单片机原理及接口技术学习辅导

朱定华　刘 玉　编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
URL: http://www.phei.com.cn

单片机原理及接口技术学习辅导

朱定华　刘　玉　编著

電子工業出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书是为配合《单片机原理及接口技术》教材而编写的辅导资料。书中对该教材中的重点和难点进行了详细论述，尤其是“难点分析”采用了问答形式，针对性很强。

本书各章都配有“例题详析”，所举之例大都很典型，可加深对有关内容的理解，同时还给出了《单片机原理及接口技术》一书的全部习题解答。每章都附有一份单元测验，最后还给出了两套综合模拟试卷，所有单元测验及综合试题都给出了答案。

本书可作为大专院校学生学习单片机原理及应用、单片机接口技术等课程的辅助教材，也可作为报考研究生相关科目的复习参考书，对从事单片机教学和科研的教师和技术人员也很有参考价值。

图书在版编目(CIP)数据

单片机原理及接口技术学习辅导/朱定华等编著 . - 北京:电子工业出版社,2001.8

ISBN 7-5053-6957-1

Ⅰ.单 ... Ⅱ.朱 ... Ⅲ.①单片微型计算机 - 理论 - 高等学校:技术学校 - 教学参考资料
②单片微型计算机 - 接口 - 高等学校:技术学校 - 教学参考资料

Ⅳ.TP368.1

中国版本图书馆 CIP 数据核字(2001)第 056993 号

书　　名：单片机原理及接口技术学习辅导
编　　著：朱定华　刘　玉
责任编辑：束传政
特约编辑：李　莉
排版制作：电子工业出版社计算机排版室监制
印 刷 者：北京市增富印刷有限责任公司
出版发行：电子工业出版社　URL:http://www.phei.com.cn
　　北京市海淀区万寿路 173 信箱　邮编 100036
经　　销：各地新华书店
开　　本：787×1092　1/16　印张：16.25　字数：416 千字
版　　次：2001 年 8 月第 1 版　2001 年 8 月第 1 次印刷
书　　号：ISBN 7-5053-6957-1
　　TN·1466
印　　数：6000 册　　定价：19.00 元

前　言

单片机因其独特的优点而广泛应用于各行各业，越来越多的大专院校相继开设了单片机原理与接口技术的课程，越来越多的企业也把掌握单片机技术作为招聘技术人员的标准之一。但是，在学习单片机原理和接口技术的过程中，不少人感到困惑和吃力，这是因为单片机“麻雀虽小，五脏俱全”，为了弄清一个概念，往往要牵扯到更多的新概念。为此，我们将二十多年的科研和教学经验加以总结和整理而写成本书，针对学习过程中可能出现的各种疑难之处给予详细解答，相信读者看了这本书之后一定会受益匪浅。

本书共分 8 章，各章的结构一致，都由内容提要、本章重点、难点分析、例题详析、习题详解等几部分组成，书后配有单元测验和两套综合模拟试卷。这本书的独特之处有三，一是“难点分析”以问答形式编写，许多问题都是各届学生共同的疑惑，其解答很有针对性和实用性；二是书中出现的所有例题、习题和试题，都无一遗漏地全部给出了答案，其中绝大部分还列出了详细的解题过程，这将大大方便读者自学和复习应考；三是将单元测验题及模拟的两套试卷都特意设计成百分制的标准试卷，这不仅可以让读者及时检测自己的学习情况（全部答案置于附录 A），也在一定程度上减轻了教师的工作量。同时，题型的多样化，对各级主考单位也是一个较好的参考。

本书由朱定华和刘玉编写。参加本书编写工作的人员还有曹丹华、张鄂亮、朱兰、李志文、程世平、林卫、王虹、崔厚孝、陆晓华、汪宇翔、汤勇、张鹏等。

由于时间仓促，书中定有不少错漏之处，衷心欢迎读者随时反映意见和建议，来信请寄：

430074　武汉市华中科技大学电信系　朱定华，刘玉

电子邮件信箱为：jimu.room@263.net

注：书中提到的“配套教材”系指由电子工业出版社出版朱定华编写的《单片机原理及接口技术》。

编　者

于武昌喻家山

2001 年 8 月

第1章 概 述

1.1 内容提要

数制以及不同数制间的转换是本章的第一个主要内容;有关计算机(特别是微型计算机、单片机)的一系列基本概念是本章的第二大内容;而本章的第三大内容则是 MCS-51 单片机的基本结构,这一部分是全新的概念,既是本章重点又是本章难点,需要认真掌握。

1. 数制及其转换的主要内容

因为二进制数及其编码是计算机运算的基础,计算机惟一能识别的数是二进制数,计算机的指令、数据、字符、地址均用二进制数表示,所以掌握二进制数是非常重要的。为了书写方便、读数直观,引入了十六进制数,但这仅是一种手段。因为人们习惯用十进制数,所以又引入了 BCD 码。对于键盘上的各种符号,在计算机内的表示法是世界统一的,称为 ASCII 码,这种代码在数据通信时也常常使用。因此,我们需要熟悉二、十和十六进制这 3 种数制之间的转换,也要掌握 BCD 码与 ASCII 码的含义和用途。

在微型计算机中可以实现二进制的加、减、乘、除等基本运算,为了简化电路、降低成本、加快运算速度,引入了补码表示法。利用补码表示法,可以将二进制数的减法运算变为加法运算,还可以将符号数和无符号数统一起来。

微型计算机不仅能进行算术运算,而且能进行逻辑运算。基本的逻辑运算有“与”、“或”、“非”、“异或”4 种。所有的逻辑运算都是按位操作的。

2. 微型计算机的结构及有关概念

建议读者以冯·诺伊曼计算机体系结构(计算机硬件的 5 大部件——运算器、控制器、存储器、输入和输出)作为引导主线,这样可以引出并记住一系列关于计算机的基本概念,例如什么是处理器(CPU),什么是主机,什么是计算机,什么是计算机系统,前四代计算机的主要区别是什么等。并给出了微型计算机、单板微机和单片微机的定义,特别强调了微处理器、微型计算机、微机系统之间的差异,以及单片微机的独特优点和用途。

3. MCS-51 单片机基本结构

本书选用的教学机型是 MCS-51 系列 8 位单片微机,它的硬件组成仍符合冯·诺伊曼体系结构,只是 5 大部件集成在一块硅片内,其独特优点是控制能力强、性价比高、可靠性好。特别需要说明的是 MCS-52 子系列是 MCS-51 子系列的增强型,内部结构大体相同,区别仅是片内存储单元的容量扩大以及增加了一个定时器,片内所含部件详见表 1-1。

表 1-1 MCS-51 单片机片内所含部件

子系列	片内 ROM 形式			ROM 容量	RAM 容量	SFR 字节数	寻址范围	I/O 特性			中断源
	无	ROM	EPROM					定时器	并行口	串行口	
51 子系列	8031	8051	8751	4KB	128B	21	2×64KB	2×16	4×8	1	5
	80C31	80C51	87C51	4KB	128B	21	2×64KB	2×16	4×8	1	5
52 子系列	8032	8052	8752	8KB	256B	26	2×64KB	3×16	4×8	1	6
	80C32	80C52	87C52	8KB	256B	26	2×64KB	3×16	4×8	1	6

在 MCS-51 单片机的片内结构中，尤以片内 RAM 区的划分最为复杂、细致，牵扯到的概念也相当多，如果能彻底看懂、理解和牢牢掌握图 1-1 的内容，那就说明你对 MCS-51 单片机内部结构已经具有清晰的概念了。

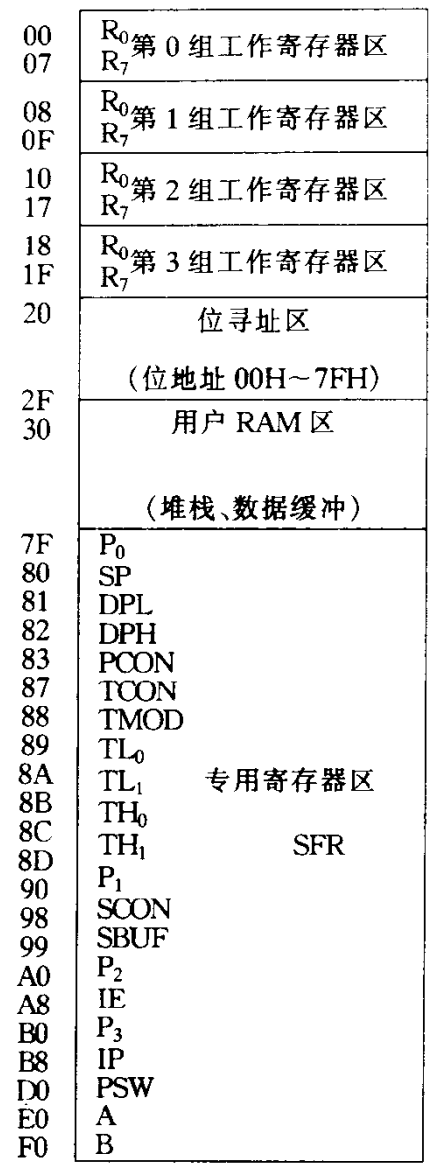

图 1-1　MCS-51 子系列机片内 RAM 空间划分

1.2　本章重点

1.2.1　数制及转换

1. 数的表示

真值——一般书写时表示的数(即带有 ± 号的数)。

机器数——计算机中编码表示的数(正数和负数的符号以最高位为 0 或 1 来表示)。

无符号数——最高位不留作符号位。

符号数——最高位是符号位，不参与表示数值的大小，后续位才是数值位。

二进制符号数的表示法有多种：原码表示法、反码表示法和补码表示法。其共同点是：最高位均为符号位，“0”表示正数而“1”表示负数，且它们对正数的表示完全一致。不同点是：原码的数值位将负数的绝对值照搬；反码是将负数的绝对值取反；而补码是将负数的绝对值求补，即取反后再做尾部加 1 处理。由于补码在运算时的合理性，计算机内通常选用补码表示法。

2. 数的转换

N 进制转换为十进制——将各位之权与对应之幂相乘展开，再累计求和即可。

十进制转换为 N 进制——将整数和小数分开，分别转换后再拼接。

转换口诀：整数部分除基取余(首次余数为代码整数最低位)；小数部分乘基取整(首次整数为小数点后最高位)。

特别需要说明的是，求十进制分数的二进制数，不必先转换为小数再求，完全可以根据“乘基取整”的方法直接求出，既快又准。

N_1 进制转换为 N_2 进制通常有两种办法，一是利用十进制当做转跳板，二是利用 N_1 与 N_2 进制间的某种联系而直接转换。例如二进制转十六进制或二进制转八进制，可以用“以 4 当 1”或“以 3 当 1”的办法迅速转换。

数制转换的实例详见本章“例题详析”中例 1-1～1-4。

3. 两种常用编码

BCD 码——以二进制数表示十进制数的编码，一般采用 4 位二进制数来表示 1 位十进制数(即压缩 BCD 码)。其特点是：4 位之内为二进制关系，每 4 位之间为十进制关系。用途：方便计算机对十进制数的直接输入、输出、存储及运算，处理每 4 位之间的合理进位靠的是一条特殊指令 DA A，称为二至十进制调整指令。

ASCII 码——美国标准信息代码，它是一种字符编码，为计算机中 100 多个字母、符号（包括常用键盘控制符）规定了特殊的 7 位二进制码，如数字“0”的 ASCII 码为 30H，数字“9”的 ASCII 码为 39H 等。

1.2.2 有关计算机的一系列概念

牢记计算机硬件结构包括五大部件：运算器、控制器、存储器、输入接口和输出接口，这根主线贯穿全书始终。其中，运算器、控制器、存储器合称为主机，运算器、控制器合称为中央处理单元或处理器（CPU）。若这两部分集成在同一块集成电路硅片上，则称为微处理器（MPU）。

五大部件集中在一块印刷电路板（且含 MPU）的计算机称为单板微型计算机；将五大部件集中在同一块集成电路硅片上的计算机称为单片机；这五大部件加上对应的系统软件和应用软件等合称为（微型）计算机系统。

用电子管实现这五大部件的计算机称为第一代电子计算机；用晶体管实现这五大部件的计算机称为第二代电子计算机；用中小规模集成电路实现这五大部件的计算机称为第三代电子计算机；用大规模和超大规模集成电路实现这五大部件的计算机称为第四代电子计算机。

1.2.3 MCS-51 单片机内部结构

1）运算器和控制器（即 CPU）的主要作用。

2）MCS-51 存储器的特点（四大物理空间或三大逻辑空间）。

3）片内程序存储空间和数据存储空间的地址安排、容量大小和各自用途，特别是低端 128 字节的数据 RAM 的各单元用途。

4）21 个特殊功能寄存器（SFR）的用途。

5）从运算器、控制器和存储器的作用入手，结合程序计数器 PC、累加器 A 以及有关的 SFR（如堆栈指针 SP，程序状态字 PSW，数据总线复用口 P_0，地址总线复用口 P_1 和 P_2，控制总线复用口 P_3 等）来深刻理解指令执行过程。

1.3 难点分析

- **补码的概念和用途**

实际上，补码的概念适用于任何进制。因为其最本质的定义是：若一个数 x 与另一个数 y 相加正好得到 x 或 y 的最小进位（注意要保持 x 与 y 的长度相同），则称 x 与 y 互为补数，这个最小进位的数值称为“模”。例如，十进制数 9 的补数是 1，其“模”是 10；而 09 的补数是 91，其“模”就是 100。

对 x 求补的运算其实很简单，就是用“模”减去 x 所得到的差。

补码能使减法变成加法的概念也适用于任何进制数。例如，9 − 3 的运算可以改用 $9+(3)_{补}$ 并丢弃模的运算来求得，即变成 9 + 7 = 16，丢弃模 10 之后得到 6。当然，在一般情况下我们不会舍近求远采用补码方式来进行十进制数的运算，但在二进制数中补码就有用武之地了。由于二进制数的特殊性，使得“用模减掉二进制数 x”的运算可以简单地转化为“对 x 取反加 1”，即计算机可以用一反相器和加法器轻易求补，从而使二进制数的减法能用加法器实

现。

补码常用于:①符号数的存储;②符号数的运算。

• 计算机能自动识别无符号数和符号数吗?

凡是能在计算机内存储或参与运算的都是二进制形式的机器数,计算机只能识别"0"和"1",对于某个二进制数的最高位究竟应看做为符号位还是数值位,理论上是无法自动识别的。只能靠用户在编写程序时做到"心中有数"。对于8位二进制数,看成字节无符号数,其表示范围是0~255(00H~FFH);看成字节符号数,其表示范围是-128~+127(80H~7FH)。

计算机所进行的运算都是无符号数运算,既把符号数的符号位当做数值进行运算,又把所有数的运算结果当做符号数来影响溢出标志位。

但是,由于引入了补码概念,使得计算机在进行无符号数和符号数的运算时能够实现操作的一致性,且结果合理。例如,将无符号数1FH与D0H相加,或是将符号数1FH与D0H相加,其结果都是EFH。编制无符号数加法程序的用户会将该结果的真值认为是239,即31+208,而编制符号数加法程序的用户则会将该结果的真值认为是-17,即31+(-48)。

• 补码和求补操作有区别吗?怎样根据真值求补码,或根据补码求真值?

补码是表示符号数的一种码制,而"求补"则是一种操作,即是指计算机机械地"取反加1"的操作。求补操作可以对任何数(不论数是正还是负,也不论数是何种码)执行求补操作。事实上我们只会遇到两种求补数的情况:一是求负数的补码,用绝对值"取反加1"的求补运算得其补码;二是求负数(补码)的真值,可先将该补码数用"取反加1"的求补运算得其绝对值,再在绝对值前添加一负号。

求补码的一个通俗口诀是"正数不变,负数求绝对值之补"。

要求一个十进制数(即真值)的补码,应当先判断该十进制数是负数还是正数。若该十进制数是正数,则其补码就是原数本身,也就是说,只需将它转换为二进制数,而不需再执行取反加1的求补操作。若是负数,则要先将原数的绝对值用二进制数表示出来,然后对这个二进制数执行求补操作。例如,求68的补码时,就只需将68转换为二进制数01000100B(或十六进制数44H);而求-68的补码时,则应先写出-68的绝对值(即68)的二进制数01000100B(或十六进制数44H),然后对01000100B执行"取反加1"的求补操作,最后得到的补码为BCH。

还有一种情况是初学者容易困惑的,那就是如何根据某个补码还原其真值,但如果对前面介绍的补码概念已经很清楚,那就不会对这种逆运算感到太困难。

根据补码求真值的一个通俗口诀是"正码不变,负码求补,补后勿忘添负号"。

请首先识别该补码(即机器数)的最高位是否为"0",若为"0",则意味着对应的真值必为正数(符号为+),而正数在当初写成补码时就是它本身,并没有多余的求补动作。所以,将该补码直接按权按幂展开即可。例如要求补码44H的真值,就可以简单地计算出其真值为+(4×16+4)=+68。但若要求BCH的真值,则由于其最高位为"1",真值必为负数(符号为-),该数在当初写成补码时一定是将原数求补了的(即对其绝对值取反加1过),现在就要对该补码再次求补以恢复其绝对值(是不是有点"负负得正"的味道?),因此我们对BCH求补操作后得到44H,这就是真值x的绝对值,而真值x为-44H=-(4×16+4)=-68。千万别忘记添加负号!

要说明的是，这种“对 x 的补码再求补以得到$|x|$”的方法，其理论根据是“$|x|$与$|x|_{补}$显然互为补数，二者之和必定产生最小进位(模)”。

• 8 位补码数的表示范围为何是不对称的 -128～127？80H 究竟是 +128 还是 -128？

8 位原码和反码的表示范围都是对称的 -127～+127，而补码不对称的原因是，“0”(无论是 +0 还是 -0)只占用了一个编码。这便节省下来一个编码值，可以让 -127～+127 的某一边界再拓宽一点。那么，该编码值加给哪一边更合适呢？如果让 +127 变为 +128 的话，会导致对应的 8 位二进制代码的最高位为“1”，变成负数的含义，不可取。所以只能加到负数边界，让 -127 变为 -128，此时对应的 8 位二进制代码是 10000000B，其十六进制数为 80H。

要回答 80H 对应的十进制数究竟是多少，首先要确定 80H 究竟是指无符号数还是符号数。如果指的是无符号数，则 80H 的大小就是 +128；如果 80H 是补码数(计算机内的符号数一律用补码表示)，其真值的计算应当遵循“取反加 1”后添加一负号的规则，这便得到 -80H=-128 的结果。

初学者常常感到困惑：为何对 80H 进行求补操作之后得到的还是 80H？没错，因为 80H+80H 可以得到一个最小进位嘛，所以无符号数 80H(+80H)与补码数 80H(-80H)当然就互为补数了。

请注意“补码”与“补数”的区别，符号数在计算机内必须以补码形式来存放和参与运算，而负数转化为补码的方法便是求补操作——计算其绝对值的补数(绝对值取反加 1)。

• 对符号数的运算结果怎样才能“保号”？

在进行加减类算术运算时，只要保证两个原始操作数是正确的补码形式，则结果必然能“保号”；若不能保号则说明此时“溢出”，计算机会自动置位 OV 标志。

在进行乘除、移位等运算时，需要用户在编程时多加小心，人为添加保号动作，确保符号正确。例如，将 -4 除以 2 的操作，计算机内通常是将 -4 的补码 11111100B 右移一次得到，如果右移时没有注意从左边添加“1”，则会出现不合理的结果 0111110。因此，一个能“保号”的具体技巧是每次右移时用符号位(D_7 位)从左边加以弥补。

当需要将单字节符号数扩展为双字节时，也需要注意“保号”。对于用补码表示的数，正数的符号扩展应该在其前面补 0；而负数的符号扩展，则应该在前面补 1。例如：68 用 8 位表示为 44H，用 16 位表示为 0044H；-68 用 8 位表示为 BCH，用 16 位表示为 FFBCH。

原码数的扩展则比较简单，例如单字节扩展为双字节时，仅将符号位移至双字节的最高位(D_{15})即可。

无符号数是正数，无符号数的扩展是在其前面补 0。如无符号数 255 用 8 位表示为 FFH，用 16 位表示为 00FFH。

• 进位和溢出有什么不同？

两数运算的结果若没有超出字长的表示范围，则由此产生的进位是自然进位；若两数的运算结果超出了字长的表示范围(即结果不合理)，则称为溢出。例如将正数 3FH 和负数 D0H 相加，其结果不会超出 8 位字长的表示范围，所以其结果 1 0FH 中的进位是正常进位(也就是模)。但是，若正数 3FH 与正数 70H 相加，其结果为 AFH，最高位为“1”，成了负数的含义，这就不合理了，这种情况称为溢出。

判别结果是否溢出，除了配套教材中介绍的 $C_y \oplus C_s$ 外，还可以采用双符号补码，即：将 1 个符号位扩展为 2 个符号位，将 8 位二进制数的运算变成 9 位运算(舍去比双符号更高的进

位位)，若结果中的两个符号位相同(异或为0)，则表示无溢出；若不同(异或为1)，则表示有溢出。且双符号位为“01”时表示正溢出；为“10”时表示负溢出。例如用双符号位法计算3FH+D0H，相当于[00] 0111111B+[11] 1010000B=1[00] 0001111B，两符号位相同，无溢出。再如，用双符号位法计算3FH+70H，相当于[00] 0111111B+[00] 1110000B=[01] 0101111B，两符号位不相同且为“01”形式，说明有溢出，而且是正溢出(即超出了正数的字长表示范围)。

若结合程序控制字(PSW)中的进(借)位标志C和溢出标志位OV的概念，此问题可以表述为：参与运算的数不论是无符号数还是符号数，计算机都按符号数来置位溢出标志、按无符号数来置位进位标志。当运算结果超出了8位字长所能表示的范围时，即字节运算结果超出了80H～7FH(－128～＋127)时溢出，OV将置1，否则OV置0。对加法运算来说，如最高位向前产生了进位，则进位标志置1，否则置0；对减法运算来说，最高位向前产生了借位(即不够减)时，进位标志置1，不产生借位(即够减)时进位标志置0。

• “十转N”和“N转十”的数制转换运算有什么快速方法吗？

把一个十进制数转换为二进制数，可以先把该数转换为十六进制数，然后再转换为二进制数，这样可以减少计算次数；反之，要把一个二进制数转换为十进制数，也可以采用同样的办法。若记住2^n和十六进制数、十进制数的对应关系(如表1-2所示)以及某些十进制整数(如50，100，200等)和十六进制数的对应关系(如表1-3所示)，转换起来则更为快捷方便，具体实现方法请见本章的例题详析和习题详解。

表1-2　部分二进制数与十进制数的对应关系

2^n	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
十六进制数	20	40	80	100	200	400	800	1000	2000	4000	8000	10000
十进制数	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536

表1-3　部分十进制数与十六进制数的对应关系(可自行添加)

无符号数	十进制	50	100	127	200	255	500	1000	65535		
	十六进制	32H	64H	7FH	C8H	FFH	01F4H	03E8H	FFFFH		
符号数	十进制	50	100	127	－1	－50	－100	－127	－128		
	十六进制	32H	64H	7FH	FFH	BEH	9CH	81H	80H		

还需要说明的是，在汇编语言源程序中可以使用二进制数、十六进制数，也可以使用十进制数，不需要编程做数制的转换，它们的转换工作可由宏汇编程序完成。但是，若需用键盘输入的十进制数参与运算，或需将运算的结果以十进制数形式输出，则要编写程序将输入的十进制数字串(ASCII码)转化为二进制数，或将二进制数转化为十进制数字串。

• BCD数是十进制数，为何也以H为单位？与十六进制数的区别何在？

BCD数的特点是用4位二进制数编码来表示1位十进制数，那么8位二进制数可以表示两位BCD数。例如十进制数29的BCD数在计算机内存放的二进制码形式为：00101001B，为

了书写方便，再将二进制数的式样写成十六进制数的式样，那就变成了 29H。

有的教科书为了将 BCD 数 29 与普通十六进制数 29H 区分开，规定将 BCD 数 29 写成 29BCD 的形式，但由于它在计算机中的存放形式仍然是 00101001B，用十六进制数来记忆和书写仍然是 29H，所以很少有人用麻烦的“BCD”书写形式，而是约定俗成地沿用“H”这个单位。

也就是说，十进制数 29 的 BCD 数本应该写成 29BCD，但因为它在计算机内存放为二进制数 00101001B，即十六进制数 29H，所以我们可以简单地把任意一个两位十进制数 XY 写成机内表示形式：XYH。

或许有人会问，当计算机内放置的是 29H 时，它到底是普通的十六进制数呢，还是 BCD 数 29？这要看当初这个数据是以何种“身份”存入的，或者要根据程序的上下文来判断。

例如，欲让计算机来执行 29＋18＝?，可以先将两个加数分别转为二进制数（1DH 和 12H）之后，再调用二进制数的加法程序计算 1DH＋12H，最后再将结果转回十进制数；也可以用 BCD 形式直接存入两个加数 29H 和 18H，再调用 BCD 数的加法程序来完成 29H＋18H。这时的结果就是 BCD 数，不必再转换了。

• 单片机与一般微型计算机最根本的不同之处及其独特优点是什么?

单片机与一般计算机的本质区别是，它的五大部件是集成在一块集成电路硅片上的。其独特优点有三：控制功能强、有优异的性能价格比、有很高的可靠性。其应用范围早已涵盖各行各业，特别是广泛用于在线控制系统。事实上，单片机的应用正从根本上改变着传统的控制系统设计思想和设计方法，形成一种全新的概念——微控制技术。

• 8052 机型属于 MCS-51 系列吗？它与 8051 机型有哪些不同?

8052 机型仍属于 MCS-51 系列，工作原理和指令集与 8051 机型完全一样，只是扩充了片内部分硬件资源，例如片内 RAM 从 128B 扩展为 256B，片内 ROM 从 4KB 扩展为 8KB，定时器从 2 个扩展为 3 个，与扩充部件配套的 SFR 由 21 个字节增加到 26 个字节。

• 单片机内部的存储空间究竟有多少?

MCS-51 单片机的设计是将程序存储空间（简称 Mp）与数据存储空间（简称 Md）从物理上分开（称为哈佛结构）。除了 8031/8032 机型没有片内 Mp 外，其他型号均有片内 Mp 且容量达 4KB 或 8KB。至于片内数据空间则小得多，仅使用 8 位地址，未超过 256B。而且，这类存储空间按不同用途又细分为不同小区，低端的 128B 是数据 RAM，而高端的 128B 作为特殊功能寄存器（SFR）空间，其中只有 21 个字节有效，其他字节形同坏区，无法读写。

• 片内低端 128 字节的数据空间能用来存放和执行程序吗？它们究竟有什么用途?

简略回答是：能存放，但不能执行！片内数据 RAM 用处非常大，大到离了它几乎无法执行指令的地步！

只要定义为“数据存储”的空间，无论位于片内还是片外，肯定不能执行指令，因为程序空间的地址选通信号（$\overline{\text{PSEN}}$）根本就没接到这里来！但是，若把程序代码当做普通数据放入数据区暂存（注意，仅仅是存放！）也没什么不可以，只是别试图运行它，除非把$\overline{\text{PSEN}}$选通信号预先接到数据存储器芯片的片选端上。可真要如此接线，此数据空间的性质也就演变成为程序空间了。

片内低端 128 字节的数据 RAM 分为 3 个地址区间，各有不同用途，请参见图 1-1。但也可作为普通寄存器使用，其中：

字节地址 00H～1FH——分配给 4 组通用寄存器，每组寄存器名称相同，都是 R_0～R_7；4 组 R_n 不能同时使用，当前选择使用哪一组 R_n，要由 PSW 的 RS_1 和 RS_0 两位来决定。

字节地址 20H～2FH——既可作为普通的字节寄存器，也可作为位寻址区，每个字节均可分为 8 个可位寻址的存储元(bit，位)，此区间 128 个存储元按位另外统一编址(即位地址)，分别从 00H～7FH，常用来存放实际控制中的各种状态信息。

字节地址 30H～7FH——普通的字节寄存器，常用来存放临时数据，又称数据缓冲区。其中最高端的若干字节常用于教学开发机各 LED 数码管的数据缓冲区，配合监控程序中的显示模块使用。例如江苏启东厂的教学开发机就是将 7EH～79H 这 6 个存储单元当做 LED 显示缓冲区，无论用户预置内容是 BCD 码还是七段代码，都可以在监控程序中找到对应的显示模块，通过调用来完成用户所需的数值显示。

片内高端 128 字节的 RAM 区只有 21 个字节有效，这 21 个字节既有地址，又有各自独特的名称，且用途各异，因此称为特殊功能寄存器(SFR)空间，其中凡字节地址能被 8 整除的 SFR 均可实现位寻址，其也有独立的位地址，且编号规律与低端位寻址区不同，与各自所在 SFR 字节地址相关。

• 在 21 个字节的 SFR 中，哪些寄存器最重要？

应该说，全都很重要，别忘了它们都叫做“特殊功能”寄存器。

不过，在 MCS-51 指令集内，涉及面最宽的一个寄存器是 ACC(累加器)，其次是 PSW，而一旦涉及到片外扩展，P_0～P_3 锁存器又显得分外重要，因为它们将代行系统总线(AB、DB、CB)的功能。

在本书前 5 章中可能用到的寄存器只有 6 个，即：A，B，PSW，SP，DPH 和 DPL(合称为 DPTR)。

与并行接口或片外扩展有关的寄存器是 P_0～P_3。

与中断控制有关的是 IE 和 IP，还有 TCON。

与定时器有关的 SFR 有 TCON，TMOD，TH_0，TL_0，TH_1，TL_1。

与串行接口有关的是 SCON、PCON 和 SBUF。

这 21 个 SFR 在上电复位时除了 SP＝07H、P_0～P_3 锁存器为 FFH 外，其余全部为 00H。

• 把累加器 A 写成 A 与写成 ACC 有什么不同？

A 和 ACC 虽指的是同一个寄存器，但在指令中它们是有区别的。ACC 在汇编后的机器指令中必有一个字节的操作数是 ACC 的字节地址 EOH，A 则隐含在指令操作码中。所以符号指令中的 A 不能用 ACC 代替；反之，特殊功能寄存器直接寻址和位名称寻址要用 ACC，而不能用 A 代替。例如，符号指令 INC A 的机器码是 04H，写成 INC ACC 之后则成了 INC direct 的格式，对应的机器码成为 05E0H。

• 程序状态字 PSW 中各个状态位是从哪里得到的？其中两个未安排的自定义位(PSW.5 和 PSW.2)用户能任意使用吗？

PSW 又称标志寄存器，主要用来存放指令执行过程中产生的各种特殊信息，这些信息分别来自于：

PSW.7(C 标志)——来自运算器最近一次算术运算操作，表示运算结果有无进(借)位？

PSW.6(AC 标志)——来自运算器最近一次算术运算操作，表示运算结果的低 4 位与高 4 位之间有无进位？

PSW.4 和 PSW.3(工作寄存器选择)——来自用户指令,由用户指定当前即将选用哪组通用寄存器?

PSW.2(OV 标志)——来自运算器最近一次算术运算操作,表示运算结果有无溢出?

PSW.0(P 标志)——来自最近一次涉及对累加器 A 的操作指令(包括传送操作),表示操作后 A 中数据有奇数个"1"吗?

至于 PSW 中两个未安排的位 PSW.5(F0)和 PSW.2(F1),用户可以视其为普通存储元(bit 位)而单独使用,例如在编程时用来存放某些二态信息。

• 堆栈和堆栈指针是不是一回事?堆栈与一般存储空间有什么不同?不用堆栈不行吗?

简答:堆栈和堆栈指针不是一回事,堆栈是一段存储空间,而堆栈指针只是这段存储空间的地址指示器,它与一般存储器读写规则不同,用途也不同。如果没有堆栈,子程序调用和中断实现等重要功能就很难实现。

详述:堆栈由一块数据存储区和一个地址指针(即堆栈指针 SP)组成。该存储区一端封闭而另一端开口,数据的存取只能在开口端(即栈顶)进行,存取原则为"先进后出"。MCS-51 把堆栈设在片内数据 RAM 中,且设计为"向上生成"式(即从低地址向高地址增加),堆栈最深只有 128 字节。

SP 寄存器内所装的数据,其含义永远是栈顶地址,也就是说,堆栈存储器的位置是由堆栈指针 SP 给定的。应特别注意栈顶是随着堆栈指示器 SP 的变化而变化的,因此在使用堆栈时,应特别注意堆栈指针的内容,不可随意改变,否则会造成混乱。

堆栈与一般的数据存储器的相同之处是:它们都属于 Md(数据存储器)空间,都可以存储数据信息,并且都能对数据进行读写操作。不同之处是:各自的读写规则不同,普通 Md 可以随机读写其中的任一字节,所用指令是 MOV 类指令,而堆栈操作只能在栈顶单元进行,遵循的是"先进后出"原则,所用指令是 PUSH 或 POP。另外,二者在用途上也很不相同,一般 Md 用来存放普通数据,而堆栈常用来存放程序断点地址以及作为保护现场和恢复现场之用,在计算机自动进行存放 16 位断点地址的栈操作时,它是自动进行双字节(或字)操作的。

计算机内如果没有堆栈,就无法实现子程序调用、子程序嵌套和中断控制等。

请注意,由于堆栈也是存储器的一个区域,所以,只要不对堆栈区进行特殊的栈操作,也就没有"先进后出"和字操作的限制,此时可以把它当做普通存储器,并使用任何其他访问 Md 的指令来访问这个区间。

1.4 例题详析

例 1-1 将十六进制数 lF3DH 转换为十进制数。

方法 1:将十六进制数按权展开相加。

$$\begin{aligned}1F3DH &= 1\times16^3+15\times16^2+3\times16^1+13\times16^0\\ &= 4096+3840+48+13\\ &= 7997\end{aligned}$$

方法 2:用 2^n 和十六进制数、十进制数的对应关系写出十进制数再相加(减)。

$$\begin{aligned}1F3DH &= 2000H-100H+3DH\\ &= 2000H-100H+40H-3\\ &= 8192-256+64-3\end{aligned}$$

$=7997$

例 1-2 将十进制数 38947 转换为十六进制数。

方法 1：用除 16 取余求。

16 | 38947	余数为 3	低位	↑
16 | 2434	余数为 2		
16 | 152	余数为 8		
16 | 9	余数为 9	高位	
0			

即 $38947=9823H$

方法 2：用 2^n 和十六进制数、十进制数的对应关系求。

$$\begin{aligned}38947 &= 32768+6179\\ &= 32768+4096+2048+35\\ &= 8000H+1000H+800H+20H+3\\ &= 9800H+23H\\ &= 9823H\end{aligned}$$

例 1-3 请计算无符号数：

① 1100111.101B= _____ D= _____ H

② 18.6= _____ B= _____ H

解 第①小题属数制转换中的“N 转十”情况。所以对于①，我们将各位之权与对应之幂相乘展开，再累计求和即可。不过，在展开时要注意整数部分的最低位是 2 的 0 次幂，小数部分的最高位是 2 的 −1 次幂，这两处是初学者最容易出错的地方。

$$\begin{aligned}1100111.101B &= 2^6+2^5+2^2+2^1+2^0+2^{-1}+2^{-3}\\ &= 103.625\end{aligned}$$

对于①中的“2 转 16”要求，可以利用 4 位二进制码相当于一位十六进制码的关系迅速求解，但注意“以 4 当 1”时应当以小数点为界，整数朝左划分，不足 4 位则高位补 0；小数部分朝右边划分，不足 4 位则低位补 0。具体做法如下：

1100111.101B=0110 0111.1010B=67.AH

第②小题属于“十转 N”情况，我们可利用转换口诀：“整数部分除基取余；小数部分乘基取整”。

对整数部分 18 除 2 取余的结果是

2 | 18	余数	↑ 0	低位
2 | 9	余数	1	
2 | 4	余数	0	
2 | 2	余数	0	
2 | 1	余数	1	高位
0				

即 18=10010B

对小数部分 0.6 乘 2 取整的结果是：

0.6×2=1.2	整数	1	高位
0.2×2=0.4	整数	0	
0.4×2=0.8	整数	0	
0.8×2=1.6	整数	1	
0.6×2=1.2	整数	1	
0.2×2=0.4	整数	0	低位

……

即 0.6=0. 1001 1001 ……B(这说明二进制码不能精确表示所有的十进制小数)

若小数部分取 8 bit 二进制码表示(且 0 舍 1 入),则 18.6D=10010.1001 1001B,其对应的十六进制数按照第①题的方法很易求出为 12.99H。

所以本题答案为:① 1100111.101B =103.625D=67.AH

② 18.6 =10010.1001 1001B =12.99H

例 1-4 求 ±68 的补码以及补码 C9H 的真值。

①已知真值求补码,要根据口诀"正数不变,负数求绝对值之补"来操作。

所以,对 +68,其补码就是它本身的二进制码,可轻松求出为 44H; 而对 −68,则应当先求其绝对值之补码(即 01000100B 或 44H 之补,请注意数据要保持 1 字节长度!)为 10111100B 或 BCH。

② 已知补码求真值,要根据口诀"正码不变,负码求补,补后勿忘添负号"来操作。

所以,对补码 C9H,首先要判断它的正负性,因其最高符号位为"1",属负数,所以应该再次求补以得到真值的绝对值,C9H 之补数为 100H − C9H(或 FFH − C9H + 1) = 37H = 55,然后勿忘给 55 添上负号,即 C9H= −55。

例 1-5 设 $x = -111/35$,当整数和小数部分各取 8 bit(0 舍 1 入)时,求 $[x]_{补}$ = _____ B= ____ H。

本题仍属"已知真值求补码",但真值为分数,比较特殊。

首先根据求补规则"正数不变,负数求绝对值之补"来求 $|x|$ 之补码。那么,要将 $|x|$ 表示为相应的二进制码或十六进制码,才能顺利求补。一般的求法是先将 x 值转为小数后再转换,但我们直接利用"整数部分除基取余;小数部分乘基取整"的原则来求解,计算过程反而显得更加简单。

首先把带分数 $\frac{111}{35}$ 中的整数分离出来,即 $|x| = 111/35 = 3 + 6/35$,3 的二进制码为 00000011,而 6/35 的二进制数求法为:

6/35×2=12/35	整数部分为 0	小数部分的高位
12/35×2=24/35	整数部分为 0	
24/35×2=48/35	整数部分为 1	
13/35×2=26/35	整数部分为 0	
26/35×2=52/35	整数部分为 1	
17/35×2=34/35	整数部分为 0	
34/35×2=68/35	整数部分为 1	
33/35×2=66/35	整数部分为 1	小数部分的低位
31/35×2=62/35	整数部分为 1	供 0 舍 1 入用(保留 8bit 小数)

即 6/35 的二进制数为 0.00101100B。

$|x|=3+6/35=00000011.00101100$B（因题意要求用两字节表示）

$|x|$的补数为$|x|$取反加 1，请特别注意，这个“1”应该加在小数（而不是整数）的尾部！

$|x|+1=$ 00000011.00101100 + 0.00000001 = 1111 1100.1101 0100B = FC.D4H

所以，当 $x=-111/35$ 时，其$[x]_{补}=11111100.11010100B=FC.D4H$。

例 1-6 请判断下列说法的正误。

(1) PC 可以看做指令存储区的地址指针。

(2) PC 是一个不可寻址的特殊功能寄存器。

(3) 一般情况下，用户无法直接修改 PC 内容，但可直接修改 SP 内容。

(4) 所有计算机系统的堆栈都是向地址高端逐渐生长的，即均为“向上生成”堆栈。

(5) SP 内装的是栈顶的数据内容。

(6) 在程序执行过程中，堆栈与 PC 没有任何联系。

答

(1) 正确。因为 PC 内存放的数据永远都是指令地址。

(2) 正确。PC 是位于片内的一个 16 位的寄存器，它专门存放当前要执行的指令地址，且能够自动加 1，具有特殊功能。因其内容之重要以及从系统安全性考虑，未分配地址，不让用户随意读写。

(3) 正确。因为 PC 不可寻址而 SP 可以被用户寻址。

(4) 不一定。在 MCS-51 系统中，数据压入堆栈时，其指针 SP 确实是逐渐增加（不断加 1）的，即堆栈是向上生成的，但在其他计算机系统中，也有将压栈后的 SP 设计为逐渐减少（不断减 1），那么堆栈就是向下生成的了。

(5) 错误。SP 内的数据永远是堆栈顶部的地址，而不是顶部地址中的数据，也就是说，SP 是一个地址指针。

(6) 错误。当涉及到子程序调用或中断响应等操作时，堆栈中将会自动装入当前主程序断点之处的 PC 值（连续装两字节）；而从子程序或中断服务子程序返回时，系统又会自动把栈顶的内容送往 PC 之中。可以说，能够正确保存和恢复主程序断点地址 PC 的内容，这才是堆栈最重要和最神奇的用途。

例 1-7 设 SP = 32H，片内 RAM 的 30H～32H 单元内容分别为 20H、23H、01H。试问执行下列指令后，堆栈指针 SP = ？ DPH = ？ DPL = ？ A = ？

地址	
30H	20H
31H	23H
SP→32H	01H

图 1-2 例 1-7 图

```
POP DPH
POP DPL
POP A
```

本题如图 1-2 所示。执行完上述 3 条指令后，SP = 2FH，DPH = 01H，DPL = 23H，A = 20H（如果答案是 SP = 29H，请自己找出错误的原因）。

1.5 习题详解

1.1 将表中的十进制数按表中要求转换后用十六进制数填入下表中。

十进制数	十六进制数	非压缩 BCD 数(H)	压缩 BCD 数(H)	ASCII 码(H)
10				
64				
78				
80				
93				
100				
125				
255				

解 部分数值转换的计算过程如下：

$80 = 10 \times 8 = 1010B \times 1000B = 01010000B = 50H$

$78 = 80 - 2 = 50H - 2 = 4EH$

$93 = 100 - 7 = 64H - 7 = 5DH$

$125 = 128 - 3 = 80H - 3 = 7DH$

填表如下：

十进制数	十六进制数	非压缩 BCD 数(H)	压缩 BCD 数(H)	ASCII 码(H)
10	0A	0100	10	3130
64	40	0604	64	3634
78	4E	0708	78	3738
80	50	0800	80	3830
93	5D	0903	93	3933
100	64	010000	0100	313030
125	7D	010205	0125	313235
255	FF	020505	0255	323535

1.2 将下列十六进制无符号数转换为十进制数：

2CH,D5H,CADH,4FEH,378H,64H,100H。

解 $64H = 100$

$100H = 256$

$2CH = 20H + 0CH = 32 + 12 = 44$

$D5H = 80H + 40H + 15H = 128 + 64 + 21 = 213$

$CADH = 1000H - 353H$

$= 1000H - 400H + ADH$

$= 1000H - 400H + 80H + 20H + 0DH$

$= 4096 - 1024 + 128 + 32 + 13$

$= 3245$

$4FEH = 500H - 2H = 400 + 100H - 2H = 1024 + 256 - 2 = 1278$

$378H = 400H - 88H = 400H - 80H - 8H = 1024 - 128 - 8 = 888$

1.3 写出下列十进制数的原码和补码，用二位、四位或八位十六进制数填入下表中。

十进制数	原码	补码	十进制数	原码	补码
18			928		
-18			-928		
30			8796		
-30			-8796		
347			65530		
-347			-65530		