

单片机笔记

这是我以前学习8051的时候所做的一些笔记，今天整理了一下，转成了PDF共享出来！

希望志同道合的人能和我多交流！

我的BLOG:<http://hi.baidu.com/idiotlw>

《平凡的单片机教程》

一. 概述

1. 何谓单片机

一台能够工作的计算机要有这样几个部份构成：CPU（进行运算、控制）、RAM（数据存储）、ROM（程序存储）、输入/输出设备（例如：串行口、并行输出口等）。在个人计算机上这些部份被分成若干块芯片，安装一个称之为主板的印刷线路板上。而在单片机中，这些部份，全部被做到一块集成电路芯片中了，所以就称为单片（单芯片）机，而且有一些单片机中除了上述部份外，还集成了其它部份如 A/D，D/A 等。

2. 单片机的历史

MCS51是指由美国 INTEL 公司（对了，就是大名鼎鼎的 INTEL）生产的一系列单片机的总称，这一系列单片机包括了好些品种，如 8031，8051，8751，8032，8052，8752 等，其中 8051 是最早最典型的产品，该系列其它单片机都是在 8051 的基础上进行功能的增、减、改变而来的，所以人们习惯于用 8051 来称呼 MCS51 系列单片机，而 8031 是前些年在我国最流行的单片机，所以很多场合会看到 8031 的名称。INTEL 公司将 MCS51 的核心技术授权给了很多其它公司，所以有很多公司在做以 8051 为核心的单片机，当然，功能或多或少有些改变，以满足不同的需求，其中 89C51 就是这几年在我国非常流行的单片机，它是由美国 ATMEL 公司开发生产的。以后我们将用 89C51 来完成一系列的实验。

二. 存储器

1. 数的本质和物理现象

我们来看，这个 000，001，101 不就是我们学过的的二进制数吗？本来，灯的亮和灭只是一种物理现象，可当我们把它们按一定的顺序排更好后，灯的亮和灭就代表了数字了。让我们再抽象一步，灯为什么会亮呢？看电路 1，是因为输出电路输出高电平，给灯通了电。因此，灯亮和灭就可以用电路的输出是高电平还是低电平来替代了。这样，数字就和电平的高、低

联系上了。(请想一下,我们还看到过什么样的类似的例子呢?(海军之)灯语、旗语,电报,甚至红、绿灯)

2. 存储器的构造

(1) 存储器就是用来存放数据的地方。它是利用电平的高低来存放数据的,也就是说,它存放的实际上是电平的高、低,而不是我们所习惯认为的 1234 这样的数字,这样,我们的一个谜团就解开了,计算机也没什么神秘的吗。

(2) 一个存储器就象一个个的小抽屉,一个小抽屉里有八个小格子,每个小格子就是用来存放“电荷”的,电荷通过与它相连的电线传进来或释放掉,至于电荷在小格子里是怎样存的,就不用我们操心了,你可以把电线想象成水管,小格子里的电荷就象是水,那就好理解了

(3) 有了这么一个构造,我们就可以开始存放数据了,想要放进一个数据 12,也就是 00001100,我们只要把第二号和第三号小格子里存满电荷,而其它小格子里的电荷给放掉就行了

三. 单片机的内,外部结构

1. 单片机的外部结构

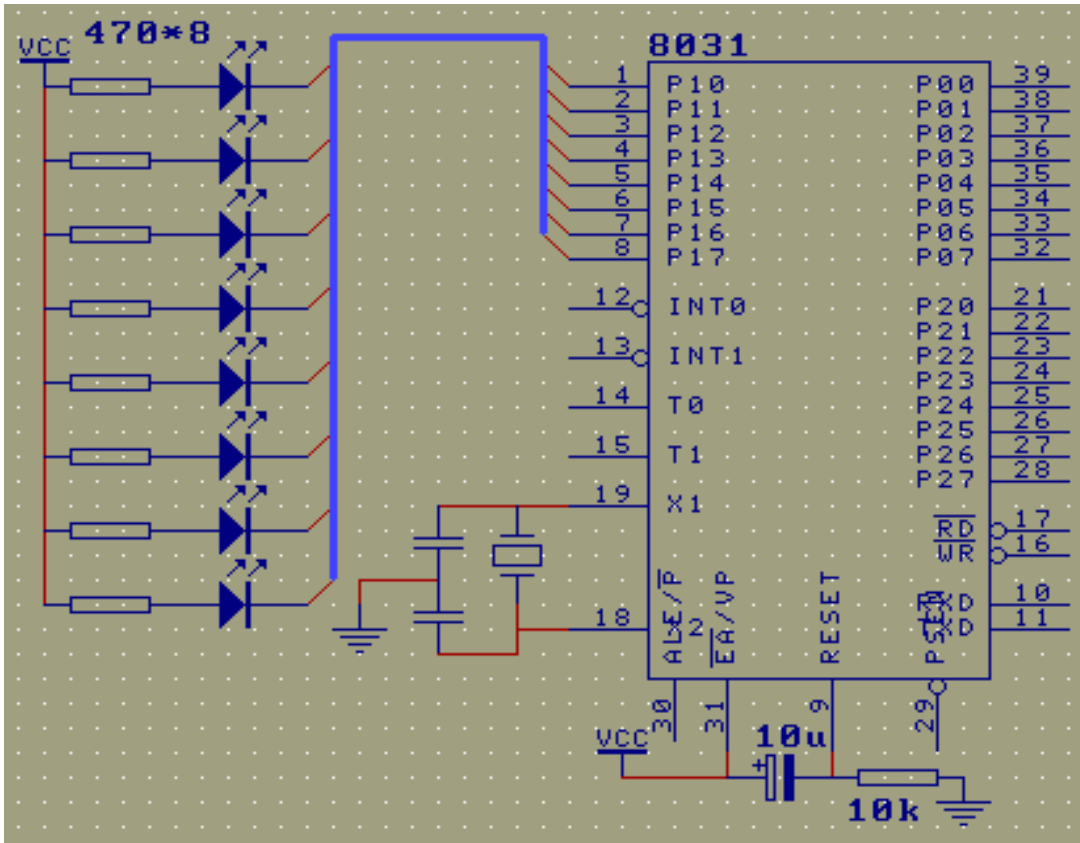
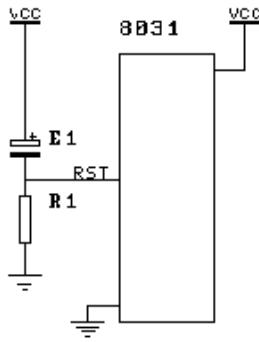
拿到一块芯片,想要使用它,首先必须要知道怎样连线,我们用的一块称之为 89C51 的芯片,下面我们就看一下如何给它连线。

(1) 电源:这当然是必不可少的了。单片机使用的是 5V 电源,其中正极接 40 引脚,负极(地)接 20 引脚。

(2) 振荡电路:单片机是一种时序电路,必须提供脉冲信号才能正常工作,在单片机内部已集成了振荡器,使用晶体振荡器,接 18、19 脚。只要买来晶振,电容,连上就可以了,按图 1 接上即可。

(3) 复位引脚:按图 1 中画法连好,至于复位是何含义及为何需要复要复位,在单片机功能中介绍。

(4) EA 引脚:EA 引脚接到正电源端。至此,一个单片机就接好,通上电,单片机就开始工作了。



2. 时序分析

(1) 机器周期:

我们已知，计算机工作时，是一条一条地从 ROM 中取指令，然后一步一步地执行，我们规定：计算机访问一次存储器的时间，称之为一个机器周期。这是一个时间基准，好象我们人用“秒”作为我们的时间基准一样，为什么不干脆用“秒”，多好，很习惯，学下去我们就会知道用“秒”反而不习惯。

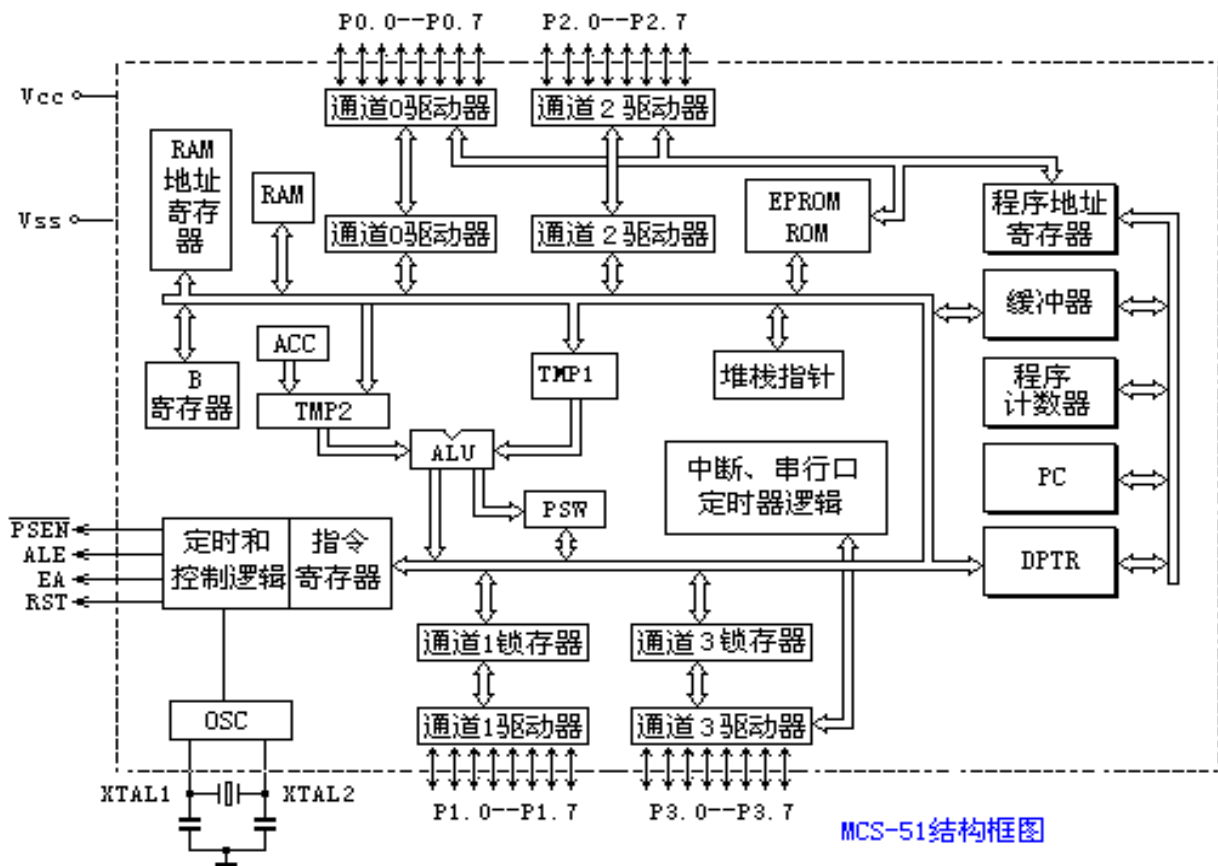
(2) 时钟周期:

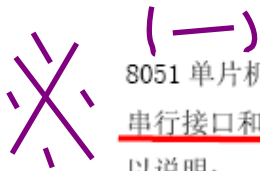
一个机器周期包括 12 个时钟周期。下面让我们算一下一个机器周期是多长时间吧。设一个单片机工作于 12M 晶振，它的时钟周期是 1/12（微秒）。它的一个机器周期是 12*（1/12）也就是 1 微秒。（请计算一个工作于 6M 晶振的单片机，它的机器周期是多少）。

(3)指令周期：

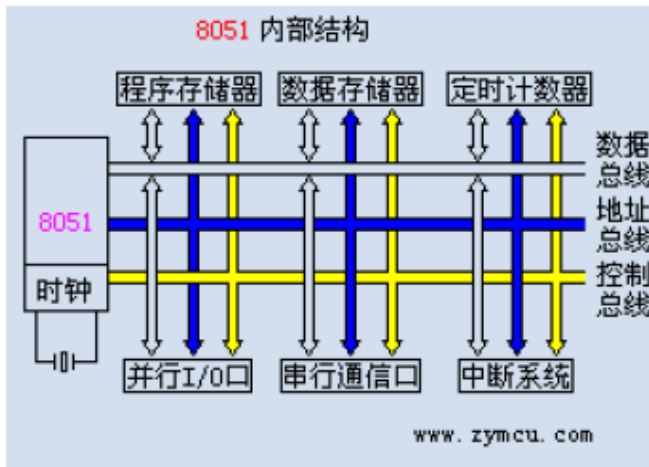
为了恒量指令执行时间的长短，又引入一个新的概念：指令周期。所谓指令周期就是指执行一条指令的时间。INTEL 对每一条指令都给出了它的指令周期数，这些数据，大部份不需要我们去记忆，但是有一些指令是需要记住的，如 DJNZ 指令是双周期指令。

3. 单片机的内部结构





8051 单片机包含中央处理器、程序存储器 (ROM)、数据存储器 (RAM)、定时/计数器、并行接口、串行接口和中断系统等几大单元及数据总线、地址总线和控制总线等三大总线，现在我们分别加以说明：



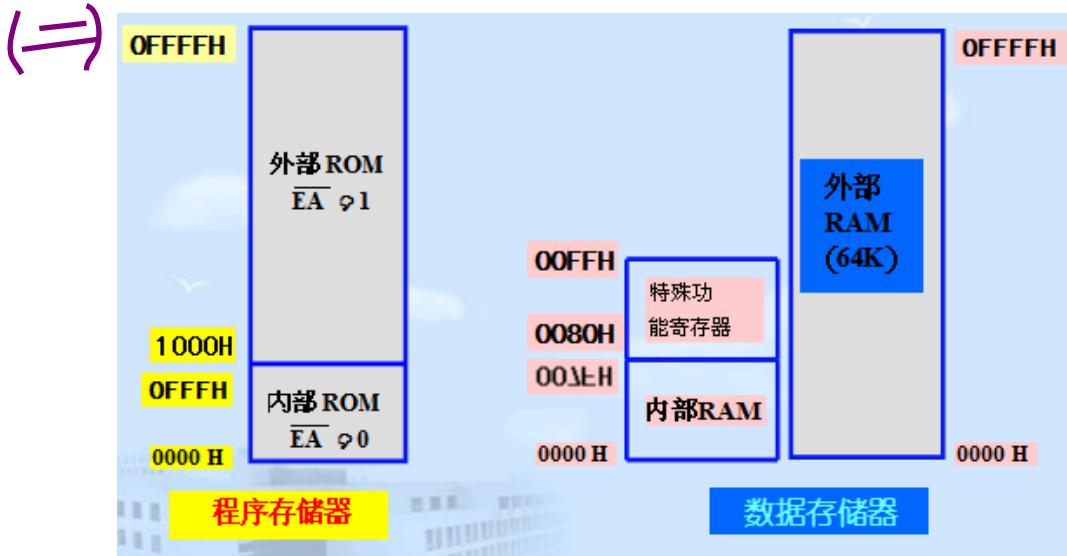
1. **中央处理器：**
中央处理器 (CPU) 是整个单片机的核心部件，是 8 位数据宽度的处理器，能处理 8 位二进制数据或代码，CPU 负责控制、指挥和调度整个单元系统协调的工作，完成 运算和控制输入输出功能等操作。
2. **数据存储器 (RAM)：**
8051 内部有 128 个 8 位用户数据存储单元和 128 个专用寄存器单元，它们是 统一编址的，(专用寄存器只能用于存放控制指令数据，用户只能访问，而不能用于存放用户数据。) 所以，用户能使用的 RAM 只有 128 个，可存放读写的数据，运算的中间结果或用户定义的字型表。
3. **程序存储器 (ROM)：**
8051 共有 4096 个 8 位掩膜 ROM，用于存放用户程序，原始数据或表格。
4. **定时/计数器 (ROM)：**
8051 有 两个 16 位的可编程定时/计数器，以实现定时或计数产生中断用于控制程序转向。
5. **并行输入输出 (I/O) 口：**
8051 共有 4 组 8 位 I/O 口 (P0、P1、P2 或 P3)，用于对外部数据的传输。
6. **全双工串行口：**
8051 内置一个 全双工串行通信口，用于与其它设备间的串行数据传送，该串行口既可以用作异步通信收发器，也可以当同步移位器使用。

7. 中断系统:

8051 具备较完善的中断功能，有两个外中断、两个定时/计数器中断和一个串行中断，可满足不同的控制要求，并具有 2 级的优先级别选择。

8. 时钟电路:

8051 内置最高频率达 12MHz 的时钟电路，用于产生整个单片机运行的脉冲时序，但 8051 单片机需外置振荡电容。



R1 被称之为工作寄存器。什么是工作寄存器呢？让我们从现实生活中来找找答案。如果出一道数学题： $123+567$ ，让你回答结果是多少，你会马上答出是 690，再看下面一道题： $123+567+562$ ，要让你马上回答，就不这么容易了吧？我们会怎样做呢？如果有张纸，就容易了，我们先算出 $123+567=690$ ，把 690 写在纸上，然后再算 $690+562$ 得到结果是 1552。这其中 1552 是我们想要的结果，而 690 并非我们所要的结果，但是为了得到最终结果，我们又不得不先算出 690，并记下来，这其实是一个中间结果，计算机中做运算和这个类似，为了要得到最终结果，往往要做很多步的中间结果，这些中间结果要有个地方放才行，把它们放哪呢？放在前面提到过的 ROM 中可以吗？显然不行，因为计算机要将结果写进去，而 ROM 是不可以写的，所以在单片机中另有一个区域称为 RAM 区（RAM 是随机存取存储器的英文缩写），它可以将数据写进去。

四. 指令

1. 寻址

MOV A, Rn ;寄存器寻址

MOV A, direct ;直接寻址

MOV A, @Ri ;间接寻址

MOV A, #data ;立即数

2. 数据传递指令

(1) 与片内 RAM

MOV

(2) 累加器 A 与片外 RAM 之间的数据传递类指令

MOVX A, @Ri

MOVX @Ri, A

MOVX A, @DPTR

MOVX @DPTR, A

(3) 程序存储器向累加器 A 传送指令

MOVC A, @A+DPTR

例：有一个数在 R0 中，要求用查表的方法确定它的平方值（此数的取值范围是 0-5）

MOV DPTR, #TABLE(ROM 中 table 的地址值)

MOV A, R0

MOVC A, @A+DPTR

.

TABLE:DB 0,1,4,9,16,25

设 R0 中的值为 2，送入 A 中，而 DPTR 中的值则为 TABLE，则最终确定的 ROM 单元的地址就是 TABLE+2，也就是到这个单元中去取数，取到的是 4，显然它正是 2 的平方。其它数据也可以类推。

(4) 堆栈操例：

```
MOV SP, #5FH
```

```
MOV A, #100
```

```
MOV B, #20
```

```
PUSH ACC
```

```
PUSH B
```

则执行第一条 PUSH ACC 指令是这样的：

- 1) 将 SP 中的值加 1，即变为 60H，
- 2) 然后将 A 中的值送到 60H 单元中。

因此执行完本条指令后，内存 60H 单元的值就是 100，同样，执行 PUSH B 时，是将 SP+1，即变为 61H，然后将 B 中的值送入到 61H 单元中，即执行完本条指令后，61H 单元中的值变为 20。

POP 指令的执行是这样的：

- 1) 首先将 SP 中的值作为地址，并将此地址中的数送到 POP 指令后面的那个 direct 中
- 2) 然后 SP 减 1。

3. 位寻址区

(1)在 8031 中，有一部份 RAM 和一部份 SFR是具有位寻址功能的，也就是说这些 RAM 的每一个位都有自己的地址，可以直接用这个地址来对此进行操作。

字节地址	位地址								
2FH	7FH								78H
2EH	77H								70
2DH	6FH								68H
2CH	67H								60H
2BH	5FH								58H
2AH	57H								50H
29H	4FH								48H
28H	47H								40H
27H	3FH								38H
26H	37H								30H
25H	2FH								28H
24H	27H								20H
23H	1FH								18H
22H	17H								10H
21H	0FH								08H
20H	07H	06H	05H	04H	03H	02H	01H		00H

内部 RAM 的 20H-2FH 这 16 个字节，就是 8031 的位寻址区。看图 1。可见这里的每一个 RAM 中的每个位我们都可能直接用位地址来找到它们，而不必用字节地址，然后再用逻辑指令的方式。

(2) 可以位寻址的特殊功能寄存器

位寻址的方法：`sbit P10=P1^0;`

8031 中有一些 SFR 是可以进行位寻址的，这些 SFR 的特点是其字节地址均可被 8 整除，如 A 累加器，B 寄存器、PSW、IP（中断优先级控制寄存器）、IE（中断允许控制寄存器）、SCON（串行口控制寄存器）、TCON（定时器/计数器控制寄存器）、P0-P3（I/O 端口锁存器）。以上的一些 SFR 我们还不熟，等我们讲解相关内容时再作详细解释。

五. 计数器与定时器

1. 介绍

8031 单片机中有两个计数器，分别称之为 T0 和 T1，这两个计数器分别是由两个 8 位的 RAM 单元组成的，即每个计数器都是 16 位的计数器，最大的计数量是 65536

2. 单片机中的定时器和计数器是一个东西

(1) 只不过 计数器 是记录的外界发生的事情

(2) 而 定时器 则是由单片机提供一个非常稳定的计数源。

3. 提供定时器的是计数源是什么呢？

原来就是由单片机的 晶振经过 12 分频后获得的一个脉冲源。晶振的频率当然很准，所以这个计数脉冲的时间间隔也很准。问题：一个 12M 的晶振，它提供给计数器的脉冲时间间隔是多少呢？当然这很容易，就是 $12\text{M}/12$ 等于 1M，也就是 1 个微秒。

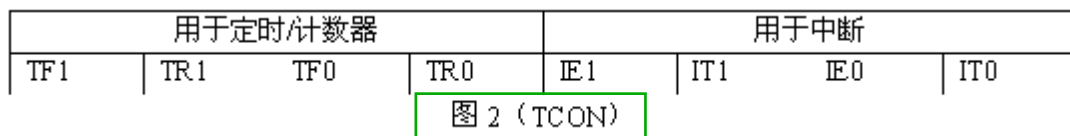
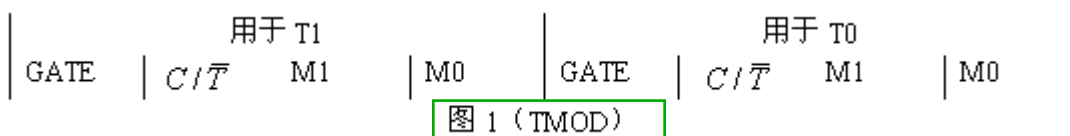
4. 计数器溢出

水溢出是流到地上，而计数器溢出后将使得 TF0 变为“1”。至于 TF0 是什么我们稍后再谈。一旦 TF0 由 0 变成 1，就是产生了变化，产生了变化就会引发事件，就象定时的时间一到，闹钟就会响一样。

5. 方式控制字

单片机中的定时/计数器都可以有多种用途，那么我怎样才能让它们工作于我所需要的用途呢？这就要通过定时/计数器的方式控制字来设置。

在单片机中有两个特殊功能寄存器与定时/计数有关，这就是 TMOD 和 TCON



1. **M1M0**: 定时/计数器一共有四种工作方式，就是用 M1M0 来控制的，2 位正好是四种组合。
2. **C/T**: 前面我们说过，定时/计数器即可作定时用也可用计数用，到底作什么用，由我们根据需要自行决定，也说是决定权在我们编程者。如果 C/T 为 0 就是用作定时器（开关往上打），如果 C/T 为 1 就是用作计数器（开关往下打）。顺便提一下：一个定时/计数器同一时刻要么作定时用，要么作计数用，不能同时用的，这是个极普通的常识，几乎没有教材会提这一点，但很多初学者却会有此困惑。
3. **GATE**: 看图，当我们选择了定时或计数工作方式后，定时/计数脉冲却不一定能到达计数器端，中间还有一个开关，显然这个开关不合上，计数脉冲就没法过去，那么开关什么时候过去呢？有两种情况

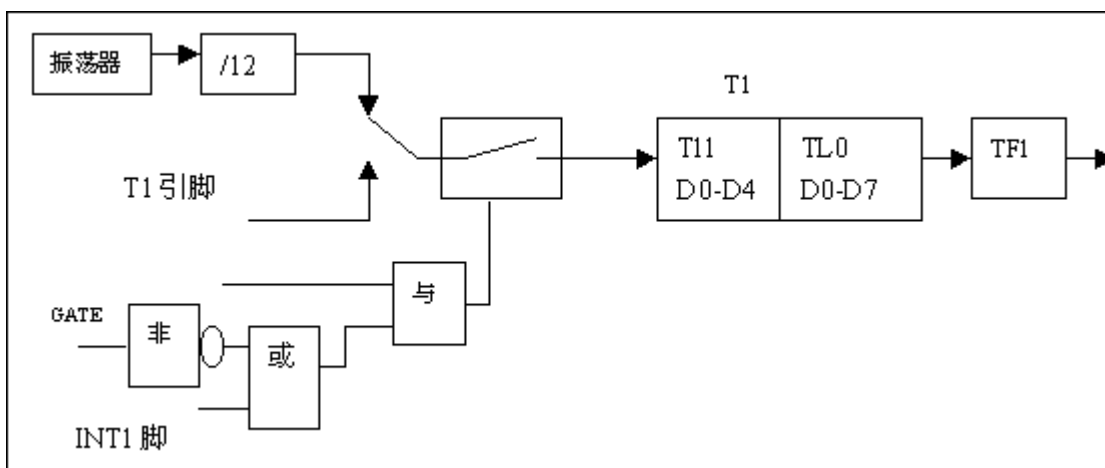
(1)GATE=0，分析一下逻辑，GATE 非后是 1，进入或门，或门总是输出 1，和或门的另一个输入端 INT0 无关，在这种情况下，开关的打开、合上只取决于 TR0，只要 TR0 是 1，开关就合上，计数脉冲得以畅通无阻，而如果 TR0 等于 0 则开关打开，计数脉冲无法通过，因此定时/计数是否工作，只取决于 TR0。

(2)GATE=1，在此种情况下，计数脉冲通路上的开关不仅要由 TR0 来控制，而且还要受到 INT0 引脚的控制，只有 TR0 为 1，且 INT0 引脚也是高电平，开关才合上，计数脉冲才得以通过。这个特性可以用来测量一个信号的高电平的宽度，想想看，怎么测？

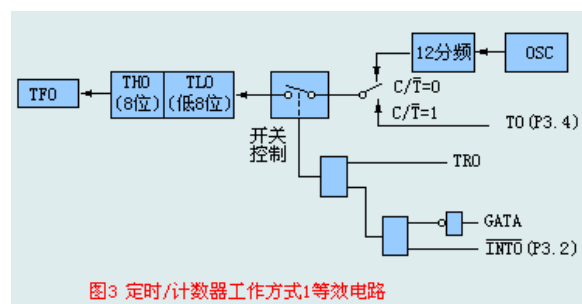
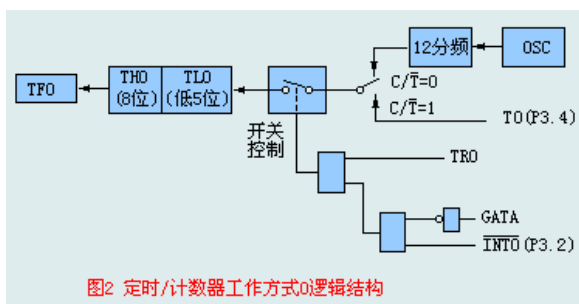
6. 计数脉冲要进入计数器

有层层关要通过，最起码，就是 TR0 (1) 要为 1，开关才能合上，脉冲才能过来。因此，TR0

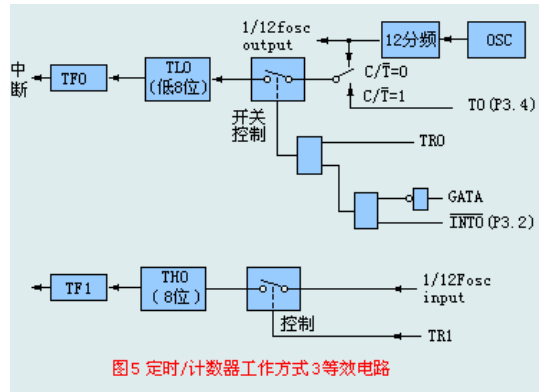
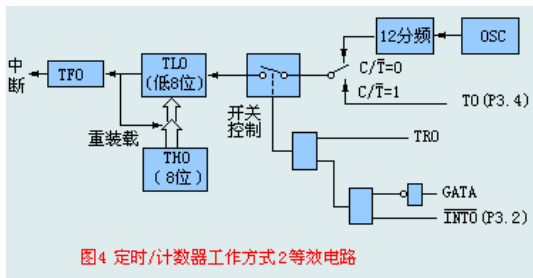
(1) 称之为运行控制位，可用指令 SETB 来置位以启动计数器/定时器运行，用指令 CLR 来关闭定时/计数器的工作，一切尽在自己的掌握中



7. 工作方式



(方式 1---最常用)



8. 利用定时器实现灯的闪烁

```
ORG 0000H
```

```
AJMP START
```

```
ORG 30H
```

```
START:
```

1.

```
MOV P1,#0FFH ;关所有灯
```

```
MOV TMOD,#00000001B ;定时/计数器 0 工作于方式 1
```

```
MOV TH0,#15H
```

```
MOV TL0,#0A0H ;即数 5536
```

```
SETB TR0 ;定时/计数器 0 开始运行
```

步骤：

- 1、设置TMOD：工作方式。
- 2、设置TH0,TL0：初值。
- 3、设置运行控制位：开始定时。

2.

```
LOOP:JBC TF0,NEXT ;如果 TF0 等于 1，则清 TF0 并转 NEXT 处
```

JBC bit,rel

若bit为1，则清bit，且跳转

```
AJMP LOOP ;否则跳转到 LOOP 处运行
```

3.

NEXT:CPL P1.0

MOV TH0,#15H

MOV TL0,#9FH;重置定时/计数器的初值

AJMP LOOP

END AJMP LOOP

END

六 . 并行口和串行口

0. 8051 单片机的通讯方式有两种:

并行通讯:数据的各位同时发送或接收。

串行通讯:数据一位一位顺序发送或接收。

1.并行通讯

(1)除了可以作为输出外,这 32 个引脚还可以做什么呢?下面再来做一个实验,程序如下:

MAIN: MOV P3, #0FFH

LOOP: MOV A, P3

MOV P1, A

LJMP LOOP

先看一下实验的结果:所有灯全部不亮,然后我按下一个按钮,第()个灯亮了,再按下另一个按钮,第()个灯亮了,松开按钮灯就灭了。从这个实验现象结合电路来分析一下程序。

P3 口确实起到了输入作用，这样，我们可以看到，以 P 字开头的管脚，不仅可以用作输出，还可以用作输入，其它的管脚是否可以呢？是的，都可以。这 32 个引脚就称之为并行口，下面我们就对并行口的结构作一个分析，看一下它是怎样实现输入和输出的。

(2)并行口结构分析:

[1]输出结构

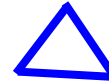
先看 P1 口的一位的结构示意图（只画出了输出部份）：从图中可以看出，开关的打开和合上代表了引脚输出的高和低，如果开关合上了，则引脚输出就是低，如果开关打开了，则输出高电平，这个开关是由一根线来控制的，这根数据总线是出自于 CPU，让我们回想一下，数据总线是一根大家公用的线，很多的器件和它连在一起，在不同的时候，不同的器件当然需要不同的信号，如某一时刻我们让这个引脚输出高电平，并要求保持若干时间，在这段时间里，计算机当然在忙个不停，在与其它器件进行联络，这根控制线上的电平未必能保持原来的值不变，输出就会发生变化了。怎么解决这个问题呢？我们在存储器一节中学过，存储器中是可以存放电荷的，我们不妨也加一个小的存储器的单元，并在它的前面加一个开关，要让这一位输出时，就把开关打开，信号就进入存储器的单元，然后马上关闭开关，这样这一位的状态就被保存下来，直到下一次命令让它把开关再打开为止。这样就能使这一位的状态与别的器件无关了，这么一个小单元，我们给它一个很形象的名字，称之为“**锁存器**”。

[2]输入结构

这是并行口的一位输出结构示意图，再看，除了输出之外，还有两根线，一根从外部引脚接入，另一根从锁存器的输出接出，分别标明读引脚和读锁存器。这两根线是用于从外部接收信号的，为什么要两根呢？原来，在 51 单片机中输入有两种方式，分别称为‘读引脚’和‘读锁存器’，第一种方式是将引脚作为输入，那是真正地从外部引脚读进输入的值，第二种方式是该引脚处于输出状态时，有时需要改变这一位的状态，则并不需要真正地读引脚状态，而只是读入锁存器的状态，然后作某种变换后再输出。

请注意输入结构图，如果将这一根引线作为输入口使用，我们并不能保证在任何时刻都能得到正确的结果（为什么？）参考图 2 输入示意图。接在外部的开关如果打开，则应当是输入 1，而如果闭合开关，则输入 0，但是，如果单片机内部的开关是闭合的，那么不管外部的开关是开还是闭，单片机接受到的数据都是 0。可见，要让这一端口作为输入使用，要先

做一个‘准备工作’，就是先让内部的开关断开，也就是让端口输出‘1’才行。正因为要先做这么一个准备工作，所以我们称之为“准双向 I/O 口”。



2. 串行通讯

(1) 通讯的方式

[1] 异步通讯

在一帧格式中，先是一个起始位 0，然后是 8 个数据位(规定低位在前，高位在后)，接下来是奇偶校验位（可以省略），最后是停止位 1。用这种格式表示字符，则字符可以一个接一个地传送。

波特率即数据传送的速率，其定义是每秒钟传送的二进制数的位数。例如，数据传送的速率是 120 字符/s，而每个字符如上述规定包含 10 数位，则传送波特率为 1200 波特。

[2] 同步通讯

在异步通讯中，每个字符要用起始位和停止位作为字符开始和结束的标志，占用了时间；所以在数据块传递时，为了提高速度，常去掉这些标志，采用同步传送。由于数据块传递开始要用同步字符来指示，同时要求由时钟来实现发送端与接收端之间的同步，故硬件较复杂。

(2) 引脚

8051 单片机通过引脚 **RXD (P3.0)**，串行数据接收端)

引脚 **TXD (P3.1)**，串行数据发送端) 与外界通讯

(3) 控制和状态寄存器

串行口控制寄存器 **SCON**

它用于定义串行口的工作方式及实施接收和发送控制。字节地址为 98H，其各位定义如下表：

D7	D6	D5	D4	D3	D2	D1	D0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

[1]SM0、SM1：串行口工作方式选择位，其定义如下：

SM0、SM1	工作方式	功能描述	波特率
00	方式0	8位移位寄存器	Fosc/12
01	方式1	10位UART	可变
10	<u>方式2</u>	<u>11位UART</u>	<u>Fosc/64 或 fosc/32</u>
11	方式3	11位UART	可变

其中 fosc 为晶振频率

[2]SM2：多机通讯控制位

在方式0时，SM2一定要等于0。

在方式1中，当(SM2)=1则只有接收到有效停止位时，RI才置1。

在方式2或方式3当(SM2)=1且接收到的第九位数据RB8=0时，RI才置1。

[3]REN：接收允许控制位

由软件置位以允许接收，又由软件清0来禁止接收。

[4]TB8：是要发送数据的第9位

在方式2或方式3中，要发送的第9位数据，根据需要由软件置1或清0。例如，可约定作为奇偶校验位，或在多机通讯中作为区别地址帧或数据帧的标志位。

[5]RB8：接收到的数据的第9位。

在方式 0 中不使用 RB8。在方式 1 中，若 (SM2) = 0，RB8 为接收到的停止位。在方式 2 或方式 3 中，RB8 为接收到的第 9 位数据。

[6] TI: 发送中断标志

在方式 0 中，第 8 位发送结束时，由硬件置位。在其它方式的发送停止位前，由硬件置位。TI 置位既表示一帧信息发送结束，同时也是申请中断，可根据需要，用软件查询的方法获得数据已发送完毕的信息，或用中断的方式来发送下一个数据。TI 必须用软件清 0。

[7] RI: 接收中断标志位

在方式 0，当接收完第 8 位数据后，由硬件置位。在其它方式中，在接收到停止位的中间时刻由硬件置位（例外情况见于 SM2 的说明）。RI 置位表示一帧数据接收完毕，可用查询的方法获知或者用中断的方法获知。RI 也必须用软件清 0。

[8] 特殊功能寄存器 PCON

PCON 是为了在 CMOS 的 80C51 单片机上实现电源控制而附加的。其中最高位是 SMOD。

(4) 常用波特率

下表列出了定时器 T1 工作于方式 2 常用波特率及初值。

常用波特率	F _{osc} (MHZ)	SMOD	TH1初值
19200	11.0592	1	FDH
9600	11.0592	0	FDH
4800	11.0592	0	FAH
2400	11.0592	0	F4h
1200	11.0592	0	E8h

3. 串行的异步通讯实例

```
org 0000H
```

AJMP START

ORG 30H

START:

mov SP,#5fh ;

mov TMOD,#20h ;T1: 工作模式 2 ;设置参数, 准备工作

mov PCON,#80h SMOD=1

mov TH1,#0FDH ;初始化波特率 (参见表一->查表得: 波特率=19200)

mov SCON,#50h Standard UART settings(0101,0000 工作方式 1)

MOV R0,#0AAH ;准备送出的数

SETB REN ;允许接收

SETB TR1 ;T1 开始工作

波特率的选择时:

1、首先考虑工作方式: 这里选择的是波特率可变的工作方式1。

2、然后设置:

- (1) TMOD-----让定时器1工作于方式1
- (2) PCON, TH1---选择波特率

WAIT:

MOV A,R0 ;发送

CPL A

MOV R0,A

MOV SBUF,A

发送

LCALL DELAY

JBC TI,WAIT1 ;如果 TI 等于 1, 则清 TI 并转 WAIT1(TI 由 0->1 表明发送完毕, 然后跳转, 开始接收)

发送完毕中断

AJMP WAIT

3.

有数据来了中断

WAIT1: JBC RI,READ ;如果 RI 等于 1[表可以接收], 则清 RI 并转 READ

AJMP WAIT1

READ: ;接收

MOV A,SBUF ;将取得的数送 P1 口 **接受**

MOV P1,A

LJMP WAIT

DELAY: ;延时子程序

MOV R7,#0ffH

DJNZ R7,\$

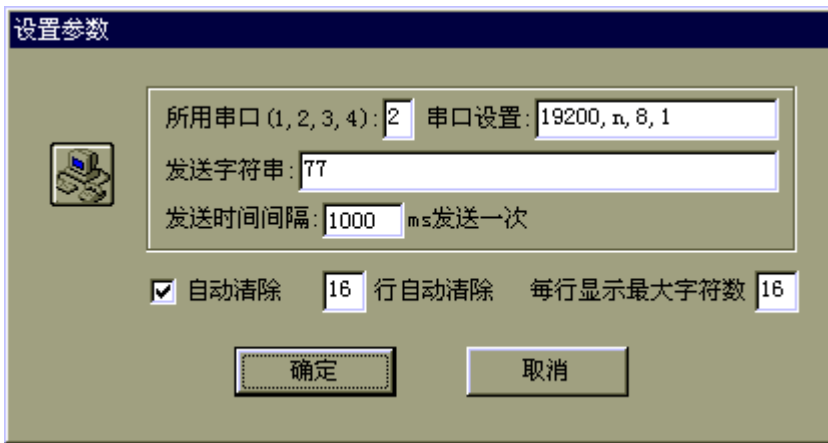
RET

END

将程序编译通过，写入芯片，插入实验板，用通读电缆将实验板与主机的串口相连就可以实验了。上面的程序功能很简单

就是每隔一段时间向主机轮流送数 55H 和 AAH，并把主机送去的数送到 P1 口。

可以在 PC 端用串口精灵来做实验。串口精灵在我主页上有下载。运行串口精灵后，按主界面上的“设置参数”按钮进入“设置参数”对话框，按下面的参数进行设置。注意，我的机器上用的是串口 2，如果你不是串口 2，请自行更改串口的设置。



设置完后，按确定返回主界面，注意右边有一个下拉列表，应当选中“按 16 进制”。然后按“开始发送”、“开始接收”就可以了。按此设置，实验板上应当有两只灯亮，6 只灯灭。大家可以自行更改设置参数中的发送字符如 55, 00, FF 等等，观察灯的亮灭，并分析原因，也可以在主界面上更改下拉列表中的“按 16 进制”为“按 10 进制”或“按 ASCII 字符”来观察现象，并仔细分析。这对于大家理解 16 进制、10 进制、ASCII 字符也是很有好处的。程序本身很简单，又有注释，这里就不详加说明了。

7. 中断系统

1. 与中断有关的寄存器

(1)总结

一. 中断请求源:

1. 外中断:IT0, IE0.. IT1, IE1
2. 定时器:TF0.. TF1
3. 串行口:TI, RI

二. 中断允许寄存器IE:

1. 外中断:EX0, EX1
2. 定时器:ET0, ET1
3. 串行口:ES
4. 总:EA

三. 中断优先级IP:

1. 外中断:PX0, PX1
2. 定时器:PT0, PT1
3. 串行口:PS

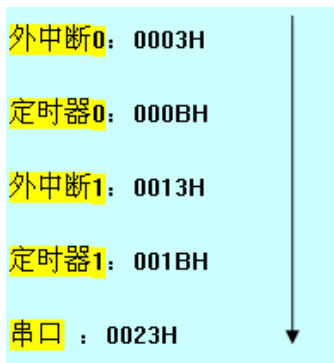
(2)说明:

前面说到的 TCON 中的

IT0: INT0 触发方式控制位，可由软件进和置位和复位，IT0=0, INT0 为低电平触发方式，IT0=1, INT0 为负跳变触发方式。这两种方式的差异将在以后再谈。

IE0: INT0 中断请求标志位。当有外部的中断请求时，这位就会置 1（这由硬件来完成），在 CPU 响应中断后，由硬件将 IE0 清 0。

2. 中断入口地址



写到这里，大家应当明白，为什么前面有一些程序一开始我们这样写：

```
ORG 0000H
```

```
LJMP START
```

```
ORG 0030H
```

```
START:
```

这样写的目的，就是为了让出中断源所占用的向量地址。

3. 中断响应过程

(1)CPU 响应中断时，首先把当前指令的下一条指令（就是中断返回后将要执行的指令）的地址送入堆栈

(2)然后根据中断标记，将相应的中断入口地址送入 PC，所以程序就会转到中断入口处继续执行。这些工作都是由硬件来完成的，不必我们去考虑。

(3)这里还有个问题，大家是否注意到，每个中断向量地址只间隔了 8 个单元，如 0003—000B，在如此少的空间中如何完成中断程序呢？很简单，你在中断处安排一个 LJMP 指令。

(4)一个完整的主程序看起来应该是这样的：

```
ORG 0000H
```

```
LJMP START
```

```
ORG 0003H
```

```
LJMP INT0 ; 转外中断 0
```

```
ORG 000BH
```

```
RETI ; 没有用定时器 0 中断，在此放一条 RETI，万一“不小心”产生了中断，不会有太大的后果。
```

中断程序完成后，一定要执行一条 RETI 指令，执行这条指令后，CPU 将会把堆栈中保存着的地址取出，送回 PC，那么程序就会从主程序的中断处继续往下执行了。

注意：CPU 所做的保护工作是很有限制的，只保护了一个地址，而其它的所有东西都不保护，所以如果你在主程序中用到了如 A、PSW 等，在中断程序中又要用它们，还要保证回到主程序后这里面的数据还是没执行中断以前的数据，就得自己保护起来。

4. 典型的中断服务程序

```
INT:      PUSH  PSW  ; 现场保护
          PUSH  ACC  ;
          _____
          中断处理程序段
          _____
          POP   ACC  ; 现场恢复
          POP   PSW
          _____
          RETI      ; 中断返回，恢复断点
```


5.中断实例

(1)定时器中断

```
ORG 0000H
```

```
AJMP START
```



```
ORG 000BH;定时器 0 的中断向量地址
```

```
AJMP TIME0;跳转到真正的定时器程序处
```

2.

```
ORG 30H
```

```
START:
```

```
MOV P1,#0FFH;关所 灯
```



```
MOV TMOD,#00000001B;定时/计数器 0 工作于方式 1
```

```
MOV TH0,#15H
```

```
MOV TL0,#0A0H;即数 5536
```

```
SETB EA;开总中断允许
```

```
SETB ET0;开定时/计数器 0 允许
```

```
SETB TR0;定时/计数器 0 开始运行
```

```
LOOP:
```

```
.....;真正工作时,这里可写任意程序
```

```
AJMP LOOP
```

3.

TIME0: ;定时器 0 的中断处理程序

PUSH ACC

PUSH PSW ;将 PSW 和 ACC 推入堆栈保护

CPL P1.0

MOV TH0,#15H

MOV TL0,#0A0H ;重置定时常数

POP PSW

POP ACC

RET

END

(2) 串行口中断

org 0000H

AJMP START

org 0023h

[串口中断入口地址]

AJMP SERIAL:

ORG 30H

START:

mov SP,#5fh ;

```
mov TMOD,#20h ;T1: 工作模式 2      [准备工作->设置参数]

mov PCON,#80h ;SMOD=1

mov TH1,#0FDH ;初始化波特率（参见表）

mov SCON,#50h ;Standard UART settings
```

```
MOV R0,#0AAH ; [准备送出的数]
```

```
SETB REN ;允许接收
```

```
SETB TR1 ;T1 开始工作
```

```
SETB EA ; [开总中断]

SETB ES ;开串口中断
```

SJMP \$

SERIAL:

```
MOV A,SBUF [接收]

MOV P1,A

CLR RI
```

RETI

```
END
```

(3)外中断

```
ORG 0000H
```

```
AJMP START
```

ORG 0003H ;外部中断地直入口

AJMP INT0

P3.2 的中断(INT0),

入口地址:0003H

需要设置:EX0

~~ORG 30H~~

START: MOV SP,#5FH

MOV P1,#0FFH ;灯全灭

MOV P3,#0FFH ;P3 口置高电平

SETB EA

SETB EX0

AJMP \$

INT0:

PUSH ACC

PUSH PSW

CPL P1.0

POP PSW

POP ACC

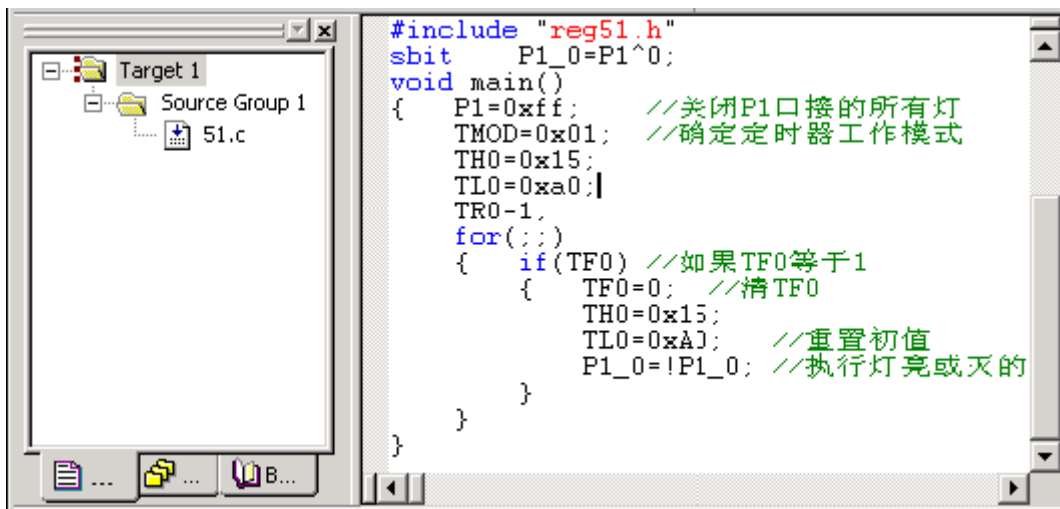
RETI

END

本程序的功能很简单，按一次按键 1（接在 12 引脚上的）就引发一次中断 0，取反一次 P1.0，因此理论上按一下灯亮，按一下灯灭，但在实际做实验时，可能会发觉有时不“灵”，按了它没反应，但在大部份时候是对的，这是怎么回事呢？我们在讲解键盘时再作解释，这个程序本身是没有问题的。

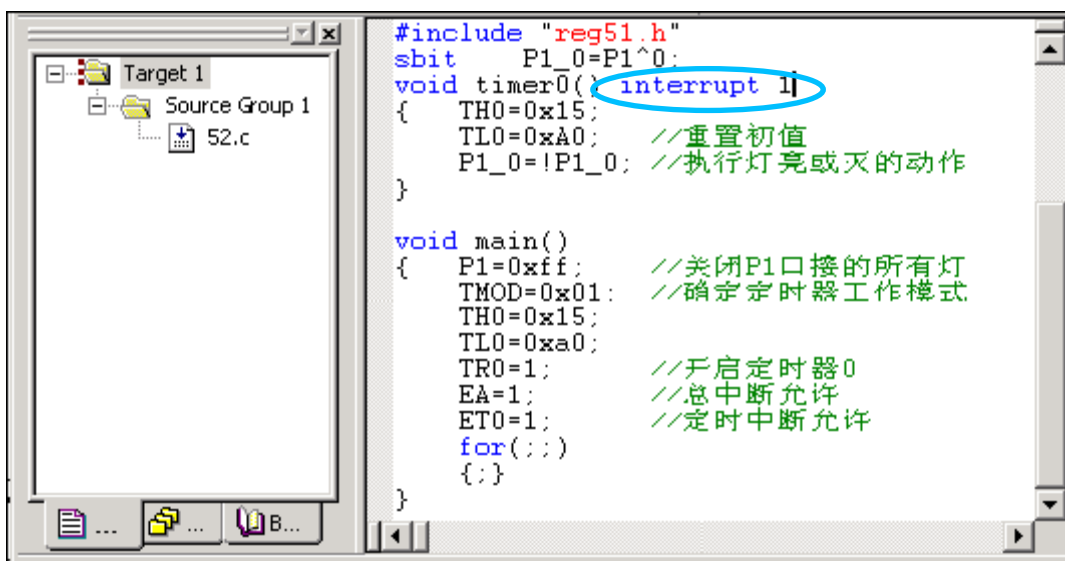
八. 内部资源的 C 编程

1. 定时器编程



```
#include "reg51.h"
sbit P1_0=P1^0;
void main()
{
    P1=0xff; //关闭P1口接的所有灯
    TMOD=0x01; //确定定时器工作模式
    TH0=0x15;
    TL0=0xa0;
    TR0=1;
    for(;;)
    {
        if(TF0) //如果TF0等于1
        {
            TF0=0; //清TF0
            TH0=0x15;
            TL0=0xa0; //重置初值
            P1_0=!P1_0; //执行灯亮或灭的
        }
    }
}
```

2. 中断编程



```
#include "reg51.h"
sbit P1_0=P1^0;
void timer0() interrupt 1
{
    TH0=0x15;
    TL0=0xa0; //重置初值
    P1_0=!P1_0; //执行灯亮或灭的动作
}

void main()
{
    P1=0xff; //关闭P1口接的所有灯
    TMOD=0x01; //确定定时器工作模式
    TH0=0x15;
    TL0=0xa0;
    TR0=1; //开启定时器0
    EA=1; //总中断允许
    ET0=1; //定时中断允许
    for(;;)
    {}
}
```

为进行中断的现场保护，80C51 单片机除采用堆栈技术外，还独特地采用寄存器组的方式，在80C51 中一共有4 组名称均为R0~R7 的工作寄存器，中断产生时，可以通过简单地设置RS0、RS1 来切换工作寄存器组，这使得保护工作非常简单和快速。

在C51 中，寄存器组选择取决于特定的编译器指令，即使用using n 指定，其中n 的值是0~3，对应使用四组工作寄存器。

例如上述例子中可以这么来写：

```
void timer0() interrupt 1 using 2
{
    ...
}
```

即表示在该中断程序中使用第2 组工作寄存器。

3. 串行口编程

```
#define uchar unsigned char
#include "string.h"
#include "reg51.h"
void SendData(uchar Dat)
{
    uchar i=0;
    SBUF=Dat;
    while(1){
        if(TI)
        {
            TI=0;
            break;
        }
    }
void mDelay(unsigned int DelayTime)
{
    unsigned char j=0;
    for(;DelayTime>0;DelayTime--)
    {
        for(j=0;j<125;j++)
        {
            ;
        }
    }
}
uchar Key()
{
    uchar KValue;
```

```
P3|=0x3e; //中间 4 位置高电平
if((KValue=P3|0xe3)!=0xff)
{
    mDelay(10);
    if((KValue=P3|0xe3)!=0xff)
    {
        for(;;)
            if((P3|0xe3)==0xff)
                return(KValue);
    }
}
return(0);
}

void main()
{
    uchar KeyValue;
    P1=0xff; //关闭 P1 口接的所有灯
    TMOD=0x20; //确定定时器工作模式
    TH1=0xFD;
    TL0=0xFD; //定时初值
    PCON&=0x80; //SMOD=1
    TR1=1; //开启定时器 1
    SCON=0x40; //串口工作方式 1
    REN=1; //允许接收
    for(;;)
    {
        if(KeyValue=Key())
        {
            if((KeyValue|0xfb)!=0xff) //K1 按下
                SendData(0x55);
            if((KeyValue|0xf7)!=0xff)
                SendData(0xaa);
        }
        if(RI)
            P1=SBUF;
            RI=0;
    }
}
```

