

高等学校教材

数字逻辑 与计算机设计基础

刘 真 蔡懿慈 毕才术

高等教育出版社

本书编委会

主任 吕福源

副主任 张尧学

编 委 周立柱 杨士强 程 旭 刘 真
王 洪 蔡懿慈 李红燕

内 容 提 要

本书以一个计算机硬件系统的设计为主线,先介绍数字电路设计的基本知识,然后使用这些知识和计算机结构的基本知识设计一个简单的计算机,最后利用已有的芯片设计完成一个较完整的计算机硬件系统。全书共9章,分别为:数制及编码、组合逻辑电路、时序逻辑电路、计算机设计引论、总线、存储器、简易计算机设计、基于微处理器的计算机设计、接口与通信等,并用附录的形式介绍了8086微机原理及接口实验系统和8086指令系统的内容。本书包含了数字逻辑、计算机原理和微机原理等三门课程的必备知识,并有机地结合在一起,适合计算机软件专业、非计算机专业的本科生和各类专科生作为教材或自学教材使用。

前　　言

硬件设计课程由于设备、师资、学时等众多因素所限，在许多大学计算机系的课程中处于相对次要的地位。同时，由于软件产业的发展创造了更多的就业机会，而软件开发对于硬件知识的依赖较少，因此学生们对软件的兴趣正在增加，而忽视了硬件设计的实践环节。但是，作为专业的计算机工作者和高级软件开发人员，如果不能很好地理解计算机的结构和工作原理，对于许多问题，包括软件设计方面的问题，就不可能找到最优的系统解决方案。

在硬件课程教学实践中，由于学生们很少有机会接触具体的工程问题和实际设备，同时由于各课程过分强调技术的细节，而忽视了相互之间的关系，学生们往往“只见树木不见森林”。本书编写的出发点正是为了弥补这些不足。

本书以一个计算机硬件系统的设计为主线，先介绍数字电路设计的基本知识，然后使用这些知识和计算机结构的基本知识设计一个简单的计算机，最后利用已有的芯片设计完成一个较完整的计算机硬件系统。为了使这条主线清晰，我们尽量略去了不必要的知识和内容，必备知识也尽可能用通俗、简单的方式叙述，而新兴的可编程逻辑器件技术和硬件描述语言由于篇幅所限没有介绍，对于想深入了解相关知识的读者，请参考有关的书籍。

原教育部副部长吕福源同志（现任国家商务部部长）提出了编写本书的思想，教育部高等教育司张尧学司长亲自主持了本书的编写组织工作，并在百忙之中仔细审阅了全部书稿，提出了许多指导性的修改意见，使本书得以进一步完善。北京邮电大学白中英先生也仔细审阅了全稿，并提出了许多宝贵意见，在此表示衷心的感谢。

本书针对软件学院学生的特点和要求，同时考虑到其他相关专业的教学需求组织编写。本书内容涉及数字逻辑、计算机原理、微机原理等三方面的知识，编者力图将它们有机地结合在一起，以期达到融汇贯通、深入浅出的目的。但限于编者水平，虽尽力而为，几经修改，仍与当初的设想和期望差距很大。鉴于时间所限，只好仓促付梓。不足之处，请读者批评指正。

编　　者

2003年6月

目 录

第一章 数制及编码	(1)		
1.1 进位计数制与数制转换	(1)		
1.1.1 二进制	(2)		
1.1.2 八进制	(2)		
1.1.3 十六进制	(2)		
1.1.4 二进制与八进制、十六 进制之间的转换	(3)		
1.1.5 二进制数与十进制数 之间的转换	(4)		
1.2 编码	(6)		
1.2.1 十进制数的二进制 编码	(6)		
1.2.2 带符号的二进制数 的编码	(7)		
1.2.3 带小数点的数的 编码	(12)		
1.2.4 字符编码	(13)		
习题	(15)		
第二章 组合逻辑电路	(16)		
2.1 逻辑代数	(16)		
2.1.1 逻辑代数的基本运算	(16)		
2.1.2 逻辑代数的基本公式 和运算规则	(16)		
2.1.3 布尔函数的化简与 实现	(18)		
2.2 典型的组合电路	(30)		
2.2.1 译码器	(31)		
2.2.2 多路开关(多路选择 器)	(32)		
2.2.3 加法器	(33)		
2.2.4 带有快速进位生成的加 法器	(36)		
2.2.5 乘法器	(37)		
2.3 组合电路的竞争与冒险	(39)		
2.3.1 竞争与冒险的产生	(40)		
2.3.2 竞争与冒险的识别	(40)		
2.3.3 竞争与冒险的消除	(42)		
习题	(42)		
第三章 时序逻辑电路	(44)		
3.1 锁存器与触发器	(44)		
3.1.1 基本 R-S 触发器的电路 结构与动作特点	(45)		
3.1.2 同步 R-S 触发器	(46)		
3.1.3 J-K 触发器	(48)		
3.1.4 D 触发器	(49)		
3.1.5 T 触发器	(49)		
3.2 时序机(状态机)	(50)		
3.2.1 同步时序电路的结构	(50)		
3.2.2 激励表、状态表及 状态图	(51)		
3.3 同步时序逻辑电路	(55)		
3.3.1 同步时序逻辑电路 分析	(55)		
3.3.2 同步时序电路的设计	(56)		
3.4 典型的同步时序电路	(61)		
3.4.1 移位寄存器	(61)		
3.4.2 计数器	(63)		
3.4.3 节拍信号发生器	(68)		
习题	(70)		
第四章 计算机设计引论	(73)		
4.1 存储程序控制原理	(73)		
4.2 计算机硬件系统组成	(75)		

4.3 简易计算机设计总体构想	(78)	7.3.1 硬件逻辑结构总体 设计	(125)
习题	(78)	7.3.2 指令时序的控制方式 ...	(127)
第五章 总线	(79)	7.3.3 指令周期和时标系统 ...	(129)
5.1 总线的概念	(79)	7.3.4 指令周期和时钟周期的 确定	(130)
5.2 总线的组成	(80)	7.3.5 确定微操作时间表与微操 作命令逻辑表达式	(135)
5.2.1 总线通道	(81)	习题	(137)
5.2.2 总线上的设备	(81)	第八章 基于微处理器的计算机 设计	(138)
5.2.3 总线控制器	(82)	8.1 引言	(138)
5.3 集电极开路总线和三态 门总线	(82)	8.2 Intel 8086 微处理器	(138)
5.3.1 集电极开路与非门 (OC 门).....	(83)	8.2.1 8086/8088 微处理器 结构	(139)
5.3.2 三态门电路	(85)	8.2.2 8086 微处理器的总线 周期	(141)
5.4 常用总线简介	(86)	8.2.3 8086 的引脚与功能	(142)
习题	(92)	8.2.4 8086 的存储器及 I/O 组织	(144)
第六章 存储器	(93)	8.3 8086 最小系统组成与总线 周期波形	(144)
6.1 概述	(93)	第九章 接口与通信	(148)
6.1.1 存储器的分类	(94)	9.1 接口的基本概念及基本 技术	(148)
6.1.2 存储器的层次结构	(97)	9.1.1 接口的概念	(148)
6.2 半导体 RAM 位元电路	(98)	9.1.2 接口信息	(148)
6.2.1 静态 RAM 位元电路.....	(98)	9.1.3 输入/输出传送方式	(149)
6.2.2 动态 RAM 位元电路.....	(99)	9.1.4 可编程定时器/计数器 芯片 8253	(150)
6.3 主存储器结构与工作原理 ...	(100)	9.1.5 可编程并行输入/输出 接口芯片 8255A	(157)
6.4 只读存储器结构与布尔 函数的实现	(107)	9.2 串行通信	(163)
习题	(109)	9.2.1 异步通信方式 ASYNC ...	(163)
第七章 简易计算机设计	(111)	9.2.2 同步通信方式	(164)
7.1 指令系统设计	(111)	9.2.3 异步通信的标准接口 ...	(166)
7.1.1 指令系统设计的基本 原则	(111)	9.2.4 可编程异步通信接口	
7.1.2 指令格式	(113)		
7.1.3 指令类型和基本指令 的设计	(114)		
7.2 运算器设计	(118)		
7.2.1 运算器设计	(119)		
7.2.2 乘法和除法运算	(121)		
7.3 控制器设计	(124)		

8250	(170)	附录.....	(193)
9.3 中断技术与 DMA 技术	(183)	附录一 8086 微机原理及接口	
9.3.1 中断的基本原理与 8259A		实验系统	(193)
中断控制器	(183)	附录二 8086 指令系统	(197)
9.3.2 DMA 技术与 8237DMA		参考文献	(211)
控制器	(188)		

第一章 数制及编码

1.1 进位计数制与数制转换

数制是人们表示数值大小的各种方法的统称。迄今为止，人类都是按照进位方式来实现计数的，这种数制称为进位计数制。大家熟悉的十进制就是一种典型的进位计数制。在数字系统中，广泛采用的是二进制、八进制和十六进制。

一种进位计数包含着两个基本的因素：基数和位权。基数是计数制中所用到的数码的个数，如十进制中使用的0、1、2、…、9十个字符，所以十进制数的基数为10。一般地说，基数为 R 的计数制中，包含的是0、1、…、 $R - 1$ 等数码。位权则表示在一个进位计数制表示的数中，处在不同数位的数码代表着不同的数值，某一个数位的数值是由这一位数码的值乘上处在该位的一个固定常数。不同数位上的固定常数称为位权值，简称位权。例如，十进制数个位的位权值是 10^0 ，十位的位权值是 10^1 ，百位的位权值是 10^2 ，等等。一个 R 进制数 N ，可以有两种表示方式：

(1) 并列表示方式或位置计数法

$$(N)_R = (K_{n-1} K_{n-2} \dots K_1 K_0. K_{-1} K_{-2} \dots K_{-m})_R$$

其中， n 为整数部分的数位， m 为小数部分的数位， R 表示基数， K_i 为不同数位的数值：

$$0 \leq K_i \leq R - 1$$

(2) 多项式表示法或以权展开式

$$(N)_R = K_{n-1}R^{n-1} + K_{n-2}R^{n-2} + \dots + K_1R^1 + K_0R^0 + K_{-1}R^{-1} + \dots + K_{-m}R^{-m}$$

或者写成和式：

$$(N)_R = \sum_{i=-m}^{n-1} K_i R^i$$

其中， R 代表进位制的基数， m, n 为正整数， n 代表整数部分的位数， m 代表小数部分的位数， K_i 代表 R 进位制中 R 个数字符号中的任何一个：

$$0 \leq K_i \leq R - 1$$

1.1.1 二进制

1. 二进制数的表示

基数 $R = 2$ 的数制为二进制。二进制数的数值表示只有“1”和“0”，进位规律是“逢二进一”，任意一个二进制数 N 的多项式表示为

$$\begin{aligned}(N)_2 &= K_{n-1}2^{n-1} + K_{n-2}2^{n-2} + \dots + K_12^1 + K_02^0 + K_{-1}2^{-1} + \dots + K_{-m}2^{-m} \\ &= \sum_{i=-m}^{n-1} K_i 2^i\end{aligned}$$

其中， K_i 为 0 或 1， i 为位权。

例如，二进制数 1111.001 可以展开为：

$$1111.001 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

2. 二进制数的运算

二进制数的算术运算规律有加法规律和乘法规律。

加法规律为

$$0 + 0 = 0 \quad 0 + 1 = 1 + 0 = 1 \quad 1 + 1 = 10$$

乘法规律为

$$0 \times 0 = 0 \quad 0 \times 1 = 1 \times 0 = 0 \quad 1 \times 1 = 1$$

1.1.2 八进制

基数 $R = 8$ 的数制为八进制。八进制数的数位符号有 8 个，即 0、1、…、7，进位规律是“逢八进一”，任意的一个八进制数 N 的多项展开式为

$$\begin{aligned}(N)_8 &= K_{n-1}8^{n-1} + K_{n-2}8^{n-2} + \dots + K_18^1 + K_08^0 + K_{-1}8^{-1} + \dots + K_{-m}8^{-m} \\ &= \sum_{i=-m}^{n-1} K_i 8^i\end{aligned}$$

其中， K_i 表示为 0 ~ 7 中的任意一个。

1.1.3 十六进制

基数 $R = 16$ 的数制为十六进制。十六进制数的表示符号有 16 个，分别为 0 ~ 9 以及用 A 、 B 、 C 、 D 、 E 和 F 表示的 10 ~ 15。进位规律为“逢十六进一”，任意的一个十六进制数 N 的多项表示式为

$$\begin{aligned}(N)_{16} &= K_{n-1}16^{n-1} + K_{n-2}16^{n-2} + \dots + K_116^1 + K_016^0 + K_{-1}16^{-1} + \dots + K_{-m}16^{-m} \\ &= \sum_{i=-m}^{n-1} K_i 16^i\end{aligned}$$

其中 K_i 表示为 0 ~ 9 以及 A、B、C、D、E 和 F 中的任意一个。

1.1.4 二进制与八进制、十六进制之间的转换

1. 二进制数转换为八进制数

二进制数转换为八进制数时，整数部分从低位向高位每 3 位分为一组，最高一组不够 3 位时，用 0 补足；小数部分从高位向低位每 3 位一组，最后不足 3 位的，在低位补 0，然后分别把每 3 位的二进制数用相应的八进制数表示。

例如， $(11110.11)_2$ 可表示为

$$\begin{array}{ccccccc} 011 & & 110 & . & 110 \\ & 3 & & 6 & . & 6 \end{array}$$

即 $(11110.11)_2 = (36.6)_8$

2. 八进制数转换为二进制数

八进制数转换为二进制数时，把每位八进制数用三位二进制数表示。

例如， $(413.62)_8$ 可表示为

$$\begin{array}{ccccccc} 4 & & 1 & & 3 & . & 6 & 2 \\ 100 & & 001 & & 011 & . & 110 & 010 \end{array}$$

即 $(413.62)_8 = (100001011.11001)_2$

3. 二进制数转换为十六进制数

二进制数转换为十六进制数时，整数部分由小数点向左，每四位一组，最高一组不足 4 位时，前面补 0；小数部分由小数点向右，每 4 位一组，最后不足 4 位的，在低位补 0，然后分别把每 4 位二进制数用相应的十六进制数表示。

例如， $(1110111.101)_2$ 可表示为

$$\begin{array}{ccccccc} 0111 & & 0111 & . & 1010 \\ 7 & & 7 & . & A \end{array}$$

即 $(1110111.101)_2 = (77.A)_{16}$

4. 十六进制数转换为二进制数

十六进制数转换为二进制数时，把每位十六进制数用相应的 4 位二进制数表示。

例如， $(41B.2)_{16}$ 可表示为

$$\begin{array}{ccccccc} 4 & & 1 & & B & . & 2 \\ 0100 & & 0001 & & 1011 & . & 0010 \end{array}$$

即 $(41B.2)_{16} = (10000011011.001)_2$

1.1.5 二进制数与十进制数之间的转换

1. 二进制数转换为十进制数

采用多项式替代法把二进制数按权展开, 利用十进制运算法则求出其值, 即可将二进制数转换为十进制数。

例如 :

$$\begin{aligned}(101.101)_2 &= 1 \times 2^2 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-3} \\&= 4 + 1 + 0.5 + 0.125 \\&= (5.625)_{10}\end{aligned}$$

2. 十进制数转换为二进制数

采用基数除 / 乘法, 分别将十进制数的整数部分和小数部分转换为二进制数, 整数部分用基数除法, 小数部分用基数乘法, 然后用小数点将两部分连接起来。

例如, 将十进制数 $(125.843)_{10}$ 转换为二进制数。

对于整数部分,

$$(125)_{10} = K_{n-1}2^{n-1} + K_{n-2}2^{n-2} + \dots + K_12^1 + K_02^0$$

显然, 等式右边除 K_0 项外都有 2 的因子。因此, 用 2 除 $(125)_{10}$, 所得余数即为 K_0 , 即

$$\begin{array}{r} 2 \mid 125 \\ \hline 62 \dots \text{余数 } 1 = K_0 \end{array}$$

并得到等式:

$$(62)_{10} = K_{n-1}2^{n-2} + K_{n-2}2^{n-3} + \dots + K_22^1 + K_1$$

同样, 再用 2 除 $(62)_{10}$, 余数则为 K_1 , 即

$$\begin{array}{r} 2 \mid 62 \\ \hline 31 \dots \text{余数 } 0 = K_1 \end{array}$$

再用这样的方法一直继续下去, 直到商为 0 为止。

2	125	余数为 1, 所以 $K_0=1$
2	62	余数为 0, 所以 $K_1=0$
2	31	余数为 1, 所以 $K_2=1$
2	15	余数为 1, 所以 $K_3=1$
2	7	余数为 1, 所以 $K_4=1$
2	3	余数为 1, 所以 $K_5=1$
2	1	余数为 1, 所以 $K_6=1$
	0	

得 $(125)_{10} = (1111101)_2$ 。

十进制小数转换为二进制小数的方法是：不断用 2 乘要转换的十进制小数，将每次所得的整数（0 或 1），依次记为 K_{-1}, K_{-2}, \dots 。若乘积的小数部分最后能为 0，那么最后一次乘积的整数部分记作 K_{-m} ，则 $0.K_{-1}K_{-2}\dots K_{-m}$ 即为十进制小数的二进制表达式。因为十进制小数并不都是能用有限位的二进制小数精确表示，通常则是根据精度要求 m 位，作为十进制小数的二进制表示的近似表达式。

例如：将 $(0.843)_{10}$ 转换为二进制数。

$$(0.843)_{10} = K_{-1}2^{-1} + K_{-2}2^{-2} + \dots + K_{-m}2^{-m}$$

两边乘以 2，得

$$(1.686)_{10} = K_{-1} + K_{-2}2^{-1} + \dots + K_{-m}2^{-m+1}$$

所以 $K_{-1} = 1$ ；再对 $(0.686)_{10} = K_{-2}2^{-1} + K_{-3}2^{-2} + \dots + K_{-m}2^{-m+1}$

两边再乘以 2，得

$$(1.372)_{10} = K_{-2} + K_{-3}2^{-1} + K_{-4}2^{-2} + \dots + K_{-m}2^{-m+2}$$

所以 $K_{-2} = 1$ ；假如精度要求 $m = 2$ ，转换过程如下：

0.843	
$\times \quad 2$	
<hr style="border: 0.5px solid black; width: 100%; margin-bottom: 5px;"/>	
1.686	整数部分为 1，所以 $K_{-1} = 1$ ；
0.686	
$\times \quad 2$	
<hr style="border: 0.5px solid black; width: 100%; margin-bottom: 5px;"/>	
1.372	整数部分为 1，所以 $K_{-2} = 1$ ；

因此 $(0.843)_{10} = (0.11)_2$

$$(125.843)_{10} = (1111101.11)_2$$

1.2 编码

虽然计算机是采用二进制数进行处理,但人们输入的不仅仅是二进制数,而是数字、字母甚至符号,这些数字、字母和符号也必须用二进制数来表示,各种不同的表示方法就称为编码。

1.2.1 十进制数的二进制编码

用二进制数码按照不同规律编码来表示十进制数,使其既具有二进制的形式,又具有十进制数的特点,便于传递、处理。

一位十进制数有0~9十个不同数码,至少需要4位二进制数编码。当采用4位二进制数进行编码时,共有16种代码。从16种代码中取10种代码来表示十进制的10个字符的编码方式很多。一般分为有权码和无权码。有权码是指4位二进制数中的每一位都对应有固定的权。无权码是指4位二进制数中的每一位无固定的权,而要遵循另外的规则。最常用的十进制数的二进制编码有以下几种。

1. 8421 码

8421码为有权码。它是十进制代码中最常见的代码,也称二—十进制码,简称BCD码(Binary Code Decimal)。4位二进制码从高位至低位每位的权分别为 2^3 、 2^2 、 2^1 、 2^0 ,即为8、4、2、1。

2. 5421 码

5421是一种有权码,其权自高位至低位,每位分别为5、4、2、1。所以,十进制数X用5421码 $a_3a_2a_1a_0$ 表示为

$$X = a_3 \times 5 + a_2 \times 4 + a_1 \times 2 + a_0 \times 1$$

3. 2421 码

2421码为另一种有权码,也是四位代码,每位权从高位到低位为2、4、2、1。若一个2421编码的二进制数码为 $a_3a_2a_1a_0$ 时,它表示的十进制数值为

$$X = a_3 \times 2 + a_2 \times 4 + a_1 \times 2 + a_0 \times 1$$

4. 余三码

余三码是一种无权码,因为它是将4位8421码首位各多余的3组去掉而得到的,所以称为余三码,它可由8421码加0011得到。

以上四种十进制数的二进制编码格式如表1.1表示。其中,8421码的1010~

1111 是没有意义的 ,余三码表中 0 和 9、1 和 8、2 和 7、3 和 6、4 和 5 互为反码。因此 ,用余三码做十进制数的算术运算是比较方便的。

表 1.1 常用的 BCD 编码

十进制数	8421	5421	2421	余三码
0	0000	0000	0000	0011
1	0001	0001	0001	0100
2	0010	0010	0010	0101
3	0011	0011	0011	0110
4	0100	0100	0100	0111
5	0101	1000	0101	1000
6	0110	1001	0110	1001
7	0111	1010	0111	1010
8	1000	1011	1110	1011
9	1001	1100	1111	1100

1.2.2 带符号的二进制数的编码

在数字系统中 ,正、负数的表示方法是 :把一个数的最高位作为符号位 ,并用“ 0 ”表示“ + ”;用“ 1 ”表示“ - ”连同符号位在一起作为一个数 称之为机器数 ,它的原来的数值形式则称为这个机器数的真值。

例如 : $X_1 = + 0.1011$; $X_2 = - 0.111$

表示成机器数为 $X_1 = 0.1011$; $X_2 = 1.111$

常用的表示机器数的方法有原码、反码和补码。

1. 原码

正数的符号位用“ 0 ”表示 负数的符号位用“ 1 ”表示 ,数值部分保持不变。

(1) 小数原码的定义

若二进制数 X 为小数 , $X = \pm 0.X_{-1}X_{-2}\dots X_{-m}$ 则 X 的原码表示为

$$[X]_{\text{原}} = \begin{cases} X & \text{当 } 0 \leqslant X < 1 \\ 1 - X & \text{当 } -1 < X \leqslant 0 \end{cases}$$

例如 : $X_1 = + 0.11011$ 则 $[X]_{\text{原}} = 0.11011$

$X_2 = - 0.1101$ 则 $[X]_{\text{原}} = 1 - (- 0.1101) = 1.1101$

(2) 整数原码的定义

若二进制数 X 为整数 , $X = \pm X_{n-1}X_{n-2}\dots X_0$ 则的原码定义为

$$[X]_{\text{原}} = \begin{cases} X & \text{当 } 0 \leq X < 2^n \\ 2^n - X & \text{当 } -2^n < X \leq 0 \end{cases}$$

例如： $X = -11011$

$$\begin{aligned}[X]_{\text{原}} &= 2^n - X \\ &= 10000 - (-11011) \\ &= 10000 + 11011 \\ &= 111011\end{aligned}$$

(3) 零的原码有两种表示形式

$$\begin{aligned}[+0]_{\text{原}} &= 0.00\dots0 \\ [-0]_{\text{原}} &= 1.00\dots0\end{aligned}$$

2. 反码

正数的符号位用“0”表示，负数的符号位用“1”表示；反码数值部分的形成和它的符号位有关。正数反码的数值和原码的数值相同，而负数反码的数值是原码的数值按位求反。

(1) 整数反码的定义

若二进制数 X 为整数， $X = \pm X_{n-1}X_{n-2}\dots X_0$ ，则 X 的反码定义为

$$[X]_{\text{反}} = \begin{cases} X & \text{当 } 0 \leq X < 2^n \\ (2^{n+1} - 1) + X & \text{当 } -2^n < X \leq 0 \end{cases}$$

例如： $X_1 = +11011$ ，则 $[X_1]_{\text{反}} = 011011 = 11011$

$$\begin{aligned}X_2 &= -1101 \text{，则 } [X_2]_{\text{反}} = (2^5 - 1) + X \\ &= (100000 - 000001) + (-1101) \\ &= 11111 - 1101 \\ &= 10010\end{aligned}$$

(2) 小数反码的定义

若二进制数 X 为小数， $X = \pm 0.X_{-1}X_{-2}\dots X_{-m}$ ，则 X 的反码定义为

$$[X]_{\text{反}} = \begin{cases} X & \text{当 } 0 \leq X < 1 \\ 2 - 2^{-n} + X & \text{当 } -1 < X \leq 0 \end{cases}$$

例如： $X_1 = +0.11011$ ，则 $[X_1]_{\text{反}} = 0.11011$

$$\begin{aligned}X_2 &= -0.1101 \text{，则 } [X_2]_{\text{反}} = 2 - 2^{-4} + X \\ &= 10.0000 - 0.0001 - 0.1101 \\ &= 1.0010\end{aligned}$$

(3) 零的反码表示有两种形式

$$[+0]_{\text{反}} = 0.00\dots0$$

$$[-0]_{\text{反}} = 1.11\dots1$$

3. 补码

正数的符号用“0”表示;负数的符号用“1”表示。正数的补码就是二进制数值本身;负数的补码表示方法为将数值部分按位求反,再在最低位加1。

(1) 整数的补码定义

若二进制数 X 为整数, $X = \pm X_{n-1}X_{n-2}\dots X_0$, 则 X 的补码定义为

$$[X]_{\text{补}} = \begin{cases} X & \text{当 } 0 \leq X < 2^n \\ 2^{n+1} + X & \text{当 } -2^n \leq X < 0 \end{cases}$$

例如: $X_1 = +110111$, 则 $[X_1]_{\text{补}} = 110111$

$$\begin{aligned} X_2 = -1101 \text{, 则 } [X_2]_{\text{补}} &= 2^5 + X_2 \\ &= 100000 - 1101 \\ &= 10011 \end{aligned}$$

(2) 小数的补码定义

若二进制数 X 为小数, $X = \pm 0.X_{-1}X_{-2}\dots X_{-m}$, 则 X 的补码定义为

$$[X]_{\text{补}} = \begin{cases} X & \text{当 } 0 \leq X < 1 \\ 2 + X & \text{当 } -1 \leq X < 0 \end{cases}$$

(3) 零的补码只有一种形式:

$$[\pm 0]_{\text{补}} = 0.000\dots0$$

表 1.2 给出了几个典型数的真值、原码、反码和补码的表示。

表 1.2 几个典型数的真值、原码、反码和补码的表示

X	$[X]_{\text{原}}$	$[X]_{\text{反}}$	$[X]_{\text{补}}$	X	$[X]_{\text{原}}$	$[X]_{\text{反}}$	$[X]_{\text{补}}$
+ 1001	1001	1001	1001	- 0.0000	1.0000	1.1111	0.0000
+ 0001	0001	0001	0001	- 0.0010	1.0010	1.1101	1.1110
+ 0.1101	0.1101	0.1101	0.1101	- 0011	1.0011	1.1100	1.1101
+ 0.0000	0.0000	0.0000	0.0000	- 1010	1.1010	1.0101	1.0110

4. 补码的补充说明

由于补码加减运算的简便性,导致了它在计算机中的广泛应用,因而有必要对补码的性质做进一步的说明。从数学上讲,补码与其真值构成了一个以某一值(由计算机字长决定)为模的“模数系统”,或者说构成一个“同余”结构的代数系统。因此,为了进一步理解补码,需先了解什么叫模数系统,什么叫同余。在此基础上,不仅能容易地构成二进制数的各种不同模值的补码,而且能容易地构成十

进制或任何 R 进制中的补码。

(1) 模数系统

所谓模 是指一个计量器的容量 ,或称模数。例如 ,若某一计算机的字长为 4 位 ,则它所能表示的二进制数为 0000 ~ 1111 ,共 16 个数 ,这就是说它的容量为 16 称 4 位字长的模为 16。同理 ,若某一计算机的字长为 l 位 ,则其模为 2^l 。

在 4 位字长的机器中 ,由于模为 2^4 ,故大于等于 2^4 的数就表示不出来。例如 ,要在该计算机中计算 $1001 + 1000$,其结果将为 0001 ,而不是 10001 ,如下所示 :

1	0	0	1	9
+				+
1	0	0	0	8
1	0	0	0	17
丢失	↑	↓		↓
	0	0	0	1

这就是说 ,在 4 位字长的机器中 ,存入 17 与存入 1 是相同的。或者说 ,在模 16 系统中 ,17 就等于 1 ,可表示为

$$17 = 1 \pmod{16}$$

圆括号中的 $\pmod{16}$ 为模 16 的记号。

显然 ,在模 16 系统中 ,下式是成立的

$$16k + N = N \pmod{16} \quad (1.1)$$

式中 $k = 0, 1, 2, \dots, n$ 。

若将式(1.1)两边都除以 16 ,则其余数必相等 :

$$\frac{16k}{16} + \frac{N}{16} = \frac{N}{16}$$

余数 余数

$$k + \frac{N}{16} = \frac{N}{16}$$

↑

整数 (商)

故也称 $(16k + N)$ 和 N 对模 16 是同余的 ,式(1.1)为模 16 的同余式。下面列举某些模数系统。

例 1 钟表是以 12 为模的模数系统。

显然，在该系统中

$$12 \text{ 点} = 0 \text{ 点} (\bmod 12)$$

$$18 \text{ 点} = 6 \text{ 点} (\bmod 12)$$

$$12k + N = N (\bmod 12)$$

例 2 式(1.2)定义的补码：

$$[x]_{\text{补}} = \begin{cases} x & \text{当 } 0 \leq x < 1 \\ 2 + x & \text{当 } -1 \leq x < 0 \end{cases} \quad (1.2)$$

是以 2 为模的模数系统。可写为

$$[x]_{\text{补}} = x (\bmod 2)$$

(2) 十进制数的补码

设 A 为 n 位十进制整数，则其模 10^{n+1} 补码的定义如下：

$$[A]_{10\text{补}} = \begin{cases} A & \text{当 } 0 \leq A < 10^n \\ 10^{n+1} + A & \text{当 } -10^n \leq A < 0 \end{cases} \quad (1.3)$$

或写成

$$[A]_{10\text{补}} = A (\bmod 10^{n+1})$$

$[A]_{10\text{补}}$ 与 A 的关系如下：若 A 为正数，则 $[A]_{10\text{补}}$ 的符号位为 0，数值位与 A 相同；若 A 为负数，则 $[A]_{10\text{补}}$ 的符号位为 9，数值位为 A 的各位对 9 取反并在最低位加 1。

例如， $A = +0063$ ($n = 4$)

$$\begin{aligned} [A]_{10\text{补}} &= 0,0063 \\ A &= -0063 \quad (n = 4) \\ [A]_{10\text{补}} &= 10^{4+1} - 0063 \\ &= 100000 - 0063 \\ &= 9,9937 \end{aligned}$$

与二进制补码加减运算一样，可以证明十进制补码的加减运算规则如下：

$$[A + B]_{10\text{补}} = [A]_{10\text{补}} + [B]_{10\text{补}}$$

例 3 已知十进制数 $A = +1356$, $B = +0978$ 求 $A - B$ 。

解：因

$$\begin{aligned} [A - B]_{10\text{补}} &= [A + (-B)]_{10\text{补}} \\ &= [A]_{10\text{补}} + [-B]_{10\text{补}} \end{aligned}$$

而

$$[A]_{10\text{补}} = 0,1356$$

$$[-B]_{10\text{补}} = 9,9022$$

故

$$\begin{array}{r}
 & 0,1356 \\
 + & 9,9022 \\
 \hline
 & 1 0,0378
 \end{array}$$

丢掉

$$[A - B]_{10\text{补}} = 0,0378$$

$$A - B = +0378$$

例 4 已知十进制数 $A = -1356$, $B = +0978$, 求 $A + B$ 。

解: 因

$$[A + B]_{10\text{补}} = [A]_{10\text{补}} + [B]_{10\text{补}}$$

而

$$[A]_{10\text{补}} = 9,8644$$

$$[B]_{10\text{补}} = 0,0978$$

故

$$\begin{array}{r}
 & 9,8644 \\
 + & 0,0978 \\
 \hline
 & 9,9622
 \end{array}$$

$$[A + B]_{10\text{补}} = 9,9622$$

$$A + B = -0378$$

1.2.3 带小数点的数的编码

在数字系统中, 表示一个既有小数部分又有整数部分的数一般有两种方式: 定点表示和浮点表示。所谓定点数(fixed-point number)就是指在所采用的数据描述格式中小数点的位置是固定不变的数据, 所谓浮点数(floating-point number)就是指在采用的数据描述格式中小数点的位置是浮动的数据。

任何数制的数 N 均可以表示为

$$N = R^E \times M$$

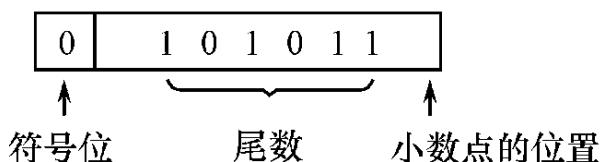
其中, R 为进制的基数; E 为阶码, 取值为整数; M 为数 N 的尾数, 取值为整数或小数。

1. 定点表示法

定点表示法就在一个数中小数点的位置在数中是固定不变的, 这个固定的位置是事先约定好的, 不必用符号表示。在定点表示中, 阶码 E 为零。

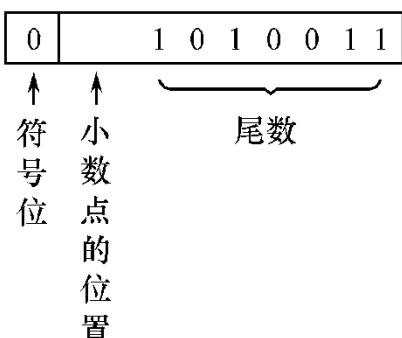
当 $E = 0$, 尾数 M 为纯整数时, 则认为小数点在尾数 M 最低位右边, 为整数定点。

例如： $N = + 1010011$ 则表示为



当 $E = 0$, 尾数 M 为纯小数时, 则认为小数点的位置在 M 最高位的左边, 定点数只能表示小数, 为小数定点。

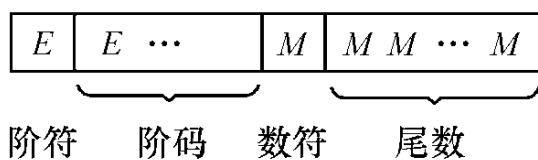
例如： $N = -0.1010011$ 则表示为



定点数表示法, 数 N 范围是限定的, 在小数定点时, 当用 8 位二进制数表示一个数时, 1 位符号位, 7 位表示数值, 若只考虑绝对值, 最大数取值为 $(0.1111111)_2 = (127 \div 128)_{10}$ 最小数取值为 $(0.0000001)_2 = (1 \div 128)_{10}$

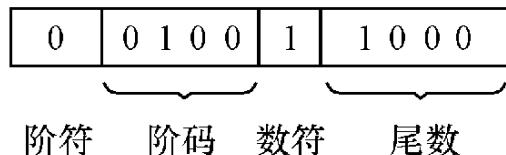
2. 浮点表示法

在浮点表示法数中小数点的位置不是固定不变的, 而是可以变化的。阶码 E 和尾数 M 各自可分别采用原码、反码和补码的形式。格式如下：



例如： $N = 2^4 \times (-8)$, 用二进制数表示为 : $N = 2^{0100} \times (-1000)$

浮点表示的格式如下：



尾数 M 表示了数 N 的全部有效数字, 而阶码 E 指明了小数点的位置。小数点移动的原则是小数点向左移 1 位, 相当于尾数的数码向右移 1 位, 而阶码加 1。

1.2.4 字符编码

在数字系统中, 还需要把符号、文字、图像等用二进制数表示, 这样的二进制

数称为字符代码。目前在国际上用得最多的字符有：十进制数 0 ~ 9 ;大写和小写英文字母各 26 个 ;一些通用的运算符号(+、-、×、÷ 等) 及标点符号等共有 128 种。可用 7 位二进制数对它们进行编码。

1. 7 位 ASCII 编码(American Standards Committee of Information Interchange , 美国信息交换标准委员会)

编码方式 : 英文字母由 A ~ Z 及由 a ~ z 顺序编码 ; 十进制数采用高 3 位 $b_6 b_5 b_4$ 相同 , 为 011 , 低 4 位 $b_3 b_2 b_1 b_0$ 按二进制数顺序编码。7 位 ASCII 的码表如表 1.3 所示。

表 1.3 7 位 ASCII 编码表

$b_6 b_5 b_4$	000	001	010	011	100	101	110	111
$b_3 b_2 b_1 b_0$	NUL	DEL	SP	0	@	P	,	p
0 0 0 0	SOM	DC	!	1	A	Q	a	q
0 0 0 1	STX	DC	"	2	B	R	b	r
0 0 1 0	ETX	DC	#	3	C	S	c	s
0 0 1 1	EOT	DC	\$	4	D	T	d	t
0 1 0 0	ENQ	NAK	%	5	E	U	e	u
0 1 0 1	ACK	SYN	&	6	F	V	f	v
0 1 1 0	BEL	ETB	,	7	G	W	g	w
0 1 1 1	BS	CAN	(8	H	X	h	x
1 0 0 0	HT	EM)	9	I	Y	i	y
1 0 0 1	LF	SUB	*	:	J	Z	j	z
1 0 1 0	VT	ESC	+	;	K	[k	
1 0 1 1	FF	FS	,	<	L	\	l	
1 1 0 0	CR	GS	-	=	M]	m	
1 1 0 1	SO	RS	.	>	N		n	~
1 1 1 0	SI	US	/	?	O	←	o	DEL
1 1 1 1								

2. 8 位 ASCII 编码

用最高 2 位($b_7 b_6$) 把字符分为 4 类 :

00 : 表示控制字符 ;

01 : 表示数字及通用符号 ;

10 : 表示大写英文字母 ;

11 : 表示小写英文字母。

第五位(b_5) 与最高位(b_7) 相同 编码的其余 5 位($b_4 ~ b_0$) 与 7 位 ASCII 编

码相同。

3. EBCDIC 编码

EBCDIC 编码是扩展二—十进制交换码。8 位码都用来表示字符。编码方式类似 7 位 ASCII 码的编码方式。低 4 位与 BCD 8421 码相同。

习 题

1. 将下列数按权展开。

$$(1992.713)_{10} (1101.101)_2$$

2. 按二进制运算规则求： $A + B$; $B - C$; $A * D$; C/D 。其中： $A = 10110100$; $B = 1011110$;
 $C = 1001011$; $D = 011$

3. 将下列二进制数转换为八进制和十六进制数。

$$(10111.110)_2 (1001010.101)_2$$

4. 将下列十进制数转换为二进制、八进制、十六进制数。

$$(17)_{10} (0.5918)_{10} (125)_{10} (234.125)_{10}$$

5. 将下列数转换为十进制数。

$$(101011.110)_2 (63.2)_8 (A8C)_{16}$$

6. 写出下列数的原码、补码、反码。

$$(1001)_2 (-1001)_2 (110.1)_2 (-110.1)_2$$

$$(1000)_2 (-1000)_2 (+1000)_2 (-0000)_2$$

7. 已知下列机器数，写出相对应的真值。

$$[X]_{\text{原}} = 10111 [X]_{\text{反}} = 10111 [X]_{\text{补}} = 10111$$

8. 将下列字符串表示为 ASCII 码。

YES ;COMPUTER \$IN(3.14/N)

第二章 组合逻辑电路

2.1 逻辑代数

逻辑代数是研究逻辑变量及其相互关系的一门科学,由于它是英国数学家乔治·布尔(George Boole)于1849年提出来的,因此也称为布尔代数。逻辑代数是分析和设计数字电路的基础。电子计算机是对“0”或“1”进行处理的,它们是通过电子开关线路实现的。这些开关电路具有下列特点:从线路内部看,或是管子导通,或是管子截止;从线路的输入输出看,或是高电平,或是低电平。这种开关电路的工作状态可以用二元布尔代数描述,通常又称为开关代数,或逻辑代数。

2.2.1 逻辑代数的基本运算

逻辑代数中的变量有三种运算,即“或”运算、“与”运算和“非”运算。其定义如表2.1所示。

表2.1 逻辑代数的三种运算

“或”运算		“与”运算		“非”运算	
+	0 1	.	0 1	—	
0	0 1	0	0 0	0	1
1	1 1	1	0 1	1	0

2.1.2 逻辑代数的基本公式和运算规则

1. 基本公式

逻辑代数有些常用的基本公式,这些公式均可用真值表证明。熟练掌握这些公式对逻辑函数的化简是非常有用的。逻辑代数的基本公式如表2.2所示。

表 2.2 逻辑代数的基本公式

公式名称	公 式	
交换律	$A \cdot B = B \cdot A$	$A + B = B + A$
结合律	$A \cdot (BC) = (AB) \cdot C$	$A + (B + C) = (A + B) + C$
分配律	$A(B + C) = AB + AC$	$A + BC = (A + B)(A + C)$
吸收律	$A(A + B) = A$	$A + AB = A$
	$A(\bar{A} + B) = AB$	$A + \bar{A}B = A + B$
包含律	$(A + B)(\bar{A} + C)(B + C) = (A + B)(\bar{A} + C)$	$AB + \bar{A}C + BC = AB + \bar{A}C$
互补律	$A \cdot \bar{A} = 0$	$A + \bar{A} = 1$
0-1律	$0 \cdot A = 0$	$0 + A = A$
	$1 \cdot A = A$	$1 + A = 1$
对合律	$\bar{\bar{A}} = A$	
重叠律	$A \cdot A = A$	$A + A = A$
反演律	$\overline{A + B} = \bar{A} \cdot \bar{B}$	$\overline{AB} = \bar{A} + \bar{B}$

2. 运算规则

(1) 代入规则

对于任何一个逻辑等式,以某个逻辑变量或逻辑函数同时取代等式两端的任何一个逻辑变量后,等式依然成立。

(2) 反演规则

在逻辑代数中,常将 \bar{F} 叫做逻辑函数 F 的反函数或补函数。反演规则是将一个逻辑函数表达式 F 中所有“与”符号换为“或”符号;所有“或”符号换为“与”符号;所有原变量换为反变量;所有反变量换为原变量;“0”换成“1”,“1”换成“0”,所得新的逻辑表达式为原函数的反函数。获得反函数的规则就是反演规则。利用反演规则可以方便地求得函数的反函数 F 。

例 1 求函数 $F = ABC + A\bar{B}\bar{C} + A\bar{B}C + A\bar{B}\bar{C}$ 的反函数。

按照反演规则,得:

$$\bar{F} = (\bar{A} + \bar{B} + \bar{C})(\bar{A} + \bar{B} + C)(\bar{A} + B + \bar{C})(\bar{A} + B + C)$$

如果由逻辑代数的基本公式求 \bar{F} ,则有:

$$\begin{aligned} \bar{F} &= \overline{ABC + A\bar{B}\bar{C} + A\bar{B}C + A\bar{B}\bar{C}} \\ &= \overline{ABC} \cdot \overline{A\bar{B}\bar{C}} \cdot \overline{A\bar{B}C} \cdot \overline{A\bar{B}\bar{C}} \\ &= (\bar{A} + \bar{B} + \bar{C})(\bar{A} + \bar{B} + C)(\bar{A} + B + \bar{C})(\bar{A} + B + C) \end{aligned}$$

比较两种求反函数的方法,显然,应用反演规则要方便得多。

(3) 对偶规则

设 F 为一个逻辑表达式,若将 F 中的“与”符号换为“或”符号,将“或”符号换为“与”符号,将“1”换为“0”,将“0”换为“1”,所得新的逻辑函数表达式称为 F 的对偶式,记作 F' ,获得对偶式的规则称为对偶规则。如果两个逻辑函数表达式相等,那么它们的对偶式也一定相等。利用对偶式规则,可以帮助人们减少公式的记忆量。

例如 求 $F = A \cdot \overline{BC} + B \overline{CD}$ 的对偶式。

解: $F' = A + (\overline{B} + C) \cdot (\overline{B} + \overline{C} + D)$

2.1.3 布尔函数的化简与实现

1. 逻辑门电路的符号表示

将与、或、非三种基本的逻辑运算进行组合,可以得到各种形式的符号逻辑运算,其中常用的有:“与非”门、“或非”门、“与或”门、“与或非”门、“异或”门、“异或非”门等。本节就它们的逻辑关系及有关特性作一阐述。

(1)“与”门

二输入“与”门的逻辑函数表达式为: $F = A \cdot B$,它的图形符号如图 2.1 所示,真值表如表 2.3 所示。

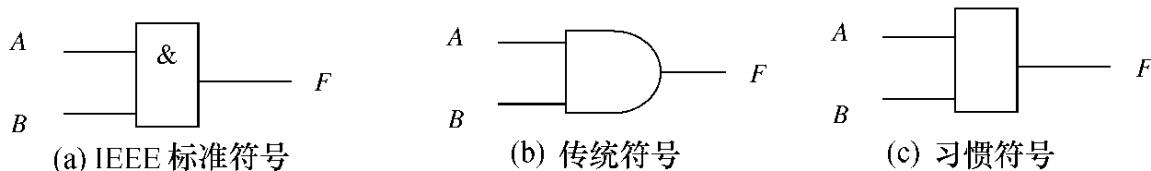


图 2.1 二输入“与”门

表 2.3 二输入“与”门真值表

A	B	F
0	0	0
1	0	0
0	1	0
1	1	1

(2)“或”门

二输入“或”门的逻辑函数表达式为: $F = A + B$,它的图形符号如图 2.2 所示,真值表如表 2.4 所示。

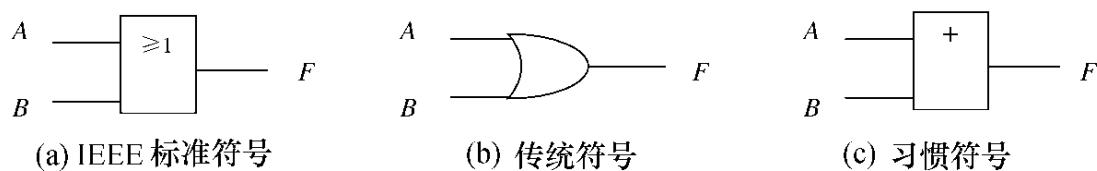


图 2.2 二输入“或”门

表 2.4 二输入“或”门真值表

A	B	F
0	0	0
1	0	1
0	1	1
1	1	1

(3)“非”门

二输入“非”门的逻辑函数表达式为： $F = \bar{A}$ ，它的图形符号如图 2.3 所示，真值表如表 2.5 所示。

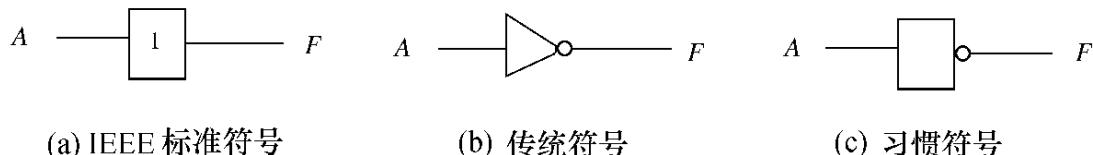


图 2.3 “非”门

表 2.5 二输入“非”门真值表

A	F
0	1
1	0

(4)“与非”门

二输入“与非”门的逻辑函数表达式为： $F = \overline{AB}$ ，它的图形符号如图 2.4 所示，真值表如表 2.6 所示。

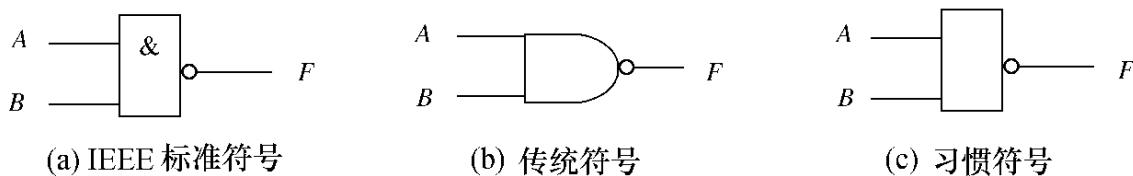


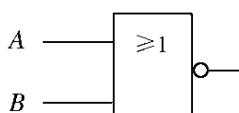
图 2.4 二输入“与非”门

(5)“或非”门

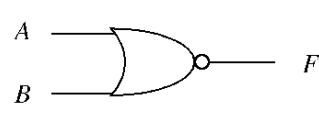
二输入“或非”门的逻辑函数表达式为： $F = \overline{A + B}$ ，它的图形符号如图 2.5 所示，真值表如表 2.7 所示。

表 2.6 二输入“与非”门真值表

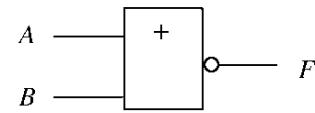
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0



(a) IEEE 标准符号



(b) 传统符号



(c) 习惯符号

图 2.5 二输入“或非”门

表 2.7 二输入“或非”门的真值表

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

2. 代数化简法

逻辑函数的代数表达式存在多种形式。对于同一个逻辑电路来说，尽管这些函数表达式描述的功能相同，但其电路实现时的复杂性和成本却各不相同。表达式越简单，实现电路也越简单，成本也将越低。逻辑函数有三种化简方法：代数化简法、卡诺图化简法、表格化简法。

代数化简法中常用的方法有：合并乘积项法、吸收项法和配项法。

(1) 合并乘积项法 利用互补律的公式，把“头部”相同的乘积项归为一类，使它们形成 $A + \bar{A} = 1$ 的形式，从而使函数化简。

例如：化简 $F = A(BC + \bar{B}\bar{C}) + ABC + A\bar{B}C$ 。

将函数表达式展开，得

$$F = ABC + A\bar{B}\bar{C} + AB\bar{C} + A\bar{B}C$$

合并有关乘积项得

$$\begin{aligned} F &= (ABC + A\bar{B}C) + (A\bar{B}\bar{C} + AB\bar{C}) \\ &= AC(B + \bar{B}) + A\bar{C}(\bar{B} + B) \\ &= AC + A\bar{C} \\ &= A(C + \bar{C}) \\ &= A \end{aligned}$$

(2) 吸收项法：利用吸收律和包含律等有关公式来减少“与”项数。

例如：化简 $F = A(B + C) + \bar{B}\bar{C}$

利用反演律，得 $F = A\bar{B}\bar{C} + \bar{B}\bar{C}$

用吸收律，得 $F = A + \bar{B}\bar{C}$

(3) 配项法：利用互补律 $A + \bar{A} = 1$ 配在乘积项上，然后再化简。

例如：化简 $F = A\bar{B} + B\bar{C} + \bar{B}C + \bar{A}B$

将 $\bar{B}C$ 配以 $(\bar{A} + A)$ ，将 $\bar{A}B$ 配以 $(\bar{C} + C)$ ，得：

$$\begin{aligned} F &= A\bar{B} + B\bar{C} + \bar{B}C(A + \bar{A}) + \bar{A}B(C + \bar{C}) \\ &= A\bar{B} + B\bar{C} + A\bar{B}C + \bar{A}\bar{B}C + \bar{A}BC + \bar{A}B\bar{C} \\ &= (A\bar{B} + A\bar{B}C) + (B\bar{C} + \bar{A}\bar{B}C) + (\bar{A}\bar{B}C + \bar{A}BC) \\ &= A\bar{B}(1 + C) + B\bar{C}(1 + \bar{A}) + \bar{A}C(\bar{B} + B) \\ &= A\bar{B} + B\bar{C} + \bar{A}C \end{aligned}$$

3. 卡诺图化简法

由于卡诺图的特殊结构，使得任意两个在几何位置上相邻或以中心轴对称的小方格代表的最小 / 最大项都有一个变量取值不同，这种特性称为相邻性。根据逻辑代数的吸收律，有

$$AB + A\bar{B} = A \quad (A + B)(A + \bar{B}) = A$$

即任意两个只有一个变量取值不同的最小项或最大项结合，都可以消去取值不同的那个变量而合并为一项，这就是卡诺图化简逻辑函数的原理。在阐述卡诺图之前，引入逻辑代数的两个重要概念：最小项和最大项。

(1) 最小项

设有 n 个变量，它们所组成的具有 n 个变量的“与”项中，每个变量或者以原变量或者以反变量的形式出现一次，且仅出现一次，这个乘积项称为最小项。

n 个变量应有 2^n 个最小项。例如 2 个变量 A, B ，有如下 4 个最小项： $\bar{A}\bar{B}$ 、 $A\bar{B}$ 、 $\bar{A}B$ 、 AB 。把最小项记作 m_i 。 i 是如下确定的：把乘积项的原变量记作“1”，反变量记作“0”，把每个乘积项表示为一个二进制数，这个二进制数所对应的十进制数就是 i 的值。例如： $\bar{A}\bar{B}$ 为 00，即为 0，所以写成 $\bar{A}\bar{B} = m_0$ 。2 变量的最小项为：

$$\bar{A}\bar{B} = m_0 \quad \bar{A}B = m_1$$

$$A\bar{B} = m_2 \quad AB = m_3$$

任何一个逻辑函数 F 都可用最小项之和（即逻辑“或”），来表示 n 个变量应有的 2^n 个最小项，它们不是包含在 F 的“与或”表达式中便是包含在 \bar{F} 的“与或”表达式中。

例如，一个 3 变量逻辑函数的表达式为：

$$F = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + \bar{A}BC + \bar{A}\bar{B}C$$

可变成 $F = m_2 + m_6 + m_3 + m_0 = \Sigma m^3(2, 6, 3, 0)$

这里“ Σ ”表示逻辑“或”运算， m^3 表示三变量的最小项。而 \bar{F} 则应包含除 m_2, m_6, m_3, m_0 之外的其余最小项：

$$\bar{F} = m_1 + m_4 + m_5 + m_7 = \Sigma m^3(1, 4, 5, 7)$$

表 2.8 列出 3 个变量全部最小项的真值表。

表 2.8 3 变量全部最小项真值表

ABC	m_0 ($\bar{A}\bar{B}\bar{C}$)	m_1 ($\bar{A}\bar{B}C$)	m_2 ($\bar{A}B\bar{C}$)	m_3 ($\bar{A}BC$)	m_4 ($A\bar{B}\bar{C}$)	m_5 ($A\bar{B}C$)	m_6 ($AB\bar{C}$)	m_7 (ABC)
000	1	0	0	0	0	0	0	0
001	0	1	0	0	0	0	0	0
010	0	0	1	0	0	0	0	0
011	0	0	0	1	0	0	0	0
100	0	0	0	0	1	0	0	0
101	0	0	0	0	0	1	0	0
110	0	0	0	0	0	0	1	0
111	0	0	0	0	0	0	0	1

从表 2.8 看出 最小项具有如下性质：

- ① 任一最小项，只有一组变量取值使其值为“1”。
- ② 任意两个最小项 m_i 和 m_j ，其逻辑“与”为“0”，即 $m_i \cdot m_j = 0$

- ③ n 个变量的全部最小项之逻辑“或”为“1”，即 $\sum_{i=0}^{2^n-1} m_i = 1$

- ④ 某一个最小项不是包含在函数 F 中，就是包含在反函数 \bar{F} 中。

(2) 最大项

设有 n 个变量，由它们组成的具有 n 个变量的“或”项中，每个变量以原变量或者以反变量的形式出现一次，且仅出现一次，这个“或”项称为最大项。

n 个变量有 2^n 个最大项。例如，两个变量的 4 个最大项为 $\bar{A} + \bar{B}, A + \bar{B}, \bar{A} + B, A + B$ 。最大项常以 M_i 来表示。 i 是这样确定的：把或项中的原变量记作“0”，反变量记作“1”，形成一个二进制数，此二进制数所对应的十进制数就是 i 。2 个变量的最大项为：

$$M_0 = A + B$$

$$M_1 = A + \bar{B}$$

$$M_2 = \bar{A} + B$$

$$M_3 = \bar{A} + \bar{B}$$

任何一个逻辑函数 F 都可用最大项之积来表示。通过一个实例来加以说明。

例：把 $F = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + \bar{A}BC + \bar{A}\bar{B}C = \Sigma m^3(2, 6, 3, 0)$ 以最大项之积来表示。

对 F 进行两次求反 ,并利用基本公式得 :

$$\begin{aligned}
 F &= \overline{\overline{F}} = \overline{\sum m^3(5,1,4,7)} \\
 &= \overline{A\bar{B}C + A\bar{B}\bar{C} + A\bar{B}\bar{C} + ABC} \\
 &= \overline{A\bar{B}C} \cdot \overline{A\bar{B}\bar{C}} \cdot \overline{A\bar{B}\bar{C}} \cdot \overline{ABC} \\
 &= (\bar{A} + B + \bar{C})(A + B + \bar{C})(\bar{A} + B + C)(\bar{A} + \bar{B} + \bar{C}) \\
 &= M_5 \cdot M_1 \cdot M_4 \cdot M_7 = \prod M^3(5,1,4,7)
 \end{aligned}$$

这里“ \prod ”表示逻辑“与”运算, M^3 表示 3 变量的最大项。由该例可知 ,一个以最小项表示的逻辑函数 F 转换成以最大项表示的方法如下 :先把 \bar{F} 以最小项表示 ,然后取与最小项有相同下标的最大项进行逻辑“与”,即可得 F 的最大项表示形式。 n 个变量的 2^n 个最大项中的某一个 ,不是包含在函数 F 的“或与”表达式中 ,便是包含在 \bar{F} 的“或与”表达式中。

表 2.9 列出 3 个变量 A, B, C 的全部最大项的真值表。

表 2.9 三变量全部最大项真值表

ABC	M_0 ($A + B + C$)	M_1 ($A + B + \bar{C}$)	M_2 ($A + \bar{B} + C$)	M_3 ($A + \bar{B} + \bar{C}$)	M_4 ($\bar{A} + B + C$)	M_5 ($\bar{A} + B + \bar{C}$)	M_6 ($\bar{A} + \bar{B} + C$)	M_7 ($\bar{A} + \bar{B} + \bar{C}$)
000	0	1	1	1	1	1	1	1
001	1	0	1	1	1	1	1	1
010	1	1	0	1	1	1	1	1
011	1	1	1	0	1	1	1	1
100	1	1	1	1	0	1	1	1
101	1	1	1	1	1	0	1	1
110	1	1	1	1	1	1	0	1
111	1	1	1	1	1	1	1	0

从真值表看出 :

- ① 任意一个最大项 ,只有一组变量取值使其为“0”。
- ② 任意两个最大项 M_i 和 M_j ,其逻辑“或”必为“1”,即 $M_i + M_j = 1$ 。
- ③ n 个变量的全体最大项之“与”必为“0”,即 $\prod M_i = 0$ 。
- ④ 某一个最大项不是包含在函数 F 中 ,便是包含在反函数 \bar{F} 中。

(3) 最小项和最大项的关系

- ① 相同 i 的最小项和最大项为互补的

例如 ,对 3 变量来说 , $m_3 = \bar{A}\bar{B}C$, $M_3 = A + \bar{B} + \bar{C}$ 。显然 , $m_3 = \bar{M}_3$; $M_3 = \bar{m}_3$ 。推广而言 ,便有 : $M_i = \bar{m}_i$; $m_i = \bar{M}_i$

- ② $\sum m_i$ 和 $\prod M_i$ 互为对偶式

例如 ,对两个变量来说 ,有 4 个最小项为 $\bar{A}\bar{B}$ 、 $A\bar{B}$ 、 $\bar{A}B$ 、 AB 。那么 ,全体最小项

之和为：

$$\sum_{i=0}^{2^n-1} m_i = \bar{A}\bar{B} + A\bar{B} + \bar{A}B + AB$$

求其对偶式为

$$\left(\sum_{i=0}^{2^n-1} m_i \right)' = (\bar{A} + \bar{B})(A + \bar{B})(\bar{A} + B)(A + B) = \prod_{i=0}^{2^n-1} M_i$$

同理： $\prod_{i=0}^{2^n-1} M_i$ 的对偶式为：

$$\begin{aligned} \left(\prod_{i=0}^{2^n-1} M_i \right)' &= [(\bar{A} + \bar{B})(A + \bar{B})(\bar{A} + B)(A + B)] \\ &= \bar{A}\bar{B} + A\bar{B} + \bar{A}B + AB = \sum_{i=0}^{2^n-1} m_i \end{aligned}$$

(4) 卡诺图

卡诺图是由真值表变换而来的，真值表有多少行，卡诺图就有多少个小方格。卡诺图上的每一个小方格代表真值表上的一行，因而也代表一个最小项或一个最大项。

2变量卡诺图的结构如图2.6(a)所示，每个小方格中的数字表示该小方格所示的真值表中的行号，行号实际上就是真值表中自变量取值的等值十进制数，因而也就是它所代表的最大项或最小项的下标。2变量、3变量、4变量以及5变量的卡诺图分别示于图2.6(a)、(b)、(c)、(d)。

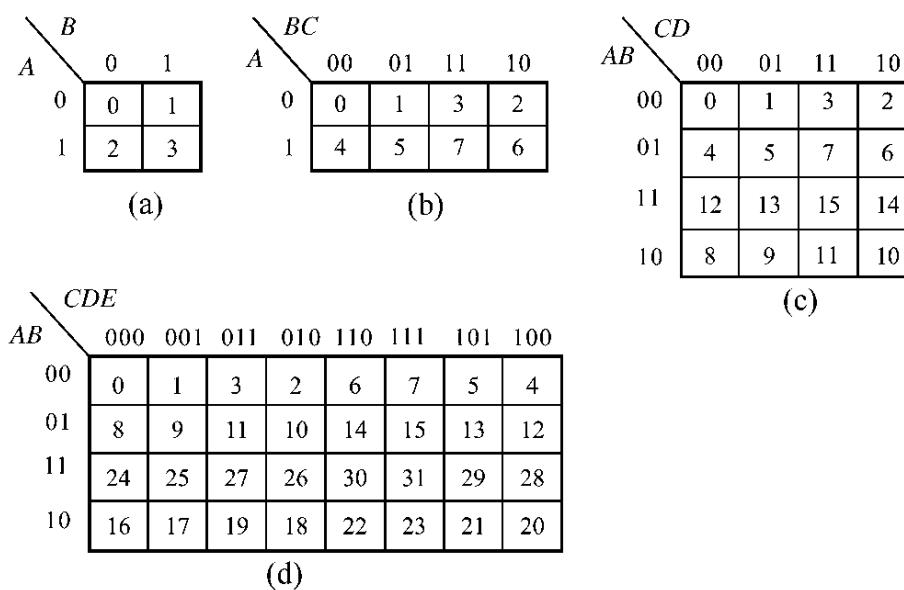


图 2.6 简化 2、3、4、5 变量卡诺图

(a) 2变量 (b) 3变量 (c) 4变量 ;(d) 5变量

从分析画出的卡诺图可知，其最大的两个特点是：两个相邻最小项只有一

个变量是互为相反的,而其余变量是相同的。如4变量卡诺图上的 m_5 和 m_7 为相邻项,其中 $m_5 = \bar{A}\bar{B}\bar{C}D$, $m_7 = \bar{A}B\bar{C}D$,它们只有 C 是互为相反变量,其余变量都是相同的,称 m_5 、 m_7 在逻辑上是相邻的。两个相邻的最小项叠加后可以消去一个变量。仍以4变量卡诺图的 m_5 、 m_7 为例:

$$\begin{aligned} m_5 + m_7 &= \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}D \\ &= \bar{A}BD(\bar{C} + C) = \bar{A}BD \end{aligned}$$

叠加后,消去了一个变量 C ,成为3变量“与”项。

常常把代表一个最小项的小方块叫0维块;两相邻最小项合并后所构成的块叫1维块,1维块比0维块少一个变量;把两个相邻的1维块合并所构成的块叫2维块,2维块比1维块少一个变量。图2.7给出了三变量卡诺图中一些维块的位置。

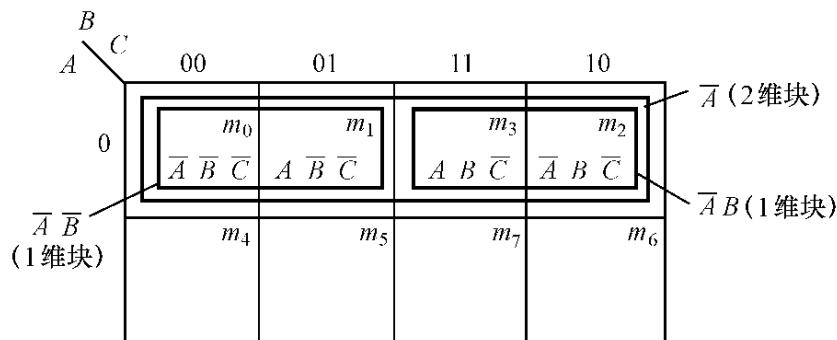


图 2.7 维块的构成

由上面分析可知,把相邻的低维块合并成高维块可以消去变量。 m 维块所表示的“与”项可以比最小项减少 m 个变量。卡诺图化简函数正是利用了这一原理。

(5) 逻辑函数的卡诺图表示及其卡诺图化简

用卡诺图来化简逻辑函数,首先要把逻辑函数在卡诺图上画出。然后根据合并相邻块可消去变量的思路,进行逻辑函数的化简。

如果逻辑函数以最小项之和的形式给出,那么,首先根据变量数画出对应的变量卡诺图框,在卡诺图中找到函数所包含的每一个最小项,并在对应的小方块上填1,其余小方块中均填0或不填任何标记。

然后,利用卡诺图化简函数,尽量把小块合并成大块。例如,图2.8(a)所示3变量卡诺图有5个“1”,用小方格1、3、5、7构成的大格才是最大的,剩下的最后一个小方格6,由它和7构成的方格才是包含6的最大格。只有用这两个大格(1、3、5、7和6、7)(图2.8(b))才能写出最简“与或”表达式: $F = C + AB$ 。

例1 化简逻辑函数 $F = \sum m(0, 1, 2, 4, 6, 7)$

① 画出3变量卡诺图框,如图2.9所示。

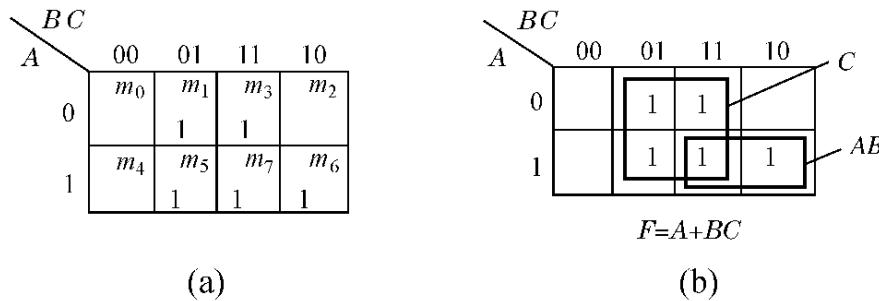
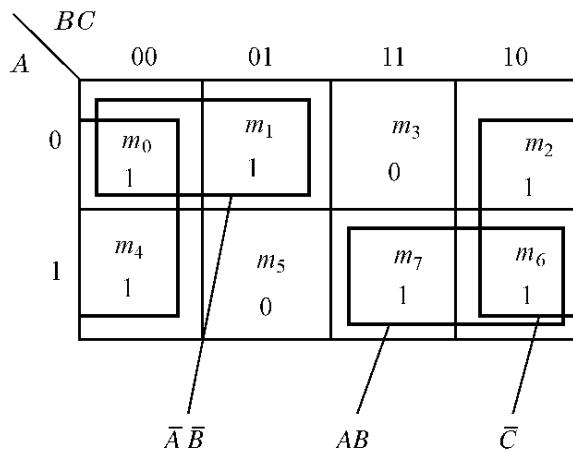


图 2.8 3 变量函数的卡诺图表示

② 找出函数的最小项 $m_0, m_1, m_2, m_4, m_6, m_7$ 的对应小方块，并填 1，其余的小方块上填 0。

③ 合并填 1 的相邻小方块。其中 m_0 和 m_1 为相邻项，可包一个圈，叠加后可消去 C ，为 $\bar{A}\bar{B}$ ； m_6, m_7 为相邻项，叠加后可消去 C ，为 AB ； m_0, m_2, m_4, m_6 为“相对”，也包一圈，可消去 B, A ，得 \bar{C} 。所以函数化简结果为 $F = \bar{C} + \bar{A}\bar{B} + AB$ 。

图 2.9 F 的卡诺图表示

$$\text{化简函数 } F = \overline{(A \oplus B) \cdot (C + D)}$$

先将函数 F 变换成“与或”表达式：

$$\begin{aligned} F &= \overline{(A \oplus B) \cdot (C + D)} \\ &= \overline{(A \oplus B)} + \overline{(C + D)} \\ &= \overline{\bar{A}B + A\bar{B}} + \overline{C + D} \\ &= \bar{A}\bar{B} + AB + \bar{C}\bar{D} \end{aligned}$$

乘积项 $\bar{C}\bar{D}$ 包含着最小项 m_0, m_4, m_8, m_{12} ； AB 包含着最小项 $m_{12}, m_{13}, m_{14}, m_{15}$ ； $\bar{A}\bar{B}$ 包含着最小项 m_0, m_1, m_2, m_3 。画出卡诺图，如图 2.10 所示。寻找相邻项化简，得 $F = \bar{C}\bar{D} + AB + \bar{A}\bar{B}$ 为最简式。

卡诺图化简逻辑函数的关键是在卡诺图上寻找相邻项，并尽量将它们合并

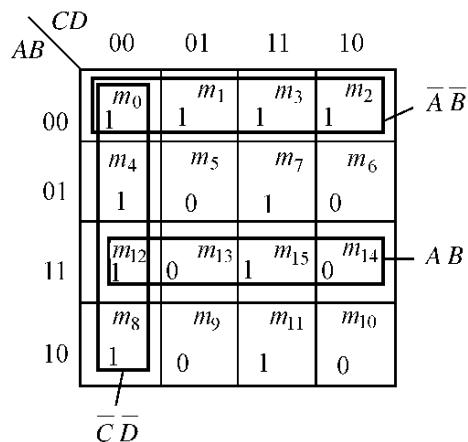


图 2.10 4 变量卡诺图化简

为大区域块(高维块)相邻块的维数越高,消去的变量愈多。此外,特别要注意的是同一区域可以重复使用多次,即同一区域块可以包含在多个更大的区域块中。函数的所有最小项必须包含在合并的块中,合并不到大块中的最小项不能丢掉。当函数变量数较少时,卡诺图化简法的优点是明显的,即直观、简便、快捷,因而在实践中得到很广泛的应用。但是当变量数较多(在4个以上)时,就很不方便,而且还容易出错。

4. 表格法化简输出逻辑函数

对于变量较多的逻辑函数化简可以采用表格化简法。表格法化简逻辑函数和卡诺图化简逻辑函数的基本思想是相同的,即在两个“与”项中,只要有一个变量互补,而其余变量均相同,则合并这两个“与”项即可消去一个变量,从而形成一个新的较简的“与”项。

表格法化简有两个步骤:

- (1) 求出函数全部的质蕴涵项;
- (2) 从质蕴涵项中选出必要质蕴涵项。

例如:化简逻辑函数 $F = f(A, B, C, D) = \sum m(0, 1, 2, 5, 6, 7, 11, 13, 15)$

(1) 求全部质蕴涵项

先将各最小项 m_i 的下标 i 用二进制数表示,然后将二进制数取值中“1”的个数由少至多分组排列。如表 2.10 所示,表中组号即为“1”的个数。

求质蕴涵项主要是运用公式 $AB + A\bar{B} = A$ 。由表 2.10 可看出,满足该等式的两个相邻最小项只能在相邻组内出现。首先,在相邻组间进行搜索,寻找相邻项。先将表 2.10 中 0 组的最小项和 1 组的两个最小项逐一比较,如是相邻项,合并后为“00-0”,即表示此“与”项为“ $\bar{A}\bar{B}\bar{D}$ ”,用“-”表示新的“与”项中消去的变量。由于新“与”项包含了最小项 0 与 2,所以在表 2.10 中最小项 0 或 2 的右侧打上“√”,表示 $\bar{A}\bar{B}\bar{D}$ 能替代这两个最小项。接着逐个比较 1 组和 2 组的最小项,寻找

相邻项 把所有存在于相邻组间的相邻项都找到后 ,即可得到一组含有 3 个变量的“与”项 ,列于表 2.11。

用同样的方法再在表 2.11 各组间寻找相邻项。例如在表 2.11 中 ,0 组和 1 组没有相邻项 ;1 组和 2 组也没有相邻项 ;2 组和 3 组中 ,只有 2 组的“ -101 ”和 3 组的“ -111 ”相邻 ,它们可合并成一个新的与项“ -1-1 ” ,列于表 2.12 ,并在该两相邻项旁打“ √ ”。表 2.11 中不打“ √ ”的各项是无法合并的。在表 2.12 中只有一个乘积项 ,它是无法合并项。把函数“与或”表达式中的每个“与”项称为蕴涵项 ,把表 2.10、2.11、2.12 中无法再合并的蕴涵项称为质蕴涵项 ,在表 2.11 和 2.12 中 ,分别记作 $P_1 \sim P_7$ 。

表 2.10

组号	最小项	变量	$ABCD$
	编号		
0	0	0000	✓
	2	0010	✓
1	1	0001	✓
	6	0110	✓
2	5	0101	✓
	13	1101	✓
3	11	1011	✓
	7	0111	✓
4	15	1111	✓

表 2.11

组号	最小项	变量	$ABCD$
	编号		
0	0 2	00 - 0	P_1
	0 1	000 -	P_2
1	2 6	0 - 10	P_3
	1 5	0 - 01	P_4
2	5 ,13	- 101	✓
	5 ,7	01 - 1	P_5
3	11 ,15	1 - 11	P_6
	7 ,15	- 111	✓

表 2.12

组号	最小项	变量	$ABCD$
	编号		
2	5 ,13	- 1 - 1	P_7
	7 ,15		

对于本例 , F 的全部质蕴涵项为 :

$$P_1 = \bar{A}\bar{B} - \bar{D} \quad P_2 = \bar{A}\bar{B}\bar{C} -$$

$$P_3 = \bar{A} - \bar{C}\bar{D} \quad P_4 = \bar{A} - \bar{C}D$$

$$P_5 = \bar{A}\bar{B} - D \quad P_6 = A - CD$$

$$P_7 = - B - D$$

(2) 选出必要质蕴涵项

先列出全部质蕴涵项(表 2.13) ,表的每一行对应一个质蕴涵项 ,每一列对应一个最小项 ,把每个质蕴涵项所包含的最小项在表中打“ × ”。例如 , P_4 包含最小项 1 和 5 ,则就在 P_4 行的 m_1 、 m_5 列上打“ × ”。把这张表称为质蕴涵表。

表 2.13

$P \backslash m_i$	m_0	m_1	m_2	m_5	m_6	m_7	m_{11}	m_{13}	m_{15}
P_1	×	×							
P_2	×	×							
P_3			×		×				
P_4			×		×				
P_5				×		×			
P_6						×		×	
P_7			×		×		×	×	×

表 2.14

$P \backslash m_i$	m_0	m_1
P_1	×	
P_2		×
P_4		×

为寻找必要质蕴涵项,先寻找出哪些最小项仅仅属于一个质蕴涵项。在表 2.13 中, m_6 属于 P_3 , m_{11} 仅属于仅 P_6 , m_{13} 仅属于 P_7 。如果质蕴涵表的一列中只有一个“ \times ”,那么就表示此最小项仅属于一个质蕴涵项,因而该质蕴涵项就一定是必要的。由于 P_3 、 P_6 、 P_7 从质蕴涵表中删去,又因为 P_3 不仅包含 m_6 ,还包含 m_2 ; P_6 不仅包含 m_{11} ,还包含 m_{15} ; P_7 不仅包含 m_{13} ,还包含 m_5 、 m_7 、 m_{15} ,因而在从表 2.13 中删去 P_3 、 P_6 、 P_7 的同时,可把它们所包含的最小项 m_2 、 m_5 、 m_6 、 m_7 、 m_{11} 、 m_{13} 、 m_{15} 均删去,这样表 2.13 就可简化为表 2.14。下面就是要在 P_1 、 P_2 、 P_4 中选出其余的必要质蕴涵项,使之包含 m_0 、 m_1 。

有两个办法从简化质蕴涵表中选取必要质蕴涵项:表达式法与行列消去法。

① 表达式法

由表 2.14 可知, m_0 包含在 P_1 或 P_2 中,即质蕴涵的组合为($P_1 + P_2$), m_1 包含在 P_2 或 P_4 中,即($P_2 + P_4$)。要同时包含 m_0 、 m_1 的质蕴涵组合为:

$$(P_1 + P_2) \cdot (P_2 + P_4) = P_1P_2 + P_1P_4 + P_2 + P_2P_4$$

此式表明要包含 m_0 、 m_1 有 4 种选取方法:或选取 P_1 与 P_2 或选取 P_1 与 P_4 ;或选取 P_2 或选取 P_2 与 P_4 。显然,为了满足最简“与或”式的要求,选 P_2 为必要质蕴涵项。因此用表格法化简的 F 最简式为:

$$F = P_2 + P_3 + P_6 + P_7 = \overline{ABC} + \overline{ACD} + ACD + BD$$

② 行列消去法

此法用于简化质蕴涵表中行数和列数都比较多的情况。以表 2.15 所示的简化质蕴涵表为例。

先进行行消去。检查表 2.15 发现, P_5 行所包含的 m_4 也包含在 P_4 之中,而且 P_4 除包含 m_4 外,还包含了 m_6 ,因此若选 P_4 就不必再考虑 P_5 了,这样 P_5 行就可从表 2.15 中删去。也就是说,如果有两行(P_i 和 P_j 行),其中 P_i 行的“ \times ”全部包

含在 P_j 行中, 那么 P_i 行就可从表中删去。同理, P_6 所包含的 m_{10} 也是 P_3 所包含的, 因此 P_6 行也可删去。这样, 行消去后的简化质蕴涵表如表 2.16 所示。这就是行消去。

表 2.15

m_i	m_2	m_4	m_6	m_{10}
P				
P_2	×		×	
P_3	×			×
P_4		×	×	
P_5	×			
P_6				×

表 2.16

m_i	m_2	m_4	m_6	m_{10}
P				
P_2	×		×	
P_3	×			×
P_4		×	×	

检查表 2.16 发现, m_4 列的“ \times ”完全包含在 m_6 列中, 当考虑 m_4 而选择 P_4 时, P_4 必然同时包含了 m_6 , 因此可以把 m_6 列删去。同理 m_{10} 列的“ \times ”也完全包含在 m_2 中, 因此可把 m_2 删去。也就是说, 如果 m_i 列中记有“ \times ”号的各行中, 在 m_j 中也记有“ \times ”号, 那么 m_j 列就可以删去。这就是列消去。

经行、列消去后剩下的 P_3 、 P_4 就是必要质蕴涵项。

对于复杂的质蕴涵表, 行列消去法要反复进行。行、列消去次序并不影响简化结果。

通常可以将表达式法和行列消去法结合进行。先用行列消去法把质蕴涵项尽量简化, 然后再用表达式法。行列消去法对未简化的质蕴涵表也适用。

2.2 典型的组合电路

数字逻辑电路分为两大类: 一类是组合逻辑电路, 它的输出只和当时的输入逻辑状态有关, 而和电路过去的状态无关; 另一类是时序逻辑电路, 它的输出不仅和当时的输入逻辑状态有关, 而且还和电路过去的状态有关。本章将介绍组合逻辑电路的原理及应用。

目前在数字系统中使用的组合逻辑电路, 按照用途来分, 大体有以下几种:

- (1) 译码器;
- (2) 数据选择器和数据分配器;
- (3) 加法器;
- (4) 带有进位的加法器;

(5) 乘法器。

下面分别介绍这些组合逻辑电路。

2.2.1 译码器

译码器是计算机中最常用的逻辑部件之一,用来完成对操作码的译码。图2.11是一个由与非门组成的译码器,它能对3个输入信号进行译码。

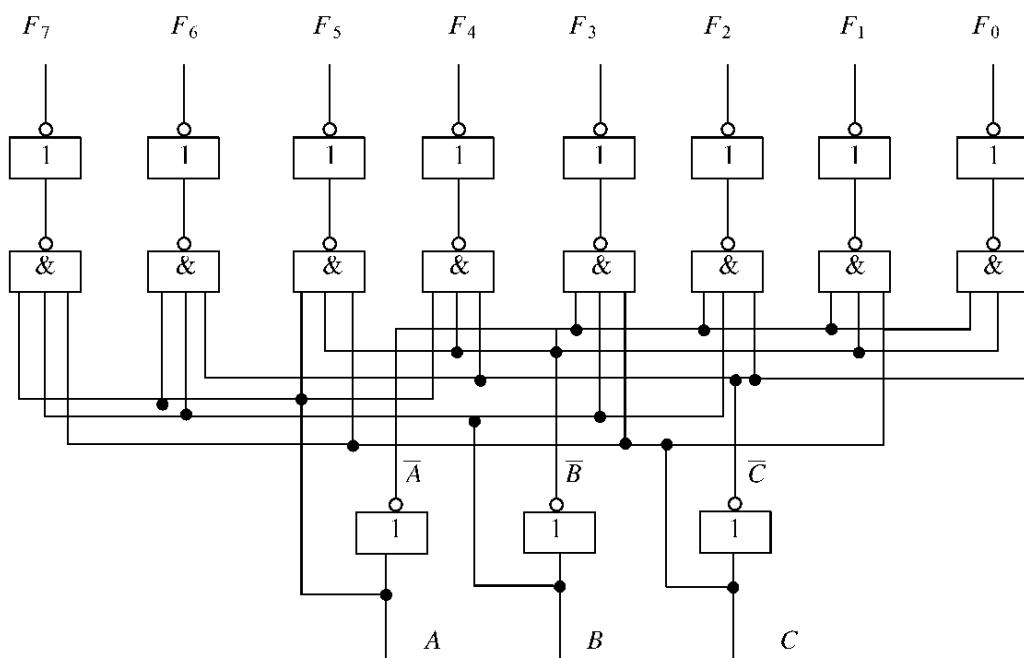


图 2.11 译码器

由图可以写出各输出函数的逻辑表达式：

$$\begin{array}{ll}
 F_0 = \bar{A}\bar{B}\bar{C} & F_1 = \bar{A}\bar{B}C \\
 F_2 = \bar{A}B\bar{C} & F_3 = \bar{A}BC \\
 F_4 = A\bar{B}\bar{C} & F_5 = A\bar{B}C \\
 F_6 = AB\bar{C} & F_7 = ABC
 \end{array}$$

根据该逻辑表达式可以写出译码器的真值表(表2.17)。由表可知,当输入 $ABC = 000$ 时,只有 $F_0 = 1$,其他输出都为0;当输入 $ABC = 001$ 时,只有 $F_1 = 1$,依次类推,从而实现了将二进制代码译为某一条输出线上的高电平。

译码器的种类很多,按输入、输出信号的数目可将译码器分为多一译码器,一多译码器及多多译码器。多一译码器是一种将某一时刻的多个输入信号译为一个输出信号的译码器;一多译码器与其相反,它是将某一时刻的一个输入信号译为多个输出信号的译码器;多多译码器是一种将某一时刻的多个输入信号译为多个输出信号的译码器。

表 2.17 译码器的真值表

A	B	C	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

译码器也叫做信号分离器。信号分离器是根据一个选择器信号 s 的值 , 把一个源信号 x 赋值给多个目标信号 y_i 。显然 , s 是一个数 , 表示下标。为了得到一个信号分离器电路 , 必须规定一个用数字信号对整数进行编码的方法。标准的编码方法被称为二进制编码。它的基本设想是 , 任一信号的数值(“ 0 ”和“ 1 ”)由一个二进制数字(Bit)的数值来表示 , 在总和 s 中 , 每个信号(s_0, s_1, \dots)均被加权。也就是

$$s = s_0 \times 2^0 + s_1 \times 2^1 + s_2 \times 2^2 + \dots + s_i \times 2^i + \dots$$

那么信号分离器的函数表达式如下 :

$$y_i = (\text{if } i = s \text{ then } x \text{ else } 0) \quad y_i = x * (i = s)$$

最终的二输出电路和信号分离器的符号表示如图 2.12 所示。

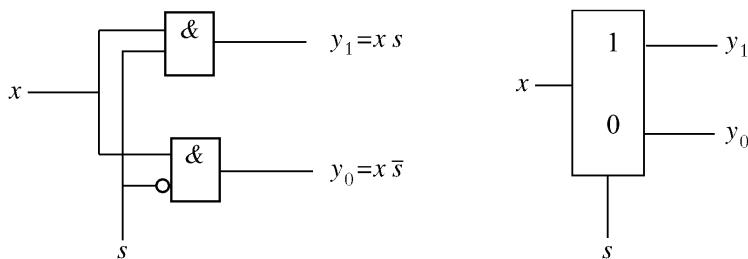


图 2.12 信号分离器的电路及符号

为了得到有很多输出的信号分离器 , 使用了重要的级联技术。实际上 , 一个有 2^n 个输出的信号分离器很容易通过级联 n 级二输出的信号分离器来获得的。这点如图 2.13 所示 , 通常称为 n 到 2^n 的信号分离器。

如果输入 x 保持常值 1 , 那么 $y_s = 1$, 其他所有输出为 0。在这种情况下 , 信号分离器成为 s 的译码器(x 被称为使能信号)。

2.2.2 多路开关(多路选择器)

与信号分离器相反 , 多路开关的功能是根据一个选择信号 s , 从若干个源信

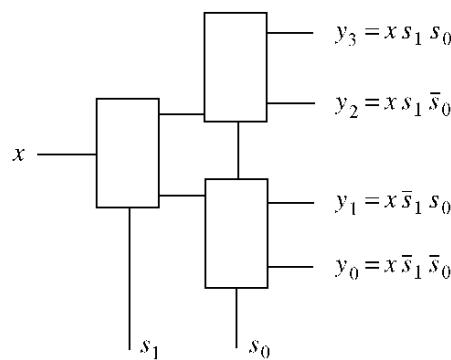


图 2.13 信号分离器的级联

号 x_i 选择一个成为目标信号 y 。多路开关因此也被称为多路选择器，多路转换函数定义为 $y = x_i$ 。

仍旧从存在 2 个源信号的简单情况开始考虑，这里 $s = s_0$, $y = x_i$ 被转换为 $y = \text{MUX}(s, x_0, x_1) = (\text{if } s \text{ then } x_1 \text{ else } x_0) = x_0 * \sim s + x_1 * s$ 二输入多路开关和它的符号表示如图 2.14 所示。

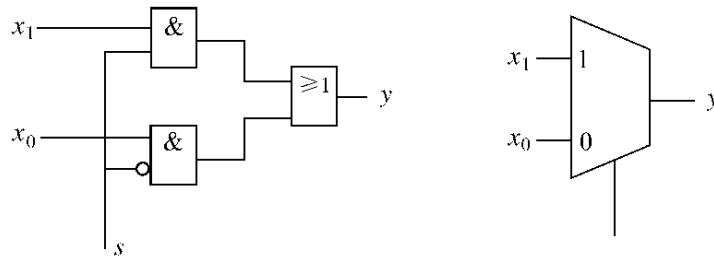


图 2.14 2 - to - 1 多路选择器的逻辑电路及符号

通过级联 n 级二输入多路开关，可以得到一个有 2^n 个输入的多路开关（见图 2.15）。这个过程有复合函数相对应。

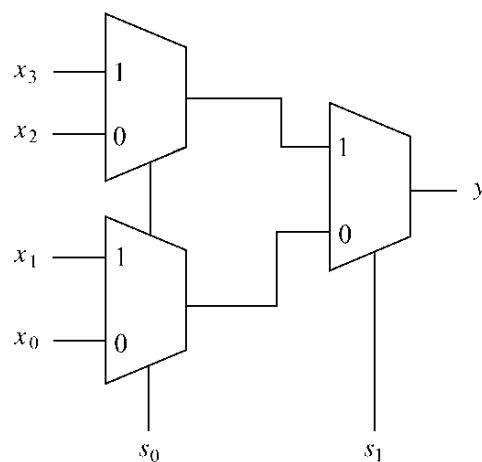


图 2.15 多路选择器的级联

2.2.3 加法器

加法器是基于整数的二进制编码的标准元件。二进制编码的优点是能用同样的电路来处理所有的数字。因此，我们从最简单的情况入手，讨论两个 1 位二进制数相加的情况，由于其最大和是 2，显然需要两个二进制位（信号）来表示，其真值表见表 2.18 所示。

表 2.18 中，最后得到的两个信号为 s （和） c （进位），不难看出，它们分别与“异或”和“与”这两个基本函数相对应，即

$$s = x \oplus y \quad c = xy$$

表 2.18 加法器的真值表

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

这种电路被称为半加器，如图 2.16 所示。

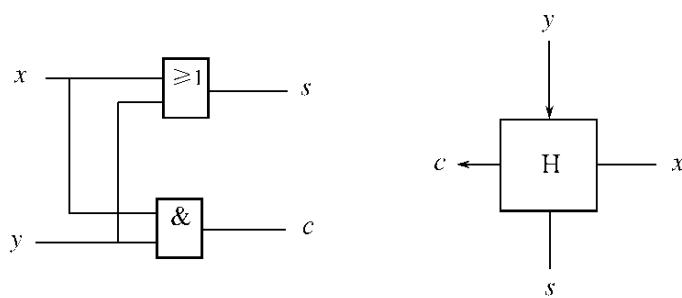


图 2.16 半加器的逻辑电路及符号

半加器级联产生增量器（incrementer）。在一个多位数字系统里，高位的进位输入是其低位的进位输出。因此，某一位的进位输出要连接到其上一位半加器的进位输入上（见图 2.17）。增量器把最低位的进位输入与输入 $x_0 \dots x_{n-1}$ 相加，也就是在 $x_0 \dots x_{n-1}$ 上加 0 或 1。

显然，全加器由能够使 3 个二进制输入相加的元件组成，即两个算子 x 和 y ，还有从其低一级元件来的进位。全加器元件可以通过连接两个半加器构成，如图 2.18 所示。进位输出函数根据 8 种可能的输入组合，通过辅助变量 $u = x \oplus y$, $v = xy$ 和 $w = uc_{in}$ 得到，即 $s = u \oplus c_{in}$, $c_{out} = w + v$ 。

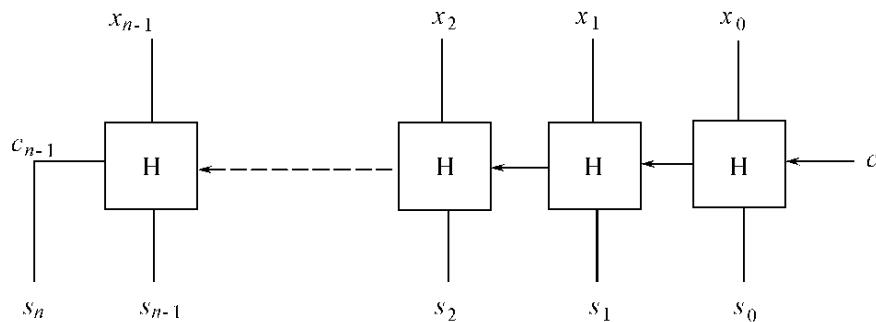


图 2.17 增量器

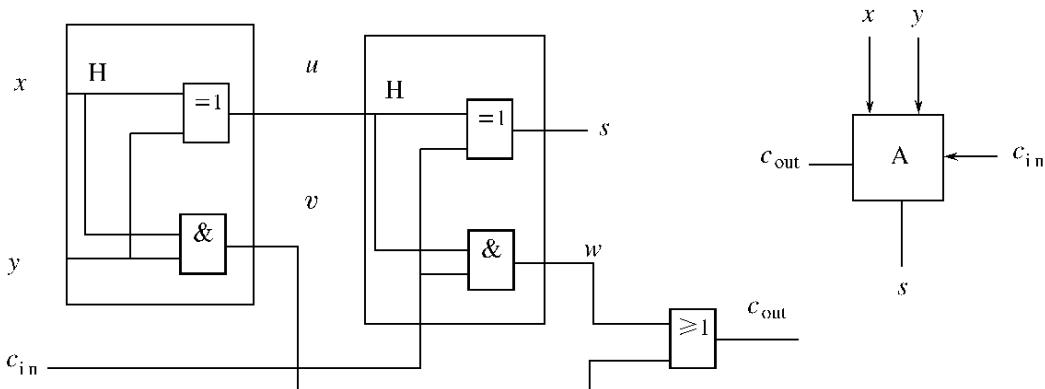


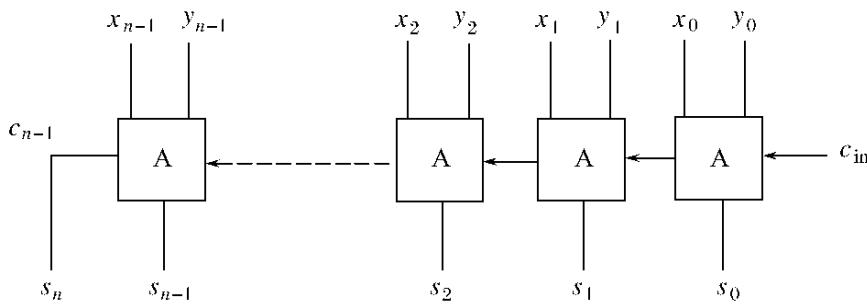
图 2.18 全加器元件

表 2.19 全加器的真值表

x	y	c_{in}	u	v	w	c_{out}	s
0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	1
1	0	0	1	0	0	0	1
1	1	0	0	1	0	1	0
0	0	1	0	0	0	0	1
0	1	1	1	0	1	1	0
1	0	1	1	0	1	1	0
1	1	1	0	1	0	1	1

如图 2.19 所示，两个 n 位加数的加法器是由 n 个全加器元件级联得到的，将每一位的进位输出连接到其上一位的进位输入上，并让 $c_{in} = 0$ 。如前所述，得到最高位的进位是通过了 n 个元件，也就是 $3n$ 个门电路。因为由进位所引起的延迟是影响整个电路运算速度一个关键因素，因此，对于较大的 n ，这个通路的长度很显然就成为了整个电路的关键。

减法器的设计依赖于负数表示方法的选择。理想的形式是让按位的加权与

图 2.19 n 位加法器

数字的符号无关,因为这样就可以根据等式 $x - y = x + (-y)$,使用相同的电路作为加法器和减法器。二进制的补码表示法满足了这一点,它获得了广泛而高效的应用。记数法将最高位 x_{n-1} 设为符号位,为了表示数 x 是负数, x_{n-1} 必须为 1。如果 x_{n-1} 是 0,所有位都是正的,因此 x 也是正的。

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \dots + x_12^1 + x_02^0$$

负数在补码中是通过将每一位二进制数在逻辑上取反得到的,即把所有的 x_i 用 $1 - x_i$ 代替,并加 1(进位位 c_{in} 作此用途):

$$\begin{aligned} & -(1 - x_{n-1})2^{n-1} + (1 - x_{n-2})2^{n-2} + \dots + (1 - x_1)2^1 + (1 - x_0)2^0 + 1 = \\ & -(1 - x_{n-1})2^{n-1} + 2^{n-1} - (x_{n-2}2^{n-2} + \dots + x_12^1 + x_02^0) = \\ & -(-x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \dots + x_12^1 + x_02^0) = -x \end{aligned}$$

如果和(或差)无法用 n 位数字表示,就产生了溢出。在两个无符号数的加法里一个值 1 的进位输出的产生就意味着溢出,但这不适合有符号数运算的情况。而有符号算术的溢出是通过比较最高位(符号位)进位与次高位进位的异同来表示。很明显,一个异或门就可以满足这个功能。

无符号算术: $ov = c_{n-1}$

有符号算术: $ov = c_{n-1} \oplus c_{n-2}$

2.2.4 带有快速进位生成的加法器

如前所述,加法器有一个特性:进位输出的产生是从最低位开始,直到通过所有的加法器元件。这就说明信号延迟与所包含的位数 n 成线性关系,并且 n 越大,电路运算速度越慢。已知在理论上每个布尔函数都可以表示成规范形式,使用一级多输入与门,跟着一级多输入或门。因此使得最小路径长度为 2,并且与函数的复杂性无关。然而,当电路具体实现时,门电路的数量却成 n 的指数倍增长,使得这个结果在实际中难以应用。

快速进位生成法是一种实用的解决方法,它能显著缩短路径长度,同时只需

少量地增加门电路数量。它是基于下面的观察得出的：

在加法器元件链的每一级中，或者产生进位，或者传递进位。如果两个加数都是 1，就会产生进位，定义 $g = xy$ ；如果两个加数中只有一个的值是 1，就传递进位，定义 $p = x + y$ 。因此，每一级中，定义进位位为

$$c_i = g_i + p_i c_{i-1}$$

其中， $g_i = x_i y_i$ ， $p_i = x_i + y_i$ 。

运用这种思想，设计一个 4 位数字的加法器，这里只注意进位的产生，可以得到：

$$c_0 = g_0 + p_0 c_{\text{in}}$$

$$c_1 = g_1 + p_1 c_0 = g_1 + p_1 g_0 + p_1 p_0 c_{\text{in}}$$

$$c_2 = g_2 + p_2 c_1 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_{\text{in}}$$

$$c_3 = g_3 + p_3 c_2 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_{\text{in}}$$

最终得到的新的加法器由 4 个元件组成，分别产生一个和信号、一个生成进位信号、一个传递进位信号和一个快速进位生成器，如图 2.20 所示。

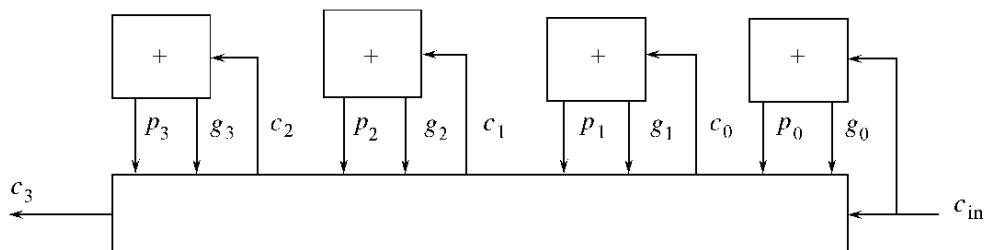


图 2.20 快速进位生成器

就像加法器元件的情况一样，现在也让快速进位生成器本身产生一个传递和一个发生信号，代替进位输出，这就是：

$$c_3 = g + p c_{\text{in}}$$

从 c_3 的表达式中得到 p 和 g 为

$$g = g_3 + p_3 c_2 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$$

$$p = p_3 p_2 p_1 p_0$$

使用这种替换，这个元件目前已具有级联能力。例如使用两级快速进位生成器构成的 16 位数字快速加法器（图 2.21）和使用三级快速进位生成器构成的 64 位数字快速加法器。显然，进位通道的长度只以 n 的对数级增加。这样即使是最严格的要求也是可以接受的。一般来说，4 位加法器是包含第一级快速进位生成器的完整元件，而一个 16 位加法器是由 4 个这样的元件和一个快速进位生成器构成的。

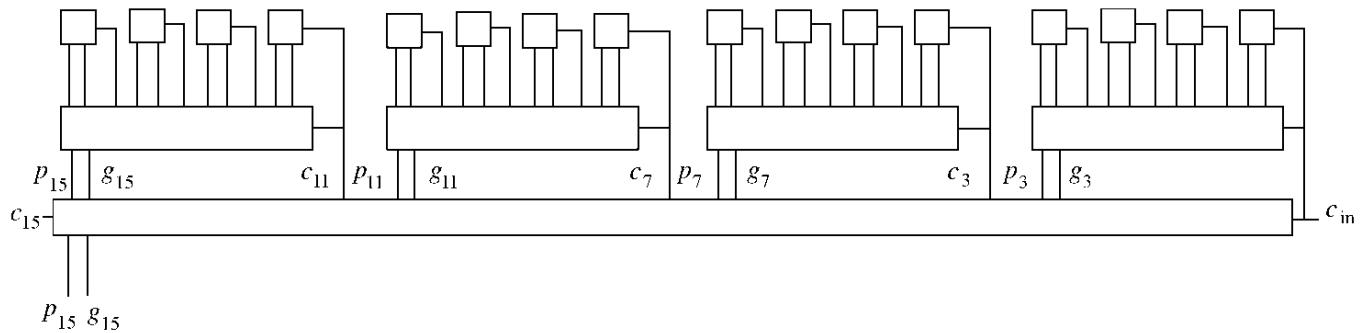


图 2.21 快速进位生成器的级联

2.2.5 乘法器

乘法比加法复杂，因此电路在实现乘法方面也应该比加法明显复杂得多。运用数字的多项式表示形式，把乘法转换为 n 个加法。其中 n 是乘法器中乘数的位数。

$$x = x_{n-1} \times 2^{n-1} + \dots + x_2 \times 2^2 + x_1 \times 2^1 + x_0 \times 2^0$$

$$x \times y = x_{n-1} \times y \times 2^{n-1} + \dots + x_2 \times y \times 2^2 + x_1 \times y \times 2^1 + x_0 \times y \times 2^0$$

由此把两个 n 位算子的乘法转换为 $n - 1$ 个 n 位算子的加法，再加上 n 个 n 位乘数 y 与一个 1 位被乘数 x_i 的乘积。后者可证实为与运算。这个思路在图 2.22 中表示出来，其中权值(2 的幂)表示为被乘数的适当移位。

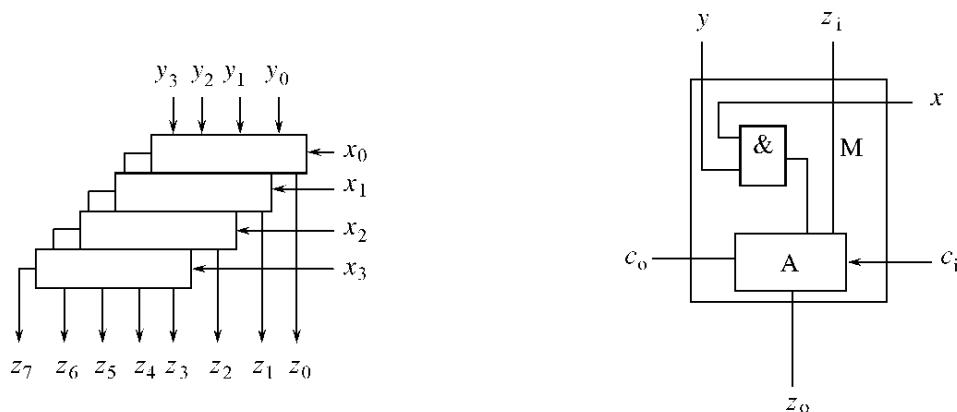


图 2.22 乘法器的示意图和元件

把乘法器看做 i 行 j 列元素的矩阵 M ，其关系由下面的等式表示：

$$M[i, j] \cdot x = x[i]$$

$$M[i, j] \cdot y = y[j]$$

$$M[i, j] \cdot z_i = M[i-1, j+1] \cdot z_o \quad M[0, j] \cdot z_i = 0$$

$$M[i, j] \cdot c_i = M[i, j-1] \cdot c_o \quad M[i, 0] \cdot c_i = 0$$

$$M[i, N-1].z_i = M[i-1, N-1].c_o$$

从图 2.23 明显看出，整个乘法器需要大量的电路。因此，乘法一般由按时间顺序执行的一系列加法来实现。但对于专门为数字计算设计的计算机是个例外，由于乘法是经常执行的一项运算，因此在计算机里使用大量的电路实现乘法是合理的。当使用快速进位传递的措施时，电路变得更加复杂。在图 2.23 所示的电路中，最长的进位通路有 n 个元件，也就是 $6n$ 个门。因此使用快速进位传递是合理的。

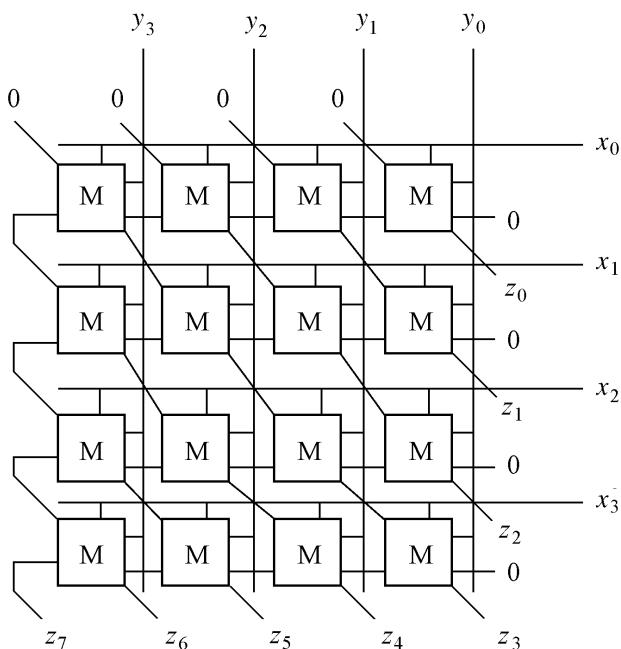


图 2.23 4×4 位乘法器

2.3 组合电路的竞争与冒险

前面分析和设计组合逻辑电路时，均未考虑信号在线路及器件中的传输与变换时所发生的延迟现象。事实上，正是由于这些延迟，当输入信号发生变化时，其输出信号不能同步地跟随输入信号的变化而变化，而是经过一段过渡时间后才能达到原先所希望的状态，从而产生瞬间的错误输出，造成逻辑功能的瞬间紊乱。这种现象就像瞬间冒出来的危险动作一样，因此被称为逻辑电路的“冒险现象”。

逻辑电路的冒险现象是一种逻辑故障，这种逻辑故障在电路和连线上是找不出原因的，只能从逻辑上采取措施加以解决。这是一个比较复杂的问题，实际工作中又不可回避，所以在此对冒险现象的形成原因和消除方法做一简单介绍。

2.3.1 竞争与冒险的产生

在组合电路中,当一个输入信号经过多路传输后又会重新回到某个门上,由于传输路径不同而导致时延不同,到达会合点的时间有先有后,这一现象称为竞争,这种变量称为有竞争力的变量。

不产生错误输出的竞争称为非临界竞争,产生错误输出的竞争称为临界竞争。临界竞争产生的错误输出即波形上的尖峰脉冲(毛刺),有可能引起后级电路(时序电路)的错误动作,称为冒险。

根据尖峰脉冲极性的不同,可以把冒险现象分为0型冒险和1型冒险两种类型。

输出尖峰为负向脉冲的冒险现象称为0型冒险,它主要出现在与或、与非、与或非型电路中;输出尖峰为正向脉冲的冒险现象称为1型冒险,它主要出现在或与、或非型电路中。

2.3.2 竞争与冒险的识别

判断一个组合电路是否存在冒险,可采用代数法或卡诺图法。

1. 代数法

当某些逻辑变量取特定值(1或0)时,组合逻辑电路输出函数表达式为下列形式之一,则存在冒险现象,此时A是有竞争力的变量:

$$F = A + \bar{A} \quad \text{存在0型冒险}$$

$$F = A\bar{A} \quad \text{存在1型冒险}$$

例如:试判断图2.24所示的电路是否存在冒险,如有,指出冒险类型,画出波形。

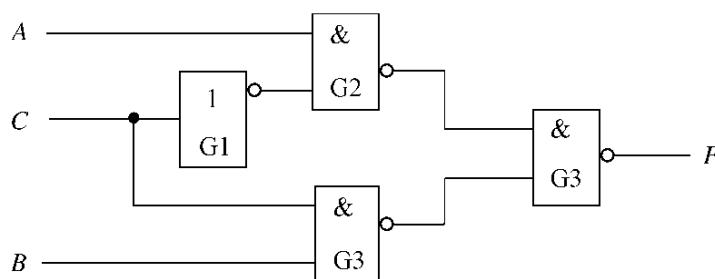


图 2.24 逻辑电路

解:在图中, $F = \overline{AC} \cdot \overline{BC} = A\bar{C} + BC$,若输入变量 $A = B = 1$,则有

$$F = 1 \cdot \bar{C} + 1 \cdot C = \bar{C} + C$$

因此,该电路存在0型冒险,C是有竞争力的变量。

下面来看此时 F 的波形。在稳态下 ,无论 C 取何值 , F 恒为 1 ,但当 C 变化时 ,由于信号的各输出路径的时延不同 ,将会出现如图 2.25 所示的情况。图中假设每个逻辑门的时延相同 ,均为 t_{pd} 。

由图 2.25 可见 ,当变量 C 由高电平变为低电平时 ,输出将会产生负毛刺 ,即存在 0 型冒险 ;但当 C 由低电平变为高电平时 ,却没有产生毛刺 ,只有竞争 ,没有冒险。这说明 ,有竞争力的变量发生变化时并不一定都产生冒险。

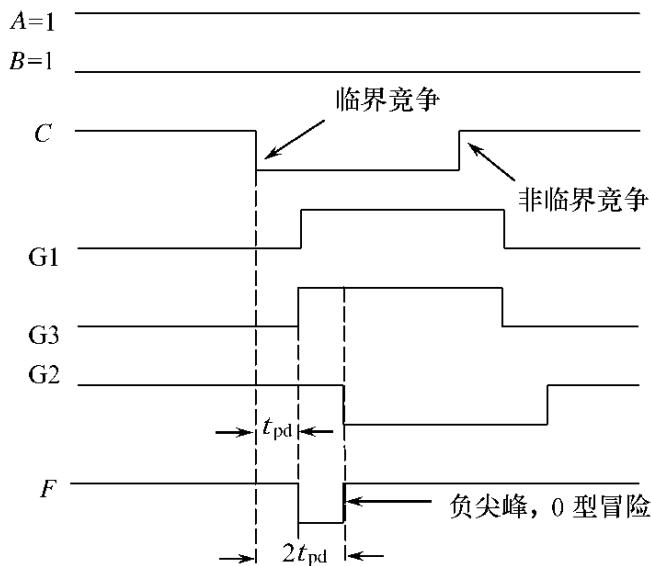


图 2.25 电路的 0 型冒险

2. 卡诺图法

在逻辑函数的卡诺图中 ,函数表达式的每个积项(或和项)对应于卡诺图上的一个卡诺圈。如果两个卡诺圈存在着相切部分 ,且相切部分又未被其他卡诺圈圈住 ,那么该电路必然存在冒险。

例如 :用卡诺图法判断函数 $F = A\bar{C} + B\bar{C} + \bar{A}CD$ 是否存在冒险。

解 : F 的卡诺图如图 2.26 所示。从图中可见 ,代表 $B\bar{C}$ 和 $\bar{A}CD$ 的两个卡诺圈相切 ,且相切部分又未被其他卡诺圈圈住 ,因此 ,当 C 从 0 到 1 或从 1 到 0 变化时 , F 将从一个卡诺圈进入另一个卡诺圈 ,从而产生冒险。至于是哪类冒险 ,可以

CD \ AB	00	01	11	10
00			1	
01		1	1	
11	1	1		
10	1	1		

图 2.26 用卡诺图判断逻辑函数是否存在冒险

用代数法确定。

2.3.3 竞争与冒险的消除

当组合逻辑电路存在冒险现象时,可以采取修改逻辑设计、增加选通电路、增加输出滤波等多种方法来消除冒险现象。后两种方法或增加电路复杂性,或使输出波形变坏,实际中极少使用。因此,这里只介绍通过修改逻辑设计来消除冒险现象的方法。

修改逻辑设计方法实际上是通过增加冗余项的办法,使函数在任何情况下都不可能出现 $F = A + \bar{A}$ 或 $F = A \cdot \bar{A}$ 的情况,从而达到消除冒险现象的目的。从卡诺图上看,相当于在相切的卡诺圈间增加一个冗余圈,将相切处的 0 或 1 圈起来。

例如:采用修改逻辑设计的方法,修改上例中函数 $F = A\bar{C} + B\bar{C} + \bar{A}CD$ 存在的冒险现象。

解:在原卡诺图中相切的两个卡诺圈相切处,增加一个冗余的卡诺圈(虚线),将相切处的两个 1 圈起来,如图 2.27 所示。此时, $F = A\bar{C} + B\bar{C} + \bar{A}CD + \bar{A}BD$ 。当 $A = 0, B = 1, D = 1$ 时, $F = 0 + \bar{C} + C + 1 = 1$,从而消除了 0 型冒险。

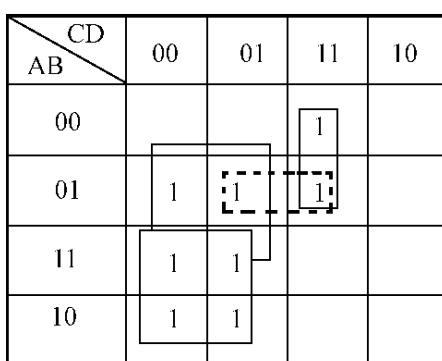


图 2.27 修改逻辑设计后的卡诺图

习 题

1. 用布尔代数的基本公式和规则证明下列等式。

$$A\bar{B} + BD + \bar{A}D + DC = A\bar{B} + D ; ABC + \bar{A}\bar{B}\bar{C} = \overline{A\bar{B} + B\bar{C} + \bar{C}A}$$

2. 求出下列函数的反函数。

$$F = AB + \bar{A}\bar{B} ; F = A\bar{B} + B\bar{C} + C(\bar{A} + D)$$

3. 写出下列函数的对偶式。

$$F = (A + B)(\bar{A} + C)(C + DE) + E; F = \overline{\overline{AB}} \cdot \overline{\overline{CB}} \cdot \overline{\overline{D}\bar{A}\bar{B}}$$

4. 函数的真值表如表 2.20 所示,列出该函数最小项及最大项的表达式。

表 2.20

A	B	C	D	F	A	B	C	D	F
0	0	0	0	1	0	0	0	1	0
1	0	0	0	0	1	0	0	1	1
0	1	0	0	0	0	1	0	1	1
1	1	0	0	1	1	1	0	1	0
0	0	1	0	1	0	0	1	1	1
1	0	1	0	0	1	0	1	1	0
0	1	1	0	0	0	1	1	1	1
1	1	1	0	1	1	1	1	1	1

5. 用公式将下列函数化简为最简“与或”式。

$$F = \overline{AB} + (AB + A\bar{B} + \bar{A}B)C; F = (X + Y)Z + \overline{XY}W + ZW$$

6. 将下列函数展开为最小项之和。

$$F = AB + A\bar{B} + \bar{A}B + \overline{CD}; F = A(\bar{B} + \bar{C}\bar{D}) + \overline{ABC}D$$

7. 用卡诺图将下列函数化为最简“与或”式。

$$F = \sum m^3(0, 1, 2, 4, 5, 7); F = \sum m^4(0, 1, 2, 3, 4, 6, 7, 8, 9, 11, 15);$$

8. 设计一个监视交通信号灯工作状态的逻辑电路。每一组信号灯由红、黄、绿三盏灯组成,如图 2.28 所示。正常工作情况下,任何时刻必有一盏灯点亮。而当出现其他五种点亮状态时,电路发生故障,这时要求发出故障信号,以提醒维修人员前去修理。

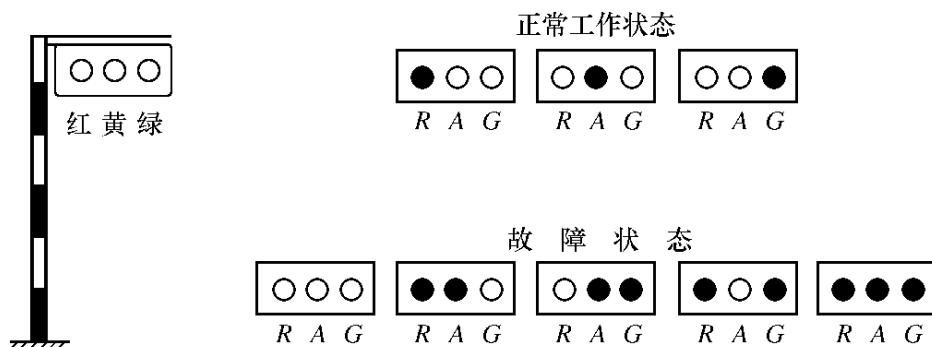


图 2.28 交通信号灯的正常工作状态与故障状态

第三章 时序逻辑电路

时序电路是计算机及其他电子系统中常用的一种电路。它和组合电路是类型完全不同的两种电路。组合电路的输出仅取决于电路当时的输入 ,而与电路过去的输入无关 ;时序电路的输出不仅取决于电路当时的输入 ,还与电路过去的输入有关。由于时序电路有这一特点 ,因此在电路的内部必然有记忆元件 ,用来记忆与过去输入信号有关的信息或电路的过去输出状态。

时序电路分为两大类 :同步时序电路和异步时序电路。在同步时序电路中有一个公共的时钟信号 ,电路中各记忆元件受它统一控制。只有在该时钟信号到来时 ,记忆元件的状态才能发生变化 ,从而使时序电路的输出发生变化 ,而且每来一个时钟信号 ,记忆元件的状态和电路输出状态才可能改变一次。如果时钟信号没有来到 ,输入信号的改变不能引起电路输出状态的变化。在异步时序电路中 ,电路没有统一的时钟信号 ,各记忆元件也不受同一时钟控制 ,电路状态的改变由输入信号引起 ,且输入信号的一次变化可引起状态变化多次。

本章将介绍同步时序电路的分析及设计方法。

3.1 锁存器与触发器

在各种复杂的数字电路中不但需要对二值信号进行算术运算和逻辑运算 ,还经常需要将这些信号和运算结果保存起来。为此 ,需要使用具有记忆功能的基本逻辑单元。能够存储 1 位二值信号的基本单元电路统称为触发器。

为了实现记忆 1 位二值信号的功能 ,触发器必须具备以下两个基本特点 :第一 ,具有两个能自行保持的稳定状态 ,用来表示逻辑状态的 0 和 1 ,或二进制数的 0 和 1 。第二 ,根据不同的输入信号可以置成 1 或 0 状态。

由于控制方式的不同 ,即信号的输入方式以及触发器状态随输入信号变化的规律不同 ,触发器的逻辑功能在细节上有所不同。因此根据触发器逻辑功能的不同分为 $R - S$ 触发器、 $J - K$ 触发器、 T 触发器、 D 触发器等几种类型。

3.1.1 基本 $R - S$ 触发器的电路结构与动作特点

基本 $R - S$ 触发器,又称 $R - S$ 锁存器,是各种触发器电路中结构形式最简单的一种。同时,它又是许多复杂电路结构触发器的一个组成部分。

图 3.1(a)为由两个或非门所组成的基本 $R - S$ 触发器电路。 Q 和 \bar{Q} 称为输出端,并且定义 $Q = 1, \bar{Q} = 0$ 为触发器的 1 状态, $Q = 0, \bar{Q} = 1$ 为触发器的 0 状态。 S_D 称为置位端或置 1 输入端, R_D 称为复位端或置 0 输入端。

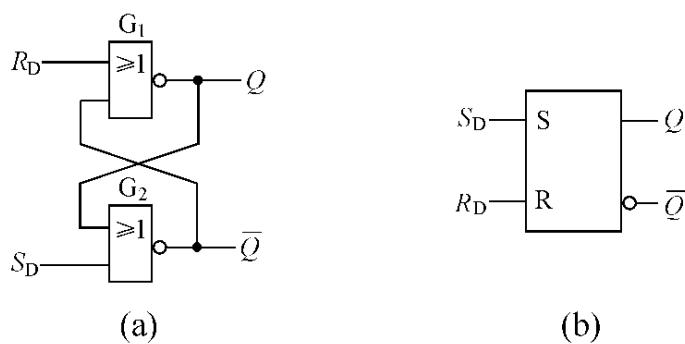


图 3.1 用或非门组成的基本 $R - S$ 触发器

(a) 电路结构 (b) 图形符号

当 $S_D = 1, R_D = 0$ 时, $Q = 1, \bar{Q} = 0$ 。在 $S_D = 1$ 信号消失以后(即 S_D 回到 0),由于有 Q 端的高电平接回到 G_2 的另一个输入端,因而电路的 1 状态得以保持。

当 $S_D = 0, R_D = 1$ 时, $Q = 0, \bar{Q} = 1$ 。在 $R_D = 1$ 信号消失以后,电路保持 0 状态不变。

当 $S_D = R_D = 0$ 时,电路维持原来的状态不变。

当 $S_D = R_D = 1$ 时, $Q = \bar{Q} = 0$,这既不是定义的 1 状态,也不是定义的 0 状态。而且,在 S_D 和 R_D 同时回到 0 以后无法断定触发器将回到 1 状态还是 0 状态。因此,在正常工作时输入信号应遵守 $S_D R_D = 0$ 的约束条件,亦即不允许输入 $S_D = R_D = 1$ 的信号。

将上述逻辑关系列成真值表,就得到表 3.1,表中 Q_n 表示在某一时钟脉冲到来前,时序电路的状态,称为现态; Q_{n+1} 表示在该时钟脉冲到来后,时序电路的状态,称为状态。

基本 $R - S$ 触发器也可以用与非门构成,如图 3.2 所示。这个电路是以低电平作为输入信号的,所以用 \bar{S}_D 和 \bar{R}_D 分别表示置 1 输入端和置 0 输入端。在图 3.2(b)的图形符号上,用输入端的小圆圈表示用低电平作输入信号,或者叫做低电平有效。表 3.2 是它的特性表。

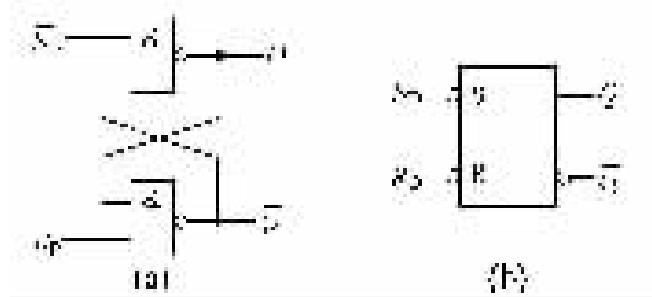


图 3.2 用与非门组成的基本 $R - S$ 触发器
(a) 电路结构 (b) 图形符号

表 3.1 用或非门组成的基本 $R - S$ 触发器的特性表

S_D	R_D	Q^n	Q^{n+1}
0	0	0	0
0	0	1	1
1	0	0	1
1	0	1	1
0	1	0	0
0	1	1	0
1	1	0	0*
1	1	1	0*

* S_D, R_D 的 1 状态同时消失后状态不定

表 3.2 用与非门组成的基本 $R - S$ 触发器的特性表

\overline{S}_D	\overline{R}_D	Q^n	Q^{n+1}
1	1	0	0
1	1	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
0	0	0	1*
0	0	1	1*

* $\overline{S}_D, \overline{R}_D$ 的 0 状态同时消失以后状态不定

3.1.2 同步 $R - S$ 触发器

在数字系统中,为协调各部分的动作,常常要求某些触发器于同一时刻动作。为此,必须引入同步信号,使这些触发器只有在同步信号到达时才按输入信号改变状态。通常把这个同步信号叫做时钟脉冲,或称为时钟信号,简称时钟,用 CP (Clock Pulse 的缩写) 表示。这种受时钟信号控制的触发器统称为时钟触发器。

实现时钟控制的最简单方式是采用图 3.3 所示的同步 $R - S$ 触发器结构。该电路由两部分组成:由与非门 G_1, G_2 组成的基本 $R - S$ 触发器和由与非门 G_3, G_4 组成的输入控制电路。

当 $CP = 0$ 时,门 G_3, G_4 截止,输入信号 S, R 不会影响输出端的状态,故触发器保持原状态不变。

当 $CP = 1$ 时, S, R 信号通过门 G_3, G_4 反相后加到由 G_1 和 G_2 组成的基本 $R - S$ 触发器上,使 Q 和 \overline{Q} 的状态跟随输入状态的变化而改变。它的特性表如表 3.3 所示。

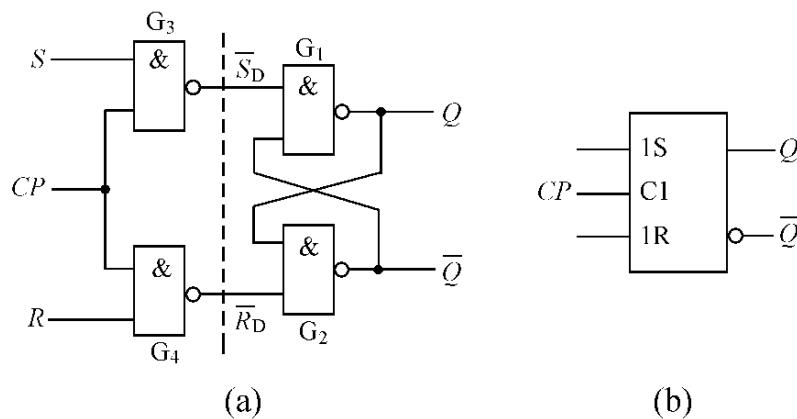


图 3.3 同步 R-S 触发器
 (a) 电路结构 (b) 图形符号

表 3.3 同步 $R - S$ 触发器的特性表

CP	S	R	Q^n	Q^{n+1}
0	\times	\times	0	0
0	\times	\times	1	1
1	0	0	0	0
1	0	0	1	1
1	1	0	0	1
1	1	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	1	0	1^*
1	1	1	1	1^*

* CP 回到低电平后状态不定。

从表3.3中可见,只有 $CP = 1$ 时触发器输出端的状态才受输入信号的控制,而且在 $CP = 1$ 时这个特性表和基本 $R - S$ 触发器的特性表相同。输入信号同样需要遵守 $SR = 0$ 的约束条件。

凡在时钟信号作用下逻辑功能符合表 3.4 特性表所规定的逻辑功能者 ,叫做 $R - S$ 触发器。如果把表 3.4 特性表所规定的逻辑关系写成逻辑函数式 则得到

$$\begin{cases} Q^{n+1} = \bar{S}\bar{R}Q^n + S\bar{R}\bar{Q}^n + S\bar{R}Q^n = S\bar{R} + \bar{S}\bar{R}Q^n \\ SR = 0 \quad (\text{约束条件}) \end{cases}$$

利用约束条件将上式化简,于是得出下式称为 $R - S$ 触发器的特性方程。

$$\begin{cases} Q^{n+1} = S + \bar{R}Q^n \\ SR = 0 \quad (\text{约束条件}) \end{cases}$$

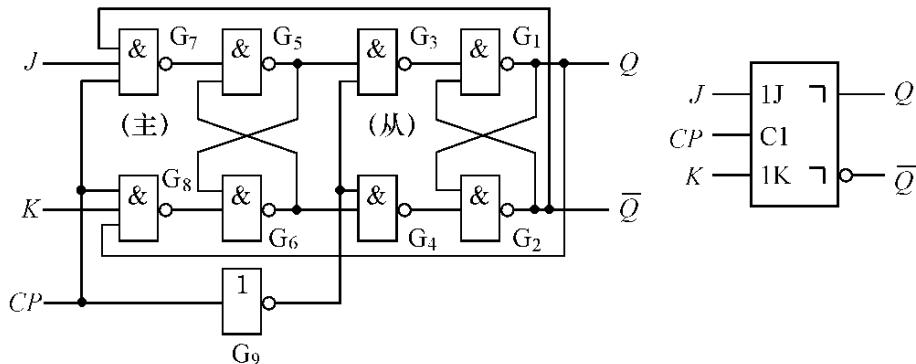
表 3.4 $R - S$ 触发器特性表

S	R	Q^n	Q^{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	不定
1	1	1	不定

3.1.3 $J - K$ 触发器

图 3.4 电路称为主从结构 $J - K$ 触发器(简称主从 $J - K$ 触发器)。

若 $J = 1, K = 0$ 则 $CP = 1$ 时主触发器置 1(原来是 0 则置成 1, 原来是 1 则保持 1), 待 $CP = 0$ 以后从触发器亦随之置 1, 即 $Q^{n+1} = 1$ 。

图 3.4 主从 $J - K$ 触发器

若 $J = 0, K = 1$ 则 $CP = 1$ 时主触发器置 0, 待 $CP = 0$ 以后从触发器也随之置 0, 即 $Q^{n+1} = 0$ 。

若 $J = K = 0$ 则由于门 G_7, G_8 被封锁, 触发器保持原状态不变, 即 $Q^{n+1} = Q^n$ 。

若 $J = K = 1$ 时, 需要分别考虑两种情况。第一种情况是 $Q^n = 0$, 这时门 G_8 被 Q 端的低电平封锁, $CP = 1$ 时仅 G_7 输出低电平信号, 故主触发器置 1。 $CP = 0$ 以后从触发器也跟着置 1, 即 $Q^{n+1} = 1$ 。

第二种情况是 $Q^n = 1$, 这时门 G_7 被 \bar{Q} 端的低电平封锁, 因而在 $CP = 1$ 时仅 G_8 能给出低电平信号, 故主触发器被置 0。当 $CP = 0$ 以后从触发器跟着置 0, 故 $Q^{n+1} = 0$ 。

综合以上两种情况可知,无论 $Q^n = 1$ 还是 $Q^n = 0$,触发器的次态可统一表示为 $Q^{n+1} = \overline{Q^n}$ 。就是说,当 $J = K = 1$ 时,CP 下降到到达后触发器将翻转为与初态相反的状态。

将上述的逻辑关系用真值表表示,即得到表 3.5 所示的主从 $J - K$ 触发器的特性表。

凡在时钟信号作用下逻辑功能符合表 3.6 特性表所规定的逻辑功能者,叫做 $J - K$ 触发器。根据表 3.6 可以写出 $J - K$ 触发器的特性方程,化简后得到

$$Q^{n+1} = J\overline{Q^n} + \overline{K}Q^n$$

表 3.5 主从 $J - K$ 触发器的特性表

CP	J	K	Q^n	Q^{n+1}
\times	\times	\times	\times	Q^n
	0	0	0	0
	0	0	1	1
	1	0	0	1
	1	0	1	1
	0	1	0	0
	0	1	1	0
	1	1	0	1
	1	1	1	0

表 3.6 $J - K$ 触发器特性表

J	K	Q^n	Q^{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

3.1.4 D 触发器

凡在时钟信号作用下逻辑功能符合表 3.7 特性表所规定的逻辑功能者,叫做 D 触发器。从特性表写出 D 触发器的特性方程为:

$$Q^{n+1} = D$$

表 3.7 D 触发器特性表

D	Q^n	Q^{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

3.1.5 T 触发器

在某些应用场合下,需要这样一种逻辑功能的触发器,当控制信号 $T = 1$ 时,

每来一个 CP 信号它的状态就翻转一次 而当 $T = 0$ 时 , CP 信号到达后它的状态保持不变。具备这种逻辑功能的触发器叫做 T 触发器。它的特性表如表 3.8 所示。

表 3.8 T 触发器的特性表

T	Q^n	Q^{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

从特性表写出 T 触发器的特性方程为

$$Q^{n+1} = T \overline{Q^n} + \bar{T} Q^n$$

事实上 ,只要将 $J - K$ 触发器的两个输入端连在一起作为 T 端 ,就可以构成 T 触发器。正因为如此 ,在触发器的定型产品中通常没有专门的 T 触发器。

当 T 触发器的控制端接至固定的高电平时(即 T 恒等于 1)则上式变为

$$Q^{n+1} = \overline{Q^n}$$

即每次 CP 信号作用后触发器必然翻转成与初态相反的状态。有时也把这种接法的触发器叫做 T' 触发器。其实 T' 触发器只不过是处于一种特定工作状态下的 T 触发器而已。

3.2 时序机(状态机)

3.2.1 同步时序电路的结构

通过图 3.5(a)所示实例来讨论同步时序电路的基本结构。图 3.5(a)所示是一个用正沿触发的 D 触发器构成的可控二进制计数器。当输入信号 $X = 1$ 时 ,在时钟脉冲 CP 的正沿作用下 ,计数器计数 ,计数顺序是 $00 \rightarrow 10 \rightarrow 01 \rightarrow 11 \rightarrow 00 \rightarrow \dots$ (数字的左右位分别为 D 触发器 1、2 的状态);当 $X = 0$,触发器输出 Q_1 、 Q_2 保持原有的状态不变。电路输出 Z 是计数器进位标志 ;当 $X = 1$,且 Q_1 、 $Q_2 = 11$ 时 , $Z = 1$;否则 $Z = 0$ 。触发器的数据输入 D_1 、 D_2 以及输出 Z 的逻辑表达式为 :

$$D_1 = \bar{X}Q_1 + X\bar{Q}_1$$

$$D_2 = X(\bar{Q}_1Q_2 + Q_1\bar{Q}_2) + \bar{X}Q_2$$

$$Z = XQ_1Q_2$$

图 3.5(b)是电路的波形图(又称时序图),它以波形图的形式描述在输入序列作用下,在 CP 到来时电路输出情况及内部状态转换关系。图中 Q_1 、 Q_2 及 Z 的变化发生在 CP 前沿到来的时刻。

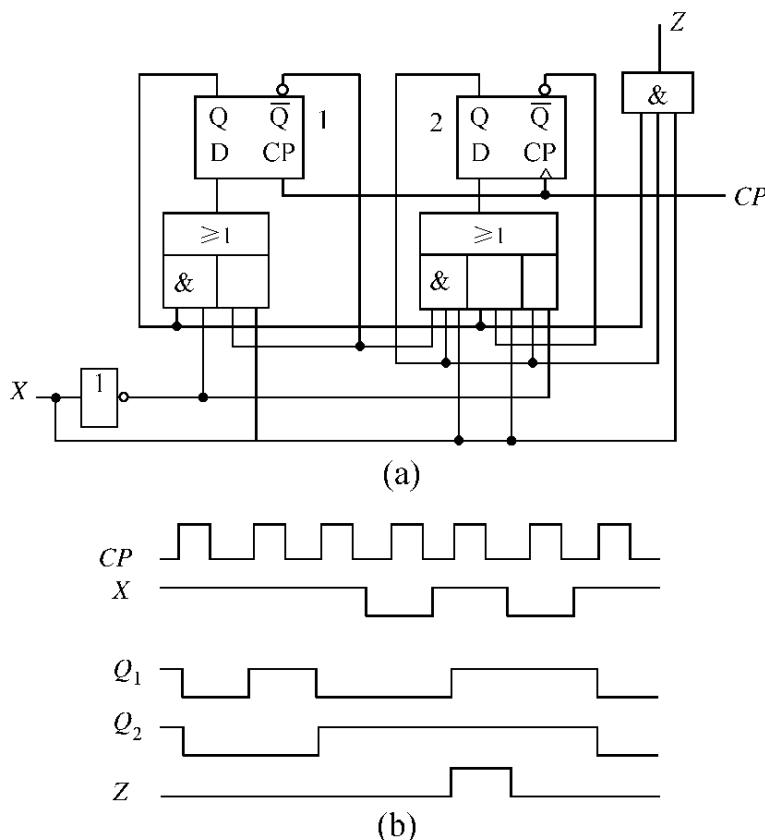


图 3.5 可控二进制计数器

(a) 逻辑图 (b) 时序图

显然,时序电路内触发器的状态(简称时序电路的状态,又称状态) Q_1 、 Q_2 及输出 Z 不仅取决于输入 X 的状态,而且与电路过去的输入有关。例如,在 CP 作用下 Z 由“0”变为“1”,此时除 $X = 1$ 以外,电路的原态一定是 $Q_1 Q_2 = 01$ 。

把图 3.5(a)所示逻辑图画成如图 3.6(a)所示形式,它由两部分组成,一部分是组合电路,另一部分是记忆电路。

图中组合电路的外部输入信号为 X ,而 Q_1 、 Q_2 、 \bar{Q}_1 、 \bar{Q}_2 为组合电路的内部输入。组合电路提供两种输出:一是时序电路外部输出 Z ,二是触发器的 D_1 、 D_2 输入。称 D_1 、 D_2 为组合电路的内部输出。记忆电路的输入、输出分别是组合电路的内部输出和内部输入。图 3.6(b)给出了时序电路的一般结构图。组合电路和记忆电路之间存在反馈关系是时序电路的一个特点。组合电路的输出和输入的关系可用熟知的逻辑表达式来描述。由于记忆电路的输出不仅取决于当时的输入而且还取决于过去的输入,而两者又不发生在同一时刻,因此对它的描述就要用到激励表、状态表及状态图等工具。

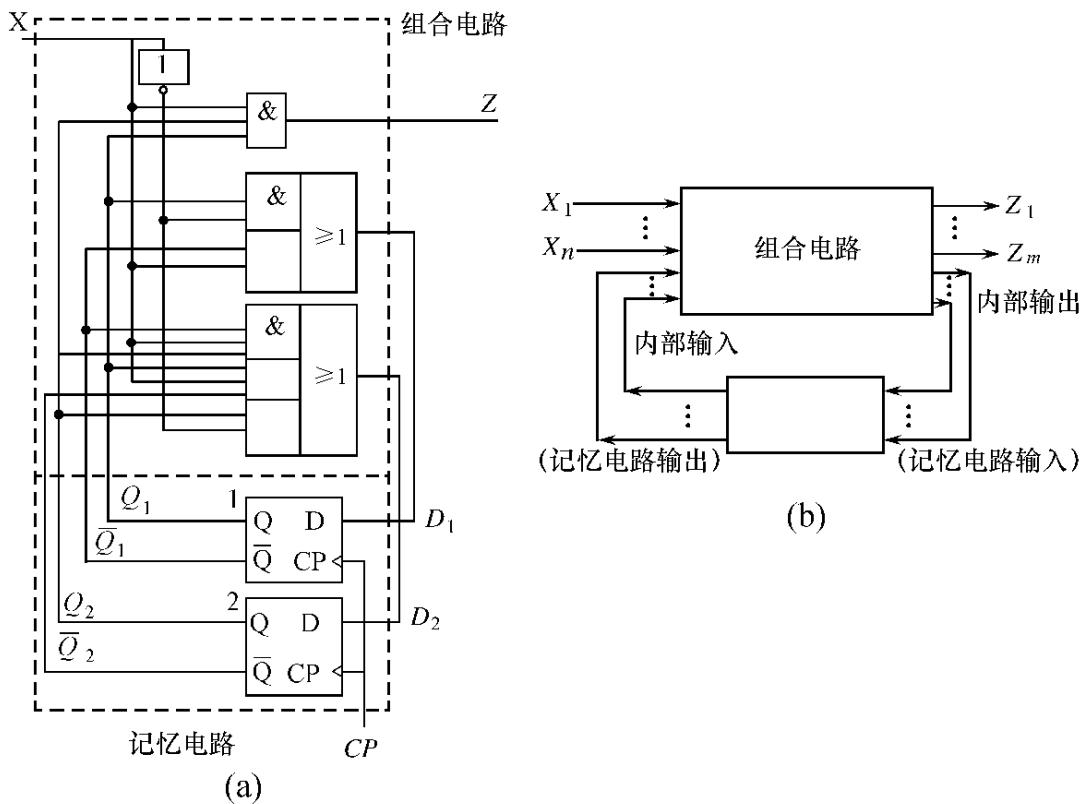


图 3.6 图 3.5 的另一种形式

(a)图 3.5 的另一种画法 (b)时序电路的一般结构

3.2.2 激励表、状态表及状态图

在介绍本节的主要内容之前,先定义时序电路的现态和次态。

在某一时钟脉冲到来前,时序电路的状态称为电路的现态,用 Q^n 表示。在该时钟脉冲来到后,时序电路的状态称为电路的次态,用 Q^{n+1} 表示。有时为了简单起见,常将现态 Q^n 表示成 Q 。

触发器是最简单的时序电路。它的激励表描述触发器从现态转换到次态时,对数据输入的要求。激励表把触发器的现态和次态作自变量,把触发器的数据输入作因变量。触发器的激励表可从触发器的功能表推出。

D 型功能触发器的激励表如图 3.7(a)所示。它说明,不论触发器处于什么现态,要转换到次态“1”,在时钟脉冲来到前,数据输入(即激励) D 应为“1”;不论触发器处于什么现态,要转换到次态“0”,在时钟脉冲来到前, D 应为“0”。

$J-K$ 功能触发器和 T 功能触发器的激励表分别示于图 3.7(b)和图 3.7(c)。对 T 触发器,若要求 $Q^n = Q^{n+1}$, T 应为“0”,否则 T 应为“1”。

现态	次态	输入	现态	次态	输入	现态	次态	输入
Q^n	Q^{n+1}	D	Q^n	Q^{n+1}	$J \quad K$	Q^n	Q^{n+1}	T
0	0	0	0	0	0 ×	0	0	0
0	1	1	0	1	1 ×	0	1	1
1	0	0	1	0	× 1	1	0	1
1	1	1	1	1	× 0	1	1	0

(a)

(b)

(c)

图 3.7 三种触发器的激励表

(a) D 功能触发器 (b) $J - K$ 功能触发器 (c) T 功能触发器

描述时序电路输入与状态转换关系的表格称为状态转换表,简称状态表。触发器的状态表以输入和现态为自变量,以次态为因变量。图 3.8 给出了 D 功能触发器、 $J - K$ 功能触发器和 T 触发器的状态表。

Q^n	D	0	1
0	0	1	
1	0	1	

(a)

Q^n	JK	00	01	11	10
0	0	0	1	1	
1	1	0	0	1	

(b)

Q^n	T	0	1
0	0	1	
1	1	0	

(c)

图 3.8 三种触发器的状态表

(a) D 功能触发器 (b) JK 功能触发器 (c) T 功能触发器

时序电路的输入与状态转换关系还可以图的形式给出。在状态图中,用圆圈表示状态,圈内以文字或数字注明状态的标志。圆圈之间用箭头线相连,它表示状态的转换。箭尾端圆圈内标明的应是现态,箭上注明发生转移的输入条件,箭头端圆圈内标明的则应是次态。图 3.9(a)(b)(c) 分别给出三种功能的触发器的状态图。图中触发器的“0”、“1”两个状态用圈内标注“0”、“1”的小圆来表示。例如,使 D 触发器由“0”转换到“1”的条件是 $D = 1$,在图中用由“0”圈指向“1”圈的箭表示状态转换,箭上标注“1”表示转换条件。如果 D 触发器的现态为“0”,输入 D 为“0”,则状态不发生变换,在图中用起、终点都是“0”圈的箭来表示,并在箭上标注“0”。使 $J - K$ 触发器由现态“0”转换成次态为“1”的条件有两个, $JK = 10$ 或 $JK = 11$,因此在由“0”圈指向“1”圈的箭上标注“10”、“11”。

下面作图 3.5(a) 所示电路的状态表和状态图。

状态表的自变量是时序电路的输入 X 和 4 个现态($Q_2 Q_1 = 00, 10, 01, 11$),因变量是次态和输出 Z 。把 X 排在状态表的上侧,把现态排在左侧。表中填入相应的次态和输出 Z (斜线的右侧为 Z)。图 3.10(a) 给出了它的状态表。

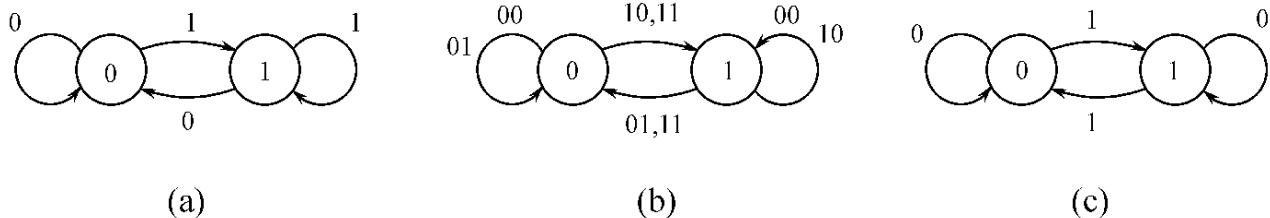


图 3.9 三种触发器的状态图

(a) D 功能触发器 (b) $J - K$ 功能触发器 (c) T 功能触发器

例如,若 $Q_2 Q_1 = 10$,如此时 $X = 1$,则来 CP 正跳变, $Q_2 Q_1$ 将由 10 变为 11, $Q_2 Q_1$ 变为 11 后, Z 即为 1,故在状态表的 $Q_2^n Q_1^n X = 101$ 小格中填 11/1。

由上例可看到,状态表虽简单,但却完全反映了在时序电路输入端上加信号后,电路的输出是如何变换的,因此它是分析时序电路外部特性的一个方便的工具。正因为它以简洁的形式反映电路的外部特性,所以,它还是设计时序电路的一个有力工具。

图 3.10(b)是 3.10(a)所示时序电路的状态图。4 个标有 00、01、10、11 的圆圈表示电路的状态。箭上标以 X 和 Z (斜线的右侧为 Z)。

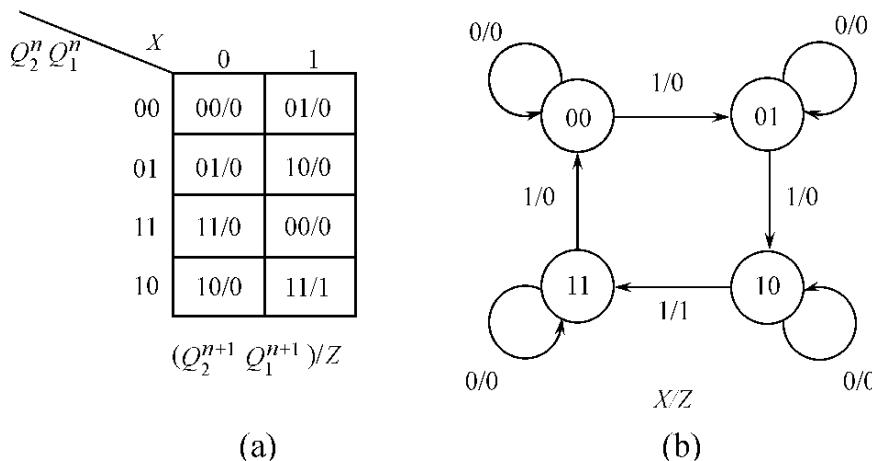


图 3.10 图 3.5(a)的状态表和状态图

(a) 状态表 (b) 状态图

状态图只是状态表的另一种表示方式。它也是分析和设计时序电路常用的一种工具。

3.3 同步时序逻辑电路

3.3.1 同步时序逻辑电路分析

同步时序电路的分析是根据已有的电路图,通过列出状态表或画出状态图来分析电路的工作过程以及其输入与输出之间的关系。

同步时序电路的分析步骤为:

(1) 根据给定的同步时序电路,列出电路中组合电路的输出函数,列出电路中各触发器的激励函数(描述触发器数据输入的逻辑函数,又称控制函数)。

(2) 列出组合电路的状态真值表。真值表的输入是时序电路的输入和时序电路的现态,输出是时序电路的输出及各触发器的数据输入。

(3) 列出时序电路的次态。

(4) 作状态表和状态图。

(5) 分析时序电路的外部性能。

下面通过对一些典型电路的分析,来进一步说明分析过程。

例 1 分析图 3.11 所示同步时序电路。

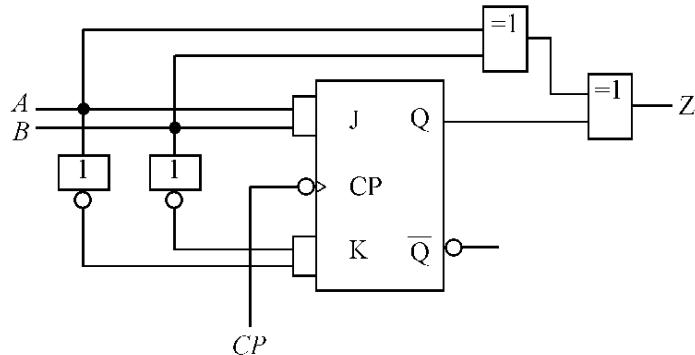


图 3.11 同步时序电路逻辑图

(1) 列出电路的输出函数和触发器的激励函数。

$$Z = A \oplus B \oplus Q$$

$$J = AB$$

$$K = \bar{A}\bar{B}$$

(2) 列出组合电路的状态真值表,如表 3.9 所示。

(3) 列出时序电路的次态,如表 3.10 所示。

表 3.9 例 1 的状态真值表

现态	输入	触发器输入	输出
Q^n	A B	J K	Z
0	0 0	0 1	0
0	0 1	0 0	1
0	1 0	0 0	1
0	1 1	1 0	0
1	0 0	0 1	1
1	0 1	0 0	0
1	1 0	0 0	0
1	1 1	1 0	1

表 3.10 例 1 的次态表

现态	输入	次态
Q^n	A B	Q^{n+1}
0	0 0	0
0	0 1	0
0	1 0	0
0	1 1	1
1	0 0	0
1	0 1	1
1	1 0	1
1	1 1	1

它以时序电路的输入 A 、 B 和触发器的现态所有可能的组合为输入，对照状态真值表，查得对应的 J 、 K 值，再由 $J-K$ 触发器的功能表，即可得触发器的次态。

(4) 由表 3.10 作状态表和状态图。

(5) 分析时序电路的外部性能。

由状态表或状态图可知，当 A 、 B 和 Q^n 中有奇数个“1”时，输出 $Z = 1$ ，否则 $Z = 0$ ；当 A 、 B 和 Q^n 中有两个及两个以上的“1”时，则 $Q^{n+1} = 1$ ，否则 $Q^{n+1} = 0$ 。所以此电路是一个串行二进制加法器，其中 A 、 B 为被加数和加数， Z 为和数， $J-K$ 触发器存放进位值。

3.3.2 同步时序电路的设计

时序电路的设计可归纳为如下步骤：

(1) 作原始状态表。根据给定的电路设计条件构成原始状态表。

(2) 状态表的简化。原始状态表通常不是最小化状态表，它往往包括多余的状态，因此必须首先对它进行简化。

(3) 状态分配。即对简化后的状态给以编码。这就要根据状态数确定触发器的数量并对每个状态指定一个二进制数构成的编码。

(4) 作激励函数和输出函数。根据选用的触发器激励表和电路的状态表，综合出电路中各触发器的激励函数和电路的输出函数。

(5) 画逻辑图。

1. 原始状态表的构成

同步时序电路设计的第一步，就是把用文字或用波形图表达的设计要求转变为状态表。这是设计中十分重要的一步。在这一步骤中，要指定最少的且是必

要的状态是困难的。在所确定的状态中往往会有多余的状态,这是允许的,可以在下一步“状态简化”中去掉多余的状态。但是在编制原始状态表时应确保不能遗漏状态。

下面通过一些实例来说明如何形成原始状态表。

例 2 “101”序列检测器(一)。

该电路有一个输入端 X 和一个输出端 Z 。在输入端 X 加上 0、1 信号序列,当信号序列中出现“101”信号时, Z 为“1”,否则 Z 为“0”。例如,在 X 上加上如下信号序列,则检测器的输出序列应为:

$$X : 010101101$$

$$Z : 000100001$$

确定状态表的过程如下:

首先假设检测器有一个初始状态 A 。若输入的第一个信号是“1”,它是“101”序列的第一个元素,应该把这个情况记忆下来,检测器进入状态 B ,检测器输出为“0”;若输入的第一个信号是“0”,它不是“101”序列的第一个元素,不必把这个情况记下来,检测器仍停留在状态 A ,检测器输出为“0”。

若电路处于 B 态(即检测器已经接收了“101”序列的第一个元素),输入信号是“0”,它是被测序列的第二个元素,检测器应把这个情况记录下来,并进入状态 C ,同时检测器的输出仍应为“0”;若输入信号是“1”,它不是被测序列的第二个元素,而仍相当于是第一个元素,所以电路仍停留在状态 B ,电路输出为“0”。

若电路处于 C 态,输入信号是“1”,它是被测序列的第三个元素,这时检测器已检出序列“101”,电路输出为“1”,检测器把第三个元素记录下来后,进入状态 D ;若输入信号是“0”,它不是被测序列的第三个元素,此时电路不但输出为“0”,而且还将“报废”以前已接收的第一、二个元素“10”,使电路回到状态 A 。

若电路处于 D 态,输入信号为“1”,它是第二个被测序列的第一个元素,检测器应把它记下来,电路进入 B 态;若输入为“0”,它不是第二个被测序列的第一个元素,电路应进入 A 态,等待接收第二个被测序列的第一个元素,输出为“0”。

根据以上描述,可知该检测器有 4 个原始状态,它的原始状态图和原始状态表如图 3.12 所示。

2. 状态表的简化

(1) 状态合并的条件

原始状态表往往包含多余的状态,为此需要进行状态简化,以求得最小化状态表。

比较例 2 的原始状态表第一行和第四行可以看到,状态 A 、 D 是等效的。这是因为不仅状态 A 、 D 在输入 $X = 0$ 、 $X = 1$ 作用下次态相同,而且其输出 Z (即电

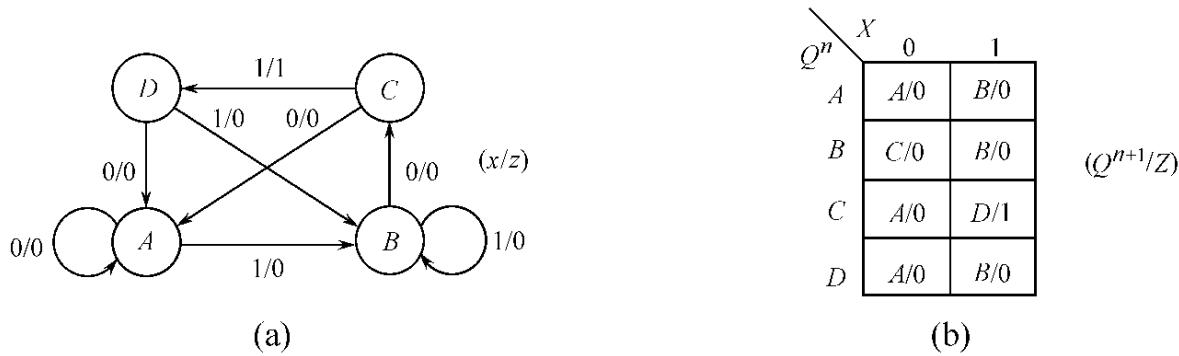


图 3.12 例 2 的原始状态图和原始状态表

(a) 原始状态图 (b) 原始状态表

路的外部表现)也相同。因此就可以将状态 A 、 D 合并,令合并后的状态为 A' ,这样,例 2 的状态图和状态表就成为图 3.13 所示形式。

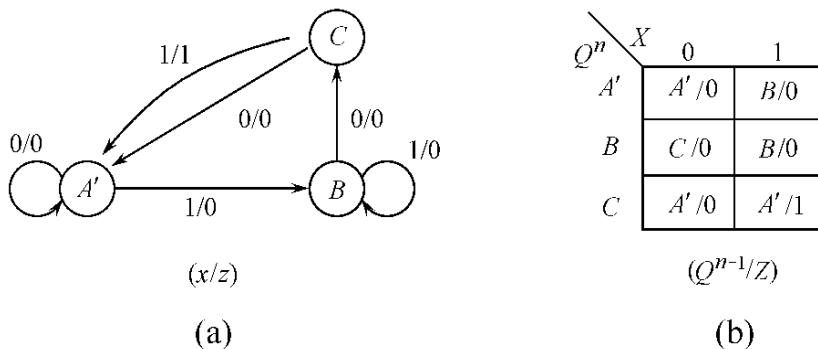


图 3.13 例 5 的状态图和状态表

(a) 状态图 (b) 状态表

(2) 状态化简的方法

对复杂一些的原始状态表仅用观察对比来判断状态等效关系是不行的,需要有一种系统的方法。下面介绍用隐含表来化简状态的方法,分三个步骤进行:顺序比较、关连比较和状态合并。

① 顺序比较

状态化简中采用的工具是隐含表。隐含表是一种正直角三角形网格,两直角边格数相同。图 3.14 是适合于 5 个状态(A 、 B 、 C 、 D 、 E)的隐含表,每直角边的格数为 4,水平边的网格自左至右按状态 A 、 B 、 C 、 D 顺序标注(不标注 E),垂直边的网格自上至下按 B 、 C 、 D 、 E 顺序标注(不标注 A)。对隐含表中的所有状态进行比较。先由水平向的 A 依次同垂直向的 B 、 C 、 D 、 E 进行比较,然后由水平向的 B 依次和垂直向的 C 、 D 、 E 进行比较,再后由水平向的 C 依次和垂直向的 D 、 E 比较,最后由 D 和 E 进行比较。比较结果写在相应的格子内。

下面对图 3.15(a)所示原始状态表进行顺序比较,它有如下三种结果:

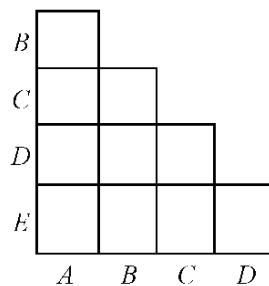


图 3.14 适合 5 个状态的隐含表

- a) 输出不相同 ,在隐含表相应的格内打“ \times ”,如 AD 、 AE 、 BD 、 BE 、 CD 、 CE 。
- b) 输出完全相同 ,次态相同或呈交错 ,在隐含表相应的格内打“ \checkmark ”,表示状态等效 ,如 BC 、 DE 。
- c) 输出完全相同 ,次态不相同但又非交错 ,此时将“ 次态对 ”填入相应的网格中以待在下一步骤中进一步比较 ,如在 AB 网格内填入 BE ,在 AC 网格中填入 BC 、 BE 。

顺序比较后的隐含表如图 3.15(b) 所示。

$Q^n \setminus X$	0	1
$A \setminus Q^{n+1}/Z$	$C/1$	$B/0$
B	$C/1$	$E/0$
C	$B/1$	$E/0$
D	$D/1$	$B/1$
E	$D/1$	$B/1$

(a)

$B \setminus Q^{n+1}/Z$	BE
$C \setminus B$	BC
$C \setminus E$	BE
$D \setminus A$	\times
$D \setminus B$	\times
$D \setminus C$	\times
$E \setminus D$	\times
$E \setminus A$	\times
$E \setminus B$	\times
$E \setminus C$	\times
$E \setminus D$	\checkmark

(b)

$B \setminus Q^{n+1}/Z$	BE
$C \setminus B$	BC
$C \setminus E$	BE
$D \setminus A$	\times
$D \setminus B$	\times
$D \setminus C$	\times
$E \setminus D$	\times
$E \setminus A$	\times
$E \setminus B$	\times
$E \setminus C$	\times
$E \setminus D$	\checkmark

(c)

$Q^n \setminus X$	0	1
$A' \setminus Q^{n+1}/Z$	$B'/1$	$B'/0$
$B' \setminus Q^{n+1}/Z$	$B'/1$	$C'/0$
$C' \setminus Q^{n+1}/Z$	$D'/1$	$B'/1$

(d)

图 3.15 原始状态表

- (a) 原始状态表 (b) 顺序比较后的隐含表
 (c) 关连比较后的隐含表 (d) 简化后的状态表

② 关连比较

检查隐含表中所填次态是否等效。例如 ,图 3.15(b) 的隐含表中 AB 格内填的是 BE , AB 是否等效要看 BE 是否等效 ,进一步检查隐含表的 BE 格 ,发现 BE 是不等效的 ,所以 AB 也不等效 ,并在 AB 格打上斜线(图 3.15(c))。又如 , AC 是否等效要看 BC 和 BE 是否都等效 ,已知 BC 是等效的 ,但 BE 是不等效的 ,所以 AC 不等效 ,并在 AC 格打上斜线。关连比较后的隐含表如图 3.15(c) 所示。

③ 状态合并 ,求得简化后的状态表

在关连比较后,就可进行状态合并,合并后状态是(A) \wedge (BC) \wedge (DE)。对它们重新命名,把状态 A 称为 A' ,状态(BC)称为 B' ,状态(DE)称为 C' ,就可构成简化后的状态表(图3.15(d))。

3. 状态分配、求激励函数与输出函数

(1) 状态分配

给简化后的状态表中的各状态以二进制数进行编码,称为“状态分配”。例如,对图3.16(a)所示状态表进行状态分配,令 A 为 $Q_0Q_1 = 00$, B 为 $Q_0Q_1 = 10$, C 为 $Q_0Q_1 = 11$ (注意:不是“01”), D 为 $Q_0Q_1 = 01$,则其状态分配表如图3.16(b)所示。

	X	0	1
Q^n			
A	$A/0$	$B/0$	
B	$D/1$	$C/0$	
C	$B/1$	$A/1$	
D	$C/0$	$D/0$	
(Q^{n+1}/Z)			

(a)

	Q_0	0	1
Q_1			
0	A	B	
1	D	C	

(b)

	X	0	1
Q_0			
Q_1			
00	$00/0$	$01/1$	
01	$10/1$	$11/0$	
11	$01/1$	$00/1$	
10	$11/0$	$10/0$	
$(Q^{1n+1} Q^{0n+1}/Z)$			

图3.16 状态分配的例子

(a) 状态表 (b) 状态分配表

一般说来,有好几种状态分配方案。不同的编码方案对于电路的复杂性是有影响的。由于篇幅所限这里就不介绍如何选择最佳编码方案的方法了,下面的工作是选择所用的触发器。

(2) 求激励函数和输出函数

以图3.16(b)的状态分配表为例,介绍如何由状态分配表和选用的触发器求激励函数和输出函数。

假定选用的是 D 功能触发器。首先把现态和输入 X 作为卡诺图的左标和上标,按状态分配表中对应的现态和次态,在 D 触发器的激励表中找到相应位的 D 值填入其卡诺图中。

把图3.16(b)的状态分配表重画在图3.17(a)上,图3.17(b)是所用 D 功能触发器的激励表。图3.17(c)(d)分别是第0位触发器及第1位触发器的 D 卡诺图。由卡诺图就能得到第0位和第1位触发器的激励函数:

$$D_0 = X\bar{Q}_1 + \bar{X}Q_1$$

$$D_1 = Q_0\bar{Q}_1 + \bar{Q}_0Q_1$$

再在另一张卡诺图上把状态分配表中的 Z 值填入卡诺图的格中,即为 Z 的卡诺

图(图 3.17(e))。由 Z 卡诺图便可得输出函数 Z :

$$Z = \bar{X}Q_0 + Q_0Q_1 + X\bar{Q}_0\bar{Q}_1$$

最后做逻辑图(图 3.17(f))。

X	0	1
Q_1	00/0	01/1
Q_0	10/1	11/0
	01/1	00/1
	11/0	10/0

$(Q_1^{n+1}Q_0^{n+1}/Z)$

(a)

Q^n	Q^{n+1}	D
\times	1	1
\times	0	0

(b)

X	0	1
Q_1	0	1
Q_0	0	1
	1	0
	1	0

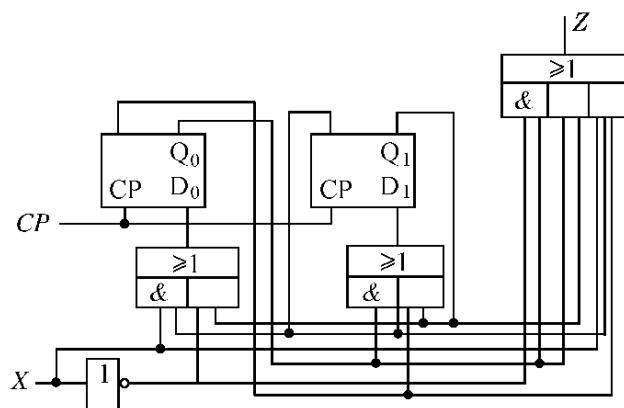
(c)

X	0	1
Q_1	0	0
Q_0	0	0
	1	1
	0	0
	1	1

(d)

X	0	1
Q_1	0	1
Q_0	0	1
	1	0
	1	1
	0	0
	0	0

(e)



(f)

图 3.17 由图 3.16(b) 所示状态分配表设计同步时序电路

(a) 状态分配表 (b) D 功能触发器激励表 (c) D_0 卡诺图

(d) D_1 卡诺图 (e) Z 卡诺图 (f) 逻辑图

3.4 典型的同步时序电路

3.4.1 移位寄存器

移位寄存器除了具有存储代码的功能以外, 还具有移位功能。所谓移位功能, 是指寄存器里存储的代码能在移位脉冲的作用下依次左移或右移。因此, 移位寄存器不但可以用来寄存代码, 还可以用来实现数据的串行—并行转换、数值的运算以及数据处理等。

图 3.18 所示电路是由边沿触发结构的 D 触发器组成的 4 位移位寄存器。其

中第一个触发器 FF_0 的输入端接收输入信号 ,其余的每个触发器输入端均与前边一个触发器的 Q 端相连。

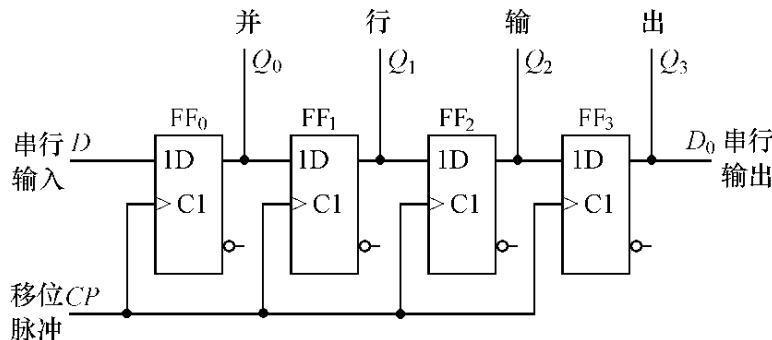


图 3.18 用 D 触发器构成的移位寄存器

因为从 CP 上升沿到达开始到输出端新状态的建立需要经过一段传输延迟时间 ,所以当 CP 的上升沿同时作用于所有的触发器时 ,它们输入端(D 端)的状态还没有改变。于是 FF_1 按 Q_0 原来的状态翻转 , FF_2 按 Q_1 原来的状态翻转 , FF_3 按 Q_2 原来的状态翻转。同时 ,加到寄存器输入端 Q_1 的代码存入 FF_0 。总的效果相当于移位寄存器里原有的代码依次右移了一位。

例如 ,在 4 个时钟周期内输入代码依次为 1011 ,而移位寄存器的初始状态为 $Q_0Q_1Q_2Q_3 = 0000$,那么在移位脉冲(也就是触发器的时钟脉冲)的作用下 移位寄存器里代码的移动情况将如表 3.11 所示。图 3.19 给出了各触发器输出端在移位过程中的电压波形图。

表 3.11 移位寄存器中代码的移动状况

CP 的顺序	输入 D	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0	0
1	1	1	0	0	0
2	0	0	1	0	0
3	1	1	0	1	0
4	1	1	1	0	1

可以看到 ,经过 4 个 CP 信号以后 ,串行输入的 4 位代码全部移入了移位寄存器中 ,同时在 4 个触发器的输出端得到了并行输出的代码。因此 ,利用移位寄存器可以实现代码的串行—并行转换。

如果首先将 4 位数据并行地置入移位寄存器的 4 个触发器中 ,然后连续加入 4 个移位脉冲 ,则移位寄存器里的 4 位代码将从串行输出端 D_0 依次送出 ,从而实现了数据的并行—串行转换。

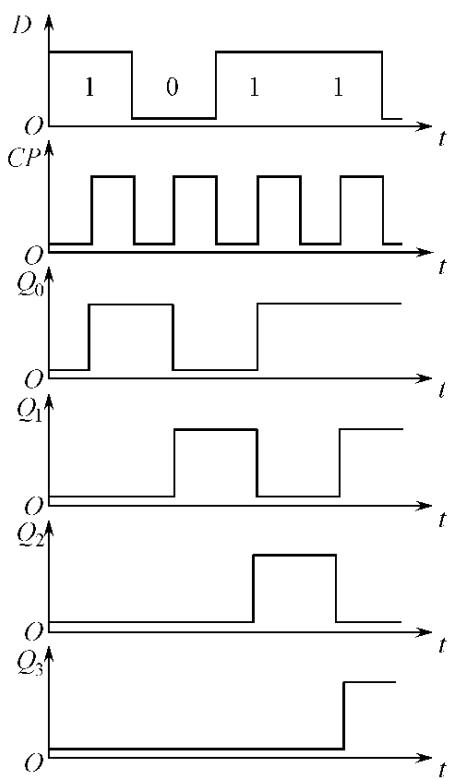


图 3.19 图 3.18 所示电路的电压波形

3.4.2 计数器

在数字系统中使用得最多的时序电路要算是计数器了。计数器不仅能用于对时钟脉冲计数，还可以用于分频、定时、产生节拍脉冲和脉冲序列以及进行数字运算等。

计数器的种类非常繁多。如果按计数器中的触发器是否同时翻转分类，可以把计数器分为同步式和异步式两种。在同步计数器中，当时钟脉冲输入时触发器的翻转是同时发生的。而在异步计数器中，触发器的翻转有先有后，不是同时发生的。

如果按计数过程中计数器中的数字增减分类，又可以把计数器分为加法计数器、减法计数器和可逆计数器（或称为加／减计数器）。随着计数脉冲的不断输入而作递增计数的叫加法计数器，作递减计数的叫减法计数器，可增可减的叫可逆计数器。

如果按计数器中数字的编码方式分类，还可以分成二进制计数器、二—十进制计数器、循环码计数器等。

此外，有时也用计数器的计数容量来区分各种不同的计数器，如十进制计数器、六十进制计数器等。

1. 同步二进制计数器

目前生产的同步计数器芯片基本上分为二进制和十进制两种。首先讨论同步二进制计数器。

根据二进制加法运算规则可知，在一个多位二进制数的末位上加 1 时，若其中第 i 位（即任何一位）以下各位皆为 1 时，则第 i 位应改变状态（由 0 变成 1，由 1 变成 0）。而最低位的状态在每次加 1 时都要改变。例如

$$\begin{array}{r} 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \\ + \qquad \qquad \qquad 1 \\ \hline 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \end{array}$$

按照上述原则，最低的 3 位数都改变了状态，而高 4 位状态未变。

同步计数器既可用 T 触发器构成，也可以用 T' 触发器构成。如果用 T 触发器构成，则每次 CP 信号（也就是计数脉冲）到达时应使该翻转的那些触发器输入控制端 $T = 1$ ，不该翻转的 $T = 0$ 。如果用 T' 触发器构成，则每次计数脉冲到达时只能加到该翻转的那些触发器的 CP 输入端上，而不能加给那些不该翻转的触发器。

由此可知，当计数器用 T 触发器构成时，第 i 位触发器输入端的逻辑式应为

$$\begin{aligned} T_i &= Q_{i-1} \cdot Q_{i-2} \dots Q_1 \cdot Q_0 \\ &= \prod_{j=0}^{i-1} Q_j \quad (i = 1, 2, \dots, n-1) \end{aligned} \quad (3.1)$$

只有最低位例外，按照计数规则，每次输入计数脉冲时它都要翻转，故 $T_0 = 1$ 。

图 3.20 所示电路就是按式(3.1)接成的 4 位二进制同步加法计数器。由图可见，各触发器的驱动方程为：

$$\begin{cases} T_0 = 1 \\ T_1 = Q_0 \\ T_2 = Q_0 Q_1 \\ T_3 = Q_0 Q_1 Q_2 \end{cases} \quad (3.2)$$

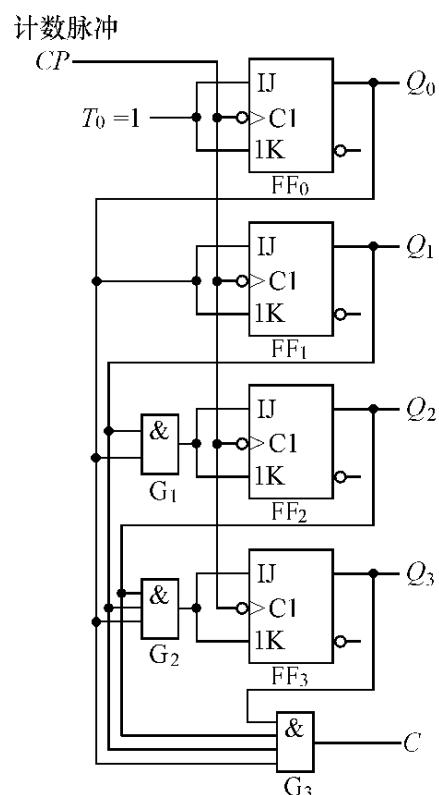


图 3.20 用 T 触发器构成的同步二进制加法计数器

将上式代入触发器的特性方程式得到电路的状态方程

$$\begin{cases} Q_0^{n+1} = \bar{Q}_0 \\ Q_1^{n+1} = Q_0\bar{Q}_1 + \bar{Q}_0Q_1 \\ Q_2^{n+1} = Q_0Q_1\bar{Q}_2 + \overline{Q_0Q_1}Q_2 \\ Q_3^{n+1} = Q_0Q_1Q_2\bar{Q}_3 + \overline{Q_0Q_1Q_2}Q_3 \end{cases} \quad (3.3)$$

电路的输出方程为

$$C = Q_0Q_1Q_2Q_3 \quad (3.4)$$

根据式(3.3)和式(3.4)求出电路的状态转换表如表3.12所示。利用第16个计数脉冲到达时 C 端电位的下降沿可作为向高位计数器电路进位的输出信号。

表 3.12 图 3.20 电路的状态转换表

计数顺序	电路状态				等效 十进制数	进位输出 C
	Q_3	Q_2	Q_1	Q_0		
0	0	0	0	0	0	0
1	0	0	0	1	1	0
2	0	0	1	0	2	0
3	0	0	1	1	3	0
4	0	1	0	0	4	0
5	0	1	0	1	5	0
6	0	1	1	0	6	0
7	0	1	1	1	7	0
8	1	0	0	0	8	0
9	1	0	0	1	9	0
10	1	0	1	0	10	0
11	1	0	1	1	11	0
12	1	1	0	0	12	0
13	1	1	0	1	13	0
14	1	1	1	0	14	0
15	1	1	1	1	15	1
16	0	0	0	0	0	0

图3.21和图3.22是图3.20电路的状态转换图和时序图。由时序图上可以看出，若计数输入脉冲的频率为 f_0 ，则 Q_0 、 Q_1 、 Q_2 和 Q_3 端输出脉冲的频率将依次

为 $\frac{1}{2}f_0$ 、 $\frac{1}{4}f_0$ 、 $\frac{1}{8}f_0$ 和 $\frac{1}{16}f_0$ 。针对计数器的这种分频功能 ,也把它叫做分频器。

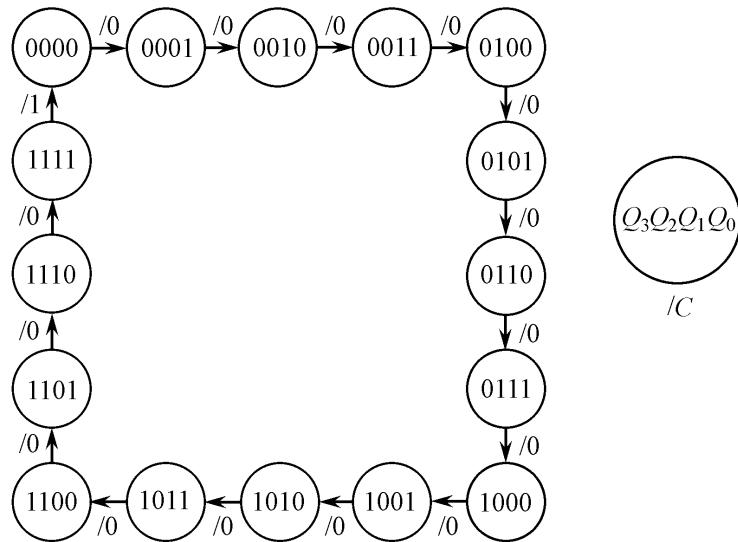
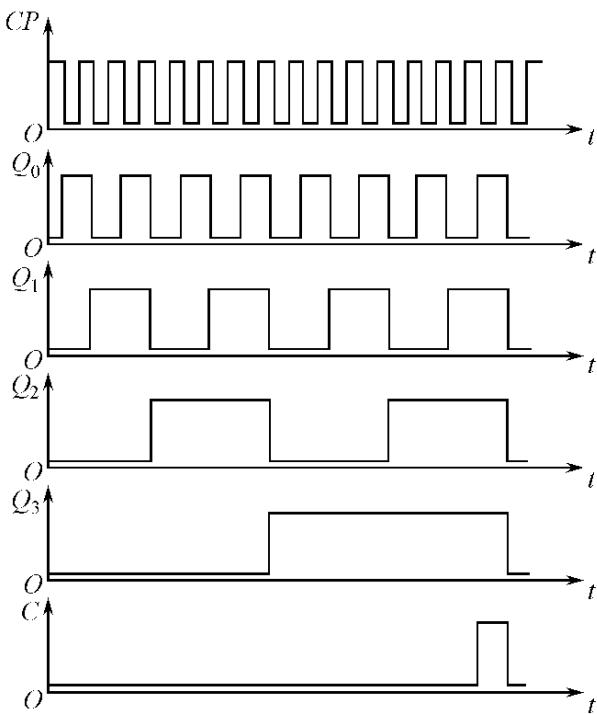


图 3.21 图 3.20 所示电路的状态转换图



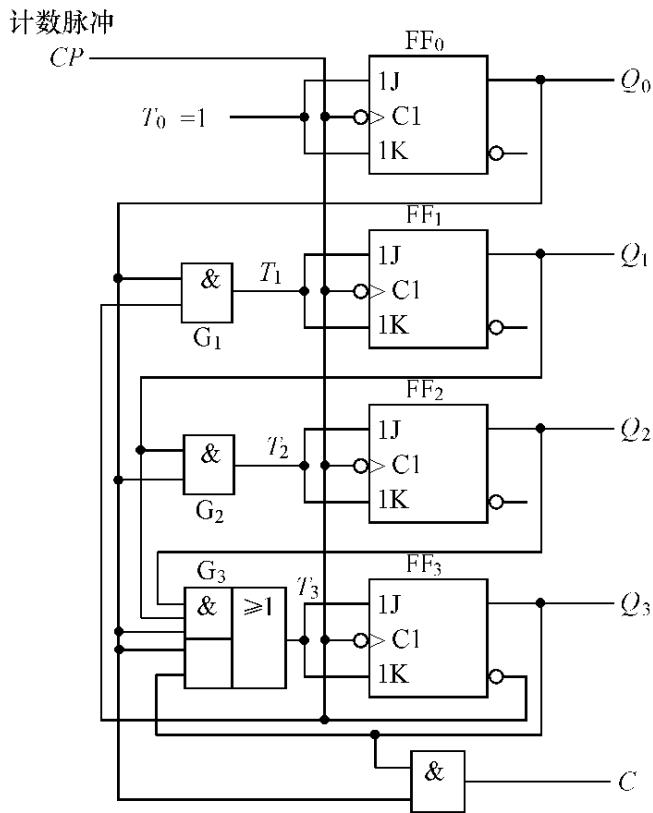


图 3.23 同步十进制加法计数器电路

输出为 1, 所以 4 个触发器的输入控制端分别为 $T_0 = 1$ 、 $T_1 = 0$ 、 $T_2 = 0$ 、 $T_3 = 1$ 。因此, 当第 10 个计数脉冲输入后, FF_1 和 FF_2 维持 0 状态不变, FF_0 和 FF_3 从 1 翻转为 0, 故电路返回 0000 状态。

从逻辑图上可写出电路的驱动方程为

$$\begin{cases} T_0 = 1 \\ T_1 = Q_0 \bar{Q}_3 \\ T_2 = Q_0 Q_1 \\ T_3 = Q_0 Q_1 Q_2 + Q_0 Q_3 \end{cases} \quad (3.5)$$

将上式代入 T 触发器的特性方程, 即可得电路的状态方程

$$\begin{cases} Q_0^{n+1} = \bar{Q}_0 \\ Q_1^{n+1} = Q_0 \bar{Q}_3 \bar{Q}_1 + \bar{Q}_0 \bar{Q}_3 Q_1 \\ Q_2^{n+1} = Q_0 Q_1 \bar{Q}_2 + \bar{Q}_0 Q_1 Q_2 \\ Q_3^{n+1} = (Q_0 Q_1 Q_2 + Q_0 Q_3) \bar{Q}_3 + (\bar{Q}_0 Q_1 Q_2 + \bar{Q}_0 Q_3) Q_3 \end{cases} \quad (3.6)$$

根据式(3.6)还可以进一步列出表 3.13 的电路状态转换表, 并画出如图 3.24 所示的电路状态转换图。由状态转换图上可见, 这个电路是能够自启动的。

表 3.13 图 3.23 电路的状态转换表

计数顺序	电路状态				等效十进制数	输出 C
	Q_3	Q_2	Q_1	Q_0		
0	0	0	0	0	0	0
1	0	0	0	1	1	0
2	0	0	1	0	2	0
3	0	0	1	1	3	0
4	0	1	0	0	4	0
5	0	1	0	1	5	0
6	0	1	1	0	6	0
7	0	1	1	1	7	0
8	1	0	0	0	8	0
9	1	0	0	1	9	1
10	0	0	0	0	0	0
0	1	0	1	0	10	0
1	1	0	1	1	11	1
2	0	1	1	0	6	0
0	1	1	0	0	12	0
1	1	1	0	1	13	1
2	0	1	0	0	4	0
0	1	1	1	0	14	0
1	1	1	1	1	15	1
2	0	0	1	0	2	0

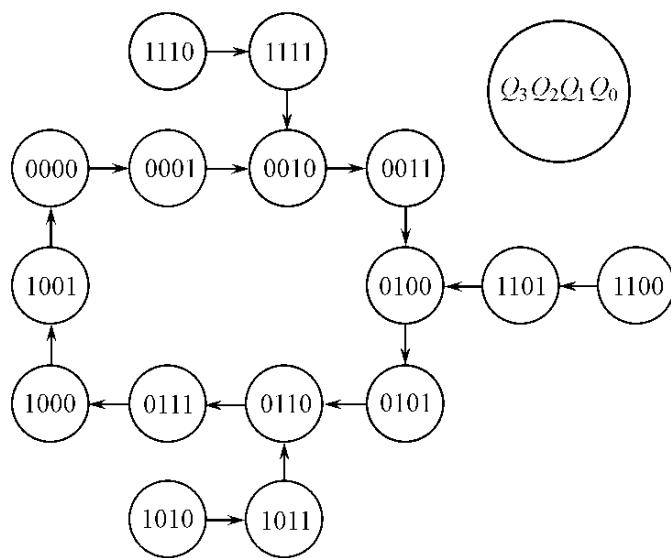


图 3.24 图 3.23 所示电路的状态转换图

3.4.3 节拍信号发生器

在数字信号的传输和数字系统的测试中,有时需要用到一组特定的串行数字信号。通常把这种串行数字信号叫做序列信号。产生序列信号的电路称为序列

信号发生器。序列信号发生器的构成方法有多种。一种比较简单、直观的方法是用计数器和数据选择器组成。例如,需要产生一个8位的序列信号00010111(时间顺序为自左而右),则可用一个八进制计数器和一个8选1数据选择器组成,如图3.25所示。其中八进制计数器取自74LS161(4位二进制计数器)的低3位。74LS152是8选1数据选择器。

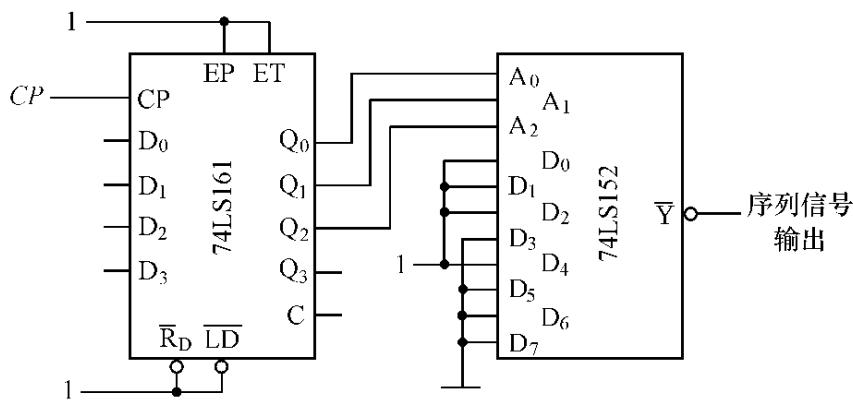


图3.25 用计数器和数据选择器组成的序列信号发生器

当 CP 信号连续不断地加到计数器上时, $Q_2Q_1Q_0$ 的状态(也就是加到74LS152上的地址输入代码 $A_2A_1A_0$)便按照表3.14中所示的顺序不断循环。 $\bar{D}_0 \sim \bar{D}_7$ 的状态就循环不断地依次出现在 \bar{Y} 端。只要令 $D_0 = D_1 = D_2 = D_4 = 1$ 、 $D_3 = D_5 = D_6 = D_7 = 0$,便可再 \bar{Y} 端得到不断循环的序列信号00010111。在需要修改序列信号时,只要修改加到 $\bar{D}_0 \sim \bar{D}_7$ 的高、低电平即可实现,而不需对电路结构做任何变动。因此,使用这种思路既灵活又方便。

表3.14 图3.25所示电路的状态转换表

CP 顺序	Q_2 (A_2)	Q_1 (A_1)	Q_0 (A_0)	\bar{Y}
0	0	0	0	$\bar{D}_0(0)$
1	0	0	1	$\bar{D}_1(0)$
2	0	1	0	$\bar{D}_2(0)$
3	0	1	1	$\bar{D}_3(1)$
4	1	0	0	$\bar{D}_4(0)$
5	1	0	1	$\bar{D}_5(1)$
6	1	1	0	$\bar{D}_6(1)$
7	1	1	1	$\bar{D}_7(1)$
8	0	0	0	$\bar{D}_0(0)$

构成序列信号发生器的另一种常见方法是采用带反馈逻辑电路的移位寄存器。如果序列信号的位数为 m , 移位寄存器的位数为 n , 则应取 $2^n \geq m$ 。例如, 若仍然要求产生 00010111 这样一组 8 位的序列信号, 则可用 3 位的移位寄存器加上反馈电路构成所需的序列信号发生器, 如图 3.26 所示。移位寄存器从 Q_2 端输出的串行输出信号就应当是所要求的序列信号。

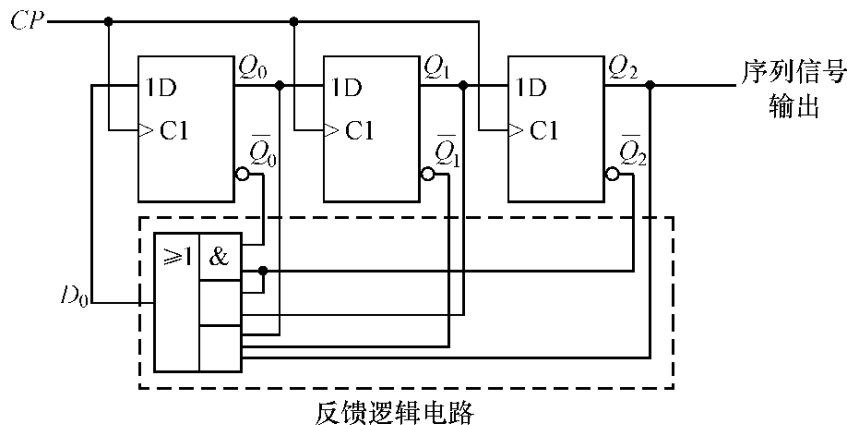


图 3.26 采用带反馈逻辑电路的移位寄存器

根据要求产生的序列信号, 即可列出移位寄存器应具有的状态转换表, 如表 3.15 所示。再从状态转换的要求出发, 得到对移位寄存器输入端 D_0 取值的要求, 如表 3.15 中所示。表中也同时给出了 D_0 与 Q_2 、 Q_1 、 Q_0 之间的函数关系。利用图 3.27 的卡诺图将 D_0 的函数式化简, 得到

$$D_0 = Q_2 \bar{Q}_1 Q_0 + \bar{Q}_2 Q_1 + \bar{Q}_2 \bar{Q}_0 \quad (3.7)$$

图 3.25 即是按式 3.7 接成的逻辑电路。

表 3.15 图 3.26 所示电路的状态转换表

CP 顺序	Q_2	Q_1	Q_0	D_0
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	1	0	1	1
4	0	1	1	1
5	1	1	1	0
6	1	1	0	0
7	1	0	0	0
8	0	0	0	0

Q_2	$Q_1 Q_0$	00	01	11	10
0	00	1	0	1	1
1	01	0	1	0	0

图 3.27 图 3.26 中的卡诺图

习 题

1. 分析图 3.28 所示电路的逻辑功能, 写出输出的逻辑函数式, 列出真值表, 说明电路逻辑功能的特点。

2. 图 3.29 所示是一个多功能函数发生电路。试写出当 $S_0 S_1 S_2 S_3$ 为 0000 ~ 1111 ,16 种不同状态时输出 Y 的逻辑函数式。

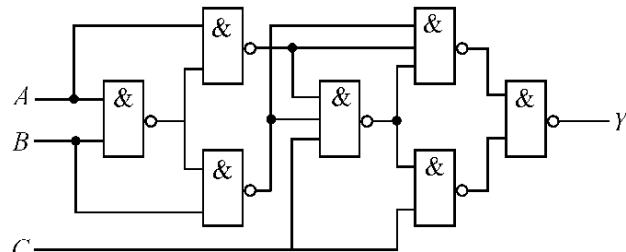


图 3.28

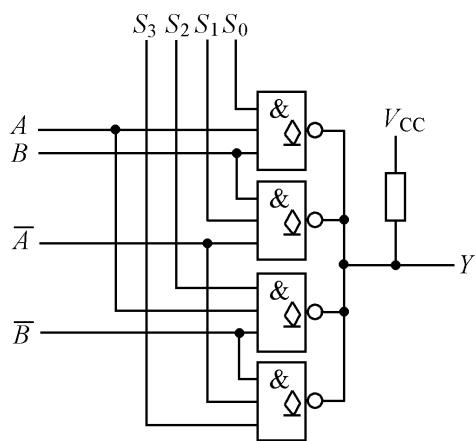


图 3.29

3. 试用数据选择器设计一个“逻辑不一致”电路, 要求 4 个输入逻辑变量取值不一致时

输出为 1 , 取值一致时输出为 0。

4. 什么叫竞争—冒险现象 ?当门电路的两个输入端同时向相反的逻辑状态转换(即一个从 0 变为 1 ,另一个从 1 变成 0)时 输出端是否一定有干扰脉冲产生 ?
5. 触发器有哪几种常见的电路结构形式 ?它们各有什么样的动作特点 ?
6. 分别写出 $R - S$ 触发器、 $J - K$ 触发器、 T 触发器和 D 触发器的特性表和特性方程。
7. 触发器的逻辑功能和电路结构形式之间的关系如何 ?
8. 试比较时序逻辑电路和组合逻辑电路在逻辑功能上和电路结构上有何不同。
9. 分析图 3.30 时序电路的逻辑功能 ,写出电路的驱动方程、状态方程和输出方程 ,画出电路的状态转换图 ,并说明该电路能否自启动。

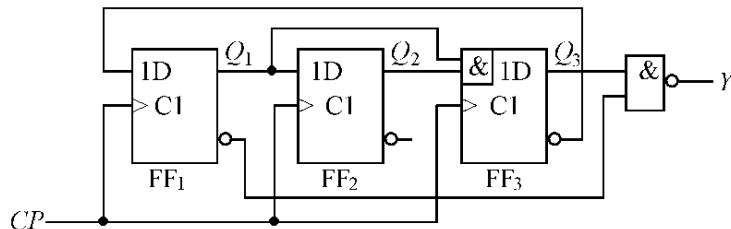


图 3.30

第四章 计算机设计引论

经过前三章的学习,我们已经掌握了数字系统设计的基本原理和基本方法。计算机作为人们最熟悉的数字系统范例之一,其主要数字电路构成、结构原理、设计方法,无疑是计算机专业学生学习数字系统设计的重要目的和最佳对象。在应用数字系统设计的基本原理和基本方法开始简易计算机设计之前,首先必须明确要设计的对象到底是什么,即它具有什么功能;其次要确定如何组织和实现这些功能是最高效和经济的。完全抛开前人的实践经验和研究成果,按数字系统的设计原理和方法设计计算机不是没有可能而是没有必要,因此掌握现代计算机的基本特征和硬件系统结构组成是非常必要的。

早在 1812 年,英国人 Babbage 首先提出了计算过程自动化的概念,指出自动计算机必须具有输入、输出、处理、存储、控制五大功能,以及计算机只有具有记忆功能,能记住数据和要进行的操作步骤,并按这些步骤规定对机器进行自动控制,才能实现自动计算的思想。1854 年,英国数学家 George Boole 发表了《布尔代数》,为采用二进制的数字计算机提供了理论基础。1945 年美籍匈牙利数学家冯·诺依曼等发表的《关于电子计算装置逻辑结构初探》一文,提出了以二进制和存储程序控制为核心的通用电子数字计算机体系结构原理,奠定了当代电子计算机体系结构的基础。

4.1 存储程序控制原理

存储程序控制原理的基本思想是:计算机要自动完成解题任务,必须将事先设计好的,用于描述计算机解题过程、为计算机所理解的一组命令的有序集合(即程序)如同数据一样,采用二进制形式存储在计算机中;计算机在工作时自动高速地从机器中取出命令并加以执行,下一条命令的选取依赖于当前命令执行的结果。

存储程序控制是计算机能自动工作的关键,也是计算机区别于计算器的根本所在。计算器由于没有存储功能,它的解题过程就必须由人工在机器外干预和控制执行。

存储程序控制原理概括起来有如下特点：

- 使用单一处理部件完成计算、存储和通信功能；
- 线性组织的定长存储单元；
- 存储空间的单元是直接寻址的；
- 使用低级的机器语言，其指令完成简单的基本操作；
- 数据和指令均采用二进制形式，且它们的存储是相同的；
- 对计算进行集中的顺序控制。

通常称这种结构的计算机为“存储程序计算机”。

按照存储程序控制原理，计算机必须具有五大功能，如图 4.1 所示。

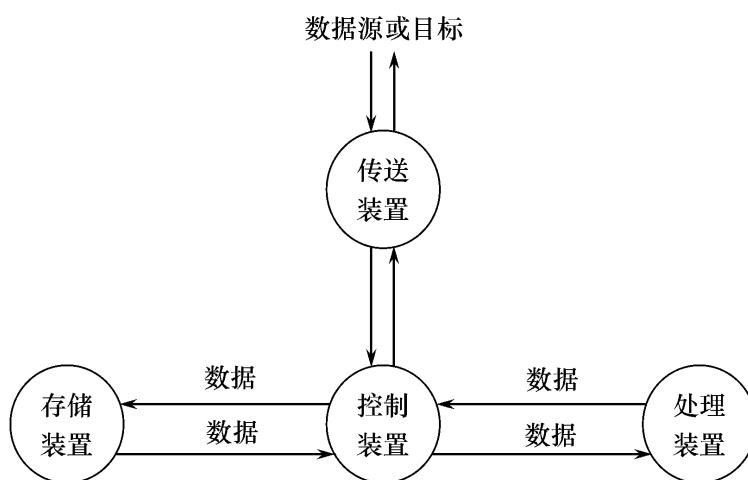


图 4.1 计算机功能描述

1. 传送功能

计算机必须有能力将原始数据和解题程序输入到机器中，而计算的结果和计算过程中出现的情况也能随时输出给用户。即计算机必须具有输入和输出功能。

2. 存储功能

计算机应能“记住”用户所提供的原始数据和解题程序，以及解题过程中的一些中间结果，即具备数据存储功能，这是计算机能实现自动运算的关键。

3. 处理功能

计算机应能进行一些最基本的算术运算和逻辑运算，从而组合成人类所需要的一切复杂的运算和处理。这是计算机进行运算、处理和控制的基础。

4. 操作控制功能

计算机应能控制和协调其各部件的操作，以保证程序执行的正确性。

5. 操作判断功能

在完成一步操作后，计算机应能从预先无法确定的几种方案中选择一种方

案 ,从而保证解题操作正确完成。

4.2 计算机硬件系统组成

根据存储程序计算机的五大功能 ,计算机硬件系统由运算器、控制器、存储器、输入设备、输出设备以及将它们连接为有机整体的总线构成 ,如图 4.2 所示。

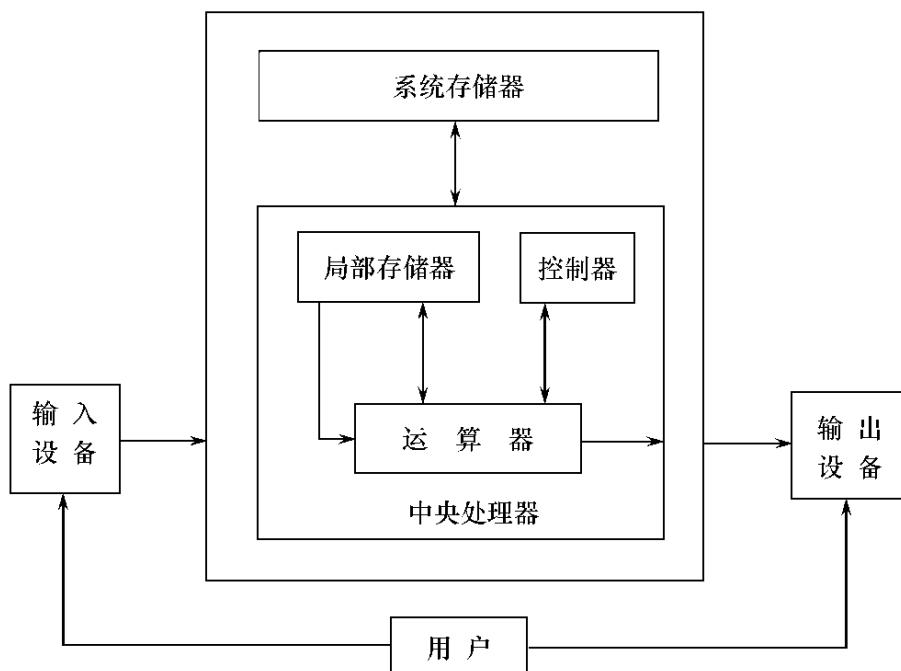


图 4.2 计算机硬件系统组成

1. 运算器

运算器是进行数据处理的部件 ,它不仅要完成算术运算和逻辑运算 ,而且还作为事件处理期间数据流动的通道 ,故又称为数据通道。

运算器通常由算术逻辑单元 ALU(Arithmetic and Logic Unit)和一系列寄存器 (Register)组成。 ALU 是具体完成算术逻辑运算的部件 ,由加法器和逻辑运算器件组成 ,它是运算器的核心。寄存器是由一组独立的存储单元构成的 ,用于存放操作数的器件。累加器(Accumulator)是一种特殊的寄存器 ,它不仅用于存放操作数 ,还用于存放中间结果和最后结果 ,因具有连续“累加”功能 ,被称为累加器。寄存器和累加器均从存储器取数据 ,累加器中的结果也送回存储器。

2. 控制器

控制器是计算机的管理机构和指挥中心 ,它指挥协调计算机各部件自动地工作。控制器的实质就是解释程序 ,它按程序规定的顺序 ,每次从存储器中读取一条指令 ,经过分析译码 ,产生一系列的时序控制信号 ,发向各个部件以控制它

们的操作。

完成上述功能的控制器由指令控制部件、地址形成部件、时序部件和操作控制部件组成。其中 指令控制部件包括程序计数器 PC(Program Counter) , 指令寄存器 IR(Instruction Register) 和指令译码器 ID(Instruction Decoder) ; 时序部件也称节拍信号发生器 , 包括时钟 CP(Clock Pulse) 和时序信号产生器 TSG(Timing Signal Generator) , 如图 4.3 所示。

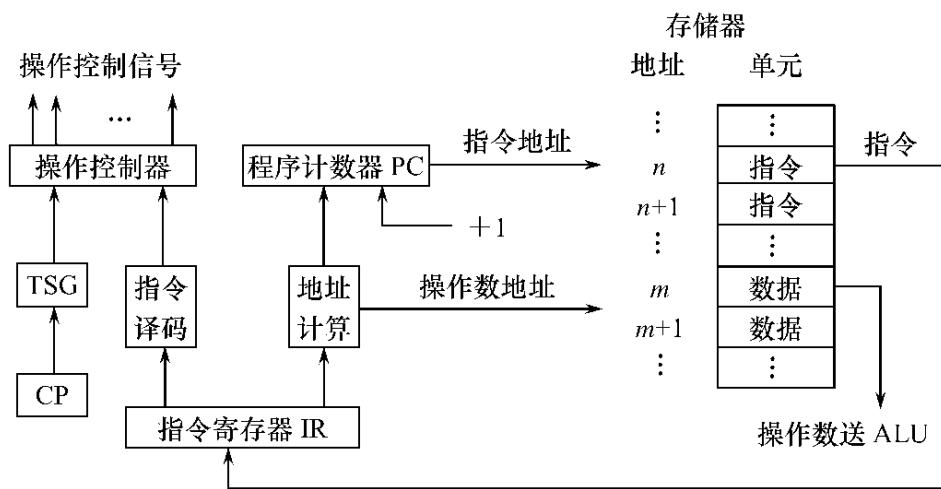


图 4.3 控制器工作原理图

程序计数器给出下一条准备执行指令在存储器中的单元地址。控制器根据 PC 的内容从存储器取出指令到 IR , PC 将自动加 1 , 指向下一条指令。若非顺序指行 , 只要将 PC 的内容作相应改变 , 就可按新的序列顺序执行。 IR 保存当前正在执行的指令 , 并分别把指令中的操作码和操作数地址送指令译码器和地址计算部件。指令译码器又称为操作码译码器 , 它将指令的操作码转换为相应的控制电位信号 , 指示各部件做什么操作。

计算机在执行一条指令时 , 总是把一条指令分成若干基本操作 , 由控制器发出一系列时钟脉冲和节拍电位 , 每个时钟脉冲和节拍电位信号控制计算机完成一个或几个基本操作。节拍信号发生器就是用来产生时钟脉冲和节拍电位的逻辑部件。时钟脉冲由 CP 产生 , 它是协调计算机各部件进行操作的同步时钟 , 其工作频率称为计算机的主频。主频的高低直接影响计算机的工作速度。节拍电位由 TSG 产生 , 其宽度为主频周期 , 个数为单条指令分解的基本操作个数的最大值。 TSG 不断重复产生这些节拍电位信号 , 各部件在不同的节拍信号控制下自动地工作。

控制器与运算器一起构成计算机的中央处理器 CPU(Central Processing Unit)。在图 4.2 所示的中央处理器中 , 包括一个局部存储器 , 它是介于中央处理器和主存储器之间的小容量高速缓冲存储器(Cache) , 它允许中央处理器执行操

作而不必总是从主存储器中检索数据。现代高性能微处理器芯片都把一级 Cache 纳入其中。

3. 存储器

电子计算机的特点之一是具有存储记忆功能 ,理所当然要有存储信息的装置——存储器。存储器的主要功能是存放数据和程序。目前的计算机系统 ,大都采用多种类型存储器 ,几乎没有只用单一存储器的情形。原因是要在容量、速度、价格三者之间折衷。

计算机的存储器分为内存和外存。内存包括高速缓冲存储器和主存储器 ,早期的计算机由于没有高速缓冲存储器 ,因此将主存储器简称为内存 ;外存也称辅助存储器 ,包括磁盘、光盘和磁带等。高速缓冲存储器用于存放当前正在执行程序的部分程序段和数据 ;主存储器用于存放当前处于活动状态的程序和有关数据 ,CPU 通过指令可直接访问它。外存与内存的最大区别就在于它不能被 CPU 直接访问 ,而必须通过专门的程序或通道把信息调入主存后才能使用。

主存储器由一个一个的存储单元组成 ,每个存储单元包含若干存储位元。一个存储单元可存放一个机器字(Word)或一个字节(Byte)。主存储器的存储单元按某种顺序编号 此编号称为单元地址。只要给定一个存储单元地址 就可以通过地址译码器译码 找到对应的存储单元 ,从而对该单元进行读写操作。

4. 输入设备

输入设备的作用是将数据和程序送入计算机。常见的输入设备有键盘类、指点类(如鼠标、光笔、触摸技术等)、扫描类(如扫描仪、条形码等)、传感类(如传感器、调制解调器)和语音类。它们多是机电混合装置 ,与运算器、存储器等纯电子部件相比 ,速度较慢 ,一般须通过某种界面(接口)与之相连。

5. 输出设备

与输入设备正好相反 ,输出设备是将计算机处理的结果转化为人或其他设备所能识别或接收的信息形式的装置。例如 ,显示器能将信息转化为字符、汉字、图形、图像 ;打印机能将结果打印成文件的形式 ,绘图机能绘出五彩缤纷的高质量图形 ,音像系统能将声音和图像直接输出或记录到磁带上。和输入设备一样 ,输出设备也多为机电装置 ,也需通过某种界面(接口)与运算器和存储器相连。

某些设备同时具有输入和输出的功能。例如 ,调制解调器(modem)允许用户发送和接收数据。

6. 总线

总线(Bus)是计算机系统中各模块之间传送信息的公共通路 ,由传输信息的电路和管理信息传输的协议组成。在现代计算机系统中 ,总线往往是计算机数

据交换的中心,总线的结构和性能直接影响着计算机系统的性能和效率。

4.3 简易计算机设计总体构想

计算机是一个复杂的数字系统,综上所述,应当包含运算器、控制器、存储器、输入设备、输出设备以及将它们连接为有机整体的总线。各部分完全从零开始设计不仅非常繁琐,而且影响对计算机工作原理这条主线的把握。中央处理器作为计算机的核心,无疑是设计的重点,将在第七章详细叙述。总线由于采用了前面章节没有介绍的 OC 门电路,因此有必要专门介绍,见第五章。存储器因用途不同而种类繁多、结构复杂,采用专门的芯片进行设计较为合理,第六章给出了存储器的工作原理,以便于正确选用。输入/输出设备多为机电混合装置,且基于微处理器的计算机设计和输入/输出接口将在第八、九章介绍,简易计算机的设计将不包含输入/输出部分,但为了使简易计算机具有一定的输入/输出功能,可以利用微机将输入程序写入 ROM 并取代简易计算机的 RAM,从而实现输入,或从微机中直接下载程序到简易计算机的 RAM(如果实验平台支持),输出可以采用最简单的发光二极管显示二进制代码,或从简易计算机的 RAM 上载到微机进行显示(如果实验平台支持)。

习 题

1. 计算机与计算器的根本区别是什么?
2. 什么是存储程序原理?按照该原理,计算机应具备哪些功能?
3. 简述计算机硬件系统的组成。
4. 控制器的作用是什么?它主要由哪些部件组成?简述各部件的功能。
5. 什么是程序计数器?计算机如何区分存储器中存放的指令和数据?
6. 解释下列概念:
CPU ;ALU ;主频 ;指令寄存器 ;指令译码器 ;节拍信号发生器

第五章 总 线

5.1 总线的概念

总线是计算机系统的骨架,是连接各功能部件的纽带,它为系统部件之间的数据传送提供公共通路。自从1970年美国DEC公司在其PDP11/20小型计算机上采用Unibus以来,各种标准的、非标准的总线纷纷面世。由于它能简化系统设计,便于组织不同专业化厂家大规模生产,降低产品成本,提高产品的性能和质量,便于产品的更新换代,满足不同用户需求以及提高可维修性和可扩充性等,因而得到迅速发展。采用总线结构是计算机系统结构的一大进步,总线结构的好坏对系统性能有很大的影响。

我们知道,计算机的工作过程就是信息在计算机各部件(器件)间不断有序流动的过程。参考图4.2所示的计算机硬件系统组成,假若各系统部件不采用总线结构连接,要实现每个部件都能够向其他任一部件传送数据,最显然的解决方案是每个部件都连有一个多路选择器(MUX),如图5.1所示。该方案的最大缺陷是线路较多,且与连接的对象成正比,尤其是这些线路用于连接物理上分离的器件,因此相对较长。分析该连接方式不难发现,造成线路多的原因是部件(器件)间的传输是一对一进行的,即传输线路是独占的,即使传输路径相同也是如此。独占的好处是数据的传输不受其他设备的限制。

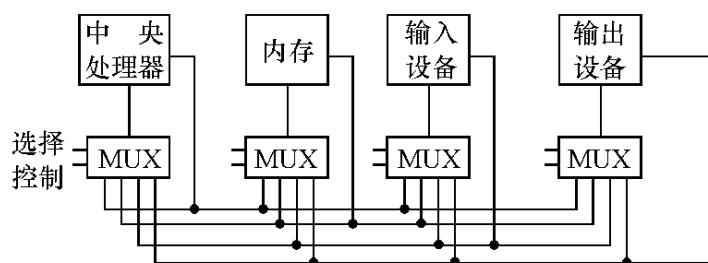


图5.1 部件间非总线连接结构

自从计算机诞生以来,人们对计算机各部件间的互连结构进行了大量的研究,产生了各种各样的互连模式,其中最常用的就是总线结构。它通过对各部件的收发方式进行一定的限制,从而实现传输线路的共享,得到了更为经济的解决

方案。所作的限制是在任何时刻只能有一对部件(器件)进行通信 ,或者放宽这个限制 ,在任何时刻最多只有一个部件(器件)作为数据源 ,而数据接收则通过接收信号控制。这种限制在大多数的电路中是很容易被接受的。图 5.2 展示了不同单元所构成的系统由共享的线路 ,即所谓的总线实现的连接结构。由于在任何时刻最多只有一个部件(器件)被作为数据源而进行连接 ,因此其他部件(器件)必须被断开。这样开关必不可少。本书到目前为止还没有介绍这样的器件 ,在本章后面的小节中将讨论两种被广泛使用的门电路实现开关的功能 ,那就是集电极开路电路和三态门电路。

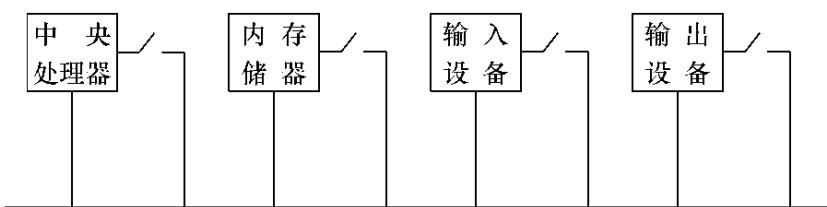


图 5.2 部件间总线连接结构

综上所述 ,总线是连接两个以上设备进行通信的共享数据通路 ;它的主要特征是传输介质是多设备共享的 ;从数据传输的角度看 ,连接到总线上的设备不能同时使用总线。总线的使用具有如下主要特点 :

(1) 多部件之间采用总线连接方式 ,可以大幅度减少连线数量 ,大大降低部件之间互连的复杂性。

(2) 多部件间的总线连接方式 ,使任一部件与其他多部件间的多个控制接口变成每个部件与总线之间的一个控制接口 ,接口数量和器材量大幅度减少 ,且有利于接口的标准化。

(3) 适合设备之间没有或者很少有多个部件同时进行信息交换的场合 ,否则 ,大量同时工作的部件采用分时共享信息通道的总线方式 ,会严重降低部件间进行信息交换的效率。

5.2 总线的组成

从逻辑构成上看 ,总线包含三个部分 :一是连接设备的信号线 ,即总线通道 ;二是总线上的设备与接口 ;三是管理总线使用的部件 ,即总线控制器。

5.2.1 总线通道

总线是信号线的集合,典型总线包含大约 50~200 根信号线,这些信号线可分为五种类型:

1. 数据线

数据线又称为数据总线,是总线设备进行数据信息交换的通道。常见总线的数据线有 1 根(串行总线)、8 根、16 根、32 根和 64 根,某些高性能总线的数据线数量达到 128 根甚至更多。一般称总线中数据线的根数为总线宽度,单位用“位”表示。总线宽度决定了总线一次能够传送的数据位数。

2. 地址线

地址线又称为地址总线,用于指示发送或接收数据的设备。常见总线的地址线有 8 根、16 根和 32 根。与数据总线相同,总线中地址线的根数称为地址总线宽度,单位也用“位”表示。地址总线宽度决定了总线的直接寻址范围及总线上连接设备的能力。例如,ISA 总线具有 24 位访问存储器的地址线和 16 位访问 I/O 设备的地址线,因此它的最大存储器扩展能力为 16 MB,I/O 设备的最大扩展能力为 64K 个设备。

3. 控制线

控制线又称为控制总线,用于传送各种控制信号,控制总线设备对数据线和地址线的使用。控制信号主要包括定时信号和命令信号,其中定时信号标明有效的地址和数据出现在总线上的时间,命令信号定义总线上所要完成的操作。控制总线决定了总线功能的强弱和适应性的好坏。

4. 电源线

为连接到总线上的设备提供电源。电源线不是必备的信号线。

5. 备用线

留给用户进行性能扩充,满足特殊需求。备用线不是必备的信号线。

5.2.2 总线上的设备

连接到总线上的设备是多种多样的,根据从不同侧面看设备在总线上发挥的作用,可以对总线设备进行不同的分类。

按有无对总线的控制功能,连接在总线上的设备可以分为总线主设备和总线从设备两种。前者能够申请并获得总线的使用权,引发一次总线操作,一般具有较完备的总线控制功能;而后者不能申请总线使用权,不能引发总线操作,它

只能在总线操作中作为被操作的对象。在任何时候,一条总线上工作的总线主设备不允许超过一个,否则将导致总线使用权的混乱,从而引发总线上信息的混乱。多个总线主设备同时要求使用总线时,总线控制器必须按一定的判定原则,对总线的使用权进行仲裁。

按信息的流向,连接在总线上的设备可以分为总线源设备和总线目标设备两种。前者是发生数据的设备,后者是接收数据的设备。总线源设备与总线主设备没有必然的联系,总线源设备可以是某一总线主设备启动的总线从设备;同样,总线目标设备与总线从设备也没有必然的联系。

按访问设备的方法,连接在总线上的设备可以分为存储器设备和I/O设备两种。前者以访问存储器的方法访问,后者以访问I/O的方法访问。

5.2.3 总线控制器

总线控制器是总线系统的核心,它的任务概括地说是管理总线的使用,包括总线上设备的管理和设备使用总线的过程管理。具体的功能有,总线系统资源的管理、总线系统的定时、总线使用权的仲裁和不同总线协议的转换。图5.3展示了逻辑概念上的总线控制器,因为在总线控制器的具体实现中,并不一定存在一个独立的控制器,它的功能可能被分布到总线的各个部件或者各个设备上了。

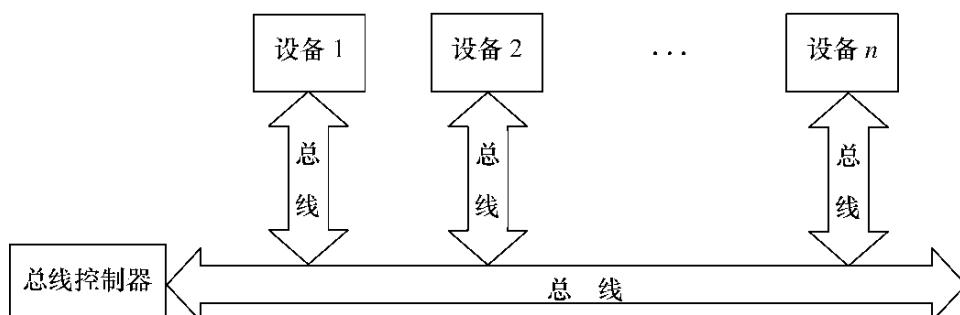


图 5.3 逻辑概念上的总线控制器

5.3 集电极开路总线和三态门总线

为了使多个设备共享总线,必须将这些设备的输出都连接到总线上,即要实现多个元件的输出都作为另一元件的一个输入进行连接,且在任一时刻最多有一个输出被选中。但是,到目前为止,本书介绍的所有门电路都只能进行单向连接,即元件的输出只能连接到另外某一元件作为输入,而不能进行输出间的互连。对于这个问题,有两种被广泛采用的解决方案,那就是集电极开路与非门

(OC门)电路和三态门电路。

5.3.1 集电极开路与非门(OC门)

典型的 TTL(Transistor – Transistor Logic)与非门电路如图 5.4 所示。它们的输出端是不能直接相连的,因为无论输出的是高电平还是低电平,它们的输出电阻都很低,直接相连就可能出现图 5.5 所示的情况。当门 1 输出低电平、即门 1 的 V_5 导通,门 2 输出高电平、即门 2 的 V_4 导通时,就会形成一个很大的电流,经门 2 的 V_4 管到门 1 的 V_5 管,该电流不仅会使门 1 的输出低电平抬高,破坏电路的逻辑关系,而且还会因功耗过大使器件很快烧坏。

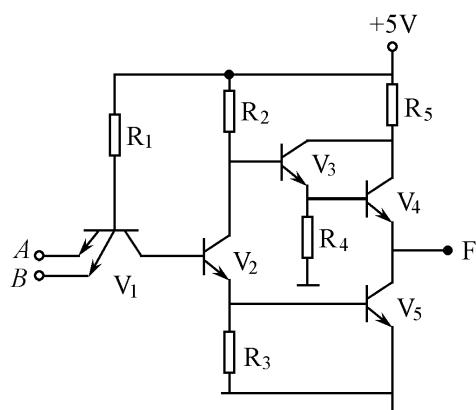


图 5.4 TTL 与非门

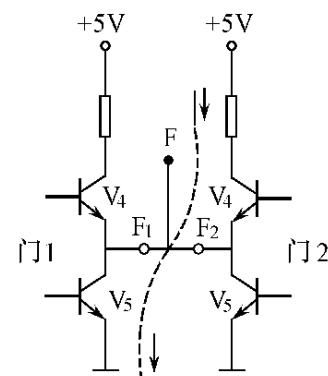
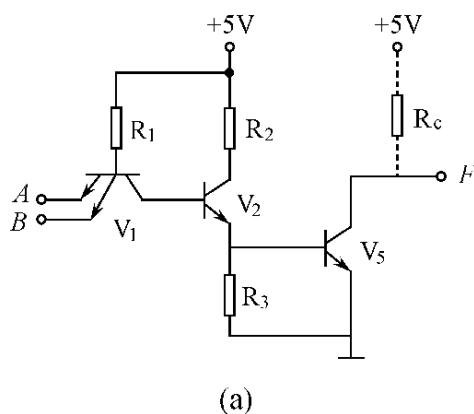
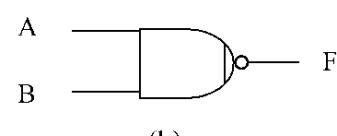


图 5.5 TTL 与非门输出相连产生的电流

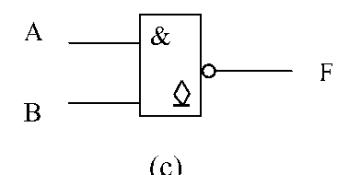
对 TTL 与非门进行改进,去掉晶体管 V_3 、 V_4 及电阻 R_4 、 R_5 ,让 V_5 的集电极直接输出,使它的输出端可连在一起而器件不损坏,这种门电路称为集电极开路与非门电路,简称 OC(Open Collector)门。由于“开路”,其值没有定义,如果要输出 1,必须在其他的地方限定,所以实际使用时,OC 门的输出端必须接一个(外部)上拉电阻并接上正电源,如图 5.6 所示。



(a) 电路图



(b)



(c)

图 5.6 TTL 集电极开路与非门
(a) 电路图 (b) 传统逻辑符号 (c) IEEE 逻辑符号

多个 OC 门的输出端连接在一起，并加接一公用负载电阻，可实现线与功能构成与或非门，如图 5.7 所示。当 OC 门有一个输入端为低电平时，其输出管截止，相当于此 OC 门和负载电阻 R_c 相脱离；所有 OC 门的输出管都截止， F 端为高电平。当 OC 门的所有输入均为高电平时，其输出管饱和导通，使 F 端为低电平。

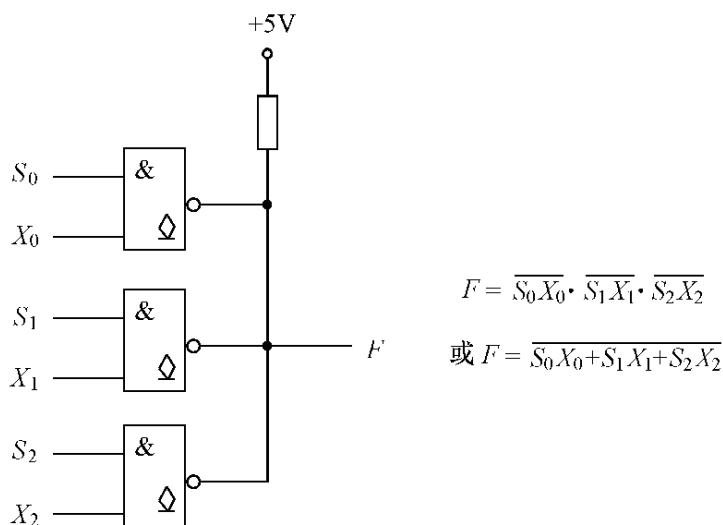


图 5.7 集电极开路总线

这实际上就是一个一位的集电极开路总线，OC 门的一个输入起着选择器的作用（有效控制图 5.2 所示的开关），另一个输入是数据输入 X 。假定在任一时刻最多只有一个选通信号有效，也就是这个信号决定当前的数据源。推广到更一般的情况，不考虑 OC 门个数的限制，总线信号 F 由下面的表达式给出：

$$F = \overline{S_0 X_0 + S_1 X_1 + \dots + S_n X_n}$$

注意，总线上的数据是数据源值的非。

集电极开路总线现在较少使用，因为其速度较慢。参见图 5.8 所示。当输入由低变高时，三极管立即导通，寄生电容（对于长总线导线而言较高）由于三极管内部电阻较小而快速放电。但是，当相反的电平转换发生时，电容通过上拉电阻充电。由于这个电阻一般不能取得太小（该电阻值与输出电平相关，电阻越小，输出高电平越高；电阻越大，输出低电平越低），因此充电的时间相对放电的时间要长。

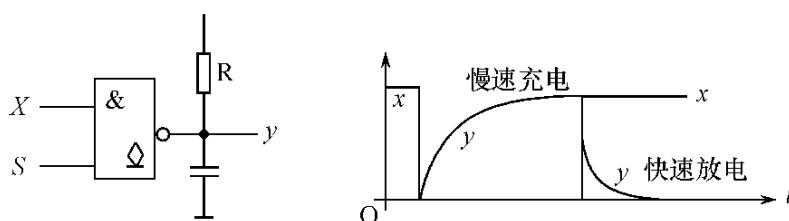


图 5.8 OC 门输出与输入的时间关系图

5.3.2 三态门电路

三态门具有三种输出状态：高电平、低电平和高阻态。图 5.9 展示了三态门电路的一种实现方式。与普通 TTL 与非门相比，它多了一个控制端 e 和控制二极管 VD。

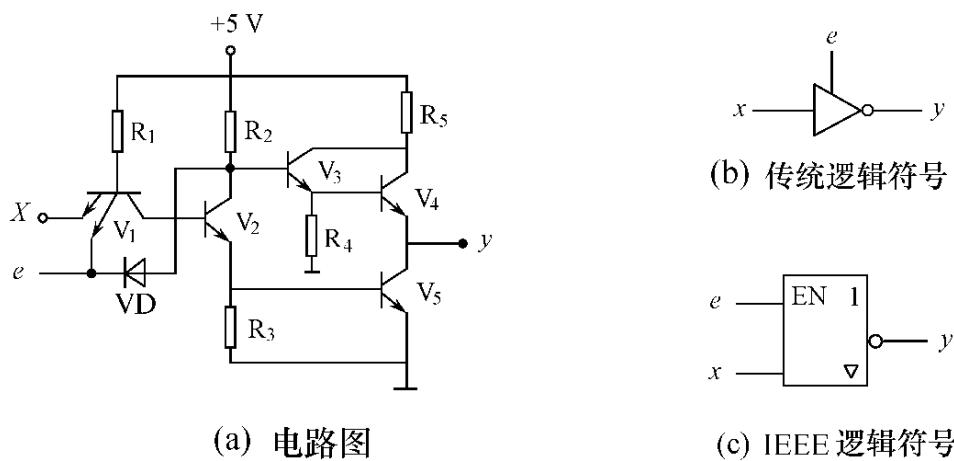


图 5.9 TTL 三态与非门

当控制输入端 e 为高电平时，由于二极管 VD 的隔离作用， e 端对电路状态没有影响，此时三态门的逻辑功能与普通 TTL 与非门完全相同；当控制输入端 e 为低电平时，由于控制端 e 不仅和 V_1 的发射极相接，而且通过二极管 VD 与 V_3 的基极相连，使 V_3 的基极钳位于低电位，于是 V_3 、 V_4 和 V_5 均截止，电路输出端呈现高阻态。

表 5.1 图 5.9 所示三态门真值表

e	x	y
1	0	1
1	1	0
0	0	高阻态
0	1	高阻态

值得注意的是，三态门电路有多种不同的实现方式，其真值表各不相同。使用时须注意控制端是高电平有效还是低电平有效，这可以从三态门逻辑符号控制端是否标有小圆圈来区分。同样，三态门的输出也不一定反向，需从三态门逻辑符号输出端是否标有小圆圈来区分。按照惯例，三态门控制端一般采用低电平有效，输出端不反向。

与 OC 门不同的是，三态门不需外接负载电阻，多个三态门的输出连接在一

起时，在同一时刻绝对不能有多于一个门电路被选中。如果两个门电路被选中，可能其中一个欲使总线为低电平，而另一个则欲使总线为高电平，或者反过来，这将会造成短路并产生大电流。使用三态门不当，不仅会产生错误的操作，甚至有可能会造成电路的永久性损坏，因此需要特别注意。

目前计算机中广泛采用三态门构成数据总线。图 5.10 是使用三态门实现的四位双向总线。

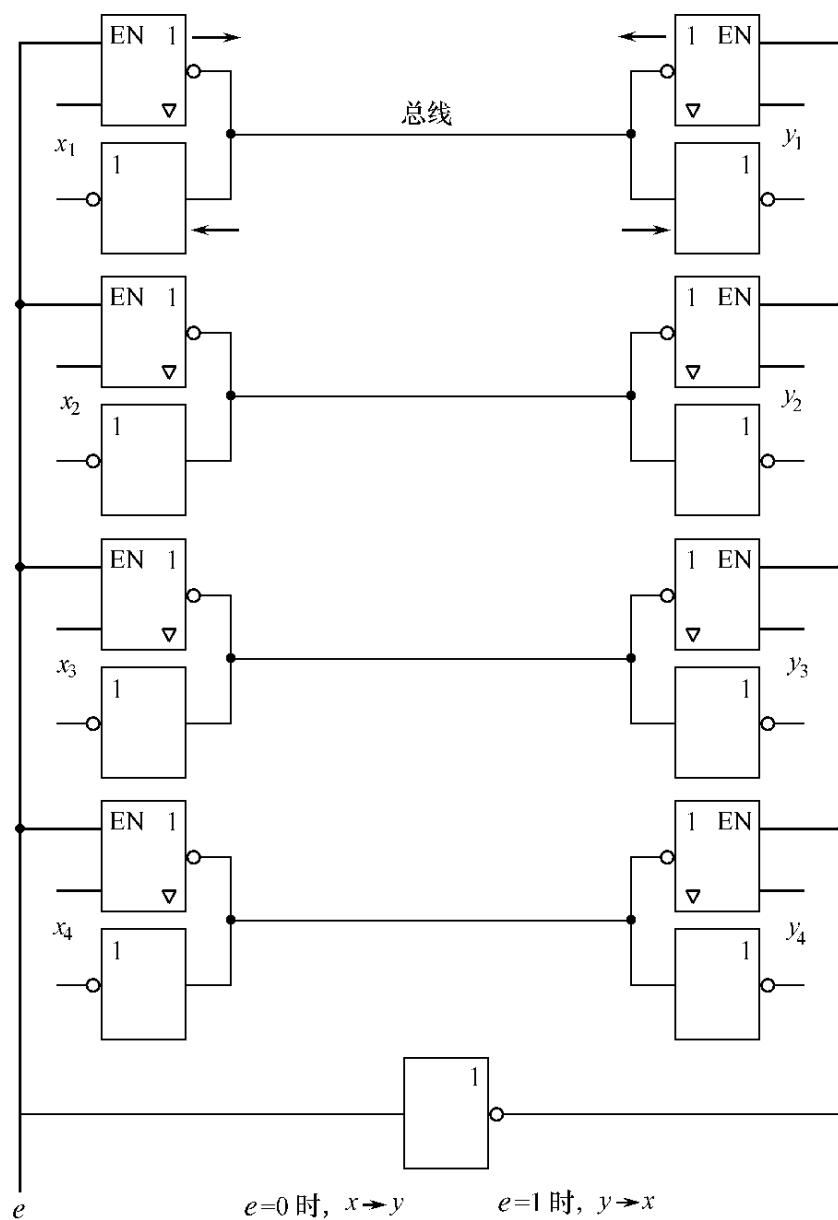


图 5.10 三态门实现的 4 位双向总线

5.4 常用总线简介

微机中总线一般有(芯)片总线、系统总线和外部总线。(芯)片总线是微处

理器芯片内部引出的总线 ,用于芯片一级的互连 ;系统总线是微机中各插件板之间的总线 ,用于插件板一级的互连 ;外部总线则是微机系统之间或微机与其他系统/外部设备之间的总线 ,微机作为一种设备 ,通过该总线和其他设备进行信息与数据交换 ,它用于设备一级的互连。

计算机通信方式可以分为并行通信和串行通信 ,相应的通信总线被称为并行总线和串行总线。并行通信速度快、实时性好 ,但由于占用的口线多 ,不适用于小型化产品 ;而串行通信速率虽低 ,但在数据通信吞吐量不是很大的应用中则显得更加简易、方便、灵活。串行通信一般又可分为异步模式和同步模式。

常用的总线介绍如下。

1. ISA 总线

ISA 总线(Industry Standard Architecture)是在 8 位 PC 总线的基础上扩展而成的 16 位总线。它是 80286 微机系统中使用的总线 ,虽然在 80386 以上 32 位微机系统中也采用 ISA 作为外围总线 ,但目前除在工控机中仍使用 ISA 总线外 ,在现代微机中已很少使用。 ISA 总线的插槽有长短两个插口 ,长插口有 62 个引脚 ,短插口有 36 个引脚。

2. EISA 总线

EISA 总线(Extended Industry Standard Architecture)是 ISA 总线的扩展 ,是与 ISA 总线完全兼容的扩展总线。凡是 ISA 总线具有的信号 ,EISA 总线均予以保留。为了兼容 ISA 总线 ,EISA 总线使用 8 MHz 的时钟速率 ,增加了突发式数据传送方式 ,可以以三倍于 ISA 总线的速率传送数据 ,总线提供的 DMA(直接存储器访问)速度可达 33 Mb/s 。 EISA 总线的输入/输出(I/O)总线和微处理器总线是分离的 ,因此 I/O 总线可保持低时钟速率以支持 ISA ,而微处理器总线则可以高速率运行。

EISA 总线将 ISA 总线的数据线从 16 位扩展为 32 位 ,地址线从 24 位扩展为 32 位 ,具有 198 条信号线 ,分成为 4 组——地址总线和数据总线组、数据传送控制组、总线仲裁信号组及其他功能组。 EISA 总线扩展槽的插脚分为上、下两层 ,上层是原 ISA 总线的接线 ,下层是 EISA 总线新增信号的接线 ,但上、下两层触点交错排列。当 ISA 卡插入时 ,它仅能接触到槽的上层接线 ,而 EISA 卡插入时则能连接到全部接线 ,因而总线能自动识别 ISA 卡和 EISA 卡。

3. VESA 总线

采用总线结构的微机系统 ,各个总线模块均挂接在系统总线上 ,模块间的信息传输均经系统总线进行 ,这就要求系统总线的传输率(带宽)很高。随着图形用户界面(GUI)软件的普及 ,原有的总线标准不能满足系统性能的要求 ,成了系

统性能的瓶颈。解决系统总线拥挤问题的一个有效办法,就是在处理模块中配置专供其 CPU 使用的局部总线,在局部总线上挂有局部存储器和局部 I/O 接口,而系统总线上挂有公共存储器(共享存储器)和公共 I/O 接口。共享存储器主要用于模块间的通信,而大部分数据传输可通过局部总线来完成。

VESA(Video Electronics Standard Association)总线是 1992 年由 60 家附件卡制造商联合推出的一种局部总线,简称为 VL(VESA Local bus)总线。它定义了 32 位数据线,且可通过扩展槽扩展到 64 位,使用 33 MHz 时钟频率,最大传输率达 132 MB/s ~ 246 MB/s,可与 CPU 同步工作,主要是面向 i486 设计的,不适合 Pentium 以上的系统。

4. PCI 总线

PCI(Peripheral Component Interconnect)总线是当前最流行的总线之一,它是由 Intel 公司推出的一种局部总线。它定义了 32 位数据总线,且可扩展为 64 位。PCI 总线主板插槽的体积比原 ISA 总线插槽还小,其功能比 VESA、ISA 有极大的改善,支持突发读写操作,最大传输速率可达 132 MB/s ~ 246 MB/s,可同时支持多组外围设备,且支持即插即用。PCI 局部总线不能兼容 ISA、EISA 总线,但 PCI 总线部件和插件接口相对处理器是独立的,支持目前和将来不同结构的处理器。

PCI 提供三个互相独立的物理地址空间:存储器、I/O 与配置空间。配置空间是 PCI 特有的,所有 PCI 设备必须提供配置空间。PCI 是地址/数据复用总线,每一个 PCI 总线传送由一个地址节拍和一个或多个数据节拍组成。

PCI 总线通过桥接器实现总线间的相互通信,桥接器的主要作用是把一条总线的地址空间映射到另一条总线的地址空间。图 5.11 给出了基于 PCI 总线的微机系统典型结构。

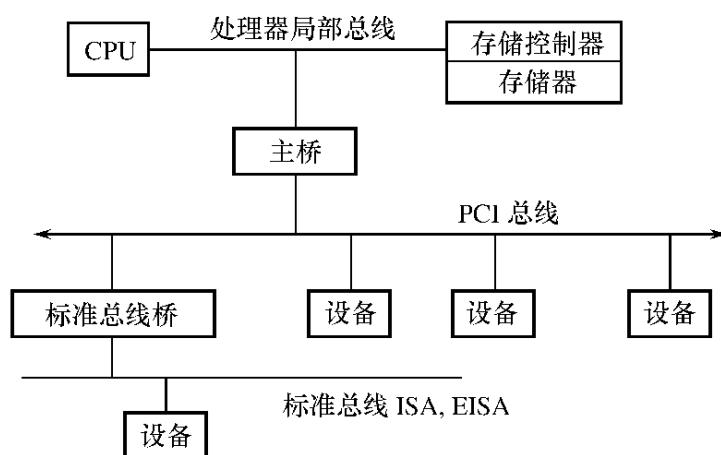


图 5.11 基于 PCI 总线的微机系统典型结构

5. Compact PCI 总线

在计算机系统总线中,还有另一大类为适应工业现场环境而设计的系统总线。Compact PCI 的意思是“坚实的 PCI”,是当今第一个采用无源总线底板结构的 PCI 系统,是 PCI 总线的电气和软件标准加欧式卡的工业组装标准,是当今最新的一种工业计算机标准。Compact PCI 是在原来 PCI 总线基础上改造而来的,它利用 PCI 的优点,提供满足工业环境应用要求的高性能核心系统,同时还考虑充分利用传统的总线产品,如 ISA、STD、VME 或 PC/104 来扩充系统的 I/O 和其他功能。

6. RS - 232 - C 总线

RS - 232 - C 是美国电子工业协会 EIA(Electronic Industry Association)制定的一种串行物理接口标准。RS 是英文“推荐标准”的缩写,232 为标识号,C 表示修改次数。RS - 232 - C 总线标准设有 25 条信号线,包括一个主通道和一个辅助通道,在多数情况下主要使用主通道,对于一般双工通信,仅需几条信号线就可实现,如一条发送线、一条接收线及一条地线。RS - 232 - C 标准规定的数据传输速率为每秒 50、75、100、150、300、600、1 200、2 400、4 800、9 600、19 200 波特。RS - 232 - C 标准规定,驱动器允许有 2 500 pF 的电容负载,通信距离将受此电容限制,例如,采用 150 pF/m 的通信电缆时,最大通信距离为 15 m;若每米电缆的电容量减小,通信距离可以增加。传输距离短的另一原因是 RS - 232 属单端信号传送,存在共地噪声和不能抑制共模干扰等问题,因此一般用于 20 m 以内的通信。

本书 9.2.3 节对 RS - 232 - C 有进一步的介绍。

7. RS - 485 总线

在要求通信距离为几十米到上千米时,广泛采用 RS - 485 串行总线标准。RS - 485 采用平衡发送和差分接收,因此具有抑制共模干扰的能力。加上总线收发器具有高灵敏度,能检测低至 200 mV 的电压,故传输信号能在千米以外得到恢复。RS - 485 采用半双工工作方式,任何时候只能有一点处于发送状态,因此,发送电路须由使能信号加以控制。RS - 485 用于多点互连时非常方便,可以省掉许多信号线。应用 RS - 485 可以联网构成分布式系统,其允许最多并联 32 台驱动器和 32 台接收器。

本书 9.2.3 节对 RS - 485 有进一步的介绍。

8. IEEE - 488 总线

RS - 232 - C 和 RS - 485 是串行总线,而 IEEE - 488 总线是并行总线接口标准。它按照位并行、字节串行双向异步方式传输信号,仪器设备直接并联于总线

上而不需中介单元 ,但总线上最多可连接 15 台设备。最大传输距离为 20 m ,信号传输速度一般为 500 KB/s ,最大传输速度为 1 MB/s。

IEEE - 488 总线用来连接如微机、打印机、数字电压表、数码显示器等设备。

9. IDE 总线

IDE(Integrated Device Electronics)即集成设备电子部件 ,也称为 ATA 接口(AT - Attachment),是 1984 年由 Compaq 开发并由 Western Digital 公司生产的控制器接口。最大特点是把控制器集成到驱动器内。好处是可以消除驱动器和控制器之间的数据丢失问题 ,使数据传输十分可靠。这就可以提高每磁道的扇区数到 30 以上 ,从而增大容量。IDE 采用了 40 线的单组电缆连接 ,在 IDE 的接口中 ,除了对 AT 总线上的信号作必要的控制之外 ,基本上是原封不动地送往硬盘驱动器。现在的微机系统中已不再使用适配卡 ,而把适配电路集成到系统主板上 ,并留有专门的 IDE 连接器插口 ,用于连接磁盘驱动器和光驱。

10. SCSI 总线

SCSI 直译为小型计算机系统专用接口(Small Computer System Interface),是一种连接主机和外围设备的接口 ,支持包括磁盘驱动器、磁带机、光驱、扫描仪在内的多种设备。它由 SCSI 控制器进行数据操作 ,SCSI 控制器相当于一块小型 CPU ,有自己的命令集和缓存。SCSI 连接器分为内置和外置两种 ,内置数据线的外型和 IDE 数据线一样 ,只是针数和规格稍有差别。SCSI 设备在 SCSI 总线上的惟一识别是其 ID ,ID 绝对不允许重复 ,可选范围从 0 到 15 。SCSI 主控制器通常占用 ID 7 。SCSI 主控制器通过总线终结器判断整条总线在何处终结 ,必须在两个物理终端做一个终结信号才能使用 SCSI 总线。SCSI 的优点是适应面广、性能高 ,在一块 SCSI 控制卡上可以同时挂接 15 个设备 ,具有多任务、带宽宽及 CPU 占用率低等特点 ,缺点是价格昂贵 ,安装复杂。

11. USB 总线

通用串行总线 USB(Universal Serial Bus)是由 Intel、Compaq、Digital、IBM、Microsoft、NEC、Northern Telecom 等 7 家世界著名的计算机和通信公司于 1994 年 11 月联合开发的一种新型串行接口总线标准 ,1996 年 1 月颁布了 USB 1.0 版本规范。它基于通用连接技术 ,实现外设的简单快速连接 ,达到方便用户、降低成本、扩展 PC 连接外设范围的目的。USB 接口的主要特点是 :即插即用 ,可热插拔。USB 连接器将各种各样的外设 I/O 端口合而为一 ,使之可热插拔 ,具有自动配置能力 ,用户只要简单地将外设插入到 PC 以外的总线中 ,PC 就能自动识别和配置 USB 设备。而且带宽更大 ,增加外设时无需在 PC 内添加接口卡 ,多个 USB 集线器可相互传送数据 ,使 PC 可以用全新的方式控制外设。它可以为外设提

供电源 ,而不像普通的使用串、并口的设备需要单独的供电系统。另外 ,快速是 USB 技术的突出特点之一 ,USB 的最高传输率可达 12 Mb/s ,比串口快 100 倍 ,比并口快近 10 倍 ,而且 USB 还能支持多媒体。

USB 采用“级联”方式连接各个外部设备 ,每个 USB 设备用一个 USB 插头连接到前一个外设的 USB 插座上 ,而其本身又提供一个 USB 插座供下一个 USB 外设连接用。连接多达 127 个外设 ,两个外设间的线缆长度可达 5 m。USB 总线标准由 1.1 版升级到 2.0 版后 ,传输率由 12 Mb/s 增加到了 240 Mb/s ,更换介质后连接距离由原来的 5 m 增加到近百米。USB 总线结构简单 ,其总线电缆仅有 4 根导线 ,由一对标准尺寸的双绞信号线和一对标准尺寸的电源线构成。

USB 是一种简单实用的计算机外部设备接口标准 ,目前大多数主板均有提供。

12. IEEE – 1394 总线

IEEE – 1394 是 Apple 在 1986 年开始构思的 ,当时的目的是简化其计算机的连线 ,并且为实时数字数据提供一个高速界面。Apple 最初称之为 FireWire ,这个名字至今仍然经常和 IEEE 1394 一起使用。

IEEE – 1394 与 USB 类似 ,也是一种高效的串行接口标准。它采用“级联”方式连接各个外部设备 ,可以在一个端口上连接最多 63 个设备 ,设备间采用树形或菊花链结构。设备间电缆的最大长度是 4.5 m ,采用树形结构时可达 16 层 ,从主机到最末端外设总长可达 72 m。IEEE – 1394 的连接电缆中共有 6 条芯线 ,2 条为电源线 ,4 条被包装成两对双绞线 ,用来传输信号。电源的电压范围是 8 ~ 40 V 直流电压 ,最大电流 1.5 A。

IEEE – 1394 标准定义了两种总线数据传输模式 ,即 :Backplane 模式和 Cable 模式。其中 Backplane 模式支持 12.5、25、50 Mb/s 的传输速率 ;Cable 模式支持 100、200、400 Mb/s 的速率。目前正在开发 1 Gb/s 的版本。在 400 Mb/s 时 ,只要利用 50% 的带宽就可以支持不经压缩的高质量数字化视频信息流。IEEE – 1394 可同时提供同步(Synchronous)和异步(Asynchronous)数据传输方式 ,采用点对点结构 ,任何两个支持 IEEE 1394 的设备可以直接连接 ,不需要通过电脑控制。

IEEE – 1394 和 USB 的相似之处主要表现在 :

- (1)都可以提供即插即用热插拔的功能 ,安装十分简单。
- (2)都提供统一的通用接口 ,都可提供总线供电方式。
- (3)都采用了串接方式 ,可以连接多台设备。

它们之间的主要差别在于 :

(1) IEEE – 1394 的传输速度很高 ,可达 100 ~ 400 Mb/s ,目前 1 Gb/s 的协定正在制定。因此 ,它可连接高速设备如 DVD 播放机、数码相机、硬盘等 ;而 USB

的 12 Mb/s 因传输速度限制 ,只能连接低速设备 ,如键盘、麦克风、软驱、电话等。

(2) IEEE – 1394 的拓扑结构中 ,不需要集线器(HUB)就可连接 63 台设备 ,并且可以由网桥再将这些独立的子网连接起来。 IEEE 1394 并不强制要用电脑控制这些设备。而在 USB 的拓扑结构中 ,必须通过 HUB 来实现多重连接 ,每个 HUB 有 7 个连接头 整个 USB 网络中最多可连接 127 台机器 ,而且一定要有电脑的存在。

(3) IEEE – 1394 的拓扑结构在其外部设备增减时 ,会自动重设网络 ,其中包括网络短暂的等待状态 ;而 USB 以 HUB 来判明其连接设备的增减 ,因此可以减少 USB 网络动态重设的状况。

习 题

1. 什么是总线 ? 总线传输有何特点 ?
2. 使用总线的好处是什么 ?
3. 简述总线的组成。
4. 什么是总线主设备和总线从设备 ?
5. 什么是总线控制器 ? 它的主要功能是什么 ?
6. 简述集电极开路总线和三态门总线的电路工作原理。
7. 为什么集电极开路总线的速度较慢 ?
8. 为什么使用三态门不当 ,可能会造成电路的永久性损坏 ?

第六章 存 储 器

6.1 概 述

目前绝大多数计算机硬件系统采用的仍然是冯·诺依曼体系结构,计算机自动工作是按照存储程序控制原理实现的。计算机通过执行保存在主存储器中的程序(指令的有序集合),完成特定任务的自动处理。在工作过程中,计算机每执行一条指令至少访问一次主存储器,以取出指令;对于某些数据操作指令,还需要从主存储器中存取操作数,这些指令需要多次访问主存储器,所以主存储器的速度对计算机的速度有很大的影响。

早期的冯·诺依曼计算机是以运算器为中心的,系统内各部件间的数据传送都要经过运算器。随着计算机解题能力的增强,应用领域的扩大,工作效率的提高,用户界面的友好灵活以及外部设备的发展,内存与外部设备间的信息交换日益频繁,如果外设与存储器打交道都要通过CPU来实现,将大大降低CPU的工作效率。为了适应这一情况,出现了外设与存储器的直接存取方式(DMA),形成了以存储器为中心的系统结构。共享主存的多处理机系统的出现,更进一步加强了存储器作为计算机系统的中心地位。这时,存储器除了要向一台或多台高速运行的CPU提供所需的指令和数据外,还要同并行工作的外存及其他外设和终端等设备交换信息。存储系统的特性,已经成为影响整个系统最大吞吐量的决定性因素。

由于计算机应用对存储器的容量和速度的要求几乎是无止境的,理想的存储系统应当具有充足的容量和与CPU相匹配的速度。但是现有主存储器都不能满足这样的要求。从整个计算机技术发展来看,一方面主存储器的工作速度总是落后于CPU,通常至少慢一个数量级,且CPU芯片以每年60%的速度增长,而存储芯片仅以每年7%的速度递增。另一方面主存储器的容量总是落后于软件的需求。容量、速度、成本折衷的结果,迫使存储系统不得不从经济的角度考虑采用分层结构,而存储器访问的局部性(包括时间上的局部性和空间上的局部性)保证了分层结构在技术上的可用性。

6.1.1 存储器的分类

计算机问世 50 多年来 ,存储技术发展迅速 ,不断有新的存储介质和存储元器件出现。从初期的汞、镍延迟线 ,到磁芯存储器 ,直到目前广泛使用的超大规模集成电路存储芯片等。存储器种类越来越多 ,功能越来越强 ,容量也大幅度提高。由于存储器的存取方式多种多样 ,速度差异悬殊 ,保存信息的方式和手段千差万别 ,在计算机中所起的作用各不相同 ,所以 ,从不同的角度对存储器可作不同的分类。

1. 按存储介质分类

存储介质是指具有两种稳定状态 ,可用来记录“ 0 ”、“ 1 ”两种代码 ,并能方便地检测和转化两种状态的物质和元器件。存储介质主要有半导体器件、磁性材料和光盘等。

(1) 半导体存储器

由半导体器件组成的存储元件叫半导体存储器。早期的半导体存储器采用典型的晶体管触发器作为存储位元 ,加上选择、读写等电路构成存储器。现代半导体存储器都用超大规模集成电路工艺制成存储芯片 ,每个芯片包含相对数量的存储位元 ,再由若干芯片构成存储器。其优点是体积小、功耗低、存取时间短 ;其缺点是当电源消失时 ,所存信息也随即丢失(半导体只读存储器例外),它是一种易失性存储器 ,也叫做挥发性存储器。

根据集成电路中开关元件的不同 ,半导体存储器可分为晶体管双极型和场效应管 MOS 型两种。双极型有射极耦合逻辑 ECL 、晶体管逻辑 TTL 和集成注入逻辑 I2L 三种 ;MOS 型有 PMOS 、 NMOS 和 CMOS 三种。

根据保持信息时间的不同 ,半导体存储器又可分为静态存储器(以触发器原理寄存信息)和动态存储器(以电容充放电原理寄存信息)。加电后 ,只要电源正常供电 ,静态存储器可长时间保持信息 ,而动态存储器只能保持几个或十几个毫秒 ,要长时间保持信息 ,必须在规定的时间内不断动态地刷新信息。双极型存储器都是静态存储器 ,MOS 型则既有静态存储器又有动态存储器。

双极型存储器比 MOS 存储器速度快 ,通常至少高一个数量级 ,但相对来说功耗大 ,集成度低 ,适合用作快速小容量存储器 ,如 Cache 。 MOS 存储器具有集成度高、功耗小、制造工艺简单、成本较低的优点 ,被广泛用作主存储器。

(2) 磁表面存储器

磁表面存储器是在金属或塑料基体的表面涂上一层磁性材料作为记录介质 ,用磁层的两种剩磁状态记录信息“ 0 ”和“ 1 ”。由于剩磁状态不会轻易消失 ,故

这类存储器具有非易失性的特点。工作时,磁层随载磁体高速运转,用软磁材料制作的磁头进行读写操作。机械运动的读写方式,使磁表面存储器的读写速度较慢,通常为毫秒级,远低于半导体存储器。

按载磁体形状的不同,磁表面存储器可分为磁盘存储器、磁卡存储器、磁带存储器和磁鼓存储器4种。目前广泛采用的是磁盘和磁带存储器。磁表面存储器因价格低廉、存储容量很大,被广泛用作辅助存储器。

(3) 光盘存储器

与磁表面存储器类似,光盘存储器也是将用于记录的薄层(磁光材料)涂敷在基体上构成记录介质,通过激光进行信息的读写,它具有非易失性的特点。与硬盘相比,光盘的存储速度虽然慢一个数量级,但由于具有记录密度高、耐用性好、可靠性高和互换性强等优点,已经成为一种重要的辅助存储器。

2. 按存取方式分类

存储器按存取方式可分为随机存取存储器、只读存储器、顺序存取存储器和直接存取存储器4类。

(1) 随机存取存储器

随机存取存储器(Random Access Memory)是一种可读写存储器,它以存储单元为单位组织信息和提供访问,其特点是存储器的任何一个存储单元都可以随机存取,而且存取时间与存储单元的物理位置无关。计算机系统中的主存储器都采用这种存储器,通常简称为RAM。由于随机存取存储器目前广泛使用半导体存储器,因此, RAM又分为静态RAM和动态RAM。

(2) 只读存储器

只读存储器(Read Only Memory)除正常工作时只能随机读取信息而不能随机写入信息外,其他特性与RAM相同。它的信息是在正常工作前事先写入的,在工作过程中不会发生改变。因此,通常用它来存放系统引导程序、管理监控程序、常数和汉字库等。它与RAM共同构成计算机的主存储器,享有统一的地址域空间。只读存储器通常简称为ROM。

目前广泛使用的是半导体大规模集成电路只读存储器。随着半导体技术的发展和用户需求的变化,只读存储器出现了很多不同的种类。根据写入方式的不同,可分为掩膜型只读存储器MROM、可编程只读存储器PROM、可擦除可编程只读存储器EPROM以及电可擦除可编程只读存储器EEPROM。MROM是在制造时使用掩膜工艺将信息写入存储器,一旦制建成后,存储的信息无法改变;PROM是一次可改写ROM,由生产厂家在制造时写入全“0”或全“1”,用户在使用前按需要做一次且仅能做一次改写;EPROM是多次可改写ROM,由生产厂家在制造时写入全“0”或全“1”,用户可根据需要做多次擦除和改写;EEPROM进一步

改进了 EPROM 需离线擦除全部信息且擦除时间较长的不足 ,可实现联机在线部分信息擦除。

进入 20 世纪 80 年代 ,在 EPROM 和 EEPROM 工艺基础上产生了一种新型的、具有更高性能价格比、更可靠的可擦写非易失性存储器——闪速存储器 (Flash Memory),又称快擦型存储器。它既有 EPROM 价格便宜、集成度高的优点 ,又有 EEPROM 电可擦洗重写的特性。其位元尺寸与 EPROM 相当 ,比 EEPROM 小 10 倍 ;用电擦除整个存储矩阵(片擦除)或部分存储矩阵(块擦除)的时间 ,与 EEPROM 擦除一个地址的时间相当 ,速度很快。它是目前惟一具有大容量、非易失性、低价格、可在线改写和较高速度几个特性共存的存储器。与动态 RAM 相比 ,它在可擦除次数和存取速度两方面还有一定距离 ,无法取代动态 RAM。但它是理想的文件存储介质 ,当它代替硬盘时 ,除具有存取快、体积小、重量轻、功耗低和不易损坏的优点外 ,还具有就地执行的能力(即存放操作系统、实用程序和数据文件时 ,开机后无需像硬盘那样 ,先要将它们调入主存再执行 ,而是立即执行)。闪速存储器在便携式计算机、工控及单片机系统中得到越来越广泛的应用。

(3) 顺序存取存储器

顺序存取存储器中的信息按文件组织 ,一个文件包含若干个数据块 ,一个数据块又包含若干个字节 ,它们顺序地记录在存储介质上 ,存取时以块为单位 ,只能顺序查找块号 ,找到后即成块顺序读取 ,存取时间与信息所在物理位置有较大的关系。

这类存储器容量大、成本低 ,但速度慢 ,常用作后援辅存。如磁带存储器。

(4) 直接存取存储器

直接存取存储器的信息组织形式与顺序存取存储器相同。它是介于随机存取和顺序存取之间的一种存储器。它对信息的存取分两步进行 ,首先随机寻址信息块(存储器上的一个区域) ,然后对这一部分进行顺序存取。如磁盘、光盘。

3. 按在计算机中的作用分类

存储器按在计算机中的作用可分为高速暂存存储器、高速缓冲存储器、主存储器和辅助存储器。

(1) 高速暂存存储器

由 CPU 内部的寄存器组成 ,其速度与 CPU 相匹配 ,用来暂时存放即刻要执行的指令、马上要使用的数据和处理得到的结果。它是 CPU 不可分割的组成部分。如先行寄存器、通用寄存器组等。

这种类型的存储器容量一般不超过 2 KB。

(2) 高速缓冲存储器

位于 CPU 和主存储器之间的小容量高速存储器 ,称为 Cache。它存放当前正在执行程序的部分程序段和数据 ,以便向 CPU 快速提供即刻要执行的指令或马上要使用的数据 ,允许 CPU 执行操作而不必总是从主存储器中检索数据 ,避免 CPU 与 I/O 争抢访问主存。现代高性能微处理器芯片都把一级 Cache 纳入其中。

目前 Cache 一般采用双极型半导体存储器 ,也有的采用 CMOS 半导体存储器 ,速度可与 CPU 匹配。其存储容量一般在几 KB 到几百 KB 之间。

(3) 主存储器

主存储器用于存放当前处于活动状态的程序和有关数据 ,包括操作系统的常驻部分和目前正在运行的程序和要处理的数据 ,CPU 通过指令可直接访问它。

从物理结构上看 ,主存储器由一个一个的存储单元组成 ,每个存储单元包含若干存储位元。一个存储单元可存放一个机器字(Word)或一个字节(Byte)。主存储器的存储单元按某种顺序编号 ,此编号称为单元地址。只要给定一个存储单元地址 ,就可以通过地址译码器译码 ,找到对应的存储单元 ,从而对该单元进行读写操作。

目前 ,主存储器大多为 MOS 半导体存储器 ,存取速度较 CPU 慢一个数量级。容量比 Cache 大得多 ,一般在几百 KB(2^{10} B)到几百 MB(2^{20} B)之间。

(4) 辅助存储器

属于外围设备的范畴 ,因此也称外存。外存与内存的最大区别就在于它不能由 CPU 的指令直接访问 ,而必须通过专门的程序或通道把所需的信息与主存进行成批交换 ,调入主存后才能使用。外存用来存放需要联机保存但暂不使用的程序和数据。

辅存的存取速度比主存至少慢两个数量级 ,但容量相当大 ,通常在几十 MB 到几百 GB(2^{30} B)之间 ,有的甚至达到几 TB(2^{40} B)。

6.1.2 存储器的层次结构

从整个计算机技术的发展来看 ,存在着这样一个明显的事 实 ,即主存的速度总是落后于 CPU 的需要 ,主存的容量总是落后于软件的需求。因此 ,仅靠改进存储技术来提高存储器性能的途径 ,不能马上满足计算机系统对存储器提出的速度快、容量大、成本低的要求 ,还必须建立合理的存储结构 ,即采用多种类型的存储器组成存储系统。多种类型存储器的采用 ,形成了存储层次的概念。典型的存储层次结构如图 6.1 所示。

存储器的层次结构主要体现在缓存—主存和主存—辅存两个存储层次上。从 CPU 角度看 ,缓存—主存这一层次的速度接近于缓存 ,高于主存 ;其容量和价位却接近于主存。从整体角度看 ,主存—辅存这一层次的速度接近于主存 ,其容

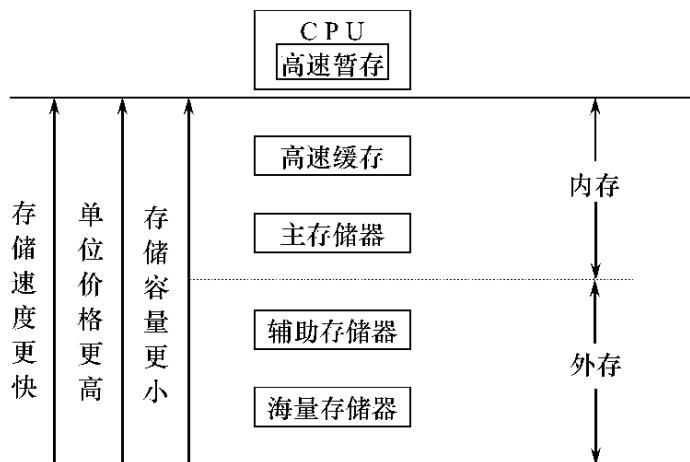


图 6.1 典型的存储层次结构

量和价位却接近于辅存。这就解决了速度、容量和成本这三者之间的矛盾。现代的计算机系统几乎都具有这两个存储层次 构成缓存、主存、辅存三级存储系统。

在主存—辅存这一层次的不断发展中 ,以存储器访问的局部性为基础 ,逐渐形成了虚拟存储系统。它的基本思想是通过某种策略使应用程序员可以按程序的需要使用逻辑地址编程 ,其地址空间为虚拟的主存地址空间 ,所编程序和要处理的数据先保存在辅存中 程序运行时 ,该策略负责将当前需要运行的部分从辅存中调入主存 ,暂不运行的部分仍留在辅存中 ;随着程序运行的需要 ,该策略按一定的替换算法将主存中暂不运行的部分调往辅存 ,将辅存中要运行的部分调入主存 ;CPU 执行程序时 ,每次访存操作都必须由特定机构判别该地址内容是否在主存中 ,若在 ,将逻辑地址转换成物理地址 ,并据此访存 ;若不在 ,则需将辅存中的相应部分调入主存。上述过程 ,应用程序员是看不见的 ,即是透明的 ,他的感觉就好像拥有一个容量极大的主存一样。

6.2 半导体 RAM 位元电路

现代计算机的主存储器都由半导体集成电路构成 ,其最小逻辑单位是存储位元 ,它存储一位二进制信息。半导体 RAM 分为静态 RAM 和动态 RAM。

6.2.1 静态 RAM 位元电路

静态 RAM 又简称 SRAM。SRAM 电子记忆元件有双极型和 MOS 型两种 ,现代的静态存储器大多采用 CMOS (Complementary MOS) 技术。

MOS 开关元件是一种金属(Metal) 氧化层(Oxide)和半导体(Semiconductor)

组成的场效应管,它的符号如图 6.2(a)所示,当 W 为高电位时导通,即 R 点与 V_{CC} 同电位,当 W 为低电位时截止。由于 W 上所加的电位信号是脉冲的,脉冲去掉以后便处于不确定状态。为了能稳定地记忆 W 上曾加过高电压,不能仅使用单管的 MOS 开关,必须使用具有双稳态的触发器。常用的是六管 MOS 记忆电路,如图 6.2(b)所示。

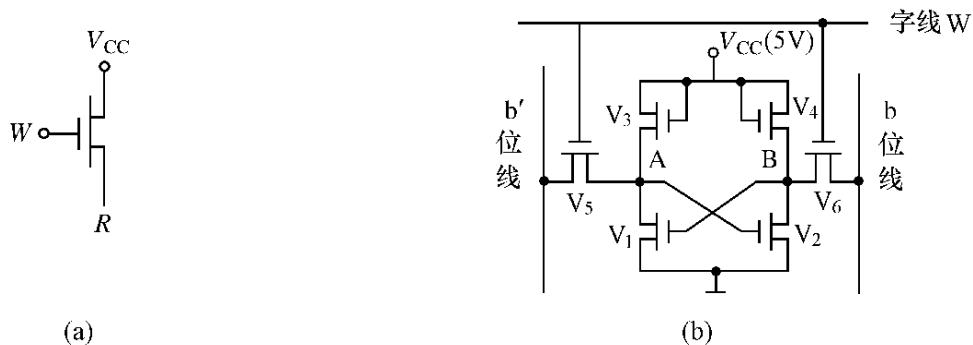


图 6.2 MOS 开关管及其实现的存储位元电路

六管静态 MOS 存储位元电路由 T_1 、 T_2 管组成双稳态的触发器, T_3 、 T_4 作为阻抗, T_5 、 T_6 作为记忆单元的选中开关(读/写控制门)。当记忆单元未被选中,字线保持低电平, T_5 、 T_6 管截止,触发器与位线隔开,原来保存的状态不变,即具有保持功能。字线上加上高电平时, T_5 、 T_6 管导通,该记忆单元被选中,可进行读/写操作。当写入时,字线选中(加正脉冲),读/写控制门打开,如果要写“1”,位线 b' 加低电平,位线 b 加高电平,此时,不管触发器原来处于何种状态,都是 T_1 导通、 T_2 截止,成为“1”状态;如果要写“0”,位线 b' 加高电平,位线 b 加低电平,此时,不管触发器原来处于何种状态,都是 T_1 截止、 T_2 导通,成为“0”状态。这种状态即使在字线上的写脉冲消失后也不会改变,因为 T_1 、 T_2 反向耦合构成双稳态的触发器, T_5 、 T_6 又被关闭,触发器保持刚写入的信息状态不变。当读出时,字线选中(加正脉冲), T_5 、 T_6 导通,位线 b' 和 b 分别与 A 点和 B 点相通,触发器所存的信息(A 点和 B 点的电位)就可以传送到位线上,即被读出;当字线驱动脉冲消失后, T_5 、 T_6 截止,触发器仍保持原有状态,即读出过程不破坏触发器状态。

MOS 静态存储位元电路具有非破坏性读出的特点,且抗干扰能力强、可靠性高。但电路所用管子数目较多,功耗和占用硅片面积大,一般集成度不高。

6.2.2 动态 RAM 位元电路

存储器是最规则和最密集的元件。由于没有办法用更少的晶体管来实现一个触发器,为了进一步提高它们的集成度,必须寻找另一种存储数据的介质。显而易见,电路中电容所带的电荷是一个候选者,即电容上存有足够的电荷表示

“1”,电容上无电荷表示“0”。图6.3所示是一个单管动态MOS存储位元电路。

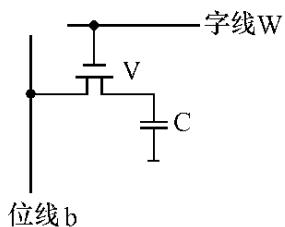


图6.3 单管动态MOS存储位元电路

当写入时,字线选中(加正脉冲), T 导通。如果要写“1”,位线b上加高电平,对电容C充电,使C充满正电荷,即写入了“1”;如果要写“0”,位线b上加低电平,电容C放电,使C上的电荷几乎放光,即写入了“0”。

当读出时,字线选中(加正脉冲), T 导通。如果电容C上有电荷,电容C通过T向位线b上放电,产生电流,可视为读出“1”;如果电容C上无电荷,则位线b上无电流,可视为读出“0”。读操作结束后,电容C上的电荷泄放完毕,故是破坏性读出,必须马上恢复,否则信息会丢失。

其实,即使读出不是破坏性的,用电容保存信息也必须考虑恢复的问题。因为所有电容都会漏电和缓慢地放电,电容上的电荷不可能长久保存,需周期性地对电容进行充电,以补充泄漏的电荷。这个问题可以通过重复读写的方式得以解决,即存储器周期性地读出存储的数据然后重新写回去,从而使电容器得到充电。这一过程被称为再生或刷新,其刷新周期一般为2 ms。

6.3 主存储器结构与工作原理

主存储器的结构如图6.4所示,主要由存储体、地址译码器及驱动器、读/写电路和时序控制电路等组成。

在实际电路中,驱动器、译码器和读/写电路均制作在存储芯片内,而地址寄存器MAR和数据寄存器MDR制作在CPU芯片内。存储芯片和CPU芯片通过总线连接,如图6.5所示。

当要从存储器中读出某一信息时,CPU先将存放该信息的存储器地址送入MAR,然后发读出命令;主存接到该命令后,经地址译码,将相应存储单元的内容送到数据总线上便完成操作;至于将该信息从MDR送到何处,由CPU决定。若要向存储器写入信息,由CPU将信息和存放信息的地址分别送入MDR和MAR,然后发写入命令;主存接到该命令后,经地址译码找到相应存储单元,将数据总

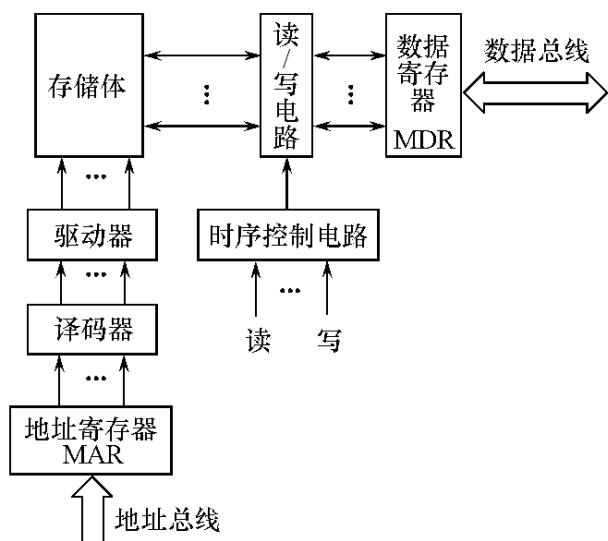


图 6.4 主存储器基本组成

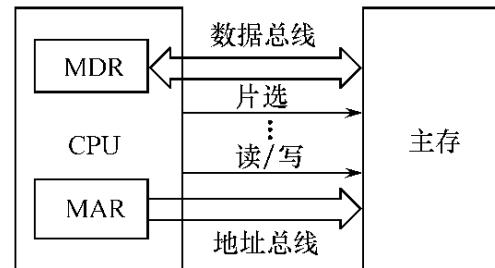


图 6.5 主存与 CPU 的连接

线上的内容写入即可。

1. 存储体的组织与地址译码

存储体可以将存储位元组织成二维存储矩阵的形式,也可以组织成三维存储矩阵的形式。对应这两种组织形式的地址译码方法分别是线选法和重合法。下面以 2^k 个存储单元、每个单元存储 n 位信息为例,说明这两种组织形式和译码方法。

(1) 线选法

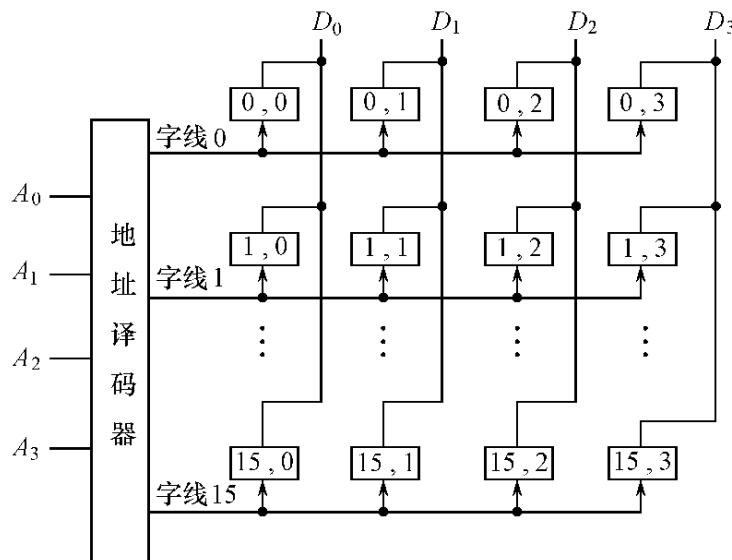
线选法采用的是二维存储矩阵一维地址译码选址技术。它将 k 位地址译码,形成 2^k 根地址线,每根线对应一个存储单元,并仅与该存储单元的 n 个存储位元的字线(参见图 6.2(b)或图 6.3)相连。某条地址线被选中,则选中且仅选中对应存储单元的所有位元。所以称这些地址线为字线,这是二维存储矩阵的一维。若要对存储单元的 n 位信息进行读写,还需要 n 根数据线,称这些线为位线,这是二维存储矩阵的另一维。

$k = n = 4$ 时的线选法二维存储矩阵如图 6.6 所示。图中小矩形表示存储位元,其中的第一个数据指明该存储位元所在存储单元的地址,第二个数据指明该存储位元是存储单元的第几位。

当存储容量很大时,地址线太多。因此,线选法只适合在小容量存储器中使用。

(2) 重合法

重合法采用三维存储矩阵、二维地址译码选址、一维读写。其构成方法是将 k 位地址分成接近相等的两组,分别译码,产生一组行地址线 $X(X_0, X_1, \dots, X_i)$

图 6.6 线选法 16×4 位二维存储矩阵示意图

和一组列地址线 $Y(Y_0, Y_1, \dots, Y_j)$, X 作为一维, Y 作为另一维, 在 X 和 Y 的每一相交处放一存储位元构成的平面, 叫做位面, 它包含所有 2^k 个存储单元的同一位, 所有存储位元用一根位线连接在一起, X 和 Y 的“与”唯一确定一个位面上被选中的存储位元。 n 个这样的位面构成一个 2^k 个存储单元 $\times n$ 位的三维存储矩阵。

$k = n = 4$ 时的重合法三维存储矩阵如图 6.7 所示。图中小矩形表示存储位元, 其中的第一个数据指明该存储位元所在存储单元的地址, 第二个数据指明该存储位元是存储单元的第几位。

重合法实质上是把部分译码功能转移到存储矩阵内部, 从而使外部的译码线路大大简化, 特别是容量越大, 效果越明显。具体实现时, 没有必要像图 6.7 那样让每个存储位元具有“与”运算功能, 可采用图 6.8 的连接方式。

重合法译码方式由于线路简单而得到广泛的应用。

2. 主存容量的扩展

存储器的最小逻辑单位是存储位元, 其次是由若干存储位元组成的存储单元, 再大一点的逻辑单位是存储芯片, 它由一定数量(一般为 2 的整数次幂)的存储单元和相应的外围电路构成, 如图 6.9 所示。其中, 片选引脚用于连接本存储芯片是否被选中的控制信号, 读/写控制引脚用于指明当前对存储芯片的访问是读操作还是写操作; 为了减少引脚数量, 读写操作共用数据引脚, 即数据线路是双向复用的, 因此须通过三态门进行连接, 这就要增加一个用于控制三态门的信号——输出使能引脚。

由于单片存储芯片的容量总是有限的, 很难满足实际的需要, 因此主存储器

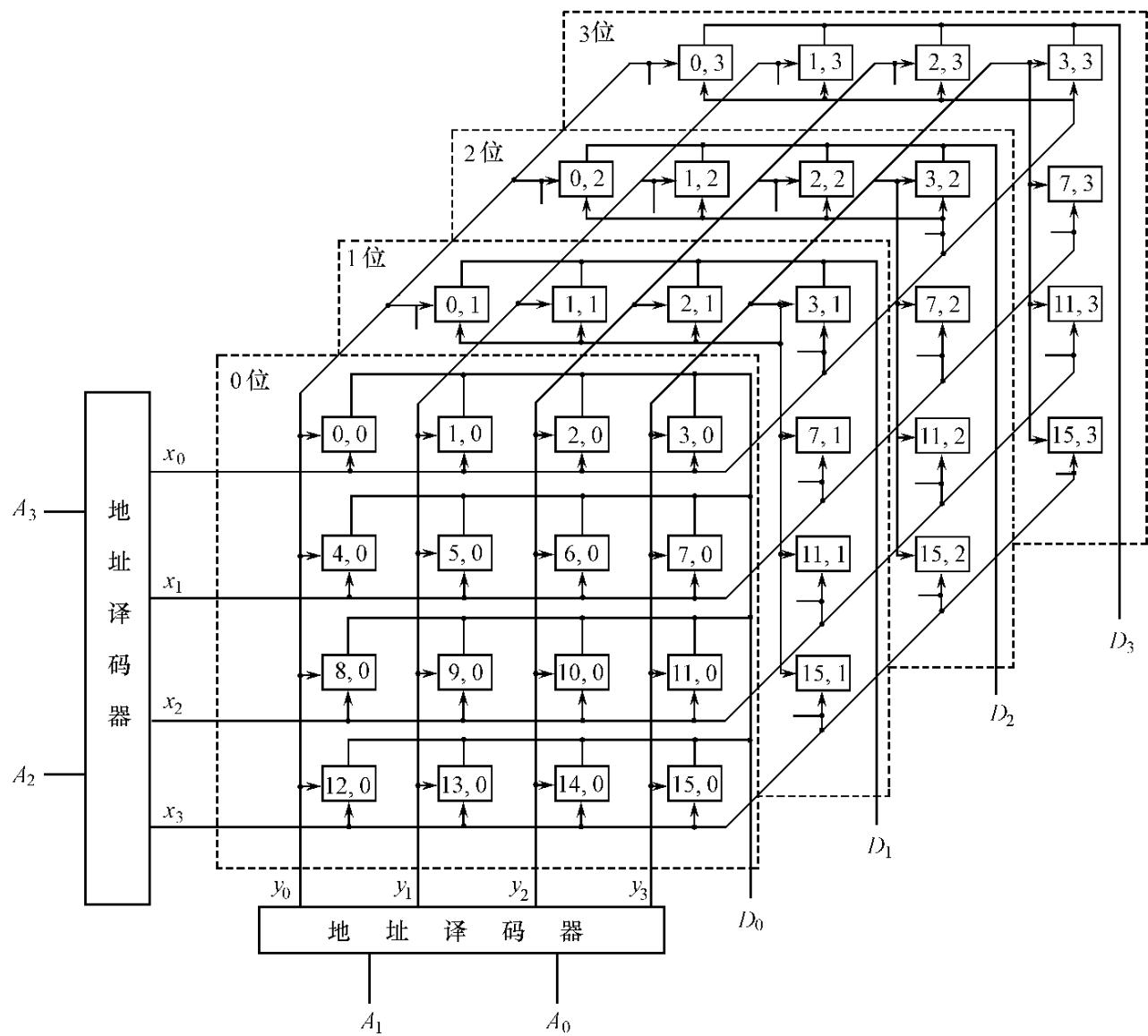
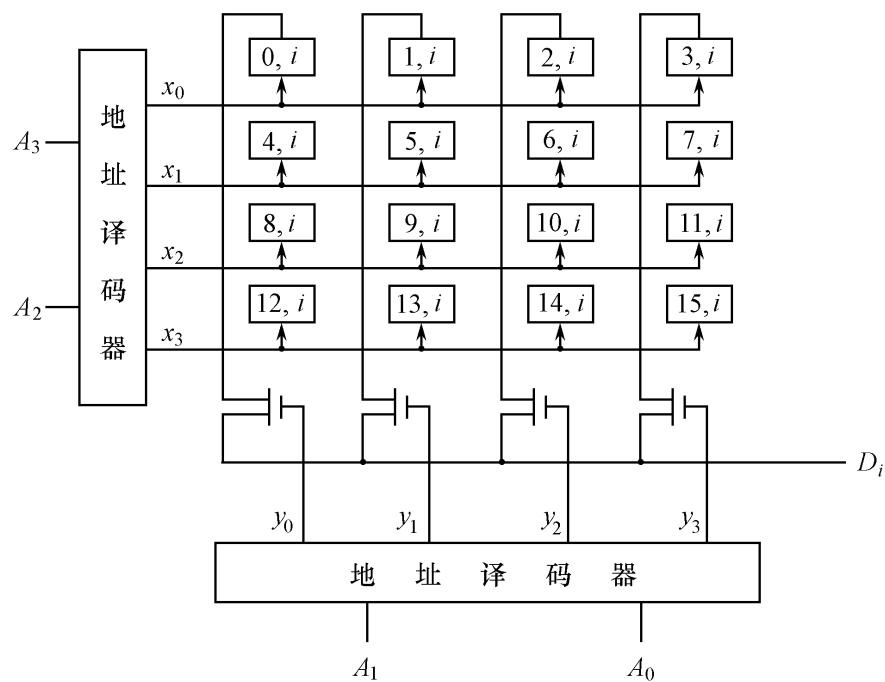
图 6.7 重合法 16 字 \times 4 位三维存储矩阵示意图

图 6.8 重合法一个位面的连接示意图

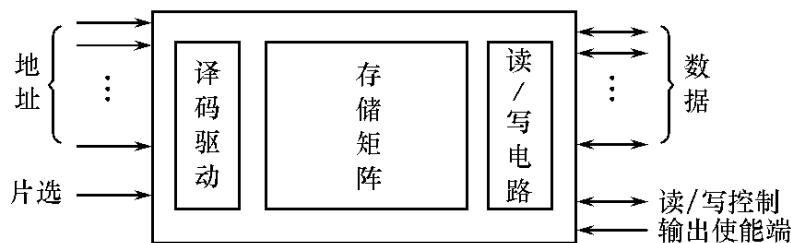


图 6.9 存储芯片的基本结构

通常都是由若干存储芯片连接在一起扩展而成。存储容量的扩展常用的有位扩展、字扩展和字位扩展三种方式。

位扩展方式是每个字中的各位都在不同的芯片上，扩展芯片时，只增加存储器的字长而不改变存储器字数的扩展方式。如用 2 块 $1K \times 4$ 位的芯片扩展成 $1K \times 8$ 位的存储器。字扩展方式是每个字的各位均在同一个芯片上，扩展时字长不变、字数增加的扩展方式。如用 2 块 $1K \times 4$ 位的芯片扩展成 $2K \times 4$ 位的存储器。字位扩展指的是既增加存储器的字长又增加存储器字数的扩展方式。大多数存储器都是在选定存储芯片的基础上通过字和位同时扩展而成的。图 6.10 给出了一个用 4 块芯片组成的具有双倍地址范围和双倍字宽的存储器。图中上下两部分进行的是字扩展，左右两部分进行的是位扩展。

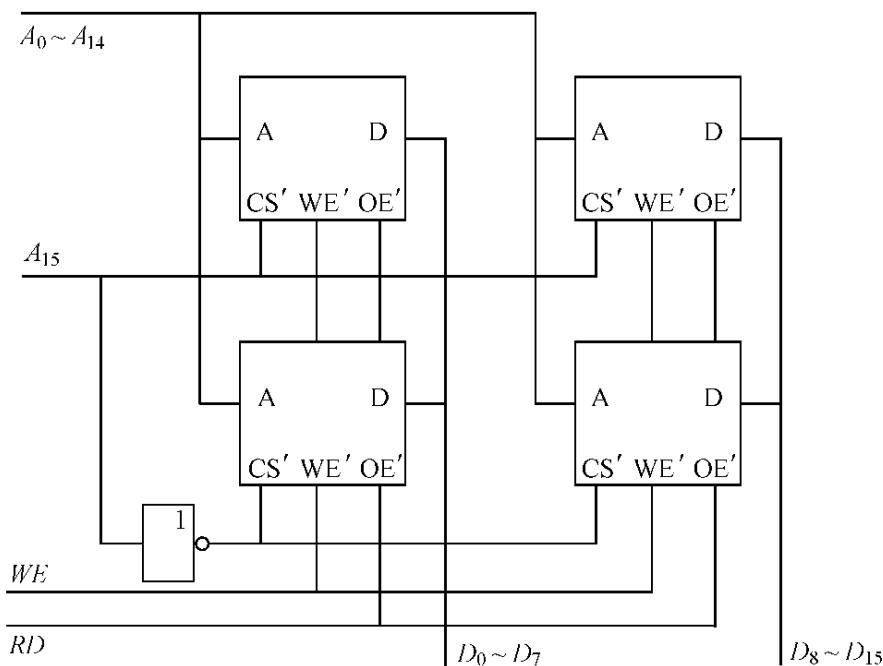


图 6.10 字位扩展的存储器示意

3. 工作时序

存储器的时间特性是衡量其好坏的一个重要指标，它有两个关键的特性：

存储周期和存取时间。前者指的是连续启动两次独立的存储器操作(如连续两次读操作)所必须经过的延迟时间。后者指的是片选信号变为有效并且地址信号稳定后,数据出现的延迟时间。

(1) 静态 RAM 工作时序

静态 RAM 的读周期时序如图 6.11 所示。由于各位地址电平有高有低,输出数据的各位有“0”有“1”,所以采取高、低电平都画的方法。数据输出端在输出使能无效时处于高阻态,图中用中间位置表示。整个读周期,读/写控制 \bar{W} 始终为高电平,故省略不画。图中, t_{RC} 为读周期时间,表示连续两次读出的最小时间间隔; t_{AA} 为地址取数时间,表示从地址有效到读出数据稳定在输出端的时间间隔; t_{AC} 为片选取数时间,表示片选有效到读出数据稳定在输出端的时间间隔; t_{OE} 为输出使能取数时间,表示输出使能有效到读出数据稳定在输出端的时间间隔; t_{OLZ} 为输出使能有效到输出低阻态的时间; t_{OH} 为地址变化后读出数据保持时间; t_{ZHZ} 为片选无效后至输出高阻态的时间; t_{OHZ} 为输出使能无效后至输出高阻态的时间。

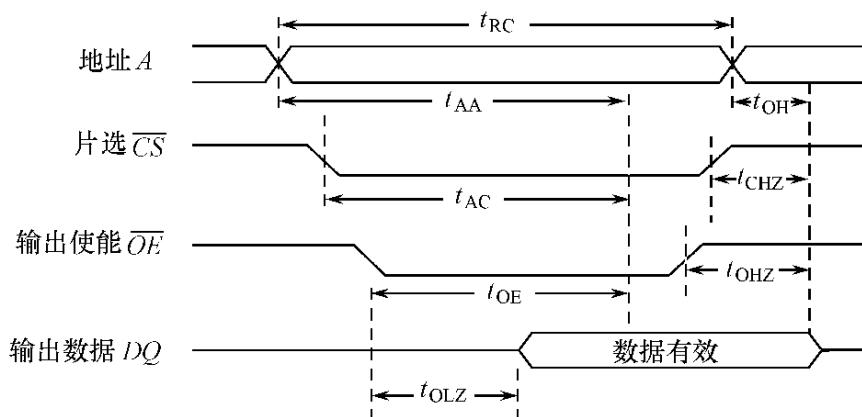


图 6.11 静态 RAM 的读周期时序示意图

静态 RAM 的写周期时序如图 6.12 所示。写周期 t_{WC} 表示连续两次写入的最小时间间隔。写入时,在有效数据出现前,数据线上存在前一时刻的数据(如图 6.11 所示的数据保持时间 t_{OH}),故在地址发生变化后, \bar{CS} 和 \bar{W} 需滞后 t_{AS} 再有效,以避免错误的写入; t_{AS} 为地址建立时间,表示从地址稳定到写使能和片选都有效的时间间隔; t_{WP} 为写脉冲宽度,是写使能和片选都有效的时间段; t_{WR} 为写恢复时间,表示写使能和片选都撤销后,地址必须保持不变的时间; t_{DW} 为数据有效时间,表示稳定的写入数据至少应在写使能和片选撤销前的 t_{DW} 时刻出现,并保持一段时间 t_{DH} 。

(2) 动态 RAM 工作时序

动态存储器技术已经发展到芯片容量按 GB 计的惊人程度,与此同时,印刷

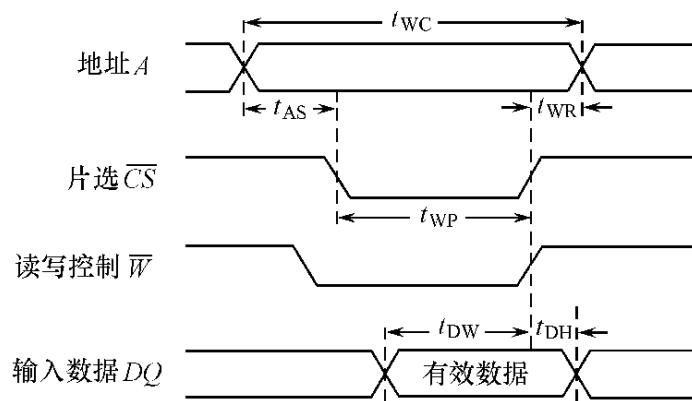


图 6.12 静态 RAM 的写周期时序示意图

电路板也有提高集成度的趋势,更小的芯片得到更多人的青睐,这样就要求芯片有更少的引脚。由于地址可以分为行地址和列地址,如果地址按两部分依次提供,则地址线的引脚可以削减一半。现今的 DRAM 芯片大都采用了这种分时复用技术,其原理结构如图 6.13 所示。

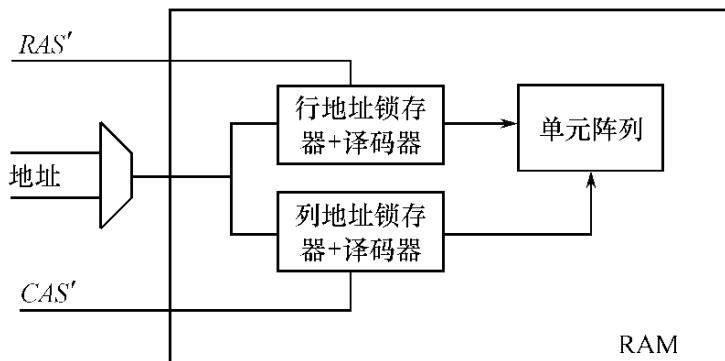


图 6.13 具有分时复用行地址和列地址的存储器

动态 RAM 的读周期时序如图 6.14 所示。首先准备好行地址,发行选通 \overline{RAS} 并使行地址保持一段时间,确保行地址正确打入行地址锁存器;发读命令即 \bar{W} 为高电平,以便 CAS 到来后可快速读出;准备好列地址,发列选通 \overline{CAS} 并使列地址保持一段时间,将列地址正确打入列地址锁存器,此时 \overline{RAS} 和 \bar{W} 应保持有效,完成数据读出。图中, t_{RC} 为读周期,表示完成一次读所需的时间; t_{RAS} 为行选通 \overline{RAS} 的脉冲宽度; t_{RP} 为 \overline{RAS} 预充电恢复时间; t_{RAC} 为从 \overline{RAS} 有效到稳定数据出现的时间。

动态 RAM 的写周期时序如图 6.15 所示。与读类似,先准备好行地址,发行选通 \overline{RAS} 并使行地址保持一段时间,确保行地址正确打入行地址锁存器;发写命令即 \bar{W} 为低电平,因未发 CAS ,没有列地址被选中,所以并未真正执行写操作;准备好列地址,在发列选通 \overline{CAS} 前的 t_{DS} 时间准备好要写入的数据(t_{DS} 为数据输入建立时间),发列选通 \overline{CAS} ,列地址保持一定时间,要写入的数据保持一段时间。

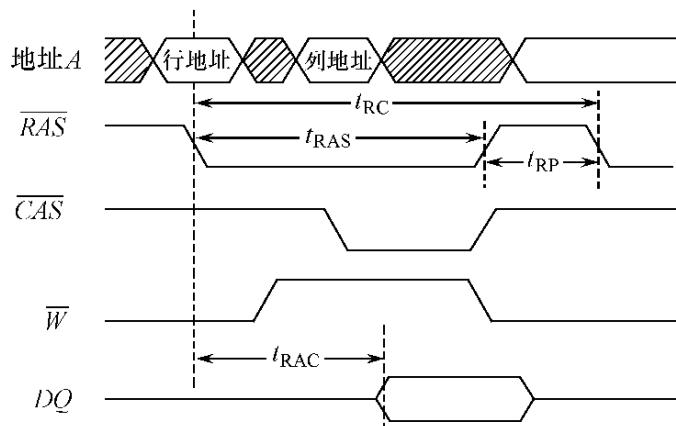


图 6.14 动态 RAM 读周期时序示意图

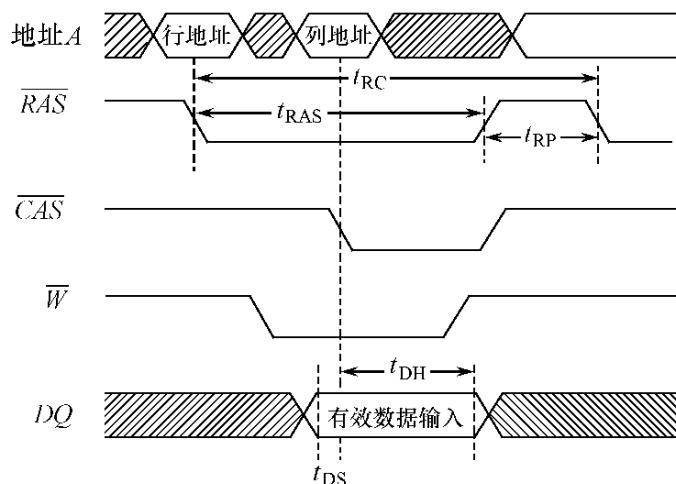


图 6.15 动态 RAM 写周期时序示意图

t_{DH} ，以便可靠地完成写操作。

由于不同芯片的性能参数是不同的，具体使用时，请参考相关手册的说明。

6.4 只读存储器结构与布尔函数的实现

从前面的介绍可知，只读存储器也是由译码器和存储体构成的。图 6.16 给出了一个 PROM 的原理结构图。

在存储矩阵中，每一个存储位元处的晶体管串联一个熔断丝。当熔断丝被烧断的存储位元被选中时，位线悬空，没有电流输出，表示存储信息“0”；当熔断丝保留的存储位元被选中时，三极管导通，位线上有电流输出，表示存储信息“1”。

使用前，用户对 PROM 进行写入。这时按地址译码后加到驱动线 W_i 上的是个比较高的电位，根据要写入信息的不同，在位线 D_j 上加不同的电位。如要

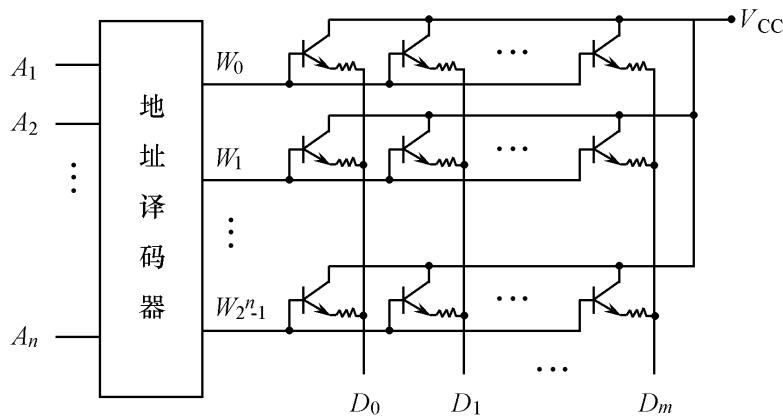


图 6.16 双极熔丝型 PROM 原理结构图

写“1”则 D_j 悬空, 如要写“0”则 D_j 加负电位, 产生大电流, 将熔断丝烧断。

在计算机中, 只读存储器主要用于存放常用的固定信息。在数字电路中, 只读存储器可用于实现组合逻辑网络, 其原理可以从“存储器”和“组合网络”两个角度来理解。

从存储器的角度看, 只要将逻辑函数的真值表事先存入 ROM, 便可以用 ROM 实现该函数。具体地说, 只要用真值表的变量取值作为存储位元的地址, 把对应的函数取值作为数据存入该位元中, 这样, 按地址读出的数据, 便是真值表中相应变量取值时的函数值。

从组合网络的角度看, ROM 中的地址译码器形成了输入变量的所有最小项, 即实现了逻辑变量的“与”运算; ROM 存储矩阵中的存储位元通过位线连接实现了最小项的“或”运算。所以, ROM 可看作为由“与”门阵列和“或”门阵列组成的逻辑网络, 如图 6.17 所示。

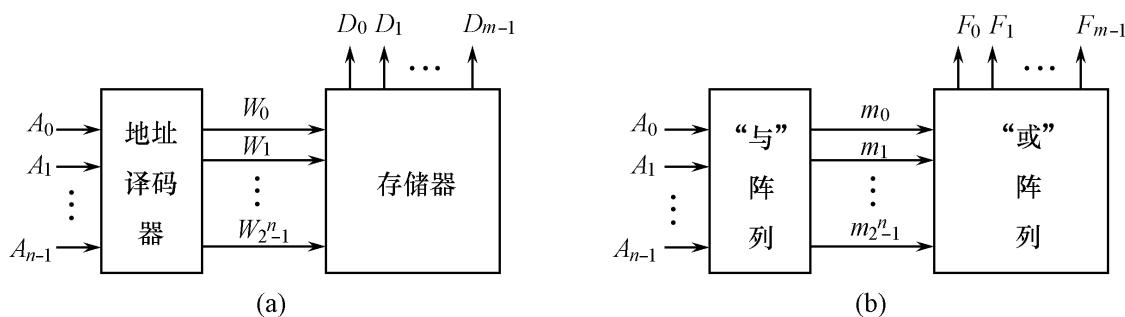


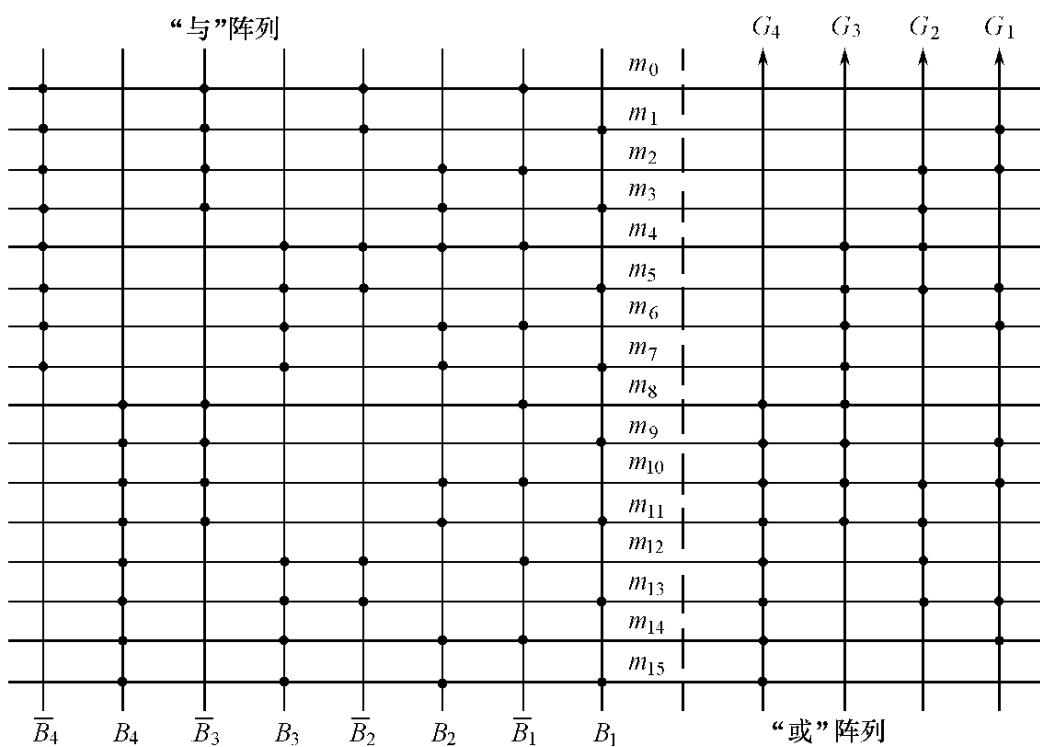
图 6.17 ROM 的逻辑结构

(a) 作为存储器 (b) 作为逻辑函数

若要实现表 6.1 所示的 4 位二进制码到 Gray 码的转换, 只要将 4 位二进制码作为地址, 将 Gray 码作为数据存入 ROM 即可。图 6.18 给出了该转换的阵列逻辑图。

表 6.1 4 位二进制码与 Gray 码的转换真值表

4 位二进制码				Gray 码			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

图 6.18 二进制码到 Gray 码的 16×4 ROM 阵列逻辑图

习 题

1. 解释下列术语：

存储位元	存储单元	存储容量	挥发性存储器
主存	辅存	Cache	破坏性读出存储器
RAM	ROM	SRAM	DRAM
PROM	EPROM	EEPROM	Flash Memory

2. 存储器有哪些分类方法？它们是如何分类的？

3. 存储器的层次结构主要体现在什么地方？为什么要分这些层次？计算机如何管理这些层次？

4. 半导体静态 RAM 和动态 RAM 的主要差别是什么？为什么动态 RAM 芯片的地址一般要分两次接收？

5. 为什么 DRAM 需要刷新？如何刷新？

6. 一个容量为 $16K \times 32$ 位的存储器，其地址线和数据线的总和是多少？

7. 用存储芯片 2114($1K \times 4$)构成 $16K \times 16$ 位存储器，画出它的逻辑图。

8. 反映存储器时间特性的两个重要指标是什么？并说明其差别。

9. 用 ROM 实现余 3 码到 BCD 码的转换。

10. 用 ROM 实现下列函数：

$$\begin{cases} F_1 = \overline{A(B + \bar{C})} + D \\ F_2 = \overline{\overline{AB} + C + \bar{B} + \bar{D}} \end{cases}$$

第七章 简易计算机设计

计算机设计包括指令集设计、硬件结构设计和逻辑电路设计等。但在开始上述工作之前,必须先确定计算机需求的功能。计算机需求的功能实际上就是用户需求的功能,这些功能需求来自计算机应用的各个领域,它们中的大部分是最基本的功能。

虽然满足用户的功能需求是计算机设计者努力追求的目标,但保持较高的性价比是设计者必须遵循的基本原则。因此,设计者必须综合考虑成本、技术、利用率、兼容性和市场大小等因素,对用户的功能需求进行分析,决定是硬件直接实现还是用软件间接实现;是优先高速实现还是一般实现。据此,计算机设计者可以决定计算机的指令系统和硬件结构。

7.1 指令系统设计

指令系统是一台计算机中所有机器指令的集合,故又称为指令集。机器指令通常简称为指令,是要计算机执行某种操作的命令,它是程序设计的最小单位,是计算机硬件能够理解并加以执行的语言。机器指令的符号表示形式称为汇编指令。

指令系统是表征一台计算机性能的重要因素,是设计计算机硬件的重要依据。它的格式与功能不仅直接影响到机器的硬件结构,而且也直接影响到系统软件的设计以及机器的适用范围。

指令系统的设计必须先根据需求确定设计的指导思想、基本目标,然后再进行具体的功能设计,主要包括指令基本功能、操作类型、寻址方式和指令格式的设计等内容。

7.1.1 指令系统设计的基本原则

不同的应用需求和使用环境,使不同类型计算机拥有的机器结构和指令系统差异很大,各有特色,很难建立一个统一的衡量标准来确定指令系统的好坏。但是,在指令系统的设计过程中,必须遵循如下基本原则:

1. 完备性原则

指令系统的完备性是指,对于任何可解的问题,都应当可以在一个可用的有限存储空间中,使用该指令系统中的指令编出(汇编语言)程序,并能运行得出正确的结果。完备性原则要求指令系统功能齐全、使用方便。

指令系统是否完备没有一个明确的标准,它只是一个指导性的原则。事实上,计算机中最基本、必不可少的指令不多,许多指令可用最基本的指令编程实现。如乘除运算指令就可以用加减和移位等指令来实现。现实计算机的指令系统一般都远远超过了基本完备性的要求,成为功能完备的指令系统。

2. 有效性原则

有效性是指利用该指令系统所编写的程序能够高效率地运行。高效率主要表现为程序占用存储空间小、执行速度快。有效性不仅反映了指令功能的强弱,而且也反映了指令系统的完备性,是一个很复杂的问题,因此也很难确定一个统一的标准。一般来说,一个功能更强、更完备的指令系统就会有更高的效率。这正是传统的复杂指令系统计算机(CISC)的设计出发点。但是,后来人们通过对CISC指令系统运行的统计分析,发现了一个80-20规律,即典型程序中80%的语句仅使用指令系统中20%的指令,而且这些指令都属于简单指令。据此,产生了精简指令系统计算机(RISC)。

3. 规整性原则

规整性包括指令系统的对称性、匀齐性和一致性三个方面。

对称性指的是指令的操作对象具有对等的关系,即指令系统中所有的寄存器和存储单元可同等对待,所有的指令都可使用各种寻址方式。如加法指令既有 $A + B \rightarrow A$,也有 $A + B \rightarrow B$ 。这种操作的对称性对于提高软件效率和便于使用是很有利的。

匀齐性指的是一种操作性质的指令可以支持各种数据类型。因此不用根据数据类型选用指令,可缩小程序空间和加快程序执行速度。但过分追求匀齐性会造成指令系统非常庞杂。

一致性指的是指令长度和数据长度有一定的关系,通常都是字节长度的整数倍,以方便存取和处理。

4. 兼容性原则

兼容性指的是不同品牌、不同机种的计算机拥有共同的基本指令系统,因此各机种上的软件可以通用。由于计算机技术的发展,不同机种推出的时间差异使得其组成和性能不同,相应的其体系结构多少存在一些差异,要做到指令系统完全兼容是不可能的,也是没有必要的。但是有两种兼容在计算机系统中是

最重要的,即现在的软件可以在以后的新型机器上运行(向后兼容)和低档计算机上的软件可以在高档机器上运行(向上兼容),其中向上兼容是系列计算机的本质特征之一。

兼容性保证了用户使用的连续性,使用户的软件投资不致于浪费。

7.1.2 指令格式

指令是要计算机执行某种操作的命令,它包含有CPU执行所需要的信息,其基本成分是:

(1) 操作码(Operation Code):计算机各种特定操作(如加、减、传送)的编码表示,英文缩写为OP。有些指令的操作码中也隐含有操作数地址。

(2) 源操作数地址(Source Operand Address/Reference):指令可对一个或多个数据进行操作,这些数据称之为源操作数。源操作数地址指出当前从何处取操作数。源操作数地址常用S标记。

(3) 目的操作数地址(Result/Target Operand Address):指出当前操作产生的结果存放在何处。目的操作数地址常用R或者T标记。

源操作数地址和目的操作数地址统称操作数地址。

每条指令在计算机内部都是用一串二进制位来表示的,称为指令字。指令字被划分为若干区域,分别对应于指令的不同构成成分。不同指令的指令字长度可以是不等长的。指令字各构成成分的结构组成形式称为指令格式。图7.1是一个典型指令格式的例子。



图7.1 典型的指令格式

一条指令中至少有一个操作码字段,而地址码字段则可以是零个、一个或多个。实际上操作数地址码也具有多样性,它可以是存储器地址、寄存器地址或编号和设备地址。根据指令中含操作数地址码的数量,称相应指令为零地址指令、一地址指令、二地址指令等等。

指令格式的设计与计算机硬件紧密相关,需考虑三个方面的问题:指令长度、操作码结构和地址码结构。

1. 指令长度

指令长度指的是一条指令中包含的二进制代码的位数。指令长度的设计受到机器字长的制约。机器字长是计算机能直接处理的二进制数据的位数,是数

据通路和寄存器的宽度。通常指令长度与机器字长有简单的倍数关系。

指令长度的选择是各种因素折衷的结果。短指令具有占用空间省、访问存储器次数少、指令执行速度快的优点，但包含信息少，指令功能弱。长指令功能强，便于程序设计，但可能造成利用率不高的浪费。解决的办法是指令系统中不同的指令字采用不同的长度，使用频率高的指令被设计成为短指令。这种变长指令格式结构灵活，能充分利用指令长度，但指令的控制复杂。而等长指令（指令系统中所有指令字长度相等）具有结构简单、便于实现的优点，但长度不好掌握——太短，则信息少；太长，则浪费大。

2. 操作码结构

操作码表明指令要计算机完成的操作性质和种类，其长度决定了指令系统中完成不同操作的指令条数。操作码长度受到指令长度和地址数的制约。操作码也可采用固定长度操作码和可变长度操作码。

3. 地址码结构

地址码结构涉及地址个数和寻址方式。

地址个数的选择是指令系统设计的一个基本决策。地址越少，指令的功能就越基本、越简单，CPU的复杂性也就越低，指令长度也越短，但指令的条数就越多，会增加程序的执行时间和复杂度。目前的机器中，地址个数一般不超过3个。

寻址方式指的是由指令中的地址（称为形式地址或逻辑地址）获得存储器地址（称为物理地址或有效地址）的方式。它用于确定本条指令的数据地址以及下一条将要执行的指令地址。本书只使用最简单、直观的直接寻址方式，即形式地址就是物理地址。有关寻址方式的详细内容请参考汇编语言程序设计和计算机原理方面的相关书籍。

指令系统应具有哪几种寻址方式，能否为用户编程提供较强的寻址能力，是指令系统设计的关键问题之一。寻址能力指的是寻址方式的多样性、灵活性以及访问地址范围的大小。寻址方式的选择，一方面需考虑寻址能力，另一方面则要考虑地址计算的复杂度，因为寻址方式与计算机硬件结构紧密相关，它涉及硬件实现的复杂度和计算机的速度。

7.1.3 指令类型和基本指令的设计

计算机要具有让用户进行任何数据处理的能力，其指令系统应包含以下四大类型的指令：

（1）算逻指令：算术运算和逻辑运算指令。

（2）传送指令：存储器与寄存器、累加器间的数据传送指令。

(3) 输入/输出指令：输入/输出设备或端口与存储器或寄存器间的信息交换指令。

(4) 控制指令：测试和转移指令。

算逻指令提供对数据的运算能力 这些运算主要是在 CPU 内的寄存器或累加器中进行的 因此需要传送指令完成存储器与寄存器、累加器间的数据传送；输入/输出指令用来把用户程序和数据传送到存储器中和把计算结果返回给用户；控制指令用于测试数据的值或者计算的状态，以决定转移到某个分支处理程序。

假设要设计的是一个称之为“小精灵”的计算机，它可做基本的加、减、乘、除运算和逻辑运算，可根据计算结果进行相应的处理和编制简单的程序在其上运行，并具有连续运行和单步运行的能力，以方便调试程序。考虑到 8 位以上的运算可由几个 8 位的运算组合而成，将“小精灵”确定为八位计算机是比较合适的，它既简单又能够满足上述基本的功能需求。另一个重要的参数是存储空间的大小，它不仅决定了用户的使用空间，而且决定了地址总线的宽度。这里，为了简单和统一，将地址总线的宽度也确定为 8 位，即存储器空间为 256 个单元。若“小精灵”计算机的构建环境(实验平台)具有将程序和数据加载进入 RAM 的能力，则主存可只包含 RAM，否则，主存由 ROM 和 RAM 两部分构成，程序和数据的输入由编程器写入 ROM。结果的输出由七段数码管显示，这样就可以省去输入/输出指令。由于完成上述功能需求的指令不会超过 32 条，因此，指令操作码占 5 位，余下的 3 位留作通用寄存器组的编码；对于含存储器地址的指令，采用两字节。综上所述，“小精灵”计算机的运算器、PC、地址寄存器、指令寄存器、累加器均为 8 位，存储器为 256×8 个单元，通用寄存器组为 8×8 个单元。乘除运算用硬件实现线路比较复杂，可以用软件实现。

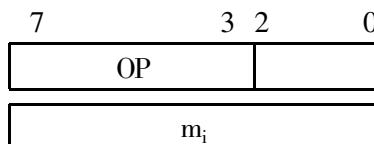
基于前面的分析，设计“小精灵”计算机的指令系统如下：

1. 传送指令

(1) 取存储器数指令

汇编形式：MOV AC ,mi；

指令格式：



指令意义：将地址为 m_i 的存储器内容送到累加器 AC 中。

(2) 送存储器指令

汇编形式：MOV mi , AC；

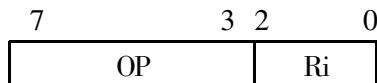
指令格式：同上。

指令意义：将累加器 AC 的内容送到地址为 m_i 的存储器中。

(3) 取寄存器数指令

汇编形式：MOV AC, R_i；

指令格式：



指令意义：将寄存器 R_i 的内容送到累加器 AC 中。

(4) 送寄存器指令

汇编形式：MOV R_i, AC；

指令格式：同上。

指令意义：将累加器 AC 的内容送到寄存器 R_i 中。

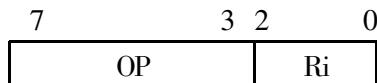
所有传送指令都不影响结果状态寄存器 C 和 Z 的值。

2. 算逻指令

(1) 加法指令

汇编形式：ADD AC, R_i；

指令格式：



指令意义：将累加器 AC 的内容与寄存器 R_i 的内容相加，结果送回到累加器 AC 中。若结果为零，则零标志寄存器 Z = 1，否则 Z = 0；若运算产生进位，则进/借位标志寄存器 C = 1，否则 C = 0。

(2) 减法指令

汇编形式：SUB AC, R_i；

指令格式：同上。

指令意义：将累加器 AC 的内容减去寄存器 R_i 的内容，结果送回到累加器 AC 中。若结果为零，则零标志寄存器 Z = 1，否则 Z = 0；若运算产生借位，则进/借位标志寄存器 C = 1，否则 C = 0。

(3) 带进位加法指令

汇编形式：ADDC AC, R_i；

指令格式：同上。

指令意义：将累加器 AC 的内容与寄存器 R_i 的内容和进位寄存器 C 的内容三者相加，结果送回到累加器 AC 中。若结果为零，则零标志寄存器 Z = 1，否则

$Z = 0$ 若运算产生进位, 则进/借位标志寄存器 $C = 1$, 否则 $C = 0$ 。

(4) 带进位减法指令

汇编形式 : SUBC AC ,R_i ;

指令格式 : 同上。

指令意义 : 将累加器 AC 的内容减去寄存器 R_i 的内容, 再减去进位寄存器 C 的内容, 结果送回到累加器 AC 中。若结果为零, 则零标志寄存器 Z = 1, 否则 Z = 0; 若运算产生借位, 则进/借位标志寄存器 C = 1, 否则 C = 0。

(5) 逻辑与指令

汇编形式 : AND AC ,R_i ;

指令格式 : 同上。

指令意义 : 将累加器 AC 的内容与寄存器 R_i 的内容按位进行逻辑“与”操作, 结果送回到累加器 AC 中。若结果为零, 则零标志寄存器 Z = 1, 否则 Z = 0; 进/借位标志寄存器 C = 0。

(6) 异或指令

汇编形式 : XOR AC ,R_i ;

指令格式 : 同上。

指令意义 : 将累加器 AC 的内容与寄存器 R_i 的内容按位进行逻辑“异或”操作, 结果送回到累加器 AC 中。若结果为零, 则零标志寄存器 Z = 1, 否则 Z = 0; 进/借位标志寄存器 C = 0。

(7) 带进位循环右移指令

汇编形式 : SHCR AC ,R_i ;

指令格式 : 同上。

指令意义 : 将寄存器 R_i 的内容与进位寄存器 C 的内容构成闭环, 进行循环右移一位操作, 结果送到累加器 AC 和进/借位标志寄存器 C 中。即 C 的内容进入 AC 的最高位, 寄存器 R_i 的内容右移一位送 AC, 寄存器 R_i 的原最低位移入 C 中。若 AC 中的结果为零, 则零标志寄存器 Z = 1, 否则 Z = 0。

(8) 带进位循环左移指令

汇编形式 : SHCL AC ,R_i ;

指令格式 : 同上。

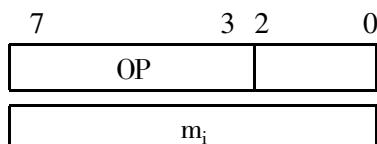
指令意义 : 将寄存器 R_i 的内容与进位寄存器 C 的内容构成闭环, 进行循环左移一位操作, 结果送到累加器 AC 和进/借位标志寄存器 C 中。即 C 的内容进入 AC 的最低位, 寄存器 R_i 的内容左移一位送 AC, 寄存器 R_i 的原最高位移入 C 中。若 AC 中的结果为零, 则零标志寄存器 Z = 1, 否则 Z = 0。

3. 控制指令

(1) 无条件转移指令

汇编形式 : JMP m_i ;

指令格式 :



指令意义 : 无条件转移到地址为 m_i 的存储器处执行。

(2) 结果非零转移指令

汇编形式 : JNZ m_i ;

指令格式 : 同上。

指令意义 : 结果非零即零标志位 $Z = 0$ 则转移到地址为 m_i 的存储器处执行。

(3) 结果无进位转移

汇编形式 : JNC m_i ;

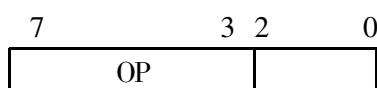
指令格式 : 同上。

指令意义 : 结果无进位即进位标志位 $C = 0$ 则转移到地址为 m_i 的存储器处执行。

(4) 停机指令

汇编形式 : HLT

指令格式 :



指令意义 : 停止执行。

所有控制指令都不影响结果状态寄存器 C 和 Z 的值。

该指令系统共有 16 条指令 , 由于操作码为 5 位 , 若需要 , 可扩充到 32 条指令。

7.2 运算器设计

7.2.1 运算器设计

运算器的设计主要是围绕算术/逻辑单元(ALU)的设计以及它与寄存器、数

据总线之间如何传递操作数和运算结果而进行的。

在进行算术/逻辑单元的设计之前,进一步明确两变量函数输入与输出的关系是很有必要的。设有两变量 X 和 Y ,它们可形成 4 个最小项,在二值空间可组合成 16 种逻辑函数,其通式为:

$$F = \alpha_0 \bar{X} \bar{Y} + \alpha_1 \bar{X} Y + \alpha_2 X \bar{Y} + \alpha_3 X Y$$

其中,系数 $\alpha_0 \sim \alpha_3$ 为特征值,共有 16 种组合。将其代入上式,可得 16 种逻辑函数,如表 7.1 所示。

表 7.1 两变量组成的逻辑函数

$\alpha_0 \alpha_1 \alpha_2 \alpha_3$	函数	功能	$\alpha_0 \alpha_1 \alpha_2 \alpha_3$	函数	功能
0 0 0 0	$F_0 = 0$	清零	1 0 0 0	$F_8 = \bar{X} + \bar{Y}$	或非
0 0 0 1	$F_1 = XY$	与	1 0 0 1	$F_9 = X \odot Y$	同或
0 0 1 0	$F_2 = X\bar{Y}$	与 Y 非	1 0 1 0	$F_{10} = \bar{Y}$	Y 非
0 0 1 1	$F_3 = X$	传送 X	1 0 1 1	$F_{11} = X + \bar{Y}$	或 Y 非
0 1 0 0	$F_4 = \bar{X}Y$	与 X 非	1 1 0 0	$F_{12} = \bar{X}$	X 非
0 1 0 1	$F_5 = Y$	传送 Y	1 1 0 1	$F_{13} = \bar{X} + Y$	或 X 非
0 1 1 0	$F_6 = X \oplus Y$	异或	1 1 1 0	$F_{14} = \overline{XY}$	与非
0 1 1 1	$F_7 = X + Y$	或	1 1 1 1	$F_{15} = 1$	置 1

虽然这些功能对“小精灵”计算机来说并不都是必需的,但考虑到这些功能的实现并不复杂,且有利于指令功能的快速实现和今后功能的扩展,在 ALU 中全部实现这些功能是个好主意。

由于 ALU 不仅要完成逻辑运算,还必须完成算术运算,因此在图 7.2 给出的一位 ALU 基本单元逻辑图中增加了一个算术/逻辑运算选择控制。

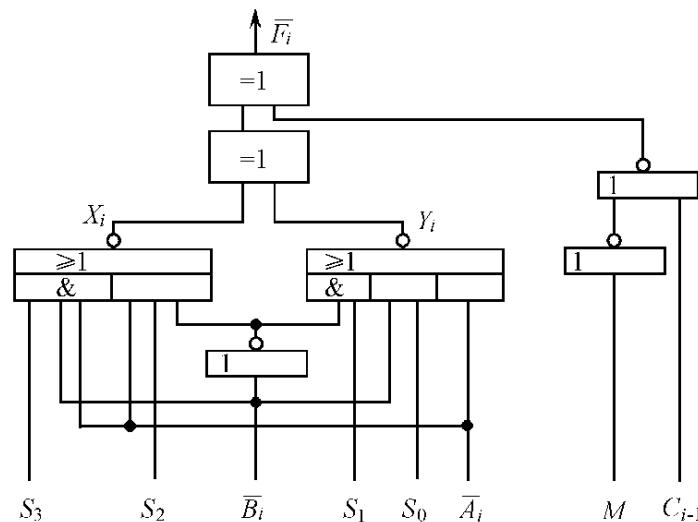


图 7.2 一位 ALU 基本单元逻辑图

该一位 ALU 由全加器、算术/逻辑运算选择控制 M 和基本操作功能选择控制 $S_3S_2S_1S_0$ 三部分组成。其中全加器由两个半加器(异或门)实现。算术/逻辑运算选择控制 $M = 0$, 开门接收低位来的进位信号 C_{i-1} , ALU 进行算术运算 ; $M = 1$, 关门不接收 C_{i-1} , ALU 进行逻辑运算。基本操作功能选择控制 $S_3S_2S_1S_0$ 通过对输入数据 A_i 和 B_i 的不同组合, 实现不同的功能, 其关系如表 7.2 所示(注意, 运算符“ + ”表示逻辑或)。

表 7.2 功能表

$S_3S_2S_1S_0$	$M = 1$ 逻辑运算	$M = 0$ 算术运算	
		$C_{i-1} = 0$	$C_{i-1} = 1$
0 0 0 0	\bar{A}	A 减 1	A
0 0 0 1	\bar{AB}	AB 减 1	AB
0 0 1 0	$\bar{A} + B$	A 减 1	A
0 0 1 1	逻辑 1	减 1	0
0 1 0 0	$\bar{A} + \bar{B}$	A 加 $(A + \bar{B})$	A 加 $(A + \bar{B})$ 加 1
0 1 0 1	\bar{B}	AB 加 $(A + \bar{B})$	AB 加 $(A + \bar{B})$ 加 1
0 1 1 0	$\bar{A} \oplus \bar{B}$	A 减 B 减 1	A 减 B
0 1 1 1	$A + \bar{B}$	$A + \bar{B}$	$(A + \bar{B})$ 加 1
1 0 0 0	$\bar{A}\bar{B}$	A 加 $(A + B)$	A 加 $(A + B)$ 加 1
1 0 0 1	$A \oplus B$	A 加 B	A 加 B 加 1
1 0 1 0	B	$\bar{A}\bar{B}$ 加 $(A + B)$	$\bar{A}\bar{B}$ 加 $(A + B)$ 加 1
1 0 1 1	$A + B$	$A + B$	$(A + B)$ 加 1
1 1 0 0	逻辑 0	A 加 A	A 加 A 加 1
1 1 0 1	$\bar{A}\bar{B}$	AB 加 A	AB 加 A 加 1
1 1 1 0	AB	$\bar{A}\bar{B}$ 加 A	$\bar{A}\bar{B}$ 加 A 加 1
1 1 1 1	A	A	A 加 1

与最小项的控制关系是 :

$$F_i = \bar{S}_3\bar{A}_i\bar{B}_i + \bar{S}_2\bar{A}_iB_i + S_0A_i\bar{B}_i + S_1A_iB_i$$

特别需要说明的是, $S_3S_2S_1S_0$ 的选择还充分考虑了并行进位链的需要, 让 X_i 中包含进位传递条件 $P_i = A_i + B_i$, Y_i 中包含本地进位 $G_i = A_iB_i$, 当 $S_3S_2S_1S_0 = 1001$ 和 $M = 0$ 时, 利用 X_i 和 Y_i 不仅可进行求和, 即 $(A_i + B_i) \oplus A_iB_i = A_i \oplus B_i$, 还可方便地形成并行进位。表 7.3 给出了 $S_3S_2S_1S_0$ 与 X_i 、 Y_i 的逻辑关系。

在运算器中, ALU 不仅是算术/逻辑运算的部件, 而且还是数据传送的主要通路, 有许多待加工处理或传送的数据汇集在它的入口, 而 ALU 只是一个纯粹的组合逻辑电路, 参加运算的操作数在运算期间须保持不变, 此外运算的结果也

需要保存,因此运算器中必须包含存储部件。根据“小精灵”计算机指令系统设计的结果,存储部件设置了8个通用寄存器 R_i ,一个功能强大的累加器 AC ,一个进/借位标志寄存器 C 和一个结果为零标志寄存器 Z 。其中,累加器 AC 具有加载、保持、附带进/借位标志寄存器 C 一起进行循环左移和循环右移的功能。其结构如图7.3所示。

表7.3 $S_3S_2S_1S_0$ 与 X_i, Y_i 的逻辑关系

S_3	S_2	X_i	S_1	S_0	Y_i
0	0	1	0	0	A_i
0	1	$A_i + \bar{B}_i$	0	1	A_iB_i
1	0	$A_i + B_i$	1	0	$A_i\bar{B}_i$
1	1	A_i	1	1	0

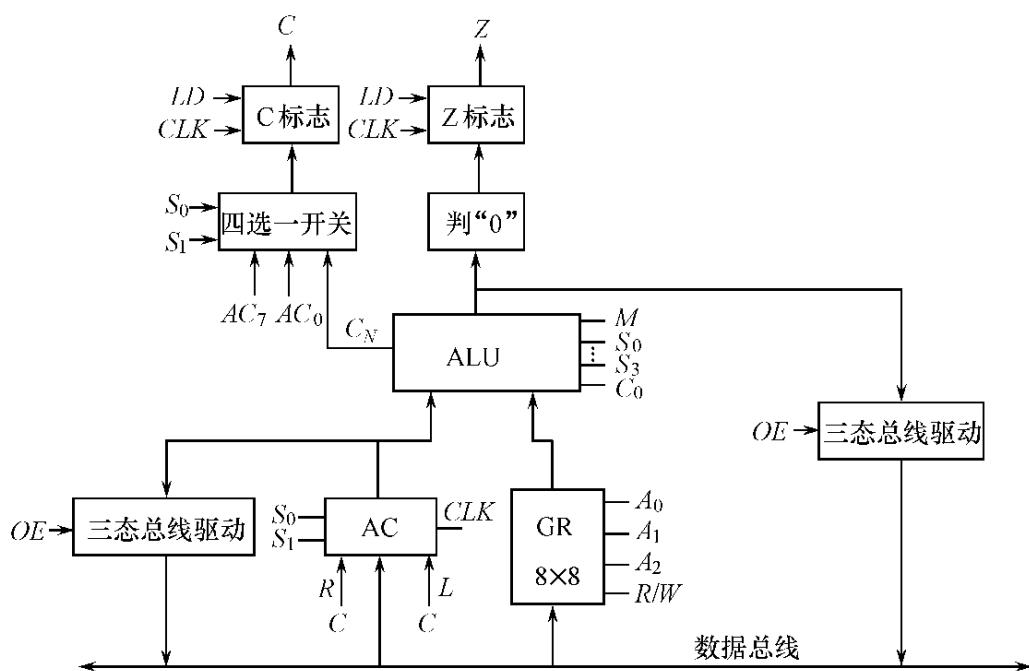


图7.3 运算器的构成

该运算器采用原码运算,对多字节加减法,利用 ADDC 和 SUBC 指令实现;对于乘除法则利用 SHCR 和 SHCL 指令完成。目前实现的 ALU 充分考虑了系统功能扩展的需要,若要增加其他的寻址方式,ALU 还可用于地址计算。

7.2.2 乘法和除法运算

乘法和除法运算用电路实现要比加、减运算复杂得多,而乘法和除法运算可

以被看成是加法和减法的运算序列,通过重复的加、减运算可以实现有效的乘除运算。因此,在基本的微型计算机结构中不包含乘、除指令。

1. 乘法运算

用加法运算实现乘法运算 $Z = X * Y$,最简单、直观的方法是把 Y (被乘数)和部分和 Z (初值为零)重复相加 X 次。这种算法尽管简单,但效率也差。因为要计算 X 步,而 X 可能会很大。所以必须找出一种方法能使 X 尽快减少,即用更大的幅度减少。显而易见的方法是使 X 减半而不是减 1(这要求 X 为偶数),部分和加 $2 * Y$ 而不是 Y ,而减半或加倍的操作可以通过简单的移位操作实现。用算法可描述如下:

```
a = X ;b = Y ;z = 0 ;
while( a != 0 )
    if ODD( a ) then //a 为奇数
        {a = a - 1 ;z = z + b ;}
    else           //a 为偶数
        {a = a/2 ;b = 2 * b ;}
```

显然,若 X, Y 为单字节长度的数,则 z 为双字节长度的数,而 b 需要做加倍运算,因此也是双字节长度的数, $z + b$ 则为双字节的加法。为方便在计算机上用汇编语言编程实现上述乘法运算,假定 X, Y 的最大位数为 N ,则 z 的最大位数为 $2 * N$,对上面的算法变化如下:

```
a = X ;b = Y ;z = 0 ;i = N ;
while( i != 0 )
{
    if ODD( a ) then //a 为奇数
        z = z + b * 2N ;
    a = a/2 ;z = z/2 ;i = i - 1 ;
}
```

这里,若 $N = 8$,则 $z = z + b * 2^N$ 只需在 z 的高 8 位进行加 b 的运算,且 a 和 z 都是朝同一个方向移位。在“小精灵”计算机上具体实现时,积的高 8 位存放在一个寄存器中(相当于 z),积的低 8 位和乘数 X 可共用一个寄存器(相当于 a, z 中右移出的数移入 a 的高位),被乘数 Y 使用一个寄存器(相当于 b)。

采用 4 位寄存器($N = 4$),计算 $Y = 13$ 和 $X = 11$ 乘积的过程如下所示:

b	z	a	说明
1101	0000	1011	初始化 $i = 4$
01101	1011		a 为奇数 $z + b * 2^N$ 和包含进位

0110	1101	a、z 右移一位 $i = 3$
10011	1101	a 为奇数 $z + b * 2^N$ 和包含进位
1001	1110	a、z 右移一位 $i = 2$
1001	1110	a 为偶数
0100	1111	a、z 右移一位 $i = 1$
10001	1111	a 为奇数 $z + b * 2^N$ 和包含进位
1000	1111	a、z 右移一位 $i = 0$

所以 积为 10001111 ,十进制数为 143。

用“小精灵”计算机指令系统编制的 8 位乘法程序如下：

```

MOV AC ,R0 ;          ; AC 清零
XOR AC ,R0 ;          ; AC 清零
MOV R0 ,AC ;          ; R0 放积的高 8 位 ,初值为零 ,作为 z
MOV AC ,100 ;           ; 乘数 X 存放在 100 号地址空间中
MOV R1 ,AC ;           ; R1 放乘数 X ,作为 a
MOV AC ,101 ;           ; 被乘数 Y 存放在 101 号地址空间中
MOV R2 ,AC ;           ; R2 放被乘数 Y ,作为 b
MOV AC ,102 ;           ; 在 102 号地址空间中存放有数字 8 ,指明位数
MOV R3 ,AC ;           ; R3 作为 i
MOV AC ,103 ;           ; 在 103 号地址空间中存放有数字 1
MOV R4 ,AC ;           ; R4 存放数字 1 ,为 i - 1 作准备
LOOP : SCHR AC ,R1 ;   ; 移出 a 的低位 ,准备检查是否奇数
                           ; a 移出的低位为零 ,说明是偶数
                           ; 从 R0 中取出 z
                           ; 完成 z + b * 2N
                           ; 送回 R0 中保存
NEXT : SCHR AC ,R0 ;   ; z/2 移出的低位进入标志寄存器 C
                           ; 送回 R0 中保存
                           ; a/2 标志寄存器 C 中的值移入 a
                           ; 送回 R1 中保存
                           ; 从 R3 中取出 i
                           ; i - 1
                           ; 送回 R3 中保存
                           ; i 不等于零 ,继续
                           ; 停机

```

从这段程序可以看出 ,如果“小精灵”计算机指令系统含有减 1 和立即数传送指令 ,该程序段还可进一步简化。

2. 除法运算

就像乘法运算可以通过重复的加法运算实现一样,通过重复进行减法运算可以实现除法。令 $X = \text{被除数}$ 、 $Y = \text{除数}$ 、 $q = \text{商}$ 、 $r = \text{余数}$,则有算法:

```

r = X ; q = 0 ;
while( r > = Y )
    {r = r - Y ; q = q + 1 ;}

```

和乘法运算一样,这个算法虽然简单,但效率不高。更好的算法在于更快地减少 r ,选择 Y 和 2 的幂次乘积为减数,这样就能尽可能快速地减少 r 。假定除数的最大位数为 N ,则算法如下:

```

r = X ; q = 0 ; i = 0 ;
while( i < N )
{
    r = 2 * r ; q = 2 * q ; i = i + 1 ;
    if( r > = Y * 2N ) then {r = r - Y * 2N ; q = q + 1 ;}
}

```

同样,在初始化时,用一个双倍长度的寄存器存放余数,为节省资源,该寄存器同时存放了余数(递减)和商(递增)。

7.3 控制器设计

根据存储程序控制原理,计算机是由存储在存储器中的程序控制自动进行工作的,而程序又是由有序的指令序列构成,指令通过控制器的处理产生相应的控制信号协调计算机中各部件的工作。从逻辑实现的角度来看,这一过程包括对以下步骤的控制:

- (1) 控制取指令字(指令流出控制);
- (2) 解释指令字(指令分析控制);
- (3) 组织控制计算机各部件动作的信号序列,完成指令的执行(指令执行控制);
- (4) 确定下一条指令的地址(指令流向控制)。

从这里可以看出,计算机对指令的执行不是一步完成的,它需要将其分解成为若干最基本的操作,并产生一系列相应的控制信号,控制计算机各部件完成这些操作。相对指令而言,这种对一个部件的控制操作,在计算机控制器的设计中被称之为微操作。这样,一条指令的执行过程实际上就是一系列微操作的控制过程,而不同的指令对应着不同的微操作序列。虽然不同计算机系统包含的微

操作可以是不同的,但是计算机系统所有微操作覆盖的功能应该是相同或相近的。正确合理地分析和设置微操作,是设计计算机控制器的前提。而控制器设计的主要工作就是要确定指令与微操作序列的对应关系,以及各种微操作控制信号产生的时间和方法。

现代计算机中微操作序列的形成方法有两种,一种是传统的组合逻辑设计方法,另一种是微程序设计方法。前者为硬连线逻辑,一旦控制部件构成以后,便难以修改。后者为存储逻辑,具有较好的可修改性和可扩充性,其原理与存储程序控制原理一致,它将每一个微操作用一位二进制码表示,1表示执行该微操作,0表示不执行,所有这些控制位组合在一起构成一条微指令(微指令直接编码方式),对应一个或几个微操作。然后,将每一条指令的微操作序列用微指令编制成微程序,存放到控制存储器中,控制器处理一条指令的工作过程,就是启动该指令对应的微程序的执行过程。有关微程序控制器的设计技术请参考有关书籍,本书不做详细论述。

组合逻辑设计方法在简单指令系统的情况下非常好用,但在复杂指令系统的情况下,设计复杂性较大,控制电路的设计质量难以保证;此外,其设计方法不便于形式化实现,设计效率较低。与组合逻辑控制相比,微程序控制具有规整性、灵活性、可维护性等许多优点,控制器的调试和修改非常方便,但速度比组合逻辑控制慢,几乎所有指令处理的速度都是一样的。

用组合逻辑方法设计控制器的基本步骤为:

- (1) 根据功能要求和指令系统设计硬件逻辑总体结构图;
- (2) 确定指令时序的控制方式;
- (3) 根据指令系统确定计算机指令周期;
- (4) 列出微操作清单,建立微操作时间表;
- (5) 写出微操作命令的最简逻辑表达式;
- (6) 画出逻辑图。

7.3.1 硬件逻辑结构总体设计

根据“小精灵”计算机的功能需求和指令系统设计的结果,利用第四章介绍的计算机结构组成的知识,可以确定“小精灵”计算机控制器应包含脉冲源、节拍信号发生器、启停电路(用于手动控制)、程序计数器 PC、指令寄存器 IR、地址寄存器 AR、指令译码器和组合控制电路等部分。结合前面介绍的运算器、存储器和总线,构造“小精灵”计算机硬件总体结构如图 7.4 所示。

图中,通用寄存器组 GR 的选择控制端 $A_2A_1A_0$ 与指令寄存器 IR 的低 3 位相连,累加器 AC 与标志寄存器 C 相连用于循环左/右移位。为了控制简单,

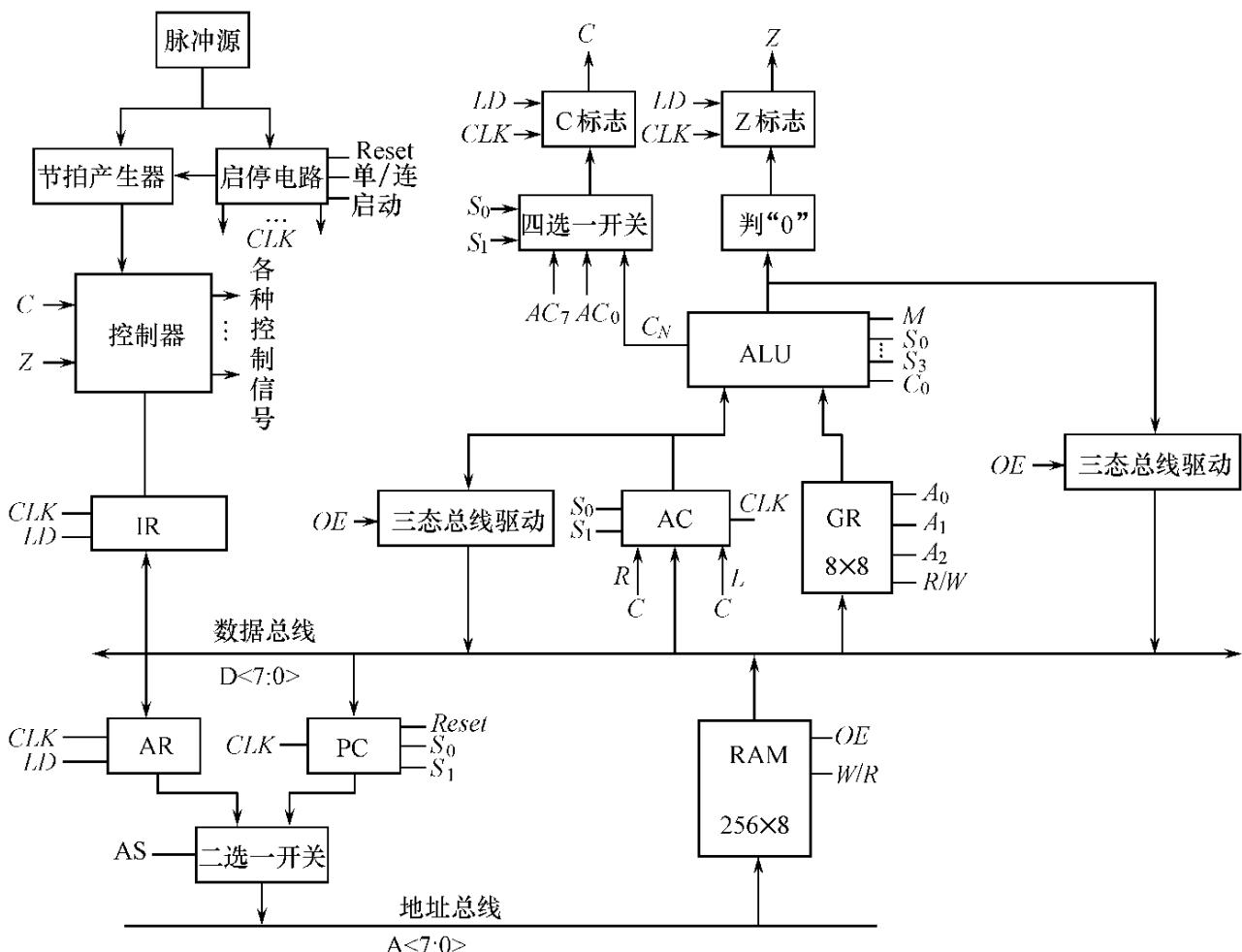


图 7.4 “小精灵”计算机硬件逻辑结构图

RAM 采用静态存储器。启停电路用于控制产生时钟脉冲和节拍信号，其时钟脉冲的输出端 **CLK** 为多个，输出相同的时钟信号，用于为不同的器件提供时钟，防止单个时钟脉冲输出为多器件使用而引起负载过大的问题。**Reset** 为复位按钮，将 **PC** 和节拍计数器置零。算逻单元 **ALU** 的控制端含义参见前面的表 7.2，其他器件各控制端的具体含义为：

$$\text{MUX2} \cdot \text{AS} = \begin{cases} 0 & \text{选择 PC} \\ 1 & \text{选择 AR} \end{cases}$$

$$\text{MUX4} \cdot \text{S}_1\text{S}_0 = \begin{cases} 00 & \text{选择 AC}_0 \\ 01 & \text{选择 AC}_7 \\ 10 & \text{选择 C}_N \\ 11 & \text{接地} \end{cases}$$

$$\text{RAM} \cdot \text{OE} = \begin{cases} 0 & \text{使能} \\ 1 & \text{禁止} \end{cases}$$

$$\text{RAM} \cdot \text{W/R} = \begin{cases} 0 & \text{读} \\ 1 & \text{写} \end{cases}$$

$$AC \cdot S_1 S_0 = \begin{cases} 00 & \text{保持} \\ 01 & \text{加载} \\ 10 & \text{左移} \\ 11 & \text{右移} \end{cases}$$

$$PC \cdot S_1 S_0 = \begin{cases} 00 & \text{保持} \\ 01 & \text{加载} \\ 10 & \text{加 1} \\ 11 & \text{未定义} \end{cases}$$

$$IR \cdot LD = \begin{cases} 0 & \text{保持} \\ 1 & \text{加载} \end{cases}$$

$$HLT = \begin{cases} 0 & \text{停时钟} \\ 1 & \text{无影响} \end{cases}$$

地址寄存器 AR、进/借位标志寄存器 C 和零标志寄存器 Z 的 LD 端定义与指令寄存器 IR 的 LD 端定义相同。累加器 AC 输出到数据总线的三态门控制 OE、算逻单元 ALU 输出到数据总线的三态门控制 OE 与存储器 RAM 的 OE 定义相同。

需要指出的是,计算机硬件总体结构的设计必须考虑指令系统微操作的需要,看能否为其提供相应的硬件支持;同时,指令系统微操作序列的确定也必须结合硬件总体结构来进行。7.3.4 小节给出了“小精灵”计算机指令系统微操作序列的分析。

7.3.2 指令时序的控制方式

指令中的每一个微操作都要由相应的时序信号激发,控制器最重要的功能就是进行指令的时序控制。由于不同指令对应的微操作个数及复杂程度不同,因此每条指令和每个微操作所需的执行时间也不同。采用什么方式形成不同微操作序列的时序控制,是控制器设计要解决的首要问题。

常见的时序控制方式有同步控制方式、异步控制方式和人工控制方式三种。

1. 同步控制方式

指令系统有统一的时钟信号,所有微操作控制信号都与时钟信号同步。其特点是每个微操作的执行,都由基准时钟的时序信号所控制。每个时序信号的结束,标志着这个时序信号所代表的时间段中的操作已完成,可以开始后续的微操作。简单地说,就是“以时定序”。图 7.5 展示了微操作控制信号与系统统一时钟的同步关系。

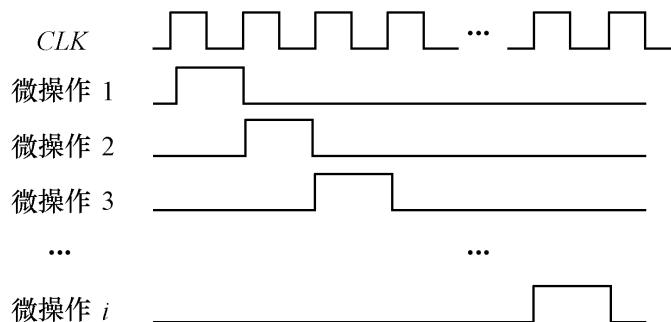


图 7.5 同步控制方式

在同步控制方式下,要求所有的微操作必须在一个时钟周期内完成,即必须选择系统的同步时钟周期,使其不小于所有微操作单个工作时间的最大值。显然,这对于其他所有耗时较少的微操作,必然造成时间的浪费,这是同步控制方式的缺点。但相对其他控制方式来说,其优点是设计简单、节省器材、便于调试、系统较为可靠。目前计算机控制器设计中主要采用这种控制方式。

2. 异步控制方式

系统没有统一的时钟,微操作控制信号采用“以序定时”,即一个微操作是用其前一个微操作的结束信号启动的,而不是由统一的时钟周期来控制。这样,每个微操作需要多少时间就占用多少时间,几乎没有时间的浪费,这是异步控制方式的优点。相对同步控制方式来说,其缺点是设计比较复杂,消耗较多的器材,系统调试难度大,且工作过程中的可靠性不易保证。

异步控制方式一般用于各自具有不同的时序系统的设备,这时,设备之间的信息交换采用应答方式,如图 7.6 所示。

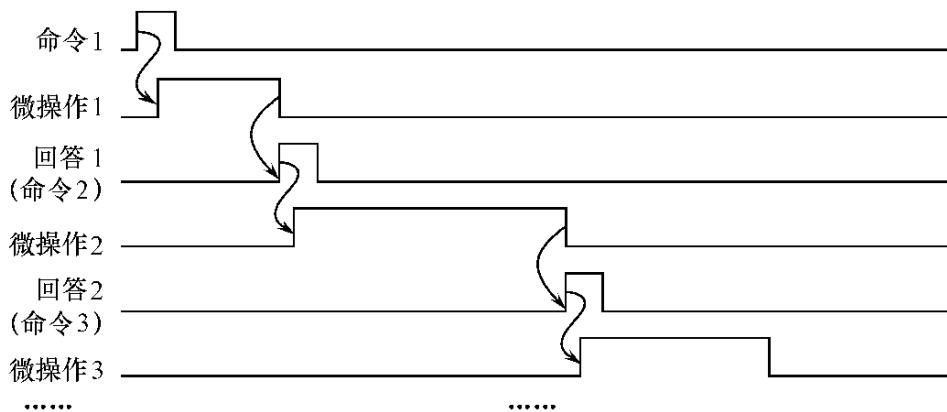


图 7.6 异步控制方式

3. 人工控制方式

为了机器调试和软件开发的需要,可在机器面板或内部设置一些开关或按

键,实现人工控制。如机器复位键,连续或单步执行开关。

7.3.3 指令周期和时标系统

一条指令的执行时间,即从取指令、分析指令直到指令执行结束所花的时间,称为指令周期。不同的指令,对应的微操作不同,执行时间的长短也不同,因此具有不同的指令周期。通常,把一个指令周期划分为若干时间段(称为机器周期),将指令执行过程中几个较大的动作放入其中,如取指令、分析指令、间接寻址、执行指令等,相应地,这些机器周期分别称为取指周期(含分析指令)、间址周期、执行周期。在同步控制中,为了精细地控制微操作的执行,将一个机器周期中的若干微操作进一步分配到一个同步时钟周期(称为节拍)。这样,一个机器周期内就包含若干时钟周期(节拍),而每个节拍可以完成一个或几个允许同时执行的微操作。

图 7.7 展示了同步时钟、节拍、机器周期、指令周期和微操作的关系。形成节拍电位的电路(见第三章 3.5.3 节“节拍信号发生器”)。

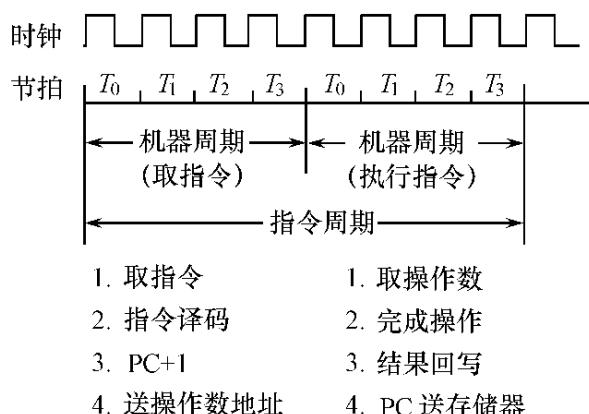


图 7.7 同步时钟、节拍、机器周期、指令周期和微操作

虽然不同的指令含有不同的机器周期,但是所有指令的第一个机器周期一定是取指周期,所以任何一条指令最少需要两个机器周期,复杂的指令需要较多的机器周期。

面对不同的指令有不同的指令周期,在同步控制方式中可采用下面的三种策略确定指令周期:

(1) 中央控制

这是一种“统一划齐”的控制方式。即不论指令所对应的微操作序列有多长,也不管微操作的简繁,一律以最长的微操作序列和最繁的微操作为标准,采用完全统一的、具有相同时间间隔和相同数目的节拍作为机器周期来运行各种不同的指令,所有指令的指令周期都是相同的。这种方法的优点是控制器的逻

辑简单,时间便于控制,缺点是在效率和功能上给控制器设计带来矛盾。考虑效率就必须使所有指令的执行时间相同或相近,限制采用功能强、处理时间长的指令;考虑功能,就需要选择一些功能强、处理时间长的指令,那么其他简单指令就存在时间浪费、效率不高的问题。

一些功能简单的控制器常采用这种方法。本书要设计的“小精灵”计算机,因指令系统简单,采用这种方法是比较合适的。

(2) 局部控制

这是一种“按需分配”的方式。即根据不同类型指令所需微操作的种类和数目的不同,控制器在解释执行时按实际所需分配相应数目的节拍,且不同的指令周期中含有不同的机器周期。这种方法是相对中央控制的另一个极端,因此它的优缺点正好与中央控制相反,即处理效率高,控制器逻辑复杂。

(3) 混合控制

这是一种折衷的方案。它根据绝大多数指令的需要,规定一个基本的节拍数作为各种指令共同要执行的周期,称为中央周期;对少数在中央周期内不能完成的指令,可根据需要采用插入节拍的方法,对其处理时间较长的微操作进行局部控制,这些延长的节拍称为局部周期。即混合控制设计的控制器对所有指令都采用中央控制,对中央控制难以处理的指令,当处理到该指令需要长时间的微操作时,中央控制时序被暂停,该指令的局部控制时序被启动,局部控制时序终止时,再次启动中央控制时序,完成指令处理的所有步骤。

混合控制兼顾了中央和局部两种控制方式的优点,在控制器设计的简单性、高效性和功能的完整性上取得了良好的一致,因此得到广泛的应用。

7.3.4 指令周期和时钟周期的确定

要确定指令的周期,必须先分析指令系统,确定其微操作序列。这里结合图7.4对“小精灵”计算机指令系统微操作序列分析如下:

1. 取存储器数指令 MOV AC ,m_i;

(1)所有指令的第一个节拍是共同的,都是完成取指令的操作。即控制多路选择器MUX2的AS端选择PC,使PC中的地址经地址总线送到存储器,同时发送读信号和输出使能信号给存储器的W/R端和OE端,将指令读到数据总线上。置指令寄存器IR的加载控制端LD,准备接收指令;控制程序计数器PC上的S₀S₁,准备完成PC加1的操作。

(2)第二个节拍对应的时钟脉冲上升沿将数据总线上的指令码打入指令寄存器IR并实现PC加1。然后,指令寄存器中的指令随即被译码,产生后续操作

的控制信号。对本指令来说,就是控制多路选择器 MUX2 的 AS 端选择 PC,使 PC 中的地址送存储器,读信号和输出使能信号送 W/R 和 OE,将该指令的第二个操作数的地址 m_i 读到数据总线上。置地址寄存器 AR 的加载控制端 LD,准备接收操作数地址 m_i ,控制程序计数器 PC 上的 S_0S_1 ,准备完成 PC 加 1 的操作。

(3) 第三个节拍对应的时钟脉冲上升沿将数据总线上的操作数地址 m_i 打入地址寄存器 AR 并实现 PC 加 1。控制多路选择器 MUX2 的 AS 端选择 AR,使 AR 中的地址送存储器,读信号和输出使能信号送 W/R 和 OE,将第二个操作数读到数据总线上。置累加器 AC 的控制端 S_0S_1 ,准备接收操作数。

(4) 第四个节拍对应的时钟脉冲上升沿将数据总线上的操作数打入累加器 AC。

需要说明的是,由于第四个节拍只用时钟脉冲上升沿将数据总线上的操作数打入累加器 AC,并没有其他的操作,因此,该节拍实际上可以与下一指令的取指操作共用。其指令周期可以确定为三个节拍。

2. 送存储器指令 MOV m_i ,AC ;

(1) 与第一条指令的第一节拍相同。

(2) 与第一条指令的第二节拍相同。

(3) 第三个节拍对应的时钟脉冲上升沿将数据总线上的操作数地址 m_i 打入地址寄存器 AR 并实现 PC 加 1。控制多路选择器 MUX2 的 AS 端选择 AR,使 AR 中的地址送存储器,写信号和输出禁止信号送存储器的 W/R 和 OE,累加器 AC 输出的三态使能 OE 允许输出。

需要说明的是,存储器的输出禁止 OE 和累加器的输出使能 OE 安排在同一个节拍,电路延迟的差异性可能造成在某一时刻两个设备同时向总线输出,因其时间短暂瞬变,只会造成器件寿命的下降,不会造成电路损坏,为了简单“小精灵”计算机的设计没有对此做特别的处理。事实上,这种冲突不仅在该指令中存在,在指令间的转换中也存在。从后面小节列出的微操作时间表(表 7.4)中可以清楚地发现这种冲突。正常情况下,应通过增加节拍数来消除这种冲突。

3. 取寄存器数指令 MOV AC ,R_i ;

(1) 与第一条指令的第一节拍相同。

(2) 第二个节拍对应的时钟脉冲上升沿将数据总线上的指令码打入指令寄存器 IR 并实现 PC 加 1。指令译码 R_i 的编码送通用寄存器组 GR 的 A₀A₁A₂ 端,读信号送 GR 的 W/R 端,置 ALU 的 S₀S₁S₂S₃ 端和三态使能 OE 允许输出,进行数据 R_i 的传送操作,置累加器 AC 的 S₀S₁ 端准备接收数据。

(3) 第三个节拍对应的时钟脉冲上升沿将数据总线上的数据打入 AC。

4. 送寄存器指令 MOV R_i ,AC ;

(1) 与第一条指令的第一节拍相同。

(2) 第二个节拍对应的时钟脉冲上升沿将数据总线上的指令码打入指令寄存器 IR 并实现 PC 加 1。指令译码 ,置 ALU 的 $S_0S_1S_2S_3$ 端和三态使能 OE 允许输出 ,进行数据 AC 的传送操作 , R_i 的编码送通用寄存器组 GR 的 $A_0A_1A_2$ 地址端 ,写信号送 GR 的 W/R 端。

5. 加法指令 ADD AC , R_i ;

(1) 与第一条指令的第一节拍相同。

(2) 第二个节拍对应的时钟脉冲上升沿将数据总线上的指令码打入指令寄存器 IR 并实现 PC 加 1。指令译码 , R_i 的编码送通用寄存器组 GR 的 $A_0A_1A_2$ 端 , 读信号送 GR 的 W/R 端 , 置 ALU 的 $S_0S_1S_2S_3$ 端和 M 端 , 选择算术加法运算功能 ; 置 ALU 的三态使能 OE 允许结果输出到数据总线 , 置累加器 AC 的 S_0S_1 端准备接收结果 , 置多路选择器 MUX4 的 S_0S_1 端选择进位 C_N , 置进位/借位标志寄存器 C 和零标志寄存器 Z 的加载控制端 LD , 准备接收结果状态。

(3) 第三个节拍对应的时钟脉冲上升沿将数据总线上的结果打入 AC、进位打入 C、零标志打入 Z。

6. 减法指令 SUB AC , R_i ;

(1) 与第一条指令的第一节拍相同。

(2) 除选择算术减法运算外 , 其余与上条指令的第二节拍相同。

(3) 第三个节拍对应的时钟脉冲上升沿将数据总线上的结果打入 AC、借位打入 C、零标志打入 Z。

7. 带进位加法指令 ADDC AC , R_i ;

(1) 与第一条指令的第一节拍相同。

(2) 第二个节拍对应的时钟脉冲上升沿将数据总线上的指令码打入指令寄存器 IR 并实现 PC 加 1。指令译码 , R_i 的编码送通用寄存器组 GR 的 $A_0A_1A_2$ 端 , 读信号送 GR 的 W/R 端 , 进位/借位标志寄存器 C 送 ALU 的 C_0 , 置 ALU 的 $S_0S_1S_2S_3$ 端和 M 端 , 选择算术加法运算功能 ; 置 ALU 的三态使能 OE 允许结果输出到数据总线 , 置多路选择器 MUX4 的 S_0S_1 端选择进位 C_N , 置进位/借位标志寄存器 C 和零标志寄存器 Z 的加载控制端 LD , 准备接收结果状态。

(3) 与 ADD 指令的第三节拍相同。

8. 带进位减法指令 SUBC AC , R_i ;

(1) 与第一条指令的第一节拍相同。

(2) 除选择算术减法运算外 , 其余与上条指令的第二节拍相同。

(3) 与 SUB 指令的第三节拍相同。

9. 逻辑与指令 AND AC ,Ri ;

(1) 与第一条指令的第一节拍相同。
(2) 第二个节拍对应的时钟脉冲上升沿将数据总线上的指令码打入指令寄存器 IR 并实现 PC 加 1。指令译码 ,Ri 的编码送通用寄存器组 GR 的 A₀A₁A₂ 端 , 读信号送 GR 的 W/R 端 , 置 ALU 的 S₀S₁S₂S₃ 端和 M 端 , 选择逻辑与运算功能 , 置 ALU 的三态使能 OE 允许结果输出到数据总线 , 置零标志寄存器 Z 的加载控制端 LD , 准备接收结果状态。

(3) 第三个节拍对应的时钟脉冲上升沿将数据总线上的结果打入 AC、零标志打入 Z。

10. 异或指令 XOR AC ,Ri ;

(1) 与第一条指令的第一节拍相同。
(2) 除选择逻辑异或运算功能外 , 其余与 AND 指令的第二节拍相同。
(3) 与 AND 指令的第三节拍相同。

11. 带进位循环右移指令 SHCR AC ,Ri ;

(1) 与第一条指令的第一节拍相同。
(2) 第二个节拍对应的时钟脉冲上升沿将数据总线上的指令码打入指令寄存器 IR 并实现 PC 加 1。指令译码 , 置 ALU 的 S₀S₁S₂S₃ 端和三态使能 OE 允许输出 , 进行数据 R_i 的传送操作 , 置累加器 AC 的 S₀S₁ 端准备右移接收数据 , 置多路选择器 MUX4 的控制端 S₀S₁ 选择 AC₀ , 置标志寄存器 C 的加载控制端 LD 准备接收 AC₀。
(3) 第三个节拍对应的时钟脉冲上升沿将数据右移打入 AC 和 C。

12. 带进位循环左移指令 SHCL AC ,Ri ;

(1) 与第一条指令的第一节拍相同。
(2) 第二个节拍对应的时钟脉冲上升沿将数据总线上的指令码打入指令寄存器 IR 并实现 PC 加 1。指令译码 , 置 ALU 的 S₀S₁S₂S₃ 端和三态使能 OE 允许输出 , 进行数据 R_i 的传送操作 , 置累加器 AC 的 S₀S₁ 端准备左移接收数据 , 置多路选择器 MUX4 的控制端 S₀S₁ 选择 AC₇ , 置标志寄存器 C 的加载控制端 LD 准备接收 AC₇。
(3) 第三个节拍对应的时钟脉冲上升沿将数据左移打入 AC 和 C。

13. 无条件转移指令 JMP mi ;

(1) 与第一条指令的第一节拍相同。
(2) 第二个节拍对应的时钟脉冲上升沿将数据总线上的指令码打入指令寄

存器 IR 并实现 PC 加 1。指令译码 ,控制多路选择器 MUX2 的 AS 端选择 PC ,使 PC 中的地址送存储器 ,读信号和输出使能信号送 W/R 和 OE ,将该指令的操作数 m_i 读到数据总线上。置程序计数器 PC 上的 S_0S_1 ,准备接收 m_i 。

(3) 第三个节拍对应的时钟脉冲上升沿将 m_i 打入 PC。

14. 结果非零转移指令 JNZ m_i ;

(1) 与第一条指令的第一节拍相同。

(2) 第二个节拍对应的时钟脉冲上升沿将数据总线上的指令码打入指令寄存器 IR 并实现 PC 加 1。指令译码 ,控制多路选择器 MUX2 的 AS 端选择 PC ,使 PC 中的地址送存储器 ,读信号和输出使能信号送 W/R 和 OE ,将该指令的操作数 m_i 读到数据总线上。如果 $Z = 1$,则控制程序计数器 PC 上的 S_0S_1 ,准备完成 PC 加 1 的操作 ,否则控制程序计数器 PC 上的 S_0S_1 ,准备接收总线上的 m_i 。

(3) 第三个节拍对应的时钟脉冲上升沿修改 PC 的值。

15. 结果无进位转移 JNC m_i ;

(1) 与第一条指令的第一节拍相同。

(2) 除将 $Z = 1$ 改为 $C = 1$ 外 ,其余与 JNZ 指令的第二节拍相同。

(3) 与 JNZ 指令的第三节拍相同。

16. 停机指令 HLT

(1) 与第一条指令的第一节拍相同。

(2) 第二个节拍对应的时钟脉冲上升沿将数据总线上的指令码打入指令寄存器 IR 并实现 PC 加 1。指令译码 ,产生控制信号 $HLT = 0$ 。

综合上面的分析可以发现 ,最长的指令周期为 3 个节拍 ,最短的指令周期为 2 个节拍 ,可以采用中央控制的方式统一为 3 个节拍 ,也可以在执行时间较短的指令微操作序列中加上对节拍发生器的控制 ,使其在指令执行完毕时重新从第一个节拍开始。

由于节拍宽度与时钟周期相同 ,因此时钟周期的选取有一个基本的原则 ,那就是除访存微操作外 ,必须保证一般的微操作在一个节拍中完成。即有 :

$$T \geq n \times t_d + \triangle$$

其中 , T 为时钟周期 , n 为电路最大的逻辑级数 , t_d 为电路的门级延迟 , \triangle 为信号时间裕量 ,包括电路的线传输延时(3 ns/m)和电路的离散误差 ,在不需要精确分析时 ,一般取 $\triangle = (1/3 \sim 1/4)n \times t_d$ 。

由于“小精灵”计算机的速度没有严格的要求 ,在上面指令系统微操作序列的分析中 我们将访问存储器的操作安排在一个节拍内完成 ,因此其时钟周期必须大于访存周期。

7.3.5 确定微操作时间表与微操作命令逻辑表达式

从图 7.4 所示的“小精灵”计算机硬件逻辑结构简图可以看出,进位/借位标志寄存器 C、零标志寄存器 Z、指令的操作码和节拍是控制器的输入,各种控制信号(微操作命令)是控制器的输出,结合上一小节对指令系统的分析,不难列出微操作命令与输入之间的关系表(微操作时间表),见表 7.4。该表实质上就是组合逻辑控制器输入和输出的真值表,据此可以写出所有控制信号(微操作命令)的逻辑表达式。如:

ALU 的控制端 S_3 只在三种输入下为 0,其余位置为 1 或任意项,因此可以采用或与式的形式,并化简为:

$$ALU \cdot S_3 = P_4 + P_3 + \overline{P_2} + \overline{P_0} + T_1 + \overline{T_0}$$

PC 的控制端 S_0 只在三种输入下为 1,其余位置为 0,因此可以采用与或式的形式,并化简为:

$$PC \cdot S_0 = \overline{P_4} P_3 P_2 \overline{P_1} \overline{P_0} \overline{T_1} T_0 + \overline{P_4} P_3 P_2 \overline{P_1} \overline{T_1} T_0 \bar{Z} + \overline{P_4} P_3 P_2 \overline{P_0} \overline{T_1} T_0 \bar{C}$$

同理,可写出其他微操作命令的逻辑表达式。

注意,为了表达式的化简,可以根据需要对指令操作码进行重新编码。

需要指出的是,图 7.3 给出的是单总线结构的运算器,这种结构的运算器操作速度较慢,但控制比较简单。它在同一时间只能有一个操作数放在总线上,需分两次将操作数放入锁存器 A 和 B,ALU 才能运算,其结果出现在总线上。而双总线结构的运算器由两条总线分别提供 ALU 所需的操作数,因此 ALU 的入口处可不用锁存器,但结果不能马上输出到总线上,须由缓冲器暂存,等运算完成后再送到目的寄存器。三总线结构的运算器由两条总线分别提供 ALU 所需的操作数,一条总线输出运算结果,整个操作可在一步控制中完成,因此速度较快,但控制相对复杂。

习 题

1. 解释下列术语：

机器指令	汇编指令	寻址方式	微操作
节拍	指令周期	机器周期	异步控制方式
同步控制方式	中央控制	局部控制	混合控制

2. 简要说明指令系统设计的基本原则。

3. 通常一条指令的执行过程分几个阶段？每个阶段完成什么功能？

4. 简述用软件方法实现乘除法的基本原理。

5. 控制器的功能是什么？其输入受什么控制？

6. 说明微操作、节拍、指令周期、机器周期和时钟周期之间的关系。

7. 能不能说机器的主频越快 机器的速度就越快 ,为什么？

8. 分别说明控制器组合逻辑设计和微程序设计的特点和优缺点。

9. 简述计算机控制器组合逻辑设计的步骤。

10. 试设计一个简易计算机。

第八章 基于微处理器的计算机设计

8.1 引言

随着半导体微电子技术的高速发展，整个 ALU 和一个控制单元可以放置于一个半导体芯片中，这个高度集成的元件称为微处理器。微处理器、存储器和 I/O 接口等部件之间通过总线连接，就构成了一个完整的具有现代编程设备特性的计算机，如图 8.1 所示。

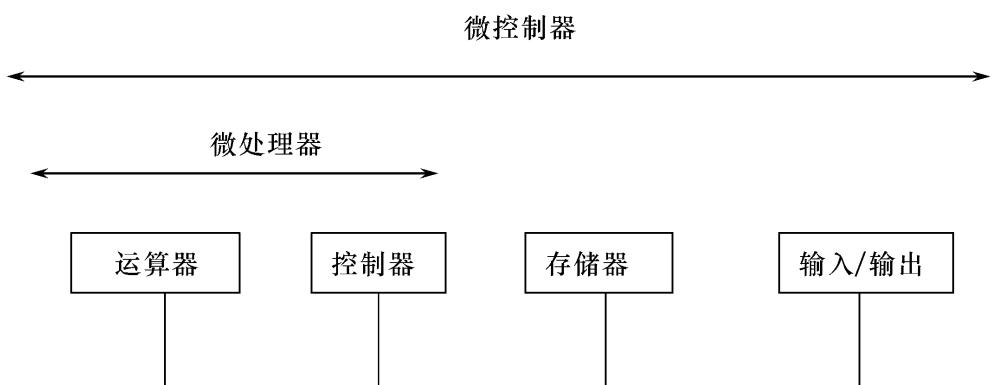


图 8.1 微处理器和微控制器

微型计算机系列系统组成，除了 CPU 之外，还必须有存储器、接口电路、外部设备、系统总线及软件等部分。存储器及各类外设都是通过各自的接口电路连接到微机系统的总线上的，也就是说，接口是 CPU 和外界连接的部件，是 CPU 和外界交换信息的通道。随着超大规模集成电路工艺的飞速发展，各种专用、通用接口芯片应运而生，为微机应用的发展奠定了坚实的硬件基础。

8.2 Intel 8086 微处理器

Intel 8086/8088 为代表的 16 位微处理器是第三代产品，以它们为核心部件组成的微机系统，其性能已达到中、高档小型计算机的水平。Intel 系列 CPU 一直占着主导地位，尽管其后续的 80286、80386、80486 以及 Pentium 系列 CPU 结构

与功能同 8086/8088 相比已经发生很大变化 ,但从基本概念与结构以及指令格式上讲 ,它们仍然是经典的 8086/8088CPU 的延续与提升。并且 ,其他系列流行的 CPU(如 AMD 公司的 6X86MX/MII 等)也与 80x86CPU 兼容。

8.2.1 8086/8088 微处理器结构

Intel 系列的 16 位微处理器 ,是 iAPX86/88 系列微机的基础。它采用高速运算性能的 HMOS 工艺制造 芯片上集成有 2.9 万个晶体管 ,用单一的 +5 V 电源和 40 条引脚的双列直插式封装 ;时钟频率为 5 MHz ~ 10 MHz ,最快的指令执行时间为 $0.4 \mu s$ 。 8086 有 16 根数据线和 20 根地址线 ,可以处理 8 位或 16 位数据 ,寻址 1 MB 的存储单元和 64 KB 的 I/O 端口。它的主机设计较之 8 位机的性能大约提高了 10 倍。

如图 8.2 所示 ,可以看出 8086CPU 由执行部件 EU(Execution Unit)和总线接口部件 BIU(Bus Interface Unit)两部分组成。

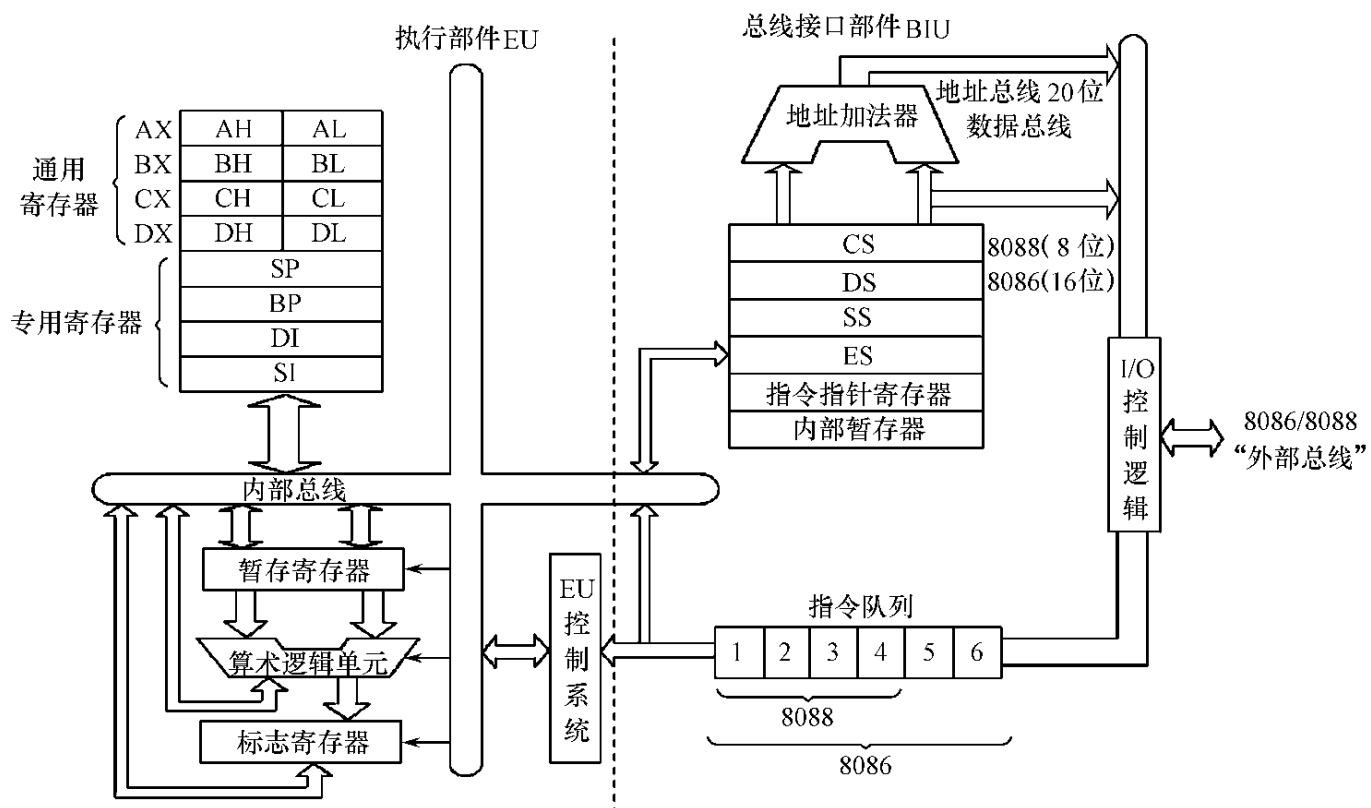


图 8.2 8086/8088CPU 内部结构示意图

1. 执行部件(EU)

执行部件是负责指令执行的部件。

执行部件由通用寄存器组、专用寄存器组、算术逻辑单元(ALU) 标志寄存器(FR)和内部控制逻辑(EU)五部分组成。

(1) 通用寄存器组

8086 共有 4 个 16 位的通用寄存器 , 分别记为 AX、BX、CX、DX 。这 4 个通用寄存器既可用作 16 位寄存器也可用做 8 位寄存器。用做 8 位寄存器时分别记为 AH、AL、BH、BL、CH、CL、DH、DL。

AX(AH、AL) : 累加器。有些指令约定 AX(或 AL) 寄存器为累加器 , 如乘法、除法、输入 / 输出等指令。实际上大多数情况下 , 所有通用寄存器均可充当累加器。

BX(BH、BL) : 基址寄存器。可用作间接寻址的地址寄存器和基址寄存器 , BH、BL 可用作 8 位通用数据寄存器。

DX(DH、DL) : 数据寄存器。除用作通用寄存器外 , 在 I/O 指令中用作口地址寄存器 , 乘除指令中用作辅助累加器。

(2) 专用寄存器组

8086CPU 除有 4 个 16 位的通用寄存器外 , 还有 4 个 16 位专用寄存器 , 分别是 : 基数指针寄存器 BP(Base Pointer Register) , 堆栈指针寄存器 SP(Stack Pointer Register) , 源变址寄存器 SI(Source Index Register) 和目的变址寄存器 DI(Destination Index Register) 。

BP、SP 常用来指示相对于段起始地址的偏移量。 BP 一般用于访问堆栈段任意单元 , SP 用于访问堆栈段栈顶单元。 SI、DI 可用作寄存器间接寻址、相对寻址、基址变址寻址、相对基址变址寻址寄存器 , 访问数据段任意单元。

(3) 算术逻辑单元(Arithmetic Logic Unit)

算术逻辑单元完成 16 位或 8 位算术逻辑运算。

(4) 标志寄存器(Flag Register)

标志寄存器共有 16 位 , 基保 7 位未用。

(5) 内部控制逻辑电路

从指令队列缓冲中取出指令 , 进行译码 , 产生各种控制信号 , 控制各部件的工作。

2. 总线接口部件(BIU)

总线接口部件负责 CPU 与存储器、输入 / 输出设备之间的数据传送 , 包括对存储器读 / 写数据操作、对 I/O 接口的读 / 写操作以及取指令操作。

总线接口部件由段寄存器(CS、DS、SS、ES) 指令指针寄存器(IP) 地址加法器、内部暂存器、指令队列缓冲器及 I/O 控制逻辑等部分组成。

(1) 段寄存器

8086CPU 内部数据结构是 16 位的 , 即所有的寄存器都是 16 位的 , 而外部寻址空间为 1 MB , 即需要 20 位地址线。为了能寻址 1 MB 空间 , 将 1 MB 空间以 16

字节为一个内存节(Paragraph),共分成 64 K 个节。每个节用一个段地址来标识。用于存放段地址的寄存器称为段寄存器 ,根据其主要用途 ,设有 4 个段寄存器 : CS 代码段寄存器(Code Segment Register),DS 数据段寄存器(Data Segment Register),SS 堆栈段寄存器(Stack Segment Register),ES 附加段寄存器(Extra Segment Register)。

(2) 地址加法器

用于产生 20 位物理地址。

(3) 指令队列缓冲器

8086 有 6 字节缓冲器 8088 有 4 字节缓冲器。

(4) I/O 控制逻辑(总线控制逻辑)

输入/输出控制电路控制 CPU 与外部电路的数据交换。

(5) 内部暂存器

用于内部数据的暂存 ,用户无权访问。

(6) 指令指针寄存器

又称指令计数器 ,表征和计算指令的序号。

8.2.2 8086 微处理器的总线周期

计算机是由一串脉冲控制进行工作的 称为计算机的时钟 ,每个脉冲称为一个时钟周期或一个 T 状态 ,它是由计算机的主频决定的 ,如 8086 - 1 主频为 10 MHz ,一个时钟周期为 100 ns。若干个时钟脉冲完成一个基本操作 ,一种基本操作为一个总线周期。CPU 有若干种典型操作 ,构成相应的总线周期。如存储器读总线周期、存储器写总线周期、I/O 读总线周期、I/O 写总线周期等。完成一条指令所需的时间称为指令周期 ,通常需要若干个时钟周期 ,如总线操作指令 ,可能需要多个总线周期。

在 8086 中 ,一个最基本的总线周期一般由 4 个时钟周期组成 ,或者称为 T_1 状态、 T_2 状态、 T_3 状态、 T_4 状态。有时根据需要在 T_3 与 T_4 之间插入若干个等待状态 T_w 如图 8.3 所示。

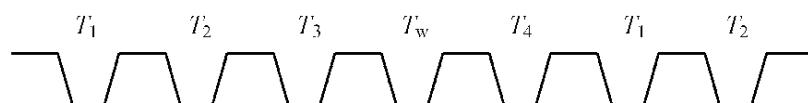


图 8.3 8086 典型的总线周期时序

T_1 状态 :CPU 向多路复用总线上发送地址信息 ,指出要寻址的内存单元地址或 I/O 端地址。这期间 CPU 还要送出地址锁存信号 ALE(正向脉冲) ,在 ALE 的下降沿将内存单元地址或 I/O 端口地址打入地址锁存器。

T_2 状态 :CPU 从总线上撤销有效地址 ,使地址总线低 16 位呈高阻状态 ,为数据传输作准备。总线高 4 位($A_{19} \sim A_{16}$)输出总线周期的状态信息 ,用以表示中断允许状态及正在使用的段寄存器名等。

T_3 状态 : $A_{19} \sim A_{16}$ 上状态信息不变 ,地址总线低 16 位上出现 CPU 要写出的数据或准备读入的数据。若外设或内存来不及与总线交换数据 ,以便在 T_4 状态下结束该总线周期 ,则应通过 CPU 的 READY 信号 ,在 T_3 前沿(下降沿)之前向 CPU 申请插入等待状态 T_w 。在 T_3 及 T_w 的前沿查询 READY 信号 ,查到高电平则结束等待状态 ,进入下一状态。否则继续插入等待状态 T_w 。

T_4 状态 :总线周期结束。

8.2.3 8086 的引脚与功能

8086 芯片为 40 引脚双列直插封装 ,图 8.4 为 8086 的引脚信号图。为解决功能与引脚之间的矛盾 ,许多引脚具有多功能。其实现方法一种是总线复用 ,即在不同时钟周期内 ,引脚的作用不同 ,如地址、数据线的分时复用 ;另一种是按工作模式的不同确定引脚的功能。

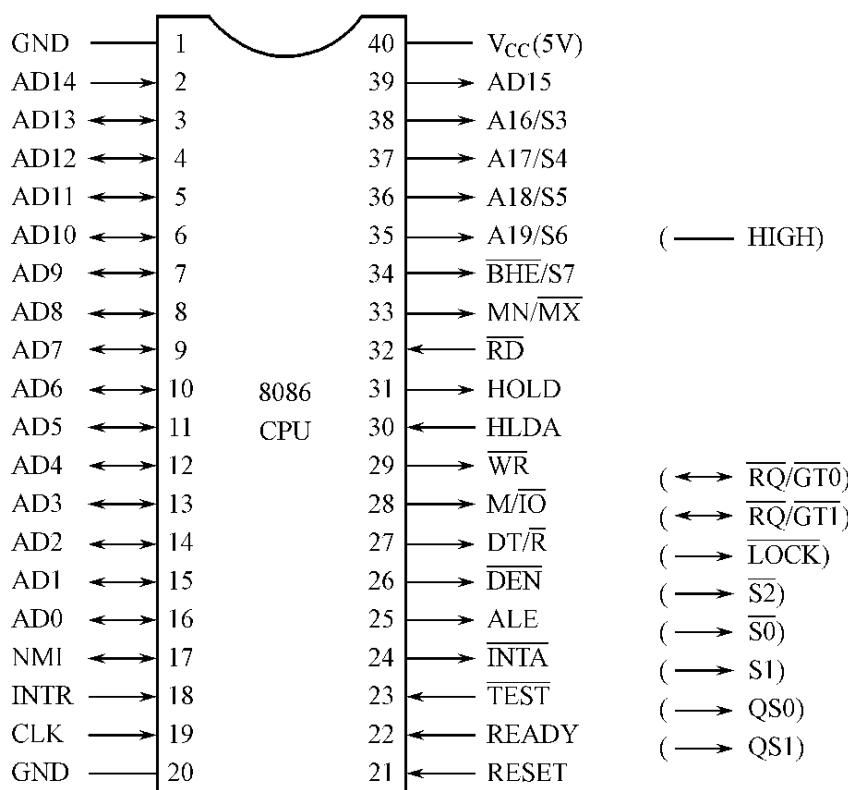


图 8.4 8086/8088 引脚分配图(括号中为最大模式名称)

8086CPU 的引脚功能分别介绍如下。

(1) GND(地)和 V_{CC} (电源)

第 1、20 脚接地 ,第 40 脚接 +5 V 电源。

(2) $AD_{15} \sim AD_0$ 地址/数据复用总线引脚(双向)

8086 由于采用地址、数据总线复用方式,所以才可能在 40 引脚的条件下提供 20 位地址总线,16 位数据总线, $AD_{15} \sim AD_0$ 在 T_1 状态输出地址信号,在 T_2 、 T_3 状态若为总线读周期则总线浮空,若为总线写周期,则总线上为输出数据。

(3) $A_{19}/S_6 \sim A_{16}/S_3$ 地址/状态复用引脚(输出)

在总线周期的 T_1 状态, $A_{19} \sim A_{16}$ 输出最高 4 位地址信号,在 T_2 、 T_3 、 T_w 、 T_4 状态 $S_6 \sim S_3$ 输出状态信息,其意义如下。

- S_6 为 0 表示 8086CPU 占用总线。
- S_5 表明中断允许标志 IF 的设置情况。若 $IF = 1$, 则为允许可屏蔽中断 $S_5 = 1$; 若 $IF = 0$, 则为禁止可屏蔽中断 $S_5 = 0$ 。
- S_4 、 S_3 指明正在使用的段寄存器。

(4) $\overline{BHE}/S7$ 高 8 位数据线允许/状态复用引脚(输出)

(5) NMI 非屏蔽中断(输入)

该信号为一个由低到高的上升沿有效信号,不受中断允许标志 IF 的影响。一旦产生 NMI 信号,CPU 处理完一条指令后,立即进入非屏蔽中断处理。

(6) INTR 可屏蔽中断(输入)

高电平有效。CPU 在每条指令的最后一个时钟脉冲对 INTR 信号进行采样,当 INTR 为高电平时,如果 CPU 的中断允许标志 $IF = 1$,则在一条指令结束后即响应中断请求,进入中断处理。

(7) \overline{RD} 读信号(三态输出)

低电平有效。在 CPU 执行读操作时, \overline{RD} 在 T_2 、 T_3 、 T_w 期间有效。到底是读存储器还是读 I/O 端口取决于 M/\overline{IO} 信号。 M/\overline{IO} 为高时,读存储器; M/\overline{IO} 为低时,读 I/O 端口。

(8) CLK 时钟信号(输入)

8086 要求时钟的占空比为 1/3(一个周期中 1/3 为高电平,2/3 为低电平)。CPU 的所有操作均是在时钟同步下进行的。

(9) RESET 复位信号(输入)

(10) READY 准备好信号(输入)

(11) \overline{TEST} 测试信号(输入)

低电平有效。 \overline{TEST} 和 WAIT 指令配合使用,执行 WAIT 指令。 \overline{TEST} 与 WAIT 配合可以实现 CPU 与外设同步工作。

(12) MN/\overline{MX} 最小/最大模式控制信号(输入)

该引脚接 +5 V 时 CPU 工作在最小模式,该信号接地时 CPU 工作于最大模式。

8.2.4 8086 的存储器及 I/O 组织

1. 8086 存储器的组织

8086CPU 有 20 条地址线 , 存储器地址的编址范围是 00000H ~ FFFFFH , 共 1 MB 的存储空间。但 8086 内部的寄存器都是 16 位的 , 显然无法直接对 1 MB 的内存空间进行寻址 , 因而引入了分段的概念。

引入分段后 , 一个存储器单元的地址就可以用段地址和偏移量两部分来表示 , 段地址由段寄存器(CS、DS、SS、ES)提供 , 偏移量由 IP、SP、BP、SI、DI、BX 等寄存器提供 , 不同的指令有不同的组合方式。计算一个存储单元的物理地址时 , 先将其段寄存器的内容左移 4 位(相当于乘十进制数 16) , 得到一个 20 位的值 , 然后加上 16 位的偏移量。物理地址的计算公式如下 :

$$\text{物理地址} = \text{段地址} \times 16 + \text{偏移量}$$

段地址的引入 , 为程序在内存中浮动创造了条件。因为一般用户程序只涉及偏移地址 , 段地址在程序装入内存时由操作系统分配 , 所以一个程序可在内存中任何一个逻辑段(64 KB 空间) 中运行。

同一物理地址可以由不同的段地址和偏移量表示。

2. 8086 的 I/O 组织

8086 系统中 , 对外部设备的端口编址采用了与存储器非统一编址方式 , 即有专门的输入指令(IN)和输出指令(OUT) , 用于对外设端口的寻址。外部设备端口的地址空间为 64 KB 地址范围 0000 – FFFFH 。两个编址相邻的 8 位端口可以组成一个 16 位端口。指令系统中既有读 / 写 8 位端口的输入 / 输出指令 , 也有 16 位端口的操作指令。

8.3 8086 最小系统组成与总线周期波形

图 8.5 是按最小方式组成的计算机系统示意框图 , 它们由时钟发生器 , 地址锁存器、数据驱动器、外设、内存储器、中断控制器和 8086 微处理器组成。

最小系统工作波形主要介绍存储器读总线周期波形、存储器写总线周期波形以及带 T_w 的存储器写总线周期波形。

1. 存储器读总线周期波形

存储器读总线周期是 8086 从存储器读取指令和数据的操作过程 , 该操作的地址、数据和控制信号的时序关系如图 8.6 所示 , 一般一个存储器读总线周期需

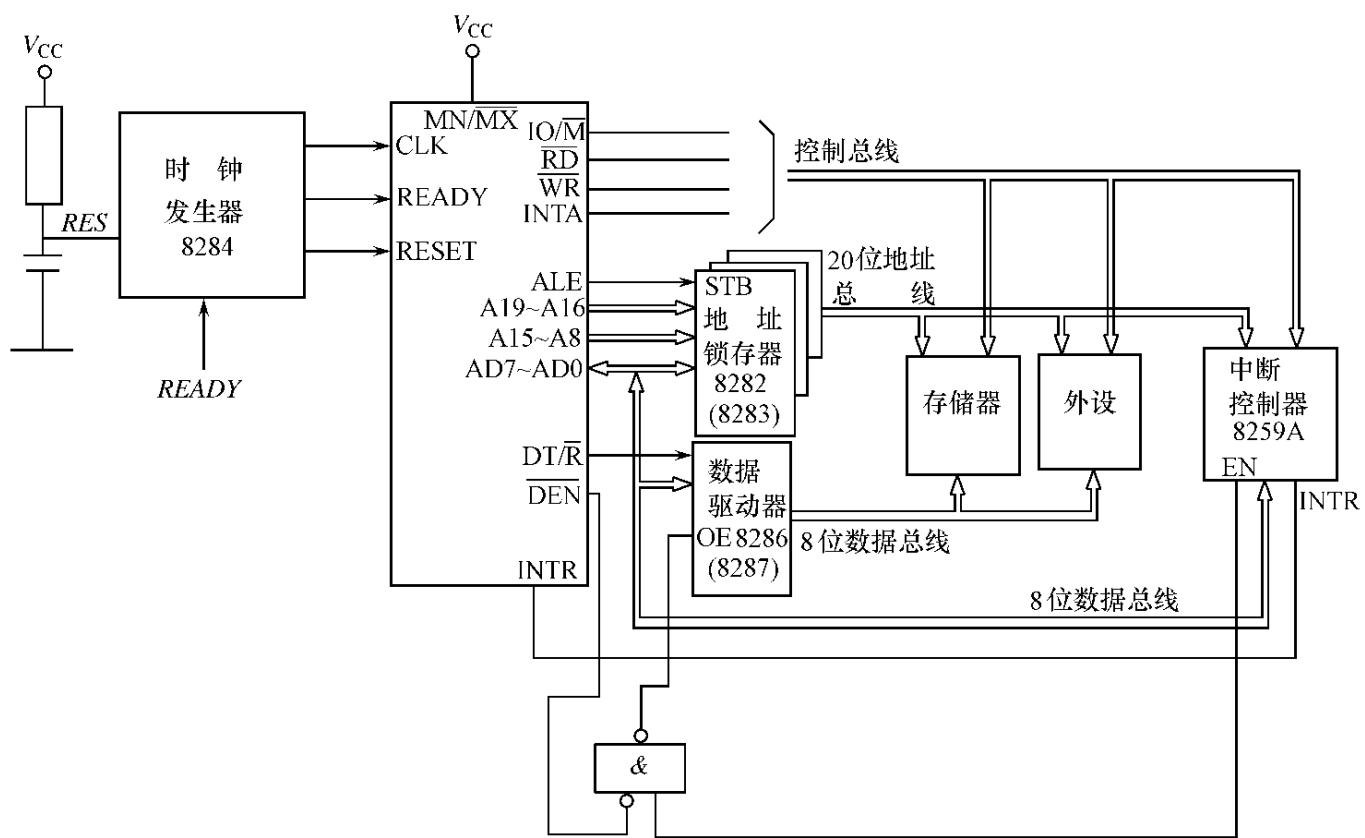


图 8.5 8086 最小方式组成系统框图

要四个时钟周期 T 即 $T_1 \sim T_4$,从 T_1 开始到 T_4 结束。

(1) IO/M : IO/M 有效时间为前一周期 T_4 上升沿至本周期的 T_4 上升沿 , 现为存储器读 , $IO/M = 0$ 。

(2) $A19(S6) \sim A16(S3)$: 在 T_1 下降沿至 T_2 下降沿期间 , 它输出地址码最

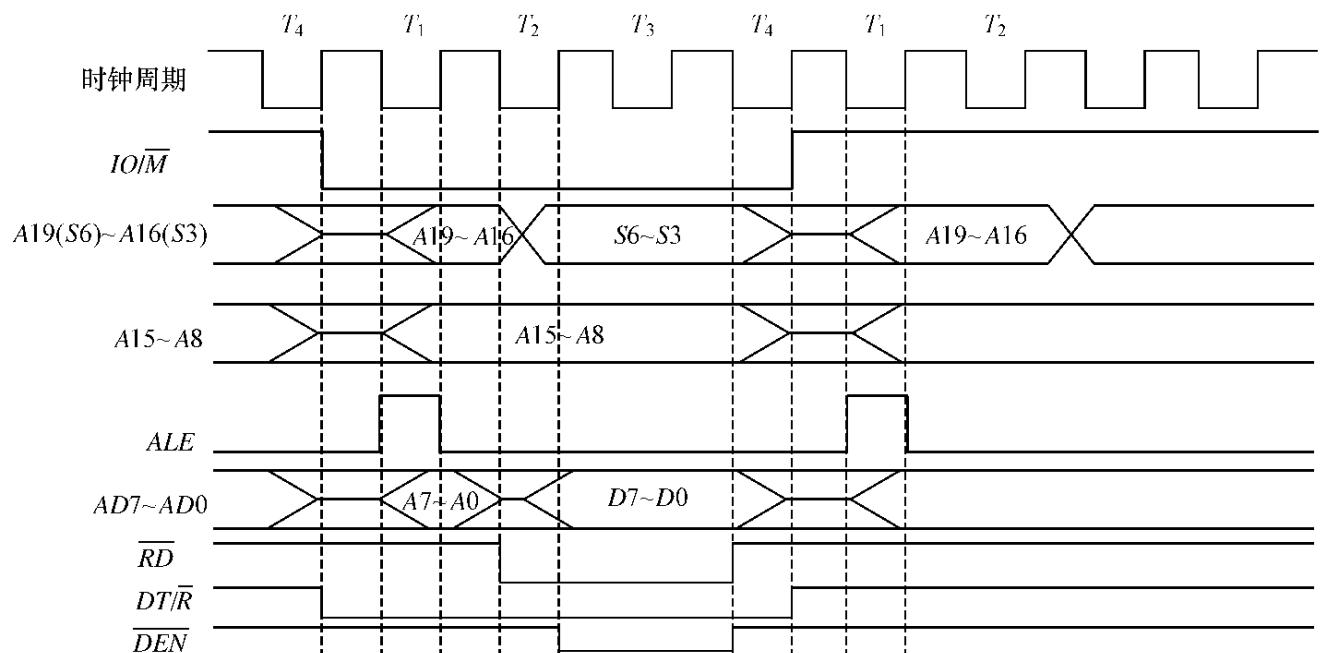


图 8.6 存储器读总线周期波形

高 4 位 $A_{19} \sim A_{16}$,而在 T_2 下降沿至 T_4 上升沿,它输出状态信号 $S_6 \sim S_3$ 。由于 $A_{19} \sim A_{16}$ 在 T_2 下降沿后就消失了,因此必须加地址锁存器将其保存起来。

(3) $A_{15} \sim A_8$: 在 T_1 下降沿至 T_4 上升沿期间输出地址码 $A_{15} \sim A_8$,由于在整个读周期此地址始终存在,可以不用锁存。

(4) ALE : 地址锁存允许信号,在 T_1 下降沿至 T_1 上升沿有效,锁存器用其下降沿将地址码 $A_{19} \sim A_{16}, A_7 \sim A_0$ 打入锁存器,予以保存。

(5) $AD_7 \sim AD_0$: 为地址数据共用信号线,在 T_1 下降沿至 T_2 下降沿输出地址码 $A_7 \sim A_0$,在 T_2 以后该线悬浮,准备接收由存储器读出的信号。

(6) \overline{RD} : 读命令 \overline{RD} 的有效时间为 T_2 下降沿到 T_4 下降沿。为了确保 $AD_7 \sim AD_0$ 输出地址 $A_7 \sim A_0$ 后能进入悬浮状态,读命令 \overline{RD} 发出的时间较 T_2 下降沿稍滞后一点,在地址和读命令 \overline{RD} 的联合作用下,数据在 $T_2 \sim T_4$ 之间出现在 $AD_7 \sim AD_0$ 线上,在 T_4 的下降沿将其读入 8088。

(7) DT/\overline{R} : DT/\overline{R} 有效时间是前一总线周期 T_4 上升沿到本周期 T_4 上升沿,由于是读存储器操作,数据由外进入 8088, $DT/\overline{R} = 0$ 。

(8) \overline{DEN} : 数据允许信号在 T_2 上升沿至 T_4 下降沿之间有效,在此期间数据驱动器是工作的。

如果存储器在 T_4 下降前不能读出数据,可以通过 $READY$ 向 8088 发一未准备好信号 8086 检测到此信号后,在 T_3 之后插入一个等待时钟周期 T_W 。

$READY$ 信号由存储器或专门电路提供,加到 8086,8086 在 T_3 和 T_W 的下降沿检测 $READY$ 信号,若 $READY = 1$,则由 T_3 转入 T_4 , $READY = 0$,表示数据未准备好,在时钟周期 T_3 之后,8088 自动插入一个等待时钟周期 T_W ,在 T_W 的下降沿继续检测,发现 $READY = 1$ 就结束等待,进入 T_4 。原则上,插入 T_W 的个数不限,插入 T_W 后,各控制信号相应予以延长。

2. 存储器写总线周期波形

存储器写周期是 8086 向存储器输出数据周期,其各控制信号波形如图 8.7 所示。该图是一带 T_W 的存储器写总线周期波形图,图中与读周期相同的信号就不再重复,下面仅就其不同点作一点介绍。

(1) $AD_7 \sim AD_0$: 在写周期,输出地址码 $A_7 \sim A_0$ 之后,不是总线变为悬浮状态,而是在 T_2 下降沿到 T_4 上升沿期间输出欲写入存储器的数据。

(2) \overline{WR} : 这是写命令信号,它有效时间为 T_2 下降沿到 T_4 下降沿,因为在 T_2 下降沿之前地址已输出,在 T_2 下降沿至 T_4 下降沿数据已送上数据总线,可以用 \overline{WR} 控制写入存储器。

(3) \overline{DEN} : 数据驱动器允许信号在 T_1 上升沿至 T_4 上升沿有效。图中未画

出 DT/R 波形，在存储器写时，它为高电平。

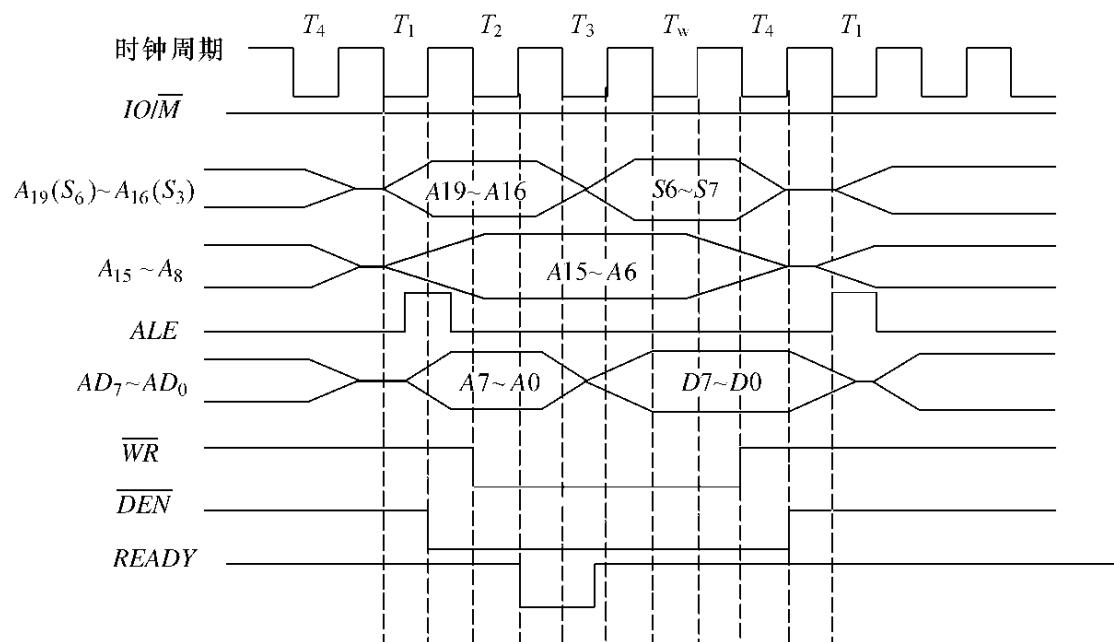


图 8.7 带 T_w 的存储器写周期波形图

第九章 接口与通信

9.1 接口的基本概念及基本技术

输入/输出设备是计算机系统的重要组成部分。程序、原始数据和各种现场采集到的信息要通过输入装置输入到计算机;计算结果和各种控制信号要输出给各种输出装置,以便显示、打印和实现各种控制。外设的接口通常做成扩展板插入总线插槽中,以实现各种功能。

9.1.1 接口的概念

所谓接口就是指 CPU 和存储器、外部设备或者两种外部设备、两种机器之间通过系统总线进行连接的逻辑部件(或称电路),它是 CPU 与外界进行信息交换的中转站。源程序、原始数据通过接口从输入设备(例如键盘)送入,运算结果通过接口向输出设备(显示器、打印机)送出去;控制命令通过接口发出去(例如步进电机)。现场信息通过接口取进来(例如温度值、转速值等)。要使计算机外部设备正常工作,就要设计正确的接口电路,编制相应的软件,因此,接口技术是采用硬件与软件相结合的方法,研究微处理器如何与外部设备进行最佳耦合与匹配,以实现 CPU 与外界高效且可靠的信息交换的一门技术。CPU 与外设之间的接口一般具有以下几个功能:数据的寄存和缓冲功能;设备选择功能;信号转换功能;对外设的控制和监测功能;中断或 DMA 管理功能;可编程功能等。

9.1.2 接口信息

CPU 与输入/输出设备之间传送的信息,通常包括数据信息、状态信息和控制信息。

1. 数据信息(Data)

微机中的数据通常为 8 位、16 位或 32 位。它大致包括数字量、模拟量、开关量三种类型。

2. 状态信息(Status)

状态信息反映了当前外设所处的工作状态。

数据输入时,CPU 常要先查询输入设备的信息是否准备好(Ready),当准备好时才传送。输出时常要查询输出装置是否空闲(Empty),如空闲则接收 CPU 送来的信息,否则 CPU 等待。

3. 控制信息(Control)

控制信息是 CPU 通过接口传送给外设的,CPU 通过发送控制信息,控制外设的工作。

9.1.3 输入/输出传送方式

CPU 与外设之间传送数据的方式大致分为以下几种:

(1) 无条件传送方式。这是最简单的一种传送方式,适合于外设总是处于准备好的状态。

(2) 查询传送方式。查询传送方式就是在传送前先要查询一下外设的状态,当外设准备好了才传送,若未准备好,则 CPU 等待。查询式输入程序流程图如图 9.1 所示。

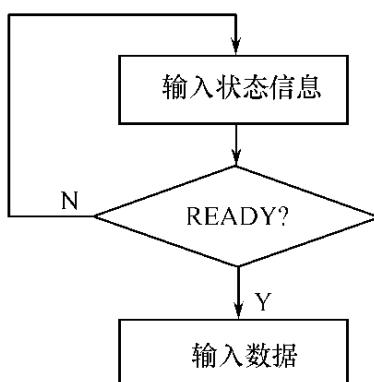


图 9.1 查询式输入程序流程图

(3) 中断传送方式。在查询传送方式中,CPU 要不断地询问外设,当外设未准备好时,CPU 就得等待,从而浪费了 CPU 大量的时间。采用中断方式则可以免去 CPU 的查询等待时间,提高 CPU 的工作效率。在中断方式下,当外设没有准备好时,CPU 可以去做其他的工作。

(4) 直接存储器存取方式(DMA)。中断传送方式相对于查询传送方式来说,大大提高了 CPU 的利用率,但是中断传送方式仍然是由 CPU 通过指令来传送的。在中断时,CPU 要进行保护断点、保护现场,传送数据、存储数据以及最后

恢复现场、返回主程序等操作,需要执行多条指令,大大影响了速度。DMA 传送方式即在 DMA 控制下,外设与内存之间直接进行数据交换而不通过 CPU。这样数据传送的速度上限将主要取决于存储器的存取速度。

9.1.4 可编程定时器/计数器芯片 8253

IBM – PC/XT 中许多部分需要用到定时/计数功能,如动态存储器刷新、磁盘驱动器等,为了支持这些功能,系统板上设置有定时/计数系统,其核心元件是 8253 可编程定时/计数器。

1. 8253 的功能

8253 的基本功能是能在软件的控制下产生一系列准确的时间延迟,程序员用不着在系统软件中设置定时循环,而只要适当地设置 8253 的参数即能达到要求。8253 还具有其他定时器/计时器功能,例如,可编程频率发生器、事件计数器、二进制频器、实时时钟、数字单稳、复杂的电机控制器等。

2. 8253 的内部结构和引脚功能

8253 具有三个功能相同的 16 位减法计数器 0 号、1 号和 2 号,可进行二进制或二—十进制计数或定时操作,工作方式和计数常数可由软件编程来选择。最高计数时钟频率为 2.6 MHz,可以方便地与 PC 总线连接,其内部结构和外部引脚如图 9.2 所示。每个计数器有三个引脚:*CLK* 为时钟输入线,在计数方式时是计数脉冲输入端;*OUT* 为计数器输出端,当计数器减为零时,根据所置的工作方式输出相应信号;*GATE* 为门控信号,用于启动或禁止计数器操作。控制字寄

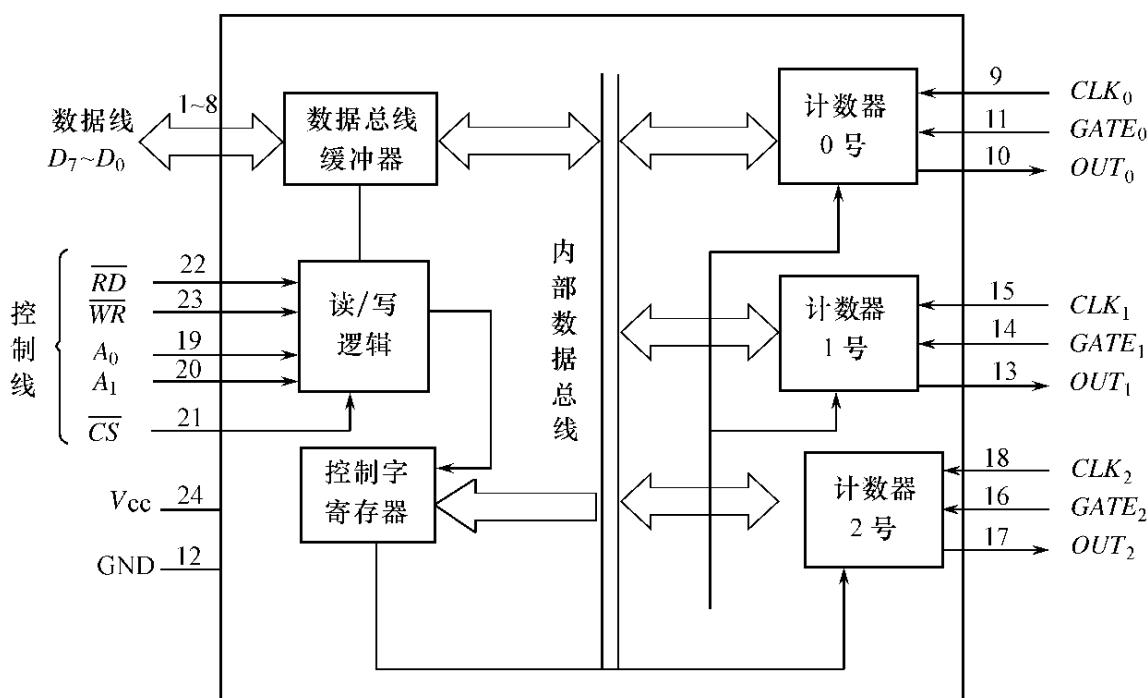


图 9.2 8253 的内部结构及引脚

存器用来寄存工作方式控制字 ,只能写入不能读出。

8253 与 PC 总线的接口线共 15 条。 $D_0 \sim D_7$ 为双向三态数据总线 ,是 PC 总线与 8253 之间的数据传输线 ; \overline{RD} 、 \overline{WR} 为数据读、写控制线 ,低电平有效 ; \overline{CS} 是片选线 ,通常接地址译码器输出 ; A_0 、 A_1 是地址选择线 ,四种组合分别选择三个计数器和控制字寄存器。 A_0 、 A_1 和 \overline{CS} 一起确定 8253 的地址。 8253 的计数通道及操作地址分配如表 9.1 所示。

表 9.1 8253 计数通道及地址分配

\overline{CS}	\overline{RD}	\overline{WR}	A_1	A_0	操作
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(禁止读)
0	1	0	0	0	计数常数写入计数器 0
0	1	0	0	1	计数常数写入计数器 1
0	1	0	1	0	计数常数写入计数器 2
0	1	0	1	1	写入方式控制字
1	×	×	×	×	禁止(三态)
0	1	1	×	×	不操作

3. 8253 的控制字

8253 的工作方式是由主机编程设定的。将给定的工作方式控制字写入控制寄存器 就可以使 8253 某通道按给定的方式工作。控制字的定义如图 9.3 所示。

8253 控制字寄存器是 8 位的。最高两位 SC_1 和 SC_0 用于选择计数器。因为 3 个计数器是完全独立的 ,所以需要有 3 个控制字寄存器来存放。但是控制字寄存器地址是惟一的 ,即 $A_1A_0 = 11$ 对应的地址。因此 , SC_1 、 SC_0 一方面选择了计数器 ,同时也指明了该控制字将写入所选择的计数器的控制寄存器中。

操作类型位(RL_1 、 RL_0)规定了数据读/写格式。当 $RL_1 RL_0 = 00$ 时是计数值锁存操作 ,用在计数过程中读计数值时 ,先送出锁存命令 ,再读取计数值。其他三种组合规定了读/写格式。

工作方式位(M_2 、 M_1 、 M_0)用来指定所选择计数器的工作方式。 8253 共有 6 种工作方式 ,将在下面逐一介绍。

计数类型位(BCD)用以确定计数是采用二进制还是二—十进制。

MOV AL, 76H



图 9.3 8253 的控制字定义

OUT 53H, AL 53H 是某 8253 的控制字寄存器地址

以上两条指令执行后, 它规定 8253 的 1 号计数器以方式 3 工作, 用二进制计数, 读/写格式是先读或写低字节后读或写高字节。

4. 8253 的工作方式与操作时序

(1) 方式 0(计数结束中断方式)

采用 0 方式时, 计数器在减为零时使输出端 OUTPUT 变为高电平, 向 CPU 发出中断申请, 如图 9.4 所示。

当方式控制字写入后, 输出端 OUTPUT 变为低电平; 计数常数写入后, 计数器开始减 1 计数并且维持 OUTPUT 为低电平, 当计数器减为零时, 输出端 OUTPUT 变为高电平。如果 OUTPUT 端接 INTR 中断请求输入端, 则向 CPU 发出中断申请, 直至 CPU 写入新的控制字和新的计数值为止。若在计数过程中 GATE 出现低电平, 则暂停计数。在计数过程中可以改变计数值。若是 8 位计数, 在写入新的计数值后, 计数器将按新的计数值重新开始计数。如果是 16 位计数, 在写入第一字节后计数器停止计数, 在写入第二字节后, 计数器按新的数值开始计数, 即改变计数值是立即有效的。

(2) 方式 1(可编程单稳态)

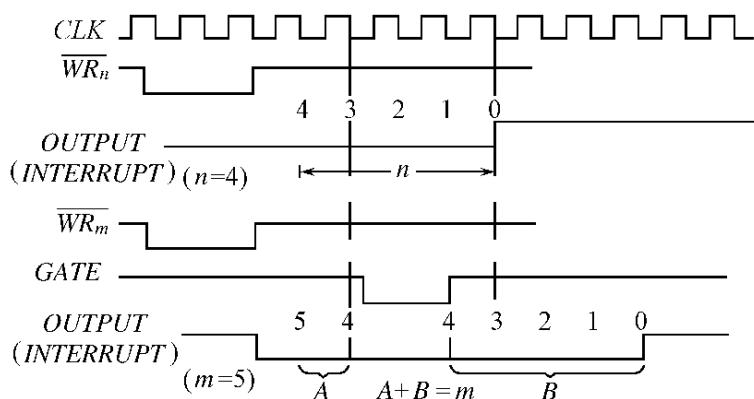


图 9.4 方式 0 的操作时序图

方式 1 可以输出一个宽度可程控的负脉冲。当 CPU 写入控制字后，OUTPUT 即变高，但是当 CPU 写入计数常数后，计数器并不开始计数，而要到外部门控脉冲 GATE 启动之后的下一个 CLK 输入脉冲的下降沿才开始计数；这时输出 OUTPUT 变低，直至计数到 0，输出 OUTPUT 再变高。若外部 GATE 再次触发，则将再产生一个负脉冲，如图 9.5 所示。如果在输出保持低电平期间，写入一个新的计数值，不会影响低电平的持续时间，只有当下一个触发脉冲（GATE）到来时，才使用新的计数值。如果计数尚未结束又出现新的触发脉冲，则从新的触发脉冲之后的 CLK 下降沿开始重新计数，因此使输出负脉冲加宽。CPU 在任何时候都可读出计数器的内容，对单拍脉冲的宽度没有影响。

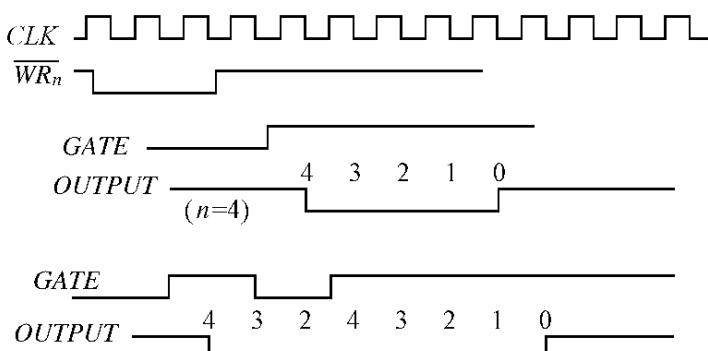


图 9.5 方式 1 操作时序

(3) 方式 2(频率发生器)

采用方式 2 时能在 OUTPUT 端输出连续的负脉冲，其宽度等于一个时钟周期。脉冲周期等于写入计数器的计数值和时钟周期的乘积。由图 9.6 可以看出，CPU 送出控制字后输出将变为高，在写入计数值后，计数器对输入时钟 CLK 计数，直至计数器减至 1 时，输出变低，经过一个时钟周期输出恢复为高，计数器从初值开始重新计数。计数过程受门控脉冲 GATE 控制，GATE 变低时停止计数；

在 *GATE* 变高后的下一个 *CLK* 脉冲使计数器恢复初值 ,重新计数。在计数过程中改变计数值 ,对正在进行的计数过程没有影响 ,但计数到 1 ,*OUTPUT* 变低一个 *CLK* 周期后 ,计数器将按新的计数值计数。

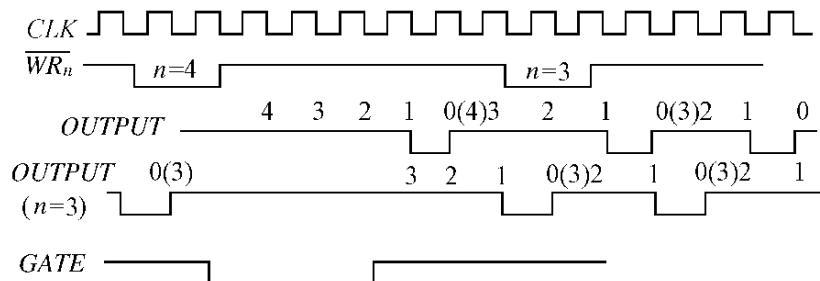


图 9.6 方式 2 的操作时序

(4) 方式 3(方波发生器)

采用方式 3 时 ,*OUTPUT* 端输出方波 ,当计数值 n 为偶数 ,则输出对称方波 ,前 $n/2$ 计数期间 *OUTPUT* 输出高电平 ,后 $n/2$ 计数期间输出低电平 ;当 n 为奇数 ,则前 $(n+1)/2$ 计数期间输出高电平 ,后 $(n-1)/2$ 计数期间输出低电平 ,其余同方式 2。方式 3 操作如图 9.7 所示。

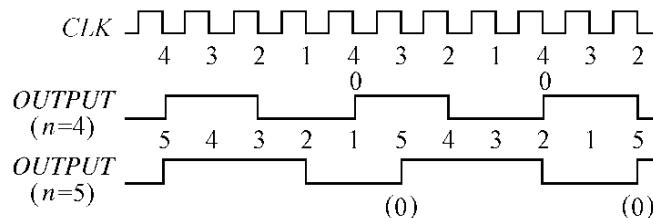


图 9.7 方式 3 的操作时序

(5) 方式 4(软件触发选通)

当方式 4 控制字写入 8253 后 ,*OUTPUT* 输出即变高 ,写入计数值后开始计数 (相当于软件启动) ,当计数到零时输出一个时钟周期的负脉冲 ,计数器停止计数。这种方式计数是一次性的。只有输入新的计数值才开始新的计数。在计数期间 ,如果写入新的计数值 ,将影响下一个计数周期 (对本次无影响)。当门控信号 *GATE* 输入低电平时 ,计数停止 ;*GATE* 恢复为高电平时继续计数。方式 4 操作如图 9.8 所示。

(6) 方式 5(硬件触发选通脉冲)

此方式类似于方式 4 ,所不同的是 *GATE* 端输入信号的作用不同。按方式 5 工作时 ,由 *GATE* 输入触发脉冲 ,从其上升沿开始 ,计数器作减 1 计数 ,计数结束时 ,在 *OUTPUT* 端输出一个宽度等于一个计数脉冲周期的负脉冲。在此方式中 ,

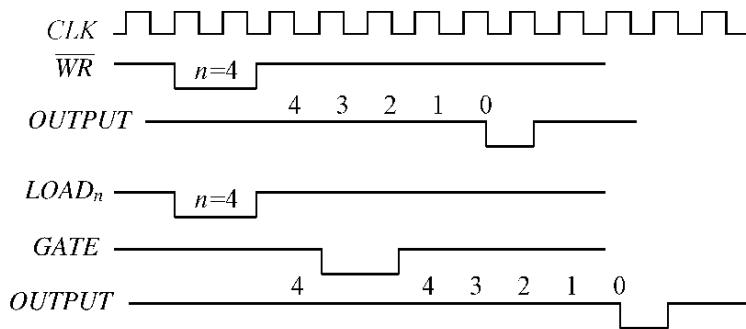


图 9.8 方式 4 的操作时序

计数器可重新触发。在任何时刻，当 **GATE** 触发脉冲上升沿到来时，将把计数初值重新送入计数器，然后开始计数过程。图 9.9 是方式 5 的时序图。

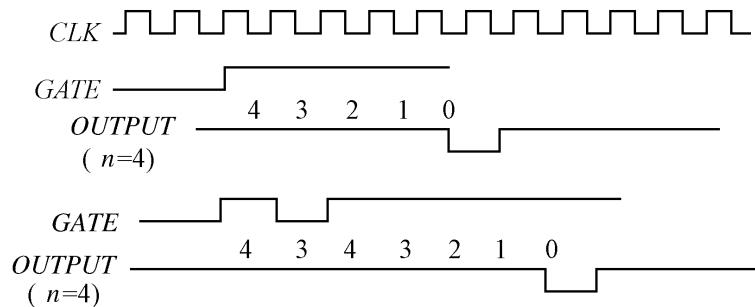


图 9.9 方式 5 的操作时序

以上分别说明了 8253 的 6 种工作方式的工作过程，表 9.2 列出了各种工作方式中门控制信号 **GATE** 的控制作用。

表 9.2 GATE 信号控制功能

信号状态方式	低电平或负跳变	正跳变	高电平
0	禁止计数	—	允许计数
1	—	1. 启动计数； 2. 在下一个脉冲后使输出变低	—
2	1. 禁止计数； 2. 立即将输出置高	1. 重新装入计数常数 2. 启动计数	允许计数
3	1. 禁止计数； 2. 立即将输出置高	启动计数	允许计数
4	禁止计数	—	允许计数
5	—	启动计数	—

5. 8253 实验举例

8253 有多种工作方式,其中方式 3 为方波发生器,当设好初值后,自动将所设周期平分为两个部分,前一部分保持为高,后一部分保持为低,输出为一方波。 CLK_0 、 CLK_1 的频率均为 $1/8 \text{ MHz}$,设计数器 0 的初值为 $0F424H$ (十进制为 62500)时,方波周期为 0.5 s 。在计数器 2 中设置不同的初值 2^n 时,可得周期 $n \times 0.5 \text{ s}$ 的方波, n 的最大值为 $7FFFH$ (十进制为 32767)时,周期最长为 $16\ 383.5 \text{ s}$ (4 小时 33 分 3.5 秒),此时的初值为 $7FFFH$ 。因此,采用两级计数叠加后,输出周期范围可大幅度提高,如能合理设置初值,这其中广域范围的周期设定在实际控制中非常有用。下面为实验程序框图及程序举例:

(1) 实验程序框图(实验程序名 T3.ASM)

实验程序框图见图 9.10。

(2) 实验程序举例

```

1                                assume cs : code
2 0000                            code segment public
3                                org 100h
4 0100    BA 04A6      start : mov dx ,04a6h      8253 控制端口地址
5 0103    B8 0036      mov ax ,36h      设置计数器 0 工作方式
6 0106    EF          out dx ,ax
7 0107    BA 04A0      mov dx ,04a0h
8 010A    B8 0024      mov ax ,24h
9 010D    EF          out dx ,ax
10 010E   F8 00F4      mov ax ,f4h
11 0111   EF  out      dx ,ax
12
13 0112   BA 04A6      mov dx ,04a6h
14 0115   B8 000A      mov ax ,76h
15 0118   EF          out dx ,ax
16 0119   BA 04A2      mov dx ,04a2h
17 011C   B8 000A      mov ax ,0ah
18 011F   EF          out dx ,ax

```



图 9.10 8253 实验程序框图

```

19 0120    B8  0000      mov ax 0
20 0123    EF            out dx ax
21
22 0124    BA  04A6      mov dx 04a6h
23 0127    B8  00B6      mov ax 0b6h      ;设置计数器 2 工作方式
24 012A    EF            out dx 04a4h
25 012B    BA  04A4      mov dx 04a4h      ;计数器 2 地址
26 012E    B8  0004      mov ax 04h      ;设初值
27 0131    EF            out dx ax
28 0132    B8  0000      mov ax 0
29 0135    EF            out dx ax
30 0136    90            next :nop      ;空操作
31 0137    EB  FD        jmp next
32 0139
33          code ends
end start

```

9.1.5 可编程并行输入/输出接口芯片 8255A

1. 8255A 的作用和特性

8255A 是与 IBM 公司的微处理器配套的通用可编程并行 I/O 接口芯片 ,通过它可直接将 CPU 总线与外设联系起来 ,实现在 CPU 与外设间传送数据、控制信号和状态信号 .8255A 有三种工作方式 ,并通过程序来改变工作方式。它的通用性强 ,使用灵活 ,与 TTL 电路完全兼容 ,减少了系统器件 ,提高了直流驱动能力。

2. 8255A 的内部结构与引脚功能

8255A 的内部结构如图 9.11 所示 ,它由以下几部分构成。

(1) 3 个数据端口 A、B、C ,分为 A、B 两组。 A 组包括端口 A 和端口 C 的高 4 位 ,B 组包括端口 B 和端口 C 的低 4 位。

(2) A 组控制和 B 组控制。这两组控制部件都从读/写控制逻辑接收命令 ,从内部数据总线接收控制字 ,并向有关的口发出适当的命令。

(3) 读/写控制逻辑。该部件的功能是管理所有的内部和外部的传送过程 ,包括数据及控制字。它接收来自 CPU 地址总线和控制总线的输入信号 ,然后向 A 和 B 两组的控制部件发送命令。

(4) 数据总线缓冲器。该三态双向 8 位缓冲器是 8255A 用以与系统数据总线连接的部件。 CPU 通过输入/输出指令接收和发送的数据、控制字和状态信息都是通过该缓冲器传送的。

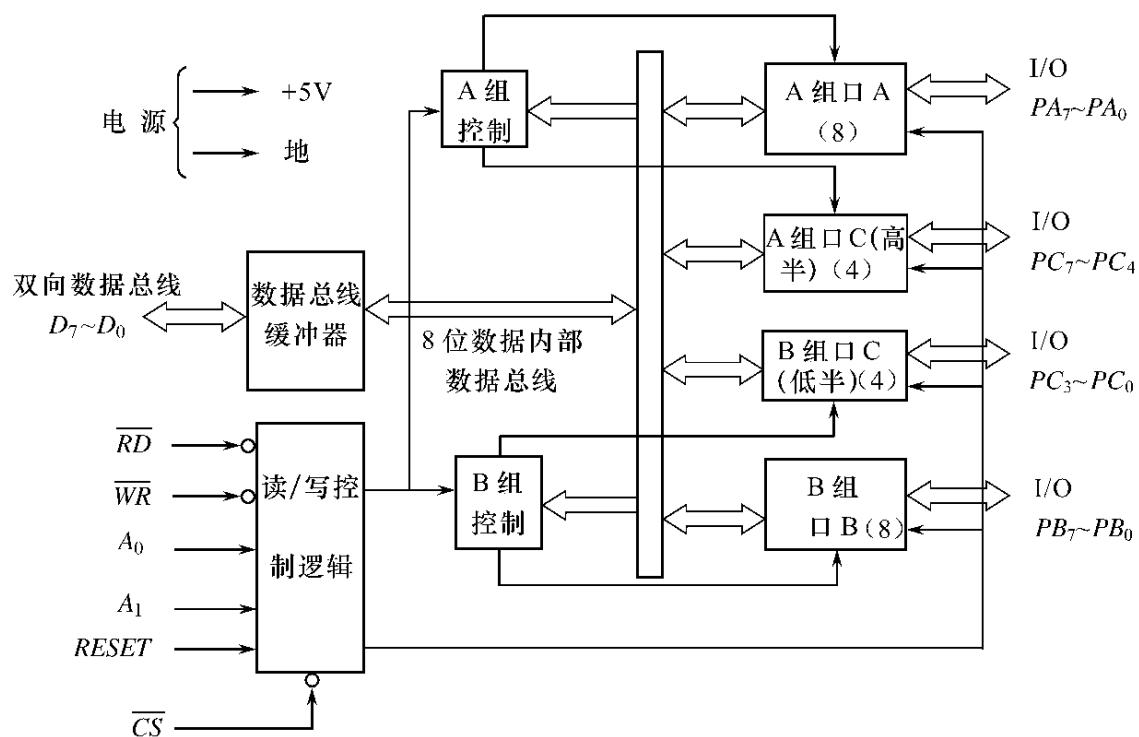


图 9.11 8255A 的内部结构框图

8255A 的引脚如图 9.12 所示。

PA ₃	1	40	PA ₄
PA ₂	2	39	PA ₅
PA ₁	3	38	PA ₆
PA ₀	4	37	PA ₇
RD	5	36	WR
CS	6	35	RESET
GND	7	34	D ₀
A ₁	8	33	D ₁
A ₀	9	32	D ₂
PC ₇	10	8255A	31
PC ₆	11		D ₃
PC ₅	12		D ₄
PC ₄	13		D ₅
PC ₀	14		D ₆
PC ₁	15		D ₇
PC ₂	16		V _{CC}
PC ₃	17		PB ₇
PB ₀	18		PB ₆
PB ₁	19		PB ₅
PB ₂	20		PB ₄
		21	PB ₃

图 9.12 8255A 的引脚图

引脚分两大部分：

(1) 与 CPU 相连的引脚

RESET：复位信号，高电平有效。当该输入端处于高电平时，所有内部寄存器（包括控制寄存器）均被清除，所有的 I/O（A、B、C）均被置成输入方式。

$D_7 \sim D_0$ ：数据线，和系统总线相连。

\overline{CS} ：芯片选择信号，低电平有效。当该引脚处于低电平时，允许 8255A 与 CPU 进行通信。一般接端口地址译码器输出端。

A_0 和 A_1 ：口选线 0 和口选线 1，用来选择 3 个数据口和控制字寄存器。选择功能见表 9.3。

表 9.3 口选线选择功能

\overline{CS}	A_1	A_0	选中的对象
0	0	0	A 口
0	0	1	B 口
0	1	0	C 口
0	1	1	控制寄存器

\overline{RD} ：读允许信号，低电平有效。当该脚为低电平时，允许 CPU 把数据或控制字写入 8255A。

(2) 与外设相连的引脚

$PA_0 \sim PA_7$ 、 $PB_0 \sim PB_7$ 、 $PC_0 \sim PC_7$ 分别对应端口 A、B 和 C。

3. 8255A 的控制字与状态字

8255A 有三种工作方式可供选择：

- 方式 0：基本的输入/输出方式；
- 方式 1：带选通的输入/输出方式；
- 方式 2：双向传输方式。

当 *RESET* 输入端处于高电平时，所有的 I/O 均被置成输入方式；*RESET* 信号撤销后，8255A 仍处于输入状态而不必再预置。要改变方式，只须用一条输出指令，向其控制字寄存器写入控制字就可以了。

下面介绍 8255A 的控制字与状态字。

(1) 方式选择控制字

方式选择控制字格式如图 9.13 所示。最高位 $D_7 = 1$ 是方式控制字的特征位。 $D_6 D_5$ 是 A 组的 A 口方式选择，2 位选择三种方式。 D_4 位是 A 口输入/输出

选择 , $D_4 = 0$ 表示 A 口输出 , $D_4 = 1$ 代表 A 口输入。 D_3 是 $PC_4 \sim PC_7$ 的输入/输出选择 ,1 代表输入 ,0 代表输出。指定 C 口高半字节中的具体输入/输出位数与 A 口选择的方式有关 ,详细见后面工作方式说明。 $D_2D_1D_0$ 是 B 组的方式与输入/输出选择 ,类似于 A 口。从图中可以看出 ,只有 A 口可以选择方式 2。

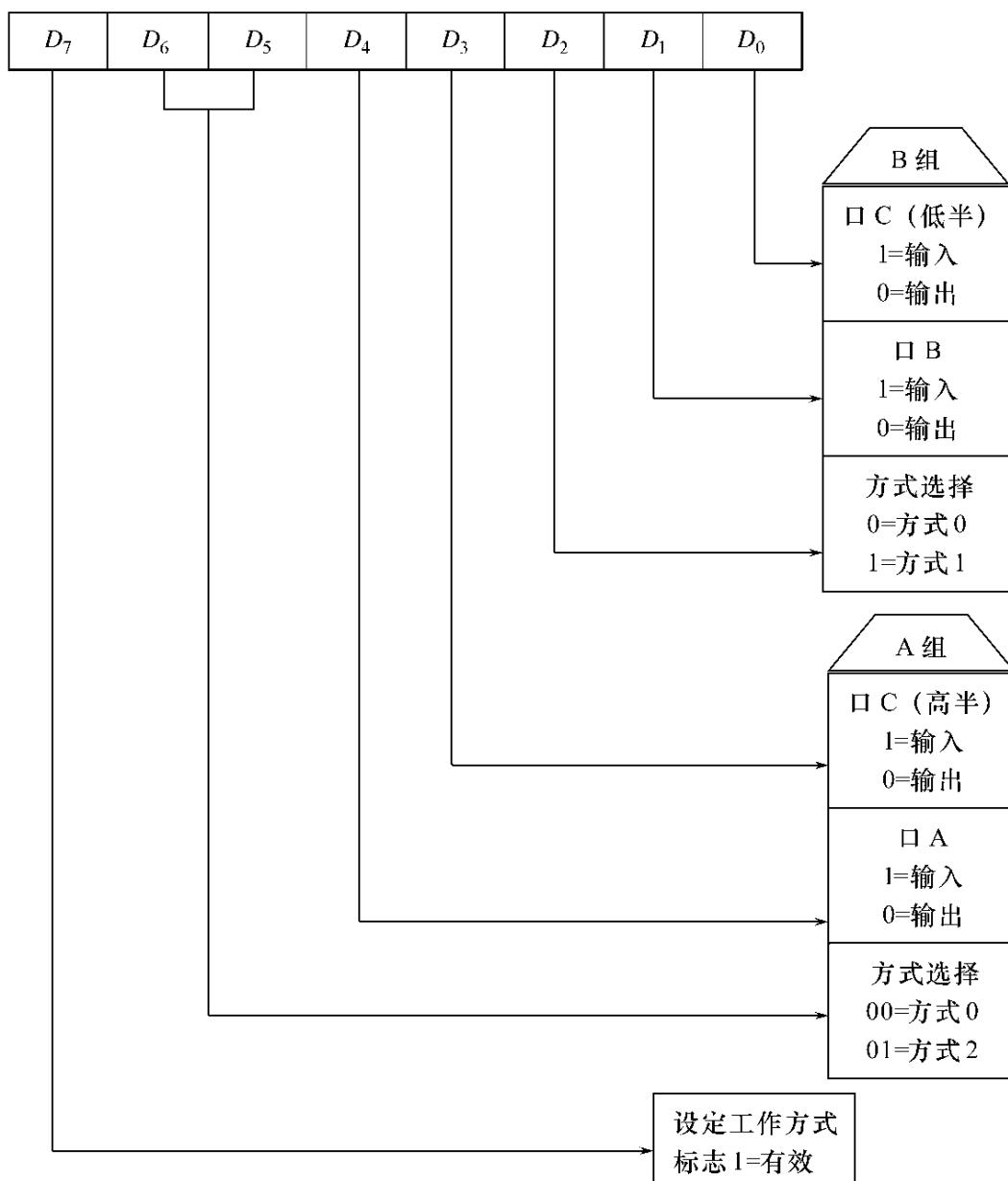


图 9.13 8255A 方式控制字格式

如果要求 A 口以方式 0 输出 ,B 口以方式 0 输入 ,C 口高 4 位输入 ,低 4 位输出 ,则方式选择控制字应是 8AH。若 8255A 地址为 320 ~ 323H ,则用以下指令即可写入控制字 :

```
MOV AL, 8AH
```

```
MOV DX, 323H
```

```
OUT DX, AL
```

(2) C 口个别位置 1/置 0 控制字

C 口的任何一位都可用一条输出指令置成 1 或 0。置 1 或置 0 控制字也是写入控制字寄存器。当 8255A 接收到控制字时会对最高位进行测试。当 $D_7 = 1$ 时, 则确认方式控制字; 若 $D_7 = 0$, 则是 C 口的置 1/置 0 控制字。端口 C 置 1 置 0 控制字格式如图 9.14 所示。例如, 要使 PC_7 置 1, PC_3 置 0, 8255A 地址为 320 ~ 323H, 则执行以下程序即可:

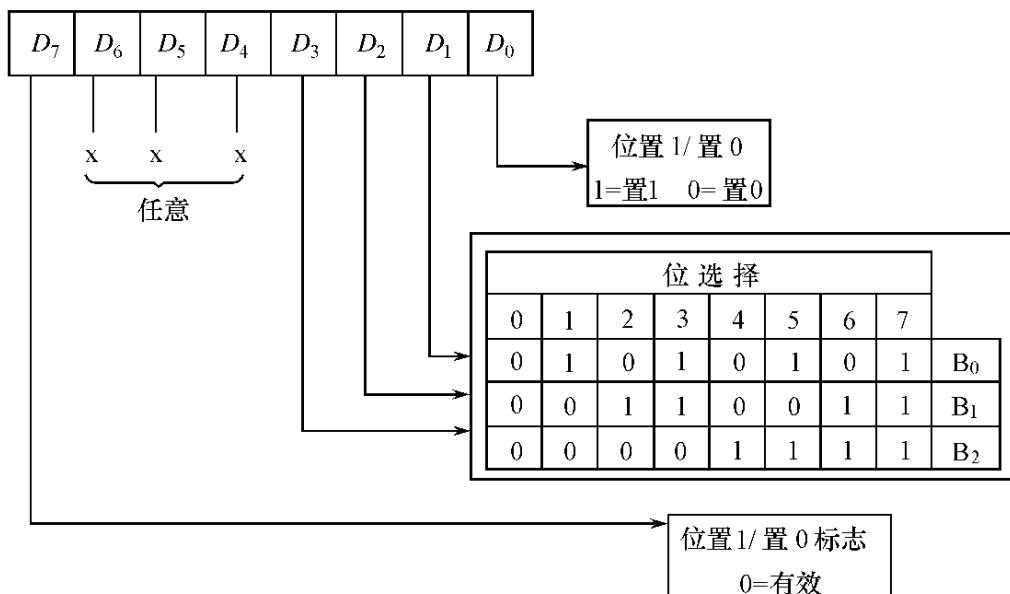


图 9.14 C 口置 1/0 控制字格式

MOV AL 0FH ; PC_7 置 1 的控制字

MOV DX 323H

OUT DX ,AL

MOV AL 06H ; PC_3 置 0 的控制字

OUT DX ,AL

在方式 1 和方式 2 的中断方式工作时, C 口的置 1/置 0 控制字可用来允许中断和禁止中断详见工作方式说明。

(3) 读 C 口的状态字

在使用方式 1 和方式 2 时, C 口发送或者接收与外部设备进行联络的信号。执行 C 口的正常读操作就能得到外部设备的状态。下面给出 3 个状态字的格式。每位的信号说明见工作方式一节。

4. 8255A 实验举例

8255A 有 3 个 8 位输入/输出端口, 由于内部电路原因, 通常将 A 端口(PA_0 ~ PA_7)作为输入用, B 端口(PB_0 ~ PB_7)作输出用, C 端口作辅助控制用。在实验举例中, 输入/输出都比较简单, 控制也不太复杂, 因此可选择在基本输入/输出方式(方式 0)下工作。在设计程序时, 先设置 8255 的工作方式(控制端口地址为

方式1的输入状态字

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
I/O	I/O	IBF_A	$INTE_A$	$INTR_A$	$INTE_B$	IBF_B	$INTR_B$
A组				B组			

方式1的输出状态字

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
\overline{OBF}_A	$INTE_A$	I/O	I/O	$INTR_A$	$INTE_B$	\overline{OBF}_B	$INTR_B$
A组				B组			

方式2状态字

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
\overline{OBF}_A	$INTE_A$	IBF_A	$INTE_2$	$INTR_A$			
A组				B组			

04A6H) ,然后循环读 PA 端口开关状态(地址为 04A2H),输出至 PB 端口发光二极管(地址为 04A2H)。

实验程序框图与程序如图 9.15 所示。

(8255 实验程序)

```

assume cs : code

code      segment public
          org 100h
start :   mov dx, 04a6h
          mov ax, 90h
          out dx, ax
start1 :  mov dx, 04a0h
          in ax, dx
          mov dx, 04a2h
          out dx, ax
          mov dx, 04a4h
          out dx, ax
          jmp start1
code ends
end start

```

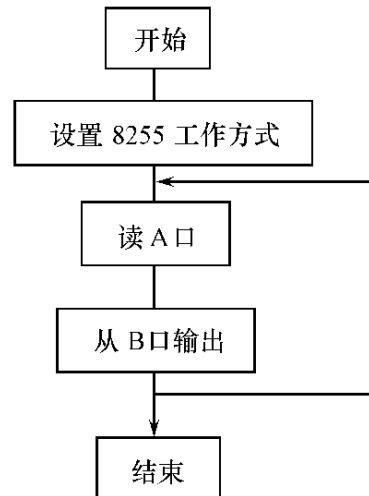


图 9.15 8255A 实验程序框图

9.2 串行通信

CPU与外设的通信有两种基本方式：并行通信和串行通信。并行通信是同时传送数据的位数，传输线的根数与数据的位数相同；串行通信则是将传输数据的每一个字符一位接一位地传送。串行通信速度比较慢，但所用的传输线较少，被广泛应用。串行通信从原理上又可以分为两种：同步串行通信与异步串行通信。

9.2.1 异步通信方式 ASYNC

在通信的数据流中，字符间异步，字符内部各位间同步。也就是说，每个字符出现在数据流中的相对时间是随机的，接收端预先并不知道，而每个字符一经开始发送，收/发双方则以预先固定的时钟速率传送各位。所以，所谓异步通信，主要指字符与字符间的传送是完全异步的，位与位之间还是基本同步传送的。异步通信时CPU与外设必须统一字符格式和波特率。

1. 字符格式

图9.16给出异步串行通信中一个字符的传送格式。开始前，线路处于空闲状态，送出连续“1”。传送开始时首先发一个“0”作为起始位，然后出现在通信线上的是字符的二进制编码数据。每个字符的数据位长可以约定为5位、6位、7位或8位，一般采用ASCII编码。后面是奇偶校验位，根据约定，用奇偶校验位将所传字符中为“1”的位数据凑成奇数个或偶数个。也可以约定不要奇偶校验，这样就取消奇偶校验位。最后是表示停止位的“1”信号，这个停止位可以约定持续1位、1.5位或2位的时间宽度。至此一个字符传送完毕，线路又进入空闲，持续为“1”。经过一段随机的时间后，下一个字符开始传送又发出起始位。

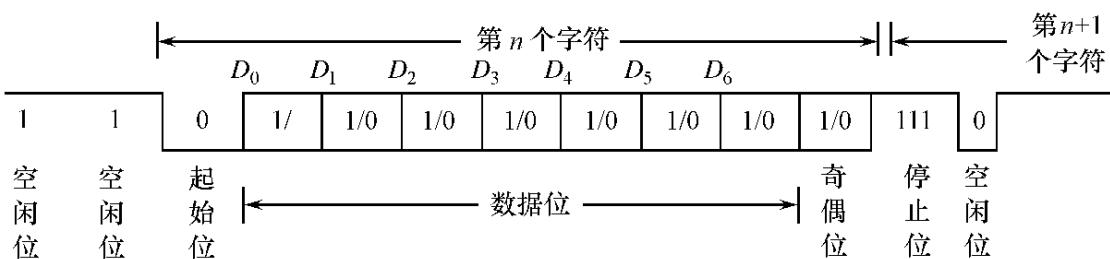


图9.16 异步串行通信字符格式

2. 波特率

所谓波特率 ,是指单位时间内传送二进制数据的位数 ,以 b/s(位 / 秒)为单位 ,所以有时也叫数据位率。它是衡量串行数据传送速度快慢的重要指标和参数。如有人说将某某终端的波特率调到 4 800 或 9 600 ,就是说把它的传送速度调到 4 800 或 9 600 b/s。有时也用“ 位周期 ”来表示传送速度 ,位周期就是每一个数据位的宽度 ,它等于传送波特率的数据。微机异步串行通信中 ,常用的波特率为 50、75、110、150、300、600、12 00、2 400、4 800、9 600 b/s ,有的可达 38 400 b/s 或更高。

9.2.2 同步通信方式

在异步传送中 ,每一个字符要用起始位和停止位作为字符的开始和结束 ,占用了时间 ,所以在数据块传送时 ,为了提高速度 ,就去掉这些标志 ,采用同步传送。同步传送时 ,在数据块开始处要用同步字符来指示 ,并在发送端和接收端之间要用时钟来实现同步 ,故硬件较为复杂。同步传送的速度高于异步 ,通常为几十 ~ 几百 kb/s。

同步通信控制规程可分为两类 :面向字符型(Character – Oriented)和面向位型(Bit – Oriented)。

1. 面向字符型的数据格式

面向字符型控制规程的特点是 ,规定一些字符作为传输控制用 ,信息长度为 8 的整数倍位 ,传输速率为 200 ~ 4 800 b/s。面向字符型的数据格式又有单同步、双同步和外同步之分 ,如图 9.17 所示。

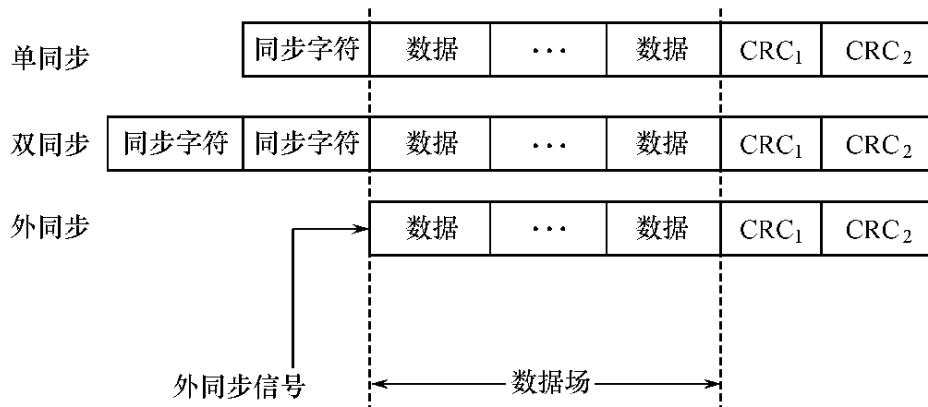


图 9.17 面向字符型同步通信数据格式

单同步是指在传送数据块之前先传送一个同步字符 SYNC ,接收端检测到该同步字符后开始接收数据 ;双同步则是安排两个同步字符 ;外同步格式中数据之

前不含同步字符,而是用一条专用控制线来传送同步字符,以实现收发双方的同步操作,任何一帧信息都以两个字节的循环控制码 CRC 为结束。

2. 面向位型的数据格式

面向位型控制规程的概念是由 IBM 公司在 1969 年提出的。它的特点是没有采用传输控制字符,而是采用某些位组合作为控制用,其信息长度可变,传输速率在 2 400 b/s 以上。这一类型中最有代表性的规程是 IBM 的同步数据链路控制规程 SDLC(Synchronous Data Link Control),国际标准化组织 ISO(International Standards Organization)的高级数据链路控制规程 HDLC(High Level Data Link Control)。

SDLC/HDLC 规程规定,所有信息传输必须以一个标志字符开始,且以同一个字符结束,这个标志字符为 01111110。从开始标志到结束标志之间构成一个完整的信息单位,称为一帧(Frame)。在 SDLC/HDLC 方式中,所有的信息都是以帧的形式传输的,而标志字符提供了每一帧的边界,接收器用每个标志字符建立帧同步。SDLC/HDLC 帧格式如图 9.18 所示。



图 9.18 SDLC/HDLC 的帧格式

(1) 地址场和控制场

SCLC/HDLC 的一帧信息是由若干场(Field)组成的。标志字符也称标志场,或简称为 F 场。在标志场之后,可以有一个地址场,简称 A(Address)场,和一个控制场简称 C(Control)场。SDLC 规定 A 场和 C 场的宽度为 8 位,而 HDLC 则允许 A 场可为任意长度,而 C 场为 8 位或 16 位。接收方必须检查每个地址字节的第一位,如果为“0”则后边跟着另一个地址字节;若为“1”则该字节就是最后一个地址字节。同理,如果控制场第一个字节的第一位为“0”,则还有第二个控制字节,否则就只有一个字节。

SDLC/HDLC 的所有场都是从最低有效位开始传送。

地址场用来规定与之通信的次站的地址。控制场可规定若干个命令。

(2) 信息场

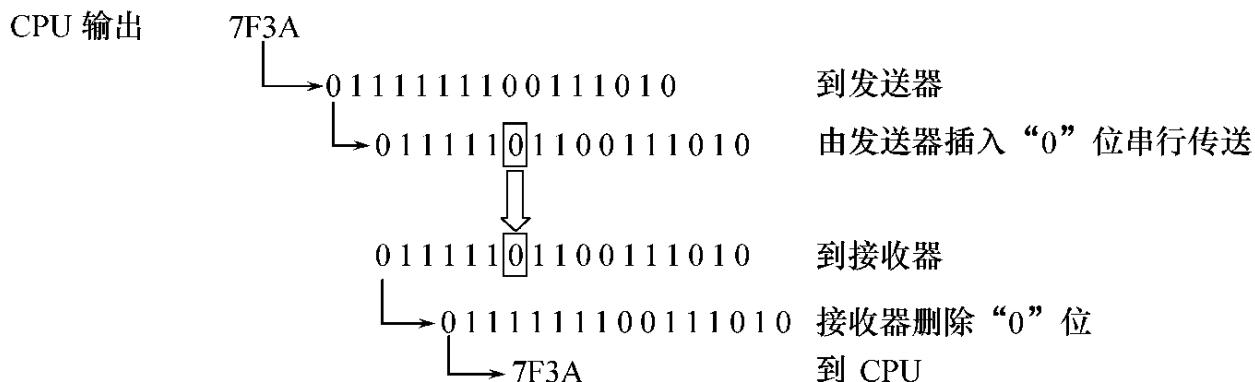
信息场跟在控制场之后,简称 I(Information)场。I 场包含有要传送的数据,但不是每一帧都必须有信息场。SDLC/HDLC 传送数据是作为任意位长的串行位流传输的。

(3) 帧校验场

SDLC/HDLC 规定的每一帧必须有两字节的帧校验场 ,紧跟在信息场之后。帧校验场称为 FC(Frame Check)场或称为帧校验序列 FCS(Frame Check Sequence)。 SDLC/HDLC 不采用 16 位循环冗余校验码(CRC)作为出错校验 ,其生成多项式为 CCITT 多项式 $X^{16} + X^{12} + X^5 + 1$ 。除了标志场和自动插入的“ 0 ”位外 ,所有的信息都参加 CRC 计算。

(4)“ 0 ”位插入 / 删 除技术

SDLC/HDLC 规程规定以 01111110 为标志字节 ,但在信息场中也完全有可能含有同一种格式的字符 ,为了把它与标志区分开来 ,采取了“ 0 ”位插入技术。发送方在发送标志字符外的所有信息(包括地址场、控制场、信息场和帧校验场) 时 ,只要遇到连续 5 个“ 1 ”就自动插入一个“ 0 ”。当然 ,当接收方在接收数据时 (除去标志场) ,如果连续收到 5 个“ 1 ” ,就自动将其后的一个“ 0 ”字删除 ,以恢复信息的原有形式 ,其过程如下 :



这种“ 0 ”位的插入和删除是由硬件自动完成的。

9.2.3 异步通信的标准接口

1. RS - 232 接口

EIA RS - 232C 是异步通信中应用最广的标准接线 ,它包括了按位串行传输的电气和机械方面的规定。适用于数据终端设备(DTE)和数据通信设备(DCE)之间的接口。一个完整的 RS - 232C 接口有 22 根线 ,采用标准的 25 芯插座。 25 芯插座的信号引脚定义如表 9.4 所示。其中 15 根引线(表中打 * 者)组成主信道通信 ,其他则为未定义的和供辅信道使用的引线。辅信道也是一个串行通道 ,但其速率比主信道低得多 ,一般不使用。如果要使用的话 ,主要是传送通信线路两端所接的调制解调器(Modem)的控制信号。

表 9.4 RS-232C 的接口信号

引脚号	说明
* 1	保护地
* 2	发送数据
* 3	接收数据
* 4	请求发送(RTS)
* 5	允许发送(CTS 或清除发送)
* 6	数传机(DCE)准备好
* 7	信号地(公共回线)
* 8	接收线信号检测
* 9	(保留供数传机测试)
10	(保留供数传机测试)
11	未定义
12	(辅信道)接收线信号检测
13	(辅信道)允许发送(CTS)
14	(辅信道)发送数据
* 15	发送信号无定时(DCE 为源)
16	(辅信道)接收数据
17	发送信号无定时(DCE 为源)
18	未定义
19	(辅信道)请求发送(RTS)
* 20	数据终端准备好
* 21	信号质量检测
* 22	振铃指示
* 23	数据信号速率选择(DTE/DCE 为源)
* 24	发送信号无定时(DTE 为源)
25	未定义

由于 RS-232C 是早期为促进公用电话网络进行数据通信而制定的标准 , 其逻辑电平对地是对称的 , 与 TTL、CMOS 逻辑电平不同。逻辑 0 电平规定为 +5 ~ +15 V 之间 , 逻辑 1 电平为 -5 ~ -15 V 之间。因此 RS-232C 驱动器与 TTL 电平连接必须经过电平转换。

RS-232C 由于发送器和接收器之间具有公共信号地 , 不可能使用双端信号 , 因此 , 共模噪声会耦合到信号系统中。这是迫使 RS-232C 使用较高传输电压的主要原因 , 即便如此 , 该标准的信号传输速率也只能达到 20 KB/s , 而且最大距离仅 15 m。只有在这种条件下才能可靠地进行数据传输。

大多数计算机应用系统或智能单元之间只需要使用 3 ~ 5 根信号线路即可工作。

目前大部分微机是 TTL 电平的 , 必须经过电平转换才能具有 RS-232C 的

电平。1488 和 1489 芯片能实现从 TTL 到 RS - 232C(发送器)及 RS - 232C 到 TTL(接收器)的转换,但转换时要用 3 个电源、2 个芯片,且每片只用了 1/4,使用很不方便。现在已有仅用 +5 V 供电的电平转换芯片,如 MAS232 和 MAX232A(高速)双组 RS - 232 发送器和接收器,多组的 RS - 232 发送器和接收器也已有产品,可在 MAX220 ~ MAX249 中选择。

2. RS - 449 与 RS - 423/422/485 接口

到目前为止,应用最广泛的串行接口标准还是 RS - 232C。它既是一种电气标准,又是一种物理接口功能标准。RS - 449 及 RS - 423/422/485 在概念上与 RS - 232C 不同。RS - 449 是一物理接口功能标准,而 RS - 423A、RS - 422A、RS - 485 则是电气标准。例如,采用 RS - 423A 电气标准,既可以通过 RS - 232C 的物理接口标准来实现,也可以通过 RS - 449 的物理接口来实现。

针对 RS - 232C 在数据传输时最大距离仅为 15 m,信号最高速率不能超过 20 kb/s,且未规定标准的连接器,从而出现互不兼容的 25 芯连接器以及接口处各信号间突然串扰等缺陷,美国电子工业协会 EIA 制定了 RS - 449 标准,并于 1980 年成为国家标准。RS - 449 在传输距离上最大可达 1 200 m,信号最高速率 100 kb/s 以上。EIA 同时明确规定了两种标准接口连接器:一种为 37 脚,一种为 9 脚。

RS - 232C 采用所谓单端驱动、单端接收的单端双极性电路标准,即当信号传输线上的是 -5 V 以下电平信号时为逻辑“1”,+5 V 以上电平信号时为逻辑“0”;且仅用一条线路传输一种信号。对于多条信号线来讲,它们的地线是公共的,这种共地传输方式抗干扰能力很差。尽管采用电平转换器来提高信号传输电平,但在较长距离时,由于电压损失,仍不可避免这一故障。另外,当信号穿过电磁干扰环境时,也可能因附加的干扰信号电平使发送的“0”变为“1”,或“1”变为“0”。故其信号波特率不能过高,仅限在 20 kb/s 以下,距离达 1 200 m。图 9.19 示出了单端驱动差分接收电路。

RS - 422A 标准规定了差分平衡的电气接口,它采用平衡驱动和差分接收的方法,从根本上消除了信号地线,这相当于两个单端驱动器输入同一个信号时,其中一个驱动器的输出永远是另一个驱动器的反相信号。于是两条线上传送的信号电平,当一条表示逻辑“1”时,另一条为逻辑“0”。在干扰信号作为共模信号出现时,接收器接收差分输入电压,只要接收器有足够的抗共模电压工作范围,就能识别两个信号并正确接收传送的信息。因此,RS - 422A 能在长距离、高速率下传输数据,它能够在 1 200 m 距离内把速率提高到 100 kb/s,在较短距离内,其传输速率可高达 10 Mb/s。这种性能的改善是由于平衡结构而产生的,差分平衡结构可以从地线的干扰中分离出有效信号,其最大可区分 0.20 的电位差值。

因此,一般不受地电位的波动和共模电磁干扰。图 9.20 为平衡驱动差分接收电路。采用 RS - 422A 实现两点之间远程通信时,其连接方式如图 9.21(a)所示,需要两对平衡差分电路形成全双工传输电路。

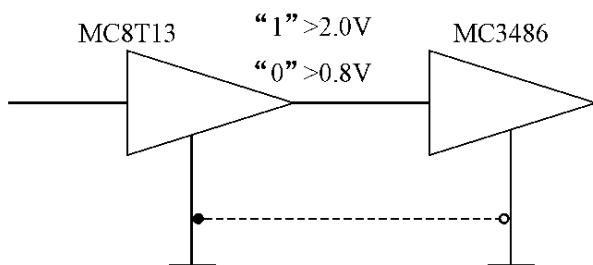


图 9.19 单端驱动差分接收电路

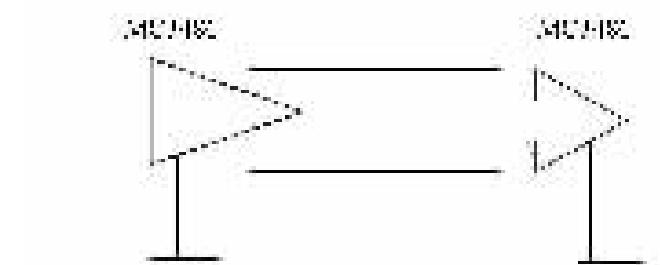
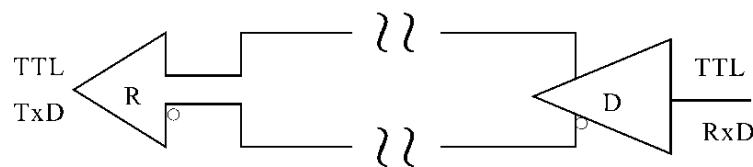
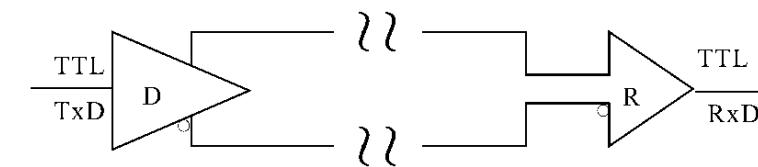
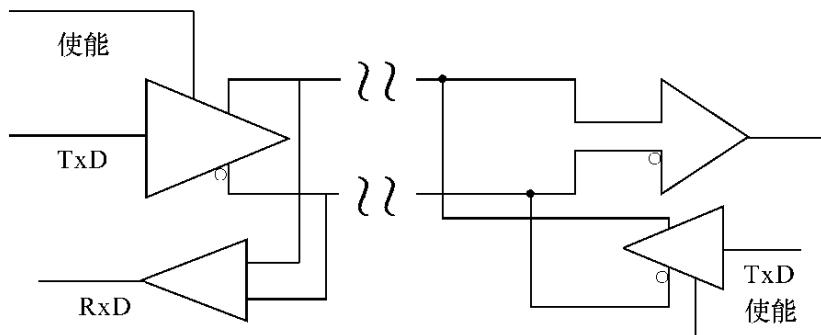


图 9.20 平衡驱动差分接收电路



(a)



(b)

图 9.21 (a) RS - 422A 传输电路 (b) RS - 485 传输电路

在许多工业控制及通信联络系统中,往往有多点互连而不是两点直连,而且大多数情况下,在任一时刻只有一个主控模块(点)发送数据,其他模块(点)处在接收数据的状态,于是便产生了主从结构形式的 RS - 485 标准。实际上,RS - 485 是 RS - 422A 的变型,它与 RS - 422A 都是采用平衡差分电路,区别在于按照上述的工作要求,RS - 485 为半双工工作方式,因而可以采用一对平衡差分信号线来连接。由于任何时候只能有一点处于发送状态,因此发送电路必须由使能信号加以控制。

RS - 485 用于多点互连时非常方便 ,可以省掉许多信号线。应用 RS - 485 可以联网构成分布式系统。RS - 422A 和 RS - 485 的驱动/接收电路没有多大区别 ,在许多情况下 ,RS - 422A 可以和 RS - 485 互连。如在环形系统中 ,采用 RS - 422/485 可以构成数据链路系统。

9.2.4 可编程异步通信接口 8250

1. 8250 的功能

8250 异步通信控制器是一种功能很强的串行接口芯片 ,常用在 16 位计算机中构成 RS - 232C 串行接口 ,8250 可以编程选择工作方式 ,主要任务是完成串并行转换、选择波特率等 ,主要功能如下 :

- (1) 可装配或拆卸串行数据格式 ;
- (2) 具有缓冲能力 ;
- (3) 有独立中断控制系统 ;
- (4) 有独立的接收时钟输入 ;
- (5) 可编程选择波特率 ;
- (6) 具有连接 Modem 功能 ;
- (7) 内部有自诊断能力。

2. 8250 的引脚与结构

(1) 8250 的引脚

INS8250 的引脚见图 9.22 所示 ,除电源线 V_{CC} (+5 V)和地线(GND)外 ,INS8250 的引脚信号可以分成系统和通信设备两方面信号。

① 系统的引脚信号

$D_7 \sim D_0$ 双向三态数据线 ,可直接连到系统数据总线。

CS_0 、 CS_1 、 \overline{CS}_2 片选信号输入。当 $CS_0 = CS_1 = 1$ 且 $\overline{CS}_2 = 0$ 时选中此片 ,即 3 个片选条件是相“与”关系 ,一般由高位地址译码 ,再加进必要的 I/O 控制信号产生。在 PC/XT 机中只用到 \overline{CS}_2 , CS_0 和 CS_1 都经电阻接 +5 V 。

CS_{OUT} 片选输出。当 3 个片选输入同时有效时 , $CS_{OUT} = 1$,作为选中此片的指示 ,在 PC/XT 机中未用。

$A_2 \sim A_0$ 地址信号输入 ,参加 INS8250 内部译码 ,一般接系统地址总线 $A_2 \sim A_0$ 。

\overline{ADS} 地址选通信号输入。当 $\overline{ADS} = 0$ 时选通上述片选和地址输入信号 ;当 $\overline{ADS} = 1$ 时锁存以上信号 ,以保证内部稳定译码。在 PC/XT 机中 ,此信号固定接地。

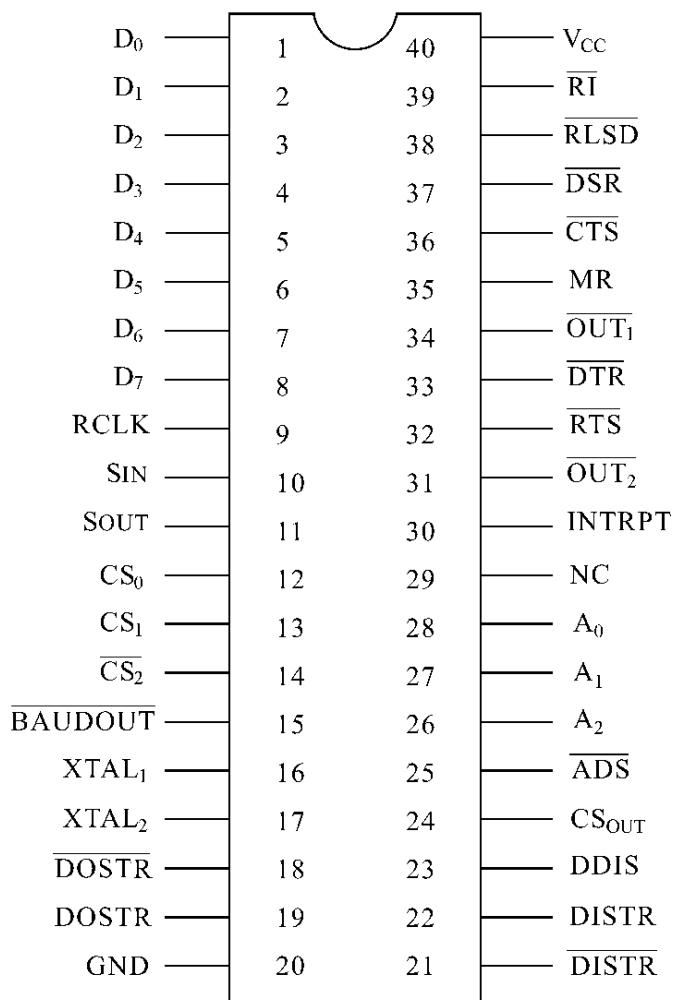


图 9.22 INS8250 引脚图

DISTR 和 *DISTR* 数据输入选通信号 ,二者作用相同 ,但有效极性相反。在芯片选中时 ,或者 *DISTR* = 1 ,或者 *DISTR* = 0 ,系统对芯片进行读操作。

DOSTR 和 *DOSTR* 与上面类似 ,当二者之一有效时 ,系统写入本片。

在 PC/XT 机中 *DISTR* 接 *IOR* ,*DOSTR* 接 *IOW* ,而 *DISTR* 和 *DOSTR* 都接地未用。

DDIS 驱动器禁止信号输出 ,高电平有效。当系统读 INS8250 时 ,*DDIS* = 0 (解除禁止) ,其他时间始终为高电平(禁止驱动)。因此若芯片向系统传送数据的通道上有三态驱动器 ,可用此信号来作其控制信号 ,平时禁止 INS8250 干扰系统数据总线。PC/XT 机中将此信号悬空未用。

MR 主复位信号输入 ,高电平有效。一般接系统复位信号 *RESET* ,用以复位芯片内部寄存器及有关信号 ,如表 9.5 所列。表中未列出的数据发送寄存器、数据接收寄存器及除数寄存器不受复位信号影响。

INTRPT 中断请求信号输出 ,高电位有效。INS8250 内部的中断控制电路在条件满足时对系统发出中断请求。在 PC/XT 机中 ,*INTRPT* 输出后还要经过 *OUT₂* 信号控制 ,*OUT₂* = 0 时 ,才能最终对系统形成中断请求。

表 9.5 INS8250 内部寄存器的复位

寄存器或信号	复位控制	复位结果
中断允许寄存器	MR	$D_7 \sim D_0$ 全为零
中断识别寄存器	MR	$D_0 = 1$, 其余位全为零
线路控制寄存器	MR	全为零
线路状态寄存器	MR	$D_5 = D_6 = 0$, 其余位全为 1
Modem 控制寄存器	MR	全为零
Modem 状态寄存器	MR	$D_3 \sim D_0$ 为零, 其位取决于输入
中断识别寄存器的 $D_2 \sim D_0$ 三位状态	110 100 010 000	MR 或读线路状态寄存器 MR 或读接收寄存器 MR 或写发送寄存器或读中断识别寄存器 MR 或读 Modem 状态寄存器
信号 S_{OUT} 、 \overline{OUT}_1 、 \overline{OUT}_2 、 RTS 、 DTR	MR	全为 1

② 外部通信设备的引脚信号

S_{OUT} 串行数据输出。系统写入的数据以字符为单位, 加进起始位、奇偶位及停止位等, 按一定的波特率逐位由此送出。

S_{IN} 串行数据输入。接收的串行数据从此进入 INS8250。

以上两数据信号分别和 RS - 232C 标准中的 TXD 及 RSD 对应。由于计算机内部使用正逻辑而 RS - 232C 使用负逻辑, 故中间加进了反相驱动器。

RTS 和 CTS 请求发送和清除发送, 是一以低电平有效的握手信号, 与 RS - 232C 中的 RTS 和 CTS 对应。当 INS8250 准备好发送时, 输出 RTS 信号, 对方的设备收到信号后, 若允许发送, 则回答一个低电平信号作为 CTS 输入, 于是握手成功, 传送可以开始。

DTR 和 DSR 数据终端准备好和数据装置准备好, 也是一对低电平有效的握手信号, 工作过程与前述类似。

$RLSD$ 接收线路信号检测输入, 低电平有效, 与 RS - 232C 的 DCD 信号对应, 从通信线路上检测到数据信号时有效, 指示应开始接收。

RI 振铃信号输入, 低电平有效, 与 RS - 232C 中 RI 同义。

在 PC/XT 机中以上 6 个联络信号全部引至 RS - 232C 接口。

OUT_1 和 OUT_2 芯片内部调制控制寄存器的 $D_2 D_3$ 两位的输出信号, 用户可以编程对其置位或复位, 以灵活地适应外部的控制要求。在 PC/XT 机中, OUT_2 用以控制 INS8250 的中断请求 $INTRP$ 信号。

$XTAL_1$ 和 $XTAL_2$ 时钟输入信号和时钟输出信号。也可以在两端间接一个石英晶体振荡器，在芯片内部产生时钟。此时钟信号是 IN8250 传输速率的时钟基准，其频率除以除数寄存器的值(分频)后得到的发送数据的工作时钟。PC/XT 机用外部时钟 1.8432 MHz 方波输入 $XTAL_1$ 。

$BAUDOUT$ 波特率输出信号，即上述发送数据的工作时钟，其频率是发送波特率的 16 倍。因此在 PC/XT 机中：

$$\text{发送波特率} = 1.8432 \text{ MHz} \div \text{除数寄存器值} \div 16$$

$RCLK$ 接收时钟输入，要求其频率为接收波特率的 16 倍。通常将其与 $BAUDOUT$ 信号短接，使接收和发送的波特率相等。

3. 8250 功能模块

IN8250 的功能框图如图 9.23 所示。除与系统相连的数据缓冲、地址选择及控制信号外，还可分成 5 个功能模块，每模块内又包含两个寄存器，共 10 个寄存器。但芯片只引入 3 根地址线，在内部至多产生 8 个地址。因此将两个除数寄存器和其他寄存器共用地址，在寻址除数寄存器时先设立特征，即使线路控制寄存器的最高位 $DLAB = 1$ 。当 $DLAB = 0$ 时，寻址除数寄存器以外的寄存器（见表 9.6）。将表中 3F8 ~ 3FFH 改成 2F8 ~ 2FFH 即是 2 号异步串行通信口 COM2 的地址表。

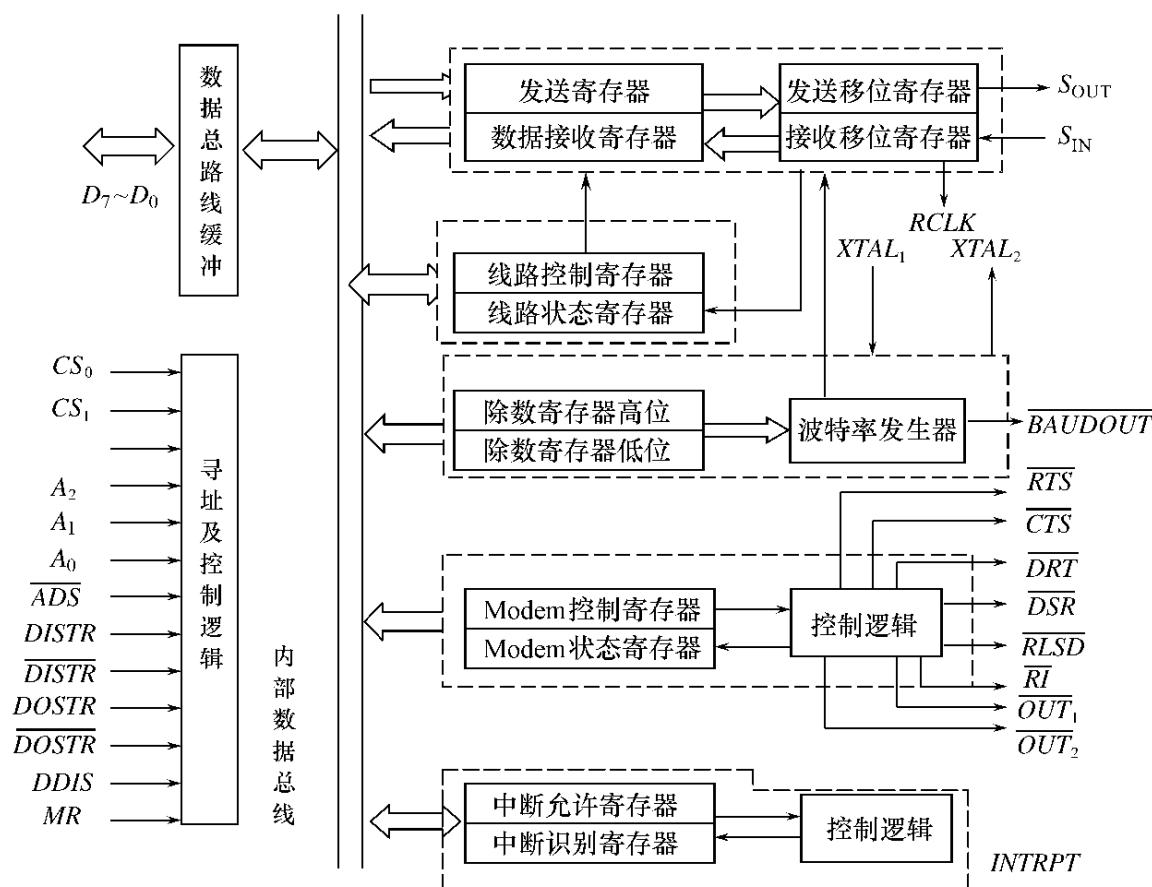


图 9.23 IN8250 的功能图

表 9.6 INS8250 内部寄存器寻址

地址信号 $A_2A_1A_0$	标志位 $DLAB$	COM_1 地址(H)	寄存器
000	0	3F8	写发送寄存器/读接收寄存器
000	1	3F8	除数寄存器低字节
001	1	3F9	除数寄存器高字节
001	0	3F9	中断允许
010	×	3FA	中断识别
011	×	3FB	线路控制
100	×	3FC	Modem 控制
101	×	3FD	线路状态
110	×	3FE	Modem 状态
111	×	3FF	不用

(1) 数据发送和接收部分

① 数据发送

数据发送部分可分为数据发送保持寄存器和发送移位寄存器。

输出数据以字符为单位首先送到数据发送保持寄存器中,再进入发送移位寄存器,以上过程都是并行方式传送的。在发送移位寄存器中,按照事先和接收方约定的字符传输格式,加上起始位,奇偶校检位和停止位,然后再以约定的波特率(由波特率控制部分产生)先低位后高位由 S_{OUT} 端串行和移位送出。

数据发送保持寄存器在将数据传给发送移位寄存器后(即发送寄存器空),CPU 即可对它写入下一个字符,而发送移位寄存器完全送出第一个字符位(即发送移位寄存器空)后,又立即接收第二个字符,开始第二个字符的发送。“发送寄存器空”和“发送移位寄存器空”状态,都在下面讲到的线路状态寄存器中有对应位反映。使 CPU 可以用查询或中断方式了解,继续输出后续字符。

② 数据接收

数据接收部分包括接收移位寄存器和数据接收缓冲寄存器。

串行数据从 S_{IN} 端逐位进入接收移位寄存器。接收数据时,首先搜寻起始位,然后才读入数据位。这个过程如图 9.24 所示。

接收电路始终用接收时钟 RCLK 选通采样串行输入 S_{IN} 的状态,每 16 个 RCLK 脉冲对应一个数据位。在检测到由“1”到“0”的变化时,若连续采样 8 次, S_{IN} 一直都保持为“0”,则认定是数据起始位;否则认为是干扰信号,将重新采样。以后再每隔 16 个 RCLK 周期读取一次数据位(正好在每个数据位的中点读),读至停止位,一个字符接收完毕,然后开始搜寻第二个字符的起始位。这样的安排

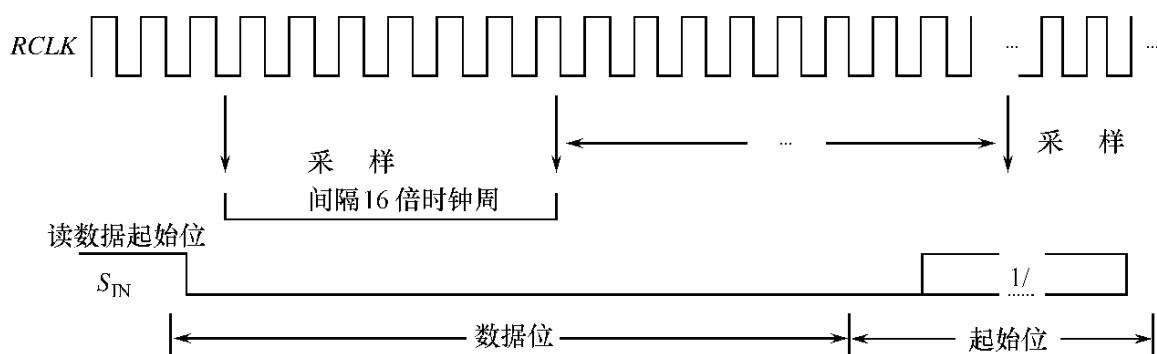


图 9.24 异步串行接收数据过程

除了可以减少误判起始信号以外,还允许发送时钟和接收时钟的频率有一定误差,每个字符单独起始又避免了时钟误差的积累。

接收移位寄存器接收一个字符后,要进行格式检查,若不正确,则通过线路状态寄存器设置出错标志位;若格式正确则将真正的数据位保留并传给数据接收缓冲寄存器,然后将线路状态寄存器中的“接收数据可用”位置“1”,CPU 可以通过查询或中断方式取走这个字符,清除“接收数据可用”位,以接收下一字符。显然,若接收的前一个字符在数据接收缓冲寄存器中尚未被 CPU 取走,后一个字符经接收移位寄存器接收完毕又要送至接收缓冲寄存器,就会丢失字符,这种情况称为“溢出错”,在线路状态寄存器中也有相应位记录。

(2) 线路控制及状态部分

① 通信线路控制寄存器

CPU 用 OUT 指令将一个 8 位的控制字写入通信线路控制寄存器,以决定通信中字符的格式。控制寄存器的内容也可以用 IN 指令读出。其各位的作用如图 9.25 所示。

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
除数标志	中止位	奇偶校验选择		停止位长		数据位长	
0 正常				0 1 位	00	5 位	
1 访问标志		000 无校验		1.5 位	01	6 位	
0 正常		001 奇校验		或 2 位	01	6 位	
1 发中止符		011 偶校验		或 2 位	10	7 位	
		101 附加位为 1					
		111 附加位为 0					

图 9.25 线路控制寄存器

最高位为访问除数寄存器的标记 $DLAB$ 。 $D_7 = 1$ 时执行的 I/O 指令应是访问波特率控制部分的除数寄存器; $D_7 = 0$,即正常寻址。

$D_6 = 0$ 时正常发送 ; $D_6 = 1$ 则中止正常发送 ,串行输出端 S_{OUT} 保持为“ 0 ”。

$D_5 \sim D_3$ 这 3 位规定了通信数据的奇偶校验规则。 D_3 表示校验有或无 , D_4 表示校验的奇偶性 , D_5 的设置可以把发送方校验的奇偶性规定通过发送数据中的附加位去告诉接收方(即不必事先约定), 当 $D_5 = 1$ 时 , 在发送数据的奇偶校验位和停止位之间附加一个标志位 : 若采用偶校验则附加位为“ 0 ”; 若采用奇校验 , 则附加位为“ 1 ”。接收方收到数据后 , 只要将附加位分离出来 , 便可得知发送数据的奇偶校验规定。正常情况下 , 数据的奇偶性是事先约定的 $D_5 = 0$, 也不附加标志位。图中列出了这 3 位的几种常用组合。

$D_2 = 0$ 时表示只有一位停止位 ; $D_2 = 1$ 时 , 若数据位长为 5 则表示有一位半停止位 , 若数据位长为 6、7 或 8 , 则停止位应是两位。

$D_1 \sim D_0$ 规定了数据位的长度 , 如图 9.25 中所列。

② 通信线路状态寄存器

CPU 读入通信线路状态寄存器 , 便可了解数据发送和接收的情况 , 如图 9.26 所示 , 其中 D_7 无用。

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	发送移位寄存器空	发送寄存器空	中止符检测	帧格式错	奇偶错	溢出错	接收数据就绪

图 9.26 线路状态寄存器

$D_5 = 1$ 反映发送寄存器已将字符传送给移位寄存器 , 当发送移位寄存器将字符各位全部从 S_{OUT} 送出后 , $D_6 = 1$ 。这两位不全为 1 时说明发送工作没有真正结束。

其余位都反映接收数据的状态。当接收移位寄存器收够一个字符规定的位数时 , 使 $D_0 = 1$, 设置“ 接收数据就绪 ”(亦称“ 接收移位寄存器满 ”) 状态标记。这个数据是否正确还要经过多方面检查 , 若发生错误 , 则将 $D_3 \sim D_1$ 相应位置“ 1 ”。若接收连续的“ 0 ”信号超过一个字符宽度时 , 认为对方已中止发送 , 则使 $D_4 = 1$ 。

以上各位状态在 CPU 读线路状态寄存器后即被清零。除 D_6 外其他位还可以被 CPU 写入 , 也可以产生中断请求。

③ 波特率控制部分

这部分的可编程寄存器即除数寄存器 , 实际上是分频系数。外部输入时钟 XTAL1 的频率(PC/XT 系列中为 1.843 2 MHz)除以除数寄存器中的双字节数后 , 得到数据发送器的工作频率 , 再除以 16 , 才是真正的发送波特率 , 在 PC/XT 中也就是接收波特率。 PC/XT 中波特率和除数之间的关系见表 9.7 所示。

表 9.7 波特率与除数的关系

波特率	除数		波特率	除数	
	高字节	低字		高字节	低字
50	09	00	1800	00	40
75	06	00	2000	00	3A
110	04	17	2400	00	30
134.5	03	59	3600	00	20
150	03	00	4800	00	18
300	01	80	7200	00	10
600	00	C0	9600	00	0C
1200	00	60	19200	00	06

④ Modem 控制与状态

此模块实现通信过程中的联络功能 ,包括联络信号的生成及检测。

(a) 参看图 9.27 ,该寄存器的高 3 位无用。 D_4 决定 INS8250 的工作方式 : $D_4 = 0$,INS8250 正常工作 ; $D_4 = 1$,INS8250 处于自检状态 ,即其数据输入 S_{IN} 同外部断开 ,而在芯片内部同数据输出 S_{OUT} 接通 ,同时 4 个输入信号 \overline{DSR} 、 \overline{CTS} 、 \overline{RLSD} 、 \overline{RI} 分别和 4 个输出信号 \overline{DTR} 、 \overline{RTS} 、 OUT_1 、 OUT_2 在内部相联 ,于是就可以用自发自收的方式来检查芯片。

$D_3 \sim D_0$ 每一位控制一个输出信号。

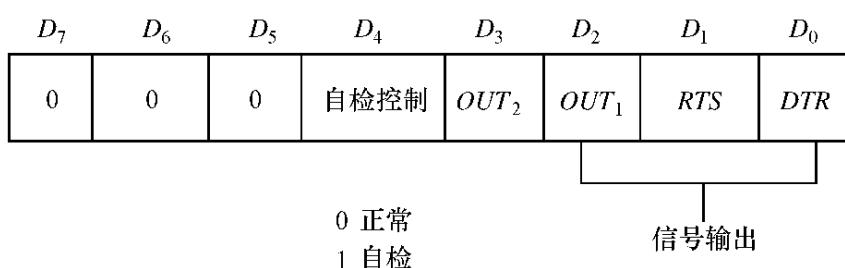


图 9.27 Modem 控制寄存器

(b) Modem 状态寄存器

参看图 9.28 ,其高 4 位即 4 个外部输入信号的状态 ,而低 4 位记录高 4 位的变化。每次读 Modem 状态寄存器时 ,低 4 位被清零。以后若高 4 位中有某状态发生改变(由“ 0 ”变到“ 1 ”或由“ 1 ”变到“ 0 ”),则低 4 位中的相应位就置“ 1 ”。这些状态位的变化 除了可以让 CPU 用输入指令查询外 ,也可以引起中断。

⑤ 中断允许及识别

INS8250 有很强的可编程中断管理功能 ,用户可以通过对中断允许寄存器及

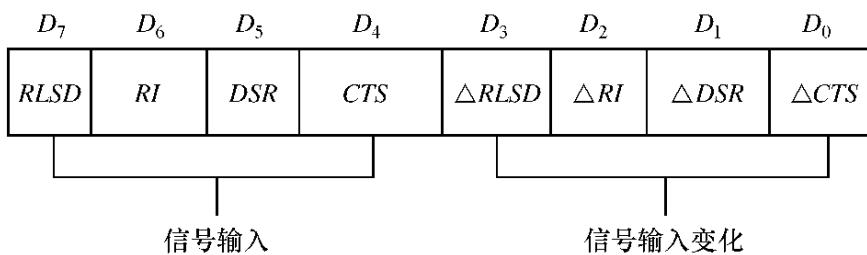


图 9.28 Modem 状态寄存器

中断识别寄存器的读/写操作来设置和利用。

(a) 中断允许寄存器

INS8250 将芯片内的各种中断源分为 4 类 ,用中断允许寄存器的低 4 位来对各类中断源实现允许或者屏蔽控制。其对应关系见图 9.29 所示。

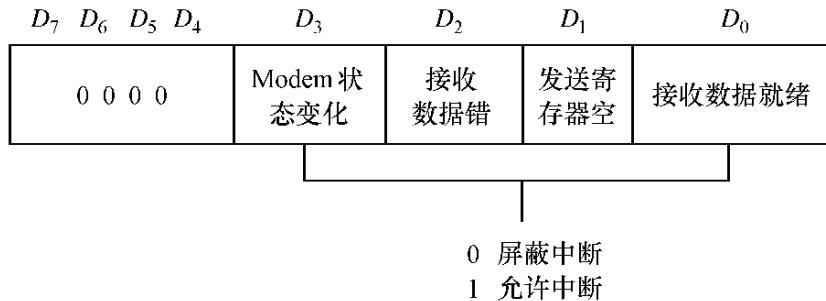


图 9.29 中断允许寄存器

中断允许寄存器的高 4 位固定为“ 0 ” ,没有使用。

若 $D_3 = 1$,则 Modem 状态寄存器的高 4 位状态发生改变时 ,允许发出中断请求信号 INTRPT 。若 $D_3 = 0$,则 Modem 状态中断被屏蔽。

$D_2 \sim D_0$ 决定线路状态寄存器引起的中断是否允许 ,同样也是为“ 1 ”的位允许中断 ,为“ 0 ”的位屏蔽中断。其中 D_2 对应接收数据错(包括溢出错、奇偶错及帧格式错)及中止符检测中断。

中断允许寄存器的相应位为“ 1 ” ,只是允许中断源产生 INTRPT 信号 ,后面还要经过 OUT_2 信号控制才可能最终产生中断请求 IRQ 信号送 GC 8259A 。

(b) 中断识别寄存器

INS8250 对内部 4 类中断源各以两位二进制编码 ,在中断允许的前提下 ,将当前中断类型的识别码写入中断识别寄存器的 D_2D_1 两位中 ,同时将中断指示位置零(表示有中断请求)。4 类中断源具有不同的中断优先级。当不同级别的多个中断源同时申请时 ,仅将最高优先级的识别码写入中断识别寄存器中。

中断识别寄存器的内容只可读出。其低 3 位实时反映中断的发生情况 ,而高 5 位始终固定为“ 0 ”。这个特点常用来检查 INS8250 在系统中是否存在 ,或是

否安装了异步串行通信口。

程序如下所列：

MOV DX 3FAH	指向 COM1 的中断识别寄存器
IN AL ,DX	读中断识别寄存器
TEST AL 0F8H	测试高 5 位
JZ INITIALIZATION	全零则转初始化

4. 8250 实验举例

(1) 下位机程序

```

1 0000           code segment
2                         assume cs : code
3                         org 0100h
4 0100 BB 0480      start : mov bx 0480h
5 0103 8B D3        mov dx ,bx
6 0105 8B c2 06      add dx ,6
7
8 0108 B8 0080      mov ax 80h
9 010B EF          out dx ,ax
10 010C 8B D3       mov dx ,bx
11 010E B8 000C      mov ax 0ch
12 0111 EF          out dx ,ax  设置波特率 9 600 bit/s
13
14 0112 83 C2 02      add dx ,2
15 0115 B8 0000      mov ax 0h
16 0118 EF          out dx ,ax  设置中断方式
17
18 0119 83 C2 04      add dx ,4
19 011C B8 0007      mov ax 07
20 011F EF          out dx ,ax  设置线路控制寄存器
21 0120 8B D3       mov dx ,bx
22
23 0122 E8 0005      crd : call recv
24 0125 E8 0011      call send
25 0128 EB F8        jmp crd
26
27 012A 83 C2 0A      recv : add dx 0ah
28 012D ED          recv3 :in    ax ,dx

```

```

29 012E A9 0001      test    ax ,   01h
30 0131 75 02       jnz     recv1  接收器数据是否就绪 ,
                           就绪则转移
31 0131 75 02       jmp     recv3
32 0135 8B D3       recv1 :mov dx ,bx
33 0137 ED           in      ax ,dz  接收数据
34 0138 C3           ret
35
36 0139 50           send : push ax      接收数据压栈
37 013A 83 C2 0A     add     dx 0ah
38 013D ED           sendl :in  ax ,dx
39 013E A90020       test    ax 20h
40 0141 75 02       jmp     secv2  发送器保持器是否为空 ,
                           为空则转移
41 0143 EB FB         jmp     sendl
42 0145 58           recv2 :pop ax      接收数据出栈 , 并准备
                           将其发送
43 0146 8B D3         mov     dx ,bx
44 0148 EF           out    dx ,ax  发送数据
45 0149 C3           ret
46
47 014a               code   ends
48                   end    start

```

(2) 下位机程序

```

data segment
    ttt      dw 0          out dx ,al
    ttt1     dw 0          mov dx ,bx
    com      dw 0          mov al 0ch
    messag1 db 'please dey in  com( 1 2 ):' ien dx
    '$'messag2 db 'send : ' ,'$'recvstr db255dup( ?) mov al ,oh
    messag3 db 'receive : ' ,'$' out dx ,al
data ends
code segment
    assume cs : code ,ds : data add dx 2
    main proc far          mov al 07
                                out dx ,al
start :
    rush      ds             inc dx

```

sub	ax ,ax	mov al ,0
posh	ax	
mov	ax ,data	out dx ,al
mov	ds ,ax	add dx ,A
main0 :		in al, dx
mov	ah ,0	mov dx ,bx
mov	al ,3	in al ,dx
int	10h	mov ax ,0a15h
mov	al ,3	lea dx ,messag3
int	10h	call disp1
mov	ah ,5	mov bh ,0
mov	ah ,15	mov ahm3
lea	dx ,messagl	int 10h
call	displ	mov si ,offset ttl
wkey : mov ah ,1		mov word ptr [si] ,dx
int	21h	mov ax815h
cmp	al ,31h	lea dx ,messag2
je	coml	call duspl
jmp	wkey	crd : mov ah ,1
coml : mov si ,offset com		int 21h
mov	word ptr [si] ,03f8h	cmp al ,0dh
jmp	intcom	je exit
com2 : mov si ,offset com		push ax
mov	word rth[si] ,02f8h	mov bh ,0
intcom : mov si ,offset com		mov ah ,3
mov	bx [si]	int 10h
mov	dx ,bx	mov si ,offset ttt
add	dx ,3	mov word ptr [si] ,dx
mov	al ,80h	pop ax
mov	dx [si]	' call send
mov	ah ,2	call recv
int	10h	push ax
pop	ax	mov si ,offset ttl
mov	ah ,2	out dx ,al
mov	dl ,al	ret
int	21h	send endp
mov	bh ,0	recv proc near

mov	ah 3		mov cx ,1000h
int	10h	rss :	mov si ,offset com
mov	si ,offset ttl		mov bx [si]
mov	word pt[si],dx		mov dx ,bx
mov	si ,offset ttl		add dx ,5
mov	word pt[si],dx		in al ,dx
mov	si ,offset ttt		test al ,01h
mov	dx [si]		jnz recv1
mov	bh 0		dec cx
mov	ah 2		cmp cx ,0
int	10h		jne rss
jmp	crd		mov al ,0
exit :	mov ah 4ch		jmp retl
	int 21h	recv1 :	mov dx ,bx
main	endp		in al ,dx
send proc	near	retl :	ret
srss : push	ax	recv	endp
mov	si ,offset com	displ	proc near
mov	bx [si]		push dx
mov	dx ,bx		mov dh ,ah
add	dx ,5		mov dl ,al
in	al ,dx		mov ah ,2
test	al ,20h		mov bh ,0
jnz	recv2		int 10h
push	bx		pop dx
mov	ah ,1		mov ah ,9
int	16h		int 21h
pop	bx		ret
jz	sretl	displ	endp
pop	ax	code	ends
jmp	srss	end	start
recv2 : mov	dx ,bx		
pop	ax		

9.3 中断技术与 DMA 技术

9.3.1 中断的基本原理与 8259A 中断控制器

1. 中断的基本原理

(1) 中断过程

与 CPU 的查询传送方式相反 , 中断方式是 CPU 被动地等待外设请求服务的一种方式。当外设准备好向 CPU 传送数据或已准备就绪请求接收 CPU 的数据 , 或者有某些紧急情况需要 CPU 处理时 , 外设向 CPU 发出中断请求 ,CPU 接收到中断请求 , 在一定条件下 , 暂时停止执行当前的主程序 , 转到中断服务程序为外设服务 , 服务结束后返回并继续执行主程序。一个中断过程分为三个阶段 : 中断请求 , 中断响应和中断处理。

(2) 中断优先权

实际的系统中常常有多个中断源 , 而中断申请引脚往往只有一条中断请求线。现在多个中断源同时请求时 ,CPU 要能辨别优先权最高的中断源并响应之 , 常用的优先权识别分析主要是两种 , 一是用软件查询分析方法确定中断优先权 ; 二是采用硬件优先权排队电路。

2. 8259A 中断控制器

8259A 是一种可编程中断控制器 , 能管理 8 级向量优先权中断 , 也即能管理 8 个中断请求 , 安排它们的优先顺序 , 在中断响应时发出相应的中断类型码 (中断矢量) , 在不增加其他电路的情况下 , 可用多片 (最多 9 片) 8259A 级联构成最多 64 级的向量优先权中断系统。

3. 8259A 的引脚与结构

(1) 8259A 的逻辑框图如图 9.30 所示。

8 位数据总线缓冲器 : 它是 8259A 与系统数据总线的接口 , 为双向三态缓冲器。 CPU 对 8259A 的控制字是通过它写入的 ; 8259A 的状态信息通过它读入 CPU ; 在中断响应周期 , 8259A 也是通过它送出中断矢量。

读 / 写控制逻辑 : 该部件接收来自 CPU 的读 / 写命令 , 配合 \overline{CS} 端的片选信号和 A_0 端的地址输入信号 , 完成规定的操作。

级联缓冲器 / 比较器 : 这个功能部件在级联方式的主从结构中 , 用来存放和

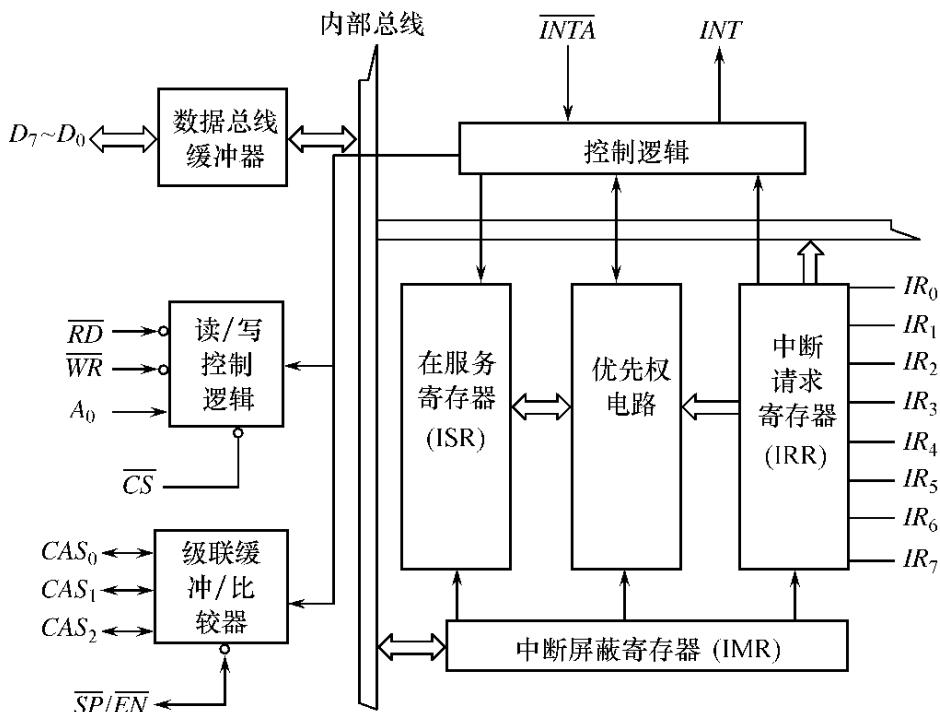


图 9.30 8259A 逻辑框图

比较系统中各 8259A 的从设备标志(ID)。与此相关的是 3 条级联线 $CAS_0 \sim CAS_2$ 和从片编程/允许缓冲器 $\overline{SP}/\overline{EN}$ 线。

中断屏蔽寄存器(IMR): 它是对 8 级中断请求分别独立地加以禁止和允许的寄存器。若某位置 1 则与之对应的中断请求被禁止。

中断请求寄存器(IRR): 8259A 有 8 条外界中断请求线 $IR_0 \sim IR_7$, 每一条请求线有一个相应的触发器来保存请求信号 , 8 个触发器组合成为 IRR。

在服务寄存器(ISR): 该寄存器用来存放当前正在进行服务的中断。若某位为“ 1 ” , 表示正在为相应的中断源服务。

优先权电路 : 该电路对保存在 IRR 中的各个中断请求 , 经过判断确定最高的优先权 , 使 ISR 中相应位置 1 , 即为之服务。

控制逻辑 : 按照编程设置的工作方式管理 8259A 的全部工作。在 IRR 中有未被屏蔽的中断请求位时 , 控制逻辑输出高电平的 INT 信号 , 向 CPU 申请中断。在中断响应期间 , 它允许 ISR 的相应位置位 , 并发出相应的中断类型号 , 通过数据总线缓冲器输出到系统总线上。在中断服务结束时 , 它按照规定的方式对 ISR 进行处理。

(2) 8259A 引脚如图 9.31 所示 , 各引脚功能说明如下 :

\overline{CS} 片选输入信号 , 低电平有效。 \overline{CS} 为低电压时 , CPU 可以通过数据总线设置命令和对内部寄存器进行读出。当进入中断响应时 , 这个引脚的状态与进行的处理无关。一般接译码器输出。

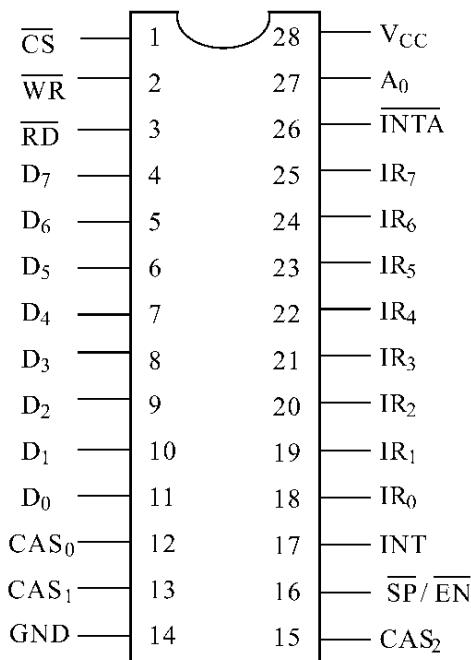


图 9.31 8259A 引脚图

\overline{WR} 写控制输入信号。同系统总线上的 IOW 信号相连接 ,向 8259 写入命令和初始化字。

\overline{RD} 读控制输入信号。同系统总线上的 IOR 信号相连接 ,由 CPU 读 8259A 的内部寄存器。

$D_0 \sim D_7$ 双向三态数据线。与系统数据总线相接。

$CAS_0 \sim CAS_2$ 级联信号线。在多片 8259A 连接时用来构成 8259A 主 – 从式级联控制结构。当 8259A 作为主片时 ,它们是输出信号 ;当 8259A 作为从片时 ,它们是输入信号。编程时设定的从设备标志保存在级联缓冲器内。在中断响应期间 ,主 8259A 把所有申请中断的从片中优先级最高的从 8259A 的从设备标志输出到级联级的 $CAS_0 \sim CAS_2$ 上 ,当片 8259A 把这个从设备标志与级联缓冲器内保存的从设备标志进行比较。在第二个 INTA 脉冲期间 ,被选中的从 8259A 的中断类型号送到数据总线上。这个中断号也是在编程时预先设定的 ,保存在控制逻辑部件内。只使用一片 8259A 时 ,不使用这些引脚。

SP/EN 从片编程/允许缓冲器 ,双向 ,低电平有效。这些信号线有两种功能 :当工作在缓冲方式时 ,它是输出信号 ,用做允许缓冲器接收和发送的控制信号(EN) ;当工作在非缓冲器方式时 ,它是输入信号 ,用来指明该 8259A 是作为主片工作($SP/EN = 1$) ,还是作为从片工作($SP/EN = 0$)。

INT : 中断请求信号输出引脚 ,向 CPU 发中断申请。

8259A 的读写操作见表 9.8 所示。

表 9.8 8259A 的读/写操作

A_0	D_4	D_3	\overline{RD}	\overline{WR}	\overline{CS}	操作
0			0	1	0	IRR、ISR 或中断级别送数据总线(*)
1			0	1	0	IMR 送数据总线
0	0	0	1	0	0	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—ICW1
1	x	x	1	0	0	数据总线—OCW1、ICW2、ICW3、ICW4(**)
x	x	x	1	1	x	高阻

* IRR、ISR 或中断级别的选择 取决于在读操作前写入 OCW3 的内容。

** 以一定的顺序来区分。

4. 8259A 实验举例

(1) 实验程序框图

主程序、中断服务程序的框图见图 9.32 所示。

(2) 实验程序举例

```

1                                assume cs : code
2 0000                            code      segment    public
3                                org       100h
4 0100  BA 03F0      start :   mov        dx ,3F0h
5 0103  B8 0013                mov        ax ,13h
6 0106  EF                   out       dx ,ax
7 0107  BA 03F2                mov        dx ,3F2h
8 010A  B8 0080                mov        ax ,80h
9 010D  EF                   out       dx ,ax      ;ICW2 set int type 80h
10 010E B8 0001               mov        ax ,01
11 0111  EF                   out       dx ,ax      ;ICW4
12                                init 8259
13 0112  B8 0000               mov        ax ,0h
14 0115  EF                   out       dx ,ax      ;set int enable
15 0116  B8 0000               mov        ax ,0
16 0119  8E D8                 mov        ds ,ax
17 011B  BE 0200               mov        si ,200h   ;int enter 0 + 200h
18 011E  B8 0136r              mov        ax ,offset hint
19 0121  89 04                 mov        ds :[ si ],ax
20 0123  83 C6 02               add        si ,2

```

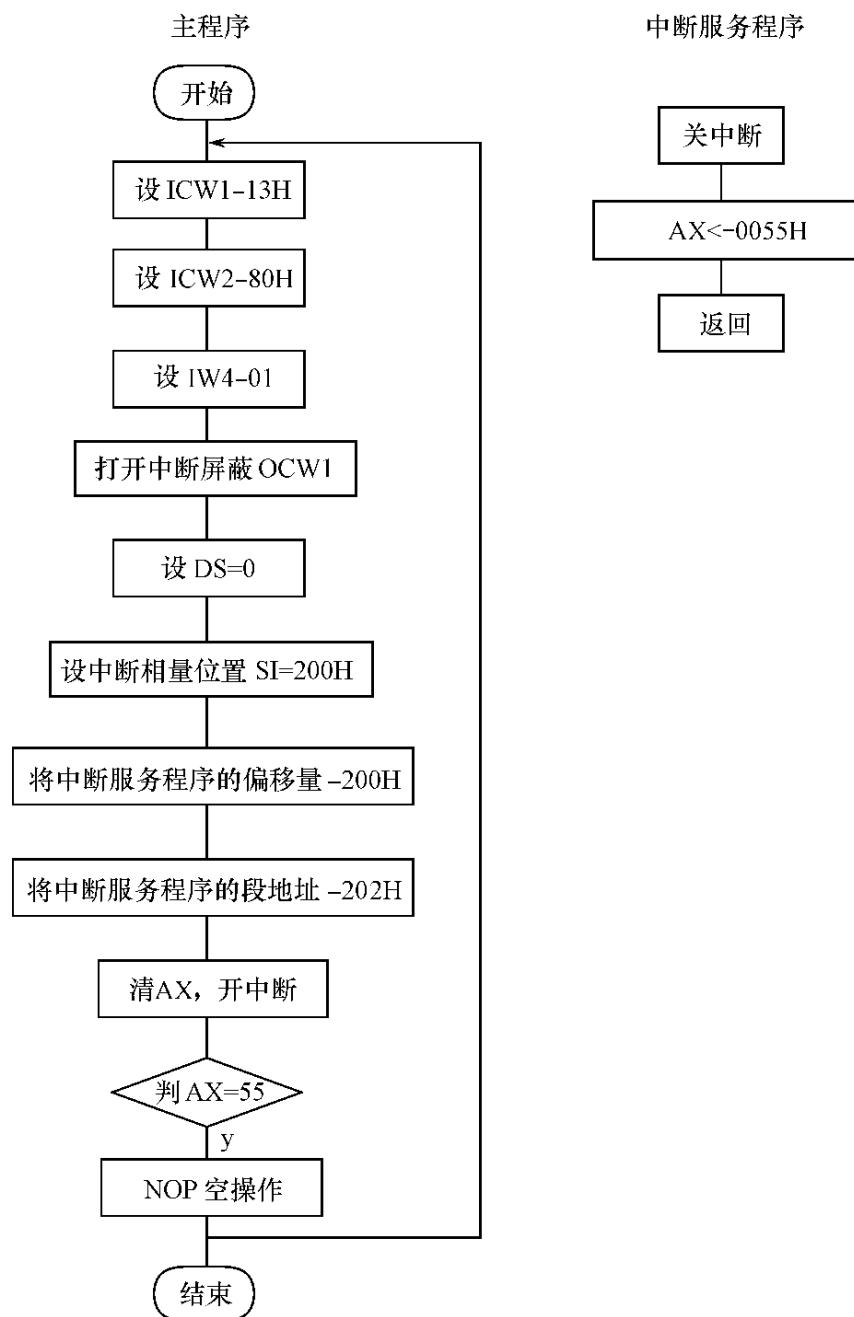


图 9.32 8259A 实验程序框图

21 0126 C7 04 0100	mov	ds :[si] 0100h
22 012A B8 0000	mov	ax 0
23 012D FB	sti	
24 012E 3D 0055	waiting :	cmp ax 55h
25 0131 75 FB	jne	waiting
26 0133 90	nop	
27 0134 EB	jmp	start
28 0136 B8 0055	hint :	mov ax 55h
29 0139 90	nop	
30 013A FA	cli	

31	013B	CF		iret
32	013C		code	ends
33			end	start

9.3.2 DMA 技术与 8237DMA 控制器

直接存储器存取方式 ,即 DMA 方式 ,指存储器与外设在 DMA 控制器控制下 ,直接传送数据而不通过 CPU ,传输速率主要取决于存储器的存取速度 ,IBM PC/AT 机中用了两片 8237A ~ 5 提供 7 个 DMA 通道。

1. 8237A 的功能

在 8237A 内部有 4 个独立的 DMA 通道 ,每一通道的 DMA 请求都可以分别允许和禁止 ,有不同的优先权 ,每一通道一次传送的最大长度可达 64 KB。

2. 8237A 的结构及引脚

8237A 的内部包括 4 个独立的通道 ,每个通道包括一个 16 位的寄存器 ,它们是基地址寄存器、现行地址寄存器、基本字节数寄存器和现行字节数寄存器 ;还包括一个 8 位的模式寄存器 4 个通道公用控制寄存器和状态寄存器。

基地址寄存器用来存放本通道 DMA 传输时的地址初值。这个初值是在 CPU 编程时写入的。编程时 ,初值同时被写入现行地址寄存器。现行地址寄存器的值在每次 DMA 传输后自动加 1 或减 1。CPU 可以用输入指令分两次读出现行地址寄存器中的值 ,每次读 8 位。但基地址寄存器中的值不能读出。当一个通道被置成自动预置模式 ,一旦现行字节数寄存器内容减至 0 时基地址寄存器内容会自动复制到现行地址寄存器中。

基本字节数寄存器用来存放 DMA 传输字节数的初值。它是芯片初始化时由 CPU 写入的 ,此值也同时被写入现行字节数计数器。在 DMA 传送时 ,每传送一个字节 ,现行字节数计数器内容减 1 ,当减至 0 时 ,产生 DMA 传输结束信号 ,在 EOP 引线上输出一个有效脉冲。若通道被置成自动预置模式 ,基本字节数寄存器内容会自动复制到现行字节数计数器中。现行字节数计数器内容可由 CPU 通过两条输入指令读出。

8237A 的内部寄存器如表 9.9 所示。

8237A 是 40 个引脚的双列直插式器件。现将引脚介绍如下。

CLK 时钟输入端 8237A 的时钟频率为 4 MHz 8237A - 5 为 5 MHz。

CS 片选输入端 ,低电平有效。

RESET 复位输入端。高电平有效。芯片复位时 ,屏蔽寄存器被置 1 ,其他寄存器均清 0 ,复位后 8237A 工作在空闲周期。

表 9.9 8237 内部寄存器

寄存器名	容量	数量
基地址寄存器	16 位	4
基字节数寄存器	16 位	4
现行地址寄存器	16 位	4
现行字节数计数器	16 位	4
临时地址寄存器	16 位	1
临时字节计数器	8 位	1
状态寄存器	8 位	1
命令寄存器	8 位	1
临时寄存器	8 位	1
模式寄存器	6 位	4
屏蔽寄存器	4 位	1
请求寄存器	4 位	1

READY 准备就绪信号输入端。当所用的存储器或 I/O 设备的速度比较慢时 ,设计一个等待电路使 *READY* 端在 S_3 状态后处于低电平 ,则 8237A 插入 S_W 状态 ,直至 *READY* 线有效(高电平)才进入 S_4 状态完成数据传送。

ADSTB 地址选通输出信号 ,高电平有效。此信号有效时 ,DMA 控制器把当前地址寄存器中的高 8 位地址(通过 $DB_0 \sim DB_7$)锁存到外部锁存器中。

AEN 地址允许输出信号 ,高电平有效。*AEN* 使外部地址锁存器中锁存的高 8 位地址送到地址总线上 ,与芯片直接输出的低 8 位地址共同构成内存单元地址的偏移量。*AEN* 信号也使与 CPU 相连的地址锁存器无效 ,这样 ,就保证了地址总线上的信号是来自 DMA 控制器 ,而不是来自 CPU 的。

MEMR 存储器读信号 ,低电平有效。此信号有效时 ,所选中的存储器单元的内容被读到数据总线。

MEMW 存储器写信号 ,低电平有效。此信号有效时 ,数据总线上的内容被写入选中的存储单元。

IOR 输入/输出设备读信号 ,双向 ,三态 ,低电平有效。在 DMA 控制器作为从设备时 ,*IOR* 作为输入控制信号送入 DMA 控制器 ,此信号有效时 ,CPU 读取 DMA 控制器中寄存器的值 ;在 DMA 控制器作为主设备时 ,*IOR* 作为输出控制信号由 DMA 控制器送出 ,此信号有效时 ,I/O 接口部件中的数据被读出送往数据总线。

IOW 输入/输出设备写信号。和 *IOR* 类似 ,在 DMA 控制器作为从设备时 ,*IOW* 的方向是送入 DMA 控制器 ,此信号有效时 ,CPU 往 DMA 控制器的内部寄存

器中写入的信息,即进行编程;在 DMA 控制器作为主设备时,IOW 的方向是由 DMA 控制器送出的,此信号有效时,存储器中读出的数据被写入 I/O 接口中。

\overline{EOP} 为 DMA 传输过程结束信号。 \overline{EOP} 是双向的,低电平有效。当由外部往 DMA 控制器送一个 \overline{EOP} 信号时,DMA 传输过程被外部强迫性地结束;另一方面,当 DMA 控制器的任一通道中计数结束时,会从 EOP 引脚输出一个有效电平,作为 DMA 传输结束信号。不论是从外部终止 DMA 过程,还是内部计数结束引起终止 DMA 过程,都会使 DMA 控制器内部的请求寄存器复位。

$DREQ$ 通道 DMA 请求输入信号,每个通道对应一个 $DREQ$ 信号端。 $DREQ$ 的极性可通过编程来选择。当外设的 I/O 接口要求 DMA 传输时,便使 $DREQ$ 处于有效电平,直到 DMA 控制器送来 DMA 响应信号 $DACK$ 以后,I/O 接口才能撤除 $DREQ$ 的有效电平。

$DACK$ 是 DMA 控制器送给 I/O 接口的回答信号,每个通道对应一个 $DACK$ 信号端。DMA 控制器获得 CPU 送来的总线允许信号 $HLDA$ 以后,便产生 $DACK$ 信号送到相应的外设接口。 $DACK$ 信号的极性也是可以通过编程选择的。

HRQ 总线请求信号。当外设的 I/O 接口要求 DMA 传输时,往 DMA 控制器发送 $DREQ$ 信号,如果相应通道的屏蔽位为 0,则 DMA 控制器的 HRQ 端输出为有效电平,从而向 CPU 发总线请求。

$HLDA$ 总线响应信号。DMA 控制器向 CPU 发总线请求信号 HRQ 以后,至少再过一个时钟周期,CPU 才能发出总线响应信号 $HLDA$,这样,DMA 控制器便可获得总线控制权。 $HLDA$ 在许多时候也称为总线保持回答信号。

$A_3 \sim A_0$ 最低的 4 位地址线,它们是双向信号端。在 DMA 控制器作为从设备时,它们作为输入端对 DMA 控制器内部寄存器进行寻址,这样,CPU 可以对 8237 编程。在 8237A 作为主设备时,这 4 条线输出低 4 位地址。

$A_7 \sim A_4$ 三态地址输出线。在 DMA 传送时输出高 4 位地址。在 8237A 作为从设备时处于高阻状态。

$DB_7 \sim DB_0$ 8 位双向三态数据线,与系统总线相连。在 8237A 作为从设备时,CPU 可通过使 IOR 有效,从数据总线上读取 8237A 内部寄存器内容,也可能通过使 IOW 有效从数据总线输出控制字,对 8237A 编程。在 8237A 为主设备控制 DMA 传送时, $DB_7 \sim DB_0$ 输出当前地址寄存器中的高 8 位地址,并通过信号 $ADSTB$ 打入外部锁存器中,这样和 $A_7 \sim A_0$ 输出的低 8 位地址构成 16 位地址。

3. 8237A 的实验举例

(1) 实验要求

用 DMA 方式将 02000H 到 020FFH 共 100H 个字节的传送到 02100H。

(2) 实验程序

```

1                               assume cs : code
2     0000                      code    segment public
3                               org 100h
4     0100  BB 0000             start :  mov      bx ,0
5     0103  BE 1000             mov      si ,1000h
6     0106  B0 55               mov      al ,55h
7     0108  B9 0100             mov      cx ,100h
8     010B  88 04              fil :   mov      [ si ] ,al
9     010D  46                 inc      si
10    010E  E2 FB              loop    fil
11    0110  BE 1100             mov      si ,1100h
12    0113  B0 00               mov      al ,0h
13    0115  B9 0100             mov      cx ,100h
14    0118  88 04              fill :  mov      [ si ] ,al
15    011A  46                 inc      si
16    011B  E2 FB              loop    fill
17    011D  B9 0100             mov      cx ,100h
18    0120  BA 03FA             rdma :  mov      dx ,3fah
19    0123  B8 0000             mov      ax ,0
20    0126  EF                 out     dx ,ax      ;clear main
21    0127  BA 03E0             mov      dx ,3e0h
22    012A  8B C3               mov      ax ,bx
23    012C  EF                 out     dx ,ax
24    012D  B8 0020             mov      ax ,20h
25    0130  EE                 out     dx ,al      ;source
26    0131  BA 03E4             mov      dx ,3e4h
27    0134  8B C3               mov      ax ,bx
28    0136  EF                 out     dx ,ax
29    0137  B8 C3               mov      ax ,21h      ;destr
30    013A  EF                 out     dx ,ax
31    013B  BA 06E6             mov      dx ,3e6h
32    013E  B8 0000             mov      ax ,0
33    0141  EF                 out     dx ,ax
34    0142  B8 0000             mov      ax ,0
35    0145  EF                 out     dx ,ax      ;length
36    0146  BA 03F6             mov      dx ,3f6h      ;mode 0

```

37	0149	B8	0088		mov	ax 88h
38	014C	EF			out	dx ,ax
39	014D	B8	0085		mov	ax 85h ;mode 1
40	0150	EF			out	dx ,ax
41	0151	BA	03FE		mov	dx 3feh ;mask reg
42	0154	B8	000F		mov	ax 0fh
43	0157	EF			out	ax 3h
44	0158	BA	03F0		mov	dx 3f0h ;command
45	015B	B8	0003		mov	ax 3h
46	015E	EF			out	dx ,ax
47	015F	BA	03F2		mov	dx 3f2h ;request
48	0162	B8	0004		mov	ax 4
49	0165	EF			out	dx ,ax
50	0166	90			nop	
51	0167	90			nop	
52	0168	43			inc	bx
53	0169	E2	B5		loop	rdma
54	016B	90			nop	
55	016C			code	ends	
56					end	start

附录

附录一 8086 微机原理及接口实验系统

一、实验系统介绍

实验系统由功能实验板、CPU 板、两块小面包板三部分构成，安装在实验箱内。其中通用接口板由若干相对独立的功能接口电器组成，它们是：D/A 电路、A/D 电路、发光二级管电路、开关量输入电路、RAM/ROM 电路、简单 I/O 电路、8253 可编程定时器/计数器电路、8255 并行接口电路、总线驱动电路、8279 接口电路、单脉冲发生器、LED 显示电路、键盘电路、复位电路、8250 串行接口电路，下面对通用接口板实验电路进行较为详细的介绍。

1. 输出显示电路

(1) 数码显示电路：该电路由 6 位共阴极数码管、3 片 75452、2 片 74LS07 组成，74LS07 为段驱动器，相应输入插孔为 CZ4，75452 为位驱动器，相应输入插孔为 CZ3(LD1 ,LD2 ,LD3 ,LD4 ,LD5 ,LD6)。

(2) LED 灯显示电路：该电路由 2 片 74LS04、12 支绿色二极管组成。12 只红、黄、绿色二极管相应的输入插孔为 CZ2(LI1 ,LI2 ,LI3 ,LI4 ,LI5 ,LI6 ,LI7 ,LI8 ,LI9 ,LI10 ,LI11 ,LI12)。

2. 信号发生电路

(1) 开关量输入电路：该电路由 8 只开关组成，每只开关有两个位置，一个位置代表高电平，一个位置代表低电平。该电路的输出插孔为 CZ1(K1 ,K2 ,K3 ,K4 ,K5 ,K6 ,K7 ,K8)。

(2) 时钟输入电路：该电路由 1 片 74LS161 组成。当 CPU 为 PC 总线时，输入时钟为 AT 总线的 CLK。当 CPU 为 8051、8098、80C198/C196 时，CLK 的输入时钟为晶振频率。当 CPU 为 8086 时，CLK 是 2 MHz。输出时钟为该 CLK 的 2 分频(CLK0)、4 分频(CLK1)、8 分频(CLK2)、16 分频(CLK3)，相应输出插孔 CZ47(CLK0 ,CLK1 ,CLK2 ,CLK3)。

(3) 单脉冲发生器电路：该单脉冲发生器由一个按钮，1片74LS04,74LS132组成，具有消颤功能，正反相脉冲，相应输出插孔CZ3(CP₀,P₁)。

(4) 模拟量输入电路：该电路由3只可变电位器组成，输出为0~5V连续可调。相应的输出插孔CZ29,CZ28,CZ27(KB1,KB2,KB3)。

(5) 键盘输入电路：该电路由28只通用键，1只Shift键，1只Ctrl键组成，28只通用键采用8根列扫线，4根行扫线，无外部信号输入时，均为高电平，有外部信号输入时，电平状态由外部输入信号决定，作键盘实验时，一般行、列扫线分别定义为输入、输出，即8根列扫线的插孔为CZ6(KA0,KA1,KA2,KA3,KA4,KA5,KA6,KA7)4根行扫线的插孔为CZ5(KB0,KB1,KB2,KB3)。

(6) 复位电路：按动复位键，将对8255,8279,8250复位，对8051,80C198,8086CPU板起复位作用。

3. 可编程定时器8253电路

该电路由1片8253组成，8253的片选输入端插孔CS8253\，数据口，地址，读/写线均已接好，T0,T1时钟输入为CLK3, T2的时钟用户可自己接。定时器输出，GATE控制孔对应如下：OUT0、GATE0、GATE1、OUT2、GATE2、CLK2。

注：GATE信号无输入时为高电平。EL-1型T2的时钟为CLK3。

4. 可编程并行口8255电路

该电路由1片8255组成，8255的数据口，地址，读/写线，复位控制线均已接好，片选输入端插孔为CS8255\，A,B,C三端口的插孔分别为：

A : CZ16(PA0,PA1,PA2,PA3,PA4,PA5,PA6,PA7)

B : CZ15(PB0,PB1,PB2,PB3,PB4,PB5,PB6,PB7)

C : CZ17(PC0,PC1,PC2,PC3,PC4,PC5,PC6,PC7)

5. 可编程键盘显示控制器

该电路由1片8279,74LS138组成，8279的数据口，地址，读/写线，复位，时钟，片选都已经接好，显示输出，键盘行列扫描线均有插孔输出。

两组显示输出的插孔标号与芯片标号一致。键盘行扫描线译码线插孔标号为CZ21,CZ25(完全相同)，相应标号为(KS0,KS1,KS2,KS3,KS4,KS5,KS6,KS7)。

另外该电路在进行实验时还要用到一组反相器，该反相器是输入插孔标号CZ24(S0,S1,S2,S3,S4,S5)输出插孔标号CZ23(S0\,S1\,S2\,S3\,S4\,S5\)。

6. 可编程串行口8050电路

该电路由1片8250,1片1488,1489组成，该电路中8250的输入端均已接好，串行口输入/输出插座为J1，T为发送，R为接收。

7. 简单数字量输出缓存实验电路

该电路由 2 片 74LS273 及一片 74LS04 ,74LS02 组成 ,该电路中 74LS273 的输入均已接好数据线 ,锁存端片选入控制 ,两片 74LS273 的片选输入插孔标号分别为 CSU8 \ ,CSU9 \ \ ,输出插孔标号分别为(SO0 ,SO1 ,SO2 ,SO3 ,SO4 ,SO5 ,SO6 ,SO7 ,SO8 ,SO9 ,SO10 ,SO11 ,SO12 ,SO13 ,SO14 ,SO15)。

8. 简单数字量输入缓冲实验电路

该电路由一片 74LS244 ,一片 74LS02 构成 ,该电路中 ,74LS244 的输入均已接在数据总线上 输入端接着插线孔 ,其标号 CZ9(S10 ,S11 ,S12 ,S13 ,S14 ,S15 ,S16 ,S17) ,片选控制端插孔标号(CSU10 \)。

9. 8 路 8 位 A/D 实验电路

该电路由一片 ADC0809 ,一片 74LS02 组成 ,该电路中 ,ADC0809 的参考电压 数据总路线输出 通道控制线均已接好 ,其他信号线由插孔接入 ,ADC0809 的片选控制插孔标号为 CZ31(CS0809 \),转换结束标志输出插孔标号为 CZ26 (EOC) 模拟量输入通道孔标号为(IN0 ,IN1 ,IN2 ,IN3 ,IN4 ,IN5 ,IN6 ,IN7)

10. 8 位双缓冲 D/A 实验电路 0832

该电 路由一片 DAC0832 ,一片 74LS00 ,一片 LM324 组成 ,该电 路中除 DAC0832 的片选未接好外 其他信号均已接好 ,片选插孔标号 CZ32(CS0832 \) ,输出插孔标号 CZ33(VOUT)。该电 路为非偏移二进制 D/A 转换电 路 ,通过调节 KB4 ,可调节 D/A 转换器的满偏值 ,调节 KB5 ,可调节 D/A 转换器的零偏值。

11. 存储器扩展实验电 路

该电 路由一片 62256 和一片 6264 组成 ,该电 路的所有信号线均已接好 ,可直接进行存储器读/写实验。对 8051 和 8098 实验来说 ,62256 的起始地址为 :4000H ,长度为 32K ,6264 系统已使用。8051 和 8098 的数据与程序存储区都为已编址 64K ,其中 0 ~ 16K 为系统使用 ,后 48K 用户作为仿真可实验用。

二、实验练习

1. 简单 I/O 口扩展实验

利用 74LS244 和 74LS273 扩展 I/O 口 ,掌握用锁存器、三态门扩展简单并行输入、输出口的方法。

2. 8255 并行口实验

8255A 的 PA0 ~ PA7 分别与逻辑电平开关电路的 K1 ~ K8 相连 ;PB0 ~ PB7 分

别与发光二极管电路的 L1 ~ L8 相连。从 CS0 \ ~ CS7 \ 中任选一个与 8255A 的片选(CS8255)端相连(如 CS0 \)其他线路均已连好。

3. 8253 定时器/计数器接口实验

编程将计数器 0、1、2 设置为模式 2(分频方式), 用示波器或 A/D、D/A 卡观察不同模式下的输出波形。

4. A/D 实验

熟悉 A/D 转换的基本原理 掌握 ADC0809 的使用方法。

5. D/A 实验

熟悉 D/A 转换的基本原理 掌握 DAC0832 的使用方法。

6. 8250 串行实验

在实验箱与 PC 机之间实现串行通信 , 对 8250 芯片进行编程 , 从而了解 PC – 232 串行接口标准及连接方法。

7. DAM 实验

利用 6264 芯片掌握 8086 十六位数据存储的方法。

8. 8259 中断控制器实验

利用 8259A 中断控制器芯片编写中断服务程序 掌握初始化中断向量的方法。

9. 8279 键盘扩展实验

将 8279 的扫描码放入 BX 寄存器 程序每循环一次 将读键码一次。

10. 8279 显示器接口实验

利用 8279 实现在 6 位 LED 上循环显示 8。

11. DMA 实验

对 DMA 控制器 8237 – 5 进行编程实验 , 将 02000HGC 02FFH 字节的数据传送到 02100H 。

附录二 8086 指令系统

8086/8088 指令系统一览表

助记符	汇编语言格式	功能	操作数	时间周期数	字节数	标志位 ODITSZAPC
MOV	MOV dst,src	(dst) \leftarrow (src)	mem ,ac	10	3	
			ac ,mem	10	3	
			reg ,reg	2	2	
			reg ,mem	8 + EA	2 - 4	
			mem ,reg	9 + EA	2 - 4	
			reg ,data	4	2 - 3	
			mem ,data	10 + EA	3 - 6	
			segreg ,reg	2	2	
			segreg ,mem	8 + EA	2 - 4	
			reg ,segreg	2	2	
PUSH	PUSH src	(sp) \leftarrow (sp) \leftarrow 2 ((sp)+1 (sp)) \leftarrow (src)	reg	11	1	
			segreg	10	1	
			mem	16 + EA	2 - 4	
POP	POP src	(dst) \leftarrow ((sp)+1 (sp)) (sp) \leftarrow (sp)+2	reg	8	1	
			segreg	8	1	
			mem	17 + EA	2 - 4	
XCGH	XCGH opr1 opr2	(opr1) $\leftarrow\rightarrow$ (opr2)	reg ,ac	3	1	
			reg ,mem	17 + EA	2 - 4	
			reg ,reg	4	2	

助记符	汇编语言格式	功能	操作数	时间周期数	字节数	标志位 ODITSZAPC
IN	IN ac ,port IN ac ,dx	(ac)<-(port) (ac)<-(dx)	10 8	2 1		
OUT	OUT port ,ac OUT dx ,ac	(port)<-(ac) (dx)<-(ac)		10 8	2 1	
XLAT	XLAT			11	1	
LEA	LEA reg ,src	(reg)<-src)	reg ,mem	2 + EA	2 - 4	
LDS	LDS reg ,src	(reg)<-(src) (ds)<-(src + 2)	Reg ,mem	16 + EA	2 - 4	
LES	LES reg ,src	(reg)<-(src) (ES)<-(src + 2)	reg ,mem	16 + EA	2 - 4	
LAFH	LAFH reg ,src	(AH)<-(psw 低字节)		4	1	
SAFH	SAFH	(PSW 低字节)<-AH		4	1	----- r r r r
PUSHF	PUSHF	(sp)<-(SP)- 2 ((sp)+ 1 (sp))<-(psw)		10	1	
POPF	POPF	(psw)<-((sp)+ 1 (sp)) (sp)<-(sp)+ 2		8	1	r r r r r r r r
ADD	ADD dst ,src	(dst)<-(src)+(dst)	reg ,reg reg ,mem mem ,reg reg ,data mem ,data ac ,data	3 9 + EA 16 + EA 4 17 + EA 4	2 2 - 4 2 - 4 3 - 4 3 - 6 2 - 3	x --- x x x x x

助记符	汇编语言格式	功能	操作数	时间周期数	字节数	标志位 ODITSZAPC
ADC	ADC drc src	dst \leftarrow (src) + (dst) + CF	reg ,reg reg ,mem mem ,reg reg ,data mem ,data ac ,data	3 9 + EA 16 + EA 4 17 + EA	4 2 2 - 4 2 - 4 3 - 4 3 - 6 2 - 3	x --- x x x x x
INC	INC opr	(opr) \leftarrow (opr) + 1	reg mem	2 - 3 15 + EA	1 - 2 2 - 4	x --- x x x x -
SUB	SUB dst src	(dst) \leftarrow (dst) - (src)	reg ,reg reg ,mem mem ,reg ac ,data reg ,data mem ,data	3 9 + EA 6 + EA 4 4 7 + EA	3 2 - 4 2 - 4 2 - 3 3 - 4 3 - 6	x --- x x x x x
SBB	SBB dst src	(dst) \leftarrow (dst) - (src) - CF	reg ,reg reg ,mem mem ,reg ac ,data reg ,data mem ,data	3 9 + EA 6 + EA 4 4 7 + EA	3 2 - 4 2 - 4 2 - 3 3 - 4 3 - 6	x --- x x x x x
DEC	DEC opr	(opr) \leftarrow (dst) - (src) - CF	reg mem	2 - 3 15 + EA	1 - 2 2 - 4	x --- x x x x -

助记符	汇编语言格式	功能	操作数	时间周期数	字节数	标志位 ODITSZAPC
NEG	NEG opr	(opr) \leftarrow 0 - (opr)	reg mem	3 6 + EA	2 2 - 4	x --- x x x x x
CMP	CMP opr1 ,opr2	(opr1) - (opr2)	reg ,reg reg ,mem mem ,reg reg ,data mem ,data ac ,data	3 9 + EA 9 + EA 4 10 + EA 4	2 2 - 4 2 - 4 3 - 4 3 - 6 2 - 3	x --- x x x x x
MUL	MUL src	(AX) \leftarrow (AL)*(src) (DX ,AX) \leftarrow (AX)* (src)	8 位 reg 8 位 mem 16 位 reg 16 位 mwm	70 - 77 (76 - 83)+ EA 118 - 133 (124 - 139)+ EA	2 2 - 4 2 2 - 4	x --- u u u u x
IMUL	IMUL src	(AX) \leftarrow (AX)*(src) (DX ,AX) \leftarrow (AX)* (src)	8 位 reg 8 位 mem 16 位 reg 16 位 mwm	80 - 98 (86 - 104)+ EA 128 - 154 (134 - 160)+ EA	2 2 - 4 2 2 - 4	x --- u u u u x
DIV	DIV src	(AL) \leftarrow (AX \ src)的商 (AH) \leftarrow (AX)(SRC)的余数 (AX) \leftarrow (DX ,AX \ src)的商 (DX) \leftarrow (DX ,AX)/ (src)的余数	8 位 reg 8 位 mem 16 位 reg 16 位 mwm	80 - 90 (86 - 96)+ EA 144 - 162 (150 - 168)+ EA	2 2 - 4 2 2 - 4	x --- u u u u u

助记符	汇编语言格式	功能	操作数	时间周期数	字节数	标志位 ODITSZAPC
IDIV	IDIV src	(AX) \leftarrow (AX)(src)的商 (AH) \leftarrow (AX)(src)的余数 (AX) \leftarrow (DX ,AX)(src)的商 (DX) \leftarrow (DX ,AX)(src)的余数	8 位 reg 8 位 mem 16 位 reg 16 位 mwm	101 – 112 107 – 118)+ EA 165 – 184 (171 – 190)+ EA	2 2 – 4 2 2 – 4	u --- u u u u u
DAA	DAA	(AL) \leftarrow 把 AL 的和调到压缩 BCD 格式		4	1	u --- x x x x x
DAS	DAS	(AL) \leftarrow 把 AL 的差调到非压缩的 BCD 格式	4	1	u --- x x x x x	
AAA	AAA	(AL) \leftarrow 把 AL 的和调整到非压缩的 BCD 码格式。 (AH) \leftarrow (AH) – 调整产生的进位值	4	1	u --- u u x u x	
AAS	AAS	(AL) \leftarrow 把 AL 的差调整到非压缩的 BCD 格式。 (AH) \leftarrow (AH) – 调整产生的借位值	4	1	u --- u u x u x	
AAM	AAM	(AX) \leftarrow 把 AH 的积调整到非压缩的 BCD 码格式	83	2	u --- u u x u x	
AAD	AAD	(AL) \leftarrow 10 * (AH) + (AL) (AH) \leftarrow 0 实现除法的非压缩的 BCD 调整		60	2	u --- u u x u x

助记符	汇编语言格式	功能	操作数	时间周期数	字节数	标志位 ODITSZAPC
AND	AND dst ,src	(dst) \leftarrow (dst) \wedge (src)	reg ,reg reg ,mem mem ,reg reg ,data mem,data ac ,data	3 9 + EA 16 + EA 4 17 + EA 4	2 2 - 4 2 - 4 3 - 4 3 - 6 2 - 3	0 --- x x u x 0
OR	OR dst ,src	(dst) \leftarrow (dst) \vee (src)	reg ,reg reg ,mem mem ,reg ac ,data reg ,data mem ,data	3 9 + EA 16 + EA 4 4 17 + EA	2 2 - 4 2 - 4 2 - 3 3 - 4 3 - 6	0 --- x x u x 0
NOT	NOT opr	(opr) \leftarrow (opr)	reg , mem	3 16 + EA	2 2 - 4	
XOR	XOR dst ,src	(dst) \leftarrow (dst) \wedge (src)	reg ,reg reg ,mem mem ,reg ac ,data reg ,data mem ,data	3 9 + EA 16 + EA 4 4 17 + EA	2 2 - 4 2 - 4 2 - 3 3 - 4 3 - 6	0 --- x x u x 0

助记符	汇编语言格式	功能	操作数	时间周期数	字节数	标志位 ODITSZAPC
TEST	TEST opr1 ,opr2	(opr1)\/(opr2)	reg ,reg reg ,mem ac ,data reg ,data mem ,data	3 9 + EA 4 5 11 + EA	2 2 - 4 2 - 3 3 - 4 3 - 6	0 --- x x u x 0
SHL	SHL opr1 SHL opr ,CL	逻辑左移	reg mem reg mem	2 15 + EA 8 + 4/位 20 + EA + 4/位	2 2 - 4 2 2 - 4	x --- x x u x 0
SAL	SAL opr ,1 SAL opr ,CL	算术左移	reg mem reg mem	2 15 + EA 8 + 4/位 20 + EA + 4/位	2 2 - 4 2 2 - 4	x --- x x u x x
SHR	SHR opr ,1 SHR opr ,CL	逻辑右移	reg mem reg mem	2 15 + EA 8 + 4/位 20 + EA + 4/位	2 2 - 4 2 2 - 4	x --- x x u x x
SAR	SAR opr ,1 SAR opr ,CL	算术右移	reg mem reg mem	2 15 + EA 8 + 4/位 20 + EA + 4/位	2 2 - 4 2 2 - 4	x --- x x u x x

助记符	汇编语言格式	功能	操作数	时间周期数	字节数	标志位 ODITSZAPC
ROL	ROL opr ,l ROL opr ,CL	循环左移	reg mem reg mem	2 15 + EA 8 + 4/位 20 + EA + 4/位	2 2 - 4 2 2 - 4	x ----- x
ROR	ROR opr ,l ROR opr ,CL	循环右移	reg mem reg mem	2 15 + EA 8 + 4/位 20 + EA + 4/位	2 2 - 4 2 2 - 4	x ----- x
RCL	RCL opr ,l RCL opr ,CL	带进位循环左移	reg mem reg mem	2 15 + EA 8 + 4/位 20 + EA + 4/位	2 2 - 4 2 2 - 4	x ----- x
RCR	RCR opr ,l RCR opr ,CL	带进位循环右移	reg mem reg mem	2 15 + EA 8 + 4/位 20 + EA + 4/位	2 2 - 4 2 2 - 4	x ----- x
MOVS	MOVSB MOVSW	((DI))<((SI)) (SI)<(SI)± 1 或 2 (DI)<(DI) ± 1 或 2		不重复 : 18 重复 : 9 + 17/rep	1	
STOS	STOSB STOSW	((DI))<(AC) (DI)<(DI)± 1 或 2		不重复 : 11 重复 : 9 + 10/rep	1	
LODS	LODSB LODSW	(AC)<((SI)) (SI)<(SI) ± 1 或 2		不重复 : 12 重复 : 9 + 13/rep	1	

助记符	汇编语言格式	功能	操作数	时间周期数	字节数	标志位 ODITSZAPC
REP	REP string primitive	当(CX)= 0 ,退出重复 ;否则 (CX)←(CX)- 1 执行其后的串指令		2	1	
CMPS	CMPSB CMPSW	((DI))-((SI)) (SI)←(SI)± 1 或 2 (DI)←(SI)± 1 或 2		不重复 :22 重复 :9 + 22/rep	1	x --- x x x x x
SCAS	SCASB SCASW	(AC)←((DI)) (DI)←(DI) ± 1 或 2		不重复 :15 重复 :9 + 15/rep	1	
REPE 或 REPZ	REPE/REPX string primitive	当(CX)= 0 或 ZF = 0 退出重 复 ;否则 (CX)←(CX)- 1 执 行其后的串指令		2	1	
REPNE 或 REPNZ	REPNE/REP NX string primitive	当(CX)= 0 或 ZF = 1 退出重 复 ;否则 (CX)←(CX)- 1 执 行其后的串指令		2	1	
JPM	JMP short opr JMP near ptr opr JMP far ptr opr JMP word ptr opr JMP dword ptr opr	无条件转移	reg mem	15 15 15 11 18 + EA 24 + EA	2 3 5 2 2 - 4 2 - 4	-----
JZ 或 JE	JZ/JE opr	ZF = 1 则转移		16/4	2	-----
JNZ 或 JNE	JNZ/JNE opr	ZF = 0 则转移		16/4	2	-----
JS	JS opr	SF = 1 则转移		16/4	2	-----
JNS	JNS opr	SF = 0 则转移		16/4	2	-----

助记符	汇编语言格式	功能	操作数	时间周期数	字节数	标志位 ODITSZAPC
JO	JO opr	OF = 1 则转移		16/4	2	-----
JNO	JNO opr	OF = 0 则转移		16/4	2	
JP 或 LPE	LP/LPE opr	PF = 1 则转移		16/4	2	-----
LNP LPO	LNP/LPO opr	PF = 0 则转移		16/4	2	-----
JC 或 JB 或 JNAE	JC/JB/JNAE opr	cF = 1 则转移		16/4	2	-----
JNC 或 JNB 或 JAE	JNC/JNB/JNE opr	CF = 0 则转移		16/4	2	-----
JBE 或 JNA	JBE/JNA opr	CF ∨ ZF = 1 则转移		16/4	2	-----
JNBE 或 JA	JNBE/JA opr	CF ∨ ZF = 0 则转移		16/4	2	-----
JL 或 JNGE	JL/JNGE opr	SF^OF = 1 则转移		16/4	2	-----
JNL 或 JGE	JNL/JNGE opr	SF^OF = 0 则转移		16/4	2	-----
JLE 或 JNG	JLE/JNG opr	(SF^OF) ∨ ZF = 1 则转移		16/4	2	-----
JNLE 或 JG	JNLE/JG opr	(SF^OF) ∨ ZF = 0 则转移		16/4	2	-----
JCXZ	JCX opr	((CX) = 0 则转移		18/6	2	-----
LOOP	LOOP opr	((CX) ≠ 0 则循环		17/5	2	-----
LOOPZ 或 LOOPE	LOOPZ/LOOPE opr	ZF = 1 且 (CX) ≠ 0 则循环		18/6	2	-----
LOOPNZ 或 LOOPNE	LOOPNZ/LOOPNE opr	ZF = 0 且 (CX) ≠ 0 则循环		19/5	2	-----

助记符	汇编语言格式	功能	操作数	时间周期数	字节数	标志位 ODITSZAPC
CALL	CALL dst	段内直接： $(SP) \leftarrow (SP)-2$ $((SP)+1(SP)) \leftarrow (IP)$ $(IP) \leftarrow (IP)+D16$ 段内间接： $(SP) \leftarrow (SP)-2$ $((SP)+1(SP)) \leftarrow (IP)$ $(IP) \leftarrow EA$	Reg mem	19 16 $21 + EA$	3 2 $2 - 4$	-----
		段内直接： $(SP) \leftarrow (SP)-2$ $((SP)+1(SP))-(CS)$ $(SP) \leftarrow (SP)-2$ $(IP) \leftarrow \text{转向偏移地址}$ $(CS) \leftarrow \text{转向段地址}$ 段内间接： $(SP) \leftarrow (SP)-2$ $((SP)+1(SP)) \leftarrow (CS)$ $(SP) \leftarrow (SP)-2$ $((SP)+1(SP)) \leftarrow (IP)$ $(IP) \leftarrow (EA)$ $(CS) \leftarrow (EA+2)$	28 37 + EA	5 $2 - 4$		

助记符	汇编语言格式	功能	操作数	时间周期数	字节数	标志位 ODITSZAPC
RET	RET RET exp	<p>段内：</p> $(IP) \leftarrow ((SP) + 1 \& SP)$ $(SP) \leftarrow (SP) + 2$ <p>段间：</p> $(IP) \leftarrow ((SP) + 1 \& SP)$ $(SP) \leftarrow (SP) + 2$ $(CS) \leftarrow ((SP) + 1 \& SP)$ $(SP) \leftarrow (SP) + 2$ <p>段内：</p> $(IP) \leftarrow ((SP) + 1 \& SP)$ $(SP) \leftarrow (SP) + 2$ <p>段间：</p> $(IP) \leftarrow ((SP) + 1 \& SP)$ $(SP) \leftarrow (SP) + 2$ $(CS) \leftarrow ((SP) + 1 \& SP)$ $(SP) \leftarrow (SP) + 2$ $((SP) \leftarrow (SP) + D16$	16 24 20 23		1 1 3 3	-----

助记符	汇编语言格式	功能	操作数	时间周期数	字节数	标志位 ODITSZAPC
INT	INT type INT (TYPE = 3 时)	(SP) \leftarrow (SP) - 2 ((SP) + 1 (SP)) \leftarrow (PSW) (SP) \leftarrow (SP) - 2 ((SP) + 1 (SP)) \leftarrow (CS) (SP) \leftarrow (SP) - 2 ((SP) + 1 (SP)) \leftarrow (IP) (IP) \leftarrow (TYPE * 4) (CS) \leftarrow (TYPE * 4 + 2)	TYPE ≠ 3 TYPE = 3	52 51	1 2	-- 0 0 -----
INTO	INTO	若 OF = 1 则 (SP) \leftarrow (SP) - 2 ((SP) + 1 (SP)) \leftarrow (PSW) (SP) \leftarrow (SP) - 2 ((SP) + 1 (SP)) \leftarrow (CS) (SP) \leftarrow (SP) - 2 ((SP) + 1 (SP)) \leftarrow (IP) (IP) \leftarrow (10H) (CS) \leftarrow (12H)		53 (OF = 1) (OF = 0)	1	-- 0 0 -----
IRET	IRET	(IP) \leftarrow ((SP) + 1 (SP)) (SP) \leftarrow (SP) + 2 (CS) \leftarrow ((SP) + 1 (SP)) (SP) \leftarrow (SP) + 2 (PSW) \leftarrow ((SP) + 1 (SP)) (SP) \leftarrow (SP) + 2		24	1	rr r r r r r r r

助记符	汇编语言格式	功能	操作数	时间周期数	字节数	标志位 ODITSZAPC
CBW	CBW	(AL) 符号扩展到(AH)		2	1	-----
XWD	XWD	(AX) 符号扩展到(DX)		5	1	-----
CLC	CLC	进位位置 0		2	1	-----0
CMC	CMC	进位位求反		2	1	-----x
STC	STC	进位位置 1		2	1	-----1
CLD	CLD	方向标志置 0		2	1	-0-----
STD	STD	方向标志置 1		2	1	-1-----
CLI	CLI	中断标志置 0		2	1	--0-----
STI	STI	中断标志置 1		2	1	--1-----
NOP	NOP	无操作		3	1	-----
HLT	HLT	停机		2	1	-----
WAIT	WAIT	等待		3 或更多	1	-----
WSC	WSC ESC mem	换码		8 + EA	2 - 4	-----
LOCK	LOCK SEGREG	封锁 段前缀		2 2	1 1	-----

(注 : 符号说明如下 : 0——置 0 ; 1——置 1 ; x——根据结果设置 ; -——不影响 ; u——无定义 ; r——恢复原先保存的值)

参 考 文 献

1. 王尔乾 ,巴林凤 . 数字逻辑及数字集成电路 . 北京 清华大学出版社 ,1994 年
2. 王玉龙 . 数字逻辑 . 北京 高等教育出版社 ,1987 年
3. 刘宝琴 . 数字电路与系统 . 北京 清华大学出版社 ,1993 年
4. 阎石 . 数字电子技术基础 . 北京 高等教育出版社 ,1981 年
5. Walter A. Triebel 著 . 80X86/Pentium 处理器硬件、软件及接口技术教程 . 王克义 ,王钧 ,方晖 等译 . 北京 清华大学出版社 ,1999 年
6. 戴梅萼 . 微型计算机技术及应用 . 北京 清华大学出版社 ,1991 年
7. 曾家智 . 微型计算机系统与接口 . 北京 电子科技大学出版社 ,1992 年
8. 宋焕章 , 张春元 , 王保恒 . 计算机原理与设计 . 长沙 国防科技大学出版社 ,2000 年
9. 孙德文 . 微型计算机技术 . 北京 高等教育出版社 ,2001 年
10. Michael Hordeski . Personal Computer Interfaces . Macmillan : McGRAW-Hill ,1995 .