《单片机 C 语言入门》（修订版）

作者：华清远见

第 1 章 MCS-51 单片机概述

本章目标

单片微型计算机简称单片机，又称为微控制器（MCU），是微型计算机的一个重要分支。单片机是 20 世纪 70 年代中期发展起来的一种大规模集成电路芯片，将 CPU、RAM、ROM、I/O 接口和中断系统等集成于同一硅片内。20 世纪 80 年代以来单片机发展迅速，各类新产品不断涌现，出现了许多高性能新型机种，在工业控制领域、家电产品、智能化仪器仪表、计算机外部设备和机电一体化产品中都有重要的用途。

单片机的开发主要分为两个方面，即按产品功能要求设计电路和编写程序。本书主要介绍 51 系列单片机的 C51 语言编程。但由于单片机的编程是和其硬件结构紧密相关的，因此有必要对单片机的硬件结构作概括性的介绍。汇编语言作为一种仍然广泛使用并且有生命力的语言，其基本知识对于单片机的应用而言也是必不可少的。本章主要讲解以下 3 个方面。

- MCS-51 单片机的结构
- MCS-51 单片机的指令系统
- MCS-51 单片机的汇编程序设计

1.1 MCS-51 单片机结构

MCS-51 是指美国 Intel 公司生产的一系列单片机的总称。这一系列单片机包括了多个种类，如 8031、8051、8751、8032、8052、8752 等。其中 8051 是最早、最典型的产品，该系列其他单片机都是以 8051 为核心电路发展起来的，都具有 8051 的基本结构和软件特征，所以人们习惯于用 8051 来称呼 MCS-51 系列单片机。

8051 单片机内部包含了微型计算机所必需的基本功能部件，各部件相互独立地集成在同一块芯片上，其基本功能特性有：

- 8 位 CPU；
- 32 条双向可独立寻址的 I/O 线；
- 4KB 程序存储器（ROM），外部可扩充至 64KB；
- 128 个字节数据存储器（RAM），外部可扩充至 64KB；
- 2 个 16 位定时/计数器；
- 5 个中断源；
- 全双工的串行通信口；
- 具有布尔运算能力。

8051 单片机内部结构框图如图 1.1 所示：

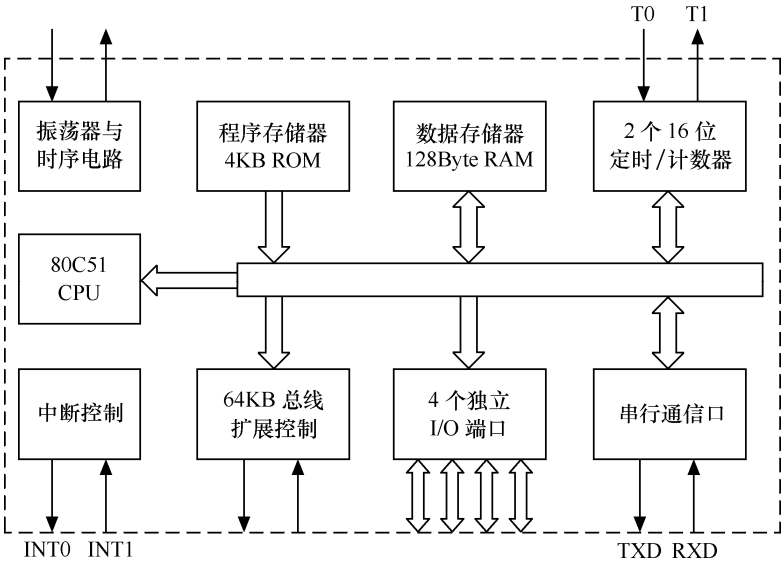


图 1.1　8051 单片机框图

从图 1.1 中可以看到，8051 单片机主要包含中央处理器（CPU）、程序存储器（ROM）、数据存储器（RAM）、定时器/计数器、并行接口、串行接口和中断系统以及数据总线、地址总线和控制总线，下面分别加以说明。

1.1.1 中央处理器（CPU）

中央处理器（CPU）是整个单片机的核心部件，是 8 位数据宽度的处理器，能处理 8 位二进制数据或代码，CPU 负责控制、指挥和调度整个系统的工作，完成运算和控制输入输出功能等操作，是单片机的核心部件，由运算器、控制器（定时控制部件）和专用存储器组 3 部分组成。

1．运算器（ALU）

运算器的功能是进行算术运算和逻辑运算。可以对半字节（4 位）、单字节等数据进行操作，既能够完成加、减、乘、除等四则运算，也可以完成加 1、减 1、BCD 码十进制调整、比较等算术运算和与、或、异或、求补、循环等逻辑运算。

8051 运算器还包含一个布尔处理器，用来处理位操作，以进位标志位 C 为累加器，可执行置位、复位、取反、等于 1 转移、等于 0 转移、等于 1 转移且清 0 以及进位标志位与其他可寻址的位之间进行数据传送等位操作，也能使进位标志位与其他可寻址的位之间进行逻辑与、或操作。

2．控制器

（1）时钟电路

8051 片内设有一个由反向放大器所构成的振荡电路，XTAL1 和 XTAL2 分别为振荡电路的输入端和输出端，时钟可以由内部方式或外部方式产生。内部方式时钟电路如图 1.2 所示。在 XTAL1 和 XTAL2 引脚上外接定时元件，内部振荡电路就产生自激振荡。定时元件通常采用石英晶体和电容组成的并联谐振回路。晶振频率在 1.2M～12MHz 范围内，电容在 5～30pF 范围内，电容的大小可起频率微调作用。

外部方式的时钟很少用，若要用时，只需将 XTAL1 接地，XTAL2 接外部振荡器即可，如图 1.3 所示。对外部振荡信号无特殊要求，只要保证脉冲宽度，一般采用频率低于 12MHz 的方波信号。

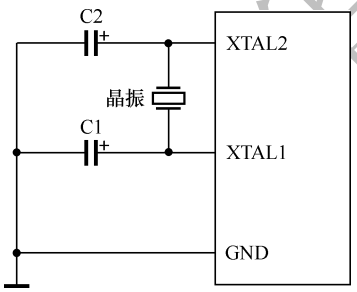


图 1.2　内部方式时钟电路

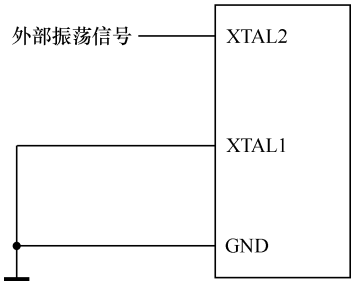


图 1.3　外部方式时钟电路

时钟频率越高，单片机控制器的控制节拍就越快，运算速度也越快，但同时消耗的功率也更大，对外界的干扰也更强。因此，不同型号、不同场合的单片机所需要的时钟频率是不一样的。

（2）振荡周期、时钟周期、机器周期和指令周期

一条指令译码产生的一系列微操作信号在时间上有严格的先后次序，这种次序就是计算机的时序。8051 的主要时序将在第 11 章中介绍，这里先介绍其基本时序周期。

- 振荡周期是为单片机提供时钟信号的振荡源的周期，是时序中最小的时间单位。
- 时钟周期是振荡源信号经二分频后形成的时钟脉冲信号的周期。
- 机器周期是完成一个基本操作所需的时间。一个机器周期包含 6 个时钟周期，也就等于 12 个振荡周期。
- 指令周期是指 CPU 执行一条指令所需要的时间，是时序中的最大时间单位。由于单片机执行不同指令所需的时间不同，因此不同指令所包含的机器周期数也不相同，一个指令周期通常含有 1～4 个机器周期。

振荡周期、时钟周期、机器周期和指令周期的关系如图 1.4 所示。

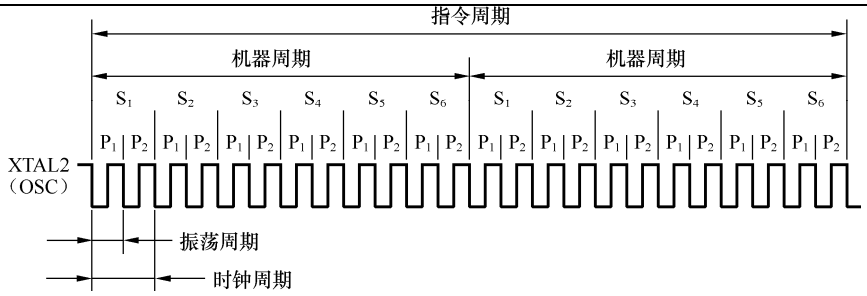


图 1.4　8051 单片机各种周期之间的关系

若 MCS-51 单片机外接晶振为 12MHz 时，则单片机的 4 个周期的具体值为：

振荡周期=1/12μs=0.0833μs；

时钟周期=1/6μs=0.167μs；

机器周期=1μs；

指令周期=1～4μs。

3．专用寄存器组

专用寄存器组主要用来指示当前要执行的内存地址、存放操作数和指示指令执行后的状态等，是任何一台计算机的 CPU 不可或缺的组成部件。8051 的专用寄存器组主要包括程序计数器（PC）、累加器（ACC）、通用寄存器（B）、程序状态字（PSW）、堆栈指示器（SP）和数据指针（DPTR）等，下面分别对这些寄存器进行介绍。

（1）累加器（ACC）

累加器是最常用的特殊功能寄存器，是一个二进制 8 位寄存器，大部分单操作数指令的操作数取自累加器，很多双操作数指令的一个操作数取自累加器。加、减、乘、除算术运算指令的运算结果都存放在累加器 ACC 或 A、B 寄存器中。指令系统中用 A 或 ACC 作为累加器的助记符。例如“3+5”加法程序如下所示：

```
MOV A,#03H   ;A←3
ADD A,#05H   ;A←A+05H
```

指令“MOV　A,#03H”把加数 3 预先送到累加器 A，为指令“ADD　A,#05H”的执行做准备，因此，指令“ADD　A,#05H”执行前累加器 A 中为加数 3，在执行后变为两数之和，即为 8。

（2）通用寄存器（B）

B 寄存器是乘除法指令中常用的寄存器。乘法指令的两个操作数分别取自 A 和 B，其结果存放在 B（高 8 位）、A（低 8 位）寄存器中。除法指令中，被除数取自 A，除数取自 B，商数存放于 A，余数存放于 B。在其他指令中，B 寄存器可作为 RAM 中的一个单元来使用。下面以乘法运算为例加以说明：

```
MOV A,#03H   ; A←3
MOV B,#05H   ; B←5
MUL AB       ; BA←A*B=5*3
```

前面两条是传送指令，是进行乘法前的准备指令，乘法指令执行前累加器 A 和通用寄存器 B 中分别存放了两个乘数，乘法指令执行后，积的高 8 位自动存放在 B 中，低 8 位自动存放在 A 中。

（3）程序状态字（PSW）

程序状态字是一个 8 位寄存器，包含了程序的状态信息，寄存器各位的含义如图 1.5 所示：

PSW7	PSW6	PSW5	PSW4	PSW3	PSW2	PSW1	PSW0
CY	AC	F0	RS1	RS0	OV	—	P

图 1.5　程序状态字各位的含义

其中 PSW1 未用，其他各位说明如下。

- CY（Carry）：进位标志。在执行某些算术和逻辑指令时，可以被硬件或软件置位或清零。具体地说：在加法运算时，若累加器 A 中最高位 A.7 有进位，则 CY=1，否则 CY=0；在减法运算时，若 A.7 有借位，则 CY=1，否则 CY=0；CPU 在进行移位操作时也会影响这个标志位。在布尔处理机中被认为是位累加器，其重要性相当于一般中央处理器中的累加器 A。
- AC（AuxiliaryCarry）：辅助进位标志。当进行加法或减法操作而产生由低 4 位数（BCD 码一位）向高 4 位数进位或借位时，AC 将被硬件置位，否则就被清零。AC 被用于 BCD 码调整。
- F0（Flag zero）：用户标志位。F0 是用户定义的一个状态标志，用软件来使其置位或清零。该标志状态一经设定，可由软件测试 F0，以控制程序的流向。
- RS1、RS0（Registers Selection）：寄存器区选择控制位。8051 共有 8 个 8 位工作寄存器，分别命名为 R0～R7。工作寄存器 R0～R7 常常被用来进行程序设计，但其在 RAM 中的实际物理地址是可以根据需要选定的。RS1 和 RS0 就是为了这个目的提供给用户使用的，用户通过改变 RS1 和 RS0 的状态可以方便地决定 R0～R7 的实际物理地址。可以用软件来置位或清零以确定工作寄存器区。RS1 和 RS0 与寄存器区的对应关系如表 1-1 所示。
- OV（Overflow）：溢出标志。可以指示运算过程中是否发生了溢出，当执行算术指令时由硬件置位或清零，溢出标志常用于作加减运算时。OV=1 表示加减运算的结果超出了目的寄存器 A 所能表示的带符号数（2 的补码）的范围（−128～+127）。当执行加法指令 ADD 时，位 6 向位 7 有进位而位 7 不向 CY 进位，或位 6 不向位 7 进位而位 7 向 CY 进位时，溢出标志 OV 置位，否则清零。

无符号数乘法指令的执行结果也会影响溢出标志：若置于累加器 A 和寄存器 B 的两个数的乘积超过 255 时，OV=1，否则 OV=0。此积的高 8 位放在 B 内，低 8 位放在 A 内。因此，OV=0 意味着只要从 A 中取得乘积即可，否则要从 B、A 寄存器对中取得乘积。除法指令也会影响溢出标志：当除数为 0 时，OV=1，否则 OV=0。

表 1-1　　通过 RS1 和 RS0 选择工作寄存器组

RS1、RS0	R0～R7 的组号	R0～R7 的物理地址
00	0	00H～07H
01	1	08H～0FH
10	2	10H～17H
11	3	18H～1FH

- P（Parity）：奇偶标志。每个指令周期都由硬件来置位或清“0”，以表示累加器 A 中值为 1 的位数的奇偶。若 1 的位数为奇数，P 置“1”，否则 P 清零。

P 标志位对串行通信中的数据传输有重要的意义，在串行通信中常用奇偶校验的办法来检验数据传输的可靠性。在发送端可根据 P 的值对数据的奇偶置位或清零。通信协议中规定采用奇偶校验的办法，则 P=0 时，应对数据（假定由 A 取得）的奇偶位置位，否则就清零。

（4）堆栈指针（SP）

堆栈指针 SP 是一个 8 位特殊功能寄存器，其作用为指示堆栈顶部在内部 RAM 中的位置。系统复位后，SP 初始化为 07H，使得堆栈事实上由 08H 单元开始。考虑到 08H～1FH 单元分属于工作寄存器区 1～3，若程序设计中要用到这些区，则最好把 SP 值改置为 1FH 或更大的值，SP 的初始值越小，堆栈深度就可以越深。堆栈指针的值可以由软件改变，因此堆栈在内部 RAM 中的位置比较灵活。

（5）数据指针（DPTR）

数据指针 DPTR 是一个 16 位特殊功能寄存器，其高位字节寄存器用 DPH 表示，低位字节寄存器用 DPL 表示，既可以作为一个 16 位寄存器 DPTR 来处理，也可以作为两个独立的 8 位寄存器 DPH 和 DPL 来处理。DPTR 主要用来存放 16 位地址，当对 64KB 外部存储器寻址时，可作为间址寄存器用。

（6）程序计数器（PC）

程序计数器（PC）用来存放即将要执行的指令地址，共 16 位，可对 64KB 的程序存储器直接寻址。读取存储在外部程序存储器中的指令时，PC 内容的低 8 位经 P0 端口输出，高 8 位经 P2 端口输出。

1.1.2 存储器结构

MCS-51 单片机的存储器编址方式采用与工作寄存器、I/O 端口锁存器统一编址的方式，程序存储器和数据存储器空间是互相独立的，各有自己的寻址系统和控制信号，物理结构也不同。程序存储器为只读存储器（ROM），数据存储器为随机存储器（RAM）。

程序存储器用来存放程序和始终要保留的常数，例如所编程序经汇编后的机器码。数据存储器通常用来存放程序运行中所需要的常数和变量。例如，做加法时的加数和被加数、做乘法时的乘数和被乘数、模/数转换时实时记录的数据等。

从物理地址空间看，MCS-51 有 4 个存储器地址空间，片内程序存储器、片外程序存储器、片内数据存储器和片外数据存储器，其组织结构如图 1.6 所示。

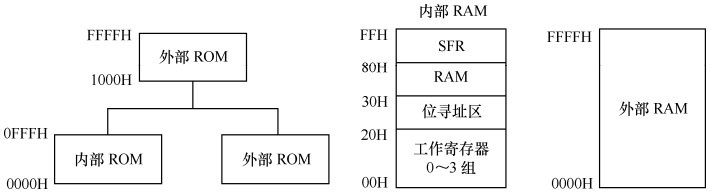


图 1.6　8051 存储器组织结构

1．程序存储器

程序存储器用来存放程序和表格常数。程序存储器以程序计数器 PC 作地址指针，通过 16 位地址总线，可寻址的地址空间为 64KB，片内、片外统一编址。

在 8051/8751 片内带有 4KB 的 ROM/EPROM 程序存储器(内部程序存储器)，4KB 可存储约两千多条指令，对于一个小型的单片机控制系统来说就足够了，不必另加程序存储器，若空间不够，还可选 8KB 或 16KB 内存的单片机芯片（例如 89C52 等），总之，尽量不要扩展外部程序存储器，这会增加成本、增大产品体积。

若开发的单片机系统较复杂，片内程序存储器存储空间不够用时，可外扩展程序存储器，具体扩展多大的芯片由两个条件决定：一是看程序容量大小；二是看扩展芯片容量大小，64KB 总容量减去 4KB 即为外部能扩展的最大容量。

对 8051/8751 而言，外部程序存储器地址空间为 1000H～FFFFH。对这类单片机，若将 $\overline{EA}$ 接低电平，可用于调试程序，即把要调试的程序放在与内部 ROM 空间重叠的外部程序存储器内，进行调试和修改。调试好后再分段存储，将 $\overline{EA}$ 接高电平，即可运行整个程序。

在程序存储器中有些特殊的单元在使用中应加以注意。其中一组特殊单元是 0000H～0002H 单元，系统复位后，PC 为 0000H，单片机从 0000H 单元开始执行程序，该单元是系统执行程序的起始地址，通常在该单元中存放一条跳转指令，而用户程序从跳转地址开始存放程序。

另一组特殊单元是 0003H～002AH，这 40 个单元被均匀地分为 5 段，其定义如下。

- 0003H～000AH：外部中断 0 的中断地址区。
- 000BH～0012H：定时器/计数器 0 的中断地址区。
- 0013H～001AH：外部中断 1 的中断地址区。
- 001BH～0022H：定时器/计数器 1 的中断地址区。
- 0023H～002AH：串行中断地址区。

可见以上的 40 个单元是专门用于存放中断处理程序的地址单元，中断响应后，按中断的类型自动转到各自的中断区去执行程序。从上面可以看出，每个中断服务程序只有 8 个字节单元，用 8 个字节来存放一个中断服务程序显然是不可能的。通常情况下是在中断响应的地址区存放一条无条件转移指令，指向程序存储器的真正存放中断服务程序的空间去执行。这样中断响应后，CPU 读到这条转移指令，便转向其他地方去继续执行中断服务程序。

2．数据存储器

MCS-51 单片机的数据存储器无论在物理上或逻辑上都分为两个地址空间，一个为内部数据存储器，访问内部数据存储器用 MOV 指令；另一个为外部数据存储器，访问外部数据存储器用 MOVX 指令。

MCS-51 系列单片机各芯片内部都有数据存储器，是最灵活的地址空间，分成物理上独立的且性质不同的几个区，如图 1.6 所示。8051 内部有 128 个 8 位用户数据存储单元和 128 个专用寄存器单元，这些单元是统一编址的，专用寄存器只能用于存放控制指令数据。所以，用户能使用的 RAM 只有 00H～7FH（0～127）单元组成的 128 字节地址空间，可存放读写的数据或运算的中间结果；80H～FFH（128～255）单元组成的高 128 字节地址空间的特殊功能寄存器（又称 SFR）区只能访问，而不能用于存放用户数据。

注意　8032/8052 单片机将 80H ~ FFH（128 ~ 255）单元组成的高 128 字节地址空间作为 RAM 区。

片内 RAM 的低 128 字节（00H～7FH）还可以分成工作寄存器区、可位寻址区和一般 RAM 区 3 个区域，其功能、特点如下。

- 工作寄存器区：在 00H～1FH 安排了 4 组工作寄存器，每组占用 8 个 RAM 字节，记为 R0～R7。在某个时刻，CPU 只能使用其中的一组工作寄存器，工作寄存器的选择由程序状态字（PSW）中的两位来确定。
- 可位寻址区：占用 20H～2FH 共 16 个字节，从 20H 单元的第 0 位起到 2FH 单元的第 7 位至共 128 位，用位地址 00H～7FH 分别与之对应。这个区域除了可以作为一般的 RAM 单元按字节读写外，还可以对每个字节的每一位进行操作，一般存放需要按位操作的数据。
- 一般 RAM 区：地址为 30H～7FH，共 80 个字节，可作为一般用途的 RAM，如存放程序变量等。

特殊功能寄存器中只有一部分是定义了的，对其他没有定义的地址进行操作会导致不确定的结果。8051 内部特殊功能寄存器符号及地址如表 1-2 所示，其中带“*”号的特殊功能寄存器都是可以位寻址的，并可以用“寄存器名.位”来表示，如 ACC.0、B.7 等。

MCS-51 具有扩展 64KB 外部数据存储器的能力，这对很多应用领域已足够使用，对外部数据存储器的访问采用 MOVX 指令，用间接寻址方式，R0、R1 和 DPTR 都可用做间址寄存器。

表 1-2　　8051 特殊功能寄存器一览表

符　号	物 理 地 址	名　称
*ACC	E0H	累加器
*B	F0H	通用寄存器
*PSW	D0H	程序状态字
SP	81H	堆栈指针
DPL	82H	数据存储器指针低 8 位
DPH	83H	数据存储器指针高 8 位
*P0	80H	通道 0
*P1	90H	通道 1
*P2	A0H	通道 2
*P3	B0H	通道 3
*IP	D8H	中断优先级控制器
*IE	A8H	中断允许控制器
TMOD	89H	定时器方式选择
*TCON	88H	定时器控制
TH0	8CH	定时器 0 高 8 位
TL0	8AH	定时器 0 低 8 位

TH1	8DH	定时器 1 高 8 位
TL1	8BH	定时器 1 低 8 位
*SCON	98H	串行口控制器
SBUF	99H	串行数据缓冲器
PCON	87H	电源控制及波特率选择

1.1.3 定时器/计数器

8051 有两个 16 位的可编程定时器/计数器 T0 和 T1 以实现定时或计数产生中断用于控制程序转向，各由两个独立的 8 位寄存器组成。T0 由两个 8 位寄存器 TH0 和 TL0 拼装而成，其中 TH0 为高 8 位，TL0 为低 8 位。和 T0 类同，T1 也由 TH1 和 TL1 拼装而成，其中 TH1 为高 8 位，TL1 为低 8 位。TH0、TL0、TH1 和 TL1 均为 SFR 中的一个，用户可以通过指令对它们存取数据。因此，T0 和 T1 的最大计数模值为 $2^{16}-1$，即需要 65 535 个脉冲才能将其从全“0”变为全“1”。

T0 和 T1 有定时器和计数器两种工作模式，在每种模式下又分为若干工作方式。在定时器模式下，T0 和 T1 的计数脉冲可以由单片机时钟脉冲经 12 分频后提供，故定时时间和单片机时钟频率有关。在计数器模式下，T0 和 T1 的计数脉冲可以从 P3.4 和 P3.5 引脚上输入。

对 T0 和 T1 的控制由两个 8 位特殊功能寄存器完成：一个称为定时器方式选择寄存器 TMOD，用于确定是定时器工作模式还是计数器工作模式；另一个称为定时器控制寄存器 TCON，用于决定定时器或计数器的启动、停止以及进行中断控制。

第 10 章中还将对定时器/计数器的使用作进一步的介绍。

1.1.4 并行端口

I/O 端口也叫做 I/O 通道或 I/O 通路，是 MCS-51 单片机对外部实现控制和信息交换的必经之路。I/O 端口有串行和并行之分，串行 I/O 端口一次能传送一位二进制信息，并行 I/O 端口一次能传送一组二进制信息。MCS-51 单片机有 P0、P1、P2、P3 等 4 个 8 位双向 I/O 端口，每一条 I/O 线都能独立地用作输入和输出，其内部结构和功能如下。

- P0 端口：P0 端口的位结构如图 1.7 所示。电路中包含一个数据输出锁存器和两个三态数据输入缓冲器，另外还有一个数据输出的驱动和控制电路。这两组口线用来作为 CPU 与外部数据存储器、外部程序存储器和 I/O 扩展口的总线接口，而不能像 P1、P3 那样直接用作输出端口。该 8 位都为漏级开路输出，每个引脚可以驱动 8 个 LS 型 TTL 负载且内部没有上拉电阻，执行输出功能时外部必须接上拉电阻（10kΩ即可）；若要执行输入功能，必须先输出高电平方能读取该端口所连接的外部数据；若系统连接外部存储器，则 P0 可作为地址总线（A0～A7）及数据总线（D0～D7）。

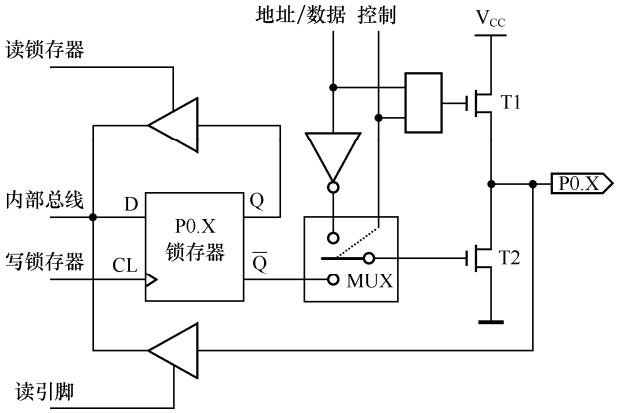


图 1.7 P0 端口的位结构

• P1 端口：P1 端口的位结构如图 1.8 所示。P1 端口为 8 位准双向端口，每一位均可单独定义为输入或输出端口，当作为输出端口，1 写入锁存器时，$\overline{Q}$=0，T2 截止，内上拉电阻将电位拉至“1”，此时该端口输出为 1；当 0 写入锁存器时，$\overline{Q}$=1，T2 导通，输出为 0。作为输入端口时，锁存器置 1，$\overline{Q}$=0，T2 截止，此时该位既可以由外部电路拉成低电平，也可由内部上拉电阻拉成高电平。需要说明的是，作为输入端口使用时有两种情况：

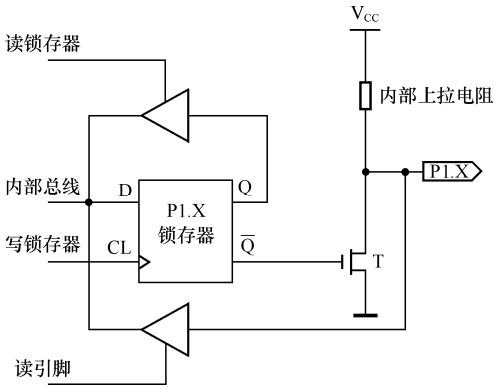


图 1.8　P1 端口的位结构

★首先读锁存器的内容，进行处理后再写到锁存器中，这种操作即读—修改—写操作，如 JBC（逻辑判断）、CPL（取反）、INC（递增）、DEC（递减）、ANL（与逻辑）和 ORL（逻辑或）等指令均属于这类操作；
★读 P1 端口线状态时打开三态门 G2 将外部状态读入 CPU。

• P2 端口：P2 端口的位结构如图 1.9 所示，电路结构与 P0 端口相似，但内部有 30kΩ上拉电阻，执行输出功能时不必连接外部上拉电阻；每个引脚可以驱动 4 个 LS 型 TTL 负载；若要执行输入功能，必须先输出高电平方能读取该端口所连接的外部数据；若系统连接外部存储器的地址线超过 8 条时，则 P2 端口可作为地址总线（A15～A8）引脚。

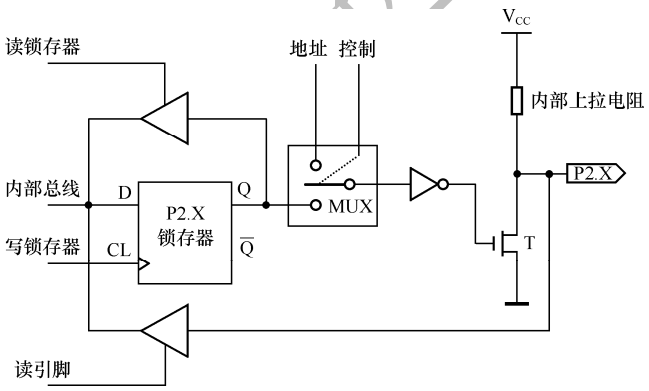


图 1.9　P2 端口的位结构

• P3 口：P3 端口的位结构如图 1.10 所示，内部有 30kΩ上拉电阻，执行输出功能时不必连接外部上拉电阻；该 8 位都为漏级开路输出，每个引脚可以驱动 4 个 LS 型 TTL 负载；若要执行输入功能，必须先输出高电平，方能读取该端口所连接的外部数据。

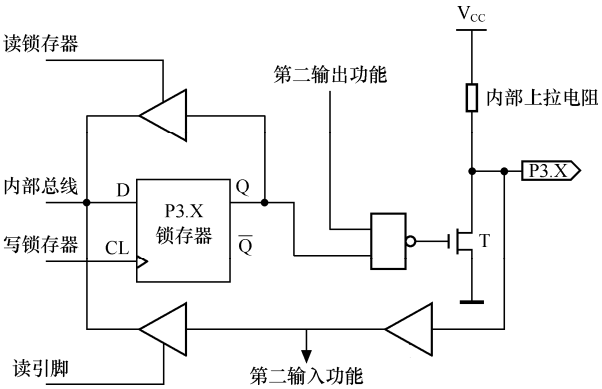


图 1.10　P3 端口的位结构

在实际应用中，P3 端口的第二功能更为重要，表 1-3 列出了 P3 端口的各位的第二功能。

表 1-3　　P3 端口各位的第二功能

P3 端口的位	第 二 功 能	注　　释
P3.0	RXD	串行数据接收口
P3.1	TXD	串行数据发送口
P3.2	INT0	外部中断 0 输入
P3.3	INT1	外部中断 1 输入
P3.4	T0	计数器 0 计数输入
P3.5	T1	计数器 1 计数输入
P3.6	WR	外部存储器写选通信号
P3.7	RD	外部存储器读选通信号

每个 I/O 端口内部都有一个 8 位数据输出锁存器和一个 8 位数据输入缓冲器，4 个数据输出锁存器与端口号 P0、P1、P2 和 P3 同名，都是特殊功能寄存器。因此，CPU 数据从并行 I/O 端口输出时可以得到锁存，数据输入时可以得到缓冲。

4 个并行 I/O 端口作为通用 I/O 端口使用时，共有写端口、读端口和读引脚 3 种操作方式。写端口实际上就是输出数据，是将累加器 A 或其他寄存器中数据传送到端口锁存器中，然后由端口自动从端口引脚线上输出；读端口不是真正的从外部输入数据，而是将端口锁存器中输出数据读到 CPU 的累加器；读引脚才是真正的输入外部数据的操作，是从端口引脚线上读入外部的输入数据。

1.1.5　串行端口

8051 有一个全双工的可编程串行 I/O 端口。这个串行 I/O 端口既可以在程序控制下将 CPU 的 8 位并行数据变成串行数据一位一位地从发送数据线 TXD 发送出去，也可以把串行接收到的数据变成 8 位并行数据送给 CPU，而且这种串行发送和串行接收可以单独进行，也可以同时进行。

8051 串行发送和串行接收利用了 P3 端口的第二功能，即利用 P3.1 引脚作为串行数据的发送线 TXD，利用 P3.0 引脚作为串行数据的接收线 RXD，如表 1-3 所示。串行 I/O 端口的电路结构还包括串行口控制器 SCON、电源及波特率选择寄存器 PCON 和串行数据缓存器 SBUF 等，这些寄存器都属于特殊功能寄存器（SFR）。

其中 PCON 和 SCON 用于设置串行口工作方式，确定数据的发送和接收波特率，串行数据缓冲器 SBUF 用于存放欲发送或已接收的数据。SBUF 实际上由两个相互独立的发送缓冲器和接收缓冲器组成，当要发送的数据传送到 SBUF 时，进的是发送缓冲器；当要从 SBUF 读数据时，则取自接收缓冲器，取走的是刚接收到的数据。

1.1.6　中断系统

8051 的中断系统可以接受 5 个独立的中断源的中断请求，这 5 个中断源即 2 个外部中断、2 个定时器/计数器中断和 1 个串行口中断。

外部中断源产生的中断请求信号可以从 P3.2 和 P3.3 引脚上输入，有电平或边沿两种触发方式；内部中断源 T0 和 T1 的两个中断是在其从全“1”变为全“0”溢出时自动向中断系统提出的；内部串行口中断源的中断请求是串行口每发送完一个 8 位二进制数据或接收到一组输入数据（8 位）后自动向中断系统提出的。

8051 的中断系统主要由 IE（Interrupt Enable，中断允许）控制器和中断优先级控制器 IP 等电路组成。其中，IE 用于控制 5 个中断源中哪些中断请求被允许向 CPU 提出，哪些中断源的中断请求被禁止，IP 用于控制 5 个中断源的中断请求的优先级。

1.1.7 总线

MCS-51 单片机属总线型结构，其总线通常分为地址总线、数据总线和控制总线等 3 种，其功能如下。

- 地址总线（AB）：地址总线宽度为 16 位，由 P0 端口经地址锁存器提供低 8 位地址（A0～A7）；P2 端口直接提供高 8 位地址（A8～A15）。地址信号是由 CPU 发出的单方向信号。
- 数据总线（DB）：数据总线宽度为 8 位，用于传送数据和指令，由 P0 端口提供。
- 控制总线（CB）：随时掌握各种部件的状态，并根据需要向有关部件发出命令。

在访问外部存储器时，P2 端口输出高 8 位地址，P0 端口输出低 8 位地址，由 ALE（地址锁存允许）信号将 P0 端口（地址/数据总线）上的低 8 位锁存到外部地址锁存器中，从而为 P0 端口接收数据做准备。在访问外部程序存储器指令时，$\overline{PSEN}$（外部程序存储器选通）信号有效；在访问外部数据存储器指令时，由 P3 端口自动产生读/写（$\overline{RD}$ / $\overline{WR}$）信号，通过 P0 端口对外部数据存储器单元进行读/写操作。

1.1.8 8051 的芯片引脚

在 MCS-51 系列单片机中，各个型号的单片机是相互兼容的，只是引脚功能略有差异。双列直插式封装的 8051 芯片引脚如图 1.11 所示。

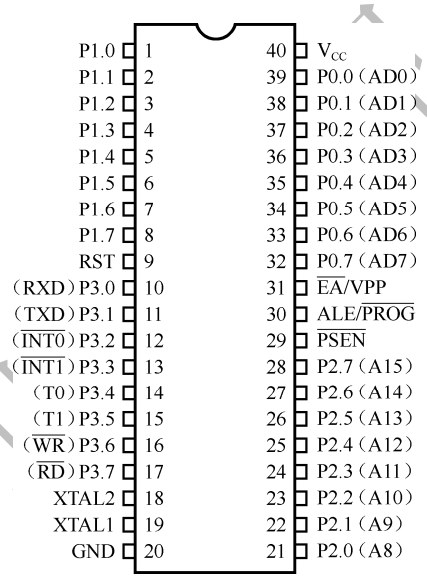


图 1.11 8051 引脚排列图

40 条引脚共分为电源线、端口线和控制与其他电源复用线 3 类，具体的功能说明如下。

1．电源线

V_{CC} 为+5V 电源线，GND 为接地线。

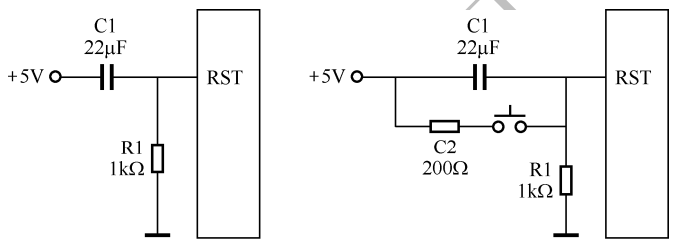
2．端口线

- P0 端口：P0 端口是开漏双向口，可以写为“1”使其状态为悬浮，用作高阻输入。P0 端口也可以在访问外部程序存储器时作地址的低字节，在访问外部数据存储器时作数据总线，此时通过内部强上拉传送“1”。
- P1 端口：P1 端口是带内部上拉的双向 I/O 端口。向 P1 端口写入“1”时，P1 端口被内部上拉为高电平，可用作输入端口。当作为输出脚时，被外部拉低的 P1 端口会因为内部上拉而输出电流。

- P2 端口：P2 端口是带内部上拉的双向 I/O 端口。向 P2 端口写入“1”时，P2 端口被内部上拉为高电平，可用作输入端口。当作为输出脚时，被外部拉低的 P2 端口会因为内部上拉而输出电流。在访问外部程序存储器和外部数据时分别作为地址高位字节和 16 位地址（MOVX @DPTR），此时通过内部强上拉传送“1”。当使用 8 位寻址方式（MOV @Ri）访问外部数据存储器时，P2 端口发送 P2 特殊功能寄存器的内容。
- P3 端口：P3 端口是带内部上拉的双向 I/O 端口。向 P3 端口写入 1 时，P3 端口被内部上拉为高电平，可用作输入端口。当作为输出脚时，被外部拉低的 P3 端口会因为内部上拉而输出电流。P3 端口的引脚更多使用其第二功能，如表 1-3 所示。

3．控制与其他电源复用线

- XTAL1 和 XTAL2：片内振荡电路输入线。这两个引脚用来外接石英晶体和微调电容，即用来连接 8051 片内 OSC 的定时反馈回路，其中 XTAL1 接晶振和内部时钟输入，XTAL2 接晶振输出；也可以将 XTAL1 接地，XTAL2 接外部振荡信号以提供 MCS-51 所需的时钟。具体的电路如图 1.2 和图 1.3 所示。
- RST/VPD：当振荡器运行时，在此引脚上出现两个机器周期的高电平（由低到高跳变），将使单片机复位。RST/VPD 的第二功能是作为备用电源输入端。当主电源 V_{CC} 发生故障而降低到规定的低电平时，RST/VPD 线上的备用电源自动投入使用，以保证片内 RAM 中的信息不会丢失。通常，8051 的复位有自动上电复位和人工按键复位两种，电路分别如图 1.12（a）和图 1.12（b）所示。



（a）上电复位电路　　（b）按键复位电路

图 1.12　MCS-51 的复位电路

- ALE/PROG：地址锁存/编程线，配合 P0 端口的第二功能使用。在访问外部存储器时，输出脉冲锁存地址的低字节。在正常情况下，ALE 输出信号恒定为 1/6 振荡频率并可用作外部时钟或定时。注意每次访问外部数据时一个 ALE 脉冲将被忽略。
- PSEN：程序存储使能，读外部程序存储。当从外部读取程序时，PSEN 每个机器周期被激活两次，PSEN 只对外部程序存储器有控制作用，对于外部数据存储器和内部程序存储器均不起作用。
- EA/VPP：外部寻址使能/编程电压。在访问整个外部程序存储器时 EA 必须外部置低。

1.2　MCS-51 单片机的指令系统

单片机应用系统完成任何一项人们预先为其规定的任务，都是靠运行程序来实现的。程序是完成任务全过程的具体描述，是由一条条单片机能够识别的指令组成的。一台单片机所具有的所有指令的集合，就构成了指令系统。

1.2.1　8051 的指令格式

汇编指令是指令系统最基本的书写方式，由助记符、目的操作数、源操作数组成。格式如图 1.13 所示：

标号:	操作码	操作数	注释

图 1.13　汇编指令的一般格式

标号（LABEL）是程序员根据编程需要给指令设定的符号地址，标记该行指令在程序中的位置，可有可无。标号可以是以英文字母开头的字母、数字和某些特殊符号的序列，由 1～8 个字符组成。标号后必须用冒号。某条指令一旦赋予标号，则在其他指令的操作数中即可引用该标号作为地址。
操作码（OP CODE）用来表达指令的操作功能。如 MOV 表示数据传送操作，ADD 表示加法操作等。
操作数（OPERAND）是操作码所要处理的数据，也是可有可无的。注释是对指令的解释说明，用以提高程序的可读性，方便阅读与修改。注释前必须加分号。

1.2.2 8051 的寻址方式

操作数是指令的重要组成部分，指出了参与操作的数据或数据的地址。寻找操作数地址的方式称为寻址方式。一条指令采用什么样的寻址方式是由指令的功能决定的。寻址方式越多指令功能就越强。
MCS-51 单片机有寄存器寻址、直接寻址、立即寻址、间接寻址、基址变址寻址、相对寻址和位寻址等 7 种寻址方式，下面对这 7 种寻址方式进行详细介绍。

1．寄存器寻址方式

8051 内部的 RAM 的每个工作寄存器组均含有 8 个寄存器，称为 R0～R7，操作数使用 R0～R7 或 A、B、DPTR 的指令寻址操作都称为寄存器寻址方式。例如：

```
MOV     A,R0;A ←（R0）
CLR     A        ;A←0
```

指令“MOV A,R0”的作用是把 R0 寄存器中的数据传送到累加器 A 中，其操作数存放在寄存器 R0 中，所以为寄存器寻址。设寄存器 R0 中的内容为 20H，则执行指令后，累加器 A 中的值就变为 20H。
指令“CLR A”的作用是把通用寄存器 A 中的内容清空，其操作数为 A，所以也是寄存器寻址。执行指令后，A 中的值为 0。

2．直接寻址方式

指令中直接给出操作数地址的寻址方式，能进行直接寻址的存储空间有 SFR 寄存器和片内 RAM 的 128 个单元。例如：

```
MOV     A,P1; A←（P1）
MOV     A,33H    ; A←（33H）
```

指令“MOV A,P1”的作用是把 SFR 中 P1 端口的内容送 A；指令“MOV A,33H”的作用是把 RAM 中地址为 33H 的存储单元的内容送 A。

3．立即寻址方式

指令中直接给出操作数的寻址方式。立即操作数用前面加有“#”的 8 位或 16 位数来表示，以区别于直接寻址。立即数在寻址操作中只能作源操作数，其寻址空间为程序存储器 ROM。例如：

```
MOV     A,#30H           ;A←30H
MOV     DPTR,#1234H      ;DPTR←1234H
MOV     33H,#40H     ;33H 单元←40H
```

上述 3 条指令执行完后，累加器 A 中数据为立即数 30H，DPTR 寄存器中数据为 1234H，33H 单元中数据为立即数 40H。

4．间接寻址方式

寄存器间接寻址是指将存放操作数的内存单元的地址放在寄存器中，指令中只给出该寄存器。执行指令时，首先根据寄存器的内容找到所需要的操作数地址，再由该地址找到操作数并完成相应操作，例如：

```
MOV     R0,#30H
MOV     A,@R0
MOV     DPTR,#1234H
MOVX    A,@DPTR
```

这里的指令“MOV A,@R0”和“MOVX A,@DPTR”都是寄存器间接寻址,其作用分别是将内部数据区 30H 单元里的数据和外部数据区 1234H 单元里的数据送入累加器 A 中。

假设 30H 单元中的数据是 23H，1234H 单元中的数据是 90H，则这两条指令的执行过程及存储单元的内容分别如图 1.14（a）和 1.14（b）所示：

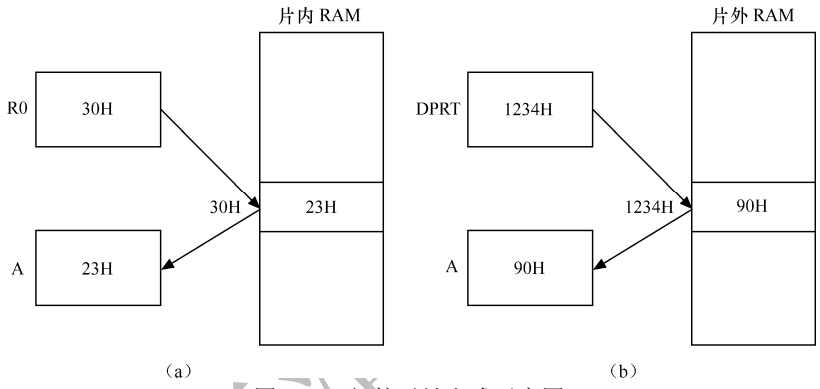


图 1.14 间接寻址方式示意图

间接寻址的地址空间有片内数据存储器的 00H～7FH 和片外数据存储器的 0000H～0FFFFH。间接寻址的寄存器有 R0、R1 和 DPTR，间接寻址时要在间接寻址寄存器前面加上符号“@”。如上面的“MOV A,@R”和“MOVX A,@DPTR”指令。

5．基址变址寻址方式

基址变址寻址是将基址寄存器（A）与变址寄存器（PC 或 DPTR）的内容相加，结果作为操作数的地址的寻址方式。变址寻址只能对程序存储器中数据进行操作，由于程序存储器是只读的，因此变址寻址只有读操作而无写操作，在指令符号上采用 MOVC 的形式，这种寻址方式多用于查表操作。具有此种寻址方式操作的指令共有 3 条：

```
MOVCA,@A+DPTR
MOVCA,@A+PC
JMP     @A+DPTR
```

指令“MOVC A,@A+DPTR”的作用是将累加器 A 和变址寄存器 DPTR 的内容相加，相加结果作为操作数存放的地址，按此地址再将操作数取出来送到累加器 A 中；指令“MOVC A,@A+PC”与第一条指令相比只是将变址寄存器换作了 PC，其他操作与第一条指令相同；指令“JMP @A+DPTR”使得程序跳转到程序存储器中地址为“A+DPTR”的地方继续执行。

指令“MOVC A,@A+DPTR”的示意图如图 1.15 所示，假设执行指令前 A 的内容为 10H，DPTR 的值为 2000H，则执行指令后 A 中存放了程序存储器中地址为 2010H 的存储单元的值，即为 64H。

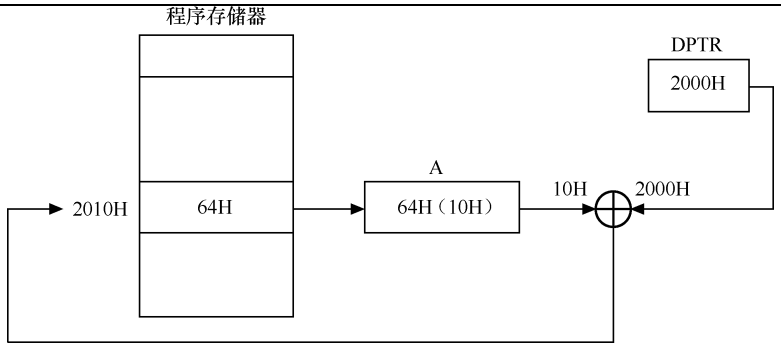


图 1.15　基址变址寻址方式

6．相对寻址方式

相对寻址方式是以当前程序计数器 PC 的内容为基础，加上指令给出的一字节补码数（偏移量）形成新的 PC 值的寻址方式。相对寻址用于修改 PC 值，主要用于实现程序的分支转移。例如：

```
SJMP08H ;PC←PC+2+08
```

指令操作示意图如图 1.16 所示：

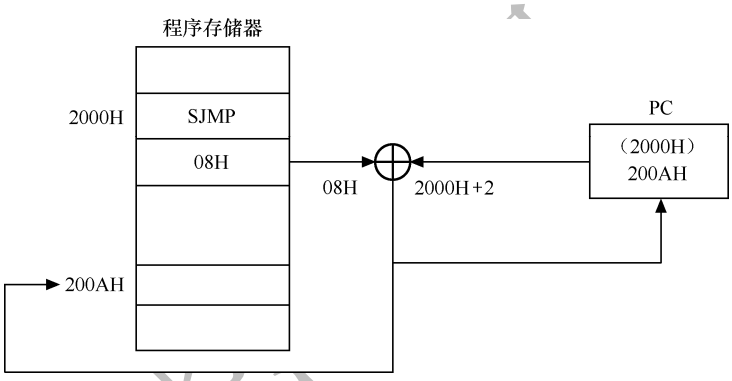


图 1.16　相对寻址方式

需要说明的是，在通常的汇编语言编程中，操作数并不是以具体的值 08H 给出来，而是以一个符号地址给出，这个符号地址就是要转移到的那条要执行的指令的标号。标号的具体值在进行汇编语言程序的汇编过程中由计算机自动算出来并取代在程序中给出的那个符号地址。

另外，使用中应注意操作数的范围不要超出−128～+127，即以转移指令下一条指令第一个指令代码的地址为基准，回跳 128 个字节单元或者下跳 127 个字节单元。

7．位寻址方式

位寻址是按位进行的寻址操作，而上述介绍的指令都是按字节进行的寻址操作。在 MCS-51 系列单片机中，操作数不仅可以按字节为单位进行操作，也可以按位进行操作。

当把某一位作为操作数时，这个操作数的地址称为位地址。位寻址区包括专门安排在内部 RAM 中的两个区域：一是内部 RAM 的位寻址区，地址范围是 20～2FH，共 16 个 RAM 单元，位地址为 00H～7FH，共 128 个位地址；二是特殊功能寄存器 SFR，有 11 个寄存器，可以位寻址，如表 1-2 所示。例如：

```
CLR     00H
ANL     C,5FH
```

在指令“ANL C,5FH”中，进位位 C 作为位操作的位累加器。指令的功能是将 5FH 中的位状态与进位位 C 相与，结果放在 C 中。

在位寻址操作中，位单元也可以使用位地址名。例如指令“SETB TR0”是将特殊功能寄存器 TCON 中的 TR0（其位地址为 8CH）置 1。

1.2.3 8051 的指令说明

8051 单片机共有 111 条指令，按功能可以分为数据传送指令（28 条）、算术运算指令（24 条）、逻辑运算指令（25 条）、位操作指令（17 条）和控制转移指令（17 条）。下面是指令中符号的含义。

Rn（n=0～7）：表示当前选中的 8 个工作寄存器。

Ri（i=0，1）：表示当前选中的可用于 8 位地址指针的两个工作寄存器。

direct：表示片内 RAM（00H～0FFH）或 SFR 区的直接地址。

#data：表示 8 位常数。

#data16：表示 16 位常数。

addrl6/ addrll：表示外部程序存储器的 16 位或 11 位地址。

rel：表示 8 位偏移地址，即相对跳转的偏移字节数。

DPTR：表示 16 位数据指针。

bit：表示直接位地址。

@：表示间接寻址寄存器的前缀。

/：表示位操作的取反前缀。

1．数据传送指令

数据传送类指令共 28 条，是将源操作数送到目的操作数。指令执行后，源操作数不变，目的操作数被源操作数取代。

数据传送类指令有 MOV、MOVC、MOVX、XCH、XCHD、PUSH、POP，共 7 种。

（1）MOV（Move）

MOV 指令执行后，源操作数不变，目的操作数被源操作数取代。按照目的操作数的不同，可以将 MOV 指令分为以下几类。

- 以 A 为目的操作数，具体的指令形式及举例如表 1-4 所示。

表 1-4　　以 A 为目的操作数的 MOV 指令

汇 编 指 令	功　　能	举例及注释
MOV　A,Rn	将 Rn 中的数送入 A 中	MOV A,R1　　;A←R1
MOV　A,direct	将直接地址 direct 中的数送入 A 中	MOV A,30H　　;A←(30H)
MOV　A,#data	将 8 位常数送入 A 中	MOV A,#40H　;A←40H
MOV　A,@Ri	将 Ri 指示的地址中的数送入 A 中	MOV A,@R1　;A←(R1)

- 以 Rn 为目的操作数，具体的指令形式及举例如表 1-5 所示。

表 1-5　　以 Rn 为目的操作数的 MOV 指令

汇 编 指 令	功　　能	举例及注释
MOV　　Rn,direct	将直接地址 direct 中的数送入 Rn 中	MOV R1,24H　　;R1←(24H)
MOV　　Rn,#data	将立即数送入 Rn 中	MOV R2,#30H　;R2←30H
MOV　　Rn,A	将 A 中的数送入 Rn 中	MOV R1,A　　;R1←A

- 以直接地址（direct）为目的操作数，具体的指令形式及举例如表 1-6 所示。

表 1-6　　以直接地址为目的操作数的 MOV 指令

汇 编 指 令	功 能	举例及注释
MOV direct,Rn	将 Rn 中的数送入 direct 中	MOV 24H,R1 ;(24H)←R1
MOV direct,A	将 A 中的数送入 direct 中	MOV 34H,A ;(34H)←A
MOV direct,@Ri	将 Ri 指示单元中的数送入 direct 中	MOV 34H,@R1 ;(34H)←(R1)
MOV direct,#data	将立即数送入 direct 中	MOV 72H,#04H ;(72H)←04H
MOV direct,direct	将一个 direct 中的数送入另一个 direct 中	MOV 34H,23H ;(34H)←(23H)

- 以间接地址（@Ri）为目的操作数，具体的指令形式及举例如表 1-7 所示。

表 1-7 以间接地址为目的操作数的 **MOV** 指令

汇 编 指 令	功 能	举例及注释
MOV @Ri,A	将 A 中的数送入 Ri 指示的地址中	MOV @R1,A ;(R1)←A
MOV @Ri,direct	将 direct 中的数送入 Ri 指示的地址中	MOV @R1,22H ;(R1)←(22H)
MOV @Ri,#data	将立即数送入 Ri 指示的地址中	MOV @R0,#16H ;(R0)←16H

- 以 DPTR 为目的操作数，这是惟一的 16 位立即数传送指令，把 16 位常数送入 DPTR。其形式及举例如下：

```
MOV DPRT,#3000H ; DPRT←3000H
```

（2）MOVC（Move Code）

MOVC 称为程序查表指令，常用 DPTR 的传送来取程序存储器中的查表数据。MOVC 指令有以下两重格式：

```
MOVC A,@A+DPTR   ;将以 DPTR 为基址、A 为偏移地址中的数送入 A 中
MOVC A,@A+PC ;将以 PC 为基址、A 为偏移地址中的数送入 A 中
```

指令“MOVC A,@A+DPTR”的查表范围可达 ROM 的 64KB 空间，指令“MOVC A,@A+PC”只能查找指令所在地址以后 256 字节范围内的常数或代码。

（3）MOVX（Move External）

在 8051 指令系统中，访问片外 RAM 只能与累加器 A 配合用寄存器间接寻址，包括“MOVX @Ri,A”、“MOVX A,@Ri”、“MOVX @DPRT,A”、“MOVX A,@DPRT”等 4 条指令，片外 RAM 指令的具体格式、功能和举例如表 1-8 所示。

表 1-8 片外 **RAM** 访问指令

汇 编 指 令	功 能	举例及注释
MOVX @Ri,A	将 A 中的数送入 Ri 指示的地址中	MOV @R1,A ;(R1)←A
MOVX A,@Ri	将 Ri 指示的地址中的数送入 A	MOV A,@R1 ;A←(R1)
MOVX @DPRT,A	将 A 中的数送入 DPRT 指示的地址中	MOV @DPRT,A ;(DPRT)←A
MOVX A,@DPRT	将 DPRT 指示的地址中的数送入 A	MOV A,@DPRT ;A←(DPRT)

（4）XCH（Exchange）

XCH 指令交换两个字节，所有的操作的目的操作数都是累加器。XCH 指令的具体格式、功能和举例如表 1-9 所示。

表 1-9 字节交换指令

汇 编 指 令	功 能	举例及注释
XCH A,Rn	A 中的数和 Rn 中的数全交换	XCH A,R1 ; A←→R1
XCH A,direct	A 中的数和 direct 中的数全交换	XCH A,30H ; A←→(30H)
XCH A,@Ri	A 中的数和 Ri 指示单元中的数全交换	XCH A,@R1 ; A←→(R1)

（5）XCHD

该指令仅仅把两个操作数的低 4 位互换，而高 4 位不变，只有下面一种形式：

```
XCHD  A,@Ri       ; A 中的数和 Ri 指示单元中的数半交换
```

（6）PUSH 与 POP

这两条指令是堆栈操作指令。所谓堆栈是在片内 RAM 中按“先进后出，后进先出”原则设置的专用存储区。数据的进栈出栈由指针 SP 统一管理。可以把任何直接地址的内容压入堆栈，包括 SFR，即可以把累加器、B、PSW 和各种硬件控制寄存器压入堆栈。不可以使用 R0～R7 的名字把其内容压入堆栈，因为改变 PSW 的两位会切换工作寄存器组。

这两条指令的格式如下：

```
PUSH  direct        ;SP←(SP+1),(SP)←(direct)
POP    direct       ;(direct)←(SP),SP←(SP-1)
```

下面通过一个例子来说明数据交换指令的具体应用。

【例 1-1】将片内 RAM 30H 单元与 40H 单元中的内容互换。

方法 1：直接地址传送法，要用到一个中间地址 31H，代码如下：

```
MOV     31H,30H     ;(31H)←(30H)
MOV     30H,40H     ;(30H)←(40H)
MOV     40H,31H     ;(40H)←(31H)
SJMP  $
```

方法 2：间接地址传送法，代码如下：

```
MOV     R0,#40H     ;R0←40H
MOV     R1,#30H     ;R1←30H
MOV     A,@R0       ;A←(40H)
MOV     B,@R1       ;B←(30H)
MOV     @R1,A       ;(30H)←A
MOV     @R0,B       ;(40H)←B
SJMP     $
```

方法 3：字节交换传送法，代码如下：

```
MOV     A,30H       ;A←(30H)
XCH     A,40H       ;A←→(40H)
MOV     30H,A       ;(30H)←A
SJMP $
```

方法 4：堆栈传送法，代码如下：

```
PUSH    30H
PUSH    40H
POP         30H
POP         40H
SJMP    $
```

2．算术运算指令

MCS-51 单片机有比较丰富的算术运算指令，可以分为加法、减法、十进制调整和乘除法 4 类。所有的算术运算都是 8 位的，除了加 1 和减 1 指令外其他的算术运算指令都把结果放到累加器中，覆盖累加器中原来的内容并影响标志位。

（1）加法指令

加法指令有 13 条，由不带 CY 加法指令、带 CY 加法指令和加 1 指令等组成，不带 CY 的加法指令有“ADD A,Rn”、“ADD A,direct”、“ADD A,#data”和“ADD A,@Ri”等 4 条。不带 CY 的加法指令的具体指令格式、功能和举例如表 1-10 所示：

表 1-10　　不带 CY 的加法指令

汇 编 指 令	功 能	举例及注释
ADD A,Rn	Rn 中的数与 A 相加，结果在 A 中	ADD A,R1 ;A←A+R1
ADD A,direct	direct 中的数与 A 相加，结果在 A 中	ADD A,30H ;A←A+(30H)
ADD A,#data	立即数与 A 相加，结果在 A 中	ADD A,#30H ; A←A+30H
ADD A,@Ri	Ri 指示单元中的数与 A 相加，结果在 A 中	ADD A,@R0 ; A←A+(R0)

带 CY 的加法指令有“ADDC A,Rn”、“ADDC A,direct”、“ADDC A,#data”和“ADDC A,@Ri”等 4 条。带 CY 的加法指令的指令格式、功能和举例如表 1-11 所示。

表 1-11 带 CY 的加法指令

汇 编 指 令	功 能	举例及注释
ADDC A,Rn	Rn 中的数与 A 带进位相加，结果在 A 中	ADDC A,R5 ;A←A+R5+CY
ADDC A,direct	direct 中的数与 A 带进位相加，结果在 A 中	ADDC A,37;A←A+(37H)+CY
ADDC A,#data	立即数与 A 带进位相加，结果在 A 中	ADDC A,#56H ; A←A+56H+CY
ADDC A,@Ri	Ri 指示单元中的数与 A 带进位相加，结果在 A 中	ADDC A,@R0;A←A+(R0)+CY

加 1 指令有“INC A”、“INC Rn”、“INC direct”、“INC @Ri”和“INC DPTR”，共 5 条。加 1 指令的指令格式、功能和举例如表 1-12 所示。

表 1-12 加 1 指令

汇 编 指 令	功 能	举例及注释
INC A	A 中的数加 1	INC A ; A←A+1
INC Rn	Rn 中的数加 1	INC R4 ; R4←R4+1
INC direct	direct 中的数加 1	INC 34H ; (34H)←(34H)+1
INC @Ri	Ri 指示单元中的数加 1	INC @R0 ; (R0)←(R0)+1
INC DPTR	dPTR 中的数加 1	INC DPTR ; DPTR←DPTR+1

使用加法指令时应注意以下 3 个问题。

- 参加运算的两个操作数必须是 8 位二进制数，操作结果也是一个二进制数。
- 用户既可以根据编程需要把参与运算的两个操作数看作是无符号数（0～255），也可以将其看作是带符号数（−128～+127）。不论把这两个参加运算的操作数看作是无符号数还是带符号数，单片机总是按照带符号数法则进行运算并产生 PSW 中的标志位。
- 若将参加运算的两个操作数看作无符号数，则应根据 CY 判断结果是否溢出；若将参加运算的两个操作数看作带符号数，则运算结果是否溢出应根据 OV 位来判断。

【例 1-2】把存放在 M1M2 和 M3M4 中的两个 16 位数相加，结果存于 M5M6 中，上述 3 个数都是高位在前。程序如下：

```
MOV     A,M2;取第一个数的低 8 位
ADD     A,M4;与第二个数的的低 8 位相加，结果存在 A 中
MOV     M6,A;保存两数和的低 8 位到 M6 中
MOV     A,M1;取第一个数的高 8 位
ADDCA,M3;两数的高 8 位相加，并把低 8 位相加时的进位位加进来
MOV     M5,A;把相加的高 8 位存入 M5 中
SJMP $        ;停机
```

（2）减法指令

减法指令有 8 条，包括带 CY 减法指令和减 1 指令两类。带 CY 的减法指令包括“SUBB A,Rn”、“SUBB A, direct”、“SUBB A, #data”、“SUBB A, @Ri”，共 4 条。带 CY 的减法指令的指令格式、功能和举例如表 1-13 所示。

表 1-13 带 CY 的减法指令

汇 编 指 令	功 能	举例及注释

SUBB A,Rn	Rn 中的数与 A 中的数带借位减，结果在 A 中	SUBB A,R5 ;A←A-R5-CY
SUBB A, direct	direct 中的数与 A 中的数带借位减，结果在 A 中	SUBB A,37;A←A-(37H)-CY
SUBB A,#data	立即数与 A 中的数带借位减，结果在 A 中	SUBB A,#56H ; A←A-56H-CY
SUBB A,@Ri	Ri 指示单元中的数与 A 中的数带借位减	SUBB A,@R0;A←A-(R0)-CY

减 1 指令包括“DEC A”、“DEC Rn”、“DEC direct”、“DEC @Ri”，共 4 条。减 1 指令的指令格式、功能和举例如表 1-14 所示。

表 1-14　　减 1 指令

汇 编 指 令	功　　能	举例及注释
DEC A	A 中的数减 1	DEC A ; A←A−1
DEC Rn	Rn 中的数减 1	DEC R4 ; R4←R4−1
DEC direct	direct 中的数减 1	DEC 34H ; (34H)←(34H) −1
DEC @Ri	Ri 指示单元中的数减 1	DEC @R0 ; (R0)←(R0) −1

在使用减法指令时应注意以下 3 个问题。

- 在 MCS-51 单片机中，没有不带 CY 的减法指令。要使用不带 CY 的减法指令，可以在合法的带 CY 的减法指令前预先用一条能够清零 CY 的指令，指令如下：

```
CLR     C           ;将 CY 位清零
```

- 在单片机内部，减法操作实际上是在控制器控制下采用补码加法来实现的。但在实际应用中，若要判断减法的操作结果，仍可按二进制减法法则进行。
- SUBB 指令影响 PSW，而 DEC 指令不影响 PSW。

【例 1-3】把存放在 M1M2（高位在前）中的一个 16 位数和存放在 M3 中的一个 8 位数相减，结果存于 M1M2 中，参考程序如下：

```
MOV     A,M2;取被减数的低 8 位
CLR     C         ;将 CY 位清零
SUBBA,M3;A←M2-M3-0
MOV     M2,A;保存两数差的低 8 位到 M2 中
MOV     A,M1;取被减数的高 8 位
SUBBA,0       ;A←M1-CY
MOV     M1,A;保存两数差的高 8 位到 M1 中
```

（3）十进制调整指令

前面提到的所有算术运算指令都是二进制数的运算指令，但是人们最常用的是十进制数，这样，当用单片机进行计算时，必须先把十进制数转换成二进制数，然后再进行二进制数的计算，计算结果又转换成十进制数输出。为了便于十进制数的计算，MCS-51 还提供了一条十进制数调整指令，这组指令在二进制计算的基础上，给予十进制调整，可以直接得到十进制的结果。此指令在实际中较少使用。指令格式如下：

```
DA      A         ;若 AC=1 或[A3A2A1A0]>9，则 A←A+06H
                  ;若 CY=1 或[A7A6A5A4]>9，则 A←A+60H
```

（4）乘除法指令

乘法和除法指令各一条，乘法指令为“MUL AB”，除法指令为“DIV AB”，其格式功能如表 1-15 所示。

表 1-15　　乘除指令

汇 编 指 令	功　　能
MUL AB	A、B 中的两无符号数相乘，结果低 8 位在 A 中，高 8 位在 B 中
DIV AB	A、B 中的两无符号数相除，商在 A 中，余数在 B 中

【例 1-4】已知两个 8 位无符号数分别放在 R1、R2 中，将这两个数相乘并把乘积的高 8 位放入 30H 单元，低 8 位放入 31H 单元。参考程序如下：

```
MOV     A,R1;A←第一个乘数
MOV     B,R2;B←第二个乘数
MUL     AB        ;BA←A×B
MOV     R0,#30H
MOV     @R0,B     ;(30H)←B
MOV     R0,#31H
MOV     @R0,A     ;(31H)←A
```

3．逻辑运算指令

逻辑运算指令共有 25 条，可以对两个 8 位二进制数进行与、或、非、异或等逻辑运算，还可以对一个字节进行移位操作。在这类指令中，除以累加器 A 为目标寄存器的指令外，其他指令均不会改变 PSW。

（1）逻辑“与”运算指令

逻辑“与”运算指令又称逻辑乘指令。若逻辑“与”的两个位都为 1，结果才为 1。逻辑“与”运算指令共有“ANL A,Rn”、“ANL A,direct”、“ANL A,#data”、“ANL A,@Ri”、“ANL direct,A”、“ANL direct,#data”，共 6 条。逻辑“与”运算指令的指令格式、功能和举例如表 1-16 所示。

表 1-16　　逻辑“与”运算指令

汇 编 指 令	功　　能	举　　例
ANL　A,Rn	Rn 中的数与 A 相“与”，结果在 A 中	ANL A,R4　;A←A∧R4
ANL　A,direct	direct 中的数与 A 相“与”，结果在 A 中	ANL A,30H　;A←A∧(30H)
ANL　A,#data	立即数与 A 相“与”，结果在 A 中	ANL A,#30H　;A←A∧30H
ANL　A,@Ri	Ri 指示单元中的数与 A 相“与”，结果在 A 中	ANL A,@R0　;A←A∧(R0)
ANL　direct,A	A 和 direct 中的数相“与”，结果在 direct 中	ANL 30H,A ;(30H)←A∧(30H)
ANL　direct,#data	立即数和 direct 中的数相“与”，结果在 direct 中	ANL 30H,#30H

（2）逻辑“或”运算指令

逻辑“或”运算指令和逻辑“与”运算指令类似，只要把表 1-16 中的指令 ANL 换作“ORL”，就构成了逻辑“或”运算的 6 条指令。只要逻辑“或”的两个位有一个为 1，结果就为 1。逻辑“或”运算指令的具体格式、功能和举例如表 1-17 所示。

表 1-17　　逻辑“或”运算指令

汇 编 指 令	功　　能	举　　例
ORL　A,Rn	Rn 中的数与 A 相“或”，结果在 A 中	ORL A,R4　;A←A∨R4
ORL　A,direct	direct 中的数与 A 相“或”，结果在 A 中	ORL A,30H　;A←A∨(30H)
ORL　A,#data	立即数与 A 相“或”，结果在 A 中	ORL A,#30H　;A←A∨30H
ORL　A,@Ri	Ri 指示单元中的数与 A 相“或”，结果在 A 中	ORL A,@R0　;A←A∨(R0)
ORL　direct,A	A 和 direct 中的数相“或”，结果在 direct 中	ORL 30H,A ;(30H)←A∨(30H)
ORL　direct,#data	立即数和 direct 中的数相“或”，结果在 direct 中	ORL 30H,#30H

（3）逻辑“异或”运算指令

逻辑“异或”运算指令和前面两类运算指令类似，只要把表 1-16 中的指令 ANL 换作“XRL”，就构成了逻辑“异或”运算的 6 条指令。参与逻辑“与”运算的的两个位互不相同，即一个为 1、一个为 0，结果才为 1。逻辑“异或”运算指令的格式、功能和举例如表 1-18 所示。

表 1-18　　逻辑“异或”运算指令

汇 编 指 令	功　　能	举　　例
XRL　A,Rn	Rn 中的数与 A 相“异或”，结果在 A 中	XRL A,R4　;A←A ⊕ R4
XRL　A,direct	direct 中的数与 A 相“异或”，结果在 A 中	XRL A,30H　;A←A ⊕ (30H)
XRL　A,#data	立即数与 A 相“异或”，结果在 A 中	XRL A,#30H　;A←A ⊕ 30H
XRL　A,@Ri	Ri 指示单元中的数与 A 相“异或”，结果在 A 中	XRL A,@R0　;A←A ⊕ (R0)
XRL　direct,A	A 和 direct 中的数相“异或”，结果在 direct 中	XRL 30H,A;(30H)←A ⊕ (30H)
XRL　direct,#data	立即数和 direct 中的数“异或”，结果在 direct 中	XRL 30H,#30H

“与”运算经常用来将一个字节的某些位清零或保留某些位而屏蔽其他位；“或”运算经常用来将一个字节的某些位置 1；“异或”运算经常用来将一个字节的某些位取反。这 3 种逻辑运算举例如下。

【例 1-5】分别将 P1 端口的第 0 位、第 1 位清零，第 2 位、第 3 位、第 4 位置 1，第 5 位、第 6 位、第 7 位取反。参考程序如下：

```
MOV     A,#0FCH      ;A←(11111100)b
ANL     P1,A     ;P1←P1∧(11111100)b
MOV     A,#1CH       ;A←(00011100)b
ORL     P1,A     ;P1←P1∨(11111100)b
MOV     A,#0E0H      ;A←(11100000)b
XRL     P1,A     ;P1←P1⊕ (11111100)b
```

（4）累加器清零和取反指令

在 MCS-51 中，专门安排了累加器清零和累加器取反指令，这两条指令皆为单字节单周期指令。虽然采用数据传送或者逻辑“异或”运算指令也可以达到同样的目的，但后两种方法至少需要两个字节。累加器清零、累加器取反这两条指令的格式如下：

```
CLR     A        ;功能上等价于 MOV  A,0
CPL     A        ;功能上等价于 XRL  A,A
```

（5）移位指令

MCS-51 虽然只有 5 条对累加器 A 中数据进行移位操作的指令，但足够用来处理所有移位问题。移位指令包括以下 5 条指令：

```
RL      A        ;循环左移
RR      A        ;循环右移
RLC     A        ;带 CY 位循环左移
RRC     A        ;带 CY 位循环右移
SWAPA            ;交换 A 的高 4 位和低 4 位
```

5 条指令共分 3 类：前两条属于不带 CY 标志位的循环移位指令，累加器 A 中最高位 A7 和最低位 A0 连接后进行左移或右移；后面两条指令为带 CY 标志位的循环左移或右移；最后一条指令称为半字节交换指令，用于累加器 A 中的高 4 位和低 4 位。这组指令功能如图 1.17 所示。

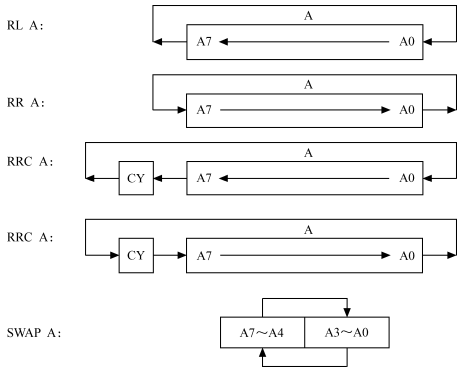


图 1.17　移位指令示意图

【例 1-6】已知 30H 和 31H 单元中存放了一个 16 位的二进制数，30H 中为高 8 位。要将此数乘以 2 后仍存放在这两个存储单元中，只需将其进行一次左移。参考程序如下：

```
CLR     C           ; CY←0
MOV     R0,#31H     ;操作数低 8 位送 R0
MOV     A,@R0       ; A←操作数低 8 位
RLC     A           ;低 8 位操作数左移，移出的位放在 CY 中
MOV     @R0,A       ;送回 31H 单元
MOV     R0,#30H     ;操作数高 8 位送 R0
MOV     A,@R0       ; A←操作数高 8 位
RLC     A           ;低 8 位操作数左移，最低位移入的是 CY
MOV     @R0,A       ;送回 30H 单元
SJMP$           ;停机
```

4．控制转移指令

控制转移指令是任何指令系统都具有的一类指令，主要以改变程序计数器 PC 中的内容为目的，以便控制程序执行流向。8051 有一些非常有效的控制转移指令，包括无条件转移指令、条件转移指令、子程序调用和返回指令、空操作指令，共 4 类。

（1）无条件转移指令

这组指令共有如下 4 条：

```
LJMP    addr16      ;PC←addr16
AJMP    addr11      ;PC←PC+2,PC10~PC0←addr11
SJMP    rel         ;PC←PC+2,PC←PC+rel
JMP         @A+DPTR     ;PC←A+DPTR
```

第一条指令称为长转移指令，其功能是把 addr16 代表的数字送入程序计数器 PC，使机器执行下条指令时无条件转移到 addr16 处执行程序。由于 addr16 是一个 16 位二进制地址（地址范围为 0000H～0FFFFH），因此长转移指令可以在 64KB 的范围内转移。为了使程序易编写，addr16 常采用符号地址（如 LOOP、LOOP1 等）表示。例如如下的代码使程序转到标号为 LOOP1 的地方执行：

```
LJMPLOOP1
```

第二条指令称为绝对转移指令。这条指令只能把 addr11 的低 11 位送入 PC 的低 11 位，PC 其余的高 5 位是把原 PC 加 1 两次后的高 5 位，因此绝对转移指令可以在 2KB 的范围内向前或向后转移。同样，addr11 也可以采用符号地址。

第三条指令称为短转移指令，其功能是先使程序计数器 PC 加 1 两次（即取出指令码），然后把加 1 两次后的地址和 rel 相加作为目标转移地址。短转移指令的实际转移范围　　为−126～+129。

第四条指令称为变址转移指令。在指令执行前，用户应预先把目标转移地址的基址送入 DPTR，目标转移地址对基地址的偏移量放在累加器 A 中。在指令执行时，MCS-51 单片机把 DPTR 中的基地址和累加器 A 中的地址偏移量相加，以形成目标转移地址送入程序计数器 PC。通常，DPTR 中的基地址是一个确定的值，常常是一张转移指令表的起始地址，累加器 A 中之值为表的偏移量地址，单片机通过变址转移指令便可实现程序的分支转移。

条件转移指令是指在满足一定条件时进行相对转移。条件指令共有 8 条，分为累加器 A 的判零转移指令、比较条件转移和减 1 条件转移，共 3 类。

（2）累加器 A 判零转移指令

这组指令执行时均要判断累加器 A 中内容是否为 0，并将其作为转移条件，共有以下两条：

```
JZ      rel     ;若 A=0，则 PC←PC+2+rel
```

```
                    ;若 A≠0，则 PC←PC+2
JNZ      rel        ;若 A≠0，则 PC←PC+2+rel
                ;若 A=0，则 PC←PC+2
```

例如下面的代码：

```
                    MOV      A,R0
                    JZ       L1
                    MOV      R1,#00H
                    AJMP     L2
            L1:
              MOV      R1,#0FFH
            L2:
                    SJMP     L2
```

在执行上面这段程序前如果 R0 中的值是 0，就转移到 L1 执行，因此最终的执行结果是 R1 中的值为 0FFH。而如果 R0 中的值不等于 0，则顺序执行，也就是执行“MOV R1,#00H”指令。最终的执行结果是 R1 中的值等于 0。

（3）比较条件转移指令

比较条件转移指令共有如下 4 条：

```
CJNE A,#data,rel      ;若 A=data，则 PC←PC+3
                      ;若 A≠data，则 PC←PC+3+rel
CJNE A,direct,rel         ;若 A=(direct)，则 PC←PC+3
                      ;若 A≠(direct)，则 PC←PC+3+rel
CJNE Rn,#data,rel         ;若 Rn=#data，则 PC←PC+3
                      ;若 Rn≠#data，则 PC←PC+3+rel
CJNE @Ri,#data,rel        ;若(Ri)=#data，则 PC←PC+3
                          ;若(Ri)≠#data，则 PC←PC+3+rel
```

指令“CJNE A,#data,rel”的功能是将 A 中的值和立即数 data 比较，如果两者相等，就顺序执行；如果不相等，就转移。同样地，也可以将 rel 理解成标号，即“CJNE A，#data, 标号”，这样利用这条指令就可以判断两数是否相等。这种用法在很多场合是非常有用的，但有时还想知道两数比较之后哪个大，哪个小，本条指令可以用 CY（进位位）来实现这样的功能：如果前面的数（A 中的）大，则 CY=0；否则 CY=1。因此在程序转移后再次利用 CY 就可判断出 A 中的数比 data 大还是小。

例如在下面的代码段中，指令“CJNE A,#0AH,LOOP1”使用的是第一种功能，即判断 A 是否等于#0AH；指令“CJNE A,#32H,CMP_R2”使用的则是第二种功能，即比较 A 与#32H 的大小，随后的 JC 指令才是真正的判断选择指令，CJNE 仅仅是影响了 CY 位以供 JC 指令使用。

【例 1-7】CJNE 的两种用途。

```
LOOP1:
   INC      R1            ;R1←R1+1
   MOV      A,R1      ;A←R1
   CJNE     A,#0AH,LOOP1;若 A≠0AH，转到 LOOP1 执行
   MOV      R1,0      ;否则，R1←0
   INC      R2            ; R2←R2+1
   MOV      A,R2      ; A←R2
   CJNE     A,#32H,CMP_R2;比较 R2 与 32H，若 A<32H 则 CY=0
CMP_R2:
   JC       LOOP1         ;对 CY 进行判断，若 CY=0 转到 LOOP1
   MOV      R2,0      ;否则，R2←0
   SJMP     R1            ;跳转到 LOOP1
```

（4）减 1 条件转移指令

减 1 条件转移指令有如下两条：

```
DJNZ Rn,rel       ;若 Rn-1≠0，则 PC←PC+2+rel
                  ;若 Rn-1=0，则 PC←PC+2
DJNZ direct,rel   ;若(direct)-1≠0，则 PC←PC+3+rel
                  ;若(direct)-1=0，则 PC←PC+3
```

指令“DJNZ　Rn,rel”执行时先把 Rn 中的内容减 1，然后判断 Rn 中的内容是否为 0，若不为 0，则程序发生转移；若为 0，则程序继续执行。“DJNZ　direct,rel”与之类似，只是减 1 的操作数不是在 Rn 中，而是在 direct 中。

【例 1-8】DJNE 的用法。将片内 RAM 中以 DAT 为起始地址的数据块中的连续 10 个无符号数相加，并将和的高 8 位送到 SUM_H 单元，低 8 位送到 SUM_L 单元。参考程序如下：

```
    MOV     R2,#0AH  ;数据块长度送 R2
    MOV     R1,#DAT  ;数据块起始地址送 R1
    CLR     A        ;累加器清零
    MOV     SUM_H,A  ;SUM_H 单元清零
    MOV     SUM_L,A  ; SUM_L 单元清零
LOOP:
    CLR     C        ;CY 位清零
    MOV     A,SUM_L  ;SUM_L 单元中数据送入累加器
    ADD     A,@R1    ;加一个数
    MOV     SUM_L,A  ;将和的低 8 位送 SUM_L
    MOV     A,SUM_H  ; SUM_H 单元中数据送入累加器
    ADDC    A,#0H    ;与进位相加
    MOV     SUM_H,A  ;将和的高 8 位送 SUM_H
    INC     R1       ;修改加数地址指针
    DJNZ    R2,LOOP  ;若 R2-1≠0，则跳转到 LOOP
    SJMP    $        ;否则，停机
```

注意　条件转移指令均为相对转移指令，因此指令的转移范围十分有限。若要实现 64KB 范围内的转移，则可以借助于一条长转移指令的过渡来实现。

（5）子程序调用和返回指令

为了减少编写和调试程序的工作量，以及减少程序在程序存储器中所占的存储空间，常常把具有完整功能的程序段定义为子程序供主程序在需要时调用。为实现主程序对子程序的一次完整调用，主程序应该能够在需要的时候通过子程序调用指令自动转入子程序执行，子程序执行完毕后应能通过返回指令自动返回调用指令的下一条指令执行。因此，子程序调用指令是在主程序需要调用子程序时使用的，返回指令则需要放在子程序末尾。

子程序调用指令有如下两条：

```
ACALL    addr11      ; PC←PC+2，SP←SP+1，(SP)←PC7~PC0
                     ;SP←SP+1，(SP)←PC15~PC8
                     ; PC10~PC0←addr11
LCALL    addr16      ; PC←PC+3，SP←SP+1，(SP)←PC7~PC0
                     ;SP←SP+1，(SP)←PC15~PC8
                     ; PC15~PC0←addr16
```

指令“ACALL　addr11”为短跳转指令，该指令执行时，PC 先加 1 两次，然后分别把 PC 的低 8 位和高 8 位压入堆栈，最后把 PC 加 1 两次后的高 5 位地址 PC15～PC11 作为子程序起始地址的高 5 位，把 addr11 作为子程序起始地址的低 11 位，并按该地址转入子程序执行。在实际编程时，addr11 可用标号表示。需要注意的是本调用指令应与被调子程序起始地址在同一个 2KB 范围内。

指令“LCALL　addr16”为长跳转，该指令执行时，PC 先加 1 三次，然后把 PC+3 后的地址压入堆栈，最后把 addr16 送入程序计数器 PC 执行。长调用指令是一种 64KB 范围内的调用指令，主程序和被调用子程序可以任意地放在 64KB 范围内。

返回指令也有两条：

```
RET          ;PC15~PC8←(SP)，SP←SP-1
             ;PC7~PC7←(SP)，SP←SP-1
RETI         ;PC15~PC8←(SP)，SP←SP-1
             ;PC7~PC7←(SP)，SP←SP-1
```

指令“RET”和“RETI”的功能相同，都是把堆栈中的断点地址恢复到程序计数器 PC 中，从而使单片机回到断点地址处执行程序。需要注意的是，RET 称为子程序返回指令，只能用在子程序末尾；RETI 称为中断返回指令，只能用在中断服务程序末尾。

【例 1-9】用子程序技术编写将 20H～2AH 和 30H～3EH 两个存储单元段清零的程序。参考程序如下：

```
    MOV      SP,#60H ;令堆栈的栈底地址为#60H
    MOV      R0,#20H ;第一清零区起始地址送 R0
    MOV      R2,#0BH ;第一清零区单元数送 R2
    ACLAA    CLEAR   ;调用子程序清空第一个区域
    MOV      R0,#30H ;第二清零区起始地址送 R0
    MOV      R2,#0FH ;第二清零区单元数送 R2
    ACLAA    CLEAR   ;调用子程序清空第二个区域
    SJMP     $
CLEAR:
    MOV      @R0,#00H;清零
    INC      R0      ;修改清零区指针
    DJNZ     R2,CLEAR;若 R2-1≠0，则跳转到 CLEAR
    RET              ;返回
```

（6）空指令

空指令仅使程序计数器 PC 加 1，不进行任何操作，共消耗 12 个时钟周期时间，故经常在延时程序中使用，其指令格式为：

```
NOP      ;PC←PC+1
```

5．位操作指令

MCS-51 单片机的硬件结构中有一个位处理器（又称布尔处理器）包含一套位变量处理的指令集。在进行位处理时，CY 称“位累加器”。有自己的位 RAM，也就是内部 RAM 的 20H～2FH 这 16 个字节单元，即 128 个位单元，还有自己的位 I/O 空间（即 P0.0～P0.7，P1.0～P1.7，P2.0～P2.7，P3.0～P3.7）。

位操作指令分为位传送、位置位和位清零、位运算以及位控制转移指令，共 4 类。

（1）位传送指令

位传送指令有“MOV C,bit”和“MOV bit,C”两条。位传送指令的指令格式和举例如表 1-19 所示。

表 1-19　　位传送指令

汇 编 指 令	功　　能	举　　例
MOV　C,bit	bit 中状态送入 C 中	MOV　C,60H;CY←(60H 位)
MOV　bit,C	C 中状态送入 bit 中	MOV　00H,C; (00H 位)←CY

（2）位置位和位清零指令

这类指令共有“CLR C”、“SETB C”、“CLR bit”和“SETB bit”，共 4 条。位置位和位清零指令的指令格式和举例如表 1-20 所示。

表 1-20　　位置位和位清零指令

汇 编 指 令	功　　能	举　　例
CLR　C	C 中状态清零	CLR　C　;CY←0
SETB　C	C 中状态置 1	SETB　C　;CY←1

CLR bit	bit 中状态清零	CLR 10H ; (10H 位)←0
SETB bit	bit 中状态置 1	SETB 10H ; (10H 位)←1

（3）位运算指令

这类指令共有 6 条，分为“与”、“或”、“非”3 组。位运算指令的指令格式和举例如表 1-21 所示。

表 1-21 位运算指令

汇 编 指 令	功 能	举 例
ANL C,bit	bit 中状态与 CY 相“与”，结果在 CY 中	ANL C,07H ;CY←CY∧(07H 位)
ANL C,/bit	bit 中状态取反与 CY 相“与”，结果在 CY 中	ANL C,/07H ;CY←CY∧(07H 位取反)
ORL C,bit	bit 中状态与 CY 相“或”，结果在 CY 中	ORL C,10H;CY←CY∨(10H 位)
ORL C,/bit	bit 中状态取反与 CY 相“或”，结果在 CY 中	ORL C,/10H;CY←CY∨(10H 位取反)
CPL C	CY 取反	CPL C ; CY←CY 取反
CPL bit	bit 中状态取反	CPL 65H ; (65H 位)←(65H 位取反)

（4）位控制转移指令

位控制转移指令有“JC rel”、“JNC rel”、“JB rel”、“JNB rel”、“JBC rel”，共 5 条，位控制转移指令的指令格式和举例如表 1-22 所示。

表 1-22 位控制转移指令

汇 编 指 令	功 能	举 例
JC rel	进位位为 1 时，程序转至 rel	JC LOOP ;若 CY=1，转到 LOOP
JNC rel	进位位不为 1 时，程序转至 rel	JNC CLEAER ;若 CY=0，转到 CLEAR
JB rel	bit 状态为 1 时，程序转至 rel	JB LOOP1 ;若 bit=1，转到 LOOP1
JNB rel	bit 状态不为 1 时，程序转至 rel	JNB SEND ;若 bit=0，转到 SEND
JBC rel	bit 状态为 1 时，程序转至 rel，同时 bit 位清零	JBC TEST ; 若 bit=1，转到 LOOP，bit=0

指令“JC rel”执行时，单片机先判断 CY 中的值，若 CY=1，则程序发生跳转，否则程序不跳转，继续执行原程序。

指令“JNC rel”执行时的情况与“JC rel”恰好相反：若 CY=0，则程序发生跳转，否则程序不跳转，继续执行原程序。

上述这两条指令是相对转移指令，都是以 CY 中的值来决定程序是否需要转移，常常与比较条件转移指令 CJNE 连用，以便根据 CJNE 指令执行过程中形成的 CY 进一步决定程序的流向。

指令“JB rel”和“JC rel”类似，“JNB rel” “JNC rel”类似，只是根据 bit 中的内容而不是 CY 中的内容来决定程序的走向。

指令“JBC rel”和“JB rel”作用相同，只是程序执行后还可以将 bit 清零，一条指令起到了两条指令的作用。

1.3 MCS-51 单片机的伪指令

汇编语言必须通过汇编器的处理才能转换为计算机所能识别并执行的机器语言。伪指令是汇编器用的指令，可以用来对机器的汇编过程进行某种控制，令其进行一些特殊操作。8051 汇编器常用的伪指令有 ORG、END、EQU、DATA、DB、DW、DS、BIT，共 8 种，下面逐一进行介绍。

1.3.1 ORG 伪指令

ORG 指令出现在源程序或数据块的开始，用以指明此语句后面的目标程序或数据块存放的起始地址。在一个源程序中可以多次使用 ORG 以规定不同程序段的起始地址，但定义的地址顺序应该是从小到大的。ORG 指令的一般形式为：

```
[标号:]ORG 表达式
```

表达式可以是一个具体的 16 位的数值，也可以是存放 16 位数据的变量名。如果包含变量名，则必须保证当第一次遇到这条伪指令时，其中的变量必须已有定义（已有具体的数值），否则无定义的值将由 0 替换，这将会造成错误。例如：

```
    ORG      0030H
START:
    MOV      SP,#60H
    …
    ….
    END
```

ORG 伪指令规定了 START 的起始地址为 0030H，第一条指令及其后续指令汇编后的机器码便从 0030H 开始依次存放。

1.3.2 END 伪指令

END 语句标志源代码的结束，汇编程序遇到 END 语句即停止运行。若没有 END 语句，汇编将报错。因此，一个源程序只能在主程序最后使用一个 END 指令，其格式为：

```
[标号:]END
```

1.3.3 EQU（或=）伪指令

EQU 伪指令用于给一个表达式的值或一个字符串起一个名字，之后这个名字可以用做程序地址、数据地址或立即数使用，因此表达式可以是 8 位或 16 位二进制数。名字必须是以字母开头的字母数字串，可以包含下划线，必须是先前没有定义过的。EQU 的一般形式为：

```
名字 EQU 表达式
```

或

```
名字=表达式
```

例如：

```
STACK_BTTM   EQU 060H;此后 STACK_BTTM 即为 60H
COUNTER=10              ;此后 COUNTER 即为 60H
```

1.3.4 DATA 伪指令

DATA 伪指令称为数据地址赋值伪指令，也可以用来把其右边的表达式的值赋给其左边的字符名称。DATA 伪指令与 EQU 伪指令的主要区别是：EQU 定义的字符名称必须先定义后使用，而 DATA 定义的字符名称没有这种限制，故 DATA 伪指令通常用在源程序的开头或末尾。例如：

```
DALAY_TIME   DATA 0AH;DALAY_TIME 即为 0AH
```

1.3.5 DB 伪指令

DB 称为定义字节伪指令，用于定义一个连续的以字节为单元的存储区，给该存储区的存储单元赋值。该伪指令的参数即为存储单元的值，在表达式中对变元个数没有限制，只要此条伪指令能容纳在源程序的一行内。只要表达式不是字符串，每一表达式值都被赋给一个字节。计算表达式值时按 16 位处理，但其结果只取低 8 位，若多个表达式出现在一个 DB 伪指令中，表达式之间必须以逗号分开。表达式中有字符串时，以单引号作分隔符，每个字符占一个字节，字符串不加改变地被存在各字节中，并不进行大小写之间的转换，其一般形式为：

```
[标号:]DB 项或项表
```

例如：

```
DB  02H
DB  01111110B,00110000B,01101101B
DB  'C','5','1'
```

1.3.6 DW 伪指令

DW 称为定义字伪指令，用于为源程序在存储器某个区域定义一个或一串字，和 DB 的主要区别在于 DW 定义的是一个字（即两个字节）。DW 伪指令主要用来定义 16 位地址（高 8 位在前，低 8 位在后），其格式为：

```
[标号:]DW 项或项表
```

例如：

```
DW 12341,54379,10110100101110B
DW 456*375H,83+295H,'YZ',72h-456
```

1.3.7 DS 伪指令

DS 称为定义存储空间伪指令。其一般形式为：

```
[标号:]DS 表达式
```

DS 指令可以指示汇编程序从其标号地址（或实际物理地址）开始预留一定数量的内存单元，以备源程序执行过程中使用。这个预留单元的数量（字节数）由 DS 语句中的“表达式”的值决定。例如：

```
RESERVE:DS  05H
```

汇编程序遇到 DS 语句就自动从 RESERVE 地址开始预留 5 个连续的内存单元，从第 6 个存储单元继续存储编译后的代码。

1.3.8 BIT 伪指令

BIT 称为位地址赋值伪指令，用于给以符号形式的位地址赋值。BIT 指令的格式是：

```
字符名称 BIT 位地址
```

该语句的功能是把 BIT 右边的位地址赋给其左边的字符名称。因此，用 BIT 语句定义过的字符名称是一个符号位地址。例如：

```
MS500_FLG   BIT 07H
SIGNAL      BIT P0.4
```

MS500_FLG 和 SIGNAL 经 BIT 语句定义后便可以作为位地址使用，其中 MS500_FLG 的物理位地址是 07H，SIGNAL 的物理位地址为 P0.4。

以上几节简要介绍了 51 单片机汇编语言的指令集和伪指令。但是，仅仅知道这些知识还是不够的，还必须在实践中多加练习，才能透彻地理解并掌握单片机汇编语言的编程方法和技巧。

1.4 MCS-51 汇编语言编程实例

单片机通常用在控制领域，最简单的应用就是通过某些口线的输入去控制另外一些口线的输出。本实例就是通过一根端口线的输入去控制另外 4 根端口线的输出，具体功能如下：在端口线 P0.0 上接一个按键，单片机计算按键闭合的次数，将这个数字通过 P2.3～P2.0 这 4 根口线所驱动的发光二极管显示出来，其中 P2.3 是最高位，P2.0 是最低位，指示灯亮表示该位为 1，否则表示该位为 0。显然，由于只有 4 个指示灯，只能表现 16 种状态，因此能表现的计数只能在 0～15 之间。例如，4 只指示灯全灭表示 0，即二进制的 0000；4 只指示灯全亮表示 15，即二进制的 1111，依此类推。

电路图如图 1.18 所示。电路中有几点需要说明。

- 89S51 是 51 系列单片机的一种，引脚功能和指令集等完全与 8051 兼容。
- 按键的检测方法为：当键没有被按下时，P0.0 端口通过上拉电阻接 V_{CC}，此时为高电平；键按下时，P0.0 通过按键接地，为低电平，因此可以通过检测 P0.0 端口的电平来判断当前键是否被按下。
- 由于单片机的引脚和端口驱动能力都有限，因此外接了一个反相器以提供发光二极管所需要的电流。当某个引脚输出为高电平时，经过反相器反相变为低电平，从而电流从 VCC 通过发光二极管和电阻流入反相器，发光二极管发光。

在程序中，首先检测当前键是否被按下。但是仅仅检测到键被按下并不意味着就应该改变显示的数字，因为可能这次按键已经被检测到了一次，被处理过了，因此必须还要判断此次检测到按键是否是第一次检测到。程序流程如图 1.19 所示。

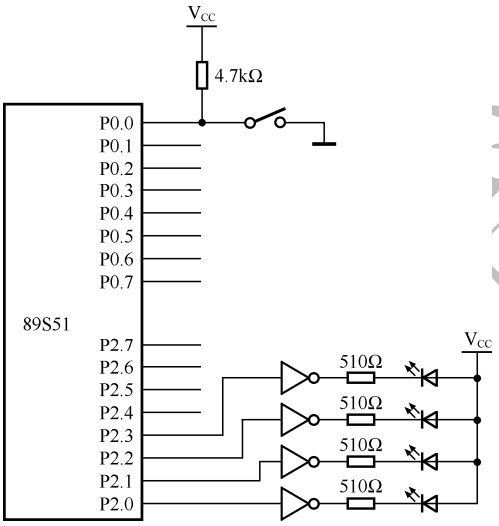


图 1.18 通过按键控制发光二极管的电路原理图

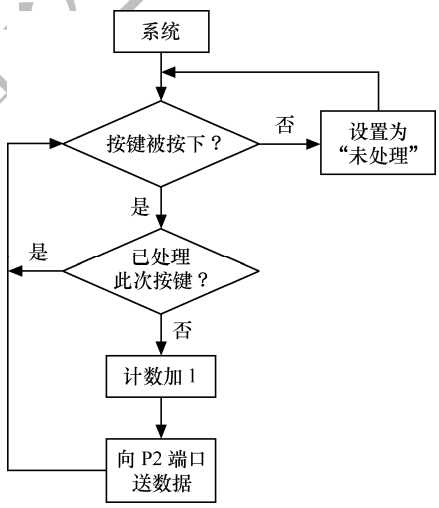


图 1.19 通过按键控制发光二极管的程序流程

完整的汇编语言程序如例 1-10 所示。

【例 1-10】用 4 个发光二极管显示按键次数的汇编语言程序。在程序中有一个主程序 MAIN 和两个子程序 KEY_TEST 和 DISPLAY。MAIN 程序是一个无限循环，反复调用 KEY_TEST 和 DISPLAY。KEY_TEST 的功能是检测按键是否被按下以及是否被处理，如果没有被处理就设置标志位 INC_FLG，使能 DISPLAY 子程序将计数加 1 并显示。DISPLAY 完成这项工作时，要将 KEY_PROCESSED 标志位置 1，以通知 KEY_TEST 子程序此次按键已被处理。代码如下所示：

```
INC_FLG          BIT  00H      ;将 INC_FLG 赋值为位地址 00H
KEY_PROCESSED    BIT  01H      ;将 KEY_PROCESSED 赋值为位地址 01H
KEY              BIT  P0.0;将 KEY 赋值为位地址 P0.0
STACK_BTTM       EQU  60H      ;将 STACK_BTTM 赋值为 60H
;----------------------------------------
```

```
    ORG      0000H            ;上电后从 0000H 开始执行
    LJMP RESET                ;0000H~0030H 为中断向量，不能放置其他代码
    ORG      0030H            ;跳转到 0030H 执行
RESET:
    MOV  SP,#STACK_BTTM       ;修改栈底地址
    MOV  R0,#0FFH          ; R0←0FFH
CLR_RAM:                   ;以下两行清空 RAM 区
    MOV  @R0,#0H              ;将 R0 所指向的 RAM 单元清零
    DJNZ CLR_RAM              ;修改 RAM 并重复上一语句，直至 R0=0
     MOV P3,#0H               ;初始时，4 个指示灯全灭
MAIN:                         ;主程序
    ACALL    KEY_TEST      ;调用按键检测子程序
    ACALL    DISPLAY          ;调用计数与显示子程序
    AJMP MAIN              ;跳转到地址 MAIN，无限循环
;-----------------------------------
KEY_TEST:                     ;按键检测子程序
    SETB KEY                  ;读引脚状态前要先将引脚置高
    MOV  C,KEY                ;将引脚状态读入 CY 中
    JC       CLEAR_RET        ;若 CY=1，说明键没有按下，返回
    JB       KEY_PRESSED,KEY_TEST_RET
;有键按下，还要检查此次按键是否已被处理过。若已被处理，则返回
    SETB INC_FLG              ;将 INC_FLG 置 1，允许计数加 1
KEY_TEST_RET:
    RET                       ;返回
CLEAR_RET:
    CLR      KEY_PROCESSED    ;若按键松开，清零 KEY_PROCESSED
    RET                       ;返回
;-----------------------------------
DISPLAY:                   ;计数与显示子程序
    JNB      INC_FLG,DIS_RET  ;若不允许计数加 1，返回
    CLR      INC_FLG          ;处理一次后就不再处理
    SETB KEY_PROCESSED        ;将代表按键已处理的标志位置 1
    INC      COUNTER          ;计数加 1
    MOV  A,#0FH               ;A←0FH
    ANL      A,COUNTER        ;取 COUNTER 的低 4 位
    MOV  P3,A              ;送显示数字
DIS_RET:
    RET                       ;返回
    END                       ;源程序文件结束
```

1.5 MCS-51 单片机 C 语言简介

1.5.1 用 C 语言开发单片机的优势

尽管汇编语言在控制底层硬件方面有着良好的性能且执行效率高，但其本身是一种低级语言，编程效率低下，且可移植性和可读性差，维护极不方便，从而导致整个系统的可靠性也较差。而 C 语言以其结构化和能产生高效代码等优势满足了开发人员的需要，成为开发人员进行单片机编程的首选开发语言，得到了广泛的支持。C 语言具有汇编语言所不可比拟的优势，如下所示。

- 可以大幅度加快开发进度，特别是开发一些复杂的系统，程序量越大，用 C 语言就越有优势。
- 可以实现软件的结构化编程，C 语言使得软件的逻辑结构变得清晰、有条理，便于开发小组计划项目、分工合作。
- 省去了人工分配单片机资源（包括寄存器、RAM 等）的工作。在汇编语言中要为每一个子程序分配单片机的资源，这是一个复杂、乏味而且容易出差错的工作。在使用 C 语言后，只要在代码中声明一下变量的类型，编译器就会自动分配相关资源，根本不需要人工干预，从而有效避免了人工分配单片机资源所可能带来的差错。
- 当写好了一个算法（在 C 语言中称为函数）后，需要移植到不同种类 MCU 上时，在汇编语言中只有重新编写代码，因而汇编语言的可移植性很差；而用 C 语言开发时，符合 ANSI C 标准的程序基本不必修改，只要将一些与硬件相关的代码作适度的修改，就可以方便地移植到其他种类的单片机上。
- C 语言提供 data、bdata、idata、pdata、xdata 和 code 等存储器类型，针对单片机的内部数据存储空间、外部数据存储空间和程序存储空间自动为变量合理地分配空间，而且 C 语言提供复杂的数据类型，如数组、指针、结构体等，极大地增强了程序处理能力和灵活性。C 语言编译器能够自动实现中断服务程序的现场保护和恢复，并且提供了常用的标准函数库供用户使用，使用户节省了重复编写相同代码的时间，并且 C 语言编译器能够自动生成一些硬件初始化代码。

虽然使用 C 语言写出来的代码会比用汇编语言占用的空间大 5%～20%，但是由于半导体技术的发展，芯片的容量和速度都有了大幅度的提高，占用空间大小的差异已经不是很关键，相比之下，应该更注重软件是否具有长期稳定运行的能力，注重使用先进开发工具所带来的时间和成本的优势。因此，使用 C 语言开发单片机程序已经成为一种趋势。

1.5.2 C51 程序的例子

例 1-10 给出了如图 1.18 所示的电路图的汇编语言程序，可以看到，使用汇编语言设计单片机程序时，需要对单片机的硬件资源和指令集比较熟悉，程序设计的技巧性较强。另外，汇编语言编写的程序可读性不好，需要添加大量的注释语句以帮助理解，否则其他开发人员难于读懂。如果使用 C 语言编写程序，则无需精通单片机指令集和具体的硬件也能够编写出符合硬件实际的专业水平的程序，源程序的可读性和可维护性都很好，基本上可以杜绝因开发人员变化而给项目进度或后期维护以及升级所带来的影响，从而保证整个系统的稳定性。

下面给出了功能完全相同的用 C51 语言编写的程序。

【例 1-11】用 C51 语言重写例 1-10。相对于汇编语言编程而言，程序的可读性大大增强，也更加容易维护。

```
#include <reg51.h>              //文件包含，将头文件 reg51.h 包括进来
sbitkey=P0^0;               //将变量 key 定义为 P0^0
bit processed;                  //定义一个位变量 processed
void main(void)                 //主函数
{
   unsigned char counter;       //定义一个变量 counter 用来计数
   while(1)                     //无限循环
   {
    if(key==1)                  //如果没有键按下，执行大括号里的语句
    {
        processed=0;        //设置为“未处理”
        continue;               //跳转到“while(1)”语句
    }
    if(processed==0)        //如果有键按下且尚未处理，执行大括号里的语句
    {
        counter+=1;             //按键计数加 1
        P2=counter&0x0F;    //在 P2 端口显示数据
        processed=1;        //设置为“已处理”
```

```
        }
      }
    }
```

1.5.3 C51 程序在 Keil C51 下的编译、仿真

现在广泛使用的 C51 集成开发环境是 Keil 软件。Keil 软件的详细说明见本书第 9 章，这里仅作一简要说明，以便于读者在后面的章节中实际动手调试、仿真程序。

1．程序的编辑和编译

要使用 Keil 软件，首先要正确安装 Keil 软件，该软件的 Eval 版本可以直接去 http://www. keil.com 下载，其步骤与一般 Windows 程序安装类似，这里就不再赘述了。

启动μVison2，如果打开后就有打开的文件，单击“Project→Close Project”菜单将其关闭。单击“File→New…”，打开一个新的文件输入窗口，在这个窗口里输入例 1-11 中的源程序。输入完毕之后，选择“File→Save”，给这个文件取名保存，取名字的时候必须要加上扩展名，一般 C 语言程序均以“.c”为扩展名，这里将其命名为 mytest.c。

单击“Project→New Project…”，出现对话框，要求给将要建立的工程起一个名字，这里起名为 test，不需要输入扩展名。单击“保存”按钮，出现第二个对话框，如图 1.20 所示：

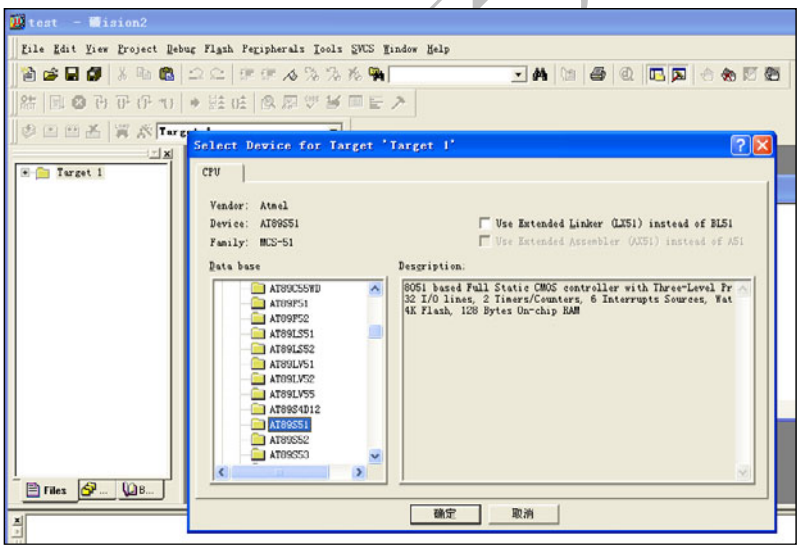


图 1.20 CPU 选择窗口

这个对话框要求选择工程中所用的单片机型号，这里选择 Atmel 公司的 89S51 芯片。单击 ATMEL 前面的“+”号，展开该层，单击其中的 89S51，然后再单击“确定”按钮，会出现如图 1.21 所示的一个对话框，询问是否要将标准的 8051 启动代码加入工程，单击“是”按钮。

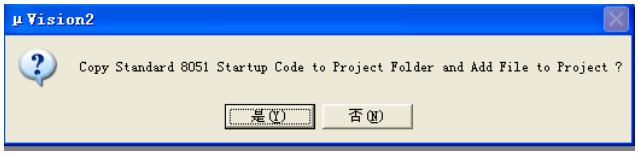


图 1.21 标准启动代码选择窗

此时，在工程窗口的文件页中出现了“Target 1”，前面有“+”号，单击“+”号展开，可以看到下一层的“Source Group1”，需要手动把刚才编写好的源程序加入，方法如下：

单击“Source Group1”，使其反白显示，然后单击鼠标右键，出现一个下拉菜单，如图 1.22 所示。

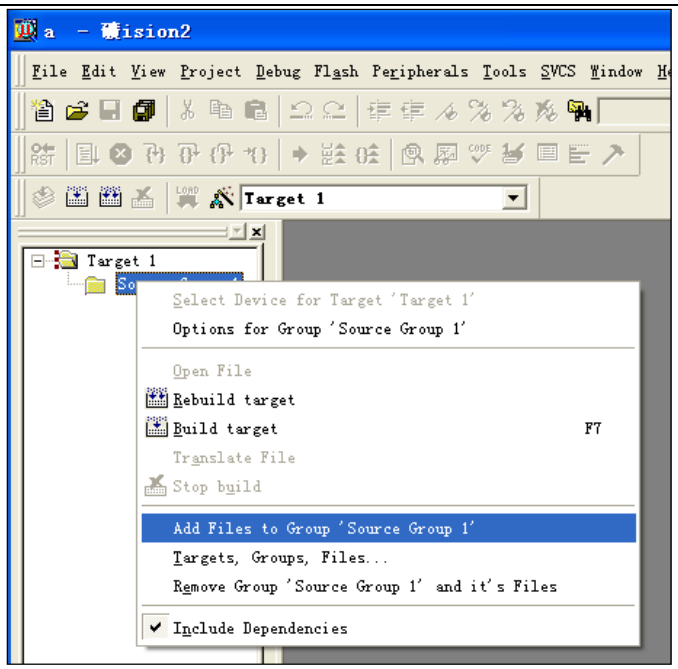


图 1.22　添加文件菜单

选中其中的“Add file to Group”Source Group1”，出现一个对话框，要求寻找源文件。双击 mytest.c 文件，将文件加入项目，然后单击“Close”按钮即可返回主窗口。返回后单击“Source Group 1”前的加号，可以看到 mytest.c 文件已在其中，双击文件名即可以打开该源程序。

接下来要对一些工程选项进行设置。单击鼠标右键，选择资源窗口中的“Target 1”，然后选择弹出菜单中的“Options for Target ’Target 1’”，在弹出的对话框中选择“Output”标签页，选中该页的“Creat HEX File”前面的复选框，这样在编译后就会生成可以用于烧写的.hex 文件。“Name of Executable”表示将要生成的.hex 文件的名称，这里改为“mytest”，如图 1.23 所示。

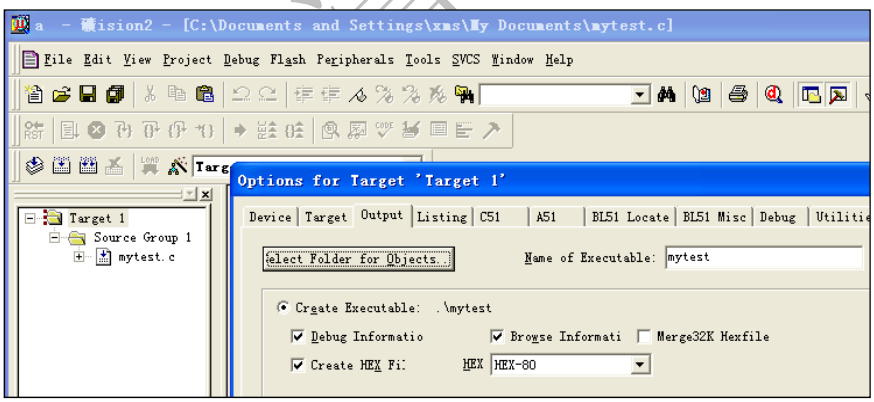


图 1.23　选择输出 HEX 文件

设置好工程选项后即可进行编译、连接。选择“Project→Build target”可以对当前工程进行连接。如果当前文件已修改，将先对该文件进行编译，然后再连接以产生目标代码；如果选择“Project→Rebuild All target files”将会对当前工程中的所有文件（无论是否修改过）重新进行编译，然后再连接以产生目标代码；而如果选择“Project→Translate ….”，项则仅对当前文件进行编译，不进行连接，也就不会产生新的目标代码。

以上操作也可以通过工具栏按钮进行。编译、设置的工具栏按钮如图 1.24 所示，从左到右分别是：编译、编译连接、全部重建、停止编译和对工程进行设置。

编译过程中的信息将出现在输出窗口中的 Build 页中，可以得到如图 1.25 所示的结果，提示获得了名为 mytest.hex 的文件，该文件即可被编程器读入并写到芯片中。同时还可看到该程序的代码量（code=37）、内部 RAM 的使用量（data=10.1，小数表示“位”）、外部 RAM 的使用量（xdata=0）等一些信息。除此之外还产生了一些其他相关的文件可被用于 Keil 的仿真与调试。

图 1.24　有关编译、设置的工具栏按钮

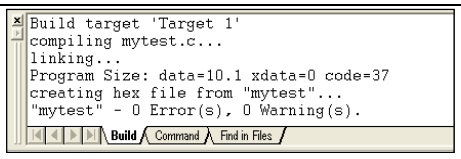


图 1.25　有关编译过程的信息窗口

2．程序的仿真与调试

程序的仿真和调试在开发中有着非常重要的作用，可以使开发者在有完整的硬件电路之前就可以验证自己的程序是否能达到预期的目标。

工程成进行汇编、连接成功以后，按组合键【Ctrl+F5】或者选择“Debug→Start/Stop DebugSession”即可进入调试状态，菜单项中出现了一个用于仿真运行和调试的工具条，如图 1.26 所示。

图 1.26　运行和调试工具条

Debug 菜单上快捷按钮从左到右依次是复位、运行、暂停、单步、过程单步、执行完当前子程序、运行到当前行、下一状态、打开跟踪、观察跟踪、反汇编窗口、观察窗口、代码作用范围分析、1#串行窗口、内存窗口、性能分析、工具按钮等命令的快捷按钮。

对于例 1-11 的程序，由于要用到 P0 和 P2 端口的输入输出，因此在仿真的时候要仿真或观察 P0、P2 端口的状态。选择“Peripherals→I/0-Ports→Port 0”和“Peripherals→ I/0-Ports→Port 2”，在主窗口中会出现 P0、P2 口的状态，需要在程序运行的时候改变 P0.0（即“Parallel Port 0”对话框中最右下角的复选框），并观察 Parallel Port 0 对话框中各位的变化情况，其中有对号的表示该位为逻辑“1”，没有对号的表示该位为逻辑“0”，如图 1.27 所示。

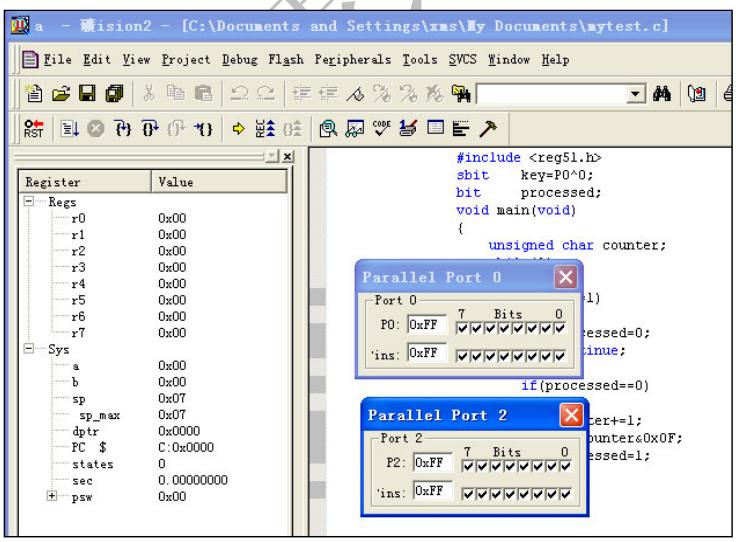


图 1.27　并行端口仿真对话框

依次单击图 1.27 中最左边的两个快捷按钮，即复位和运行，程序就开始仿真执行，图 1.26 中的“暂停”快捷按钮由灰色变为红色，如图 1.28 所示。

图 1.28　仿真运行时的运行和调试工具条

此时单击“Parallel Port 0”对话框中最右下角的复选框将对号去掉就相当于按键按下，再次单击又会出现对号，相当于按键松开。随着 P0.0 的变化，“Parallel Port 0”对话框的状态也会发生变化，如图 1.29 所示。

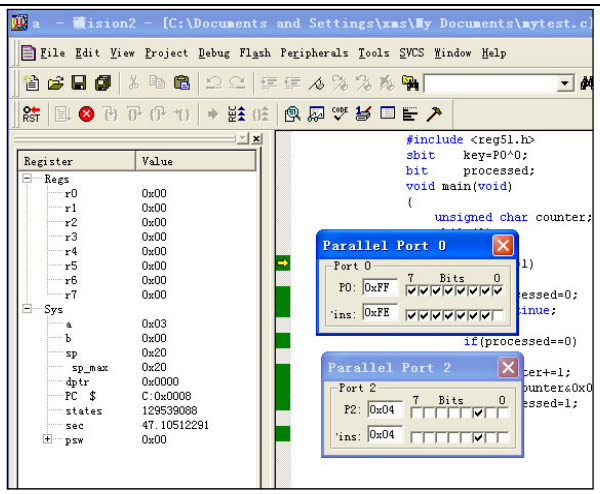


图 1.29　仿真端口输入状态的改变

3．串口程序的仿真与调试

由于单片机的输出是通过串口完成的，因此经常需要观察串口输出窗口。在以后的章节的例子中，凡是有用到 printf()语句的程序都要打开串口窗口以观察输出。

打开串口输出窗口的方法就是单击运行和调试工具条中的“1#”串行窗口，之后就会在主窗口中出现标题为“Serial #1”的子窗口，串口的输出就是在这个窗口中显示出来。为方便观察，可以选择“Window→Cascade”，使多个子窗口成为层叠的形式，如图 1.30 所示。

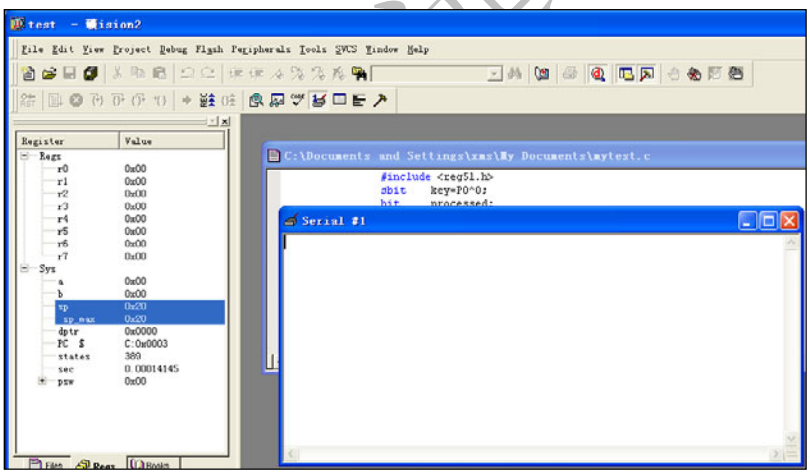


图 1.30　串口调试窗口

1.6　小结

作为入门章节，本章主要介绍了 MCS-51 系列单片机的内部结构、指令集和汇编伪指令，然后通过一个实际的例子说明了汇编语言的编程方法。

C 语言相对于汇编语言在许多方面有着巨大的优势。C 语言经不同的编译器编译后可以用于多种 CPU，用于 51 单片机的 C 语言就称为 C51 语言。本章实例也给出了用 C51 语言编写的功能完全相同的程序。

本书采用 Keil C51 作为开发环境，因此简要介绍了 Keil C51 的使用方法，以便于读者在以后的章节中实际仿真、调试程序。

联系方式

集团官网：www.hqyj.com 嵌入式学院：www.embedu.org 移动互联网学院：www.3g-edu.org

企业学院：www.farsight.com.cn 物联网学院：www.topsight.cn 研发中心：dev.hqyj.com

集团总部地址：北京市海淀区西三旗悦秀路北京明园大学校内 华清远见教育集团

北京地址：北京市海淀区西三旗悦秀路北京明园大学校区，电话：010-82600386/5

上海地址：上海市徐汇区漕溪路 250 号银海大厦 11 层 B 区，电话：021-54485127

深圳地址：深圳市龙华新区人民北路美丽 AAA 大厦 15 层，电话：0755-25590506

成都地址：成都市武侯区科华北路 99 号科华大厦 6 层，电话：028-85405115

南京地址：南京市白下区汉中路 185 号鸿运大厦 10 层，电话：025-86551900

武汉地址：武汉市工程大学卓刀泉校区科技孵化器大楼 8 层，电话：027-87804688

西安地址：西安市高新区高新一路 12 号创业大厦 D3 楼 5 层，电话：029-68785218

广州地址：广州市天河区中山大道 268 号天河广场 3 层，电话：020-28916067