教育科学“十五”国家规划课题研究成果

微机原理及应用

徐晨　陈继红　王春明　徐慧　编著

高等教育出版社

内容提要

本书是教育科学“十五”国家规划课题研究成果。全书共13章，包括：基础知识，微型计算机概论，8086/8088指令系统与寻址方式，汇编语言程序设计，8086的总线操作和时序，半导体存储器，基本输入输出接口，中断，可编程接口芯片及应用，串行通信，模数、数模转换，高性能微处理器，总线标准与微型计算机。

本书内容全面、实用性强，讲述有特点和新意，同时，配以较多的程序设计实例和接口电路实例。

本书适用于工科各专业本专科生“微机原理”课程，同时可供有关工程技术人员参考使用。

图书在版编目（CIP）数据

微机原理及应用/徐晨等编著. —北京：高等教育出版社，2004.7

ISBN 7-04-014564-2

Ⅰ.微… Ⅱ.徐… Ⅲ.微型计算机-高等学校-教材 Ⅳ.TP36

中国版本图书馆CIP数据核字（2004）第053388号

出版发行 高等教育出版社
社 址 北京市西城区德外大街4号
邮政编码 100011
总 机 010-82028899
购书热线 010-64054588
免费咨询 800-810-0598
网 址 http://www.hep.edu.cn
http://www.hep.com.cn

经 销 新华书店北京发行所
印 刷

开 本 787×960 1/16
印 张 31.5
字 数 580 000
版 次 年 月第1版
印 次 年 月第 次印刷
定 价 38.90元

策划编辑　李　慧
责任编辑　陈大力
封面设计　李卫青
责任绘图　朱　静
版式设计　范晓红
责任校对　尤　静
责任印制

总　　序

为了更好地适应当前我国高等教育跨越式发展需要，满足我国高校从精英教育向大众化教育的重大转移阶段中社会对高校应用型人才培养的各类要求，探索和建立我国高等学校应用型人才培养体系，全国高等学校教学研究中心（以下简称“教研中心”）在承担全国教育科学“十五”国家规划课题——“21世纪中国高等教育人才培养体系的创新与实践”研究工作的基础上，组织全国100余所以培养应用型人才为主的高等院校，进行其子项目课题——“21世纪中国高等学校应用型人才培养体系的创新与实践”的研究与探索，在高等院校应用型人才培养的教学内容、课程体系研究等方面取得了标志性成果，并在高等教育出版社的支持和配合下，推出了一批适应应用型人才培养需要的立体化教材，冠以“教育科学‘十五’国家规划课题研究成果”。

2002年11月，教研中心在南京工程学院组织召开了“21世纪中国高等学校应用型人才培养体系的创新与实践”课题立项研讨会。会议确定由教研中心组织国家级课题立项，为参加立项研究的高等院校搭建高起点的研究平台，整体设计立项研究计划，明确目标。课题立项采用整体规划、分步实施、滚动立项的方式，分期分批启动立项研究计划。为了确保课题立项目标的实现，组建了“21世纪中国高等学校应用型人才培养体系的创新与实践”课题领导小组（亦为高校应用型人才立体化教材建设领导小组）。会后，教研中心组织了首批课题立项申报，有63所高校申报了近450项课题。2003年1月，在黑龙江工程学院进行了项目评审，经过课题领导小组严格的把关，确定了首批9项子课题的牵头学校、主持学校和参加学校。2003年3月至4月，各子课题相继召开了工作会议，交流了各校教学改革的情况和面临的具体问题，确定了项目分工，并全面开始研究工作。计划先集中力量，用两年时间形成一批有关人才培养模式、培养目标、教学内容和课程体系等理论研究成果报告和在研究报告基础上同步组织建设的反映应用型人才培养特色的立体化系列教材。

与过去立项研究不同的是，“21世纪中国高等学校应用型人才培养体系的创新与实践”课题研究在审视、选择、消化与吸收多年来已有应用型人才培养探索与实践成果基础上，紧密结合经济全球化时代高校应用型人才培养工作的实

际需要,努力实践,大胆创新,采取边研究、边探索、边实践的方式,推进高校应用型人才培养工作,突出重点目标,并不断取得标志性的阶段成果。

教材建设作为保证和提高教学质量的重要支柱和基础,作为体现教学内容和教学方法的知识载体,在当前培养应用型人才中的作用是显而易见的。探索、建设适应新世纪我国高校应用型人才培养体系需要的教材体系已成为当前我国高校教学改革和教材建设工作面临的十分重要的任务。因此,在课题研究过程中,各课题组充分吸收已有的优秀教学改革成果,并和教学实际结合起来,认真讨论和研究教学内容和课程体系的改革,组织一批学术水平较高、教学经验较丰富、实践能力较强的教师,编写出一批以公共基础课和专业、技术基础课为主的有特色、适用性强的教材及相应的教学辅导书、电子教案,以满足高等学校应用型人才培养的需要。

我们相信,随着我国高等教育的发展和高校教学改革的不断深入,特别是随着教育部‘高等学校教学质量和教学改革工程”的启动和实施,具有示范性和适应应用型人才培养的精品课程教材必将进一步促进我国高校教学质量的提高。

全国高等学校教学研究中心

2003年 4月

前　言

由于计算机技术的飞速发展，微机原理与接口技术课程作为电类本科专业的一门重要的专业基础课，如何做到既有利于基本概念的掌握和基本能力的培养，又做到推陈出新、紧跟时代、学以致用，同时还要具有系统性和完整性，这是教学中一直在探索的问题。

虽然在 8086微处理器制成之后，微处理器技术已有了巨大的发展，但由于它的典型性和以后的微处理器对它的兼容性，使 8086对于初学者来说仍是必要的。本教材以 80X86为背景，讲述了微型计算机的基本工作原理，8086指令系统及其汇编语言程序设计，半导体存储器，输入输出接口，中断，串行通信，模数、数模转换，最后对高性能微处理器、总线标准和微型计算机作了介绍。

在编写过程中，注重深入浅出、循序渐进，让读者加快基本概念的建立。配以较多的程序设计实例和接口电路实例，使读者易于接受。本教材力求内容精练，取材新颖，还介绍了一些新的芯片及其接口技术，具有一定的参考价值和实用价值。

本课程具有实践性强的特点，程序设计能力只能在程序设计的过程中学会。学习者应多上机调试程序，水平才能得到提高。在每一章后面附以适当的习题、思考题，有利于读者尽快掌握程序设计方法和计算机接口技术。

本教材的参考教学课时为 80～96学时。教材中的部分内容可选讲（如第10章中，可根据实验条件选讲 8250和 8251A中的一种），部分内容可由学生自学。

本教材由徐晨编写第 6、10、11、13章并统稿，陈继红编写第 7、8、9章，王春明编写第 2、5、12章，徐慧编写第 1、3、4章。东南大学黄清教授担任本教材主审，提出了许多宝贵意见，在此表示衷心的感谢。

由于编者水平有限，书中难免存在错误和不当之处，敬请批评指正。

编者

2004年 2月

目　　录

第1章

基础知识

1.1 数制及其相互转换

1.1.1 常用计数制

1. 计数制

计数制是指用一组固定的数字符号和统一的规则表示数的方法。一般，r进制数可以用下式表示：

$$\sum_{i=-m}^{n} a_i r^i = a_{-m} r^{-m} + \cdots + a_{-2} r^{-2} + a_{-1} r^{-1} + a_0 r^0 + a_1 r^1 + \cdots + a_n r^n$$

其中，r称为数制的基数，r^i称为数制的权，i为整数。

基数的含义为：⑴ 基数为 r的数制称为 r进制数；⑵ 该数制数由 r个不同的数字符号表示；⑶ 确定了算术运算时的进位或借位规则，即加法时逢 r进一，减法时借一当 r。

权的含义为：⑴ 表示数字在不同的位置代表的数值是不一样的。每个数字所表示的数值等于它本身乘以该数位对应的权；⑵ 权是基数的幂。

以十进制为例，十进制数具有下列属性：

⑴ r=10，由 0～9共 10个不同的阿拉伯数字表示；

⑵ i位置上的权为 10^i；

⑶ 加法运算时逢十进一,减法运算时借一当十。

十进制数 $579.43=5\times10^{2}+7\times10^{1}+9\times10^{0}+4\times10^{-1}+3\times10^{-2}$

2. 计算机中常用计数制

计算机常用的计数制除上述十进制数外,还有二进制数、八进制数、十六进制数等。它们的部分属性见表 1.1。

表 1.1　计算机中常用计数制

数制	基数	数　　码	运算规则	举例
二进制	2	0,1	逢二进一,借一当二	1011.11
八进制	8	0,1,2,3,4,5,6,7	逢八进一,借一当八	745.64
十进制	10	0,1,2,3,4,5,6,7,8,9	逢十进一,借一当十	9999.99
十六进制	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F	逢十六进一,借一当十六	0A45.B

说明:为了便于计算机识别,当十六进制数的首字符为字母时,前面加数字 0。

⑴ 不同数制数的区别表示

为了区分不同的数制,书写上有两种方法:

方法一:用后缀区分。

二进制、十进制、八进制数、十六进制数的后缀分别为字母:B、D、Q、H。

【例】

1) 123D 表示十进制数 $123=1\times10^{2}+2\times10^{1}+3\times10^{0}$

2) 123Q 表示八进制数 $123=1\times8^{2}+2\times8^{1}+3\times8^{0}$

3) 123H 表示十六进制数 $123=1\times16^{2}+2\times16^{1}+3\times16^{0}$

方法二:用括号将数字括起,加以下标标注。

【例】

1) 十进制数 123表示为 $(123)_{10}$

2) 八进制数 123表示为 $(123)_{8}$

⑵ 二进制数

1) 二进制数的属性

二进制数基数为 2,由 0、1组成,它的各个位置上的权为 2^{i},小数点左边从右至左其各位的位权依次是:2^{0}、2^{1}、2^{2}、2^{3}、…,小数点右边从左至右其各位的位权依次是:2^{-1}、2^{-2}、2^{-3}、…

【例】　$1011.11B=1\times2^{3}+0\times2^{2}+1\times2^{1}+1\times2^{0}+1\times2^{-1}+1\times2^{-2}$

2) 二进制正整数的表示范围

二进制正整数的表示范围由其位数决定。n位二进制数可以表示 2^n 个正整数。例如 n=3,可以表示 8个不同的正整数;n=4,可以表示 16个不同的正整数。见表 1.2、表 1.3。

表 1.2 3位二进制数能表示的数

二进制数	000	001	010	011	100	101	110	111
对应的十进制数	0	1	2	3	4	5	6	7

表 1.3 4位二进制数能表示的数

二进制数	0000	0001	0010	0011	0100	0101	0110	0111
对应的十进制数	0	1	2	3	4	5	6	7
二进制数	1000	1001	1010	1011	1100	1101	1110	1111
对应的十进制数	8	9	10	11	12	13	14	15

3) 二进制数算术运算

加法规则:逢二进一。即:

0+0=0; 0+1=1; 1+0=1; 1+1=10

【例】 1101+1011=11000

减法规则:借一当二。即:

0−0=0; 1−0=1; 0−1=1; 1−1=0

【例】 1101−1011=0010

乘法规则:任何数乘以 0得 0,1乘以任何数得该数。即:

0×0=0; 0×1=0; 1×0=0; 1×1=1

【例】 1101×101=1000001

除法规则:0除以任何数得 0,任何数除以 1得该数,除数不得为 0。即:

0÷1=0; 1÷1=1

【例】 110÷10=11

4) 二进制数逻辑运算

与:0∧0=0; 0∧1=0; 1∧0=0; 1∧1=1;

或:0∨0=0; 0∨1=1; 1∨0=1; 1∨1=1;

非:$\bar{0}=1, \bar{1}=0$

异或:0 0=0;0 1=1;1 0=1;1 1=0;

这里分别用符号“∧”、“∨”、“-”、“ ”表示与、或、否、异或运算符。

由于二进制数书写长而且不易阅读,因此在计算机中经常使用与二进制之

间转换方便的八进制和十六进制数。

⑶ 八进制数

八进制数基数为 8,由 0～7共 8个数字组成,它的各个位置上的权为 8^k,小数点左边从右至左其各位的位权依次是:8^0、8^1、8^2、8^3、…,小数点右边从左至右其各位的位权依次是:8^{-1}、8^{-2}、8^{-3}、…

【例】 $753.45Q=7\times8^2+5\times8^1+3\times8^0+4\times8^{-1}+5\times8^{-2}$

【例】 $34Q+44Q=100Q$

⑷ 十六进制数

十六进制数基数为 16,由 0～9,A～F共 16个符号组成,它的各个位置上的权为 16^k,小数点左边从右至左其各位的位权依次是:16^0、16^1、16^2、16^3、…,小数点右边从左至右其各位的位权依次是:16^{-1}、16^{-2}、16^{-3}、…

【例】 $0FA3.3BH=15\times16^2+10\times16^1+3\times16^0+3\times16^{-1}+11\times168^{-2}$

【例】 $0FFH+1=100H$

1.1.2 不同数制之间的转换

1. 其他数制数转换为十进制数

其他数制数转换为十进制数的方法是“按权展开”。

【例】

1) $1011.11B=1\times2^3+0\times2^2+1\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2}=11.75D$

2) $753.4Q=7\times8^2+5\times8^1+3\times8^0+4\times8^{-1}=491.5D$

3) $0FA3.4H=15\times16^2+10\times16^1+3\times16^0+4\times16^{-1}=4003.25D$

2. 十进制数转换为其他数制数

把十进制数转换为其他数制数的方法很多,通常采用的方法有降幂法及乘除法。下面以十进制数转换为二进制数为例加以说明,十进制数转换为八进制数、十六进制数从此类推。

⑴ 降幂法

步骤:

1) 写出所有小于此数的各位二进制权;

2) 用要转换的十进制数减去与它值最近的二进制权值;

3) 如够减,相应位记为 1;如不够减,相应位记 0,并恢复该减法实施之前的数;

4) 重复 2)、3),直至该数为 0或达到所需精度。

【例】 把十进制数 117.75转换成二进制数

1) 小于 117.75D 的二进制权为：

2^6 (64)、2^5 (32)、2^4 (16)、2^3 (8)、2^2 (4)、2^1 (2)、2^0 (1)、2^{-1} (0.5)、2^{-2} (0.25) 、…

2)、3)、4) 重复过程如下：

整数部分：$117-2^6=53>0$ $a_6=1$

$53-2^5=21>0$ $a_5=1$

$21-2^4=5>0$ $a_4=1$

$5-2^3=-3<0$ $a_3=0$

$5-2^2=1>0$ $a_2=1$

$1-2^1=-1<0$ $a_1=0$

$1-2^0=0$ $a_0=1$

小数部分：$0.75-2^{-1}=0.25>0$ $a_{-1}=1$

$0.25-2^{-2}=0$ $a_{-2}=1$

转换结果为：$a_6a_5a_4a_3a_2a_1a_0.a_{-1}a_{-2}=1110101.11B$

(2) 乘除法

采用乘除法把十进制数转换为二进制具体为：整数部分除 2取余，直至商为0小数部分乘 2取整，直至积为整数或小数位数由精度定。

【例】 把十进制数 14.625转换成二进制数

整数部分：

商	余数	
14 ／2 =7……	0	$a_0=0$
7 ／2 =3……	1	$a_1=1$
3 ／2 =1……	1	$a_2=1$
1 ／2 =0……	1	$a_3=1$

小数部分：

积	整数	
0.625×2=1.25……	1	$a_{-1}=1$
0.25×2=0.5……	0	$a_{-2}=0$
0.5×2=1……	1	$a_{-3}=1$

转换结果为：$a_3a_2a_1a_0.a_{-1}a_{-2}a_{-3}=1110.101B$

3. 其他数制之间的转换

(1) 二进制与八进制数之间的转换

由于八进制数以 2^3 为基数，所以二进制数与八进制数之间的转换比较

简单。

1) 二进制数转换为八进制数:以小数点为界,整数部分向左,小数部分向右每 3位二进制数为一组,用 1位八进制数表示,不足三位的,整数部分高位补 0,小数部分低位补 0。

【例】 把 10110.11B转换为八进制

10110.11B=010 110 . 110B=26.6Q

2) 八进制数转换为二进制数与上述方法相反,把每位八进制数字用 3位二进制表示即可。

【例】 把 27.6Q转换为二进制数

27.6Q=010 111. 110B=10111.11B

(2) 二进制与十六进制数之间的转换

由于十六进制数以 2^4为基数,因而二进制数转换为十六进制数方法是:以小数点为界,整数部分向左,小数部分向右每 4位二进制数为一组,用 1位十六进制数表示,不足 4位的,整数部分高位补 0,小数部分低位补 0。反之,十六进制数转换为二进制数的方法是:把每位十六进制数用 4位二进制数表示即可。

【例】 把二进制数 10110.1转换为十六进制数

10110.1B=0001 0110.1000B=16.8H

【例】 把十六进制数 5A.7转换为二进制数

5A.7H=0101 1010.0111B=1011010.0111B

1.1.3 二进制编码的十进制数(BCD码)

尽管计算机采用的二进制数的表示法及运算规则简单,但书写冗长,不直观。有的场合需要计算机的输入输出仍采用人们习惯的十进制数。这样,在计算机输入、输出时要进行十进制数与二进制数的互相转换。为了使计算机能较方便地完成这个工作,需要将人们习惯的十进制数的每一位写成二进制的形式。凡是利用若干二进制数码来表示一位十进制数的方法,统称为“二进制编码的十进制数(BCD)”。

1. 8421码

因为十进制基数为 10,需要 10个不同的数码,所以为了能表示十进制数的某一位,选择的二进制数的位数至少需要 4位。在十进制数与若干二进制编码表示的数之间选择不同对应规律,可以得到不同形式的编码。最常用的一种编码的方法是 8421BCD码。这种编码的特点是 4位数码中的每一位对应一个固定的常数,且编码的权自左至右分别是 8,4,2,1。8421码的名称也就是由此而

来。将每位数码与对应的数相乘求和,就是它代表的十进制的数值。

【例】 9表示成:1001=1×8+1

这种编码的优点是:十进制数的每一位表示法完全和通常二进制一样,因此容易识别。这种编码的缺点是 10~15的 6个数没有意义,因此一旦在运算中出现,必须设法转为相应的数,才能得到正确的结果。

BCD编码的形式很多,本书所指 BCD码,均指 8421BCD码。

2. 8421BCD码的形式

BCD码有两种形式,即压缩 BCD码和非压缩 BCD码。表 1.4是两种 BCD的部分编码。压缩 BCD码用 4位二进制数表示一位十进制数,一个字节表示两位十进制数,如:96D表示成 1001 0110B=96H。

非压缩 BCD码用一个字节表示一位十进制数,一般只用低 4位的 0000~1001表示一位十进制数,如将 96D表示成 0000 1001 0000 0110B=0906H。

表 1.4 BCD码表

十进制编码	压缩 BCD (8421)码	非压缩 BCD (8421)码
0	0000 0000	0000 0000
1	0000 0001	0000 0001
2	0000 0010	0000 0010
3	0000 0011	0000 0011
4	0000 0100	0000 0100
5	0000 0101	0000 0101
6	0000 0110	0000 0110
7	0000 0111	0000 0111
8	0000 1000	0000 1000
9	0000 1001	0000 1001
10	0001 0000	0000 0001 0000 0000
11	0001 0001	0000 0001 0000 0001
…	…	…
99	1001 1001	0000 1001 0000 1001
100	0000 0001 0000 0000	0000 0001 0000 0000 0000 0000
…	…	…

1.2 符号数的表示及运算

数除了有上述无符号数外,还有符号数。数的符号在计算机中也用二进制数表示,通常用二进制数的最高位表示数的符号,0表示正数,1表示负数。把一个数及其符号在机器中数值化的表示称为机器数,而机器数所代表的数本身称为数的真值。机器数可以用不同方法表示,常用的有原码、反码和补码表示法。

1.2.1 符号数的表示

1. 原码

数 x的原码记作 $[x]_{原}$,如机器字长为 n,则原码的定义如下:

$$[x]_{原}=\begin{cases}x & 0\le x\le 2^{n-1}-1\\ 2^{n-1}+|x| & -(2^{n-1}-1)\le x\le 0\end{cases}$$

当机器字长 n=8时,

$[+0]_{原}=0000\ 0000$, $[-0]_{原}=1000\ 0000$

$[+1]_{原}=0000\ 0001$, $[-1]_{原}=1000\ 0001$

$[+127]_{原}=0111\ 1111$, $[-127]_{原}=1111\ 1111$

当机器字长 n=16时,

$[+0]_{原}=0000\ 0000\ 0000\ 0000$, $[-0]_{原}=1000\ 0000\ 0000\ 0000$

$[+1]_{原}=0000\ 0000\ 0000\ 0001$, $[-1]_{原}=1000\ 0000\ 0000\ 0001$

$[+32767]_{原}=0111\ 1111\ 1111\ 1111$, $[-32767]_{原}=1111\ 1111\ 1111\ 1111$

由上可知,在原码表示中,最高位为符号位,正数为 0,负数为 1,其余 n-1位表示数的绝对值。原码表示的整数范围是 $-(2^{n-1}-1)\sim+(2^{n-1}-1)$。

【例】 8位二进制原码表示的整数范围是 -127~+127,16位二进制原码表示的整数范围是 -32767~+32767。原码表示法简单直观,但由于符号位不能参与运算,所以不便于进行加减运算。

2. 反码

数 x的反码记作 $[x]_{反}$,如机器字长为 n,反码定义如下:

$$[x]_{反}=\begin{cases}x & 0\le x\le 2^{n-1}-1\\ (2^{n}-1)-|x| & -(2^{n-1}-1)\le x\le 0\end{cases}$$

当机器字长 n=8时,

$[+0]_{反}=0000\ 0000$, $[-0]_{反}=1111\ 1111$

$[+1]_{反}=0000\ 0001$，　$[-1]_{反}=1111\ 1110$

$[+127]_{反}=0111\ 1111$，　$[-127]_{反}=1000\ 0000$

当机器字长 n=16时，

$[+0]_{反}=0000\ 0000\ 0000\ 0000$，　$[-0]_{反}=1111\ 1111\ 1111\ 1111$

$[+1]_{反}=0000\ 0000\ 0000\ 0001$，　$[-1]_{反}=1111\ 1111\ 1111\ 1110$

$[+32767]_{反}=0111\ 1111\ 1111\ 1111$，　$[-32767]_{反}=1000\ 0000\ 0000\ 0000$

反码表示法中，最高位仍为符号位，正数为 0，负数为 1。正数的反码与原码相同，负数的反码，是原码的符号位不变，其他各位求反。n位反码表示整数的范围是 $-(2^{n-1}-1)\sim+(2^{n-1}-1)$。

【例】 8位二进制反码表示整数的范围是 −127～+127，16位二进制反码表示的整数范围是 −32767～+32767，与原码相同。

3. 补码

数 x的补码记作 $[x]_{补}$，如机器字长为 n，补码定义如下：

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

从定义式可见，正数的补码与其原码相同，只有负数才有求补的问题。所以，严格地说，“补码表示法”应称为负数的补码表示法。一个二进制数，以 2^n 为模，它的补码叫做 2的补码。所以，补码的定义可以修改为：

$$[x]_{补}=\begin{cases}x & 0\leq x<2^{n-1}-1\\ 2^n-|x| & -2^{n-1}\leq x<0\end{cases}$$

当机器字长 n=8时，

$[+0]_{补}=0000\ 0000$，　$[-0]_{补}=2^8-|-0|=1111\ 1111$

$[+1]_{补}=0000\ 0001$，　$[-1]_{补}=2^8-|-1|=1111\ 1110$

$[+127]_{补}=0111\ 1111$，　$[-127]_{补}=2^8-|-127|=1000\ 0001$

补码表示法中，最高位仍为符号位，正数为 0，负数为 1。补码表示的整数范围是 $-2^{n-1}\sim+(2^{n-1}-1)$。例如 8位二进制补码表示的整数范围是 −128～+127，16位二进制补码表示的整数范围是 −32768～+32767。部分 8位二进制数的数原码、反码、补码列表如表 1.5所示。

表 1.5 原码、反码、补码表

二进制数	无符号数	带符号数		
		原码	补码	反码
0000 0000	0	+0	0	+0

续表

二进制数	无符号数	带符号数		
		原码	补码	反码
0000 0001	1	+1	+1	+1
0000 0010	2	+2	+2	+2
…	…	…	…	…
0111 1110	126	+126	+126	+126
0111 1111	127	+127	+127	+127
1000 0000	128	−0	−128	−127
1000 0001	129	−1	−127	−126
…	…	…	…	…
1111 1101	253	−125	−3	−2
1111 1110	254	−126	−2	−1
1111 1111	255	−127	−1	−0

1.2.2　码制转换

反码通常是作为求补过程的中间形式,所以重点介绍原码和补码之间的转换。因正数的原码、补码和反码的表示方法相同,不存在转换问题,故只讨论负数的情况。

1. 已知 $[X]_{原}$,求 $[X]_{补}$

方法是符号位不变,数值部分逐位取反后末位加 1。

【例】　已知 $[X]_{原}=10011010$,求 $[X]_{补}$

$[X]_{原}$ = 1 0 0 1 1 0 1 0

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

1 1 1 0 0 1 0 1

+)　　　　1

$[X]_{补}$ = 1 1 1 0 0 1 1 0

还可以总结出一个更简单的规律:符号位不变,数值部分从低位开始向高位逐位行进,在遇到第一个 1以前,包括第一个 1按原码照写;第一个 1以后,逐位

取反。

【例】 $[x]_{原}$ =1 0 0 1 1 0 1 0

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

$[x]_{补}$ =1 1 1 0 0 1 1 0

↑ ↑ ↑

不变 求反 不变

可见,两种方法所得结果是一样的,读者可用定义对结论进行验证。

2. 已知 $[x]_{补}$,求 $[x]_{原}$

由补码的定义,不难得出:$[[x]_{补}]_{补} = [x]_{原}$,所以由 $[x]_{补}$ 求 $[x]_{原}$,只要求 $[[x]_{补}]_{补}$ 即可。

【例】 已知 $[x]_{补}$ =1110 0110,求 $[x]_{原}$

$[x]_{补}$ =1 1 1 0 0 1 1 0

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

$[x]_{原}$ =1 0 0 1 1 0 1 0

3. 已知 $[x]_{补}$,求 $[-x]_{补}$,即 求补

求补的方法是将 $[x]_{补}$ 连同符号位一起逐位变反,然后在末位加 1,便得到 $[-x]_{补}$。这时要注意的是,不管 $[x]_{补}$ 是正数还是负数,都应按上述方法进行。

【例】 已知 $[x]_{补}$ =0101 0110,求 $[-x]_{补}$

$[x]_{补}$ = 0 1 0 1 0 1 1 0 $(+86)_{补}$

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

$[-x]_{补}$ = 1 0 1 0 1 0 1 0 $(-86)_{补}$

已知 $[x]_{补}$,求 $[-x]_{补}$,在进行补码减法运算时,特别有用。

1.2.3 补码的运算

1. 补码加法

补码加法规则是:$[x+y]_{补} = [x]_{补} + [y]_{补}$,其中,x和 y为正数、负数皆可。

【例】 −9 +2 = −7

$[x]_{补} = [-9]_{补}$ =11110111

+) $[y]_{补} = [+2]_{补}$ =00000010

11111001 $[-7]_{补}$

2. 补码减法

补码减法规则是：$[x-y]_{补} = [x]_{补} + [-y]_{补}$，其中，x和 y为正数、负数皆可。

【例】　5－3＝2

$[x]_{补} = [5]_{补}$　＝　0 0 0 0 0 1 0 1

+)　$[y]_{补} = [-3]_{补}$ ＝　1 1 1 1 1 1 0 1

1 0 0 0 0 0 0 1 0　　$[+2]_{补}$

↑

丢掉

进行补码加、减运算时，如果最高位有进位或借位，则自动丢掉。至于这种进位或借位丢失是否会影响结果的正确性，将在溢出判断中讨论。

3. 补码运算的溢出判别

如果运算结果，超出了计算机能表示的数的范围，会得出错误的结果，这种情况称为溢出。产生错误结果的原因是溢出时数值的有效位占据了符号位。

【例】　73＋72＝145 ＞127

$[x]_{补}$ ＝0 1 0 0 1 0 0 1　(＋73)

+)　$[y]_{补}$ ＝0 1 0 0 1 0 0 0　(＋72)

1 0 0 1 0 0 0 1　$[-111]_{补}$　　结果错

上例中，参加运算的两个数为正数，结果应为正数。但由于运算结果 (145) 大于计算机能表示的数的范围 (127)，使得和的数值部分占据了符号位，计算机把结果变为负数，产生了一个错误结果。

对于字长为 n的计算机，它能表示的定点补码范围为 $-2^{n-1} \leq x \leq 2^{n-1}-1$，如果运算结果小于 -2^{n-1}或大于 $2^{n-1}-1$，则发生溢出。判定方法如下：

(1) 加法：令 A＋B＝C，A、B的符号位分别为 a_{n-1}、b_{n-1}；C的符号位为 c_{n-1}，则：

1) A＞0，B＞0：此时，$a_{n-1}=0$，$b_{n-1}=0$，c_{n-1}也应为 0。若发生溢出，数值的最高位占据了符号位，使 $c_{n-1}=1$；

2) A＜0，B＜0：此时，$a_{n-1}=1$，$b_{n-1}=1$，c_{n-1}也应为 1。若发生溢出，数值的最高位占据了符号位，使 $c_{n-1}=1$；

3) A、B异号，加法时不会产生溢出。

综上分析，补码加法运算的溢出条件为：

$$V=a_{n-1}b_{n-1}c_{n-1}+a_{n-1}b_{n-1}c_{n-1}$$

(2) 减法:对于补码减法,有:$[A]_{补} - [B]_{补} = [A]_{补} + [-B]_{补}$,可将减法运算变为加法运算,按补码加法溢出判断方法来进行。补码减法运算的溢出条件为:

$$V = a_{n-1} b_{n-1} c_{n-1} + a_{n-1} b_{n-1} c_{n-1}$$

这种判断方法不容易由硬件来实现。一般计算机定点补码加减法溢出判断,是查看有没有向符号位 C_{y-1}进位,符号位的计算结果有没有向进位标志位 C_y 进位。溢出的判断条件为:

$$V = C_y C_{y-1} + C_y C_{y-1}$$

即当次高位 C_{y-1}与最高位 C_y 同时产生进位或借位时,条件不成立,不产生溢出;当次高位 C_{y-1}与最高位 C_y不同时产生进位或借位时,条件成立,产生溢出。

1.3 定点数和浮点数

对于任意一个二进制数,都可以表示为:

$$N = 2^J \times S$$

其中 J称为数 N的阶,S称为数 N的尾数。尾数 S表示数 N的全部有效数字,阶 J表示出小数点的位置。当 J为固定值时,称数的这种表示方法为定点表示,这样的数称为定点数;当 J可变时,称数的这种表示方法为浮点表示,这样的数称为浮点数。

1.3.1 数的定点表示法

常用定点数有两种:纯整数和纯小数。

1. 纯整数

J=0,且尾数 S为纯整数,即小数点固定在尾数之后,这时定点数只能表示整数。

符号位	尾数 .

2. 纯小数

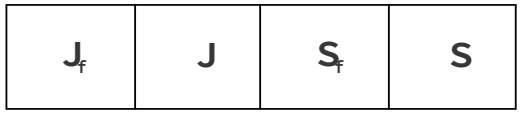
J=0,且尾数 S为纯小数,即小数点固定在尾数之前,这时定点数只能表示小数。

符号位.	尾数

在计算机中,定点表示法可以表示整数,也可以表示纯小数,一般用来表示整数,而实数则用浮点表示法。符号的表示同上,以数字 0或 1来表示,"0"代表"+","1"代表"-"。上节中,讨论整数的原码、补码与反码表示法,它们同样适用于定点小数,也适用于即将讨论的浮点数。

1.3.2　数的浮点表示

浮点数在计算机中表示为:



↑ 阶符　↑ 阶码　↑ 数符　↑ 尾数

J_f,阶符,表示阶的符号。J_f=0,阶码为正;J_f=1,阶码为负。

J,阶码,表示阶的大小。J为整数。

S_f,数符,表示数的符号。S_f=0,正数;S_f=1,负数。

S,尾数,是数字的有效数字。S为纯小数。

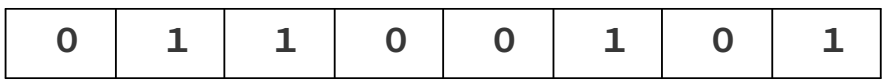
【例】　将 N=2.5表示成字长为 8的浮点数,要求阶码 2位,尾数 4位,阶符及数符各一位,阶码与尾数均用原码表示。

二进制浮点表示形式为:$N=2^{010}\times 0.101$

它在计算机中表示形式:

↑ 阶符　↑ 阶码(2位)　↑ 数符　↑ 尾数(4位)

阶码的底,在机器数表示法中不出现。阶码小数点在阶码的右端(即:阶码为整数);尾数的小数点在尾数的左端(即:尾数为纯小数)。在浮点表示法中,可以做到移动小数点而保持数的值不变,尾数左移一位或小数点右移一位,阶码减 1;尾数右移一位或小数点左移一位,阶码加 1。若将上例中的机器数尾数右移 1,同时阶码加 1,则变为:



↑ 阶符　↑ 阶码(2位)　↑ 数符　↑ 尾数(4位)

它对应的二进制数是:$N=2^{011}\times 0.0101$,显然,移位前后的数值没有变。

为了使计算机在运算过程中,不丢失有效数字,提高运算的精度,在计算机中,浮点数通常都采用规格化数的表示方法。对二进制浮点数 $2^j \times S$,若尾数 S 满足 $1/2 \le |S| < 1$ 时,则为“规格化的数”,否则就是非规格化的数。要使浮点规格化只要移动尾数并改变阶的值就可以实现。对补码来讲,规格化的数意味着什么呢?如果是正数,尾数第一位数字为 "1" 如果是负数,尾数第一个数字应为 "0" 即 1.0×…×) 或为 "1",而以后各位全为 "0" 即 1.10…0),前者小于 $-1/2$,后者正好等于 $-1/2$。为了机器判断方便,在补码表示中,往往不把 $-1/2$ 列入规格化的数。这样,补码规格化数规定如下:

对正数 $x \ge 0$,如果 $1 > x \ge 1/2$,称为规格化数,其补码表示形式为:$0.1x_2 \cdots x_n$

对负数 $x < 0$,如果 $-1/2 > x \ge 1$,称为规格化数,其补码表示形式为:$1.0x_2 \cdots x_n$

其中 $x_2 \cdots x_n$ 表示可任取 "0" 或 "1"。因此,机器只要判断运算结果的符号位与第一位数字是否相同,便可知道是否是规格化的数。

【例】 将 $(-18.75)_{10}$ 转换为二进制浮点规格化数 (用补码表示),基数为 2,其中阶符、阶码共 4位,数符 (尾符)、尾数共 8位。

解:$-18.75D = -10010.11B = -0.1001011 \times 2^5$

阶码:$+5 = 0101B$,补码 :0101

尾数:$[-0.1001011]_{补} = 1.0110101$

结果为 :0 101 1 0110101

对定点数,字长位数确定以后,表示数的范围是一定的。对于浮点数,当字长一定时,分给阶码的位数越多,则表示数的范围越大。但是由于尾数的位数越少,必定会使有效数字位数减少,从而影响数字的精度。因此,阶码与尾数分别用多少位,要视具体情况适当分配。

【例】 设某计算机用双字表示一浮点数,其中阶符、阶码 8位,用原码表示,数符 (尾符)、尾数共 8位,用补码表示,求能表示的最大正数为多少?

解 :最大正数的浮点表示为 :0 1111111 0 1111111

阶码:$+1111111 = 2^{127}$

尾数:$+0.1111111 = 1 - 2^{-7}$

最大正数十进制表示为:$(1-2^{-7}) \times 2^{127}$

浮点加减法运算要经过对阶、尾数运算、舍入和规格化三步操作。其中,对阶是将两个加数的小数点对齐,通常采用的方法是小阶向大阶看齐,阶码较小的数,其尾数向右移,每右移一位,阶码加 "1",直到两数阶码相同为止。

1.4 字 符 编 码

计算机不仅能处理数字信息,而且可以处理非数字信息,而这些非数字信息在计算机中也以代码的形式存在。一般情况下,计算机依靠输入设备把要输入的字符转换成为一定格式的代码,然后才能接收进来。输出则是相反过程,为了在输出设备上输出字符,计算机要把相应字符的编码送到外部输出设备。本节讨论字符编码。在微型计算机中最常用的是“美国标准信息交换代码”(ASCII码)和“信息交换用汉字编码”(汉字国际码)。

1.4.1 ASCII码

ASCII码,全称为“美国标准信息交换代码(American Standard Code for Information Interchange)”。基本 ASCII代码共 128个,其中控制符 32个,数字 10个,大写英文字母 26个,小写英文字母 26个以及专用符号 34个。见附录 1。

每一个 ASCII码存放在一个字节中,低 7位为有效编码位,最高位可用于校验位或用于 ASCII码的扩充。扩充后的 ASCII有 256个,除基本的 ASCII码外,还扩充了 128个字符和图形符号。

字符的 ASCII码可以看作字符的码值,如字符“A”的 ASCII代码值为 41H,“Z”的 ASCII代码值为 5AH,利用这个值的大小可以将字符排序,以后会遇到字符串大小比较,实际上是比较 ASCII码代码值的大小。

1.4.2 汉字编码

ASCII码只有 256个符号,所以一个字节完全可以表示一个 ASCII符号。汉字的数量很大,常用汉字有 6 000多,总共 60 000多,要为每个汉字给出一个唯一的编码,则至少需要 16位(2^{16}=65 536)二进制位。因此,在计算机中,用两个字节对汉字进行编码。1981年我国制定了“信息交换用汉字编码字符基本集(GB 2312—80)”。这个标准中除汉字外还收录一般符号、序号、数字、拉丁字母、日文假名、希腊字母、俄文字母、汉语拼音符号、汉语注音字母符号等,共 7 445个图形字符。其中汉字 6 763个,分两级,第一级为常用字 3 755个,第二级为次常用字 3 008个;图形符号为 628个。每个字符都采取两个字节表示,每个字节低 7位为字符编码,最高位用于校验或汉字标志。整个代码表分成 94区,每个区有 94位。区的编码从 1～94,由第一字节标志,位的编号也从 1～94,

由第二个字节标志。代码中的任何一个图形字符位置都用它所在的区号与位号标志。用区号和位号对图形字形的标识,称为“区位码”。例如,汉字“啊”用 16－01表示,也可将连字符取消,表示为 1601。

1. 汉字外部码

汉字外部码即汉字输入码,是汉字输入计算机时使用的编码。由于汉字数量大,不可能用含有那么多个键的键盘来输入,于是采用汉字编码输入,如区位码、拼音码、五笔码等。

2. 汉字内部码

汉字内部码即机内码,是计算机处理汉字时所采用的代码形式。现在流行的多种汉字输入法,一般都不单独建立自身的汉字库和构造机内码,大多采用国家标准 GB 2312—80汉字交换码的序号作为机内码。

3. 汉字交换码

汉字交换码亦称国标码,是 GB 2312—80等国家标准中采用的用于汉字信息处理的交换码。国标码与区位码有简单的换算关系,将区号位号分别加上 32,就可以得到汉字的国标代码,对于汉字“啊”(1601) + (3232) → 4833D = 3021H。称 1601D为汉字“啊”的区位码,3201H为它的国标码。

由于汉字的总数为 60 000多字,这是 GB 2312—80标准收入的总数的 10倍,为此国家标准局又颁布了 GB 7589和 GB 7590汉字标准。此外,还颁布了汉字的第一辅助集——第五辅助集。国际标准化组织 ISO／IEC10646提出用 4个字节对全世界的文字信息编码,四个字节有 2^{32}种组合,可达 20亿。中国内地及中国港台地区与日本、朝鲜联合制定了用两个字节编码的 CJK编码,收入了 20 000多汉字及符号,现已批准成为 GB 13000。随着 Windows98、Windows NT等操作系统的使用,其中的中文版也采了中西文统一编码,称之为“Unicode”编码,收入 27 000个汉字。为了做到与 GB 2312—80兼容,又能支持 Unicode编码,我国又颁布了《国标汉字扩充码 (GKB)》,现已在 Windows98、Windows NT等操作系统中广泛使用。

思考题与习题

1.1 总结计算机中十进制、二进制、八进制及十六进制数的书写形式。123D、0AFH、77Q、1001110B分别表示什么计数制的数?

1.2 字长为 8位、16位二进制数的原码、补码表示的最大数和最小数分别是什么?

1.3 把下列十进制数分别转换为二进制数和十六进制数。

(1) 125 (2) 255 (3) 72 (4) 5090

1.4 把下列二进制数分别转换为十进制数和十六进制数。

⑴ 1111 0000 ⑵ 1000 0000 ⑶ 1111 1111 ⑷ 0101 0101

1.5 把下列十六进制数分别转换为十进制数和二进制数。

⑴ FF ⑵ ABCD ⑶ 123 ⑷ FFFF

1.6 分别用 8位二进制数和 16位二进制数写出下列十进制数的原码和补码。

⑴ 16 ⑵ −16 ⑶ +0 ⑷ −0 ⑸ 127 ⑹ −128 ⑺ 121 ⑻ −9

1.7 试实现下列转换。

⑴ $[X]_{原}=10111110$,求 $[X]_{补}$ ⑵ $[X]_{补}=11110011$,求 $[-X]_{补}$

⑶ $[X]_{补}=10111110$,求 $[X]_{原}$ ⑷ $[X]_{补}=10111110$,求 $[X]_{反}$

1.8 假设两个二进制数 A=01101010,B=10001100,试比较它们的大小。

⑴ A B两数均为带符号的补码数 ⑵ A B两数均为无符号数

1.9 下列各数均为十进制数,请用 8位二进制数补码计算下列各题,用十六进制数表示其运算结果,并判断是否溢出,验证教材中所给判断依据。

⑴ 90+71 ⑵ 90−71 ⑶ −90−71 ⑷ −90+71 ⑸ −90−(−71)

1.10 完成下列 8位二进制数的逻辑运算:

⑴ 11001100∧10101010 ⑵ 11001100∨10101010 ⑶ 11001100⊕10101010

⑷ 10101100∧10101100 ⑸ 10101100⊕10101100 ⑹ 10101100∨10101100

⑺ $\overline{10101100}$

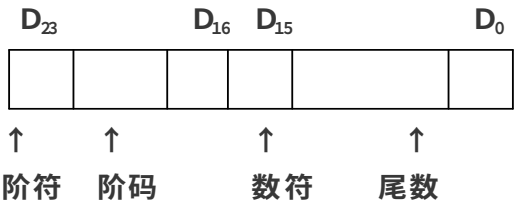
1.11 以下均为 2位十六进制数,试说明当把它们分别看作无符号数或字符的 ASCII码值,它们所表示的十进制数和字符是什么?

⑴ 30 ⑵ 39 ⑶ 42 ⑷ 62 ⑸ 20 ⑹ 7

1.12 把以下十进制数分别以压缩 BCD码、非压缩 BCD码、ASCII码串表示。

⑴ 2 ⑵ 78

1.13 设浮点数格式如下图所示:



阶码、尾数均以补码表示,基数为 2,求:+25.6和 −361.25的规格化浮点数。

1.14 设某计算机用 12位表示一个符点数,该符点数从高位到低位依次为:阶符 1位、阶码 3位 (原码表示)、数符 1位、尾数 7位 (补码表示),则 0 100 1 0110011的真值是多少?

第2章

微型计算机概论

2.1 计算机概论

1946年第一台通用电子数字计算机 ENIAC（Electronic Numerical Integrator and Computer）在美国宾夕法尼亚大学诞生。但 ENIAC的结构与后来的计算机有所不同，并且采用十进制，输入和更换程序十分不便。第一台真正意义上的存储式程序（stored program）计算机是 1949年在英国剑桥大学建成的 EDSAC计算机，它使用了 3 000只电子管，每秒钟能完成 700次加法运算。第一台商用计算机是在 1953年由 IBM公司制造的。

依据半导体技术水平和硬件、软件的技术特征，电子计算机可分为五代（现在正处于第五代）：

第一代：1945～1954年，电子管和继电器；机器语言，符号语言。

第二代：1955～1964年，晶体管和磁芯存储器；汇编语言，高级语言，监控程序。

第三代：1965～1974年，中小规模集成电路；汇编语言，高级语言，操作系统。

第四代：1975～1990年，LSI/VLSI和半导体存储器；汇编语言，高级语言，操作系统。

第五代：1990年至今 ULSI/GSI（Giga Scale Integration）巨大规模集成电路；汇编语言，高级语言，操作系统。

2.1.1 计算机硬件基本结构

目前,计算机硬件体系结构基本上还是经典的冯·诺依曼结构,由运算器、控制器、存储器、输入设备和输出设备等 5个基本部分组成,如图 2.1所示。

⑴ 运算器 (ALU, Aritmetic Logical Unit) ——能完成各种算术和逻辑运算的部件。

⑵ 控制器 (CU, Control Unit) ——能发出各种控制信息,使计算机各部件协调工作的部件。

⑶ 存储器 (M, Memory) ——能记忆程序和数据的部件。

⑷ 输入设备 (IN, Input device) ——能将程序和数据输入的部件。

⑸ 输出设备 (OUT, Output device) ——能将结果数据和其他信息输出的部件。

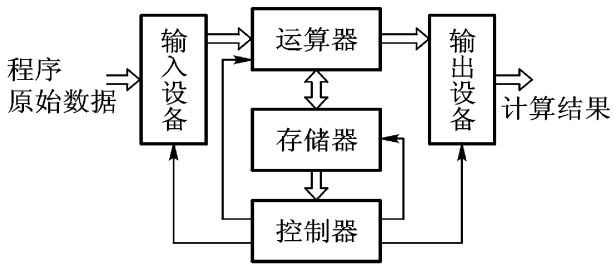


图 2.1 计算机基本结构

从图 2.1可见,计算机 5大部件之间有两类信息在流动:一类是数据信息,用双线表示,包括原始数据、中间结果、计算机结果和程序指令;另一类是控制信息,用单线表示,它是由控制器发出,指挥和协调其他各部件动作的信号。不论是数据还是控制命令,计算机中都是用 “0”和 “1”表示二进制信息。数据输入到存储器时,需经过运算器;数据输出时也要由运算器送到输出设备。计算机工作时将存放在存储器中的程序逐条取出到控制器,控制器执行指令时发出控制信号到运算器、存储器、输入设备、输出设备。

2.1.2 计算机工作原理

1945年冯·诺依曼在《关于计算仪器逻辑设计的初步探讨》一文中,第一次提出了计算机组成和工作方式的基本思想:

⑴ 计算机由运算器、控制器、存储器、输入和输出设备 5部分组成。

(2) 数据和指令以二进制代码形式不加区别地存放在存储器中,地址码也为二进制形式,计算机能自动区分指令和数据。

(3) 编写好的程序事先存入存储器。控制器根据存放在存储器中的指令序列即程序来工作,由程序计数器(PC, Program Counter)控制指令的执行顺序。控制器具有判断能力,能根据计算结果选择不同的动作流程。

其中,指令是命令计算机完成某种基本操作的代码。将各种算术运算、逻辑运算及存储器的读、写等作为基本操作,为每一个基本操作规定一个代码,这个代码被称为指令。

当需要计算机完成某项任务时,就将其分解成一系列的基本操作并用指令来表示,预先存放到存储器中。计算机工作时就逐条执行指令,完成一系列的基本操作,从而完成整个任务。我们把能完成某项任务的指令序列称为程序。

使用计算机时,首先要将程序存储,即将指令序列存放到存储器。然后在计算机工作时,控制器从存储器逐条取出指令、分析指令并执行指令。执行指令时,控制器依次发出各种控制命令信号给其他部件,使运算器完成某种算术、逻辑运算或作寄存器与存储器之间的数据传送,输入和输出等。计算机的工作过程就是执行指令的过程。

程序中的指令一般按序存放在存储器的连续区域中。计算机开始执行程序时,PC中存放着第一条指令所在存储单元的地址,然后每取出一条指令(确切地说是每取出一个指令字节),PC中的内容自动加1,指向下一条指令地址,从而保证了自动地按顺序取指令和执行指令。

2.1.3 计算机的性能指标

评价一台计算机,涉及许多因素,对于计算机的使用者来说,至少要了解以下评估计算机性能的主要指标。

1. 字长

在计算机中,所有信息都是用二进制数码(0,1)表示的。其最小单位是位(bit),即一个二进制数位。CPU在处理和传送信息时,往往是把一组二进制数码看作是一个整体来并行操作,并行处理的一组二进制数称为一个字(Word),字所含有的二进制位数称为字长。字长是CPU交换、加工和存放信息时其信息位的最基本长度,它通常与寄存器、运算器、传输线的宽度相一致。因此,字长实际上表示的是CPU并行处理的最大位数。字长是计算机的重要性能指标,也是计算机分类的依据之一。

计算机中普遍使用字节(Byte)作为单位,一个字节由8位二进制数组成,通

常用 D_7, D_6, …, D_0 来表示从最高位 (MSB) 到最低位 (LSB) 的各数位。因此,计算机字长的位数,也可以用“字节”单位取代。比如字长 8位,可说成一个字节字长或字长 1字节;字长 16位,可说成字长 2字节,用 D_{15}, D_{14}, …, D_0 表示其各数位。

2. 存储容量

存储器 (通常指内存储器) 是计算机存放二进制信息的“仓库”,由若干存储单元组成。存储单元的编号称做存储单元地址 (是二进制的数字码)。存储容量与 CPU构成的系统能够访问的存储单元数有关。存储单元的数目是由传送地址信息的传输线的条数决定的。若有 16条地址线,所能编出的地址码有 2^{16} =65 536种,由此可区分 65 536个单元。计算机中把 2^{10} =1 024规定为 1 K,因此 65 536个单元可以称为 64 K个单元。若有 20条地址线,2^{20} =1 048 576=1 024 K,1 024 K规定为 1M (1兆),即 20条地址线有 2^{20} =1 M个单元地址码。若有 30条地址线,则有 2^{30} =1 KM =1 G个单元地址码。

一般存储单元是以字节 (8 bit) 为单位的,即一个存储单元中存放一个字节信息,信息的读出和写入以字节为单位。所以存储容量可以看作是存储器能够存放信息的最大字节数。通常说存储容量为 64K、1M或 1G,分别是指 64 K字节 (64 KB)、1M字节 (1 MB) 和 1G字节 (1 GB)。

3. 指令系统

计算机在设计时,就确定了它能完成的各种基本操作。一台计算机所固有的基本操作指令的集合,称为该计算机的指令系统。计算机的指令系统一般含有几十到几百条指令。

计算机能完成的基本操作种类越多,即指令系统的指令数越多,说明其功能越强。此外,一条指令本身功能的强弱,也能说明该问题。

4. 运算速度

计算机完成一个具体任务所花费的时间就是完成该任务的时间指标,时间越短,表明计算机的速度越高。但是计算机各种指令执行时间是不一样的。以每秒执行基本指令的条数来大致地反映计算机的运算速度。单位为百万条指令/秒 (MIPS)。

现在,人们用计算机的主频——时钟频率来表示运算速度,以兆赫 (MHz) 或吉赫 (GHz) 为单位。主频越高,表明运算速度越快。

5. 系统配置

一台计算机要能正常工作,必须提供必要的人机联系手段,这包括配置相应数量的外部设备 (如键盘、显示器、磁盘驱动器、打印机、绘图仪等) 和配置实现计算机操作的软件。当然,外设配置越高档,软件配置越丰富,计算机的使用就

越便利,工作效率也就越高。特别是软件配置,在很大程度上决定了计算机能力的发挥。

2.1.4 CISC和 RISC

为了提高计算机性能,使 CPU有更大的指令系统、更多的专用寄存器、更多的寻址方式和更强的指令计算功能,CPU的结构正在沿着不断复杂化的方向发展,将它们称为复杂指令集计算机 (CISC, Complex Instruction Set Computer)。CISC技术通过增强指令功能提高计算机的性能,指令码不等长,指令数量多。CISC技术的复杂性在于硬件,在于 CPU芯片中控制器部分的设计与实现。自 PC诞生以来,主流产品一直使用 Intel公司的 CPU或其他公司生产兼容产品,而 Intel公司的 CPU沿用了 CISC技术。

当 CISC发展到一定程度后,人们发现一些复杂指令很少使用,但为了把它们加入到指令集中却使控制器的设计变得十分复杂,并占用了相当大的 CPU芯片面积。指令的执行周期也较长。因此,从处理器的执行效率和开发成本两个方面考虑,有必要对复杂指令集结构的处理器进行反思。

1980年 Patterson和 Ditzel首先提出了精简指令集计算机 (RISC, Reduced Instruction Set Computer)的概念,另觅提高计算机性能的途径。RISC具有简单的指令集,指令少、指令码等长,寻址方式少、指令功能简单;强调寄存器的使用,CPU配备大量的通用寄存器 (常称为寄存器文件 register file),以编译技术优化寄存器的使用;强调对指令流水线的优化,采用超标量和超级流水线。通过简化指令系统使控制器结构简化,进而提高指令执行速度。RISC技术的复杂性在于软件,在于编译程序的编写与优化。目前,RISC处理器产品主要用在工程工作站、嵌入式控制器和超级小型计算机上。

今后 RISC技术和 CISC技术都会继续发展,同时,RISC技术与 CISC技术的竞争使它们互相渗透,如 Power PC处理器,已不再是“纯”RISC结构;而最新的 CISC设计也融进不少 RISC特征,如 Intel Pentium系列、AMD K6系列微处理器。

2.2 微型计算机

随着 VLSI技术的发展,20世纪 70年代以后微型计算机迅速发展,广泛应用于社会生活的各个领域。它们价格不高,但系统结构中采用或借鉴了过去为开发大中型机而使用过的技术,有的已能和过去的小型机、甚至大中型计算机的性能相媲美。

2.2.1 微处理器、微型计算机和微型计算机系统

微处理器 (Microprocessor) 也被称为微处理机,它是微型计算机的核心部件,但并不是微型计算机。微处理器包括算术逻辑部件 ALU、控制部件 CU和寄存器组 R三个基本部分和内部总线。相当于图 2.1中一般计算机系统结构中的运算器和控制器的组合,一般又称为中央处理器 (CPU)。

微型计算机 (Micro Computer) 简称为微机,它是以微处理器为核心,加上由大规模集成电路制作的存储器 M (ROM和 RAM)、I/O (输入/输出) 接口和系统总线组成的。由于微处理器在微型计算机中的重要性,其性能很大程度上决定了微型计算机的性能。

微型计算机系统 (Micro Computer System) 是以微型计算机为核心,再配以相应的外围设备,电源、辅助电路和控制微型计算机工作的软件而构成的完整的计算系统。软件分为系统软件和应用软件两大类。系统软件是用来支持应用软件的开发与运行的,它包括操作系统、标准实用程序和各种语言处理程序等。应用软件是解决用户具体应用问题的程序及有关的文档和资料。

2.2.2 微处理器的发展

1971年,美国旧金山南部森特克拉郡 (硅谷) 的 Integrated Electron公司 (Intel公司) 首先制成 4004微处理器,进而研制出由它组成的第一台微型机。30多年来,微处理器的发展日新月异,下面做一个简要的回顾。

⑴ 第一代微处理器

1971年美国 Intel公司采用 MOS大规模集成电路技术生产出 4004微处理器,它本来是为高级袖珍计算器而设计的,但生产出来后,取得了意外的成功。于是 Intel公司对它作了改进,正式生产通用的 4位微处理器 4040。1972年,Intel公司又生产了 8位微处理器 8080。通常将 Intel 4004、Intel 4040和 Intel 8080等称为第一代微处理器。这些微处理器的字长为 4位或 8位,集成度大约为 2 000管/片,时钟频率为 1 MHz,平均指令执行时间约为 20 μs。

⑵ 第二代微处理器

第二代产品是 1973—1977年间的产品。Intel公司的 8080/8085、Zilog公司的 Z80、Motorola公司的 6800/6802、Rockwell公司的 6502。这些微处理器的时钟频率为 2~4MHz,平均指令执行时间为 1~2 μs,集成度超过 5 000管/片。在这个时期,微处理器的设计和生产技术已相当成熟,配套的各类器件和接口芯片

也很齐全。以微处理器为核心的微型计算机技术也比较成熟。微型机开始广泛应用于事务管理、过程控制、通信、教育等领域。

⑶ 第三代微处理器

1978—1980年微处理器进入了超大规模电路时代，从16位微处理器时代开始，一块硅片上可容纳几万个晶体管。一些厂家推出了性能可与过去中档小型计算机相比的16位微处理器。这一时期的产品通常称为第三代微处理器。这中间有代表性的产品是Intel 8086/8088、Zilog的Z8000、Motorola的M68000。这些微处理器的时钟频率为4～8 MHz，平均指令执行时间为0.5 μs，集成度为20 000～60 000管/片。1980年以后，半导体厂家在提高电路的集成度、速度和功能方面取得了很大进展，相继出现了Intel 80286，Motorola 68010这样一些集成度高达10万管/片，时钟频率10 MHz，平均指令执行时间约为0.2 μs的16位高性能微处理器。进入80年代，计算机厂家IBM（国际商用机器）公司开始进入微型机领域，其产品IBM PC/XT（使用8088微处理器）和IBM PC/AT（使用80286微处理器）成为微型机领域的主流产品。1982年推出的Intel 80286，属于高档16位微处理器，具有16位数据总线、24位地址线，有实地址模式和虚地址保护模式。许多厂家开始生产与IBM PC兼容的产品，使得IBM PC迅速得到普及并很快地成为工业标准，许多兼容机厂家已成为独立计算机厂家，这使计算机变得廉价，从而微型机进入了更多的领域。

⑷ 第四代微处理器

1984年以后进入了第四代，该代产品是32位微处理器。1984年7月，Motorola公司推出了MC68020，1985年Intel推出了80386。它们的主要特征是，数据总线32位、地址总线32位、有实地址模式、虚地址保护模式和虚拟8086模式。

1989年Intel推出了80486，其结构是80386加80387（数学协处理器），还集成了8 KB Cache（高速数据缓冲器）。使用时钟倍频技术，部分采用RISC技术、突发总线技术。它虽然在CPU结构上和80386无根本差异，但它使得微型机结构更简单、性能更佳。

⑸ 第五代微处理器

1993年Intel Pentium（奔腾）32位微处理器推出。5级超标量结构，分支预测技术，有64条数据线，32位地址线。后来推出的Pentium MMX（多能奔腾）又增加了75条多媒体应用指令。

⑹ 第六代微处理器

代表性的产品有：

Pentium Pro（高能奔腾）—— 64位数据线、36位地址线

Pentium II(奔腾 II)——双独立总线结构

Pentium III(奔腾 III)——增加了 70条 SSE指令

⑺ 第六代后的微处理器

Pentium IV仍然是 32位的微处理器,采用超级管道技术,增加了 144条 SSE2指令,ALU在 2倍的处理器核心时钟频率上运行。

Itanium是 64位微处理器,采用 EPIC技术、RISC技术和 CISC技术,三级高速缓存。

2.2.3 微型计算机的分类

1. 按微处理器的字长分类

按微处理器的字长可分为 8位、16位、32位和 64位微型计算机。

2. 按用途分类

⑴ 通用微型计算机

通用微型计算机是能够面向各种不同应用的微型计算机系统,具有强大的处理能力和较大的体积,拥有较多的外设和各种接口。主要包括个人计算机(PC)和工作站(workstation)。

个人计算机:是一种通用微型机,体积小,价格低廉,并主要为每次一人使用,用户界面“友好”。又可分为:台式、便携式、手持式。主流产品使用 Intel公司的微处理器或兼容的微处理器,使用 Windows或 Linux操作系统。1981年 8月 12日,IBM发布了第一台 PC,开创了个人电脑时代。20多年后的今天,PC已在科学、教育、商务、办公、生产等领域获得了广泛的应用。

工作站:工作站是指具有完整人机交互界面,高性能的计算,可配置大容量的内存和硬盘,I/O和网络功能完善,使用多任务、多用户操作系统的小型通用个人化的计算机系统。目前工作站普遍采用 RISC处理器芯片,32位结构或 64位结构,工作站采用的主要是 UNIX操作系统。工作站在工程领域、商业领域及办公领域中获得广泛的应用。

⑵ 嵌入式计算机

另一类微型计算机安装在特定的应用系统中,使用者甚至感觉不到其存在。一般地,把带有微处理器的专用微机系统称为嵌入式计算机(Embedded Computer)或嵌入式系统。它是以应用为中心、以计算机技术为基础、软硬件可裁剪,适于应用系统对功能、可靠性、成本、体积、功耗有较高要求的专用计算机系统。嵌入式系统具有高集成度、低功耗的特点,其操作系统和功能软件一般都固化在芯片中。软件硬件都以高效率为原则,以实现在最小的硅片面积上实现

更高的性能,以保证其竞争力。

作为系统核心的微处理器有三类:微控制器(MCU)、数字信号处理器(DSP)和嵌入式微处理器(EMPU)。嵌入式系统广泛应用于仪器仪表、工业控制、测控系统、通讯设备、医疗设备、生产控制设备、武器装备等,特别是在移动电话、MP3播放器、PDA等新型个人消费类产品中的应用有着十分巨大的市场。

3. 按微型计算机的组成和规模分类

(1) 多板机:微型计算机的各组成部分安装在多个印刷电路板上的微型计算机。如台式 PC 机、便携式 PC 机等。

(2) 单板机:微型计算机的各组成部分安装在一个印刷电路板上的微型计算机。一般用于教学、实验等。

(3) 单片机:微型计算机的各组成部分集成在一个超大规模集成电路芯片上,则称之为单片微型计算机,简称单片机。因单片机广泛应用于嵌入式系统,又被称为微控器或嵌入式计算机。

2.2.4 微型计算机的结构

微型计算机的结构采用的是总线结构。如图 2.2所示。微处理器、存储器、输入和输出接口电路构成了计算机的主机;输入设备和输出设备统称为计算机外部设备,简称外设。以微型计算机为主体,配上系统软件和外设之后,就构成了微型计算机系统。

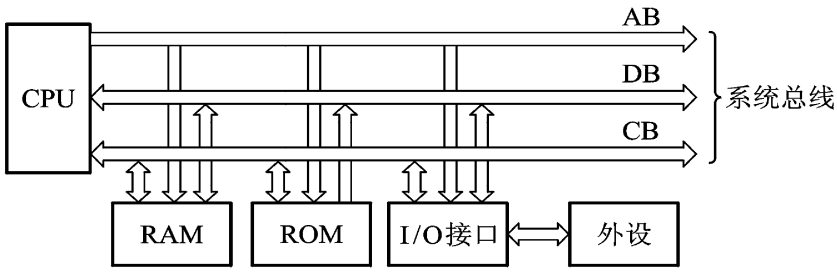


图 2.2 微型计算机的总线结构

总线是传输信号的一组导线,作为微机各部件之间信息传输的公共通道。一个部件只要符合总线标准,就可以连接到使用这种总线标准的系统中。这样的结构使得系统中各功能部件之间的相互关系变成了各个部件面向总线的单一关系,不仅简化了整个系统,而且使系统的进一步扩充变得非常方便。总线结构这种模块化(或称为积木化)特点使得微机系统部件的组成相当灵活,实现起来

也相当简捷。

微处理器通过系统总线实现和其他组成部分的联系。总线把微处理器、存储器和 I/O接口电路（外部设备与微型计算机相连的协调电路）有机地连接起来，所有的地址、数据和控制信号都经过总线传输。

微机的系统总线按功能分成三组，即数据总线（DB, Data Bus）、地址总线（AB, Address Bus）和控制总线（CB, Contol Bus）。

DB是传输数据或代码的一组通信线，其宽度（总线的根数）一般与微处理器的字长相等。例如，16位微处理器的 DB有 16根，分别以 D_{15} ～D_0 表示，D_0 为最低数据线。DB上的数据信息在微处理器与存储器或 I/O接口之间的传送可以是双向的，即 DB上既可以传送读信息，也可以传送写信息。微型计算机讲到的“读”或“写”都是以微处理器为主导地位而言的。

AB是传输地址信息的一组通信线，是微处理器访问外界用于寻址的总线。AB是单向的，其根数决定了可以直接寻址的范围。例如，某 8位微处理器的 AB有 16根，分别用 A_{15} ～A_0 表示，A_0 为最低位地址线，A_{15} ～A_0 可以组合成 2^{16} = 65 536（64 K）个不同地址值，可寻址范围 0000H～FFFFH。

CB是传送各种控制信号的一组通信线。控制信号是微处理器和其他芯片间相互联络或控制用的。其中包括微处理器发给存储器或 I/O接口的输出控制信号，如读信号（RD）、写信号（WR）等，还包括其他部件送给微处理器的输入控制信号，如时钟信号（CLK）、中断请求信号（INTR和 NMI）、准备就绪信号（READY）等。控制信号间是相互独立的，其表示方法采用能表明含义的缩写英文字母符号。若符号上有一横线，表示负逻辑有效，否则为正逻辑有效。

2.3 8086微处理器

8086是 Intel系列的 16位微处理器，有 16根数据线和 20根地址线。因为可用 20位地址，所以可寻址的地址空间达 2^{20}即 1 M字节单元。

8086工作时，只要一个 5 V的电源和一相时钟，时钟频率为 5 MHz，时钟周期为 200 ns。

几乎在推出 8086微处理器的同时，Intel公司还推出了一种准 16位的微处理器 8088，8088的内部寄存器、内部运算部件以及内部操作都是按 16位设计的，但对外的数据总线只有 8条。推出 8088的主要目的是为了与当时已有的一整套 Intel外围设备接口芯片直接兼容使用。

2.3.1 8086的编程结构

要掌握一个CPU的工作性能和使用方法,首先应该了解它的编程结构。所谓编程结构,就是指从程序员和使用者的角度看到的结构,当然,这种结构与CPU内部的物理结构和实际布局是有区别的。从功能上,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2.3是8086的编程结构图。

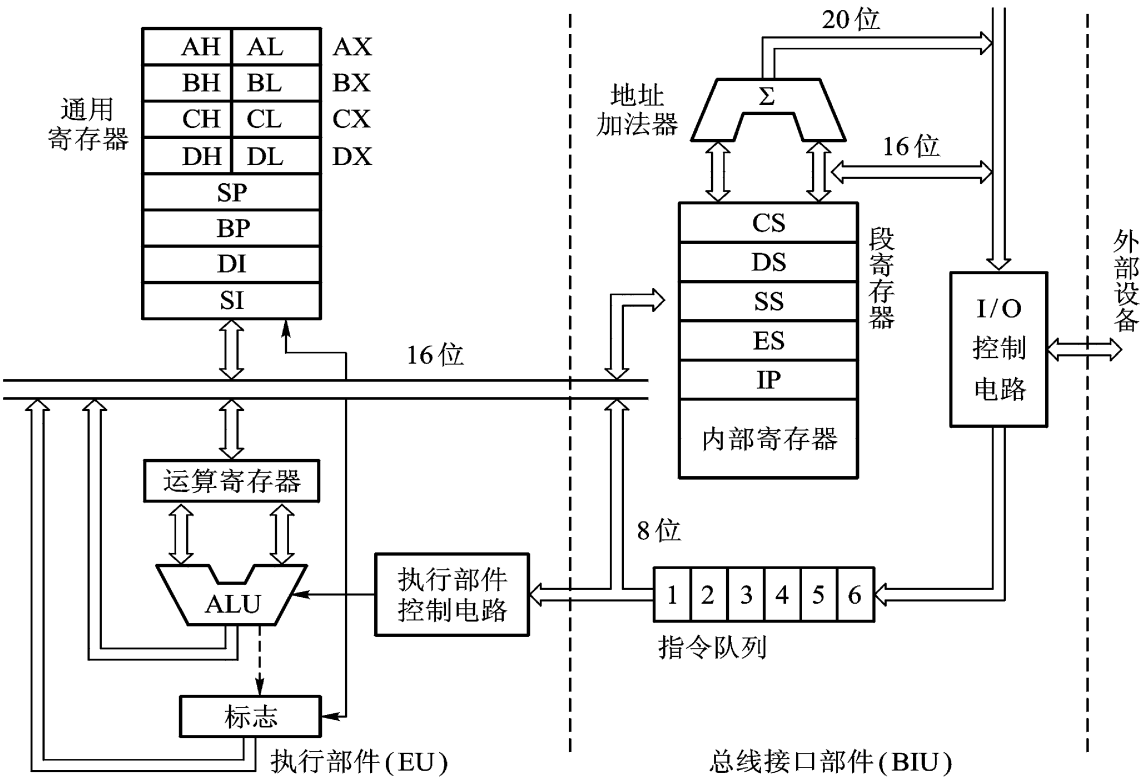


图 2.3 8086的编程结构图

1. 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,包括取指令操作以及对存储器读写数据操作和对I/O接口的读写操作。具体讲,总线接口部件要从内存取指令送到指令队列;CPU执行指令时,总线接口部件要配合执行部件从指定的内存单元或者外设端口中取数据,将数据传送给执行部件,或者把执行部件的操作结果传送到指定的内存单元或外设端口中。

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。

⑴ 段寄存器

8086 CPU采用段地址、段内偏移地址两级存储器寻址方式,段地址和段内偏移地址均为 16位。 8086内部根据需要设置了 4个段寄存器,用于存放段的高 16位地址,称为段的逻辑地址,4个段寄存器分别是:

CS:16位代码段寄存器 (Code Segment Register)

DS:16位数据段寄存器 (Data Segment Register)

SS:16位堆栈段寄存器 (Stack Segment Register)

ES: 16位附加段寄存器 (Extra Segment Register)

采用段地址的优点是:解决了 16位寄存器如何访问大于 64 KB的内存空间的问题;可以实现程序重定位,即一个小于 64 KB的程序可通过改变段寄存器的内容放到 1 MB空间中任意段位置,从而为同时运行多道程序提供了方便。

⑵ 20位地址加法器

8086可用 20位地址寻址 1 M字节的内存空间,但 8086内部所有的寄存器都是 16位的,8086 CPU采用段地址、段内偏移地址两级存储器寻址方式,由一个 20位地址加法器根据 16位段地址和 16位段内偏移地址计算出 20位的物理地址。 20位物理地址的获得方法是:将 CPU中的 16位段寄存器内容左移 4位 (×16,或写成 ×10H)得到该段的 20位物理地址,与 16位的逻辑地址 (又称偏移地址,即所寻址单元相对段首的偏移量)在地址加法器内相加,得到所寻址单元的 20位物理地址。根据寻址方式的不同,偏移地址可以来自指令指针寄存器 (IP)或其他寄存器。假设 CS=8211H, IP=1234H,则该指令单元的 20位物理地址为:

$$PA = 8311H \times 10H + 1234H = 83110H + 1234H = 84344H。$$

⑶ 16位指令指针寄存器 IP (Instruction Pointer)

其功能和 8位微处理器中的程序计数器功能相似。由于 8088取指令和执行指令同时进行,Intel公司用指令指针寄存器 IP代替 8位机的程序计数器 PC, IP总是保存着 EU要执行的下一条指令的偏移地址,而不是像 8位的 PC总是保存下一个取指令的地址。程序不能直接对指令指针寄存器进行存取,但能在程序运行中自动修正,使之指向要执行的下条指令,有些指令 (如转移、调用、中断、返回)能使 IP的值改变,或使 IP的值存进堆栈,或由堆栈恢复原有的值。

⑷ 指令队列缓冲器

8086有 6字节指令队列缓冲器,8088有 4字节指令队列缓冲器。在执行指

令的同时,可以从内存中取出下一条或下几条指令放到缓冲器中,一条指令执行完后,可立即译码执行下一条指令,从而解决了以往 CPU取指令期间,运算器的等待问题。

⑸ 输入/输出控制电路 (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。8086有 20条地址线,16条数据线,由输入/输出控制电路控制分时复用 CPU芯片的 16条引脚。

⑹ 内部暂存器

用于内部数据的暂存,该部分对用户透明 (几乎感觉不到它的存在,在编程时可不予理会),用户无权访问。

8086/8088的总线接口部件和执行部件并不是同步工作的,它们是按以下流水线技术原则管理:

1) 每当 8086的指令队列中有两个空字节,或者 8088的指令队列中有一个空字节时,总线接口部件就会自动把指令取到指令队列中。

2) 每当执行部件准备执行一条指令时,它会从总线接口部件的指令队列前部取出指令的代码,然后再用几个时钟周期去执行指令。在执行指令的过程中,如果必须访问存储器或者输入/输出设备,那么,执行部件就会请求总线接口部件,进入总线周期,完成访问内存或者输入/输出端口的操作 如果此时总线接口部件正好处于空闲状态,那么,会立即响应执行部件的总线请求。但有时会遇到这样的情况,执行部件请求总线接口部件访问总线时,总线接口部件正在将某个指令字节取到指令队列中,此时总线接口部件将首先完成这个取指令的总线周期,然后再去响应执行部件发出的访问总线的请求。

3) 当指令队列已满,而且执行部件又没有总线访问时,总线接口部件便进入空闲状态。

4) 在执行转移指令、调用指令和返回指令时,下面要执行的指令就不是在程序中紧接着的那条指令了,而总线接口部件向指令队列装入指令时,总是按顺序进行的,这样,指令队列中已经装入的字节就没有用了。遇到这种情况,指令队列中的原有内容被自动消除,总线接口部件会接着向指令队列中装入另一个程序段中的指令。

2. 执行部件 (EU)

执行部件的功能就是负责指令的执行。

从编程结构图可见到,8086CPU内部共有 14个寄存器,其中执行部件含有 9个寄存器,总线接口部件含有 5个寄存器。这些寄存器在指令中的隐含使用见表 2.1。

表 2.1　寄存器在指令中的隐含使用表

寄存器名称	含　义	所属部件
AX	累加器,还用于乘、除法和输入/输出的数据寄存器	EU
BX	基数寄存器,在间接寻址时作为地址寄存器和基址寄存器	
CX	计数寄存器,在循环和字符串操作中作为循环计数器	
DX	数据寄存器,还用于乘、除法,在输入输出指令中作间址寄存器	
BP	基数指针,在间接寻址时作为基址寄存器 (注意:默认 SS段)	EU
SP	堆栈指针,用于指示栈顶元素的地址	
SI	源变址寄存器,在间接寻址时作为地址或变址寄存器 在字符串操作中作为源变址寄存器	
DI	目的变址寄存器,在间接寻址时作为地址或变址寄存器 在字符串操作中作为目的变址寄存器	
F	标志寄存器,又称程序状态寄存器 (PSW),用来保存和反映运算操作结果的特征,以及 CPU内部的某种控制状态	
CS	代码段寄存器,用于存放程序代码段的逻辑地址	BIU
DS	数据段寄存器,用于存放数据段的逻辑地址	
ES	附加段寄存器,用于存放附加段的逻辑地址	
SS	堆栈段寄存器,用于存堆栈段的逻辑地址	
IP	指令指针寄存器,用于存放下一条要执行的指令的逻辑地址	

(1) 4个通用寄存器 AX、BX、CX、DX

EU中有 4个 16位的寄存器 AX、BX、CX和 DX,一般用来存放 16位数据,故称为数据寄存器。每个数据寄存器又可分为两个 8位的寄存器,即 AH、AL、BH、BL、CH、CL、DH、DL,用以存放 8位数据,它们均可独立使用。数据寄存器主要用来存放操作数或中间结果,以减少访问存储器的次数。

多数情况下,这些数据寄存器是用在算术运算或逻辑运算指令中,以进行算术逻辑运算。在有些指令中,它们则有特定的用途:如 AX作累加器用;BX作基址 (Base)寄存器,如在查表指令 XLAT中存放表的起始地址;CX作计数 (Count)寄存器,如在数据串操作指令的 REP中用 CX存放数据单元的个数作为循环操作的次数;DX作数据 (Data)寄存器,如在字的除法运算指令 DIV中,存放余数。

⑵ 4个专用寄存器 SP、BP、SI、DI

指针寄存器 SP和 BP用来存取位于当前堆栈段中的数据,但 SP和 BP在使用上有区别。入栈 (PUSH)和出栈 (POP)指令是由 SP给出栈顶的偏移地址,故称为堆栈指针寄存器。BP则是用来存放位于堆栈段中的一个数据区基址的偏移地址的,故称作基址指针寄存器。

变址寄存器 SI和 DI是用来存放当前数据段的偏移地址的。在字符串操作中,源操作数地址的偏置放于 SI中,所以 SI称为源变址寄存器;目的操作数地址的偏置放于 DI中,所以 DI称为目的变址寄存器。

⑶ 状态标志寄存器 F

8086 CPU的状态标志寄存器是一个十六位的寄存器,其中 9个位用作标志位:状态标志位有 6个,控制标志有 3个。状态标志寄存器如图 2.4所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

图 2.4 状态标志寄存器

状态标志中用 6位来反映 EU执行算术或逻辑运算以后的结果特征。这 6位都是逻辑值,判断结果为逻辑真 (true)时,其值为 1;判断结果为逻辑假 (false)时,其值为 0。

CF (Carry Flag)进位标志:CF=1表示指令执行结果在最高位上产生了一个进位或借位;CF=0则无进位或借位产生。

AF (Auxiliary Carry Flag)辅助进位标志:AF=1,表示结果的低 4位产生了一个进位或借位;AF=0则无此进位或借位。

ZF (Zero Flag)零标志:ZF=1,表示运算结果为零;ZF=0,则结果不为零。

SF (Sign Flag)符号标志:SF=1,表示运算结果为负数,即结果的最高位为 1;SF=0,则结果为正数,最高位为 0。

PF (Parity Flag)奇偶标志:PF=1,表示指令执行结果低 8位中有偶数个 1;PF=0,则结果中有奇数个 1。

OF (Over flow Flag)溢出标志:当运算过程中产生溢出时,会使 OF为 1。所谓溢出,就是当字节运算的结果超出了范围 -128～+127,或者当字运算的结果超出了范围 -32 768～+32 767时称为溢出。计算机在进行加法运算时,每当判断出低位向最高有效位产生进位,而最高有效位往前没有进位时,便得知产生了溢出,于是 OF为 1;或者反过来,每当判断出低位向最高位无进位,而最高位往前却有进位时,便得知产生了溢出,于是 OF为 1。在减法运算

时,每当判断出最高位需要借位,而低位并不向最高位产生借位时,OF置 1;或者反过来,每当判断出低位从最高位有借位,而最高位并不需要从更高位借位时,OF置 1。

为对上述状态标志有更具体的了解,举两个例子。

比如,执行下面两个数的加法:

$$\begin{array}{rrrrr} & 0010 & 0011 & 0100 & 0101 \\ + & 0011 & 0010 & 0001 & 1001 \\ \hline & 0101 & 0101 & 0101 & 1110 \end{array}$$

由于运算结果的最高位为 0,所以,SF=0;而运算结果本身不为 0,所以,ZF=0;低 8位所含的 1的个数为 5个,即有奇数个 1,所以,PF=0;由于最高位没有产生进位,所以,CF=0;又由于第 3位没有往第 4位产生进位,所以,AF=0;由于低位没有往最高位产生进位,最高位往前也没有进位,所以,OF=0。

又比如,执行下面两个数的加法:

$$\begin{array}{rrrrr} & 0101 & 0100 & 0011 & 1001 \\ + & 0100 & 0101 & 0110 & 1010 \\ \hline & 1001 & 1001 & 1010 & 0011 \end{array}$$

显然,由于运算结果的最高位为 1,所以,SF=1;由于运算结果本身不为 0,所以 ZF=0;由于低 8位所含的 1的个数为 4个,即含有偶数个 1,所以,PF=1;由于最高位没有往前产生进位,所以,CF=0;运算过程中,第 3位往第 4位产生了进位,所以,AF=1;由于低位往最高位产生了进位,而最高位没有往前产生进位,所以 OF=1。

当然,在绝大多数情况下,一次运算后,并不对所有标志进行改变,程序也并不需要对所有的标志作全面的关注。一般只在某些操作之后,对其中某个标志进行检测。

控制标志位有 3个,即 DF、IF、TF。

DF (Direction Flag) 方向标志:这是控制串操作指令用的标志。如果 DF为 0,则串操作过程中地址会不断增值;反之,如果 DF为 1,则串操作过程中地址会不断减值。

IF (Interrupt Enable Flag) 中断允许标志:这是控制可屏蔽中断的标志。如果 IF为 0,则中断是关闭的,此时 CPU不能对可屏蔽中断请求做出响应;如果 IF为 1,则中断是打开的,此时 CPU可以接受可屏蔽中断请求。

TF (Trap Flag) 跟踪标志:如果 TF为 1,则 CPU按跟踪方式执行指令。

这些控制标志一旦设置之后,便对后面的操作产生控制作用。

状态标志的状态表示在 IBM PC XT/AT机,可由调试程序 (DEBUG) 显示出来,所用符号如表 2.2所示。

表 2.2 状态标志表

标志位名	为 1对应符	为 0对应符
OF	OV	NV
DF	DN	UP
IF	EI	DI
SF	NG	PL
ZF	ZR	NZ
AF	AC	NA
PF	PE	PO
CF	CY	NC

2.3.2 8086的存储器组织

8086有 20根地址线,因此,具有 2^{20} =1M字节的存储器地址空间。这 1M字节的内存单元按照 00000H～FFFFFH来编址。

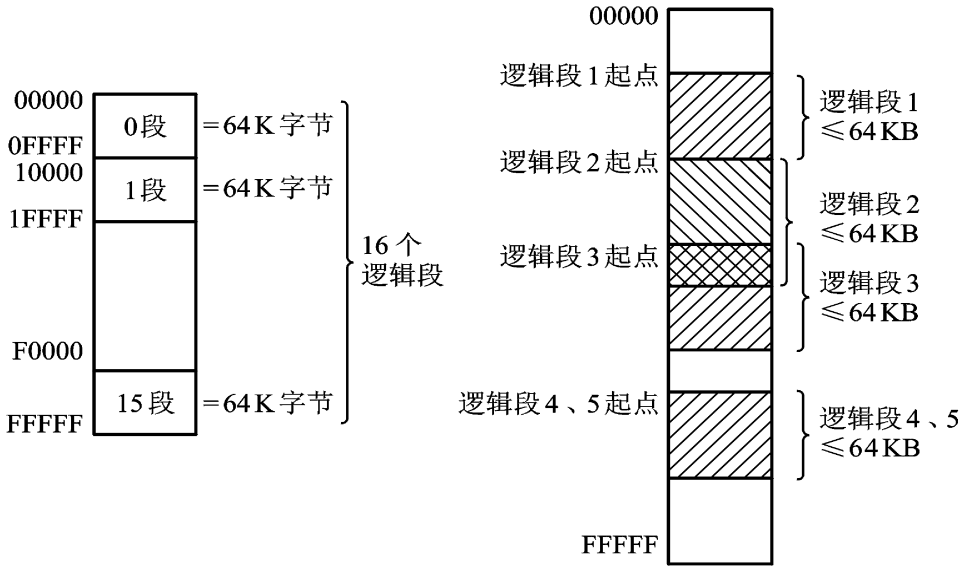
1. 存储器的分段

8086存储器操作采用了典型的存储器分段技术,对存储器的寻址操作不是直接用 20位的物理地址,而是采用段地址加段内偏移地址的二级寻址方式,即先用 16位的段地址经过简单运算 (左移 4位)得到单元所在段的物理地址,再加上该单元的 16位逻辑地址 (相对于段首的偏移地址)即可得到该单元的物理地址。如图 2.5a所示。

对于任何一个物理地址,可以唯一地被包含在一个逻辑段中,也可包含在多个相互重叠的逻辑段中,只要有段地址和段内偏移地址就可以访问到这个物理地址所对应的存储空间。如图 2.5b所示。

在 8086/8088存储空间中,把 16个字节的存储空间称作一节 (Paragraph)。为了简化操作,要求各个逻辑段从节的整数边界开始,也就是说段首地址低 4位应该是“0”,因此就把段首地址的高 16位称为“段基址”,存放在段寄存器 DS或 CS或 SS或 ES中,段内的偏移地址存放在 IP或 SP中。

若已知当前有效的代码段、数据段、附加段和堆栈段的段基址分别为 1055H、250AH、8FFBH和 EFF0H,那么它们在存储器中的分布情况如图 2.6所示。



(a) 存储器的逻辑分段　　(b) 存储器逻辑段的首地址

图 2.5　存储器的分段结构

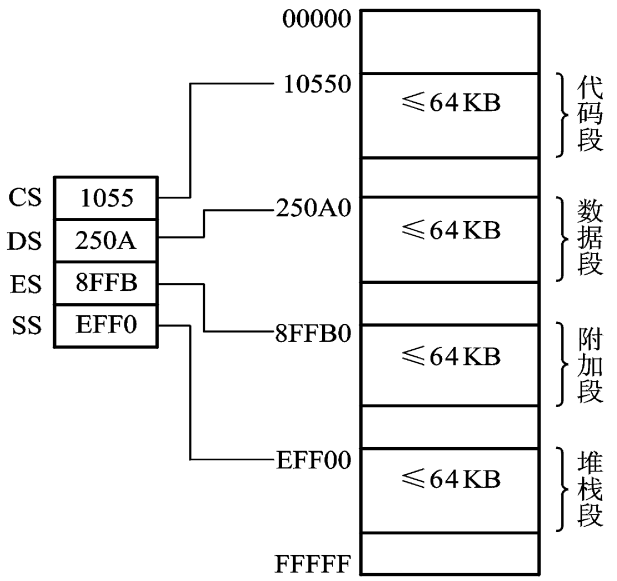


图 2.6　存储器逻辑段分布举例

2. 存储器中的逻辑地址和物理地址

采用分段结构的存储器中，存储单元的逻辑地址由段基址和偏移地址（也称有效地址 EA）两个部分构成，它们都是无符号的 16位二进制数。

任何一个存储单元对应一个 20位的物理地址（PA），也称为绝对地址，它是

由逻辑地址变换得来的。可以把每一个存储单元看成是具有两种类型的地址：物理地址和逻辑地址。物理地址就是实际地址，它具有 20位的地址值，它是唯一标志 1MB存储空间的某一个字节的地址。逻辑地址由段基址和偏移地址组成。程序以逻辑地址编址，而不是用物理地址。当 CPU需要访问存储器时，必须完成如下的物理地址运算：

物理地址 =段基址 ×10H +偏移地址

同一个物理地址可以由不同的段地址和偏移量组合来得到。比如：CS＝0000，IP＝306AH，物理地址为 0306AH；而 CS＝0300H，IP＝006AH，物理地址也是 0306AH。当然，还可以有许多别的组合。

物理地址的形成如图 2.7所示，它是通过 CPU的总线接口部件 BIU的地址加法器来实现的。

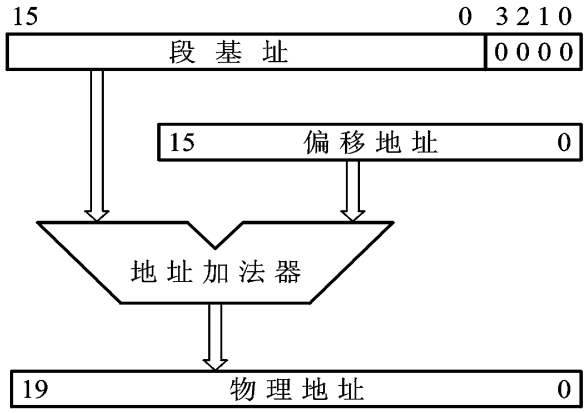


图 2.7 物理地址的形成

例如，代码寄存器 CS＝2000H，指令指针寄存器存放在偏移地址 IP＝2200H，存储器的物理地址为 20000H +2200H ＝22200H。

再以刚复位后的取指令动作为例来说明物理地址的形成。复位时，除了 CS＝FFFFH外，8086的其他内部寄存器的值均为 0，指令的物理地址应为 CS的值乘 16，再加 IP的值，所以，复位后执行的第一条指令的物理地址为：FFFF0H

因此，在存储器编址时，将高地址区分配给 ROM，而在 FFFF0H开始的单元中固化了一条无条件转移指令，转到系统初始化程序。

4个段寄存器分别指向 4个现行可寻址的分段的起始字节单元。程序指令存放在代码段中，代码段地址来源于代码段寄存器，偏移地址来源于指令指针 IP。当涉及一个堆栈操作时，堆栈段地址寄存器为 SS，偏移地址来源于栈指针寄存器 SP。当涉及一个操作数时，则由数据段寄存器 DS或附加段寄存器 ES作为段地址寄存器，而偏移地址是由 16位的偏移量得到。16位偏移量的确定与

指令的寻址方式有关。

如图 2.8所示,在 8086运行过程中,每当取指令时,CPU就会选择代码段寄存器 CS,再和指令指针 IP的内容一起形成指令所在单元的 20位物理地址;而当进行堆栈操作时,CPU就会选择堆栈段寄存器 SS,再和堆栈指针 SP或者基址指针 BP形成 20位堆栈地址;当要往内存写一个数据或者从内存读一个数据时,CPU就会选择数据段寄存器 DS,然后和变址寄存器 SI、DI或者通用寄存器 BX中的值形成操作数所在存储单元的 20位物理地址。

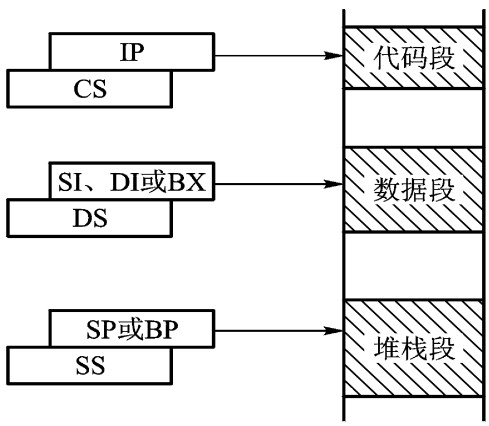


图 2.8 CS、DS、SS和其他寄存器组合指向存储单元

存储器中的操作数可以是 1个字节,也可以是 1个字。如果是字操作数,那么低位字节放在较低的地址单元,高位字节放在较高的地址单元。

扩展段 (也称附加段)一般是作为辅助的数据段来使用的,8086对数据的串操作指令多数都要用到扩展段寄存器。

存储器采用分段方法进行组织,带来了下列好处:

首先,可以使指令系统中的大部分指令只涉及 16位地址,这减少了指令长度,提高了执行程序的速度。也就是说,尽管 8086的存储空间高达 1 MB,但在程序执行过程中,一般不需要在 1M范围内转来转去,多数情况下只在一个较小的存储空间中运行,因此段寄存器的值是很少改变的。实际上,大多数指令运行时,并不涉及段寄存器的值,而只涉及 16位的偏移量。

此外,内存分段也为程序的浮动装配创造了条件。在每个模块设计时,程序员都希望系统不计较新设计的软件模块具体装在哪一个区域而都能正确运行,就是说,程序可以浮动地装配在内存任何一个区域中运行,这中间并不需要程序员为了使程序装配在某一处而修改程序代码。为了做到浮动装配,这要从两个方面提出要求:一是系统能够根据当时的内存使用情况将新引入的软件启动装置在合适的地方,这一点是由操作系统来实现的,8086的操作系统 MS-DOS具

有这个能力；二是要求程序本身是可浮动的，这就要求程序不涉及物理地址，进一步讲，就是要求程序和段地址没有关系，而只与偏移地址有关，这样的程序装在哪一段都能正常工作。要做到这一点，只需在程序设计时不牵涉段地址就行了，凡是遇到转移指令或调用指令则都采用相对转移或相对调用。可见，存储器采用分段结构之后，就可以使程序保持完整的相对性，也就具备了可浮动性。这样，操作系统对程序的浮动装配工作也变得比较简单，装配时只要根据当时的内存情况确定 CS、DS、SS、ES的值就行了。

尽管代码段、数据段、堆栈段及扩展段都可为 64 K，但实际应用中这些段之间可以有互相覆盖的部分。比如：

CS＝2000H，DS＝2100H

则

代码段的物理地址为 20000～2FFFFH；

数据段的物理地址为 21000～30FFFH。

这样，代码段和数据段之间有相当大的一个区域是相互重叠的，如图 2.9 示。

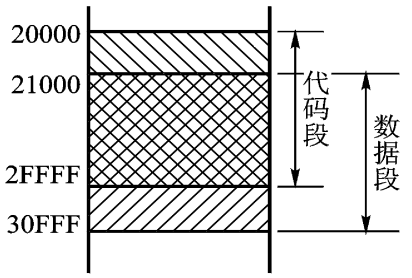


图 2.9 两个段互相重叠的情况

尽管从表面上看 CS、DS段寄存器为不同的值，而概念上代码段和数据段是在两个段中，但实际上程序代码后面就紧跟着数据，当数据所占区域也比较小时，两者或许在一个更小的范围内，还可能连同堆栈段都在这样一个范围内。

在 IBM PC/XT的 8086/8088系统中，存储器有几个部分的用处是固定的：

⑴ 00000～003FFH共 1 KB区域用来存放中断向量，这一区域称为中断向量表。中断向量实际上就是中断处理子程序的入口地址，每个中断向量占 4个字节，低地址的 2个字节为中断处理子程序入口地址的偏移地址，高地址的 2个字节为中断处理子程序的段地址。当然，对一个具体系统来说，一般并不需要多达 256个中断处理程序，因此，实际系统中的中断向量表的大部分区域是空白的。

⑵ B0000H～B0F9FH约 4KB是单色显示器的显示缓冲区,存放单色显示器当前屏幕字符所对应的 ASCII码和属性。

⑶ B8000H～BBF3FH约 16 KB是彩色显示器的显示缓冲区,存放彩色显示器当前屏幕像点所对应的代码。

⑷ 从 FFFF0H开始到存储器底部 FFFFFH共 16个单元,一般只用来存放一条无条件转移指令,转到系统的初始化程序。这样在系统加电或者复位时,就会自动转到 FFFF0H执行。

思考题与习题

2.1 解释下列名词:

⑴ 硬件、软件

⑵ 微处理器、微型计算机、微型计算机系统

⑶ 字节、字、字长

⑷ 物理地址、段地址、偏移地址

2.2 微型计算机由哪几个部分组成,各部分的基本功能是什么?

2.3 试说明标志寄存器的作用以及每个标志位的含义。

2.4 试说明 8086CPU中 EU和 BIU的功能。

2.5 试说明地址加法器的工作原理。

2.6 试说明 8086/8088存储器组织结构特点。

2.7 为什么 8086/8088CPU存储器组织采用分段结构?

2.8 说明冯·诺依曼计算机工作原理。

第3章

8086/8088指令系统与寻址方式

3.1 概　　述

1. 指令

指令是计算机能够识别和执行的指挥计算机进行操作的命令。指令系统是指微处理器能执行的各种指令的集合。微处理器主要功能是由它的指令系统来体现的,不同的微处理器有不同的指令系统,其中每一条指令对应着处理器的一种基本操作,这在设计微处理器时就已决定了。

计算机的指令有两种表示方式 机器码和助记符。机器码也称指令码,是机器能够接受的指令,但程序设计人员使用不便。助记符有利于程序编写,运行前需转换为机器码。

计算机指令码由操作码字段和操作数字段两部分组成。操作码字段指出所要执行的操作,而操作数字段指出指令操作过程中需要的操作数。例如一条加法指令机器码为:02CFH=000000　10　11001111,前 6个 0,表示加法操作,为操作码字段,剩余部分说明操作数。该指令的助记符为:

ADD　CL, BH, 即实现:CL← CL + BH

操作数可以有一个、两个或更多,通常分别称为一地址指令、二地址指令或单操作数指令、双操作数指令等。

ADD　　CL, BH 是二地址指令;

INC　　AC 是一地址指令。

在二地址指令格式中,提供两个操作数地址,分别为目标操作数(前面的那个)和源操作数(后面的那个)。当指令执行时,两个操作数同时参与运算,并把运算结果返回目标操作数,也就是说,目标操作数原有内容在运算后将会丢失。

2. 操作数

操作数是指令的操作对象。8086/8088指令系统中的操作数分为两类:数据操作数、转移地址操作数。

(1) 数据操作数

按存储位置,数据操作数分为:立即数、寄存器操作数、内存操作数、I/O操作数。

1) 立即数:指令中直接给出操作数本身。

【例】 MOV AL,8,其中 8为立即数。

2) 寄存器操作数:即操作对象是寄存器中的内容。上例所述指令中 AL为寄存器操作数。

3) 内存操作数:也称为存储器操作数,操作对象是内存中的数。

4) I/O操作数:指令中要操作的数据来自或送到 I/O端口。

(2) 转移地址操作数

这类操作数出现在程序跳转或程序调用指令中,指出程序要转移的目的地址。它也可以分为:立即数、寄存器操作数、存储器操作数,即要转移的目标地址包含在指令中或存放在寄存器、内存储器中。

3.2 数据寻址方式

指令中关于如何求出操作数有效地址的方法称为寻址方式。计算机按照指令给出的寻址方式求出操作数有效地址的过程,称为寻址操作。在程序设计中,有时需要直接写出操作数本身,有时希望给出操作数的地址,有时希望给出操作数所在地址的地址。为了满足程序设计需要,8086/8088给出了多种寻址方式,根据操作数的类型及来源大致分为三类:数据寻址、转移地址寻址、I/O寻址。本节讲解数据寻址方式,后两种分别见转移指令及 I/O指令部分。

8086/8088系列计算机有 7种基本的数据寻址方式:(1)立即寻址;(2)寄存器寻址;(3)直接寻址;(4)寄存器间接寻址;(5)寄存器相对寻址;(6)基址变址寻址;(7)相对基址变址寻址。后面 5种寻址方式属于存储器寻址,用来确定操作数所在的存储单元的有效地址 EA的计算方法。

1. 立即寻址

立即寻址即指令中直接给出操作数本身。采用该寻址方式的操作数与指令

代码一起存放在码段中。

【例】 MOV AX，1234H ；AX ← 1234H

源操作数的寻址方式为立即寻址。执行过程如图 3.1所示。

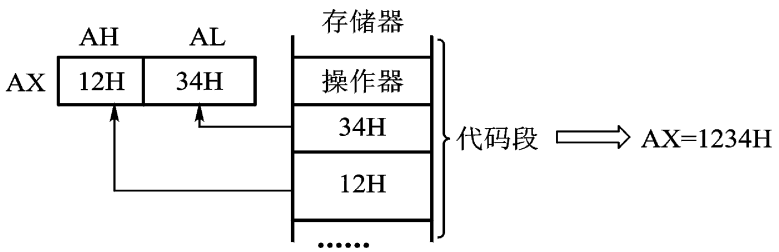


图 3.1 立即寻址示意

注意：

⑴ 立即寻址通常用于二地址指令中，且只能是源操作数。

⑵ 数据传送，应理解为复制传送，源操作数不会因为传送而失去数据。

2. 寄存器寻址

寄存器寻址是指操作数存放在寄存器中，指令中给出寄存器名。对于 16位操作数，寄存器可以是：AX, BX, CX, DX, SI, DI, SP, BP, CS, DS, SS, ES；对 8位操作数，寄存器可以是：AH, AL, BH, BL CH, CL, DH, DL。

【例】 以下指令中均采用了寄存器寻址方式，在每条指令后，说明了采用该方式的操作数。

⑴ MOV AX, 1234H ；目标操作数

⑵ MOV DX, AX ；目标操作数、源操作数

以 ⑵ 为例说明寄存器寻址执行过程。设 AX =5678H，指令执行过程如图 3.2所示。

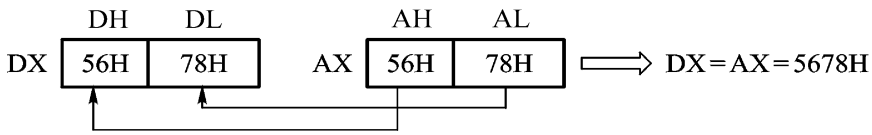


图 3.2 寄存器寻址示意

特点：

⑴ 操作数在寄存器中，寄存器在 CPU内部，指令执行时，操作就在 CPU的内部进行，不需要访问存储器来取得操作数，因而执行速度快。

⑵ 寄存器号比内存地址短，汇编后机器码长度最短。

⑶ 寄存器寻址方式既可用于源操作数，也可用于目标操作数，还可以两者都用寄存器寻址方式。

在编程中,如有可能,尽量使用寄存器寻址方式的指令。

注意:

⑴ 当指令中的源操作数和目标操作数均为寄存器时,必须采用同样长度的寄存器;

⑵ 两个操作数不能同时为段寄存器;

⑶ 目标操作数不能是码段寄存器。

除以上两种寻址方式外,下面 5种寻址方式的操作数均在存储器中,统称为内存寻址方式。当采用内存操作数时,必须注意双操作数指令中的两个操作数不能同时为内存操作数。

3. 直接寻址

直接寻址即指令中给出操作数所在存储单元的有效地址,缺省的段为数据段。为了区别于立即数,有效地址用“[]”括起。

【例】 以下指令中源或目标操作数采用了直接寻址方式

```
⑴ MOV    AX, [2000H]          ;AX ← (DS:2000H)
⑵ MOV    [1200], BL           ; (DS:1200H) ← BL
⑶ MOV    ES: [0100], AL       ; (ES:0100H) ← AL
```

说明:

DS:2000表示内存单元地址;(DS:2000)表示地址是 DS:2000内存单元的内容。同样,约定用寄存器名(如 AX)表示寄存器的内容,(寄存器名)(如(AX))表示以寄存器的内容为地址的内存单元内容。

以⑴为例,说明直接寻址的寻址过程。设 DS=4000H,则此指令将数据段中物理地址为 42000H单元的内容传送 AX寄存器。执行过程如图 3.3所示。

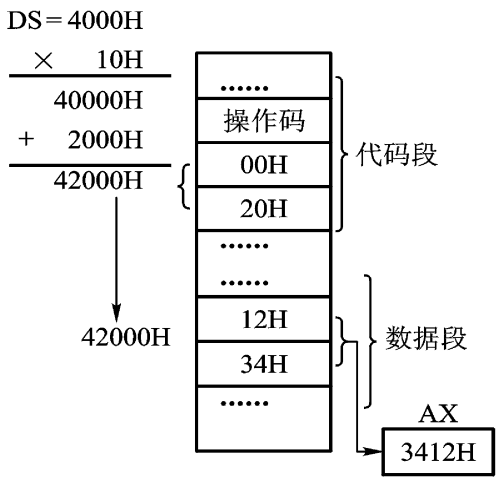


图 3.3　直接寻址过程示意

① 根据指令中给出的有效地址,得到存储单元的物理地址;

$$DS\times16+4000H=42000H$$

② 把该内存单元中的内容送到 AX 中;

③ 字在内存中的存储占两个内存单元,低字节在低地址,高字节在高地址,并以低字节的地址作为字的地址。

直接寻址允许数据存于附加段、堆栈段、码段,这称为"段跨越",此时,需要段说明,如例 (3) 中,数据存于附加段中,操作数物理地址为:ES×10H+0100H。

在汇编语言指令中,可以用符号地址代替数值地址。

【例】

```
VALUE   DB   12H, 34H, 56H        ;数据定义
MOV     AL, VALUE   或   MOV   AL, [VALUE]
```

寻址示意如图 3.4所示。

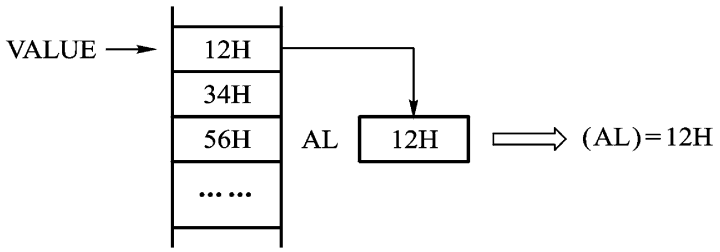


图 3.4 符号地址

4. 寄存器间接寻址

寄存器间接寻址是把内存操作数的有效地址存储于寄存器中,指令中给出存放地址的寄存器名。因为有效地址是 16位,所以,存放地址的寄存器必须是16位的。8086/8088中可以用于间接寻址的寄存器有基址寄存器 BX、BP和变址寄存器 SI、DI。为了区别于寄存器寻址,寄存器名用"[ ]"括起。

【例】

```
(1)  MOV     AX, [SI]          ;AX ← (DS:SI+1, DS:SI)
(2)  MOV     [BX], 1234H       ; (DS:BX+1, DS:BX) ← 1234H
```

不同的寄存器所隐含对应的段不同。采用 SI、DI、BX 寄存器,数据存于数据段中;采用 BP寄存器,数据存于附加段中,即操作数的物理地址计算式为:

$$物理地址=DS\times10H+SI或\ DI或\ BX$$

或

$$物理地址=SS\times10H+BP$$

以 (1) 为例,说明寄存器间接寻址的寻址过程。设 DS=3000H, SI=2000H,

寻址过程如图 3.5所示。

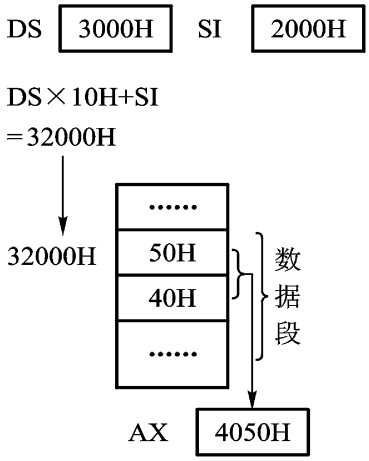


图 3.5　寄存器间址寻址示意图

① 根据指令中给出的寄存器及寄存器内容得到存储单元的物理地址

$$DS \times 16 + 2000H = 32000H$$

② 把该内存单元开始的两个字节的内容“送到”AX 中。低地址单元内容送到 AL中,高地址单元内容送到 AH 中。

5. 寄存器相对寻址

采用寄存器相对寻址时,操作数的有效地址分为两部分,一部分存于寄存器中,指令中给出该寄存器名;另一部分以偏移量的方式直接在指令中给出。

【例】

(1) MOV　　AL, 8[BX]

(2) MOV　　AX, COUNT[SI]

其中,寄存器前的值(如 8)或符号常量 COUNT为偏移量。可用于寄存器间接寻址的寄存器也可用于寄存器相对寻址,选用不同寄存器,对应的段不同,规律同寄存器寻址,即:

操作数的有效地址 EA_1 = SI/DI/BX + 8位 disp/16位 disp (disp代表偏移量)

或

$$EA_2 = BP + 8位\ disp/16位\ disp$$

操作数的物理地址　　$PA_1 = DS \times 10H + EA_1$

或

$$PA_2 = SS \times 10H + EA_2$$

以 (1) 为例,说明寄存器相对寻址的寻址过程。设 DS = 3000H, BX = 100H,

上例中源操作数的寻址过程如图 3.6所示。

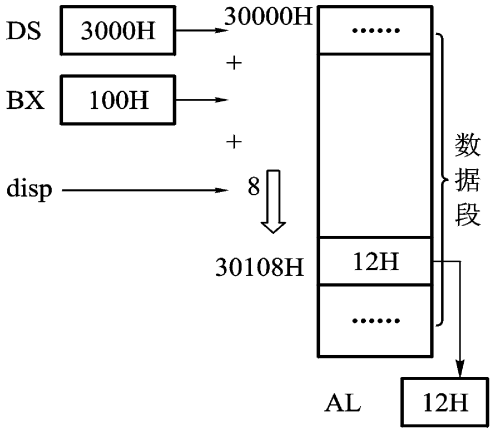


图 3.6 寄存器相对寻址示意

① 根据指令中给出的寄存器名、偏移量及寄存器内容,得到存储单元的物理地址;

$$DS\times16+BX+disp=30108H$$

② 把该内存单元中的内容传送到 AL中。

说明:

(1) 偏移量是符号数,8位偏移量的取值范围为:00～0FFH (即 +127～−128);16位偏移量的取值范围为:0000～0FFFFH (即 +32 765～−32 768)。

(2) IBM汇编允许用三种形式表示相对寻址,它们的效果是一样的,如:

```
MOV     AX, [BX] +6          ;标准格式
MOV     AX,6 [BX]            ;先写偏移值
MOV     AX, [BX +6]          ;偏移值写在括号内
```

6. 基址变址寻址

采用基址变址寻址时,操作数的有效地址分为两部分,一部分存于基址寄存器 (BX或 BP)中,另一部分存于变址寄存器 (SI或 DI)中,指令中分别给出两个寄存器名。操作数的有效地址为:

$$EA_1=BX+SI/DI$$

或

$$EA_2=BP+SI/DI$$

当基址寄存器选用 BX时,数据隐含存于数据段中;当基址寄存器选用 BP时,数据隐含存于堆栈段中,即操作数的物理地址为:

$$PA_1 = DS \times 10H + EA_1$$

$$PA_2 = SS \times 10H + EA_2$$

【例】

(1) MOV AL, [BP] [SI]

(2) MOV AX, ES: [BX] [DI]

以 (1) 为例,说明基址变址的寻址过程。设 SS=3000H, BP=100H, SI=5, 源操作数的寻址过程如图 3.7所示。

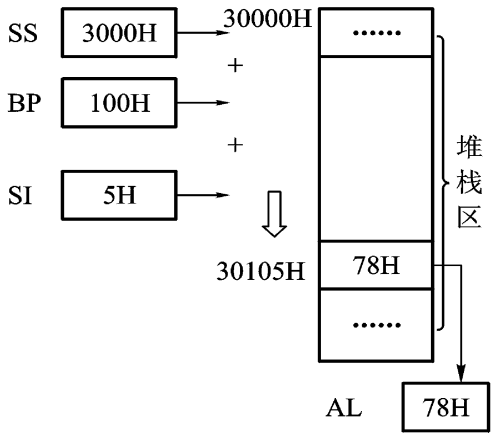


图 3.7　基址变址寻址示意

① 根据指令中给出寄存器名及寄存器内容,得到存储单元的物理地址;

$$SS \times 16 + BP + SI = 30105H$$

② 把该内存单元中的内容送到 AL中。

7. 相对基址加变址寻址

采用相对基址变址寻址时,操作数的有效地址分为三部分:一部分存于基址寄存器 SI或 DI中;一部分存于变址寄存器 BX或 BP中;一部分为偏移量。指令中分别给出两个寄存器名及 8位或 16位的偏移量。操作数的有效地址为:

$$EA_1 = BX + SI/DI + 8位/16位\ disp$$

或

$$EA_2 = BP + SI/DI + 8位/16位\ disp$$

当基址寄存器选用 BX时,数据隐含存于数据段中;当基址寄存器选用 BP时,数据隐含存于数据段中,即操作数的物理地址为:

$$PA_1 = DS \times 10H + EA_1$$

$$PA_2 = SS \times 10H + EA_2$$

【例】

(1) MOV AL,5[BP][SI]

(2) MOV AX,5[BX][SI]

以(1)为例,说明相对基址变址寻址的寻址过程。设SS=2000H,BP=1000H,SI=100H,源操作数的寻址过程如图3.8所示。

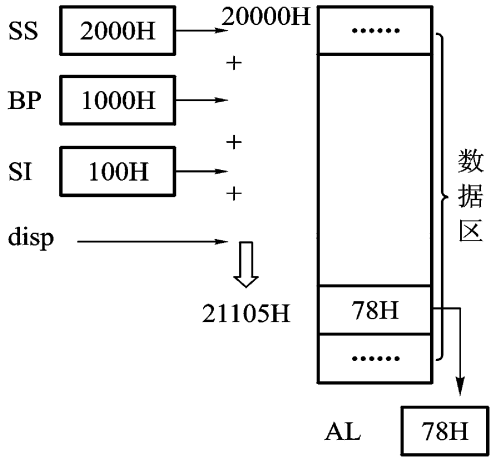


图3.8 相对基址变址寻址示意

① 根据指令中给出寄存器名、寄存器内容及偏移量,得到存储单元的物理地址:

$$SS\times 10H+BP+SI=21105H$$

② 把该地址开始连续两个内存单元中的内容送到AX中。

3.3 指令格式及指令执行时间

3.3.1 指令格式

助记符是帮助人们阅读与书写源程序的,计算机只能识别机器码,所以,由汇编语言编写的源程序需经过"汇编",将它翻译成机器指令组成的机器语言程序,才能由计算机识别并执行。一般而言,这一过程不必人工干预。这里以8086/8088指令为例简单介绍由指令助记符到机器码的基本原理,以便在必要时也可完成类似的工作或帮助你更好地掌握指令。

8086/8088指令系统用了一种灵活的,由1~6个字节组成的变字长的指令格式。每条指令包括操作码、寻址方式及操作数三个部分。如图3.9所示。通

常指令的第一字节为操作码,规定指令的操作类型,第二字节规定操作数的寻址方式,3～6字节依据指令的不同而取舍,指出存储器操作数地址的位移量或立即数,指令的字长可变主要体现在这里。

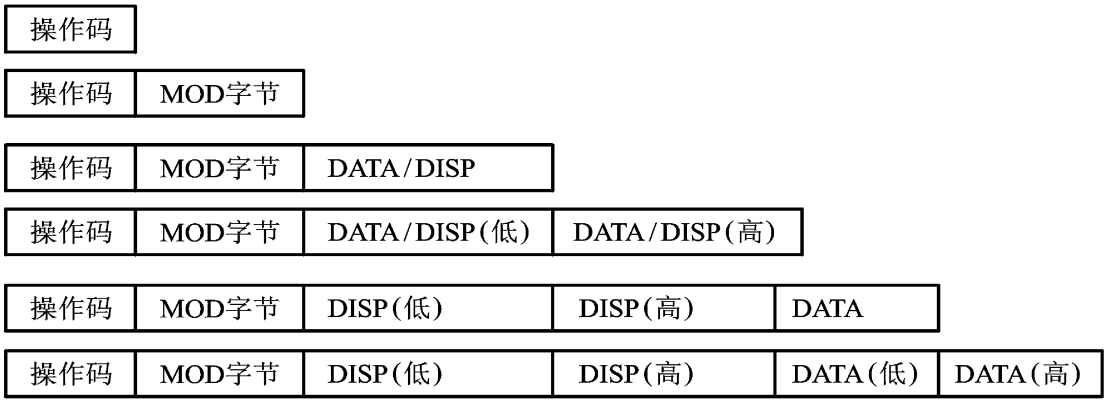


图 3.9　不同字长的指令格式

操作码/寻址方式字节格式如表 3.1。

表 3.1　操作码/寻址方式格式

第一字节								第二字节							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
操作码						D/S	W	MOD		REG			R/M		

第一字节中：

D——双操作数指令有效。D=0,源操作数为寄存器;D=1,目标操作数为寄存器。

S——使用立即寻址方式时有效。S=0,没有符号扩展;S=1,有符号扩展。

W——指示操作数类型。W=0,为字节;W=1,为字。

第二字节：

指出所用的两个操作数存放的位置,以及存储器中操作数有效地址 EA 的计算方法。

REG——规定一个寄存器操作数,见表 3.2。

MOD——指示另一个操作数在寄存器中还是在存储器中,若在存储器中,还用来指示该字节后有多少位移量。

R/M——当 MOD=11 (即寄存器寻址)时,指示第二操作数所在寄存器编号。

当 MOD=00、01或 10 (即存储器寻址)时,指示如何计算存储器中操作数地

址。见表 3.3。

表 3.2 REG 字段编码表

REG	W=1 (字操作)	W=0 (字节操作)
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

表 3.3 MOD 字段编码表

MOD	寻址方式
00	存储器寻址,没有位移量
01	存储器寻址,有 8位位移量
10	存储器寻址,有 16位位移量
11	寄存器寻址

表 3.4 各种 MOD与 R/M 字段组合编码及有关地址的计算

MOD / R/M	00	01	10	11	
				W=0	W=1
000	DS: [BX+SI]	DS: [BX+SI+D_8]	DS: [BX+SI+D_{16}]	AL	AX
001	DS: [BX+DI]	DS: [BX+DI+D_8]	DS: [BX+DI+D_{16}]	CL	CX
010	SS: [BP+SI]	SS: [BP+SI+D_8]	SS: [BP+SI+D_{16}]	DL	DX
011	SS: [BP+DI]	SS: [BP+DI+D_8]	SS: [BP+DI+D_{16}]	BL	BX
100	DS: [SI]	DS: [SI+D_8]	DS: [SI+D_{16}]	AH	SP
101	DS: [DI]	DS: [DI+D_8]	DS: [DI+D_{16}]	CH	BP
110	DS:D_{16}	SS: [BP+D_8]	SS: [BP+D_{16}]	DH	SI
111	DS: [BX]	DS: [BX+D_8]	DS: [BX+D_{16}]	BH	DI

表 3.4中的段寄存器是指无段跨越前缀的情况下所使用的隐含的段寄存器。如果指令中指定段跨越前缀,则在机器语言中使用放在指令之前的一个字节来表示。即:

00	SEG	110

SEG指定 4个段寄存器中的一个,见表 3.5。

表 3.5　段跨越时 SEG编码

SEG	段寄存器
00	ES
01	CS
10	SS
11	DS

根据以上说明,下面以加法指令为例,对机器指令做一些具体分析。加法指令的汇编语言格式为:

ADD　　dest, src　　;

操作可表示为:(dest) ← (dest) + (src),意思是:把源操作数 (src)与目标操作数 (dest)相加,结果存于目标操作数 (dest)中。

根据不同寻址方式,加法指令有三种格式。

格式一:

ADD　　mem/reg1, reg2

或

ADD　　reg1, reg2/mem

表示:

(1) 寄存器 (reg1)与寄存器 (reg2)内容相加,结果存于 reg1中;

(2) 寄存器 (reg1)与存储器 (mem)内容相加,结果存于 reg1中;

(3) 存储器 (mem)与寄存器 (reg2)内容相加,结果存于 mem中。

机器码为:

第一字节	第二字节			第三字节	第四字节
000000DW	MOD	REG	R/M	ADDR (低)	ADDR (高)

其中,第一字节中 000000为操作码;虚线框内容 (第三、四字节)位移量由寻址方式确定其有无。

【例】　ADD　　BH, CL

机器码:00000　1　0　11　001　111

①　②　③　④　⑤　⑥

①、②、③——第一字节

①——操作码

②——D = 1,目标操作数是寄存器 (目标操作数为一个操作数)

③——W =0,字节运算

④、⑤、⑥——第二字节

④——MOD =11,另一个操作数 (即源操作数)在寄存器中

⑤——REG = 001,一个操作数 (即目标操作数)所在寄存器为 CL

⑥——R/M = 111,另一个操作数 (即源操作数)所在寄存器为 BH

格式二：

ADD mem／reg, data

表示：寄存器（reg）内容或存储器（mem）内容与立即数相加，结果存于 reg／mem中。

机器码为：

第一字节	第二字节	第三字节	第四字节	第五字节	第六字节
100000SW	MOD 000 R／M	ADDR（低）	ADDR（高）	DATA（高）	DATA（低）

其中，第一、二字节均为操作码。当 W＝1（即字操作）时，S失效。

格式三：

ADD acc, data

表示：立即数与累加器（AL或 AX寄存器）内容相加，结果存于累加器中。

第一字节	第二字节	第三字节
0000010W	DATA（高）	DATA（低）

其中，000001为操作码，S位指定为 0。当 W＝0（即字节操作）时，用 AL寄存器；W＝1（即字操作）时，用 AX寄存器。

【例】 ADD AX, 123H

机器码为：0000010 1 0000001 00100011

①——操作码

②——W＝1，字操作

③——立即数的低 8位

④——立即数的高 8位

3.3.2 指令执行时间

一条指令的执行时间是取指令、取操作数、执行指令及传送结果所需时间的总和。IBM PC采用一种预取指令的方式工作，取指令时间与其他阶段的时间重叠可不予考虑，这样执行一条指令的时间即为基本执行时间和存取操作时间的总和。指令的基本执行时间因指令而异，互相之间差异很大，如表 3.6所示。而存、取操作数的时间因不同的寻址方式而有所不同，如表 3.7所示。当需要访问存储器取得操作数时，还需要考虑计算有效地址 EA的计算时间，如表 3.8所示。

表 3.6　指令执行基本时间

指令	寻址方式	时钟周期
加法 ADD	寄存器—寄存器	3
传送 MOV	寄存器—寄存器	2
整数乘法 IMUL	16位寄存器乘	128～154
整数除法 IDIV	16位寄存除	165～184
无条件转移 JMP	直接	15
条件转移	不转移	4
	转移	16

表 3.7　加法指令 ADD执行时间

寻址方式	时钟周期	访问存储器次数
寄存器—寄存器	3	0
存储器—寄存器	9+EA	1
寄存器—存储器	16+EA	2
立即数—寄存器	4	0
立即数—存储器	17+EA	2

【例】　计算 ADD指令在以下寻址方式下指令的执行时间。

⑴ 寄存器—寄存器方式；⑵ 存储器—寄存器方式；⑶ 寄存器—存储器方式

解：假设时钟频率为 5MHz，则一个时钟周期为 0.2 μs。

⑴ 寄存器—寄存器方式的 ADD指令，需要时间 t = 3 × 0.2 μs = 0.6 μs

⑵ 存储器—寄存器方式 ADD指令，如存储器使用相对基址变址方式，则

$$t = (9 + EA) \times 0.2 = (9 + 12) \times 0.2\ \mu s = 4.2\ \mu s$$

如果在此种寻址方式下，求得存储器的有效地址为奇数，则根据 IBM PC 的规定，需要多访问一次存储器，即要增加 4个时钟周期，则：

$$t = (9 + 12 + 4) \times 0.2\ \mu s = 5\ \mu s$$

⑶ 寄存器—存储器方式，如存储器使用相对基址变址方式，则：

$$t = (16 + EA) \times 0.2 = (16 + 12) \times 0.2\ \mu s = 5.6\ \mu s$$

如果此时是奇地址，则：$t = (16 + 12 + 4) \times 0.2\ \mu s = 6.4\ \mu s$

表 3.8　有效地址 EA 计算时间

寻址方式	时钟周期
直接	6
寄存器间接	5
寄存器相对	9
基址变址	7～8
相对基址变址	11～12

3.4　8086/8088指令系统

指令对于程序而言，如同人们说话与写作时的文字与必须遵循的语法。所以，指令学习过程中，应该重视指令的功能及指令的格式，在掌握了这些之后，应用它们编写程序解决实际问题，实现学习指令的真正目的。

8086/8088的指令按其功能大致可分为以下 6种：

1. 数据传送指令
2. 算术运算指令
3. 逻辑指令
4. 串操作指令
5. 程序控制指令
6. 处理机控制指令

为了表示方便，约定了一些符号，如表 3.9所示。

表 3.9 符号约定及含义

符号	含 义
i8	一个 8位的立即数
i16	一个 16位的立即数
imm	一个 8位或 16位的立即数
r8	一个 8位通用寄存器：AH、AL、BH、BL、CH、CL、DH、DL
r16	一个 16位通用寄存器：AX, BX, CX, DX, BP, SP, SI, DI
reg	一个 8位或 16位通用寄存器
seg	一个段寄存器：DS, ES, SS, CS
m8	一个 8位存储器操作数（包括所有的内存寻址方式）
m16	一个 16位存储器操作数（包括所有的内存寻址方式）
mem	一个 8位或 16位存储器操作数（包括所有的内存寻址方式）
dest	目标操作数
src	源操作数
port	I/O端口号

3.4.1 数据传送指令

数据传送指令是将数据或地址传送到寄存器或存储单元中。它分为以下 4类：①通用数据传送指令；②地址传送指令；③标志传送指令；④输入输出指令。

数据传送指令中，除第三类标志传送指令会对标志位产生影响外，其余的指令均不影响标志位。所以在下面的阐述中，对于一、二、四类指令，将不再说明它们对标志状态位的影响。

1. 通用数据传送指令：MOV、PUSH、POP、XCHG、XLAT

(1) 数据传送指令 MOV

格式：MOV　　dest, src

功能：将源操作数的内容“送”给目标操作数。即：(dest) ← (src)。

其中，src可以为：mem, seg, imm

dest可以为：mem, seg

MOV可以实现一个字节或一个字的传送，但要求 dest与 src相同类型，即长度相等。数据允许的传送方向如图 3.10所示。

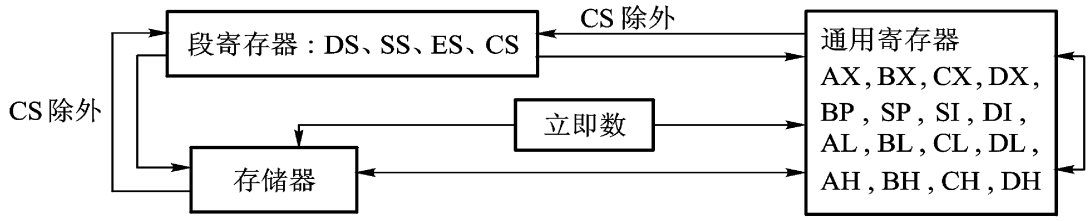


图 3.10　MOV指令数据传送方向示意

从图 3.10可知：(1) 段寄存器 CS不能作为目标操作数，即不能给 CS赋值；(2) 源操作数与目标操作数不能同时为内存操作数。

【例】　以下指令均为合法的传送指令，括号中为目标操作数与源操作数的寻址方式。

1) MOV　　AL, 5　　；(寄存器，立即数)

2) MOV　　AX, BX　　；(寄存器，寄存器)

3) MOV　　DS, AX　　；(段寄存器，寄存器)

4) MOV　　ES, DS　　；(段寄存器，段寄存器)

5) MOV　　ES:VAR, 12　　；(存储器，立即数)

6) MOV　　WORD PTR [BX], 12　　；(存储器，立即数)

说明：

5) 有段超越，VAR为符号地址；

6) 中的 WORD PTR指明存储器操作数的属性是字属性。

(2) 堆栈操作指令　PUSH、POP

堆栈是计算机的一种数据结构，数据的存取原则是“先进后出，后进先出”。好比货物堆放，先到的货物放在下面，后到的货物堆放在上面；取货物时，堆在上面的货物被先取走。

一般地，计算机系统中堆栈区建立在内存中，通过堆栈指针实现堆栈的管

理。堆栈指针总是指向栈顶。堆栈指针的初值为栈底。把数据从栈顶存入堆栈中,同时调整堆栈指针保持指向栈顶,这一操作称为入栈 (或压入);把数据通过栈顶从堆栈中取出,同时调整堆栈指针保持指向栈顶,称为出栈 (或弹出)。

在 8086系统中,堆栈最大空间为 64 KB。堆栈的基地址放在堆栈段寄存器 SS中,由堆栈指针寄存器 SP给出栈顶的偏移地址。堆栈操作是双字节的操作,即每次压入或弹出两个字节。入栈时堆栈指针 SP被减 2,出栈时堆栈指针 SP被加 2。当进行压入操作后堆栈指针回到初值,表明堆栈满;当执行弹出操作后堆栈指针回到初值,表明堆栈空。当栈满时,再压入数据,称为"堆栈溢出"。

8086指令系统中,堆栈压入操作指令为 PUSH,弹出指令为 POP。另外还有一些指令也会对堆栈操作,将在后面进行介绍。

1) 进栈指令 PUSH

格式:PUSH src

src可以是:r16、seg、m16

功能:SP ← SP - 2

(SP+1, SP) ← (src)

该指令先将堆栈指针减 2,然后将源操作数送入栈顶。

2) 出栈指令 POP

格式:POP dest

dest可以为:r16、seg、m16。

功能:(dest) ← (SP+1, SP)

SP ← SP+2

该指令实现栈顶字数据送目标操作数,同时堆栈指针加 2。

【例】

```
MOV     AX, 6688H
PUSH    AX
POP     BX
```

若执行入栈指令前,堆栈段寄存器 SS = 2100H,堆栈指针 SP=0010H;执行入栈指令后,字单元 SS:000EH 的内容 (SS:000EH) =6688H;堆栈指针 SP=000EH,指向新的栈顶。

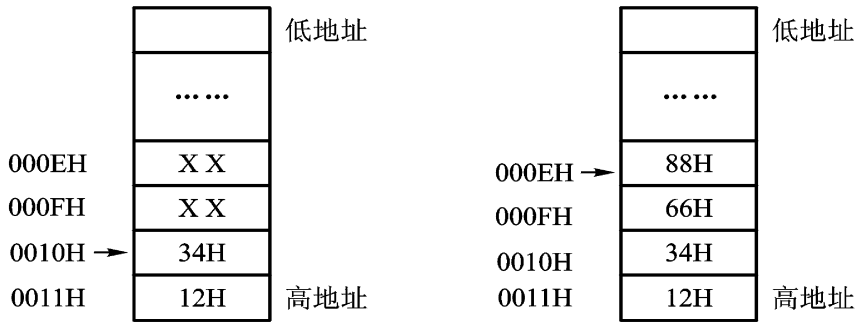
执行过程如图 3.11a、b。

当执行 POP BX后,BX = 6688H;堆栈指针 SP = 0010H,指向新的栈顶。

在调用子程序或转入中断服务程序时,堆栈被用于保存返回地址。为了实现子程序或中断嵌套,必须使用堆栈技术。堆栈还被用于数据的暂存、交换、子程序的参数传递等。

【例题 3.1】　实现 DS、ES的内容交换。

```
PUSH    DS
PUSH    ES
POP     DS
POP     ES
```



(a) 入栈前 SS:SP=2100:0010H　　(b) 入栈后 SS:SP=2100:000EH

图 3.11　入栈操作示意

(3) 数据交换指令 XCHG

格式：XCHG　　dest, src

dest 可以为：reg mem；src可以为：reg mem

功能 源操作数与目标操作数互换，即：(dest) ←→ (src)。

【例】　AX = 4A7BH, (DS:2000H) = 1234H,

指令 XCHG AX, [2000H] 执行后，AX = 1234H, (DS:2000H) = 4A7BH

【例题 3.2】　利用 XCHG,实现两个内存单元 VALUE1和 VALUE2的内容互换

```
MOV     AX,VALUE1       ①
XCHG    AX,VALUE2       ②
MOV     VALUE1,AX       ③
```

由于 XCHG指令不允许同时对两个存储单元进行操作，因而必须借助于一个通用寄存器。①先把一个存储单元中的数据传送到通用寄存器；②再将通用寄存器中的内容与另一个存储单元内容进行交换；③把通用寄存器中的内容回传给第一个存储单元。

(4) 换码指令 XLAT

格式：XLAT或 XLAT变量名或表格首址

源操作数、目标操作数均隐含。

功能：把数据段 DS中偏移地址为 BX ＋AL的内存单元的内容送到 AL中，即：AL← （BX ＋ AL）。

图示：设 BX ＝ 100H，AL ＝ 03H，以 DS:BX为首址的一段内存单元内容如图 3.12。

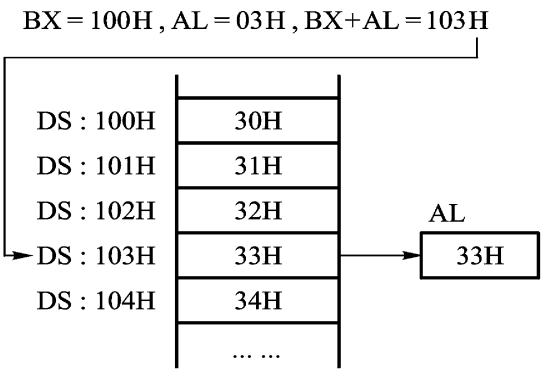


图 3.12 XLAT执行示意图

指令 XLAT执行后，

AL ＝ （DS:BX ＋ AL）

＝ （DS:103H） ＝ 33H ＝‘3’

说明：格式中变量名或表格首址只是为了提高程序的可读性而设置的，BX内容为变量的偏移地址或表格首地址。

换码指令常用于代码转换。把字符扫描码转换成 ASCII码；把数字的二进制编码转换为 7段数码管显示代码等。图示中把数字 3转换为 3的 ASCII码“33H”。

【例题 3.3】 表格 TABLE中定义了十六进制数 0～F的 ASCII码，取出 A的 ASCII码。相关的指令如下：

```
TABLE   DB‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’
        DB ‘8’ ‘9’, ‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘F’      数据定义
        LEA BX, TABLE                                  ;把表格首地址
                                                        送到 BX 寄存
                                                        器中
        MOV AL, 10                                     ;字符序号送到
                                                        AL寄存器中
        XLAT                                           ;AL ＝‘A’＝64H
```

2. 地址目标传送指令 LEA、LDS、LES

汇编中，地址是一种特殊操作数，区别于一般数据操作数，它无符号，长度为

16位。为了突出其地址特点,由专门的指令进行地址传送。

⑴ 取有效地址指令 LEA

格式:LEA　　r16,mem

功能:取内存单元 mem的有效地址,送到 16位寄存器 r16中,即:r16←EA(mem)。

【例】 设 DS = 2100H,BX = 100H,SI = 10H,(DS:110H) = 1234H

指令 LEA　　BX,[BX+SI] 执行后,BX = BX + SI = 110H

⑵ 地址指针装入 DS指令 LDS

格式:LDS　　r16,m32

功能:把内存中的 32位源操作数中低 16位送到指定寄存器 r16中,高 16位送到段寄存器 DS中。即:r16←m32低 16位;DS←m32高 16位。

【例】 执行示意如图 3.13所示。

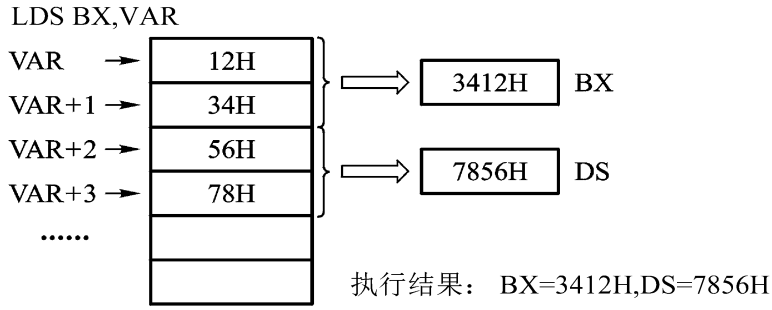


图 3.13　LDS执行示意

⑶ 地址指针装入 ES指令 LES

把上述指令中的 DS换成 ES,即成为 LES指令。

3. 标志传送指令 LAHF、SAHF、PUSHF、POPF

标志寄存器用于记载指令执行引起的状态变化及一些特殊控制位,以此作为控制程序执行的依据。所以,标志寄存器是特殊寄存器,不能像一般数据寄存器那样随意操作,以免其中的值发生变化。

⑴ 取标志指令 LAHF

格式:LAHF

该指令源操作数隐含为标志寄存器低 8位,目标操作数隐含为 AH。

功能:把 16位的标志寄存器低 8位送至寄存器 AH中,即 AH←$(FLAG)_{0\sim7}$。

LAHF执行示意如图 3.14所示。

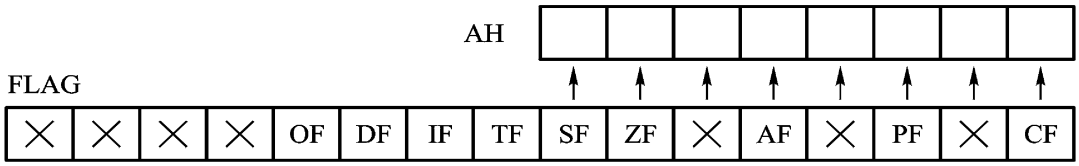


图 3.14 LAHF执行示意

⑵ 置标志指令 SAHF

格式：SAHF

源操作数隐含为 AH，目标操作数隐含为标志寄存器。

功能 把寄存器 AH中内容送至 16位的标志寄存器低 8位，即 (FLAG)$_{0\sim7}$←AH。

此操作是 LAHF的逆操作。

【例题 3.4】 利用 LAHF、SAHF把标志位 CF求反，其他位不变。

```
LAHF                    ;取标志寄存器的低 8位
XOR     AH,01H          ;最低位求反，其他位不变
SAHF                    ;送入标志寄存器的低 8位
```

⑶ 标志入栈指令 PUSHF

格式：PUSHF

源操作数隐含为标志寄存器，目标操作数隐含为堆栈区

功能 标志寄存器入栈。

① SP← SP －2

② (SP + 1, SP) ← FLAG

⑷ 标志弹出指令 POPF

格式：POPF

源操作数隐含为堆栈区，目标操作数隐含为标志寄存器

功能 数据出栈到标志寄存器。

① FLAG ← (SP+1, SP)；

② SP← SP + 2。即：

此操作是 PUSHF的逆操作。

【例题 3.5】 把标志寄存器 TF位清零，其他标志位不变。

```
PUSHF                   标志寄存器入栈
POP     AX              取标志寄存器内容
AND     AX,0FEFFH       ;TF清零，其他位不变
```

```
PUSH    AX              新值入栈
POPF                    送入标志寄存器
```

4. I/O指令 IN、OUT

I/O指令用于 CPU与输入、输出设备端口之间的数据传送。指令有两条：IN和 OUT。IN用于 CPU从外设读数据，OUT用于 CPU对外设写数据。对外设的寻址方式有两种：直接寻址和间接寻址。

(1) 直接寻址

指令中直接给出端口地址，相应的指令格式如下：

输入格式：IN　　AL/AX, port

输出格式：OUT　port, AL/AX

说明：

① port为端口地址，取值范围为：0～255 (0FFH)；

② 操作数类型为字节时，选用 AL寄存器，类型为字时，选用 AX寄存器；

【例题 3.6】　从端口 60H读字节，把它低 4位清零后，再从 60H送出。

```
IN      AL, 60H         ;从 60H读 8位数
AND     AL, 0F0H        ;低 4位清 0
OUT     60H, AL         ;从 60H送出
```

(2) 间接寻址

端口地址存于 DX寄存器中。相应的指令格式如下：

输入格式：IN　　AL/AX, DX

输出格式：OUT　　DX, AL/AX

说明：

① 端口地址，取值范围为：0～65 535 (0FFFFH)；

② 操作数类型为字节时，选用 AL寄存器，类型为字时，选用 AX寄存器；

【例题 3.7】　使用间接寻址方式实现：从端口 60H读字节，把它低 4位清零后，再从 61H送出。

```
MOV     DX, 60H     源端口地址送 DX寄存器中
IN      AL, DX      ;从 60H读 8位数
AND     AL, 0F0H    ;低 4位清 0
INC     DX          ;DX寄存器内容为目标端口地址
OUT     DX, AL      ;从 61H送出数据
```

注意：

(1) 对外设端口操作时，当端口地址在 0～255 (0FFH)范围内，寻址方式可选用直接寻址，也可选用间接寻址；当端口地址大于 255时，只能选用间接寻址，

并且地址寄存器只能用 DX。

(2) 数据寄存器只能用 AL (字节操作)或 AX (字操作)。

3.4.2 算术运算指令

1. 概述

算术运算指令完成的操作可归为三类:一类是辅助运算指令 (CBW, CBD);二类实现算术运算中的加 (ADD, ADC)、减 (SUB, SUBB)、乘 (MUL, IMUL)、除 (DIV, IDIV)、加 1 (INC)、减 1 (DEC)、数据比较 (CMP);三类实施 BCD数算术运算结果调整 (DAA, DAS, AAA, AAS, AAM, AAD)。

操作数分为 4种类型:无符号二进制数、带符号二进制数、无符号压缩 BCD数、无符号非压缩 BCD数。压缩 BCD数可以加、减,其余 3类数据可进行加、减、乘、除 4种运算,如表 3.10。

表 3.10 8086/8088算术运算适用的操作数

操作数类型	加法	减法	乘法	除法
无符号二进制数	ADD、ADC、INC	SUB、SUBB、DEC	MUL	DIV
符号二进制数	ADD、ADC、INC	SUB、SUBB、DEC	IMUL	IDIV
压缩的 BCD码	ADD/ADC AL, src DAA	SUB/SUBB AL, src DAS	—	—
非压缩的 BCD码	ADD/ADC AL, src AAA	SUB/SUBB AL, src AAS	MUL r8/m8 AAM	AAD DIV r8/m8

数据传送类指令,除标志操作指令外,都不影响标志位。与其相反的是算术运算指令基本都对标志位产生影响 (见表 3.11),并且这些影响通常成为程序控制的依据。所以,掌握指令对标志位的影响,是在算术运算指令学习中,特别要注意的,也是学习该部分的一个重点。

表 3.11 算术运算指令功能及其对标志位的影响

	分类	指令助记符	功能简介	标志位					
				OF	SF	ZF	AF	PF	CF
—	扩展	CBW	AL符号扩展到 AH	—	—	—	—	—	—
		CBD	AX符号扩展到 DX	—	—	—	—	—	—

续表

	分类	指令助记符	功能简介	标志位					
				OF	SF	ZF	AF	PF	CF
二	加 1 减 1	INC	增 1	+	+	+	+	+	−
		DEC	减 1	+	+	+	+	+	−
	加	ADD	加法	+	+	+	+	+	+
		ADC	带进位位加	+	+	+	+	+	+
	减	SUB	减	+	+	+	+	+	+
		SBB	带进位位减	+	+	+	+	+	+
		CMP	数据比较	+	+	+	+	+	+
		NEG	求补	+	+	+	+	+	+
	乘	MUL	无符号数乘	+	?	?	?	?	+
		IMUL	符号数乘	+	?	?	?	?	+
	除	DIV	无符数除	?	?	?	?	?	?
		IDIV	符号数除	?	?	?	?	?	?
三	BCD数 调整	DAA	压缩 BCD数加法调整	?	+	+	+	+	+
		DAS	压缩 BCD数减法调整	?	+	+	+	+	+
		AAA	非压缩 BCD数加法调整	?	?	?	+	?	+
		AAS	非压缩 BCD数减法调整	?	?	?	+	?	+
		AAM	非压缩 BCD数乘法调整	?	+	+	?	+	?
		AAD	非压缩 BCD数除法调整	?	+	+	?	+	?

注：+ 表示有影响；− 表示不影响；? 表示没有定义。

2. 扩展指令　CBW、CWD

扩展指令用于将字节扩展为字 (CBW) 或字扩展为双字 (CWD)，指令格式及功能见表 3.12

表 3.12 CBW、CWD指令格式及功能

格式	功 能	说 明
CBW	把寄存器 AL中数据的符号位扩到 AH寄存器中，使字节扩为字。 即：L＜80H时，AH ← 00H AL≥ 80H时，AH ← FFH	源操作数隐含为寄存器 AL，目标操作数隐含为寄存器 AX
CWD	把寄存器 AX中数据的符号位扩到 DX寄存器中，使字扩为双字。 即：X＜8000H时，DX ← 0000H AX≥ 8000H时，DX ← FFFFH	源操作数隐含为寄存器 AX，目标操作数隐含为寄存器 DX，AX

一个数经符号扩展后，就补码表示的数而言，数值大小没有变化。

【例】

```
(1) MOV   AL, 75H
    CBW                    ;执行结果为：AX = 0075H
(2) MOV   AX, 0A085H
    CBW                    ;执行结果为：DX = 0FFFFH, AX = 0A085H
```

3. 加、减

(1) 指令格式见表 3.13。

表 3.13 加、减指令格式一览表

指令	格 式	功 能	说 明
增 1	INC dest	(dest) ← (dest) + 1	
减 1	DEC dest	(dest) ← (dest) − 1	
加	ADD dest, src	(dest) ← (dest) + (src)	
带进位位加	ADC dest, src	(dest) ← (dest) + (src) + CF	用于多字节或多字加法运算
减	SUB dest, src	(dest) ← (dest) − (src)	
带进位位减	SUBB dest, src	(dest) ← (dest) − (src) − CF	用于多字节或多字减法运算
比较	CMP dest, src	(dest) − (src)	影响标志位，不保留结果
求补	NEG dest	(dest) ← 0 − (dest)	

其中，dest可以为 reg, mem, src可以为 reg, mem、imm, dest与 src不能同时为内存操作数。

(2) 举例说明

1) 加 1指令 INC

【例】

① MOV　AL, 0FFH
　INC　AL　　; L = 00H, OF = 0, SF = 0, ZF = 1, AF = 1, PF = 1, CF不变

② INC　BYTE PTR [BX]　数据段中由 BX寻址存储单元的字节内容加 1

2) 减 1指令 DEC

【例】

① MOV　AL, 0
　DEC　AL　　; L = 0FFH, OF = 0, SF = 1, ZF = 0, AF = 1, PF = 1, CF不变

② DEC　NUMB　　;数据段 NUMB单元的内容减 1, 由定义 NUMB的方法来确定这是字节减 1还是字减 1

3) 加法指令 ADD

【例】

① ADD　CX, DI　　;CX ← CX + DI

② ADD　[BP], CL　　; L加堆栈段用 BP作为偏移地址的存储单元的内容, 结果存入存储单元

③ ADD　CL, TEMP　　数据段 TEMP单元的内容加到 CL, 结果存入 CL

④ ADD　BYTE PTR [DI], 3　;3加到数据段中的 DI寻址的存储单元的字节内

4) 带进位位加法指令 ADC

【例】　由 BX－AX组成的 32位数与 DX－CX组成的 32位数相加, 和送 BX－AX,

```
ADD　AX, CX
ADC　BX, DX,
```

操作如图 3.15。

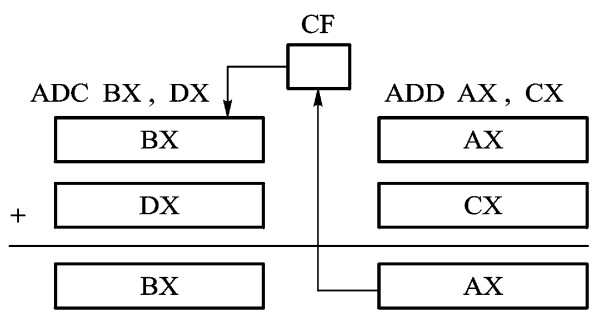


图 3.15　ADC加法示意

5) 减法指令 SUB

【例】

① SUB AX, 0CCCCH ;AX ← AX − 0CCCCH

② SUB [DI], CH ;从由 DI寻址的数据段存储单元的字节内容中减去 CH

6) 带进位位减法指令 SBB

【例】 由 BX −AX组成的 32位数与 SI−DI组成的 32位数相减,差送 BX −AX。

SUB AX, DI

SBB BX, SI

操作如图 3.16所示。

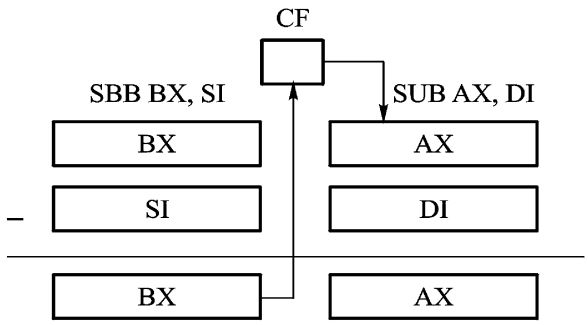


图 3.16 SBB减法示意

加、减运算指令中,不区分符号数与无符号数,即符号数与无符号数使用相同的加、减指令。CPU运算时统一使用补码运算规则。应用中是符号数还是无符号数,由编程人员看问题的视角定,可以从符号数的角度看,也可以从无符号数的角度看。但相同的编码,不同的视角,值有所不同,并且对运算结果的溢出判断标准也不同:符号数,当 OF =1时,溢出;OF =0时,不溢出;无符号数,当 CF=1时,为溢出;CF =0时,为不溢出。见表 3.14例证。

表 3.14 相同的操作,不同视角产生的不同运算结果

序号	被加数 +加数 = CPU运算结果	状态标志位						符号数		无符号数	
		OF	SF	ZF	AF	PF	CF	真值运算	溢出	真值运算	溢出
1	00000100 +00001011 =00001111	0	0	0	0	1	0	4+11=15	不	4+11=15	不
2	00000111 +11111011 =00000010	0	0	0	1	0	1	7+ (−5) =2	不	7+251 =2	是
3	00001001 +01111100 =10000101	1	1	0	1	0	0	9+124=−123	是	9+124 =133	不
4	10000111 +11110101 =01111100	1	0	0	0	0	1	−121+ (−11) =124	是	135+245=124	是

7) 比较指令 CMP

CMP dest, src通过减操作 dest−src,对标志寄存器的状态位产生影响,以此

可判别目标操作数与源操作数之间的大小关系。

由于无符号数与符号数表示规则不同（符号数的最高位为符号位，无符号数各位均为数字位），表示数的范围不同（长度为 8的操作数，视为符号数，表示范围为：－128～＋127；视为无符号数，表示范围为：0～255），使得无符号数与符号数大小判定依据不同。

无符号数相减，不可能有溢出，与大小相关的标志位仅为 ZF与 CF。无符号数 A－B可能情况如表 3.15，从中不难归纳出无符号数的大小判断依据，如表 3.16所示。

表 3.15　无符号数 A－B产生的 ZF、CF

不同情况	标志位	
	ZF	CF
A ＞ B	0	0
A ＝ B	1	0
A ＜ B	0	1

表 3.16　无符号数 A与 B大小判断依据

判断条件	大小关系
ZF ＝ 1	A ＝ B
CF ＝ 1	A＜ B
CF ∨ ZF ＝ 0	A ＞B
CF ＝ 0	A≥ B
CF ∧ ZF ＝ 1	A≤ B

符号数相减，不仅考虑正、负，并且可能存在溢出，所以与大小相关的标志位有 SF、ZF与 OF。符号数 A－B可能情况如表 3.17所示。

表 3.17　符号数 A － B的标志位 SF、ZF、OF

A	B	大、小关系	相关的标志位		
			SF	ZF	OF
A＞0	B＞0	A＞B	0	0	0
A＞0	B＞0	A＜B	1	0	0
A＞0	B＜0	A＞B	0	0	无溢出时，OF＝0；溢出时，OF＝1
A＜0	B＞0	A＜B	1	0	无溢出时，OF＝0；溢出时，OF＝1
A＜0	B＜0	A＜B	1	0	0
A＜0	B＜0	A＞B	0	0	0
A	B	A＝B	0	1	0

符号数，不发生溢出（OF＝0）时，如果 SF＝0，则 A＞B，如果 SF＝1，则 A＜B；发生溢出（OF＝1）时，如果 SF＝1，则 A＞B，如果 SF＝0，则 A＜B。符号数大小判定依据归纳如表 3.18所示。

表 3.18 符号数判断大小依据

判断条件	大小关系
ZF = 1	A = B
SF ⊕ OF = 1	A < B
(SF ⊕ OF) ∨ ZF = 0	A > B
SF ⊕ OF = 0	A ≥ B
(SF ⊕ OF) ∨ ZF = 1	A ≤ B

8) 求补指令 NEG

【例】 MOV DL,0111 1000B ;DL = 120D

NEG DL 结果:DL = 0 − 0111 1000 = 1000 1000B = −120D

【例题 3.8】 将字变量 A1转换为反码和补码,分别存入字变量 A2和 A3中。

分析:取反码要求原数按位求反,可由 NOT指令实现,补码可由反码加 1得到。也可以由 NEG指令获取相反数的补码,由补码减 1得到反码值。所以可用两种方法来实现。

方法一:先取反,再取补的程序段:

```
MOV   AX, A1
NOT   AX          ;取反码
MOV   A2, AX
INC   AX          ;形成补码
MOV   A3, AX
```

方法二:先取补再取反的程序段:

```
MOV   AX, A1
NEG   AX          ;取补码
MOV   A3, AX
DEC   AX, 1       ;形成反码
MOV   A2, AX
```

4. 乘、除指令

乘、除指令一览表,见表 3.19。

表 3.19　乘、除指令

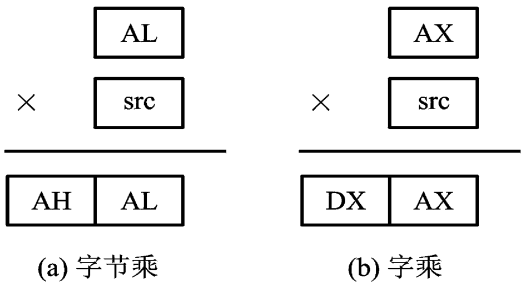
指令	助记符	格式	操　作
无符号数乘	MUL	MUL　src	src为字节：AX ← AL × src src为字：DX AX ← AX × src
符号数乘	IMUL	IMUL　src	
无符号数除	DIV	DIV　src	src为字节：AL← AX ÷ (src) 的商，AH← AX ÷ (src) 的余数 src为字：AX← DX AX ÷ (src) 的商，DX← DX AX ÷ (src) 的余数
符号数除	IDIV	IDIV　src	

src可以为：reg,mem

(1) 乘法指令 MUL、IMUL

8086/8088可实施字节与字节乘、字与字乘，指令中给出乘数，被乘数隐含。乘数可以是寄存器或内存操作数，不能为立即数。

字节乘时，乘积的高 8位存于寄存器 AH中，低 8位存于寄存器 AL中。字乘时，被乘数隐含为寄存器 AX，乘积的高 16位存于寄存器 DX中，低 16位存于寄存器 AX中。如图 3.17a b所示。



(a) 字节乘　　(b) 字乘

图 3.17　乘法示意

【例】

1) IMUL　CL　　;符号数乘法，AL乘以 CL，积在 AX中

2) MUL　WORD PTR [SI]　　;无符号数乘法，AX乘以由 SI寻址的数据段存储单元的字内容，积在 DX－AX中

乘法指令仅影响标志位 OF, CF，对其他标志位无定义。无符号数乘，字节乘时，如果 AH = 0，则 OF = CF = 0；如果 AH ≠ 0，OF = CF = 1。字乘时，如果 DX = 0，则 OF = CF = 0；DX ≠ 0，OF = CF = 1。即符号数乘时，当积的高 8位 (字节乘) 或积的高 16位 (字乘) 是低字节 (字节乘) 或低字 (字乘) 的符号扩展时，OF = CF = 0；否则，OF = CF = 1。

【例】 64H × 0A5H

1) 无符号数乘,即 100 × 165 = 16500D

```
MOV     AL, 64H              ;AL = 64H = 100D
MOV     BL, 0A5H             ;BL = 0A5H = 165D
MUL     BL                   ;AX = 4074H = 16500D, OF = CF = 1
```

2) 符号数乘,即 100 × (−91) = −9100D

```
MOV     AL, 64H              ;AL = 64H = 100D
MOV     BL, 0A5H             ;BL = 0A5H = −91D
IMUL    BL                   ;AX = 0DC74H = −9100D, OF = CF = 1
```

(2) 除法指令 DIV, IDIV

8086/8088可实施除数为字节与除数为字的两种除法。指令中给出除数,被除数隐含。除数可以是寄存器或内存操作数,不能为立即数。

除数为字节乘,被除数必须为 16位,隐含为寄存器 AX,商存于寄存器 AL中,余数存于寄存器 AH中。除数为字时,被除数必须为 32位,隐含为寄存器 DX、AX,商存于寄存器 AX中,余数存于寄存器 DX中。如图 3.18a b所示。

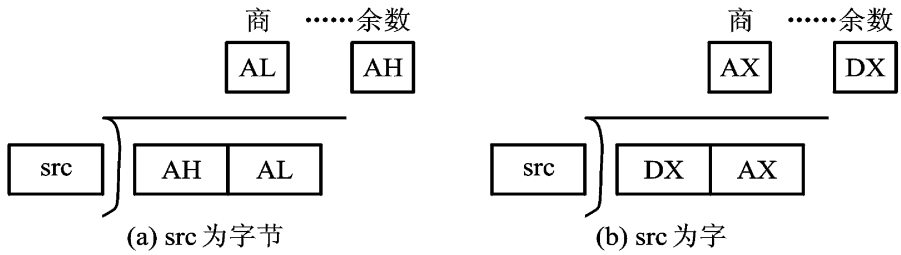


图 3.18 除法示意

【例】

```
1) IDIV     CL                  ; X被 CL除 ;AL中是有符号的商,余数在
                                  AH中
2) DIV      BYTE PTR [BP]       ; X被堆栈段用 BP寻址的存储单元的字
                                  节内容除,无符号的商在 AL中,余数在
                                  AH中
```

除非发生“溢出”,除法对所有标志位均无定义。所谓“溢出”,指除数为字节时,商大于 0FFH或除数为字时,商大于 0FFFFH时。当除法发生溢出时,OF=1,并产生 0型中断 (溢出中断)。

符号数除法中,商的符号遵循除法法则,余数的符号与被除数一致。

【例】　40003H ÷ 8000H

1) 无符号数除,即:262147 ÷ 32768 = 8……3

```
MOV     DX,4
MOV     AX,3                    ;(DXAX) = 40003H = 262147D
MOV     WORD PTR [30H],8000H    ;(DS:30H) = 8000H = 32768D
DIV     WORD PTR [30H]          ;商 AX = 8,余数 DX = 3
```

2) 符号数除,即:262147 ÷ (−32768) = −8……3

```
MOV     DX,4
MOV     AX,3                    ;(DXAX) = 40003H = 262147D
MOV     WORD PTR [30H],8000H    ;(DS:30H) = 8000H = −32768D
DIV     WORD PTR [30H]          ;商 AX = 0FFF8H = −8D,余数 DX
                                 = 3D
```

【例题 3.9】　计算表达式:(V − (X * Y + Z − 540))/X

其中 X、Y、Z、V均为 16位符号数,已分别装入 X、Y、Z、V单元中,要求上式计算结果的商存入 AX,余数存入 DX。

```
MOV     AX, X           ;取被乘数 X
IMUL    Y               ;X* Y,结果在 DX、AX中
MOV     CX, AX          ;将乘积存在 BX、CX中
MOV     BX, DX
MOV     AX, Z           ;取被加数 Z
CWD                     ;将符号扩展后的 Z加到 BX、CX中的乘积上去
ADD     CX, AX
ADC     BX, DX
SUB     CX, 540
SBB     BX, 0           ;从 BX、CX中减去 540
MOV     AX, V
CWD
SUB     AX, CX          ;从符号扩展后的 V中减去 (BX、CX)并
SBB     DX, BX          ;除以 X,商在 AX中,余数在 DX中
IDIV    X
```

5. BCD数调整指令

(1) 调整原理

8421BCD编码用4位二进制数表示一位十进制数，4位二进制数表示范围是0000～1111，对应十进制数0～15，而一位十进制数表示范围是0～9，所以，当某十进制位运算值超过9或发生进位/借位时，运算结果会出错，需要调整。BCD调整指令，用于对BCD数进行加、减、乘、除运算时的调整，使结果正确，并且仍然是BCD编码形式。

下面以具体例子，说明加、减法的基本调整原理。

【例题3.10】 非组合BCD调整

(1) 8＋7＝15

```
  0000 1000        8H
+ 0000 0111        7H
-------------------------------
  0000 1111        1FH，结果错，因为1111＞9，加6调整
+ 0000 0110
-------------------------------
  0001 0101        15H 正确结果
```

(2) 9＋8＝17

```
  0000 1001        9H
+ 0000 1000        8H
-------------------------------
  0001 0001        11H，结果错，因为低位向高位有进位(AF＝1)，需加6调整
+ 0000 0110
-------------------------------
  0001 0111        17H 正确结果
```

可见，运算中，若和大于9或有辅助进位(AF＝1)就需加6调整。同理，减法中，若差大于9或有辅助借位(AF＝1)，就需减6调整。

【例题3.11】 压缩的BCD数调整

(1) 18＋7＝25

```
  0001 1000        18H
+ 0000 0111        7H
-------------------------------
  0001 1111        1FH，结果错，因为1111＞9，需加6调整
+ 0000 0110
-------------------------------
  0010 0101        25H 正确结果
```

(2) 19 + 98 = 27

0001 1001	19H
+ 1001 1000	8H
1011 0001	B1H,结果错。首先,低位向高位有进位 (AF=1),加 6调整
+ 0000 0110	
1011 0111	B7 H 结果错。其交,高位 $(1011)_2$ >9,需加 60调整
+ 0110 0000	
1 0001 0111	117 H 结果对。

可见,在压缩的 BCD 数加法运算时,若低 4位和大于 9或有辅助进位 (AF=1) 就需在低 4位“加 6调整”;若高 4位和大于 9或有进位 (CF=1) 就需在高 4位“加 6调整”。同理,在 BCD数加法运算时,若低 4位差大于 9或有辅助借位 (AF=1) 就需在低 4位“减 6调整”;若高 4位差大于 9或有借位 (CF=1),就需在高 4位“减 6调整”。

(2) 调整指令 (表 3.20)

表 3.20　BCD 数调整指令一览表

指令格式	功能	使用说明
DAA	压缩 BCD数加法调整	紧跟在 ADD或 ADC指令之后
DAS	压缩 BCD数减法十进制调整	紧跟在 SUB或 SUBB指令之后
AAA	加法 ASCII码调整	紧跟在 ADD或 ADC指令之后
AAS	减法 ASCII码调整	紧跟在 SUB或 SUBB指令之后
AAM	乘法 ASCII码调整	紧跟在 MUL指令之后
AAD	除法 ASCII码调整	在 DIV指令之前

操作数隐含为寄存器 AL

1) 压缩 BCD数加法调整指令 DAA

DAA指令目标操作数隐含为寄存器 AL,执行流程如图 3.19所示。

① 如果低 4位大于 9或低 4位向高 4位有进位 (AF=1),低 4位上加 6调整,并且,AF置 1。

② 如果高 4位大于 9或最高位有进位 (CF=1),加 60H 调整,并且 CF置 1。

【例】 39 + 17 = 56

```
MOV     AL,39H
```

```
ADD     AL, 17H          ;AL = 50H, AF=1, CF=0
DAA                      ;AL = 56H
```

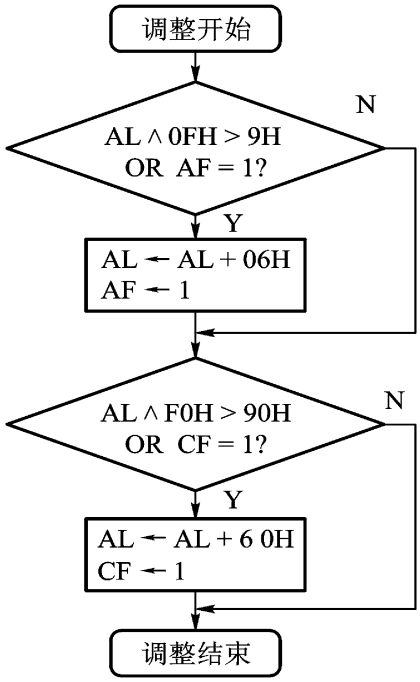


图 3.19 DAA执行流程

2) 压缩 BCD数减法调整指令 DAS

DAS指令目标操作数隐含为寄存器 AL,执行流程如图 3.20所示。

① 如果低 4位大于 9或低 4位向高 4位有借位 (AF=1),低 4位上减 6调整,并且,AF置 1。

② 如果高 4位大于 9或最高位有借位 (CF=1),减 60H调整,并且,CF置 1。

【例】 37 − 19 = 18

```
MOV     AL, 37H
SUB     AL, 19H          ;AL = 1EH, AF=1, CF=0
DAS                      ;AL = 18H, AF=1
```

3) 加法的 ASCII调整指令 AAA

非压缩的 BCD码用 8个二进制位表示一位十进制数,通常只用低 4位,高 4位置零。‘0’～‘9’的 ASCII码为 30H～39H,其低 4位的编码与 BCD编码一

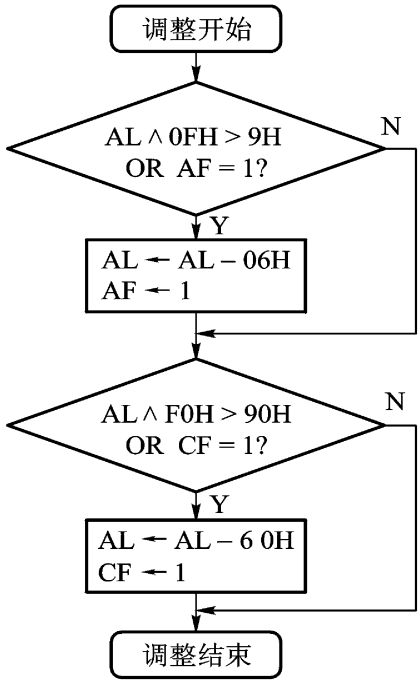


图 3.20　DAS执行流程

致,所以又把非压缩的 BCD码调整称为 ASCII码调整。

AAA调整过程如图 3.21所示。

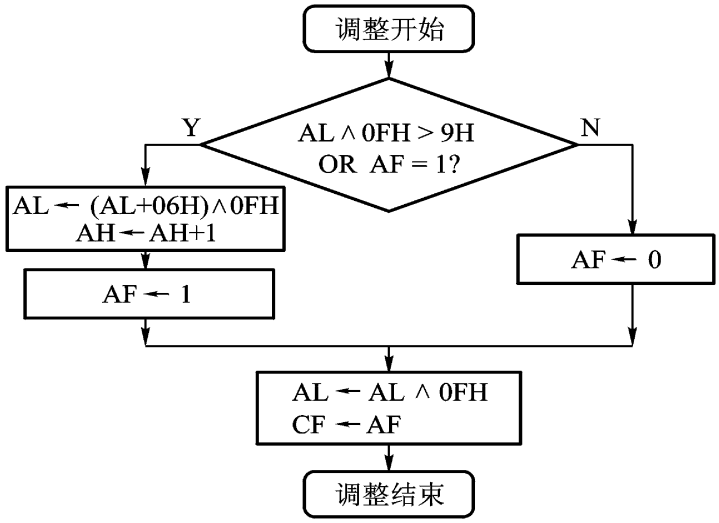


图 3.21　AAA指令执行流程

① 如果 AL中低 4位小于 9且 AF＝0,跳过②。

② 如果 AL中低 4位大于 9或 AF＝1 即低 4位向高 4位有进位),加 6调

整,并且,AF置 1。

③ 清除 AL寄存器的高 4位。

④ AF值送 CF。

【例】 7+9=16

```
MOV     AL, '7'          ;AL = 37H
ADD     AL, '9'          ;AL = 70H, AF = 1, CF = 0
AAA                      ;AL = 06H, CF = AF = 1
```

4) 减法的 ASCII调整指令 AAS

AAS调整过程如图 3.22所示。

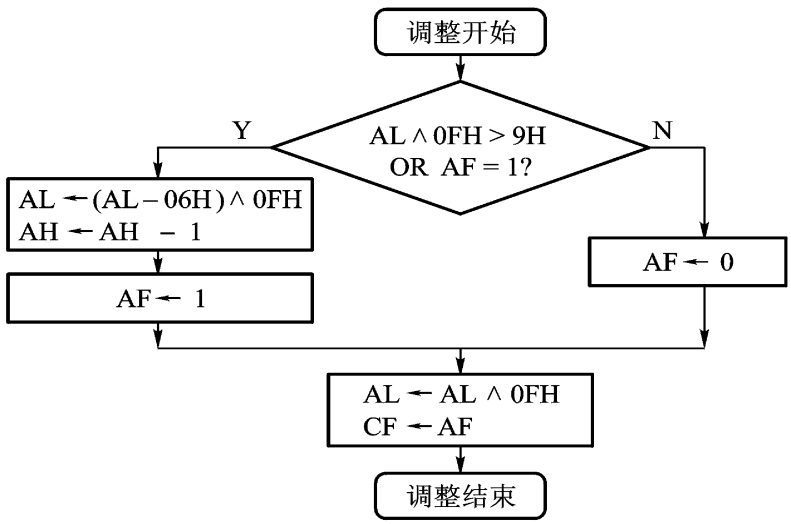


图 3.22 AAS指令执行流程

① 如果 AL中低 4位小于 9且 AF=0,跳过②。

② 如果 AL中低 4位大于 9或 AF=1 即低 4位向高 4位有借位),减 6调整,并且,AF置 1。

③ 清除 AL寄存器的高 4位。

④ AF值送 CF。

【例】 17 − 9= 8

```
MOV     AL, 7H           ;AL = 07H
MOV     AH, 1H           ;AH = 1H
SUB     AL, 9H           ;AL = 0FEH, AF = 1, CF = 1
AAS                      ;AL = 8 H, CF = AF = 1, AH = 0
```

5) 乘法 ASCII码调整指令 AAM

两个非压缩的 BCD码相乘，乘法指令用 MUL，乘积存于寄存器 AX，AAM 用于 MUL之后，把 AL中的乘积调整成非压缩的 BCD码，结果存于寄存器 AX中。指令操作流程如图 3.23所示。

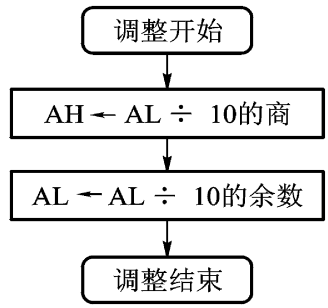


图 3.23　AAA指令执行流程

把 AL寄存器的内容除以 10，商放在 AH寄存器中，余数保存在 AL寄存器中。

【例】　9 × 3 = 27

```
MOV     AL, 9H          ;AL = 0000 1001B
MOV     BL, 3H          ;BL = 0000 0011B
MUL     BL              ;AH = 0, AL = 0001 1011B = 27D
AAM                     ;AH = 02H, AL = 07H (因为 27÷10=2……7)
```

6) 除法 ASCII码调整指令 AAD

被除数是两位十进制数，存于寄存器 AX中，AH中为十位上的数，AL中为个位上的数，除数是 1位十进制数。AAD用于 DIV之前，把 BCD码表示的被除数转为与真值对应的机器数。AAD操作流程如图 3.24。

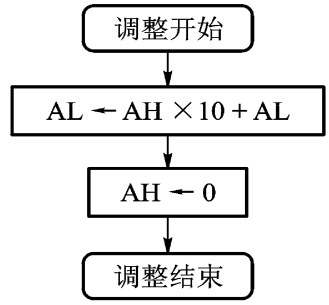


图 3.24　AAD指令执行流程

【例】　15 ÷ 3 = 5

```
MOV     AH, 1
MOV     AL, 5
MOV     BL, 3
AAD                     ;AL = 1×10+5 = 15D = 0000 1111B
DIV     BL              ;AL = 05H, AH = 0 (因为 15÷3 =5……0)
```

3.4.3　位操作指令

位操作指令分为 逻辑运算指令、移位指令、循环移位指令。

1. 逻辑运算指令

逻辑运算指令包括逻辑非 (NOT)、逻辑与 (AND)、逻辑测试 (TEST)、逻辑或 (OR) 和逻辑异或 (XOR) 指令。这些指令的操作数可以是 8位、16位，运算按位进行。对操作数的要求与 MOV指令相同。指令格式如表 3.21。

表 3.21 逻辑运算指令一览表

指令名	格 式	功 能	标志位
逻辑非	NOT dest	(dest) ← $\overline{(dest)}$	① CF、OF清零 ②影响 SF、ZF、PF ③ AF不定
逻辑与	AND dest, src	(dest) ← (dest) ∧ (src)	
测试	TEST dest, src	(dest) ∧ (src)	
逻辑或	OR dest, src	(dest) ← (dest) ∨ (src)	
逻辑异或	XOR dest, src	(dest) ← (dest) ⊕ (src)	

测试指令 TEST,执行相与操作,以便影响标志位,但不保留结果。

【例】

(1) AND AL, BL ; AL← AL∧ BL

(2) XOR AX, [DI] ; X异或数据段存储单元的字内容,结果存入 AX

(3) OR SP, 990DH ; (SP) ← (SP) ∨ 990DH

(4) NOT BYTE PTR [BX] ;数据段存储单元的字节内容求反

(5) TEST AH, 4 ; AH ∧ 4, AH不变,只影响标志位

运算定义如表 3.22。

表 3.22 逻辑运算符定义

X Y	X AND Y	X OR Y	X XOR Y	NOT X
0 0	0	0	0	1
0 1	0	1	1	1
1 0	0	1	1	0
1 1	1	1	0	0

由表 3.22知:

(1) 任何数与 0相与,得 0;与 1相与,得原值;

(2) 任何数与 0相或,得原值;与 1相或,得 1;

(3) 任何数与 0异或,得原值;与 1异或,得相反数;

【例题 3.12】 把 BCD数 8H,变成 ASCII码‘8’

```
MOV     AL, 8H
OR      AL, 30H          ;AL = 38H = ‘8’
```

【例题 3.13】 从端口 61H读取一个字节,D_1 位求反后,从 61H送出

```
IN      AL,61H
XOR     AL,00000010B
OUT     61H,AL
```

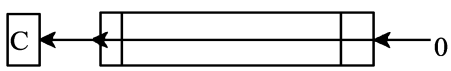
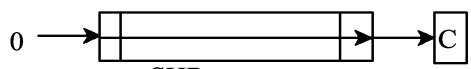
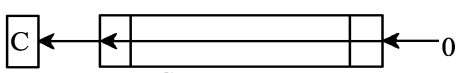
【例题 3.14】　寄存器 AX清零

```
XOR     AX,AX
```

2. 移位指令

移位指令包括:逻辑左移 SHL、逻辑右移 SHR、算术右移 SAR、算术左移 SAL。指令格式及功能见表 3.23。

表 3.23　移位指令一览表

名称	格　式	功能图示及说明	说　明
逻辑左移	SHL dest,1/CL	 目标操作数左移 CNT次,最低位补0,最高位移至标志位 CF中	① CNT 代表移动次数; ② CNT ＞ 1时,必须由寄存器 CL说明; ③CF、ZF、SF、PF由运算结果定; ④ CNT ＝ 1时,若移位后符号位发生变化,则标志位 OF＝1,否则 OF＝0; ⑤CNT ＞ 1时,对 OF无定义
逻辑右移	SHR dest,1/CL	 目标操作数右移 CNT次,最低位移至标志位 CF中,最高位补 0	
算术左移	SAL dest,1/CL	 目标操作数左移 CNT次,最低位补0,最高位移至标志位 CF中	
算术右移	SAR dest,1/CL	 目标操作数右移 CNT次,最低位移至标志位 CF中,最高位不变	

【例】　分别给出下列移位指令执行结果。设 AL ＝ 0B4H ＝ 10110100B, CF ＝ 1, CL ＝ 4。

```
(1)  SAL     AL,1          ;AL = 01101000B, CF = 1, OF = 1
(2)  SAR     AL,1          ;AL = 11011010B, CF = 0, OF = 0
```

```
(3) SHL    AL,1                ;AL = 01101000B, CF = 1, OF = 1
(4) SHR    AL,CL               ;AL = 00001011B, CF = 0, OF无定义
(5) SAL    DATA1,CL            ;数据段中的DATA1按CL的内容算术
                               左移数个位
(6) SAR    WORD PTR [BP],1     ;堆栈段由BP寻址的存储单元的字内容
                               算术右移1位
```

这组指令除了可以实现基本的移位操作外,还可以用于实现数倍增(左移)或倍减(右移),使用这种方法比直接使用乘、除法效率要高得多。用逻辑移位指令可实现无符号数的乘、除,算术移位指令可实现符号数的乘、除。只要不超出数的表示范围,SHL或SAL执行后,数为原数的2^{CNT}倍,SHR或SAR执行后,数为原数的$1/2^{CNT}$。

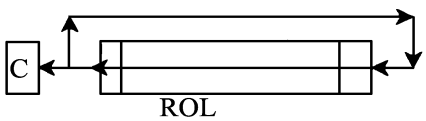
【例题3.15】 设无符号数X在寄存器AL中,用移位指令实现X × 10的运算

```
MOV    AH,0
SAL    AX,1          ;计算 2X
MOV    BX,AX
MOV    CL,2
SAL    AX,CL         ;计算 8X
ADD    AX,BX         ;计算 2X+8X = 10X
```

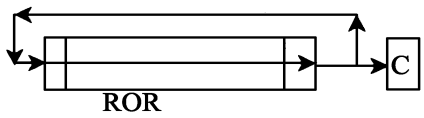
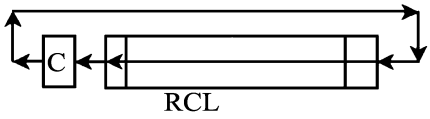
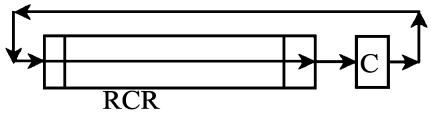
3. 循环移位指令

循环移位指令包括以下4条:不带进位的循环左移(ROL)、不带进位的循环右移(ROR)、带进位的循环左移(RCL)、带进位的循环右移(RCR)。指令格式及功能见表3.24。

表3.24 循环指令一览表

名称	格 式	功能图示及说明	说 明
不带进位循环左移	ROL dest,1/CL	 目标操作数循环左移CNT次,最高位移至最低位的同时移至标志位CF中	

续表

名称	格　式	功能图示及说明	说　明
不带进位循环右移	ROR dest,1/CL	 目标操作数循环右移 CNT次,最低位移至最高位的同时移至标志位 CF中	① CNT 代表移动次数； ② CNT ＞ 1时,必须由寄存器 CL说明； ③ CF由运算结果定； ④ 不影响 SF、ZF、AF、PF； ⑤ 对 OF 的影响同 SHL
带进位循环左移	RCL dest,1/CL	 目标操作数及标志位 CF一起循环左移 CNT次,最高位移至标志位中,标志位移至最低位	
带进位循环右移	RCR dest,1/CL	 目标操作数及标志位 CF一起循环右移 CNT次,最低位移至标志位中,标志位移至标最高位	

【例】

分别写出下列循环移位指令的执行结果,设 AL = 0101 0100B, CF = 1, CL = 4。

```
(1) ROL    AL,1        ;AL = 1010 1000B, CF = 0, OF = 1
(2) ROR    AL,1        ;AL = 0010 1010B, CF = 0, OF = 0
(3) RCL    AL,1        ;AL = 1010 1001B, CF = 0, OF = 1
(4) RCR    AL,CL       ;AL = 1001 0101B, CF = 0, OF无定义
```

3.4.4　串操作指令

所谓“串”指一组数据,所以串操作指令的操作对象不是一个字节或一个字,而是内存中地址连续的一组字节或一组字。缺省情况下,原串存于数据段中,目标串存于附加段中。在每次基本操作后,能够自动修改源及目标地址为下一次操作做好准备。串操作指令前可以加上重复前缀,此时,基本操作在满足条件的情况下得到重复,直至完成预设次数。

1. 串操作指令

串操作指令共有5条,串传送指令MOVS、串装入指令LODS、串送存指令STOS、串比较指令CMPS、串扫描指令SCAS,具体格式及功能见表3.25

表3.25 串操作指令一览表

指令名	指令格式	无前缀时基本操作	可用前缀	说明
串传送	MOVS dest, src MOVSB MOVSW	(ES:DI) ← (DS:SI) SI← SI±1/2 DI← DI±1/2	REP	①格式中的dest, src仅为了增加程序的可读性; ②字节操作时,地址调整量是1,字操作时,地址调整量是2; ③地址是增或减由标志位DF定: DF=0,地址增, DF=1,地址减; ④寻址方式规定为寄存器间接寻址,源操作数隐含为数据段,偏移地址由寄存器SI指明,允许段跨越;目标操作数隐含为附加段,偏移地址由寄存器DI指明,不允许段跨越
串装入	LODS src LODSB LODSW	AL← (DS:SI) SI← SI±1/2	REP	
串送存	STOS dest STOSB STOW	(ES:DI) ← AL DI← DI±1/2	REP	
串比较	CMPS src, dest CMPSB CMPSW	(DS:SI) − (ES:DI) SI← SI±1/2 DI← DI±1/2	REPE/REPZ REPNE/REPNZ	
串扫描	SCAS dest SCASB SCASW	AL/AX − (ES:DI) DI← DI±1/2	REPE/REPZ REPNE/REPNZ	

指令STD、CLD用于设置方向标志。STD使DF为1,CLD使DF为0。

2. 指令前缀

串操作指令的基本操作可以用于一个数据被执行一次,也可以用于一组数据重复执行,因此在指令中需要指明该指令的基本操作是否被重复、重复的条件是什么及重复的次数是多少。串操作的指令前缀用于说明前两个问题,重复次数隐含在寄存器CX中。前缀有三种:无条件重复REP、相等重复REPE或REPZ、不相等重复REPNE或REPNZ。

(1) 无条件重复前缀REP

当在串操作指令前加上前缀REP后,指令执行过程如图3.25,即:

1) 若CX=0,则结束该指令,执行后续指令;否则,

2) CX← CX−1;

3) 执行前缀后的串操作指令的基本操作（包括地址修改）；

重复第 1) ～3)。

MOVS、LODS、STOS指令前可用前缀 REP。

(2) 相等重复前缀 REPE或 REPZ

当在串操作指令前加上前缀 REPZ或 REPE后，指令执行过程如图 3.26，即：

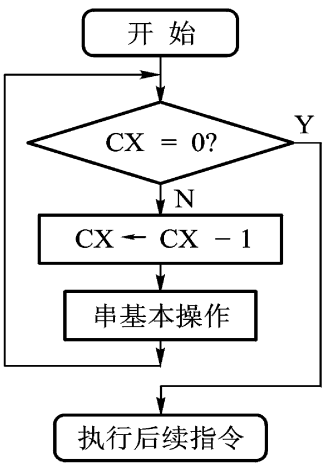


图 3.25　REP执行指令流程

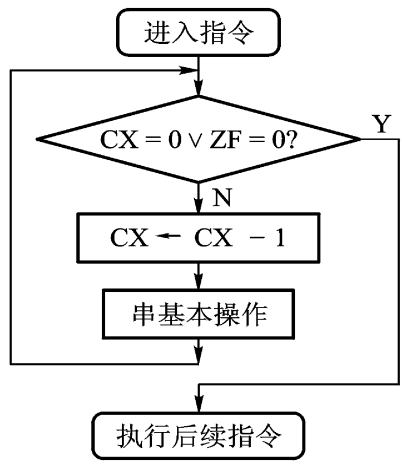


图 3.26　REPNE/REPNZ 指令执行流程

1) 若 CX = 0或 ZF=0，则结束该指令，执行后续指令；否则，

2) CX ← CX −1；

3) 执行前缀后的串操作指令的基本操作（包括地址修改）；

重复第 1) ～3)。

指令 CMPS、SCAS前可用前缀 REPE或 REPZ。

(3) 不相等重复前缀 REPNE或 REPNZ

当在串操作指令前加上前缀 REPNZ或 REPNE后，指令执行过程如图 3.27，即：

1) 若 CX = 0或 ZF=1，则结束该指令，执行后续指令；否则，

2) CX ← CX − 1；

3) 执行前缀后的串操作指令的基本操作（包括地址修改）；

重复第 1) ～3)。

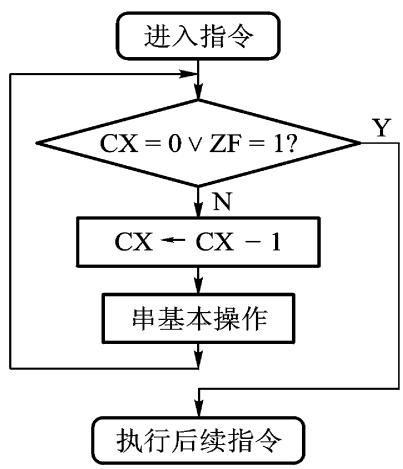


图 3.27　REPNE/REPNZ 指令执行流程

指令 CMPS、SCAS前可用前缀 REPNE或 REPNZ。

注意：有前缀的串操作中，CX －1不影响标志位

3. 加前缀的串操作指令

(1) 与 REP相配合的 MOVS、STOS、LODS

1) 重复串传送

格式：REP MOVS dest, src

或 REP MOVSB

或 REP MOVSW

操作：

① 若 CX = 0,则结束该指令，执行后续指令；否则，

② CX← CX －1;

③ (ES:DI) ← (DS:SI)；

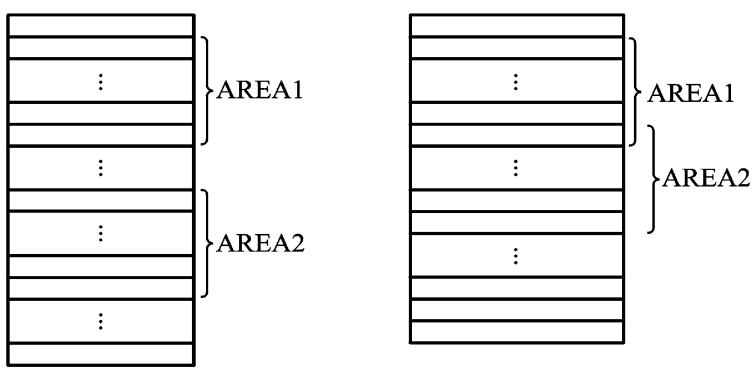
④ SI← SI ± 1／2，DI← DI ± 1／2;

重复第①～④。

说明：MOVSB时，地址增/减量为 1,MOVSW 时，地址增/减量为 2。

【例题 3.16】 把自 AREA1开始的 100个字传送到 AREA2开始的区域中。

解：源、目标区，可能没有重叠，也可能有重叠，如图 3.28所示。



(a) 源、目标区域不重叠 (b) 源、目标区域有重叠

图 3.28 源数据区，目标数据区两种情况

① 源、目标区，没有重叠，传送方向，地址增或地址减均可，以地址增为例

```
MOV     AX, SEG AREA1
MOV     DS, AX                  ;源区段地址送段寄存器 DS
MOV     AX, SEG AREA2
MOV     ES, AX                  ;目标区段地址送段寄存器 ES
```

```
LEA     SI, AREA1                ;源区首字的偏移地址送寄存器 SI
LEA     DI, AREA2                ;目标区首字的偏移地址送寄存器 DI
MOV     CX, 100                  ;串长送寄存器 CX
CLD                              ;DF = 0,地址增
REP     MOVSW                    ;串传送
```

② 源、目标区有重叠(如图 b),选择地址减

```
MOV     CX, 100                  ;串长送寄存器 CX
MOV     AX, SEG AREA1
MOV     DS, AX                   ;源区段地址送段寄存器 DS
MOV     AX, SEG AREA2
MOV     ES, AX                   ;目标区段地址送段寄存器 ES
LEA     SI, AREA1
ADD     SI, CX                   ;源区末字的偏移地址送寄存器 SI
LEA     DI, AREA2
ADD     DI, CX                   ;目标区末字的偏移地址送寄存器 DI
STD                              ;DF = 1,地址减
REP     MOVSW                    ;串传送
```

2) 重复串送存

格式:REP　STOS dest

或　REP　STOSB

或　REP　STOSW

操作:

① 若 CX = 0,则结束该指令,执行后续指令;否则,

② CX ← CX − 1;

③ (ES:DI) ← AL;

④ DI ← DI ± 1/2;

重复第①～④。

重复串送存,用于在一段地址连续的内存单元中置相同数。

[例题 3.17]　在内存 DS:2100～DS:2110中,存入符号'$'

```
MOV     ES, DS           ;目标段段地址送段寄存器 ES
MOV     DI, 2100         ;目标段首字节偏移地址送寄存器 DI
MOV     CX, 10H          ;串长送寄存器 CX
```

```
CLD                    ;设置方向增
REP     STOS           ;重复串送存
```

3) 重复串装入

格式：REP LODS src

或 REP LODSB

或 REP LODSW

操作：

① 若 CX = 0,则结束该指令,执行后续指令;否则,

② CX← CX －1;

③ AL← (DS:SI);

④ SI← SI ±1/2;

重复第①～④。

从操作可见,串装入的重复没有多大意义,最终取到的数是最后一个送入寄存器 AL中的值。

(2) 与 REPE或 REPZ联合使用的 CMPS、SCAS

1) 相等重复串比较

格式：REPE/REPZ CMPS dest,src

或 REPE/REPZ MOVSB

或 REPE/REPZ MOVSW

操作：

① 若 CX = 0或 ZF=0,则结束该指令,执行后续指令;否则,

② CX← CX －1;

③ (DS:SI) － (ES:DI),影响标志位;

④ SI← SI ±1/2,DI← DI ±1/2;

重复第①～④。

该指令把源串与目标串的数据逐个比较,相等继续比下去。退出该指令时,有两种情况：(1) 比较完毕退出,此时 CX=0;(2) 不相等退出,此时 ZF=0,表示两串不相等。所以,该指令常用于判断两串是否相等。方法是检测退出后标志位 ZF:ZF=1,两串相等;ZF=0,两串不相等。从以上分析可知,不能以是否比较完毕来判断两串是否相等。

【例题 3.18】 比较串 STR1、STR2,若相等,给寄存器 AX置全 1,否则 AX中为不相等处地址。

数据定义

```
STR1    DB      'ABCDEFGHIJK'      ;串 1,作为源串
STR2    DB      'ABCDFGKJ'         ;串 2,作为目标串
N       DW      20                 ;串长
;功能代码
        MOV     AX, SEG STR1
        MOV     DS, AX             ;源串段地址送 DS
        MOV     AX, SEG STR2
        MOV     ES, AX             ;目标串段地址送 ES
        LEA     SI, STR1           ;源串偏移地址送 SI
        LEA     DI, STR2           ;目标串偏移地址送 DI
        MOV     CX, N              ;重复次数送 CX
        CLD                        ;地址增
        REPE    CMPSB              ;相等继续比较,不相等退出
        JZ      EQUAL              ;退出时,若 ZF = 1,表示两串相等
        DEC     SI                 ;不相等,指针退 1,指向不相等处
        MOV     AX, SI             ;保存不相等处地址
        JMP     FINISH
EQUL:   MOV     AX, 0FFH           ;相等, AX为 0FFH
FINISH: HLT
```

2) 相等重复串扫描

格式 :REPE/REPZ　SCAS　dest

或　REPE/REPZ　SCASB

或　REPE/REPZ　SCASW

操作：

① 若 CX = 0或 ZF=0,则结束该指令,执行后续指令 ;否则,

② CX← CX - 1;

③ AL/AX - (DS:SI),影响标志位 ;

④ DI← DI ± 1/2;

重复第① ~④。

该指令把目标串的数据逐个与 AL中数据比较,相等继续比下去。退出时,有两种情况：(1) 碰到与 AL中数据不相等的数据,此时, ZF=0; (2) 目标串中所有数据相等,且等于 AL中的数据,此时, CX=0, ZF=1。所以,该指令可用于判断串中所有字符是否相等,实际应用价值不大。

(3) 与 REPNE或 REPNZ联合使用的 CMPS、SCAS

1) 不相等重复串比较

格式：REPNE/REPNZ CMPS dest, src

或 REPNE/REPNZ MOVSB

或 REPNE/REPNZ MOVSW

操作：

① 若 CX = 0或 ZF=1,则结束该指令,执行后续指令;否则,

② CX← CX - 1;

③ (DS:SI) - (ES:DI),影响标志位;

④ SI← SI±1/2, DI← DI±1/2;

重复第①~④。

该指令把源串与目标串的数据逐个比较,不相等继续比下去。退出该指令时,有两种情况：(1) 比较完退出,此时 CX=0,两串对应位置上字符均不相等;(2) 相等退出,此时 ZF=1,表示两串对应位置上出现相同数据。可见,该指令仅可用于说明两串对应位置上是否有相等数据,并且一旦发现,停止操作。该指令实际应用价值不大。

2) 不相等重复串扫描

格式：REPNE/REPNZ SCAS dest

或 REPNE/REPNZ SCASB

或 REPNE/REPNZ SCASW

操作：

① 若 CX = 0或 ZF=1,则结束该指令,执行后续指令;否则,

② CX← CX - 1;

③ AL/AX - (DS:SI),影响标志位;

④ DI← DI±1/2;

重复第①~④。

该指令把目标串的数据逐个与 AL中数据比较,不相等继续比下去。退出该指令时,有两种情况：(1) 碰到与 AL相等的数据,此时,ZF=1;(2) 目标串中不存在与 AL相等的数据,此时,CX=0。本指令适用于在串中寻找某特定字节或字,在该字节或字第一次出现时,停止指令操作,但数据指针 DI指在该数据的前一个数据处。

【例题 3.19】 在串 STR中寻找字符串 'AB',若有,把其出现的位置存于寄存器 BX中,否则,BX置 0FFH。

数据定义

```
STR     DW    'ABCDEFGHIJK'
KEY     DW    'AB'
N       EQU   ($ -STR)/2        ;串长
;功能代码
        CLD                     ;地址增
        MOV   AX, SEG STR       ;目标串段地址送段寄存器 ES
        MOV   ES, AX
        LEA   DI, STR           ;目标串首字的偏移地址送寄存器 DI
        MOV   AX, KEY           ;被寻找字送寄存器 AX
        MOV   CX, N             ;串长送寄存器 CX
        REPNE SCASB             ;不相等继续寻找,一旦相等退出
        JNZ   NEQUAL            ;退出时,若 ZF = 0,表示未找到
        SUB   DI, 2             ;相等,数据指针退 2,指向相等处
        MOV   BX, DI            ;保存关键字出现地址
        JMP   FINISH
NEQUL:  MOV   BX, 0FFH          ;未找到,AX为 0FFH
    ……
FINISH:…
```

3.4.5 控制转移指令

控制转移指令用来改变程序的执行顺序,执行转移就是将目的地址传送给码段寄存器 CS与指令指针寄存器 IP。如果跳转目的地与被转移点在同一代码段,称为“段内转移”,此时,只需指明目标地址的有效地址 (16位)。如果跳转目的地与被转移点不在同一代码段,称为“段间转移”,此时,需知道目标地址的段地址 (16位)及有效地址 (16位)。

控制转移指令的寻址方式分为‘直接寻址”、“间接寻址”两种。如果指令中直接给出目标地址,如地址标号或立即数 (偏移量,目标与源之间的偏移距离),称为‘直接寻址”;如果指令中,给出目标地址存放地的地址,如寄存器或内存地址,称为‘间接寻址”。

控制转移指令包括 转移指令、循环控制指令、过程调用指令和中断指令等4组。

1.转移指令

转移指令将正在被执行的指令集的执行点从一处转到另一处。源地址与目

标地址的距离称为跳转“偏移量”,偏移量是符号数。当用一个字节表示偏移量时,即源地址与目标地址之距在 +127～−128字节之内,称为“短(short)转移”,当用一个字表示偏移量时,即源地址与目标地址之距在 +32 767～−32 768字节之内,称为“近(near)转移”。

转移指令又可分为两类:无条件转移指令与条件转移指令。

(1) 无条件转移指令 JMP

格式:JMP dest

dest可以是标号、立即数、寄存器、内存操作数。

功能:跳转到 dest所指目标处。

不同寻址方式,目标地址的计算见表 3.26。

表 3.26 无条件跳转不同寻址方式下地址计算

类型	寻址方式	操作数	目标地址计算		示例	说明
段内转移	直接	地址符号	IP← IP + 偏移量 CS不变		JMP SHORT NEXT	① SHORT表示短跳; ② FAR表示段间跳; ③ DWORD PTR表明内存操作数属性为双字
		立即数(偏移量)			JMP 2100H	
	间接	寄存器	IP← 寄存器	CS不变	JMP BX	
		存储器	IP← (存储器)		JMP [BX]	
段间转移	直接	地址符号	IP← 目标偏移地址/立即数低 16位 CS← 目标段地址/立即数高 16位		JMP FAR PTR NEXT	
		立即数(32位)			JMP 2100:0100H	
	间接	内存(双字)	IP← (EA+1, EA) CS← (EA+3, EA+2)		JMP DWORD PTR [BX]	

[例题 3.20] 短转移

```
0000    33 BD              XOR     BX, BX
0002    D8 0001 START:     MOV     AX, 1
0005    03 C3              ADD     AX, BX
0007    EB 17              JMP     SHORT NEXT
0009    …
  ……
0020    8B D8    NEXT:     MOV     BX, AX
0022    EB DE              JMP     START
```

程序指出了短转移指令怎样通过控制从程序的一部分转移到另一部分,也

说明了和转移指令一起的标号的使用。用下一条指令的地址（0009H）加第一条转移指令的符号扩展位移量（0017H），就得到 NEXT位于 0017H＋0009H 的地址处，即 0020H 处。第二条转移指令（JMP START）也按短转移指令汇编，因为已知地址 START。当转移指令引用标号时，标号等效于地址。

转移指令使用实际的十六进制地址是极少的，但是汇编程序支持使用 $ ±位移量，即相对于指令指针的寻址。JMP $ ＋2就相对 JMP指令向后越过两个存储单元。

【例题 3.21】　近转移

```
        XOR     BX, BX
START:  MOV     AX, 1
        ADD     AX, BX
        JMP     NEXT
NEXT:   MOV     BX, AX
        JMP     START
```

例题 3.21给出了与例题 3.20相同的基本程序，只是转移的距离大了些。

⑵ 条件转移指令

格式：Jcc　　short－label

cc代表跳转条件，short－label表明该指令只能实现段内短转移，参数形式通常为符号地址。

根据不同条件，条件转移指令共有 19条，指令助记符及相应的跳转条件见表 3.27。

表 3.27　条件跳转指令

特征	助记符	转移条件	说明	
单标志位	JAE／JNB	CF＝0	无符号数	大于等于 或 不小于 转移
	JB／JANE	CF＝1		小于 或 不大于等于
	JC	CF＝1	有进位或借位 转移	
	JNC	CF＝0	无进位／借位 转移	
	JZ	ZF＝1	等于转移	
	JNZ	ZF＝0	不等于 转移	
	JNO	OF＝0	无溢出 转移	
	JO	OF＝1	有溢出 转移	
	JNP／JPO	PF＝0	1的个数为奇数 转移	

续表

特征	助记符	转移条件	说明	
多标志位	JP/JPE	PF=1	1的个数为偶数 转移	
	JNS	SF=0	正数 转移	
	JS	SF=1	负数 转移	
	JA/JNBE	CF∨ZF=0	无符号数	大于 或 不小于等于 转移
	JBE/JNA	CF∨ZF=1		小于等于 或 不大于 转移
	JGE/JNL	SF⊕OF=0	符号数	大于等于 或 不小于 转移
	JL/JNGE	SF⊕OF=1		小于 或 不大于等于 转移
	JG/JNLE	(SF⊕OF)∨ZF=0		大于 或 不小于等于 转移
	JLE/JNG	(SF⊕OF)∨ZF=1		小于等于 或 不大于 转移
CX	JCXZ	CX=0		

单标志位条件转移指令简单明了,它们只测试一个标志位,多标志位转移指令测试多个标志位。相对大小的比较有两套条件转移指令。比较有符号数时使用术语大于或小于,用 JG、JL、JGL、JLE、JE或 JNE指令。比较无符号数时使用高于或低于,用 JA、JB、JAE、JBE、JE及 JNE指令。无符号数集合中,FFH (255)高于 00H,而有符号数 FFH (-1)小于 00H。

【例题 3.22】 比较两个字属性的符号数 X,Y的大小,如果 X>Y,AL为 1,如果 X=Y,AL为 0,如果 X<Y,AL为 0FFH。

解:设 X、Y为内存变量,功能实现主要代码如下:

```
    MOV     AX,X
    CMP     AX,Y
    JLE     LE
    MOV     AL,1            如果 X大于 Y,AL=1
    JMP     DONE
LE: L       L
    MOV     AL,0FFH         如果 X等于 Y,AL=0FFH
    JMP     DOWN
L:  OV      AL,0            如果 X小于 Y,AL=0
    DONE:HLT
```

2. 循环控制指令

循环指程序段在一定条件下重复执行。循环指令提供了程序段循环的控制

及手段。这些指令都用 CX寄存器作为循环次数计算器,表示某程序段最大循环次数,且循环体每执行一次,CX被减去 1。8088/8086 CPU规定被循环的程序段必须在同一段内,且长度不能大于 256字节。

循环控制指令有 3条:循环指令 LOOP、相等循环指令 LOOPE/LOOPZ、不相等循环指令 LOOPNE/LOOPNZ,其格式及功能如表 3.28。

表 3.28　循环指令

名　　称	格　　式	操　　作
循环指令	LOOP　Short-label	① CX← CX －1 ② 如果 CX ＝0,结束循环,执行后续语句。否则: ③ 转移到标号处,循环体被重复
相等循环指令	LOOPZ/LOOPE Short-label	① CX← CX －1 ② 如果 CX ＝0或 ZF＝0,结束循环,执行后续语句。否则: ③ 转移到标号处,循环体被重复
不相等循环指令	LOOPNZ/LOONE Short-label	① CX← CX －1 ② 如果 CX ＝0或 ZF＝1,结束循环,执行后续语句。否则: ③ 转移到标号处,循环体被重复

Short-label通常为循环体起始位置处的标号。

【例题 3.23】　有一首地址为 Array的长度为 M字数组,试编写实现下列功能的代码:统计出数组中 0元素的个数,并存入变量 total中。

```
        MOV     CX,M            ;数组长度存入循环计数器 CX
        MOV     total,0         ;计数初始值 0送计数变量
        MOV     SI,0            ;采用寄存器相对寻址,初始偏移量送
                                 寄存器 SI
AGAIN:  MOV     AX,Array[SI]    ;取数
        CMP     AX,0            ;与 0比较
        JNZ     NEXT            ;不为 0,取下一个数
        INC     total           ;为 0,计数器累加
NEXT:   INC     SI              ;调整地址,指向下个数
        INC     SI
        LOOP    AGAIN           ;进入下一轮循环
```

显然，LOOP AGAIN 等效于下列语句：

```
DEC     CX
JNZ     AGAIN
```

但是，LOOP指令中完成的操作 CX ← CX −1，不影响标志位。

3. 过程调用和返回指令

如果有一些程序需要在不同的地方多次出现，则可以将这些程序段设计成过程（即子程序），供需要时调用，在过程中安排返回指令，使得过程结束时，返回到调用处。

过程与调用程序在同一段内，称“段内调用”；过程与调用程序不在同一段内，称“段间调用”。过程调用的寻址方式与转移指令类似，分为“直接寻址”、“间接寻址”。调用指令中直接给出被调用过程的首地址（标号或立即数），为“直接寻址”；预先把被调用过程的地址存于寄存器或内存，调用指令仅给出这些地址存放处（寄存器名或内存地址），为“间接寻址”。

过程调用指令为：CALL，返回指令为：RET，两者均不影响标志位，但影响堆栈内容。

(1) 过程调用指令 CALL

CALL指令格式及操作如表 3.29。

表 3.29 CALL不同寻址方式下地址计算

调用类型	寻址方式	格式	操作	示例	说明
段内调用	直接	CALL proc−name CALL disp16	① IP入栈； ② IP← IP +偏移量	CALL SUB1	① 段内调用，CS不变 ② FAR PTR表示段间调用 ③ DWORD PTR表明内存操作数属性为双字，用于段间调用
	间接	CALL r16／m16	① IP入栈； ② IP← (r16) ／(m16)	CALL BX CALL WORD PTR [BX]	
段间调用	直接	CALL FAR proc−name	① CS入栈；② IP入栈； ③ CS← 过程的段地址； ④ IP← 过程的偏移地址	JMP FAR PTR NEXT	
	间接	CALL mem32	① CS入栈；② IP入栈； ③ IP← (EA+1, EA)； ④ CS← (EA+3, EA+2)	CALL DWORD PTR [BX]	

(2) 返回指令 RET

返回指令用于从被调用过程返回到调用处。根据调用类型的不同，返回指

令操作有所不同。具体如表 3.30。

表 3.30　返回指令的格式及功能

返回类型	格　式	操　作	说　明
段内	RET	① IP出栈	格式 RET exp,允许在返回的同时,修改堆栈指针。
	RET exp	① IP出栈 ② SP← SP + exp	
段间	RET	① IP出栈 ② CS出栈	
	RET exp	① IP出栈 ② CS出栈 ③ SP← SP + exp	

4. 中断指令

中断指计算机暂时挂起正在执行的主程序而转向处理某件事,处理完后再恢复原进程的过程。对某件事的处理即执行一段例行程序,该程序被称为中断处理(子)程序。8086/8088的中断分为内部中断和外部中断。

中断处理子程序的入口地址,称为“中断向量”,由中断处理子程序所在段地址及偏移地址组成,共 32位,占 4个字节。CPU 8086/8088规定内存 0000:0000H～0000:3FFFH处存放中断向量,共 1 KB,可存入 256个中断向量,总称为“中断向量表”。中断向量在中断向量表中的位置由其类型决定,类型取值为 0～0FFH,共 256个。每个中断向量占 4个字节,类型为 n的中断,其中断向量存放处为 0000:4n～0000:4n+3连续的 4个单元,其中低 16位为偏移地址,高 16位为段地址。如图 3.29。

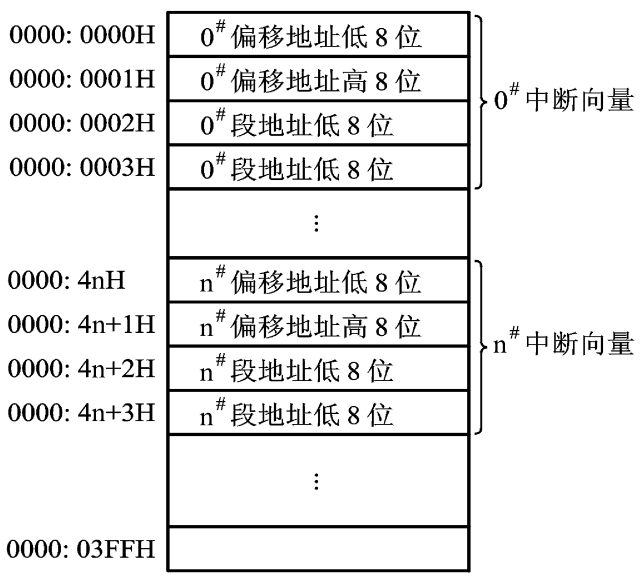


图 3.29　中断向量表示意图

与中断相关的指令有:中断调用指令 INT n、溢出中断指令 INTO、中断返回指令 IRET。

(1) 中断调用指令 INT n

格式:INT n

功能:产生一个类型为 n的软中断

操作:

1) 标志寄存器入栈;

2) 断点地址入栈,先 CS入栈,后 IP入栈;

3) 从中断向量表中获取中断服务程序入口地址,即:

IP← (0:4n+1,0:4n)

CS← (0:4n+3,0:4n+2)

(2) 溢出中断指令 INTO

格式:INTO

功能:检测 OF标志位,当 OF=1时,产生中断类型为 4的中断;当 OF=0时,不起作用。

操作:当产生中断类型为 4的中断时,有下列操作:

1) 标志寄存器入栈;

2) 断点地址入栈,先 CS入栈,后 IP入栈;

3) 从中断向量表中获取中断服务程序入口地址,即:

IP← (0:13,0:12)

CS← (0:15,0:14)

(3) 中断返回指令 IRET

格式:IRET

功能:从中断服务程序的断点处返回,继续执行原程序。用于中断处理程序中。

操作:

1) 断点出栈,先 IP,后 CS;

2) 标志寄存器出栈。

3.4.6 处理器控制指令

处理器控制指令用于控制 CPU的动作,修改标志寄存器的标志位,实现对 CPU的管理。

1. 标志位操作指令

标志位操作指令完成对标志位置位、复位等操作,共有 7条,见表 3.31。

表 3.31 标志操作指令

格 式	功 能	操 作
CLC	进位标志复位	CF←0
STC	进位标志置位	CF←1
CMC	进位标志求反	CF←$\overline{CF}$
CLD	方向标志复位	CF←0
STD	方向标志置位	DF←1
CLI	中断允许	IF←0
STI	禁止中断	IF←1

2. 外部同步指令

外部同步指令用于控制 CPU的动作,这类指令不影响标志位。

(1) 处理器暂停指令 HLT

格式 :HLT

功能 :使处理器处于暂停状态。

说明 :由 HLT引起的 CPU暂停,只有复位 (RESET信号)、外中断请求 (NMI信号或 INTR信号)可使其退出。常用于等待中断或多处理机系统的同步操作。

(2) 处理器等待指令 WAIT

格式 :WAIT

功能 :处理器检测 TEST引脚信号,当 TEST有效时,则退出等待状态执行后续指令,否则处理器处于等待状态,直到 TEST有效。

(3) 处理器交权指令 ESC

格式 :ESC ext-op,mem

ext-op是其他处理器的一个操作码,称 "外操作码 ",mem是内存操作数

功能 :使外部处理器能从 8086指令流中取得它们的操作指令,同时指示 8086/8088 CPU取出内存操作数,放到数据总线上,以供使用。

(4) 封锁总线指令 LOCK

格式 :LOCK 其他指令

LOCK指令是一个前缀,可放在任何指令的前面

功能 :封锁总线。在指令执行期间,不允许其他设备对总线进行访问。

共享资源的多处理器系统中,必须提供一些手段对这些资源的存取进行控

制,指令前缀是一种手段。

思考题与习题

3.1 8086汇编语言指令的寻址方式有哪几类?用哪一种寻址方式的指令执行速度最快?

3.2 内存寻址方式中,一般只指出操作数的偏移地址,那么,段地址如何确定?如果要用某个段寄存器指出段地址,指令中应如何表示?

3.3 在8086系统中,设DS=1000H,ES=2000H,SS=1200H,BX=0300H,SI=0200H,BP=0100H,VAR的偏移量为0060H,请指出下列指令的目标操作数的寻址方式,若目标操作数为存储器操作数,计算它们的物理地址。

(1) MOV BX,12 (2) MOV [BX],12 (3) MOV ES:[SI],AX

(4) MOV VAR,8 (5) MOV [BX][SI],AX (6) MOV 6[BP][SI],AL

(7) MOV [1000H],DX (8) MOV 6[BX],CX (9) MOV VAR+5,AX

3.4 下面这些指令中哪些是正确的?哪些是错误的?如果是错误的,请说明原因。

(1) XCHG CS,AX (2) MOV [BX],[1000]

(3) XCHG BX,IP (4) PUSH CS

(5) POP CS (6) IN BX,DX

(7) MOV BYTE [BX],1000 (8) MOV CS,[1000]

(9) MOV BX,OFFSET VAR[SI] (10) MOV AX,[SI][DI]

(11) MOV COUNT[BX][SI],ES:AX

3.5 试述以下指令的区别。

(1) MOV AX,3000H 与 MOV AX,[3000H]

(2) MOV AX,MEM 与 MOV AX,OFFSET MEM

(3) MOV AX,MEM 与 LEA AX,MEM

(4) JMP SHORT L1 与 JMP NEAR PTR L1

(5) CMP DX,CX 与 SUB DX,CX

(6) MOV [BP][SI],CL 与 MOV DS:[BP][SI],CL

3.6 设DS=2100H,SS=5200H,BX=1400H,BP=6200H,说明下面两条指令所进行的具体操作。

(1) MOV BYTE PTR [BP],2000

(2) MOV WORD PTR [BX],2000

3.7 设当前SS=2010H,SP=FE00H,BX=3457H,计算当前栈顶的地址为多少?当执行PUSH BX指令后,栈顶地址和栈顶2个字节的内容分别是什么?

3.8 设DX=78C5H,CL=5,CF=1,确定下列各条指令执行后,DX和CF中的值。

(1) SHR DX,1 (2) SAR DX,CL (3) SHL DX,CL

(4) ROR DX,CL (5) RCL DX,CL (6) RCR DH,1

3.9　设 AX=0A69H,VALUE字变量中存放的内容为 1927H,写出下列各条指令执行后 AX寄存器和 CF、ZF、OF、SF、PF的值。

(1) XOR　AX, VALUE　　(2) AND　AX, VALUE

(3) SUB　AX, VALUE　　(4) CMP　AX, VALUE

(5) NOT　AX　　(6) TEST　AX, VALUE

3.10　设 AX和 BX中是符号数,CX和 DX是无符号数,请分别为下列各项确定 CMP和条件转移指令。

(1) CX值超过 DX转移。

(2) AX值未超过 BX转移。

(3) DX为 0转移。

(4) CX值等于小于 DX转移。

3.11　阅读分析下列指令序列:

```
ADD　AX, BX
JNO　L1
JNC　L2
SUB　AX, BX
JNC　L3
JNO　L4
JMP　L5
```

若 AX和 BX的初值分别为以下 5种情况,则执行该指令序列后,程序将分别转向何处(L1～L5中一个)。

(1) AX = 14C6H, BX = 80DCH

(2) AX = 0B568H, BX = 54B7H

(3) AX = 42C8H, BX = 608DH

(4) AX = 0D023H, BX = 9FD0H

(5) AX = 9FD0H, BX = 0D023H

3.12　用普通运算指令执行 BCD码运算时,为什么要进行十进制调整?具体讲,在进行 BCD码的加、减、乘、除运算时,程序段的什么位置必须加上十进制调整指令?

3.13　在编制乘法程序时,为什么常用移位指令来代替乘除法指令?试编写一个程序段,实现将 BX中的数乘以 10,结果仍放在 BX中。

3.14　串操作指令使用时与寄存器 SI、DI及方向标志 DF密切相关。请具体就指令 MOVSB/MOVSW、CMPSB/CMPSW、SCASB/SCASW、LODSB/LODSW、STOSB/STOSW列表说明和 SI、DI及 DF的关系。

3.15　用串操作指令设计实现以下功能的程序段:首先将 100H个数从 2170H处搬到 1000H处,然后,从中检索等于 AL中字符的单元,并将此单元值换成空格符。

3.16　求双字长数 DX:AX的相反数。

3.17　试对物理地址为 53481H单元中的单字节数求补后存入 53482H,最高位不变,低

7位取反存入 53483H,高 4位置 1,低 4位不变,存入 53484H。

3.18 自 1000H单元开始有 1 000个单字节带符号数,找出其中最小值,放在 2000H单元。

3.19 试编写一个程序,比较两个字符串 STRING1和 STRING2所含字符是否完全相同,若相同则显示“MATCH”,若不同则显示“NO MATCH”。

3.20 用子程序的方法,计算 a+10b+100c+20d,其中 a,b,c,d均为单字节无符号数,存放于数据段 DATA起的 4个单元中,结果为 16位,存入 DATA+4的两单元中。

3.21 试编写一段程序把 LIST到 LIST+100中的内容传送到 BLK到 BLK+100中去。

3.22 在自 BUFFER单元开始有一个数据块,BUFFER和 BUFFER+1单元中放的是数据块长度,自 BUFFER+2开始存放的是以 ASCII码表示的十进制数码,把它们转换为 BCD码,且把两个相邻单元的数码并成一个单元 (地址高的放在高 4位),放到自 BUFFER+2开始的储存区。

3.23 设 CS:0100H单元有一条 JMP SHORT LAB指令,若其中的位移量为:

⑴ 56H ⑵ 80H ⑶ 78H ⑷ 0E0H

试写出转向目标的物理地址是多少?

3.24 不使用除法指令,将堆栈段中 10H、11H单元中的双字节带符号数除以 8,结果存入 12H、13H单元 (注:多字节数存放格式均为低位在前,高位在后)。

3.25 内存 BLOCK存有 32个双字节带符号数,试将其中的正数保持不变,负数求补后放回原处。

3.26 数据段中 3030H起有两个 16位的带符号数,试求它们的积,存入 3034H~3036H单元中。

第4章

汇编语言程序设计

汇编语言（Assembly Language）是一种采用指令助记符、符号地址、标号、伪指令等符号编写程序的程序设计语言。用汇编语言编写的程序称为汇编语言源程序（Source Program），将汇编语言源程序转换成机器能执行的语言（目标代码程序，Object Program）的过程称为汇编。完成汇编任务的软件称为汇编程序。

一般情况下，一个助记符表示一条机器指令，所以汇编语言也是面向机器的语言。用汇编语言编写的程序能够直接利用硬件的特性（如寄存器、标志位、中断系统等），直接对位、字节或字寄存器或存储单元、I/O端口进行处理，同时也能直接使用CPU指令系统和指令系统提供的各种寻址方式。汇编语言程序不但占用内存空间少，而且执行速度快。在有些应用领域，汇编语言的作用是不容置疑和无可替代的。

本章首先介绍汇编语言程序格式，展开其中每个部分，引出基本的汇编语言伪指令，然后，就顺序、循环、子程序结构等方面论述汇编语言的各种程序设计方法。

4.1 汇编语言语法

4.1.1 源程序的结构及组成

1. 汇编语言源程序结构

汇编语言介于机器语言和高级语言之间，汇编语言程序设计与高级语言程序设计有相似之处，但有很大的不同。下面通过一个具体程序例子，来看如何使用汇编语言编程。

【例题 4.1】 实现：123+456→sum的源程序。

```
DATA    SEGMENT                          行 1
    A   W       123                      行 2
    B   DW      456                      行 3
  SUM   DW      ?                        行 4
DATA    ENDS                             行 5
CODE    SEGMENT                          行 6
MAIN    PROC    FAR                      行 7
        ASSUME  CS:CODE, DS:DATA         行 8
START:  USH     DS                       行 9
        MOV     AX, 0                    行 10
        PUSH    AX                       行 11
        MOV     AX, DATA                 行 12
        MOV     DS, AX                   行 13
        MOV     AX, A                    行 14
        ADD     AX, B                    行 15
        MOV     SUM, AX                  行 16
        RET                              行 17
        MAIN    ENDP                     行 18
CODE    ENDS                             行 19
END     START                            行 20
```

⑴ 段式结构

从上述程序中，可明显地看出段式结构。该程序共有 2个段：行 1～行 5为一段、行 6～行 19为一段，DATA、CODE 分别为 2个段的名字。每一段有明显的起始语句与结束语句，这些语句称为“段定义”语句。代码段的第一个语句（本例中行 8）ASSUME，用于明确段与段寄存器的关系。由此可知，本程序中 DATA 是数据段、CODE 是代码段。一个汇编语言源程序中，代码段不可缺少，其他段视具体情况而定。

⑵ 语句

组成源程序的是语句，汇编语言源程序语句可以分为：指令语句、伪指令语句和宏指令语句。

1）指令语句

指令语句是能产生目标代码的语句，这些目标代码可供 CPU执行并完成特定操作。其格式为：

[标号：] 操作码 [操作数] [;注释]

其中,操作码和操作数是用助记符表示的指令的两个部分。标号具有该指令语句所在内存地址属性,通常在转移指令中用作目的地址。注释简单说明本语句的功能或在程序中的作用。

⑵ 伪指令语句/指示性语句

伪指令语句是一种不产生目标代码的语句,它仅仅在汇编过程中告诉汇编程序应如何汇编。例 4.1中告诉汇编程序哪些语句是属于一个段、是什么类型的段、各段存入内存应如何组装、给变量分配多少存储单元、给数字或表达式命名等。其格式为:

[名字/变量] 伪指令 参数 [;注释]

⑶ 宏指令语句

宏是若干语句组成的程序段,宏指令语句用来定义宏。一旦把某程序段定义成宏,则可以用宏名代替那段程序。在汇编时,要对宏进行宏展开,展开的过程是将宏名用程序段代替。宏指令格式为:

[标号:] 宏指令 参数 1,…, [;注释]

⑶ 设置返回操作系统的功能

计算机一旦启动成功,由操作系统掌握 CPU的控制权。应用程序只是作为操作系统的子程序,应用程序执行完,必须返回操作系统。上述程序的行 9~行 11及行 17就为了完成此功能而设计的。

2. 汇编语句中的名字

汇编语言源程序中的变量名、标号、常量名、段名、宏名等统称为“名字”。

⑴ 名字的命名规则

名字命名,具有以下规则:

1) 组成名字的合法字符有:字母(不分大小写)、数字及特殊符号(“?”,“:”,“@”,“_”,“$”)。

2) 名字的有效长度小于 31个西文字符。

3) 名字以字母开头。

4) 不能把保留字用作名字。

⑵ 名字及其属性

1) 标号名

标号在代码段中定义,写在指令语句之前,其名字之后有冒号“:”,也可用 EQU或 LABEL伪指令定义。不是所有指令语句必须有标号,只有那些是程序跳转点处的语句前必须设有标号。

标号有三种属性:段、偏移量、类型。标号实际代表一个地址,段属性的值即段地址,偏移量属性的值即偏移地址。类型属性用来指明该标号是被段内引用还是被段

间引用,如果是段内引用,其类型值为 NEAR,如是段间引用,其类型值为 FAR。

2) 变量名

变量在除代码段外的其他段定义,后面不需要冒号“:”,它也可以用 EQU 或 LABEL伪指令定义。变量名代表存储器中的一个数据区的名字,有 5种属性:段、偏移量、类型、长度、规模。

段属性:是变量所代表的数据区所在段的段基址。

偏移属性:是变量所代表的数据区首字节所在段内偏移地址。

类型属性:变量的类型有:BYTE(字节)、WORD(字)、DWORD(双字)、DQ(4字)、DT(5字)等,表示数据区中存、取操作对象的大小。

长度属性:表示该变量所代表的数据区中数据元素的个数。

规模属性:表示变量所代表的数据区中数据所占空间大小,以字节计。

3) 段名

段名,顾名思义是段的名称,在段定义中给出。源程序在进行汇编连接时,系统分配给段一个段基值。段名可作为段基值被引用。例题 4.1中行 12、13,通过段名 DATA给 DS赋值。

4) 过程名

过程名,即过程的名称,在过程定义中给出。例题 4.1中行 7、行 18为过程定义语句,MAIN为过程名。汇编连接目标程序时,系统分配给过程名一个地址,即该过程第一条指令所在内存单元的地址,称为“过程的入口地址”。

5) 符号常量名

符号常量名用于代替一常数,以增加程序的可读性及通用性。

4.1.2 汇编语言伪指令

伪指令从表示形式及其在语句中所处的位置,与 CPU指令相似,但二者有着重要的区别。首先,伪指令不像机器指令那样是在程序运行期间由 CPU来执行,它是在汇编程序对源程序汇编期间由汇编程序处理的操作;其次,汇编以后,每条 CPU指令产生一一对应的目标代码,而伪指令则不产生与之相应的目标代码。

宏汇编程序 MASM提供了几十种伪指令,大致可分为以下几类:

(1) 数据定义伪指令;

(2) 符号定义伪指令;

(3) 段定义伪指令;

(4) 过程定义伪指令;

(5) 宏处理伪指令;

⑹ 模块定义与结束伪指令；

⑺ 处理器方式伪指令；

⑻ 条件伪指令；

⑼ 列表伪指令；

⑽ 其他。

本节介绍一些常用的基本伪指令。

1. 数据定义伪指令

数据定义伪指令用来为数据分配存储单元，建立变量与存储单元之间的联系。语句格式为：

［变量名］数据定义伪指令 操作数 1［,操作数 2…］

伪指令有：DB、DW、DD、DQ、DT，分别用来定义类型属性为字节（DB）、字（DW）、双字（DD）、4字（DQ）、5字（DT）的变量。

操作数可以是：

① 数字常量，允许以十进制、八进制、十六进制、二进制等形式表示，缺省形式是十进制；

② 字符常量，用单引号括起来，被存储的是该字符的 ASCII码；

③ 符号常量，必须是预先已定义的符号；

④ 符号“?”，表示预留空间，内容不定；

⑤ DUP，表示内容重复的数据。具体形式为：

次数　DUP　（被重复内容）

【例】　数据定义如下，其存储示意如图 4.1所示。

```
DATA_B    DB    10, 'A', ?
DATA_W    DW    1234H, ?
DATA_S    DB    '1234', 2 DUP (1, 2 DUP (2))
```

从图知：

① DB定义的数据，每个数据元素占据 1个存储单元；DW定义的数据，每个数据元素占据 2个存储单元。

② 字数据存储时，低字节存储在低地址单元中，高字节存储在高地址单元中。

③ 字符被存放时为它的 ASCII码，‘A’的 ASCII码为 41H。

④ DUP可以嵌套使用。

⑤ 符号地址具有以下关系：

DATA_W＝DATA_B＋3

DATA_S＝DATA_W＋4＝DATA_B＋7

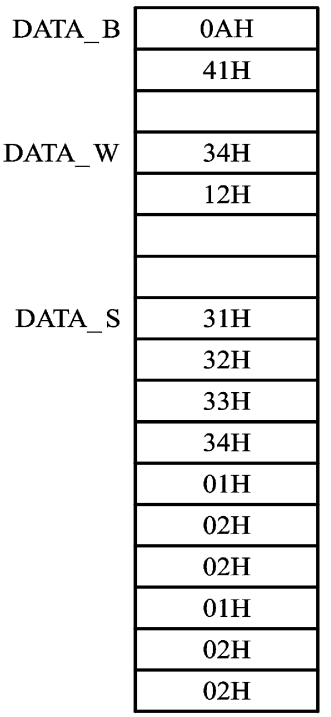


图 4.1 数据存储示意

2. 符号定义伪指令

符号包括汇编语言的变量名、标号名、过程名、寄存器名及指令助记符等。常用符号定义伪指令有：EQU、=、LABEL。

⑴ EQU

格式：名字 EQU 表达式

表达式可以是一个常数、已定义的符号、数值表达式或地址表达式。

功能：给表达式赋予一个名字。定义后，可用名字代替表达式。

【例】

```
VB      EQU   64×1024        ; B代表数值表达式的值
A       EQU   7
B       EQU   A-2
```

必须注意，在 EQU语句的表达式中，如果有变量的表达式，则在该语句前应先给出它们的定义；EQU语句不能给某一变量重复定义。

⑵ 等号 =

格式：名字 =表达式

功能：与 EQU基本相同，区别是它可以对同一个名字重新定义。

【例】

```
COUNT=10
MOV     AL,COUNT
……
COUNT=5
……
```

(3) LABEL

格式:变量/标号 LABEL类型

变量的类型有:BYTE、WORD、DWORD、DQ、DT;标号的类型有:NEAR、FAR。

功能:定义变量或标号的类型,而变量或标号的段属性和偏移属性由该语句所处的位置确定。

【例】 利用 LABEL使同一个数据区有一个以上的类型及相关属性。

```
AREAW     LABEL   WORD          ; REAW与 AREAB指向相同的数据区,AREAW类型为字,而 AREAB类型为字节
AREAB     DB      100 DUP(?)
……
MOV       AX,1234H
MOV       AREAW,AX              ; (AREAW)=1234H
……
MOV       BL,AREAB              ;BL=34H
```

3. 段定义伪指令

汇编源程序以段为其基本组织结构,段定义伪指令用于汇编源程序中段的定义,相关指令有:SEGMENT、ENDS、ASSUME。

(1) 段定义伪指令 SEGMENT、ENDS

格式:段名 SEGMENT [定位类型] [组合类型] ['类别']

```
        ……
        段名   ENDS
```

功能:定义一个逻辑段。

SEGMENT和 ENDS必须成对使用,它们前面的段名必须是一致的。SEGMENT后面中括号中的内容为可选项,告诉汇编程序和连接程序如何确定段的边界、如何连接几个程序模块。

1) 定位类型

定位类型说明段的起始地址应有怎样的边界值,它们可以是:

BYTE:×××× ×××× ×××× ××××B,即段可以从任何地址开始;

WORD:×××× ×××× ×××× ××× 0B,即段的起始地址必须为偶地址;

PARA:×××× ×××× ×××× 0000B,即段从节(PARAGRAPH)边界开始,内存中,每16个字节为1小段,所以,定位类型为PARA的段,其起始地址必为16的倍数。

PAGE:×××× ×××× 0000 0000B,即段从页边界开始,内存中,每256个字节为1页,所以,定位类型为PAGE的段,其起始地址必为256的整数倍数。

定位类型的缺省值为PARA。

2) 组合类型

组合类型说明程序连接时的段合并方法,它们可以是:

① PUBLEC:将同名(即类别名相同)段组装在一起形成一个逻辑段。

② STACK:与PUBLIC一样,只用于堆栈段。在汇编及连接后,系统自动为SS及SP分配值,在可执行程序中,SP初值指向栈底。

③ COMMON:同名段从同一个内存地址开始装入。所以,各个逻辑段将发生覆盖。连接以后,该段长度取决于同名段中最长的那个,而内容有效的是最后装入的那个。

④ MEMORY:与PUBLIC同义,只不过MEMORY定义的段装在所有同名段的最后。若连接时出现多个MEMORY,则最先遇到的段按组合类型MEMORY处理,其他段组合类型按PUBLIC处理。

⑤ PRIVATE:不组合,该段与其他段逻辑上不发生关系,即使同名,各段拥有各自的段基值。

⑥ AT exp:段地址为表达式exp的值(长度为16位)。此项不能用于代码段。

组合类型的缺省值为PRIVATE。

3) 类别

类别名必须用单引号括起来。类别的作用是在连接时决定各逻辑段的装入顺序。当几个程序模块进行连接时,其中具有相同类别名的段,按出现的先后顺序被装入连续的内存区。没有类别名的段,与其他无类别名的段一起连续装入内存。

(2) ASSUME

格式:ASSUME 段寄存器名:段名 [,段寄存器名:段名……]

段寄存器可以是:CS、DS、ES、SS。段名为已定义的段。凡是程序中使用的段,都应说明它与段寄存器之间的对应关系。

功能:用于明确段与段寄存器的关系。

注意：本伪指令只是指示各逻辑段使用段寄存器的情况，并没有对段寄存器的内容进行赋值。DS、ES的值必须在程序段中用指令语句进行赋值，而 CS、SS由系统负责设置，程序中也可对 SS进行赋值，但不允许对 CS赋值。

4. 过程定义伪指令

过程定义伪指令用于定义过程。指令格式如下：

过程名　　PROC　[类型]

……

RET

……

过程名　　ENDP

过程名按汇编语言命名规则设定，汇编及连接后，该名称表示过程程序的入口地址，供调用使用。

PROC与 ENDP必须成对出现，PROC开始一个过程，ENDP结束一个过程。成对的 PROC与 ENDP的前面必须有相同的过程名。

类型取值为：NEAR或 FAR，表示该过程是段内调用或段间调用，缺省值为：NEAR。

一个过程中，至少有一条过程返回指令 RET，一般出现在 ENDP之前。

5. 模块定义和结束伪指令

在编写规模比较大的汇编语言程序时，可以将整个程序划分为几个独立的源程序（或模块），然后将各个模块分别进行汇编，生成各自的目标程序，最后将它们连接成为一个完整的可执行程序。

⑴ NAME

格式：NAME　模块名

功能：为源程序的目标程序指定一个模块名。

如果程序中没有 NAME伪指令，则汇编程序将 TITLE伪指令定义的标题名前 6个字符作为模块名；如果程序中既没有 NAME，又没有 TITLE，则汇编程序将源程序的文件名作为目标程序的模块名。

⑵ END

格式：END　[标号]

功能：表示源程序的结束。

标号指示程序开始执行的起始地址。如果多个程序模块相连接，则只有主程序要使用标号，其他子模块则只用 END而不必指定标号。

6. 其他伪指令

⑴ 对准伪指令 EVEN

格式：EVEN

功能：使下一个分配地址为偶地址。

在 8086中，一个字的地址最好为偶地址。因为 8086 CPU同样存取一个字，如果地址是偶地址，需要一个读或写周期，如果是奇地址，需要两个读或写周期。所以，该伪指令常用于字定义语句之前。

【例】

```
DSEG    SEGMENT
……
        EVEN
ARR_W   DW   100 DUP (?)
……
DSEG    ENDS
```

(2) 定位伪指令 ORG

格式：ORG 表达式

表达式取值范围为：0～65 535，无符号数

功能：指定其后的程序段或数据块所存放的起始地址的偏移量。

【例】

```
MY_DAT          SEGMENT
        ORG     100H
ARRAY           DW   1, 2, $ +4
MY_DATA         ENDS
```

从 DS:100H 开始为变量 ARRAY 分配存储空间。存储示意如图 4.2所示。符号“$ ”代表当前地址，第 3个数据 $ +4＝104H +4＝108H。如果没有 ORG

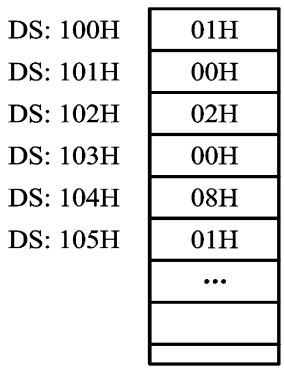


图 4.2 数据存储示意

伪指令,一般从 DS:0开始为变量分配存储空间。

⑶ 基数控制伪指令 RADIX

格式 :RADIX　表达式

表达式取值为 2～16内任何整数。

功能 指定汇编程序使用的默认数制。缺省为十进制。

【例】

```
MOV     BX, 0FFH                ;十六进制数要加后缀
MOV     BX, 178                 ;十进制数不要加后缀
RADIX   16                      ;设置十六进制为默认数制
MOV     AX, 0FF                 ;十六进制数不加后缀
MOV     BX, 178D                ;十进制数要加后缀
```

4.1.3 汇编语句

汇编语句分指令语句、指示语句、宏语句等,语句基本格式为 :

[名字]　操作　[操作数]　[;注释]

操作项是指令中不可缺少的,可以是指令、伪指令及宏指令助记符等。操作数或有或无或多个,多个操作数之间用逗号“,”隔开。指令语句中,操作数项可以是常量、寄存器、标号、变量或表达式等。在 8088 /8086指令系统中,操作数项 0～2个。伪指令和宏指令的操作数项可以是常量、变量或表达式等。

1. 常量

常量是具有一定值的量,并且其值不能改变。汇编语言程序中的常量有 :数字常量、字符常量和符号常量。

⑴ 数字常量

数字常量的表现形式有 :十进制、八进制、十六进制、二进制等。例 12D、75Q、8H 均为数字常量。

⑵ 字符常量

字符常量用单引号括起来,如‘a’、‘1’等。字符常量在操作中体现出的值是其 ASCII码值。

⑶ 符号常量

用名字来标志的常量,称“符号常量”。以符号代替常量,用以增加程序的可读性及通用性,例如 :COUNT EQU 10或 COUNT=10

2. 变量

变量是可以具有不同值的量。变量通过数据定义伪指令定义,与某个存储

区相关联。按变量值的长度的不同,变量的类型有:字节型 (BYTE)、字型 (WORD)、双字型 (DWORD)、4字型 (DQ)、5字型 (DT)等。源程序经汇编产生的目标程序中,变量名和它所定义的存储区的首地址一一对应,所以,通过变量名对内存的访问,如同通过内存地址对内存的访问 (即直接寻址方式),两者是等同的。

【例】

数据定义

```
ARRAY    DW    1234H
……
```

数据访问

```
MOV      AX, ARRAY                    ;AX =1234H
```

3. 表达式及运算符

表达式是由常量、变量和运算符等组成的具有可计算值的式子。表达式按其特性分为两种:数值表达式和地址表达式。数值表达式产生的值,只有大小;地址表达式产生的值,不仅有大小,还有段、偏移量和类型属性。

运算符是表达式中的重要部分,通过它,把常量、变量等元素联系在一起。汇编中的运算符可分为以下几类:算术运算符、逻辑运算符、关系运算符、分析运算符、属性运算符及其他,如表 4.1所示。

表 4.1 汇编运算符

运算分类	运 算 符
算术运算	+、-、*、/、MOD
逻辑运算	AND、OR、NOT、XOR
关系运算	EQ、NE、LT、GT、LE、GE
分析运算	SEG、OFFSET、TYPE、LENGTH、SIZE
属性运算	PTR、THIS、SHORT
其他	LOW、HIGH

(1) 算术运算符

算术运算符有 5种,即加 (+)、减 (-)、乘 (*)、除 (/)、求余 (MOD)。MOD的操作是将两个整数相除后取余数。这 5种运算符全部适用于数值表达

式,地址表达式中只能用加 (+)、减 (-)。

【例】

```
ARRAY    DW   1* 2+3-4,56H
……
MOV      BX,ARRAY +2                    ;AX =0056H
```

其中,1* 2+3-4为数值表达式,ARRAY +2为地址表达式。这些表达式的计算均在汇编过程中计算,不是由 CPU完成,目标程序中,以表达式的值代替这些表达式。

⑵ 逻辑运算符

逻辑运算按位操作,共有 4种运算符,与 (AND)、或 (OR)、非 (NOT)、异或 (XOR),只适用于数值表达式,操作运算的定义与逻辑指令 AND、OR、NOT、XOR类似。

【例】 从端口 86H读取一个字节,高位屏蔽后从端口 6送出。

```
PORT     EQU   86H
IN       AL,PORT
AND[1]   AL,0FH
MOV      DX,PORT AND[2] 0FH
OUT      DX,AL
```

AND[1]是逻辑指令助记符,由 CPU完成,AND[2]是逻辑运算符,在汇编中完成。

⑶ 关系运算符

关系运算符用于数的比较,汇编语言提供 6种关系运算符:相等 (EQ)、不相等 (NE)、小于 (LT)、大于 (GT)、小于等于 (LE)、大于等于 (GE)。关系运算符两边的操作数必须是两个数值或同一段中两个存储单元地址,运算结果应为逻辑值,结果为真,表示为 0FFFFH;结果为假,则表示为 0。

【例】 MOV AX,4 EQ 3 汇编结果为:MOV AX,0

⑷ 分析/数值返回运算符

分析运算符取得操作数的属性值,运算符有:SEG、OFFSET、TYPE、SIZE、LENGTH。

1) SEG

格式:SEG 变量或标号

返回值:变量或标号的段地址

2) OFFSET

格式：OFFSET 变量或标号

返回值：变量或标号的偏移量

【例】 将从 AREA1开始的数据块传送到 AREA2开始的区域。注意程序中的地址传送语句。

```
DATA    SEGMENT
        AREA1   DW      1,2,3,4,5,6,7,8,…
        COUNT   EQU     ($ -AREA1) /2
        AREA2   DW      COUNT DUP (?)
DATA    ENDS
        ……
        MOV       X, SEG AREA1          ;源数据区的段地址给 DS
        MOV     DS, AX
        MOV     ES, AX                  ;目标数据区的段地址给 ES
        MOV     SI, OFFSET  AREA1       ;源数据区指针 SI赋初值
        MOV     DI, OFFSET  AREA2       ;目标数据区指针 DI赋初值
        MOV     CX, COUNT
        REP     MOVSW
        ……
```

3) TYPE

格式：TYPE 变量或标号

返回值：变量或标号的类型值

如果该表达式是变量，则汇编程序将回送该变量的以字节表示的类型：DB为 1，DW为 2，DD为 4，DQ为 8，DT为 10。如果表达式是标号，则汇编程序将回送代表该标号类型的数值：NEAR为 -1，FAR为 -2。

4) LENGTH

格式：LENGTH 变量

返回值：DUP定义的数据占据的单元数；非 DUP定义的数据，返回 1。

5) SIZE

格式：SIZE 变量

返回值：DUP定义的数据占据的字节数；非 DUP定义的数据，取类型值。

显然，SIZE = TYPE * LENGTH

【例】 数据定义如下：

```
DATA    SEGMENT    AT 2000H
        BUF1    DB      0,1,2,3,4,5,6,7,8,9
        BUF2    DW      5 DUP (0)
DATA    ENDS
```

则：

SEG BUF1＝2000H	SEG BUF2＝2000H
OFFSET BUF1＝0000H	OFFSET BUF2＝000AH
TYPE BUF1＝1	TYPE BUF2＝2
LENGTH BUF1＝1	LENGTH BUF2＝5
SIZE BUF1＝1	SIZE BUF2＝10

⑸ 属性运算符

属性运算符用来建立或改变已定义变量、内存操作数或标号的类型属性。属性运算符有：PTR、THIS、SHORT。

1) PTR

格式：类型　PTR　变量/标号

返回值：具有规定类型属性的变量或标号。

典型应用：

① 重新指定变量类型

【例】　数据定义如下：

```
BUFW    DW      1234H,5678H
```

则下列指令合法：

```
MOV     AX,BUFW
MOV     AL,BYTE PTR BUFW        ;临时改变 BUFW 的字属性为字节属性
```

② 指定内存操作数的类型

在寄存器间接寻址、寄存器相对寻址、基址变址寻址或相对基址变址寻址等内存寻址方式中，往往很难判断出操作数的类型属性

【例】　INC　　[BX]

此时，汇编将指示出错，为了避免出错，应对操作数类型加以说明，如下所示：

```
INC     BYTE PTR [BX]                   ;字节属性
INC     WORD PTR [BX][SI]               ;字属性
```

③ 与 EQU一起定义一个新的变量

格式：变量或标号 EQU 类型 PTR

新变量或新标号的段属性与偏移属性与前面已定义的变量或标号段属性与偏移属性相同。

【例】

```
BUFW    DW   1234H,5678H          ;一个已定义的字变量 BUFW
BUFB    EQU  BYTE PTR BUFW       ; UFB的类型属性为字节,其他属
                                    性与 BUFW一样
```

进行字存取时,可用变量 BUFW,如：MOV AX,BUFW

进行字节存取时,可用变量 BUFB,如：MOV AL,BUFB

2) THIS

格式：THIS 类型

可以像 PTR一样建立一个指定类型的地址操作数,该操作数的段地址和偏移地址与下一个存储单元地址相同。例如：

```
BUFB    EQU    THIS BYTE
BUFW    DW     1234H,5678H
```

此时 BUFB的偏移地址和 BUFW完全相同,但它是字节类型的;而 BUFW则是字类型的。

3) SHORT

格式：SHORT 标号

返回值：偏移量在 -128～+127范围内的标号。

用于 JMP指令。即：JMP SHORT 标号,指明是短转移。

(6) 其他

字节分离运算符 HIGH、LOW

格式： HIGH 表达式

LOW 表达式

返回值：表达式值的高字节或低字节

【例】

```
CONST    EQU   0ABCDH
MOV      AH,HIGH    CONST      ;AH=0ABH
MOV      CL,LOW     CONST      ;CL=0CDH
```

以上介绍了表达式中使用的各种运算符,如果一个表达式同时具有多种运算符,按优先级的高低进行运算。运算符的优先级如表 4.2所示。

表 4.2　运算符的优先级

优　先　级	运　算　符
1	()、[]、< >
2	LENGTH、SIZE、WIDTH
3	PTR、OFFSET、SEG、TYPE、THIS
4	HIGH、LOW
5	*、/、MOD、SHL、SHR
6	+、-
7	EQ、NE、LT、GT、LE、GE
8	NOT
9	AND
10	OR、XOR
11	SHORT

4. 注释项

注释项用来说明一段程序或一条或几条指令的功能,此项是可有可无的。但是,对于汇编语言源程序清单来说,注释项可以使程序易于被读懂;而对编写程序的人来讲,注释项可以是一种“备忘录”,所以必须写好注释项。例如,一般在循环程序的开始都有初始化程序,置有关工作单元的初值:

```
MOV     CX,100          ;将 100送入 CX
MOV     SI,0100H        ;将 0100H送入 SI
MOV     DI,0200H        ;将 0200H送入 DI
```

这样注释没有告诉它们真正在程序中的作用,应该改为:

```
MOV     CX,100          ;循环计数器 CX置初值
MOV     SI,0100H        ;源数据区指针 SI置初值
MOV     DI,0200H        ;目标数据区指针 DI置初值
```

因此,编写好程序后,如何写好注释也是一个重要的部分。

4.2 汇编语言程序实现

4.2.1 汇编语言程序实现步骤

汇编语言程序实现过程如图 4.3所示。

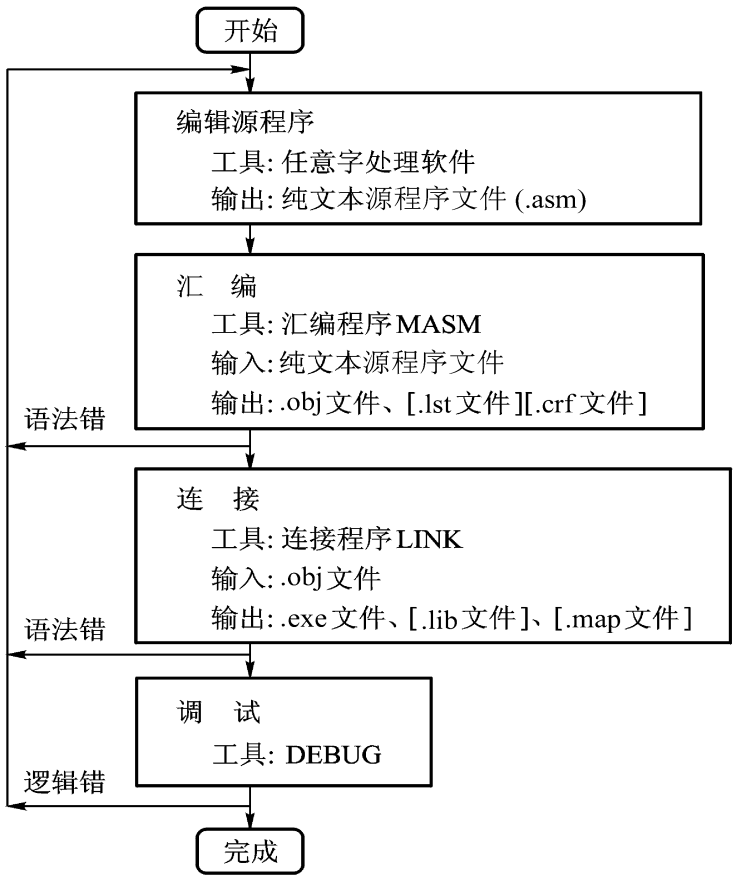


图 4.3 汇编语言程序实现流程

1. 编辑源程序

用字处理软件创建源程序。常用编辑工具有：EDIT. COM、记事本、Word等。无论采用何种编辑工具，生成的文件必须是纯文本文件，所有字符为半角。

2. 汇编

用汇编工具，对上述源程序文件（如.ASM）进行汇编，产生目标文件（OBJ）等文件。汇编程序的主要功能是检查源程序的语法，并给出错误信息；产生目标程序文件；展开宏指令。

3. 连接

汇编产生的二进制目标文件（OBJ）并不是可执行的程序，还要用连接程序把它转换为可执行的 EXE文件。

4. 程序运行

在建立了 EXE文件后，只需在提示符下键入文件名即可运行程序。若程序能够运行但不能得到预期结果，则需要静态或动态查错。静态查错即检查源程序，并用文本编辑器进行修改，然后再汇编、连接、运行。

5. 程序调试及结果查看

有时静态检查不容易发现问题，尤其是碰到复杂的程序更是如此，这时就需要使用调试工具动态查错。当程序结果不能在屏幕上显示时也需要用调试工具查看结果。常用的动态调试工具为 DEBUG。

4.2.2 COM 文件的生成

可执行文件除了上述的 EXE文件外，另有一种是 COM文件。COM文件比 EXE文件短小高效。下面介绍创建 COM文件的两种方法。

方法 1：把 EXE文件转换成 COM文件

不是任何 EXE文件都能转换，能够被转换成 COM文件的 EXE文件，其源程序必须是 TINY模式，即必须满足以下条件：

⑴ 程序长度不大于 64KB，只有一个代码段，ASSUME四个段寄存器指向同一个段；

⑵ 程序中所有过程都必须为 NEAR，程序中不得出现段间转移和段间调用指令；

⑶ 程序入口点是 0100H；

⑷ 程序中可使用 DB和 DW定义数据，且将这些语句放在指令语句之后的程序尾部。

对于符合转换条件的 EXE文件，用 EXE2BIN转换程序生成 COM文件。操作方法是在提示符下键入命令及文件名。设 AB.ASM源程序符合上述条件，且汇编及连接后，生成了可执行文件 AB.EXE，以下命令语句完成由 AB.ASM生成 AB.COM。

EXE2BIN AB.EXEf

利用高版本的 MASM6.x，对 TINY模式源程序可直接生成 COM文件。

方法 2：用 DEBUG生成 COM文件

下面举例说明。（画线部分为键盘输入内容）

```
C:\MASM>debug↵                          启动 DEBUG
-A CS:0100↵                             编辑源程序
××××:0100 MOV DX,0109↵                 本程序实现屏幕显示“Hello!”
××××:0103 MOV AH,9↵
××××:0105 INT 21↵
××××:0107 INT 20↵
××××:0109 DB 'Hello! $'↵
××××:0110 -
-N ABC.COM↵                             文件命名为 ABC.COM
-R BX↵                                  设置文件长度,高 16位存于 BX,
                                        低 16位存于 CX
BX 0000↵
:↵
-R CX↵                                  设置文件长度,高 16位存于 BX,
                                        低 16位存于 CX
CX 0000
:10↵                                    源程序共有 10H个字节
-W↵                                     存盘
Writing 0011 bytes
-Q↵                                     退出 DEBUG
C:\MASM>ABC.COM↵                        运行程序
```

4.2.3 可执行程序的装入

1. EXE程序

EXE程序有独立的代码段、数据段、堆栈段,并且每种类型的段可以有一个以上,程序大小可以超过 64 KB,程序起始地址可以由操作系统任意指定。EXE程序文件在磁盘上由两部分组成:文件头和装入模块。装入模块即程序本身,文件头则由连接程序生成,含有文件的控制信息和重定位信息,供 DOS装入 EXE文件时使用。DOS装入 EXE文件的过程如下:

⑴ DOS确定当前主存最低的可用地址作为该程序的装入起始点;

⑵ DOS在偏移地址 00H～0FFH (共 256字节)处,为该程序建立一个程序段前缀控制块 (PSP, Program Segment Prefix);

⑶ DOS利用文件头对有关数据进行重新定位,从偏移地址 100H开始装

入程序本身；

⑷ 程序装载成功，DOS将控制权交给该程序，开始执行，CS和 IP指向的第一条指令。

EXE装入内存的映像如图 4.4所示。

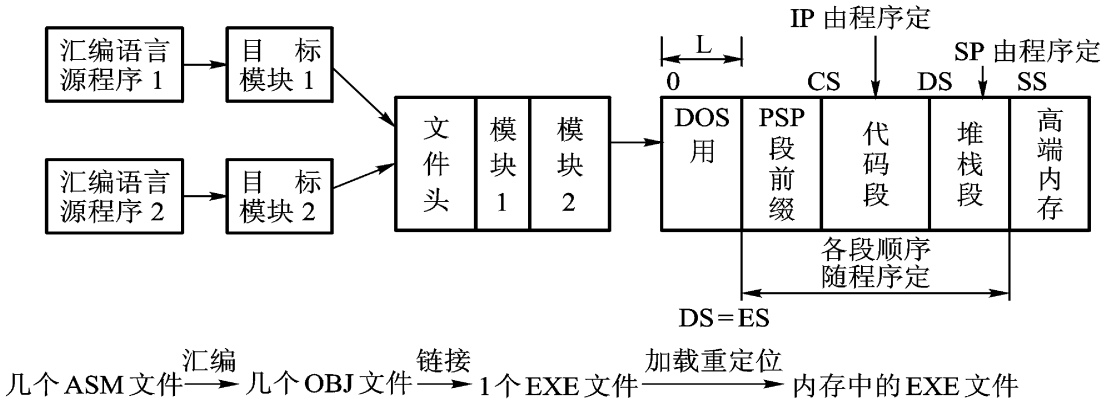


图 4.4　程序连接与定位过程图

由图知：

⑴ DS和 ES指向 PSP（程序前缀），而不是用户程序的数据段和附加段，所以需在程序中根据实际的数据段和附加段设置 DS和 ES值。

⑵ CS和 IP指向代码段第一条指令处，所以程序会从第一条指令开始执行。

⑶ SS和 SP指向堆栈段。源程序中如果没有堆栈段，则 SS为 PSP所在段的段地址，SP＝100H，即堆栈段占用 PSP中部分区域。这正是连接时出现 No Stack Segment错误提示时，并不影响程序运行的原因，当然是在用户程序本身对堆栈区要求不大的前提下。

2. COM 文件

COM文件是一种只有一个逻辑段的程序，其中包含有代码、数据和堆栈，大小不超过 64 KB。COM文件存储在磁盘上是主存的完全映像。与 EXE文件相比，其装入速度快，占用的磁盘空间少。DOS装入 COM文件的过程类似于 EXE文件的装入过程，也要建立程序段前缀 PSP，但不需要重新定位，直接将程序装入偏移地址 100H开始的区域，并从 100H处开始执行程序。如图 4.5所示。

由图知：

⑴ 所有段寄存器都指向 PSP的段地址；

⑵ 程序执行起点是 PSP后的第 1条指令，即 IP＝100H，这就要求 COM文件的第一条指令必须在 100H处；

⑶ 堆栈区设在 64 KB物理段尾部 (SP＝0FFFEH),栈底元素为 0;

⑷ COM文件长度存放在 BX:CX寄存器中,例如 BX:CX ＝0000:1000,则长度为 4 KB。

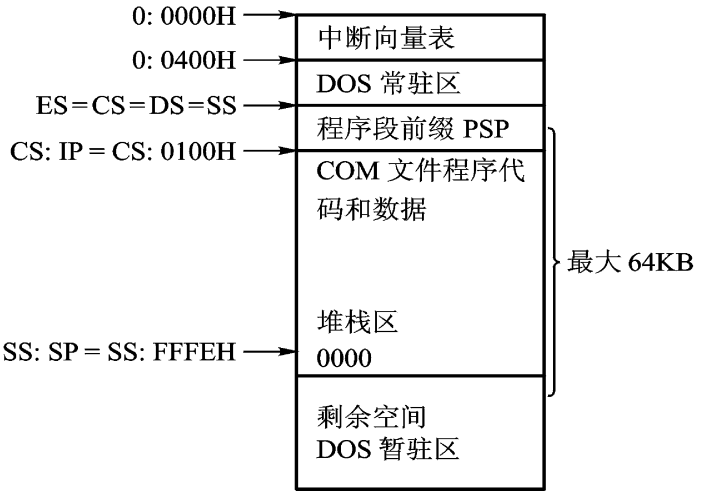


图 4.5 COM文件内存映像

3. 程序段前缀

程序段前缀结构如图 4.6所示。

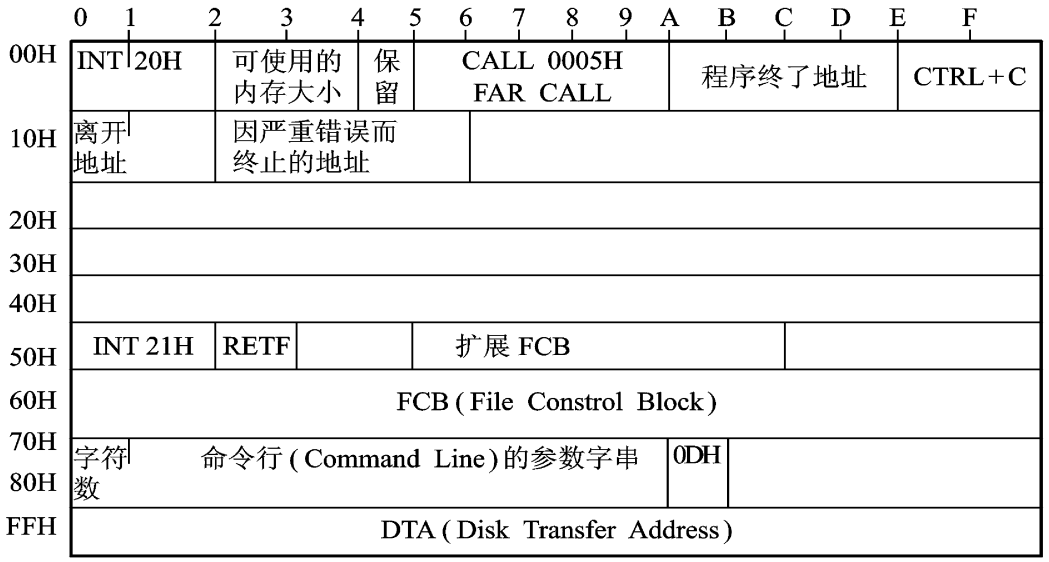


图 4.6 PSP的结构

⑴ 偏移 2DH以前各项,属于为程序运行提供的服务信息,它们包括下列三个方面:

1) 程序结束处理信息

偏移 00H处存放了中断指令 INT 20H (指令代码:CD 20),偏移 0AH处存

放了 INT 22H 的中断向量,利用它们可正常返回 DOS或主程序。

2) 程序中间停止处理信息

偏移 0EH 处存放了 INT 23H 的中断向量,处理 Ctrl + C操作,偏移 12H 处存放了 INT 24H 的向量,处理出现的严重错误。

3) 环境块参数地址

指示环境块参数地址所存段地址,借以取得恢复 COMMAND.COM 暂驻部分和 PATH路径信息。

(2) 在偏移 2EH 以后的区域为文件服务信息,内容如下:

1) 格式化的 FCB (文件控制区)

5CH ~7FH 处设置传统文件访问方式中使用的文件控制块 FCB结构。由于 FCB具有一定的格式,称为格式化区。

2) 非格式化区

80H ~0FFH,有两种用途。一是存放本文件名和路径字符串;二是作为系统默认的磁盘传输区 DATA。它们均无固定的格式,称为“非格式区”。通用于存储最初命令行 (COMMAND LINE) 的参数的原副本。

4.2.4 汇编语言和操作系统 MS-DOS的接口

为了保证应用程序执行完后,能回到 DOS,可使用如下两种方法。

1. 标准方法

本章开头程序示例中即采用了此方法。具体如下:

(1) 将应用程序的主程序定义成一个 FAR过程,其最后一条指令为 RET;

(2) 在代码段的主程序的开始部分用三条指令,把 PSP中 INT 20H 指令的段地址 (CS=DS=ES) 及偏移地址 (0) 压入堆栈:

```
PUSH    DS
MOV     AX,0
PUSH    AX
```

这样,程序执行到主程序的最后一条指令 RET时,由于过程具有 FAR属性,故存在堆栈内的两个字分别弹出到 IP及 CS,便执行 INT 20H 指令,返回到 DOS。

2. 用 DOS功能调用 4CH

在用户程序中不定义过程段,只在代码段结束之前,增加两条语句:

```
MOV     AH,4CH
```

```
INT     21H
```

内中断 21H是操作系统向用户提供服务程序的窗口，4CH号功能服务用于结束用户程序，操作系统收回 CPU的控制权。类似功能有 INT 20H。

4.3 汇编语言程序设计方法及应用

4.3.1 概述

1. 程序设计步骤

从具体问题到编程解决问题，经过如下几个步骤：

⑴ 分析问题，抽象出描述问题的数学模型。

⑵ 确定解决问题的算法或算法思想。

⑶ 绘制流程图或结构图。

⑷ 分配存储空间及工作单元（包括寄存器）。

⑸ 编写程序。

⑹ 静态检查。

⑺ 上机运行调试。

【例题 4.2】 编程实现在 100个无符号字节整数的数组中找出最大数。

⑴ 分析问题

分析问题包括找出已知条件、问题特点、问题中的规律并归纳出数学模型。本例中，有 100个整数的数组，可以把它看作一个集合，记作 {S}。在此集合中找出最大数，记为 max{S}。当然有些问题不一定非要写出数学模型，或者根本写不出数学模型。但是，当有了一个数学模型之后，就可使用很多行之有效的计算方法。

⑵ 确定解决问题的算法或算法思想

确定解决问题的算法或算法思想，指根据人们在解决实际问题时的逻辑思维中的常规推理，去找算法。如果已有数学模型，可以直接或间接利用一些现有的计算方法。

本例中，若是由人工的方法在一组数据中找出最大数。首先从第一个数据开始，一边看一边比较两个数的大小，看到较大数就记下，将较小数丢掉。再将此较大数与下一个数进行比较，始终将比较大的数记下，一直将数组中的数两两

比较完毕,就能找到最大数。根据这个人工思维的过程,可归纳算法为:

首先,建立一个数据指针指向数据区的首地址,将第一个数取入某个寄存器中(例如 AL),与一个数相比较,若下一个数比较大,就将它取到 AL中,替换掉原来的数;然后调整数据指针,将 AL的数与此指针所指的数进行比较,再重复上述步骤,两两比较下去,直到比较完毕,结果 AL中留下最大数。

(3) 绘制流程图或结构图

流程图是程序算法的图形描述,即用图形的方式把解决问题的先后次序和程序的逻辑结构直观地、形象地描述出来,使解题思路清晰,有利于理解和编制程序,还有利于修改程序和减少错误等。对于一个复杂的问题,可以画多级流程图,先画出粗框图,再逐步求精,画出细框图。

本例的程序流程图比较简单,如图 4.7所示。

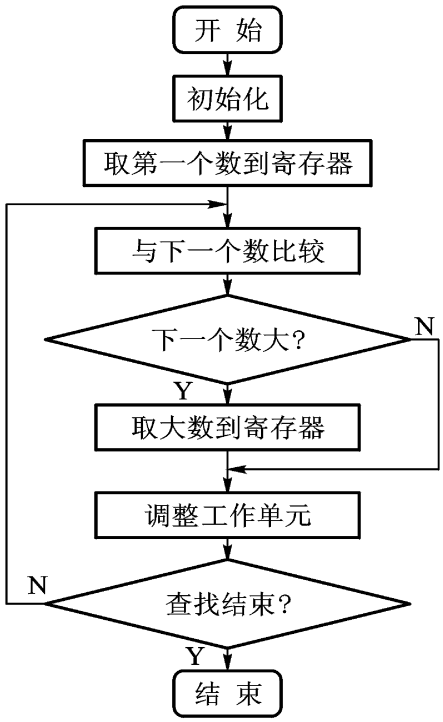


图 4.7　找最大数流程

为了判断查找工作是否结束,可以在初始化部分设计一个计数器,初值是总的比较次数,每比较一次,计数器减 1,当计数器减到零时,比较任务就完成。初始化部分还应包括建立一个指针,由它指向数据区。每比较一次,指针就修改,指向下一个数据单元。开始时,指针指向数据区的首地址。

(4) 分配存储空间及工作单元(包括寄存器)

8088/8086存储器结构要求存储空间分段使用。因此,要分别定义数据段、

堆栈段、代码段以及附加段。工作单元即可以设置在数据段或附加段中的某些存储单元,也可以设置在 CPU内部的数据寄存器中。

本例中,把数组存放在数据段中,并设置 50个字节堆栈空间。利用 BX寄存器作数据指针,用 CX作计数器,用 AL暂存较大数。

⑸ 编写代码

根据流程图和确定的算法逐条语句地编写程序,注意不同的机器有不同的指令系统。使用 8088/8086指令系统,按程序流程图编写出程序如下:

```
DSE       SEGMENT                          ;定义数据段
   ARRAY    B   X1,X2……
   COUNT    DB  $ -ARRAY                    ;数据个数
DSEG      ENDS
SSE       EGMENT  PARA STACK 'STACK'       ;定义堆栈段
   SDAT     DB        50 DUP (?)
   TOP      EQU       LENGTH SDAT
SSEG      ENDS
CSEG      SEGMENT                          ;定义代码段
          ASSUME  CS:CSEG,DS:DSEG,SS:SSEG
MAIN      PROC  FAR                        ;定义过程
START:     USH    DS                       ;为返回 DOS做准备
          MOV    AX,0
          PUSH   AX
          MOV    AX,DSEG
          MOV    DS,AX                    ;设置 DS
          MOV    AX,SSEG
          MOV    SS,AX                    ;设置 SS
          MOV    AX,TOP
          MOV    SP,AX                    ;设置 SP初值
          MOV    BX,OFFSET ARRAY          ;设置数据区指针首地址
          MOV    CX,COUNT                 ;设置计数器初值
          DEC    CX                       ;设置比较次数
          MOV    AL, [BX]                 ;取数入 AL
AGAIN:    INC    BX                       ;指向数据区下一个数据
          CMP    AL, [BX]                 ;两数比较
          JAE    NEXT                     ;AL≥ [BX],转 NEXT
```

```
        MOV     AL, [BX]        ;否则,把较大数取入 AL
NEXT:   DEC     CX              ;全部数据比较完否?
        JNZ     AGAIN           ;否,转 AGAIN
        RET                     ;全部比较完毕,返回 DOS
MAIN    ENDP
CSEG    ENDS
        END     START
```

⑹ 静态检查

静态检查包括:看程序是否具有所要求的功能;程序是否清晰易读;选用指令是否合适;程序语法和格式上是否有错;指令中引用的符号名、标号名和变量名是否定义正确;程序执行流程是否符合算法和流程图;适当考虑字节数要少,执行速度要快等。容易产生错误的地方要重点检查,如比较次数就是数组数据个数减 1,因为第一次比较是连续取出两个数,以后才是取出一个数,比较一次。

⑺ 上机运行调试

静态检查可以发现一些问题,但毕竟还受到主观因素的影响。只有在机上运行通过并且结论正确的程序,才是正确的程序。事实上,即使一个非常有经验的程序员,也不能说程序一次编写就成功,特别是一些复杂的程序,必须经过调试、修改、再调试、再修改,反复多次之后,才能得到一个比较满意的程序。

2. 结构化程序设计

为了获得一个结构良好,易于阅读和易于维护的程序,应当在设计中严格遵守结构化程序设计的要求。结构化程序设计的主要观点可概括为:

⑴ 程序质量标准"清晰第一,效率第二"

结构化程序设计理论认为,随着计算机运算速度的提高和内存价格的下降,相对而言,软件的设计和维护成本在系统中所占的比重越来越大。因此,过分追求节省时间和空间而采用若干小技巧,以致破坏程序易读性和易维护性的方法是不足取的。保持良好的程序结构,应当作为程序设计的首要考虑。

⑵ 程序设计过程"自顶向下,逐步求精"

采用'自顶向下'的设计方法,首先把问题分解为几个大的相对独立的部分,再对每个部分进一步细分,直到成为一个个功能相对简单和易于理解的小问题。

⑶ 三种基本结构

三种基本结构指:顺序、分支、循环,如图 4.8。结构化程序设计思想认为,

只需使用这三种基本结构就可以编写出任何形式的程序。

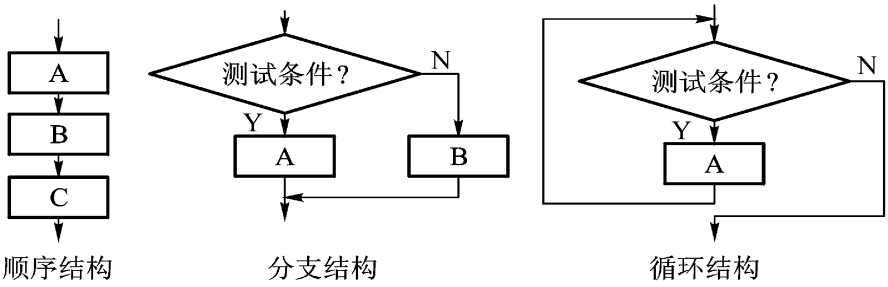


图 4.8 三种基本结构

4.3.2 顺序结构程序设计

顺序结构程序又称简单程序。这种结构的程序中无程序跳转指令、无循环指令，所有指令按其书写顺序逐条顺序执行，程序的执行路径从上到下只有一条。因此，顺序程序的设计只要依照步骤写出相应的指令即可。

【例题 4.3】 将存储单元 A中两个压缩 BCD数拆成两个非压缩的 BCD码，低位 BCD数存入 C中，高位 BCD数存于 B中，并将对应的 ASCII码存入 C1及 B1中。

分析：将一个字节中的两位 BCD码分开，可采用屏蔽高 4位、保留低 4位的方法，得到低位 BCD码；由未组合的 BCD码转为 ASCII码，只要高 4位加 30H或“OR 30H”；若将高 4位 BCD码分离，可以采用逻辑右移 4位的方法，左边移入 4个 0，即可达到目的。流程图见图 4.9。

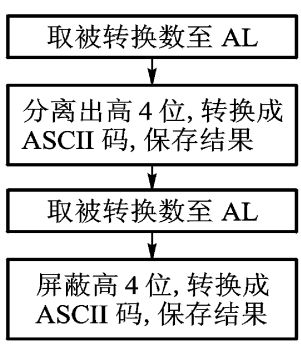


图 4.9 BCD数转换为 ASCII

程序段如下：

```
D TA      DEGMENT                    数据定义
```

```
A       B   37H                 被转换数
B       DB  ?                   ;安排结果保存空间
C       DB  ?
B1      DB  ?
C1      DB  ?
DATA    ENDS
CODE    SEGMENT
        ASSUME CS:CODE, DS:DATA
MAIN    PROC FAR
START:  PUSH    DS
        MOV     AX , 0
        PUSH    X               ;保存返回地址
        MOV     AX, DATA
        MOV     DS, AX          ;设置 DS
        MOV     AL, A
        MOV     CL, 4
        SHR     AL, CL
        MOV     B, AL
        OR      AL, 30H         ;取高 4位,转换成 ASCII码
        MOV     B1, AL          ;保存高 4位的转换结果
        MOV     AL, A
        AND     AL, 0FH
        MOV     C, AL
        OR      AL, 30H         ;取 4位,转换成 ASCII码
        MOV     C1, AL          ;保存低 4位的转换结果
        RET                     ;返回 DOS
MAIN    ENDP
CODE    ENDS
END     START
```

4.3.3 分支程序设计

所谓分支程序指程序的运行过程中,要求计算机做出一些判断,并根据判断选择不同的处理。分支程序分为双分支 (图 4.10a 4.10b)、多分支 (图 4.10c)

两种结构,多分支可转换成多个双分支的结构 图 4.10d)。

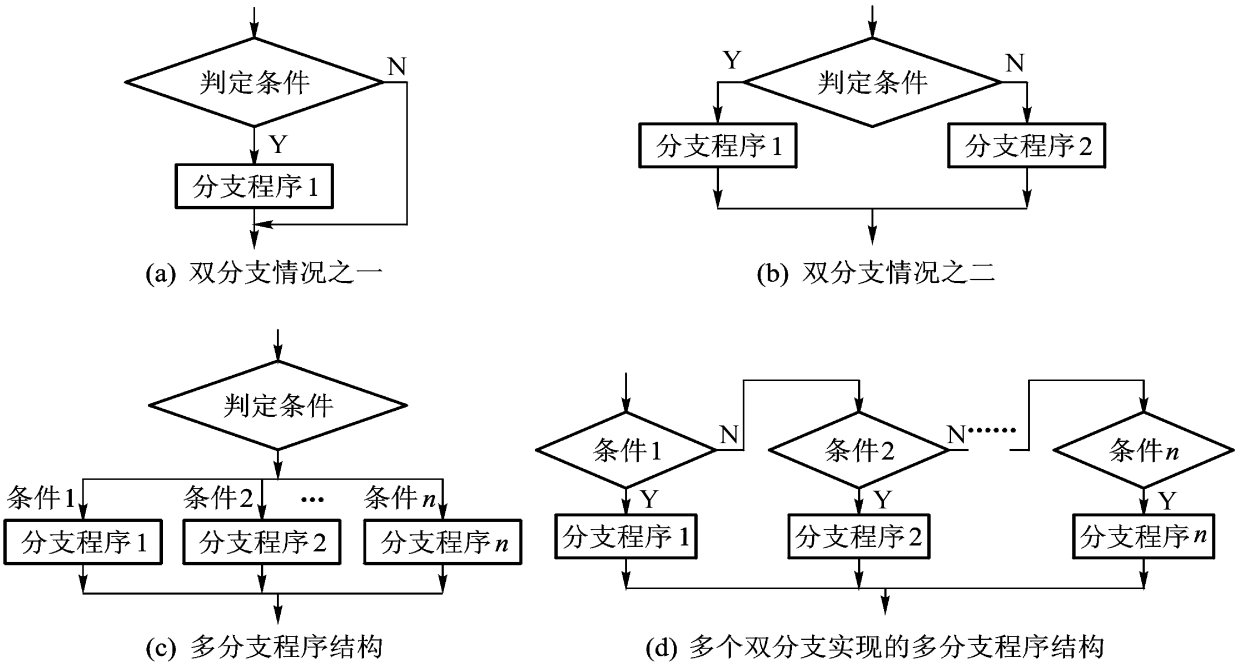


图 4.10 分支程序结构

在 8088/8086 指令系统中,判断的依据主要是运算结果及运算标志寄存器中的状态位。实现跳转的指令是无条件跳转指令:JMP label与条件跳转指令:Jcc short-label,详见 3.4.5。

1. 双分支程序实现

双分支程序由以下 5部分组成:

⑴ 产生条件的语句

在进行判定以前,必须要有产生条件的语句,如算术运算指令、比较指令、移位指令等能影响标志寄存器状态位 ZF、CF、OF、SF等,为下面条件测试做准备。

⑵ 条件测试语句

利用条件转移指令进行测试。

⑶ 定向

根据判断的条件,决定程序的不同走向,在两种可能转移方向中择其一。

⑷ 顺序控制

无论选择哪个方向,都必须保证执行分支段程序后,程序顺序执行下去。

⑸ 标号

在分支程序段前设置标号,以便于转移。

双分支程序典型结构如图 4.11所示。

```
S1:  ……
     ……      ; 程序段1, 产生条件
     ……
     Jcc  BRANCH ; 条件测试, 并定向
     ……
     ……      ; 分支程序段1
     ……
     JMP  S2 ; 顺序控制
BRANCH: ……
     ……      ; 分支程序段2
     ……
S2 : ……
```

图 4.11　双分支程序组成

【例题 4.4】　比较两个无符号数的大小,把大数存入 MAX单元。流程图见图 4.12。

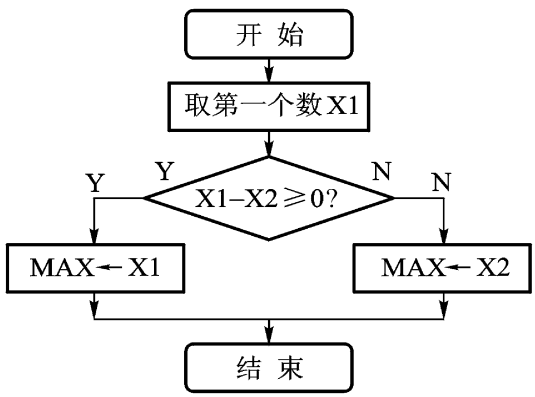


图 4.12　找出两数中较大数

程序如下：

```
DSEG        SEGMENT
NUMBER      B   X1,X2              ;X1和 X2为两个无符号数
MAX         DB  ?
DSEG        ENDS
CSEG        EGMENT
            ASSUME  CS:CSEG,DS:DSEG
            MOV     AX,DSEG
            MOV     DS,AX
```

```
        MOV     AL, NUMBER          ;取第一个数 X1
        CMP     AL, NUMBER+1        ;与第二个数 X2比数
        JNC     BRANCH              ;若 X1≥X2,转 BRANCH
        MOV     AL, NUMBER+1        ;否则,第二个数为较大数
BRANCH: MOV     AX, AL              ;保存较大数
        MOV     AH, 4CH
        INT     21H
CSEG    ENDS
        END
```

2. 多分支程序设计

多分支结构中,条件不止一个,这些条件之间存在两种情况:一是所有条件中只有一个条件成立,称“相异性条件”;二是这些条件中可能有两个以上甚至所有条件都成立,称“相容性条件”。

对于相异性条件,要保证各个条件所对应的分支程序段可执行性的互斥,每一次程序运行,只能选择其一运行;对于相容性条件,必须给出一个判断条件成立与否的次序,按次序判别条件,一旦找到成立的条件后,就执行相应的程序段,其他次序在后的条件是否成立,就不进行判断了。

从多分支程序结构可知,设计多分支程序的关键是如何按条件对多分支进行判断,从而根据不同的条件,转移到不同的入口去执行,常用方法有三种:转移表法、地址表法、逻辑分解法。

(1) 转移表法

转移表法设计思想是:把转移到分支段程序的转移指令依次存放在一张表中,这张表称为“转移表”,如图 4.13。各分支转移指令在表中的位置(常取指令距表首的位移量)作为条件,根据条件即位置选择跳转指令,如图 4.14。

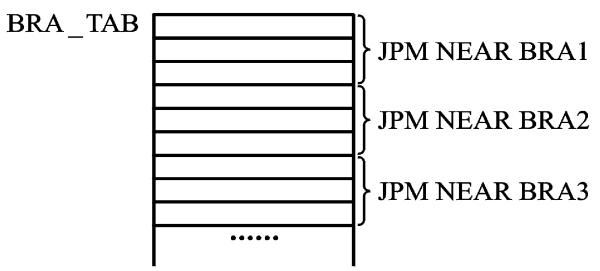


图 4.13 跳转表示意

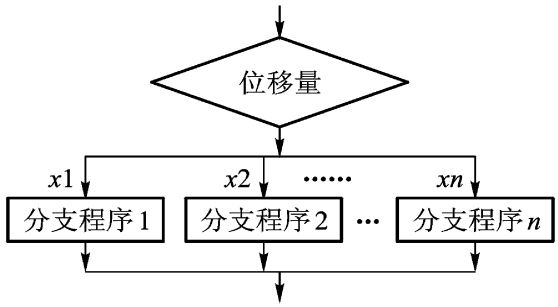


图 4.14　转移表实现多分支

【例题 4.5】　有一监控程序,有 10个控制命令键,如启动、执行、复位等,按下任一命令键相当于发出一条键盘命令,而这些命令的实现由监控程序中 10个子程序完成,这些子程序的入口地址为 ADR0～ADR9。据此建立跳转表,在表内存放 10条跳转指令, JMP ADR0～ JMP ADR9。设 10个命令键编号分别为 0～9。

跳转表已知的情况下,关键问题是要求跳转指令在跳转表中的地址,即计算表地址。表地址 =表基地址 +偏移量。表基地址即为跳转表的首地址,偏移量即对应的跳转指令在表中的地址与表基地址的距离。如果以按键号 X为条件,因为每条跳转指令长度为三个字节,则按键 X对应的命令子程序的跳转命令在跳转表中的偏移量是 3X。设命令键的编号 X已送入寄存器 AL,则实现转向相应命令子程序的程序如下:

```
MOV     AH, 0
MOV     BL, AL
ADD     AL, AL                  ;求 2X
ADD     AL, BL                  ;求 3X,即位移量
MOV     BX, OFFSET BRA_TAB      ;设 BRA_TAB为表首
ADD     BX, AX
JMP     BX
```

⑵ 地址表法

如果把上述跳转表中的跳转指令换成各分支程序的入口地址,即为地址表。跳转指令中,采用间接寻址方式,从地址表中找到分支入口地址而实现的多分支程序,即为地址表法。

【例题 4.6】　某工厂有 8种产品的加工程序 R0～R7分别存放在以 ADR0～ADR7为首地址的内存区域,这 8个首地址的偏移量连续存放在以 BASE

为首地址的地址表内,如图 4.15所示。

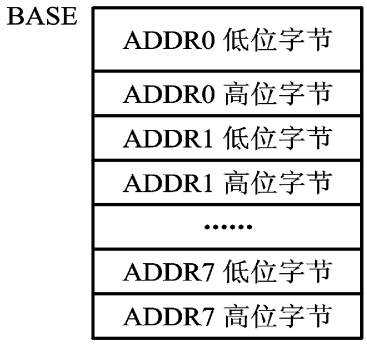


图 4.15 跳转表

从图知,偏移量 =产品编号 ×2,表地址 =表基地址 +偏移量。

据此可写出程序清单:

```
DSEG    SEGMENT
BASE    W    ADR0, ADR1, ADR2, ADR3
        DW   ADR0, ADR1, ADR2, ADR3
BN      DB   ?
DSEG    ENDS
STACK   SEGMENT PARA STACK 'STACK'
        DB   100 DUP (?)
STACK   ENDS
CSEG    SEGMENT
        ASSUME  CS:CSEG, DS:DSEG, SS:STACK
START   PROC  FAR
BEGIN:  PUSH    DS
        MOV     AX, 0
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX
        MOV     SI, BN              ;取产品编号
        ADD     SI, SI              ;计算偏移量
        MOV     BX, BASE [SI]       ;计算表地址
        JMP     BX                  ;分支跳转
START   ENDP
```

```
CSEG    ENDS
        END BEGIN
```

(3) 逻辑分解法

所谓逻辑分解法,就是采用逻辑等效的方法将多分支结构按条件的先后依次分解成一串双分支结构,然后使用双分支的实现方法进行程序设计。此方法多用于相容性条件的多分支程序。

【例题 4.7】　若寄存器 AL中存放了当前外部中断请求的情况。AL中的每一位 D_7 ～D_0 对应一个中断源的中断请求,共 8个 INT_7 ～INT_0。若有中断请求 INT_i,对应位 D_i 为 1,无中断请求,对应位为 0,并设某一刻系统只能响应一个中断请求。由于系统中可能会同时出现多个中断请求 (即 AL中同时有多位为 1),因此必须对中断请求设置优先级。假定优先级顺序与数据位低到高的顺序一致,即 D_0 上中断请求最高,D_7 上中断请求最低,则处理流程如图 4.16所示。

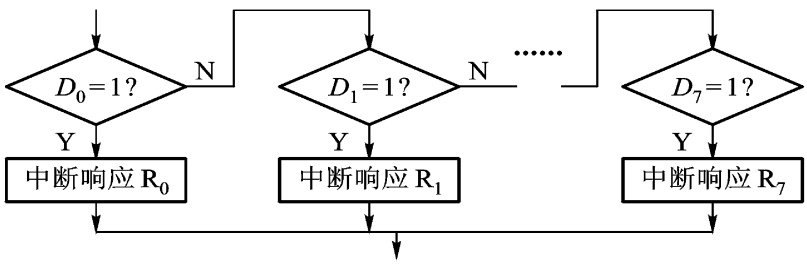


图 4.16　多个双分支实现的多分支程序结构

在进行条件测试时,通过对寄存器 AL移位,依次检测各位,程序为:

```
……
ROR     AL,1
JC      R0
ROR     AL,1
JC      R1
ROR     AL,1
JC      R2
……
ROR     AL,1
JC      R7
……
```

4.3.4 循环结构程序设计

1. 循环程序的组成

循环指程序段在一定条件下重复执行。循环程序由 5个部分组成：

⑴ 初始化部分

这是循环的准备部分,为程序操作、地址指针、循环计数、结束条件等设置初始值。

⑵ 循环体,包括以下 3个部分：

1) 循环工作部分——这是循环程序的主体,完成程序的基本操作,循环多少次,这部分语句就执行多少次；

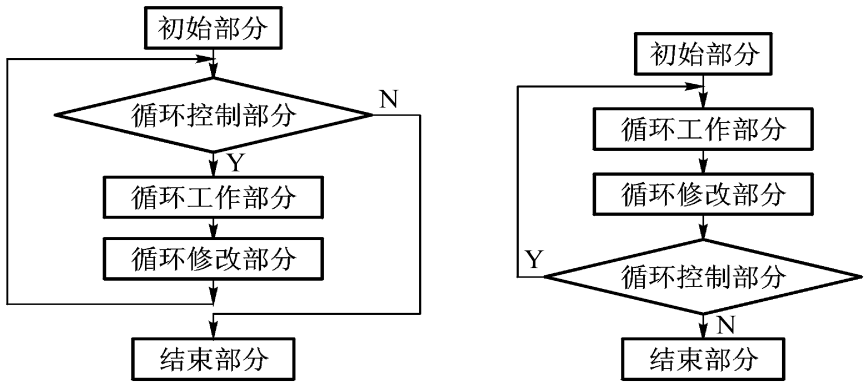
2) 循环修改部分——修改循环工作部分的变量地址等,保证每次重复时,参加执行的数据能发生有规律的变化；

3) 循环控制部分——保证循环条件满足时进入循环；循环结束条件不满足时,退出循环,执行循环体外的后续语句。

⑶ 循环结束部分

完成循环结束后的处理,如数据分析、结果的存放等。

典型的循环程序结构如图 4.17。



(a) 0 次循环程序结构　　(b) 非 0 次循环程序结构

图 4.17 循环程序典型结构

图 4.17a所示的循环结构执行时,先判断循环条件是否满足。当循环条件满足时,执行循环体；当循环条件不满足时,执行结束部分。这种结构的循环,有可能循环体一次也得不到执行,称为“零次或先判断后执行循环”结构。

图 4.17b所示的循环结构执行时,循环体至少执行一次后,才判别循环是否

结束。这种结构的循环,称为“非零次或先执行后判断”循环。

设计循环程序时,建议一般采用以下几个步骤:

1) 分析问题,确定是采用零次循环结构还是非零次循环结构;

2) 根据循环变化的规律,确定适合于循环设计的工作部分;

3) 考虑循环参数的修改部分,分析并确定参数的每次修改方法;

4) 设置循环控制部分及循环参数的置初始值部分。

【例题 4.8】　统计字符串 STRING中空格的个数源程序如下 (分号后的数字为行号)。

```
DSEG    SEGMENT                          ;1
STRING  DB 'Where there is a will, DB there is a way. $ '     ;2
RESULT  DW  ?                            ;3
DSEG    ENDS                             ;4
CSEG    SEGMENT                          ;5
        ASSUME DS:DSEG, CS:CSEG          ;6
START:  MOV     AX, DATA                 ;7
        MOV     DS , AX                  ;8
        MOV     BX, OFFSET STRING        ;9
        MOV     DX, 0                    ;10
NEXT:   MOV     AL, [BX]                 ;11
        CMP     AL, '$ '                 ;12
        JZ      FIN                      ;13
        CMP     AL, 20H                  ;14
        JNZ     CONT                     ;15
        INC     DX                       ;16
CONT:   INC     BX                       ;17
        JMP     NEXT                     ;18
FIN:    MOV     RESULT, DX               ;19
        MOV     AH, 4CH                  ;20
        INT     21H                      ;21
CSEG    ENDS                             ;22
        END     START                    ;23
```

本程序结构分析如下:

1) 数据定义 (1～4)

① 被处理的字符串 (2)

② 预留结果保存空间 (3)

2) 初始化部分：(7～10)

① 设置段地址 (7、8)

② 采用寄存器间接寻址方式，地址初值赋 BX (9)

③ 累加计数器 DX，初值为 0 (10)

3) 循环体与控制部分 (11～18)，如图 4.18所示

4) 结束部分 (19～22)

① 保存结果 (19)

② 返回到 DOS (20、21)

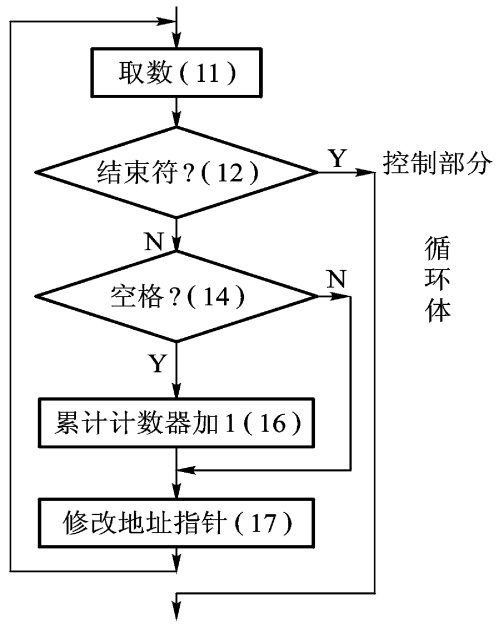


图 4.18 循环体及控制部分流程

2. 循环程序控制方法

(1) 计数控制法

当循环体重复执行次数已知或可求的情况下，可以采用计数方法控制循环。计数控制法分为正计数和倒计数法。

1) 正计数法：将计数器的初值设置为 0，每执行一遍循环，计数器加 1，然后与规定的循环次数比较，若相等，则结束循环，否则继续循环。

2) 倒/负计数法：先将计数器的初值设置为循环次数，每执行一遍循环体后，计数器减 1，并测试是否为 0，当减为 0时，结束循环，否则继续循环。

【例题 4.9】 在数据段以 BUF为首址的区域中，存放了 COUNT个字节数据。编一程序分别统计负数和正数的个数送 MINUS和 PLUS单元。

以正计数法为例，流程图如图 4.19所示，源程序如下：

```
DATA    SEGMENT
BUF     DB -32,25,36, -18, -64,0, -3
COUNT   EQU $ -BUF
PLUS    DB      ?
MINUS   DB      ?
DATA    ENDS
CODE    SEGMENT
        ASSUME CS:CODE, DS:DATA
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     BL, 0
```

```
        MOV     DL, 0
        MOV     SI, OFFSET BUF
        MOV     CX, 0
LOP1:   MOV     AL, [SI]
        CMP     AL, 0
        JGE     NEXT0
        INC     BL
        JMP     NEXT1
NEXT0:  INC     DL
NEXT1:  INC     SI
        INC     CX
        CMP     CX, COUNT
        JL      LOP1
        MOV     MINUS, BL
        MOV     PLUS, DL
        MOV     AH, 4CH
        INT     21H
CODE ENDS
END START
```

若采用负计数法,循环计数器 CX初值为 COUNT,每循环一次,计数值减 1,直至减为 0,退出循环。流程图如图 4.20所示。源程序略。

(2) 条件控制法

当只知道进入或结束循环的条件,而无法知道循环次数时,可采用条件控制法。典型情况有两种:

1) 精度:对于数值求解问题,有时无法得到精确解。通常的方法是,在迭代计算过程中,把本次迭代结果与前一次迭代结果相比,当符合精度要求时,结束迭代循环。

2) 设置结束标志:在数组和表格处理问题中,在数据表格末尾设置一结束标志,作为循环结束条件。例题 4.11统计字符串中空格个数,即采用了此方法。

(3) 逻辑尺控制法

所谓逻辑尺,指一串标志位组成的值,一种标志对应一种操作。以逻辑尺中标志位为控制条件实现的循环,称“逻辑尺控制法”。

[例题 4.10] 设在数据组 X和 Y中各存在有 10个数据元素。试编写程

序计算：

Z1＝X1＋Y1、Z2＝X2＋Y2、Z3＝X3－Y3、Z4＝X4－Y4、Z5＝X5－Y5

Z6＝X6＋Y6、Z7＝X7－Y7、Z8＝X8－Y8、Z9＝X9＋Y9、Z10＝X10＋Y10

并把结果存入数组 Z中。

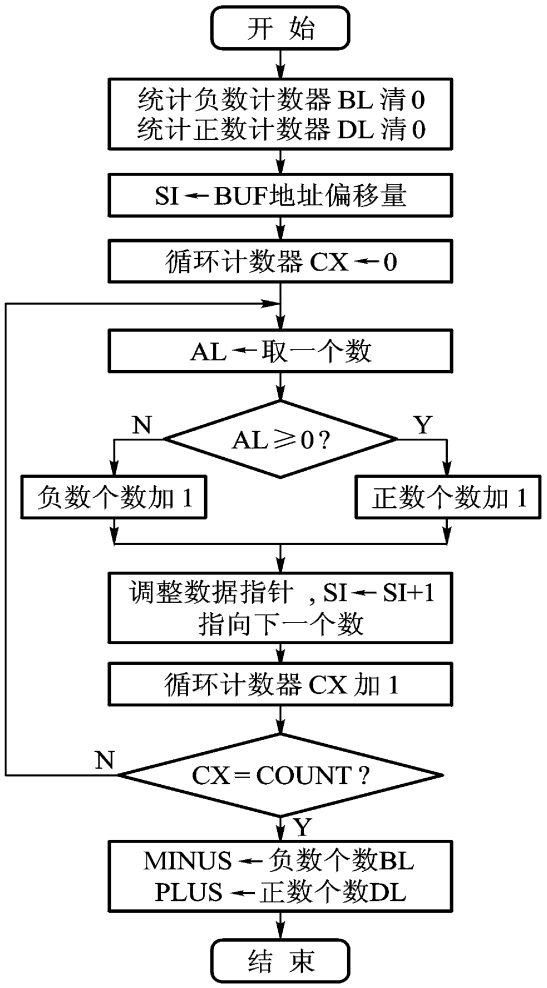


图 4.19 正计数法

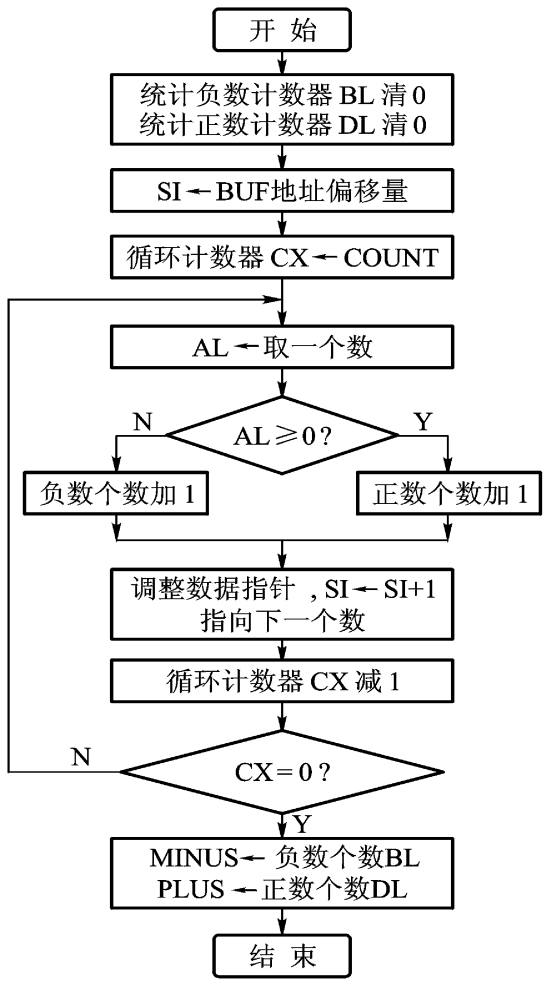


图 4.20 负计数法

分析：由于需要把数组 X和 Y中的 10个对应元素分别进行处理，所以，已知循环次数为 10次，每次循环的操作数可以顺序取出，但所作的处理却不同。为了区别每次应作的处理，可以用一个内存单元设置相应的标志，如标志为 0做加法，标志为 1做减法。在循环程序中，只要测试标志位就能确定应做的操作。这种存放逻辑操作标志的单元称为“逻辑尺”。在本例中，需要进行 10次操作，应设立 10个逻辑位，其逻辑尺存放在 LR单元中，内容为 0000 0000 1101 1100，从低位开始所设的标志表示每次要做的操作。最高 6位没有意义，可设为 0。程序流程如图 4.21所示。

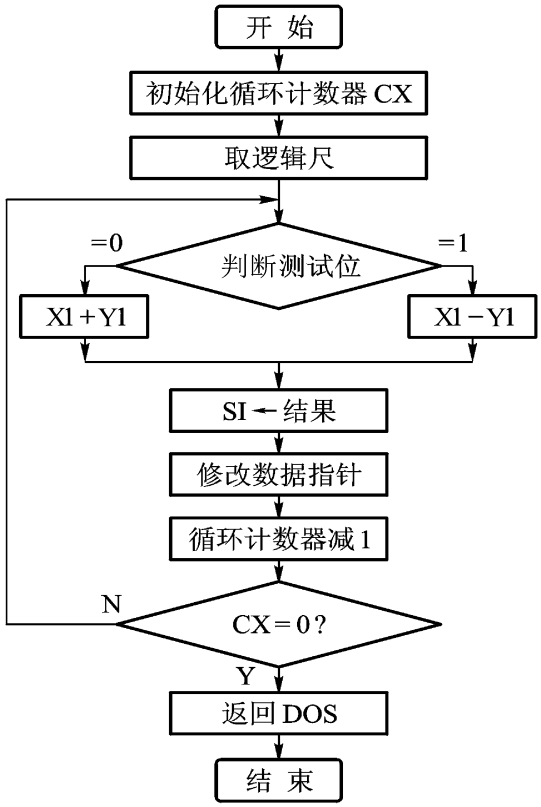


图 4.21　逻辑尺控制的循环

源程序如下：

```
DSEG    SEGMENT
X       DW X1, X2, X3, X4, X5, X6, X7, X8, X9, X10
Y       DW Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Y9, Y10
Z       DW 10 DUP (?)
LR      DW 00DCH
DSEG    ENDS
CSEG    SEGMENT
        ASSUME CS:CSEG, DS:DSEG
MAIN    PROC FAR
        PUSH    DS              ;DS:0000入栈
        SUB     AX, AX
        PUSH    AX
        MOV     AX, DSEG        ;设置数据段
        MOV     DS, AX
        MOV     CX, 10          ;设置移位次数
```

```
        MOV     DX, LR              ;取逻辑尺
        SUB     BX, BX              ;下标清零
NEXT:   MOV     AX, X [BX]
        SHR     DX, 1               ;测试逻辑尺第 i位
        JC      SB
        ADD     AX, Y [BX]
        JMP     SHORT RESULT
SB:     SUB     AX, Y [BX]
RESULT: MOV     Z [BX] , AX
        ADD     BX, 2               ;修改下标
        LOOP    NEXT                ;循环
        RET
MAIN    ENDP
CSEG    ENDS
        END MAIN
```

(4) 开关控制法

如果在一个循环结构中包含有若干个循环体,每一个循环体都对应一个条件,当满足某一条件后就执行对应循环体。这种结构非常类似于多路分支结构。进行这种循环结构的程序设计时,常用开关控制法。所谓开关控制法,是在第一次进入公共循环体之前,预置第一次循环时的开关方向,即指向这一次应执行的对应的循环体。进入该循环体执行完毕在退出本次循环前,又预置好下一个开关方向,以便下一次执行完公共循环体后,按照开关设置进入对应循环体执行。同时应根据设置的循环条件来控制整个循环执行。

【例题 4.11】 设在某一个位移量测试系统当中,需通过位移传感器循环检测 16个位移量。每次检测到的位移量通过 A/D转换后获得的数值在 -128～+127之间。现需对位移量进行线性补偿。设已有三个补偿子程序 BS1, BS2, BS3。设计规划前 5次测试值的补偿由子程序 BS1完成,后 8次补偿由 BS2完成,最后 3次补偿则由 BS3完成。试用开关控制法编程。

分析:显然 16次测量值按 5、8、3的次数进行补偿,即补偿的次数是已知的,如何用开关控制法来编制这一循环程序呢?首先设置一个开关,一开始置开关初值指向补偿子程序 BS1,当循环 5次后再把开关控制转向 BS2。当循环 8次后,再将开关控制转向 BS3。再循环 3次后退出循环,也就完成了 16个位移值的补偿。程序流程图见图 4.22。

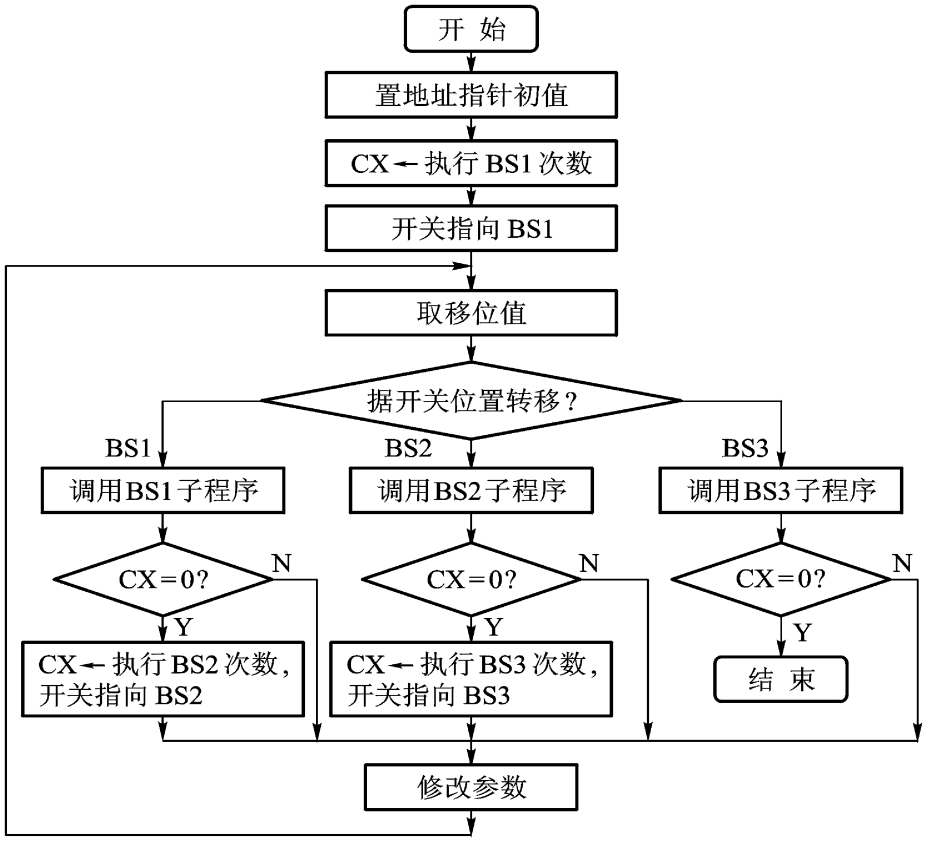


图 4.22　开关控制法

源程序：

```
DATA    SEGMENT
TDATA   DB  -65, 78, 127, -88, 12…          ;位移值存放地址
BDATA   DB  16 DUP (?)                      ;补偿值存放地址
DATA    ENDS
STAK    SEGMENT PARA  STACK 'STACK'
        DB  100 DUP (?)                     ;定义堆栈区
STAK    ENDS
CODE    SEGMENT
        ASSUME CS:CODE, DS:DATA, SS:STAK
START:  MOV     AX, DATA
        MOV     DS, AX
        MOV     CX, 5                       ;置执行 BS1五次
        MOV     SI, OFFSET TDATA            ;SI←位移值存放首址
        MOV     DI, OFFSET BDATA            ;DI←补偿值存放首址
```

```
        MOV     BX,OFFSET SUBROT1       ;开关指向 BS1
LOP1:   MOV     AL,[SI]                 ;取一位移值
SWTH:   JMP     BX                      ;开关方向转移
SUBROT1:CALL    BS1                     ;调用 BS1
        MOV     [DI],AL                 ;存补偿值
        LOOP    NEXT                    ;循环执行 BS1五次
        MOV     CX,8                    ;置执行 BS2八次
        MOV     BX,OFFSET SUBROT2       ;开关指向 BS2
        JMP     NEXT                    ;转修改参数
SUBROT2:CALL    BS2                     ;调用 BS2
        MOV     [DI],AL                 ;存补偿值
        LOOP    NEXT                    ;循环执行 BS2八次
        MOV     CX,3                    ;置执行 BS3三次
        MOV     BX,OFFSET SUBROT3       ;开关指向 BS3
        JMP     NEXT                    ;转修改参数
SUBROT3:CALL    BS3                     ;调用 BS3
        MOV     [DI],AL                 ;存补偿值
        LOOP    NEXT                    ;循环执行 BS3三次
        MOV     AH,4CH
        INT     21H                     ;返回系统
NEXT:   INC     SI                      ;修改取数地址
        INC     DI                      ;修改存数地址
        JMP     LOOP1                   ;继续补偿下一位移值
CODE    ENDS
        END     START
```

在程序中选用 CX作为计数器分三次共进行 16次循环计数。选用 SI作为地址偏移量存储器指针,DI作为补偿后的地址偏移量存储区指针。用 BX作为开关,存放调用对应子程序指令的地址。

4.3.5 子程序设计

1. 概述

如果在一个程序中的多个地方或在多个程序中都要用到同一段程序,可以

把该程序段独立出来,以供其他程序调用,这段程序称为“子程序”或“过程”。子程序是供其他程序调用的相对固定的程序段,调用子程序的程序体,称为“主程序”或“调用程序”。

采用子程序结构,具有以下优点:

⑴ 简化了程序设计过程,使程序设计时间大量节省;

⑵ 缩短程序的长度,节省计算机汇编源程序的时间和程序的存储单元;

⑶ 增加程序的可读性,便于程序修改。

在程序设计中,一个程序可以调用某个子程序,该子程序可以调用其他子程序,这就形成了子程序嵌套,如图 4.23。子程序嵌套调用的层次不受限制,其嵌套层数称“嵌套深度”。

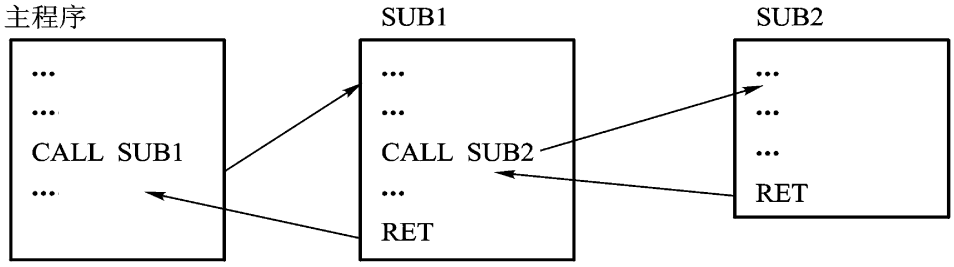


图 4.23　子程序嵌套示意

子程序嵌套调用对程序设计没有什么特殊要求。由于子程序中使用堆栈来保护断点,堆栈操作的“后进先出”特性能自动保证各个层次子程序断点的正确入栈和返回。在嵌套子程序设计中应注意寄存器的保护和恢复,避免各层子程序之间寄存器冲突。特别是在子程序中使用 PUSH、POP指令是要格外小心,以免造成子程序无法正确返回。

2. 子程序设计方法

⑴ 过程定义

过程定义伪指令见 4.1.2。

如果主程序和子程序在同一代码段中,采用如下结构:

```
……
CODE        SEGMENT
            ASSUME    ……
MAIN        PROC   FAR
            ……
            CALL   SUB1
```

```
                ……
                RET
        SUB1    PROC   NEAR
                ……
                CALL   SUB2
                ……
                RET
            SUB2  PROC   NEAR
                ……
                RET
            SUB2  ENDP
        SUB1    ENDP
    MAIN        ENDP
    CODE        ENDS
```

主程序 MAIN 嵌套子程序 SUB1,子程序 SUB1 嵌套子程序 SUB2。因为 MAIN 和 SUB1、SUB2 都在同一代码段,因此除主程序 MAIN 定义为 FAR 属性外,两个子程序 SUB1、SUB2 均定义成 NEAR 属性。

如果主程序与子程序不在一个代码段,采用如下结构:

```
……
CODE1   SEGMENT
        ASSUME  ……
MAIN    PROC    FAR
        ……
        CALL    SUB
        ……
        RET
MAIN    ENDP
CODE1   ENDS
CODE2   SEGMENT
SUB     PROC    FAR
        ……
        RET
SUB     ENDP
```

```
CODE2    ENDS
```

子程序 SUB和主程序不在同一段中,因此子程序 SUB定义成 FAR属性,这样 CALL指令和 RET指令都是 FAR属性。

⑵ 子程序的现场保护和现场恢复

所谓‘现场保护’指子程序运行时,对可能破坏的主程序用到的寄存器、堆栈、标志位、内存数据的保护。所谓‘现场恢复’指由子程序结束运行返回主程序时,对被保护的寄存器、堆栈、标志位、内存数据的恢复。保护现场和恢复现场的工作可在主程序中完成,也可在子程序中完成。若在子程序中完成,一般在子程序开始处实施保护,在子程序结尾处实施恢复。若在主程序中完成,可在主程序调用子程序前实施保护,从子程序返回后再恢复现场。通常,在主程序中保护现场,就一定在主程序恢复现场 在子程序中保护现场,则一定在子程序恢复现场。这样的安排,程序结构清楚,不易出错。常利用堆栈和空闲的存储区实现现场保护和现场恢复。

1) 利用堆栈实施现场保护和现场恢复

利用堆栈实施现场保护和现场恢复,在保护时把需保护的对象入栈,恢复时把被保护的对象出栈,所用指令有 PUSH、POP、PUSHF、POPF等。

【例】 在子程序中,对寄存器 AX、BX、CX、DX的保护与恢复。

```
SUB1     PROC     NEAR
         PUSH     AX
         PUSH     BX
         PUSH     CX
         PUSH     DX
         ……
         POP      DX
         POP      CX
         POP      BX
         POP      AX
         RET
SUB1     ENDP
```

注意 堆栈操作的先进后出的特点。

2) 利用空闲的存储区实施现场保护和现场恢复

利用空闲的存储区实施现场保护和现场恢复,保护时利用数据传递指令把

主程序所占用的寄存器的内容保存到指定的内存单元，恢复时再用数据传递指令从指定的内存单元取回到对应的寄存器中。这种方法使用时不太方便，故较少使用。

【例】 在子程序中，利用内存对寄存器 AX、BX、CX、DX 的保护和恢复。

```
BUFFER    DW  20 DU  (?)                ;为实施保护预留空间
……
SUB       PROC    NEAR
          MOV     DI, OFFSET BUFFER
          MOV      [DI] , AX
          MOV      [DI+2] , BX
          MOV      [DI+4] , CX
          MOV      [DI+6] , DX
          ……
          MOV     SI, OFFSET BUFFER
          MOV     AX,  [SI]
          MOV     BX,  [SI+2]
          MOV     CX,  [SI+4]
          MOV     DX,  [SI+6]
          RET
SUB       ENDP
```

注意：与利用堆栈的方法相比，这里没有保护与恢复顺序问题。

(3) 子程序的参数传递方法

主程序在调用子程序时，经常需要向子程序传递一些参数或控制信息，子程序执行完成后，也常常需要把运行的结果返回给调用程序，这种调用程序和子程序之间的信息传递，称为“参数传递”。参数传递的主要方法有：寄存器传递、内存变量传递和堆栈传递。传递的内容如果是数据本身，称为“值传递”；如果是数据所在单元的地址，称为“地址传递”。

1) 用寄存器传递参数

寄存器进行参数传递时，先由主程序将需要传递的参数预先装入约定的寄存器，进入子程序后，由子程序取出进行处理，执行子程序后，结果也放入规定的寄存器中带回主程序。这种方法十分方便，但由于受寄存器数量的限制，不适合有大量参数的场合。

【例题 4.12】 软件延时子程序及其调用。

所谓“软件延时”,指通过 CPU执行指令需要耗费一定时间的特点实施的延时,常用减 1循环来实现。循环的层次及循环次数,由延时时间长短及 CPU的运行速度决定。对于 8088 CPU,主频 4.77 MHz,每个时钟周期为:1/4.77 MHz = 0.21 μs。循环指令 LOOP,当 CX不为零时,执行循环转移分支,占用 17个时钟周期;当 CX为零时,退出循环,占用 5个时钟周期。如果 CX初值是 2801时,执行指令 WAIT1:LOOP WAIT1,所需时间为:

$$(0.21 \times 2801 + 0.21 \times 5)\ \text{ms} \approx 10\ \text{ms}$$

```
;主程序
CSEG    SEGMENT
        ASSUME CS:CSEG,…
MAIN    PROC    FAR
        PUSH    DS
        SUB     AX, AX
        PUSH    AX
        ……
        MOV     BX,10           延时 100ms,参数赋值
        CALL    DELAY           调用延时子程序
        MOV     BX,55           延时 550ms,参数赋值
        CALL    DELAY           调用延时子程序
        ……
        RET
MAIN    ENDP
;————————————————————————————————
;子程序:DELAY
;功能:实现软件延时,延时单位时间为 10ms
;入口参数:BX,延时常数,实际延时时间为:10* BX (ms)
;出口参数:无
;————————————————————————————————
DELAY   PROC    NEAR
        PUSH    BX              ;现场保护
        PUSH    CX
WAIT0:  MOV     CX, 2801        ;内循环次数,大小由单位时间定
WAIT1:  LOOP    WAIT1           ;延时 10ms
        DEC     BX
```

```
        JNZ     WAIT0
        POP     CX
        POP     BX
        RET
DELAY   ENDP
CSEG    ENDS
        END     MAIN
```

上述子程序用两层循环实现延时,内循环实现单位时间(10 ms)延时,外循环次数由寄存器 BX 定。BX 作为主程序与子程序之间的参数,其值由主程序根据延时多少来定。如第一次延时 100 ms,赋值 10,第二次延时 550 ms,赋值 55。

2) 用存储单元传递参数

当需要传递的参数较多时,由于寄存器数量有限,这时可以考虑在内存中开辟专门的区域用于参数传递。主程序和子程序按照事先的约定,在指定的存储单元中进行数据交换。这种方式所能传递的参数数量几乎没有限制,其缺点是需要一定数量的存储单元,由于任何程序在任何时刻都可以修改这些数据,不利于模块化信息的隔离。实现方式有两种,一是主程序与子程序在同一模块,两者均可直接访问数据区的数据;二是通过地址传递,实现参数内容的传递。第一种形式容易理解,下面举例说明第二种形式的使用。

【例题 4.13】 采用第二种形式,实现无符号数组的求和。

```
;主程序
DSEG    SEGMENT
ARRAY   DW      1,2,3,4,5,6,8,9,0
COUNT   EQU     ($ -ARRAY)/2
SUM     DW      ?
DSEG    ENDS
CSEG    SEGMENT
        ASSUME  CS:CSEG,DS:DSEG
MAIN    PROC    FAR
        PUSH    DS
        SUB     AX,AX
        PUSH    AX
        MOV     AX,DSEG
```

```
        MOV     DS,AX
        LEA     SI,ARRAY
        MOV     CX,COUNT
        CALL    FAR PTR SUM_W
        MOV     SUM,AX
        RET
MAIN    ENDP
;------------------------------------------
;子程序:SUM_W
;功能:计算无符号字数组的累加和
;入口参数:在内存数据段 ARRAY数组中,COUNT为数组长度
;出口参数:内存单元 SUM,存放和
;------------------------------------------
SUM_W   PROC    FAR
        JCXZ    EXIT
        PUSH    CX          ;以下二行,完成现场保护
        PUSH    SI
        XOR     AX,AX
        CLC
NEXT:   ADC     AX,[SI]     ;通过地址引用,传递参数
        ADD     SI,2
        LOOP    NEXT
        POP     SI          ;以下二行,完成现场恢复
        POP     CX
EXIT:   RET
SUM_W   ENDP
CSEG    ENDS
        END     MAIN
```

3) 通过堆栈实现参数传递

通过堆栈实现参数传递是先在主程序中把参数或参数地址压入堆栈,然后在子程序中取出使用。利用堆栈实现参数传递的好处是堆栈操作不占用寄存器,并且堆栈单元使用后可自动释放,反复使用。使用时应该注意,返回指令RET应选择带参数形式,即:RET exp,以便返回主程序后弹出提前压入堆栈的

参数,恢复堆栈的原始状态。

【例题 4.14】 利用堆栈进行参数传递,实现一个无符号字数组的和。

```
SSEG    SEGMENT  STACK  'STACK'
        DW 100 DUP (?)
SSEG    ENDS
DSEG    SEGMENT
ARRAY   DW 1,2,3,4,5,6,7,8,10,
COUNT   DW ($ - ARRAY)/2
SUM     DW ?
DSEG    ENDS
CSEG    SEGMENT
        ASSUME CS:CSEG, DS:DSEG, SS:SSEG
MAIN    PROC   FAR
        PUSH   DS
        SUB    AX,AX
        PUSH   AX
        MOV    AX,DSEG
        MOV    DS,AX
        LEA    BX,ARRAY      ;以下 6行,实参入栈
        PUSH   BX
        MOV    BX,COUNT
        PUSH   BX
        LEA    BX,SUM
        PUSH   BX
        CALL   FAR PTR SUM_W
        RET
MAIN    ENDP
;-------------------------------------------------
;子程序 :SUM_W
;功能 :数组元素求和
;入口参数 :堆栈栈底三个字为数组首地址、数组长度存储单元地址、结果存
 储单元地址
;出口参数 :不影响寄存器
;-------------------------------------------------
```

```
SUM_W   PROC   FAR
        PUSH   BP              ;以下 6行现场保护
        MOV    BP, SP
        PUSH   AX
        PUSH   CX
        PUSH   SI
        PUSH   DI
        MOV    SI, [BP+10]     ;以下 3行取实参,完成参数传递
        MOV    CX, [BP+8]
        MOV    DI, [BP+6]
        JCXZ   EXIT
        XOR    AX, AX
NEXT:   ADD    AX, [SI]
        ADD    SI, 2
        LOOP   NEXT
        MOV    [DI], AX
EXIT:   POP    DI              ;以下 4行,现场恢复
        POP    SI
        POP    CX
        POP    AX
        POP    BP
        RET    6
SUM_W   ENDP
CSEG    ENDS
END     MAIN
```

程序运行中,当子程序中 PUSH DI执行后,堆栈最满,堆栈内容如图 4.24。

主程序中压入堆栈的参数,在子程序中通过 BP指针,以相对寻址方式取得。子程序返回时,除了要弹出返回地址外,还应当“丢弃”主程序事先压入堆栈的三个字参数 (6个字节),所以返回指令使用了带立即数的指令格式“RET 6”。

(4) 子程序说明

由于子程序通常是要反复使用或供他人使用的,在设计子程序的同时就应当建立相应的说明文档,清楚地描述该子程序的功能和调用方法,以便于以后正

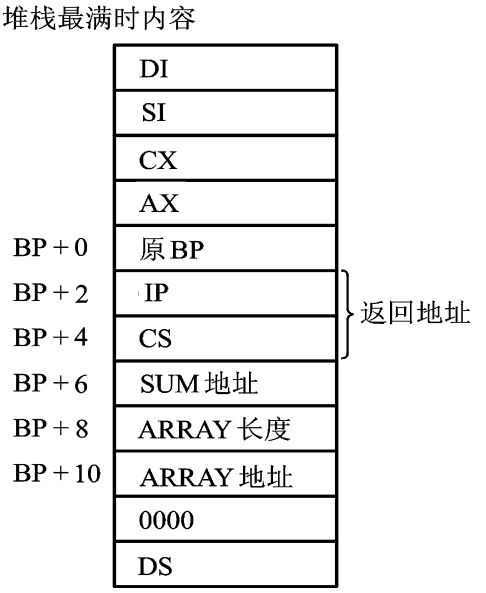


图 4.24 堆栈示意

确使用和维护。前面各例中过程定义头部的注释就是子程序说明文档的一种表示方式。一般来说子程序说明文档应包括以下内容：

1) 子程序名；

2) 功能名：说明该程序完成的具体任务；

3) 入口参数：说明该程序运行所需的参数及其存取方式；

4) 出口参数：说明该程序运行结果及其存取方式；

5) 工作寄存器及存储单元，说明该程序运行中使用寄存器和存储单元占用情况；

6) 作者及最后修改日期：注明程序作者及修改日期以便今后的维护；

7) 示例：必要时，通过具体例子示范参数的使用，并起验证作用。

4.3.6 宏

1. 宏定义、宏调用、宏扩展

(1) 宏定义：宏是源程序中一段有独立功能的程序代码。它只需在汇编语言源程序中定义一次，便可以多次反复调用。调用时只需要一条宏指令即可。

宏定义伪指令格式如下：

宏指令名 MACRO [形式参数表]

```
…… 宏指令体
……
ENDM
```

其中,MACRO是宏指令的开始,ENDM是宏指令的结束,它们之间是宏指令体,是一组具有独立功能的程序代码。形式参数表给出宏定义中所用到的形式参数,参数之间用逗号隔开。

[例题 4.15] 完成 2个压缩 BCD数加法的宏定义。

```
DECADD    MACRO OP1, OP2
      MOV     AL, OP1
      ADD     AL, OP2
      DAA
      MOV     OPR1, AL
      ENDM
```

⑵ 宏调用

在程序中对宏的使用,称为宏调用。宏调用格式为:

宏指令名 [实参数表]

其中,宏指令名是经宏定义伪指令定义的名字,实参数表给出了宏调用中要用到的参数,实参数之间用逗号隔开。实参数是形式参数的实际代替,因此,实参数和形式参数的顺序一致。原则上,实参数的个数应和形式参数个数相等,但汇编程序允许它们不等。

利用上述定义的宏,分别对存放在 8位寄存器或存储单元中的两个压缩 BCD进行加法运算。

```
DECADD    DL, BUFFER
DECADD    AREA1, AREA2
```

⑶ 宏扩展

宏扩展指宏定义体替换宏指令名,并用实参数替换形式参数。当源程序被汇编时,汇编程序对每个宏调用进行宏扩展。当宏调用中实参数个数和形式参数个数不等时,若实参数个数大于形式参数个数,在替换时多余的实参数不予考虑。若实参数个数小于形式参数个数,则多余的形式参数作为空 (字符)或零 (数字)处理。

上述宏调用,宏扩展后,得到以下指令

```
DECADD    DL, BUFFER扩展成：
    +MOV    AL, DL
    +ADD    AL, BUFFER
    +DAA
    +MOV    DL, AL
```

宏扩展后，原来宏定义中的指令前面加上符号“+”，以示区别。

2. 宏指令与子程序的区别

宏指令是用一条指令来代替一段程序以简化程序，而子程序也有类似的功能，那么这两者之间有什么区别？

⑴ 宏指令由宏汇编程序 MASM在汇编过程中进行处理，在每个宏调用处，将相应的宏定义体插入；而调用指令 CALL和返回指令 RET则是 CPU指令。执行 CALL指令时，程序被控制转移到子程序的入口地址处。

⑵ 宏指令简化源程序，但不能简化目标程序，汇编以后，在宏定义处不产生机器代码。但在每个宏调用处，通过宏扩展，在目标程序中插入与重复次数一样的目标代码。因此不节省内存。对于子程序来说，在目标程序中定义子程序的地方将产生相应的机器代码，但每次调用时，只需用 CALL指令，不再重复出现子程序的机器代码，一般来说可以节省内存单元。

3. 宏指令举例

⑴ 形式参数可以是操作数或操作码

```
宏定义：FOO   MACRO   P1, P2, P3
              MOV     AX, P1
              P2, P3
              ENDM
宏调用：FOO     WORD_VAR, INC, AX
宏展开：+MOV    AX, WORD_VAR
        +INC    AX
```

⑵ 形式参数可以是操作码的一部分，但宏定义体中必须用“&”作为连接符

```
宏定义：LEAP  MACRO   COND, LAB
              J&COND  LAB
              ENDM
宏调用：EAP   Z, THERE
```

```
        ……
        LEAP   NZ,HERE
        ……
宏展开：+JZ   THERE
        ……
        +JNZ HERE
        ……
```

“&”是一个操作符，它在宏定义体中可以作为形参前缀，展开时将把“&”前后两个符号合并而形成一个符号，这个符号可以是操作码、操作数或一个字符串。

⑶ 宏指令名可以与指令助记符或伪操作名相同。这时，宏指令的优先级最高，而同名的指令或伪操作则失效了，使用伪操作 PURGE可以在适当的时候取消宏定义，以恢复指令的原始含义。

```
宏定义：  UL      MACRO M1,M2
          MOV     AX,M1
          MOV     BX,M2
          MUL     BX
          ENDM
宏调用：VAR1    DW  345
          VAR2    DW  789
          ……
          MUL     VAR1,VAR2
          ……
宏展开：……
          +MOV   AX, VAR1
          +MOV   BX, VAR1
          +MUL   BX
```

⑷ 宏定义中的标号

由于宏定义体内允许使用标号，如果多次调用该宏定义，则宏展开后的同一标号可能会重复出现，这是不能允许的。这时，可用伪操作 LOCAL建立唯一的地址标号。其格式为：

LOCAL 局部标号 1,……]

其中局部标号表中的各标号之间必须用逗号隔开。汇编过程中会自动用一个唯一的符号（用 ?? 0000～?? FFFF）来代替每个指定的局部标号。

注意：LOCAL必须是 MACRO 后的第一个语句，且在 MACRO 和 LOCAL 之间不能有其他注释或分号标志。

```
宏定义：  BSOL    MACRO    OPR
          LOCAL    NEXT
          CMP      OPR,0
          JGE      NEXT
          NEG      OPR
NEXT:
          ENDM
宏调用：……
          ABSOL    VAR
          ……
          ABSOL    BX
          ……
宏展开：……
          +CMP     VAR,0
          +JEG     ?? 000
          +NEG     VAR
          +?? 000:
          ……
          +CMP     BX,0
          +JGE     ?? 0002
          +NEG     BX
          +?? 0001:
          ……
```

4.3.7 系统功能调用

PC－DOS系统中设置两层内部子程序供用户使用，即 DOS功能模块和基本输入输出子程序 BIOS。它们的入口都安排在中断向量表中，所以，对用户来说可将这些子程序看成是中断处理程序，程序中可以直接调用它们。这就极大地方便了用户对微机系统资源的利用。

1. DOS功能调用

DOS共提供了近 80个功能调用,大致分为:设备管理、文件管理和目录管理等几类,这些子程序通过 BIOS使用设备,从而进一步隐蔽了设备的物理特性及其接口的细节。一般,调用系统功能时总是先采用 DOS层功能模块,如果这层模块内容达不到要求,再进一步选用 BIOS层的子程序。DOS通过 INT 21H为程序员提供一系列的功能调用,作为应用编程接口。DOS功能调用的使用方法是:

⑴ 根据需要的功能调用设置入口参数。

⑵ 把功能调用号送 AH寄存器。

⑶ 发软中断指令“INT 21H”。

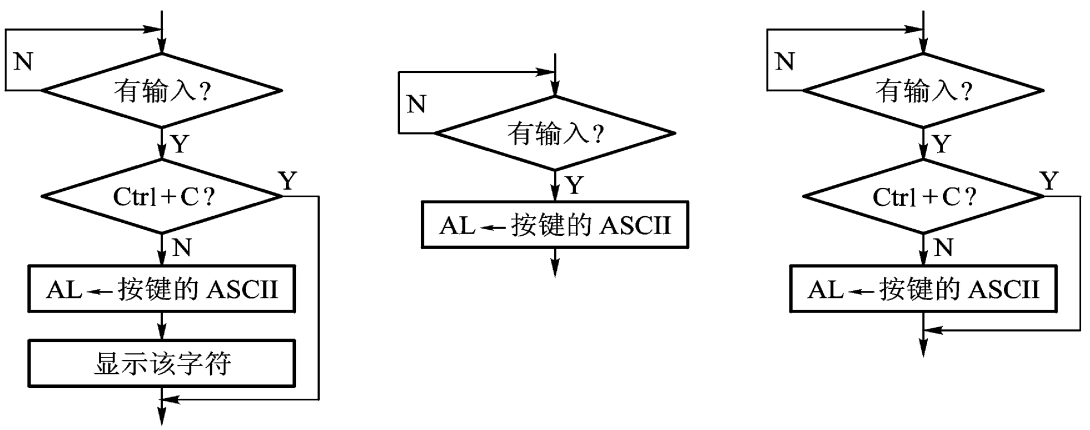
⑷ 可根据有关功能调用的说明取得出口参数。

DOS功能号无需死记,必要时可查阅有关资料。见附录。

⑴ 字符输入

1) 单个字符输入:

1号、7号、8号均能实现从键盘接受 1个字符,区别在于是否检查 Ctrl+Break或 Ctrl+C及字符是否回显。流程见示意图 4.25。



(a) 1号功能操作示意　　(b) 7号功能操作示意　　(c) 8号功能操作示意

图 4.25　1号、7号、8号功能操作流程

【例】　键盘接收一个字符,并回显

```
MOV     AH,1
INT     21H
```

2) 字符串输入

字符串输入,可选用多次单个字符的输入或选用 10号功能。

10号功能的入口参数是存放字符串的缓冲区首地址和最大字符个数。接受缓冲区首地址的是寄存器 DS和 DX,分别存放其段基值和偏移量;缓冲区第一字节置为缓冲区最大容量。

出口参数是输入的字符串及实际输入的字符个数。缓冲区第二字节存放实际读入的字符个数（不包括回车符）;第三字节开始存放接收的字符串。

字符串以回车键结束,回车符是接收到的字符串的最后一个字符。如果输入的字符数超过缓冲区所能容纳的最大字符数,则随后的输入字符被丢失并响铃,直到遇到回车键为止。如果在输入时按 Ctrl+C或 Ctrl+Break键,则结束程序。

【例】 允许最大输入为 100字节的数据定义及功能调用。内存示意见图 4.26。

```
DATA    SEGMENT
BUF     DB    100
        DB    ?
        DB    100 DUP (?)
        ……
DATA    ENDS
CODE    SEGMENT
        ……
        MOV   AX, DATA
        MOV   DS, AX
        ……
        MOV   DX, OFFSET BUF
        MOV   AH, 10
        INT   21H
        ……
CODE    ENDS
```

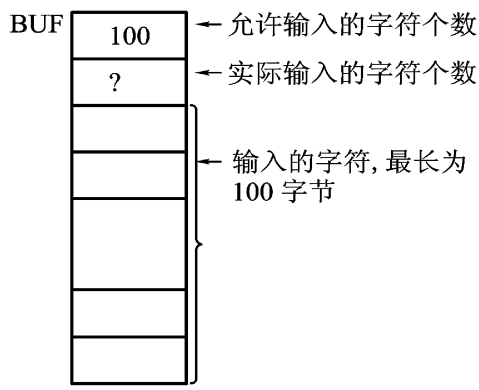


图 4.26 内存示意

(2) 字符输出

1) 单个字符,选用 2号功能

【例】 输出‘2’

```
MOV     DL, '2'
MOV     AH, 2
INT     21H
```

2) 字符串输出

字符串输出,可选用多次单个字符的输出或选用 9号功能。9号功能的入口参数是被输出字符串首址,接受入口参数的是寄存器 DS和 DX,分别存入被输出字符串首址的段基值和偏移量。采用 9号功能输出字符串,要求字符串以“$”结束,该字符作为字符串结束符,不输出。

【例】 输出 STRING中的字符

```
DATA    SEGMEN S
STRING  DB 'Where there is a will, there is a way$'   ;定义字符串
……
DATA    ENDS
CODE    SEGMENT
        ……
        MOV     AX,DATA
        MOV     DS,AX
        ……
        MOV     DX,OFFSET STRING
        MOV     AH,9
        INT     21H
        ……
CODE    ENDS
```

2. BIOS功能调用

BIOS常驻 ROM,独立于 DOS,可与任何操作系统一起工作。它的主要功能是驱动系统所配置的外部设备,如磁盘驱动器、显示器、打印机及异步通讯接口等。通过 INT 10H～INT 1AH向用户提供服务程序的入口,使用户无需对硬件有深入了解,就可完成对 I/O设备的控制与操作。BIOS的中断调用与 DOS功能调用类似。下面作为 BIOS功能调用入门,对键盘 I/O调用 (中断类型 16H)和显示调用 (中断类型 10H)加以介绍。

(1) 键盘输入

当用户按键时,键盘接口会得到一个被按键的键盘扫描码 (见附录),同时产生一个中断请求。如果键盘中断是允许的 (中断屏蔽字中的 bit1为 0),并且 CPU处于中断状态 (IF=1),那么 CPU通常就会响应中断请求,转入键盘中断处理程序。

键盘中断处理程序首先从键盘接口取得代表被按键的扫描码，然后根据扫描码判别用户所按的键并做相应的处理。把键盘上的键简单地分成 5种类型：字符键（字母、数字和符号等），功能键（如 F1和 PgUp等），控制键（Ctrl、Alt和左右 Shift键），双态键（如 Num Lock和 Caps Lock等），特殊请求键（如 PrintScreen等）。键盘中断处理程序对 5种键的基本处理方法如下：

如果用户按的是双态键，那么就设置有关标志，在 AT以上档次的系统上还要改变 LED指示状态。如果用户按的是控制键，那么就设置有关标志。如果用户按的是功能键，那么就根据键盘扫描码和是否按下某些控制键（如 Alt）确定系统扫描码，把系统扫描码和一个全 0字节一起存入键盘缓冲区。如果用户按的是字符键，那么就根据键盘扫描码和是否按下某些控制键（如 Ctrl）确定系统扫描码，并且得出对应的 ASCII码，把系统扫描码和 ASCII码一起存入键盘缓冲区。如果用户按的是一个特殊请求键，那么就产生一个相对应的动作，例如用户按 PrintScreen键，那么就调用 5H号中断处理程序打印屏幕。

键盘 I/O程序以 16H号中断处理程序的形式存在（详见附录），它提供若干功能，每一个功能有一个编号。在调用键盘 I/O程序时，把功能编号置入 AH寄存器，然后发出中断指令“INT 16H”。调用返回后，从有关寄存器中取得出口参数。

【例】 利用 BIOS中断服务，从键盘读一个字符：

```
MOV     AH,0
INT     16H
```

如果键盘缓冲区中有字符，那么中断处理程序就会极快结束，即调用就会极快返回，读到的字符是调用发出之前用户按下的字符。如果键盘缓冲区空，那么要等待用户按键后调用才会返回，读到的字符是调用之后按下的字符。如果程序员出于某种理由，要从键盘取得在调用发出之后用户按下的字符，那么就要先清除键盘缓冲区。下面的程序片段先清除键盘缓冲区，然后再从键盘读一个字符：

```
AGAIN:  MOV     AH,1
        INT     16H             ;判缓冲区空？
        JZ      NEXT            ;空，转
        MOV     AH,0
        INT     16H             ;从键盘缓冲区取一个字符
        JMP     AGAIN           ;继续
NEXT:   MOV     AH,0
```

```
INT     16H             ;等待键盘输入
……
```

当然,程序员也可以通过直接修改键盘缓冲区头指针的方法清除键盘缓冲区,但不鼓励这样做。

(2) 显示器输出

1) 文本显示方式

所谓文本显示方式是指以字符为单位的显示方式。字符通常是指字母、数字、普通符号 (如运算符)和一些特殊符号 (如菱形块和矩形块)。文本显示模式下,显示器的屏幕被划分为 80列 25行,所以每一屏最多可显示 2 000 (80* 25)个字符。用行号和列号组成的坐标来定位屏幕上的每个可显示位置,左上角的坐标规定为 (0,0),向右增加列号,向下增加行号,这样右下角的坐标便是 (79,24)。

2) 显示属性

单色显示屏幕上每个字符在存储中用两个字节表示,一个字节保存字符的ASCII码,另一个字节保存字符的属性。字符属性确定了每个要显示字符的特性。在单色显示时,显示属性定义了闪烁、反相和高亮度等显示特性,如图4.27。

在彩色显示时,属性还定义了前景色和背景色 (如图 4.28)。在彩色显示属性字节中,RGB分别表示红、绿、蓝,I表示亮度,BL表示闪烁。位 0到位 3组合16种前景颜色。亮度和闪烁只能用于前景。当 I位为 1时,表示高亮度,当 I位为 0时,表示普通亮度。当 BL为 1时,表示闪烁,当 BL为 0时,表示不闪烁。表 4.3给出彩色文本模式下的 IRGB组合成的通常颜色。前景颜色和背景颜色一起确定字符的显示效果。

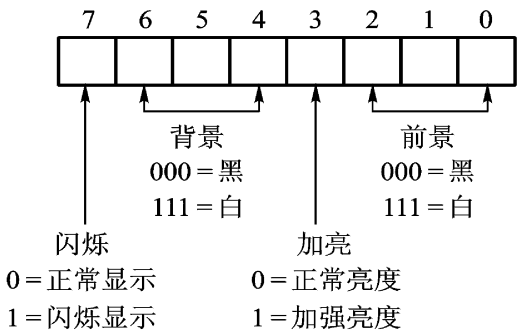


图 4.27　单色显示属性字节

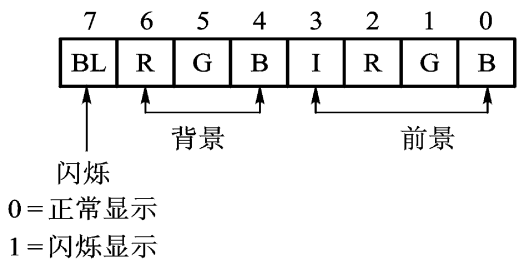


图 4.28　单色显示属性字节

表 4.3 彩色文本模式下的颜色组合

色号	IRGB	颜色	色号	IRGB	颜色
0	0000	黑	8	1000	亮灰
1	0001	蓝	9	1001	亮蓝
2	0010	绿	10	1010	亮绿
3	0011	青（深蓝）	11	1011	亮青
4	0100	红	12	1100	亮红
5	0101	品红	13	1101	亮品红
6	0110	棕	14	1110	黄
7	0111	白	15	1111	亮白

属性值可以有不同组合，表 4.4与表 4.5分别给出单色与彩色典型的属性值。

表 4.4 单色属性典型值

属性值	效　果
0000 0000	不显示
0000 0001	黑底白字，下划线
0000 0111	黑底白字，正常显示
0000 1111	黑底白字，高亮度
0111 0000	白底黑字，反相显示
1000 0111	黑底白字，闪烁
1111 0000	白底黑字，反相闪烁

表 4.5 彩色属性字节的典型值

效　果	BL	R	G	B	I	R	G	B	十六进制值
黑底蓝字	0	0	0	0	0	0	0	1	01
黑底红字	0	0	0	0	0	1	0	0	04
黑底白字	0	0	0	0	0	1	1	1	07
黑底黄字	0	0	0	0	1	1	1	0	0E
黑底亮白字	0	0	0	0	1	1	1	1	0F
白底黑字	0	1	1	1	0	0	0	0	70
白底红字	0	1	1	1	0	1	0	0	74
黑底灰白闪烁字	1	0	0	0	0	1	1	1	87
白底红闪烁字	1	1	1	1	0	1	0	0	F4

3) 显示存储区

显示适配卡带有显示存储器,用于存放显示屏幕上显示文本的代码及属性或图形信息。显示存储器作为系统存储器的一部分,可用访问普通内存的方法访问显示存储器。通常为显示存储器安排的存储地址空间的段值是 B800H 或 B000H,对应的内存区域就称为显示缓冲区。假设段值是 B800H。

文本显示模式下,屏幕的每一个显示位置依次对应显示存储区中的两个字节,这种对应关系如图 4.29所示。

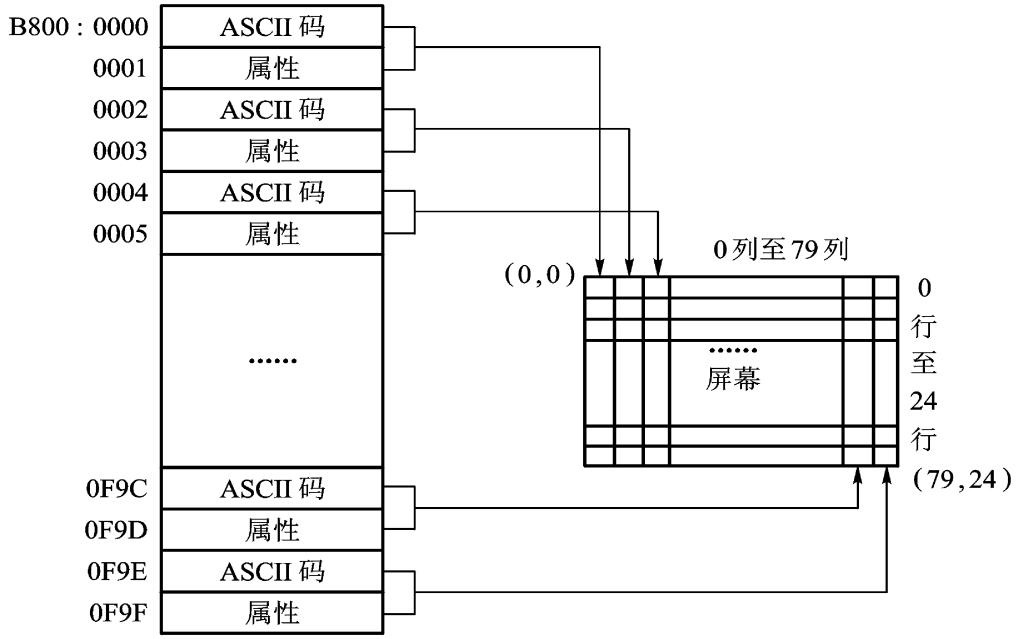


图 4.29　显示存储区与显示位置的对应关系

为了在屏幕上某个位置显示字符,只需把要显示字符的代码及属性填到显示存储区中的对应存储单元即可,这种方法称为“直接写屏”。下面的程序片段属性在屏幕的左上角以黑底白字显示字符“A”:

```
MOV     AX, B800H
MOV     DS, AX
MOV     BX, 0
MOV     AL, 'A'
MOV     AH, 07H
MOV     [BX], AX
```

为了了解屏幕上某个显示位置的字符是什么,或显示的颜色是什么,那么只要从显示存储区中的对应存储单元中取出字符的代码和属性即可。下面的程序片段取得屏幕右下角所显示字符的代码及属性:

```
MOV     AX, B800H
MOV     DS, AX
MOV     BX, (80*24+79)*2
MOV     AX, [BX]
```

④ 显示 I/O程序的功能和调用方法

利用直接写屏方法,程序可实现快速显示。但编程较复杂,并且最终的程序也与显示适配卡相关。所以,一般不采用直接写屏方法,而是调用 BIOS提供的显示 I/O程序。

BIOS提供的显示 I/O程序作为 10H号中断处理程序存在。显示 I/O程序的提供若干功能(详见附录),每一个功能有一个编号。在调用 I/O程序的某个功能时,应根据要求设置好入口参数,把功能编号置入 AH寄存器中,然后发出中断指令"INT 10H"。调用返回后,从有关寄存器中取得出口参数。

现就显示页号作说明。为了支持屏幕上显示 2 000个字符,需要的显示存储器容量为 4 KB。如果显示存储器的容量为 32 KB,那么显示存储器可存放 8屏显示内容。为此,把显示存储器再分成若干段,称为显示页。调用显示 I/O程序的 5号功能,可选择当前显示页。通常总是使用第 0页。

【例】 通过 BIOS INT 10H实现显示

调用 I/O程序的 5号功能,可选择当前显示页。下面的程序片段选择第 0页作为当前显示页:

```
MOV     AL, 0
MOV     AH, 5
INT     10H
```

如果要知道当前显示页号,则可通过调用显示 I/O程序的 0FH号功能,同时可知道当前显示模式和该模式下最大显示列数,下面的程序片段调用 0FH号功能:

```
MOV     AH, 0FH
INT     10H
```

下面的程序片段调用显示 I/O程序的 9号功能,在当前光标位置处显示指

定属性的字符,但不移动光标:

```
MOV     BH,0            ;第 0页
MOV     BL,47H          ;红底白字
MOV     CX,1            ;1个
MOV     AL,'A'          ;字符为 A
MOV     AH,9
INT     10H
```

在窗口滚屏时,如果滚屏行数为 0,就表示清除整个窗口。如果把整个屏幕作为窗口,那么就可清除整个屏幕。下面的程序段设屏幕为 80列,它先清除屏幕,然后把光标移到左上角:

```
……
MOV     CH,0            ;置最左上角坐标
MOV     CL,0
MOV     DH,24           ;置最右下角坐标
MOV     DL,79
MOV     BH,07           ;填充属性
MOV     AL,0            ;清整个窗口
MOV     AH,6
INT     10H             ;实现清屏
MOV     BH,0            ;设第 0
MOV     DH,0            ;置光标定位坐标
MOV     DL,0
MOV     AH,2
INT     10H             ;定位光标
```

4.4 汇编语言程序设计举例

4.4.1 数制和代码转换

[例题 4.16] 将 5位 ASCII码表示的十进制数 (注:不大于 65 535) 转换成

两字节二进制数。

如：'12345'→ 110000 00111001

分析设计：两字节无符号二进制数最大表示范围为 65 535，所以要求被转换的十进制数不大于 65 535。设 5位 ASCII码存放在以 ASDEC为首址的数据区中，转换好的一字节二进制数存放在同一数据区 RESULT单元中，见图 4.30。

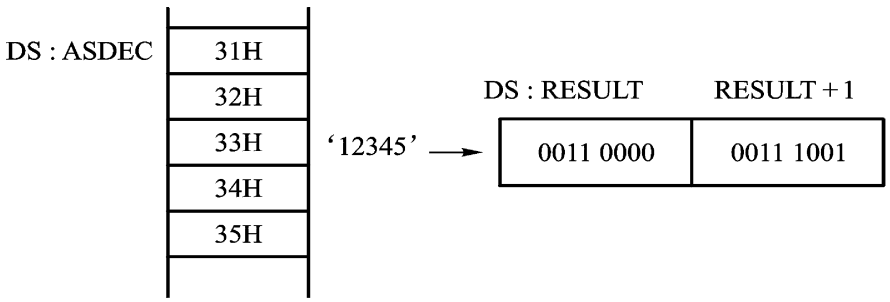


图 4.30 存储示意

算法思想：转换分为两步：

1) 把 ASCII码转换为 BCD数，方法是：减 30H；

2) 将 BCD数转换为二进制数。方法是：

设 5位十进制为 $d_4d_3d_2d_1d_0$，这样表示的数其值为：

$$d_4\times10^4+d_3\times10^3+d_2\times10^2+d_1\times10^1+d_0$$

$$=0\times10^5+d_4\times10^4+d_3\times10^3+d_2\times10^2+d_1\times10^1+d_0$$

$$=(((((0\times10+d_4)\times10+d_3)\times10+d_2)\times10+d_1)\times10+d_0$$

由此可见，设最初部分和 AX为 0，最终结果是做 5次 AX乘以 10＋d→AX的运算。

源程序：

```
DSEG      SEGMENT
  ASDEC     B     '12345'
  COUNT     EQU   $ - ASDEC
  RESULT    DW    ?
DSEG      ENDS
SSEG      SEGMENT  PARA STACK 'STACK'
          DB  10 DUP (?)
SSEG      ENDS
CSEG      SEGMENT
```

```
        ASSUME  DS:DSEG, CS:CSEG, SS:SSEG
START:  MOV     AX, DSEG
        MOV     DS, AX
        MOV     SI, OFFSET ASDEC
        MOV     CX, COUNT          ;循环计数器赋初值
        XOR     AX, AX             ;部分和 AX,初值为 0
AGAIN:  ADD     AX, AX             ;AX ×10
        MOV     BX, AX
        ADD     AX, AX
        ADD     AX, AX
        ADD     AX, BX
        MOV     BH, 0
        MOV     BL, [SI]           ;取 ASCII码,转换成二进制数
        SUB     BL, 30H
        ADD     AX, BX             ;部分和计算
        INC     SI                 ;修改数据指针,指向下一个字节
        LOOP    AGAIN
        MOV     RESULT, AX
        MOV     AH, 4CH
        INT     21H
CSEG    ENDS
        END     START
```

本算法中,用加法代替了 MUL指令,值得借鉴。

【例题 4.17】　将一字节二进制数转换成两位十六进制 ASCII码表示的十六进制数。

如 :01001111B (4FH) → '4F'

分析设计 :设要转换的一字节二进制数存放在数据区 BINA单元,转换后的两位 ASCII码表示的十六进制数存放在同一数据区以 AHEX为起始的两字节单元之中,如图 4.31。

算法思想 :二进制转换成十六进制数是每 4位一组,所以转换分两步 :

1) 把二进制数转换为十六进制数 ;

2) 把十六进制数转换为 ASCII码,方法是 :0～9的数加 30H或者与 30H相或 ;A～F的数加 37H。具体过程见图 4.32。

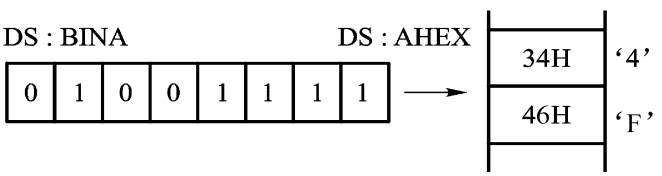


图 4.31 存储示意

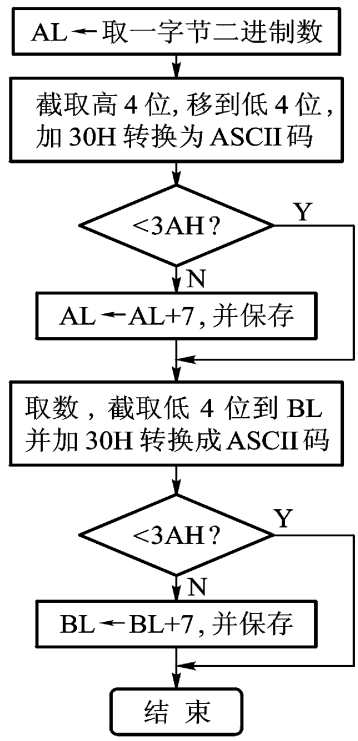


图 4.32 二进制转换成 ASCII

源程序：

```
DSEG    SEGMENT
   BINA    B   10100110B              ;定义一字节二进制数
   AHEX  DB   2 DUP (?)               ;定义结果单元
DSEG    ENDS
CSEG    SEGMENT
   ASSUME   CS:CSEG, DS:DSEG
START:  MOV     AX, DSEG
        MOV     DS, AX
        MOV     DI, OFFSET  AHEX    ;寄存器间接寻址
        MOV     AL, BINA            ;取一字节二进制数
        MOV     BL, AL              ;保存
        AND     AL, 0F0H            ;截取高 4位
        MOV     CL, 4
        SHR     AL, CL              ;移到低 4位
```

```
        ADD     AL, 30H          ;转换成 ASCII码
        CMP     AL, 3AH
        JB      NEXT1            ; <3AH时转 NEXT1
        ADD     AL, 7            ;≥3AH时,再加 7
NEXT1:  MOV     [DI], AL         ;存高位十六进制数的 ASCII码
        INC     DI               ;修改数据指针
        AND     BL, 0FH          ;截取低 4位
        ADD     BL, 30H          ;转换成 ASCII码
        CMP     BL, 3AH
        JB      NEXT2            ; <3AH时转 NEXT2
        ADD     BL, 7            ;≥3AH时,再加 7
NEXT2:  MOV     [DI], BL         ;存低位十六进制数的 ASCII码
        MOV     AH, 4CH
        INT     21H
MAIN    ENDP
CSEG    ENDS
        END     START
```

【例题 4.18】　把 16位二进制数转换成十进制的 ASCII码串。

如 :0000 0101 0001 1100B (1308D) → ‘01308’

分析设计 :设要转换的 16位二进制数存放在数据区 BINA2的字单元中,转换后的 5位十进制 ASCII码串存放在 ADEC为首址的 5个连续字单元中,见图 4.33。

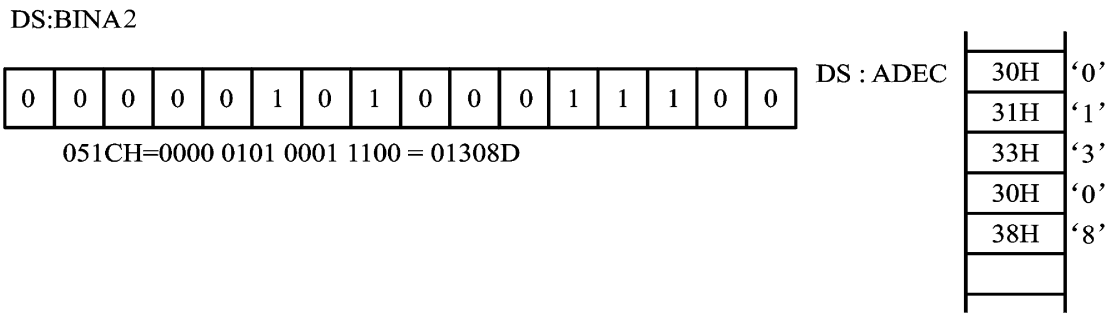


图 4.33　存储示意

算法思想 转换分 2步：

1) 求出与 16位二进制数对应的十进制数的万位、千位、百位、十位、个位上的数；

2) 把各位上的数转换为 ASCII码。实现第一步的方法之一是：通过在原数中分别减万、千、百、十，统计出其中有多少个万、多少个千、多少个百、多少个十，最后减剩下的为个位数。具体过程见图 4.34。

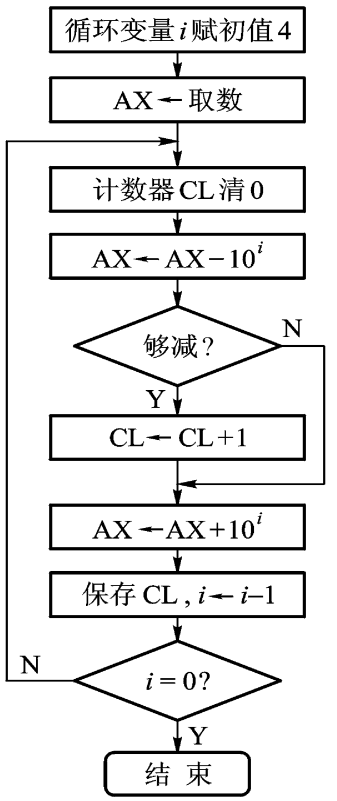


图 4.34 二进制数转换为 BCD

源程序：

```
DSEG        SEGMENT
   BINA2    DW   051CH
   ADEC     DB   5 DUP (?)
   UNIT10   DW   10000,1000,100,10,1
DSEG        ENDS
CSEG        SEGMENT
   ASSUME   CS:CSEG,DS:DSEG
```

```
MAIN    PROC    FAR
START:  PUSH    DS
        XOR     AX, AX
        PUSH    AX
        MOV     AX, DSEG
        MOV     DS, AX
        MOV     DI, OFFSET  ADEC
        MOV     SI, OFFSET  UNIT10
        MOV     AX, BINA2           ;取 16位二进制数
LOP0:   XOR     CL, CL
        MOV     BX, [SI]            ;取 10^i 次数
LOP1:   SUB     AX, BX              ;减 10^i
        JB      NEXT                ;不够减转 NEXT
        INC     CL                  ;够减时计数
        JMP     LOP1                ;重复
NEXT:   ADD     AX, BX              ;不够减时恢复
        ADD     CL, 30H             ;转换成对应位 ASCII码
        MOV     [DI], CL            ;存放结果
        INC     SI
        INC     SI
        INC     DI                  ;修改存放结果地址
        CMP     BX, 1               ;检查是否已到个位数
        JNZ     LOP0                ;不是转 LOP0继续
        RET                         ;处理完返回
MAIN    ENDP
CSEG    ENDS
END     START
```

另一算法是：将初始二进制数分别除以 10000、1000、100和 10，每次的商是一个 BCD码数，即为结果的“万位”、“千位”、“百位”和“十位”，最后一次除法的余数就是“个位”，算法流程如图 4.35。

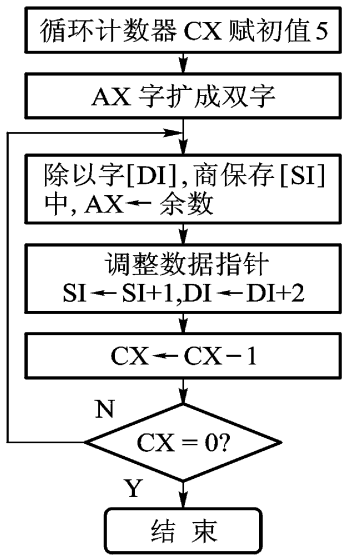


图 4.35 二进制数转换为 BCD

源程序：

```
DSEG        SEGMENT
   UNIT10   DW 10000,1000,100,10,1
   DATA     DW   12310
   BUF      DB   5 DUP (?)
DSEG        ENDS
CSEG        SEGMENT
            ASSUME   CS:CSEG,DS:DSEG
START:      MOV   AX,DSEG
            MOV   DS,AX
            MOV   AX,DATA
            LEA     SI,BUF
            LEA     DI,UNIT10
            CALL    B2BCDW
            MOV     AH,4CH
            INT     21H
;------------------------------------------
;子程序名：B2BCDW
```

```
;功能:16位二进制数转换成压缩 BCD码。
;入口参数:AX=16位二进制,SI=缓冲区首址
;DI=除数首址
;出口参数:缓冲区中存放压强 BCD码
;------------------------------------------
B2BCDW  PROC    NEAR
CONT:   PUSH    CX
        MOV     CX,5
NEXT:   CWD
        DIV WORD PTR [DI]
        MOV     [SI],AL
        INC     SI
        ADD     DI,2
        MOV     AX,DX
        LOOP    NEXT
        POP     CX
        RET
B2BCDW  ENDP
CSEG    ENDS
        END   START
```

4.4.2 BCD数的算术运算

1. 非压缩 BCD数 (ASCII码)加减运算

【例题 4.19】 编写一个 4字节非压缩 BCD码的减法运算。如:SUM←2974 + 7326

分析设计:被加数存于缓冲数据区 FIRST中,加数存于缓冲数据区 SECOND中,结果存于缓冲数据区 SUM,存放原则是低位在低字节。为了避免进位丢失,每个缓冲区开辟 5个字节单元。

算法思想:将 4字节的 BCD码分为 4个单字节数相加,从低字节开始,进行 4次循环操作,注意每次相加后必须进行 BCD修正。

源程序:

```
DSEG    SEGMENT
```

```
FIRST   DB  4,7,9,2,0
SECOND  DB  6,2,3,7,0
THIRD   DB  5 DUP (?)
DSEG    ENDS
CSEG    SEGMENT
        ASSUME CS:CSEG,DS:DSEG
START:  MOV   AX,DSEG
        MOV   DS,AX
        MOV   ES,AX
        LEA   SI,FIRST          ;被加数指针送 SI
        LEA   BX,SECOND         ;加数指针送 BX
        LEA   DI,THIRD          ;和指针送 DI
        MOV   CX,4+1            ;循环计数器赋初值
        CLD                     ;串操作时方向为增
AGAIN:  LODSB                   ;取被加数
        ADD   AL,[BX]           ;对应一位加法运算
        AAA                     ;BCD调整
        STOSB                   ;保存当前位和
        INC   BX
        ADC   BYTE PTR [BX],0   ;进位值加到加数的下一位
        LOOP  AGAIN
        MOV   AH,4CH
        INT   21H
CSEG    ENDP
        END   START
```

2. 压缩 BCD 码加减运算

【例题 4.20】 编写一个 4 字节压缩 BCD 码的加法运算。如：SUM←29385476+17653749

分析设计：被加数存于缓冲数据区 BCD1 中，加数存于缓冲数据区 BCD2 中，结果存于缓冲数据区 SUM，占 4个字节。

算法思想：将 4字节的 BCD码分为 4个单字节数相加，从低字节开始，进行 4次循环操作，注意每次相加后必须进行 BCD修正。

源程序：

```
DSEG    SEGMENT
  BCD1  DB   76H, 54H, 38H, 29H
  BCD2  DB   49H, 37H, 65H, 17H
  SUM   DB   4 DUP (?)
DSEG    ENDS
CSEG    SEGMENT
        ASSUME CS:CSEG, DS:DSEG
START:  MOV    AX, DSEG
        MOV    DS, AX
        LEA    SI, BCD1          ;被加数指针送 SI
        LEA    BX, BCD2          ;加数指针送 BX
        LEA    DI, SUM           ;和指针送 DI
        MOV    CL, 4             ;循环计数器赋初始
        CLC                      ;清进位位
AGAIN:  MOV    AL, [SI]          ;取被加数
        ADC    AL, [BX]          ;被加数加加数
        DAA                      ;BCD 调整
        MOV    [DI], AL          ;保存结果
        INC    SI
        INC    BX
        INC    DI                ;调整数据指针
        DEC    CL
        JNZ    AGAIN             ;所有字节计算未完,转 AGAIN
        MOV    AH, 4CH
        INT    21H               ;结束
CSEG    ENDS
        END    START
```

3. BCD乘法运算

非压缩型 BCD码的乘法和加减法不同,加减法可以直接用 ASCII码参加运算而不管其 ASCII码的高 4位,只要在 ADC或 SBB指令后用一条调整指令 AAA或 AAS就能得到正确的结果。而乘法运算则要求参加运算的乘数、被乘数必须是真正的非压缩型 BCD码,即高 4位为 0,低 4位表示一位十进制数。也就是说当乘数、被乘数用 ASCII码表示的话,则在进行乘法运算之前必须将 ASCII码高

4位清零。然后采用指令系统提供的十进制乘法调整指令 AAM和 MUL指令配合就可完成十进制的乘法运算。

【例题 4.21】 用非压缩型 BCD码完成 57394×8的运算,并在显示器上输出显示。

分析设计:设被乘数从低位到高位以 ASCII码的形式存放在 ADR1为首的数据区中。同样乘数也以 ASCII码形式存放在 ADR2单元中,所得运算后的乘积存放在 RESULT为首的单元中,如图 4.36所示。结果要求在屏幕显示,必须把乘积的二进制形式转换成 ASCII码形式。

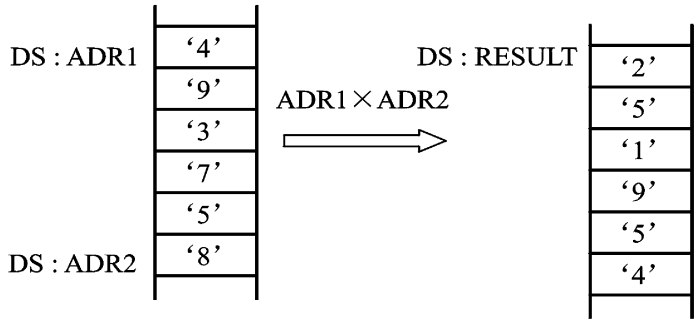


图 4.36 57394×8=459152存储示意图

算法思想:从低位到高位依次把各位 ASCII码转换成 BCD码后与乘数相乘,乘积的低 4位转换成 ASCII码保存,高 4位暂存后加到下一位数的数乘中。这里没有采用求出二进制积后,再转换成 ASCII码,而是根据乘法特点,边乘边把乘积的低 4位进行转换。

源程序:

```
DATA    SEGMENT
  ADR1    DB    '49375'          ;定义被乘数 57394
  ADR2    DB    '8'              ;定义乘数 8
  RESULT  DB    6 DUP (?)        ;定义存放结果单元
DATA    ENDS
CODE    SEGMENT
        ASSUME CS:CODE, DS:DATA, ES:DATA
MMULT:  MOV   AX, DATA
        MOV   DS, AX
        MOV   ES, AX
```

```
        CLD
        MOV   SI,OFFSET ADR1
        MOV   DI,OFFSET RESULT
        MOV   CX,5
        AND   ADR2,0FH              ;将乘数的高 4位清 0
        MOV   BYTE PTR [DI],0
L1:     LODSB                       ;取被乘数中 1位,从低位开始
        AND   AL,0FH                ;将被乘数对应的高 4位清 0
        MUL   ADR2                  ;1位乘
        AAM                         ;十进制数乘法运算调整
        ADD   AL,[DI]               ;加进位
        AAA                         ;十进制数加法运算调整
        OR    AL,30H                ;将结果位换成 ASCII码
        STOSB                       ;保存部分积
        MOV   [DI],AH               ;存放当前运算结果位
        LOOP  L1                    ;重复相乘运算 5次
        OR    BYTE PTR [DI],30H
        MOV   CX,6                  ;输出乘积结果 6位数
L2:     MOV   DL,[DI]               ;从高位到低位开始输出显示
        MOV   AH,02H                ;2号系统功能调用
        INT   21H
        DEC   DI
        LOOP  L2                    ;重复输出显示 6位乘积结果
        MOV   AH,4CH
        INT   21H                   ;返回系统
CODE    ENDS
        END   MMULT
```

以上程序仅讨论了多位被乘数和 1位乘数的乘法运算。通过学习可以很容易地掌握多位被乘数和多位乘数的相乘运算程序的编制。

4. 多字节非压缩 BCD码除法运算

非压缩型 BCD码除法运算与加、减、乘运算不同,其十进制算术除法调整指令 AAD是用在除法指令之前。即十进制数的除法运算是先将 BCD码调整成真

正的二进制数然后再作二进制除法运算。

【例题 4.22】 用非压缩型 BCD码完成 672597÷9的运算,并将结果在显示器上输出显示。

分析设计:设被除数从高位到低位以 ASCII码形式放在数据区以 ADR1为首址的单元中,同样除数也以 ASCII码形式存放在 ADR2单元中,相除后所得到的商也从高位到低位以 ASCII码形式存放在以 RESULT为首址的单元中。如图 4.37。结果在屏幕显示,必须把运算结果转换为 ASCII码。

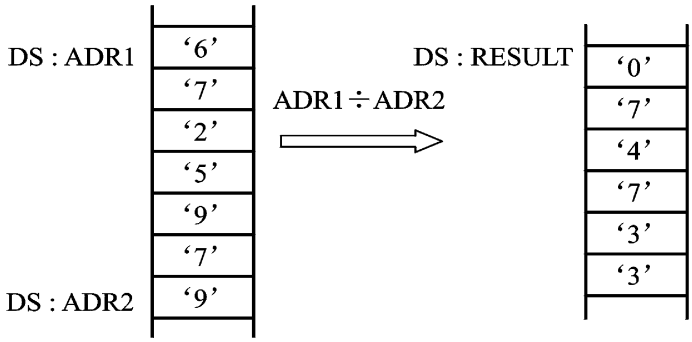


图 4.37 672597÷9=74733存储示意

算法思想:从高位到低位依次将各位非压缩的 BCD码与前一次除法的余数合在一起转换成二进制数,除以除数,商转换成 ASCII码保存,余数作为被除数的十位参与下一次除法运算。

【例】 '672597'÷'9'

1) 余数寄存器 AH=0,除数'9'转换成 9;

2) 被除数高位'6'转换成 6,送 AL;

3) BCD数 AX转换成二进制数,即:AH×10+AL=6,形成被除数;

4) 被除数 AX÷除数,即 06÷9 = 0……6

5) 商 0转换成'0',保存;

6) 取被除数下一位'7'重复 2)~6),直至做完最后一位被除数。

源程序:

```
DATA        SEGMENT
   BUF1     DB     '672597'          ;定义被除数 672597
   BUF2     DB     '9'               ;定义除数 9
   RESULT   DB     6 DUP (?)
DATA        ENDS
```

```
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA, ES:DATA
MDIVD:  MOV     AX, DATA
        MOV     DS, AX
        MOV     ES, AX
        CLD
        MOV     SI, OFFSET BUF1     ;从高位开始除
        MOV     DI, OFFSET RESULT   ;从低位开始存放商
        MOV     CX, 6
        MOV     BL, BUF2
        AND     BL, 0FH             ;将除数高 4位清 0
        MOV     AH, 0
L1:     LODSB                       ;取被除数的 1位,从高位开始
        AND     AL, 0FH             ;将被除数对应的高 4位清 0
        AAD                         ;除法调整将 AX转换为二进制数
        DIV     BL                  ;AX/BL, AL←商, AH←余数
        OR      AL, 30H             ;将对应位商转换成 ASCII码
        STOSB                       ;部分商保存
        LOOP    L1                  ;继续下一位相除运算
        MOV     CX, 6
        MOV     DI, OFFSET RESULT
L2:     MOV     DL, [DI]            ;将商从高位开始输出显示
        MOV     AH, 02H             ;2号系统功能调用
        INT     21H
        INC     DI
        LOOP    L2
        MOV     AH, 4CH
        INT     21H                 ;返回系统
CODE    ENDS
        END     MDIVD
```

4.4.3 表格处理与应用

1. 排序

排序的目的是为检索提供方便，一组数据按照由小到大的顺序排列称为升序，反之则称为降序。排序的方法很多，如交换排序、选择排序、快速排序等，本书主要介绍交换排序。所谓交换排序，就是将相邻的两个数进行比较，如果它们不是按规定的顺序（例如升序）排列，则交换它们的位置，否则就不交换。一种典型的交换排序称为气泡排序法，又称冒泡排序，其基本思想如下：

采用两两比较的方法，先拿第 N个数 d_N 与第 N－1个数 d_{N-1} 比较，若 $d_N > d_{N-1}$ 则不变动；反之，则交换。然后拿 d_{N-1} 与第 d_{N-2} 个数相比，按同样方法决定是否交换，这样一直比较到 d_2 与 d_1 比较。当第一次大循环结束时，数组中最小值冒到顶部。但数组尚未按大小排列好，还要进行第二次大循环，循环结束时，数组中第 2小值也上升到顶部的相应位置……，这样不断地循环下去，若数组的长度为 N，则最多经过 N－1次的大循环，就可以使数组按由小到大的升序排列整齐。在每一次大循环中，两两比较的次数，在第一次大循环时为 N－1，在第二次大循环时为 N－2次，……

【例】 数字：2，－1，6，3，－5的冒泡排序过程如图 4.38（图中黑体字为已排好序的数字，下一轮中不参与排序）：

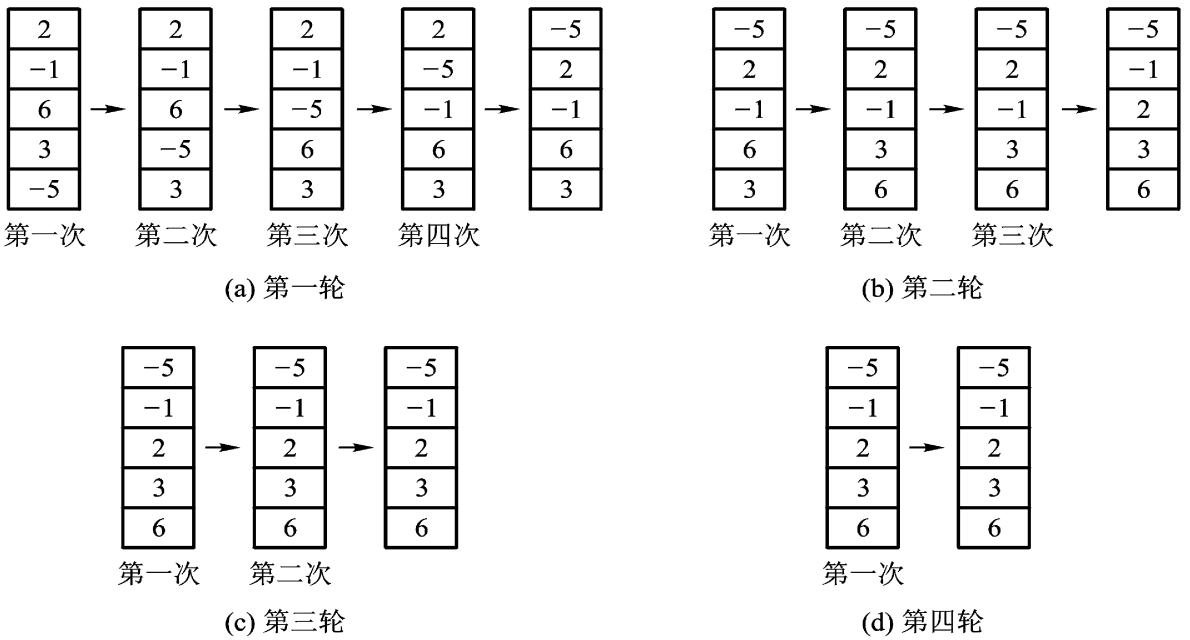


图 4.38 冒泡排序过程

从图知,有的数组不需要经过 N－1次大循环就已经排列整齐了,为此需要设置交换标志,如果在一个大循环中,一次交换都未发生,或只在 d_N 与 d_{N-1}相比较时发生交换,则说明数组在此大循环中已经有序,这样下一次大循环就不再需要。由上述操作过程不难分析出对于 N个数排序的结论:

⑴ 至多需进行 N－1轮比较 (外循环);

⑵ 第 i轮中共需进行 N－ i次比较 (内循环);

⑶ 每次比较时只比较相邻两数,并且将小数变换至上面;

⑷ 每次比较必须由上至下即只向下移动一个位置;

⑸ 排序结束标志是当前一轮比较中无数据交换。

下面来编写一个实用的冒泡排序法程序。

【例题 4.23】　设在内存数据区 TABLE地址开始存放一列表,表长在 LEN单元,表中数据为有符号字节数据,请用冒泡排序法编程将表中数据从小到大排序。

算法设计 根据上面的有关冒泡排序法的介绍,可将排序算法归纳如下:

1) BX←外循环比较 (轮数)计数值;

2) SI←数据表首址偏移量;

3) CX←内循环比较 (次数)计数值;

4) DX←置交换标志初值;

5) 两相邻数据比较若顺序对不交换转 8);

6) 若顺序不对两数据交换位置;

7) DX←交换标志;

8) SI←SI+1修改数据地址;

9) CX←CX－1内循环比较次数计数器减 1,若不为 0转 5);

10) 检查交换标志若为初值,表示已全部排好序转结束;

11) BX←BX－1外循环比较轮数计数器减 1,若不为 0转 2);

12) 结束。

还需说明一下,以下源程序中引入的交换标志 DX,其初值为第 N个数在数据表中的偏移量,在每一轮比较过程中只要发生过数据交换则将被比较数较大下标给 DX。当每一轮比较后,检查该交换标志,若 DX等于初值,说明未发生过交换,或只在最后两个数处发生了一次交换,即有序表已形成,无需继续循环,排序结束,否则,还要进一步排序比较。

源程序:

```
DATA      SEGMENT
   TABLE  DB      12, 87, －51, 68, 0, 15
```

```
   LEN    EQU   $ -TABLE
DATA      ENDS
STAK      SEGMENT  STACK  'STACK'
STK       DB    20 DUP (0)
STAK      ENDS
CODE      SEGMENT
          ASSUME  CS:CODE, DS:DATA, SS:STAK
START:    MOV   AX, DATA
          MOV   DS, AX
          MOV   BX, LEN -1          ;BX←比较轮数
LOP0:     MOV   SI, LEN -1          ;SI←第 N个数据在数据表中偏移量
          MOV   CX, BX              ;CX←比较次数计数值
          MOV   DX, SI              ; X←置交换标志为第 N个数据在数
                                    据表中偏移量
LOP1:     MOV   AL, TABLE [SI]
          CMP   AL, TABLE [SI-1]    相邻两数据比较
          JGE   NEXT
          MOV   AH, [SI-1]          ; ABLE [SI] ←→ TABLE [SI-1] 两数
                                    据交换
          MOV   [SI-1] , AL
          MOV   [SI] , AH
          MOV   DX, SI              ; X←发生交换处的位置,给交换标志
NEXT:     DEC   SI                  修改数据地址
          LOOP  LOP1                控制内循环比较完一轮吗?
          CMP   DX, LEN -1          需要下一轮吗?
          JZ    STOP                ;不需要下一轮,已全部排好序,转程序
                                    结束
          DEC   BX                  控制外循环所有轮都比较完否?
          JNZ   LOP0                ;未完继续
STOP:     MOV   AH, 4CH
          INT   21H
CODE      ENDS
          END   START
```

2. 查找

查找就是在一列表中查询指定的数据（一般又称关键字）。例如对学生考试成绩的查询/某一职工出身年月的查询、图书目录的查询、电话号码的查询等等都要用到查表操作。查表的方法很多,对于不同的表格结构采用的查表方法也不同。最常用的有直接查表法、顺序查表法和对分查表法。

查表后有两种可能的结果。若在列表中查到指定的数据称查表成功,此时可给出数据在表格中的位置（地址）或设置查表成功的标志。若在列表中查不到指定的数据称查表失败,此时应给出数据不在表中的标志。

⑴ 直接查表法

所谓直接查表法又称计算查表法。就是可按照被查询的数据和它在表格中位置之间存在的规律进行查询。一般都可通过“关键字”在表中的地址 =表格首址 +偏移地址,这个公式进行计算查找。实际上在前面分支程序设计中用的地址表法、转移表法均属直接查表法。这里不再举例。

⑵ 顺序查表法

对于无序数据进行检索,通常采用顺序检索的方法。它将关键字与数据项中的有关数据逐个比较,若找到,则检索成功,否则检索失败。

【例题 4.24】　设在内存数据区 LINTAB单元开始存放一数据表,表中为有符号的字数据。表长在 COUNT单元,要查找的关键数据存放在 KEYBUF单元。编制程序查找 LINTAB表中是否有 KEYBUF单元中指定的关键数据。若有将其在表中的地址存入 ADDR单元,否则将此单元置全 1标志。

分析 根据题意仍可采用加前缀的检索指令:REPNZ　SCASW来完成,这和前面字符串查询操作中使用方法一样。

源程序:

```
DATA    SEGMENT
  LINTAB  DW    35, -56, 378, 1024, -32767, 512, 256, -255…
  COUNT   EQU   ($ -LINTAB) /2
  KEYBUF  DW    256                    ;关键字
  ADDR    DW    ?                      ;找到关键字的位置
DATA    ENDS
CODE    SEGMENT
        ASSUME CS:CODE, DS:DATA, ES:DATA
MAIN    PROC  FAR
START:  PUSH  DS
        XOR   AX, AX
```

```
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX
        MOV     ES, AX
        MOV     DI, OFFSET LINTAB       ;DI←表首址
        MOV     CX, COUNT               ;CX←表长
        MOV     AX, KEYBUF              ;AX←查询关键字
        CLD                             ;DF←0地址增量
        REPNZ   SCASW                   ;重复查表
        JZ      FOUND                   ;找到时转 FOUND
        MOV     DI, 01H                 ;找不到时
FOUND:  DEC     DI
        DEC     DI                      ;计算关键字在表中的地址
        MOV     ADDR, DI                ;保存地址
        RET
MAIN    ENDP
CODE    ENDS
        END     START
```

(3) 对分查找

若表中的内容已排序（设为升序），可采用对分查找。这种方法可大大减少查找次数。对分的思想是：先取表中间的值（N/2处的值）与要查找的值 X 比较，看是否相等，若相等则搜索到；若不等则比较两数的大小，若 $X>d_{N/2}$，则下次取 N/2～N 中间值 $d_{N/4}$ 与 X 比较；若 $X<d_{N/2}$，则下次取 0～N/2 中间值 $d_{N/4}$ 与 X 比较，这样每查找一次使区间缩小 1/2，如此一直进行下去，直至或者是被搜索的字找到，或者是搜索的区间变为 0，则表示搜索不到所要找的数。

【例题 4.25】 一有序数组：0、11、15、21、34、57、60、78、90、97 共 10 个元素，数的排列序号为 0～9，从中搜索数据 78，若找到，记下搜索次数，未找到，标记为全 1。

分析设计：取区间上限 L=0，区间下限 R=10，则序号 $J_1=(L+R)/2=5$ 取出数 $d_{J1}=d_5=57$ 与 X=78 相比，两者不等；而且 $X>d_5$，于是下一次要到下半区间去寻找，则取 $L=J_1=5$，R=10 不变，求出新的序号 $J_2=(L+R)/2=7.5$ 往下取整，取 $J_2=7$，则 $d_{J2}=d_7=78$ 与 X 相等，找到。程序中，用 DI 寄存器作为下限 R，用 SI 寄存器作为上限 L，用寄存器 BX 作为序号 J。

源程序：

```
DSEG    SEGMENT
  BUFF  DB  00,11,15,21,34,57,60,78,90,97
  COUNT EQU $-BUFF
  PTRN  DW  ?                    ;存放查找次数或未找到标记
  KEY   EQU 78                   ;关键字
DSEG    ENDS
SSEG    SEGMENT  PARA  STACK 'STACK'
STOP    DB  100 DUP(?)
SSEG    ENDS
CSEG    SEGMENT
        ASSUME  CS:CSEG,DS:DSEG,SS:SSEG,ES:DSEG
MAIN    PROC  FAR
START:  PUSH  DS
        MOV   AX,0
        PUSH  AX
        MOV   AX,DSEG
        MOV   DS,AX
        MOV   ES,AX
        MOV   SI,OFFSET BUFF     ;区间上限送 SI
        MOV   CX,COUNT
        MOV   DX,1               ;查找次数
        MOV   AX,SI
        ADD   AX,CX              ;最后数的地址加 1
        MOV   DI,AX              ;下限送 DI
        MOV   AL,KEY
COUNT1: MOV   BX,SI
        ADD   BX,DI
        SHR   BX,1               ;BX 中为 J
        CMP   AL,[BX]
        JZ    FOUND              ;找到转 FOUND
        PUSHF                    ;保护标志
        CMP   BX,SI              ;J等于上限
        JZ    NOFID              ;相等未找到
```

```
        POPF
        JL      LESS
        MOV     SI, BX          ;J作上限
        JMP     NEXT
LESS:   MOV     DI, BX          ;J作下限
NEXT:   INC     DX              ;查找次数加 1
        JMP     CONT1
NOFID:  MOV     DX, 0FFFFH      ;未找到标记 0FFFFH
        POPF
FOUND:  MOV     AX, DX
        MOV     PTRN, AX        ;结果送 PTRN单元
        RET
MAIN    ENDP
CSEG    ENDS
        END     START
```

当数据较多时,对分查找比顺序查找速度快得多,但它要求表的内容有序。表长 N=65 536时,顺序查找平均次数为 N/2=32 768,而对分查找平均次数(公式证明从略)为 Log(N-1)=15。

4.4.4 功能调用

【例题 4.26】 写一个程序,它先接受一个字符串,然后显示其中数字字符的个数、英文字母的个数和字符串的长度。

分析:先利用 0AH号功能调用接受一个字符串,然后分别统计其中数字字符的个数和英文字母的个数,最后用十进制数的形式显示它们。整个字符串的长度可从 0AH号功能调用的出口参数中取得。

源程序:

```
MLENGTH=128                     ;缓冲区长度
DSEG    SEGMENT                 ;数据段
BUFF    DB      MLENGTH         ;符合 0AH号功能调用所需的缓
                                 冲区
        DB      ?               ;实际键入的字符数
        DB      MLENGTH DUP(0)
```

```
  MESS0  DB    'Please input:$'
  MESS1  DB    'Length = $'
  MESS2  DB    'X = $'
  MESS3  DB    'Y = $'
DSEG     ENDS
CSEG     SEGMENT                      ;代码段
         ASSUME  CS:CSEG, DS:DSEG
START:   MOV   AX, DSEG
         MOV   DS, AX                 ;设置 DS
         MOV   DX, OFFSET  MESS0      ;显示提示信息
         CALL  DISPMESS
         MOV   DX, OFFSET  BUFF
         MOV   AH, 0AH                ;接受一个字符串
         INT   21H
         CALL  NEWLINE
         MOV   BH, 0                  ;清数字字符计数器
         MOV   BL, 0                  ;清字符计数器
         MOV   CL, BUFF+1             ;取字符串长度
         MOV   CH, 0
         JCXZ  COK                    ;若字符串长度等于 0,不统计
         MOV   SI, OFFSET  BUFF+2     ;指向字符串首
AGAIN:   MOV   AL, [SI]               ;取一个字符
         INC   SI                     ;调整数据指针,指向下一个数据
         CMP   AL, '0'                ;判断是否是数字符
         JB    NEXT                   ;小于 '0',不属于统计字符,转向
                                       取一个字符
         CMP   AL, '9'
         JA    NODEC                  ;大于 '9',不是数字字符,转向字
                                       母字符判断
         INC   BH                     ;'0'～'9',数字符计数加 1
         JMP   SHORT  NEXT            ;转向取一个字符
NODEC:   OR    AL, 20H                ;转小写
         CMP   AL, 'a'                ;判断是否是字符
         JB    NEXT
```

```
        CMP   AL,'z'
        JA    NEXT
        INC   BL                    ;字符计数加 1
NEXT:   LOOP  AGAIN                 ;下一个
COK:    MOV   DX,OFFSET  MESS1
        CALL  DISPMESS
        MOV   AL,BUFF+1             ;取字符串长度
        XOR   AH,AH
        CALL  DISPAL                ;显示字符串长度
        CALL  NEWLINE
        MOV   DX,OFFSET  MESS2
        CALL  DISPMESS
        MOV   AL,BH
        XOR   AH,AH
        CALL  DISPAL                ;显示数字符个数
        CALL  NEWLINE
        MOV   DX,OFFSET  MESS3
        CALL  DISPMESS
        MOV   AL,BL
        XOR   AH,AH
        CALL  DISPAL                ;显示字符个数
        CALL  NEWLINE
        MOV   AX,4C00H              ;程序正常结束
        INT   21H
;-------------------------
        ;子程序名:DISPAL
        ;功能:用十进制数的形式显示 8位二进制数
        ;入口参数:AL=8位二进制数
        ;出口参数:无
;-------------------------
DISPAL  PROC  NEAR
        MOV   CX,3                  ;8位二进制数最多转换成 3位十
                                     进制数
        MOV   DL,10
```

```
DISP1:  DIV   DL
        XCHG  AH, AL          ;使 AL=余数,AH=商
        ADD   AL, '0'         ;得 ASCII码
        PUSH  AX              ;压入堆栈
        XCHG  AH, AL
        MOV   AH, 0
        LOOP  DISP1           ;继续
        MOV   CX, 3
DISP2:  POP   DX              ;弹出一位
        CALL  ECHOCH          ;显示之
        LOOP  DISP2           ;继续
        RET
DISPAL  ENDP
                              ;显示由 DX所指的提示信息,其
                               他子程序说明信息略
DISPMESS PROC NEAR
        MOV   AH, 9
        INT   21H
        RET
DISPMESS ENDP
;------------------------------
        ;子程序名 :ECHOCH
        ;功能 :调用 DOS 2号功能,显示一个字符
        ;入口参数 :DL =显示字符
        ;出口参数 :无
        ;说明 :通过显示回车符形成回车,通过显示换行符形成换行
;------------------------------
ECHOCH  PROC  NEAR
        MOV   AH, 2
        INT   21H
        RET
ECHOCH  ENDP
;------------------------------------------------
        ;子程序名 :NEWLINE
```

```
                ;功能:形成回车和换行
                ;入口参数:无
                ;出口参数:无
                ;说明:通过显示回车符形成回车,通过显示换行符形成换行
;------------------------------------------------------------
NEWLINE  PROC   NEAR
         PUSH   AX
         PUSH   DX
         MOV    DL,0DH          ;回车符的ASCII码是0DH
         MOV    AH,2            ;显示回车符
         INT    21H
         MOV    DL,0AH          ;换行符的ASCII码是0AH
         MOV    AH,2            ;显示换行符
         INT    21H
         POP    DX
         POP    AX
         RET
NEWLINE  ENDP
CSEG     ENDS
         END START
```

【例题 4.27】 写一个程序完成如下功能:读键盘,并把所按键显示出来,在检测到按下 Shift键后,就结束运行。

调用键盘 I/O程序的 2号功能取得变换键状态字节,进而判断是否按下了 Shift键。在调用 0号功能读键盘之前,先调用 2号功能判键盘是否有键可读,否则会导致不能及时检测到用户按下的 SHIFT键。

源程序:

```
                                        ;常量定义:
L_SHIFT = 00000010B
R_SHIFT = 00000001B
                                        ;代码段
CSEG     SEGMENT
         ASSUME  CS:CSEG
START:   OV      AH,2                   ;取变换键状态字节
```

```
        INT     16H
        TEST    AL,L_SHIFT+R_SHIFT      ;判是否按下 SHIFT键
        JNZ     OVER                    ;按下,转
        MOV     AH,1
        INT     16H                     ;是否有键可读
        JZ      START                   ;没有,转
        MOV     AH,0                    ;读键
        INT     16H
        MOV     DL,AL                   ;显示所读键
        MOV     AH,6
        INT     21H
        JMP     START                   ;继续
OVER:   MOV     AH,4CH
        INT     21H
CSEG    ENDS
        END     START
```

【例题 4.28】　采用直接写屏法在屏幕上用多种属性显示字符串“HELLO”.

先用一种属性在屏幕上显示指定信息然后在用户按一键后再换一种属性显示,如按 ESC键,则结束。其中的显示子程序 ECHO采用了直接写屏方法实现显示。

源程序：

```
;常量定义
ROW  = 5                                ;显示信息的行号
COLUM  = 10                             列号
ESCKEY  = 1BH                           ;ESC键的 ASCII码值
                                        数据段
DSEG        SEGMENT
  MESS      DB 'HELLO'                  ;显示信息
  MESS_LEN  = $ -OFFSET MESS            ;显示信息长度
  COLORB    DB 07H,01H,0FH,70H,74H      颜色
  COLORE    LABEL   BYTE
DSEG        ENDS
```

```
                                        ;代码段
CSEG    SEGMENT
        ASSUME  CS:CSEG, DS:DSEG, ES:DSEG
START:  MOV     AX, DSEG
        MOV     DS, AX                  ;设置数据段段值
        MOV     ES, AX
        MOV     DI,OFFSET COLORB-1      ;颜色指针初值
NEXTC:  INC     DI                      ;调整颜色指针
        CMP     DI,OFFSET  COLORE       ;是否超过指定的最后一种颜色
        JNZ     NEXTE                   ;否
        MOV     DI,OFFSET  COLORB       ;是,重新指定第一种颜色
NEXTE:  MOV     BL, [DI]                ;取颜色
        MOV     SI,OFFSET MESS          ;取显示信息指针
        MOV     CX, MESS_ LEN           ;取显示信息长度
        MOV     DH, ROW                 ;置显示开始行号
        MOV     DL, COLUM               ;置显示开始列号
        CALL    ECHO                    ;显示
        MOV     AH, 0
        INT     16H
        CMP     AL, ESCKEY              ;是否为 ESC键
        JNZ     NEXTC                   ;不是,继续
        MOV     AX, 4C00H               ;结束
        INT     21H
;-----------------------------------------
        ;子程序名 :ECHO
        ;功能 :直接写屏显示字符串
        ;入口参数 :DS:SI=字符串首地址
        ; X =字符串长度 ;BL =属性
        ; H =显示开始行号 ;DL=显示开始列号
        ;出口参数 :无
;-----------------------------------------
ECHO    PROC    NEAR
        MOV     AX, 0B800H
        MOV     DS, AX
```

```
        MOV   AL,80                ;计算显示开始位置偏移
        MUL   DH                   ;偏移 = (行号*80+列号)*2
        XOR   DH,DH
        ADD   AX,DX
        ADD   AX,AX
        XCHG  AX,BX
        MOV   AH,AL                ;属性值保存到 AH寄存器
        JCXZ  ECHO2                ;显示信息长度是否为 0
ECHO1:  MOV   AL,ES:[SI]           ;取一要显示字符代码
        INC   SI                   ;调整指针
        MOV   [BX],AX              ;送显示存储区,即显示
        INC   BX                   ;准备显示下一个字符
        INC   BX
        LOOP  ECHO1                ;循环显示
ECHO2:  RET                        ;返回
ECHO    ENDP
CSEG    ENDS
        END   START
```

【例题 4.29】　调用 BIOS显示程序来实现例题 4.28。

方法一：调用 BIOS显示程序中的 13H号功能直接显示字符串。

源程序：

```
;-------------------------------------------
        ;子程序名:ECHOA
        ;调用 BIOS显示程序的 13H号功能显示字符串
        ;入口参数:ES:BP=字符串首地址
        ;CX =字符串长度 ;BL=属性 ;BH=页号
        ; H =显示开始行号 ;DL=显示开始列号
        ;AL=写方式 (0~3)
        ;出口参数:无
;-------------------------------------------
ECHOA   PROC  NEAR
        PUSH  ES
        PUSH  BP                   ;保护有关寄存器内容
```

```
        PUSH    DS
        POP     ES
        MOV     BP,SI           ;满足 13H 号功能入口参数要求
        MOV     BH,0            ;指定显示页
        MOV     AL,0            ;采用 0号显示方式 (不移动光标)
        MOV     AH,13H          ;字符串中不含属性
        INT     10H
        POP     BP
        POP     ES              ;恢复有关寄存器内容
        RET
ECHOA   ENDP
```

该子程序与 ECHO稍有不同 :第一,13H 号功能解释回车和换行等控制码,所以如果要显示的字符中含有这样的控制码,那么显示效果就会有差异 ;第二,当显示超出最后一行的最后一列时,13H功能要引起滚屏。

方法二 :先调用 BIOS显示程序的 2号功能把光标定到指定位置中,然后利用 BIOS显示程序的 0EH 号功能逐个显示字符串中的字符。但由于该方式显示不含属性,因此先调用 9号功能把指定属性写到显示字符串的位置处。

源程序:

```
;--------------------------------
;子程序名 :ECHOB
;入口参数 :DS:SI=字符串首地址
;CX = 字符串长度 ;BL = 属性
;  H = 显示开始行号 ;DL = 显示开始列号
;出口参数 :无
;--------------------------------
ECHOB   PROC    NEAR
        JCXZ    ECHO2           如果字符串长度为 0,则结束
        MOV     BH,0
        MOV     AH,2            ;设置光标位置
        INT     10H
        MOV     AL,20H          ;用指定属性写一空格
        MOV     AH,9
        INT     10H
```

```
        MOV     AH,0EH
ECHO1:  MOV     AL,[SI]
        INC     SI
        INT     10H             ;逐个显示字符
        LOOP    ECHO1
ECHO2:  RET
ECHOB   ENDP
```

该子程序与 ECHO也稍有不同:第一,0EH号功能显示也解释控制符,所以如果要显示的字符串中含有控制符,则显示效果就不一样;第二,当在屏幕的右下角显示一个字符后要上滚屏幕;第三,字符串显示完后,光标定位在字符串之后。

方法三:先读当前光标位置且保存;定位光标到指定位置;调用 BIOS显示程序的 9号功能逐个显示字符,每显示一个字符后,把光标向后移一个位置;最后把光标回到原位。注意,在把光标向后移一个位置时,要判别是否超越屏幕右边界和是否已达到屏幕的右下角。请读者作为练习完成子程序 ECHOC。

思考题与习题

4.1　什么是标号?它有哪些属性?

4.2　什么是变量,它有哪些属性?

4.3　什么叫伪指令?什么叫宏指令?伪指令在什么时候被执行?宏指令在程序中如何调用?

4.4　汇编语言表达式中有哪些运算符?它所完成的运算是在什么时候进行的?

4.5　画出下列语句中的数据在存储器中的存储情况。

```
VARB    DB   34,34H,'GOOD',2 DUP(1,2 DUP(0))
VARW    DW   5678H,'CD',$ + 2,2 DUP(100)
VARC    EQU  12
```

4.6　按下列要求,写出各数据定义语句。

(1) DB1为 10H个重复的字节数据序列:1,2,5个 3,4。

(2) DB2为字符串'STUDENTS'。

(3) BD3为十六进制数序列:12H,ABCDH。

(4) 用等值语句给符号 COUNT赋以 DB1数据区所占字节数,该语句写在最后。

4.7　指令 OR　AX,1234H　OR 0FFH中两个 OR有什么差别?这两个操作分别在什么

时候执行？

4.8 对于下面的数据定义，各条 MOV 指令单独执行后，有关寄存器的内容是什么？

```
PREP    DB   ?
TABA    DW   5 DUP (?)
TABB    DB   'NEXT'
TABC    DD   12345678H
```

(1) MOV AX, TYPE PREP　　(2) MOV AX, TYPE TABA

(3) MOV CX, LENGTH TABA　　(4) MOV DX, SIZE TABA

(5) MOV CX, LENGTH TABB　　(6) MOV DX, SIZE TABC

4.9 设数据段 DSEG 中符号及数据定义如下，试画出数据在内存中的存储示意图。

```
DSEG     SEGMENT
  DSP = 100
  SAM = DSP + 20
  DAB    DB     '/GOTO/', 0DH, 0AH
  DBB    DB     101B, 19, 'a'
  .RADIX 16
  CCB    DB     10 DUP (?)
  EVEN
  DDW    DW     '12', 100D, 333, SAM
  .RADIX 10
  EDW    DW     100
  LEN    EQU    $ - DAB
DSEG     ENDS
```

4.10 若自 STRING 单元开始存放有一个字符串（以字符“$”结束）：

(1) 编程统计该字符串长度（不包含 $ 字符，并假设长度为两字节）。

(2) 把字符串长度放在 STRING 单元，把整个字符串往下移两个储存单元。

4.11 将字符串 STRING 中的“&”字符用空格符代替，字符串 STRING 为：“The data is FEB&03”。

4.13 考虑以下调用序列：

(1) MAIN 调用 NEAR 的 SUBA 过程（返回的偏移地址为 150BH）；

(2) SUBA 调用 NEAR 的 SUBB 过程（返回的偏移地址为 1A70H）；

(3) SUBB 调用 FAR 的 SUBC 过程（返回的偏移地址为 1B50H，段地址为 1000H）；

(4) 从 SUBC 返回 SUBB；

(5) 从 SUBB 返回 SUBA；

(6) 从 SUBA 返回 MAIN。

请画出每次调用或返回时，堆栈内容和堆栈指针变化情况。

4.14 设计以下子程序：

⑴ 将 AX中的 4位 BCD码转换为二进制码，放在 AX中返回。

⑵ 将 AX中无符号二进制数转换为十进制数 ASCII码字符串，放在 AX中返回。

⑶ 将 AX中有符号二进制数转换为十进制数 ASCII码字符串，DX和 CX返回串的偏移地址和长度。

4.15 试编写一个汇编语言程序，要求对键盘输入的小写字母用大写字母显示出来。

4.16 键盘输入 10个学生的成绩，试编制一个程序统计 60～69分，70～79分，80～89分，90～99分及 100分的人数，分别存放到 S6, S7, S8, S9及 S10单元中。

4.17 分别实现满足下面要求的宏定义：

⑴ 可对任一寄存器实现任意次数的左移操作。

⑵ 任意两个字单元中数据相加存于第三个单元中。

⑶ 将任意一个 8位寄存器中的压缩 BCD码转换为两个 ASCII码，在屏幕显示。

⑷ 重新定义指令助记符 ADD，完成双字的加法。

4.18 什么叫系统功能调用？在 MS－DOS中，可以用 INT指令进行系统功能调用，这样做的依据是什么呢？这种方法对你今后的软件开发有何启发作用？

第5章

8086的总线操作和时序

5.1 概　　述

5.1.1 时钟周期（T状态）、总线周期和指令周期

1. 时钟周期（T状态）

计算机是一个复杂的时序逻辑电路，时序逻辑电路都有“时钟”信号。计算机的“时钟”是由振荡源产生的、幅度和周期不变的节拍脉冲，每个脉冲周期称为时钟周期，又称为 T状态。计算机是在时钟脉冲的统一控制下，一个节拍一个节拍地工作的。在 IBM PC中，时钟频率为 4.77 MHz，一个 T状态为 210 ns。

2. 总线周期

当 CPU访问存储器或输入输出端口，需要通过总线进行读或写操作。与 CPU内部操作相比，通过总线进行的操作需要较长的时间。把 CPU通过总线进行某种操作的过程称为总线周期（Bus Cycle）。根据总线操作功能的不同，有多种不同的总线周期。如存储器读周期、存储器写周期、I/O读周期、I/O写周期等。

8086的一个基本的总线周期的时序如图 5.1所示。

该总线周期包含 4个 T状态（T state），即图 5.1中的 T_1、T_2、T_3、T_4。

⑴ 在 T_1状态，CPU往多路复用总线上发出地址信息，以指出要寻址的存

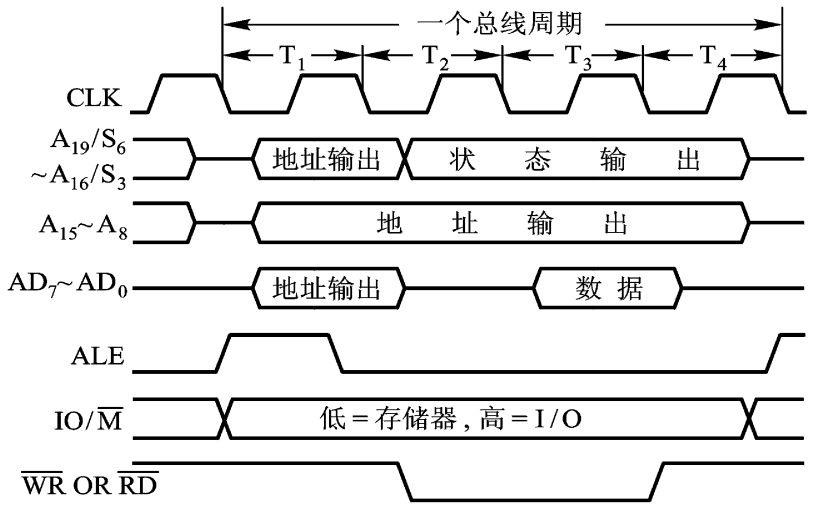


图 5.1　一个基本的总线周期时序

储单元或外设端口的地址。

⑵ 在 T_2状态，CPU从总线上撤销地址，而使总线的低 16位浮空，置成高阻状态，为传输数据做准备。总线的最高 4位（A_{19} ～A_{16}）用来输出本总线周期状态信息。这些状态信息用来表示中断允许状态，当前正在使用的段寄存器名等。

⑶ 在 T_3状态，多路总线的高 4位继续提供状态信息，而多路总线的低 16位（8088则为低 8位）上出现由 CPU写出的数据或者 CPU从存储器或端口读入的数据。

⑷ 在有些情况下，外设或存储器速度较慢，不能及时地配合 CPU传送数据。这时，外设或存储器会通过“READY”信号线在 T_3状态启动之前向 CPU发一个“数据未准备好”信号，于是 CPU会在 T_3之后插入 1个或多个附加的时钟周期 T_W。T_W也叫等待状态，在 T_W状态，总线上的信息情况和 T_3状态的信息情况一样。当指定的存储器或外设完成数据传送时，便在“READY”线上发出“准备好”信号，CPU接收到这一信号后，会自动脱离 T_W状态而进入 T_4状态。

⑸ 在 T_4状态，总线周期结束

需要指出，只有在 CPU和内存或 I/O接口之间传输数据，以及填充指令队列时，CPU才执行总线周期。可见，如果在一个总线周期之后，不立即执行下一个总线周期，那么，系统总线就处在空闲状态，此时，执行空闲周期。

在空闲周期中，可以包含一个时钟周期或多个时钟周期。这期间，在高 4位上，CPU仍然驱动前一个总线周期的状态信息，而且，如果前一个总线周期为写

周期，那么，CPU会在总线低16位上继续驱动数据信息；如果前一个总线周期为读周期，则在空闲周期中，总线低16位处于高阻状态。

图5.2为一个典型的总线周期序列。

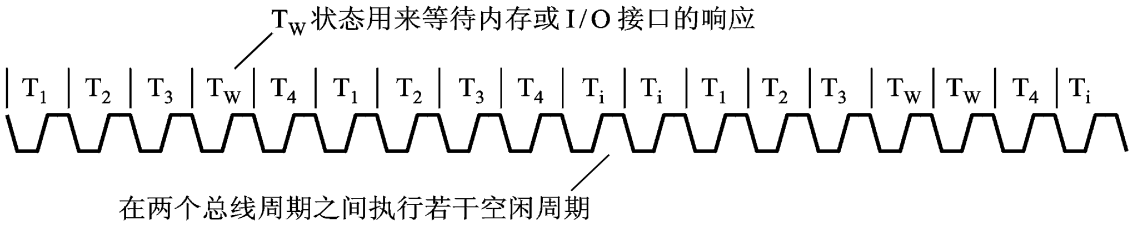


图5.2　一个典型的总线周期序列

3. 指令周期

每条指令的执行包括取指令（fetch）、译码（decode）和执行（execute）。

执行一条指令所需要的时间称为指令周期（Instruction Cycle）。指令周期是由1个或多个总线周期组合而成。或者说，指令周期可以被划分为若干个总线周期。

8086中不同指令的指令周期是不等长的。由于8086中的指令码最短的只需要一个字节，多的有6个字节，多字节指令，取指（存储器读）就需要多个总线周期；在指令的执行阶段，由于各种不同寻址方式，需要的总线周期个数也各不相同，因此8086的指令周期是不等长的。

对于8086 CPU来说，在EU执行指令的时候，BIU可以取下一条指令。由于EU和BIU可以并行工作，8086指令的最短执行时间可以是两个时钟周期，一般的加、减、比较、逻辑操作是几十个时钟周期，最长的为16位乘除法约要200个时钟周期。

5.1.2　8086/8088的引脚信号

图5.3是8086和8088微处理器的引脚信号图。

8088 CPU是继8086之后推出的准16位微处理器，被畅销全球的IBM PC选作CPU，它与8086 CPU具有类似的体系结构。两者的执行部件EU完全相同，其指令系统，寻址能力及程序设计方法都相同，所以两种CPU完全兼容。这两种CPU的主要区别如下：

⑴ 外部数据总线位数的差别；8086 CPU的外部数据总线有16位，在一个总线周期内可输入/输出一个字（16位数据），使系统处理数据和对中断响应的

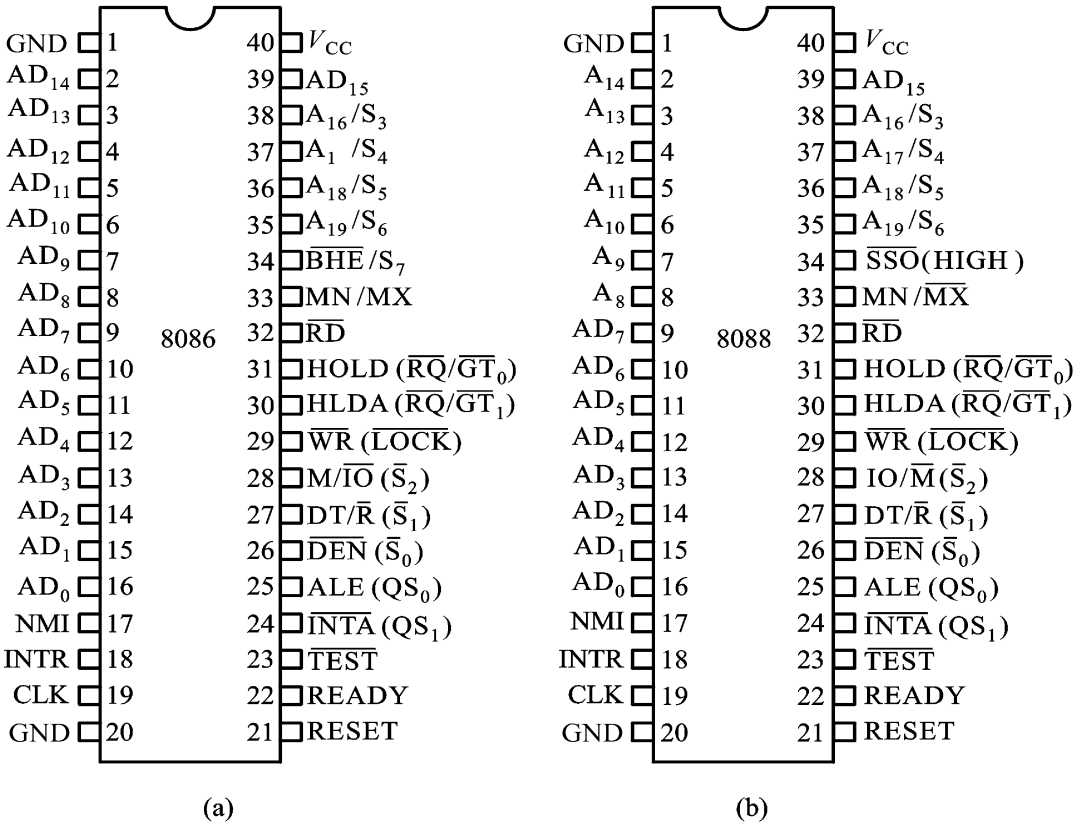


图 5.3　8086和 8088的引脚信号图

速度得以加快;而 8088 CPU的外部数据总线为 8位,在一个总线周期内只能输入/输出一个字节 (8位数据)。

(2) 指令队列容量的差别:8086 CPU的指令队列可容纳 6个字节,且在每个总线周期中从存储器中取出 2个字节的指令代码填入指令队列,这可提高取指操作和其他操作的并行率,从而提高系统工作速度;而 8088 CPU的指令队列只能容纳 4个字节,且在每个总线周期中能取一个字节的指令代码,从而增长了总线取指令的时间,在一定条件下可能影响取指令操作和其他操作的并行率。

(3) 引脚特性的差别:两种 CPU的引脚功能是相同的,但有以下几点不同:

1) AD_{15} ~AD_0的定义不同:在 8086中都定义为地址/数据复用总线;而 8088中,由于只需 8条数据总线,因此,对应于 8086的 AD_{15} ~AD_8这 8条引脚定义为 A_{15} ~A_8,只作地址线使用。

2) 34号引脚的定义不同:在 8086中定义为 $\overline{BHE}$信号;而在 8088中定义为 $\overline{SS_0}$,它与 DT/R、IO/M一起用作最小方式下的周期状态信号。

3) 28号引脚的相位不同,在 8086中为 M/IO;而在 8088中被倒相,改为

I/O/M,以便与 8080/8085系统的总线结构兼容。

5.2 8086的两种模式

5.2.1 最小模式和最大模式的概念

当把 8086 CPU与存储器和外设构成一个计算机系统时,根据所连的存储器和外设的规模,8086具有两种不同的工作模式,即最小模式和最大模式。8086到底工作在最大模式还是最小模式,由硬件设计决定。

1. 最小模式

当要利用 8086构成一个较小的系统时,在系统中只有 8086一个微处理器,所连的存储器容量不大、片子不多,所要连的 I/O端口也不多,系统中的总线控制电路被减到最少。系统的地址总线可以由 CPU的 AD_0 ~AD_{15}、A_{16} ~A_{19}通过地址锁存器 8282构成;数据总线可以直接由 AD_0 ~AD_{15}供给,也可以通过发送/接收接口芯片 8286 (或 74LS245)供给 (增大总线的驱动能力);系统的控制总线就直接由 CPU的控制线供给,这种模式就称为 8086的最小模式,如图 5.4所示。

2. 最大模式

最大模式是相对最小模式而言的。最大模式用在中等规模的或者大型的8086系统中。若要构成的系统较大,就要求有较强的总线驱动能力,这样 8086要通过一个总线控制器 8288来形成各种总线周期,控制信号由 8288供给。在最大模式系统中,总是包含有两个以上总线主控设备,其中一个就是 8086或者8088微处理器,其他的通常是协处理器,它们是协助微处理器工作的。8086最大工作模式如图 5.5所示。

和 8086配合的协处理器有两个,一个是数值运算协处理器 8087,一个是输入/输出协处理器 8089。

8087是一种专用于数值运算的处理器,它能实现多种类型的数值操作,比如高精度的整数和浮点运算,也可以进行超越函数 (如三角函数、对数函数)的计算。在通常情况下,这些运算往往通过软件方法来实现,而 8087是用硬件方法来完成这些运算的,所以,在系统中加入协处理器 8087之后,会大幅度地提高系统的数值运算速度。

8089在原理上有点像带有两个直接存储器存取 (DMA)通道的处理器,它有一套专门用于输入/输出操作的指令系统。但是,8089又和 DMA控制器不同,

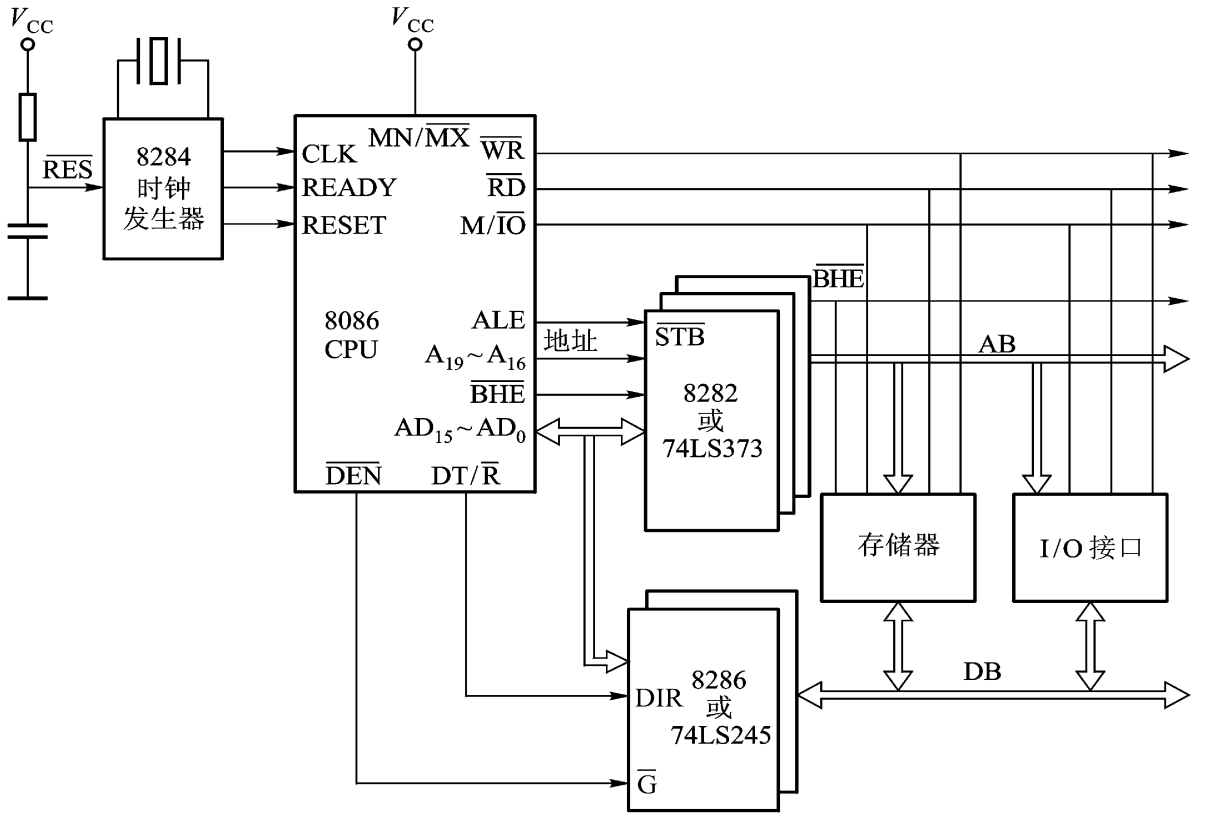


图 5.4　8086的最小模式

它可以直接为输入/输出设备服务,使 8086或/8088不再承担这类工作。所以,在系统中增加协处理器 8089后,会明显提高主处理器的效率,尤其是在输入/输出频繁的场合。

在这两种组态下,8086引脚中的脚 24~脚 31有不同的名称和意义,所以需要有一个引脚 MN/$\overline{MX}$来规定 8086处在什么模式,若把 MN/$\overline{MX}$引脚连至电源(+5 V),则为最小模式;若把它接地,则 8086处在最大模式。

5.2.2　8086 CPU引脚功能

1. 与工作模式无关的引脚功能

(1) AD_{15} ~AD_0 (双向,三态)

低 16位地址/数据的复用引脚线。采用分时的多路转换方法来实现对地址线和数据线的复用。在总线周期的 T_1状态,这些引线表示为低 16位地址线,在总线周期的 T_2, T_3, T_W状态,这些引线用作数据总线。可见对复用信号是用时间来加以划分的,它要求在 T_1状态先出现低 16位的地址时,用地址锁存器加以锁

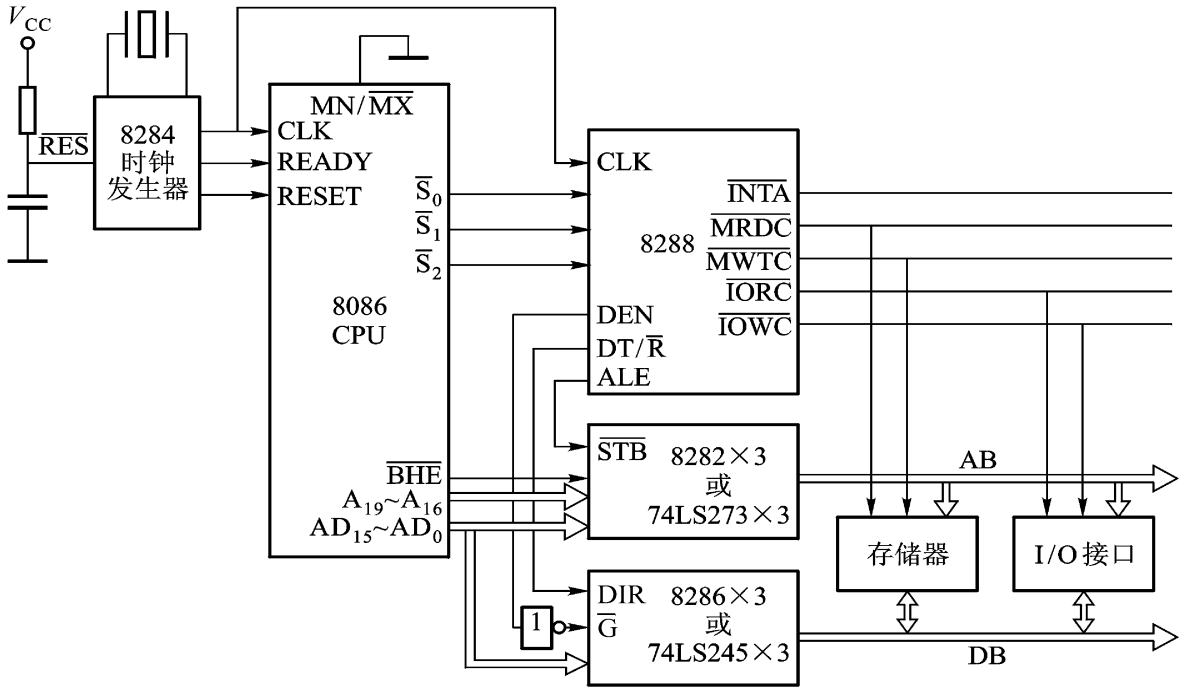


图 5.5 8086的最大模式

存,这样在随后的 T状态,即使这些线用作数据线,而低 16位地址线的地址状态却被记录保存下来,并达到地址总线上。在 DMA方式时,这些引线被浮空,置为高阻状态。

(2) A_{19}/S_6 ~A_{16}/S_3 (输出、三态)

为地址/状态复用引脚线,在总线周期的 T_1状态,这些引线表示为最高 4位地址线 (在读/写外设端口时,最高 4位地址线不用,故这些引线全为低电平),在总线周期的其他 T状态 (T_2, T_3, T_W和 T_4)时,这些引线用作提供状态信息。同样需要地址锁存器对 T_1状态出现的最高 4位地址加以锁存。

状态信息 S_6总是为低电平。S_5反映当前允许中断标志的状态。S_4与 S_3一起指示当前哪一个段寄存器被使用。其规定如表 5.1所示。

表 5.1 S_4、S_3代码组合

S_4	S_3	当前正在使用的段寄存器名
0	0	ES
0	1	SS
1	0	CS或未用
1	1	DS

在 DMA方式时,这些引线被浮置为高阻。

(3) $\overline{BHE}/S_7$ (输出,三态)在 T_1周期时,它作为 CPU访问存储器高位库的允许信号,低电平有效。它与 AD_0 结合在一起,决定访问存储器高位库和低位库。在其他 T状态时,作为一条状态信号线。

(4) $\overline{RD}$ (输出,三态)

读信号,当其有效时 (低电平)表示正在对存储器或 I/O端口进行读操作,若 IO/$\overline{M}$为低电平,表示读取存储器的数据;若 IO/$\overline{M}$为高电平,表示读取 I/O端口的数据。

在 DMA方式时,此线被浮置为高阻。

(5) READY (输入)

为准备就绪信号。是由选中的存储器或 I/O端口送来的响应信号,当有效时 (高电平),表示被访问的存储器或 I/O端口已准备就绪,可完成一次数据传送。CPU在读操作总线周期的 T_3状态开始处,采样 READY信号,若发现为低电平,则在 T_1状态结束后,插入等待状态 T_W。然后在 T_W开始处,继续采样 READY信号,直至变为有效 (高电平),才进入 T_4状态,完成数据传送,以结束总线周期。

(6) $\overline{TEST}$ (输入)

为检测信号,低电平有效。本信号由等待指令 WAIT来检查。$\overline{TEST}$信号和 WAIT指令配合使用。当 CPU执行 WAIT指令时,CPU处于等待状态,并且每隔 5个 T对该信号进行一次测试,一旦检测到$\overline{TEST}$号为低,则结束等待状态,继续执行 WAIT指令下面的指令。WAIT指令是使 CPU与外部硬件同步的,$\overline{TEST}$相当于外部硬件的同步信号。

(7) INTR (输入)

可屏蔽中断请求信号,高电平有效。CPU在执行每条指令的最后一个 T状态时,去采样 INTR信号,若发现为有效,而中断允许标志 IF又为 1,则 CPU在结束当前指令周期后响应中断请求,转去执行中断处理程序。

(8) NMI(输入)

非屏蔽中断请求信号,为一个边缘触发信号,不能由软件加以屏蔽。只要在 NMI线上出现由低到高的变化信号,则 CPU就会在结束当前指令后,转去执行非屏蔽中断处理程序。

(9) RESET (输入)

复位信号,高电平有效。复位时该信号要求维持高电平至少 4个时钟周期,若是初次加电,则高电平信号至少要保持 50 μs。复位信号的到来,将立即结束 CPU的当前操作,内部寄存器恢复到初始状态,如表 5.2所示。

当 RESET信号从高电平回到低电平时,即复位后进入重新启动时,便执行从内存 FFFF0H处开始的指令,通常在 FFFF0H存放一条无条件转移指令,转移到系统程序的实际入口处。这样只要系统被复位启动,就自动进入系统程序。

表 5.2 复位时各内部寄存器的值

标志寄存器	清零
指令指针 (IP)	0000H
CS寄存器	FFFFH
其他寄存器	0000H
指令队列	空

(10) CLK (输入)

时钟信号,它为 CPU和总线控制电路提供基准时钟。

(11) 电源和地

V_{CC}为电源引线,单一的 +5 V电源。

引脚 1和 20为两条 GND线,要求均要接地。

(12) MN/$\overline{MX}$ (输入)

为最小/最大模式信号。它决定 8086的工作模式。将此引线接电源 +5 V,则 8086工作于最小模式;若此引线接地,则 8086工作在最大模式。

引脚 24~31在不同模式下有不同的功能含义。下面分别加以介绍。

2. 最小模式下的引脚功能

把 MN/$\overline{MX}$引脚连至电源,8086处于最小模式,此时引脚 24~31的功能含义如下述。

(1) $\overline{INTA}$ (Interrupt Acknowledge,输出)

CPU向外输出的中断响应信号,用于对外部中断源发出中断请求做出的响应,中断响应周期由两个连续的总线周期组成,在每个响应周期的 T_2, T_3和 T_W状态,$\overline{INTA}$均为有效 (低电平),在第二个中断响应周期,外设端口往数据总线上发送中断类型号,CPU根据中断向量而转向中断处理程序。

(2) ALE (Address Latch Enable,输出)

地址锁存允许信号,高电平有效。在总线周期的 T_1状态,当地址/数据复用线 AD_{15}~AD_0和地址/状态复用线 A_{19}/S_6~A_{16}/S_3上出现地址信号时,CPU提供 ALE有效电平,将地址信息锁存到地址锁存器中。

(3) $\overline{DEN}$ (Data Enable,输出,三态)

数据允许信号,在使用 8286或 74LS245数据收发器的最小模式系统中,在存储器访问周期,I/O访问周期或中断响应周期,此信号有效,用来作为 8286或 74LS245数据收发器的输出允许信号,即允许收发器和系统数据总线进行数据传送。

在 DMA方式时,此线被浮置为高阻。

(4) DT/R (Data Transmit/Receive,输出,三态)

数据发送/接收控制信号。在使用 8286或 74LS245数据收发器的最小模式系统中,用 DT/R来控制数据传送方向。DT/R为低电平,进行数据接收 (CPU读),即收发器把系统数据总线上的数据读进来。

当 CPU处在 DMA方式时,此线浮空。

(5) IO/M (输出,三态)

访问存储器或 I/O端口的控制信号。若 IO/M为高电平,则访问的是 I/O端口;若 IO/M为低电平,则访问的是存储器。

(6) $\overline{WR}$ (输出,三态)

写信号。当其有效时 (低电平)表示 CPU正在对存储器或 I/O端口进行写操作,具体对谁进行写操作,由 IO/M信号决定。本信号在总线周期的 T_2, T_3及 T_W状态有效。

(7) HOLD (输入)

总线保持请求信号。当系统中 CPU之外的总线主设备要求占用总线时,通过 HOLD引线向 CPU发出高电平的请求信号。如果 CPU允许让出总线,则 CPU在当前周期的 T_4状态,由 HLDA引线向总线主设备输出一高电平信号作为响应,同时使地址总线、数据总线和相应的控制线处于浮空状态,总线请求主设备取得了对总线的控制权。一旦总线使用完毕,总线请求主设备让 HOLD变为低电平,CPU检测到 HOLD为低后,把 HLDA也置为低电平,CPU又获得了对总线的控制权。

(8) HLDA (输出)

总线保持响应信号。当 HLDA有效 (高电平)时,表示 CPU对总线请求主设备做出响应,同意让出总线,与 CPU相连的三态引线都被浮置为高阻态。

(9) $\overline{SSO}$ (System Status Output)

系统状态信号,它与 IO/M, DT/R共同组合反映当前总线周期执行的是什么操作,如表 5.3所示。

表中除无源外,其他均对应了一个明确的总线操作过程。而无源不是这种情况,它只是表示一个总线操作过程的结束,下一个总线周期还未开始。

表 5.3 8086的 IO/M, DT/R, $\overline{SSO}$代码组合及对应的操作

IO/M	DT/R	$\overline{SSO}$	操 作
1	0	0	发中断响应信号
1	0	1	读 I/O端口
1	1	0	写 I/O端口
1	1	1	暂停
0	0	0	取指令
0	0	1	读内存
0	1	0	写内存
0	1	1	无源状态

3. 最大模式下的引脚功能

把 MN/$\overline{MX}$引脚接地,则系统就处在最大模式下。此时引脚 24～31具有另外的功能含义,介绍如下。

(1) QS_1, QS_0 (输出)

指令队列状态 (Queue Status)信号。QS_1和 QS_0的组合提供了总线周期前一个 T状态中指令队列的状态 (队列状态只在队列操作执行以后的时钟周期有效),允许外部设备跟踪 8086内部指令队列状况。QS_1和 QS_0的组合所代表的指令队列状态输出如表 5.4所示。

表 5.4 QS_1和 QS_0的代码组合和对应的操作

QS_1	QS_0	含 义
0	0	无操作
0	1	从指令队列的第一个字节中取走代码
1	0	队列空
1	1	除第一个字节外,还取走了后续字节中的代码

(2) $\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$的组合及其对应的操作如表 5.5所示。

当 CPU处在 DMA传送方式时,这三根引线浮置为高阻。

(3) $\overline{LOCK}$ (输出,三态)

总线封锁信号。当本信号有效 (低电平)时,封锁了系统中别的总线主设备对系统总线的占有。$\overline{LOCK}$输出信号是由前缀指令 LOCK产生的,且保持有效直

至 LOCK指令的下面一条指令执行后。

另外,在 8086的中断响应时,在两个连续响应周期之间,$\overline{LOCK}$信号亦变为有效,以防止一个完整的中断过程被外部主设备占用总线而破坏。

表 5.5　$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$的代码组合和对应的操作

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	操作过程
0	0	0	发中断响应信号
0	0	1	读 I/O端口
0	1	0	写 I/O端口
0	1	1	暂停
1	0	0	取指令
1	0	1	读内存
1	1	0	写内存
1	1	1	无效状态

在 DMA操作时,$\overline{LOCK}$引线端被浮空。

(4) $\overline{RQ}/\overline{GT_0}$, $\overline{RQ}/\overline{GT_1}$ (Request/Grant,双向)

总线请求/允许信号。为两个信号端,每个信号端可供 CPU以外的一个总线设备来发出使用总线请求信号,以及接收来自 CPU的允许总线请求响应信号。类似于最小系统中的 HOLD和 HLDA信号,但 $\overline{RQ}/\overline{GT_0}$和 $\overline{RQ}/\overline{GT_1}$都是双向的,即在同一引脚上送总线请求信号 (对 CPU为输入),后传送允许信号 (对 CPU为输出)。

$\overline{RQ}/\overline{GT_0}$的优先权高于 $\overline{RQ}/\overline{GT_1}$。

5.3　最小模式下的 8086时序分析

5.3.1　最小模式下的读周期时序

读周期操作是指 CPU从存储器或 I/O端口读取数据的操作,其时序如图 5.6所示,分析如下:

(1) 在 T_1状态　CPU首先要判断是从存储器读取数据,还是从 I/O端口读取数据。这可用控制信号 M/$\overline{IO}$指出,若是存储器读,则 M/$\overline{IO}$为高,若是从 I/O端口读,则 M/$\overline{IO}$为低。其次,要给出所读取的存储器或 I/O端口的地址。8086有 20根地址线,前面已经指出,由于受封装引线数目的限制,其中最高 4根地址

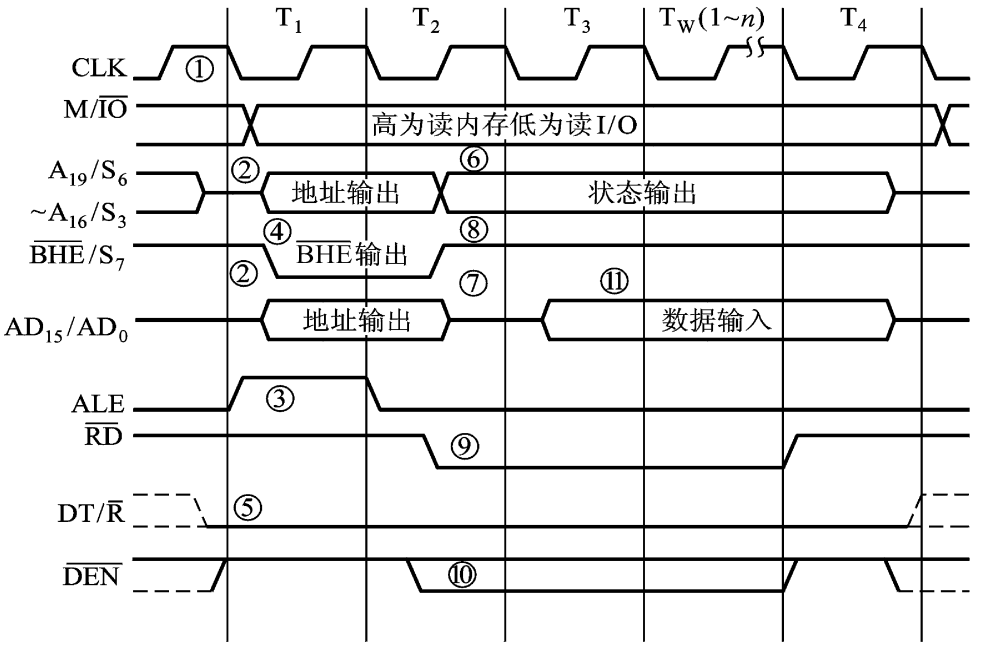


图 5.6 8086存储器或 I/O端口读周期时序

线 A_{19}/S_6 ～A_{16}/S_3和最低 16根地址线 AD_{15} ～AD_0是复用的。但在 T_1状态的开始,对存储器来说,这 20根引线上出现的全是地址信息;对 I/O端口来说,其最大寻址范围为 0000～FFFFH,故送往 I/O端口的是低 16位地址信息,最高 4位地址线 A_{19} ～A_{16}全为低电平。

在 T_1状态 CPU还必须在 ALE引脚上输出一个正脉冲,将 20位地址信息中的最高 4位和最低 8位锁存到地址锁存器 8282或 74LS373中。这样才能保证在总线周期 T_1以后的 T状态往这些复用线上传送状态信息和数据时,不造成地址信息的破坏。

$\overline{BHE}$信号也在 T_1状态通过 $\overline{BHE}$/引脚送出,它用来表示高 8位数据总线上的信息可以使用。$\overline{BHE}$信号常常作为奇地址存储体的片选信号,配合地址信号来实现存储单元的选址,因为奇地址存储体中的信息总是通过高 8位数据线来传输。而偶地址存储体的片选信号是最低位地址 A_0。

若系统中接有数据收发器 8286时,要用到信号 DT/R来控制数据传输方向,用信号 $\overline{DEN}$来控制数据的选通。为此,在 T_1状态 DT/R应输出低电平,即表示为读周期。

⑵ 在 T_2状态 $\overline{BHE}$/S_7及 A_{19}/S_6 ～A_{16}/S_3引脚上输出状态信息 S_7 ～S_3;AD_{15} ～AD_0转为高阻,为下面读出数据做准备。

读信号 $\overline{RD}$在 T_2状态变为有效,允许将被地址信息选中的存储单元或 I/O端

口中的数据读出。

在接有数据收发器 8286的系统中，在 T_2状态 $\overline{DEN}$变为低电平，以选通 8286，允许数据来传送。

⑶ 在 T_3状态　被选中存储单元或 I/O端口把数据送到数据总线，以备为 CPU来读取。

⑷ T_W状态　当系统中所用的存储器或外设的工作速度较慢，从而不能用最基本的总线周期执行读操作时，系统中就要用一个电路来产生 READY信号，READY信号通过时钟发生器 8284A传递给 CPU。CPU在 T_3状态开始时刻对 READY信号进行采样。如果 CPU没有在 T_3状态的一开始采样到 READY信号为高电平（当然，在这种情况下，在 T_3状态，数据总线上不会有数据），那么，就会在 T_3和 T_4之间插入等待状态 T_W。T_W可以为 1个，也可以为多个。以后，CPU在每个 T_W开始时刻对 READY信号进行采样，等到 CPU接收到高电平的 READY信号后，再把当前 T_W状态执行完，便脱离 T_W进入 T_4。

在最后一个 T_W状态，数据肯定已经出现在数据总线上。所以，最后一个 T_W状态中总线的动作和基本总线周期中 T_3状态的完全一样。而在其他的 T_W状态，所有控制信号的电平和 T_3状态一样，但数据信号尚未出现在数据总线上。

⑸ T_4状态

在 T_4状态和前一个状态交界的下降沿处，CPU对数据总线进行采样，从而获得数据。

5.3.2　最小模式下的写周期时序

写周期操作是指 CPU往存储器或外设端口写入数据的操作。其时序如图 5.7所示。

和读周期时序一样，最基本的写周期亦由 4个 T状态组成。当存储器或外设端口的工作速度较慢时，亦需在 T_3与 T_4状态之间插入一个或几个 T_W状态。

⑴ 在 T_1状态　M/$\overline{IO}$变为有效，以指出是对存储器还是对外设端口进行写操作，若写入存储器，M/$\overline{IO}$为高；若写入 I/O端口，则 M/$\overline{IO}$为低。

与读周期一样，在 T_1状态需提供存储器和外设端口地址，并同样要用 ALE信号把地址信息、M/$\overline{IO}$信号和 $\overline{BHE}$信号锁存到地址锁存器 8282中。

当系统中接有数据收发器 8286时，CPU在 T_1状态使 DT/$\overline{R}$为高电平，以表示执行的是写操作。

⑵ 在 T_2状态 CPU把要写入存储器或外设端口的数据，送到 AD_{15}～AD_0上

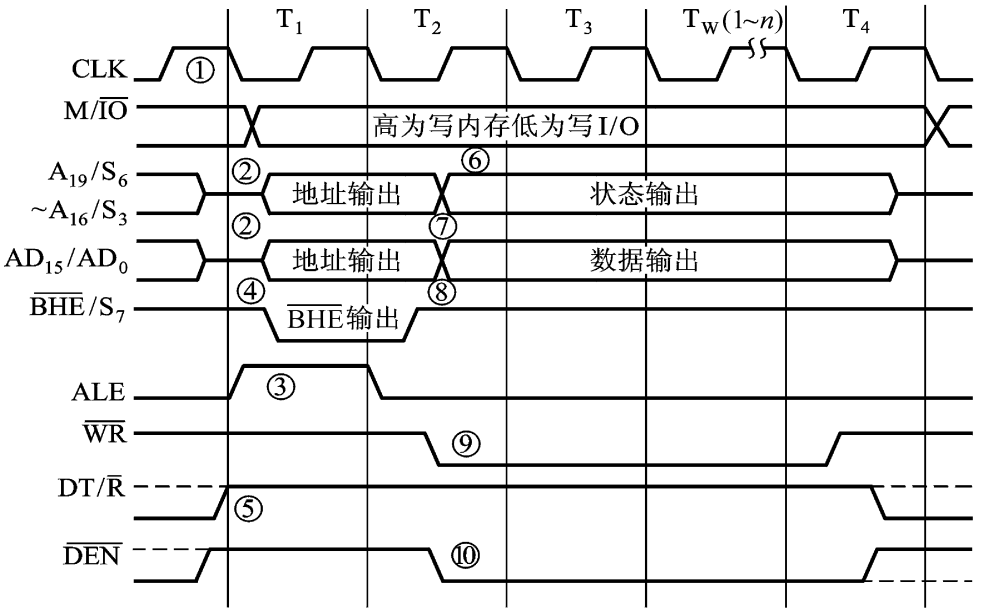


图 5.7 8086CPU写周期时序

在此之前低 16位地址已被锁存)。同时,CPU在高 4位地址/状态复用线上送出状态信息 S_6 ~S_3。

写信号$\overline{WR}$在 T_2状态变为有效,此信号与读周期中的$\overline{RD}$信号一样,一直维持有效到 T_4状态。$\overline{DEN}$也在 T_2状态变为有效,并维持到 T_4状态。

(3) 在 T_3状态 CPU继续提供 S_6 ~S_3状态信息和数据,并维持有关控制信号不变。

(4) 在 T_4状态 CPU完成对指定的存储器或外设端口的数据写入。数据和控制信号被撤除。

5.3.3 中断响应周期时序

一般外部设备的中断是通过$\overline{INTA}$引脚向 CPU发出中断请求的,这个可屏蔽中断请求信号的有效电平 高电平)必须维持到 CPU响应中断为止。若标志 IF =1,表示 CPU允许中断,则 CPU在执行完当前指令后响应中断。其中断响应周期时序如图 5.8所示。

中断响应周期由两个连续响应周期组成,在每一个响应周期的 T_2状态至 T_4状态开始,从$\overline{INTA}$引脚向外设接口发出一个负脉冲。在第一个响应周期,使 AD_7 ~AD_0浮空,在第二个响应周期,外设接口收到$\overline{INTA}$后,立即往数据总线

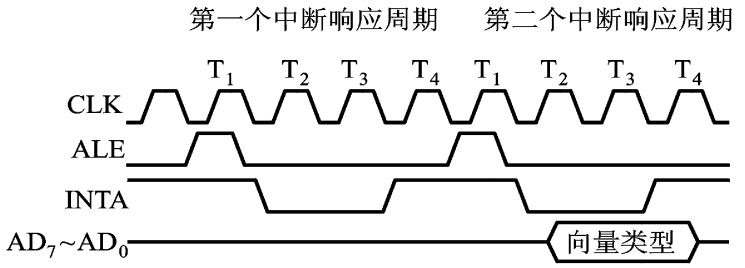


图 5.8　中断响应周期时序

AD_7 ~AD_0上送出一个中断类型码，CPU读入该类型码后，根据中断向量表找到该设备的处理程序的入口地址，转入中断处理。

对非屏蔽中断 NMI而言，其请求信号为一上升沿触发信号，并要求维持两个时钟周期的高电平。CPU响应 NMI的过程与响应 INTR请求的过程基本类似，不同点有二，一是前者无需判断 IF是否为 1，均要求响应；二是 CPU响应 NMI时，不需要从外设读取中断向量，而是按规定对应中断向量表中的中断类型 2，即直接从 0000:0008H开始的 4个单元中读取 NMI中断服务程序的入口地址。

5.3.4　8086的复位时序

前面在介绍 RESET引脚时，知道 RESET是用来启动或再启动系统的。这里着重介绍一下复位时序。

前面提到外部来的 RESET信号经 8284同步为内部 RESET，具体来讲是按时钟脉冲来同步的，如图 5.9所示。外部 RESET有效后的时钟脉冲上升沿使得内部 RESET变为有效。

在内部 RESET有效后，经过半个时钟周期，即用时钟脉冲下降沿驱动所有

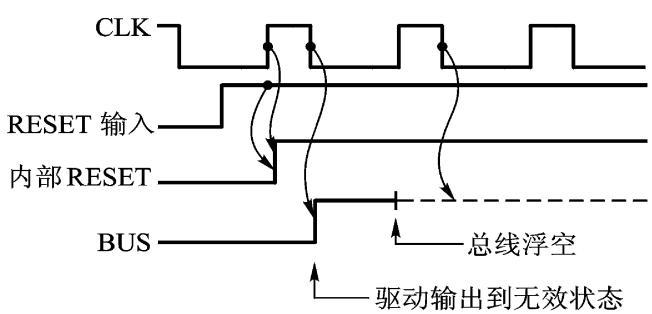


图 5.9　8086的复位时序

的三态引脚输出信号为不作用状态。这个不作用状态的时间为半个时钟周期（时钟周期的低电平期间），等到时钟脉冲由低变高时，三态输出线浮空为高阻状态，直到 RESET信号回到低电平时为止。

5.3.5 总线保持请求与保持响应时序

当系统中 CPU之外的总线主设备需要占用总线时，就向 CPU发出一个有效的总线保持请求信号 HOLD，这个 HOLD信号可能与时钟信号不同步，当 CPU在每个时钟周期的上升沿检测到该信号时，在当前总线周期的 T_4后或下一个总线周期的 T_1后，CPU发出一个有效的保持响应信号 HLDA，并让出总线，如图 5.10所示。

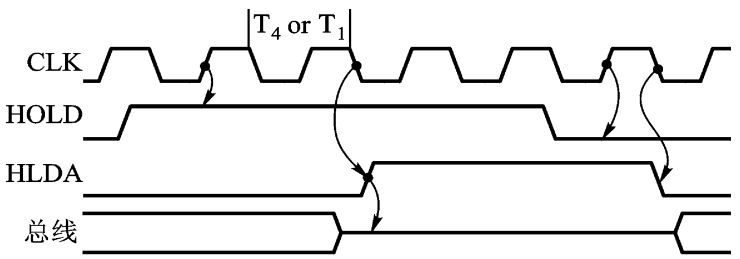


图 5.10 总线保持请求与保持响应的时序

在外部主设备 DMA传送结束后，使 HOLD变低，当 CPU在下一个时钟上升沿检测到 HOLD变低后，则在紧接着的下降沿让 HLDA变低，CPU又收回总线控制权。8237A DMA（直接存储器存取）芯片就是一种代表外设向 CPU发请求获得对总线控制权的器件。8086一旦让出总线控制权，便将所有具有三态的输出线 AD_{15}～AD_0，A_{19}/S_6～A_{16}/S_3，$\overline{RD}$，$\overline{WR}$，$\overline{INTA}$，$M/\overline{IO}$，$\overline{DEN}$及 $DT/\overline{R}$都置于高阻状态。即 CPU暂时与总线断开。但是，这里要注意：输出信号 ALE不是高阻的。

对于总线保持请求/保持响应操作时序，有下面几点需注意：

⑴ 当某一总线主模块向 CPU发来的 HOLD信号变为高电平（有效）后，CPU将在下一个时钟周期的上升沿检测到，若随后的时钟周期正好为 T_4或为 T_1，则在其下降沿处将 HLDA变为高电平；若 CPU检测到 HOLD后不是 T_4或 T_1，则可能会延迟几个时钟周期，等到下一个 T_4或 T_1出现时，才发出 HLDA信号。

⑵ 在总线保持请求/响应周期中，因三态输出线处于高阻状态，这将直接影响 8086的 BIU部件的工作，但是执行部件 EU将继续执行指令队列中的指令，直到遇到一条需要使用总线的指令时，EU才停止工作，或者把指令队列中指

令执行完,也会停止工作。由此可见,CPU和获得总线控制权的其他主模块之间,在操作上有一段小小的重叠。

⑶ 当 HOLD变为无效后,CPU也接着将 HLDA变为低电平。但是,不会马上驱动已变为高阻的输出引脚,只有等到 CPU新执行一个总线操作周期时,才结束这些引脚的高阻状态,因此,就可能出现一小段时间总线没有任何总线主模块的驱动,这种情况很可能导致这些线上的控制电平漂移到最小电平以下,为此,在控制线 HOLD和电源之间需要连接一个上拉电阻。

5.4　最大模式下的 8086时序分析

关于最大模式下的 8086时序,只介绍对存储器和外设端口的读/写操作时序。

与最小模式下的读/写操作时序一样,最大模式下的基本总线周期也是由 4个 T状态组成的。当存储器或外设端口的工作速度较慢时,也需在 T_3状态后插入一个或几个等待周期 T_W。与最小模式下时序相比,最大模式不同之处在于:最大模式下的时序,除 CPU有关信号外,还要特别考虑总线控制器 8288所产生的有关控制信号和命令信号。为此,在介绍最大模式的读/写时序之前,先介绍一下 8288总线控制器。

5.4.1　总线控制器 8288

8288的框图、引脚图和工作时序分别如图 5.11、图 5.12和图 5.13所示。

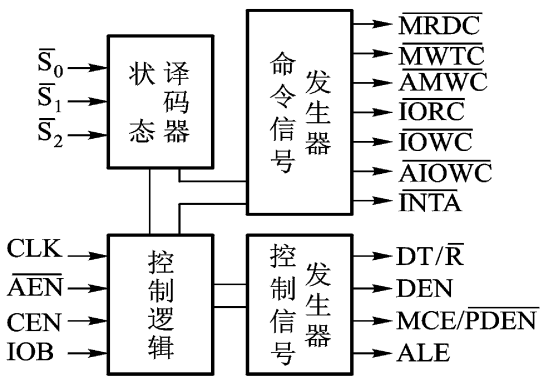


图 5.11　8288的框图

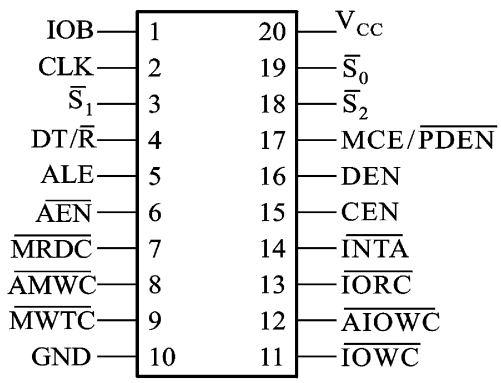


图 5.12　8288的引脚图

下面根据框图来介绍 8288的外接信号,它们可分为三组 输入信号、命令输

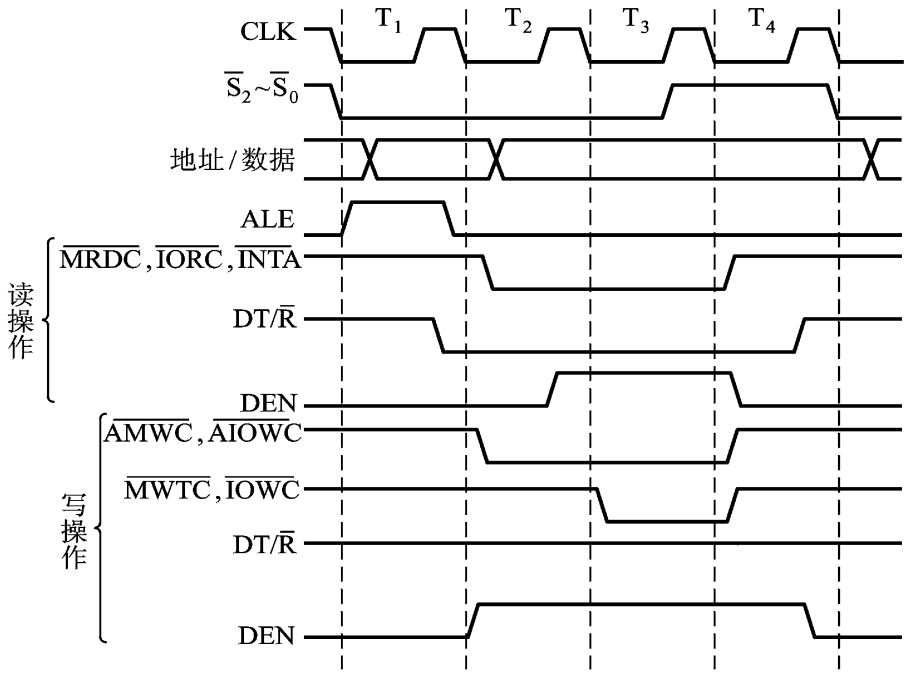


图 5.13 8288的工作时序

出信号以及控制输出信号。

1. 输入信号

状态输入信号$\overline{S_2}$,$\overline{S_1}$,$\overline{S_0}$由 8086送来。在最小模式系统中,控制信号$\overline{INTA}$,ALE,$\overline{DEN}$,DT/$\overline{R}$,M/$\overline{IO}$和$\overline{WR}$是直接从 8086的引脚 24～29输出的,它们提供了中断响应信号、地址锁存控制信号以及总线收发器的控制信号等。在最大模式系统中,上面这些信息隐含在状态信号$\overline{S_2}$,$\overline{S_1}$,$\overline{S_0}$中。8288总线控制器接收 8086发出的$\overline{S_2}$,$\overline{S_1}$,$\overline{S_0}$后,发出相应的控制命令,并确定 CPU执行何种操作。状态信号$\overline{S_2}$,$\overline{S_1}$,$\overline{S_0}$的组合所代表的总线周期及其对应的命令输出,如表 5.6所示。

从表中关系不难看出,除$\overline{S_1}$=$\overline{S_0}$=1这种情况外,可用$\overline{S_2}$来区分是对 I/O端口还是对内存执行操作。若$\overline{S_2}$=0,表示是在 CPU和 I/O端口之间执行操作;若$\overline{S_2}$=1,则表示是在 CPU与内存之间执行操作。进一步还不难发现,当$\overline{S_1}$=0,表示执行的操作为读操作;而$\overline{S_1}$=1,表示执行的操作为写操作。

总线控制器 8288可工作于两种方式,即系统总线方式和 I/O总线方式。由引脚 IOB来进行选择。

如果 IOB接低电平,则 8288工作在系统总线方式。这种情况主要用在多处理器使用一组总线的系统中,当存储器和 I/O设备都是被多个处理器共享,此时总线仲裁逻辑电路向 8288的$\overline{AEN}$端送一个低电平来表示总线可供使用。例

IBM PC/XT中，8288就工作在系统总线方式。若 IOB接高电平，则 8288就工作在 I/O总线方式。此时所有 I/O命令，如 $\overline{IORC}$, $\overline{INTA}$, $\overline{IOWC}$, $\overline{AIOWC}$都是允许的，即处理器进行 I/O操作时，不需要仲裁逻辑电路送 $\overline{AEN}$信号。这种情况主要用在多处理器系统中，若外部设备都不是共享的，而是某些外部设备从属于某一个处理器，则采用 I/O总线方式比较合适。

表 5.6　$\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$的组合

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	8088总线周期	8288的命令输出
0	0	0	发中断响应信号	$\overline{INTA}$
0	0	1	读 I/O端口	$\overline{IORC}$
0	1	0	写 I/O端口	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	暂停	—
1	0	0	取指令代码	$\overline{MRDC}$
1	0	1	读内存	$\overline{WRDC}$
1	1	0	写内存	$\overline{MWTC}$, $\overline{AMWC}$
1	1	1	无源状态	—

CEN称为命令允许输入信号，若 CEN为高电平，则 8288处于正常工作状态；若 CEN为低电平，则 8288的所有命令信号端都处于无效电平。

CLK为时钟输入信号，用来确定 8288各信号的时序关系。

2. 命令输出信号

8288接收 CPU送来的状态信号 $\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$，发出相应的命令，以实现对存储器和 I/O接口的读/写操作。这些命令信号都是低电平有效。分别说明如下：

$\overline{MRDC}$为存储器读命令，此命令有效时，通知被选中的存储单元，把数据送到数据总线上。

$\overline{MWTC}$为存储器写命令，此命令有效时，把数据总线上的数据，写入到所选中的存储单元。

$\overline{AMWC}$为存储器超前写命令。此命令的功能和 $\overline{MWTC}$一样，只是超前一个时钟周期输出。

$\overline{IORC}$为 I/O读命令。此命令有效时，通知被选中的 I/O端口，把数据送到数据总线上。

$\overline{IOWC}$为 I/O写命令。此命令有效时，把数据总线上的数据，写入到所选中的 I/O端口。

$\overline{AIOWC}$为 I/O超前写命令。此命令的功能和 $\overline{IOWC}$一样,只是超前一个时钟周期输出。

$\overline{INTA}$为中断响应信号。此命令有效时,告诉请求中断的设备,它所发出的中断请求信号已被响应。

3. 控制输出信号

8288输出的控制信号有 ALE, DEN, DT/$\overline{R}$和 MCE/$\overline{PDEN}$。

ALE为地址锁存允许信号。前面已经介绍,8086有些引脚是分时复用的,例 AD_{15} ~AD_0, A_{19}/S_6 ~A_{16}/S_3,这些引脚在每个总线周期的 T_1状态输出地址信息,在 T_1状态 CPU通过 8288发出 ALE信号将出现的地址信号锁存至地址锁存器中。

DEN为数据总线允许信号。为了增加数据总线的驱动能力,往往要用到数据收发器,如在 IBM PC/XT中采用的是 74LS245,8288要发出 DEN送至数据收发器的使能输入端 (又称片选端),把数据收发器与数据总线接通。

DT/$\overline{R}$ 为数据发送/接收信号,用来控制数据传输方向。当执行写操作时,DT/$\overline{R}$为高电平;当执行读操作时,DT/$\overline{R}$为低电平。DT/$\overline{R}$接数据收发器的方向控制端。

MCE/$\overline{PDEN}$ 具有两种功能:当 8288工作于系统总线方式,并且系统中又使用了多个中断控制器 8259A构成的级联式中断系统,用 MCE作为级联允许信号。当 8288工作于 I/O总线方式时,用 $\overline{PDEN}$作为允许信号,允许数据收发器为 I/O总线使用。

如果 8288工作于系统总线方式,且系统中未采用级联中断,则 MCE/$\overline{PDEN}$信号不用。在 IBM PC/XT中就属这种情况。

5.4.2 最大模式下的读周期时序

在读周期时序开始之前,CPU将状态信号 $\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$按照对应操作设置好相应电平 (若是读存储器,则 $\overline{S_2}\,\overline{S_1}\,\overline{S_0}$ =101;若是读 I/O端口,则 $\overline{S_2}\,\overline{S_1}\,\overline{S_0}$ =001) 送给总线控制器 8288,由 8288产生出相应的控制信号 ALE, DEN, DT/$\overline{R}$和 $\overline{MRDC}$或 $\overline{IORC}$。读周期时序如图 5.14所示。

读周期时序分析如下:

(1) 在 T_1状态 CPU发出 20位地址信息 (若是 I/O端口只需 16位地址,则 A_{19} ~A_{16}为 0),即 A_{19}/S_6 ~A_{16}/S_3, AD_{15} ~AD_0引线上地址有效。总线控制器 8288发出地址锁存信号 ALE,将复用线上出现的地址信息锁存到地址锁存器

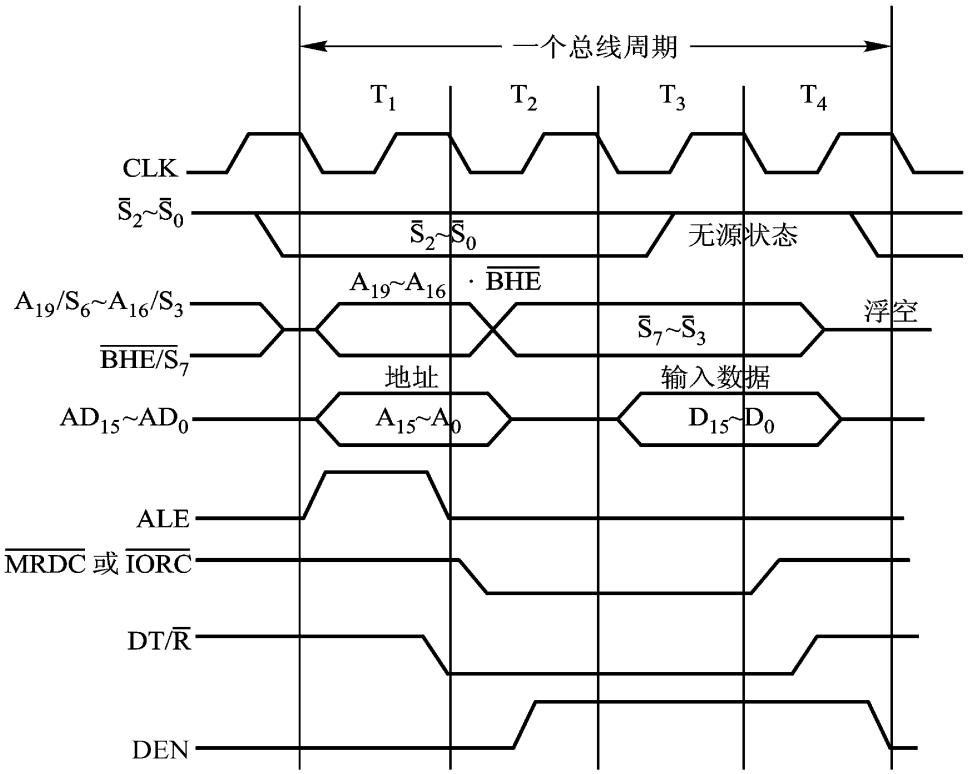


图 5.14　8086最大模式下读周期时序

中。此外，8288还为数据收发器发出控制数据传送方向的信号 DT/R，因是读周期，此信号应是低电平有效。

(2) 在 T_2状态　CPU通过复用线 A_{19}/S_6～A_{16}/S_3输出状态信息 S_6～S_3。总线控制器 8288发出数据收发器允许信号 DEN，注意此信号为高电平有效，允许数据总线与 8086的数据总线接通。8288在 T_2状态还发出存储器读信号$\overline{MRDC}$或 I/O端口读信号 $\overline{IORC}$，送到存储器或 I/O端口。此外，CPU还将 AD_{15}～AD_0置为高阻状态，为 T_4状态读入数据做好准备。

(3) 在 T_3状态　若存储器或 I/O端口的工作速度能与 CPU的速度相匹配，则读取的数据已送到数据总线上。但一般情况下，I/O端口工作速度较慢，往往要插入等待状态。例如在 IBM PC/XT中，对 I/O端口的读/写操作均插入了一个 T_W状态。与最小模式时序类似，也是在 T_3状态开始的下降沿去采样 READY线，若为低电平，在 T_3与 T_4状态之间插入一个或几个 T_W；若 READY为高电平，则在 T_3状态结束后，进入 T_4状态。在 T_3状态下，$\overline{S_2}$，$\overline{S_1}$，$\overline{S_0}$进入无源状态。

(4) 在 T_4状态　在 T_4状态开始的时钟下降沿，把数据总线上的数据读入 CPU。在 T_4状态，DEN，DT/R及读命令变为无效，至此读周期结束。准备启动

一个新的总线周期。

5.4.3 最大模式下的写周期时序

在写周期时序开始之前，$\overline{S_2}$，$\overline{S_1}$，$\overline{S_0}$按照对应操作设置好相应电平（若是写存储器，$\overline{S_2}\,\overline{S_1}\,\overline{S_0}$=110；若是写 I/O端口，则$\overline{S_2}\,\overline{S_1}\,\overline{S_0}$=010），送总线控制器 8288。在写周期内，8288除输出相应的控制及命令信号 ALE, DEN, DT/$\overline{R}$, $\overline{MWTC}$或 $\overline{IOWC}$之外，还将产生超前的存储器写信号$\overline{AMWC}$或超前的 I/O端口写信号$\overline{AIOWC}$。这两个超前写信号比普通的写信号$\overline{MWTC}$（存储器写）和 $\overline{IOWC}$（I/O端口写）要超前一个时钟周期。

写周期时序如图 5.15所示，分析如下：

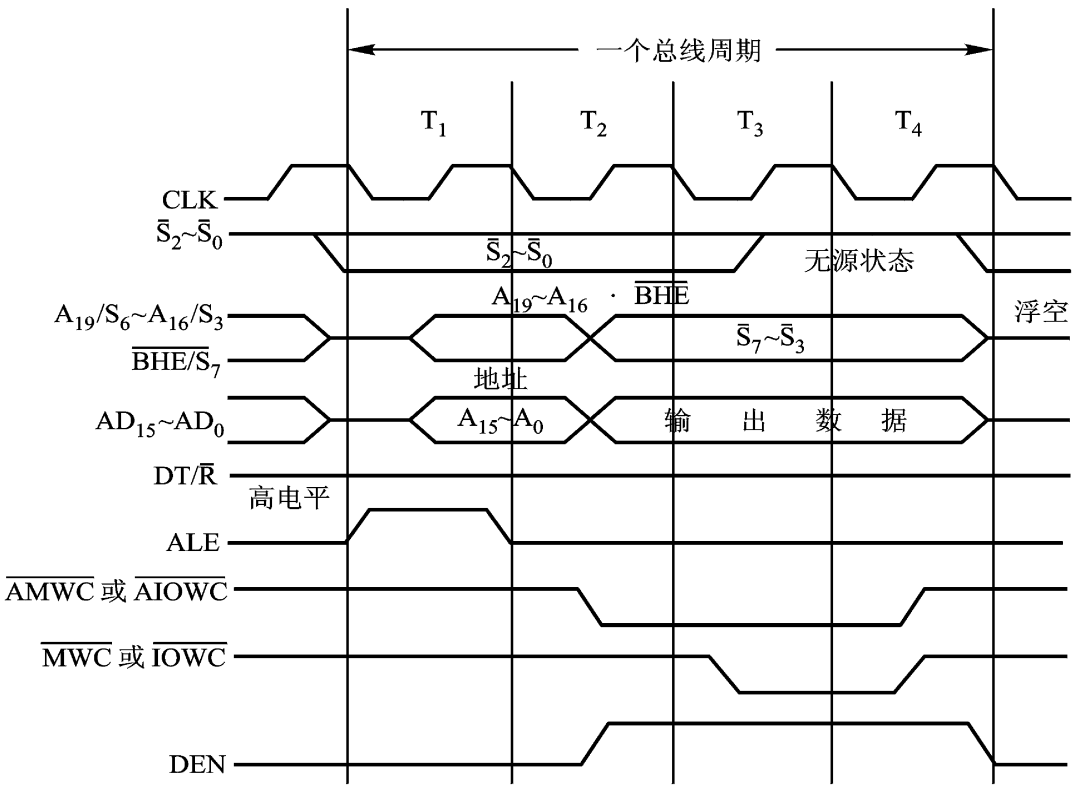


图 5.15 8086最大模式下的写周期时序

⑴ 在 T_1状态 CPU通过 A_{19}/S_6～A_{16}/S_3，AD_{15}～AD_0输出 20位地址（若是 I/O端口写，只需输出 16位地址）。总线控制器 8288输出地址锁存信号 ALE，将复用线上的地址信号锁存到 8282中。8288还输出控制数据传送方向的信号 DT/$\overline{R}$，此信号应为高电平，表示是写周期。

⑵ 在 T_2状态　CPU通过 $A_{19}/\overline{S_6}$～$A_{16}/\overline{S_3}$复用线输出状态信息 $\overline{S_6}$～$\overline{S_3}$。总线控制器送出有效的 DEN信号,允许 8086的数据总线接到系统的数据总线上,于是 CPU通过 AD_{15}～AD_0把数据送到数据总线上,与此同时,超前写信号$\overline{AMWC}$或$\overline{AIOWC}$变为有效。

⑶ T_3状态　8288使普通写信号$\overline{MWTC}$或$\overline{IOWC}$变为有效。从时序上看,普通写信号比起超前写信号慢了整整一个时钟周期,这样,对那些工作速度较慢的存储器或 I/O端口来说,利用超前写信号就可以额外得到一个时钟周期的时间。在 T_3状态,$\overline{S_2}$,$\overline{S_1}$,$\overline{S_0}$进入无源状态。

⑷ 在 T_4状态　超前写信号及普通写信号被撤销,DEN和 DT/$\overline{R}$变成无效。在此之前,写操作过程已经完成。

若存储器或外设来不及在指定的时间内完成写操作,可利用 READY信号在 T_3与 T_4状态之间插入一个或几个等待状 T_W。

5.4.4　最大模式下的总线请求/允许/释放操作

8086/8088在最大方式下提供的总线控制联络信号不再是 HOLD和 HLDA,而是把这两个引脚变成功能更加完善的两个具有双向传输信号的引脚$\overline{RQ}/\overline{GT_0}$和$\overline{RQ}/\overline{GT_1}$,称之为总线请求/总线允许/总线释放信号。它们可以分别连接到两个其他的总线主模块。在最大方式下,可发出总线请求的总线主模块包括协处理器和 DMA控制器等。

8086/8088最大方式下的总线请求/总线允许/总线释放的操作时序如图 5.16所示。

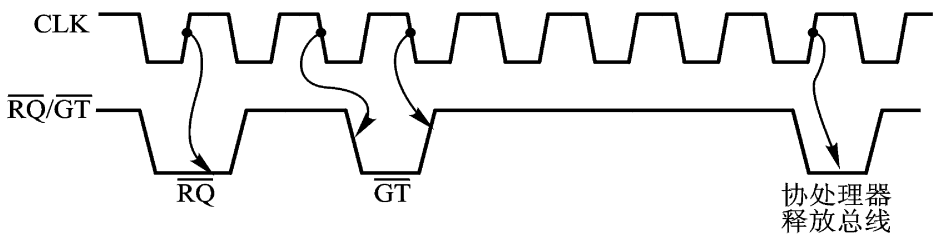


图 5.16　8086/8088最大方式下的总线请求/总线允许/总线释放的操作时序

由图可见:CPU在每个时钟周期的上升沿处对$\overline{RQ}/\overline{GT}$引脚进行检测,当检测到外部向 CPU送来一个“请求”负脉冲时(宽度为一个时钟周期),则在下一个 T_4状态或 T_1状态从同一引脚上由 CPU向请求总线使用权的主模块回发一个“允许”负脉冲(宽度仍为一个时钟周期),这时的全部具有三态的输出线——包

括 AD_{15} ～AD_0，A_{19}/S_6 ～A_{16}/S_3，和 $\overline{RD}$，$\overline{LOCK}$，$\overline{S_2}$，$\overline{S_1}$，$\overline{S_0}$，$\overline{BHE}/S_7$都进入高阻状态，CPU暂时与总线断开。

外部主模块得到总线控制权后，可以对总线占用一个或几个总线周期。当外部主模块准备释放总线时，便又从$\overline{RQ}/\overline{GT}$线上向 CPU发一个“释放”负脉冲（其宽度仍为一个时钟周期）。CPU检测到释放脉冲后，于下一个时钟周期收回对总线的控制权。

由$\overline{RQ}/\overline{GT}$线上的三个负脉冲，即请求—允许—释放，就构成了最大方式下的总线请求/总线允许/总线释放操作。三个脉冲虽都是负的，宽度也都为一个时钟周期。但是，它们的传输方向并不相同。

对于此操作，有下面几点需注意：

⑴ 8086/8088有两条$\overline{RQ}/\overline{GT_0}$和$\overline{RQ}/\overline{GT_1}$，其功能完全相同，但前者的优先级高于后者。当两条引脚都同时向 CPU发总线请求时，CPU将会在$\overline{RQ}/\overline{GT_0}$上先发允许信号，等到 CPU再次得到总线控制权时，才去响应$\overline{RQ}/\overline{GT_1}$引脚上的请求。不过，当接于$\overline{RQ}/\overline{GT_1}$上的总线主模块已得到了总线的控制权时，也只有等到该主模块释放了总线，CPU收回了总线控制权后，才会去响应$\overline{RQ}/\overline{GT_0}$引脚上的总线请求。

⑵ 与最小模式下执行总线保持请求/保持响应操作一样，8086/8088通过$\overline{RQ}/\overline{GT}$发出响应负脉冲，CPU让出了对总线的控制权后，CPU内部的 EU仍可继续执行指令队列中的指令，直到遇到一条需执行总线操作周期的指令为止。另外，当 CPU收到其他主模块发出的释放脉冲后，也并不是立即恢复驱动总线的，而是要等遇到执行涉及总线的指令时才恢复驱动总线。和 HLDA控制线不同的是：$\overline{RQ}/\overline{GT_0}$和$\overline{RQ}/\overline{GT_1}$都设置了上拉电阻与电源相连，如果系统中不用它们，则可将之悬空。

思考题与习题

5.1 解释下列名词和术语：

⑴ 时钟周期、总线周期、指令周期

⑵ 三态、数据收发器、地址锁存器

⑶ 最小工作模式和最大工作模式

⑷ 总线保持请求、总线保持响应

⑸ 总线控制器 8288

5.2 基本的总线周期有哪几种？

5.3 8088和 8086有何区别？

5.4 8086是怎样解决地址线和数据线的复用问题的？ALE信号有效电平是什么？

5.5 试分析 8086最小模式下的读周期时序。

5.6 试分析 8086最小模式下的写周期时序。

5.7 试分析 8086中断响应周期时序。

5.8 试分析 8086的复位时序。

5.9 分析 8086总线保持请求与保持响应的时序。

第6章

半导体存储器

本章将主要介绍各种半导体存储器的结构、工作原理和主要外特性，讲述存储器与系统接口的原理。

6.1 内存和外存

计算机系统中，按存储器与 CPU 的关系，分为内存和外存。内存是内部存储器的简称，又称主存。内存直接与控制器、运算器相连接，是计算机的组成部分。已编制的程序、需要处理的数据、处理过程中产生的中间结果等均存放于内存中。计算机工作的过程就是不断地由控制器从内存取出指令，然后分析指令、执行指令的过程。因此内存应具有快速存取的能力以保证计算机的工作速度。

计算机硬件系统中的外存即外部存储器，也称辅存。外存不直接与 CPU 相连接，而是通过 I/O 接口与 CPU 连接，其主要特点是大容量，往往达到几百兆以上。如一张 CD 盘为 650 MB，一张 DVD 光盘为 4.7 GB，硬盘一般为几十 GB 以上。

计算机内存一般都以半导体存储器作为存储介质。外存的存储介质包括磁带、磁盘、光盘等，也可以由半导体存储器构成。近年来，大容量半导体存储器如 Flash 存储器（闪存）价格迅速下降，用闪存制成的“优盘”成为了一种很受欢迎的外存。

6.2 半导体存储器

6.2.1 半导体存储器的分类

1. RAM 与 ROM

半导体存储器按存储信息的特性可分为随机存储器（Random Access Memory）和只读存储器（Read Only Memory）两类。

随机存储器又称为读写存储器，它在计算机基本读、写周期内可完成读或写数据的操作。只读存储器在计算机基本读周期内可完成数据的读操作，但不具备数据写入功能，或不能在计算机基本写周期内完成写操作。换言之，随机存储器可以“随时”进行读、写操作，而只读存储器只能“随时”读出数据、不能写入或不能“随时”写入数据（即写入操作需较长的时间）。

对于有些种类的只读存储器而言，可以在脱机状态或较慢的速度下将数据写入芯片，这种写入过程被称为对 ROM 芯片的编程。

ROM 中存储的信息具有非易失性，芯片断电后所存的信息不会改变和消失。而 RAM 必须保持供电，否则其保存的信息将消失。

目前，按半导体存储器制造工艺主要可分为 NMOS、CMOS、TTL、ECL、砷化镓等。TTL 工艺制造的存储器速度较高，但功耗较大，集成度不高。ECL 存储器的优点是速度快，砷化镓的存储器更快，但功耗大、价格高、集成度低。以 CMOS 工艺制造的半导体存储器具有集成度高、功耗低的特点，读写速度达几纳秒至几十纳秒，随着工艺水平的提高，读写速度还在不断提高。以 CMOS 工艺制造的半导体存储器是应用得最多的半导体存储器。

2. RAM 的分类

静态 RAM（SRAM，Static RAM）：SRAM 的记忆单元是具有两种稳定状态的触发器，以其中一个状态表示“1”，另一个状态表示“0”。SRAM 的读写次数不影响其寿命，可无限次读写。当保持 SRAM 的电源供给的情况下，其内容不会丢失。但如果断开 SRAM 的电源，其内容将全部丢失。

动态 RAM（DRAM，Dynamic RAM）：DRAM 的记忆单元是 MOS 管的栅极与衬底之间的分布电容，以该电容存储电荷的多少来表示“0”和“1”。DRAM 的一个 bit 数据可由一个 MOS 管构成，因此具有集成度高、功好低的特点。

DRAM 的一个缺点是需要刷新。芯片中存储的信息会因为电容的漏电而消失，因此应确保在信息丢失以前进行刷新。所谓刷新就是对原来存储的信息进

行重新写入,因此使用 DRAM 的存储体需要设置刷新电路。刷新周期随芯片的型号而不同,一般为 1至几十毫秒。DRAM 的另一个缺点是速度比 SRAM 慢。

目前 PC 中的内存都采用 DRAM,因为它价格低,容量大,耗电少。

为了克服动态 RAM 需设置刷新电路的缺点,又开发了能够自动刷新的 DRAM,芯片中集成了动态 RAM 和自动刷新控制电路。

3. ROM 的分类

掩膜 ROM:掩膜 ROM 简称 ROM,是由芯片制造的最后一道掩模工艺来控制写入信息。因此这种 ROM 的数据由生产厂家在芯片设计掩膜时确定,产品一旦生产出来其内容就不可改变。由于集成电路生产的特点,要求一个批次的掩膜 ROM 必须达到一定的数量(若干个晶圆)才能生产,否则将极不经济。掩膜 ROM 既可用双极性工艺实现,也可以用 CMOS 工艺实现。掩膜 ROM 的电路简单,集成度高,大批量生产时价格便宜。掩膜 ROM 一般用于存放计算机中固定的程序或数据,如引导程序、BASIC 解释程序、显示、打印字符表、汉字字库等。

PROM(Programmable ROM):可由用户一次性写入的 ROM。如熔丝 PROM,新的芯片中所有数据单元的内容都为 1,用户将需要改为 0的 bit以较大的电流将熔丝烧断即实现了数据写入。这种数据的写入是不可逆的,即一旦被写入 0则不可能重写为 1。因此熔丝 PROM 是一次性可编程的 ROM。

EPROM(Erasable Programmable ROM):可擦除的可编程只读存储器。如紫外线擦除型的可编程只读存储器,20世纪 80年代到 20世纪 90年代曾经广泛应用。这种芯片的上面有一个透明窗口,紫外线照射后能擦除芯片内的全部内容。当需要改写 EPROM 芯片的内容时,应先将 EPROM 芯片放入紫外线擦除器擦除芯片的全部内容,然后对芯片重新编程。

E^2PROM(Electrically Erasable Programmable ROM):也称为 EEPROM,是可以电擦除的可编程只读存储器。由于能以电信号擦除数据,并且可以对单个存储单元擦除和写入(编程),因此使用十分方便,并可以实现在系统擦除和写入。

闪速存储器(Flash Memory):闪速存储器是新型非易失性存储器,在系统电可重写。它与 E^2PROM 的一个区别是 E^2PROM 可按字节擦除和写入,而闪速存储器只能分块进行电擦除。目前闪速存储器产品的容量比 E^2 PROM 更大,价格更优,是一种很有前途的大容量存储器。

6.2.2 半导体存储器的主要技术指标

1. 存储容量

存储器的存储容量是表示存储器大小的指标,通常以存储器单元数与存储器字长之积表示。每个存储器单元可存储若干个二进制位,二进制位的长度称为存储器字长。存储器字长一般与数据线的位数相等。每个存储器单元具有唯一的地址。因此,通常存储容量越大,地址线的位数越多。

由于存储器的容量一般都比较大,因此常以 K表示 2^{10},以 M表示 2^{20},G表示 2^{30}。如 256 KB等于 $256\times2^{10}\times8$ bit,32 MB等于 $32\times2^{20}\times8$ bit。

2. 最大存取时间

从接受地址码、地址译码、选中存储单元,到该单元读/写操作完成所需要的总时间被称为存取时间。存储器的存取时间越短,工作速度就越快,价格也越高。存储器厂家一般给出某种芯片的最大存取时间。设计计算机的存储器系统时,为读/写操作留出的时间应大于存储器最大存取时间,一般还应有一定的富余量以确保存储器读写操作的可靠性。

3. 功耗

存储器被加上的电压与流入的电流之积是存储器的功耗。存储器的功耗又分为操作功耗和维持功耗(或备用功耗)。前者是存储器被选中进行其中某个单元的读/写操作时的功耗,后者是存储器未被选中时的功耗。当芯片被选中时,地址译码、读写控制等电路工作,有一个单元被选中作读或写操作,因此操作功耗比维持功耗大。

虽然存储器中的存储单元很多,但由于 CMOS电路的在不发生电平翻转时的功耗几乎为零,因此 CMOS存储器的维持功耗很低。而 TTL工艺的存储器虽然速度快,但功耗大,当需要的存储器容量大的时候功耗就成为系统的一个严重问题。随着 CMOS工艺的提高,其工作速度也进一步提高,能够满足系统的要求。因此,目前 CMOS存储器已成为应用最多的存储器,TTL工艺的存储器基本被淘汰。

4. 可靠性

可靠性一般是指存储器对温度、电磁场等环境变化的抵抗能力和工作寿命。半导体存储器由于采用大规模集成电路工艺,具有较高的可靠性。

6.3 随机存储器 RAM

6.3.1 基本结构

1. RAM 芯片内部结构

典型的 RAM 芯片内部结构如图 6.1所示。半导体存储器由地址译码、存储矩阵、读写控制逻辑、三态双向缓冲器等部分组成。

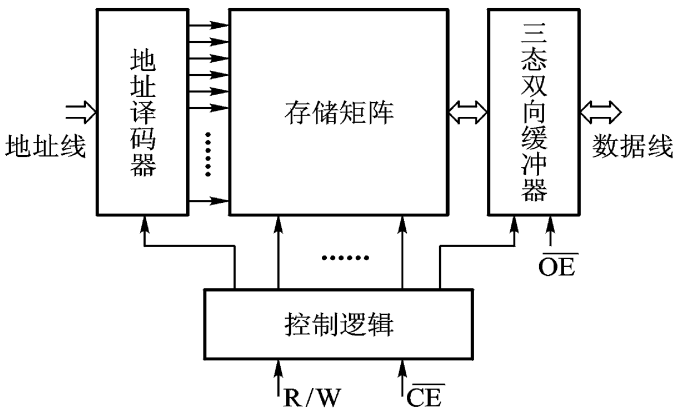


图 6.1 RAM 芯片内部结构

每个基本存储电路可存储一位二进制数,由若干个基本存储电路组成一个存储器单元。每个存储器单元具有唯一的地址。存储单元按一定的结构排列,如线列结构、行列结构等,称为存储矩阵。

地址译码器接受来自 CPU 的地址信息,译码后产生选中信号,选中某个存储单元。译码一般有线译码、行列译码等模式。

存储器的字长是指每个存储单元所包含的二进制数的位数,数据缓冲器应与存储器的字长相同。

读写控制逻辑接受来自 CPU 的控制信号,对地址译码、存储矩阵、数据缓冲等进行控制。通常控制信号有片选信号$\overline{CE}$、读信号$\overline{RD}$、写信号$\overline{WE}$等。

当系统由多个存储器芯片组成时,CPU 的地址信号经译码器译码后产生片选信号送存储器的$\overline{CE}$,用来控制存储器内的译码电路。当$\overline{CE}$不等于 0时,本存储器芯片未被选中,存储器内的译码电路不工作,不选中存储矩阵中的任何单元。当$\overline{CE}$等于 0时,本存储器芯片被选中,存储器内的译码电路根据送到的地址信息选中存储矩阵中的某个单元。

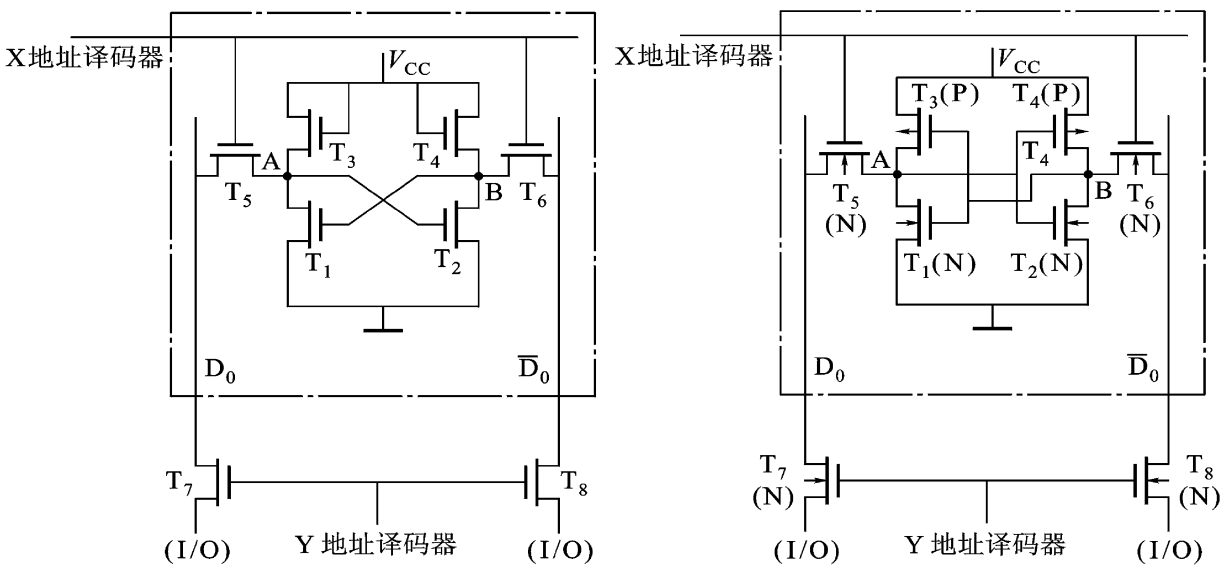
2. SRAM 基本存储电路

SRAM的基本存储电路是一个双稳态触发器，可以存储一位二进制数据。典型的 NMOS静态存储器基本存储电路如图 6.2a所示。

典型的 CMOS静态存储器基本存储电路如图 6.2b所示，下面分析其基本存储电路工作原理。

T_1、T_3是一对互补的 MOS管，T_2、T_4是一对互补的 MOS管。当外部写入时，I/O线上的数据使 A点为低、B点为高，则 T_4导通时 T_2关断，同时 T_1导通时 T_3关断，A点为低。这是一种稳定状态，当写入操作结束，状态不变。当外部写入的数据是 B点为低、A点为高，则导通、关断的情况相反。读出时，A、B两点的状态被送到 I/O线上。

由于 T_1、T_3总是一个导通、一个关断，同时 T_2、T_4也是一个导通、一个关断。这样，除了在状态改变时有电流通过，其他时间基本没有电流，所以 CMOS的存储器功耗很低。



(a) NMOS型 SRAM 基本存储电路　　(b) CMOS型 SRAM 的基本存储电路

图 6.2

3. 单管 DRAM 基本存储电路

单管 DRAM基本存储电路如图 6.3所示。信息存放在电容 C_S中，T_S是选通开关。当 C_S中充有电荷时表示信息 “1”，当 C_S中没有电荷时表示信息 “0”。

写入数据时，地址选通信号为高，T_S导通，数据经数据线向 C_S充电或放电从

而完成写操作。

数据读出时，地址选通信号为高，T_S导通，C_S上的电压送到数据线。C_S上表示“1”的信号电平只有 0.2 V左右，当 T_S导通时数据线上的分布电容还要被充电，使数据线上的电平更低，因此，需要经放大电路后输出。同时，由于读出时使 C_S中的电荷被部分泄放，对于 C_S来说这是一种破坏性读出，因此需要将经放大电路放大后的数据重写入。

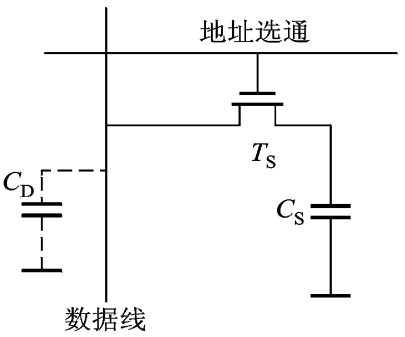


图 6.3 单管 DRAM基本存储电路

当 DRAM不作读写操作时，C_S中的电荷也会由于 T_S的漏电或充电而改变，在一定的时间之后将丢失信息。解决的方法是在信息丢失之前对 C_S中的数据进行放大后再重新写入即“刷新”。一般的刷新周期为 1～2 ms，即 1～2 ms内应对每个 C_S进行一次重写。

6.3.2 典型 SRAM芯片

1. IDT6116

IDT6116是一种 2 048×8 bit的 CMOS工艺的静态 RAM，内部功能框图与引脚图见图 6.4a和图 6.4b。主要指标如下：

(1) 最大存取时间：早期的 6116速度较低，近几年的产品性能有所提高。例如 IDT6116SA 15/20/25/35/45的读、写时间分别为 15、20、25、35、45 ns。

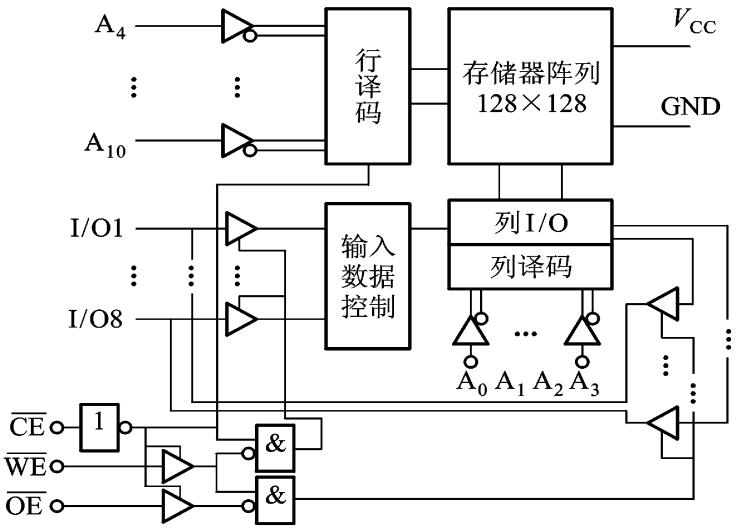
(2) 功耗：随着芯片最大存取时间的不同，操作时 I_{CC}为 80～150 mA，全待用模式（Full standby power mode）时 I_{CC}为 2 mA。

(3) TTL兼容。

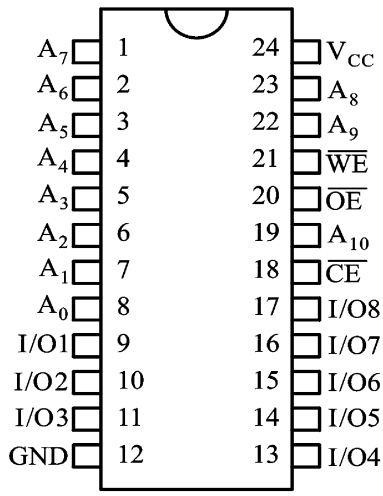
2. HM62256

HM62256是 32 K×8 bit的静态 RAM，0.8 μm工艺 CMOS工艺。具有高速、低功耗的特点。读、写时间相同，最大存取速度分 45、55、70、85 ns等几挡。单一的 5 V电源，备用状态功耗 1 μW，操作时 25mW。全静态，无需时钟或选通信号，双向 I/O端口，三态输出，与 TTL兼容。HM62256有 28条引脚，引脚图见图 6.5。

与其同一系列的还有 HM6264、62128、62512、621000、621400、628511、6216255等，它们的容量分别为 8 K×8 bit、16 K×8 bit、64 K×8 bit、4 M×1 bit、128 K×8 bit、512 K×8 bit、256 K×16 bit。它们的控制信号基本相同。



(a) 6116内部功能框图



(b) 6116引脚图

图 6.4

3. HM628511HC

HM628511是 CMOS工艺的 4Mbit静态 RAM,结构为 512 K×8 bit。

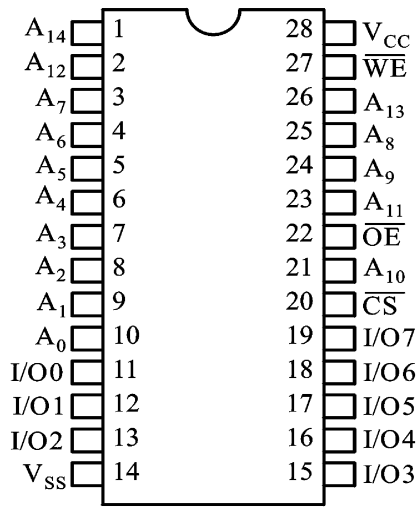


图 6.5　62256引脚图 (顶视图)

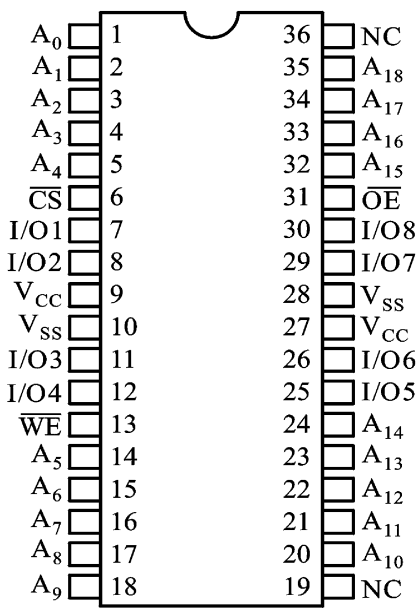


图 6.6　HM62628511的引脚

主要特点：

高速,读写时间 10 ns或 12 ns;低功耗,操作时 130～140 mA,备用时 5 mA。

使用单电源,电压为 5 V。

引脚说明:

HM628511为 36脚 SOJ封装,引脚图见图 6.6,图 6.7是内部结构图。

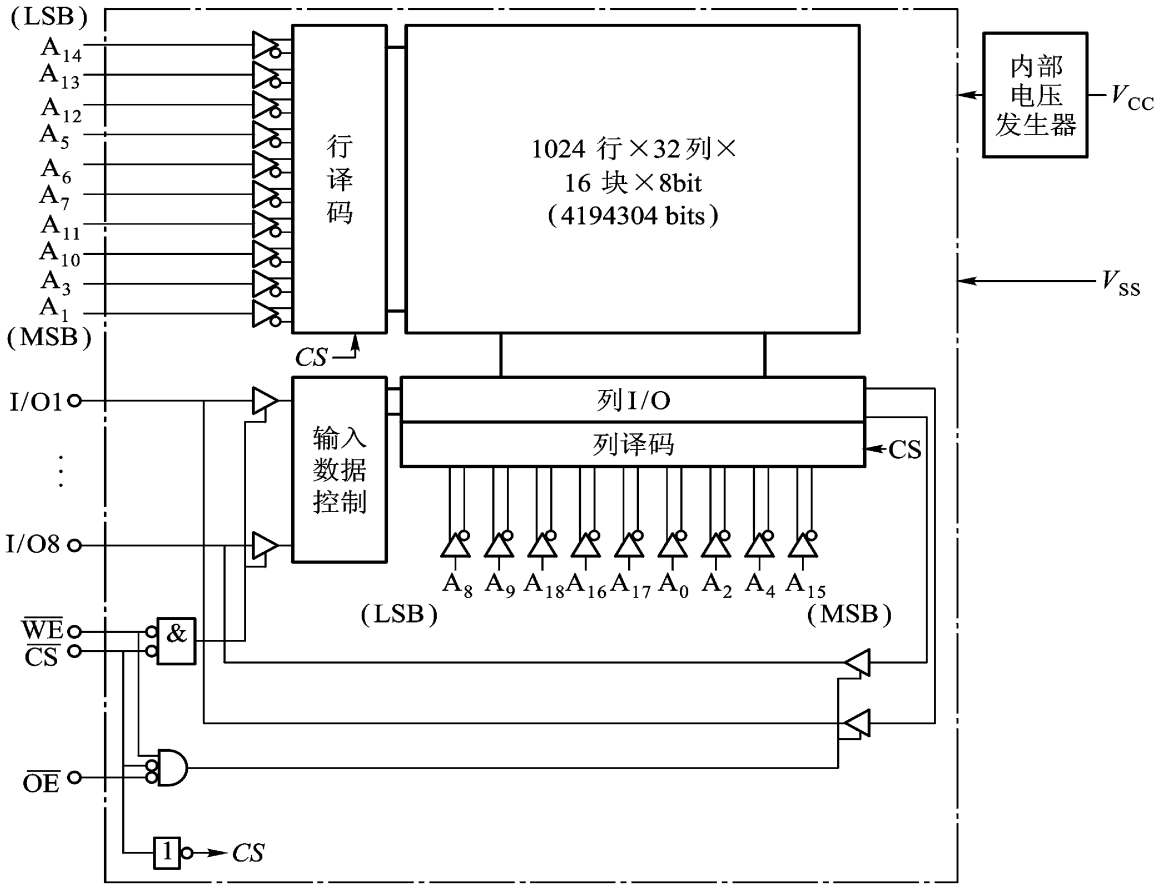


图 6.7 HM628511的内部结构

HM628511有 A_{18} ~A_0共 19条地址线,数据线为 I/O_8 ~I/O_1。$\overline{CS}$为片选,$\overline{WE}$为写允许,$\overline{OE}$为输出允许。

4. HM62W16255

HM62W16255是 CMOS工艺的 4Mbit静态 RAM,结构为 256 K×16 bit。电源电压为 3.3 V。主要特点:高速 10 ns/12 ns;低功耗,操作时 145 mA/130 mA,备用时 5 mA。HM62W16255的引脚图见图 6.8。

6.3.3 典型 DRAM 芯片

DRAM由于集成度高、功耗低、价格低,得到广泛的应用。DRAM的发展很快,单片容量越来越大,表 6.1列出了部分 DRAM芯片的型号、容量和结构。

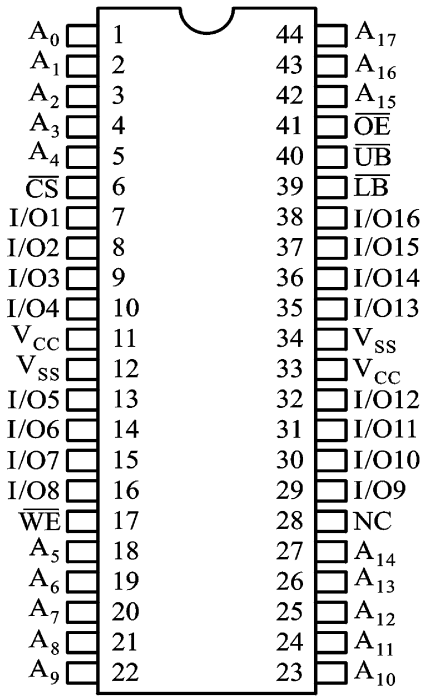


图 6.8 HM62W16255的引脚图（顶视图）

表 6.1 常见 DRAM 芯片

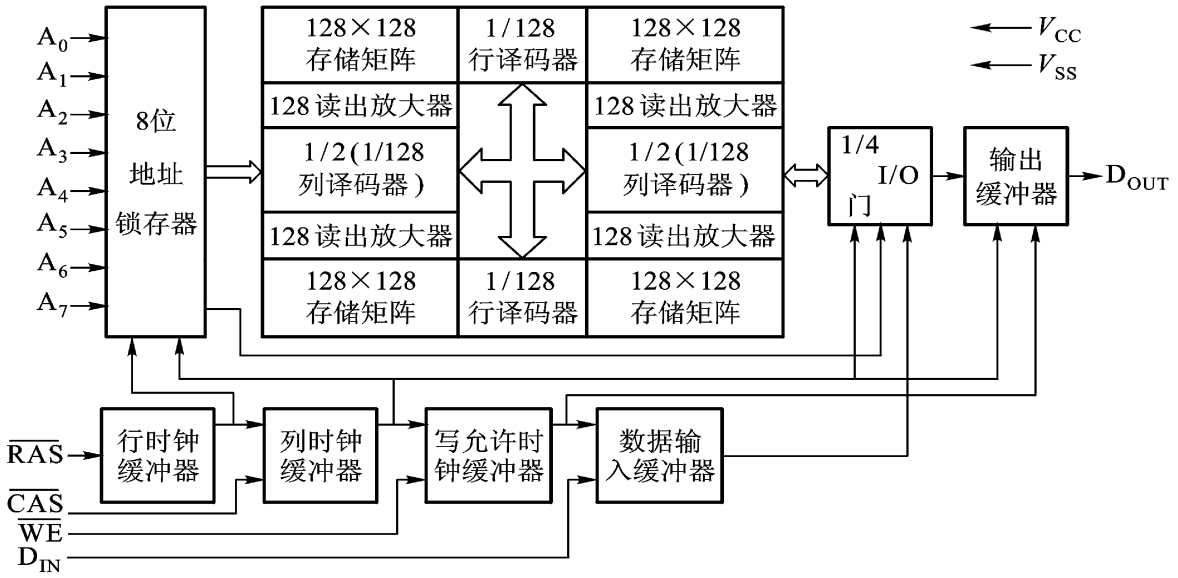
DRAM 型号	容量/bit	结构
2164	64 K	64 K×1 bit
21256	256 K	256 K×1 bit
21464	256 K	64 K×4 bit
421000	1 M	1 M×1 bit
424256	1 M	256 K×4 bit
44100	4 M	4 M×1 bit
44400	4 M	1 M×4 bit
44160	4 M	256×16 bit
416800	16 M	8 M×2 bit
416400	16 M	4 M×4 bit
416160	16 M	1 M×16 bit

1. 2164

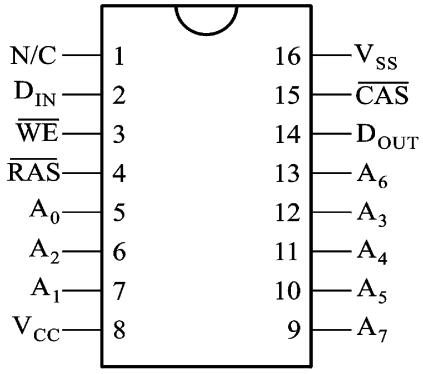
Intel 2164是 64 K×1 bit的 DRAM，是 Intel公司的早期产品，当时 IBM 公司的 PC使用该芯片作为其内存。

对于 64 K的存储空间应该有 16位的地址信号，Intel 2164的地址线只有 8

位，16位的地址信号分为行地址和列地址，分两次送入芯片。这样的设计，减少了引脚数，降低了成本。缺点是地址译码电路变得复杂，降低了工作速度。图6.9a是2164的内部结构图，存储矩阵为128×128×4 bit，还包括读出放大器与I/O门控制电路、行地址锁存器、行地址译码器、列地址锁存器、列地址译码器、数据输入缓冲器、输出缓冲器、行时钟缓冲器、列时钟缓冲器、写允许时钟缓冲器等。图6.9b是其引脚图。



(a) 2164的内部结构



(b) 2164的引脚图

图 6.9

主要特征

存取时间150～200 ns，操作时的功耗275 mW，备用时27.5 mW，5 V单一电源，每次同时刷新512个存储单元（512×1 bit），刷新128次可将全部单元刷

新一遍。应在 2 ms内将全部单元刷新一遍,以确保数据不丢失。

读/写操作

先由$\overline{RAS}$信号将地址线输入的 8位行地址 (例如 A_0 ~A_7)锁存到内部行地址寄存器,再由$\overline{CAS}$信号将地址线输入的 8位列地址 (例如 A_8 ~A_{15})锁存到内部列地址寄存器,选中一个存储单元,由$\overline{WE}$决定读或写操作。由于动态存储器读出时须预充电,因此每次读/写操作均可进行一次刷新,刷新 4个矩阵中的 128×4 bit。

刷新操作

当芯片的$\overline{RAS}$为低,动态存储器对部分单元进行刷新操作。2164内部由 4个 128×128的矩阵组成,刷新操作时 A_7不用,行地址由 A_0 ~A_6送入,4个矩阵中的 128×4 bit同时刷新。因此只要 128次 (每次的地址不同)就能完成全部的刷新操作。IBM PC中,定时器 8253的 1号通道每 15 μs向 4号 DMA控制器请求,由该控制器送出刷新地址,进行一次刷新操作。完成全部的刷新操作的时间为 128×15 μs。

2. 414256

414256的内部组成如图 6.10所示。

414256的基本组成是 512×512×4 bit的存储器阵列,还包括读出放大器与 I/O门控制电路、行地址缓冲器、行地址译码器、列地址缓冲器、列地址译码器、数据输入/输出缓冲器、刷新控制/计数器以及时钟发生器等。

地址信号线同样只有一半的位数,由行地址和列地址分两次输入。

存储器访问时,首先由$\overline{RAS}$信号锁存由地址线 A_0 ~A_8输入的 9位行地址,然后再由$\overline{CAS}$信号锁存由地址线 A_0 ~A_8输入的 9位列地址,经译码选中某一存储单元,在读/写控制信号$\overline{WE}$的控制下,可对该单元的 4位数据进行读出或者写入。

由于动态存储器读出时须预充电,因此每次读/写操作均可进行一次刷新。414256必须每 8 ms刷新一次,414256每 64 ms进行一次快速刷新。刷新时通过在 512个行地址间按顺序循环进行刷新,可以分散刷新,也可以连续刷新。分散刷新是指每隔一定的时间刷新一行;连续刷新也称猝发方式刷新,该方式对 512行进行连续刷新。刷新地址可以由外部输入,也可以使用内部刷新控制器产生刷新地址。

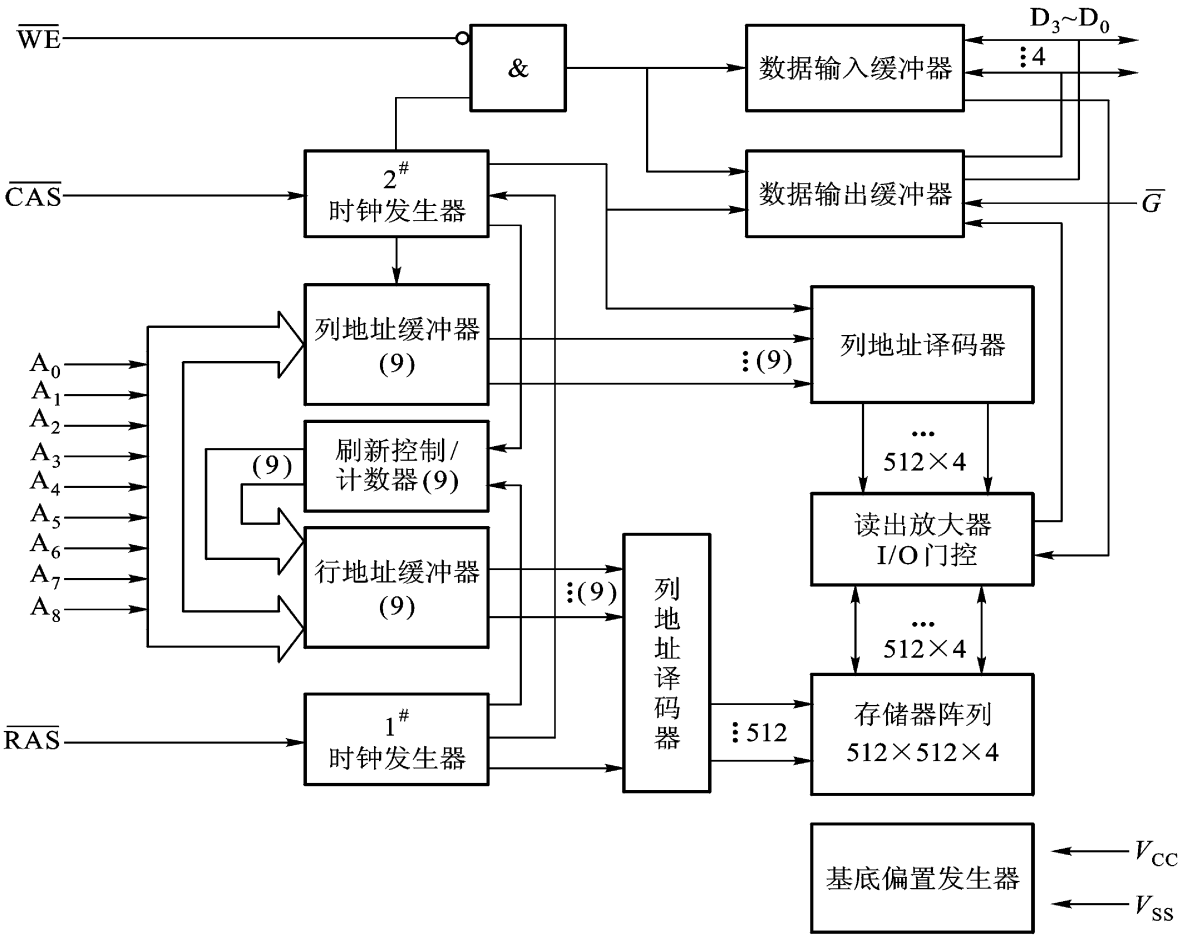


图 6.10 414256内部结构

6.4 只读存储器

只读存储器包括掩膜 ROM、PROM、EPROM、E^2PROM、Flash Memory等，本节重点介绍后3种类型的典型芯片。

6.4.1 EPROM

1. EPROM的工作原理

下面说明NMOS型EPROM的工作原理。图6.11为一个N沟道浮栅雪崩注入MOS管，有一个浮栅和一个控制栅。控制栅与行选信号相连，用于地址选中，浮栅被置于SiO_2层内，与四周绝缘。在较高的编程电压作用下，电荷由控制

栅进入浮栅。浮栅上积存的电荷引起 MOS管导通阈值电压的改变。浮栅上无电荷时 MOS管导通阈值电压低,浮栅上有电荷时 MOS管导通阈值电压高。

当以紫外线照射时,浮栅中的电荷以光电子的形式释放,从而实现信息的擦除。为此,采用特殊的封装形式,在双列直插封装的顶部设有一个圆形石英玻璃窗口,以使紫外线能够照射到芯片上。紫外线的波长为 2.537×10^{-7} ?,需要的照射能量为 15 J/cm^2。

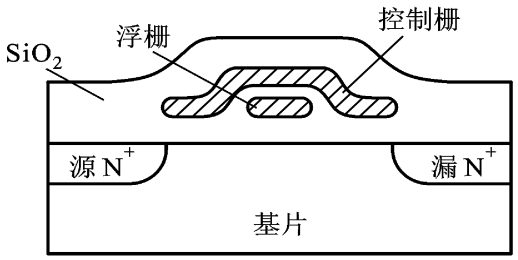


图 6.11　N沟道浮栅雪崩注入 MOS管

2. 典型芯片

典型 NMOS工艺的 EPROM芯片有 2708、2716、2732等,容量分别为 1 K×8 bit、2 K×8 bit、4 K×8 bit,24脚封装。HMOS工艺的 EPROM芯片有 2764、27128、27256、27512等,容量分别为 8 K×8 bit、16 K×8 bit、32 K×8 bit、64 K×8 bit,这几种芯片的引脚兼容,都是 28脚封装。还有 27C128、27C256、27C512等 CMOS工艺的 EPROM,具有低功耗的优点,可优先选用。

(1) 2764

2764的结构如图 6.12所示。表 6.2列出了 2764、27128、27256的引脚。其中 2764是 8KB的存储空间,因此地址线为 13条,由 A_0 ~A_{12}。NC为空脚。O_0

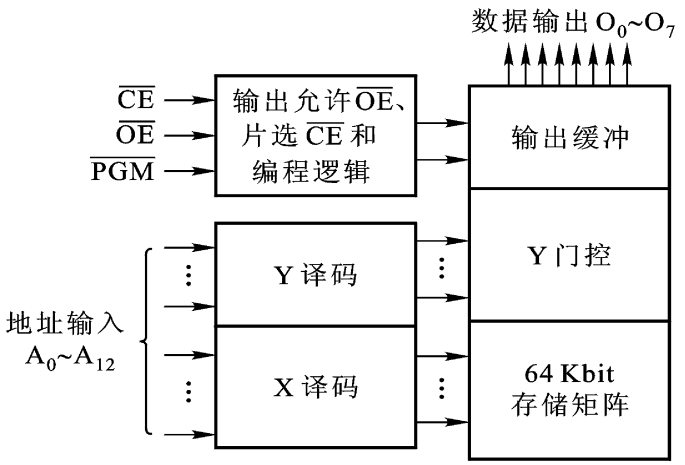


图 6.12　2764的内部结构

～O_7是双向数据线。$\overline{CE}$是片选信号，当它为低时选中。$\overline{OE}$是数据输出允许，当它为低时从芯片中读出的数据被送到 O_0～O_7。$\overline{PGM}$是编程有效信号，当对芯片编程时$\overline{PGM}$被置为低。

表 6.2 2764、27128、27256、27512的引脚

引脚号	2764	27128	27256	27512	引脚号	2764	27128	27256	27512
1	V_{PP}	V_{PP}	V_{PP}	A_{15}	15	O_3	O_3	O_3	O_3
2	A_{12}	A_{12}	A_{12}	A_{12}	16	O_4	O_4	O_4	O_4
3	A_7	A_7	A_7	A_7	17	O_5	O_5	O_5	O_5
4	A_6	A_6	A_6	A_6	18	O_6	O_6	O_6	O_6
5	A_5	A_5	A_5	A_5	19	O_7	O_7	O_7	O_7
6	A_4	A_4	A_4	A_4	20	$\overline{CE}$	$\overline{CE}$	$\overline{CE}$	$\overline{CE}$
7	A_3	A_3	A_3	A_3	21	A_{10}	A_{10}	A_{10}	A_{10}
8	A_2	A_2	A_2	A_2	22	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$
9	A_1	A_1	A_1	A_1	23	A_{11}	A_{11}	A_{11}	A_{11}
10	A_0	A_0	A_0	A_0	24	A_9	A_9	A_9	A_9
11	O_0	O_0	O_0	O_0	25	A_8	A_8	A_8	A_8
12	O_1	O_1	O_1	O_1	26	NC	A_{13}	A_{13}	A_{13}
13	O_2	O_2	O_2	O_2	27	$\overline{PGM}$	$\overline{PGM}$	A_{14}	A_{14}
14	GND	GND	GND	GND	28	V_{CC}	V_{CC}	V_{CC}	V_{CC}

表 6.3列出了 2764的 8种工作模式。下面分别加以说明。

读操作：$\overline{CE}=0, \overline{OE}=0, \overline{PGM}=1, V_{PP}=V_{CC}$，根据 A_0～A_{12}选中的单元的内容被送到 O_0～O_7。

编程模式：$\overline{CE}=0, \overline{OE}=0, \overline{PGM}=0, V_{PP}=V_{PP}$，将 O_0～O_7上的数据（从 CPU送来的）写入由 A_0～A_{12}选中的单元。编程需持续 50ms。

Intel编程：这是 Intel公司提出的一种快速编程方法。控制信号与“编程模式”相同，但采用边写入边校验的方法，使编程时间大大缩短。

校验：在 $V_{PP}=V_{PP}$的情况下进行读操作，以便与写入的数据进行比较。

输出禁止：$\overline{OE}=1$，输出端 O_0～O_7高阻。

备用模式：$\overline{CE}=1$，芯片未被选中，输出端 O_0～O_7高阻。

编程禁止：虽然 V_{PP}端被加上了编程电压，但 $\overline{CE}=1$，芯片未被选中。

Intel标识符：$\overline{CE}$、$\overline{OE}$、$\overline{PGM}$、V_{PP}与读操作相同，但 $A_9=V_{ID}$，器件将工作于 Intel标识符模式。Intel标识符为两字节，包括制造厂商信息和器件类型编码。读取 Intel标识符模式时，$A_0=0$，其他位为低，读出的是制造厂商信息；$A_0=1$，其他位为低，读出的是器件类型编码。

表 6.3　2764的工作模式

引脚 / 模式	$\overline{CE}$	$\overline{OE}$	$\overline{PGM}$	A_9	V_{PP}	V_{CC}	$O_7\sim O_0$
读	L	L	H	X	V_{CC}	V_{CC}	数据输出
输出禁止	L	H	H	X	V_{CC}	V_{CC}	高阻
备用模式	H	X	X	X	V_{CC}	V_{CC}	高阻
编程禁止	H	X	X	X	V_{PP}	V_{CC}	高阻
编程模式	L	H	L	X	V_{PP}	V_{CC}	数据输入
Intel编程	L	H	L	X	V_{PP}	V_{CC}	数据输入
校验	L	L	H	X	V_{PP}	V_{CC}	数据输出
Intel标识符	L	L	H	V_{ID}	V_{CC}	V_{CC}	标识符输出

其中，V_{CC}是 5 V电源电压，V_{PP}是编程电压。编程电压随不同的生产厂家有区别，一般为 12 V左右。V_{ID}为加在 A_9脚上的标识符识别电压，电压值与编程电压相同。

⑵ 27系列的其他芯片

27128是 16 K×8 bit的 EPROM，其 26脚是 A_{13}，即 2764的空脚 NC改为了 A_{13}。

27256是 32 K×8 bit的 EPROM，27脚改为了 A_{14}，无 $\overline{PGM}$端。是否编程模式由 V_{PP}的电压决定，当 V_{PP}端加上合适的编程电压时工作于编程模式。

6.4.2　E^2PROM

1. 简介

E^2PROM的工作原理与 EPROM类似，当浮栅中没有电荷时，MOS管不导通，若使浮栅中带有电荷，MOS管就导通。如图 6.13所示，E^2PROM中漏极上面增加了一个隧道二极管，在第二栅与漏极之间的电压 V_G的作用下，不同的电压

极性使电荷流向浮栅（编程）或者流出浮栅（擦除）。

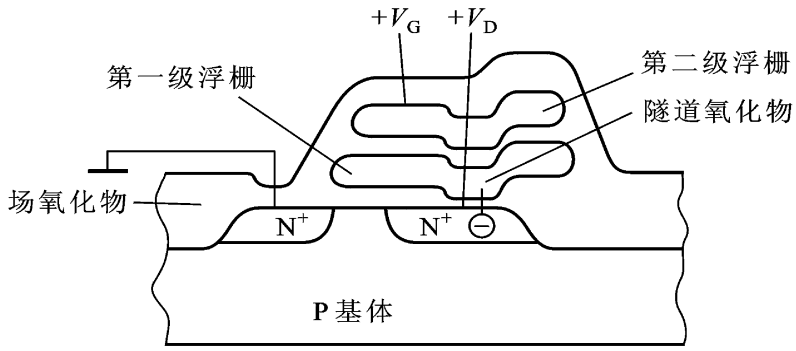


图 6.13 E^2PROM工作原理

E^2PROM具有很高的可靠性，擦写次数 10^4 ～10^5以上，数据保持期大于10年。

2. 典型 E^2PROM 芯片 AT28C64

AT28C64、28C128、28C256、28C512的容量分别是 8 KB、16 KB、32 KB、64 KB的 E^2PROM，28脚封装，管脚与 2764、27128、27256、27512兼容，可直接替换。由于 E^2PROM使用方便，现已替代了 EPROM。下面介绍 28C64芯片特性及读写操作。

(1) 28C64的引脚

图 6.14是两种不同封装的 28C64的顶视图。A0～A12为地址线，I/O0 ～I/O7为双向的数据输入/输出。$\overline{CE}$为芯片使能，$\overline{OE}$为输出允许、$\overline{WE}$为写允许。NC为空脚。RDY/$\overline{BUSY}$为状态信号，低电平表示芯片“忙”，芯片的地址和数据缓冲器不能接受新的输入，即 CPU不能对芯片进行写操作；高电平表示芯片“准备好”，CPU可以进行写操作。因此当对 28C64进行写操作前要先检查状态信号 RDY/$\overline{BUSY}$。

(2) 28C64的读操作

28C64的读操作类似于 SRAM，当 $\overline{CE}$和 $\overline{OE}$为低，$\overline{WE}$为高，地址确定的存储单元的内容输出到 I/O0～I/O7。当 $\overline{CE}$或 $\overline{OE}$为高，I/O0～I/O7为高阻状态。

(3) 28C64的其他操作

字节写（BYTE Write）：28C64的写操作类似于 SRAM，当 $\overline{CE}$和 $\overline{OE}$为高，地址信号和由 I/O0～I/O7输入的数据在$\overline{WE}$的上升沿被锁存到芯片内，开始写入由地址确定的存储单元。与读出时间相比，写入时间 TWC是比较长的。完成字节写所需要的时间小于 1 ms，快速的 AT28C64E系列，完成字节写所需要的时间小

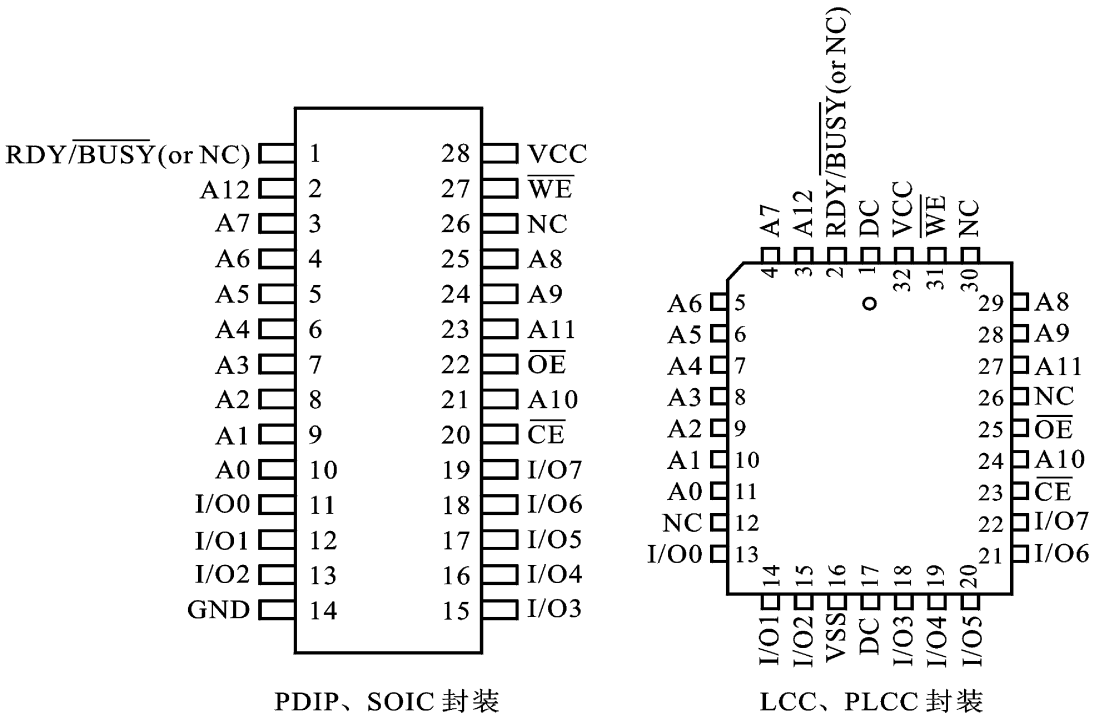


图 6.14　28C64的顶视图

于 200 μs。写操作分成擦除和写入两个步骤,它们是芯片内部自动实现的。一旦开始写操作,芯片内部自动进行该单元的擦除和写入操作直到完成。

RDY/$\overline{\text{BUSY}}$:RDY/$\overline{\text{BUSY}}$ (引脚 1)是漏极开路的输出端,可以被用来测试芯片的写操作是否结束。当存储器正在进行内部写操作时,RDY/$\overline{\text{BUSY}}$变为低电平表示芯片“忙”,外部对芯片不能进行写操作;当存储器内部写操作完成后RDY/$\overline{\text{BUSY}}$变为高电平表示芯片“准备好”,可以进行写操作。由于它是漏极开路的,所以可以将多片 28C64的 RDY/$\overline{\text{BUSY}}$并联,它们的逻辑关系是“线或”。即只要有一片存储器的 RDY/$\overline{\text{BUSY}}$输出为低,并联起来的 RDY/$\overline{\text{BUSY}}$就为低,当并联起来的 RDY/$\overline{\text{BUSY}}$为高,就表示全部芯片都处于“准备好”状态。

数据轮询 (DATA POLLING):28C64在写周期中提供完成写操作的数据检测功能。在写周期中,不断地读数据,读出的数据的 I/O7总是相反的,直到写操作完成,读出的数据就正确了。因此可以采用反复检测写入数据的方法判断写操作是否完成。

写保护 (Write Protection):下列三种情况下存储器不能进行写操作:

1) 电源电压低于 3.8 V;

2) 电源电压延时期间不能进行写操作,即当电源电压由低于 3.8 V到达到 3.8 V时,将延时 5ms后才能进行写操作;

3) $\overline{OE}$为低、$\overline{CE}$为高或$\overline{WE}$为高，这三个信号任意一个符合，都不能进行写操作。

全片清除（Chip Clear）：将存储器内的全部存储单元中的每一位置为“1”就是全片清除操作。方法是使$\overline{CE}$为低、$\overline{OE}$为 12 V，在$\overline{WE}$端加上 10 ms低脉冲。

6.4.3 Flash Memory

1. 简介

闪速（Flash）存储器是 1988年 Intel公司采用 ETOX（EPROM TunnelOxide，EPROM 沟道氧化）技术研制成功的新型非易失性存储器。闪速存储器在系统电可重写，但只能分块进行电擦除。闪速存储器采用单管存储单元结构，比 EPROM结构简化。它是从 EPROM演化而来，工艺过程与 EPROM的大部分是相同的。目前，由于闪速存储器的性能不断提高，价格不断降低，得到了广泛应用，特别是在固态大容量存储器领域有极大的市场。Flash存储器的接口形式有并行和串行两类。

2. 典型闪速存储器芯片 TMS29F040

目前，Flash存储器有多种系列的产品，还有多种类型的 Flash存储卡。下面介绍 29F系列的 Flash存储器，容量为 4 Mbit的 TMS29F040。

TMS29F040是 4 194 304位可编程只读存储器，由 8个独立的 64 K字选段组成。随着芯片的速度分档的不同，访问时间在 60 ns与 120 ns之间。TMS29F040使用 5 V单电源，TMS29LF040使用 3.3 V电源。可编程/擦除 100 000次。

片内的状态机控制编程与擦除器件，字节编程与区段/芯片擦除功能是全自动的。命令集与 JEDEC 4 Mbit E^2PROM 兼容。使用硬件区段保护特性可实现任何区段组合的数据保护。图 6.15为 29F040的内部结构。

(1) 引脚说明

TMS29F040有多种封装，图 6.16是 PLCC－32脚封装，引脚说明如下：

A0～A18——地址输入，其中 A18、A17、A16选择区段

DQ0～DQ7——输入（编程）/输出

$\overline{E}$——芯片使能

$\overline{G}$——输出使能

$\overline{W}$——写使能

V_{CC}——5 V电源

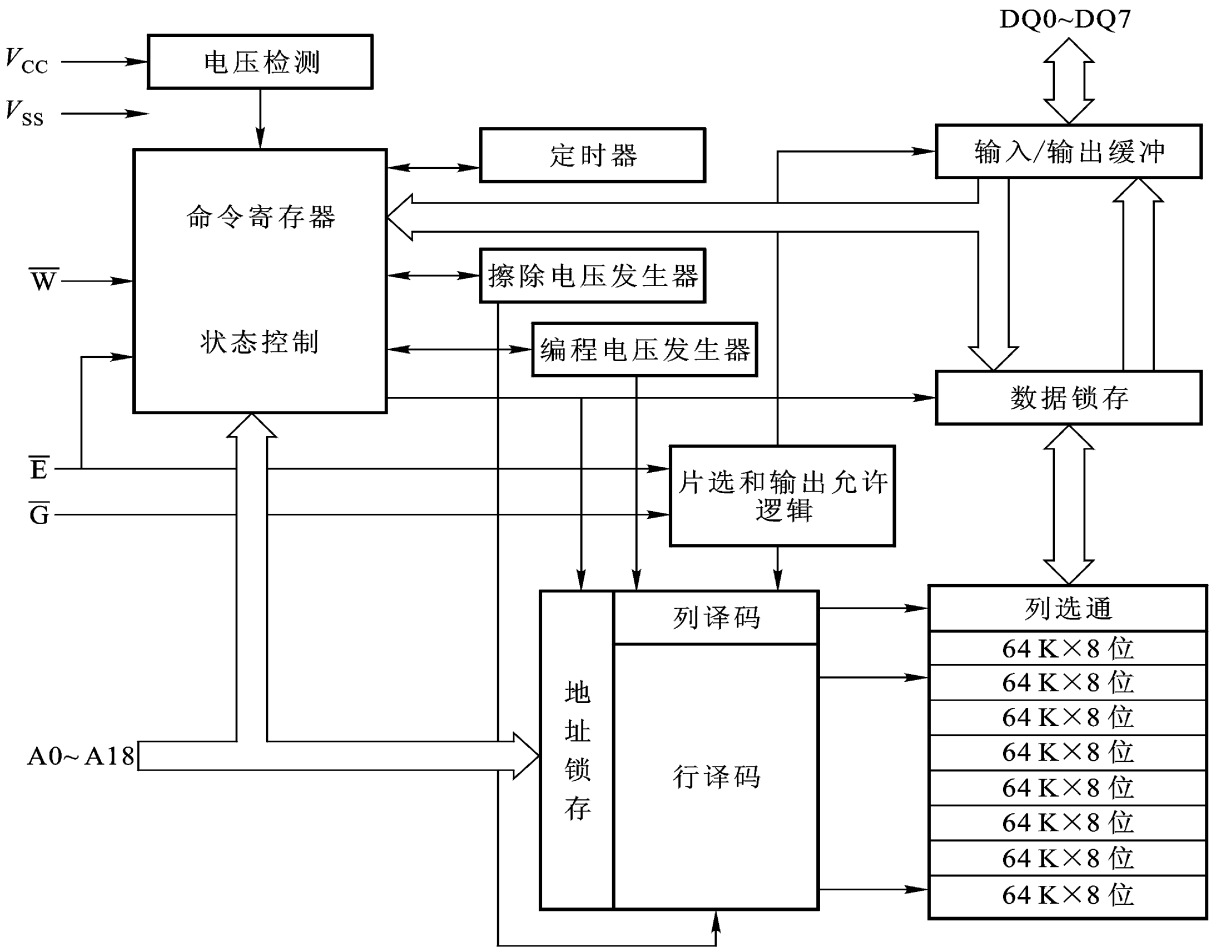


图 6.15　29F040的内部结构

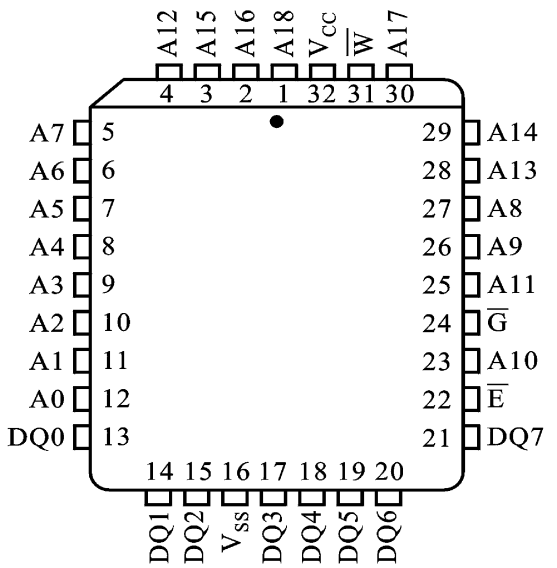


图 6.16　29F040引脚图

V_{SS}——地

⑵ 操作命令

通过使用标准的微处理器写操作时序把 JEDEC标准命令写入命令寄存器，以选择器件的工作方式。命令寄存器用作内部状态机的输入，内部状态机解释命令、控制擦除与编程操作、输出器件的状态、输出存储在器件中的数据以及输出器件算法选择代码。在初始上电操作时，器件缺省方式为读方式。表 6.4列出了各种操作命令，下面介绍几个主要命令。

表 6.4 操作命令表

命令	总线周期	第一周期（写）		第二周期（写）		第三周期（写）		第四周期（读/写）		第五周期（写）		第六周期（写）	
		地址	数据	地址	数据	地址	数据	地址	数据	地址	数据	地址	数据
读/复位	1	XXXXH	F0H										
	4	5555H	AAH	2AAAH	55H	5555H	F0H	RA	RD				
算法选择	4	5555H	AAH	2AAAH	55H	5555H	90H	RA	RD				
字节编程	4	5555H	AAH	2AAAH	55H	5555H	A0H	PA	PD				
芯片擦除	6	5555H	AAH	2AAAH	55H	5555H	80H	5555H	AAH	2AAAH	55H	5555H	10H
区段擦除	6	5555H	AAH	2AAAH	55H	5555H	80H	5555H	AAH	2AAAH	55H	SA	30H
区段擦除暂停		XXXXH	B0H	在区段擦除期间暂停有效									
区段擦除恢复		XXXXH	30H	仅在区段擦除暂停之后擦除恢复									

其中：A＝被读出的存储单元地址（A18～A0）；

PA＝被编程的存储单元地址（A18～A0）；

RD＝所选地址单元被读出的数据；

PD＝所选地址单元被编程的数据；

SA＝被擦除的区段地址，地址位 A18、A17、A16唯一地选择 8个区段之一。

⑴ 读/复位命令

把表 6.4中两种读/复位命令序列的任何一个写入命令寄存器，可以激活读或复位方式。芯片保持在此方式直至其他有效命令序列之一输入到命令寄存器为止。在读方式下，存储器内的数据可用标准的微处理器读周期时序读出，与普通的 ROM一样。

上电时，器件缺省为读/复位方式，不需要向芯片写入读/复位命令。若执行了其他操作，当需要转为读方式时，则应写入读/复位命令。

⑵ 字节编程命令

如表 6.4所示，字节编程是由 4个总线写周期构成的命令序列。前三个总

线周期把器件置于编程建立状态。第 4个总线周期把要编程的单元地址和数据装入器件。嵌入式字节编程 (Embedded Byte-programming) 功能自动提供编程所需的电压和时序并校验单元界限。字节编程操作需要较长的时间,字节编程操作周期最小值为 16 μs。在编程操作期间内写入的任何其他命令均被忽略。

被擦除的单元的所有位都为逻辑 1,编程时将逻辑 0写入。试图把先前已编程为 0的位编程为 1将导致内部脉冲计数器超出脉冲计数极限。这将把超出定时极限 (Exceed timing limit) 指示位 DQ5置为逻辑高状态。只有擦除操作可把位从逻辑 0变为逻辑 1。因此,应先擦除后编程。

利用数据轮询特性或跳转位特性可以监视自动编程操作期间器件的状态以了解字节编程操作是否完成。

3) 芯片擦除命令

芯片擦除是 6总线周期的命令序列。前三个总线周期把器件置为擦除建立状态。接着的两个总线周期开启擦除方式。第 6个总线周期装载芯片擦除命令。在芯片擦除操作期间内写入的任何其他命令均被忽略。

同样,利用数据轮询特性或跳转位特性可以监视自动芯片擦除操作期间器件的状态以了解操作是否完成。

芯片擦除操作周期时间典型值为 14 s,最大 120 s。

4) 区段擦除命令

区段擦除是 6总线周期的命令序列。前三个总线周期把器件置为擦除建立状态。接着的两个总线周期开启擦除方式。第 6个总线周期装载区段擦除命令以及要擦除的区段地址,区段地址仅与 A18、A17、A16有关,与 A15～A0无关。

区段擦除操作周期时间典型值为 2 s,最大 30 s。

(3) 操作状态查询

在编程和擦除操作期间内,可以读出状态信息,状态信息反映器件正在进行的操作的状态。下面介绍其中的数据轮询 (data polling) 位 (DQ7) 和跳转位 (toggle bit) (DQ6)。

1) 数据轮询位 DQ7

在字节编程操作期间保持地址不变,从同一单元读出的最高位是被编程的数据 DQ7的反 ($\overline{DQ7}$/DQ7)。当字节编程操作完成后,从该单元读出的最高位是被编程的 DQ7数据本身。因此,数据位从 $\overline{DQ7}$变为 DQ7指示了字节编程操作的结束。

在擦除操作期间内,在被擦除区段内的进行数据轮询,读出的最高位 DQ7将为 0。擦除操作完成后,读出的最高位 DQ7将为 1。因此,可根据 DQ7从 0变为 1来判断擦除操作的结束。

2) 跳转位 DQ6

在进行编程或擦除操作期间,跳转位 DQ6在 1和 0之间跳转。当对同一地址两次连续读,发现跳转位 DQ6停止跳转时,表示操作完成。

6.5 存储器与系统的连接

存储器与系统之间通过 AB、DB及有关的控制信号线相连接,设计系统的存储器体系时需要将这三类信号线正确连接。一是数据线的连接,要使存储器与 CPU的数据线匹配。二是地址线的连接,通过将地址总线中部分高位地址线经过适当的译码电路后选中相应的芯片的方法,使存储器芯片位于符合要求的地址范围,即存储器芯片的片选信号如何正确产生的问题。控制信号一般包括读信号 (如 $\overline{RD}$)、写信号 (如 $\overline{WR}$) 等。

设计系统的存储器体系时还要考虑速度匹配问题。为了保证存储器读写正确,存储器芯片的读写时间应短于 CPU读写周期中留给存储器操作的时间。若 CPU读写周期中可以插入等待周期 T_W,则也可使用慢速的存储器,代价是牺牲 CPU的存储器读写速度。

6.5.1 8位微机系统中存储器与系统的连接

1. 存储器系统设计举例

Intel8088 CPU是与 8086兼容的 CPU,主要区别是其数据总线为 8位。下面以 8088系统为例说明存储器与 8位 CPU的连接方法。

【例题 6.1】 某 8088系统 (最大组态) 的存储器系统如图 6.17所示,图中 8088 CPU芯片上的地址、数据信号线经锁存、驱动后成为地址总线 A_{19} ~A_0、数据总线 D_7 ~D_0。U0、U1是两片 EPROM,型号为 27128。U2、U3是两片 RAM,型号为 62256。两片译码器 74HC138担任片选译码。

(1) 数据线的连接

27128和 62256的数据线都是 8位,因此它们可与 CPU的数据线直接相连。

(2) 地址线的连接

根据存储器芯片上地址线的位数,把 CPU的地址信号分为片内地址和片外地址 (就存储器芯片而言)。

本例中 27128的容量为 16 KB,存储器芯片内的地址线为 14位 (A_{13} ~A_0),它们与 CPU的地址线中的同名端直接相连。存储器片外的地址线,即 8088

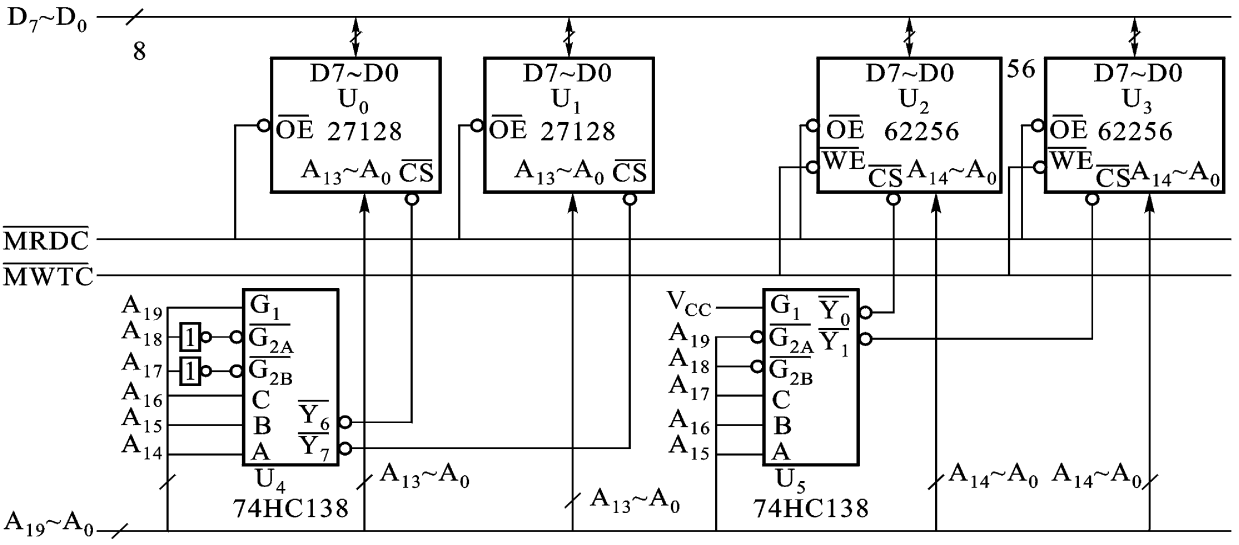


图 6.17　例题 6.1电路图（CPU为 8088最大组态）

CPU其余的 6条地址线 A_{19}～A_{14}通过译码器 U4进行地址译码。A_{18}、A_{17}经反相，当 $A_{19}A_{18}A_{17}$ =111时 U_4 的使能端都有效。$A_{16}A_{15}A_{14}$接 U_4 的 C、B、A端。

根据 74LS138的逻辑功能可以知道，当 A_{19}～A_{14} =111000时 $\overline{Y_0}$ =0；相应地，由 $\overline{Y_0}$选中的存储器的地址范围为 E0000H～E3FFFH。当 A_{19}～A_{14} =111001时 $\overline{Y_1}$ =0相应地，由 $\overline{Y_1}$选中的存储器的地址范围为 E4000H～E7FFFH。这样，可以确定全部输出端与存储器地址的关系，表 6.5列出了 U_4 的输出 $\overline{Y_0}$～$\overline{Y_7}$作存储器片选时被选中存储器的地址范围。其中阴影部分的地址线 A_{19}～A_{14}是存储器芯片片外的地址线。

表 6.5　U4的输出端与被选中存储器的地址范围

输出	被选中存储器的地址范围		
	地址 A_{19}～A_0（二进制）		地址 A_{19}～A_0（十六进制）
$\overline{Y_0}$	起始地址	1110, 00 00, 0000, 0000, 0000	E0000
	末地址	1110, 00 11, 1111, 1111, 1111	E3FFF
$\overline{Y_1}$	起始地址	1110, 01 00, 0000, 0000, 0000	E4000
	末地址	1110, 01 11, 1111, 1111, 1111	E7FFF
$\overline{Y_2}$	起始地址	1110, 10 00, 0000, 0000, 0000	E8000
	末地址	1110, 10 11, 1111, 1111, 1111	EBFFF

续表

输出	被选中存储器的地址范围		
	地址 A_{19}～A_0（二进制）		地址 A_{19}～A_0（十六进制）
$\overline{Y_3}$	起始地址	1110, 1100, 0000, 0000, 0000	EC000
	末地址	1110, 1111, 1111, 1111, 1111	EFFFF
$\overline{Y_4}$	起始地址	1111, 0000, 0000, 0000, 0000	F0000
	末地址	1111, 0011, 1111, 1111, 1111	F3FFF
$\overline{Y_5}$	起始地址	1111, 0100, 0000, 0000, 0000	F4000
	末地址	1111, 0111, 1111, 1111, 1111	F7FFF
$\overline{Y_6}$	起始地址	1111, 1000, 0000, 0000, 0000	F8000
	末地址	1111, 1011, 1111, 1111, 1111	FBFFF
$\overline{Y_7}$	起始地址	1111, 1100, 0000, 0000, 0000	FC000
	末地址	1111, 1111, 1111, 1111, 1111	FFFFF

U_0 由 $\overline{Y_6}$ 作片选，因此其地址范围为 F8000H～FBFFFH，U_1 由 $\overline{Y_7}$ 作片选，因此其地址范围为 FC000H～FFFFFH，两片构成 32 KB的 ROM 存储体。

根据同样的分析，本例中 62256的容量为 32 KB，存储器芯片内的地址线为15位，片内地址线与 CPU的地址线中的同名端直接相连。存储器片外的地址线，即 8088CPU其余的 5条地址线 A_{19}～A_{15} 通过译码器 U_5 进行地址译码。注意 A_{19}、A_{18} 与 U_5 的 G_{2A}、G_{2B} 相连，当 $A_{19}A_{18}$ =00时 U_5 使能。

表 6.6根据 74HC138的逻辑功能，列出了 U_5 的输出 $\overline{Y_0}$～$\overline{Y_7}$ 作存储器片选时被选中存储器的地址范围。

表 6.6 U_5 的输出端与被选中存储器的地址范围

输出	被选中存储器的地址范围		
	地址 A_{19}～A_0（二进制）		地址 A_{19}～A_0（十六进制）
$\overline{Y_0}$	起始地址	0000, 0000, 0000, 0000, 0000	00000
	末地址	0000, 0111, 1111, 1111, 1111	07FFF
$\overline{Y_1}$	起始地址	0000, 1000, 0000, 0000, 0000	08000
	末地址	0000, 1111, 1111, 1111, 1111	0FFFF

续表

输出	被选中存储器的地址范围		
	地址 A_{19} ～A_0（二进制）		地址 A_{19} ～A_0（十六进制）
$\overline{Y_2}$	起始地址	0001, 0 000, 0000, 0000, 0000	10000
	末地址	0001, 0 111, 1111, 1111, 1111	17FFF
$\overline{Y_3}$	起始地址	0001, 1 000, 0000, 0000, 0000	18000
	末地址	0001, 1 111, 1111, 1111, 1111	1FFFF
$\overline{Y_4}$	起始地址	0010, 0 000, 0000, 0000, 0000	20000
	末地址	0010, 0 111, 1111, 1111, 1111	27FFF
$\overline{Y_5}$	起始地址	0010, 1 000, 0000, 0000, 0000	28000
	末地址	0010, 1 111, 1111, 1111, 1111	2FFFF
$\overline{Y_6}$	起始地址	0011, 0 000, 0000, 0000, 0000	30000
	末地址	0011, 0 111, 1111, 1111, 1111	37FFF
$\overline{Y_7}$	起始地址	0011, 1 000, 0000, 0000, 0000	38000
	末地址	0011, 1 111, 1111, 1111, 1111	3FFFF

U_2 由 $\overline{Y_0}$作片选，因此其地址范围为 00000H～07FFFH，U_3 由 $\overline{Y_1}$作片选，因此其地址范围为 08000H～0FFFFH，两片构成 64 KB的 RAM存储体。

(3) 控制信号的连接

27128上的$\overline{OE}$与 CPU的存储器读信号$\overline{RD}$相连，62256上的$\overline{OE}$与 CPU的读信号$\overline{RD}$相连，当 CPU执行存储器读操作时被选中的存储器单元的内容由 DB送入 CPU内。62256上的写允许$\overline{WE}$与 CPU的$\overline{WE}$相连，当 CPU执行存储器写操作时 CPU内的数据经由 DB送入存储器写入选中的单元。

【例题 6.2】　试在 8088系统（最小模式）中设计 256 KB RAM、32 KB E^2PROM。RAM区的首地址为 40000H，E^2PROM区的首地址为 F8000H。

分析：RAM选 62512，容量为 64 KB，共需 4片，片内的地址线为 A_{15} ～A_0。E^2PROM选用 28C256，容量为 32 KB，只需一片。M/$\overline{IO}$为高电平时选中存储器。

RAM区的首地址为 40000H，高 4位地址线即片外的地址线 A_{19} ～A_{16} ＝0100。译码电路采用 74HC138，A_{19}接$\overline{G_{2B}}$。A_{18} ～A_{16} ＝100，即$\overline{Y_4}$选择 RAM0，A_{18} ～A_{16} ＝101，即$\overline{Y_5}$选择 RAM1，A_{18} ～A_{16} ＝110，即$\overline{Y_6}$选择 RAM2，A_{18} ～A_{16} ＝111，即$\overline{Y_7}$选择 RAM3。

E^2PROM0的片选信号 $\overline{CE}$ 在 A_{19} ～ A_{15} = 11111 时被选中，即 $\overline{CE} = \overline{A_{19}\,A_{18}\,A_{17}\,A_{16}\,A_{15}}$。

完整的电路图如图 6.18所示。

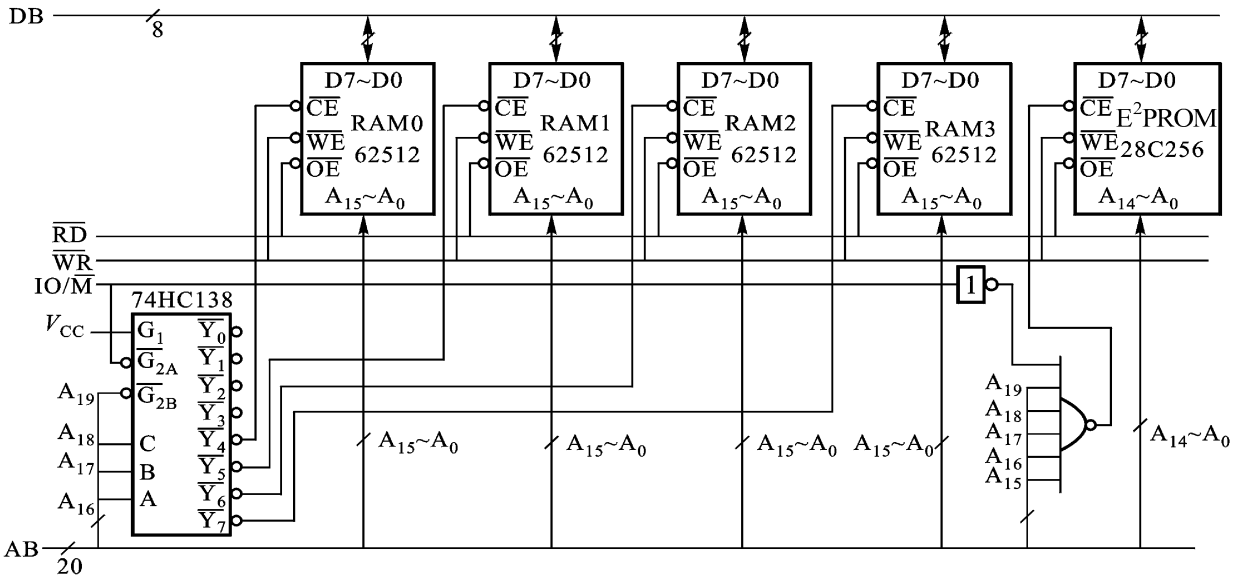


图 6.18 例题 6.2电路图 (CPU为 8088最小模式)

2. 片选信号的产生

无论 8位、16位或其他存储器系统，一般都由多片存储器芯片构成，那么就需要译码电路进行片选。译码电路可使用中集成的译码器如 74HC138等，也可使用逻辑门，近年来已更多使用灵活方便的可编程逻辑器件，如 GAL、CPLD或 FPGA。

有三种片选控制的方法，全译码、部分译码和线选。

(1) 全译码

片选信号由地址线中所有不在存储器上的地址线译码产生，称为全译码。也就是说，除了已经连接到存储器芯片上的地址线，其他高位的地址线都被送入译码电路，以产生片选信号。这样，存储器芯片中的每一个单元的地址将是唯一的。

(2) 部分译码

片选信号不是由地址线中所有不在存储器上的地址线译码产生，即只有部分高位地址线被送入译码电路产生片选信号，称为部分译码。由于有地址线未参与译码，则该地址线是 0或 1将不影响芯片的选中与否。例题 6.2中，若 A_{19} 不参与 E^2 PROM的片选信号的译码，则它的地址为 F8000H～FFFFFH，或者

78000H～7FFFFH。也就是说,同一个单元有两个地址。若 A_{19}、A_{18}都不参与 E^2PROM的片选信号的译码,则同一个单元将有 4个地址。

⑶ 线选

以不在存储器上的高位地址线直接作为存储器芯片的片选信号,称为线选。使用线选法的好处是译码电路简单。但线选不仅导致一个存储单元有多个地址,还有可能一个地址同时选种多个单元,这会引起数据总线的冲突。因此,使用该方法时,程序中应避免出现后一种情况。通常线选法使用在系统的地址空间很大、所用存储器容量很小的情况下。

6.5.2 16位微机系统中存储器与系统的连接

16位微机的存储器体系有两种。一种是标准的 16位存储体,另一种是 8位存储体。

1. 16位存储体

标准的 16位存储体,每个存储单元为 16位,数据总线 16位,每次存储器操作都是 16位的,如图 6.19所示。这样的系统中,存储器与 CPU的连接原理与上节所述的 8位微机系统相同。只要把数据总线扩展到 16位即可。

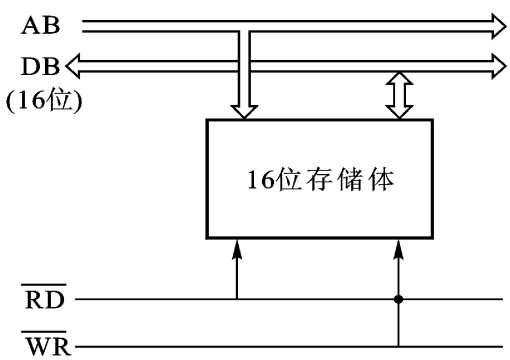


图 6.19 典型的 16位存储体

2. 8位存储体

这种 16位微机的数据总线 16位,但存储器体系是 8位存储体,即每个地址确定的存储单元为 8位,存储器操作可能是 8位的也可能是 16位的。8086系统就是这样的结构,下面以 8086系统为例介绍其原理。

⑴ 奇体、偶体

8086 CPU有 20位地址线,可直接寻址 1MB的存储器地址空间。当把存储器看作字节序列时,每个字节单元地址相连,即每个地址对应一个存储单元,每

个存储单元为一个字节。当把存储器看作字序列时,每个字单元地址不相连,每个字包括地址相连的两个字节。而 8086 CPU的数据总线是 16位的,需要设计一种合理的存储体结构,既能适合做 8位的存储器操作 (字节访问),又能适合做 16位的存储器操作 (字访问)。

8086系统将 1 MB地址空间分成两个 512 KB地址空间,一半是偶数地址另一半是奇数地址,相应的存储体称为偶体和奇体。偶体和奇体的地址线都是 19位。将数据总线的低 8位 D_7 ~D_0与偶体相连,高 8位 D_{15} ~D_8与奇体相连。地址总线的 A_{19} ~A_1与这两个存储体的 19条地址线 A_{18} ~A_0相连。用 CPU的 A_0 作偶体的选中信号,$\overline{BHE}$作奇体的选中信号。如图 6.20所示。

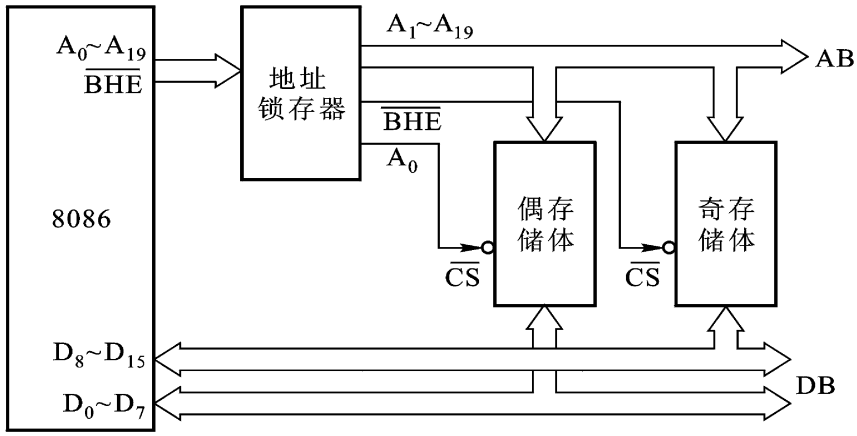


图 6.20 8086系统的存储器

(2) 字节访问

8086 CPU进行存储器访问有 8位的也有 16位的。当进行字节访问即 8位的访问时,如果地址的 A_0 =0,选中偶体中的某个单元,数据通过 D_7 ~D_0传送。如果地址的 A_0 =1,则 CPU的$\overline{BHE}$=0,选中奇体中的某个单元,数据通过 D_{15} ~D_8传送。

(3) 字访问

当 CPU进行 16位的字访问时,设低字节的地址为 n,则高字节的地址为 n+1。若地址 n为偶数,即 A_0 =0,称为对准的字若地址 n为奇数,即 A_0 =1,称为非对准的字。

当 CPU访问对准的字时,由 A_0 =0选中偶体中的地址为 n的单元,低字节数据通过 D_7 ~D_0传送;同时由$\overline{BHE}$=0选中奇体中的地址为 n+1的单元,高字节数据通过 D_{15} ~D_8传送。这样,两个字节的数据在一个总线周期中同时进行读或写操作。

当 CPU访问非对准的字时即地址 n为奇数,要由两个总线周期完成一个字的读或写操作。第一个总线周期发出 A_0 =1和 $\overline{BHE}$=0,访问奇体中的地址为 n的单元,低字节数据通过 D_{15} ~D_8传送;第二个总线周期发出 A_0 =0和 $\overline{BHE}$=1,访问偶体中的地址为 n+1的单元,高字节数据通过 D_7 ~D_0传送。这样,两个字节分别由两个总线周期进行读或写操作。

两种数据传送方式,对于程序员来说是“透明的”,编写程序时不必考虑CPU的取数过程。但 CPU访问对准的字和非对准的字,所需的总线周期数是不同的,因此应将字型数据的低 8位放在偶地址以提高访问内存的速度。

同样,8086系统的 I/O地址空间的组织也类似于 8086的存储体系。由 $\overline{BHE}$和 A_0选择奇体或偶体,对准的字可以在一个总线周期内完成输入或输出操作。

【例题 6.3】　某 8086系统 (最大模式)的存储器系统如图 6.21所示,图中 8086 CPU芯片上的地址、数据信号线经锁存、驱动后成为地址总线 A_{19} ~A_0、数据总线 D_{15} ~D_0。 ROM0、ROM1是两片 E^2PROM,型号为 28C256。 RAM0、RAM1是两片 RAM,型号为 62256。译码器 74HC138担任片选译码。

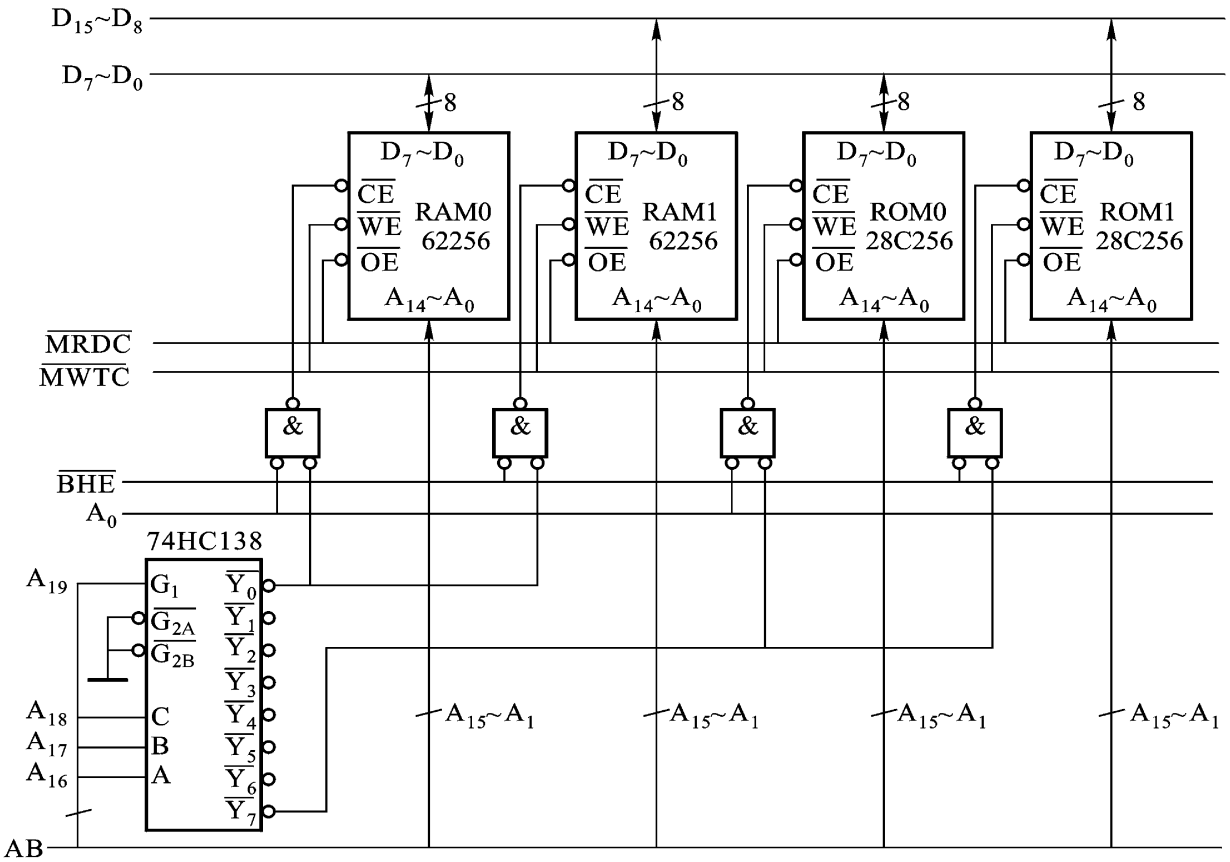


图 6.21　例题 6.3电路图 (CPU为 8086最大模式)

RAM0和 RAM1组成的存储器的地址范围为 80000H～8FFFFH，RAM0是偶存储体，RAM1是奇存储体。ROM0和 ROM1组成的存储器的地址范围为 F0000H～FFFFFH，ROM0是偶存储体，ROM1是奇存储体。

6.5.3 32位微机系统中存储器与系统的连接

32位微机系统中的存储体是 8位的时候，其接口的基本原理与 6.5.2节中介绍的类似，下面以 80386 CPU为例说明。

1. 存储体结构

80386 CPU 的地址线为 32 位，存储器地址空间从 00000000H～FFFFFFFFH，存储器字长 8位。存储器容量为 4 GB。数据总线是 32位，当把存储器看作字节序列时，每个字节单元地址相连。当把存储器看作字序列时，每个字单元地址不相连，每个字包括地址相连的两个字节。当把存储器看作双字序列时，每个双字单元地址不相连，每个双字包括地址相连的 4个字节。在 386系统中存储器被分成 4个存储体，每个存储体的字长为 8位，每个存储体的容量为 1 GB。4个存储体的选通信号分别是 $\overline{BE0}$、$\overline{BE1}$、$\overline{BE2}$、$\overline{BE3}$，对应的数据线为 D_7～D_0、D_{15}～D_8、D_{23}～D_{16}、D_{31}～D_{24}，如图 6.22所示。

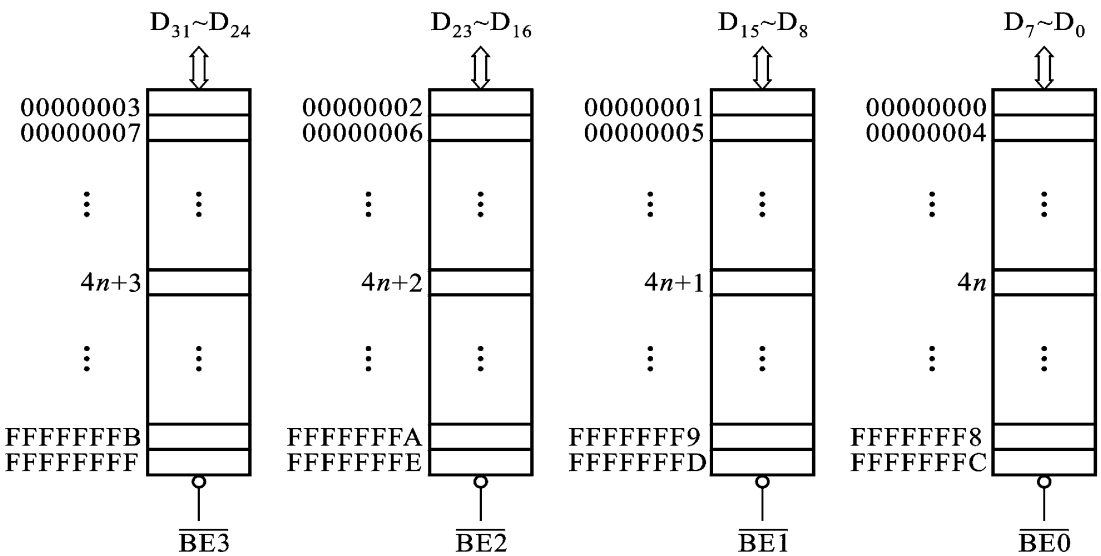


图 6.22 32位微机系统中的存储体

CPU的 A_{31}～A_2对存储器中的双字进行寻址，最低的两位在 CPU内进行译码以控制选通输出信号 $\overline{BE0}$、$\overline{BE1}$、$\overline{BE2}$、$\overline{BE3}$。在每个数据传输总线周期中，数据可以用 32位、24位、16位或 8位的方式传输。

386CPU只能以字节、字、双字为操作单位,不会出现$\overline{BE3}$和$\overline{BE0}$同时有效的情况,也不会出现$\overline{BE2}$和$\overline{BE0}$同时有效的情况。

2. 对齐的传送与非对齐的传送

如果访问的是字节,那么传输都是对齐的。

如果访问的双字的地址是 4n,即最低两位 A_1A_0 =00,那么传输是对齐的,传输能在一个总线周期内完成。如果访问的字的地址是 2n、2n+1和 2n+2,那么传输是对齐的,传输能在一个总线周期内完成。

如果访问的字或双字跨越了 386CPU的双字边界,那么传送是非对齐的。表 6.7说明了 32位总线上非对齐的数据传送。非对齐的数据传送需两个总线周期。

表 6.7 32位总线上非对齐的数据传送

传送类型	物理地址	第一周期		第二周期	
		地址总线	选通信号	地址总线	选通信号
字	4n+3	4n+4	$\overline{BE0}$	4n	$\overline{BE3}$
双字	4n+1	4n+4	$\overline{BE0}$	4n	$\overline{BE1}$、$\overline{BE2}$、$\overline{BE3}$
双字	4n+2	4n+4	$\overline{BE0}$、$\overline{BE1}$	4n	$\overline{BE2}$、$\overline{BE3}$
双字	4n+3	4n+4	$\overline{BE0}$、$\overline{BE1}$、$\overline{BE2}$	4n	$\overline{BE3}$

思考题与习题

6.1 试说明半导体存储器的分类。

6.2 试说明 CMOS静态存储器基本存储电路数据读、写的原理。

6.3 试说明单管 DRAM基本存储电路数据读、写的原理。

6.4 试说明如何对 28C64进行字节编程,并说明怎样使用数据轮询的方法判断写操作完成。

6.5 试说明 29F040的区段擦除流程,并说明如何判断擦除操作完成。

6.6 试说明 29F040的字节编程流程,并说明如何判断编程完成。

6.7 试使用 62512和 27256,在 8088系统(最小模式)中设计具有 128 KB的 RAM、64 KB的 EPROM的存储体,RAM的地址从 0000:0000H开始、EPROM的地址从 F000:0000H开始。

6.8 试使用 62512和 27512,在 8088系统(最大模式)中设计具有 256 KB RAM、64 KB EPROM的存储体,RAM的地址从 0000:0000H开始、EPROM的地址从 F000:0000H开始。

6.9 8086系统中存储器偶地址体及奇地址体之间应该用什么信号区分?怎样区分?

6.10 8086系统中对外设端口的读/写操作时,$\overline{BHE}$信号和地址线 A_0如何起作用?

6.11 试使用62512和28512,在8086系统(最小模式)中设计具有256 KB RAM、128 KB E^2PROM的存储体,RAM的地址从0000:0000H开始、E^2 PROM的地址从E000:0000H开始。

6.12 试使用621000和28C256,在8086系统(最大模式)中设计具有512 KB RAM、64 KB E^2PROM的存储体,RAM的地址从0000:0000H开始、E^2 PROM的地址从F000:0000H开始。

6.13 某16位微机的存储器体是标准的16位存储体,数据总线为D_{15}～D_0,地址总线为A_{23}～A_0。读信号为$\overline{RD}$,写信号为$\overline{WR}$。试在该系统中使用HM62W16255设计1 M×16 bit的存储体,地址从40000H开始。

6.14 某系统以8088CPU(最大模式)为核心。(1)试在该系统中使用29F040设计512 KB的存储体,地址从8000:0000H开始;(2)试写出进行字节编程的子程序;(3)试写出进行区段擦除的子程序。

第7章

基本输入输出接口

外部设备是微机系统的重要组成部分，微机通过它们与外界进行数据交换。各种外部设备通过输入输出接口（I/O Interface）与系统相连，并在接口电路的支持下实现数据传送和操作控制。

最常用的外部设备有键盘、鼠标、显示器、打印机、绘图仪、调制解调器、软/硬盘驱动器、模/数转换器、数/模转换器等，这些设备通过挂接在总线上的各种接口电路与微处理器相连。接口电路按功能可分为两大类：一类是微处理器工作所需要的辅助/控制电路，通过这些辅助/控制电路，使微处理器得到所需要的时钟信号或接收外部的多个中断请求等；另一类是输入/输出接口电路，利用这些接口电路，微处理器可接收外部设备送来的信息或将信息发送给外部设备。

7.1 I/O接口概述

I/O接口电路是计算机和外设之间传送信息的部件，每个外设部件都要通过相应的接口与系统总线相连，实现与CPU之间的数据交换。接口技术专门研究CPU和外设之间的数据传送方式、接口电路的工作原理和使用方法等。

7.1.1 输入/输出信息

CPU与I/O设备之间传送的信息可分为数据信息、状态信息和控制信息三类。

1. 数据信息

CPU和外设交换的基本信息是数据。数据信息大致可分为如下三种类型：

⑴ 数字量 数字量是用二进制形式表述的数据、图形、文字等信息，通常以并行的 8位或 16位进行传输。

⑵ 模拟量 连续变化的物理量，如温度、湿度、压力、流量等。这些物理量一般通过传感器先变成电压或电流，经过放大，由模/数转换器进行由模拟信号到数字信号的转换，最后送给计算机。因为数字计算机对连续变化的模拟量无法直接接收和处理。

⑶ 开关量 开关量可表示成两个状态，如开关的接通和断开、电机的运转和停止、阀门的打开和关闭、三极管的导通和截止等等，这样的量只要用一位二进制数表示就可以了。

2. 状态信息

状态信息反映了外设当前所处的工作状态，是外设发送给 CPU的，用来协调 CPU和外设之间的操作。对于输入设备来说，通常用准备好 (READY)信号来表示输入数据是否准备就绪；对于输出设备来说，通常用忙 (BUSY)信号表示输出设备是否处于空闲状态，若为空闲，则可接收 CPU送来的信息，否则 CPU等待。

3. 控制信息

控制信息是 CPU发送给外设的，以控制外设的工作。如对外设的初始化，外设的启动和停止等控制信息。

7.1.2 I/O接口的主要功能

⑴ 对输入输出数据进行缓冲和锁存

在微机中 CPU通过接口与外设交换信息。因为输入接口连接在数据总线上，只有当 CPU从该接口输入数据时才允许选定的输入接口将数据送到总线上由 CPU读取，其他时间不得占用总线。因此一般使用三态缓冲器 (三态门)作输入接口，当 CPU不选中该接口时三态缓冲器的输出为高阻。

输出时，CPU通过总线将数据传送到输出接口内的数据寄存器中，然后由外设读取。在 CPU向它写入新数据之前该数据将保持不变。数据寄存器一般由锁存器实现，如 74LS373。

⑵ 对信号的形式和数据的格式进行变换

由计算机直接处理的信号为一定范围内的数字量、开关量和脉冲量，它与外设所使用的信号可能不同。所以，在输入输出时，必须将它们转变成适合对方的

形式。

⑶ 对 I/O端口进行寻址

在一个微机系统中,通常会有多个外设。而在一个外设的接口电路中,又可能有多个端口 (Port),每个端口用来保存和交换不同信息。每个端口必须有各自的端口地址以便 CPU访问。因此,接口电路中应包含地址译码电路使 CPU能够寻址到每个端口。

⑷ 提供联络信号

I/O接口处在 CPU和外设之间,既要面向 CPU进行联络,又要面向外设进行联络。联络的目的是使 CPU与外设之间数据传送的速度匹配。联络的具体内容有:状态信息、控制信息和请求信息。

7.1.3　I/O接口的结构

每个 I/O接口内部一般由 3类寄存器组成,CPU与外设进行数据传输时,各类信息在接口中进入不同的寄存器,一般称这些寄存器为 I/O端口,每一个端口有一个端口地址。I/O接口结构如图 7.1所示。

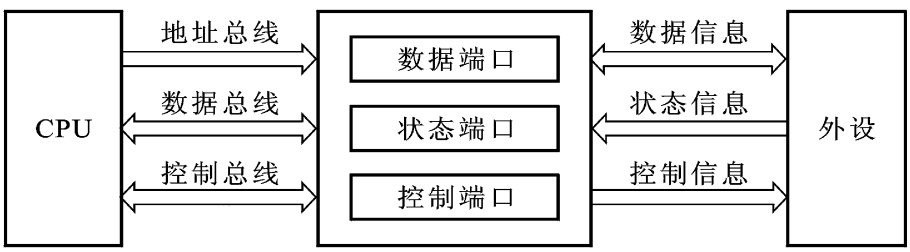


图 7.1　I/O接口结构

数据端口:用于数据信息输入/输出的端口。CPU通过数据接收端口输入数据,有的能保存外设发往 CPU的数据;CPU通过数据输出端口输出数据,一般能将 CPU发往外设的数据锁存。

状态端口:CPU通过状态端口了解外设或接口部件本身的状态。

控制端口:CPU通过控制端口发出控制命令,以控制接口部件或外设的动作。

7.1.4　I/O的寻址方式

在不同的微机系统中,I/O端口的地址编排有两种形式:一是 I/O端口与内

存统一编址,二是 I/O端口独立编址。

1. I/O端口与内存统一编址

I/O端口与内存统一编址,即 I/O端口的地址和内存的地址在同一个地址空间内。所有访问内存的指令都可访问 I/O端口,其缺点是占去内存部分空间且难以区分某条指令访问的是内存还是 I/O端口。

2. I/O端口独立编址

I/O端口有独立的地址空间,即 I/O端口的地址和内存的地址不在同一个地址空间内。系统需有专门的 I/O指令,需要相应的控制电路和控制信号。好处是 I/O端口不占用内存部分地址空间,缺点是需增加硬件电路的复杂性,并且 I/O指令一般较少、不如访问内存的指令丰富。

80X86系列微处理器采用 I/O端口独立编址方法,提供 I/O读/写控制信号,有专门的 I/O指令用于访问 I/O端口。

7.2 简单 I/O接口芯片

在外设接口电路中,经常需要对传输过程中的信息进行缓冲、锁存以及增大驱动能力,能实现上述功能的接口芯片最简单的就是缓冲器、锁存器和数据收发器等,下面介绍几种典型芯片。

1. 锁存器 74LS373

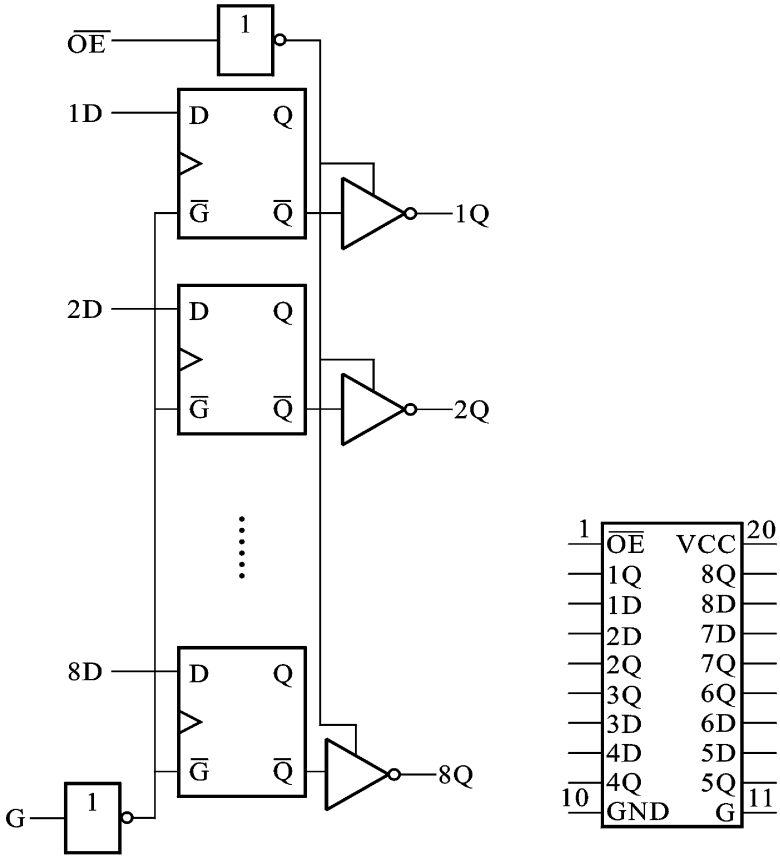
74LS373是由 8个 D触发器组成的具有三态输出和驱动的锁存器,74LS373逻辑电路及引脚如图 7.2所示,使能端 G有效时,将输入端 (D端)数据打入锁存器,当输出允许端$\overline{OE}$有效时,将锁存器中锁存的数据送到输出端 Q;当$\overline{OE}$=1时输出为高阻。常用的锁存器还有 74LS273, Intel 8282等。

2. 缓冲器 74LS244

74LS244是一种三态输出的缓冲器 (或称单向线驱动器),74LS244逻辑电路及引脚如图 7.3所示,内部线驱动器分为两组,分别有 4个输入端 (1A1～1A4, 2A1～2A4)和 4个输出端 (1Y1～1Y4, 2Y1～2Y4),分别由使能端$\overline{1G}$, $\overline{2G}$控制。当$\overline{1G}$为低电平,1Y1～1Y4的电平与 1A1～1A4的电平相同,当$\overline{2G}$为低电平,2Y1～2Y4的电平与 2A1～2A4的电平相同;当$\overline{1G}$ (或$\overline{2G}$)为高电平时,输出 1Y1～1Y4 (或 2Y1～2Y4)为高阻态。常用的缓冲器还有 74LS240, 74LS241等。

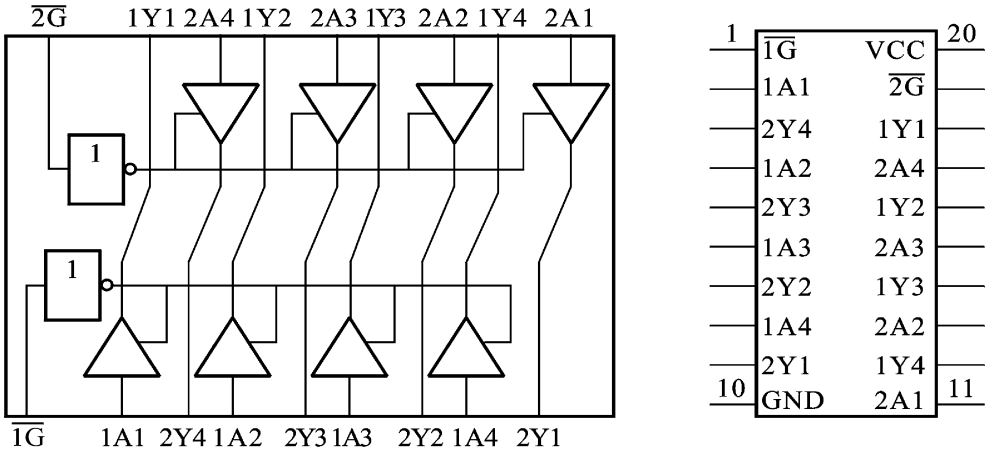
3. 数据收发器 74LS245

74LS245是一种三态输出的数据收发器 (或称双向线驱动器),74LS245逻辑电路及引脚如图 7.4所示,16个三态门每两个三态门组成一路双向驱动。由端, DIR两个控制端控制,端控制驱动器有效或高阻态;当端有效时 DIR控制



(a) 逻辑电路　　(b) 引脚图

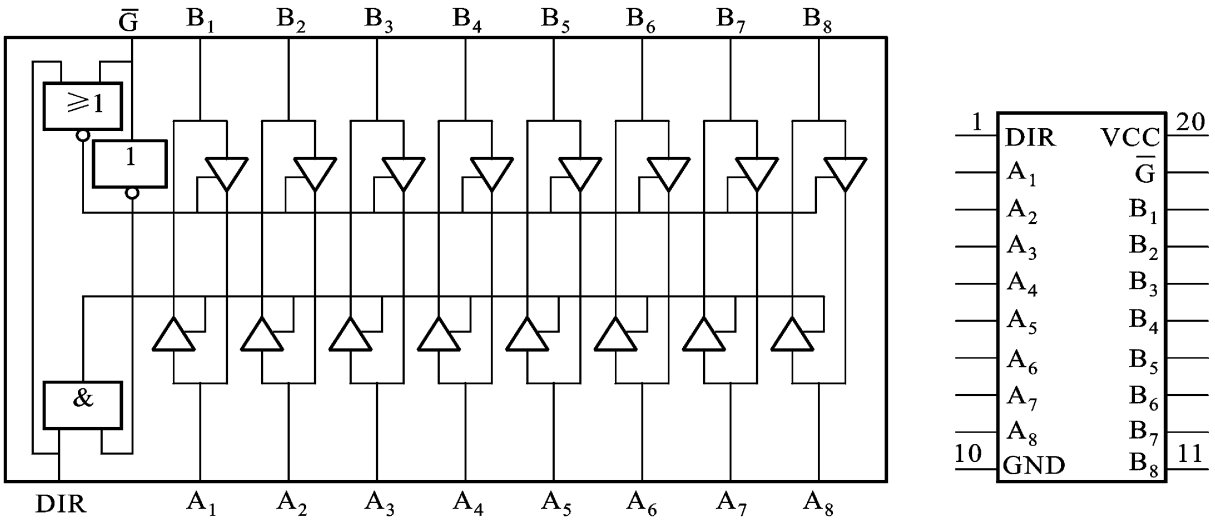
图 7.2　74LS373锁存器



(a) 逻辑电路　　(b) 引脚图

图 7.3　74LS244缓冲器

驱动器的驱动方向，DIR＝0时，驱动方向为 B→A；DIR＝1时，驱动方向为 A→B。74LS245的真值表如表 7.1所示。常用的数据收发器还有 74LS243、Intel 8286、Intel 8287等。



(a) 逻辑电路　　(b) 引脚图

图 7.4 74LS245数据收发器

表 7.1 74LS245真值表

使能端	方向控制 DIR	传送方向
L	L	B→A
L	H	A→B
H	×	隔开

7.3 CPU与外设之间的数据传输方式

在微机控制外设工作期间，最基本的操作是数据传输。但各种外设的工作速度相差很大，如何解决 CPU与各种外设之间的速度匹配，以确保数据传输过程的正确和高效是很重要的问题。通常微机系统与外设之间数据传输采用程序方式、中断方式和 DMA方式来解决上述问题。

7.3.1　程序方式

程序方式是指微机系统与外设之间的数据传输过程在程序的控制下进行。其特点是以 CPU为中心,通过执行预先编制的输入/输出程序实现数据传输。程序方式可分为无条件传输和条件传输两种方式。

1. 无条件传输方式

无条件传输方式是指传输数据过程中,发送/接收数据一方不查询判断对方的状态,进行无条件的数据传输。这种传输方式程序设计简单,一般用于能够确信外设已经准备就绪的场合。如读取开关的状态,LED的显示等。

【例题 7.1】　硬件如图 7.5所示,不断扫描开关 S_i,当开关闭合时,点亮相应的 LED_i,当地址为 200H时,Y为低电平。

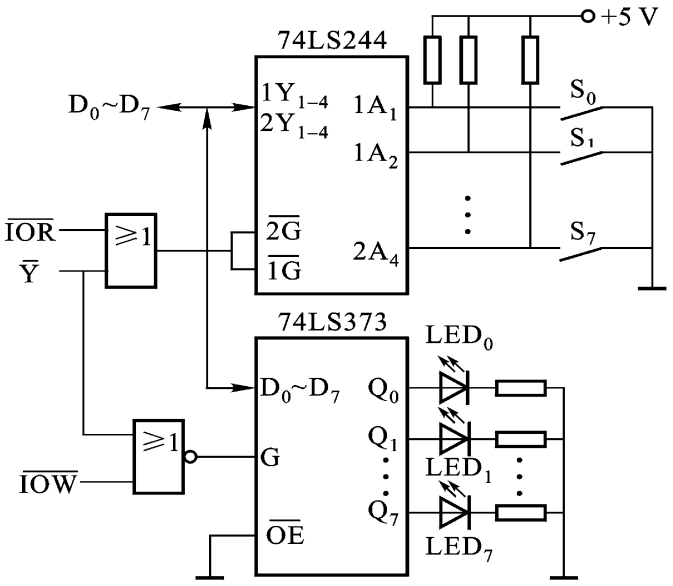


图 7.5　无条件传送接口举例

分析:开关 S_i闭合时,输入为低电平“0”,而点亮相应 LED_i,则输出为高电平“1”,输入与输出的关系相反。编写程序时,若采取先读入开关状态,再分析每一位的状态,然后决定 LED的亮灭,则该程序显得非常繁琐。具体程序如下:

```
CODE SEGMENT
    ASSUME CS:CODE
    MAIN PROC FAR
    START:  PUSH   DS
```

```
        MOV     AX, 0
        PUSH    AX
AGAIN:  MOV     AH, 1           ;读键盘缓冲区字符
        INT     16H
        CMP     AL, 1BH         ;若为 "Esc"键,则退出
        JZ      EXIT
        MOV     DX, 200H
        IN      AL, DX          ;读取开关状态
        NOT     AL              ;取反
        OUT     DX, AL          ;输出控制 LED
        JMP     AGAIN
EXIT:   RET     ;返回 DOS
MAIN    ENDP
CODE    ENDS
        END     START
```

2. 条件传输方式

条件传输方式,也称为查询传输方式,使用这种方式,CPU不断读取并测试外设的状态,如果外设处于"准备好"状态(输入设备)或"空闲"状态(输出设备),则CPU执行输入指令或输出指令与外设交换信息。为此,接口电路中除数据端口外,还必须有状态端口。对于条件传输来说,一个条件传输数据的过程一般由三个环节组成:

(1) CPU从接口中读取状态字;

(2) CPU检测状态字的相应位是否满足"就绪"条件,如果不满足,则转(1);

(3) 如状态位表明外设已处于"就绪"状态,则传输数据。

【例题 7.2】 从终端往缓冲区输入1行字符,当遇到回车符(0DH)或超过81个字符时,输入结束,并自动加上一个换行符(0AH)。如果在输入的81个字符中没有回车符,则在终端上输出信息"BUFFER OVERFLOW"。设终端接口的数据输入端口地址为32H,数据输出端口地址为34H,状态端口地址为36H。状态寄存器的 $D_1=1$,表示输入缓冲器已准备好数据,CPU可读取数据;状态寄存器的 $D_0=1$,表示输出缓冲器已空,CPU可往终端输出数据。终端接口电路具有根据相应操作对状态寄存器自动置1和清0功能。具体程序如下:

```
DATA SEGMENT
  MESS   DB 'BUFFER OVERFLOW', 0DH, 0AH
```

```
    BUFFER DB 82 DUP (?)
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA
    START:  MOV     AX, DATA
            MOV     DS, AX
            LEA     SI, BUFFER
            MOV     CX, 81
    INPUT:  IN      AL, 36H         ;读状态端口
            TEST    AL, 02H         ;测输入状态 D1位
            JZ      INPUT           ;未 "准备好 "转 INPUT
            IN      AL, 32H         ;读取输入字符
            MOV     [SI], AL        ;输入字符存缓冲区
            INC     SI
            CMP     AL, 0DH         ;输入字符为回车否？
            LOOPNE  INPUT           ;不是回车且接收字符个数未超过 81,
                                     转 INPUT
            JNE     OVERFLOW        ;不是回车且接收字符个数超过 81,转
                                     OVERFLOW
            MOV     AL, 0AH         ;是回车且接收字符个数≤81,存换行
                                     符
            MOV     [SI], AL
            JMP     EXIT            ;转程序结束处理
OVERFLOW:   OV      CX, 17          ;初始化输出字符个数
            LEA     SI, MESS        ;初始化显示字符串首址
OUTPUT:     IN      AL, 36H         ;读状态端口
            TEST    AL, 01H         ;测输出状态 D0位
            JZ      OUPUT           ;输出缓冲器未空,转 OUTPUT
            MOV     AL, [SI]        ;取出输出字符
            INC     SI
            OUT     34H, AL         ;输出字符
            LOOP    OUTPUT
EXIT:       MOV     AH, 4CH         ;返回 DOS
            INT     21H
```

```
CODE ENDS
    END START
```

7.3.2 中断方式

程序控制传输方式的缺点是CPU和外设只能串行工作,各外设之间也只能串行工作。为了使CPU和外设以及外设和外设之间能并行工作,以提高系统的工作效率,充分发挥CPU高速运算的能力,在计算机系统中引入了“中断”系统,利用中断来实现CPU与外设之间的数据传输方式即为中断传送方式。

在中断传输方式下,当输入设备将数据准备好或输出设备可以接收数据时,便可向CPU发出中断请求,使CPU暂时停止执行当前程序,而去执行一个数据输入/输出的中断服务程序,与外设进行数据传输操作,中断服务程序执行完后,CPU又返回继续执行原来的程序。这样在一定程度上实现了主机与外设的并行工作。同时,若某一时刻有几个外设发出中断请求,CPU可根据预先安排的优先顺序,按轻重缓急处理几个外设的请求,这样在一定程度上也可实现几个外设的并行工作。

利用中断方式进行数据传输,CPU不必花费大量时间在两次输入或输出过程间对接口进行状态测试和等待,从而大大提高了CPU的效率。

7.3.3 直接存储器存取(DMA)方式

在程序控制的传送方式中,所有传送均通过CPU执行指令来完成,而CPU指令系统只支持CPU和内存或外设间的数据传输。如果外设要和内存进行数据交换,即使使用效率较高的中断传送,也免不了要走外设→CPU→内存这条路线或相反的路线,这样限制了传输的速度。若I/O设备的数据传输速率较高(如硬盘驱动器),那么CPU和这样的外设进行数据传输时,即使尽量压缩程序查询方式或中断方式中的非数据传输时间,也仍然不能满足要求。为此,提出了在外设和内存之间直接进行数据传输的方式,即DMA方式。

DMA方式是指不经过CPU的干预,直接在外设和内存之间进行数据传输的方式。一次DMA传输需要执行一个DMA周期(相当于一个总线读或写周期)。数据的传输速度基本上取决于外设和内存的速度,因此能够满足高速外设数据传输的需要。

实现DMA方式,需要一个专门的接口器件来协调和控制外设接口和内存之间的数据传输,这个专门的接口器件称为DMA控制器(DMAC)。

在采用 DMA方式进行数据传输时,当然也要利用系统的数据总线、地址总线和控制总线。系统总线原来是由 CPU控制管理的。在用 DMA方式进行数据传输时,DMAC向 CPU发出申请使用系统总线的请求,当 CPU同意并让出系统总线控制权后,DMAC接管系统总线,实现外设与内存之间的数据传输,传输完毕,将总线控制权交还给 CPU。DMAC是一个专用接口电路,在系统中的连接如图 7.6所示。

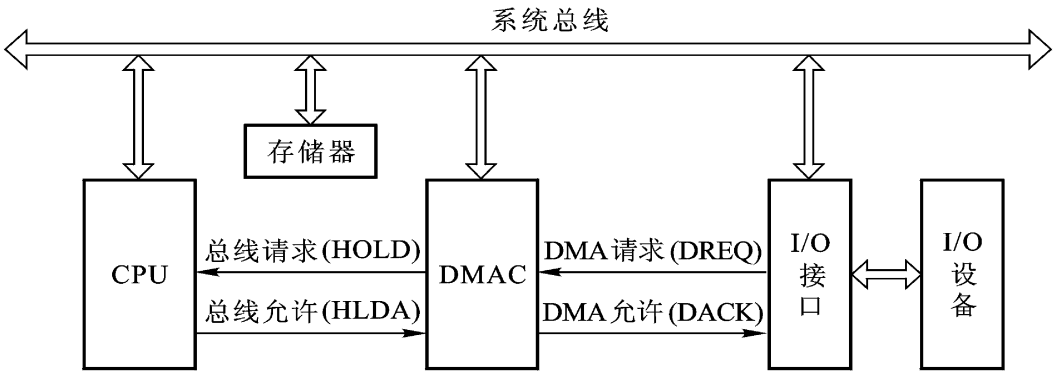


图 7.6　DMAC与系统的连接

DMA操作的基本方法有三种:

⑴ CPU停机方式　指在 DMA传送时,CPU停止工作,不再使用总线。该方式比较容易实现,但由于 CPU停机,可能影响到某些实时性很强的操作,如中断响应等。

⑵ 周期挪用方式　利用窃取 CPU不进行总线操作的周期,来进行 DMA传送。这一方式不影响 CPU的操作,但需要复杂的时序电路,而且数据传送过程是不连续的和不规则的。

⑶ 周期扩展方式　该方式需要专门时钟电路的支持,当传送发生时,该时钟电路向 CPU发送加宽的时钟信号,CPU在加宽时钟周期内操作不往下进行;另一方面,仍向 DMAC发送正常的时钟信号,DMAC利用这段时间进 DMA传送。

7.4　DMA控制器 8237A

直接存储器存取 (DMA)是一种外设与存储器之间直接传输数据的方法,适用于需要数据高速大量传送的场合。DMA数据传送利用 DMA控制器进行控制,不需要 CPU直接参与。

Intel8237A是一种高性能的可编程 DMA控制器芯片。在 5 MHz时钟频率下,其传送速率可达 1.6 MB/s。每片 8237A有 4个独立的 DMA通道,即有 4个

DMA控制器（DMAC）。每个DMA通道具有不同的优先权，都可以允许和禁止。每个通道有4种工作方式，一次传送的最大长度可达64 KB。多片8237A可以级连，任意扩展通道数。

7.4.1 8237A的内部结构和引脚

8237A要在DMA传送期间作为系统的控制器件，所以，它的内部结构和外部引脚都相对比较复杂。从应用角度看，内部结构主要由两类寄存器组成。一类是通道寄存器，它们是现行地址寄存器、现行字节数寄存器和基地址寄存器、基字节数寄存器，这些寄存器都是16位的寄存器。另一类是控制和状态寄存器，它们是方式寄存器（6位寄存器）、命令寄存器（8位）、状态寄存器（8位）、屏蔽寄存器（4位）、请求寄存器（4位）、临时寄存器（8位）。内部寄存器的作用在7.4.4节中介绍。

8237A的内部结构框图及外部引脚如图7.7所示。

时钟（CLK）：输入信号，提供8237A正常工作所需的时钟。

复位（RESET）：高电平有效，输入信号。此信号有效时，屏蔽寄存器被置位，其它寄存器被复位，且芯片处于空闲周期。

准备好（READY）：高电平有效，输入信号。在DMA传送的第3个时钟周期S_3的下降沿检测READY信号，若READY信号为低，则插入等待状态S_W，直到READY信号为高才进入第4个时钟周期S_4。

片选（$\overline{CS}$）：低电平有效，输入信号。该信号有效，CPU访问8237A。

DMA通道请求（$DREQ_0$～$DREQ_3$）：每个通道对应一个DREQ信号，用以接受外设的DMA请求。当外设需要请求DMA服务时，将DREQ信号置成有效电平，并要保持到产生响应信号。DREQ有效极性由编程选择。8237A芯片被复位后，初始为高电平有效。

DMA通道响应（$DACK_0$～$DACK_3$）：每个通道对应一个DACK信号，是8237A对外设DMA请求的响应信号。8237A一旦获得HLDA有效信号，便使请求服务的通道产生相应的DMA响应信号以通知外设。DACK输出信号的有效极性由编程选择。8237A被复位后，初始为低电平有效。

总线请求（HRQ）：当外设的I/O接口要求DMA传输时，向8237A发送DMA请求DREQ，若允许该通道产生DMA请求，则8237A输出有效的HRQ高电平，向CPU申请使用系统总线。

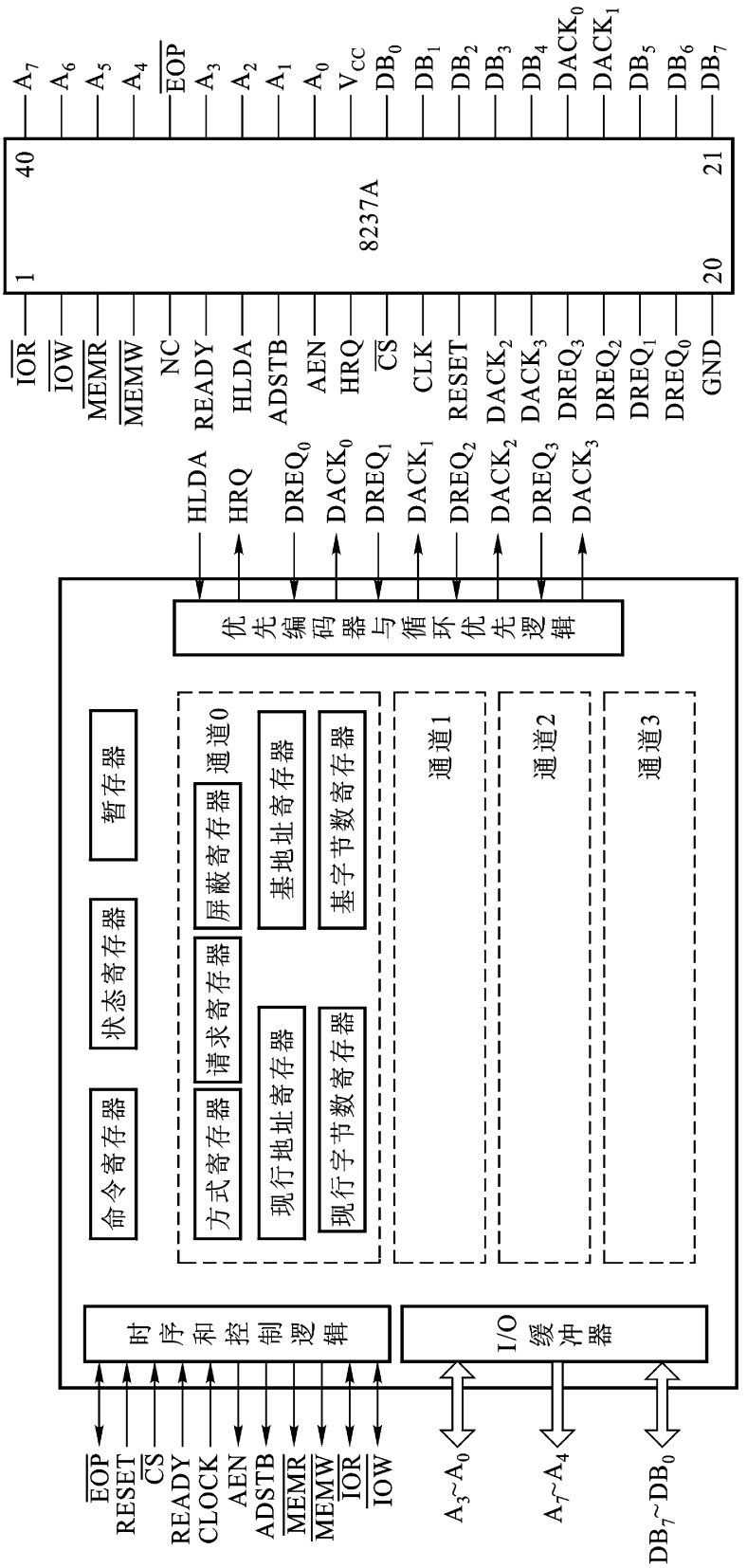


图 7.7 8237A 内部结构框图及引脚图

总线响应（HLDA）：当 8237A向 CPU发出总线请求信号后，至少再过一个时钟周期，CPU才发出总线响应信号 HLDA，这样，8237A可获得总线的控制权。

低 4位地址线（A_0 ～A_3）：双向地址线。当 8237A为从器件时，A_0 ～A_3 作为输入信号，CPU可通过它们对 8237A的内部寄存器进行寻址，从而实现对 8237A的编程；当 8237A为主器件时，A_0 ～A_3 作为输出信号，输出低 4位地址。

高 4位地址线（A_4 ～A_7）：当 8237A为主器件时，A_4 ～A_7 输出低 8位地址中的高 4位地址。

数据线（DB_0 ～DB_7）：双向三态数据线。当 8237A为主器件时，用于 8237A与微处理器进行数据交换；当 8237A为从器件时，输出当前地址寄存器的高 8位地址。

地址选通（ADSTB）：高电平有效。此信号有效时，把 8237A当前地址寄存器中的高 8位地址锁存到 DMA外部地址锁存器。

地址允许（AEN）：高电平有效。此信号有效时，将 8237A的外部地址锁存器中的高 8位地址送到地址总线 A_{15} ～A_8 上，与芯片直接输出的低 8位地址组成内存单元的偏移地址。AEN在 DMA传送期间也可作为使其他处理器输出的地址无效的控制信号。

存储器读（$\overline{MEMR}$）：三态输出信号，低电平有效。有效时，所选中的存储单元的内容被读到数据总线上。

存储器写（$\overline{MEMW}$）：三态输出信号，低电平有效。有效时，数据总线上的内容被写入选中的存储单元。

输入输出读（$\overline{IOR}$）：双向三态，低电平有效。当 8237A为从器件时，$\overline{IOR}$作为 8237A的输入信号，此信号有效时，CPU读取 8237A内部寄存器的值；当 8237A为主器件时，$\overline{IOR}$作为 8237A的输出信号，此信号有效时，请求 DMA的 I/O接口部件中的数据被读出并送往数据总线。

输入输出写（$\overline{IOW}$）：双向三态，低电平有效。当 8237A为从器件时，$\overline{IOW}$作为 8237A的输入信号，此信号有效时，CPU往 8237A内部寄存器写入信息；当 8237A为主器件时，$\overline{IOW}$作为 8237A的输出信号，此信号有效时，从指定存储单元中读出的数据被写入 I/O接口中。

过程结束（$\overline{EOP}$）：低电平有效，双向信号，在 DMA传送时，当当前字节数寄存器的计数值从 0减到 FFFFH时（即内部 DMA过程结束），从 $\overline{EOP}$引脚上输出一个负脉冲。若由外部输入 $\overline{EOP}$信号，DMA传送过程被强迫终止。不论是内部还是外部产生 $\overline{EOP}$信号，都会终止 DMA数据传送。

7.4.2 8237A的工作周期和时序

8237A有两种工作周期:即空闲周期和有效周期,每一个周期由多个时钟周期组成。

1. 空闲周期

当 8237A的任一通道无 DMA请求时就进入空闲周期,在空闲周期 8237A始终处于 S_1状态,每个 S_1状态都采样通道的请求输入线 DREQ。此外,8237A在 S_1状态还采样片选信号 $\overline{CS}$,当 $\overline{CS}$为低电平,且 4个通道均无 DMA请求,则 8237A进入编程状态,即 CPU对 8237A进行读/写操作。8237A在复位后处于空闲周期。

2. 有效周期

当 8237A在 S_1状态采样到外设有 DMA请求时,就脱离空闲周期进入有效周期,8237A作为系统的主控芯片,控制 DMA传送操作。由于 DMA传送是借用系统总线完成的,所以,它的控制信号以及工作时序类似 CPU总线周期。图 7.8为 8237A的 DMA传送时序,每个时钟周期用 S状态表示,而不是 CPU总线周期的 T状态。

⑴ 当在 S_1脉冲的下降沿检测到某一通道或几个通道同时有 DMA请求时,则在下一个周期就进入 S_0状态;而且在 S_1脉冲的上升沿,使总线请求信号 HRQ有效。在 S_0状态 8237A等待 CPU对总线请求的响应,只要未收到有效的总线请求应答信号 HLDA,8237A始终处于 S_0状态。当在 S_0的上升沿采样到有效的 HLDA信号,则进入 DMA传送的 S_1状态。

⑵ 典型的 DMA传送由 S_1,S_2,S_3,S_4四个状态组成。在 S_1状态使地址允许信号 AEN有效。自 S_1状态起,一方面把要访问的存储单元的高 8位地址通过数据线 DB_0~DB_7输出,另一方面发出一个有效的地址选通信号 ADSTB,利用 ADSTB的下降沿把在数据线上的高 8位地址锁存至外部的地址锁存器中。同时,地址的低 8位由地址线 A_0~A_7输出,且在整个 DMA传送期间保持不变。

⑶ 在 S_2状态,8237A向外设输出 DMA响应信号 DACK。在通常情况下,外设的请求信号 DREQ必须保持到 DACK有效。即自 S_2状态开始使“读写控制”信号有效。

如果将数据从存储器传送到外设,则 8237A输出 $\overline{MEMR}$有效信号,从指定的存储单元读出一个数据并送到系统数据总线上,同时 8237A还输出 $\overline{IOW}$有效信号将系统数据总线的这个数据写入请求 DMA传送的外设中。

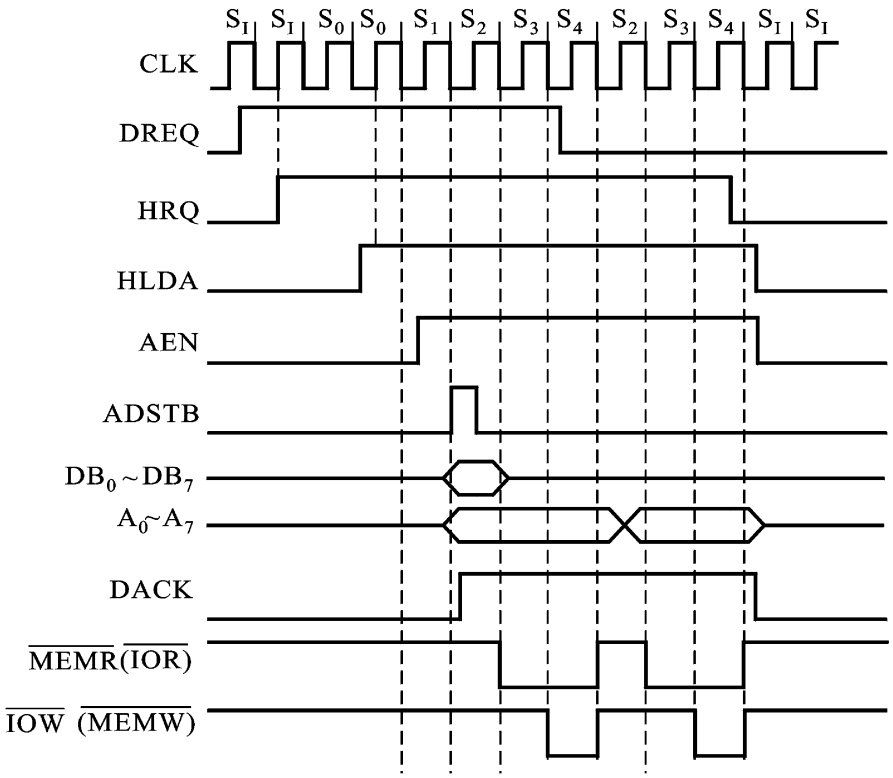


图 7.8 8237A的 DMA传送时序

如果将数据从外设传送到存储器,则 8237A 输出 $\overline{IOR}$有效信号,从请求 DMA传送的外设读取一个数据并送到系统数据总线上,同时 8237A 还输出 $\overline{MEMW}$有效信号将系统数据总线的这个数据写入指定的存储单元。

由此可见,DMA传送实现了外设与存储器之间的直接数据传送,传送的数据不进入 8237A内部,也不进入 CPU。另外,DMA传送不提供 I/O端口地址(地址线上总是存储器地址),请求 DMA传送的外设需要利用 DMA响应信号进行译码以确定外设数据缓冲器。

(4) 在 8237A输出信号的控制下,利用 S_3 和 S_4 状态完成数据传送。若存储器和外设不能在 S_4 状态前完成数据的传送,则只要设法使 READY信号变低,就可以在 S_3 和 S_4 状态间插入 S_W 等待状态。在此状态,所有控制信号维持不变,从而加宽 DMA传送的周期。

(5) 在数据块传送方式下,S_4 后面应接着传送下一个字节。因为 DMA传送的存储器区域是连续的,通常情况下地址的高 8位不变,只是低 8位增量或减量。所以,输出和锁存高 8位地址的 S_1 状态不需要了,直接进入 S_2 状态,由输出地址的低 8位开始,在读写信号的控制下完成数据传送。这种过程一直继续到把规定的数据个数传送完。此时,一个 DMA传送过程结束,8237A又进入空

闲周期,等待新的请求。

7.4.3 8237A的工作方式和传送类型

1. 8237A 工作方式

⑴ 单字节传送方式

单字节传送方式是 8237A 每传送一个字节之后就释放总线。传送一个字节之后,字节计数器减 1,地址寄存器加 1或减 1,HRQ变为无效。这样,8237A 释放系统总线,将控制权还给 CPU。当字节计数器从 0减到 FFFFH,产生终止计数信号,使$\overline{EOP}$变为低电平,从而结束 DMA传输。单字节传送方式的优点是保证在两次 DMA操作之间 CPU有机会获得至少一个总线周期的总线控制权,达到 CPU与 DMA控制器并行工作的状态。

⑵ 数据块传送方式

在这种方式下,8237A由 DREQ启动,连续传送数据,当字节计数器从 0减到 FFFFH,产生终止计数信号,使$\overline{EOP}$变为低电平,或者由外部输入有效的$\overline{EOP}$信号,终止 DMA传送。在数据块传送方式中,要求 DREQ保持到 DACK变为有效时即可,且一次 DMA操作最多传送 64KB。数据块传送方式的特点是:一次请求传送一个数据块,效率高;但在整个 DMA传送期间,CPU无法控制总线。

⑶ 请求传送方式

与数据块传送方式类似。但当 DREQ信号变为无效时,则暂停 DMA传送;当 DREQ再次变为有效时,DMA传送继续进行,直至字节计数器从 0减到 FFFFH,或者由外部送来一个有效的$\overline{EOP}$信号。

⑷ 级连方式

8237A可以多级级联,扩展 DMA通道。第二级的 HRQ和 HLDA信号连到第一级某个通道的 DREQ和 DACK上,第二级芯片的优先权与所连接的通道相对应。

2. DMA传送类型

⑴ 读传输:是指从指定的存储器单元读出数据写入到相应的 I/O设备。DMA控制器发出$\overline{MEMR}$和$\overline{IOW}$信号。

⑵ 写传输:是指从 I/O设备读出数据写入到指定的存储器单元。DMA控制器发出$\overline{MEMW}$和$\overline{IOR}$信号。

⑶ DMA传送:是一种伪传送操作,用于校验 8237A的内部功能。它与读传输和写传输一样产生存储器地址和时序信号,但存储器和 I/O的读写控制信号无效。

⑷ 存储器到存储器的传送:使用此方式 8237A可实现存储器内部不同区域之间的传输。这种传送类型仅适用于通道 0和通道 1,此时通道 0的地址寄存器存源数据区地址,通道 1的地址寄存器存目的数据区地址,通道 1的字节数计数器存传送的字节数。传送由设置通道 0的 DMA请求(设置请求寄存器)启动,8237A按正常方式向 CPU发出 HRQ请求信号,待 HLDA响应后传送就开始。每传送一个字节需用 8个状态,前 4个状态用于从源存储器中读取数据并存放于 8237A中的数据暂存器,后 4个状态用于将数据暂存器的内容写入目的存储器中。

7.4.4 8237A的寄存器

8237A共有 10种内部寄存器,对它们的操作有时需要配合三个软件命令,它们由最低地址 A_0 ~A_3 区分,如表 7.2所示。

表 7.2 8237A寄存器和软件命令的寻址

$A_3A_2A_1A_0$	读操作	写操作
0000	通道 0现行地址寄存器	通道 0地址寄存器
0001	通道 0现行字节数寄存器	通道 0字节数寄存器
0010	通道 1现行地址寄存器	通道 1地址寄存器
0011	通道 1现行字节数寄存器	通道 1字节数寄存器
0100	通道 2现行地址寄存器	通道 2地址寄存器
0101	通道 2现行字节数寄存器	通道 2字节数寄存器
0110	通道 3现行地址寄存器	通道 3地址寄存器
0111	通道 3现行字节数寄存器	通道 3字节数寄存器
1000	状态寄存器	命令寄存器
1001	—	请求寄存器
1010	—	单通道屏蔽字
1011	—	方式寄存器
1100	—	清先/后触发器命令
1101	暂存器	复位命令
1110	—	清屏蔽寄存器命令
1111	—	综合屏蔽字

1. 基地址寄存器

用于保存 DMA传送的起始地址,不能被 CPU读出。

2. 现行地址寄存器

保存 DMA传送的当前地址,每次传送后这个寄存器的值自动加 1或减 1。这个寄存器的值可由 CPU写入和读出。其初值就是基地址寄存器内容。

3. 基字节数寄存器

用于保存每次 DMA操作需要传送数据的字节总数,不能被 CPU读出。

4. 现行字节数寄存器

保存 DMA还需传送的字节数,每次传送后减 1。这个寄存器的值可由 CPU写入和读出。当这个寄存器的值从 0减到 FFFFH时,产生终止计数信号,使 $\overline{EOP}$变为低电平。

5. 方式寄存器

存放相应通道的方式控制字。方式控制字的格式如图 7.9所示,用于设置某个 DMA通道的工作方式,其中最低 2位选择 DMA通道。

D_4:自动预置功能选择位。若 8237A被设置为允许自动预置功能,则当 DMA传送结束 $\overline{EOP}$有效时,现行地址寄存器和现行字节数寄存器会从基地址寄存器和基字节数寄存器中重新取得初值,从而又可以进入下一个数据传输过程。

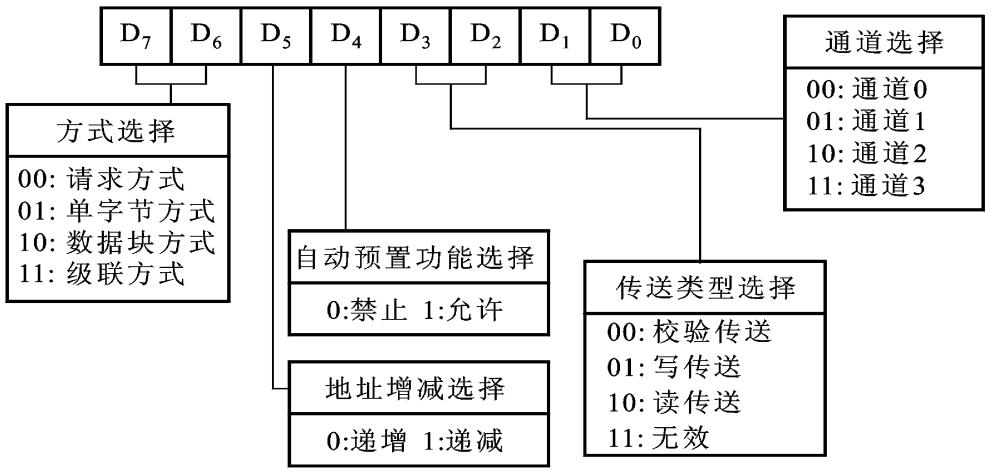


图 7.9　8237A方式控制字

6. 命令寄存器

存放 8237A的命令字。命令字格式如图 7.10所示,用于设置 8237A的操作方式。

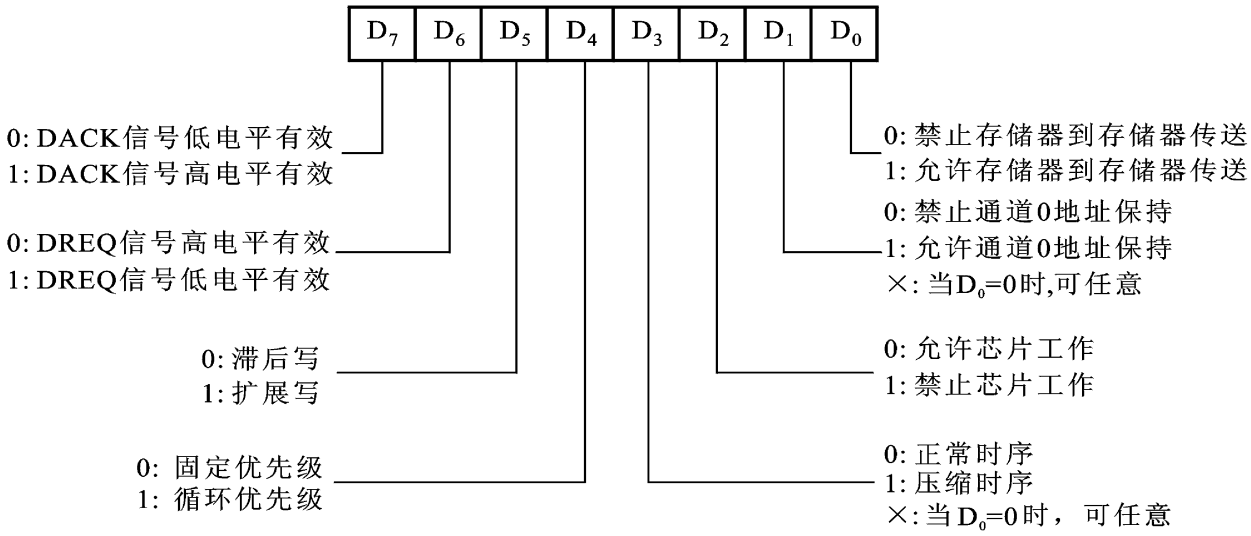


图 7.10 8237A命令字

当 D_0 =1时选择存储器到存储器的传送方式,此时,通道 0的地址寄存器存放源地址,通道 1的地址寄存器和字节计数器存放目的地址和计数值。若 D_1 也为“1”,则整个存储器到存储器的传送过程始终保持同一个源地址,以便实现将一个目的存储区域设置为同一个值。

D_3 :决定是压缩时序还是普通时序。8237A工作于压缩时序时,进行一次DMA传输需 2个时钟周期,而工作于普通时序时,进行一次 DMA传输需 3个时钟周期。

D_4 :决定 4个通道的优先级方式。一种是固定优先级方式,即通道 0的优先级最高,通道 3的优先级最低。另一种是循环优先级方式,即某通道进行一次传输以后,其优先级降为最低。

D_5 :决定是否扩展写信号。关于扩展写信号说明如下:如果外部设备的速度较慢,必须用普通时序工作,若普通时序仍不能满足要求,就要在硬件上通过READY信号使 8237A插入 S_W 状态。有些设备是用 8237A送出的 $\overline{IOW}$ 或 $\overline{MEMW}$ 信号的下降沿产生 READY信号响应的,而这两个信号是在 S_4 状态才送出的。为使 READY信号早点到来,将这两个信号扩展到 S_3 状态开始有效。

7. 请求寄存器

除可以利用硬件提出 DMA请求外,还可通过软件发出 DMA请求,其格式如图 7.11所示。

8. 屏蔽寄存器

用于控制每个通道的 DMA请求是否有效。对屏蔽寄存器的写入有 3种

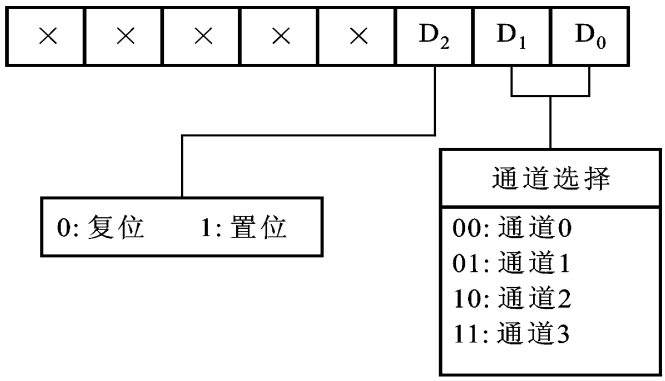
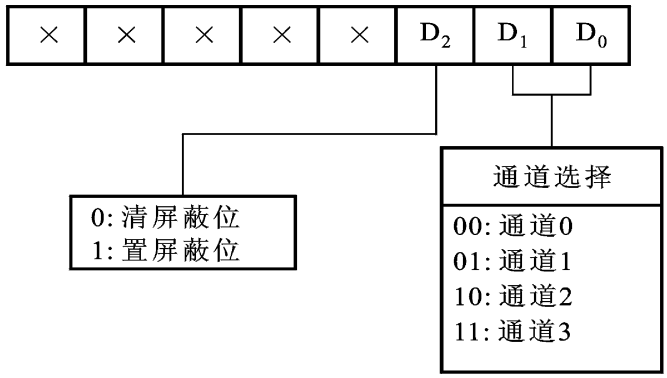


图 7.11　8237A请求字

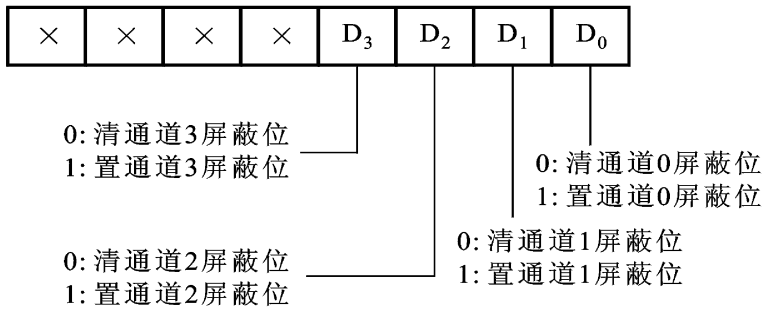
方法：

⑴ 单通道屏蔽字，实现对某一通道 DMA屏蔽标志的设置，其格式如图7.12a所示。

⑵ 综合屏蔽字，实现对 4个通道 DMA屏蔽标志的设置，其格式如图7.12b



(a) 单通道屏蔽字



(b) 综合屏蔽字

图 7.12　8237A屏蔽字

所示。

9. 状态寄存器

状态寄存器的各位分别表示各通道是否有 DMA 请求及是否终止计数，其格式如图 7.13所示。

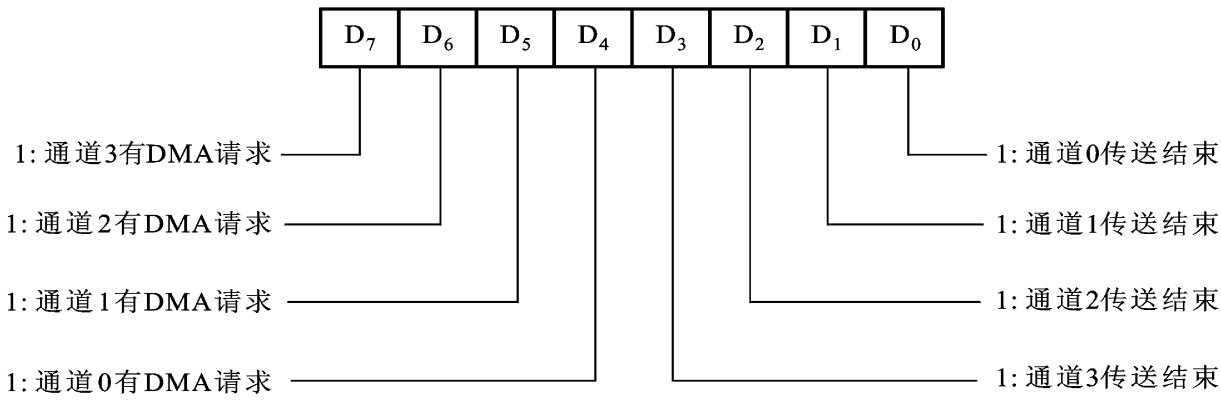


图 7.13 8237A状态字

10. 暂存器

在存储器到存储器的传送方式下，暂存器用于保存从源存储单元读出的数据。

7.4.5 8237A的软件命令

1. 复位命令

复位命令也叫综合清除命令，它的功能和 RESET信号相同。复位命令使命令寄存器、状态寄存器、请求寄存器、暂存器以及先/后触发器清 0，而使屏蔽寄存器置位。

2. 清除先/后触发器命令

先/后触发器是用来控制 DMA 通道中地址寄存器和字节计数器的初值设置的。由于 8237A 只有 8位数据线，所以，一次只能传输一个字节。而地址寄存器和字节计数器都是 16位的，这些寄存器都要通过两次传输才能完成初值设置。为了保证能正确设置 16位初值，应先发出清除先/后触发器命令，写入低 8位数据后，先/后触发器自动置 1，写入高 8位数据后，先/后触发器自动复位为 0。

3. 清除屏蔽寄存器命令

使 4个屏蔽位都清 0，即 4个通道的 DMA请求都被允许。

7.4.6　8237A的应用

1. 8237A在 IBM PC/XT上的应用

IBM PC/XT机使用一片 8237A。通道 0用来对动态存储器刷新;通道 2和通道 3分别用于软盘和硬盘驱动器与内存之间的数据传输;通道 1用作同步数据链路通信 (SDLC)卡与存储器之间的数据传输,若系统不使用该通信卡,则可供用户使用。

根据系统板 I/O译码电路所产生的 DMA片选信号,DMAC的端口地址范围是 00H～1FH,DMAC的 A_3～A_0脚同系统地址线 A_3～A_0相连,A_4未参加译码,取 A_4=0时的地址 00～0FH为 DMAC的端口地址。

8237A只提供 16位地址,系统的高 4位地址由附加逻辑电路 (页面寄存器)提供,以形成整个微机系统需要的所有存储器地址。系统分配给页面寄存器的端口地址为 80H～83H。

【例题 7.3】　在 IBMPC/XT中,利用 8237A通道 0输出存储器地址进行 DRAM的刷新操作,其 DMA传送程序如下:

```
MOV     AL,0
OUT     0DH,AL          ; MAC复位命令
MOV     AL,0            ; MAC命令字:固定优先权,DREQ高有效、DACK
                          低有效、滞后写、正常时序
OUT     08H,AL
MOV     AL,0
OUT     00H,AL          ;写入通道 0的地址寄存器低字节
OUT     00H,AL          ;写入通道 0的地址寄存器高字节
MOV     AL,0FFH
OUT     01H,AL          ;写入通道 0的字节数寄存器低字节
OUT     01H,AL          ;写入通道 0的字节数寄存器高字节
MOV     AL,58H          通道 0方式字:单字节传送、DMA读、地址增量、
                          自动初始化
OUT     0BH,AL
MOV     AL,0            通道 0屏蔽字:允许 DREQ0提出申请
OUT     0AH,AL
```

2. DMA写传输

【例题 7.4】　假设采用 IBM PC/XT中 DMA通道 1,传送 2KB外设数据,内

存起始地址为 45000H。其程序如下：

```
        MOV     AL, 45H         通道 1方式字 :单字节 DMA 写传送,地址增量,非
                                自动初始化
        OUT     0BH, AL
        OUT     0CH, AL         清先 /后触发器
        MOV     AL, 0
        OUT     02H, AL         ;写入低 8位地址到地址寄存器
        MOV     AL, 50H
        OUT     02H, AL         ;写入中 8位地址到地址寄存器
        MOV     AL, 04H
        OUT     81H, AL         ;写入高 4位地址到页面寄存器
        MOV     AX, 2047        ; X←传送字节数减 1
        OUT     03H, AL         送字节数低 8位到字节数寄存器
        MOV     AL, AH
        OUT     03H, AL         送字节数高 8位到字节数寄存器
        MOV     AL, 01H
        OUT     0AH, AL         ;单通道屏蔽字 :允许通 1的 DMA请求
        ……                      ;其他工作
DMALP:
        IN      AL, 08H         ;读状态寄存器
        AND     AL, 02H         ;判断通道 1是否传送结束
        JZ      DMALP           ;没有结束,则循环等待
        ……                      ;传送结束,处理转换数据
```

思考题与习题

7.1 简述 I/O接口的功能。

7.2 CPU与外设之间的数据传输方式有哪些？简要说明各自含义。

7.3 什么是端口？通常有哪几类端口？计算机对 I/O端口编址时通常采用哪两种方法？在 80X86系统中,采用哪一种方法？

7.4 现有一输入设备,其数据端口地址为 FFE0H,状态端口地址为 FFE2H,当其 D_0 位为 1时表明输入数据准备好。试采用查询方式,编程实现从该设备读取 100个字节数据并保存到 2000H:2000H开始的内存中。

7.5 硬件如图 7.5所示,试编程实现 :S_0 控制 8个发光二极管 1亮 7暗,S_1 控制 8个发

光二极管 7亮 1暗，S_2 控制某一亮点 (或暗点)以一定时间间隔循环向左移动，S_3 控制某一亮点 (或暗点)以一定时间间隔循环向右移动，两个或两个以上开关闭合，则结束程序。

7.6 用 DMA控制器 8237A进行内存到内存传输时，有什么特点？

第8章

中　断

中断是微机系统中非常重要的一种技术，是对微处理器功能的有效扩展。利用外部中断，微机系统可以实时响应外部设备的数据传送请求，能够及时处理外部意外或紧急事件。利用内部中断，微处理器为用户提供了发现、调试并解决程序执行时异常情况的有效途径。

本章讨论微机中断系统的功能、中断过程、中断管理以及 80X86的中断系统，并详细介绍中断控制器 8259A的工作原理及应用。

8.1 概　述

8.1.1 中断的基本概念

1. 中断和中断源

所谓“中断”是指 CPU中止正在执行的程序，转去执行请求 CPU为之服务的内、外部事件的服务程序，待该服务程序执行完后，又返回到被中止的程序中继续运行的过程。

引起 CPU中断的事件称为“中断源”。常见的中断源有：

⑴ 外部设备的请求，如 CRT终端、键盘、打印机等；

⑵ 由硬件故障引起的，如电源掉电，硬件损坏等；

⑶ 实时时钟，如定时器芯片等；

⑷ 由软件引起的,如程序错、运算错、为调试程序而设置的断点等。

2. 中断系统功能

为满足上述中断要求,中断系统应具有以下功能:

⑴ 能实现中断响应、中断服务及中断返回。当某一中断源发出中断请求时,CPU能决定是否响应这一中断请求,若允许响应这个中断请求,CPU在保护断点后,将转移到相应的中断服务程序中,中断处理完,CPU返回到原断点处继续执行原程序。

⑵ 能实现中断优先权排队。当有两个或多个中断源同时提出中断请求时,中断系统能根据各中断源的性质分清轻重缓急,给出处理的先后顺序,确保优先级别较高的中断请求先处理。

⑶ 能实现中断嵌套。若在中断处理过程中又有新的优先级较高的中断源提出请求,中断系统要能使 CPU暂停当前中断服务程序的执行,转去响应和处理优先级较高的中断请求,处理完后再返回原优先级较低的中断服务程序中。

8.1.2 中断处理过程

对于不同的微型计算机系统,CPU进行中断处理的具体过程不完全一样,即使是同一台微型计算机,由于中断方式的不同 (如可屏蔽中断、不可屏蔽中断等),中断处理也会有差别,但一个完整的中断处理的基本过程应包括:中断请求、中断判优、中断响应、中断处理及中断返回 5个基本阶段。

1. 中断请求

中断请求是中断过程的第一步。中断源产生中断请求的条件,因中断源而异。

2. 中断判优

由于中断产生的随机性,可能出现两个或两个以上的中断源同时提出中断请求的情况。设计者必须根据中断源的轻重缓急,给每个中断源确定一个中断级别,CPU首先响应优先级别最高的中断源的请求,处理完毕后,再响应级别较低的中断源的请求。中断判优的另一作用是决定可否实现中断嵌套。当 CPU响应某一中断请求并为之服务时,若有一个优先权更高的中断源发出请求,CPU应能及时响应;反之,若有一个优先权较低的中断源发出请求,中断判优电路应屏蔽这一中断请求,直至原有中断请求服务完再响应优先权较低的中断请求。

3. 中断响应

CPU收到中断请求后,首先判断能否接受。若能接受,则响应该中断请求。

通常中断响应的操作过程应包括：保留断点地址、关闭中断允许、转入中断服务程序。

8086微处理器有两个引脚接收中断请求信号，一个是非屏蔽中断（NMI），另一个是可屏蔽中断（INTR）。NMI引脚一旦接收到请求，CPU立即予以响应；INTR引脚接收到的请求，受标志寄存器的IF标志位控制，当IF＝1，CPU允许中断；而当IF＝0，CPU禁止中断。

CPU响应中断的条件：① 接收到中断请求信号；② 若是INTR类中断，CPU必须允许响应；③ 等现行指令执行完。

4. 中断处理

中断处理通常是由中断服务程序完成的，一般按以下模式设计：

⑴ 保护现场　为不使中断服务程序的运行影响主程序的状态，将中断服务程序中用到的寄存器内容压入堆栈保护。

⑵ 执行中断服务程序　这是中断处理的核心部分，完成中断源要求完成的任务。

⑶ 恢复现场　将中断服务程序执行前保护的信息从堆栈中弹出恢复到原寄存器。

5. 中断返回

执行中断返回指令，返回到原程序断点处继续运行。

8.1.3 中断优先级（优先权）

在微机系统中，常常遇到多个中断源同时提出中断请求的情况，此时CPU必须确定首先为哪个中断源服务，以及服务的顺序，这些都由中断判优逻辑来解决。

中断优先权管理有两层含义：一是多个中断源同时提出请求时，应首先响应优先权高的中断请求；二是当CPU正在处理某一级中断请求时，又有其他的中断请求产生，这时应能响应更高一级的中断请求，而屏蔽掉同级或较低级的中断请求。通常，中断判优逻辑的具体实现方法有以下三种。

1. 软件查询方式

软件查询方式是在CPU响应中断后执行查询程序，以确定请求中断的中断源的优先权。使用软件查询方式需相关硬件接口电路，将若干个中断源经“或门”后，形成一个公共的中断请求INTR，这样，只要有一个中断请求，就可向CPU发中断请求INTR信号。CPU响应中断后，进入一个公用的中断处理程序，该程序读出外设中断请求状态信息，依次检测中断请求状态位，若有请求，则转相应

中断处理程序。先检测的优先权高,后检测的优先权低。硬件接口电路和软件查询程序流程如图 8.1所示。

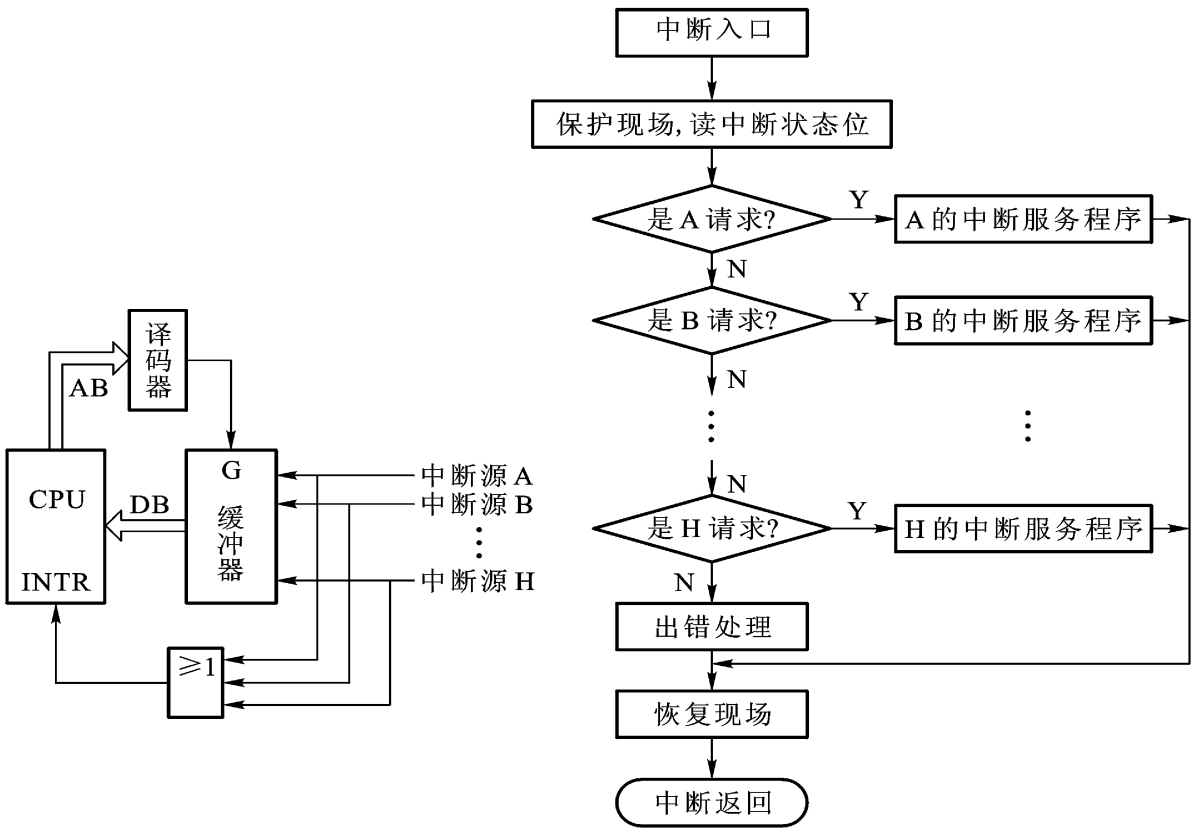


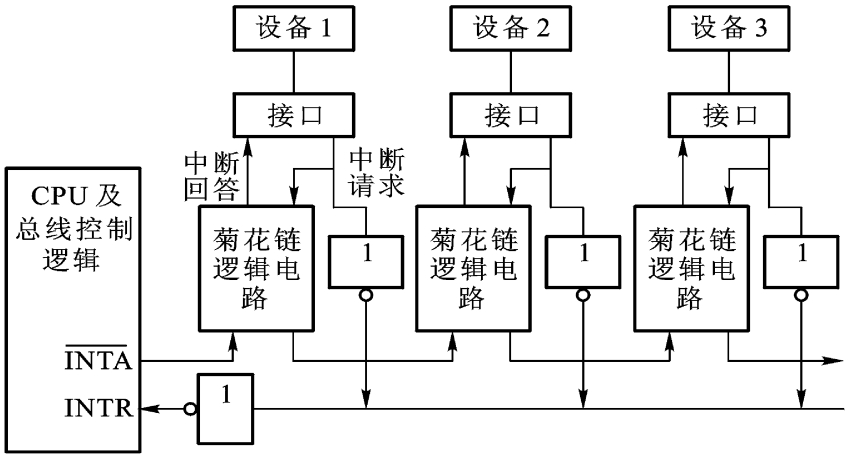
图 8.1 硬件接口电路和软件查询程序流程

2. 链式优先权排队 (菊花链法)

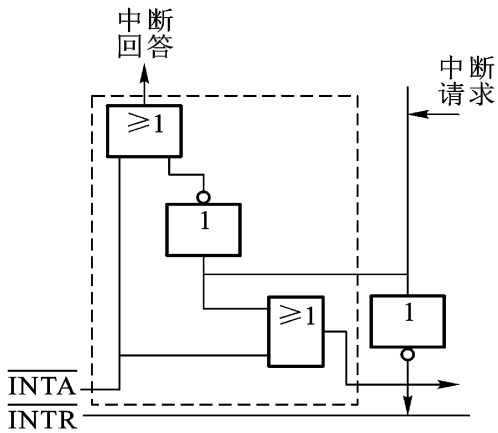
菊花链法是得到中断优先级控制的硬件方法。其原理是在每个中断源的接口电路中设置一个菊花链逻辑电路。当某一接口有中断请求时,会向CPU发送中断请求信号,若 CPU允许中断,则 CPU发出中断响应信号 $\overline{INTA}$。$\overline{INTA}$信号在菊花链中传递,如果某接口中无中断请求信号,则 $\overline{INTA}$信号通过菊花链逻辑电路,原封不动地向后传递;如果某接口中有中断请求信号,则该接口的菊花链逻辑电路阻塞 $\overline{INTA}$信号向后传递。显然,在多个中断请求同时发生时,最靠近 CPU的接口,优先权最高。菊花链式优先排队电路如图 8.2所示。

3. 可编程中断控制器

中断控制器是集中断请求、中断屏蔽、中断判优、中断源类型码提供等功能



(a) 菊花链



(b) 菊花链逻辑电路

图 8.2 菊花链式优先排队电路

于一身的专用大规模集成芯片。采用可编程中断控制器是当前微型计算机中解决中断的最常用方案。Intel公司的 8259A,就是具有上述功能的可编程中断控制器。可编程中断控制器 8259A将在 8.3节中详细论述。

8.2 80X86中断系统

80X86微机具有一个简单而灵活的中断系统,可处理 256种不同的中断请求。这些中断可分为两大类,即外部中断 (硬件中断)和内部中断 (软件中断)。中断源如图 8.3所示,中断优先级如表 8.1所示。

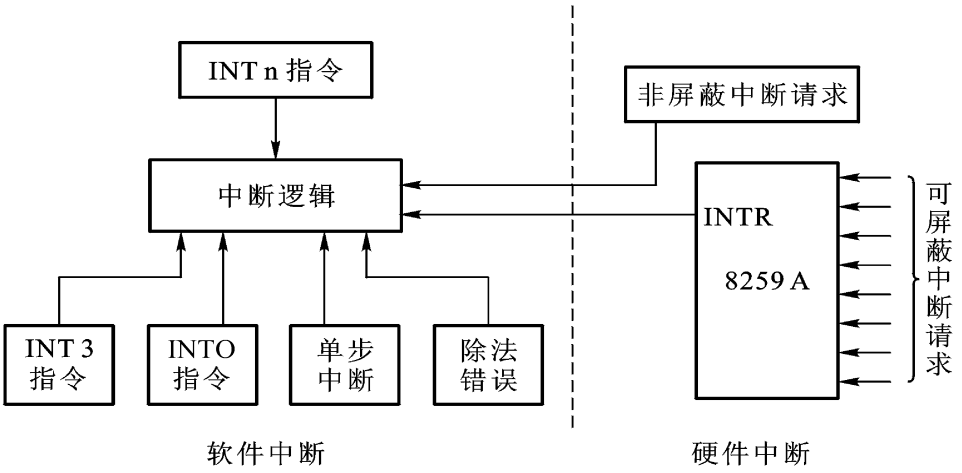


图 8.3 80X86中断源

表 8.1 中断优先级

中断	优先级
除法出错, INT n, INTO	最高
NMI	↑
INTR	↓
单步	最低

8.2.1 外部中断 (硬件中断)

1. 可屏蔽中断 INTR

可屏蔽中断 INTR信号连到 CPU的 INTR引脚,它受 CPU中断允许标志位 IF的控制,即 IF=1时,CPU才能响应 INTR引脚上的中断请求。当可屏蔽中断被响应时,CPU需执行 7个总线周期,即:

⑴ 执行第一个 $\overline{\text{INTA}}$总线周期,通知外部中断系统做好准备;

⑵ 执行第二个 $\overline{\text{INTA}}$总线周期,从外部中断系统获取中断类型号,并乘以 4,形成中断向量地址;

⑶ 执行一个总线写周期,将标志寄存器内容压栈,同时使 IF为 0, TF为 0;

⑷ 执行一个总线写周期,把 CS内容压栈;

⑸ 执行一个总线写周期,把当前 IP内容压栈;

⑹ 执行一个总线读周期,从中断向量表中读取中断服务程序的偏移地址并送 IP;

⑺ 执行一个总线读周期,从中断向量表中读取中断服务程序的段地址并送 CS。

2. 非屏蔽中断

非屏蔽中断 NMI信号连到 CPU的 NMI引脚,它不受 CPU中断允许标志位 IF的控制。一旦发生,立即转至中断类型号为 2的中断处理服务程序。NMI的优先级高于 INTR。当 CPU采样到 NMI有请求时,在内部将其锁存,并自动提供中断类型号 2,然后按以下顺序处理:

⑴ 将中断类型号乘以 4,得到中断向量地址 0008H;

⑵ 将标志寄存器内容压入堆栈保护;

⑶ 清 IF和 TF标志,屏蔽 INTR中断和单步中断;

⑷ 保存断点,即把断点处的 IP和 CS内容压栈;

⑸ 从中断向量表中取中断服务程序的入口地址,分别送至 CS和 IP;

⑹ 转入相应中断服务程序并执行;

⑺ 恢复断点及标志寄存器内容,中断返回。

IBM PC/XT系统中,NMI主要用于解决系统主板上 RAM出现的奇偶错,或 I/O通道中扩展选件板上出现的奇偶校验错等。

8.2.2 内部中断(软件中断)

内部中断是由于 80X86内部执行程序出现异常引起的程序中断,包括除法错中断、溢出中断、INT n指令中断、单步中断和断点中断。内部中断响应后不需要 $\overline{INTA}$总线周期,处理过程与 NMI过程基本相同。

⑴ 除法错中断。在执行除法指令时,若除数为 0或商超过寄存器所能表达的范围,则 CPU立即产生一个 0型中断。

⑵ 溢出中断。如果上一条指令使溢出标志位 OF为 1,则执行 INTO指令产生中断,溢出中断的中断类型号为 4。

⑶ INT n指令中断。在执行中断指令 INT n时产生的一个中断类型号为 n的内部中断。

⑷ 单步中断。当陷阱标志 TF置“1”时,80X86处于单步工作方式。在单步工作时,每执行完一条指令,CPU自动产生中断类型号为 1的中断。

⑸ 断点中断。断点中断是 80X86提供的一种调试程序的手段。用于设置程序中的断点,中断类型号为 3。

8.2.3 中断向量表

中断向量表是存放中断服务程序入口地址的表格。它存放于系统内存的最低端,共 1 KB,每 4个字节存放一个中断服务程序的入口地址,较高地址的 2个字节存放中断服务程序入口的段地址,较低地址的 2个字节存放中断服务程序入口的偏移地址,这 4个单元的最低地址称为中断向量地址,其值为中断类型号乘 4。80X86系统的中断向量表结构如表 8.2所示。

CPU响应中断后,将中断类型号 ×4,在中断向量表中“查表”得到中断服务程序入口地址,分别送 CS和 IP,从而转入中断服务程序。

设置中断向量的方法有两种,一是自编一段程序将中断服务程序的入口地址直接写入中断向量表中的相应单元;二是利用 DOS功能调用完成中断向量的设置。

⑴ 直接写入

```
MOV     DS,0000H
MOV     SI,中断类型号 * 4
MOV     AX,中断服务程序偏移地址
MOV     [SI],AX
MOV     AX,中断服务程序段地址
MOV     [SI+2],AX
```

⑵ 利用 DOS功能调用

设置中断向量 (DOS功能调用 INT 21H)

功能号:AH = 25H

入口参数:AL = 中断类型号,DS:DX = 中断向量(段地址:偏移地址)

获取中断向量 (DOS功能调用 INT 21H)

功能号:AH = 35H

入口参数:AL = 中断类型号

出口参数:ES:BX = 中断向量(段地址:偏移地址)

表 8.2 中断向量表

地址	表项	向量定义
000	IP_0	除法出错
	CS_0	
004	IP_1	单步中断
	CS_1	
008	IP_2	NMI 中断
	CS_2	
00C	IP_3	断点中断
	CS_3	
010	IP_4	溢出中断
	CS_4	
014	IP_5	系统保留
	CS_5	
07C	IP_{31}	
	CS_{31}	
080	IP_{32}	用户定义
	CS_{32}	
0FC	IP_{255}	
	CS_{255}	

8.2.4 80X86中断响应过程

80X86对一个中断请求响应和处理过程如图 8.4所示。当响应中断后，按图 8.4左半部分的顺序查询，并从内部或外部得到反映该中断的中断类型号。尽管中断类型号不同，但 80X86对它们的响应过程一样，如图 8.4右半部分

所示。

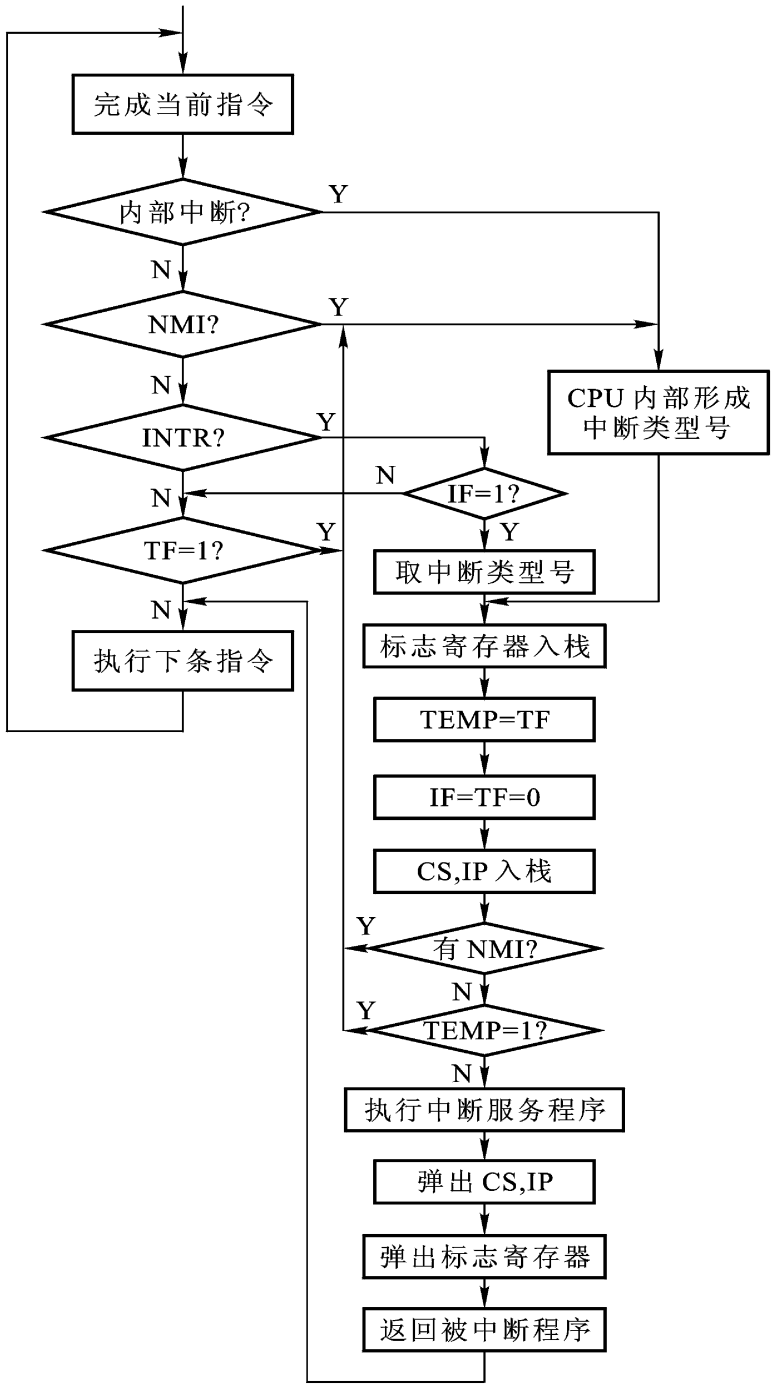


图 8.4　80X86中断处理流程

在获取中断类型号后,执行中断服务程序前,进一步判断是否存在单步中断,目的是在系统单步工作时又产生其他中断的情况下,尽管系统首先识别其他

中断，但在执行该服务程序前，还可识别出单步中断，并首先开始执行单步中断服务程序。当单步中断处理结束后，才返回原先被挂起的其他中断处理程序。另外，还判断是否存在非屏蔽中断的原因，也是为了系统能够及时处理外部紧急事件提出的中断请求。

8.3 中断控制器 8259A

Intel 8259A是一种可编程中断控制器。在PC/XT系统中使用了一片8259A，在PC/AT系统中使用了两片8259A。目前的PC系列，其外围接口芯片（如80C286）都集成有与两片8259A相当的中断控制电路。

8.3.1 8259A的功能

Intel 8259A是一种可编程的、具有强大中断管理功能的大规模集成电路芯片，主要功能有：

⑴ 具有8级优先权控制，通过级联可扩展至64级。

⑵ 每一级均可通过编程实现屏蔽或开放。

⑶ 能向CPU提供相应的中断类型号。

⑷ 可通过编程选择不同的工作方式。

8.3.2 8259A的内部结构和引脚功能

1. 8259A的内部结构

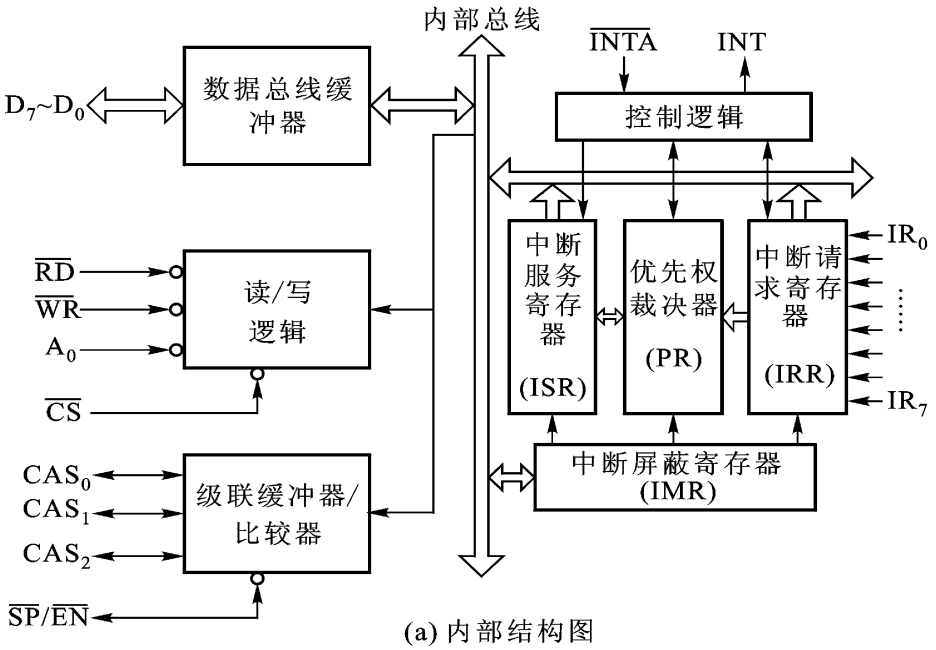
8259A的内部结构如图8.5a所示，主要由8个功能模块组成。

⑴ 中断请求寄存器（IRR） IRR是一个具有锁存功能的8位寄存器，用于寄存外部设备提出的中断请求。IR_0～IR_7可连接8个外设的中断请求信号，当IR_0～IR_7中任何一个变为高电平时，IRR中的相应位置“1”。

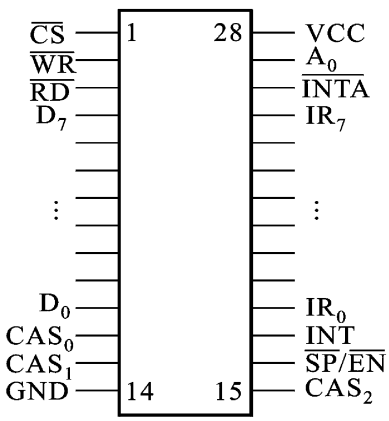
⑵ 中断服务寄存器（ISR） ISR是一个8位寄存器，用于寄存所有正在被服务的中断请求。8259A在接收到第一个$\overline{INTA}$信号后，使当前被响应的中断请求所对应的ISR置“1”，而相应的IRR复位。在中断嵌套时，ISR中有多个位为“1”。

⑶ 中断屏蔽寄存器（IMR） IMR是一个8位寄存器，用于寄存要屏蔽的中断。某位为“1”，表示屏蔽相应中断请求，为“0”，表示开放相应中断请求。

⑷ 优先权裁决器（PR） 用于识别和管理IRR中各位的优先权级别。各



(a) 内部结构图



(b) 引脚图

图 8.5 8259A内部结构图及引脚图

个信号的优先权级别可通过编程定义和修改。当有中断请求使 IRR中某些位置“1”时,优先权裁决器选出其中级别最高的中断级,当中断允许嵌套时,选出的中断级若优先权高于正在服务的中断,则发中断请求信号 INT,并中止当前的中断处理,执行高一级的中断处理;若优先权低于正在服务的中断,则不发中断请求信号 INT。

⑸ 数据总线缓冲器 用于连接系统的数据总线,是一个 8位双向三态缓冲器,传输写入 8259A的控制字、读取的 8259A的状态信息,以及 CPU读取的中

断类型号。

⑹ 读/写逻辑 用于接收端口地址信息和 CPU 的读写控制信号 $\overline{IOR}$和 $\overline{IOW}$,产生相应的控制信号,控制命令字的写入和状态字的读取。

⑺ 控制逻辑 根据编程设定的工作方式管理 8259A,负责向 CPU 发中断请求信号 INT和接收来自 CPU 的中断响应信号 $\overline{INTA}$,并将 $\overline{INTA}$信号转换成内部所需的各种控制信号。

⑻ 级联缓冲/比较器 用于控制多片 8259A 的级联,使得系统的中断级可以扩展。最多可用 9片实施级联,一片为主片,其余为从片。

2. 8259A 的引脚及功能

8259A为 28脚双列直插式封装,其引脚信号如图 8.5b所示。

$D_7 \sim D_0$:双向三态数据线,在系统中与数据总线相连。

$IR_7 \sim IR_0$:中断请求输入信号。

$\overline{RD}$:读控制信号,输入,与系统控制总线相连。

$\overline{WR}$:写控制信号,输入,与系统控制总线相连。

$\overline{CS}$:片选信号,输入,与地址译码电路相连。

A_0:地址线,输入,在使用中 8259A 占用相邻两个端口地址,A_0 与 $\overline{CS}$配合,A_0 =1选中奇地址端口,A_0 =0选中偶地址端口。在 80X86的 PC系列机中,主片 8259A 的端口地址为 20H和 21H。

$CAS_2 \sim CAS_0$:级联信号线,对主片 8259A,它为输出;对从片 8259A,它为输入。主、从片 8259A的 $CAS_2 \sim CAS_0$ 对应相连,主片 8259A在第一个 $\overline{INTA}$响应周期内通过 $CAS_2 \sim CAS_0$ 送出识别码,而和此识别码相符的从片 8259A在接收到第二个 $\overline{INTA}$信号后,将中断类型码发送到数据总线上。

$\overline{SP}/\overline{EN}$:从编程/缓冲器允许信号,双向。$\overline{SP}/\overline{EN}$作为输入还是输出,取决于 8259A是否采用缓冲方式,若采用缓冲方式,$\overline{SP}/\overline{EN}$作为输出,反之,作为输入。作为输入的 $\overline{SP}$使用时,用于区分主、从片 8259A。主片 8259A的 $\overline{SP}$=1,从片 8259A的 $\overline{SP}$=0。作为输出的 $\overline{EN}$使用时,作为数据总线缓冲器的使能信号。

INT:中断请求信号,输出。与 CPU 的 INTR引脚连接。

$\overline{INTA}$:中断响应信号,输入。与 CPU 的 $\overline{INTA}$引脚连接。

8.3.3 8259A的工作方式

8259A有多种工作方式,可以通过编程来设定。用户可根据系统工作的要求来选择相应的工作方式,然后通过对 8259A写入初始化命令字来确定其工作

方式。

1. 中断嵌套方式

⑴ 全嵌套方式

全嵌套方式是 8259A 最常用的一种工作方式,中断优先级别固定,IR_0 最高,IR_7 最低。当 IR_i 中断请求响应时,相应的 ISR_i 位置 1,在中断服务过程中禁止同级和优先级低于本级的中断请求。

⑵ 特殊全嵌套方式

特殊全嵌套方式与全嵌套方式基本相同,只是在特殊全嵌套方式下,可响应同级的中断请求。

特殊全嵌套方式一般用于 8259A 的级联情况。此时,从片的 INT连接到主片的 IR_i 上,每个从片的 IR_0 ~IR_7 有不同的优先级别,但从主片看来,每个从片作为同一优先级。如果采用全嵌套方式,则从片中某一较低级的中断请求经主片得到响应后,主片会把该从片的所有其他中断请求作为同一级而屏蔽掉,包括优先级较高的中断请求,因此无法实现从片上各级中断的嵌套。所以,系统中只有一片 8259A时,通常采用全嵌套方式,而系统中有多片 8259A 时,主片则必须采用特殊全嵌套方式,而从片可采用全嵌套方式。

2. 循环优先方式

⑴ 优先级自动循环方式

初始时,优先次序为 $IR_0 \rightarrow IR_7$,IR_0 最高,IR_7 最低。当某级中断响应后,则优先级降为最低。而其后的与之相邻的优先级升为最高。如 IR_3 响应后的优先级次序变为 IR_4,IR_5,IR_6,IR_7,IR_0,IR_1,IR_2,IR_3。

⑵ 优先权特殊循环方式

优先权特殊循环方式与优先权循环方式相比仅有一点不同,就是在优先权特殊循环方式下,一开始的最低优先权是由编程确定的。如编程时确定 IR_5 为最低优先权,则 IR_6 优先权最高。

3. 中断屏蔽方式

⑴ 普通屏蔽方式

这种屏蔽方式是通过编程将中断屏蔽字写入 IMR而实现的。若写入某位为 1,则对应的中断请求被屏蔽;为 0,则对应的中断请求被开放。

⑵ 特殊屏蔽方式

此方式用于这样一种特殊要求的场合,即在执行较高级的中断服务时,希望开放较低级的中断请求。采用普通屏蔽方式是不能实现这一要求的,因为用普通方式时,即使把较低级的中断请求开放,但由于 ISR中当前正在服务的较高中断级的对应位仍为“1”,它会禁止所有优先级比它低的中断请求。采用特殊屏

蔽方式，可在中断服务程序中用中断屏蔽命令字来屏蔽当前正在服务的中断级别（即设置 IMR 的相应位为“1”），同时使 ISR 中对应位清“0”，这样就不但屏蔽了当前正在服务的中断级，同时真正开放了其他优先级较低的中断请求。

4. 结束中断处理方式

当某个中断服务完成时，必须给 8259A 一个中断结束命令，使 ISR 的相应位清“0”，从而结束中断。8259A 有两种不同的结束中断处理方式。

⑴ 自动中断结束方式（AEOI）

此种方式只能用于单片 8259A 的系统中，8259A 在第二个 $\overline{\text{INTA}}$ 信号的上升沿，自动清除 ISR 的相应位。

⑵ 非自动中断结束方式（EOI）

在这种工作方式下，中断服务程序返回前，必须向 8259A 发送中断结束命令，清除 ISR 的相应位，表示该中断处理的结束。

非自动中断结束方式又分一般和特殊结束中断处理方式两种。一般结束中断处理方式只要在程序中往 8259A 的偶地址端口输出一个操作命令字 OCW_2，并使得 OCW_2 中的 EOI＝1，SL＝0，R＝0即可。在特殊全嵌套方式下，因无法确定哪一级中断为最后响应和处理的，因此特殊结束中断处理方式也是在程序中往 8259A 的偶地址端口输出一个操作命令字 OCW_2，并使得 OCW_2 中的 EOI＝1，SL＝1，R＝0，且 L_2 L_1 L_0 指明清 ISR 中的哪一位。

5. 程序查询方式

在程序查询方式下，8259A 不向 CPU 发 INT 信号，而是靠 CPU 不断查询 8259A。当查询到有中断请求时，转入相应的中断处理程序。设置查询方式的过程为：写入查询方式命令字，然后读取 8259A 的查询字（IRR 寄存器），其格式为：

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
I					W_2	W_1	W_0

I＝1表示有中断请求，$W_2W_1W_0$ 表示 8259A 请求服务的最高优先级编码。

D_6 ～D_3 无实际意义。

6. 中断请求触发方式

⑴ 边沿触发方式

在边沿触发方式下，8259A 将中断请求输入端出现的上升沿作为中断请求信号。

⑵ 电平触发方式

在电平触发方式下，8259A 将中断请求输入端出现的高电平作为中断请求信号。在中断请求得到响应后必须及时撤除高电平，如果在 CPU 进入中断处理过程并且开放中断前未去掉高电平信号，则可能引起不应有的第二次中断。

8.3.4 8259A的编程

8259A的编程包括初始化编程和工作方式编程两部分，其中初始化命令字 4个（ICW$_1$～ICW$_4$），操作命令字 3个（OCW$_1$～OCW$_3$）。

1. 8259A 的初始化命令字

(1) 初始化命令字 ICW$_1$

ICW$_1$ 的主要功能是设置 8259A 中断请求 IR$_i$ 的触发方式，是单片 8259A 还是多片 8259A。当写入 ICW$_1$ 后，自动清除中断屏蔽寄存器 IMR，并默认为全嵌套方式。ICW$_1$ 写入 8259A 的偶地址端口。其格式如下：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0				1	LTIM	ADI	SNGL	ICW$_4$

D_4 为 ICW$_1$ 的特征标志位。

D_3 (LTIM) 表示中断请求信号起作用的触发方式。D_3 ＝1为电平触发，D_3 ＝0为边沿触发。

D_1 (SNGL) 表示系统是使用单片 8259A 还是多片 8259A。D_1 ＝1为单片，D_1 ＝0为多片。

D_0 (ICW$_4$) 表示是否需要 ICW$_4$。D_0 ＝1为需要，D_0 ＝0为不需要。

D_2 (ADI) 在 8080/8085CPU模式下用，80X86 CPU模式下不用。

(2) 初始化命令字 ICW$_2$

ICW$_2$ 的功能是设定 8259A 的中断类型号。ICW$_2$ 写入 8259A 的奇地址端口。其格式如下：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	T_7	T_6	T_5	T_4	T_3			

D_7～D_3为中断类型号的高 5位,由用户给出。低三位由 8259A按 IR_0～IR_7三位编码值自动填入。三位编码定义如下：

中断源	编码 $D_2D_1D_0$
IR_0	0 0 0
IR_1	0 0 1
IR_2	0 1 0
IR_3	0 1 1
IR_4	1 0 0
IR_5	1 0 1
IR_6	1 1 0
IR_7	1 1 1

(3) 初始化命令字 ICW_3

ICW_3仅用于 8259A的级联方式,其功能是用来表明主片 8259A的 IR_i与从片 8259A的 INT之间连接关系。ICW_3写入 8259A的奇地址端口。

8259A作为主片的格式：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

D_7～D_0 (S_7～S_0)表示相应的 IR_7～IR_0中断请求线上有无从片。D_i=1表示 IR_i接有从片。

8259A作为从片的格式：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	ID_2	ID_1	ID_0

D_2～D_0为从片的识别码,表示从片的 INT输出是和主片 8259A中的哪一个 IR_i相连接。

(4) 初始化命令字 ICW_4

ICW_4的功能是用来设定 80X86系统中的 8259A在级联方式下的优先权管理方式、主/从状态以及中断结束方式等。ICW_4写入 8259A的奇地址端口。其

格式如下：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	SFNM	BUF	M／S	AEOI	μPM

D_0（μPM）：定义 8259A工作的系统，D_0 ＝1为 80X86系统，D_0 ＝0为 8080／8085系统。

D_1（AEOI）：表示是否采用自动结束中断方式，D_1 ＝1为自动中断结束方式，D_1 ＝0为非自动中断结束方式。

D_2（M／S）：表示本片 8259A是主片还是从片，D_2 ＝1为主片，D_2 ＝0为从片。

D_3（BUF）：表示本片 8259A和系统数据总线之间是否有缓冲器，D_3 ＝1表示有缓冲器，因此必须产生控制信号，使缓冲器启动，D_3 ＝0表示没有缓冲器。

D_4（SFNM）：用于设定级联方式下的优先权管理方式，D_4 ＝1为特殊全嵌套方式，D_4 ＝0为全嵌套方式。

写入初始化命令字的流程如图 8.6所示。

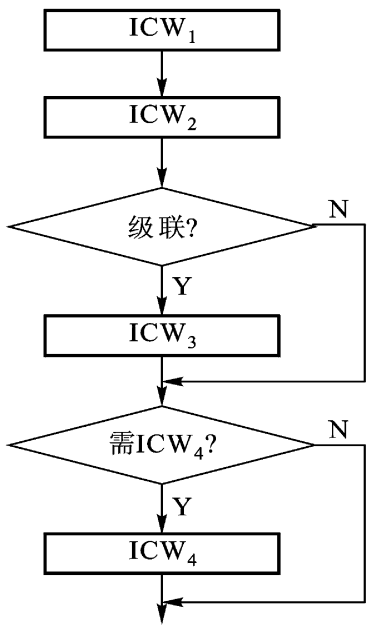


图 8.6　8259A初始化流程

2. 8259A的操作命令字

(1) 操作命令字 OCW_1

操作命令字 OCW_1，也称屏蔽操作命令字，写入 8259A的奇地址端口。其格式如下：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0

M_i =1表示 IR_i 上的中断请求被屏蔽，M_i =0表示 IR_i 上的中断请求被允许。

(2) 操作命令字 OCW_2

操作命令字 OCW_2，也称中断方式命令字，用来设置优先级是否循环、循环的方式及中断结束的方式。OCW_2 写入 8259A 的偶地址端口，其格式如下：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	R	SL	EOI	0	0	L_2	L_1	L_0

D_7 (R)：中断排队是否循环的标志。R =1为优先级循环方式，R =0为固定优先级方式。

D_6 (SL)：选择 $L_2 L_1 L_0$ 编码是否有效的标志。若 SL =1，则 $L_2 L_1 L_0$ 编码有效，若 SL =0，则无效。

D_5 (EOI)：中断结束命令。D_5 =1时，则使现行的 ISR 中最高优先级的相应位复位（一般中断结束方式），或由 $L_2 L_1 L_0$ 指定的 ISR 相应位复位（特殊中断结束方式）。

$D_2 D_1 D_0$ ($L_2 L_1 L_0$)：对应 8个二进制编码，有两个作用，一是用在特殊 EOI命令中，表示清除的是 ISR 的哪一位；二是用在特殊循环方式中，表示系统中最低优先级编码。

(3) 操作命令字 OCW_3

操作命令字 OCW_3，也称状态操作命令字，其功能是用来设置特殊屏蔽方式和查询方式，以及用来读取 8259A 的中断请求寄存器 IRR 和中断服务寄存器 ISR 的当前状态。OCW_3 写入 8259A 的偶地址端口，其格式如下：

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	ESMM	SMM	0	1	P	RR	RIS

$D_6 D_5$ 两位决定 8259A 是否工作于特殊屏蔽方式。$D_6 D_5$ 为 11时，8259A 为特殊屏蔽方式；$D_6 D_5$ 为 10时，8259A 为一般屏蔽方式。

D_1D_0 两位规定随后读取的寄存器。D_1D_0 为 11时,表示要读 ISR;D_1D_0 为 10时,表示要读 IRR。

D_2 决定 8259A是否处于程序查询方式,D_2 =1时,8259A处于程序查询方式。当 8259A发出查询命令后,随后从偶地址读出的数据即为中断请求状态字,其格式见 8.3.3中的程序查询方式。

3. 8259A编程举例

【例题 8.1】　8259A在 IBM PC/XT中的应用。

在 IBM PC/XT中只有一片 8259A,可连接 8个外部中断源,其连接方法、中断源名称及中断类型码如图 8.7所示。

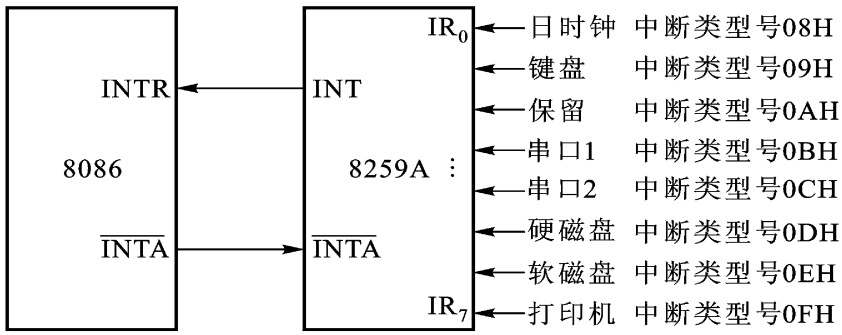


图 8.7　8259A在 IBM PC/XT中连接图

系统分配给 8259A的 I/O端口地址为 20H和 21H,采用边沿触发方式、缓冲方式、非自动中断结束方式以及完全嵌套方式。对 IBM PC/XT微机 8259A的初始化程序段如下:

```
MOV     AL, 00010011B       ;设置 ICW1,边沿触发,
                            单片 8259A,需 ICW4
OUT     20H, AL
MOV     AL, 00001000B       ;设置 ICW2,中断类型号的高 5位为 00001
OUT     21H, AL
MOV     AL, 00001101B       ;设置 ICW4,非自动中断结束方式,完全嵌
                             套方式,缓冲方式
OUT     21H, AL
```

【例题 8.2】　读中断请求寄存器 IRR内容。设 8259A偶地址端口为 20H,奇地址端口为 21H。

若要对 IRR或 ISR读出时,则必须先写一个 OCW_3 命令字,以便 8259A处于被读状态,然后再从偶地址端口读出 IRR或 ISR中的内容。程序段如下:

```
MOV     AL,0AH
OUT     20H,AL          ;设置 OCW3
NOP
IN      AL,20H          ;读 IRR内容
```

【例题 8.3】 试编程实现主机每次响应 8259A的 IR_2 中断请求,显示字符串“This is a 8259A interrupt!”,中断 10次结束。8259A偶地址端口为 20H,奇地址端口为 21H,IR_2 的中断类型号为 0AH。

程序流程如图 8.8所示。

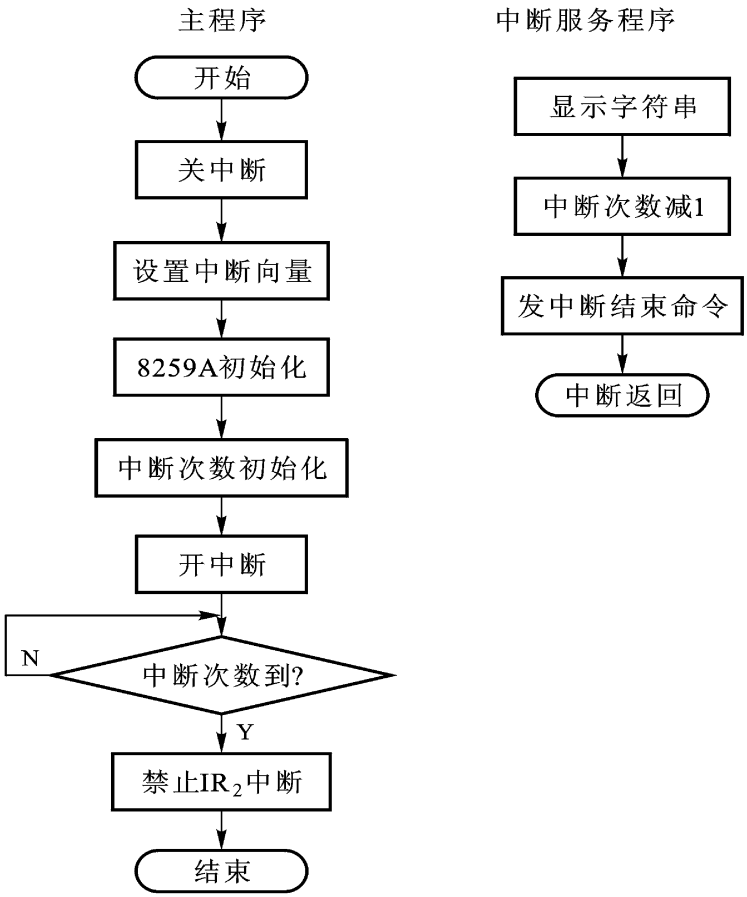


图 8.8 例题 8.3程序流程图

```
DATA SEGMENT
    MESS DB 'This is a 8259A interrupt!',0Ah,0Dh,'$'
    DATA ENDS
    CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
```

```
START: OV      AX, DATA
       MOV     DS, AX
       CLI                          ;关中断
       PUSH DS
       MOV     AX, SEG DISPLAY      ;取中断服务程序入口段地址
       MOV     DS, AX
       MOV     DX, OFFSET DISPLAY   ;取中断服务程序入口偏移地址
       MOV     AX, 250AH            ;设置中断向量
       INT     21H
       POP     DS
       MOV     AL, 13H              ;设置 ICW1,边沿触发,单片 8259A,需
                                     ICW4
       OUT     20H, AL
       MOV     AL, 08H              ;设置 ICW2,中断类型号的高 5位为
                                     00001
       OUT     21H, AL
       MOV     AL, 05H              ;设置 ICW4,非 EOI方式,完全嵌套方式
       OUT     21H, AL
       IN      AL, 21H              ;读取 IMR
       AND     AL, 0FBH             ;开放 IR2
       OUT     21H, AL
       MOV     BL, 10               ;初始化中断次数
       STI                          ;开中断
WAIT1: CMP     BL, 0
       JNZ     WAIT1
       CLI
       IN      AL, 21H
       OR      AL, 04H              ;禁止 IR2 中断
       OUT     21H, AL
       STI
       MOV     AH, 4CH              ;返回 DOS
       INT     21H
DISPLAY PROC   NEAR
       LEA     DX, MESS             ;显示字符串
```

```
        MOV     AH,09H
        INT     21H
        DEC     BL                      ;中断次数减 1
        MOV     AL,20H                  ;发送中断结束命令
        OUT     20H,AL
        IRET
DISPLAY ENDP
CODE    ENDS
        END     START
```

8.3.5 8259A的级联

在一个中断系统中,可以使用多片 8259A,采用级联方法,使中断优先级从8级可扩展到 64级。在级联时,只能有一片 8259A作为主片,其余的 8259A均作为从片。

主 8259A的三条级联线 CAS_0 ～CAS_2 作为输出线,通过驱动器连接到每个从片的 CAS_0 ～CAS_2 的输入端。如只有一个从片,也可不加驱动器。

图 8.9为 80X86微机系统中,使用 2片 8259A构成的级联中断系统。系统分配给主片 8259A的端口地址为 20H和 21H,从片 8259A的端口地址为 A0H和 A1H,系统加电后,BIOS对它们的初始化程序如下:

```
;主片 8259A
        MOV     AL,11H                  ;设置 ICW1,边沿触发,需 ICW4
        OUT     20H,AL
        MOV     AL,08H                  ;设置 ICW2,中断类型号的高 5位为
                                         00001
        OUT     21H,AL
        MOV     AL,04H                  ;设置 ICW3,从片连到主片的 IR2上
        OUT     21H,AL
        MOV     AL,15H                  ;设置 ICW4,非缓冲,非 EOI,特殊全嵌
                                         套方式
        OUT     21H,AL
;从片 8259A
        MOV     AL,11H                  ;设置 ICW1,边沿触发,需 ICW4
        OUT     0A0H,AL
```

```
MOV   AL, 70H          ;设置 ICW2,中断类型号的高 5位为 01110
OUT   0A1H, AL
MOV   AL, 02H          ;设置 ICW3,设定从片级联于主片的 IR2
OUT   0A1H, AL
MOV   AL, 01H          ;设置 ICW4,非缓冲,非 EOI,全嵌套方式
OUT   0A1H, AL
```

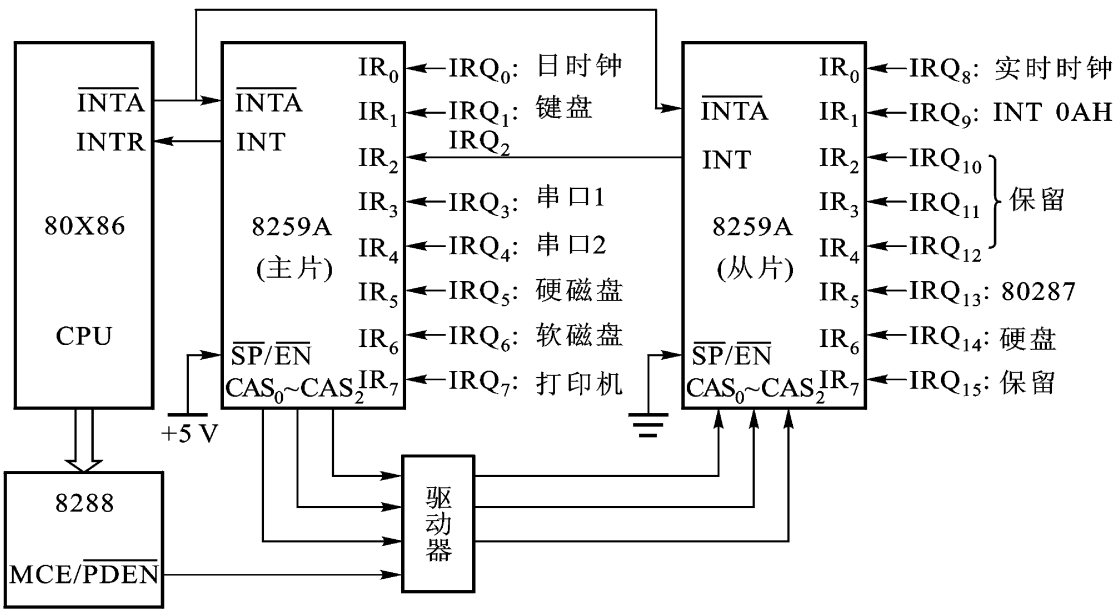


图 8.9 8259A在 IBM PC/XT中的应用

思考题与习题

8.1 80X86的中断系统有哪几种类型中断？其优先次序如何？

8.2 已知中断向量表中地址 0020H～0023H的单元中依次是 40H, 00H, 00H, 01H,并知 INT 08H指令本身所在的地址为 9000H:00A0H。若 SP=0100H, SS=0300H,标志寄存器内容为 0240H,试指出在执行 INT 08H指令,刚进入它的中断服务程序时, SP, SS, IP, CS和堆栈顶上三个字的内容 (用图表示)。

8.3 某一用户中断源的中断类型号为 40H,其中断服务程序名为 INTR40,请用两种不同方法设置它的中断向量。

8.4 试编写一段将 8259A中 IRR, ISR, IMR的内容读出,存入到 BUFFER开始的数据缓冲区去的程序。8259A端口地址为 30H, 31H。

8.5 某一 8086 CPU 系统中，采用一片 8259A 进行中断管理。设 8259A 工作在全嵌套方式，发送 EOI 命令结束中断，边沿触发方式，IR_0 对应的中断向量号为 90H。8259A 在系统中的端口地址为 FFDCH ($A_0=0$) 和 FFDDH ($A_0=1$)。试编写 8259A 的初始化程序段。

8.6 中断服务程序的入口处为什么通常要使用开中断指令？

第9章

可编程接口芯片及应用

本章主要介绍可编程接口芯片的基本概念，以及可编程计数器/定时器接口芯片 8253、可编程并行接口芯片 8255A 的组成、功能和应用。

9.1 可编程接口芯片概述

CPU要与外设交换信息，必须通过接口电路，在接口电路中一般具有如下电路单元：

⑴ 输入/输出数据缓冲器和锁存器，以实现数据的 I/O。

⑵ 控制命令和状态寄存器，用以存放对外设的控制命令，以及外设的状态信息。

⑶ 地址译码器，用来选择接口电路中的不同端口（寄存器）。

⑷ 读写控制逻辑。

⑸ 中断控制逻辑。

1. 片选

微机系统中，所有 I/O接口及内存储器都挂接在系统总线上，要访问某一接口芯片中的某个端口，必须要有地址信号选中该接口芯片才能使该接口芯片进入工作状态，访问芯片中的某个端口。CPU的地址信号经地址译码器后接到接口芯片的片选端 CE（Chip Enable）或 CS（Chip Select，CS 或 CE）究竟是高电平有效还是低电平有效视接口芯片而定。

2. 读/写操作

接口芯片的地址码经译码后接通芯片的片选端,对读操作而言,使输入口信息由数据总线进入 CPU,数据何时读入 CPU,由读信号控制。对于输出口,当 CPU对接口进行输出数据的操作时,发出写信号。在 PC系统中,对 I/O接口的操作由 IN、OUT指令完成。

3. 可编程

目前所用的接口芯片大部分是多通道、多功能的。所谓多通道就是指一个接口芯片同时可接几个外设;所谓多功能是指一个接口芯片能实现多种功能,实现不同的电路工作状态。而这些通道和电路工作状态的选择可由计算机通过指令来设定。

4. "联络"

CPU通过外设接口芯片同外设交换信息时,接口芯片常常需要和外设间有一定的"联络",以保证信息的正常传输。

9.2 可编程计数器/定时器 8253

Intel 8253是具有三个独立的 16位计数器,使用单一 +5 V电源,采用 NMOS工艺,24脚双排直插式封装的大规模集成电路。

9.2.1 8253功能及结构

1. 8253主要功能

⑴ 每片有三个独立的 16位计数通道。

⑵ 每个计数器可按二进制或十进制来计数。

⑶ 每个计数器最高计数速率可达 2.6 MHz。

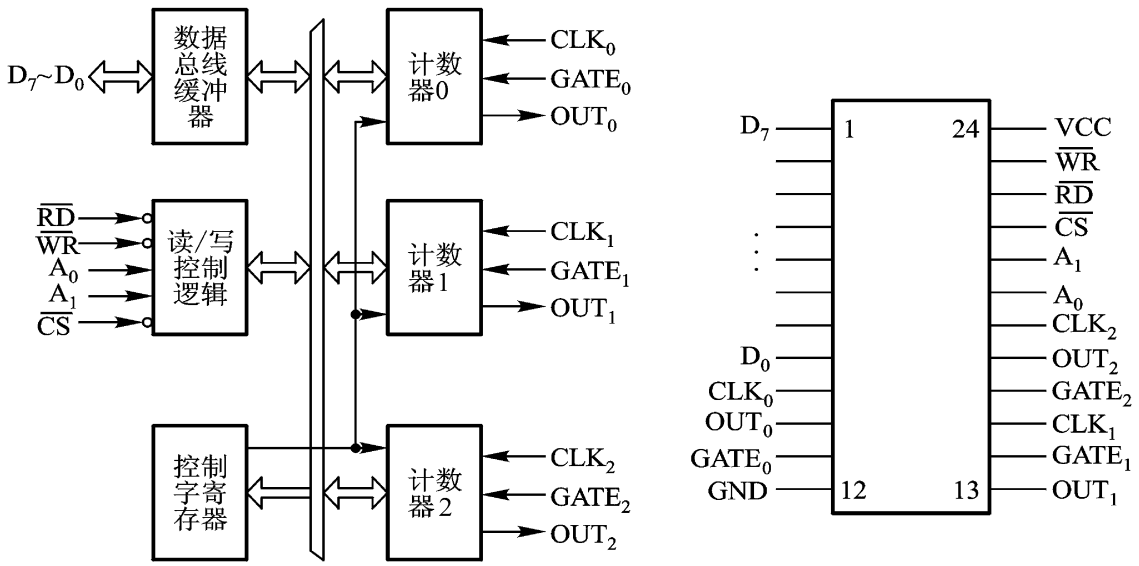
⑷ 每个计数器可编程设定 6种工作方式之一。

⑸ 所有输入、输出均与 TTL电平兼容,便于与外围接口电路相连。

2. 8253的内部结构和引脚特性

8253的内部结构如图 9.1a所示。与内部总线相连的分为 4部分:数据总线缓冲器、读写控制逻辑、控制字寄存器以及三个独立的 16位的计数器通道。这三个计数器分别是计数器 0、计数器 1和计数器 2。

⑴ 数据总线缓冲器 数据总线缓冲器是 8位的双向三态缓冲器,主要用于 8253与 CPU之间进行数据传送。该数据包括三个方面:一是向 8253写入控制字,二是向计数器设置计数初值,三是从计数器读取计数值。



(a) 内部结构图　　(b) 引脚图

图 9.1　8253内部结构及引脚图

⑵ 读写控制逻辑　读写控制逻辑电路接受输入到 8253的 $\overline{RD}$, $\overline{WR}$, $\overline{CS}$, A_1, A_0 信号,经过逻辑控制电路的组合产生相应操作,具体操作如表 9.1所示。

表 9.1　8253控制信号与执行的操作

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	执行的操作
0	1	0	0	0	对计数器 0设置初值
0	1	0	0	1	对计数器 1设置初值
0	1	0	1	0	对计数器 2设置初值
0	1	0	1	1	写控制字
0	0	1	0	0	读计数器 0当前计数值
0	0	1	0	1	读计数器 1当前计数值
0	0	1	1	0	读计数器 2当前计数值

⑶ 控制字寄存器　接收 CPU发来的对 8253的初始化控制字。对控制字寄存器只能写入,不能读出。

⑷ 三个计数器　每个计数器内部都包含一个计数初值寄存器,一个减 1计数寄存器,一个当前计数输出寄存器和一个控制寄存器。当前计数输出寄存器跟随减 1计数寄存器内容而变化,当有一个锁存命令出现后,当前计数输出寄存器锁定当前计数,直到被 CPU读走之后,又随减 1计数寄存器的变化而变化。

8253引脚如图 9.1b所示,每个引脚的功能定义如表 9.2所示。

表 9.2 8253引脚信号及功能定义

信号	信号方向	功能定义
$D_7 \sim D_0$	双向	8位三态数据线
$CLK_0 \sim CLK_2$	输入	计数器 0,1,2的时钟输入
$GATE_0 \sim GATE_2$	输入	计数器 0,1,2的门控输入
$OUT_0 \sim OUT_2$	输出	计数器 0,1,2的输出
$\overline{CS}$	输入	片选信号,低电平有效。CPU通过该信号有效选中8253,对其进行读写操作
$\overline{RD}$	输入	读信号,低电平有效。有效时表示正读取某个计数器的当前计数值
$\overline{WR}$	输入	写信号,低电平有效。有效时表示正对某个计数器写入计数初值或写入控制字
A_1, A_0	输入	8253端口选择线,可对三个计数器和控制寄存器寻址

9.2.2 8253控制字

1. 8253控制字格式

为让 8253计数器工作,必须先设置控制寄存器的控制字,用来选择计数器,设置工作方式,计数方法以及 CPU访问计数器的读写方法等。8253控制字 (8位)的格式如图 9.2所示:

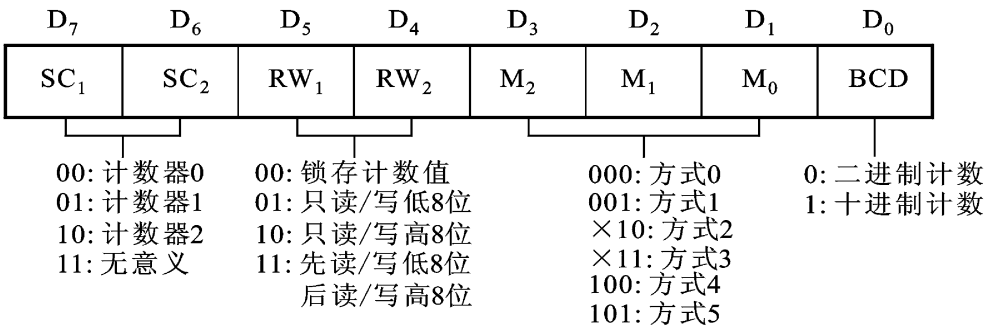


图 9.2 8253控制字格式

其中:D_7 D_6 用于选择定时器;D_5 D_4 用于确定时间常数的读/写格式;D_3 D_2 D_1 用来设定计数器的工作方式;D_0 用来设定计数方式。

2. 8253初始化编程原则

8253的控制寄存器和三个计数器分别具有独立的编程地址,由控制字的内容确定使用的是哪个寄存器以及执行什么操作。因此 8253在初始化编程时并没有严格的顺序规定,但在编程时,必须遵守两条原则:①在对某个计数器设置初值之前,必须先写入控制字。②在设计初始值时,要符合控制字中规定的格式,即只写低位字节,还是只写高位字节,或高、低位字节都写(分两次写,先低字节后高字节)。

8253编程命令有两类:一类是写入命令,包括设置控制命令字、设置计数器的初始值命令和锁存命令。另一类是读出命令,用来读取计数器的当前值。锁存命令是配合读出命令使用的,在读计数值前,必须先用锁存命令锁定当前计数输出寄存器的当前计数。否则,在读数时,减 1计数寄存器的值处在动态变化过程中,当前计数输出寄存器随之变化,就会得到一个不确定的结果。当 CPU将此锁定值读走之后,锁存功能自动失锁,于是当前计数输出寄存器的内容又跟随减 1计数寄存器而变化。在锁存和读出计数值的过程中,减 1计数寄存器仍在作正常减 1计数,这样,保证了计数器在运行中被读取而不影响计数的进行。

9.2.3 8253工作方式与工作时序

8253共有 6种工作方式,对它们的操作都遵守以下三条基本原则:

⑴ 当控制字写入 8253时,所有的控制逻辑电路自动复位,这是输出端 OUT进入初始态。

⑵ 当初始值写入计数器后,要经过一个时钟周期,减法计数器才开始工作,时钟脉冲的下降沿使计数器进行减 1操作。计数器的最大初始值是 0,用二进制计数时 0相当于 2^{16},用十进制计数时 0相当于 10^4。

⑶ 一般情况下,在时钟脉冲 CLK的上升沿采样门控信号。门控信号的触发方式有边沿触发和电平触发两种。

门控信号为电平触发的有:方式 0,方式 4。

门控信号为上升沿触发的有:方式 1,方式 5。

门控信号可为电平触发也可为上升沿触发的有:方式 2,方式 3。

1. 方式 0(计数结束产生中断)

采用这种工作方式,8253可完成计数功能,且计数器只计一遍。当控制字写入后,输出端 OUT为低电平,当计数初值写入后,在下一个 CLK脉冲的下降沿将计数初值寄存器内容装入减 1计数寄存器,然后计数器开始计数,在计数期间,当计数器减为 0之前,输出端 OUT维持低电平。当计数值减到 0时,OUT输

出端变为高电平,可作为中断请求信号,并保持到重新写入新的控制字或新的计数值为止。

在计数过程中,若 GATE信号变为低电平,则在低电平期间暂停计数,减 1计数寄存器值保持不变。

在计数过程中,若重新写入新的计数初值,则在下一个 CLK脉冲的下降沿,减 1计数寄存器以新的计数初值重新开始计数过程。

8253方式 0下三种情况的时序波形图如图 9.3所示:

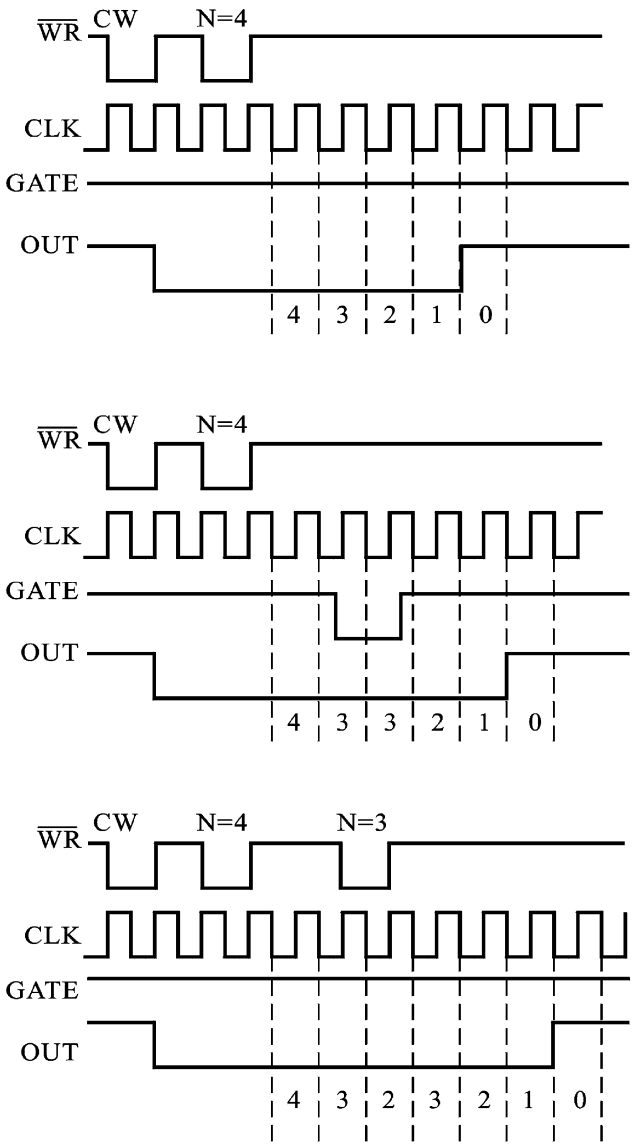


图 9.3 8253方式 0时序波形图

2. 方式 1(可重触发单稳态方式)

采用这种工作方式可输出单个负脉冲信号,脉冲的宽度可通过编程来设定。

当写入控制字后,输出端 OUT变为高电平,并保持高电平状态。然后写入计数初值,只有在 GATE信号的上升沿之后的下一个 CLK脉冲的下降沿,才将计数初值寄存器内容装入减 1计数寄存器,同时 OUT端变为低电平,然后计数器开始减 1计数,当计数值减到 0时,OUT端变为高电平。

如果在 OUT端输出低电平期间,又来一个门控信号上升沿触发,则在下一个 CLK脉冲的下降沿,重新将计数初值寄存器内容装入减 1计数寄存器,并开始计数,OUT端保持低电平,直至计数值减到 0时,OUT端变为高电平。

在计数期间 CPU又送来新的计数初值,不影响当前计数过程。计数器计数到 0,OUT端输出高电平。一直等到下一次 GATE信号的触发,才会将新的计数初值装入,并以新的计数初值开始计数过程。

8253方式 1下三种情况的时序波形图如图 9.4所示：

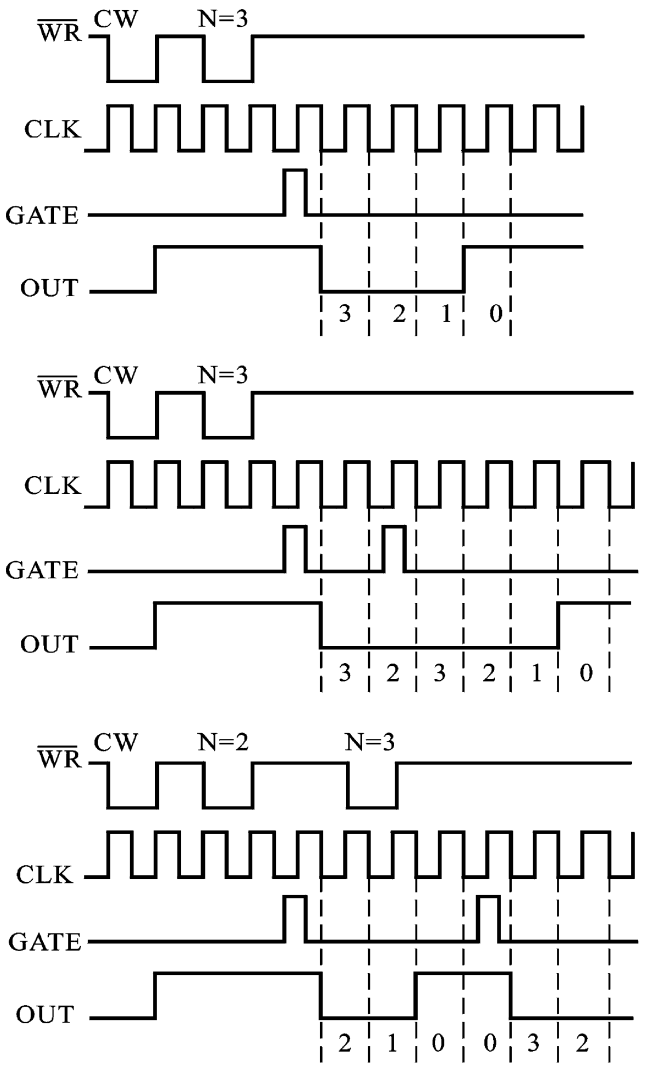


图 9.4　8253方式 1时序波形图

3. 方式 2 (频率发生器)

采用方式 2,可产生连续的负脉冲信号,负脉冲宽度为一个时钟周期。写入控制字后,OUT端变为高电平,若 GATE为高电平,当写入计数初值后,在下一个 CLK的下降沿将计数初值寄存器内容装入减 1计数寄存器,并开始减 1计数,当减 1计数寄存器的值为 1时,OUT端输出低电平,经过一个 CLK时钟周期,OUT端输出高电平,并开始一个新的计数过程。

在减 1计数寄存器未减到 1时,GATE信号由高变低,则停止计数。但当 GATE由低变高时,则重新将计数初值寄存器内容装入减 1计数寄存器,并重新开始计数。

GATE信号保持高电平,但在计数过程中重新写入计数初值,则当正在计数的一轮结束并输出一个 CLK周期的负脉冲后,将以新的初值进行计数。

8253方式 2下三种情况的时序波形图如图 9.5所示:

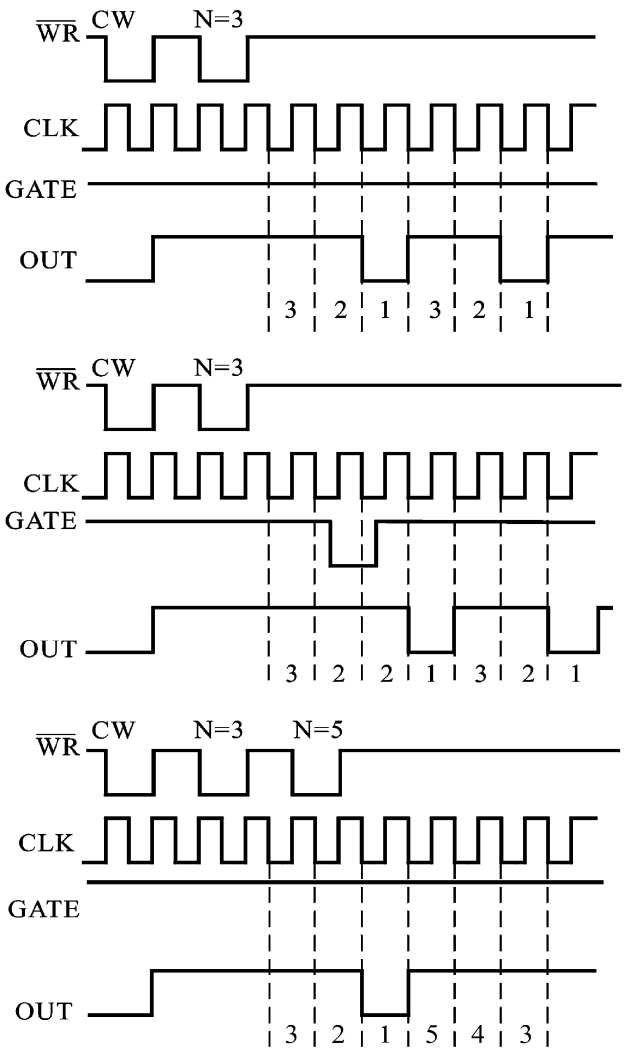


图 9.5 8253方式 2时序波形图

4. 方式 3 方波发生器）

采用方式 3,OUT端输出方波信号。当控制字写入后,OUT输出高电平,当写入计数初值后,在下一个 CLK的下降沿将计数初值寄存器内容装入减 1计数寄存器,并开始减 1计数,当计数到一半时,OUT端变为低电平。减 1计数寄存器继续作减 1计数,计数到 0时,OUT端变为高电平。之后,周而复始地自动进行计数过程。当计数初值为偶数时,OUT输出对称方波;当计数初值为奇数时,OUT输出不对称方波。

在计数过程中,若 GATE变为低电平,则停止计数;当 GATE由低变高时,则重新启动计数过程。如果在输出为低电平时,门控信号 GATE变为低电平,减 1计数器停止,而 OUT输出立即变为高电平。在 GATE又变成高电平后,下一个时钟脉冲的下降沿,减 1计数器重新得到计数初值,又开始新的减 1计数。

在计数过程中,如果写入新的计数值,那么,将不影响当前输出周期。但是,如果在写入新的计数值后,又受到门控上升沿的触发,那么,就会结束当前输出周期,而在下一个时钟脉冲的下降沿,减 1计数器重新得到计数初值,又开始新的减 1计数。

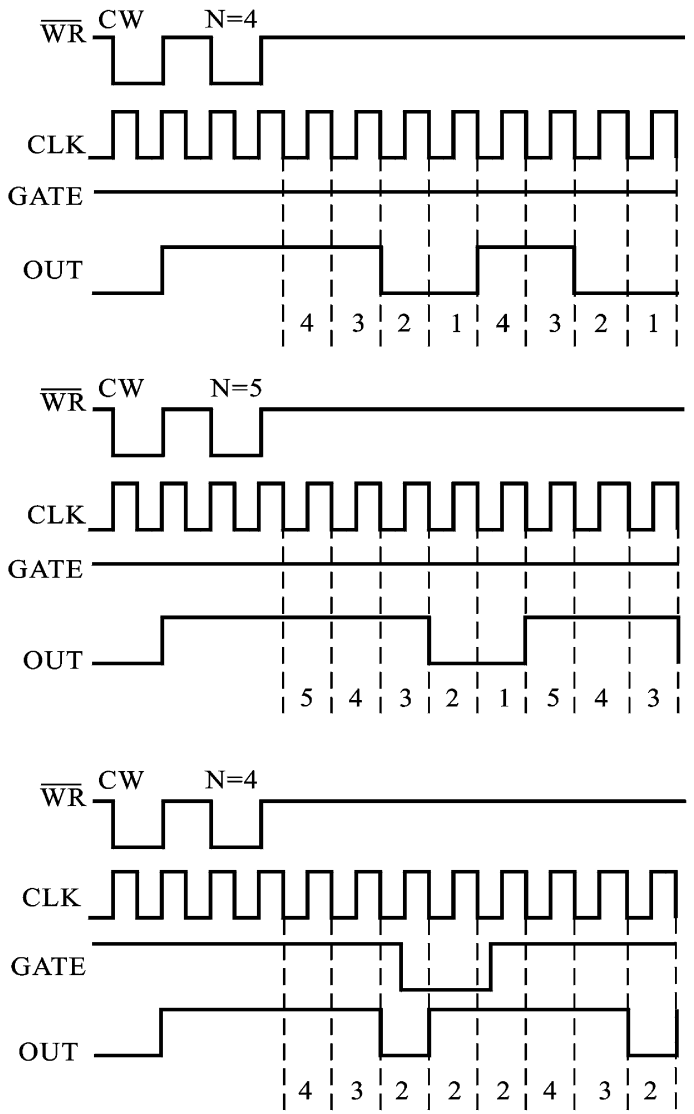


图 9.6　8253方式 3时序波形图

8253方式 3下三种情况的时序波形图如图 9.6所示。

5. 方式 4 软件触发的选通信号发生器）

采用方式 4,可产生单个负脉冲信号,负脉冲宽度为一个时钟周期。写入控制字后,OUT端变为高电平,若 GATE为高电平,当写入计数初值后,在下一个 CLK的下降沿将计数初值寄存器内容装入减 1计数寄存器,并开始减 1计数,当

减 1计数寄存器的值为 0时，OUT端输出低电平，经过一个 CLK时钟周期，OUT端输出高电平。

如果在计数时，又写入新的计数值，则在下一个 CLK的下降沿此计数初值被写入减 1计数寄存器，并以新的计数值作减 1计数。

8253方式 4下三种情况的时序波形图如图 9.7所示：

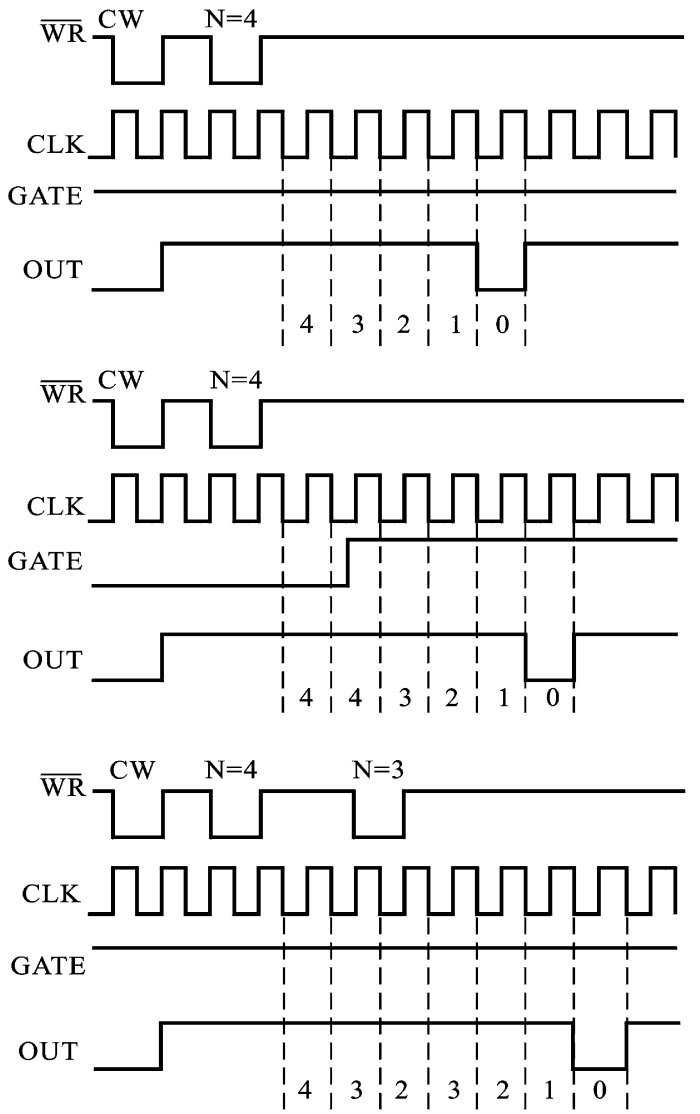


图 9.7 8253方式 4时序波形图

6. 方式 5 硬件触发的选通信号发生器）

方式 5的计数过程由 GATE的上升沿触发。当控制字写入后，OUT端输出高电平，并保持高电平状态。然后写入计数初值，只有在 GATE信号的上升沿之后的下一个 CLK脉冲的下降沿，才将计数初值寄存器内容装入减 1计数寄存

器,并开始减 1计数,当计数值减到 0时,OUT端变为低电平,并持续一个 CLK周期,然后自动变为高电平。

若在计数过程中,GATE端又来一个上升沿触发,则在下一个 CLK脉冲的下降沿,减 1计数寄存器将重新获得计数初值,并按新的初值作减 1计数,直至减为 0为止。

若在计数过程中,写入新的计数值,但没有触发脉冲,则当前输出周期不受影响。当前周期结束后,在再触发的情况下,将按新的计数初值开始计数。

若在计数过程中,写入新的计数值,并在当前周期结束前又受到触发,则在下一个 CLK脉冲的下降沿,减 1计数寄存器将获得新的计数初值,并按此值作减 1计数操作。

8253方式 5下三种情况的时序波形图如图 9.8所示:

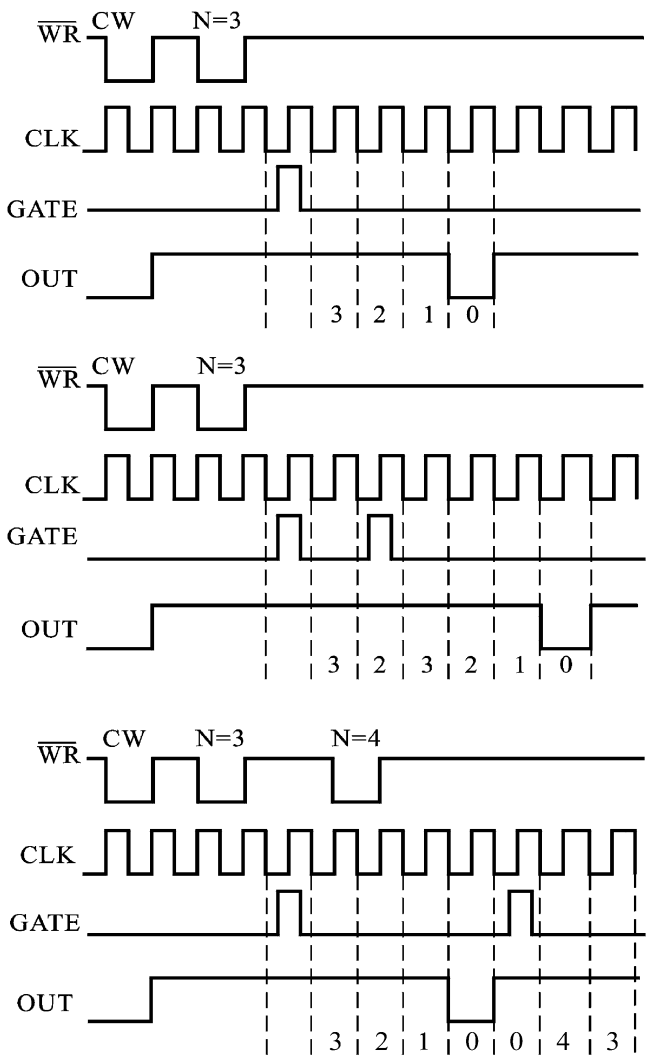


图 9.8　8253方式 5时序波形图

9.2.4 8253的初始化编程

1. 写入控制字

以便选择计数器和规定计数器的工作方式,任一计数通道的控制字都要从8253的控制端口写入。

2. 写入计数初值

某个计数器写入控制字后,任何时候都可按控制字中的 RW_1, RW_2 规定写入计数初始值。写入计数初值时,还必须注意:如果在方式控制字中的 BCD 位为 1,则写入的计数初值应为十六进制数。例如:计数初值为 50,采用 BCD 码计数,则指令中的 50必须写为 50H。

计数初值(TC)的计算公式为:$t=1/f\times TC$,其中 t为定时时间,TC为计数初值,f为输入时钟频率。

3. 读计数值

在计数过程中,若要读取当前的计数值,则需采用以下方法。先写入一个方式控制字,该方式控制字的 SC_1 SC_2 指明要读取的计数通道,RW_1 RW_2 设为 00;然后再按照初始化该计数器时的读写方法读取计数值。

9.2.5 8253应用

【例题 9.1】 8253在 IBM PC中的应用。IBM PC系统板上 8253的接口电路如图 9.9所示,三个计数器的时钟输入频率为 1.1932 MHz,系统分配给8253的端口地址为 40H~43H。

计数器 0为方式 3,先写低字节,后写高字节,二进制计数,计数初值为 0。输出端 OUT_0 接至中断控制器 8259A的 IR_0,OUT_0 输出的脉冲周期约为 55 ms(65536÷1193200),即计数器 0每隔 55 ms产生一次中断请求。

计数器 1为方式 2,只写低字节,二进制计数,计数初值为 18。输出端 OUT_1 接至 DMA控制器 8237A通道 0的 DMA请求 $DREQ_0$,作为定时(15.08 μs)刷新动态存储器的启动信号。

计数器 2为方式 3,先写低字节,后写高字节,二进制计数,计数初值为 0533H。$GATE_2$ 由 8255A的 PB_0 控制,当 $GATE_2$ 为高电平时,OUT_2 输出频率为 896 Hz的方波,经功率放大器和滤波后驱动扬声器发声。

【例题 9.2】 利用 PC内部定时器 0,设计并实现一个数字式计时时钟。显

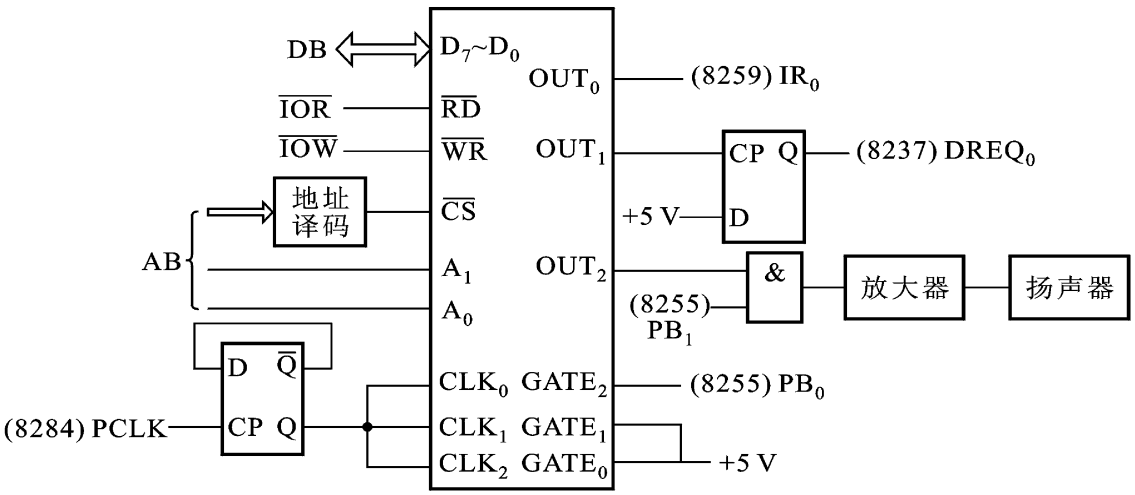


图 9.9　IBM PC系统板上 8253接口电路

示格式为 hh:mm:ss(hh代表时,mm代表分,ss代表秒)。

解题分析

PC内部定时器 0初始设定为每隔 55 ms中断一次,即 1 s中断 18.2 (1000÷55)次,若按初始设定,无法实现 1 s精确定时。需重新设定定时器 0的时间常数,让其每隔 10ms中断一次;再设定一个软件计数器,初始值为 100。每中断一次,软件计数器减 1,当软件计数器减为 0,则定时 1 s。在 PC系统中,定时器 0的中断类型号为 08H,但在中断向量表中,存放 08H中断服务程序入口地址的单元中实际存放的是 INT 1CH指令,因此,当定时器 0中断时,实际是转至 INT 1CH的中断服务程序入口处。其程序如下:

```
DATA        SEGMENT
    COUNT   DB  100                 软件计数器
    TENH    DB  '0'
    HOUR    DB  '0',     ':
    TENM    DB  '0'
    MINU    DB  '0',':
    TENS    DB  '0'
    SECO    DB  '0'
DATA        ENDS
CODE        SEGMENT
  MAIN      PROC    FAR
    ASSUME CS:CODE,DS:DATA
```

```
START: USH   DS
       MOV   AX, 0
       PUSH  AX
       MOV   AX, DATA
       MOV   DS, AX
       CLI                          ;关中断
       MOV   AX, 351CH
       INT   21H                    ;取中断类型号为 1CH 的原系
                                     统中断服务程序入口地址
       PUSH  ES                     ;保存中断服务程序入口地址
                                     的段地址
       PUSH  BX                     ;保存中断服务程序入口地址
                                     的偏移地址
       PUSH  DS
       MOV   DX, OFFSET TIMER       ;设置用户中断服务程序偏移
                                     地址
       MOV   AX, SEG TIMER          ;设置用户中断服务程序段地址
       MOV   DS, AX
       MOV   AX, 251CH              ;设置中断向量
       INT   21H
       POP   DS
       MOV   AL, 36H                ;重设 8253定时器 0控制字
       OUT   43H, AL
       MOV   AX, 11932              ;重写 8253定时器 0时间常数
       OUT   40H, AL
       MOV   AL, AH
       OUT   40H, AL
       STI                          ;开中断
FORE:  MOV   AH, 1                  ;检测键盘按键
       INT   16H
       CMP   AL, 1BH                ;是按下“ESC”键吗
       JZ    EXIT                   ;是,退出
       MOV   BX, OFFSET TENH        ;显示 hh:mm:ss
       MOV   CX, 8
```

```
DISPCLK: MOV   AL, [BX]
         CALL  DISP
         INC   BX
         LOOP  DISPCLK
         MOV   AL,0DH          ;显示回车
         CALL  DISP
         MOV   AL,SECO         ;取秒计数单元值
WAIT1:   CMP   AL,SECO         ;秒计数单元变化否?
         JZ    WAIT1           ;无,等待
         JMP   SHORT FORE      ;有,显示新时间
EXIT:    CLI
         MOV   AL,36H          ;恢复定时器 0的初始设定值
         OUT   43H,AL
         MOV   AX,0
         OUT   40H,AL
         MOV   AL,AH
         OUT   40H,AL
         POP   DX              ;恢复中断类型号 1CH的系统
                               ;初始值
         POP   DS
         MOV   AX,251CH
         INT   21H
         STI
         RET
MAIN     ENDP
TIMER    PROC  NEAR
         PUSH  AX
         DEC   COUNT           ;软件计数器减 1
         JNZ   RETURN          ;软件计数器不为 0,中断返回
         MOV   COUNT,100
         INC   SECO            ;秒加 1
         CMP   SECO,'9'
         JLE   RETURN
```

```
        MOV     SECO,'0'
        INC     TENS
        CMP     TENS,'6'
        JL      RETURN
        MOV     TENS,'0'
        INC     MINU            ;分加 1
        CMP     MINU,'9'
        JLE     RETURN
        MOV     MINU,'0'
        INC     TENM
        CMP     TENM,'6'
        JL      RETURN
        MOV     TENM,'0'
        INC     HOUR            ;小时加 1
        CMP     HOUR,'9'
        JA      ADJHOUR
        CMP     HOUR,'4'        ;判断是否计时到 24时？
        JNZ     RETURN
        CMP     TENH,'2'
        JNZ     RETURN
        MOV     HOUR,'0'        ;若计时到 24时,则回到 00时
        MOV     TENH,'0'
        JMP     SHORT RETURN
ADJHOUR:INC     TENH
        MOV     HOUR,'0'
RETURN: MOV     AL,20H          ;送中断结束命令
        OUT     20H,AL
        POP     AX
        IRET                    ;中断返回
TIMER   ENDP
DISP    PROC    NEAR            ;显示字符子程序
        PUSH    BX
        MOV     BX,0
        MOV     AH,0EH
```

```
        INT     10H
        POP     BX
        RET
DISP    ENDP
CODE    ENDS
END     START
```

【例题 9.3】　利用 8253的通道 0和通道 1,设计并产生周期为 1 Hz的方波。设通道 0的输入时钟频率为 2 MHz,8253所占端口地址为 80H,81H,82H,83H。

解题分析

根据题意可知通道 0的输入时钟周期为 0.5 μs,其最大定时时间为 0.5 μs×65536,即为 32.768 ms,要产生频率为 1 Hz 周期为 1 s 的方波,单独利用一个通道是无法实现的。但可利用通道级联的方法,将通道 0的输出 OUT_0 作为通道 1的输入时钟。

若让 8253通道 0工作于方式 2 (速率发生器),输出脉冲周期为 10 ms,则通道 0的计数值为 20 000。周期为 4 ms的脉冲作为通道 1的输入,要求输出端 OUT_1 的波形为方波且周期为 1 s,则通道 1的计数值为 100。

通过以上分析,硬件连接图如图 9.10所示：

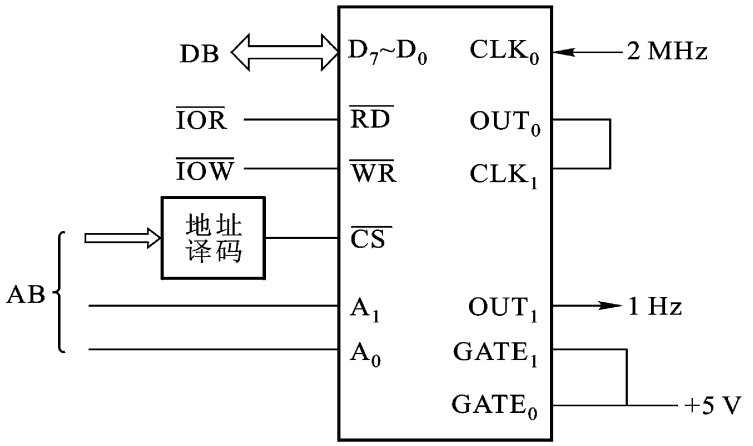


图 9.10　8253级联应用

8253初始化程序如下：

```
MOV     AL,34H          通道 0控制字
OUT     83H,AL
MOV     AX,20000        通道 0时间常数
```

```
OUT     80H, AL
MOV     AL, AH
OUT     80H, AL
MOV     AL, 56H         通道 1控制字
OUT     83H, AL
MOV     AL, 100         通道 1时间常数
OUT     81H, AL
```

9.3 可编程并行接口芯片 8255A

Intel 8255A是一个广泛用于微机系统的、具有 24条 I/O引脚的、可编程并行接口芯片。8255A采用双排直插式封装,使用单一 +5 V电源,全部输入输出与 TTL电平兼容。

9.3.1 8255A内部结构及引脚功能

1. 8255A内部结构

8255A的内部结构如图 9.11a所示,它由 4部分组成:

(1) 数据总线缓冲器

数据总线缓冲器是一个双向三态的 8位数据缓冲器,8255A通过它与系统总线相连。输入数据、输出数据、CPU发给 8255A的控制字都是通过这个缓冲器进行的。

(2) 数据端口 A、B、C

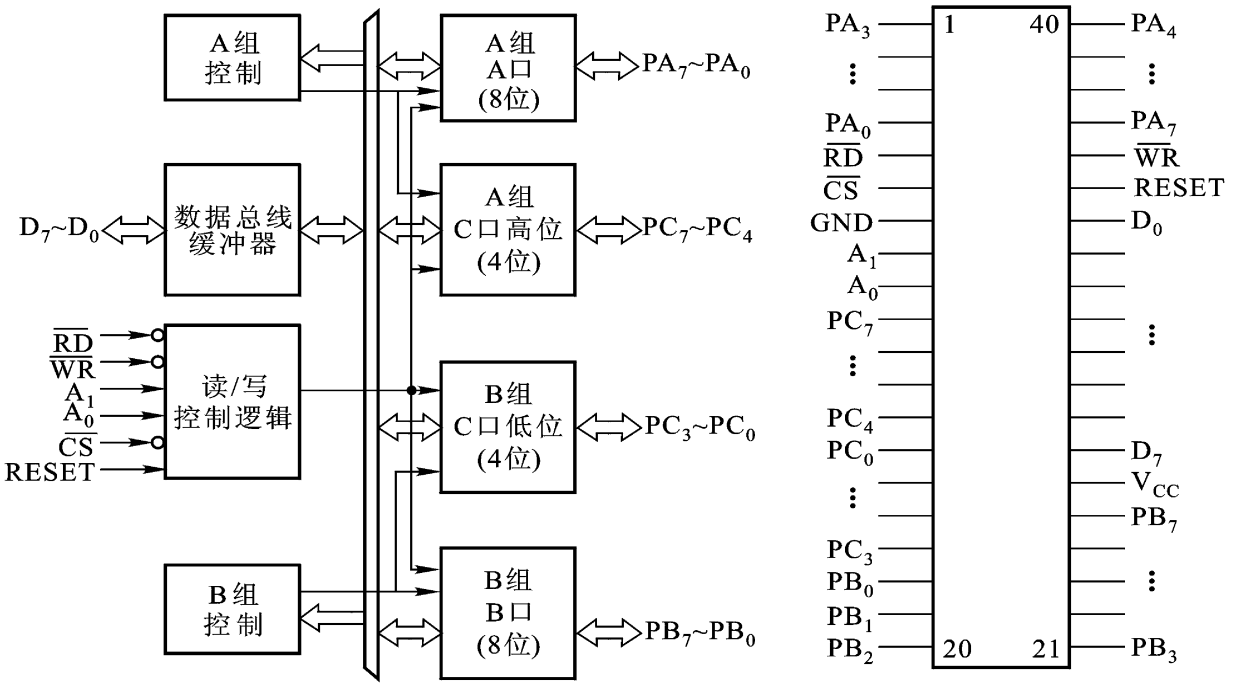
8255A由三个 8位数据端口,即端口 A、端口 B、端口 C。设计人员可通过编程使它们分别作为输入端口或输出端口。不过,这三个端口有各自的特点。

端口 A对应一个 8位数据输入锁存器和一个 8位数据输出锁存器/缓冲器。用端口 A作为输入或输出时,数据均受到锁存。

端口 B和端口 C均对应一个 8位输入缓冲器和一个 8位数据输出锁存器/缓冲器。

在使用中,端口 A和端口 B常常作为独立的输入或者输出端口。端口 C除了可以做独立的输入或输出端口外,还可配合端口 A和端口 B的工作。具体说,端口 C可分成两个 4位的端口,分别作为端口 A和端口 B的控制信号和状态信号。

(3) A组控制和 B组控制



(a) 内部结构图　　(b) 引脚图

图 9.11　8255A内部结构及引脚图

这两组控制电路一方面接收 CPU发来的控制字并决定 8255A 的工作方式；另一方面接收来自读/写控制逻辑电路的读/写命令，完成接口的读/写操作。

A组控制电路控制端口 A和端口 C的高 4位的工作方式和读/写操作。

B组控制电路控制端口 B和端口 C的低 4位的工作方式和读/写操作。

⑷ 读/写控制逻辑

读/写控制逻辑负责管理 8255A的数据传输过程。它接收译码电路的$\overline{CS}$和来自地址总线的 A_1，A_0 信号，以及控制总线的 RESET，$\overline{WR}$，$\overline{RD}$信号，将这些信号进行组合后，得到对 A组控制部件和 B组控制部件的控制命令，并将命令发给这两个部件，以完成对数据信息、状态信息和控制信息的传输。

2. 8255A 引脚功能

8255A芯片除电源和地引脚以外，其他引脚可分为两组，引脚如图 9.11b所示：

⑴ 8255A 与外设连接引脚

8255A与外设连接的有 24条双向、三态数据引脚，分成三组，分别对应于 A，B，C三个数据端口：PA_7 ～PA_0，PB_7 ～PB_0，PC_7 ～PC_0。

⑵ 8255与 CPU连接引脚

$D_7 \sim D_0$：双向、三态数据线。

RESET：复位信号，高电平有效。复位时所有内部寄存器清除，同时 3个数据端口被设为输入。

$\overline{CS}$：片选信号，低电平有效。该信号有效时，8255A被选中。

$\overline{RD}$：读信号，低电平有效。该信号有效时，CPU可从 8255A读取输入数据或状态信息。

$\overline{WR}$：写信号，低电平有效。该信号有效时，CPU可向 8255A写入控制字或输出数据。

A_1，A_0：片内端口选择信号。8255A内部有三个数据端口和一个控制端口。

8255A的 $\overline{CS}$，$\overline{RD}$，$\overline{WR}$，A_1，A_0 控制信号和传送操作之间的关系表 9.3所示：

表 9.3 8255A的控制信号和传送操作的对应关系

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	执行的操作
0	0	1	0	0	读端口 A
0	0	1	0	1	读端口 B
0	0	1	1	0	读端口 C
0	0	1	1	1	非法状态
0	1	0	0	0	写端口 A
0	1	0	0	1	写端口 B
0	1	0	1	0	写端口 C
0	1	0	1	1	写控制端口
1	×	×	×	×	未选通

9.3.2 8255A控制字

8255A有两个控制字：方式选择控制字和端口 C置位/复位控制字。这两个控制字公用一个地址，即控制端口地址。用控制字的 D_7 位来区分这两个控制字，D_7 =1为方式选择控制字；D_7 =0为端口 C置位/复位控制字。

1. 方式选择控制字

方式选择控制字的格式如图 9.12所示：$D_0 \sim D_2$ 用来对 B组的端口进行工作方式设定，$D_3 \sim D_6$ 用来对 A组的端口进行工作方式设定。最高位为 1是方式选择控制字标志。

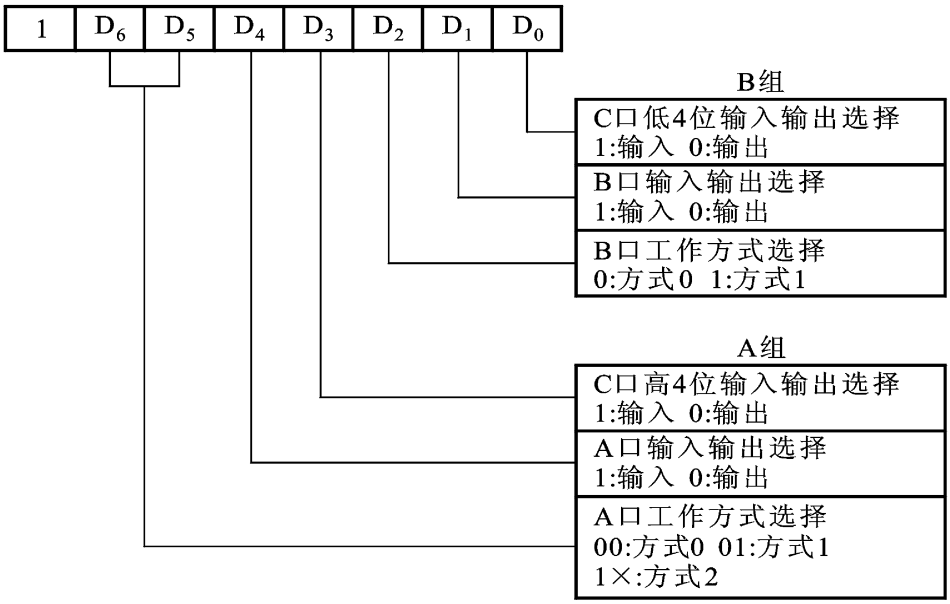
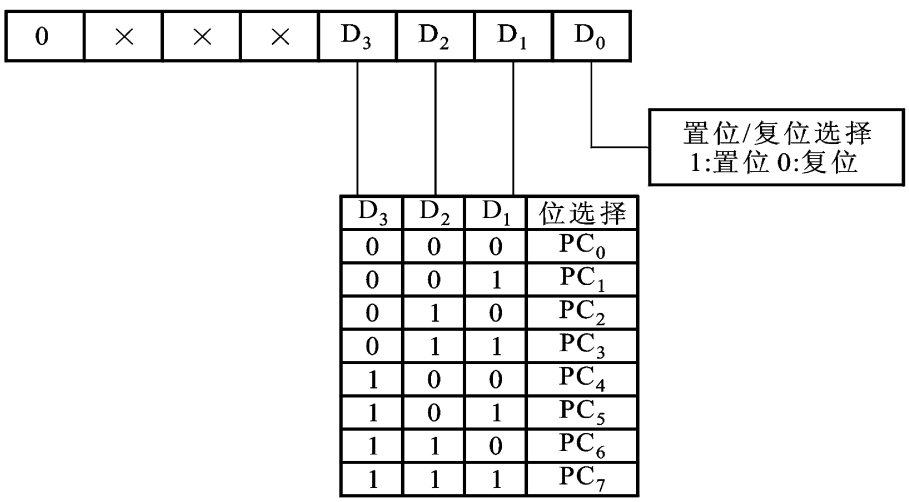


图 9.12　8255A方式选择控制字

2. 端口 C置位/复位控制字

端口 C置位/复位控制字的格式如图 9.13所示：D_3～D_1 三位的编码与端口 C的某一位相对应，D_0 决定置位或复位操作。最高位为 0是端口 C置位/复位控制字标志。



D_3	D_2	D_1	位选择
0	0	0	PC_0
0	0	1	PC_1
0	1	0	PC_2
0	1	1	PC_3
1	0	0	PC_4
1	0	1	PC_5
1	1	0	PC_6
1	1	1	PC_7

图 9.13　8255A端口 C置位/复位控制字

9.3.3 8255A工作方式

1. 方式 0——基本输入输出

方式 0下，每一个端口都作为基本的输入或输出口，端口 C的高 4位和低 4位以及端口 A、端口 B都可独立地设置为输入口或输出口。4个端口的输入/输出可有 16种组合。

8255A工作于方式 0时，CPU可采用无条件读写方式与 8255A交换数据，也可采用查询方式与 8255A交换数据。采用查询方式时，可利用端口 C作为与外设的联络信号。

2. 方式 1——选通输入输出

方式 1下三个端口分为 A、B两组，端口 A、端口 B仍作为输入或输出口，端口 C分成两部分，一部分作为端口 A和端口 B的联络信号，另一部分仍可作为基本的输入输出口。

⑴ 方式 1输入

端口 A、端口 B都设置为方式 1输入时的情况及时序如图 9.14所示：PC_3、PC_4、PC_5 作为端口 A的联络信号，PC_0、PC_1、PC_2 作为端口 B的联络信号。

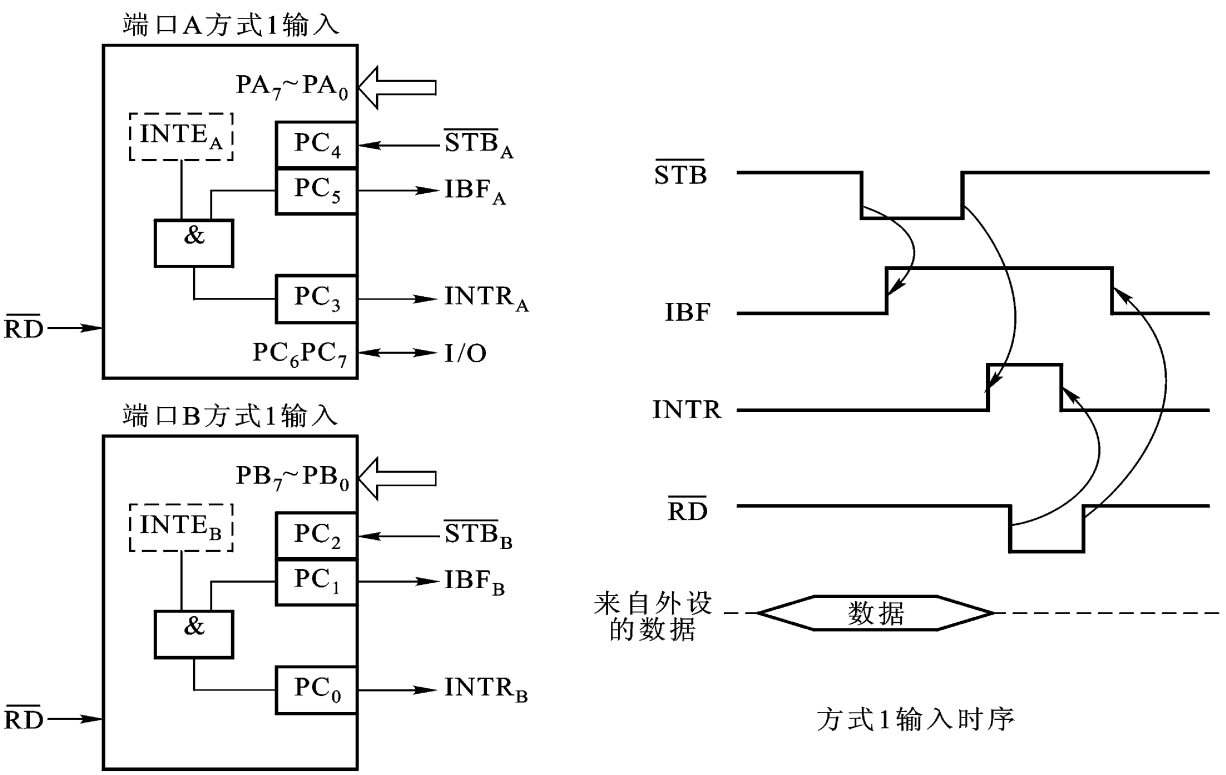


图 9.14 8255A方式 1输入的控制信号和时序

$\overline{STB}$:选通输入,低电平有效。该信号有效时,输入数据被送入锁存端口 A或端口 B的输入锁存器/缓冲器中。

IBF:输入缓冲器满,高电平有效。该信号由 8255A发出,作为$\overline{STB}$信号的应答信号。该信号有效时,表明输入缓冲器中已存放数据,可供 CPU读取。IBF由$\overline{STB}$信号的下降沿置位,由$\overline{RD}$信号的上升沿复位。

INTR:中断请求信号,高电平有效。当 IBF和 INTE均为高电平时,INTR变为高电平。INTR信号可作为 CPU的查询信号,或作为向 CPU发出中断请求的信号。$\overline{RD}$的下降沿使 INTR复位,上升沿又使 IBF复位。

INTE:中断允许信号。端口 A用 PC_4 的置位/复位控制,端口 B用 PC_2 的置位/复位控制。需特别说明的是,对 INTE信号的设置,虽然使用的是对端口 C的置位/复位操作,但这完全是 8255A的内部操作,对已作为$\overline{STB}$信号的引脚 PC_4、PC_2 的逻辑状态没有影响。

⑵ 方式 1输出

端口 A、端口 B都设置为方式 1输出时的情况如图 9.15所示:PC_3、PC_6、PC_7 作为端口 A的联络信号,PC_0、PC_1、PC_2 作为端口 B的联络信号。

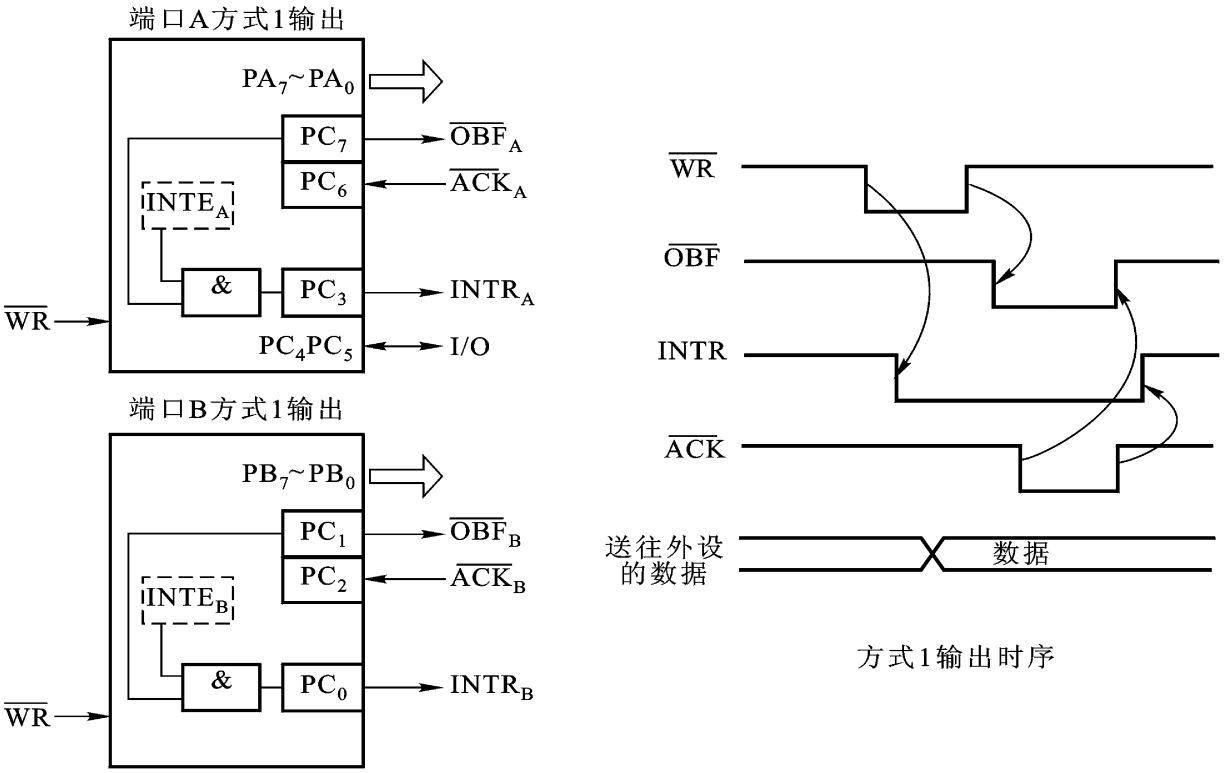


图 9.15　8255A方式 1输出的控制信号和时序

$\overline{OBF}$ 输出缓冲器满，低电平有效。该信号有效时，表明 CPU 已将待输出的数据写入到 8255A 的指定端口，通知外设可从指定端口读取数据。该信号由 $\overline{WR}$ 的上升沿置为有效。

$\overline{ACK}$：响应信号，低电平有效。该信号由外设发给 8255A，有效时，表示外设已取走 8255A 的端口数据。

INTR：中断请求信号，高电平有效。当输出缓冲器空（$\overline{OBF}=1$），中断允许 INTE＝1 时，INTR 变为高电平。INTR 信号可作为 CPU 的查询信号，或作为向 CPU 发出中断请求的信号。$\overline{WR}$ 的下降沿使 INTR 复位。

INTE：中断允许信号。端口 A 用 PC_6 的置位/复位控制，端口 B 用 PC_2 的置位/复位控制。

3. 方式 2——双向选通输入输出

方式 2 为双向传输方式。8255A 的方式 2 可使 8255A 与外设进行双向通信，既能发送数据，又能接收数据。可采用查询方式和中断方式进行传输。

方式 2 只适用于端口 A，端口 C 的 PC_7～PC_3 配合端口 A 的传输，其联络信号如图 9.16 所示。INTE1 为输出中断允许，由 PC_6 置位/复位；INTE2 为输入中断允许，由 PC_4 置位/复位。

当端口 A 工作于方式 2，端口 B 工作于方式 0 时，PC_7～PC_3 作为端口 A 的联络信号，PC_0～PC_2 可工作于方式 0；当端口 A 工作于方式 2，端口 B 工作于方式 1 时，PC_7～PC_3 作为端口 A 的联络信号，PC_0～PC_2 作为端口 B 的联络信号。端口 A 方式 2 和端口 B 方式 1 时端口 C 各位的功能如表 9.4 所示。

表 9.4 端口 A 方式 2 和端口 B 方式 1 时端口 C 的功能

端口 C	端口 A 方式 2 和端口 B 方式 1	
	输入	输出
PC_7	$\overline{OBF_A}$	
PC_6	$\overline{ACK_A}$	
PC_5	IBF_A	
PC_4	$\overline{STB_A}$	
PC_3	$INTR_A$	
PC_2	$\overline{STB_B}$	$\overline{ACK_B}$
PC_1	IBF_B	$\overline{OBF_B}$
PC_0	$INTR_B$	

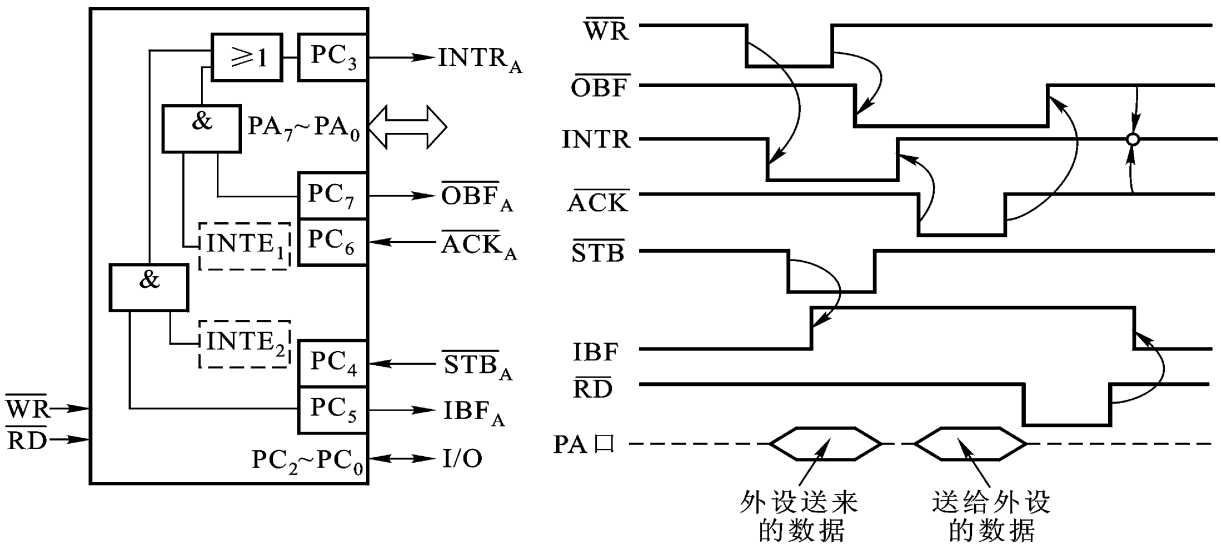


图 9.16　8255A方式 2的控制信号和时序

9.3.4　8255A应用

【例题 9.4】　扫描键盘按键,并保存相应键值,硬件图如图 9.17所示。设 8255A的端口地址为 300H～303H,接收 100个按键后结束。

检测键盘输入过程如下:PC_4～PC_7送全“0”,再读取 PC_0～PC_3,若全为“1”,则表示无键闭合。若有键闭合,则进行键扫描。键扫描方法如下:使 PC_4为 0,PC_5～PC_7为高电平,读取 PC_0～PC_3,如果是全“1”,表示该列无键闭合;否则闭合键在该列上,再进一步判断读取的数据中哪一位为“0”,从而确定闭合键。若该列无键闭合,则依次使 PC_5,PC_6,PC_7进行上述操作。

在键盘设计时,除了对键码识别外,还有抖动和重键两个问题需要解决。

对机械按键就是当用手按下一个键时,往往会出现按键在闭合和断开位置之间跳几下才稳定到闭合状态的情况;在释放一个键时,也会出现类似的情况,这就是抖动。抖动持续时间一般为 10ms左右。利用硬件,也可通过软件延时来消除抖动。

所谓重键就是指两个或多个键同时闭合。通常情况,则是只承认先识别出来的键,对同时按下的其他键均不作识别,直到所有键都释放以后,才读下一个键。

```
DATA    SEGMENT
    BUFFER    DB 100 DUP (?)
```

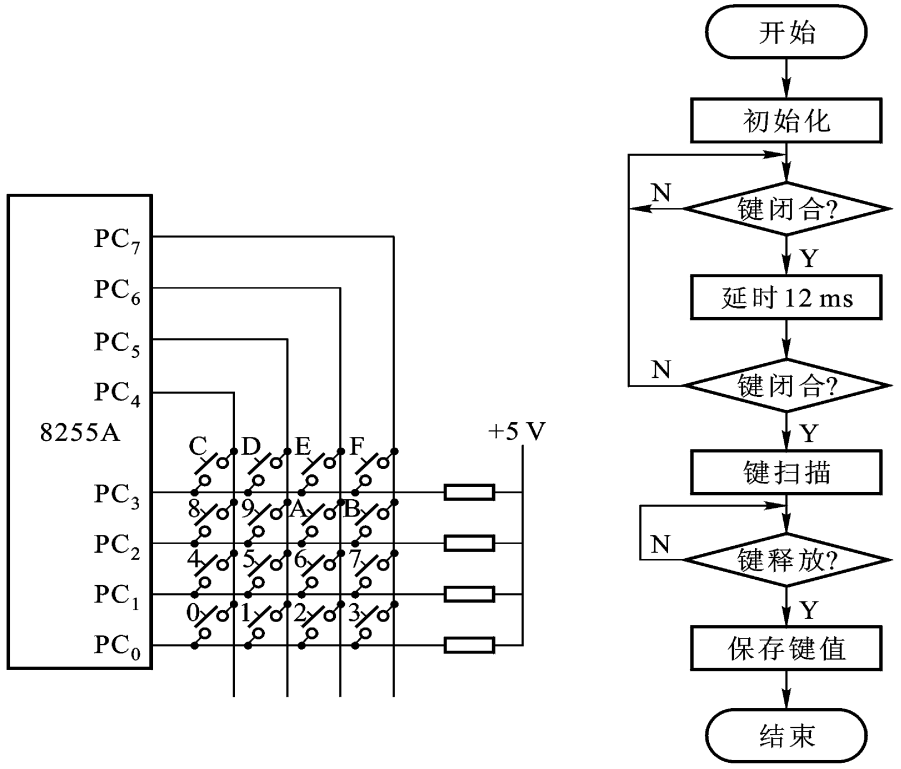


图 9.17 简单键盘接口及软件流程

```
DATA    ENDS
CODE    SEGMENT
        ASSUME      CS:CODE,DS:DATA
START:  OV      AX,DATA
        MOV     DS,AX
        LEA     SI,BUFFER
        MOV     CL,100          ;初始化按键次数
        MOV     AL,81H          ;8255A控制字
        MOV     DX,303H
        OUT     DX,AL           ;8255A初始化
KS1:    CALL    KS              ;读取按键
        CMP     AL,0FH          ;判有无键闭合
        JZ      KS1             ;无键闭合,循环等待
        CALL    DELAY           ;延时 12 ms,消除抖动
        CALL    KS
        CMP     AL,0FH          ;再次判有无键闭合
```

```
        JZ      KS1
        MOV     BL,0EFH         ;初始化列码
        MOV     BH,0            ;初始化列计数器
AGAIN:  MOV     DX,302H
        MOV     AL,BL
        OUT     DX,AL           ;输出列码
        IN      AL,DX           ;读取行码
        AND     AL,0FH
        CMP     AL,0FH
        JZ      NEXT            ;该列无键闭合,准备下一列
                                 扫描
        CMP     AL,0EH          ;判该列是否第一个键闭合?
        JNZ     TWO
        MOV     AL,0
        JMP     FREE
TWO:    CMP     AL,0DH          ;判该列是否第二个键闭合
        JNZ     THREE
        MOV     AL,4
        JMP     FREE
THREE:  CMP     AL,0BH          ;判该列是否第三个键闭合
        JNZ     FOUR
        MOV     AL,8
        JMP     FREE
FOUR:   CMP     AL,07H          ;判该列是否第四个键闭合
        JNZ     NEXT
        MOV     AL,0CH
FREE:   PUSH    AX
WAIT1:  CALL    KS
        CMP     AL,0FH
        JNZ     WAIT1           ;键未释放,则等待
        POP     AX
        ADD     AL,BH           ;按键键值 =扫描键值 +列
                                 计数值
        MOV     [SI],AL         ;保存相应按键键值
```

```
            INC     SI
            DEC     CL
            JZ      EXIT            ;判是否接收到 100个按键
            JMP     KS1
NEXT:       INC     BH              ;列计数值加 1
            ROL     BL,1            ;列码循环左移一位
            CMP     BL,0FEH         ;判该轮键扫描是否结束
            JNZ     AGAIN
            JMP     KS1
EXIT:       MOV     AH,4CH          ;返回 DOS
            INT     21H
KS          PROC    NEAR
            MOV     DX,302H
            MOV     AL,0FH
            OUT     DX,AL           ;使所有列线为低电平
            IN      AL,DX           ;读取行值
            AND     AL,0FH          ;屏蔽高 4位
            RET
KS          ENDP

DELAY       PROC    NEAR            ;延时子程序
            PUSH    BX
            PUSH    CX
            MOV     BX,2000
DEL1:       MOV     CX,0
DEL2:       LOOP    DEL2
            DEC     BX
            JNZ     DEL1
            POP     CX
            POP     BX
            RET
        DELAY   ENDP
CODE    ENDP
    END     START
```

注：延时子程序中 BX,CX的初始值随不同型号计算机应作相应调整。

【例题 9.5】　试编程实现采用动态扫描方法在 LED数码管上显示 0000～9999,硬件图如图 9.18所示。设 8255A的端口地址为 300H～303H。

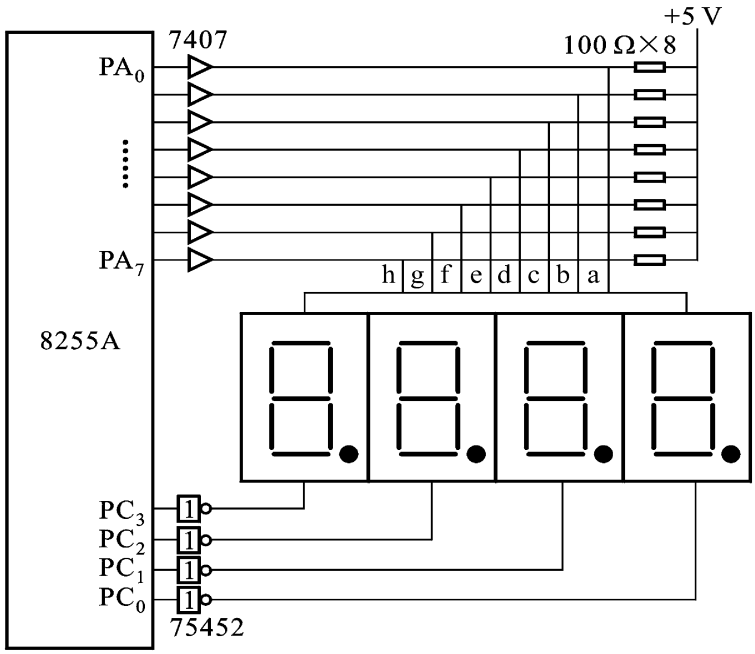


图 9.18　数码管动态显示接口

LED (Light Emitting Diode)数码管的主要部分是发光二极管,如图 9.19所示。这 7段发光管按顺时针分别称为 a、b、c、d、e、f、g,有的产品还附带小数点 h。LED数码管有共阴极和共阳极两种结构。通过 7个发光段的不同组合,可显示 0～9和 A～F以及某些特殊字符。

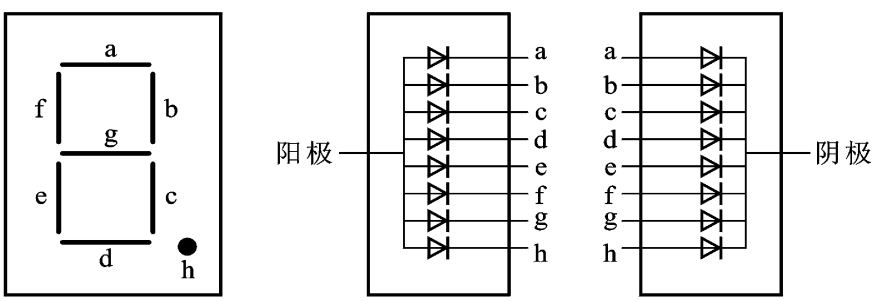


图 9.19　LED数码管

由于发光二极管发光时,通过的平均电流为 10～20 mA,而通常的输出锁存器不能提供这么大的电流,所以 LED各段必须接驱动电路。

点亮数码管有静态和动态两种方法。所谓静态显示,就是当数码管显示某一个字符时,相应的发光二极管恒定地导通或截止。这种显示方式每一个数码管都需要有一个 8位输出口控制,而当系统中数码管较多时,用静态显示所需的I/O口太多,一般采用动态显示方法。

所谓动态显示就是一位一位地轮流点亮各位数码管(扫描),对于每一位数码管来说,每隔一段时间点亮一次。数码管的亮度既与导通电流有关,也与点亮时间和间隔时间的比例有关。调整电流和时间参数,可实现亮度较高较稳定的显示。这种显示方法需有两类控制端口,即位控制端口和段控制端口。位控制端口控制哪个数码管显示,段控制端口决定显示代码。段控制端口所有数码管公用,因此,当 CPU输出一个显示代码时,各数码管的输入段都收到此代码。但是,只有位控制码中选中的数码管才得到导通而显示。

```
DATA    SEGMENT
  OUTBUFF    DB 4DUP (?)
  LEDTAB     DB 3FH, 06H, 5BH, 4FH, 66H, 6DH, 7DH, 07H, 7FH, 6FH
  COUNT      DB 100
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE, DS:DATA
        START:  MOV     AX, DATA
                MOV     DS, AX
                MOV     AL, 80H
                MOV     DX, 303H
                OUT     DX, AL              ;8255A初始化
                MOV     BX, 0               初始化显示值
        NEXT:   LEA     SI, OUTBUFF
                MOV     AX, BX              ;将显示值转换为十进制
                                            数并保存
                MOV     DX, 0
                MOV     CX, 1000
                DIV     CX
                MOV     [SI], AL
                INC     SI
                MOV     AX, DX
                MOV     CL, 100
```

```
        DIV     CL
        MOV     [SI], AL
        INC     SI
        MOV     AL, AH
        MOV     AH, 0
        MOV     CL, 10
        DIV     CL
        MOV     [SI], AL
        INC     SI
        MOV     [SI], AH
AGAIN:  MOV     CH, 08H          ;初始化位选码
        LEA     SI, OUTBUFF
LEDDISP:MOV     AL, [SI]         ;取显示值
        MOV     AH, 0
        LEA     DI, LEDTAB
        ADD     DI, AX
        MOV     AL, [DI]         ;转换为段码
        MOV     DX, 300H
        OUT     DX, AL           ;输出段码
        MOV     DX, 302H
        MOV     AL, CH
        OUT     DX, AL           ;输出位选码
        CALL    DELAY            ;延时 2ms
        INC     SI
        ROR     CH, 1            ;指向下一个数码管
        CMP     CH, 80H
        JNZ     LEDDISP          ;判该轮是否显示结束
        DEC     COUNT            ;重复显示某数 100次,便
                                 ;于看清该数
        JNZ     AGAIN
        MOV     COUNT, 100
        INC     BX               ;显示数值加 1
        CMP     BX, 10000
        JZ      EXIT
```

```
        JMP     NEXT
EXIT:   MOV     AH,4CH          返回 DOS
        INT     21H
DELAY   PROC    NEAR            延时子程序
        PUSH    BX
        PUSH    CX
        MOV     BX,10
DEL1:   MOV     CX,0
DEL2:   LOOP    DEL2
        DEC     BX
        JNZ     DEL1
        POP     CX
        POP     BX
        RET
DELAY   ENDP
CODE    ENDS
        END     START
```

注：延时子程序中 BX,CX 的初始值随不同型号计算机应作相应调整。

【例题 9.6】 采用 8255A 作为与打印机接口的电路，CPU 与 8255A 利用查询方式输出数据，硬件图如 9.20所示，试编程实现将若干个字节数据送打印机打印。设 8255A 的端口地址为 90H～93H。

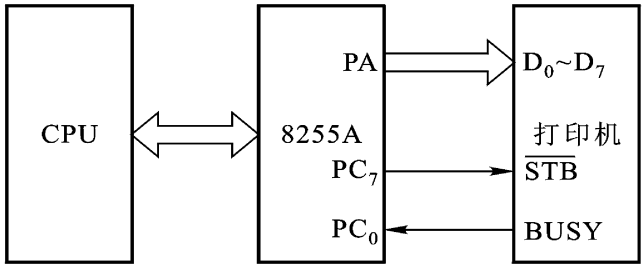


图 9.20 简单的打印机接口

解题分析

打印机一般有三个主要信号，BUSY 表示打印机是否处于“忙”状态，高电平表示打印机处于忙状态。$\overline{STB}$为选通信号，低电平有效，该信号有效时，CPU 输出的数据被锁存到打印机内部数据缓冲器。$\overline{ACK}$为打印机应答信号，当打印机处理好输入数据后发出该信号，同时撤销忙信号。CPU 可利用 BUSY 信号或

$\overline{ACK}$信号决定是否输出下一个数据。

当 CPU通过打印接口要求打印机打印数据时,一般先查询 BUSY 信号,BUSY为低电平时,输出数据至打印口,再发送$\overline{STB}$信号。具体程序如下：

```
DATA    SEGMENT
  BUFFER    DB '45A…'
  COUNT     DW  $ -BUFFER
DATA    ENDS
CODE    SEGMENT
  ASSUME    CS:CODE, DS:DATA
  START:  OV    AX, DATA
          MOV   DS, AX
          LEA   SI, BUFFER
          MOV   CX, COUNT
          MOV   AL, 81H          ;8255A初始化
          OUT   93H, AL
          MOV   AL, 0FH          ;使 PC7 ="1"
          OUT   93H, AL
  NEXT:   IN    AL, 92H          ;读 PC端口
          TEST  AL, 01H          ;测试 BUSY信号
          JNZ   NEXT
          MOV   AL, [SI]         ;读取一个数据,送入 PA端口
          OUT   90H, AL
          MOV   AL, 0EH          ;输出选通脉冲 STB
          OUT   93H, AL
          NOP
          NOP
          MOV   AL, 0FH
          OUT   93H, AL
          INC   SI
          LOOP  NEXT
          MOV   AH, 4CH          ;返回 DOS
          INT   21H
  CODE    ENDS
    END   START
```

【例题 9.7】 在两台计算机之间利用 8255A 的端口 A 实现并行数据传送。A 机的 8255A 采用方式 1 发送数据，B 机的 8255A 采用方式 0 接收数据。A 机 8255A 工作于方式 1 输出，B 机 8255A 工作于方式 0 输入。两机的 CPU 与 8255A 接口之间均采用查询方式交换数据，如图 9.21 所示。试编程实现将 A 机缓冲区 0300:0000H 开始的 1024 个字节数据发送至 B 机，并存放于 B 机从 0400:0000 开始的缓冲区中。设两机 8255A 的端口地址均为 300H～303H。

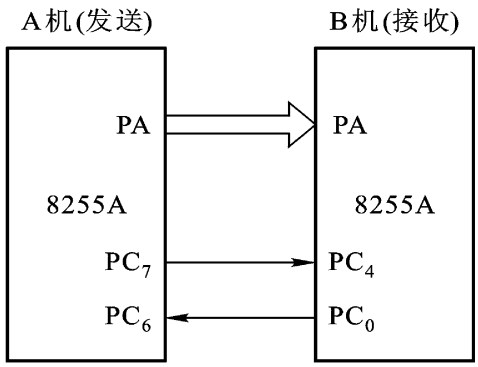


图 9.21 并行通信接口

A 机发送程序段：

```
        MOV     DX, 303H
        MOV     AL, 0A0H
        OUT     DX, AL          ;8255A 初始化，端口 A 方式 1 输出
        MOV     AL, 0DH
        OUT     DX, AL          ;使 PC6 (INTEA) =1,允许中断
        MOV     AX, 0300H
        MOV     DS, AX
        MOV     BX, 0
        MOV     CX, 1024
NEXT:   MOV     DX, 302H
WAIT1:  IN      AL, DX          ;查询 PC3 (INTRA) =1?
        TEST    AL, 08H
        JZ      WAIT1
        MOV     DX, 300H        ;发送数据
```

```
        MOV     AL, [BX]
        OUT     DX, AL
        INC     BX
        LOOP    NEXT
```

B机接收程序段：

```
        MOV     DX, 303H
        MOV     AL, 98H
        OUT     DX, AL          ;8255A 初始化,端口 A 方式 0
                                 输入
        MOV     AL, 01H
        OUT     DX, AL          ;使 PC0 (ACK) =1
        MOV     AX, 0400H
        MOV     DS, AX
        MOV     BX, 0
        MOV     CX, 1024
NEXT:   MOV     DX, 302H
WAIT1:  IN      AL, DX          ;查询 PC4 (OBF) =0?
        TEST    AL, 10H
        JNZ     WAIT1
        MOV     DX, 300H        ;接收并保存数据
        IN      AL, DX
        MOV     [BX], AL
        INC     BX
        MOV     DX, 303H        ;产生 ACK信号
        MOV     AL, 00H
        OUT     DX, AL
        NOP
        NOP
        MOV     AL, 01H
        OUT     DX, AL
        LOOP    NEXT
```

思考题与习题

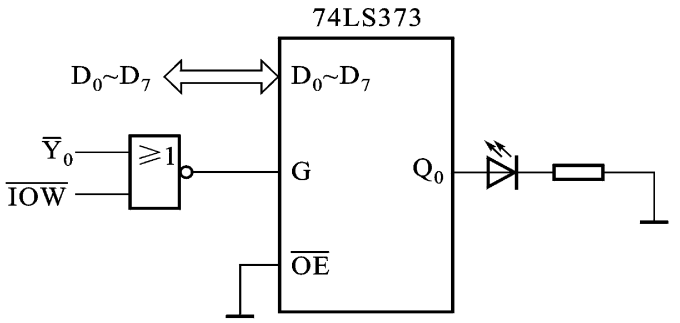
9.1 试按如下要求分别编写初始化程序，已知计数器 0～2和控制字寄存器的端口地址依次为 204H～207H。

⑴ 使计数器 1工作在方式 0，仅用 8位二进制计数，计数初值为 128。

⑵ 使计数器 0工作在方式 1，按 BCD码计数，计数值为 3000。

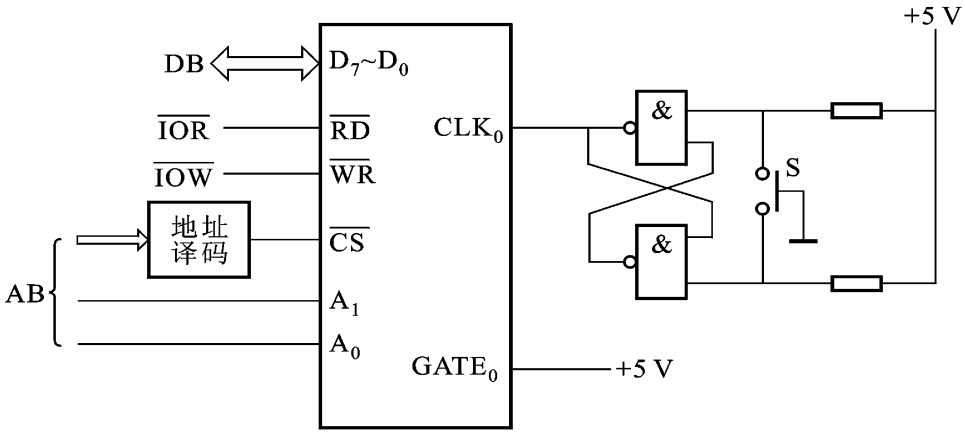
⑶ 使计数器 2工作在方式 2，按二进制计数，计数值为 02F0H。

9.2 硬件如题 9.2图，利用 PC内部 8253定时器 0，并用中断方式实现每隔 1 s使发光两管亮暗交替显示。（PC内部 8253输入时钟频率为 1.1932 MHz，8253定时器 0的端口地址为 40 H，控制寄存器端口地址为 43 H。现为 350 H）。



题 9.2图

9.3 硬件如题 9.3图，8253采用方式 0，BCD码计数方式，初始值为 1000，每按一次按钮 S，计数值减 1。试编程实现显示 8253当前计数值，直至计数值为 0。8253端口地址范围为 80 H～83 H。



题 9.3图

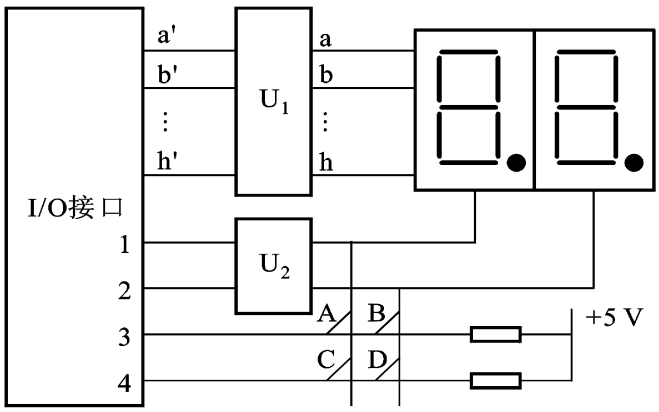
9.4　图 9.4为一简化的键盘/数码管显示接口电路,假设端口线 a′～h及 1、2脚送“1”电平,可使显示器点亮并能实现键扫描。试问 U_1、U_2及数码显示器的选用。

⑴ U_1为_____(同相驱动器/反相驱动器)。

⑵ U_2为_____(同相驱动器/反相驱动器)。

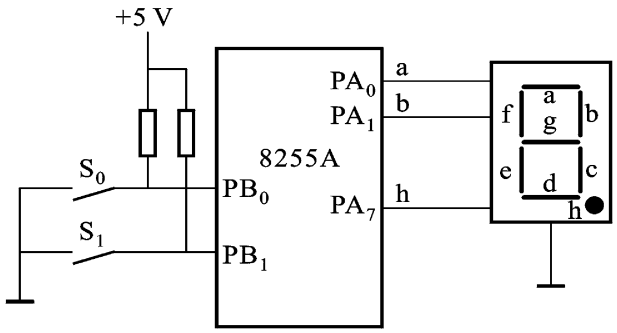
⑶ 数码显示器为_____(共阴/共阳)数码管。

⑷ 若 A键闭合,则端口线 3,4的电平为_____。



题 9.4图

9.5　硬件如题 9.5图,试编程实现,循环检测 S_0、S_1,当 S_0按下数码管显示 0,当 S_1按下数码管显示 1,S_0、S_1同时按下,则结束程序。8255A端口地址范围为 80 H～83 H。



题 9.5图

第10章

串行通信

10.1 基本概念

随着信息技术的发展,微型计算机在远程通信领域中的应用日益增多,已经成为人们不可缺少的通信工具。同时,微型计算机本身也带有若干外设(如鼠标、打印机、绘图仪、摄像头等),需要与它们进行数据通信。在计算机数据传送中,有两种基本的数据传送方式:串行通信和并行通信。采用串行通信时,数据通过一条线路传输,因此可以简化通信设备、降低使用通信线路的价格,并且能利用现有的通信系统。因此,人们为微型计算机制定了适合各类外部设备的接口标准,设计了相应的串行、并行通信接口。

10.1.1 串行通信与并行通信

并行通信是指利用多根传输线将多位数据同时进行传送。一字节的数据通过8条传输线同时发送。由于并行通信方式使用的线路多,一般用在如计算机与打印机等距离短、数据量大的场合。

串行通信是指利用一条传输线将数据一位一位地按顺序分时传输。当传送一字节的数据时,8位数据通过一条线分8个时间段发出,发出顺序一般是由低位到高位。

串行通信与并行通信的示意图见图10.1。同样传送一个字节数据,并行传

送需要 1 T时间,而串行传送需要 8 T时间。

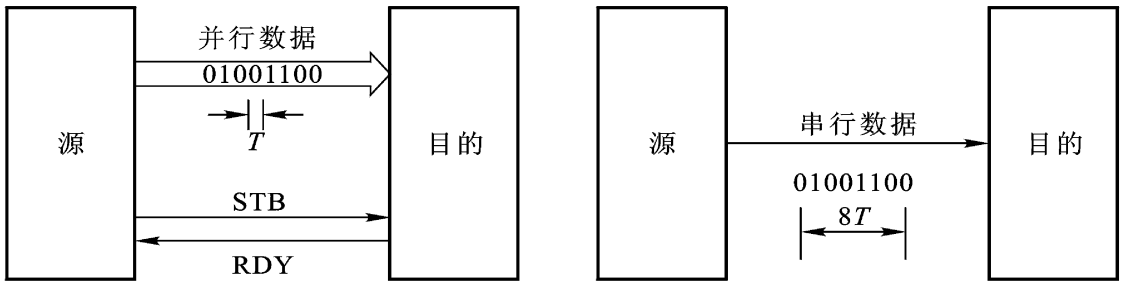


图 10.1 串行通信与并行通信的示意图

串行通信的优势是用于通信的线路少,因而在远距离通信时可以降低通信成本。另外,它还可以利用现有的通信信道 (如电话线路等),使数据通信系统遍布千千万万个家庭和办公室。串行通信适合于远距离数据传送,例如,微型机与计算中心之间、微机系统之间或与其他系统之间。串行通信也由于连线方便而常用于速度要求不高的近距离数据传送,例如,同房间的微型机之间、微型机与绘图机之间、微型机与字符显示器之间。PC系列上都有两个串行异步通信接口,键盘、鼠标器与主机之间也采用串行数据传送方式。

相对于并行通信方式,串行通信速度较慢。现在,高速的串行通信标准如USB接口标准已制定,获得了广泛应用。

10.1.2 异步串行通信

串行通信系统中为了使收发数据正确,收发两端操作必须相互协调,即收发在时间上应同步。同步方式有两种:异步串行通信 ASYNC (Asynchronous Data Communication) 和同步串行通信 SYNC (Synchronous Data Communication)。

异步传送是计算机通信中常用的串行通信方式。异步是指发送端和接收端不使用共同的时钟,也不在数据中传送同步信号。在这种方式下,收方与发方之间必须约定数据帧格式和波特率。

1. 数据帧格式

图 10.2为异步传送的数据帧格式。每帧包括:1个起始位 (低电平)、5~8个数据位、1个可选的奇偶校验位、1~2个终止位 (高电平)。

相邻两个数据帧之间的间隔称为空闲位,长度任意,为高电平。由高电平变为低电平就是起始位,后面紧跟的是 5~8位有效数据位。传送时数据的低位在前、高位在后。数据的后面跟奇偶校验位 (可选),结束是高电平的终止位 (1~2

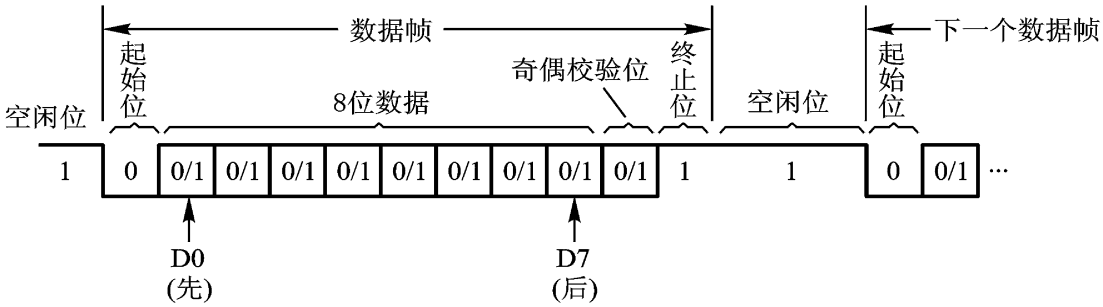


图 10.2 异步通信的数据帧格式

位)。起始位至停止位构成一帧。下一数据帧的开始又以下降沿为标志,即起始位开始。通常 5～8位数据可表示一个字符,如 ASCII码就是 7位。

2. 波特率 (Baud Rate)

波特率是衡量串行数据传送速度的参数,是指单位时间内传送二进制数据的位数,以位/秒为单位 (bps, b/s),也称为波特。PC中异步串行通信的速度一般为 50到 19 200波特之间。常用的波特率有 50、75、100、110、150、300、600、1 200、2 400、4 800、9 600、19 200。

由于每一帧开始时将进行起始位的检测,因此收发双方的起始时间是对齐的。收发双方使用相同的波特率,虽然收发双方的时钟不可能完全一样,但由于每一帧的位数最多只有 12位,因此时钟的微小误差不会影响接受数据的正确性。这就是异步串行通信能实现数据正确传送的基本原理。

【例题 10.1】 设数据帧为 1位起始位、1位终止位、7位数据位、1位奇偶校验位,传送的波特率为 1 200。用 7位数据位代表一个字符,求最高字符传送速度。

答:1 200 (位/秒)/10 (位) =120 (字符/秒)

【例题 10.2】 设数据帧为 1位起始位、2位终止位、8位数据位、1位奇偶校验位,要求每秒传送字节数大于 1 KB,则波特率应大于多少波特?

答:12 (位/秒) ×1 000=12 000 (位/秒),波特率应大于 12 000位/秒。

10.1.3 同步串行通信

异步串行通信中每一帧都需要附加起始位和停止位使数据成帧,因而降低了传送有效数据的效率。对于快速传送大量数据的场合,为了提高数据传输的效率,一般采用同步串行传送。

同步传送时，无需起始位、停止位。每一帧包含较多的数据，在每一帧开始处使用 1～2个同步字符以表示一帧的开始。一种同步串行通信的数据格式如图 10.3所示。同步传送要求对传送的每一位在收发两端保持严格同步，发送、接收端可使用同一时钟源以保证同步，或在发送端采用某种编码方式，在收端将时钟恢复。

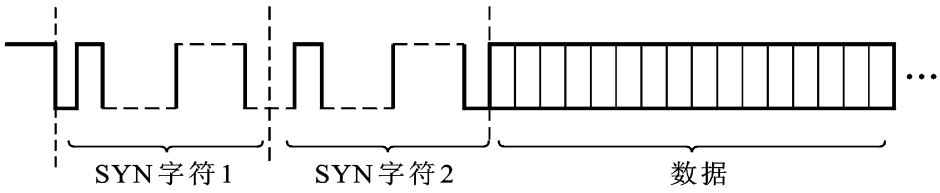


图 10.3 某种同步串行通信的数据格式

10.1.4 串行通信中的数据传送模式

串行通信中，数据在两个站 A或 B之间传送，有单工、半双工与全双工三种模式。如图 10.4所示。

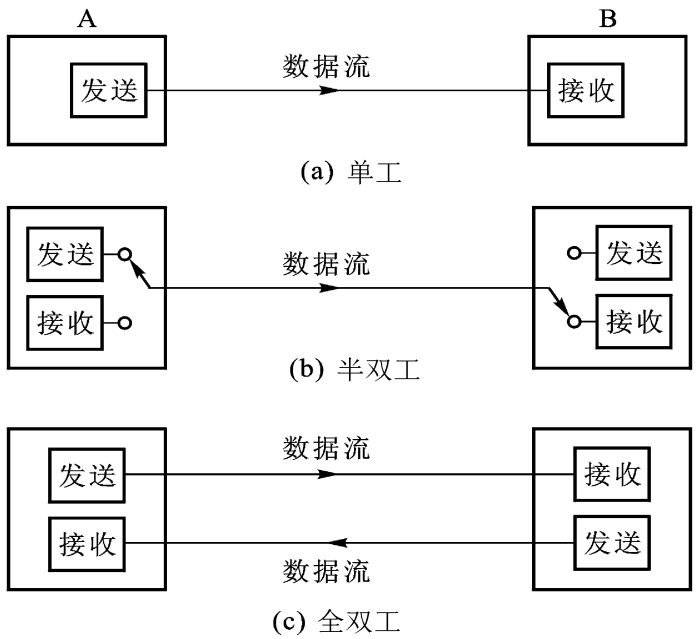


图 10.4 单工、半双工与全双工

1. 单工 (Simplex)

只能进行一个方向的数据传送。如图 10.4a所示,A作为发送器,B作为接受器,数据由 A发送到 B。

2. 半双工 (Half-Duplex)

这种方式下,A、B交替地进行双向数据传送。但由于两设备之间只有一条传输线,因此只能分时地进行收和发,不能同时进行双向数据传送。如图 10.4b所示

3. 全双工 (Full-Duplex)

两设备之间有两根传输线,对于每一个设备来讲都有一条专用的发送线和一条专用的接收线,因此可以同时实现双向数据传送。如图 10.4c所示

10.1.5 信号的调制和解调

如果直接以逻辑电平表示的数字信号进行传送,由于其频谱很宽,需要的通信线路的频带也就很宽。

在进行远程数据通信时,通信线路往往是借用现有的公用电话网或其他通信网络。而现有的通信网的带宽是一定的,如电话线路的带宽是 3.4 kHz,因此不合适直接传输二进制数据。为了利用电话线传输数字信号,必须采取一些措施,把数字信号转换为适合传输的模拟信号,而在接收端再将其转换成数字信号。前一种转换称为调制,后一种转换称为解调。完成调制、解调功能的设备称为调制解调器 (Modem)。应选择合适的调制方式,使调制器输出的信号频谱在通信线路的频带范围内,同时又具有较高的数据传输率,并且接收端易于解调。

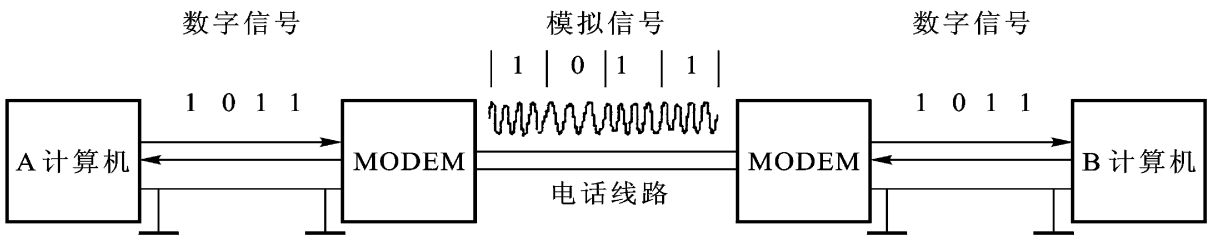


图 10.5 调制和解调

图 10.5中 Modem即调制解调器 (Modulator-Demodulator),能实现数字信号的调制和解调。调制的方式有幅移键控 ASK (Amplitude Shift Keying)、

频移键控 FSK (Frequency Shift Keying) 和相移键控 PSK (Phase Shift Keying)。调制解调器大体上可分为两类,一类是异步调制解调,另一类是同步调制解调。

异步调制解调适用于异步通信方式,常用的调制方式是频移键控 FSK。其基本原理是将“0”和“1”两种数字信号以不同频率的信号表示,如图 10.5所示。

同步调制解调器主要用于高速同步通信,常用的方式是调相和正交幅度调制等。PSK的调制方式与 FSK相比,使用的频带较窄,传送速率高,抗干扰性能强,因此高速调制解调器一般采用 PSK调制。如 V.92标准的调制解调器,可利用电话线上网,其通信速率达到 56 Kbps。

10.1.6 串行接口标准 RS-232C

RS-232C是得到广泛使用的串行异步通信接口标准。它是美国电子工业协会 (Electronic Industry Association, EIA) 于 1962年公布,并于 1969年修订的串行接口标准。事实上已经成为国际上通用的标准串行接口。1987年 1月,RS-232C经修改后,正式改名为 EIA-232D。由于标准修改并不多,因此,现在很多厂商仍沿用旧的名称。

最初,RS-232C串行接口的设计目的是用于连接调制解调器。目前,RS-232C已成为数据终端设备 DTE (例如计算机) 与数据通信设备 DCE (例如调制解调器) 的标准接口。利用 RS-233C接口不仅可以实现远距离通信,也可以近距离连接两台微机或电子设备。

1. RS-232C的引脚定义

RS-232C接口标准使用标准的 25针 D型连接器即 DB-25。表 10.1罗列了它的引脚排列和名称。PC已使用 9针连接器取代 25针连接器,因此表 10.1中也给出了 9针连接器的引脚。图 10.6为 25针连接器和 9针连接器。

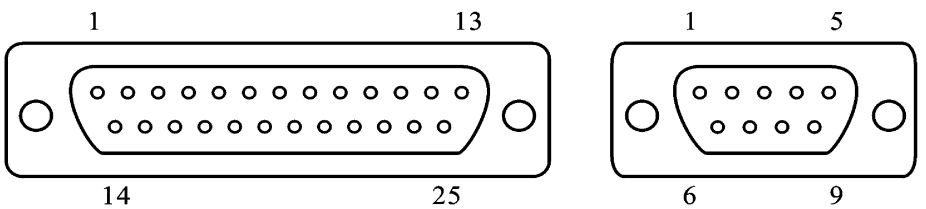


图 10.6 DB-25连接器和 DB-9连接器

表 10.1 RS－232C的引脚

9针连接器引脚号	25针连接器引脚号	名　　称	25针连接器引脚号	名　　称
	1	保护地	12	次信道载波检测
3	2	发送数据 TxD	13	次信道清除发送
2	3	接收数据 RxD	14	次信道发送数据
7	4	请求发送 RTS	16	次信道接收数据
8	5	清除发送 CTS	19	次信道请求发送
6	6	数据装置准备好 DSR	21	信号质量检测
5	7	信号地 GND	23	数据信号速率选择
1	8	载波检测 CD	24	终端发生器时钟
4	20	数据终端准备好 DTR	9、10	保留
9	22	振铃提示 RI	11	未定义
	15	发送时钟 TxC	18	未定义
	17	接收时钟 RxC	25	未定义

RS－232C接口包括两个信道：主信道和次信道。次信道为辅助串行通道提供数据控制和通道，但其传输速率比主信道要低得多，其他跟主信道相同，通常较少使用。

TxD发送数据——串行数据的发送端。

RxD接收数据——串行数据的接收端。

RTS请求发送——当数据终端设备准备好送出数据时，就发出有效的 RTS信号，用于通知数据通信设备准备接收数据。

CTS清除发送——当数据通信设备已准备好接收数据终端设备的传送数据时，发出 CTS有效信号来响应 RTS信号，其实质是允许发送。

RTS和 CTS是数据终端设备与数据通信设备间一对用于数据发送的联络信号。

DTR数据终端准备好——通常当数据终端设备一加电，该信号就有效，表明数据终端设备准备就绪。

DSR数据装置准备好——通常表示数据通信设备（即数据装置）已接通电源连到通信线路上，并处在数据传输方式，而不是处于测试方式或断开状态。

DTR和 DSR也可用做数据终端设备与数据通信设备问的联络信号，例如，

应答数据接收。

GND信号地——为所有的信号提供一个公共的参考电平。

CD载波检测——当本地调制解调器接收到来自对方的载波信号时,就从该引脚向数据终端设备提供有效信号。该引脚缩写为 DCD。

RI振铃指示——当调制解调器接收到对方的拨号信号期间,该引脚信号作为电话铃响的指示,保持有效。

保护地 机壳地)——这是一个起屏蔽保护作用的接地端,一般应参照设备的使用规定,连接到设备的外壳或机架上,必要时要连接到大地。

TxC发送器时钟——控制数据终端发送串行数据的时钟信号。

RxC接收器时钟——控制数据终端接收串行数据的时钟信号。

2. RS-232C的连接

图 10.7是数字终端设备 (例如微机)利用 RS-232C接口连接调制解调器的示意图,用于实现通过电话线路的远距离通信。实际上,数据终端设备与数据通信设备通过 RS-232C接口就是对应引脚直接相连。图中只使用 9个常用信号,并给出了 DB25连接器的引脚号。

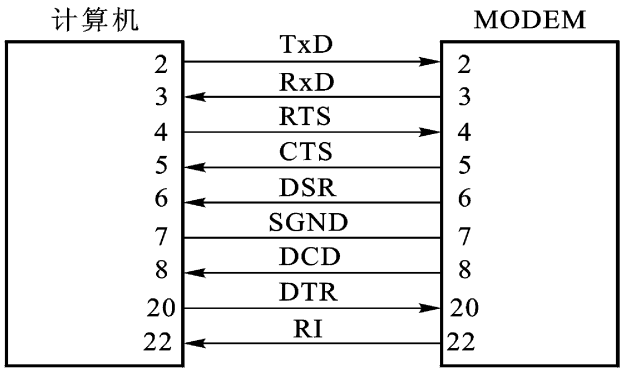


图 10.7　计算机由 RS-232C接口连接调制解调器

图 10.8是两台微机直接利用 RS-232C接口进行短距离通信的连接示意图。由于这种连接不使用调制解调器,所以被称为零调制解调器 (Null Modem)连接。

图 10.8a是不使用联络信号的 3线相连方式。很明显,为了交换信息,TxD和 RxD应当交叉连接。程序中不必使 RTS和 DTR有效,也不应检测 CTS和 DSR是否有效。

图 10.8b是“伪”使用联络信号的 3线相连方式,是常用的一种方法。图中

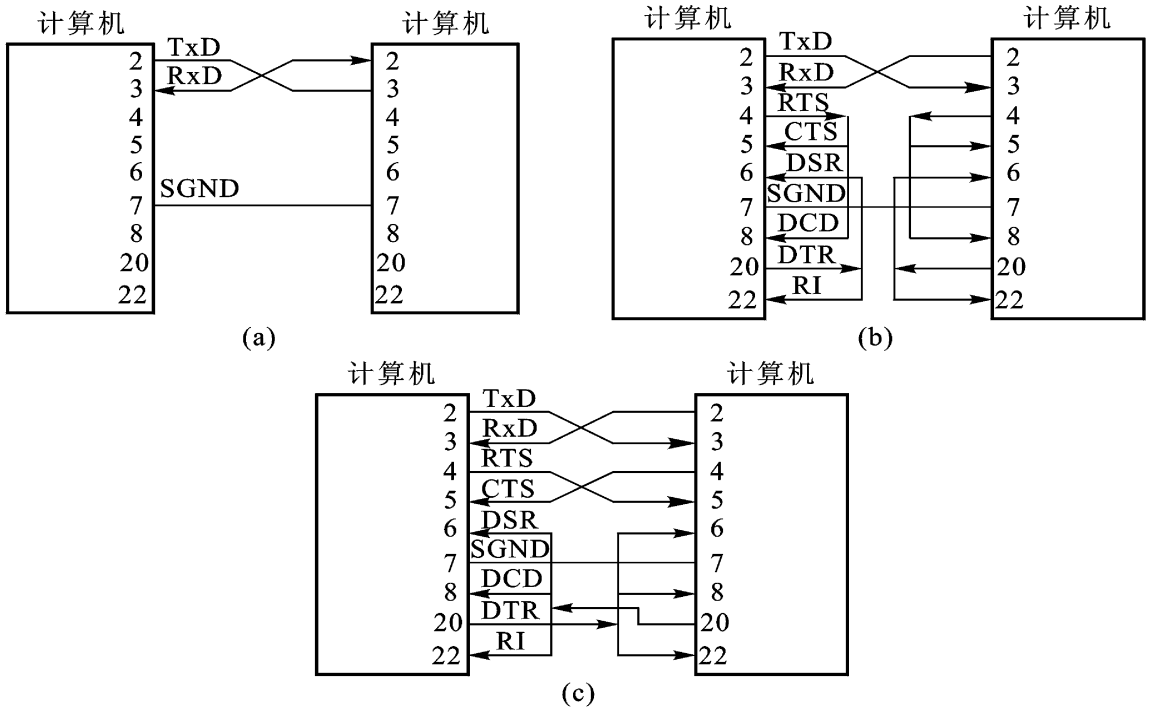


图 10.8 两台微机直接利用 RS－232C接口进行短距离通信

双方的 RTS和 CTS各自互接,用请求发送 RTS信号来产生允许发送 CTS,表明请求传送总是允许的。同样,DTR和 DSR互接,用数据终端准备好产生数据装置准备好。这样的连接可以满足通信的联络控制要求。

由于通信双方并未进行联络应答,所以采用图 10.8a和图 10.8b的连接方式,应注意传输的可靠性。发送方无法知道接收方是否可以接收数据、是否接收到了数据。传输的可靠性需要利用软件来保证,例如程序中先发送一个字符,等待接收方确认之后 回送一个响应字符)再发送下一个字符。

图 10.8c是使用联络信号的多线相连方式。这种连接方式通信比较可靠,但所用连线较多。

IBM PC系统 ROM—BIOS的异步通信 I/O功能调用 INT 14H支持图 10.8b和图 10.8c的连接方式。

3. RS－232C的电气特征

RS－232C接口标准采用 EIA电平。它规定:高电平为 +3～+15 V,低电平为 －3～－15 V。实际应用中常采用 ±12V 或 ±15V。 RS－232C可承受 ±25 V的信号电压。另外,要注意 RS－232C数据线 TxD和 RxD使用负逻辑,即高电平表示逻辑 0,用符号 SPACE (空号)表示;低电平表示逻辑 1,用符号 MARK (传号)表示。联络信号线为正逻辑,高电平有效,为 ON状态;低电平无效,为 OFF

状态。

由于 RS－232C的 EIA电平与微机的逻辑电平（TTL电平或 CMOS电平）不兼容，所以两者间需要进行电平转换。传统的转换器件有 MC1488（完成 TTL电平到 EIA电平的转换）和 MC1489（完成 EIA电平到 TTL电平的转换）等芯片。

目前已有更为方便的电平转换芯片，例如 MAX232、UN232等。MAX232电平转换电路如图 10.9所示，其外围元件很少，一块芯片就能实现两路 TTL电平到 EIA电平、两路 EIA电平到 TTL电平的转换。

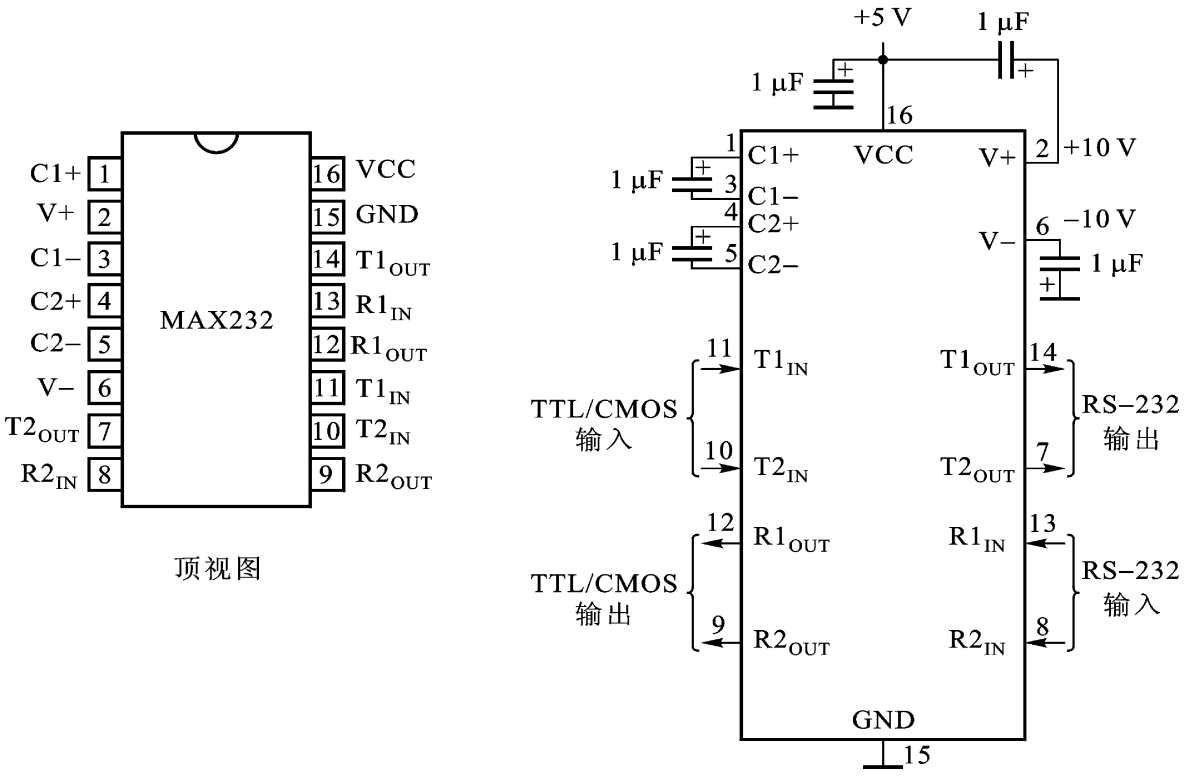


图 10.9　MAX232的封装和应用电路图

10.2　通用可编程串行通信接口芯片 NS8250

计算机进行串行通信，需要并行到串行和串行到并行的转换，并需按照传输协议发送和接收每个字符（或数据块）。这些工作可以由软件实现，也可以由硬件电路实现。通用异步接收发送器（UART, Universal Asynchronous Receiver/Transmitter）就是串行异步通信的接口电路芯片。IBM PC/XT使用的 UART芯

片为 NS 8250,后来使用NS 16550。现在的 32位 PC芯片组中使用了与NS 16550兼容的逻辑电路。NS 8250支持的数据传输速率为 50～9 600 bps;NS 16550支持的速率高达 115 200 bps。除支持的速率不同外,NS 8250和NS 16550完全兼容。

10.2.1 NS 8250概述

1. NS 8250的基本功能

8250支持串行异步通信协议,支持全双工通信。通信字符可选择数据位为5～8位,停止位可选择 1、1.5或 2位,可进行奇偶校验,具有奇偶、帧和溢出错误的检测。具有带优先级排序的中断系统,有多种中断源。发送和接收均采用双缓冲器结构。使用单一的 5 V电源,40脚双列直插型封装。

2. NS 8250的结构

8250的内部结构如图 10.10所示。发送器具有发送保持寄存器、发送移位寄存器组成的双缓冲结构,实现由并行数据到串行数据的转换。接收器具有接收缓冲寄存器、接收移位寄存器组成的双缓冲结构,它将接收的串行数据转换为并行数据。波特率发生器为发送器和接收器提供所需的同步控制时钟信号。调制解调器控制逻辑实现与调制解调器连接,中断控制逻辑实现中断控制和优先权判断,数据缓冲器和选择控制逻辑实现与 CPU的接口。

⑴ 串行数据的发送

当发送数据时,8250接受 CPU送来的并行数据,存在发送保持寄存器中。只要发送移位寄存器没有正在发送的数据,发送保持寄存器的数据就进入发送移位寄存器。与此同时,8250按照编程规定的起止式字符格式,加入起始位、奇偶校验位和停止位,从串行数据输出引脚 SOUT逐位输出。每位的时间长度由传输速率确定。另外,8250能发送中止字符 (输出连续的低电平,以通知对方中止通信)。

因为采用双缓冲寄存器结构,所以在发送移位寄存器进行串行发送的同时,CPU可以向 8250提供下一个发送数据,这样可以保证数据的连续发送。

⑵ 起始位的检测

8250需要首先确定起始位才能开始接收数据,这就是实现位同步。8250的数据接收时钟 RCLK使用 16倍波特率的时钟信号。接收器用 RCLK检测到串行数据输入引脚 SIN由高电平变低后,连续测试 8个 RCLK时钟周期,若采样到的都是低电平,则确认为起始位;若低电平的保持时间不是 8个 RCLK时钟周期,则认为是传输线上的干扰。在确认了起始位后,每隔 16个 RCLK时钟周期对 SIN输入的

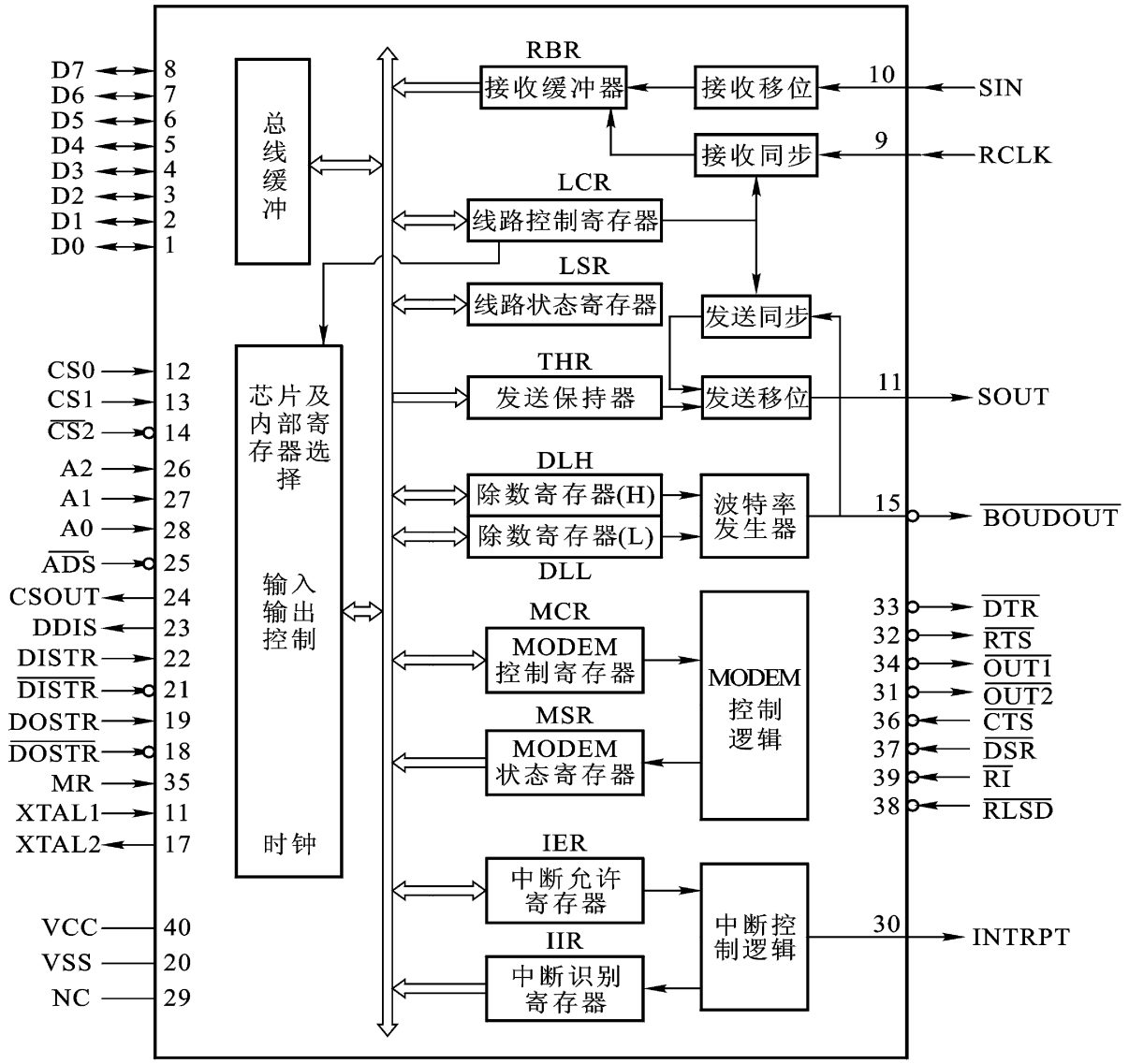


图 10.10 NS8250的内部结构和引脚信号

数据位进行采样一次,直至规定的数据格式结束,如图 10.11所示。

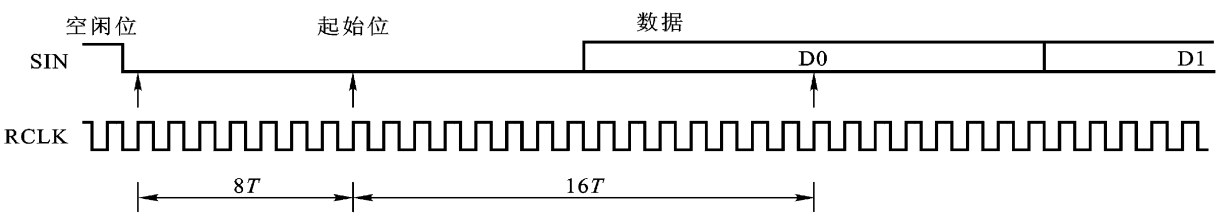


图 10.11 起始位的检测

⑶ 串行数据的接收

当接收数据时，8250的接收移位寄存器对 SIN引脚输入的串行数据进行移位接收。8250按照通信协议规定的字符格式自动删除起始位、奇偶校验位和停止位，把移位输入的串行数据转换成并行数据。接收完一个字符后，把数据送入接收缓冲寄存器。接收器在接收数据的同时，还对接收数据的正确性和接收过程进行监视。如果发现出现奇偶校验错、帧错、溢出错或接收到中止符，则在状态寄存器中置相应位，并通过中断控制逻辑请求中断，要求 CPU处理。

因为采用双缓冲寄存器结构，所以在 CPU读取接收数据的同时，8250就可以继续串行接收下一个数据，这样可以保证数据的连续接收。

⑷ 接收错误的处理

为了使传输过程更可靠，8250在接收端设立了三种出错标志：

1) 奇偶错误 PE (Parity Error) —— 若接收到的字符“1”的个数不符合奇偶校验要求，则置这个标志，发出奇偶校验出错信息。

2) 帧错误 FE (Frame Error) —— 若接收到的字符格式不符合规定 (如缺少停止位)，则置这个标志，发出帧错误信息。

3) 溢出错误 OE (Over Error) —— 若接收移位寄存器接收到一个数据，在把它送至输入缓冲器时，CPU还未取走前一个数据，就会出现数据丢失，这时置溢出错误标志。由此还可以看出，若设计较多级数的接收缓冲器，则溢出错误的概率就少。

3. 8250引脚说明

8250的外部引脚可以分成连接 CPU的部分和连接外设的部分。这里的外设是通过 RS－232C接口的。下面分别说明。

⑴ 处理器接口引脚

数据线 D_0 ～D_7 ——用于在 CPU与 8250之间交换信息。

地址线 A_0 ～A_2 ——用于寻址 8250内部寄存器，参见表 10.2左边 3列。

表 10.2 8250的寄存器寻址

DLAB	$A_2A_1A_0$	寄存器	COM1地址	COM2地址
0	000	读接收缓冲寄存器	3F8H	2F8H
0	000	写发送保持寄存器	3F8H	2F8H
×	001	中断允许寄存器	3F9H	2F9H
×	010	中断识别寄存器 (只读)	3FAH	2FAH

续表

DLAB	$A_2A_1A_0$	寄存器	COM1地址	COM2地址
×	011	通信线路控制寄存器	3FBH	2FBH
×	100	调制解调器控制寄存器	3FCH	2FCH
×	101	通信线路状态寄存器	3FDH	2FDH
×	110	调制解调器状态寄存器	3FEH	2FEH
×	111	不用	3FFH	2FFH
1	000	除数寄存器低 8位	3F8H	2F8H
1	001	除数寄存器高 8位	3F9H	2F9H

片选线—— 8250设计了 3个片选输入信号 CS0、CS1、$\overline{CS2}$和一个片选输出信号 CSOUT。3个片选输入都有效时,才选中 8250芯片,同时 CSOUT输出高电平有效。

地址选通信号 $\overline{ADS}$——当该信号低电平有效时,锁存上述地址线和片选线的输入状态,保证读写期间的地址稳定。若不会出现地址不稳定现象,则不必锁存,只将 $\overline{ADS}$引脚接地。

读控制线—— 8250被选中时,只要数据输入选通 DISTR (高有效)和 $\overline{DISTR}$ (低有效)引脚有一个信号有效,CPU就从被选择的内部寄存器中读出数据。相当于 I/O读信号。

写控制线—— 8250被选中时,只要数据输出选通 DOSTR (高有效)和 $\overline{DOSTR}$ (低有效)引脚有一个信号有效,CPU就将数据写入 8250被选择的内部寄存器。相当于 I/O写信号。

8250读写控制信号有两对,每对信号作用完全相同,但有效电平不同。

驱动器禁止信号 DDIS—— CPU从 8250读取数据时,DDIS引脚输出低电平,用来禁止外部收发器对系统总线的驱动;其他时间,DDIS为高电平。

主复位线 MR—— 该引脚输入高电平有效时,8250复位,控制部分寄存器和输出信号处于初始化状态,参见表 10.3。这个引脚就是 8250的硬件复位信号。

表 10.3 8250的复位状态

寄存器/信号	复位状态
中断允许寄存器	所有位为低
中断识别寄存器	D_0为高,其他位为低
通信线路控制寄存器	所有位为低
调制解调器控制寄存器	所有位为低
通信线路状态寄存器	D_5和 D_6为低,其他为高
调制解调器状态寄存器	D_0～D_3为低,D_4～D_7为输入信号
引脚 INTRPT	低电平
引脚 SOUT、RTS和 DTR	高电平
引脚$\overline{OUT1}$、$\overline{OUT2}$	高电平

⑵ 时钟信号

外部晶体振荡器电路产生的时钟信号送到时钟输入引脚 XTAL1,作为 8250的基准工作时钟。时钟输出引脚 XTAL2是基准时钟信号的输出端,可用做其他功能的定时控制。外部输入的基准时钟,经 8250内部波特率发生器分频后产生发送时钟,并经波特率输出引脚$\overline{BAUDOUT}$输出。接收时钟引脚 RCLK可接收外部提供的接收时钟信号,若采用发送时钟作为接收时钟,则只要将 RCLK引脚和$\overline{BAUDOUT}$引脚直接相连即可。

⑶ 串行异步接口引脚

这是一组用于实现 RS-232C接口的信号线。它们是 TTL电平,输出数据线为正逻辑,联络控制信号线为低电平有效。

串行数据输入线 SIN对应 RxD,用于接收串行数据。串行数据输出线 SOUT对应 TxD,用于发送串行数据。

调制解调器控制线包括数据终端准备好$\overline{DTR}$、数据设备准备好$\overline{DSR}$、发送请求$\overline{RTS}$、清除发送$\overline{CTS}$、接收线路检测$\overline{RLSD}$(对应载波检测 CD)和振铃指示$\overline{RI}$。

⑷ 输出线

$\overline{OUT1}$和$\overline{OUT2}$由调制解调器控制寄存器 MCR的 D_2和 D_3使其输出低电平有效,复位时为高电平。

⑸ 中断请求信号线 INTRPT

8250内部有 4种类型的中断,若 8250的中断是允许的,则其中任意一个中断源有中断请求,INTRPT输出位高电平。

10.2.2　NS 8250的寄存器

8250内部有 9种可访问的寄存器、用引脚 A_0 ～A_2来寻址;同时还要利用通信线路控制寄存器的最高位,即除数寄存器访问位 DLAB,以区别共用两个端口地址的不同寄存器 (参见表 10.2)。

1. 接收缓冲寄存器 RBR

接收缓冲寄存器存放串行接收后获得的并行数据。

2. 发送保持寄存器 THR

发送保持寄存器存放将要串行发送的并行数据。

3. 除数寄存器

8250的接收器时钟和发送器时钟由时钟输入引脚的基准时钟分频得到,而且是传输率的 16倍。不同的数据传输率,需要不同的分频系数,除数寄存器就保存设定的分频系数。计算分频系数 (即除数)的公式可以表达如下:

分频系数 =基准时钟频率 /(16×波特率)

除数寄存器是 16位的,写入前注意使 DLAB=1。

4. 通信线路控制寄存器 LCR

通信线路控制寄存器指定串行异步通信的字符格式,即数据位个数、停止位个数,是否进行奇偶校验以及何种校验。LCR可以写入,也可以读出,其格式见图 10.12。其中 D6=1将迫使 8250发送连续低电平的中止字符。最高位 D_7是 DLAB,为 1说明寻址除数寄存器;否则为寻址数据寄存器和中断允许寄存器。

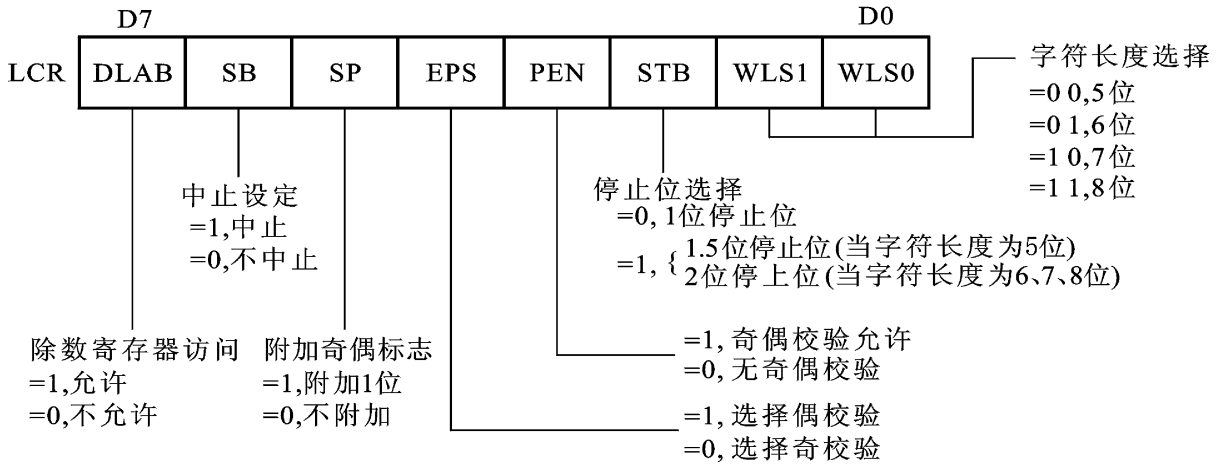


图 10.12　LCR的格式

5. 通信线路状态寄存器 LSR

通信线路状态寄存器提供串行异步通信的当前状态,供 CPU读取和处理。LSR还可以写入 (除 D_0位),人为地设置某些状态,用于系统自检,其格式见图 10.13。

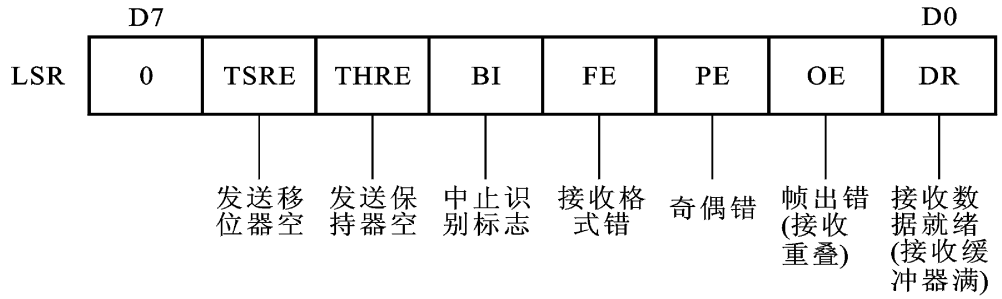


图 10.13 LSR的格式

LSR能反映接收数据是否准备就绪和发送保持寄存器是否为空,以决定CPU的下一个读写操作。当接收数据就绪或发送保持寄存器为空时,除使 LSR相应位置位外,还可以通过中断控制电路发出中断请求。LSR也反映接收数据后是否发生错误以及是哪种错误。当错误发生时,也可以产生中断请求。

6. 调制解调器控制寄存器 MCR

调制解调器控制寄存器用来设置 8250与数据通信设备 (例如调制解调器)之间联络应答的输出信号,其格式如图 10.14所示。

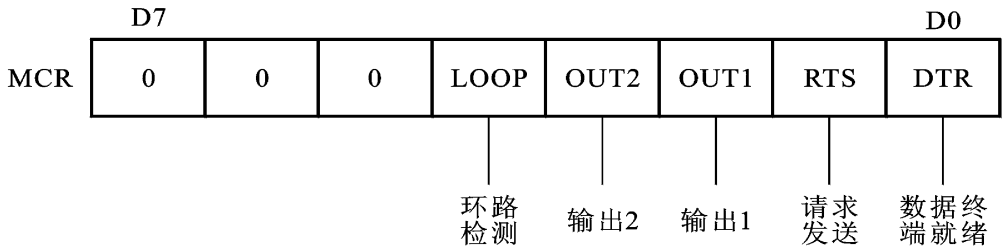


图 10.14 MCR的格式

MCR的 D_2 D_3 位分别控制 $\overline{OUT1}$和 $\overline{OUT2}$的输出,可作为一般的输出信号使用。$\overline{OUT2}$还具有中断控制作用,若 $\overline{OUT2}$输出低电平,允许 8250的 INTRPT发出中断请求信号,否则将屏蔽 8250的中断请求信号。因此 MCR的 D_3位可当作为8250的中断允许控制位。

MCR的 D_4位可控制 8250处于自测试工作状态。在自测试状态,引脚 SOUT变为高,而 SIN与系统分离,发送移位寄存器的数据回送到接收移位寄存器;4

个控制输入信号（$\overline{CTS}$、$\overline{DSR}$、$\overline{RLSD}$及$\overline{RI}$）和系统分离，并在芯片内部与 4个控制输出信号（$\overline{RTS}$、$\overline{DTR}$、$\overline{OUT1}$及$\overline{OUT2}$）相连。这样，发送的串行数据立即在内部被接收（循环反馈），故可用来检测 8250发送和接收功能正确与否，而不必外连线。

在自测试状态，有关接收器和发送器的中断仍起作用，调制解调器产生的中断也起作用。但调制解调器产生中断的源不是原来的 4个控制输入信号，而变成内部连接的 4个控制输出信号，即 MCR低 4位。中断是否允许，则仍由中断允许寄存器控制，若中断是允许的，则将 MCR低 4位的某一位置位，产生相应的中断，好像正常工作一样。

7. 调制解调器状态寄存器 MSR

调制解调器状态寄存器反映 4个控制输入信号的当前状态及其变化，如图 10.15所示。MSR高 4位中某位为 1，说明相应输入信号当前为低有效，否则为高电平有效。MSR低 4位中某位为 1，说明从上次 CPU读取该状态字后，相应输入信号已发生改变，从高变低或反之。MCR低 4位任一位置 1，均产生调制解调器状态中断，当 CPU读取该寄存器或复位后，低 4位被清零。

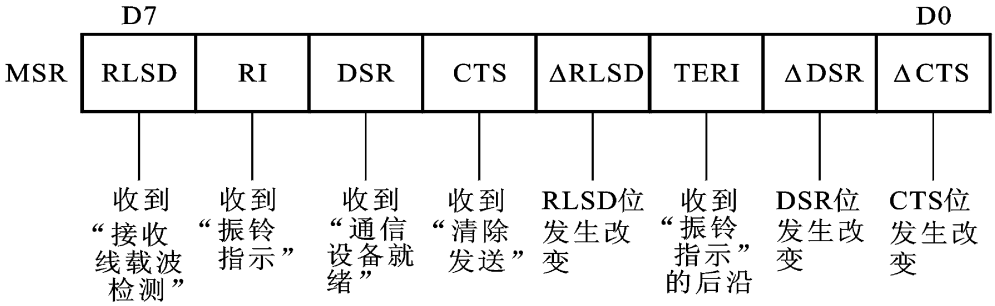


图 10.15　状态寄存器 MSR

8. 中断允许寄存器 IER

8250具有很强的中断控制和优先权判决处理能力，设计有两个中断寄存器和 4级中断。这 4级中断按优先权从高到低排列的顺序为：接收线路状态中断（包括奇偶错、溢出错、帧错和中止字符）、接收器数据准备好中断、发送保持寄存器空中断和调制解调器状态中断（包括清除发送$\overline{CTS}$状态改变、数据终端准备好$\overline{DSR}$状态改变、振铃$\overline{RI}$接通变成断开和接收线路信号检测$\overline{RLSD}$状态改变）。8250的 4级中断的优先权，是按照串行通信过程中事件的紧迫程度安排的，是固定不变的，用户可利用中断允许或禁止进行控制。

中断允许寄存器的低 4位控制 8250的 4级中断是否被允许。某位为 1，则对应的中断被允许（否则被屏蔽（禁止），如图 10.16所示。如果 IER低 4位全

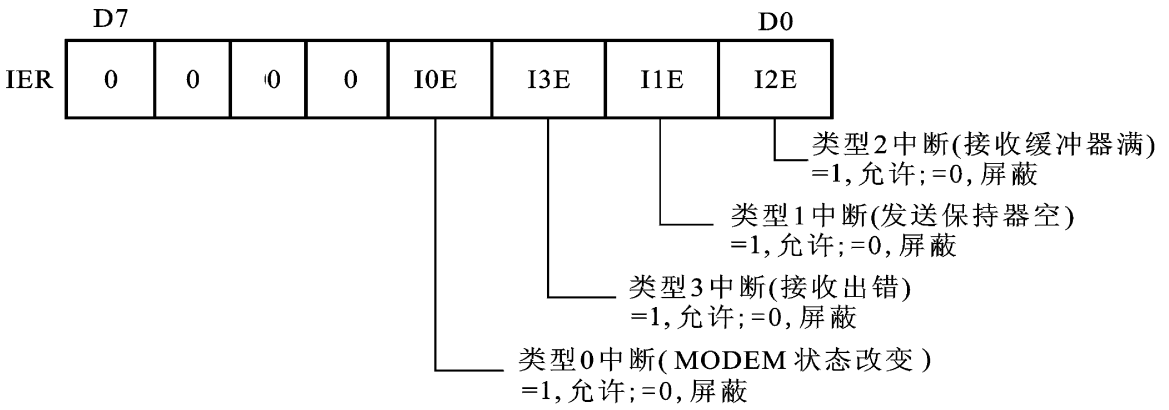


图 10.16 IER的格式

为 0,则禁止 8250产生中断,此时还将禁止中断识别寄存器和中断请求信号的输出。

9. 中断识别寄存器 IIR

8250的 4级中断中有一级或多级出现时,8250便输出高电平的 INTRPT中断请求信号。为了能具体识别是哪一级中断引起的请求,以便分别进行处理,8250内部设有一个中断识别寄存器。它保持正在请求中断的优先权最高的中断级别编码,在这个特定的中断请求由 CPU进行服务之前,不接受其他的中断请求。

中断识别寄存器的格式见图 10.17。其中,最低位 D0反映是否有中断请求,IP=0表示有待处理的中断,IP=1表示没有待处理的中断;D_2 D_1位则表示正在请求的最高优先权的中断。

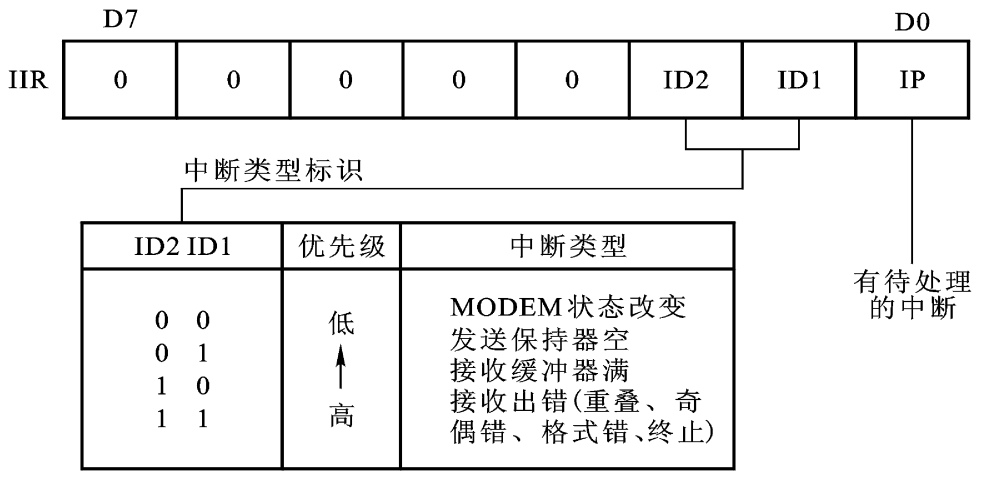


图 10.17 中断识别寄存器 IIR的格式

IIR是只读寄存器,它的低 3位随中断源而变化,但 IIR高 5位总是 0,可用作判别 8250是否存在的特征。

10.2.3 IBM PC/XT的串行异步通信适配器

IBM PC/XT机的串行异步通信适配器使用 NS 8250,还包括 TTL电平与 EIA电平转换电路等。后来 PC的串行异步通信硬件电路有很大的改变,但在软件上保持了兼容性。

1. 异步通信适配器的接口电路

考虑到允许在系统中插入两块串行异步通信适配器,两块适配器的地址、中断请求线应能按需要设定。若使异步通信适配器上跨接器的 J12和 J11分别接通,则 8250的端口地址为 3F8~3FFH(成为第 1个串行通信接口 COM1),以 IRQ4作为本适配器的中断请求线;若使异步通信适配器上跨接器的 J9和 J10分别接通,则 8250的端口地址为 2F8~2FFH(成为第 2个串行通信接口 COM2),以 IRQ3为中断请求线。参见表 10.2。8250的 A_0~A_2引脚连至系统地址总线的低 3位,以选择 8250的内部寄存器。

8250的数据输入选通与 I/O读连接,数据输出选通与 I/O写连接,以控制 8250的读写操作。系统的复位信号 RESET控制 8250的复位 MR引脚。

异步通信适配器上的晶体振荡器向 8250提供 1.843 2 MHz的基准时钟信号(XTAL)。串行接收和发送使用相同的传输率,所以 $\overline{\text{BAUDOUT}}$与 RCLK相连。用户可定义的引脚 $\overline{\text{OUT1}}$在 PC中未用。$\overline{\text{OUT2}}$用于控制 INTRPT的三态输出,可作为 8250的中断请求允许位。异步通信适配器既可工作于查询方式又可工作于中断方式。

8250与 RS-232C接口之间,使用 SN75150芯片进行 TTL电平到 EIA电平的转换,使用 SN75154进行 EIA电平到 TTL电平的转换。

2. 异步通信适配器的初始化编程

为了使串行通信接口做好准备,先要进行异步通信适配器的初始化编程,即对 8250的内部控制寄存器进行写入。以 PC的 COM1初始化编程为例,说明初始化编程的过程。

(1) 设置波特率——写入除数寄存器

根据通信双方约定的传输率和基准时钟频率,确定分频系数。为了寻址除数寄存器,必须先使通信线路控制寄存器的最高位 DLAB置 1。假设采用 1 200 bps。

```
MOV     AL,80H          ;最高位 DLAB=1
```

```
MOV     DX,3FBH       ;COM1通信线路控制寄存器的地址为 3FBH
OUT     DX,AL         ;写入通信线路控制寄存器,使 DLAB=1
MOV     AX,0096       ;分频系数:1.8432MHz/(1200×16)=96=60H
MOV     DX,3F8H       ;除数寄存器低 8位的地址为 3F8H
OUT     DX,AL         ;写入除数寄存器低 8位
MOV     AL,AH
INC     DX            ;除数寄存器高 8位的地址为 3F9H
OUT     DX,AL         ;写入除数寄存器高 8位
```

⑵ 设置通信字格式——写入通信线路控制寄存器

设数据格式为一个起始位、7个数据位、一个停止位、一个奇校验。程序段如下：

```
MOV     AL,00001010B      ;DLAB=0
MOV     DX,3FBH
OUT     DX,AL             ;写入通信线路控制寄存器
```

这段程序同时使 DLAB=0,以方便后面的初始化程序。

⑶ 设置工作方式——写入调制解调器控制寄存器

通过调制解调器控制寄存器的 D_3 位控制 $\overline{OUT2}$,可选择允许中断或禁止中断 对应通信过程采用中断或查询工作方式。但不论在何种工作方式,调制解调器控制寄存器的最低两位通常都置为 1,这样就建立数据终端准备好 $\overline{DTR}$和请求发送 $\overline{RTS}$的有效信号,即使系统中没有使用调制解调器,也无妨这样设置。

① 设置查询通信方式：

```
MOV   AL,03h        控制OUT2为高,DTR和RTS为低
MOV   DX,3FCH
OUT   DX,AL         ;写入调制解调控制寄存器
```

② 设置中断通信方式

```
MOV   AL,0BH        控制OUT2为低,允许 INTRPT产生请求
MOV   DX,3FCH
OUT   DX,AL         ;写入调制解调控制寄存器
```

③ 设置查询的循环测试通信方式

```
MOV   AL,13H        循环测试位设置为 1
MOV   DX,3FCH
```

```
OUT   DX, AL
```

8250采用循环自测试通信方式，$\overline{OUT2}$不再输出低电平有效信号。在异步通信适配器上，它就无法允许中断请求信号 INTRPT，所以不能采用中断的循环测试通信方式。

(4) 设置中断允许/中断屏蔽——写入中断允许寄存器

如果不采用中断工作方式，应设置中断允许寄存器为 0，禁止所有的中断请求。否则，根据需要，允许相应级别的中断，不使用的中断则仍屏蔽。例如：

```
MOV   AL, 0          ;禁止所有中断
MOV   DX, 3F9H
OUT   DX, AL         ;写入中断允许寄存器 (应保证此时 DLAB＝0)
```

10.2.4　8250的应用举例

1. 异步通信程序

【例题 10.3】　下面的例子实现两台 PC之间的异步串行通信，从一台 PC键盘输入的字符将在对方 PC屏幕上显示出来。每台 PC都使用 COM2口，使用相同的程序，程序采用查询工作的方式。

初始化编程时，应将 03H写入调制解调器控制寄存器 MCR，使环路检测位为 0。初始化编程后，程序读取 8250的通信状态寄存器，若数据传输出错就显示一个问号 “?” 若接收到对方送来的字符就将其显示在屏幕上；若从本机键盘输入字符，就将其发送给对方。如果按下 Esc键就返回 DOS。

本例程序不使用联络控制信号，通信时不关心调制解调器状态寄存器的内容，而只要查询通信线路状态寄存器即可。

```
START:  MOV   AL, 80H     ;异步通信适配器的初始化编程
        MOV   DX, 2FBH    ;波特率
        OUT   DX, AL
        MOV   AX, 0096
        MOV   DX, 2F8H
        OUT   DX, AL
        MOV   AL, AH
        INC   DX
        OUT   DX, AL
        MOV   AL, 0AH     ;通信字格式
```

```
        MOV     DX,2FBH
        OUT     DX,AL
        MOV     AL,03H      ;设置工作方式
        MOV     DX,2FCH
        OUT     DX,AL

STATUS: MOV     DX,2FDH
        IN      AL,DX       ;读 COM2的通信线路状态寄存器
        TEST    AL,1EH      ;接收有错误否？
        JNZ     ERROR       ;有错,则转错误处理
        TEST    AL,01H      ;接收到数据吗？
        JNZ     RECEIVE     ;有,则转接收处理
        TEST    AL,20H      ;发送保持寄存器 THR空 能输出数据）
                            吗？
        JZ      STATUS      ;不能输出,则循环查询通信线路状态
        MOV     AH,0BH      ;能,可以发送数据,检测键盘有无输入
                            字符
        INT     21H
        CMP     AL,0
        JZ      STATUS      ;无输入字符,则循环查询通信线路状态
        MOV     AH,0        ;有输入字符,读取字符。
        INT     16
        CMP     AL,1BH      ;判是否 Esc键
        JZ      DONE        ;是 Esc键,则退出程序返回 DOS
        MOV     DX,2F8H     ;否则,将字符输出给发送保持寄存器
        OUT     DX,AL       ;串行发送数据
        JMP     STATUS      ;继续查询

RECEIRVE:MOV    DX,2F8H     ;已收到字符,读取该字符并显示
        IN      AL,DX       ;从输入缓冲寄存器读取字符
        AND     AL,7FH      ;ASCII码 7个数据位,所以保留低 7位
        PUSH    AX          ;保存数据
        MOV     DL,AL       ;在屏幕上显示该字符
        MOV     AH,2
```

```
            INT     21H
            POP     AX          ;恢复数据
            CMP     AL,0DH      ;数据是回车键吗?
            JNZ     STATUS      ;不是,则循环
            MOV     DL,0AH      ;是,再进行换行
            MOV     AH,2
            INT     21H
            JMP     STATUS      ;继续查询

ERROR:      MOV     DX,2F8H     ;接收有错,显示问号"?"
            IN      AL,DX       ;读出接收有误的数据,丢掉
            MOV     DL,'?'      ;显示问号
            MOV     AH,2
            INT     21H
            JMP     STATUS      ;继续查询
DONE:       ……                  ;返回 DOS
```

【例题 10.4】 PC串行口"自发自收"。例 10.3中的程序稍作修改即可实现该操作。方法是在初始化编程中,向调制解调器控制寄存器 MCR写入 13H,即环路检测位为 1,则 8250工作于循环自测试方式。从键盘输入的字符,经8250发送后又由 8250自身接收。这时,PC后面板串行接口上无需连线,就实现了"自发自收"。上述程序可用于 8250芯片的自检。

以上例题中的异步串行通信程序采用查询工作方式。系统 BIOS的通信I/O功能调用 INT14H也采用了查询方式的实现异步通信功能,可由用户程序调用。

2. 中断通信方式的编程方法

在一些系统中,主程序的主要任务是处理所接收的输入数据、提供发送的输出数据以及其他工作,不可能花费很多时间去周期性检测通信线路状态寄存器。这时采用中断工作方式更为合适。

首先,在主程序中完成有关芯片的初始化编程,允许 8250中断。中断服务程序中,首先要根据中断识别寄存器 IIR的标志位来判别是哪个中断源引起的中断,然后执行相应的处理。在一个中断源处理完后,还应检查中断识别寄存器的最低位,判断是否还有其他未处理的中断。因为在同一时刻可能会出现一个以上的中断,而中断识别寄存器仅按优先权顺序列出较高优先权的中断。当较

高优先权的中断正在被处理时,其他优先仅较低的中断不会再次产生中断请求。

当有一批数据需要输出时,主程序先向 8250的发送保持寄存器 THR送入第一个数据。以后每当由于“发送保持寄存器空”引起中断,就在这次中断服务程序中向发送保持寄存器 THR送入一个数据,直到一批数据发送完毕。

接收数据时,每当由于“接收缓冲器满”引起中断,就在中断服务程序中从接收缓冲器 RBR输入一个数据。

由于 8250的接收缓冲器只有两级 (第一级为接收移位寄存器、第二级为接收缓冲器 RBR),当主程序无法及时取走数据时,应设置接收数据缓冲区。中断服务程序将 RBR中的数据放入接收数据缓冲区,等待主程序处理。同样,由于 8250的发送缓冲器为两级 (发送保持寄存器 THR和发送移位寄存器),当主程序一次需要发送很多数据时,可设置发送数据缓冲区,这样主程序只需将一批数据送入发送数据缓冲区而不必等待。接收数据缓冲区和发送数据缓冲区都采用“先进先出循环队列”实现。

10.3 通用可编程串行通信接口芯片 8251A

Intel 8251A是 Intel公司生产的通用可编程串行通信接口芯片,与 NS8250基本功能是类似的,都可通过编程实现串行接口的基本任务。

10.3.1 8251A的基本功能

1. 8251A是可编程的串行通信接口芯片,能够以同步方式或异步方式进行工作。能自动完成帧格式。

2. 在同步方式中,每个字符可定义为 5、6、7或 8位,可以选择进行奇校验、偶校验或不校验。内部能自动检测同步字符实现内同步或通过外部电路获得外同步,波特率为 0~64 K。

3. 异步方式中,每个字符可定义为 5、6、7或 8位,用 1位作为奇偶校验 (可选择)。时钟速率可用软件定义为通信波特率的 1、16或 64倍。能自动为每个被输出的数据增加 1个起始位,并能根据软件编程为每个输出数据增加 1个、1.5个或 2个停止位。异步方式下,波特率为 0~19.2 K。

4. 8251A能进行出错检测,它具有奇偶、溢出和帧错误等检测电路。用户可通过输入状态寄存器内容进行查询。

5. 具有独立的接收器和发送器,因此,能够以单工、半双工或全双工的方式进行通信。并且提供一些基本控制信号,可以方便地与调制解调器连接。

10.3.2　8251A的结构

1. 8251A 的内部结构

8251A的内部结构如图 10.18所示。8251A由 5个主要部分组成,包括接收器、发送器、调制控制、读写控制和系统数据总线缓冲器。8251A内部由内部数据总线实现相互之间数据传送。

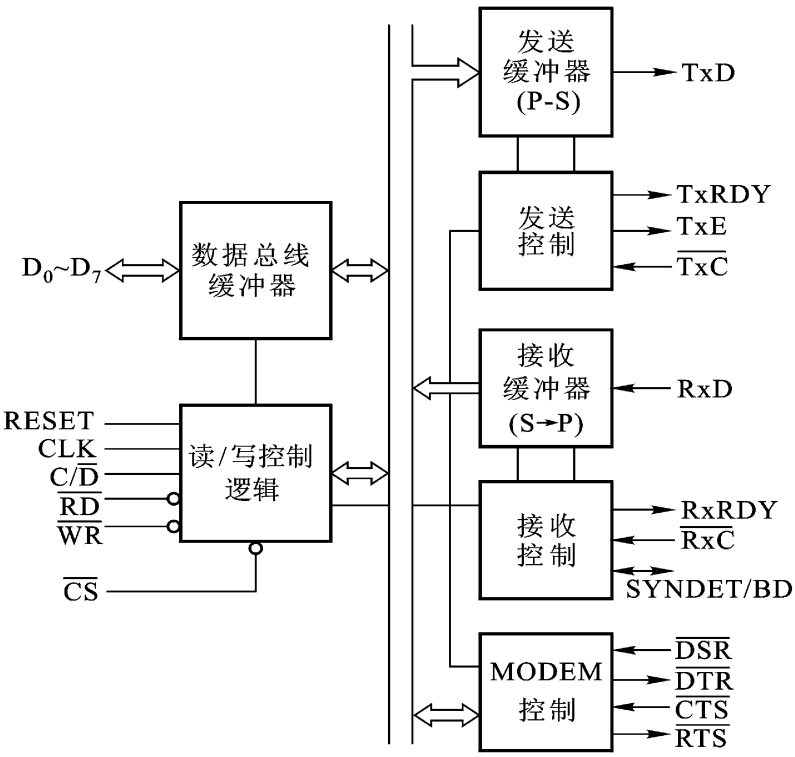


图 10.18　8251A的内部结构

⑴ 数据总线缓冲器

数据总线缓冲器是三态双向 8位缓冲器,它使 8251A与系统总线连接起来。它含有数据缓冲器和命令缓冲器,CPU通过输入/输出指令对它读/写数据,也可以写入控制字和命令字,再由它产生使 8251A完成各种功能的控制信号。另外,执行命令所产生的各种状态信息也是从数据总线缓冲器读出的。

⑵ 接收

接收器的功能是接收在 RxD引脚上的串行数据,并按规定的格式把它转换为并行数据,存放到数据总线缓冲器中。

异步方式:且允许接收和准备好接收数据时,它监视 RxD线。在无字符传

送时，RxD线为高电平。当发现 RxD 线上出现了低电平时，则认为它是一帧信息的起始位，同时启动一个内部计数器，当计数到一个数据位宽度一半时（若时钟脉冲频率为波特率的 16倍时，则为计数到第 8个脉冲），又重新采样 RxD线，若仍为低电平，则确认它就是起始位，而不是噪声信号。此后，每隔一个数据位宽度时间采样一次。RxD 线作为输入信号，送至移位寄行器，经过移位，又经过去掉停止位和奇偶校验后，变成了并行数据，再经过 8251A 内部数据总线送至接收数据缓冲器，同时发出 RxRDY信号，通知 CPU字符已经可以输入了。

鉴于接收器采用上述的方式确定起始位和检测信息，所以在异步串行通信时，大多数可编程串行接口芯片都设计了三种错误类型检测功能。当发送时钟和接收时钟的频率相差太大时，会引起在刚采样几次就造成错位，接收器在收到规定的字符位后，有可能在本应是停止位的数位上出现了低电平，这就是帧格式错 如果接收器对收到的字符位产生的奇偶校验位与收到的奇偶校验位不一致，就是奇偶校验错；当接收器将收到的一个有效字符送至接收数据缓冲器，通知 CPU读取时，若 CPU还未读走，又收到一个有效字符时，就会覆盖掉上一个字符，这就是接收缓冲器溢出错误。不管出现哪种类型的错误，接收器均会将所收到的字符数据送至接收数据缓冲器，同时置相应的错误状态标志位。

同步方式：8251A首先搜索同步字。8251A检测 RxD线，每出现一个数位就把它接收下来，并把它送入移位寄存器移位，接收一个整字符后和同步字符寄存器的内容相比较，若不等，还要重复上述部分接收下一个字符继续和同步字符寄存器的内容相比较；若相等，说明 8251A搜索到了同步字符，此时，8251A 的 SYNDET引脚就升为高电平以示同步已经实现。

有时，8251A采用双同步字符方式。这种情况下，就要测得输入的字符与第一个同步字符寄存器内容相同后，再继续检测此后输入的字符与第二个同步字符寄存器的内容是否相等，如果不等，还要从第一个同步字符寄存器开始比较，若相同，则认为同步已经实现。

在外同步的情况下，和上面过程不同。因为外同步是通过在同步输入引脚 SYNDET加一个高电位实现同步的。SYNDET引脚一出现高电平，8251A就会立即脱离对同步字符的搜索过程，只要此高电平能维持一个接收时钟周期，8251A便认为已经实现同步了。

8251A实现同步后，利用时钟采样和移位从 RxD线上接收来的数据位，且按规定的位数，把它送至接收数据输入缓冲寄存器，同时发出 RxRDY准备好信号。

⑶ 发送器

发送器接收 CPU输出的并行数据，通过移位寄存器，串行从 TxD引脚输出。

在异步方式时,发送器为每一个字符自动加上 1个起始位,并且按照编程要求加上奇偶校验位以及 1个、1.5个或者 2个停止位。数据及起始位、校验位、停止位总是在发送时钟 TxC的下降沿时,从 8251A发出,数据传输的波特率可以为发送时钟频率的 1、1/16或者 1/64,具体取决于编程时,选择方式选择字中的波特率系数。

在同步发送方式下,发送器在准备发送的数据前面插入由初始化程序设定的一个或两个同步字符,而在数据中,根据编程设定可插入奇偶校验位。然后,在发送时钟的作用下,以时钟相同的频率将数据一位一位地由 TxD引脚发送出去。当 8251A正在发送数据,而 CPU却来不及提供新数据时,8251A发送器会自动插入同步字符在 TxD引脚发出,因为在同步方式时被传送的字符间是不允许存在间隙的。

不论在同步或异步工作方式,只有当程序设置了 TxEN允许发送和 $\overline{CTS}$对调制器发出请求发送的响应信号有效时,才能发送。

⑷ 调制控制和读写控制

调制控制实现对 MODEM的控制,读写控制对 CPU有关控制信号进行译码,用来控制整个 8251A芯片的工作过程,以完成对数据、状态信息和控制信息的传输。

2. 8251A的芯片引脚

8251A是用来作为 CPU与外设或 MODEM之间的接口的,其引脚如图 10.19所示。除了地线和电源引脚外,其余的引脚可分为两类,一类是 8251A和 CPU之间的信号,另一类是 8251A和外部设备或调制解调器之间的信号。

⑴ 8251A与外部装置相连的引脚

1) TxD:发送数据信号端。CPU送往 8251A的并行数据,在 8251A内部转变为串行数据后,通过 TxD端输出。

2) RxD:接收数据信号端。RxD用来接收外部装置通过传输线送来的串行数据,数据进入 8251A后被变换成并行数,等待 CPU输入。

3) $\overline{DTR}$:数据终端准备好信号,低电平有效。是由 8251A送出一个通用输出信号,它能由初始化 8251A编程的命

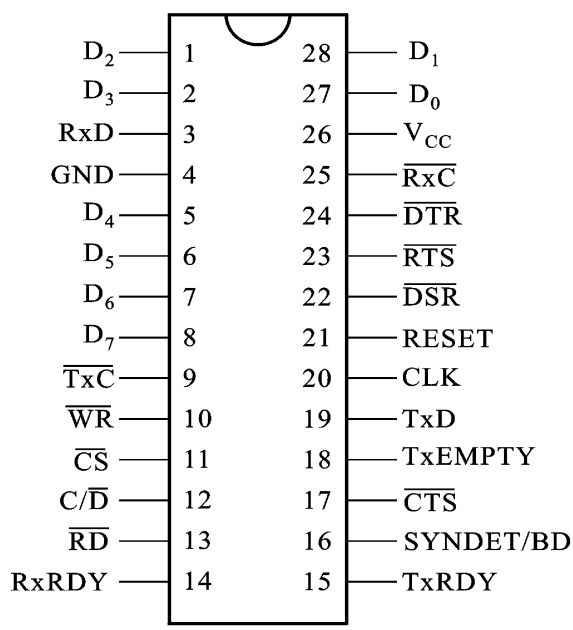


图 10.19　8251A引脚

令指令的 D_1置“1”变为有效，用以表示 CPU准备就绪。

4) $\overline{DSR}$：数据装置准备好信号，低电平有效。这是一个通用的输入信号，用以表示 MODEM或外设已经准备好。 CPU可通过读入状态操作，在状态寄存器的 D_7检测这个信号。一般情况下$\overline{DTR}$和$\overline{DSR}$是一组信号，用于接收器。

5) $\overline{RTS}$:请求发送信号，低电平有效。由 8251A发送给调制解调器或外设的，CPU可以通过编程使命令指令的 D_5置“1”，使其有效，以表示 CPU已经准备好发送。

6) $\overline{CTS}$:允许发送信号，低电平有效。这是调制器或外设对$\overline{RTS}$的响应信号，当其有效时，8251A才能执行发送操作。

以上 4个信号对于远距离串行通信时，因为要用到调制解调器，故实际上是连接 8251A和调制解调器的接口信号。当以上信号用于和计算机外部设备连接，外设不要求有联络信号时，这些信号可以不用，但$\overline{CTS}$应该接地。因为只有$\overline{CTS}$有效，才能使 TxRDY为高电平，只有 TxRDY为高电平时，CPU才能往 8251A发送数据。其他三个信号引脚可悬空不用。

(2) 8251A与 CPU相连的引脚

1) $D_7 \sim D_0$:双向数据线，与系统的数据总线相连。8251A通过它们与 CPU进行数据传输，包括 CPU对 8251A的编程命令和 8251A送往 CPU的状态信息。

2) $\overline{CS}$:片选信号，低电平有效。它是 8251A芯片的片选信号，由地址总线经地址译码器输出。只有$\overline{CS}$信号有效，CPU才能对 8251A进行读写。当$\overline{CS}$为高电平时，8251A未被选中，8251A的数据线将处于高阻状态。

3) $\overline{RD}$:读信号，低电平有效。与系统读控制线相连，当$\overline{RD}$有效时，CPU可以从 8251A的数据口中读取数据或从状态口读取状态信息。

4) $\overline{WR}$:写信号，低电平有效。与系统写控制线相连，当$\overline{WR}$时，CPU可以向 8255A的控制口写入控制字或向数据口写入数据。

5) C/$\overline{D}$:控制/数据信号。用来区分当前读写的是数据还是控制信息或状态信息，一般与地址总线的最低位 A_0相连。当 C/$\overline{D}$为高电平时，选中控制端口或状态端口，C/$\overline{D}$为低电平时，选中数据端口。

8251A共有两个端口地址，数据输入端口和数据输出端口合用一端口地址；状态端口和控制端口合用一个端口地址，它们由$\overline{RD}$和$\overline{WR}$信号区别开。总之，$\overline{CS}$、$\overline{RD}$、$\overline{WR}$和 C/$\overline{D}$这 4个信号能决定 CPU对 8251A的具体操作。

6) TxRDY:发送器准备好信号，高电平有效。它通知 CPU，8251A的发送器已经准备好，可以接收 CPU送来的数据，当 8251A收到一个数据后，TxRDY信号变为低电平。

当$\overline{RxC}$为低电平,工作命令字中的 TxEN为高电平,且发送缓冲器为空时,TxRDY有效。当使用查询方式时,CPU可以从状态寄存器的 D_0位检测这个信号;当使用中断方式时,则 TxRDY可以作为中断请求信号。

7) TxE:发送器空信号,高电平有效。它表示 8251A发送器已空,即当一个数据发送完成后 TxE变高。当 CPU向 8251A写入一个字符时,TxE变成低电平。

8) RxRDY:接收器准备好信号,高电平有效。它表示当前 8251A已经从外部设备或调制解调器上接收到一个字符,正等待 CPU取走。在中断方式下,该信号可以作为中断请求信号;在查询方式下,该信号可以作为状态信号供 CPU查询。当 CPU从 8251A的数据口读取了一个字符后,RxRDY变为低电平,表示无数据可取;当 8251A又收到一个字符后,RxRDY再次变为高电平。

9) SYNDET/$\overline{BD}$:同步和间断检测检测信号。SYNDET/$\overline{BD}$既可以是输入,又可以是输出,它取决于 8251A是工作在内同步方式,还是工作在外同步方式。

当 8251A工作在内同步方式时,SYNDET/$\overline{BD}$作为输出端,一旦 8251A搜索到所要求的同步字符,则 SYNDET/$\overline{BD}$变为高电平,表示 8251A当前已经达到同步。在双同步字符情况下,SYNDET/$\overline{BD}$会在第二个同步字符的最后位被检测到后,变为高电平。在 CPU执行一次读操作后,变为低电平。

8251A工作在外同步方式时,SYNDET/$\overline{BD}$作为输入端,当片外的检测电路找到同步字符后,从这个输入端输入一个正脉冲作为启动脉冲,使 8251A在$\overline{RxC}$的下降沿开始拼装字符。SYNDET/$\overline{BD}$的高电平状态最少要维持一个$\overline{RxC}$周期,直到$\overline{RxC}$出现一个下降沿。在外同步方式下,SYNDET/$\overline{BD}$的电平信号取决于外部启动信号。在复位时,SYNDET/$\overline{BD}$变为低电平。

10) RESET:芯片复位线,高电平有效。当 RESET上有大于等于 6倍时钟宽度的高电平时,芯片被复位处于空闲状态,直到新的编程命令到来。RESET通常与系统复位线相连。

(3) 时钟引脚

1) $\overline{RxC}$:接收器时钟输入端。$\overline{RxC}$控制 8251A接收器接收字符的速度。

同步方式下,$\overline{RxC}$等于波特率,由调制解调器供给(近距离不用调制解调器传送时由用户自行设置)。异步方式,$\overline{RxC}$是波特率的 1、16或 64倍,由方式控制命令预先选择。接收器在$\overline{RxC}$的上升沿采集数据。

2) $\overline{TxC}$:发送器时钟输入端。$\overline{TxC}$控制 8251A发送器发送字符的速度。时钟频率和波特率之间的关系同$\overline{RxC}$,数据在 $\overline{TxC}$的下降沿由发送器移位输出。

3) CLK:8251A内部工作时钟信号。由这个 CLK输入产生 8251A的内部工作时序。为了使芯片工作可靠,在同步方式工作时 CLK的频率应大于接收器和

发送器时钟频率的 30倍 在异步方式工作时，CLK的频率应大于接收器和发送器时钟频率的 4.5倍。

10.3.3 8251的编程命令

CPU可以向 8251A 写入控制命令，包括通信方式选择命令和工作命令。CPU还可以读取 8251A 的状态。下面分别加以说明。

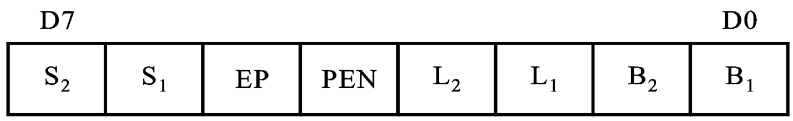


图 10.20 通信方式选择命令字

1. 通信方式选择命令字

通信方式选择命令字格式如图 10.20所示，可以可分为 4组，每组两位。下面分别说明其功能。

⑴ B_2、B_1

确定工作于同步方式还是异步方式：

0 0—— 同步方式

0 1—— 异步方式，波特率系数为 1

1 0—— 异步方式，波特率系数为 16

1 1—— 异步方式，波特率系数为 64

⑵ L_2、L_1

选择字符的位数：

0 0—— 5位

0 1—— 6位

1 0—— 7位

1 1—— 8位

⑶ EP、PEN

确定奇偶校验性质：

× 0—— 无奇偶校验

0 1—— 奇校验

1 1—— 偶校验

⑷ S_2、S_1

在异步方式下（$B_2B_1 \neq 00$），定义停止位的位数：

0 0—— 无意义

0 1—— 1个停止位

1 0—— 1.5个停止位

1 1—— 2个停止位

在同步方式下（$B_2B_1 = 00$），定义同步方式：

× 1—— 外同步

0 0—— 内同步，2个同步字符

1 0—— 内同步，1个同步字符

例如，若 8251A芯片进行异步串行通信时，要求波特率系数为 16，字符长度为 7位，奇校验，2个停止位。则方式选择字应为：

$$11011101_{0B} = 0DAH$$

若要求 8251A作为同步通信的接口，内同步且需 2个同步字符，偶校验，7位字符。其方式选择字为：

$$00111000B = 38H$$

2. 工作命令字

工作命令字用于确定 8251A的操作，使 8251A处于某种工作状态，以便接收或发送数据。工作命令格式见图 10.21。

D7							D0
EH	IR	RTS	ER	SBRK	RxE	DTR	TxEN

图 10.21 工作命令格式

(1) TxEN（允许发送位 TxEN，该位可作为发送中断屏蔽位）

1—— 允许发送

0—— 不允许发送

(2) DTR（数据终端准备就绪）

1—— 强制$\overline{\text{DTR}}$有效（低电平），表示数据终端准备好

0—— 置$\overline{\text{DTR}}$无效

(3) RxE（允许接收 RxE，该位可作为接收中断屏蔽位）

1—— 允许接收

0—— 不允许接收

(4) SBRK (发中止字符)

1—— 强迫 TxD为低电平,输出连续的空号

0—— 正常操作

(5) ER (错误标志复位)

1—— 使状态字中的错误标志 PE、OE、TE 复位

0—— 不复位

(6) RTS (发送请求标志)

1—— 强制 $\overline{RTS}$有效 (低电平)

0—— 置 $\overline{RTS}$无效

(7) IR (内部复位)

1—— 使 8251A回到方式选择命令状态

0—— 不回到方式命令

(8) EH (外部搜索方式)

1—— 启动搜索同步字符

0—— 不搜索同步字符

3. 工作状态字

8251A进行数据传输后的状态字存放在状态寄存器中,CPU通过读操作读入状态字,进行分析和判断,以决定下一步的工作。状态字格式如图 10.22。

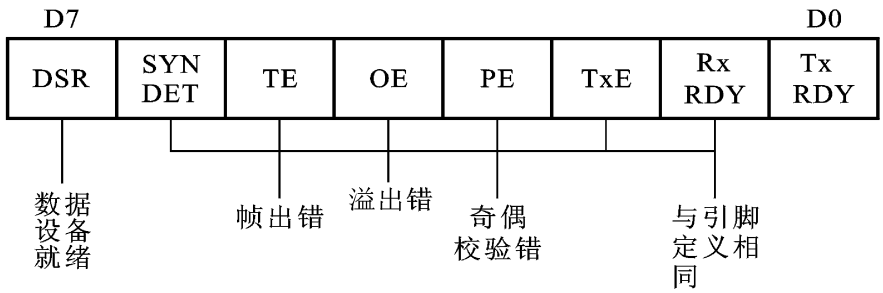


图 10.22 工作状态字

其中,状态位 RxRDY、TxE、SYNDET的定义与 8251A芯片引脚的定义完全相同,DSR与芯片引脚 $\overline{DSR}$意义相同,但有效电平相反。状态位 TxRDY,只要发送数据缓冲寄存器一空就置 1,而 8251A芯片引脚 TxRDY 为高电平的条件,除发送数据缓冲寄存器空外,还必须满足 $\overline{CTS}$=0和 TxE =1两个条件。

另外,PE为奇偶校验错误标志位,OE为溢出错误 (也称为覆盖错误)标志位,TE为帧格式错误标志位。所有状态位均置 1有效。

10.3.4 8251A初始化的步骤

8251A是可编程的多功能串行通信接口,在使用 8251A时,必须先对它进行初始化编程,选择 8251A的工作方式等。

1. 初始化编程的步骤

⑴ 芯片复位后,第一次用输出指令写入奇地址端口的应是方式选择指令。约定双方的通信方式 (同步/异步),数据格式 (数据位和停止位长度、校验特征、同步字符特征)及传输速率 (波特率系数)等参数。

⑵ 如果方式选样指令中规定了 8251A工作在同步方式,那么,CPU用执行输出指令向奇地址端口写入规定的 1个或 2个字节的同步字符。

⑶ 只要不是复位命令,不论同步方式还是异步方式,均由 CPU执行输出指令向奇地址端口写入工作命令指令,控制允许发送/接收或复位。

初始化结束后,CPU就可通过查询 8251A的状态字内容或采用中断方式,进行正常的串行通信发送/接收工作。

因为方式字、命令字及同步字均无特征标志位,且都是写入同一个命令口地址,所以在对 8251A初始化编程时,必须按一定的顺序流程,若改变了这种顺序流程,8251A就不能识别。

2. 内部复位命令

当 8251A通过写入方式选择字,规定了 8251A的工作方式后,可以根据对 8251A工作状态的不同要求随时向控制端口输出工作命令指令字。若要改变 8251A工作方式,应先使 8251A芯片复位,内部复位命令字为 40H。8251A芯片复位后,又可重新向 8251A输出方式选择字,以改变 8251A的工作方式。

下面具体介绍初始化 8251A的方式选择命令字和工作命令字。

1. 异步方式下的初始化编程

要求使 8251A工作在异步方式,波特率系数为 16,字符长度为 8位,偶校验,2个停止位。则方式选择字为:11111110B=0FEH。工作状态要求:复位出错标志、使请求发送信号 $\overline{RTS}$有效、使数据终端准备好信号 $\overline{DTR}$有效、发送允许 TxEN有效、接收允许 RxE有效。则工作命令指令字应为 37H。假设 8251A的两个端口地址分别为 0C0H和 0C2H,初始化编程如下:

```
MOV     AL, 0FEH
OUT     0C2H, AL            ;设置工作方式
MOV     AL, 37H
OUT     0C2H, AL            ;设置工作状态
```

2. 同步方式下初始化编程

要求 8251A工作在同步方式,两个同步字符 (内同步)、奇校验、每个字符 8位,则方式选择字应为 1CH。工作状态要求:使出错标志复位,允许发送和接收、使 CPU已准备好且请求发送,启动搜索同步字符,则工作命令指令应该是 0B7H。又设第一个同步字符为 0AAH,第二个同步字符为 55H。还使用上例 8251A芯片,这样要先用内部复位命令 40H,使 8251A复位后,再写入方式选择控制字。具体程序段如下:

```
MOV     AL, 40H
OUT     0C2H, AL        ;复位 8251A
MOV     AL, 1CH
OUT     0C2H, AL        ;设置方式选择字
MOV     AL, 0AAH
OUT     0C2H, AL        ;写入第一个同步字符
MOV     AL, 55H
OUT     0C2H, AL        ;写入第二个同步字符
MOV     AL, 0B7H
OUT     0C2H, AL        ;设置命令字
```

10.3.5 8251的应用举例

1. 利用查询状态字方式实现串行数据输入

[例题 10.5] 用异步串行输入方式输入 2000个数据,存放到内存 BUFFER开始的单元中。因为 8251A从外设接收一个字符 RxRDY会自动置位,因此程序中不断对状态寄存器的 RxRDY位进行测试,查询 8251A是否已经从外设接收了一个字符。若 RxRDY变为有效,即收到一个字符,CPU就执行输入指令取回一个数据存放到内存缓冲区,RxRDY在 CPU输入一个字符后会自动复位。除了对状态寄存器的 RxRDY位检测之外,程序还要检测状态寄存器的 D_3、D_4、D_5位,以判断是否出现奇/偶错、覆盖错或帧格式错误,若发现错误就转错误处理程序。下面的程序中没有给出错处程序。设 8251的地址为 80H和 81H。

```
        MOV     AL, 0FEH        ;异步方式选择字
        OUT     81H, AL         ;写入
        MOV     AL, 37H         ;工作命令字
        OUT     81H, AL         ;写入
```

```
        MOV     BX, BUFFER      ;BX指向缓冲区首址
        MOV     DI, 0           变址寄存器初值为 0
        MOV     CX, 2000        ;设置计数器初值
WAIT:   IN      AL, 81H         ;读状态字到 AL
        TEST    AL, 2           ;测试状态字的 D2位,即 RxRDY位
        JZ      WAIT            ;为 0,未收到字符,继续取状态字
        IN      AL, 80H         ;当 RxRDY为 1,则从数据口输入数据
        MOV     [BX][DI], AL    ;将字符送入缓冲区
        INC     DI              ;缓冲区指针下移一个单元
        IN      AL, 81H         ;读状态字
        TEST    AL, 38H         ;判断有无三种错误
        JNZ     ERROR           ;有错,则转出错处理程序
        LOOP    WAIT            ;没错,判是否结束循环
        JMP     EXIT            ;结束
ERROR:  CALL    ERR_PRO         ;转入错误处理程序
EXIT:   ……
```

2. 用 8251A作为串行接口的实例

【例题 10.6】　图 10.23是以 8251A作为异步串行接口的电路图。8251A的发送时钟 TxC和接收时钟 RxC由 8253的 OUT2提供。要求 8251A的波特率为 2 400,波特率系数选 16,则 8251A的 TxC和 RxC应为 38.4 kHz,8253的 CLK为 2 MHz,使 8253的计数器 2工作于方波方式,分频系数为 52,则 OUT2输出频率约为 38.461 kHz,基本满足要求。8251A的时钟 CLK频率为 2 MHz,8251A的 $C/\overline{D}$接 A0,片选信号 $\overline{CS}$由 CPU的地址线 A15～A1译码输出,数据端口地址为 0E0H,控制端口地址为 0E1H。由 MAX232实现电平转换。

下面给出完成对 8251A的初始化编程的程序。在对 8251A设置方式字之前,先进行 8251A的软件复位,程序中先送三个 0,再送 40H到控制端口使 8251A复位。初始化程序段如下:

```
MOV     AL, 00H
OUT     0E1H, AL
CALL    DELAY           ;调用延时程序
OUT     0E1H, AL
CALL    DELAY
OUT     0E1H, AL
```

```
CALL    DELAY
MOV     AL, 40H          ;内部复位命令指令字
OUT     0E1H, AL         ;确保 8251A 复位
CALL    DELAY
MOV     AL, 4EH          ;写入方式控制字,异步、8位数据
OUT     0E1H, AL         ;波特率系数为 16,不校验、1个停止位
MOV     AL, 27H          ;写入命令字,启动发送器和接收器
OUT     0E1H, AL
```

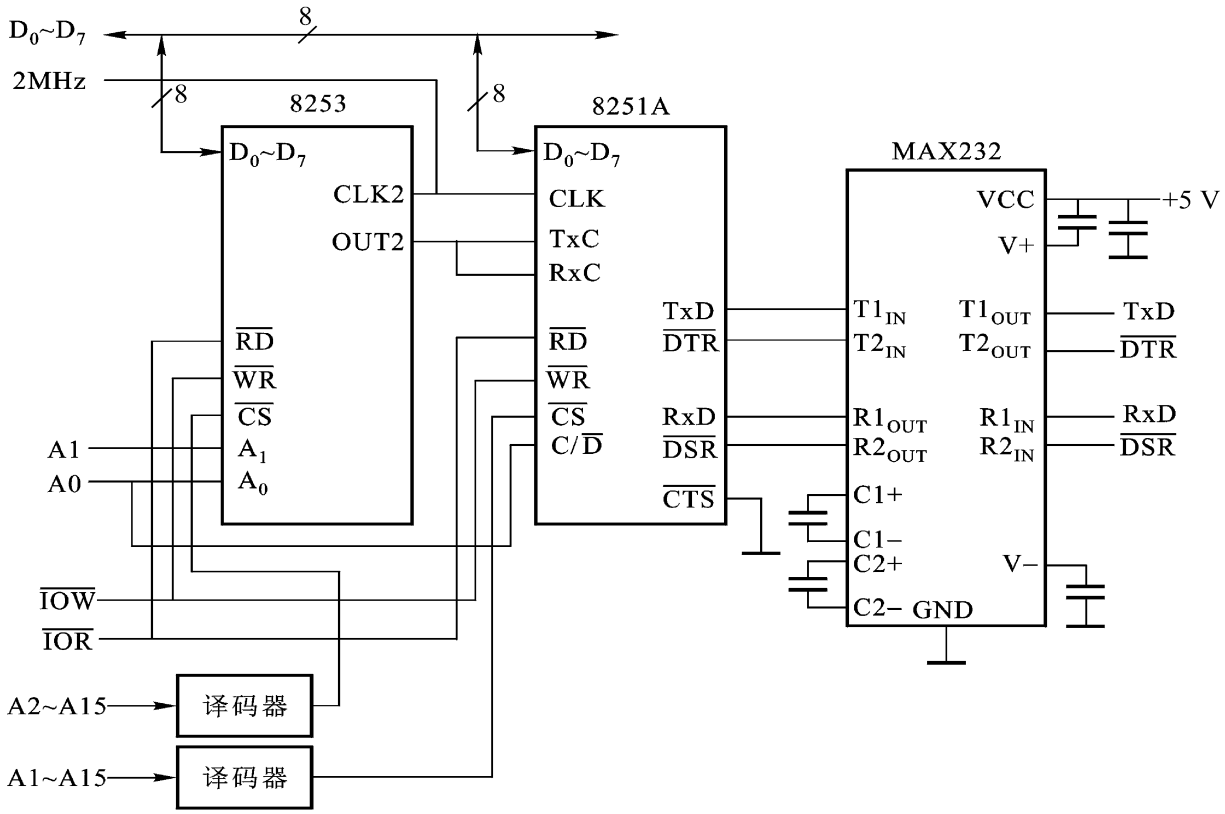


图 10.23 8251A的应用电路

若串行通信采用查询方式,则程序应先对状态字进行测试,判断 TxRDY 状态位是否有效。若 TxRDY 为高电平,说明当前数据输出缓冲器为空,CPU 可以向 8251A 输出该数据 否则就等待。

设要向外输出的数据已放在 AH 寄存器中。数据输出的程序段如下:

```
WAIT: N     AL, 0E1H          ;读状态口,输入状态字
      TEST  AL, 01H           ;测试状态字 D0 位 TxRDY
```

```
JZ      WAIT            ;若为 0则等待
MOV     AL, AH          ;取要发送的数据
OUT     0E0H, AL        ;往端口输出一个字符
```

思考题与习题

10.1 什么是同步通信方式？什么是异步通信方式？试说明各自的主要优、缺点，并说明各适合在什么场合下使用。

10.2 什么叫奇/偶校验错误？什么叫覆盖错误？什么叫帧格式错误？

10.3 什么是双工，半双工和单工通信方式？举你熟悉的通信例子说明它们属于何种方式。

10.4 在数据通信系统中，什么情况下需采用全双工方式，什么情况下可用半双工方式？

10.5 若 8250的 CLK为 1MHz，波特率为 4800时，分频系数是多少？

10.6 设异步传输时，每个字符对应 1个起始位、7个信息位、1个奇/偶校验位和 1个停止位，如果波特率为 9600，每秒能传输的最大字符数为多少个？

10.7 一个数据传输系统由哪几部分组成？每一部分各起什么作用？

10.8 异步传输的基本特征是什么？保证它正确传输的条件是什么？

10.9 利用一个异步传输系统传送文字资料，系统的波特率为 1200，待传送的资料为 5000个汉字长，设系统不用校验位，停止位只用一位，至少需要多少时间才能传完全部资料？

10.10 在 RS-232C标准中，信号电平与 TTL电平不兼容，问 RS-232C标准的 1和 0分别对应什么电平？RS-232C的电平和 TTL电平之间通常用什么器件进行转换？

10.11 要求 PCCOM1串行通信格式为每个字符包含 1个起始位、8个信息位、1个奇/偶校验位和 2个停止位，试对系统中 8250的 LCR编程。

10.12 要求 PCCOM2工作于波特率为 1200时，每个字符包含 1个起始位、7个信息位、1个奇/偶校验位和 1个停止位，采用偶校验。试完成初始化。

10.13 要求 8251A工作于异步方式，波特率系数为 16，字符长度为 7位，奇校验，2个停止位。工作状态要求：复位出错标志、使请求发送信号 $\overline{\text{RTS}}$有效、使数据终端准备好信号 $\overline{\text{DTR}}$有效、发送允许 TxEN有效、接收允许 RxE有效。设 8251A的两个端口地址分别为 0C0H和 0C2H，试完成始化编程。

第11章

模数、数模转换

计算机中处理的数据是数字量，而工程实际中需要处理的是连续变化的物理量，也称之为模拟量。所谓连续变化的量，一方面它们是随时间而连续变化的，另一方面它们的数值也是连续变化的，如温度、压力、流量、位移、速度等。通过传感器可以转化为电压、电流或频率等信号。

为了使计算机能够处理这些模拟量，需要实现模拟量与数字量之间的转换。模/数转换器（A/D转换器或ADC）能实现从模拟量到数字量之间的转换，数/模转换器（D/A转换器或DAC）能实现从数字量到模拟量之间的转换。通常A/D转换器的输入为电压信号，输出为数字信号；D/A转换器的输入为数字信号，输出为电流或电压。

11.1 A/D转换器及其接口

11.1.1 A/D转换器的基本概念

1. 量化

当以数量表示连续量的时候都会遇到量化问题。所谓量化就是以一定的量化阶距为单位，把数值上连续的模拟量转变为数值上离散量的过程。

设输入为x(t)，量化阶距是q，量化后的输出为y(t)，那么量化可以表示为：

$$y(t) = INT(x(t)/q)$$

其中 INT( )是取整函数，x(t)/q的小数部分被舍去。

为了减小量化误差，通常以“4舍 5入”的方法进行量化，量化可以表示为：

$$y(t)=\mathrm{INT}\left(\frac{x(t)+0.5q}{q}\right)$$

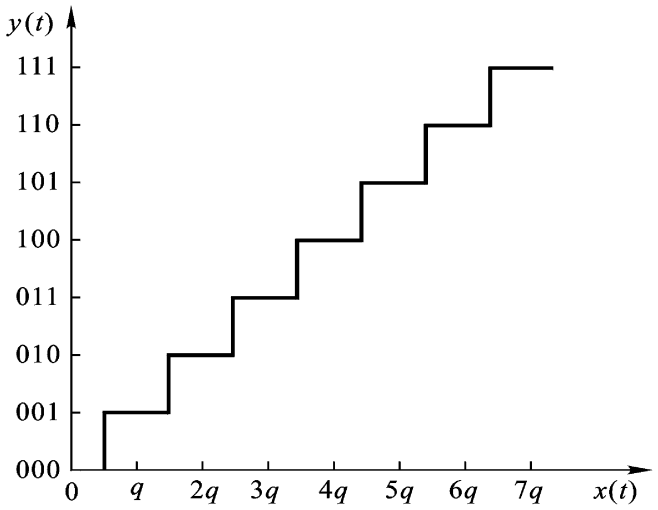


图 11.1　3位的 A/D转换器的量化关系

图 11.1说明了一个 3位的 A/D转换器的量化关系。当 x(t)<0.5q时，y(t)=000;当 0.5q≤x(t)<1.5q时，y(t)=001;当 1.5q≤x(t)<2.5q时，y(t)=010;当 2.5q≤x(t)<3.5q时，y(t)=011;……;当 6.5q≤x(t)<7.5q时，y(t)=111。

2. 输入极性与编码

当输入信号为单极性信号时，以二进制数进行量化编码。输入范围为 0～+5 V的 8位 ADC，其输入/输出关系如图 11.2a所示。

当输入为双极性信号（即输入信号的幅值可能为正、可能为负）时，对输入信号的编码通常有以下三种方式：

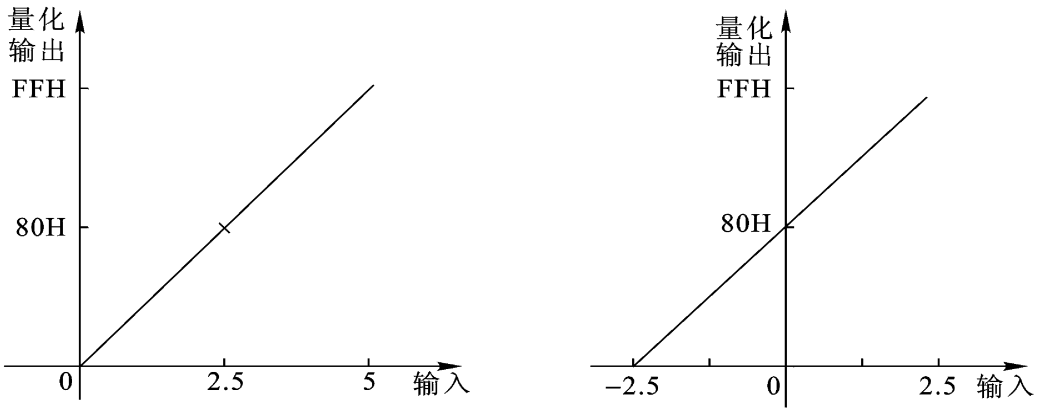
偏移二进制码——以最高位为符号位，以 1表示正，以 0表示负；后面的各位表示幅值。就相当于把单极性的 ADC的输入输出特性曲线向左平移了一半。输入为 −2.5～+2.5 V的 8位 ADC，其输入/输出之间的关系见图 11.2b。

原码——以原码来表示，当输入为正时，符号位为 0;当输入为负时，符号位为 1。后面的各位表示其幅值。

补码——以补码来表示。其符号位刚好与偏移二进制码的符号位相反，后面的各位相同。

3. A/D转换器的主要性能参数

(1) 量化误差



(a) 单极性输入　　(b) 偏移二进制码

图 11.2 单极性输入二进制编码和双极性输入偏移二进制编码

A/D转换器将连续的模拟量转换为离散的数字量,对一定范围内的连续变化的模拟量只能量化成同一个数字量,从图 11.1中可见 y(t)的量化误差是 ±0.5q,y(t)的最低位 (LSB)的变化对应于输入变化一个量化阶距 q,因此又以 ±0.5 LSB表示量化误差。这种误差是由于量化引起的,所以称为量化误差,它是量化器固有的,是不可克服的。

例如某 A/D转换器的分辨率为 8位,满量程输入电压 V_{FS} =5 V,则量化阶距是 5 V/(2^8 −1),即 0.0196 V,量化误差是 ±0.5 LSB。

(2) 分辨率

A/D变换器的分辨率能表示 A/D变换器对输入信号的分辨能力。A/D变换器的分辨率以输出二进制数的位数表示。n位输出的 A/D变换器能区分 2^n 个不同等级的输入模拟电压,能区分的输入电压最小值 (即量化阶距)为满量程输入电压的 $1/2^n$。当满量程输入电压一定时,输出的位数越多,分辨率越高。例如某 A/D转换器的分辨率为 8位,满量程输入电压 V_{FS} =5 V,则能辨别的最小模拟量是 5 V/(2^8 −1),即 0.0196 V。

(3) 转换误差

转换误差说明 A/D变换器实际的输出数字量与理论上的输出数字量之间的差别,通常以整个输入范围内的最大输出误差表示。一般用最低有效位的倍数来表示转换误差,例如转换误差≤ ±1 LSB,就说明在整个输入范围内,输出数字量与理论上的输出数字量之间的误差小于最低位的一个数字。

(4) 转换时间

转换时间是指 A/D转换器开始一次转换到完成转换得到相应的数字量输出所需的时间。

⑸ 量程

量程是指 A/D转换器能够实现转换的输入电压范围。

4. A/D转换器的类型

A/D转换器的类型较多,主要的有并行比较型、逐次比较型、双积分型等。

并行比较型的转换速度最高,但分辨率一般在 8位以内。因为 n位并行比较型 A/D中需要 2^n-1个电压比较器,当 n大于 8以后,需要的电压比较器太多使得芯片的面积大、成本高。

双积分型的分辨率高,抗干扰能力强,但转换速度低,一般为 1~1 000 ms,通常用在对速度要求不高但需要很高精度的场合。

逐次比较型的分辨率高,转换时间在 0.1~100 μs之间。转换速度比并行比较型要低,但远高于双积分型。随着集成电路工艺的提高,其转换速度也在提高。因此,逐次比较型的 A/D适合既要求精度、又要求速度的场合。

一般,将转换时间大于 1ms的称为低速 A/D,1 μs~1 ms的称为中速 A/D,小于 1 μs的称为高速 A/D。

11.1.2 典型 A/D转换器介绍

1. 8位 A/D转换器 ADC0808/0809

美国国家半导体公司 (National Semiconductor)的 ADC0808/0809是 CMOS工艺的 8位 A/D转换器。芯片内不仅包括一个 8位逐次逼近型 A/D转换器,还包括一个 8通路模拟开关,可以从 8路模拟输入中选择一路。ADC0808/0809不需要外部调零,也不需要满刻度调整。数据输出接口有三态功能,易于与微处理器接口。转换时间为 100 μs,功耗 15 mW,工作温度范围 -40~+85 ℃。图 11.3给出了 ADC0808/0809的内部逻辑框图和引脚编号。ADC0808和 ADC0809的主要区别是精度不同,ADC0808的误差为 ±1/2 LSB,ADC0809的误差为 ±1 LSB。

德州仪器公司 (Texas Instruments Incorporated)生产的 ADC0808/0809与国家半导体公司的 ADC0808/0809在内部结构上不同,但引脚、外特性相同可以互换。它也是逐次逼近型 A/D转换器,其核心包括采样/保持电路、开关电容阵列、高输入阻抗电压比较器、逐次逼近寄存器等。

⑴ 引脚介绍

8路模拟输入信号分别为 IN0~IN7。输入通路选择控制由地址锁存与译码电路进行,模拟信号的 3位地址分别为 ADDRESS C、B、A,由 ALE的上升沿将它们锁存。被锁存的 C、B和 A经译码电路后控制模拟开关,C、B、A与模拟开关输入、输

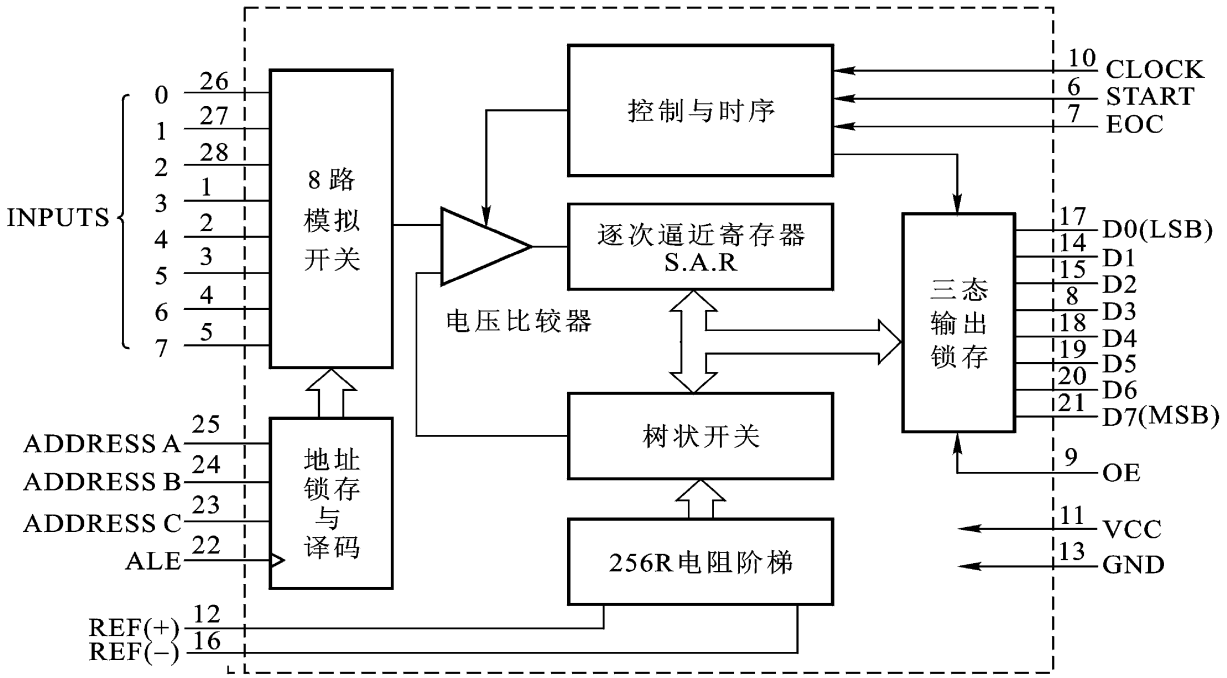


图 11.3 ADC0808/0809的内部逻辑框

出之间的关系见表 11.1。可见,8路模拟输入信号由 3位地址信号进行选择。

表 11.1 ADDRESS C、B、A与模拟开关输入、输出之间的关系

C、B、A	模拟开关输出 V_X
000	IN0
001	IN1
010	IN2
011	IN3
100	IN4
101	IN5
110	IN6
111	IN7

VCC和 GND分别为电源和地,VCC电压为 5 V。

REF (+)、REF (−)是基准电压输入,REF (+)不应大于 VCC,REF (−)不应小于 GND。若 REF (−)接 GND,则 REF (+)的电压就是满量程电压,一个 LSB的对应的输入电压等于 REF (+)/255。

START是 A/D启动信号,脉冲宽度应大于 200 ns; CLOCK为时钟脉冲,频率范围为 10 kHz~1 MHz,对应的时钟周期为 T。

引脚 D7~D0是数字量输出端,$\overline{OE}$是数据输出允许端。当$\overline{OE}$=0时 D7~D0上输出最近一次已完成的转换的结果,当$\overline{OE}$=1时输出禁止,D7~D0为高阻。

EOC是转换结束信号 (End Of Convert),其上升沿表示一次 A/D转换完成。

(2) ADC0809的时序

图 11.4是 ADC0809的时序图。ALE将模拟信号的 3位地址 ADDRESS C、B、A锁存,模拟开关选择某一路输入 IN;START脉冲的上升沿开始进行 A/D转换,经过 8T+2 μs后 EOC变低 (t_c),直到 EOC的上升沿时本次 A/D转换完成。因此,在 EOC上升沿后可以读出本次 A/D转换的结果。当以程序判别 EOC的上升沿时,应先确认 EOC为低,再判别是否变高。

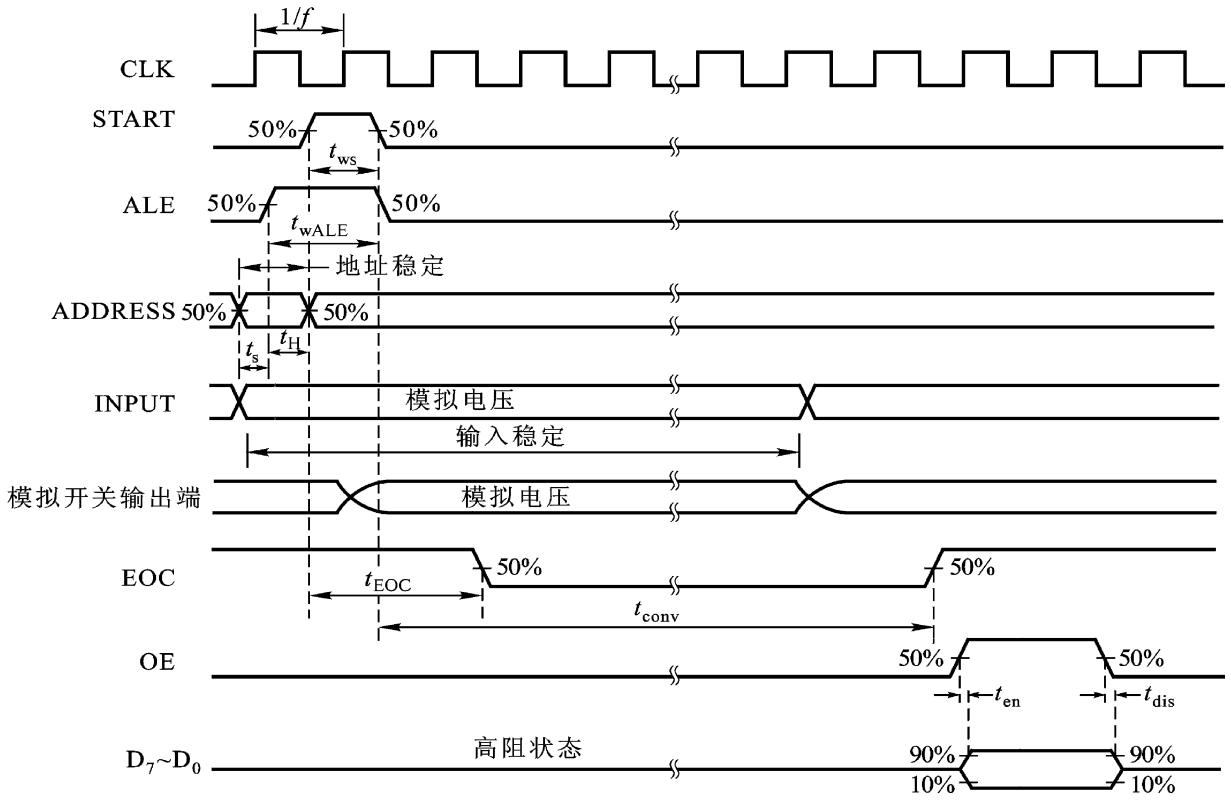


图 11.4　ADC0809的时序图

2. 12位高速逐次逼近型 A/D转换器 AD7472

AD7472是通用 12位高速、低功耗逐次逼近型 A/D转换器。采样频率由外部时钟控制,最高采样频率达到 1.5 MSPS (Mega-Sample/s)。芯片内含一个低

噪声的宽带放大器，具有跟踪/保持功能，其带宽超过 1MHz。AD7472为单通道输入，12位并行数据输出，具有标准的接口电路，能方便地与微处理器、DSP等接口。单一的 2.7～5.25 V 电源，适合于各种系统。3 V电源、1.5 MSPS情况下，AD7472典型的平均功耗仅为 4mW。

(1) 内部结构

AD7472的内部结构如图 11.5所示。T/H为跟踪/保持放大器（Track/Hold Amplifier），当 A/D启动后进入保持模式即放大器的输出保持不变，A/D完成后进入跟踪模式即放大器的输出跟随输入变化，因此 T/H放大器就是取样保持电路。12位逐次逼近型 A/D转换器在控制逻辑电路的控制下工作，REF IN是参考基准电压的输入，要求有一定的精度。A/D转换器的转换结果送输出驱动电路，当芯片被选中（$\overline{CS}$=0）且进行读操作（$\overline{RD}$=0）时，DB0～DB11输出 A/D转换的结果。输出驱动电路具有三态逻辑，当芯片未选中（$\overline{CS}$=1）或非读操作（$\overline{RD}$=1）时，DB0～DB11为高阻状态。VDRIVE是输出驱动电路的电源。

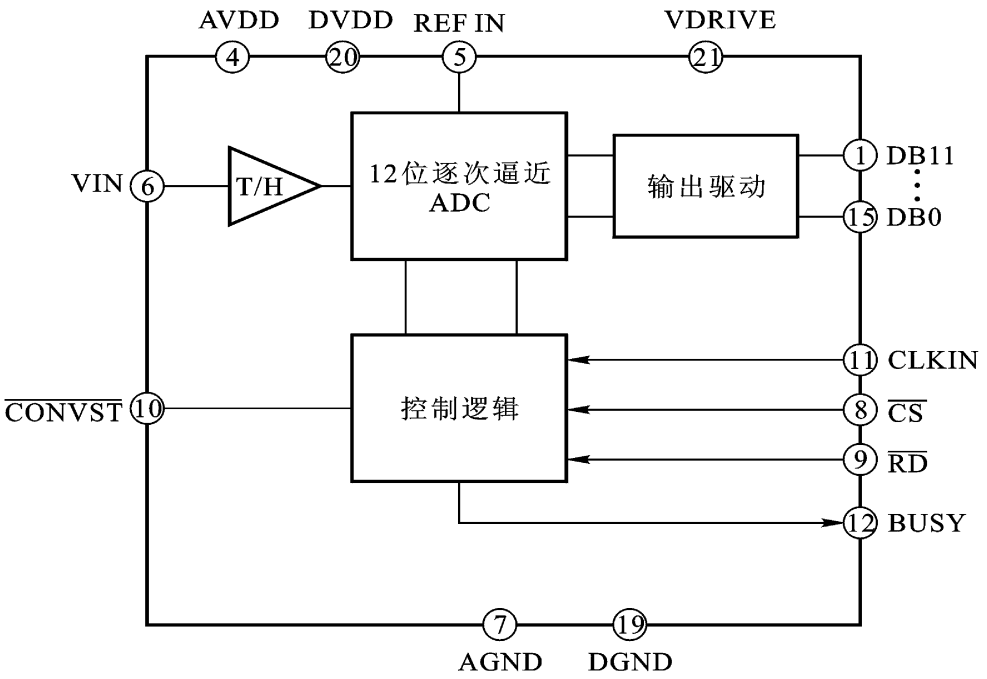


图 11.5 AD7472的内部结构框图

(2) 引脚说明

AD7472的引脚编号见图 11.5,下面对各引脚分别加以说明。

$\overline{CS}$——片选信号（输入），低电平有效。与$\overline{RD}$信号配合，使 A/D转换的结果送到数据线 DB11～DB0上。

$\overline{RD}$——读信号（输入），低电平有效。与$\overline{CS}$信号配合，使 A/D转换的结果送到数据线 DB11～DB0上，即当$\overline{CS}$和 $\overline{RD}$信号同时有效时，AD7472的数据线输出 A/D的结果，否则数据线为高阻。

$\overline{CONVST}$——转换开始信号（输入），低电平有效。在$\overline{CONVST}$的下降沿，传输/保持放大器从传输模式变为保持模式，同时 A/D转换开始。$\overline{CONVST}$的脉冲宽度至少 15 ns。如果$\overline{CONVST}$保持低电平直到转换完成，则 AD7472将进入休眠模式。此后到来的$\overline{CONVST}$的上升沿将唤醒 AD7472，典型的唤醒时间为 1 μs。

CLOCK IN——主时钟（输入），A/D转换器的时钟信号由该引脚输入。AD7472使用 14个时钟周期完成一次 A/D转换。因此，主时钟的频率决定了一次 A/D转换所需要的时间。AD7472的 CLOCK IN最高频率是 26 MHz。

BUSY——忙信号（输出），以逻辑电平表示芯片的状态。当$\overline{CONVST}$下降沿到来后，BUSY变为高，并在 A/D转换期间一直保持高。一旦 A/D转换完成，转换的结果放到输出寄存器中，BUSY就变低。从 BUSY下降沿起，跟踪/保持放大器由保持模式变为跟踪模式，以便获得对输入信号的跟踪。

REF IN——参考电压输入端，外部的参考电压必须加到这个端上。为了获得理想的性能，外部参考电压应在［2.5± (2.5×1%)］V的范围内。

AVDD——模拟电路的电源输入，电压 2.7～5.25 V。它只供芯片内的模拟电路使用，应使它与 DVDD的电压值相同。该电源端需要相对于 AGND滤波。

DVDD——数字电路的电源输入，电压 2.7～5.25 V。它为芯片内的数字电路提供电源，应使它与 AVDD的电压值相同。该电源端需要相对于 DGND滤波。

AGND——模拟地。它是 AD7472中所有模拟电路的地电压参考点。所有模拟输入信号和外部参考信号以该点为地电压。AGND应与 DGND有相同的电压，即使是瞬时差值也不能超过 0.3 V。

DGND——数字地。它是 AD7472中所有数字电路的地电压参考点。

VIN——单端模拟信号输入。输入范围是 0～REF IN。VIN端具有高输入阻抗。

VDRIVE——这是输出驱动电路的电源，+2.7～+5.25 V。它决定了数据输出端的高电平电压，这使得 AD7472的输出接口更为灵活。

DB11～DB0——数据输出线，三态逻辑。当$\overline{CS}$和 $\overline{RD}$信号同时有效时，DB11～DB0输出 A/D转换的结果，否则数据线为高阻。

(3) 操作时序

AD7472快速数据采集模式的时序图如图 11.6所示，这种模式时 AD7472

不进入休眠状态。在 CLOCK IN端加上连续的时钟信号 f_{clock}，其周期为 t_{clock}，图 11.6中未画出。$\overline{CONVST}$为启动信号，其下降沿使 A/D 开始。当 $\overline{CONVST}$由高电平变为低电平时，经过 t_2后 BUSY 变高，表示 A/D 转换正在进行中，t_2的最大值为 10 ns。从 $\overline{CONVST}$下降沿起经过 $t_{convert}$后 BUSY 变为低，表示 A/D 转换完成，$t_{convert}$等于 $t_{clock}\times14$，即 14个 t_{clock}。从 BUSY 下降沿起，跟踪/保持放大器由保持模式变为跟踪模式。需要经过一段时间才能使跟踪/保持放大器的输出跟踪输入信号，并且要使因这种由保持模式变为跟踪模式导致的误差小于 1LSB。这段时间称为获得时间 $t_{acquisition}$（图 11.6中 t_9），至少应保证 135 ns，使用时应留有余地。

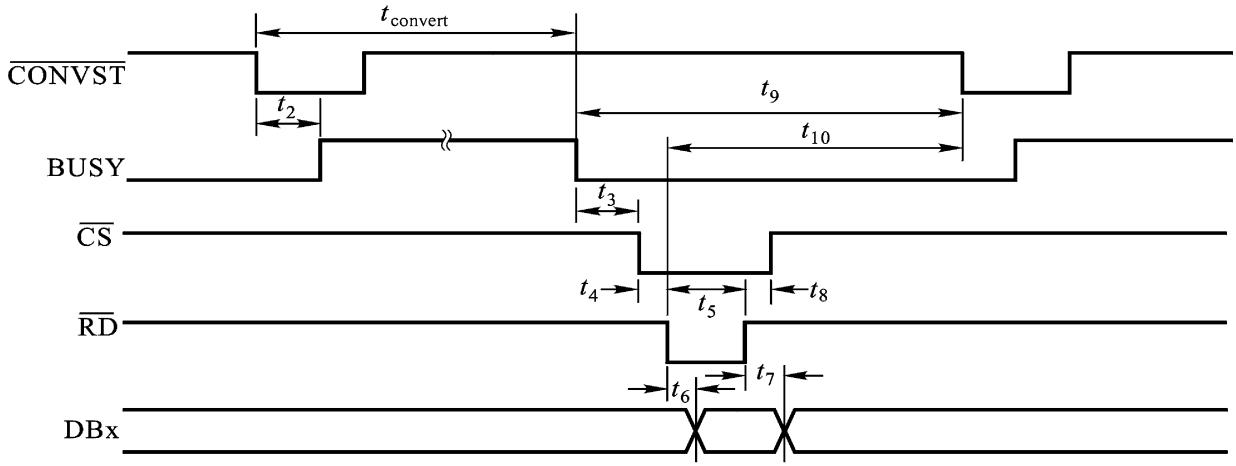


图 11.6 AD7472的时序

因此，AD7472完成一次数据采集需要的时间 t_s为：

$$t_s = t_{convert} + t_{acquisition}$$

可以计算出在 f_{clock}为 26 MHz时的最大采样频率：

$$1/t_s = 1/(14/(26\times10^6) + 0.135\times10^{-6}) = 1.4848\times10^6\ (SPS)$$

11.1.3 应用举例

【例 11.1】 图 11.7是 0809通过 8255与 CPU接口的例子。0809的 D7～D0接 8255的 PA口，PA口工作于方式 0输入。0809的 ADDC、ADDB、ADDA接 8255的 PB2～PB0，PB口工作于方式 0输出。8255的 PC口高 4位工作于方式 0输入，PC7接 0809的 EOC。8255的 PC口低 4位工作于方式 0输出，PC0接 0809的 START和 ALE。8255的地址为 70H～73H。

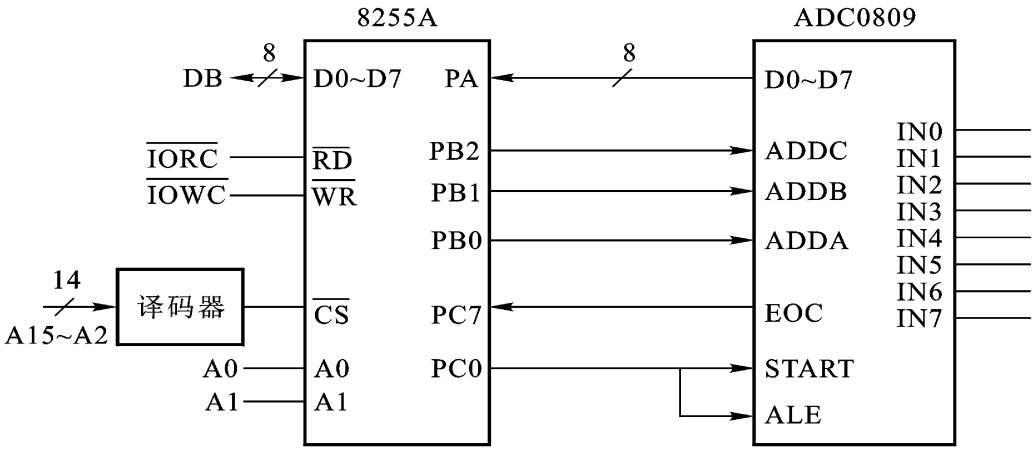


图 11.7　ADC0809与 8255的连接

⑴ 下面的一段程序以查询的方式对 IN0端进行 100次采样，数据存入 DATA开始的内存中。

```
        MOV     AL, 10011000B       ;8255编程
        OUT     73H, AL
        MOV     AL, 00H
        OUT     72H, AL             ;START、ALE=0
        MOV     BX, OFFSET DATA     ;DATA是数据区首地址
        MOV     CX, 100
        MOV     AL, 00H
        OUT     71H, AL             ;选 IN0
AGAIN:  MOV     AL, 01H
        OUT     72H, AL             ;START、ALE=1
        MOV     AL, 00H
        OUT     72H, AL             ;START、ALE=0
WAIT0:  IN      AL, 72H
        AND     AL, 80H
        JNZ     WAIT0               ;若 EOC为低，则执行下条指令
WAIT1:  IN      AL, 72H
        AND     AL, 80H
        JZ      WAIT1               ;若 EOC为高，则执行下条指令
        IN      AL, 70H             ;从 PA口输入数据
        MOV     [BX], AL            ;存入内存
```

```
        INC     BX
        LOOP    AGAIN
```

程序中先判 EOC为低,再判 EOC为高,即识别了 EOC的上升边沿。当EOC的上升沿到来后,读取 A/D转换的数据。

⑵ 下面的一段程序是二重循环结构,内循环对 IN0～IN7端进行轮流采样,一组数据存入 DATA开始的 8个内存单元中。在外循环控制下作 100次内循环,即采集 100组数据。

```
        MOV     AL,10011000B        ;8255编程
        OUT     73H,AL
        MOV     AL,00H
        OUT     72H,AL              ;START、ALE＝0
        MOV     BX,OFFSET DATA      ;DATA是数据区首地址
        MOV     CX,100
AGAIN0: MOV     DL,00
AGAIN1: MOV     AL,DL
        OUT     71H,AL              ;选 IN0
        MOV     AL,01H
        OUT     72H,AL              ;START、ALE＝1
        MOV     AL,00H
        OUT     72H,AL              ;START、ALE＝0
WAIT0:  IN      AL,72H
        AND     AL,80H
        JNZ     WAIT0               ;判 EOC为低
WAIT1:  IN      AL,72H
        AND     AL,80H
        JZ      WAIT1               ;判 EOC为高
        IN      AL,70H              ;输入数据
        MOV     [BX],AL
        INC     BX
        INC     DL
        CMP     DL,8
        JNZ     AGAIN1
        LOOP    AGAIN0
```

【例 11.2】　图 11.8是一个由 ADC0809、8255、8253及 8259组成的数据采集系统。ADC0809的 D7～D0接 8255的 PA口，PA口工作于方式 0输入。ADC0809的 ADDC、ADDB、ADDA接 8255的 PB2、PB1、PB0，PB口工作于方式 0输出。8253通道 0工作于方式 1，OUT0输出的脉冲经反相后接 ADC 0809的 START和 ALE，因此，每个脉冲启动一次 A/D变换。EOC接 8259的 IR7，EOC上升沿引起中断。8255的地址为 70H～73H，8253的地址为 74～77H，8259的地址为 78～79H。8259的 IR0～IR7对应的中断向量为 30H～37H。

试编写程序，以 8 kHz采样频率连续采样 8 000个数据，数据存入 BUFFER起的内存中。

程序：

```
DATA    SEGMENT
BUFFER  DB   8000 DUP (0)
COUNTER EQU  $ -BUFFER
DATA    ENDS
CODE    SEGMENT
ASSUME  DS:DATA, CS:CODE
MAIN    PROC     FAR                ;主程序
        MOV      AL, 00010011B      ;8259编程，ICW1，边沿触发
        OUT      78H, AL
        MOV      AL, 30H            ;ICW2，中断矢量
        OUT      79H, AL
        MOV      AL, 00001001B      ;ICW4
        OUT      79H, AL
        MOV      AL, 7FH            ;中断屏蔽字
        OUT      79H, AL
        MOV      AL, 00110010B      ;8253编程，通道 0方式 1
        OUT      77H, AL
        MOV      AX, 125            ;计数器初值
        OUT      74H, AL
        MOV      AL, AH
        OUT      74H, AL
        MOV      AL, 10010000B      ;8255编程
        OUT      73H, AL
        MOV      CX, 8000
```

```
        PUSH    DS                  ;以下程序段设置中断入口地址表
        MOV     DX, OFFSET  INTERRUPT_ SERVE
        MOV     AX, SEG  INTERRUPT_ SERVE
        MOV     DS, AX
        MOV     AX, 2537H           ;37H是 IR7的中断向量
        INT     21H
        POP     DS                  ;设置中断入口地址表完毕
        LEA     BX, BUFFER
        MOV     CX, COUNTER
        STI
WAIT:   CMP     CX, 0
        JNZ     WAIT
        CLI
        RET
MAIN    ENDP
INTERRUPT_ SERVE   PROC   FAR ;中断服务程序
INTS:   PUSH    AX
        IN      AL, 70H             ;读 8255的 PA口
        MOV     [BX], AL
        INC     BX
        DEC     CX
        MOV     AL, 20H
        OUT     78H, AL             ;正常 EOI
        POP     AX
        IRET
INTERRUPT_ SERVE   ENDP
CODE    ENDS
```

说明：

⑴ 设置数据缓冲区 BUFFER,主程序中设置数据缓冲区指针寄存器 BX指向缓冲区首地址。

⑵ 主程序中,编程使 8253通道 0工作于方式 1,计数器初值为 1 MHz/8 kHz=125。

⑶ 每次 A/D变换完成,EOC上升沿引起中断,中断服务程序中读 8255的

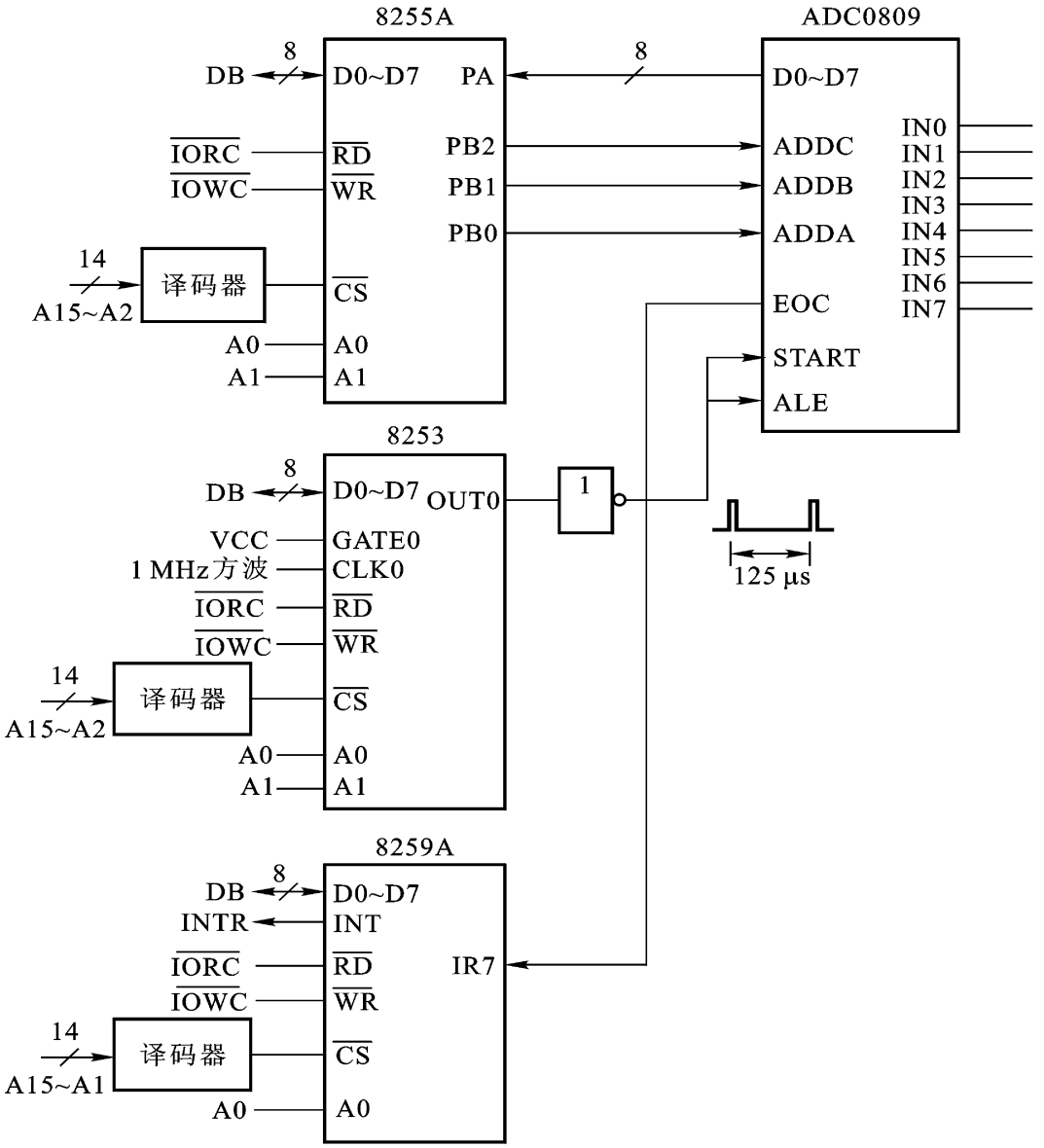


图 11.8　一个由 ADC0809、8255、8253及 8259组成的数据采集系统

PA口，读到的数据存入数据缓冲区，并修改数据缓冲区指针寄存器 BX和计数器 CX。

⑷ 中断开放后，每次中断服程序对 CX减 1。主程序不断检查 CX是否变为 0，若等于 0即表明执行了 8 000次中断。

【例 11.3】　AD7472与 8086系统的连接如图 11.9所示。AD7472的 12位数据线 DB_{11} ～DB_0与 CPU的数据总线相连，$\overline{RD}$接 $\overline{IORC}$。设译码器 1在 A_{15} ～A_1 =0000000001000000时输出为 0，与 A_0、$\overline{BHE}$配合，只有在 CPU执行 16位输入指令且地址等于 80H时，AD7472才被选中。当执行 16位的 I/O指令

时,可在一个总线周期内读取 16位数据。

由 8255作 AD7472的控制、状态接口。 AD7472的启动信号 $\overline{CONVST}$接 PC_7,状态信号 BUSY接 PC_0。因此 8255的 PC口高 4位应设置为输出,PC口低 4位应设置为输入。设 8255的地址为 90H～93H。

模拟信号接 AD7472的 VIN。时钟信号 CLKIN为 10 MHz。

设系统中定时器每 100 μs中断一次,下面的定时器中断服务程序完成一次数据采集并存放于内存中 BUFFER中。主程序应对 8255、8253等编程,并对采集的数据进行处理 (主程序略)。

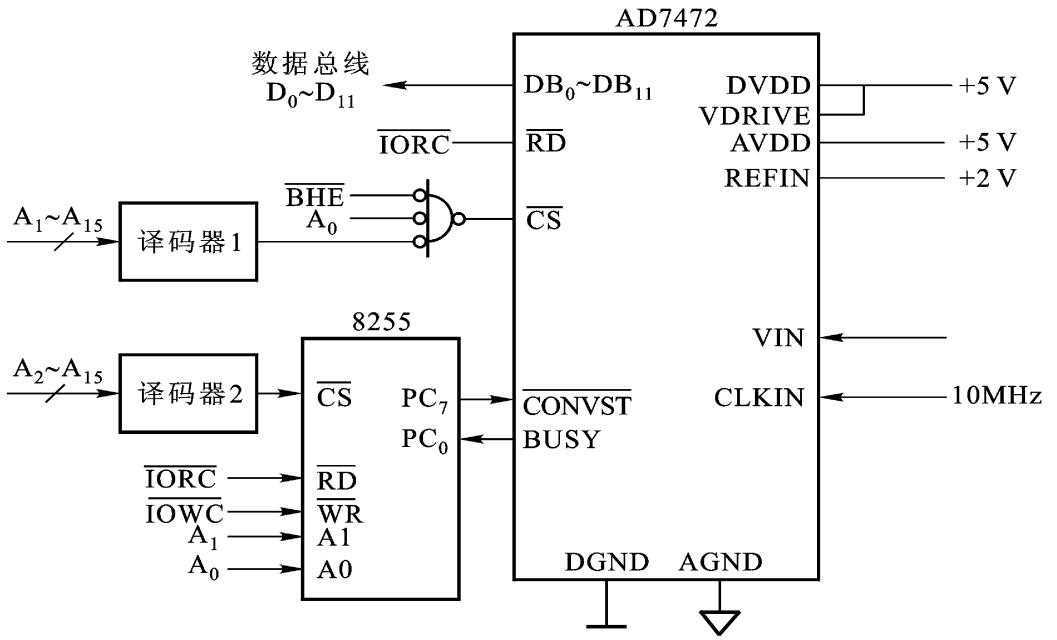


图 11.9 AD7472应用电路图

中断服务程序:

```
TIMER_INT  PROC  FAR
           PUSH  AX
           MOV   AL,00H
           OUT   92H,AL          ;CONVST=0,启动 A/D
           MOV   AL,80H
           OUT   92H,AL          ;CONVST=1,
WAIT:      IN    AL,92H
           AND   AL,01H          ;保留最低位
           JNZ   WAIT            ;若 BUSY为 1,则等待
```

```
                                    ;
        IN      AX,80H              ;16位数据读命令
        AND     AX,0FFFH            ;保留低 12位
        MOV     BUFFER,AX           ;存入内存
                                    ;
        MOV     AL,20H
        OUT     20H,AL              ;正常 EOI,20H为 8259的地址
        POP     AX
        IRET
TIMER_INT ENDP
```

11.2 D/A转换器及其应用

11.2.1 D/A转换的主要性能参数

1. 分辨率

DAC的分辨率是指 D/A转换器模拟输出电压能够被分离的等级数,输入数字量的位数越多,输出电压可分离的等级越多,因此我们用输入数字量的二进制位数来表示分辨率。在实际应用中,人们往往使用可分辨的最小输出电压与最大输出电压之比表示 DAC的分辨率,如 8位 DAC的分辨率为 1/255,就说该 DAC的分辨率是 8位。

2. 建立时间

当 DAC输入由最小的数字量变为最大的数字量时,DAC的输出达到稳定所需要的时间称为 DAC的输出建立时间。建立时间反映了 DAC的转换速度。

11.2.2 典型 D/A转换器介绍

1. 8位 D/A转换器 DAC0832

(1) 概述

DAC0832是一种 8位双缓冲型 D/A转换器,CMOS工艺。内部阶梯电阻网络形成参考电流,由输入二进制数控制 8个电流开关,CMOS的电流开关漏电很小,保证了转换器的精度。 DAC0832使用单一电源,功耗低。从输入数据到输

出电压稳定所需的时间即建立时间为 1 μs。

输入数据为 8位并行输入,有两级数据缓冲器及使能信号、数据锁存信号等,与 CPU接口方便。DAC0832的内部结构图和引脚编号见图 11.10。

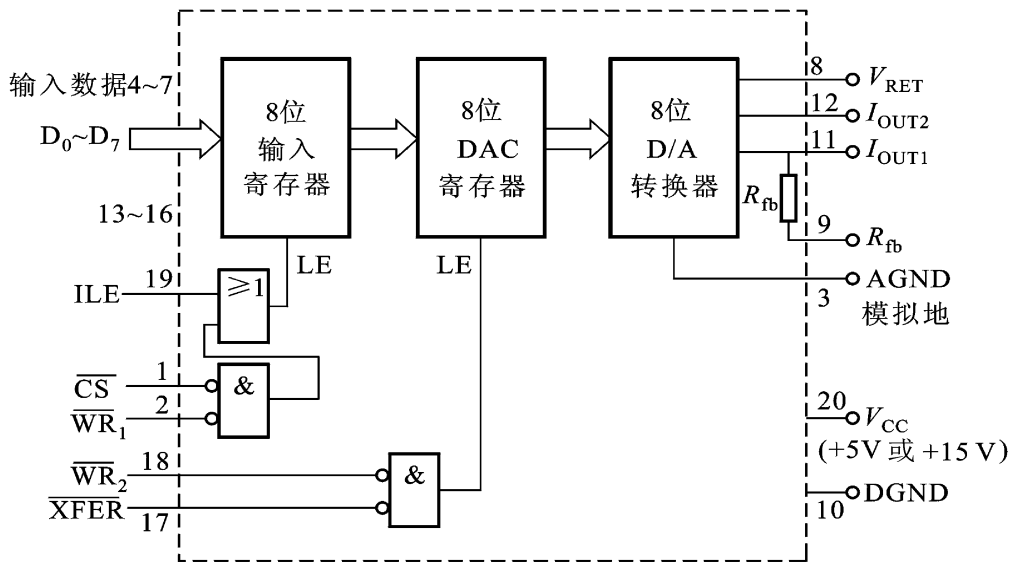


图 11.10 DAC0832的内部结构和引脚编号

(2) 引脚说明

D_0 ~D_7——数据线,输入数字量。

$\overline{CS}$——第一级数据缓冲器的片选信号,低电平有效。

$\overline{WR_1}$——第一级数据缓冲器的写信号,低电平有效。

ILE——允许输入锁存,高电平有效。

$\overline{WR_2}$——第二级数据缓冲器的写信号,低电平有效。

$\overline{XFER}$——传送控制信号。

I_{OUT1}、I_{OUT2}——DAC输出电流,若需要电压输出,要通过运算放大器进行电流—电压转换。

R_{fb}——供电流—电压转换电路使用的反馈电阻。由于它是与 DAC中的权电阻网络一起制造的,因此具有同样的温度系数,使用该反馈电阻可使电流—电压转换电路的电压输出稳定,受温度变化的影响小。

V_{RET}——基准电压输入端,允许范围 −10~+10 V。

V_{CC}——逻辑电路的电源,允许范围 +5~+15 V。

AGED——模拟电路的地。

DGND——数字电路的地。

(3) DAC0832的输出电路

1) 单极性电压输出

典型的单极性电压输出电路见图 11.11a,由运算放大器进行电流—电压转换,使用芯片内部的反馈电阻。输出电压 V_{OUT}与输入数字 D的关系为:

$$V_{OUT} = -V_{REF} \times D/256$$

$$D=0\sim255,\ V_{OUT}=0\sim-V_{REF}\times255/256$$

当 V_{REF}等于 −5 V时,V_{OUT}的输出范围是 0～+ (255/256) V ;当 V_{REF}等于 +5 V时,V_{OUT}的输出范围是 0～− (255/256) V。

2) 双极性电压输出

在有些应用中要求输出模拟电压是双极性的,这就需要用转换电路来实现。图 11.11b是能够输出双极性电压的电路,其中电阻 $R_2 = R_3 = 2R_1$。

$$V_{OUT} = 2\times V_{REF}\times D/256 - V_{REF}$$

$$= (2D/256 - 1)\ V_{REF}$$

当 $D=0,\ V_{OUT} = -V_{REF}$

当 $D=128,\ V_{OUT}=0$

当 $D=255,\ V_{OUT} = (2\times255/256-1)\times V_{REF} = (254/255)\cdot V_{REF}$

即输入 D=0～255,输出电压在 $-V_{REF}$～$+V_{REF}$之间变化。

(4) DAC0832的工作方式

直通方式:把 $\overline{CS}$、$\overline{WR1}$、$\overline{WR2}$、$\overline{XFER}$接地,即第一级、第二级数据缓冲器都直通。数据一旦加在数据线 (D_7～D_0)上,DAC的输出就立即响应。

单缓冲方式:适用于系统中只有一路 DAC,或有多路 DAC但不需要同步的情况。有两种方法使 DAC0832工作于单缓冲方式。

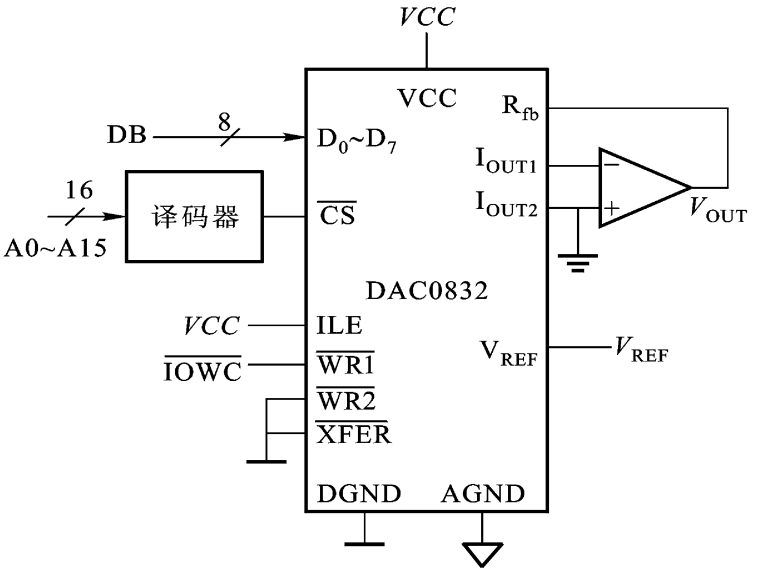
把$\overline{WR2}$、$\overline{XFER}$接地,即第二级缓冲器直通。数据由 $\overline{CS}$、$\overline{WR1}$和 ILE控制写入第一级数据缓冲器。图 11.11a b中 DAC0832与 CPU就是采用的这种方法。

把$\overline{CS}$、$\overline{WR1}$接地,ILE接高电平,即第一级缓冲器直通。数据由$\overline{WR2}$和 $\overline{XFER}$控制写入第二级数据缓冲器。

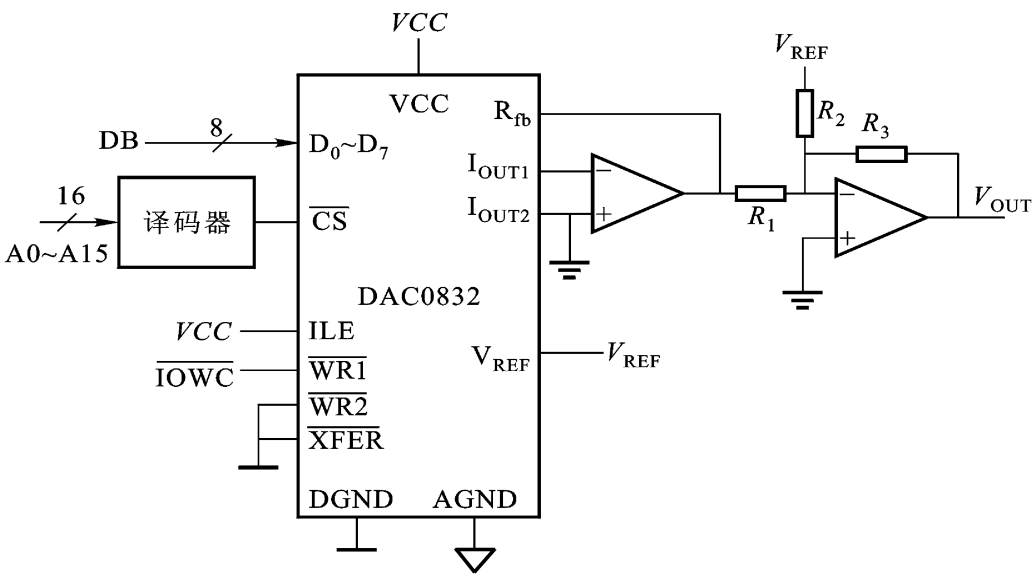
双缓冲方式:适用于系统中有多片 DAC0832,并且需要同步的情况。使用时,多片 0832的$\overline{WR2}$、$\overline{XFER}$并联在一起。先分别将每一路的数据写入各个芯片的第一级数据缓冲器,然后同时将数据锁存到每一片 0832的第二级数据缓冲器。

2. 12位 D/A转换器 AD5341

AD5341是高性能的单通道 12位 D/A转换器,并行数据输入,两级缓冲器结构。内含电流/电压转换,输出为电压信号,简化了应用电路。具有增益控制



(a) 单极性电压输出电路



(b) 双极性电压输出电路

图 11.11

端,可以选择增益为 1或 2。单一电源,电压范围 2.7~5.5 V。低功耗,5 V电源时的电流为 140 μA。

AD5341采用 20脚 TSSOP (ThinShrink SmallOutline Packages)封装,顶视图见图 11.12。

(1) 引脚图

V_{DD}是电源输入,允许范围是 2.5~5.5 V。

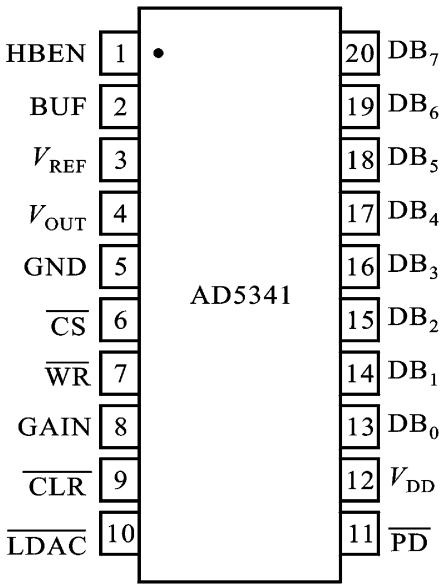


图 11.12　AD5341的引脚

DB_0 ～DB_7为数据线，HBEN为高字节有效信号。当 HBEN为低电平，写入 AD5341的是低字节，当 HBEN为高电平，写入 AD5341的是高字节。$\overline{WR}$是写信号，$\overline{CS}$是片选信号。

$\overline{LDAC}$是将输入寄存器数据装入 DAC寄存器的控制信号。

$\overline{CLR}$是异步清零端。

V_{REF}是参考电压输入端。BUF是一个控制参数输入端，其作用是控制参考电压 V_{REF}进入 DAC前是否需要使用缓冲器，参见图 11.13。当使用参考电压输入缓冲放大器时，V_{REF}端有很高的输入阻抗。当 BUF选择 1，表示使用参考电压输入缓冲放大器，这时允许的 V_{REF}输入范围是 1 V～V_{DD}。当 BUF选择 0，表示不使用参考电压输入缓冲放大器，这时允许的 V_{REF}输入范围是 0.5 V～V_{DD}。

V_{OUT}是 D/A变换后经过电流—电压转换放大器后的电压输出端。GAIN为增益控制端，它决定输出的电压范围是 0～V_{REF}或 0～2V_{REF}。

$\overline{PD}$是休眠模式控制端，低电平有效，当系统暂时不使用 AD5341时，可以使它进入休眠模式。当 V_{DD}为 5 V时，$\overline{PD}$ =0，AD5341的电源消耗只有 200 nA。

⑵ 内部结构

AD5341内部结构如图 11.13所示。AD5341接口逻辑实现与微处理器的接口，可输入 D/A变换的数据，进行增益控制等操作。内部有两级 12位寄存器，第一级为输入寄存器，第二级为 DAC寄存器。当 12位的数据分为低字节和高字节分别送入第一级寄存器后，再一并装入第二级 DAC寄存器。DAC寄存器

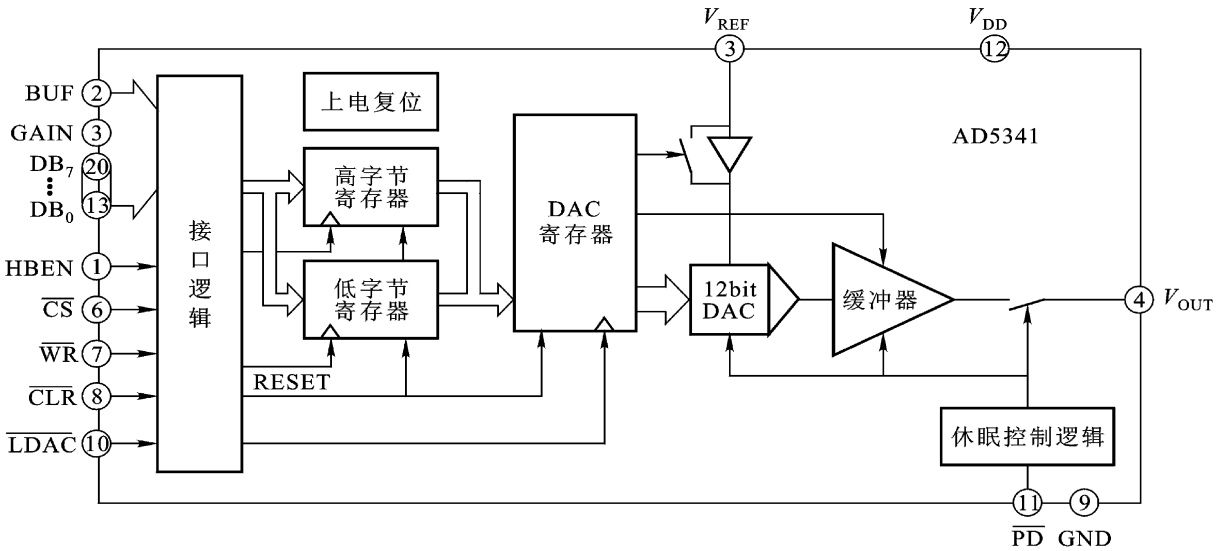


图 11.13 AD5341的内部结构

中的数据经 12位 D/A变换成为电流信号,经内部的运算放大器后转换为电压由 V_{OUT}输出。

(3) 数字—电压转换关系

AD5341的输出为电压信号,理想的数字—电压转换关系为:

$$V_{OUT}=V_{REF}\times D\times Gain/2^{12}$$

其中,D为输入的数字量,D=0~4 095;Gain等于 1或 2。

可见,AD5341的输出电压为正,输出电压随输入量的增加而增加,范围为 0~V_{REF}或 0~2V_{REF}。

(4) AD5341与 CPU的接口

AD5341的数据分为高 8位和低 8位,其意义见图 11.14。

高字节

BUF	GAIN	×	×	DB11	DB10	DB9	DB8

低字节

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0

图 11.14 AD5341的数据格式

AD5341的引脚 DB0~DB7与系统的数据总线相连,BUF与数据总线的 D_7相连,GAIN与数据总线的 D_6相连。这样,CPU送低字节数据时,HBEN=0,由数据总线送 AD5341;CPU送高字节数据时,将 BUF、GAIN并接在 D_7、D_6位,见图

11.13,使 HBEN＝1,由数据总线写入 AD5341。BUF位等于 0或 1,可选择不需要或需要使用参考电压输入缓冲放大器。GAIN位等于 0或 1,选择增益为 1或 2。

一个典型的应用电路如图 11.15所示,参考电压 V_{REF}为 2 V。A0接 HBEN,$\overline{Y_0}$接 $\overline{CS}$,$\overline{Y_1}$接 $\overline{LDAC}$,$\overline{Y_2}$接 $\overline{CLR}$。

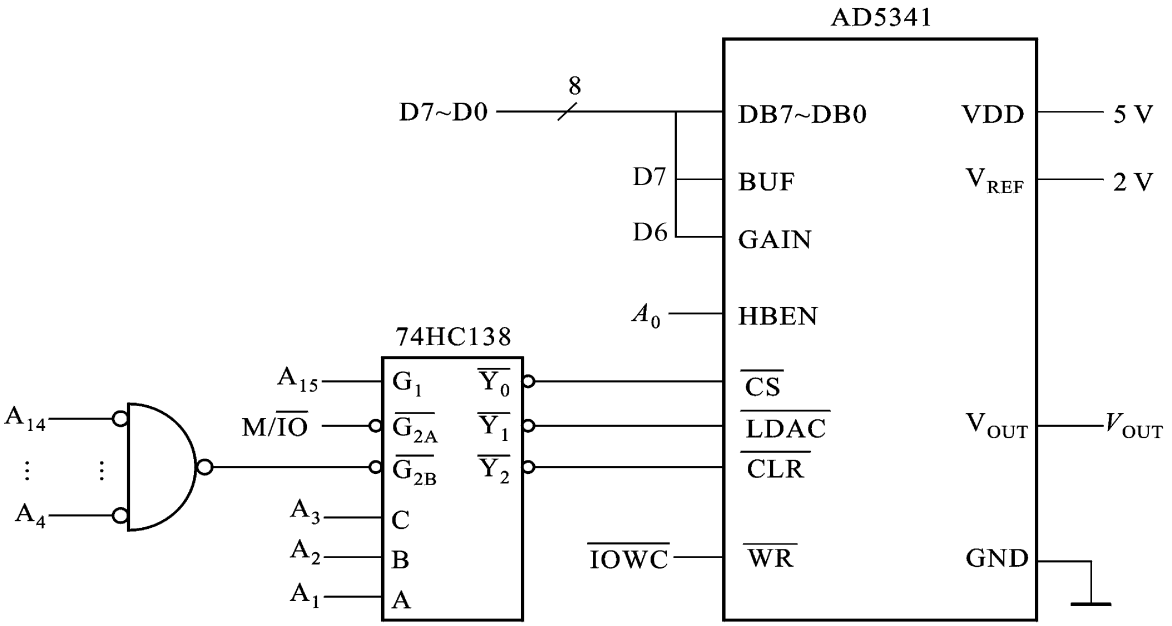


图 11.15　AD5341典型应用电路

当 A_{14} ~A_4全为 0时,$\overline{G2B}$＝0。A_3、A_2、A_1接 74HC138的 C、B、A,因此当 I/O地址为 8000H或 8001H时 $\overline{Y_0}$等于 0,芯片被选中。因为 A_0接 HBEN,当 A_0 ＝0时写低字节,A_0 ＝1时写高字节。

当 I/O地址为 8002H或 8003H时,$\overline{Y_1}$等于 0,即 $\overline{LDAC}$＝0,将输入寄存器内容装入 DAC寄存器,对 DAC寄存器进行数据更新。该操作只要以 I/O指令访问其中一个地址即可,不论执行输入还是输出指令。

当 I/O地址为 8004H或 8005H时,$\overline{Y_2}$等于 0,即 $\overline{CLR}$＝0,对 DAC寄存器清 0。同样,该操作只要以 I/O指令访问其中一个地址即可。

相应的 I/O地址分配如下:

写低字节数据到输入寄存器	8000H
写高字节数据到输入寄存器	8001H
输入寄存器内容装入 DAC寄存器	8002H (或 8003H)
DAC寄存器清 0	8004H (或 8005H)

11.2.3 应用举例

[例 11.4] 一个由 DAC0832实现的 8位 D/A转换电路见图 11.11a,设 0832的地址为 5AH,基准电压 VREF＝－5 V。试编写程序使其输出锯齿波,并画出输出波形图。

程序:

```
        MOV     AL, 00H
AGANT:  OUT     5FH, AL     ;数据送 D/A的数据口
        CALL    DELAY       ;延时
        INC     AL          ;AL加 1, 当 AL由 255加 1时,AL回到 0
        JMP     AGANT
DELAY:  MOV     CX, 10      ;延时子程序
DELAY1: LOOP    DELAY1
        RET
```

输出波形见图 11.16,锯齿波的周期与子程序 DELAY的延时时间有关。

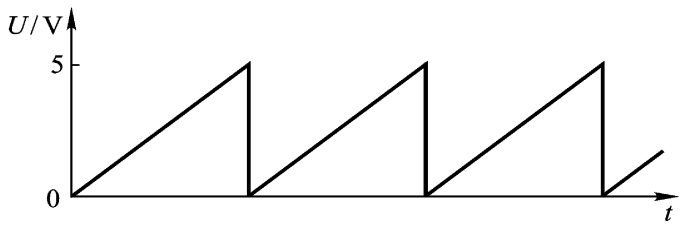


图 11.16 例 11.4输出波形图

[例 11.5] 某 8086系统中有一个由 DAC0832构成的双极性电压输出的 8位 D/A转换电路,见图 11.11b,设 0832的地址为 5AH,基准电压 VREF＝＋1 V。系统中定时器 8253与中断控制器 8259配合,每 100 μs中断一次,试编写中断服务子程序,使其输出三角波,并画出输出波形图。

程序:

```
DATA    SEGMENT             ;数据段
COUNT   DB  0
FLAG    DB  0
```

```
DATA     ENDS
CODE     SEGMENT                ;代码段
                                ;主程序略
         ……
INTSERVE PROC    FAR
         PUSH    AX
         MOV     AL, COUNT
         OUT     5FH, AL        ;数据送 D/A的数据口
         MOV     AL, FLAG
         AND     AL, AL
         JNZ     DECREASE       ;FLAG不等于 0,转减 1处理
                                ;FLAG等于 0,做加 1处理
         INC     COUNT          ;加 1
         CMP     COUNT, 255
         JNZ     NEXT           ;不等于 255则转移
         MOV     FLAG, 1        ;置减 1标志
         JMP     NEXT
DECREASE:                       ;FLAG不等于 0,做减 1处理
         DEC     COUNT          ;减 1
         JNZ     NEXT           ;COUNT不等于 0则转移
         MOV     FLAG, 0        ;COUNT等于 0,则置加 1标志
  NEXT:  MOV     AL, 20H
         OUT     20H, AL        ;假设 8259的端口地址为 20H和
                                 21H
         POP     AX
         IRET
INTSERVE EDNP
  CCODE  ENDS
```

输出的幅度为 −1～+1 V,周期为 (255＋255) ×100 μs 波形示意图见图 11.17。

【例 11.6】　某系统中使用 AD5341完成 D/A变换,其电路如图 11.15所示,V_{REF}为 2 V。试编写中断服务程序,将 DATA (字型数据,低 12位有效)输出到 D/A转换器。D/A变换的 GAIN选 0, BUF选 1。

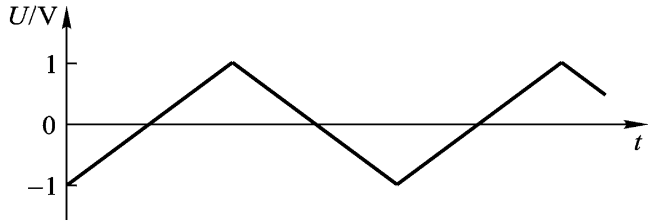


图 11.17 例 11.5输出波形图

程序：

```
INT_SERVE  PROC  FAR
        PUSH    AX
        MOV     AX, DATA
        MOV     DX, 8000H
        OUT     DX, AL      写 DATA的低 8位到 AD5341的低字节输
                            入寄存器
        MOV     AL, AH
        AND     AL, 0FH
        ADD     AL, 80H     ;BUF=1,GAIN=0
        MOV     DX, 8001H
        OUT     DX, AL      写 DATA的高 4位和 BUF、GAIN到高字
                            节输入寄存器
        MOV     DX, 8002H
        OUT     DX, AL      输入寄存器内容装入 DAC寄存器
                            ;访问 8002H端口,与 AL内容无关
        MOV     AL, 20H
        OUT     20H, AL     ;正常 EOI,20H为 8259的地址
        POP     AX
        IRET
INT_SERVE ENDP
```

思考题与习题

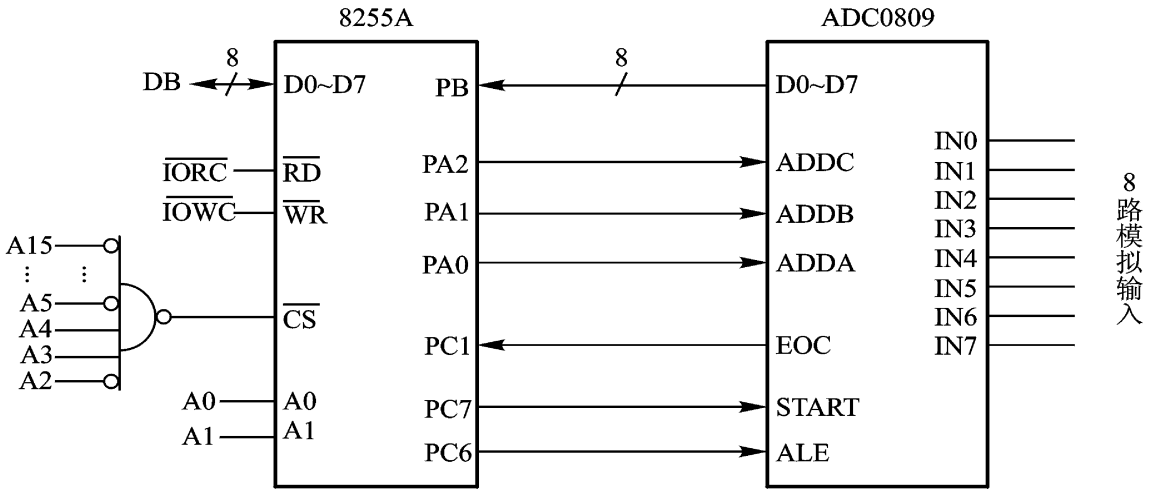
11.1 当输入为双极性信号时,对输入信号的编码主要有哪三种方式？试分别加以说

明。

11.2　试说明并行比较型和逐次比较型 A/D转换器的特点。

11.3　某 8086系统中，A/D转换器 0832只使用一个模拟信号输入通道，试设计其接口电路，并编写子程序。子程序的功能是启动 A/D转换，并以查询的方式读入一个采样数据放在寄存器 DL中。

11.4　某 8086系统中，由 DAC0832构成 A/D转换器，通过 8255与 CPU接口，如题 11.1图所示。试编写程序，以查询的方式巡回采样 8个模拟输入通道，每个通道 16次。采样数据存放在内存中。



题 11.1图

11.5　试设计一个由 AD7472、8255、8253及 8259组成的数据采集系统。AD7472的 D7～D0接 8255的 PA口，D_{11}～D_8接 8255的 PB口。8253通道 0工作于方式 1，OUT0输出的脉冲经反相后接 AD7472的 $\overline{CONVST}$，因此，每个脉冲启动一次 A/D变换。AD7472的 BUSY接 8259的 IR7，下降沿引起中断。然后编写程序（包括主程序和中断服务程序），功能是以 1 MHz的频率采集 1 000个数据，存入内存中。

11.6　某 8位 D/A转换电路见图 11.11a，设 DAC0832的地址为 5AH，基准电压 VREF＝－5 V。试编写程序使其输出梯形波。

11.7　某系统中使用 AD5341完成 12位 D/A变换，其电路如图 11.15所示，VREF为 2 V。试编写程序，使 D/A转换器的 VOUT输出三角波。三角波的幅度为 0到 4 V（对三角波的周期大小不作要求）。

第12章

高性能微处理器

12.1 80286微处理器

12.1.1 80286的内部结构

80286主要由总线部件(BU)、指令部件(IU)、执行部件(EU)和地址部件(AU)4个独立的处理部件构成一个有机的整体,并加强它们之间的并行操作程序,有效地加快了处理速度。80286内部结构图如图12.1所示。

⑴ BU是微处理器与系统之间的一个高速接口,负责管理、控制总线操作。它有效地管理、控制80286与存储器、外部设备的联系。以最高的速率传送数据和预取指令的操作,实现零等待状态,完成对外的读/写。

⑵ IU负责从存储区域中取出指令,送入预取指令队列。该队列是预取器和指令译码器之间的一个缓冲,指令译码器将指令从队列中取出、译码后送入已译码指令队列,并作好供EU执行的准备。IU连续译码,与此同时,EU执行的总是事先由IU译好的指令。这样译码和执行并行操作,改善了流水线功能,从而大大提高了80286的工作速度。

⑶ EU负责执行已译码的指令,按照所需步骤完成微处理器的算术、逻辑运算以及其他数据加工等操作。

⑷ AU由偏移量加法器、段界限值检查器、段基地址寄存器、段长度寄存

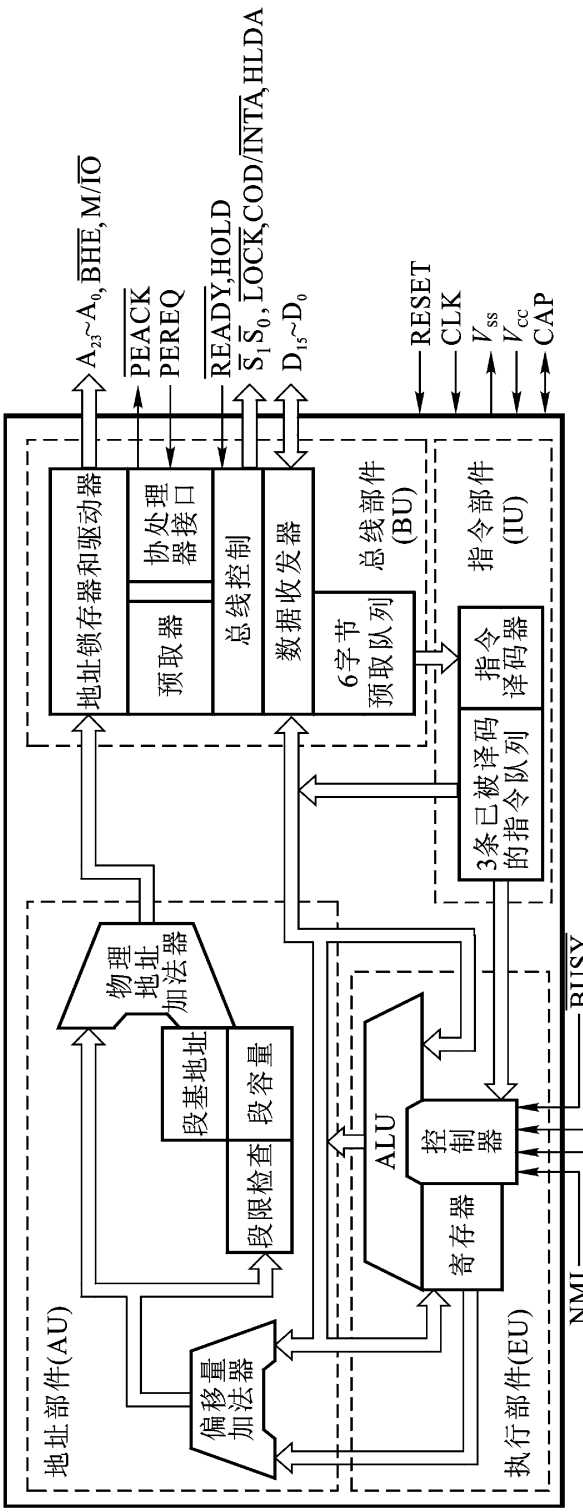


图 12.1 80286 内部结构图

器和物理地址加法器等部件构成,完成执行指令过程中的有关寻址操作。它实施存储器管理及保护功能,计算出操作数据的物理地址,同时检查保护权。在保护方式下,AU提供完全的存储管理、保护和虚拟存储等支持。AU内部有一个高速缓冲寄存器,该寄存器保存着段的基地段、段长界限和当前正在执行的任务所用的全部虚拟存储段的访问权。

12.1.2 80286的寄存器

80286的寄存器组与8086基本相同。同样有8个16位数据寄存器AX, BX, CX, DX, SP, BP, SI, DI。80286的4个段寄存器CS, DS, ES, SS各包括16位段选择器和与之相对应的48位段高速缓冲器(8位存取权域、24位基地址域和16位界限域),共64位,用于逻辑地址到物理地址的转换。

80286的标志寄存器增设了两个标志位段寄存器字段。其中,IOPL字段为特权标志,用来定义当前任务的特权层,即优先权(有0～3四级);NT位为任务嵌套标志,NT=1,表示当前执行的任务嵌套于另一任务中,否则NT=0。

80286新设了16位的机器状态字(MSW),只使用其中的低4位:D_0——保护允许位(PE);D_1——监督协处理器位(MP);D_2——仿真协处理器位(EM)和D_3——任务切换位(TS)。在系统复位时,MSW被置成FFF0H,它使80286处于实地址方式。MSW的保护允许位(PE)用来启动80286进入保护虚拟地址方式,PE=0,表示CPU当前处于实地址方式;PE=1,表示CPU当前已进入保护虚拟地址方式。

80286还增设了4个系统表寄存器:全局描述符表寄存器GDTR、局部描述符表寄存器LDTR、中断描述符表寄存器IDTR、任务状态表寄存器TR。这4个系统表寄存器只在保护方式下使用。

80286的保护方式在存储器中设置了三种类型的描述符表:全局描述符表(GDT, Global Descriptor Table)、局部描述符表(LDT, Local Descriptor Table)和中断描述符表(IDT, Interrupt Descriptor Table)。对应这三个描述符表,GDTR, LDTR和IDTR在保护方式寻址时分别作所对应的描述符表的指针。GDTR和IDTR由24位基地址和16位段界限,共40位组成,LDTR具有与段寄存器相同的位数(64位)。

TR(Task State)是一个64位的寄存器,当任务切换时,它自动地保护和恢复机器状态。TR的16位段选择器字段由CPU运行程序装入16位的段选择字,再由段选择字选择48位的段描述符装入相应的描述符高速缓存器。描述符包含了当前任务状态段的容量、基地址与访问权等信息。

与 8086比较，80286芯片引脚功能最大的差别是不采用地址、数据线复用方式，因此有 68条引脚，封装成四面都有引脚的正方形管壳。

12.1.3　80286的存储器组织

80286是在芯片内部最早实现存储管理和保护的微处理器。80286有实地址管理方式和保护虚拟地址管理方式。保护虚拟地址管理方式可对多任务操作提供可靠的支持。

80286在加电或复位时自动进入实地址方式。可以通过使用指令 LMSW和 SMSW置机器状态字 MSW中的 PE位，使 CPU进行实地址方式 (PE＝0)和保护虚拟地址方式 (PE＝1)的切换。

(1) 实地址管理方式。80826的实地址管理方式与 8086完全相同，用 A_{19}～A_0直接寻址 1MB存储器空间，这时 A_{23}～A_{20}无效。

(2) 保护虚拟地址管理方式。80286的能力在保护虚拟地址管理方式下充分发挥出来了。保护方式是集实地址方式的能力、存储管理、对虚拟存储器的支持，以及对地址空间的保护为一体而建立起来的一种特殊工作方式。80286的保护功能，可以对存储器的段边界、属性及访问权等自动进行检查，通过 4级保护环结构支持任务与任务之间和用户与操作系统之间的保护，也支持任务中程序和数据的保密，从而确保在系统中建立高可靠的系统软件。

在保护虚拟地址方式下，访问存储器的物理地址由 20位扩展为 24位，因此可直接寻址的实存空间扩大为 16MB。

80286的保护虚拟地址机制，为每个任务可提供最大为 1GB的虚拟存储器空间。这是由于系统建立了全局描述符表 (GDT)、局部描述符表 (LDT)和中断描述符表 (IDT)。每个描述符表最多由 2^{13}＝8K个段描述符组成，GDT和 LDT两个描述符表总共可包含 16K个段描述符。每个段描述符指向一个 64KB存储空间的逻辑段。因此，80286的最大虚拟存储器空间为 64KB×16K＝1024MB＝1GB。

80286的虚拟地址方式采用虚拟地址指示器寻址。32位虚拟地址指示器包含高 16位段选择字和低 16位偏移地址 (有效地址)。其中，偏移地址的功能与实地址方式相同，而 16位段选择字是为了进入一个描述符表的偏移量参数。通过它从描述符表中可得到 24位的段基地址，将它与 16位偏移地址相加，形成访问存储器的 24位物理地址，其实现过程如图 12.2所示。

在 16位的段选择字中，用 1位作描述符表选择 (TI)字段，选择当前使用 GDT表还是 LDT表。段选择字用 13位作描述符表偏移地址字段，用来确定当

前使用的段描述符在描述符表中的位置。段选择字用2位作请求特权级(RPL)字段,用来定义当前请求的优先权级别(0～3,0级为最高),特权级也称为保护环,为多任务环境下不同的任务、过程设置优先权,有效地防止系统的混乱。

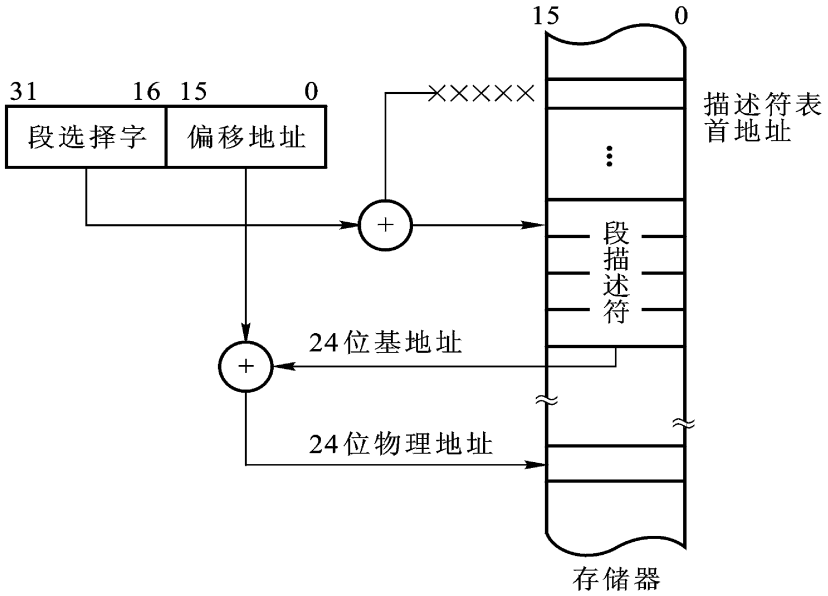


图 12.2 保护虚拟地址方式下的寻址方式

12.2 80386微处理器

如果说,微处理器从8位到16位主要是总线的加宽,那么,从16位到32位,则是从体系结构设计上的概念性的革新。32位微处理器的问世是微处理器发展史的又一里程碑。32微处理器普遍采用流水线技术、指令重叠技术、虚拟存储技术、片内存储管理技术、存储器分段、分页保护技术。这些技术的应用,使32位微机可以更有效地处理数据、文字、图像、图形、语音等各种信息,为实现多用户、多任务操作系统提供了有力支持。

12.2.1 80386的内部结构

80386采用流水线工作方式,其内部结构按功能划分由6大部件组成:总线接口部件(BIU)、指令预取部件(IPU)、指令译码部件(IDU)、指令执行部件(EU)、分段部件(SU)和分页部件(PU)。80386的内部结构如图12.3所示。其中,分段部件SU和分页部件PU统称为存储器管理部件MMU(MemoryManagementUnit)。

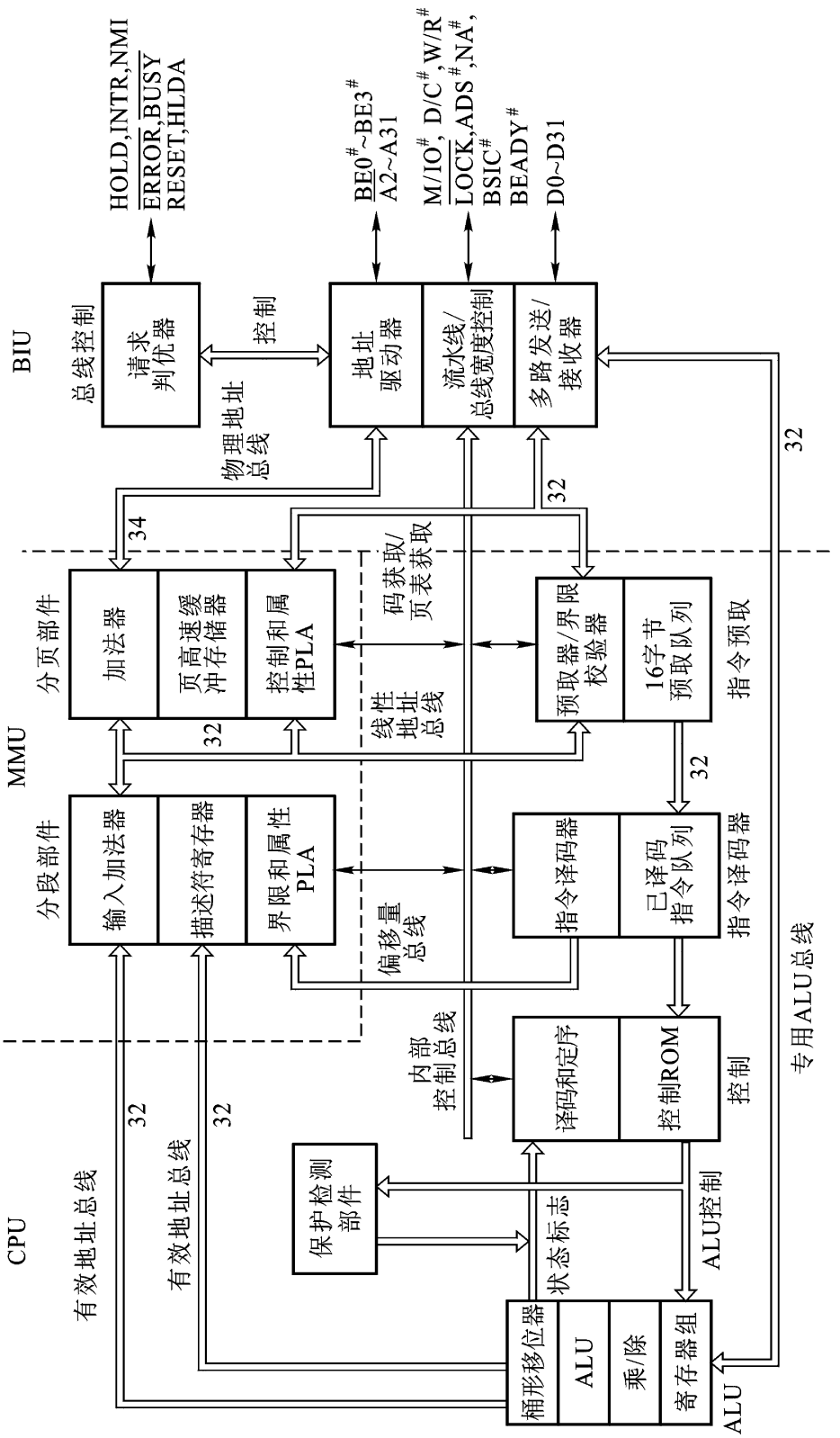


图 12.3 80386 的内部结构

总线接口部件 BIU是微处理器与系统的接口。其功能是:在取指令、取数据、分段部件请求和分页部件请求时,有效地满足微处理器对外部总线的传输要求。BIU能接收多个内部总线请求,并且能按优先权加以选择,最大限度地利用所提供的总线宽度,为这些请求服务。

指令预取部件 IPU的职责是从存储器预先取出指令。它是一个能容纳 16条指令的队列。指令译码部件 IDU的职责是从预取部件的指令队列中取出指令字节,对它们进行译码后存入自身的已译码指令队列中,并且作好执行部件处理的准备工作,如果在预译码时发现是转移指令,可提前通知总线接口部件 BIU去取目标地址中的指令,取代原预取队列中的顺序指令。

执行部件 EU由控制部件、数据处理部件和保护测试部件组成。控制部件中包含着控制 ROM、译码电路等微程序驱动机构。数据处理部件中有 8个 32位通用寄存器、算术逻辑运算器 ALU、一个 64位桶形移位器、一个乘除法器和专用的控制逻辑,它负责执行控制部件所选择的数据操作。保护测试部件用于微程序控制下,执行所有静态的与段有关的违章检验。执行部件 EU中还有一条附加的 32位的内部总线及专门的总线控制逻辑,以确保指令的正确完成。

在由 80386组成的系统中,存储器采用段、页式结构。页是机械划分的,每 4 KB为 1页,程序或数据均以页为单位进入实存。存储器按段来组织,每段包含若干页,段的最大容量可达 4 GB (2^{32})。在 80386中,分段部件根据执行部件的要求,完成有效地址的计算,以实现逻辑地址到线性地址的转换。分页部件将分段部件产生的线性地址转换成物理地址,提供对物理地址空间的管理。一个任务最多可包含 16 K个段,所以 80386可为每个任务提供 64 TB (2^{46})虚拟存储空间。为了加快访问速度,系统中还设置有高速缓冲存储器 (Cache),构成完整的 Cache—主存 —辅存的三级存储体系。

80386的存储管理部件 MMU和其他各部件集成于同一芯片中,可以把从形成有效地址到产生线性地址和物理地址的各个步骤都重叠起来,充分利用流水线与并行执行的优点,且简化了电路设计,降低了系统的复杂性和价格,提高了可靠性和速度。

12.2.2 80386的寄存器

80386共有 34个寄存器,按功能可分为:通用寄存器、段寄存器、指令指针寄存器、状态和控制寄存器、系统地址寄存器、调试寄存器和测试寄存器。

⑴ 通用寄存器。80386有 8个 32位的通用寄存器:EAX, EBX, ECX, EDX, ESI, EDI, EBP和 ESP,都是由 8086中的 16位寄存器扩充而来的,仍然支

持 8,16位的操作,用法和 8086相同。

⑵ 段寄存器。80386设置 6个 16位段寄存器,其中 CS,SS,DS和 ES与 8086中的完全相同,新增加的 FS,GS是两个支持当前数据段的段寄存器。

在保护虚拟地址方式(即支持多任务方式)下,段寄存器称为段选择器,用来存放虚拟地址指示器中的段选择字,它与段描述符寄存器配合实现段寻址。为了实现存储器分段管理,80386把每个逻辑段的基地址(32位)、长度限值、属性等信息定义成一个称为段描述符的 8字节(64位)长的数据结构,把所有的段描述符构置成系统的段描述符表。

80386有 6个段描述符寄存器,它们与 6个段寄存器一一对应。段描述符寄存器和段描述符的结构完全一样。64位的段描述符寄存器对程序员是不可操作的。当一个段选择字被装入段寄存器,系统根据选择字找到所对应的描述符项,同时装入对应的段描述符寄存器。这样,只要段选择字不变,就不需要到内存中查询描述符表,从而加快了段寻址的速度。

⑶ 系统地址寄存器。80386和 80286一样设置了 4个专用的系统表地址寄存器 GDTR,LDTR,IDTR和 TR,用于保存保护方式下所需要的有关信息。

80386设置了三种描述符表,即全局描述符表 GDT、局部描述符表 LDT、中断描述符表 IDT。前两个定义了系统中使用的所有的(最多可有 8K个)段描述符,IDT则包含了指向多达 256个中断程序入口的中断向量描述符。实际上,这些表是长度为 8~64 KB的数组,段寄存器中的选择字的高 13位就是所对应的描述符在表中的索引地址。

GDTR和 IDTR均为 48位寄存器,GDTR用来存放 GDT的 32位基地址和 16位段长限值,IDTR存放 IDT的 32位基地址和 IDT的 16位段长限值。LDTR和 TR均是 16位寄存器,LDTR存放 LDT的段选择字,而 TR用来存放任务状态段表的段选择字。

⑷ 指令指针和标志寄存器,80386设置了一个 32位的指令指针 EIP和一个 32位的标志寄存器 EFLAGS。EIP是 IP的扩充,它可直接寻址 4GB的实存空间。EFLAGS寄存器的低 16位与 80286标志寄存器完全相同,高 16位目前只设置了两个新的标志:虚拟方式标志位 VM(D_{17})和恢复标志位 RF(D_{16})。若 VM=1,表示 80386是在虚拟 8086方式。若 RF=1,表示下边指令中的所有调试故障都被忽略。

⑸ 控制寄存器。80386设置了 4个 32位的控制寄存器:CR_0,CR_1,CR_2和 CR_3。它们和系统地址寄存器一起,保存着全局性的机器状态,主要供操作系统使用。

⑹ 调试寄存器。80386有8个32位的调试寄存器DR_7～DR_0。DR_7用来设置允许或禁止断点调试的控制;DR_6用于指示断点的当前状态;DR_3～DR_0用于设置4个断点:DR_5,DR_4保留待用。

⑺ 测试寄存器。80386有两个32位测试寄存器TR_7和TR_6。TR_7用来保留存储器测试所得的数据;TR_6为测试控制寄存器,存放测试控制命令。

80386有一个在80286基础上增强和扩大的、功能很强、高度灵活的指令系统,共有152条指令,分为9类:数据传输指令、算术逻辑运算指令、移位循环指令、串处理指令、位处理指令、控制转换指令、高级语言支持指令、操作系统支持指令和处理器控制指令。指令可处理的数据类型有:整数、序数(无符号数)、BCD数、浮点数、串和位串等,覆盖了大多数高级语言中使用的数据类型。

指令分别有16位和32位寻址方式,其中16位寻址方式与8086,80286的寻址方式完全相同。32位的寻址方式有一个重要的特点:通用寄存器都可作为基址寄存器使用,通用寄存器(除ESP外)也可作为变址寄存器使用,并可乘上一个比例因子(1或2或4或8),特别方便对数组的处理。

12.2.3 80386的工作方式

80386有高性能的存储管理部件MMU,有力地支持了三种工作方式:实地址方式、虚拟地址方式和虚拟8086方式。

⑴ 实地址方式:80386在加电或复位初始化时进入实地址方式,这是一种为建立保护方式做准备的方式。它与8086,80286相同,由16位段选择字左移4位与16位偏移地址相加,得到20位物理地址,可寻址1MB存储空间。这时,段的基地址是在4 GB物理存储空间的第一个1 MB内。

⑵ 保护虚拟地址方式。80386的保护虚地址方式是其最常用的方式,一般开机或复位后,先进入实地址方式完成初始化,然后立即转入保护虚地址方式,也只有在保护虚地址方式下,80386才能充分发挥其强大的功能。

80386在保护方式下,存储器用虚拟地址空间、线性地址空间和物理地址空间三种方式来描述,可提供4 GB实地址空间,而虚拟地址空间高达64 TB。在保护方式下,80386支持存储器的段页式结构,提供两级存储管理。

80386支持两种类型的特权保护:通过给每个任务分配不同的虚拟地址空间,可实现任务之间的完全隔离;在同一个任务内,定义4种执行特权级别,高特权级的代码可以访问低特权级的代码,由此实现了程序与程序之间、用户程序与操作系统之间的隔离与保护,为多任务操作系统提供了优化支持。

⑶ 虚拟8086(V86)方式。虚拟8086方式又称为V86方式。80386把标

志寄存器中的 VM标志位置 “1”,即进入 V86方式,执行一个 8086程序,把 VM复位,即退出 V86方式而进入保护方式,执行保护方式的 80386程序。

一般情况下,80386的实地址方式主要是为初始化使用的,它还可为运行保护方式所需的数据结构做好配置和分配。真正运行 8086程序往往用 V86方式。80386把它称为 VM86任务。在 V86方式中,实现虚拟化的办法是把有关存储器、输入/输出的指令进入陷阱处理,并且使一种称为虚拟机的监控制程序对它们进行仿真。V86方式下允许使用分页方式,将 1 MB分为 256个页面,每页 4 KB。

V86方式是 80386设计的一个重要的特点,它可以使大量的 8086程序有效地与 80386保护方式的程序并行运行,从而达到 8086、80286和 80386的多任务并行操作。

12.2.4　80386的存储器管理

80386在保护虚拟地址方式下,采用分段、分页两级综合的存储管理,用分段管理组织其逻辑地址空间的结构,用分页管理来管理其物理存储。80386的分段部件把程序的逻辑地址变换为线性地址,进而由分页部件变换为物理地址。这种段管理基础上的分页管理是 80386所支持的最全面、功能最强的一种存储管理方式。由于微处理器内还设置高速缓冲存储器 (Cache) 和其他功能部件,使得这种两级地址转换的速度很快。

(1) 分段管理。80386的分段管理与 80286类似。80386的段描述符也为 8 Byte,段基地址扩大到 32位,段限值扩大到 1 MB,增添了 4位语义控制字段。80386的段描述符的格式如图 12.4所示:

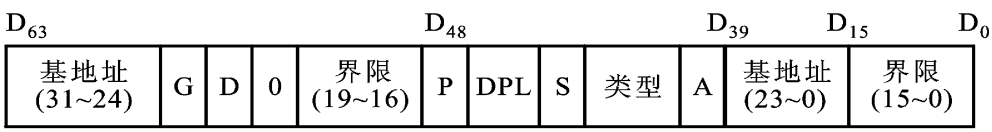


图 12.4　80386段描述符格式

80386有两种主要的段类型,即系统段和非系统段 (代码段和数据段)。段描述符中 S位 (段位) 判别某一给定段是系统段还是代码段或数据段。若 S=1,则为代码段和数据段;若 S=0,则为系统段。段描述符各字段的意义如下:

基地址　32位,指出段基地址;

界限　20位,指出段的长度的限值,表明段最大可为 1MB;

P　1位,存在位,P=1在内存,P=0不在内存;

DPL 2位,描述符特权级,其值为 0～3;

S 1位,段描述符。S=0为系统描述符,S=1为代码或数据段描述符;

类型 3位,指出段类型;

A 1位,已访问位,A=1表示已访问过;

G 1位,组织位,G=1,段长度以页面为单位,G=0,段长度以 Byte为单位;

D 1位,代码段默认操作长度,D=1为 32位代码段,D=0为 16位代码段;

O 2位,备用段,考虑与将来的处理机兼容,这两位必须为零。

80386的虚拟地址指示器提供 48位地址指针;16位段选择符和 32位的偏移量。16位段选择符由三字段组成:最低 2位为 RPL,表示请求者的特权级别,共有四级;接着 1位为 TI,是描述符表的指示符,若 TI=1,表示选中局部描述符表 LDT,若 TI=0,表示选中全局描述符表 GDT;最高 13位为描述符表的偏移量,它和 TI组合选中段描述符。

16位段选择字中用 14位作段描述符的寻址,加上偏移量的 32位寻址,80386为每个任务可提供 14+32=46位,即 64 TB逻辑地址的寻址能力。逻辑地址通过分段部件转换得到 32位线性地址,可用来寻址 4 GB物理空间。

分段存储管理就是要根据逻辑地址提供的段选择符和偏移量,通过段选择符从描述符表中找到相应的描述符,从描述符中取得段的基地址(32位),加上逻辑地址提供的偏移量(32位),形成 32位的线性地址。图 12.5所示,表明了分段存储管理的地址转换过程。

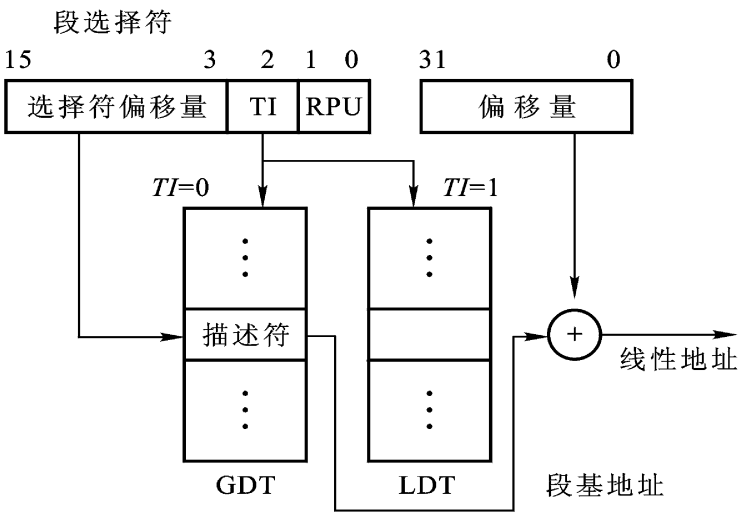


图 12.5 逻辑地址到线性地址的转换

分段部件使用保存在段描述符中的信息,实现保护校验,例如,检验使用段的权能,检测即将执行的数据。如果出现保护冲突,分段部件将出现一次事故跟踪。

(2) 分页管理。80386的物理存储器组织成若干个页面(一般每个页面为 4 KB)。

80386分页采用了页目录表、页表两级页变换机制,低一级的页表是页的映像,由若干页描述符组成,每一个页描述符指示一个物理页面;高一级的页目录表是页表的映像,由若干页目录描述符组成,每一个页目录描述符指示着不同页表,由 80386的页目录基地址寄存器 CR_3 指示页目录表在存储器中的位置。80386的页表和页目录表中最多可分别包含 2^{10}个页描述符和页目录描述符,每个描述符均由 4 Byte(32位)组成,其格式也基本相同。其中:

P (D_0)为存在位,页面或页表装入存储器时,P=1,否则 P=0;

R/W (D_1)为读/写控制位,R/W =1写,否则为读;

U/S (D_2)为用户/监控位。U/S=1用户操作,否则为监控操作;

A (D_5)为访问位,对该页面或页表进行过读写访问时,A=1;

D (D_6)为出错位,D=1出错,否则未出错;

SYSTEM (D_{11} ~D_9)为系统位,留给系统使用;

页面地址指针或页表地址指针 (D_{31} ~D_{12})分别是对应的页面或页表的基地址。

80386的页面和页表均起始于存储空间的 4 KB界上,因此,页面地址和页表地址的低 12位为全 0。在 80386分页系统中,由 CR3给出页目标表的基地址,利用 32位线性地址的高 10位在页目录表的 1 024个页目录描述符中选定 1个,从而获得对应页表的基地址;利用线性地址的中间 10位,在对应页表的 1 024个页描述符中选定 1个,得到页面地址;利用线性地址的最低 12位可在指定页面的 4 KB中选中一个物理存储单元,实现了从线性地址到物理地址的转换。这种地址转换是标准的二级查表机构,如图 12.6所示。

在这个分页系统中,通过页目录表可寻址多达 1 K个页表,每个页表可寻址多达 1 K个页面,因此可寻址 1 M个页面,而一个页面有 4 KB,即可寻址 80386整个物理空间 4 GB(4 KB×1 M)。

(3) 高速缓冲存储管理。为了加快段内地址转换速度,在 80386芯片上有高速缓冲存储器(Cache),可把当前段描述符存入 Cache中,在以后进行的地址转换中,就不用再访问描述符表,而只与 Cache打交道,这样就大大地提高了地址转换的速度。

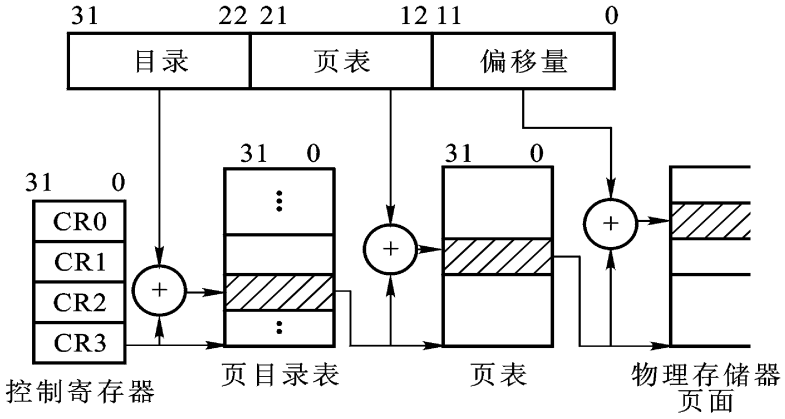


图 12.6 80386的分页机构

分页系统也支持 Cache,把最新、最常用的页表项目,自动保存在称为转换后备高速缓存 (TLB) 中。TLB共可保存 32个页表信息,32个页与对应页的 4 KB相联系,这样就覆盖了 128 KB的存储器空间。对于一般的多任务系统来说,TLB具有大约 98% 的命中率,也就是说在处理器访问存储器过程中,只有 2%必须访问两级分页机构,所以加快了地址转换的速度。

12.3 80486微处理器

Intel 80486是 Intel公司在 1989年推出的新一代 32位微处理器,是 80386的升级产品。80486相当于以 80386为核心,除包含在片内的 8 KB高速缓存 (Cache) 和相当于 80387的数值协处理器之外,还采用了易于构成多处理器系统结构的机制。这是 80486结构上的重大变革,从而使它的整体性能有了很大提高。在相同的工作频率下,其处理速度比 80386提高了 2～4倍,实现了高速度化和支持多处理器系统设计目标。

12.3.1 80486的内部结构

80486基本上沿用了 80386的体系结构,以保持与 80X86微处理器系列在机器码级上的兼容性。80486由 8个基本部件组成:总线接口部件、指令预取部件、指令译码部件、执行部件、控制部件、存储管理部件、高速缓存部件 Cache和高性能浮点处理部件 FPU,其中后两个部件是在 80486在 80386的基础上新增的。80486的内部总线有 32,64,128位三种。80486的内部结构如图 12.7所示。

80486寄存器组包含 80386的全部寄存器组,再加上 80387中的全部 FPU寄存器。只是 80486对标志寄存器和控制寄存器进行了扩充定义。

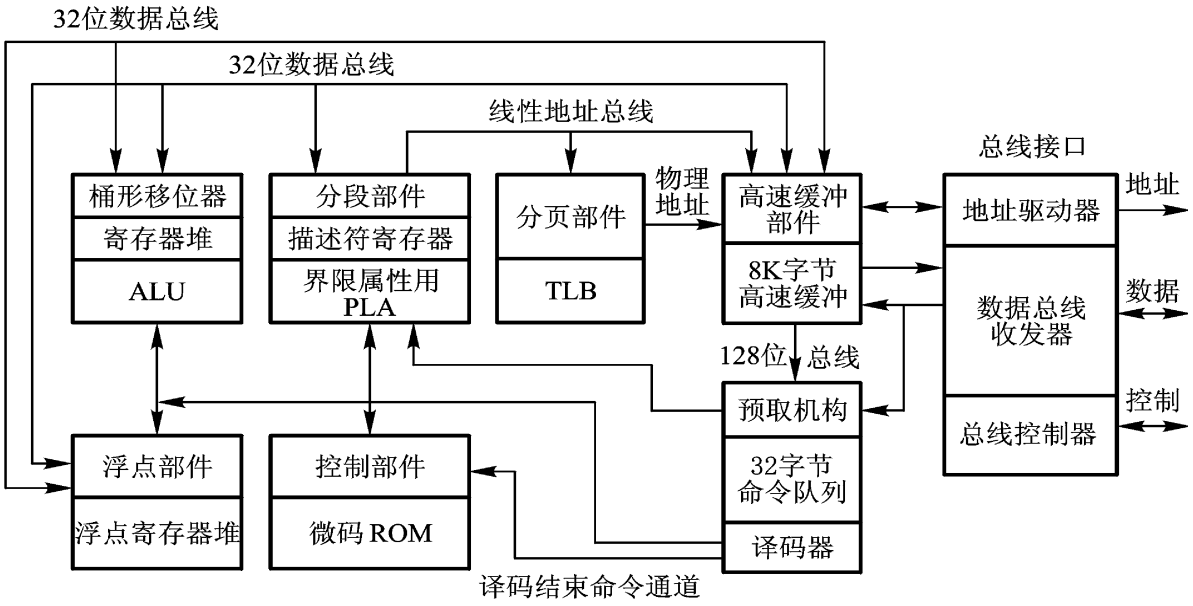


图 12.7　80486的内部结构

12.3.2　80486的技术特点

80486在 Intel微处理器的历史上首次采用了 RISC技术,有效地优化了微处理器的性能。80486采用 RISC技术并不意味着与 80386等 CPU不兼容,实际上指令也并没有精简,强调的只是 RISC技术,目的是使 80486达到 1个时钟周期执行 1条指令。目前 80486已超过了这一设计目标,平均 1个时钟周期执行 12条指令。

80486采用了突发总线 (Burst Bus)同外部 RAM进行高速数据交换。通常 CPU与 RAM进行数据交换时,取得一个地址,交换一个数据,再取得一个地址,又交换一个数据。而采用突发总线后,每取得一个地址,便将这个地址及其后地址中的数据一起参与交换,从而大大加快了 CPU与 RAM之间的数据传输率。这种技术尤其适用于图形显示和网络运用。因为在这两种情况下,所涉及的地址空间一般都是连续的。

80486配置了 8 KB的高速缓冲存储器 Cache。该高速缓存采用 4路相连的实现方案,具有较高的命中率 (约为 92%)。高速缓存由指令和数据共用,当执行某些不涉及数据访问的指令时,整个缓存都为指令所用;反之,当用简单的循

环来处理大量的数据时,则将高速缓存的大部分空间用来存放数据,这样既充分利用了缓存的空间,同时提高了缓存的命中率。若在高速缓存中未找到所需数据,可访问外部存储器,外部存储器与高速缓存间采用成组传送方式,平均每个时钟周期可传送 4 Byte。80486高速缓存采用的替换策略是“近期最少使用”(LRU)策略。

80486片内设置了一个数值协处理器,这就使得 80486不再需要片外数值处理器 80387的支持,而直接有浮点数据处理能力,从而缩短了 CPU与数值协处理器之间的通信时间,提高了浮点处理能力。该片内数值协处理器以极高的速度进行单精度或双精度的浮点运算,保持了与 80387的二进制兼容性,且浮点处理命令也完全一致。在相同的时钟频率下,80486的指令执行速度比 80386系统高出 2~3倍 。

此外,80486在内部高速缓存部件与协处理器之间设置有两条高速数据总线,这两条 32位的总线也可作为一条 64位的总线使用。高档 80486芯片的数据总线宽度甚至可达 128位。如此带宽的数据交换通道是微处理器的外部高速缓存和协处理器无法达到的。

为了适应大规模科学计算的需求,便于系统功能的扩充,80486采用了有助于构成多处理器系统的硬件结构,配置了一些构成多处理器系统所必需的功能和信号,使用户能利用 80486方便地构成一个高性能多处理器并行系统。

Intel80486有多种产品,包括 486SX,486DX,486DX2和 OverDrive升级芯片等。

486SX是 80486的入门产品,芯片内不含数值协处理器,它兼有 80486的性能和 80386的价格两大优点。

486DX属 80486的中档产品,芯片中增设了数值协处理器,是 80486标准结构。486DX2是 80486中的高档产品,它是在 486DX的基础上采用倍速技术,使 CPU的执行速度双倍于系统总线的速度,有效地提高了系统的性能。

OverDrive是一种升级芯片,用于对微型机系统升级的处理。厂家通常采用加插件板的办法,对 80486进行廉价单片升级,即在原有 80486的旁边增加一块 OverDrive芯片,从而可使原微处理器的速度提高近一倍。从原理上说 OverDrive也是一个微处理器,但只能用于对 80846系列升级,对 80386系列无效。

12.4 Pentium处理器

1993年 3月。Intel公司推出了新一代名为 Pentium的微处理器 (P5)。它拥有 32位寄存器、64位数据总线和 32位地址线、高性能浮点处理部件和多媒

体处理 MMX部件。采用 0.80 μm制造工艺,支持 60和 66 MHz前端总线速度(FSB),安全工作电压为 5 V。Pentium处理器采用了全新的设计,与 80486相比内部结构也作了很大改进,但是依然保持了和 80X86系列的二进制兼容性,在相同的工作模式上可以执行所有的 80X86程序。片内存储管理单元(MMU)也与 386和 486兼容,可以在实地址模式引导下转入保护模式和虚拟 86模式,其指令集包括了 80486的所有指令,并增加了新的指令。

其下一代产品是一年后推出的 P54,它支持 3.3 V的内核电压,使用了 0.50 μm甚至是 0.35 μm的制造工艺,处理器的时钟频率达到了 75~200 MHz,总线频率 50~66 MHz。P5带有 16 KB的一级缓存。要特别提到的是,这次英特尔首次运用了两个独立的一级高速缓存:8 KB用于数据,另 8 KB用于指令;采用 Socket5和 IA32架构。

英特尔下一个最重要的转变就是 P55处理器的推出,这是第一款采用增加了 57条 MMX指令集(主要用于多媒体和网络通信)的 CPU。随着 CPU的制造工艺继续发展,处理器已转向到 0.35 μm制造工艺上,运行电压变成 2.8 V,这就要求主板进行相应的结构上的改变以支持新的 CPU电压,也就是说要对主板增加一个电压调整器。新的 CPU的一级缓存也增加到了以前的两倍,达到 32 KB。

Pentium MMX处理器在 Socket7的架构下工作于 166~233 MHz的时钟频率,它的总线频率为 66 MHz。

12.4.1 Pentium处理器的内部结构

Pentium处理器的内部结构如图 12.8所示。Pentium处理器主要由执行单元、指令 Cache、数据 Cache、指令预取单元、指令译码单元、地址转换与管理单元、总线单元以及控制器等组成,其中核心是执行单元(又叫运算器),它的任务是高速完成各种算术和逻辑运算,其内部包括两个整数算术逻辑运算单元(ALU)和一个浮点运算器,分别用来执行整数和实数的各种运算。为了提高效率,它们都集成了几十个数据寄存器用来临时存放一些中间结果。

除执行单元外,其他的单元都可认为是一些控制单元。其中主要包括指令预取单元(为加速处理单元,Pentium提前向主存取出一些准备要执行的指令),指令译码单元,地址转换与管理单元(及时形成各种指令需要使用的数据所存放的地址,有效地使用存储器系统)以及总线单元。

总线单元为 Pentium提供了与周围其他硬件环境的接口,比如,从指令预取单元接收预取指令的请求,从执行单元接收传送数据的请求等。当有多个单元

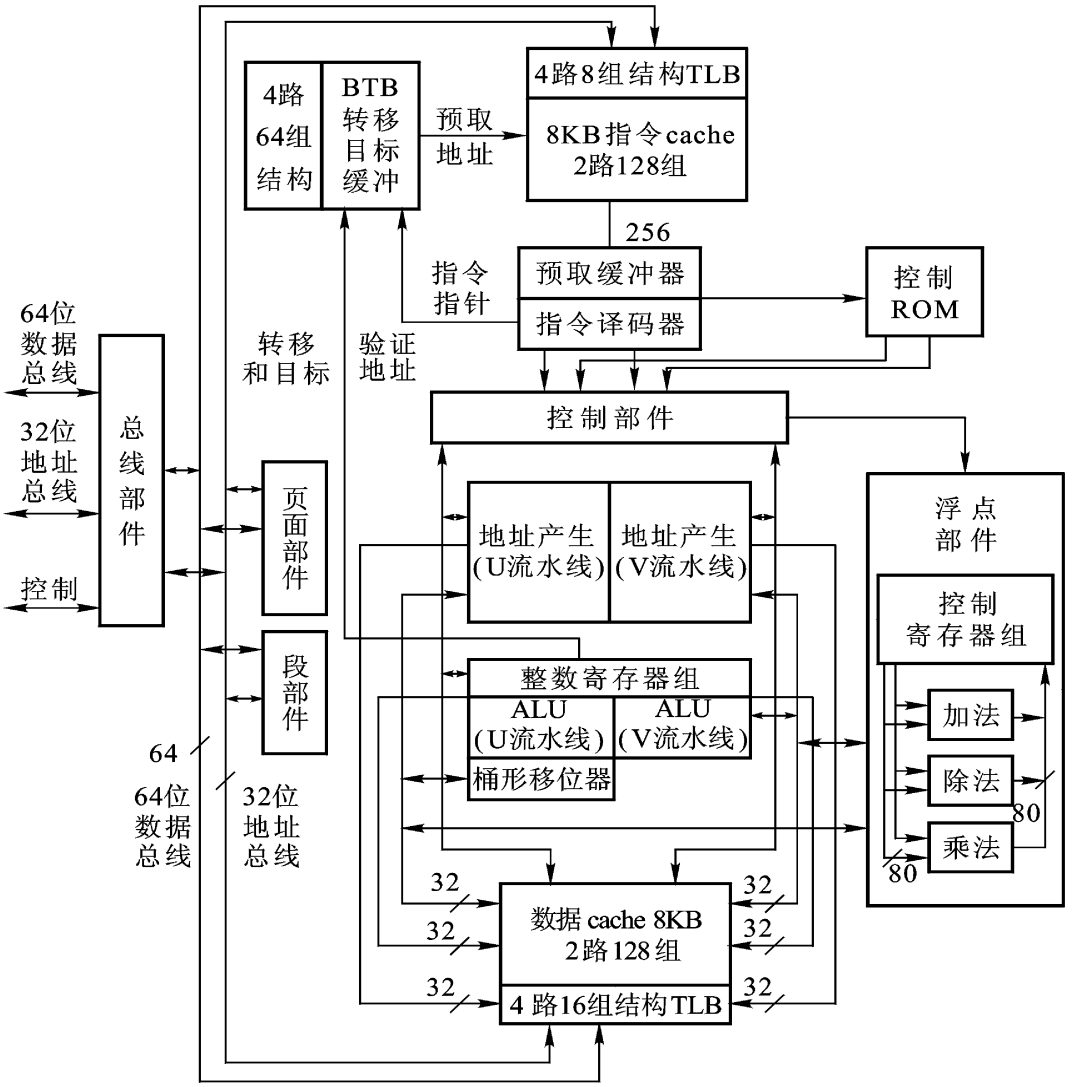


图 12.8 奔腾处理器的内部结构

同时发出请求时,总线单元将进行协调,努力为所有请求及时服务。总线单元的基本任务是使 Pentium能有效使用计算机中的总线。

12.4.2 Pentium处理器的技术特点

除了具有与 80X86系列微处理器完全兼容的特点以外,在 CPU的结构体系上,还有如下一些新的特点。

⑴ Pentium的片内高速缓存采用了分离式结构

即 Cache分为两个成组相连 8KB指令 Cache和 8KB数据 Cache,这种将片内高速缓存分开的做法使各自 Cache能够加快速度,减少等待时间及搬移数据

的次数和时间,从而提高了整体性能。

⑵ Pentium采用 RISC技术

Pentium虽然属于 CISC处理器,但在执行单元的设计中采用了较多的 RISC技术,例如 Pentium具有超标量指令流水线功能,超标量指令流水线的基本思想是 RISC技术的重要内容之一。一个处理器中有多个指令执行单元时,Intel称之为超标量结构。奔腾处理器有两个执行单元,这些执行单元也称为流水线,用于执行微机程序指令。每个执行单元都有其自己的 ALU,地址生成电路以及数据高速缓存接口。

在 Pentium处理器内设计的两条指令流水线,分别叫 U指令流水线 (U pipeline)与 V指令流水线 (V pipeline),它们是并行但独立的功能单元,在某些条件下可以并行执行两条指令。U流水线负责所有整数和浮点数指令的执行,V流水线负责简单的整数指令与 FXCH (交换寄存器内容)浮点指令的执行,每个流水线部件依次由 5个执行阶段组成:指令预取、指令译码、地址计算、执行运算、回送结果。在最佳状况下,一个处理器时钟能执行完两条指令,其效率比 80486的单一指令流水线提高一倍。

⑶ Pentium具有高性能的浮点运算部件

Pentium中的浮点运算部件完全重新设计,根据测试,Pentium中 FPU的速度是 486中 FPU的两倍,Pentium中的浮点操作已高度流水线化,并与整数流水线相结合。

⑷ 具有分支指令预测功能

在指令流水线的处理过程中,分支指令 (转移指令)有相当大的破坏作用,例如在某些指令流水线中,第 1条指令已执行到译码阶段,而此时第 2条指令已进入预取阶段,如果发现第 1条指令是分支指令 (即需要跳转到程序的某一行),则第 2条要执行的指令就被迫取消执行,并把分支目的地址处的指令装入流水线,如此将使整个指令流水线混乱或停滞,而大多数程序是每 6到 10条指令就有一条分支指令,所以如何处理分支指令就成为高速流水线执行单元的重要问题。

在 Pentium处理器中设立了能预测分支指令的分支目标缓冲器 (Branch Target Buffer,BTB),如预测无分支指令,则预取将继续按顺序执行 如预测到分支指令,则可在分支指令进入指令流水线之前,根据预测预取指令 (BTB)能记住分支目的地址,并允许分支发生前预取新指令,从而不致使指令流水线陷于混乱或停滞。

⑸ 数据总线位宽增加

Pentium处理器内部数据总线与 80486一样都是 32位 (内部供程序控制用

的寄存器组有16个,其中32位通用寄存器10个,32位状态和控制标志寄存器2个,16位段寄存器6个,另外还有更多的寄存器不再用作控制程序,而提供CPU在操作过程中使用,这类寄存器有几十个),但处理器与内存进行数据交换的外部数据总线为64位,在一个总线周期内将数据传输量增加了一倍。另外Pentium还支持多种类型的总线周期,在突发方式下,可以在一个总线周期内读入256 B的数据。Pentium的64位数据总线与内存数据交换的速度高达528 MB/s,为80486DX-50的3倍还多。

⑹ 常用指令固化

在Pentium处理器中,把一些常用指令(比如MOV、INC、DEC、PUSH等)改用硬件实现,不再使用微码进行操作,使指令的运行速度进一步加快。

⑺ 系统管理模式(SMM)

SMM最显著的应用就是电源管理,它可以使处理器和系统外围部件在暂不工作时,休眠一定时间,然后再按下一键唤醒它们,使之继续工作。通过使用管脚的信号,SMM甚至能完全控制整个系统,包括输入/输出和RAM。

12.4.3 Pentium处理器的发展

1. Pentium Pro

在Pentium处理器的基础上,1995年Intel公司宣布P6产品问世。P6即"Pentium Pro",中文名字为"高能奔腾处理器"。Pentium Pro是第一个属于第六代CPU的产品。从第六代微处理器开始,具有36条地址线,可寻址64 G的地址空间。英特尔首次将二级缓存也整合在此款CPU上,并且此二级缓存与处理器的内核捆绑在一起,使它的工作频率与CPU同步。此款处理器采用了两种制造工艺,分别是0.25 μm和0.35 μm。先进的技术可以使CPU的缓存越做越大,可从256 KB,512 KB,1 MB一直做到2 MB。

Pentium Pro具有16 KB的一级缓存,时钟频率为150~200 MHz,其系统总线为60 MHz或66 MHz,采用的是socket8结构。Pentium Pro支持所有以前的Pentium指令(不包括MMX),此款CPU还是第一款使用独立双总线结构的CPU。

Pentium Pro处理器保持了与以前的80X86处理器的二进制兼容。其性能的提高,主要是由于采用了下述一些新的设计:

⑴ 具有化复杂指令为精简指令功能,Pentium Pro能把X86(CISC结构)指令分解为较简单的指令,这些被称之为微操作,执行时间很短,并且比较长的X86指令更易用并行的方法执行,这既照顾到芯片的兼容性,又提高了运行

速度。

⑵ Pentium Pro片内除了有 16 KB (数据与指令各 8 KB)高速缓存外,还有256 KB二级高速缓存 (二级缓存与 CPU共置于 387针双腔 PGA陶瓷封装内)并有高速总线与 CPU紧密相连。高能奔腾的二级缓存与 CPU的速度一样,超过奔腾 II 350 MHz及以下的奔腾 II (奔腾 II 的二级缓存速度只有 CPU速度的一半)。高能奔腾的二级缓存最大可达 1 MB,而奔腾 II 仅有 512 KB。此外高能奔腾可以 4个 CPU同时工作,这在服务器中是必需的,但奔腾 II 只能两个 CPU同时工作。因而能够极大地提高程序运行速度,而 Pentium的二级高速缓存不在芯片上,而是在片外电路中。

⑶ 具有最新的指令动态执行技术,包括 增强型的分支预测,能向 CPU核心提供多条指令,这些指令放在指令池中,它采用调度/执行单元和指令释放单元代替传统的指令执行单元。这样指令可以不完全按照顺序执行,但执行结果仍按原有顺序产生。例如,一条正在执行的指令因等待操作数未能执行完,它能从指令池中找出其他的指令并执行。这称为随机执行,只要执行条件具备就可以执行。对多分支程序,它可以通过多分支预测技术对程序不同的分支进行预测,按预测结果调整指令执行顺序,这样使指令执行效率更高。

⑷ 具有三路超标量结构,因而并行执行指令的能力超过 Pentium,而 Pentium只有两路超标量结构。

2. Pentium II

Pentium II 是 Pentium Pro的改进产品,把多媒体增强技术 (MMX技术)融入高能奔腾处理器中,新增 57条功能强大的指令,使它能更有效地处理声音、图像、视频信号。

Pentium II 采用了动态执行技术,包括多分支跳转预测、数据流分析和推测执行技术,提高了软件的执行速度。

PentiumII还采用了双重独立总线结构,一条是二级 Cache总线,另一条是处理器至主存储器的系统总线,这使 PentiumII处理器的数据吞吐能力大大提高。

3. PentiumIII

PentiumIII仍是 32位 Intel结构 (IA-32) CPU,它最重要的技术特点在于采用了 KNI(MMX2)构架并添加了 70条附加浮点多媒体指令,以增强三维和浮点应用。主要针对中、高端市场推出,总体性能比较高。但是在现在的高主频时代和高性能的双重压力下,也开始显得平庸了。市场有 733EB、800EB、866EB、933EB、1GEB等几种频率 PIII,采用的都是 0.18 μm的制造工艺,外频为 100 MHz。

PentiumIII处理器设计时便考虑了互联网的应用。它的另一个特色便是处

理器包含了序列号,每个 PentiumIII处理器都有一个不同的号码,Intel认为这给用户带来好处是可以提高互联网上的安全性。这个全新的 64位的处理器序列号,就相当于电脑的“身份证”,用户既可以用它对电脑进行认证,也可以在商务往来或是上互联网时用它进行加密,以提高电脑应用的保密性。

Tualatin是最后一代 PentiumIII,其接口也是用 Socket370,但针脚的定义,工作电压,内核封装等又有变化,需新的主板才能支持。Tualatin采用了 0.13 μm的技术。工作电压为 1.2~1.425 V,工作频率可达到 1.33 GHz,其性能与 Pentium 4相差不大。

4. Pentium 4

Pentium 4处理器是 Intel公司全新推出的 IA-32结构处理器,目前主频已达 2.7 GHz。Pentium 4处理器没有使用 P6架构,而采用了和以往不同的全新 NetBurst架构。Pentium 4处理器采用了多种新技术:

⑴ 超级流水技术

Pentium 4处理器将流水线的深度增加了一倍,达到 20级,这显著提高了处理器的性能和频率能力。

⑵ 改进的浮点运算能力

Pentium 4处理器改进的浮点运算能力,提供了逼真的视频和三维图形处理能力,带来了更精彩的游戏和多媒体体验。

⑶ 快速执行引擎

算术逻辑单元(ALU)以双倍的时钟速度运行,从而提高了总体速度。一种全新的高速缓存系统,执行跟踪高速缓存,与高速执行运行保持一致。

⑷ 400 MHz系统总线

400 MHz的系统总线在奔腾 4处理器和内存控制器之间提供了 3.2 GB/s的传输速度,是目前最高的带宽台式机系统,提供了响应更迅捷的系统性能。

⑸ 高度动态执行

在 Pentium 4处理器推出之前,数据是以特定顺序来执行的。现在,数据能够以速度最快的顺序来执行一段程序,从而提高了总体性能。

⑹ 数据流单指令多数据扩展指令(SSE2)

拥有 144条新指令、一个 128位单指令多数据整数运算和 128位单指令多数据双精度浮点指令,可极大增强您的多媒体体验。不过这些新的指令需要应用软件的支持,而 SSE指令集到目前还没有办法全部得到支持。

⑺ 高速缓存

英特尔的 Pentium 4处理器采用了一个全新的先进一级(L1)指令高速缓存技术,以及可提供更高性能指令高速缓存的执行跟踪高速缓存。执行跟踪高速

缓存可更有效地利用高速缓存内存。此外,奔腾 4处理器的 256 KB二级高级传输高速缓存,集合了内建内存芯片,提高了总体性能。

⑻ 内存

奔腾 4处理器具有双通道 RDRAM,可实现目前最好的数据传输性能。双通道 RDRAM的数据传输速率高达 3.2 GB/s,从而可以充分享受处理器主频提高带来的所有优势,实现了全面的优化性能。

5. Celeron (赛扬)

赛扬的定位是基于影响越来越大的“基本 PC”,最初的两款产品没有二级缓存,连封装合也省掉了,走低价格低性能的路线。早期产品由于没有 L2,性能不佳,未获成功。

而 Intel稍后推出的赛扬 A,具有和目前奔腾二代处理器同等的内核,内置了 128 KB全速 L2 Cache (与 CPU同频工作),更快的二级缓存对系统沉重的数据负荷大有好处。而且同样拥有源于 Intel Pentium Pro的 D.I.B技术。

Intel的赛扬系列是 Intel面向低端市场的产品,其实就是 Pentium II 的简化版,唯一的差别在于减少了集成的 L2 Cache (仅为 Pentium II 的一半)。为了进一步降低成本,Intel又将原来 SLOT1接口的赛扬 A做成了 Socket370接口的 PPGA封装。如果主板是 SLOT1接口的,就可通过 Socket370转接板来转换。2000年后,Intel推出了赛扬二代,采用了 0.18 μm的工艺和全速的 L2缓存,工作频率也较高。

6. Xeon (至强)

1998年 7月 Intel公司推出了新的奔腾至强处理器 (Pentium Xeon processor),该处理器专为满足中高档服务器和工作站而设计的。该产品采用 0.25 μm P6微处理器结构,运行速度为 400 MHz,内装 512 KB或 1 MB的二级高速缓存,缓存可寻空间达 64 GB。

自第一个处理器问世以来,Intel公司已经连续发明制造了 10代规模化生产的微处理器芯片,这一系列的产品由 8080芯片开始,经过 8086, 80286, 80386, 80486, Pentium (奔腾), Pentium MMX (多能奔腾)、Pentium Pro (高能奔腾), Pentium II (奔腾 II), Pentium III (奔腾III)和最新的 Pentium 4 (奔腾 4)处理器。每一代产品都标志着个人电脑计算能力的又一次飞跃及技术上的重大突破。

按照摩尔定律的预测,到 2011年,一个微处理器将含有 10亿个晶体管,而系统功能将是奔腾 II 处理器的 150倍。微处理器发展到此,并不是至极,而只不过是新的开端而已,更新的处理器将会以前所未有的功能展现在面前。

思考题与习题

12.1 试述 Intel系列微处理器的发展历程。

12.2 试述 80286微处理器的技术特点。

12.3 试述 80386微处理器的技术特点。

12.4 试述 80486微处理器的技术特点。

12.5 何谓超标量双流水线结构？

12.6 何谓实地址管理方式？

12.7 试述保护虚地址管理方式及其工作过程。

12.8 何谓虚拟 8086方式？

12.9 试述 80386的分页管理工作原理。

12.10 试述 Pentium微处理器的技术特点。

第13章

总线标准与微型计算机

13.1 微型计算机系统中的总线

13.1.1 总线和总线规范

1. 总线

微型计算机的硬件系统包括微处理器、内存、I/O接口和总线。一台微型计算机可以含有几个或几十个模块或设备，总线就是在多个模块之间或多个设备之间传送信息的公共通道。信息包括指令、数据和地址。

按照总线的规模、用途和应用场合的不同，微型计算机系统中的总线可分为片总线、内部总线和外部总线三类。

⑴ 片总线

片总线是芯片与芯片之间的总线，通常包括地址总线、数据总线和控制总线。一般它是微处理器构成一个小系统或部件时的芯片级总线。

⑵ 内部总线

内部总线又称系统总线或微机总线，它是用于微机系统中各插件之间信息传送的通道。

⑶ 外部总线

又称为通信总线，是微机与微机或微机与其他设备信息传输的通道。

2. 总线规范

为了使计算机的各种模块或设备能够互连和扩展,不同厂商生产的部件能够相互替换,需要制定一定的规范,因此需要标准化的总线标准。通常,总线标准由国际组织、厂商联盟制定或推荐。

每个总线标准都有详细的规定,一般包括以下 4个特征:

⑴ 物理特性

物理特性指的是总线物理连接的方式,包括总线的根数、总线的插头、插座是什么形状的、引脚是如何排列的等。

⑵ 功能特性

功能特性描写的是这一组总线中每一根线的功能是什么。从功能上看,总线分成 3组:地址总线、数据总线和控制总线。地址总线的宽度指明了总线能够直接访问存储器的地址范围。数据总线的宽度指明了访问一次存储器或外部设备最多能够交换数据的位数。控制总线一般包括 CPU与外界联系的各种控制命令,如输入输出读写信号、存储器读写信号、外部设备与主机同步匹配信号、中断信号和 DMA控制信号等。

⑶ 电气特性

电气特性定义每一根线上信号的传递方向及有效电平范围。一般规定送入 CPU的信号叫 IN(输入信号),从 CPU送出的信号叫 OUT(输出信号)。

⑷ 时间特性

时间特性定义了每根线在什么时间有效。也就是说用户什么时间可以用总线上的信号或者用户什么时候把信号提供给总线,CPU才能正确无误地使用。

13.1.2 系统总线 ISA和 EISA

1. ISA总线

ISA(Industry Standard Architecture)总线是在最早的 IBM PC上的 PC总线基础上发展起来的。PC总线是一个 8位的开放结构总线,有 62个引脚,提供地址线、数据线、控制线及电源等。1984年,在 PC总线的基础上增加了一个 36引脚的扩展插座,成为 ISA总线。ISA总线插槽如图 13.1所示。

ISA总线有 16位数据线、24位地址线、中断线、支持 DMA通道的信号线、等待状态发生信号线、+5 V、-5 V、12 V电源线等。ISA总线工作频率为 8 MHz,最大传输率为 8 MB/s。ISA总线信号见表 13.1。

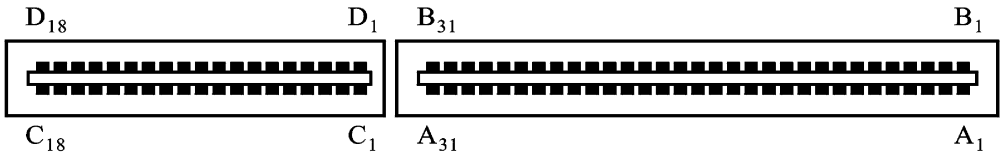


图 13.1 ISA总线插槽

表 13.1 ISA总线信号

引脚编号	信号名称	I/O类型	引脚编号	信号名称	I/O类型
A_1	$\overline{I/OCHK}$	I	B_{18}	DRQ_1	I
A_2 ～A_9	SD_7 ～SD_0	I/O	B_{19}	$\overline{REFRESH}$	I/O
A_{10}	I/OCHRDY	I	B_{20}	SYSCLK	O
A_{11}	AEN	O	B_{21} －B_{25}	IRQ_7 ～IRQ_3	I
A_{12} ～A_{31}	SA_{19} ～SA_0	O	B_{26}	$\overline{DACK_2}$	O
B_1	GND	地	B_{27}	T/C	O
B_2	RESETDRY	O	B_{28}	BALE	O
B_3	+5 V	电源	B_{29}	+5 V	电源
B_4	IRQ_2	I	B_{30}	OSC	O
B_5	－5 V	电源	B_{31}	GND	地
B_6	DRQ_2	I	C_1	SBHE	I/O
B_7	－12 V	电源	C_2 －C_8	LA_{23} ～LA_{17}	I/O
B_8	$\overline{OWS}$	I	C_9	$\overline{MEMR}$	I/O
B_9	+12 V	电源	C_{10}	$\overline{MEMW}$	I/O
B_{10}	GND	地	C_{11} －C_{18}	SD_8 ～SD_{15}	I/O
B_{11}	$\overline{SMEMW}$	O	D_1	$\overline{MEMCS_{16}}$	I
B_{12}	$\overline{SMEMR}$	O	D_2	$\overline{I/OCS_{16}}$	I
B_{13}	$\overline{IOW}$	I/O	D_3	IRQ_{10}	I
B_{14}	$\overline{IOR}$	I/O	D_4	IRQ_{11}	I
B_{15}	$\overline{DACK_3}$	O	D_5	IRQ_{12}	I
B_{16}	DRQ_3	I	D_6	IRQ_{13}	I
B_{17}	$\overline{DACK_1}$	O	D_7	IRQ_{14}	I

续表

引脚编号	信号名称	I/O类型	引脚编号	信号名称	I/O类型
D_8	$\overline{DACK_0}$	O	D_{14}	$\overline{DACK_7}$	O
D_9	DRQ_0	I	D_{15}	DRQ_7	I
D_{10}	$\overline{DACK_5}$	O	D_{16}	+5 V	电源
D_{11}	DRQ_5	I	D_{17}	$\overline{MASTER}$	I
D_{12}	$\overline{DACK_6}$	O	D_{18}	GND	地
D_{13}	DRQ_6	I			

2. EISA总线

EISA (Extended Industry Standard Architecture)总线是由 Compaq为代表的美国 9大计算机厂家联合制定的一种 32位总线结构。该总线是 ISA总线的 32位扩展,与 ISA总线兼容,它具有 32位数据线、33MB/s的数据传输率,提供多处理器控制功能,其多主控总线使一般微机的单处理器环境升级至多处理器环境,扩展卡安装方便、自动配置,无需跳线,保持与 ISA总线百分之百兼容。

EISA插座的触点分上下两层,上层是原 ISA总线信号,下层是 EISA新增的信号。当 ISA板卡插入时只能接触到上层触点,当 EISA板卡插入时能够接触到上、下两层触点。

13.1.3 PCI总线

PCI(Peripheral Component Interconnect)总线标准是 1991年由 Intel、IBM、Compaq、Apple等大公司联合制定的一种局部总线标准,1995年又推出了 PCI2.1版。

PCI总线支持 33 MHz和 66 MHz的同步总线操作,其数据宽度为 32位,可升级至 64位。其数据传输速率可高达 132 MB/s (33 MHz时钟 32位数据通路)~528 MB/s (66 MHz时钟 64位数据通路)。这就为计算机图形显示所需的大批量的数据传送和高性能的磁盘输入输出提供了硬件支持。PCI总线开放性好,具有良好的兼容性,是一种低成本、高效益、能与 ISA总线兼容的一种有前途的局部总线。

1. PCI总线的主要特点

(1) 最高操作时钟频率为 33/66 MHz,拥有 32位和 64位两种数据通道。

⑵ 支持由成组数据传送方式:若被传送的数据在内存中连续存放,则在访问第一个数据时需要两个时钟周期,第一个时钟周期内给出地址,第二个时钟周期内传送数据;从第二个数据开始不必再给出地址,可直接传送数据,即每一个时钟周期传送一个数据。这种传送方式也称为突发传送。

⑶ 支持总线主控方式,允许多处理机系统中的任何一个微处理机都可以成为总线主控设备,对总线操作进行控制。

⑷ 与 ISA、EISA、微通道等多种总线兼容。由于 PCI总线在 Pentium微处理机与其他总线间架起了一座桥梁,它也支持像 ISA、EISA以及微通道等这样的低速总线操作。

⑸ 支持所有目前的和将来的不同结构的微处理器。可以把 PCI局部总线看做是一个独立的处理器,它可以与任何一种微处理器一起使用,不局限于 80X86。这就确保了 80X86系列机在更新换代时,也不会把 PCI局部总线抛弃。因此许多大的计算机公司都宣布支持 PCI总线。

⑹ 它支持 5V和 3.3V两种扩充插件卡。可以从 5V向 3.3V进行平滑的系统转换。PCI总线上装有一个很小的断路键,使用户在插卡时不会导致在系统主板上有不同的电压电源。

⑺ 支持即插即用。PCI设备中有存放设备具体信息的寄存器,这些信息使系统 BIOS和操作系统层的软件可以实现自动配置。用户可以安装一个新的添加卡,且不用设置 DIP开关、跳线(跨接线)和选择的中断。配置软件会自动选择未被使用的地址和中断,以解决可能出现的冲突问题。

⑻ PCI总线的引线,在每两个信号之间都安排了一个地线,以减少信号间的相互干扰。

⑼ PCI总线实现了触发级的中断,这种中断可支持中断共享。

⑽ PCI总线能支持高达 10个外围设备,其中的某些外围设备必须嵌入到系统主板上。

2. 桥接器

PCI规范包括三类桥接器:主处理器与 PCI的桥,即主桥;PCI与标准总线(ISA、EISA、微通道)之间的总线桥;PCI与 PCI之间的桥。桥接器就是总线转换器,能连接两条计算机总线,实现总线之间的通信。在一个 PCI应用系统中,如果某设备取得了总线控制权,就称其为“主设备”;而被主设备选中以进行通信的设备称为“从设备”。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,使系统中的每一个总线主设备能看到同样的地址表。

3. 配置空间

PCI提供三个互相独立的物理地址空间,包括存储器地址空间、I/O地址空

间和配置地址空间。前两个是一般总线都有的通用空间;第三个是用以支持PCI硬件配置的特殊空间,是 PCI所特有的。每个 PCI总线设备必须提供配置信息存放的空间,多功能设备则应相应地提供多块配置信息存放空间。

由于 PCI总线的优良性能,获得了广泛应用。目前的 PC中一般都采用 PCI与 ISA总线并存的结构,也有的 PC已取消了 ISA总线。

13.1.4 AGP

AGP(Accelerated Graphics Port)是 Intel公司开发的新一代图形总线标准。它是建立在 PCI基础上专门针对 3D图形处理而开发的高性能图形总线。

AGP规范是 Intel公司解决电脑处理(主要是显示)3D图形能力差的问题而出台的。电脑在处理 3D图形时需要与 CPU和系统内存进行大量的数据交换,根据专家计算,在处理 1 024×768分辨率、64 K种彩色的显示方式中,显示控制器与系统之间通过 PCI总线传输的数据高达 532 MBps,而实际上 PCI总线只能保证 133 MBps的极限速率,其中还没有考虑同时安装在 PCI总线的 PCI声卡、SCSI接口等外设还需同时享用这可怜的 133 MBps的速率。另外由于需要对 3D图形中物体表面进行大量的各种纹理贴图处理或渲染,以保证物体材质表面的真实性效果,显示控制器还必须占用最多高达 16 MB的显存来保存纹理位图等数据,这对普通电脑中只有 4~8 MB显存的 3D图形显示卡来说是不可能做到的。显存的不足必将影响图像的分辨率和 3D中关键的“Z-Buffering”处理,具体表现将影响电脑 3D图形再现的速度和视觉效果。因此,Intel公司认为 PCI总线数据传输率低、显示卡显存容量不足是普通电脑提高处理和显示 3D图形速度的瓶颈。

在电气信号上,AGP标准完全兼容 PCI标准。一个 AGP设备既可通过 AGP规范,也可通过 PCI规范与内存进行数据交换。对于在 PCI标准中保留的管脚,AGP也不予以占用。但是,AGP并不是 PCI的升级版本,它的插槽与 PCI不兼容,也就是说,AGP的显示卡不能插在 PCI总线上,以前的 PCI显示卡也不能插在 AGP槽上。AGP的出现并不是为了取代 PCI,AGP是为了加快图形处理而设计的一条数据传输捷径,PCI将在除图形卡以外的部分继续存在。

与 PCI相比,AGP有以下三个重大改进:

⑴ 对内存的读写操作实行流水线处理,充分利用等待延时,大大地增加了读内存的速度,使其与写内存的速度相当。而在 PCI中,读内存的速度通常只是写内存速度的一半。

⑵ 使总线上的地址信号与数据信号分离,一方面充分利用了读写请求与

数据传输之间的空闲,使总线效率达到最高;另一方面可以有效地分配系统资源,避免了死锁的发生。

⑶ 是第一个为图形卡所设计的界面。实际上 AGP不能算是总线,因为总线可以支持多种设备,它只能算是一种端口。PCI显卡以 PCI总线速度(外频)的一半即最大 33 MHz工作,它可以达到的峰值传送率为 33×4 MHz=132 Mbps(PCI是 32位总线一次传输 4字节)。而 AGP以 66 MHz的速度和 64位的数据宽度工作,AGP1X的峰值传送率可达 66×4 MHz=264 Mbps,AGP2X的峰值传输率可以达到 532 Mbps,因为“2X”可以在一个时钟周期中传输两次数据(上升沿和下降沿各一次),而一般的工作状态只能进行一次传输,AGP4X的理论传输率为 1.066 Gbps。在主板 66 MHz总线上,芯片组和内存之间数据的最大传输率就可以达到 66×64 MHz=528 Mbps,在这种环境下 AGP4X无法发挥作用。而使用 100 MHz总线时,内存的最大数据交换率可以达到 800 Mbps,这可能会使“4X”发挥一些威力,但也是远远不够的。

⑷ AGP增加了一种使用模式——“Execute”模式(执行模式)。原来 PCI使用的 DMA模式适用于从系统内存到图形内存之间的大批量数据传输,其中系统内存中的数据并不能被图形加速器所直接调用,只有调入图形内存才能被加速芯片所寻址。而在 Execute模式中,加速芯片(以 i740为代表的一些显示芯片)将图形内存与系统内存看做一体,通过 Graphics Address Remapping机制,加速芯片可直接对系统内存进行寻址,这样可以大大减轻本地局部显存的压力。但该模式的使用究竟是否真的提高性能,需要根据情况具体分析,不能一概而论。

如果想应用 AGP技术去处理 3D图形而获得较好的效果,那么必须具备以下硬件和软件环境的条件支持。

硬件方面:支持 AGP规范的电脑主板、安装 64 MB的 SDRAM内存,使用至少符合 AGP规范 1.0/2.0标准的 3D显示卡。使用支持 AGP显卡的主板这一点不必解释,安装 64 MB内存的原因是 AGP技术只有在检测系统拥有 64 MB或更大容量时 DME技术才能得到应用,而使用 SDRAM型内存自然是追求高速的存取时间以提高显示速度,而真正的 AGP规范 3D显示卡是指所用的显卡不但支持 ×2模式的高速数据传输,而且确实支持 DME即支持“执行模式(Execute Mode)”。

软件方面:操作系统使用 Windows 95 OSR2.1或 Windows 98版本;所运行的应用软件中支持 AGP规范显示卡。对操作系统要求使用 Windows 95 OSR2.1和 Windows 98是因为这些版本的操作系统支持 AGP技术。

13.1.5 通用串行总线 USB

USB (Universal Serial Bus——通用串行总线)用于微机与外设之间的数据交换,是一种外部总线。USB允许外设在本机和其他外设工作时进行连接、配置、使用和移走。目前 USB接口的外设已十分丰富,包括键盘、鼠标、显示器、调制解调器、打印机、扫描仪、数码相机等,USB接口还可以串接,使一个 USB口串接多个 USB设备。USB的应用减少了微机与外设连接的 I/O端口,甚至仅用一个串行接口来代替,使 PC与外设之间的连接更容易。

1. 总线拓扑结构

USB的物理连接是一种分层的星形结构,集线器 (Hub)是每个星形结构的中心,PC是根集线器,外设或附加的 Hub与之相连。USB最多可支持 5个 Hub层、127个外设。

2. USB的物理层

USB采用 4线电缆传输数据和电源,如图 13.2示。其中 D+和 D-是差模信号线,VBUS为 +5 V电源,GND为电源地。USB提供 12 Mbps高速模式和 1.5式 Mbps低速模式传输数据,两种模式可并存于一个系统中。

图 13.2 USB使用的 4线电缆

3. USB数据传输方式

USB数据传输是通过管道进行的,USB提供了控制传输、同步传输、中断传输和数据块传输等 4种数据传输方式。它们在数据格式、传输方向、数据包容量限制、总线访问限制等方面有不同的特征。

(1) 控制传输

通常用于配置、命令、状态等情况,支持双向传输,允许数据包容量为 8、16、32、64字节,不能指定总线访问的频率和总线占用时间,传输可靠性高。

(2) 同步传输

是一种周期性、连续的传输方式,通常用于与时间有密切关系的信息传输,单向传输 若需要双向传输,必须使用另外一个端点),只能用于高速设备,数据容量为 0~1 023字节,具有带宽保证 (保持恒定的传输速率),没有数据重发机

制,要求具有一定的容错性。

⑶ 中断传输

用于非周期的、自然发生的、数据量小的信息传输,只有输入一种传输方式(外设到主机),对于高速设备数据包大小为 64字节(低速设备小于或等于 8字节),具有最大的服务周期保证(在规定时间内至少有一次数据传输),具有数据保证(错误可重发)。

⑷ 数据块传输

用于大量的没有时间要求的数据传输,单向传输(若需要双向传输,必须使用另外一个端点),对于高速设备数据包大小为 8、16、32、64字节,没有带宽保证(总线空闲即可传输),只有数据保证(必要时可重试、重发)。

4. USB总线协议

在 USB中,任何操作都是从主机开始的,主机以预先安排的时序,发出一个描述操作类型、方向、外设地址以及端点号的包(令牌包),然后在令牌中指定数据发送者发出一个数据包或指出它没有数据传输。而 USB外设要以一个确认包做出响应,表明传输成功。

⑴ 域类型

一个包包括:同步域、标志域、地址域、端点域、帧号域、数据域及 CRC校验等。其中同步域用于本地时钟与输入信号的同步;标志域指明包的类型及格式;地址域指明外设端点地址(外设地址及外设端点);端点域说明设备所使用的子通道;帧号域指明目前帧的序号;数据域包含传输的数据;CRC校验包含令牌校验和数据校验。

⑵ 包类型

USB中有令牌包、数据包、应答包等,其中令牌包包含输入(IN)、输出(OUT)、设置(SETUP)和帧起始(SOF)4种类型;数据包含标志域、数据域和 CRC校验域;应答包包含确认包、无效包、出错包、特殊包等。应答包用于报告数据传输状态,仅有支持流控制的传输类型。

⑶ 总线操作

USB的总线操作包含:批操作、控制操作、中断操作、同步操作等。批操作包含令牌、数据、应答三个阶段;控制操作包含设置和状态两个操作阶段;中断操作只有输入一个方向,与批操作的输入相同;同步操作包含令牌和数据两个阶段,它不支持重发功能。

⑷ 错误检验与恢复

USB具有检查错误能力,可以根据传输类型的要求进行相应的处理。如数据传输要求较高的数据准确度,因此支持所有的错误检验与重试来对端对端数据的

完整传输。USB可进行 PID检验、CRC校验、总线时间溢出及 EOP错误检验等。

13.1.6 IEEE1394

IEEE1394是 IEEE (Institute of Electrical and Electrical Engineers)制定的一个高速串行总线标准。它通过提供一个高带宽、易使用和低价格的接口,将个人电脑和外部设备、家用电器连接起来。IEEE1394提供的带宽完全可以综合现有的外部接口,将以前的多种总线标准统一起来。它提供数字设备之间高速、廉价、规格化、多用途的传输方式,是数字信息传输的一大革命,被认为是未来总线最佳选择之一。

IEEE1394是以 TI, Sony, Microsoft, Philips, Apple等公司提出,共同制定的一种高性能的串行总线,其前身是于 1986年 Apple公司用于连接打印机、调制解调器、硬盘、扫描仪等外部设备到电脑的总线设计“FireWare”。1995年正式由 TI, Sony等定义,成为支持个人电脑与多媒体连接的新一代标准,商业上仍沿用 Apple公司的“FireWare”名称。2000年 3月,新的标准 IEEE1394—2000正式通过批准。IEEE1394a支持三种传输速率:100 Mbps, 200 Mbps, 400 Mbps,并且具有更好的数据流控制和节能特性。它支持对等设备之间,如数字摄像机、SGS,数字相机、高速高分辨率打印机和扫描仪等,为高带宽视频、音频数据传输。为了消费者使用方便,IEEE1394支持即插即用和“热拔插”,同时支持更高传输速率。

IEEE1394的主要性能特点:

⑴ 采用“级联”方式连接各个外部设备。IEEE1394在一个端口上最多可以连接 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,采用树形结构时层次可达 16层,两个端点之间的最大距离为 72 m。

⑵ 采用基于内存的地址编码,具有高速传输能力。总线采用 64位的地址宽度 (16位网络 ID,6位节点 ID,48位内存地址),将资源看做寄存器和内存单元,可以按照 CPU内存的传输速率进行读写操作,因此具有高速的传输能力。IEEE1394总线的数据传输率最高可达 1 024 Mbps,适用于各种高速设备。

⑶ 采用点对点结构 (peer to peer)。任何两个支持 IEEE1394的设备可以直接连接,不需要通过电脑控制。设备之间互传数据时,不分主从设备,都是主导者和服务者。

⑷ 安装方便且适用。允许“热拔插”,支持即插即用。IEEE1394可以自动探测设备的插入与拔出动作并对系统做重新构建,即在系统工作时,IEEE1394设备也可以插入和拔出。

⑸ 兼容性好。IEEE1394总线可以适应台式电脑用户的全部 I/O要求,并可以与 SCSI并口、RS-232标准串口、IEEE1284标准并口等接口兼容。不同传

输速率的设备可以随意互连,以低速率设备的最高支持速率进行数据传输。

13.2 Pentium微型计算机

Pentium PC由主机箱、键盘、显示器、打印机等组成。主机箱包括主板、I/O接口卡、软盘、硬盘、光盘驱动器和电源等部件。

13.2.1 主板

在 PC中,主板是装有计算机主要部件的印刷电路板,除各种芯片外,还有多种接插口或接插槽。主要包括:

1. CPU芯片和 CPU插座或插槽

CPU芯片有较多的引脚,使用专用的插座或插槽将 CPU连接到主板上。Pentium CPU使用 296脚的 PGA (Pin Grid Arrange)封装,采用 ZIF (Zero Insertion 零插拔力)插座 Socket7。Socket7的侧面有一个手柄,当抬起手柄时,插座松开引脚,可插拔芯片;当按下手柄时,插座卡紧引脚。

Pentium II CPU的核心芯片和二级 Cache (静态 RAM)组合成一个长方形的插接盒,插接盒插入 Slot1插槽。Slot1是具有 242个触点的插槽,插接盒须用力插入并需要使用支架固定。

2. 内存和内存条插槽

最早的 PC直接使用存储器芯片构成内存,即芯片插入主板上的芯片插座。后来 PC改为使用内存条。每个内存条上装有多片存储器芯片,组成较大容量的存储器。

早期的内存条是 SIMM (Single In-line Memory Modular)内存条,有 30线和 72线两种。后来使用 168线的 DIMM (Dual In-line Memory Modular)内存条。各种内存条使用相应的内存条插槽。

3. 芯片组 (Chip Set)

主板上需要的各种支持芯片和接口芯片。随着 VLSI技术的发展,目前已能将 PC的总线控制 (CPU、PCI、AGP总线等)和图形显示、声卡、I/O接口功能集成在 2~3片芯片中,这些芯片组在系统中起到神经中枢的作用。有了芯片组,主板结构变得非常简洁,就一般应用而言,不再需要插入其他 I/O接口卡就能使计算机工作。

4. 扩展槽

主板上设有若干扩展插槽。由于系统的总线连接到扩展插槽,因此也称为总线插槽。插入扩展插槽的各种板卡 (接口电路)能够与系统总线相连,实现系

统的扩展。扩展插槽应遵循一定的标准,以使板卡具有通用性。现在的 PC一般包括 4～6个 PCI插槽和 1～2个 ISA插槽。

5. BIOS芯片

PC上的 BIOS (Basic InputOutputSystem) 固化于 ROM芯片中,上电后计算机先执行其中的程序。BIOS完成冷启动、热启动、上电自检、基本 I/O驱动、系统硬件配置分析、引导操作系统等。

6. 硬盘机和光盘驱动器的接口

硬盘、光驱使用 IDE接口,芯片组包含 IDE控制器,一般可连接 4个 IDE设备,如 Intel440芯片组中的 82371可挂 4个 IDE接口的硬盘或光驱。

7. 电源

PC主板由主机电源提供 4组电源,分别是 +5 V、−5 V、+12 V、−12 V。另外,主板上还有电源变换电路,以适应现在的 CPU使用低电源电压的要求。

13.2.2 440BX芯片组

芯片组不仅对 CPU、主板起着极其重要的协调、支持和控制作用,而且在很大程度上也决定了 PC的结构和性能,不同 Pentium处理器必须有相应的芯片组配合才能正常工作,因此,芯片组的发展也是十分迅速的。本节主要介绍 Intel公司的 440BX芯片组的特点和主要功能。

440系列芯片组是专为 PentiumII设计的,共包括 FX、LX、BX、GX、EX、ZX 6个版本,以适应不同类型的 PentiumII。PentiumIII发布后,440芯片组经过优化和升级,同样适用。尽管 Intel已推出 440的升级换代产品 815E,但目前使用 BX的 PentiumIII PC仍不在少数,因此具有一定的代表性,成为当代 Pentium PC主板芯片组的典型产品。

440BX芯片组由两片组成,一片是 492引脚的北桥芯片 82443BX,另一片是 324引脚的南桥芯片 82731EB (PIIXE),而 PentiumIII使用南桥芯片 82731AB。

北桥芯片 82443BX的主要功能有:

⑴ 支持单、双 PentiumII处理器,总线频率可高达 100 MHz。

⑵ 集成了内存控制器,支持 100/66 MHz的 SDRAM,最大内存 512 MB。

⑶ PCI总线接口遵循 PCI2.1版规范、3.3/5 V、33 MHz,支持除南桥芯片以外的 6个 PCI总线主设备。

⑷ 集成了 AGP (Accelerated Graphics POrt) 接口,支持 +3.3 V,66/133 MHz设备,以 133 MHz工作时带宽为 533 MBps。

南桥芯片 82731EB (PIIXE) 的主要功能有:

⑴ PCI/ISA桥接器,支持 ISA总线。

⑵ 集成了 IDE控制器,可连接 4个 IDE设备,支持 16 MBps的 PIO模式 4传输,支持 Ultra DMA33模式传输,数据传输率可达 33 MBps。

⑶ 集成了 USB控制器,支持两个 USB端口。

⑷ 集成了 7个通道的 DMA控制器、两个 82C59A中断控制器、82C54定时/计数器、实时钟等。

⑸ 遵循 ACPI(Advanced Configuration and Power Interface高级配置和电源接口)电源管理,支持系统挂起/再启动 (挂起到 RAM及磁盘)等功能。

⑹ 支持 I/O APIC (Advanced Programmable Interrupt Controller)模块。

13.2.3　采用 440BX芯片组的 PC结构

采用 440BX芯片组的典型主板结构框图如图 13.3所示,系统采用 AGP显

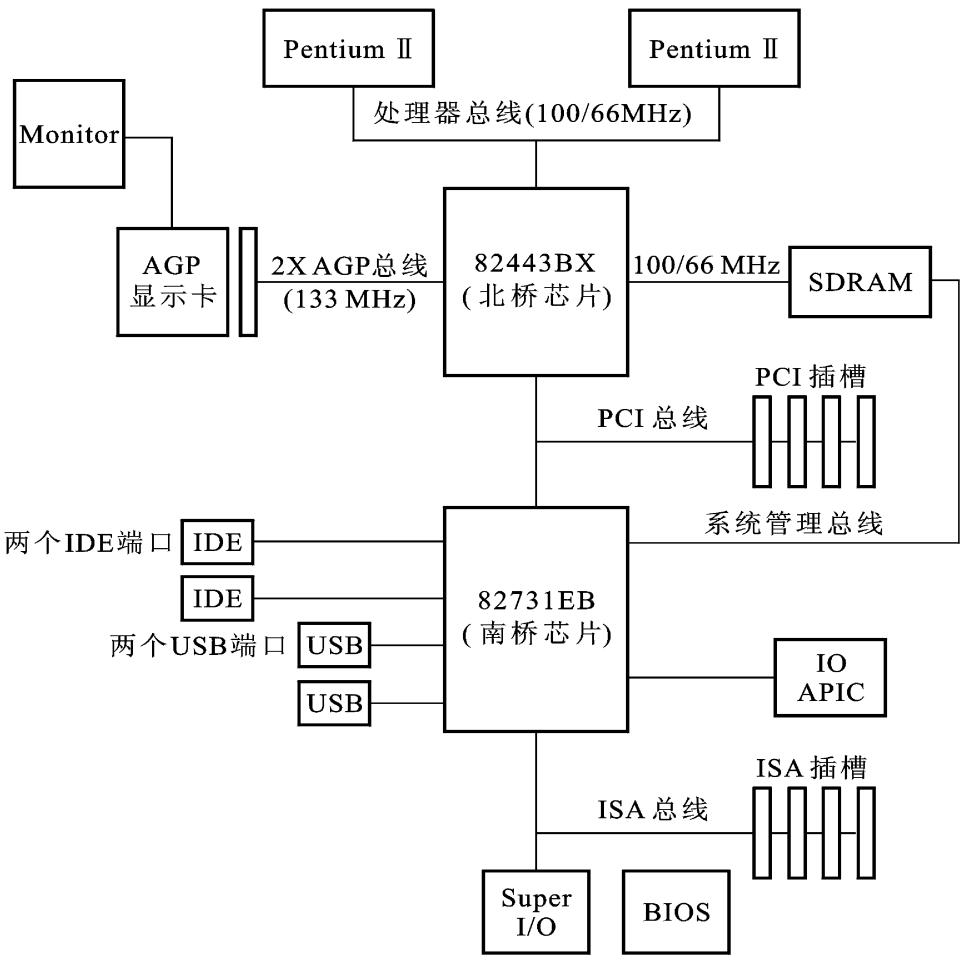


图 13.3　440BX Pentium PC主板结构

卡，系统扩展采用 PCI与 ISA总线并存的模式。

北桥芯片为 82443BX。支持单、双 Pentium II 处理器，总线频率可高达 100 MHz；内存控制器支持 100/66 MHz的 SDRAM；AGP接口以 133 MHz工作时带宽为 533 MBps，可插入 AGP接口的显示卡；南桥芯片接 PCI接口，系统中安装了 4个 PCI槽用于扩接 PCI总线设备（最多可安装 6个 PCI槽）。

南桥芯片为 82731EB（PIIXE）。它支持 ISA总线，系统中安装了 4个 ISA槽；两个 IDE控制器可连接 4个 IDE设备；支持两个 USB端口等。

思考题与习题

13.1 什么是微型计算机的总线？总线可分哪几类？

13.2 为什么需要总线标准？总线标准一般在哪几方面进行了详细的规定？

13.3 简要说明 ISA、EISA总线。

13.4 简述 PCI接口的用途和特点。

13.5 简述 AGP接口的用途和特点。

13.6 简述 USB接口的用途和特点。

13.7 简述 IEEE1394接口的用途和特点。

13.8 PC主板主要包括哪些部件？

13.9 什么是芯片组？440BX芯片组的北桥芯片有何功能？南桥芯片有何功能？

附录 1　ASCII码表

$D_6D_5D_4$ / $D_3D_2D_1D_0$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	、	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	,	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^(1)	n	~
1111	SI	US	/	?	O	_(2)	o	DEL

说明：SP 空格
NUL 空
SOH 标题开始
STX 正文结束
ETX 本文结束
EOT 传输结果
ENQ 询问
ACK 承认
DEL 作废
DLE 数据链换码
DC1 设备控制 1
DC2 设备控制 2
DC3 设备控制 3
DC4 设备控制 4
NAK 否定
SYN 空转同步

BEL 报警符　(可听见的信号)	ETB 信息组传送结束
BS 退一格	CAN 作废
HT 横向列表　(穿孔卡片指令)	EM 纸尽
LF 换行	SUB 减
VT 垂直制表	ESC 换码
FF 走纸控制	FS 文字分隔符
CR 回车	GS 组分隔符
SO 移位输出	RS 记录分隔符
SI 移位输入	US 单元分隔符

注：(1) 取决于使用这种代码的机器，它的符号可以是弯曲符号，向上箭头，或（—）标记。

(2) 取决于使用这种代码的机器，它的符号可以是在下面画线，向下箭头，或心形。

附录 2　8088／8086指令系统

附表 2.1　指令符号说明

符　号	说　明
r8	任意一个 8位通用寄存器 AH、AL、BH、BL、CH、CL、DH、DL
r16	任意一个 16位通用寄存器 AX、BX、CX、DX、SI、DI、BP、SP
reg	代表 r8、r16
seg	段寄存器 CS、DS、ES、SS
m8	一个 8位存储器单元
m16	一个 16位存储器单元
mem	代表 m8、m16
i8	一个 8位立即数
i16	一个 16位立即数
imm	代表 i8、i16
dest	目的操作数
src	源操作数
label	标号

附表 2.2　指令汇编格式

指令类型	指令汇编格式	指令功能简介	备　注
传送指令	MOV reg/mem, imm	dest← src	
	MOV reg/mem/seg, reg		CS除外
	MOV reg/seg,mem		CS除外
	MOV reg/mem, seg		
交换指令	XCHG reg, reg/mem	reg⟷ reg/mem	
	XCHG reg/mem, reg		
转换指令	XLAT label	AL← [BX＋AL]	
	XLAT		

续表

指令类型	指令汇编格式	指令功能简介	备　注
堆栈指令	PUSH r16/m16/seg	入栈	
	POP r16/m16/seg	出栈	CS除外
标志传送	CLC	CF←0	
	STC	CF←1	
	CMC	CF←$\overline{CF}$	
	CLD	DF←0	
	STD	DF←1	
	CLI	IF←0	
	STI	IF←1	
	LAHF	AH←标志寄存器低字节	
	SAHF	标志寄存器低字节←AH	
	PUSHF	标志寄存器入栈	
	POPF	出栈到标志寄存器	
地址传送	LEA r16,mem	r16←16位有效地址	
	LDS r16,mem	DS:r16←32位远指针	
	LES r16,mem	ES:r16←32位远指针	
输入	IN AL/AX, i8/DX	AL/AX←I/O端口 i8/DX	
输出	OUT i8/DX ,AL/AX	I/O端口 i8/DX←AL/AX	
加法运算	ADD reg, imm/reg/mem	dest←dest+src	
	ADD mem, imm/reg		
	ADC reg, imm/reg/mem	dest←dest+src+CF	
	ADC mem, imm/reg		
	INC reg/mem	reg/mem←reg/mem+1	
减法运算	SUB reg, imm/reg/mem	dest←dest−src	
	SUB mem, imm/reg		
	SBB reg, imm/reg/mem	dest←dest−src−CF	
	SBB mem, imm/reg		
	DEC reg/mem	reg/mem←reg/mem−1	
	NEG reg/mem	reg/mem←$\overline{reg/mem}$+1	
	CMP reg, imm/reg/mem	dest−src	
	CMP mem, imm/reg		
乘法运算	MUL reg/mem	无符号数乘法	
	IMUL reg/mem	有符号数乘法	
除法运算	DIV reg/mem	无符号数除法	

续表

指令类型	指令汇编格式	指令功能简介	备 注
	IDIV reg/mem	有符号数除法	
符号扩展	CBW	将 AL符号扩展为 AX	
	CWD	将 AX符号扩展为 DX.AX	
十进制调整	DAA	将 AL中的加和调整为压缩 BCD码	
	DAS	将 AL中的减差调整为压缩 BCD码	
	AAA	将 AL中的加和调整为非压缩 BCD码	
	AAS	将 AL中的减差调整为非压缩 BCD码	
	AAM	将 AX中的乘积调整为非压缩 BCD码	
	AAD	将 AX中的非压缩 BCD码转成二进制	
逻辑运算	AND reg, imm/reg/mem	dest←dest AND src	
	AND mem, imm/reg		
	OR reg, imm/reg/mem	dest←dest OR src	
	OR mem, imm/reg		
	XOR reg, imm/reg/mem	dest←dest XOR src	
	XOR mem, imm/reg		
	TEST reg, imm/reg/mem	dest AND src	
	TEST mem, imm/reg		
	NOT reg/mem	reg/mem←$\overline{\text{reg/mem}}$	
移位	SAL reg/mem, 1/CL	算术左移 1位/CL指定的位数	
	SAR reg/mem, 1/CL	算术右移 1位/CL指定的位数	
	SHL reg/mem, 1/CL	逻辑左移 1位/CL指定的位数	
	SHR reg/mem, 1/CL	逻辑右移 1位/CL指定的位数	
	ROL reg/mem, 1/CL	循环左移 1位/CL指定的位数	
	ROR reg/mem, 1/CL	循环右移 1位/CL指定的位数	
	RCL reg/mem, 1/CL	带进位循环左移 1位/CL指定的位数	
	RCR reg/mem, 1/CL	带进位循环右移 1位/CL指定的位数	
串操作	MOVS [B/W]	串传送	
	LODS [B/W]	串读取	
	STOS [B/W]	串存储	
	CMPS [B/W]	串比较	
	SCAS [B/W]	串扫描	
	REP	重复前缀	
	REPZ/REPE	相等重复前缀	
	REPNZ/REPNE	不等重复前缀	

续表

指令类型	指令汇编格式	指令功能简介	备　注
控制转移	JMP label	无条件直接转移	
	JMP r16/m16	无条件间接转移	
	Jcc label	条件转移	cc可为 C/NC/Z/NZ/S/NS/O/NO/B/NB/BE/NBE/L/NL/LE/NLE
循环	LOOP label	CX←CX−1;若 CX≠0,则循环	
	LOOPZ/LOOPE label	CX←CX−1;若 CX≠0且 ZF=1,则循环	
	LOOPNZ/LOOPNE label	CX←CX−1;若 CX≠0且 ZF=0,则循环	
	JCXZ label	若 CX=0,则循环	
子程序	CALL label	直接调用	
	CALL r16/m16	间接调用	
	RET	无参数返回	
	RET i16	有参数返回	
中断	INT 8	中断调用	
	INTO	溢出中断调用	
	IRET	中断返回	
处理器控制	NOP	空操作指令	
	seg:	段跨越前缀	除 CS
	HLT	停机指令	
	LOCK	封锁前缀	
	WAIT	等待指令	
	ESC men	换码指令	

附表 2.3　状态符号说明

符　号	说　明
−	标志位不受影响
0	标志位清 0
1	标志位置 1
x	标志位按定义功能设置
#	标志位按指令的特定说明设置
u	标志位不确定

附表 2.4 指令对状态标志的影响 (未列出的指令不影响标志)

指 令	OF	SF	ZF	AF	PF	CF
SAHF	—	#	#	#	#	#
POPF/IRET	#	#	#	#	#	#
ADD/ADC/SUB/CMP/NEG/CMPS/SCAS	x	x	x	x	x	x
INC/DEC	x	x	x	x	x	—
MUL/IMUL	#	u	u	u	u	#
DIV/IDIV	u	u	u	u	u	u
DAA/DAS	u	x	x	x	x	x
AAA/AAS	u	u	u	x	u	x
AAM/AAD	u	x	x	u	x	u
AND/OR/XOR/TEST	0	x	x	u	x	0
SAL/SAR/SHL/SHR	#	x	x	u	x	#
ROL/ROR/RCL/RCR	#	—	—	—	—	#
CLC/STC/CMC	—	—	—	—	—	#

附录 3　IBM PC／AT中断功能表

向量号	中 断 功 能	注　释
00H	除法错中断	其中 0～4为 CPU专用内部中断
01H	单步中断	
02H	NMI中断	
03H	断点中断	
04H	溢出中断	
05H	屏幕打印	
06H、07H	保留	
08H	定时器时钟中断（IRQ_0）	08H～0FH为外部可屏蔽中断
09H	键盘中断（IRQ_1）	
0AH	供 IRQ_8～IRQ_{15}串接（IRQ_2）	PC和 PC／XT为保留
0BH	异步通信 COM2中断（IRQ_3）	
0CH	异步通信 COM1中断（IRQ_4）	
0DH	并行打印机 LPT2中断（IRQ_5）	PC／XT为硬盘中断
0EH	软盘中断（IRQ_6）	
0FH	并行打印机 LPT1中断（IRQ_7）	
10H	显示 I/O功能程序	10H～1FH为 ROM BIOS中断
11H	设备配置检测	
12H	存储器容量检测	
13H	磁盘 I/O功能程序	
14H	串行通信 I/O功能程序	
15H	盒式磁带机 I/O功能程序	PC／XT机不使用
16H	键盘 I/O功能程序	
17H	打印机 I/O功能程序	
18H	ROM BASIC入口	
19H	引导程序入口	
1AH	日时钟 I/O功能程序	
1BH	键盘中止控制	
1CH	定时器定时	

续表

向量号	中 断 功 能	注 释
1DH	显示器初始化参数	
1EH	磁盘参数	
1FH	图形字符集	
20H	程序结束	20H～3FH为 DOS中断
21H	DOS功能调用	
22H	结束地址	
23H	中止（Ctrl+Break）处理	
24H	关键性错误处理	
25H	绝对磁盘读	
26H	绝对磁盘写	
27H	程序驻留结束	
28H～3FH	DOS内部使用或保留	
40H	硬盘 I/O功能程序	PC机不使用
41H	硬盘参数	PC机不使用
42H～6FH	保留,用户使用或未使用	
70H	实时时钟中断（IRQ_0）	70H～77H为外部可屏蔽中断
71H	软件使其重新指向 IRQ_2（IRQ_9）	PC和 PC/XT无法使用
72H	保留（IRQ_{10}）	
73H	保留（IRQ_{11}）	
74H	保留（IRQ_{12}）	
75H	协处理机中断（IRQ_{13}）	
76H	硬盘中断（IRQ_{14}）	
77H	保留（IRQ_{15}）	
78H～FFH	未使用或保留给 BASIC使用	

附录 4 常用 DOS功能调用 (INT 21H)

AH	功 能	入 口 参 数	出 口 参 数
00H	程序终止	CS=程序段前缀的段地址	
01H	键盘输入并回显		AL=输入字符
02H	显示输出	DL=输出字符	
03H	串行通信输入		AL=接收字符
04H	串行通信输出	DL=发送字符	
05H	打印机输出	DL=打印字符	
06H	直接控制台 I/O	DL=FFH (输入),DL=字符 (输出)	AL=输入字符
07H	键盘输入无回显		AL=输入字符
08H	键盘输入无回显 检测 Ctrl-Break 或 Ctrl-C		AL=输入字符
09H	显示字符串	DS:DX=字符串地址,字符串以'$'结尾	
0AH	键盘输入到缓冲区	DS:DX=缓冲区首址	(DS:DX)=缓冲区最大字符数 (DS:DX+1)=实际输入的字符数
0BH	检验键盘状态		AL=00H有输入,AL=FFH无输入
0CH	清输入缓冲区并执行指定的输入功能	AL=输入功能号 (1、6、7、8、0AH)	
0DH	磁盘复位		清除文件缓冲区
0EH	选择磁盘驱动器	DL=驱动器号 (0=A,1=B,…)	AL=系统中驱动器数
0FH	打开文件	DS:DX=FCB首地址	AL=00H文件找到,AL=FFH文件未找到
10H	关闭文件	DS:DX=FCB首地址	AL=00H目录修改成功,AL=FFH未找到
11H	查找第一个目录项	DS:DX=FCB首地址	AL=00H找到,AL=FFH未找到
12H	查找下一个目录项	DS:DX=FCB首地址,文件名中可带*或?	AL=00H文件找到,AL=FFH未找到

续表

AH 功能	入口参数	出口参数
13H 删除文件	DS:DX＝FCB首地址	AL＝00H删除成功,AL＝FFH未找到
14H 顺序读文件	Ds:DX ＝FCB首地址	AL＝00H读成功 AL＝01H文件结束,记录无数据 AL＝02H DTA空间不够 AL＝03H文件结束,记录不完整
15H 顺序写文件	DS:DX＝FCB首地址	AL＝00H写成功 AL＝01H磁盘满或只读文件 AL＝02H DTA空间不够
16H 创建文件	DS:DX＝FCB首地址	AL＝00H创建成功,AL＝FFH无磁盘空间
17H 文件改名	DS:DX＝FCB首地址 (DS:DX ＋1) ＝旧文件名 (DS:DX ＋17) ＝新文件名	AL＝00H改名成功,AL＝FFH不成功
19H 取当前磁盘驱动器		AL＝当前驱动器号 (0＝A,1＝B,…)
1AH设置 DTA地址	DS:DX＝DTA地址	
1BH取默认驱动器 FAT 信息		AL＝每簇的扇区数 DS:BX ＝FAT标识字符 CX ＝物理扇区的大小 DX ＝驱动器的簇数
21H 随机读文件	DS:DX＝FCB首地址	AL＝00H读成功 AL＝01H文件结束 AL＝02H缓冲区溢出 AL＝03H缓冲区不满
22H 随机写文件	DS:DX＝FCB首地址	AL＝00H写成功 AL＝01H盘满 AL＝02H缓冲区溢出
23H 测定文件长度	DS:DX＝FCB首地址	AL＝0成功,文件长度填入 FCB AL＝FFH未找到
24H 设置随机记录号	DS:DX＝FCB首地址	
25H 设置中断向量	DS:DX＝中断向量,AL＝中断类型号	
26H 建立程序前缀 PSP	DX＝新的 PSP	

续表

AH 功能	入口参数	出口参数
27H 随机分块读	DS:DX＝FCB首地址	AL＝00H读成功
	CX＝记录数	AL＝01H文件结束
		AL＝02H缓冲区溢出
		AL＝03H缓冲区不满
		CX＝读取的记录数
28H 随机分块写	DS:DX＝FCB首地址	AL＝00H写成功
	CX＝记录数	AL＝01H盘满
		AL＝02H缓冲区溢出
29H 分析文件名	ES:DI=FCB首地址	AL＝00H标准文件
	DS:SI=ASCII串	AL＝01H多义文件
	AL＝控制分析标志	AL＝FFH非法盘符
2AH取日期		CX:DH:DL＝年:月:日
2BH设置日期	CX:DH:DL＝年:月:日	AL＝00H成功,AL＝FFH无效
2CH取时间		CH:CL＝时:分,DH:DL＝秒:1/100秒
2DH设置时间	CH:CL＝时:分,DH:DL＝秒:百分秒	AL＝00H成功,AL＝FFH无效
2EH设置磁盘检验标志	AL＝00关闭检验,AL＝01打开校验	
2FH取 DTA地址		ES:BX＝DTA首地址
30H取 DOS版本号		AL＝版本号,AH＝发行号
31H程序终止并驻留	AL＝返回码,DX＝驻留区大小	
33H Ctrl-Break检测	AL＝00取状态	DL＝00H关闭检测,DL＝01H打开检测
	AL＝01置状态(DL＝00H关闭,DL＝01H打开)	
35H获取中断向量	AL＝中断类型号	ES:BX＝中断向量
36H取可用磁盘空间	DL＝驱动器号	成功:AX＝每簇扇区数,BX＝有效簇数,
	0＝默认,1＝A,2＝B,…	CX＝每扇区字节数,DX＝总簇数
		失败:AX＝FFFFH
38H置/取国家信息	AL＝00取国别信息	BX＝国别代码
	AL＝FFH国别代码放在 BX中	DS:DX＝返回的信息区首地址
	DS:DX＝信息区首地址	AX＝错误代码
	DX＝FFFFH设置国别代码	
39H建立子目录	DS:DX＝ASCII串地址	AX＝错误码
3AH删除子目录	DS:DX＝ASCII串地址	AX＝错误码

续表

AH 功能	入口参数	出口参数
3BH改变当前目录	DS:DX＝ASCII串地址	AX＝错误码
3CH建立文件	DS:DX＝ASCII串地址，CX＝文件属性	成功：AX＝文件代号；失败：AX＝错误码
3DH打开文件	DS:DX＝ASCII串地址 AL＝0/1/2读/写/读写	成功：AX＝文件代号；失败：AX＝错误码
3EH关闭文件	BX＝文件号	失败：AX＝错误码
3FH读文件或设备	DS:DX＝数据缓冲区地址 BX＝文件号 CX＝读取的字节数	成功：AX＝实际读出的字节数， AX＝0已到文件尾 出错：AX＝错误码
40H写文件或设备	DS:DX＝数据缓冲区地址， BX＝文件号，CX＝写入的字节数	成功：AX＝实际写入的字节数 出错：AX＝错误码
41H删除文件	DX:DX＝ASCII串地址	成功：AX＝00；失败：AX＝错误码
42H移动文件指针	BX＝文件号，CX:DX＝位移量 AL＝移动方式	成功：DX:AX＝新指针位置 出错：AX＝错误码
43H读取/设置文件属性	DS:DX＝ASCII串地址 AL＝0/1取/置文件属性，CX＝文件属性	成功：CX＝文件属性 失败：AX＝错误码
44H设备 I/O控制	BX＝文件号；AL＝0取状态，AL＝1置状态， AL＝2,4读数据，AL＝3,5写数据， AL＝6取输入状态，AL＝7取输出状态	成功：DX＝设备信息 失败：AX＝错误码
45H复制文件号	BX＝文件号 1	成功：AX＝文件号 2；出错：AX＝错误码
46H强制复制文件号	BX＝文件号 1，CX＝文件号 2	成功：AX＝文件号 1；出错：AX＝错误码
47H取当前目录路径名	DL＝驱动器号，DS:SI＝ASCII串地址	DS:SI＝ASCII串地址；失败：AX＝错误码
48H分配内存空间	BX＝申请内存容量	成功：AX＝分配内存首址 失败：BX＝最大可用空间
49H释放内存空间	ES＝内存起始段地址	失败：AX＝错误码
4AH调整已分配的内存空间	ES＝原内存起始地址 BX＝再申请的内存容量	失败：AX＝错误码 BX＝最大可用空间

续表

AH	功 能	入 口 参 数	出 口 参 数
4BH	装入/执行程序	DS:DX=ASCII串地址 ES:BX=参数区首地址 AL=0/3装入执行/装入不执行	失败:AX=错误码
4CH	带返回码终止	AL=返回码	
4DH	取返回码		AL=返回码
4EH	查找第一个匹配文件	DS:DX=ASCII串地址,CX=属性	AX=错误码
4FH	查找下一个匹配文件	DS:DX=ASCII串地址,文件名中可带*或?	AX=错误码
54H	读取磁盘写标志		AL=当前标志值
56H	文件改名	DS:DX=旧 ASCII串地址 DS:DX=新 ASCII串地址	AX=错误码
57H	设置/读取文件日期和时间	BX=文件号,AL=0读取 AL=1设置 (DX:CX)=日期:时间	DX:CX=日期:时间 失败:AX=错误码

附录 5　BIOS功能调用

INT 10H	AH	功　能	调 用 参 数	返 回 参 数
	0	设置显示方式	AL=00 40×25黑白文本,16级灰度 =01 40×25 16色文本 =02 80×25黑白文本,16级灰度 =03 80×25 16色文本 =04 320×200　4色图形 =05 320×200黑白图形,4级灰度 =06 640×200黑白图形 =07 80×25黑白文本 =08 160×200 16色图形 (MCGA) =09 320×200 16色图形 (MCGA) =0A 640×200 4色图形 (MCGA) =0D 320×200 16色图形 (EGA/VGA) =0E 640×200 16色图形 (EGA/VGA) =0F 640×350单色图形 (EGA/VGA) =10 640×350 16色图形 (EGA/VGA) =11 640×480黑白图形 (VGA) =12 640×480 16色图形 (VGA) =13 320×200 256色图形 (VGA)	
	1	置光标类型	$(CH)_{0-3}$=光标起始行，$(CL)_{0-3}$=光标结束行	
	2	置光标位置	BH=页号,DH/DL=行/列	
	3	读光标位置	BH=页号	CH=光标起始行 CL=光标结束行 DH/DL=行/列
	4	读光笔位置		AX=0光笔未触发 =1光笔触发

续表

INT AH	功　能	调用参数	返回参数
			CH/BX=像素行/列
			DH/DL=字符行/列
10　5	置当前显示页	AL=页号	
10　6	屏幕初始化或上卷	AL=0初始化窗口	
		AL=上卷行数,BH=卷入行属性	
		CH/CL=左上角行/列号	
		DH/DL=右上角行/列	
10　7	屏幕初始化或下卷	AL=0初始化窗口	
		AL=下卷行数,BH=卷入行属性	
		CH/CL=左上角行/列号	
		DH/DL=右上角行/列	
10　8	读光标位置的字符和属性	BH=显示页	AH/AL=字符/属性
10　9	在光标位置显示字符和属性	BH=显示页 AL/BL=字符/属性,CX=字符重复次数	
10　A	在光标位置显示字符	BH=显示页 AL=字符,CX=字符重复次数	
10　B	置彩色调色板	BH=彩色调色板ID BL=和ID配套使用的颜色	
10　C	写像素	AL=颜色值,BH=页号,DX/CX=像素行/列	
10　D	读像素	BX=页号,DX/CX=像素行/列	AL=像素的颜色值
10　E	显示字符(光标前移)	AL=字符,BH=页号,BL=前景色	
10　F	取当前显示方式		BH=页号,AH=字符列数 AL=显示方式
10　10	置调色板寄存器(EGA/VGA)	AL=0,BL=调色板号,BH=颜色值	
10　11	装入字符发生器(EGA/VGA)	AL=0～4全部或部分装入字符点阵集	
		AL=20～24置图形方式显示字符集	
		AL=30读当前字符集信息	ES:BP=字符集位置

续表

INT	AH	功 能	调用参数	返回参数
10	12	返回当前适配器设置的信息（EGA／VGA）	BL＝10H（子功能）	BH＝0单色方式 ＝1彩色方式 BL＝VRAM容量 （0＝64 K, 1＝128 K, …） CH＝特征位设置 CL＝EGA的开关设置
10	13	显示字符串	ES:BP＝字符串地址 AL＝写方式（0～3）, CX＝字符串长度 DH／DL＝起始行／列, BH／BL＝页号／属性	
11		取设备信息		AX＝返回值（位映像） 0＝设备未安装 1＝设备未安装
12		取内存容量		AX＝字节数（KB）
13	0	磁盘复位	DL＝驱动器号 （00, 01为软盘, 80h, 81h, …为硬盘）	失败：AH＝错误码
13	1	读磁盘驱动器状态		AH＝状态字节
13	2	读磁盘扇区	AL＝扇区数 $(CL)_{6,7}$ $(CH)_{0\sim7}$ ＝磁道号 $(CL)_{0\sim5}$ ＝扇区号 DH／DL＝磁头号／驱动器号 ES:BX＝数据缓冲区地址	读成功： AH＝0, AL＝读取的扇区数 读失败： AH＝错误码
13	3	写磁盘扇区	同上	写成功： AH＝0, AL＝写入的扇区数 写失败： AH＝错误码
13	4	检验磁盘扇区	AL＝扇区数 $(CL)_{6,7}$ $(CH)_{0\sim7}$ ＝磁道号 $(CL)_{0\sim5}$ ＝扇区号 DH／DL＝磁头号／驱动器号	成功：AH＝0 AL＝检验的扇区数 失败：AH＝错误码

续表

INT	AH	功　能	调 用 参 数	返 回 参 数
13	5	格式化盘磁道	AL=扇区数 $(CL)_{6,7}$ $(CH)_{0\sim7}$=磁道号，$(CL)_{0\sim5}$=扇区号 DH/DL=磁头号/驱动器号 ES:BX=格式化参数表指针	成功：AH=0 失败：AH=错误码
14	0	初始化串行口	AL=初始化参数 DX=串行口号	AH=通信口状态 AL=调制解调器状态
14	1	向通信口写字符	AL=字符 DX=通信口号	写成功：$(AH)_7$=0 写失败：$(AH)_7$=1 $(AH)_{0\sim6}$=通信口状态
14	2	从通信口读字符	DX=通信口号	读成功：$(AH)_7$=0 AL=字符 读失败：$(AH)_7$=1
14	3	取通信口状态	DX=通信口号	AH=通信口状态 AL=调制解调器状态
15	0	启动盒式磁带机		
15	1	停止盒式磁带机		
15	2	磁带分块读	ES:BX=数据传输区地址 CX=字节数	AH=状态字节 =00读成功 =01冗余检验错 =02无数据传输 =04无引导 =80非法命令
15	3	磁带分块读	DS:BX=数据传输区地址 CX=字节数	AH=状态字节 (同上)
16	0	从键盘读字符		AL=字符码 AH=扫描码 ZF=0　表示已按键
16	1	取键盘缓冲区状态		AL=字符码，AH=扫描码

续表

INT	AH	功 能	调用参数	返回参数
				ZF＝1键盘缓冲区空
16	2	取键盘状态字节		AL＝键盘状态字节
17	0	打印字符 回送状态字节	AL＝字符 DX＝打印机号	AH＝打印机状态字节
17	1	初始化打印机 回送状态字节	DX＝打印机号	AH＝打印机状态字节
17	2	取打印机状态字节	DX＝打印机号	AH＝打印机状态字节
1A	0	读时钟		CH：CL＝时：分 DH：DL＝秒：1/100秒
1A	1	置时钟	CH：CL＝时：分 DH：DL＝秒：1/100秒	
1A	6	置报警时间	CH：CL＝时：分（BCD） DH：DL＝秒：1/100秒（BCD）	
1A	7	清除报警		

附录 6　DEBUG 命令

DEBUG 的命令

命　令　极　其　功　能	格　　式
A (ssemble) 对语句进行汇编	A [<地址 >]
C (ompare)　比较内存内容	C<源地址范围 ><目标 >
D (ump)　　显示内存内容	D [<地址 >] 或 [<地址范围 >]
E (nter)　　修改内存内容	E<地址 > [<字节串 >]
F (ill)　　填充内存内容	F<地址范围 ><要填入的字节或字节串 >
G (o)　运行一个程序或程序段	G [=<始址 >] [<断点 >...]
H (exaritmetic)　十六进制的加减法运算	H<数值 1><数值 2>
I (nput)　读/显示输入字节	I<端口地址 >
L (oad)　装入文件或磁盘扇区	L [<地址 > [<盘号 ><相对扇区 ><扇区数 >]]
M (ove)　传送内存块	M<源地址范围 ><目标地址 >
N (ame)　定义文件名和参量	N<文件名 > [<文件名 >...]
O (utput)　输出命令	O<端口地址 ><字节 >
Q (uit)　退出 DEBUG	Q
R (egister)　显示和修改寄存器内容	R [<寄存器 >]
S (earch)　对字符进行检索	S<地址范围 ><要检索的字节或字节串 >
T (race)　跟踪执行和显示	T [=<地址 >] [<跟踪条数 >]
U (nassemble) 对指令 (二进制) 进行反汇编	U [<地址范围 >]
W (rite)　写入文件或磁盘扇区	W [<地址 > [<盘号 ><相对扇区 ><扇区数 >]]

附录 7　汇编语言程序上机过程

1. 编辑源程序

用字处理软件创建源程序。常用编辑工具有：EDIT.COM、记事本、WORD、WPS等。无论采用何种编辑工具，生成的文件必须是纯文本文件，且文件扩展名为 asm。

设有实现字节和的源程序文件 abc.asm如下：

```
DATA1    SEGMENT
     BUF   DB   47H,6AH
     SUM   DB   ?
DATA1    ENDS
CODE1    SEGMENT
         ASSUME   CS:CODE1,DS:DATA1
START:   MOV    AX,DATA
         MOV    DS,AX
         MOV    AL,BUF
         ADD    AL,BUF+1
         MOV    SUM,AL
         INT    20H
CODE1    ENDS
         END    START
```

2. 汇编

⑴ 汇编过程

当源程序建立以后，设汇编源程序为 abc.asm，用汇编程序 MASM 对 abc.asm源程序文件进行汇编，操作步骤如下：（以下有下划线的部分为用户键盘输入，“f ”代表回车，未划线部分为屏幕显示。）

```
C\MASM > MASM   abc.asmf
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) MicrosoftCrop 1981－1985, 1987, All rights reserved
```

Object filename [abc.obj] 输入目标文件名,若采用括号 [] 中的名字,按f

Source listing [NUL.LST] 若需要列表文件,输入文件名,按f ;如果不需要列表文件,按f 。

列表文件同时给出源程序和机器语言程序清单,并给出符号表,为调试程序带来方便。

Cross-reference [NUL.CRF] 若需要交叉索引文件,输入文件名 ;如果不需要交叉索引文件,按f 。

49758 + 451602 Bytes symbol space free

0 Warning Errors

0 Severe Errors

回答上述问题后,汇编程序就对源程序进行汇编。若汇编中发现源程序有语法错误,则列出错误语句所在行、错误代码及错误性质说明。错误分警告错误(Warning Errors)和严重错误(Severe Errors)。警告错误指汇编程序认为的一般性错误 ;严重错误指汇编程序认为无法进行正确汇编的错误,给出其错误个数,错误性质。这时,就要对错误进行分析,找出问题和原因,然后再调用编辑程序加以修改。修改后重新汇编,直到无错误为止。

除了用上述方法分步回答问题外,还可以用命令行的形式按顺序对四个提示予以回答,其格式是:

MASM 源文件名,目标文件名,列表文件名,交叉引用文件名

其中文件名都不必给出扩展名,汇编程序会按照缺省情况使用或产生。若只想对部分提示给出回答,则在相应位置用逗号隔开,若不想对剩余部分作答,则用分号结束。例如以下命令经汇编后在当前目录下产生 abc.obj和 abc.lst文件,不产生 .crf文件。

C\MASM>masm abc,,abc;f

(2) 显示 LST文件和 REF文件

①显示 LST文件

.LST文件是文本文件,任何文本编辑器均可打开它,DOS下,可用 TYPE命令进行查看。

C\MASM> TYPE abc.lst

Microsoft (R) Macro Assembler Version 5.00 2/6/4

Page 1-1

```
1 0000                          DATA1  SEGMENT
2 0000  47 6A                   BUF    DB   47H,6AH
3 0002  ??                      SUM    DB   ?
```

```
 4 0003                          DATA1   ENDS
 5 0000                          CODE1   SEGMENT
 6                                 ASSUME  CS:CODE1, DS:DATA1
 7 0000  B8 ---- R             START:MOV   AX, DATA1
 8 0003  8E D8                     MOV     DS, AX
 9 0005  A0 0000 R                 MOV     AL, BUF
10 0008  02 06 0001 R              ADD     AL, BUF+1
11 000C  A2 0002 R                 MOV     SUM, AL
12 000F  CD 20                     INT     20H
13 0011                         CODE1   ENDS
14                                 END     START
Microsoft (R) Macro Assembler Version 5.00                  2/6/4
Symbols-1
Segments and Groups:
                N a m e         Length   Align   Combine Class
CODE1  . . . . . .              0011   PARA   NONE
DATA1  . . . . . .              0003   PARA   NONE
Symbols:
                N a m e         Type     Value   Attr
BUF  . . . . . .                L BYTE   0000   DATA1
START  . . . . . .              L NEAR   0000   CODE1
SUM  . . . . . .                L BYTE   0002   DATA1
@ FILENAME  . . . . . .         TEXT  abc
    14 Source  Lines
    14 Total   Lines
    7 Symbols
  49758 + 451602 Bytes symbol space free
     0 Warning Errors
     0 Severe  Errors
```

② 显示 REF文件

交叉索引文件 CRF不是文本文件,不能用 TYPE 命令或文本编辑器打开。若需要阅读,必须将索引文件 (二进制形式的文件)转换为交叉引用文件 (文本文件)。方法是执行程序 CREF. EXE,将索引文件变换为交叉引用文件。操作步骤如下:

C>MASM >CREF f

Cref Filename [.CRF] : bc f 输入交叉索引文件名。

List Filename [outchar.REF] : bcf 输入交叉引用文件名。

交叉引用表给出用户定义的所有符号,对于每个符号列出其定义所在行号加上#及引用的符号。它为较大程序的修改提供了方便,而一般较小程序则用处不大。

abc.ref文件如下:

3. 连接

设汇编后产生目标程序文件为 abc.obj,连接程序操作方法和屏幕显示如下:

C\MASM > LINK abc.obj

IBM Personal Computer Linker Linker Version 3.60

Version 2.00 <C>Copyright IBM Corp 1981,1982,1983

Run File [abc.exe] 输入可执行文件名,若采用括号 [] 中的名字,按f

LIST FILE [NUL.MAP] 若需要映象文件,输入文件名,按↙ 如果不需要列表文件,按f

映象文件又称为连接程序的列表文件,它给出每个段在存储器中的分配情况。如 abc.map如下:

Warning: No STACK segment

Start	Stop	Length	Name	Class
00000H	00002H	0003H	DATA1	
00010H	00020H	0011H	CODE1	

Origin Group

Program entry point at 0001:0000

Libraries [NUL.LIB] 若需要库文件,输入文件名,按f ;如果不需要库文件,按f

Warning: No STACK segment

There was 1 error detected.

上述两行给出警告信息,表示用户程序没有定义堆栈段。该警告不影响可执行程序的生成和正常运行,因为运行时会自动使用系统提供的缺省堆栈。

回答上述问题后,连接程序开始连接,若连接过程中有错,则显示错误信息。此时,按错误提示,修改源程序,然后,回到第二步。直至汇编无错误,连接无错误(错误 No Stack Segment除外),在当前目录下产生可执行文件(.EXE),进入第四步。

同样也可以用命令行的形式按顺序对四个提示予以回答,其格式是:

LINK 目标文件名,可执行文件名,内存映象文件名,库文件名;

其中不必给出扩展名,连接程序会按照缺省情况使用。若只想对部分提示给出回答,则在相应位置用逗号隔开,若不想对剩余部分作答,由用分号结束。例如以下命令行对 abc.obj文件连接后在当前目录下产生 abc.exe文件,其余文件均不需要。

C\MASM > LINK abc;↵

从 6.0版以后,Microsoft公司把 MASM和 LINK的功能由一个 ML.EXE程序完成,只需一个命令就可以把源程序汇编并连接生成.EXE文件。而不再需要分两步操作。

ML.EXE常用格式为:

ML [/Fl] [/Fm] [/Fr] [/c] 源文件

其中,源文件名的扩展名(.asm)不能省,[ ]中的 F必须大写,l,m,r,c必须小写,各可选项含义如下:

/Fl:产生.lst列表文件,缺省时不产生。

/Fm:产生.map内存映象文件,缺省时不产生。

/Fr:产生.sbr交叉参考文件,缺省时不产生。

/c:只产生,obj目标文件,不产生.exe可执行文件。缺省时只产生.exe文件。

例 ML /Fl abc.asm

以上命令会对已存在的 abc.asm文件汇编且连接,并在当前目录下生成一个列表文件 abc.lst和一个可执行文件 abc.exe

4. 程序运行

设连接生成的执行文件为 abc.exe,运行程序只需在提示符下键入文件名即可。例:

C\MASM > abc.exe↵

若程序能够运行但不能得到预期结果,则需要静态或动态查错。静态查错即检查源程序,并在源程序级用文本编辑器进行修改,然后再汇编、连接、运行。如果静态检查无法发现错误,则需动态查错。

5. 程序调试及结果查看

这里使用 DEBUG程序进行调试,DEBUG是一种使用广泛的强有力的汇编语言程序或二进制文件的调试工具。

设 abc.asm经汇编及连接后,产生 abc.exe文件,对它的调试操作如下:

(1) 把程序装入内存,并显示程序清单

C\MASM >DEBUG abc.exef 在 DEBUG后所带文件名为调试对象。

－ “－”是提示符,等待键入命令。

－Uf 用反汇编命令 U列出程序清单

0892:0000 B81B1A MOV AX,0891

0892:0003 8ED8 MOV DS,AX

0892:0005 8A260100 MOV AL, [0000]

0892:0009 02260100 ADD AL, [0001]

0892:000D 88260200 MOV [0002] ,AL

0892:0011 CD20 INT 20

……

⑵ 查看数据段内容或查看寄存器内容

－D1A1Bf 查看数据段内容

0891:0000 47 6A 00 00 00 00 00 00－00 00 00 00 00 00 00 00 GJ.. ....

1A1B:0010 B8 91 08 8E D8 A0 00 00－02 06 01 00 A2 02 00 CD 8... X..... "..M

……

可见 0000、0001单元内容为数据段定义的内容,分别为 47H、6AH。 DEBVG状态下,缺省数制为十六进制数。

－Rf 查看寄存器内容

AX =08B1 BX =0000 CX =0080 DX =0000 SP=0000 BP=0000 SI=0000 DI=0000

DS=0891 ES=0881 SS=0891 CS=0892 IP=000C OV UP DING NZ AC PE NC

若查看指定寄存器内容,在 R命令后写上寄存器文件名,例:

－ R AXf 查看寄存器 AX的内容

AX 08B1

:f 写入新值,无需更改,按回车

⑶ 连续运行程序,查看结果

－ Gf 或 －G =0 0010f 运行程序,0为运行启点,0010为程序运行断点

－ D1A1Bf 查看内存结果

1A1B:0000 47 6A B1 00 00 00 00 00 00 00 00 00 00 00 00 00 GJ· · · · · · · · · · · · · · · ·

……

可见 0002单元中内容已变为 0000和 0001两单元内容之和。

⑷ 如果连续得不到正确结果,需分步跟踪运行

－ T=0000f　　0000这所需要运行指令的地址

AX =0891 BX =0000 CX=0000 DX =0000 SP=0000 BP=0000 SI=0000 DI =0000

DS=0881 ES=0881 SS=0891 CS=0892 IP=0003 NV UP DI PL NZ NA PO NC

0892:0003 8EDB　　MOV　DS, AX

显示单步执行后寄存器和标志状态之后,停在下一条指令开始处。

－ Tf　　顺序运行下一条指令

……

⑸ 退出 DEBUG

－ Qf

附录 8　键盘扫描码

索　　引

参考文献

1 沈美明等. IBM PC/XT汇编语言程序设计. 第 14版. 北京:清华大学出版社, 1999

2 姚君遗. 汇编语言程序设计. 第 4版. 南京:南京大学出版社, 2001

3 孙德文. 微型计算机技术. 北京:高等教育出版社, 2001

4 戴梅萼, 史嘉权. 微型计算机技术及应用. 从 16位到 32位. 第 2版. 北京:清华大学出版社, 1996

5 周明德. 微型计算机系统原理及应用. 第 3版. 北京:清华大学出版社, 1998

6 钱晓捷, 陈涛. 16/32位微机原理、汇编语言及接口技术. 北京:机械工业出版社, 2001

7 Muhammad Ali Mazidi, Janice Gillispie Mazidi. The 80×86 IBM PC AND COMPATIBLE COMPUTERS VOLOMES I & II ASSEMBLY LANGUAGE, DESIGN, AND INTERFACING. Fourth Edition. Pearson Education, Inc., publishing as Prentice Hall, Inc., 2003

8 Intel公司. Intel Component Data Catalog, 1985

9 Intel公司. Pentium Processor Family User's Manual, 1994

10 Hitachi, Ltd., Semiconductor & Integrated Circuits, HM628511HC Series 4M High Speed SRAM (Rev. 2.0). www.hitachisemiconductor.com, Feb. 3, 2003

11 Atmel Corporation. 256 K (32 K x 8) Paged Parallel EEPROM — AT28C256 (Rev. 0006H). www.atmel.com, 1999

12 Analog Devices, Inc., 10Bit/12Bit Parallel ADCs: AD7470/AD7472 (REV. A). www.analog.com, 2000

13 Analog Devices, Inc., Parallel Interface Single Voltage-Output 8/10/12Bit DACs: AD5330/AD5331/AD5340/AD5341 (REV. 0). www.analog.com, 2000

14 National Semiconductor Corporation. ADC0808/ADC0809 8-Bit µP Compatible A/D Converters with 8-Channel Multiplexer. www.national.com, October 1999

15 National Semiconductor Corporation. DAC0830/DAC0832 8Bit µP Compatible Double-Buffered D to A Converters. www.national.com, May 1999