0 即为方波输出端。

其中，30H 单元存放的数 16H 为 8253 的控制字，它的功能为选择计数器 0，只读/写最低的有效字节，选择方式 3，采用二进制。IN 单元的开关置的数 N 为计数值，即输出是 N 个 CLK 脉冲的方波。

(3) 微程序沿用复杂模型机的微代码程序，选择联机软件的“【转储】—【装载】”功能，在打开文件对话框中选择“典型 I/O 接口 8253 扩展设计实验.txt”，软件自动将机器程序和微程序写入指定单元。

(4) 运行上述程序，分两种情况：本机方式或联机方式，本机方式运行程序时，要借助示波器来观测 8253 的输入和输出波形。而在联机方式时，可用联机操作软件的示波器功能测 8253 的 OUT0 端和系统总线的 XCLK 波形。进入软件界面，选择菜单命令“【实验】—【CISC 实验】”，打开相应的数据通路图，选择相应的功能命令，即可联机运行、调试程序。当机器指令执行到 HLT 指令时，停止运行程序，再选择菜单命令“【波形】—【打开】”，打开示波器窗口，选择菜单命令“【波形】—【运行】”，启动逻辑示波器，就可观测到 OUT0 端和系统总线的 XCLK 端的波形。将开关置不同的计数值，按下 CON 单元的 CLR，再运行机器指令后，可观察到 OUT0 端输出波形的频率变化。



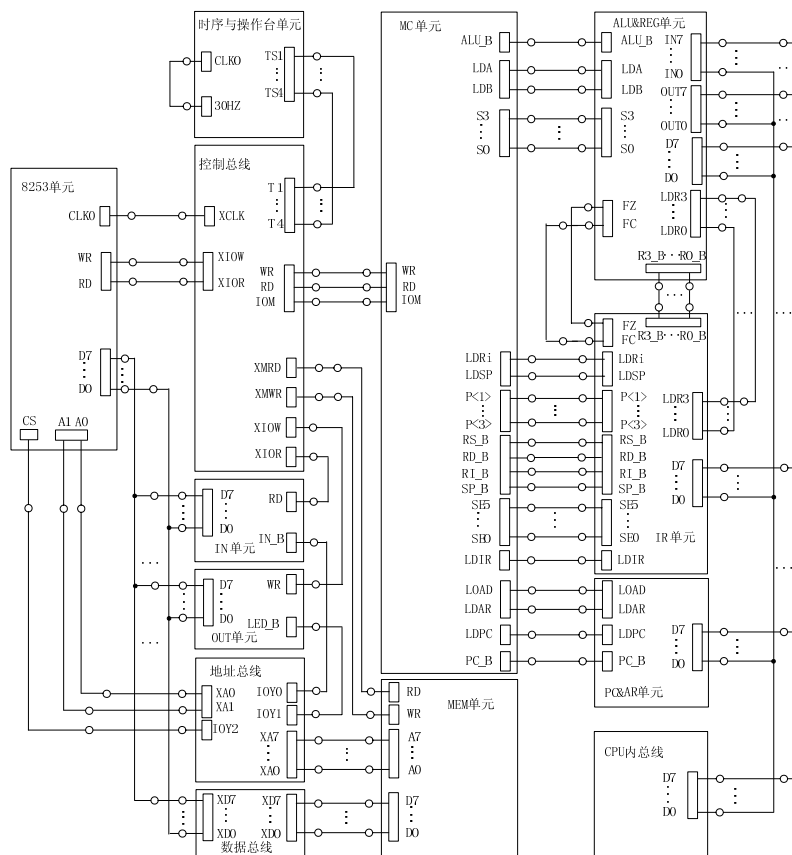


图 6-3-3 实验接线图