CHAPTER 3

3 第章

输入口及其应用

3.1 MCS-51 时序

3.1.1 MCS-51 时序定时单位

1. 时钟周期

时钟周期是单片机的基本时间单位。若时钟的晶体振荡频率为 f，则时钟周期 T 为晶体振荡频率的倒数，即 $T=1/f$。

2. 机器周期

在计算机中，为了便于管理，常把一条指令的执行过程划分为若干个阶段，每一阶段完成一项工作。例如，取指令、存储器读、存储器写等，每一项的工作都为一个基本操作。完成一个基本操作所需的时间称为一个机器周期。这是一个时间基准，好像人们用“秒”作为时间基准一样。

MCS-51 单片机的一个机器周期包括 12 个时钟周期。设一个单片机工作于 12MHz，它的时钟周期是 1/12μs。它的 1 个机器周期是 12×(1/12)，即 1μs。当单片机时钟频率为 6MHz 时，机器周期为 2μs。这是两个常用的数据，应该记住。

3. 指令周期

单片机的所有指令中，有一些完成得快，只要一个机器周期就行了；而有些完成得比较慢，要 2 个机器周期，还有两条指令要 4 个机器周期才能完成。也就是说它们所需的机器周期不相同，可能包括 1～4 个不等的机器周期，执行一条指令所需要的时间称为指令周期。每一条指令所需用的机器周期数是永远固定的，而且可以通过表格查到。

3.1.2 MCS-51 的指令时序

MCS-51 单片机执行任何一条指令时，都可以分为取指令阶段和执行指令阶段。取指令阶段简称取指阶段，单片机在这个阶段里可以把程序计数器 PC 中的地址送到程序存储器，并从中取出需要执行指令的操作码和操作数。指令执行阶段可以对指令操作码进行译码，以产生一系列控制信号完成指令的执行。

MCS-51 单片机的各种指令执行所需要的机器周期数目是不同的，主要有以下几种：单字节单机器周期指令、双字节单机器周期指令（如“ADDA，＃data”）、单字节双机器周期指令（如 INC DPTR）、双字节双机器周期指令（如位逻辑“与”和“或”类指令）、三字节双机器周期指令。

3.2 控制转移指令

控制转移指令共有 22 条。分为无条件转移（AJMP、LJMP、SJMP、JMP）指令，条件转移（JZ、JNZ、JC、JNC、JB、JNB、JBC、CJNE、DJNZ）指令，调用和返回（ACALL、LCALL、RET、RETI）指令，空操作（NOP）指令。

控制与转移指令中，除 CJNE 指令对 Cy 有影响外，其余指令都不影响标志。

控制与转移指令可改变程序计数器 PC 的值，从而使程序跳到指定的目的地址开始执行。

3.2.1 无条件转移指令

无条件转移指令如表 3.1 所示。

表 3.1 无条件转移指令

汇编指令	操作说明	代码长度（字节）	指令周期	
			T_{osc}	TM
LJMP addr16	长转移：程序转移到 addr16 指示的地址处，即 PC←addr16	3	24	2
AJMP addr16	绝对转移：程序转移到 addr16 指示的地址处，即 PC←PC＋2，PC15－11 不变，PC10－0←addr10－0	2	24	2
SJMP rel	短转移：程序转移到 rel 指示相对地址处，即 PC←PC＋2，PC←PC＋rel	2	24	2
JMP @A＋DPTR	间接长转移（程序散转）：程序转移到 DPTR 为基址加 A 偏移地址处，即 PC←A＋DPTR	1	24	2

程序执行无条件转移指令时，就无条件地转移到目的地址。

1. 长转移指令：LJMP addr16

指令的操作是将 16 位目标地址 addr16 装入 PC 中，允许转移的目标地址在 64KB 空间的任意单元。用汇编语言编写程序时，addr16 经常是一个标号。

2. 绝对转移指令：AJMP addr11

指令的操作是将 11 位的目标地址 addr11 装入 PC 中的低 11 位。要求目标地址的高

5 位与 PC+2 后 PC 中的高 5 位相同。即转移的目标地址必须和 AJMP 指令的下一条指令首字节地址位于程序存储器的同一段 2KB 字节范围内。编写程序时，addr11 也经常是一个标号。

3. 短转移指令：SJMP rel

指令中相对偏移量 ret 为 8 位的补码，将其符号扩展为 16 位后与 PC 相加得到 16 位的目标地址。转移的范围为－128～＋127 字节。编写程序时，ret 同样往往是一个标号。

MCS-51 没有专用的停机指令，若要原地踏步等待，可以用 SJMP 指令来实现。原地踏步等待指令为：

```
LP1: SJMP  LP1
```

或写成：

```
SJMP  $
```

$ 表示本指令首字节所在单元的地址。

4. 间接长转移指令 JMP @A＋DPTR

转移目标地址由数据指针 DPTR 和累加器 A(8 位无符号数)相加而得。指令的执行不影响累加器 A 和数据指令 DPTR。该指令的特点是转移地址可以在程序运行中加以改变。例如，DPTR 作为基地址。根据 A 的不同值可以实现多分支转移，因此一条指令可以完成多分支转移的功能，称为散转功能。间接长转移指令又称为散转指令。

3.2.2 条件转移指令

条件转移指令的操作是判断指定的条件，如果条件满足则转移，不满足则顺序执行。

条件转移指令共有 13 条，可分为判断 A 是否为零转移(JZ、JNZ)指令，位条件转移(JC、JNC、JB、JNB、JBC)指令，比较不等转移(CJNE)指令，减 1 循环(DJNZ)转移指令。

1. 判断累加器是否为零转移指令(2 条)

判断累加器是否为零转移指令如表 3.2 所示。

表 3.2 判断累加器是否为零转移指令

汇编指令	操作说明	代码长度(字节)	指令周期	
			T_{osc}	TM
JZ rel	A 值为零时，程序转移到相对地址 rel 处： 若(A)≠0，则 PC←(PC)＋2； 若(A)＝0，则 PC←(PC)＋2＋rel	2	24	2
JNZ rel	A 值不为零时，程序转移到相对地址 rel 处： 若(A)＝0，则 PC←(PC)＋2； 若(A)≠0，则 PC←(PC)＋2＋rel	2	24	2

例 1：使用 JZ 指令编程实现内部 RAM 30H 中数据连续加 1，直到 FFH 结束。

```
      ORG   0000H
LOOP: INC   30H
```

```
        MOV   A,30H
        CPL   A
        JNZ   LOOP                    ;A 不等于 0,继续加 1
        END                           ;A 等于 0,顺序执行
```

2. 位条件转移指令(5 条)

位条件转移指令如表 3.3 所示。

表 3.3 位条件转移指令

汇编指令	操作说明	代码长度(字节)	指令周期	
			T_{osc}	TM
JC rel	进位位 Cy 为 0 时,程序转移至 rel: 若 Cy=0,则 PC←(PC)+2; 若 Cy=1,则 PC←(PC)+2+rel	2	24	2
JNC rel	进位位为 0 时,程序转移至 rel 处: 若 Cy=0,则 PC←(PC)+2+rel; 若 Cy=1,则 PC←(PC)+2	2	24	2
JB bit, rel	Bit 位为 1 时,程序转移至 rel 处: 若(bit)=0,则 PC←(PC)+3; 若(bit)=1,则 PC←(PC)+3+rel	3	24	2
JNB bit, rel	Bit 位为 0 时,程序转移至 rel 处: 若(bit)=0,则 PC←(PC)+3+rel; 若(bit)=1,则 PC←(PC)+3	3	24	2
JBC bit, rel	Bit 位为 1 时,程序转移至 rel 处,同时将 bit 清零: 若(bit)=0,则 PC←(PC)+3; 若(bit)=1,则(bit)←0,PC←(PC)+3+rel	3	24	2

例 2:使用 JC 指令编程实现如下功能:将内部 RAM 30H 中的数据连续减 1,直到遇到 0 结束。

```
        ORG   0000H
        MOV   R0,#30H
LOOP:   MOV   A, @R0
        SUBB  A, #1
        MOV   @R0, A
        JNC   LOOP                    ;C 不等于 0,继续循环
        END                           ;C 等于 0,则顺序执行
```

3. 比较不等转移指令(4 条)

比较不等转移指令如表 3.4 所示。

表 3.4 比较不等转移指令

汇编指令	操作说明	代码长度(字节)	指令周期	
			T_{osc}	TM
CJNE A, #data, rel	#data 与 A 内容不等时转至 rel 处,同时影响 Cy 位: 若(A)=data,则 PC←(PC)+3,Cy←0; 若(A)>data,则 PC←(PC)+3+rel,Cy←0; 若(A)<data,则 PC←(PC)+3+rel,Cy←1	3	24	2

续表

汇编指令	操作说明	代码长度（字节）	指令周期	
			T_{osc}	TM
CJNE A, direct, rel	direct 值与 A 值不等时转至 rel 处，同时影响 Cy 位： 若(A)=(direct)，则 PC←(PC)+3，Cy←0； 若(A)>(direct)，则 PC←(PC)+3+rel，Cy←0； 若(A)<(direct)，则 PC←(PC)+3+rel，Cy←1	3	24	2
CJNE Rn, #data, rel	#data 与 Rn 值不等时转至 rel 处，同时影响 Cy 位： 若(Rn)=data，则 PC←(PC)+3，Cy←0； 若(Rn)>data，则 PC←(PC)+3+rel，Cy←0； 若(Rn)<data，则 PC←(PC)+3+rel，Cy←1	3	24	2
CJNE @Ri, #data, rel	data 与@Ri 值不等时转至 rel 处，同时影响 Cy 位： 若(Ri)=data，则 PC←(PC)+3，Cy←0； 若(Ri)>data，则 PC←(PC)+3+rel，Cy←0； 若(Ri)<data，则 PC←(PC)+3+rel，Cy←1	3	24	2

比较不等转移指令的功能是比较两个数，若两者不相等则转移，若相等则顺序执行。如果第二个操作数(无符号数)大于第一个操作数(无符号数)，则 Cy 置 1，否则 Cy 清零。指令的执行不影响操作数。

例 3：用 CJNE 指令编程实现将内部 RAM 30H 开始的 16 个单元数据取反。

```
        ORG   0000H
        MOV   R0,#30H
        MOV   40H,#10H                ;置数据长度
LOOP:   MOV   A,@R0
        CPL   A
        INC   R0
        CJNE  R0,#40H,LOOP            ;(40H)不等于 0,循环
        END                           ;(40H)等于 0,顺序执行
```

4. 减 1 循环指令

减 1 循环指令如表 3.5 所示。

表 3.5 减 1 循环指令

汇编指令	操作说明	代码长度（字节）	指令周期	
			T_{osc}	TM
DJNZ Rn,rel	Rn 值先减 1，即 Rn←Rn−1； 若 Rn 不为零，则 PC←PC+2+rel，程序转移到相对地址 rel 处； 若 Rn=0，则 PC←PC+2，按原顺序向下执行	2	24	2
DJNZ direct,rel	direct 值先减 1，即 direct←direct−1； 若 direct 不为零，则 PC←PC+3+rel，程序转移到相对地址 rel 处； 若 direct=0，则 PC←PC+3，按原顺序向下执行	3	24	2

例 4：用 DJNZ 指令编程实现将内部 RAM 30H 开始的 16 个单元置 1。

```
        ORG   0000H
```

```
        MOV   R0,#30H
        MOV   40H,#10H                ;置数据长度
        MOV   A,#01H
LOOP:   MOV   @R0, A
        INC   R0
        DJNZ  40H,LOOP                ;(40H)不等于 0,循环
        END                           ;(40H)等于 0,顺序执行
```

3.2.3 调用和返回指令

在程序设计中,通常将反复出现、具有通用性和功能相对独立的程序段设计成子程序。子程序可以有效地缩短程序长度、节约存储空间;可被其他程序共享以及便于模块化,便于阅读、调试和修改。

子程序调用如图 3.1 所示。子程序调用指令的组成和操作说明如表 3.6 所示。

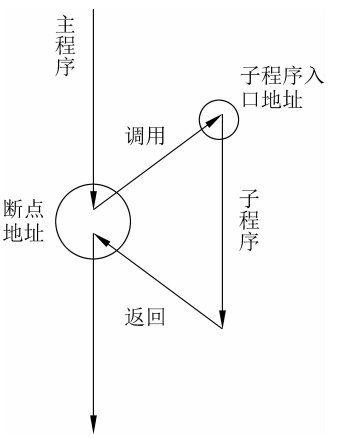

图 3.1 子程序调用

(1) 长调用指令的目标地址以 16 位给出,允许子程序放在 64KB 空间的任何地方。指令的执行过程是把 PC 加上本指令代码数(三个字节)获得下一条指令的地址,并把该断点地址入栈(断点地址保护),接着将被调子程序的入口地址(16 位目标地址)装入 PC,然后从该入口地址开始执行子程序。

(2) 绝对调用指令的执行过程是:PC 加 2(本指令代码为两个字节)获得下一条指令的地址,并把该断点地址(当前的 PC 值)入栈,然后将断点地址的高 5 位与 11 位目标地址(指令代码第一字节的高 3 位,以及第二字节的 8 位)连接构成 16 位的子程序入口地址,使程序转向子程序。调用子程序的入口地址和 ACALL 指令的下一条指令的地址,其高 5 位必须相同。因此子程序的入口地址和 ACALL 指令下一条指令的第一个字节必须在同一个 2KB 范围的程序存储器空间内。

表 3.6 调用和返回指令

汇编指令	操作说明	代码长度(字节)	指令周期	
			T_{osc}	TM
LCALL addr16	长调用:程序调用 addr16 处的子程序。 PC←PC+3 断点入栈: PC←addr16	3	24	2
ACALL addr11	绝对调用:程序调用 addr11 处的子程序。 PC←PC+2 断点入栈: PC←addr11	2	24	2
RET	子程序返回	1	24	2
RETI	中断返回	1	24	2

(3) 返回指令的功能是恢复断点地址,即从堆栈中取出断点地址送给 PC,因此在子程序、中断服务程序内使用堆栈时要特别小心,一定要确认执行返回指令时,SP 指向的是断点地址,否则程序将出错。

子程序是通过 RET 指令返回主程序。

中断服务子程序是通过 RETI 指令返回。执行 RETI 指令时,清除响应中断时置位的优先级状态触发器以开放中断逻辑等,详细内容将在第 5 章中讨论。

例 5:已知标号 LP1 的地址为 0300H,子程序 TIME1 的入口地址为 0100H,SP=70H。执行调用指令:

```
LP1: ACALL  TIME1
```

指令执行后:

```
SP=72H,(71H)=02H,(72H)=03H, PC=0100H
```

例 6:已知标号 LOOP 的地址为 268EH,子程序 TIME2 的入口地址为 0206H,SP=6AH。执行调用指令:

```
LOOP: LCALL TIME2
```

指令执行后:

```
SP=6CH,(6BH)=91H,(6CH)=26H,PC=0206H
```

例 7:已知 SP=78H,(78H)=46H,(77H)=8BH。执行指令:

```
RET
```

指令执行后:

```
SP=76H,PC=468BH
```

即 CPU 从 468BH 处开始执行程序,子程序必须通过 RET 指令返回主程序(通常情况下,子程序都以 RET 指令结束,但一个子程序也可以有多条 RET 指令)。

3.2.4 空操作指令

空操作指令如表 3.7 所示。

表 3.7 空操作指令

汇编指令	操作说明	代码长度(字节)	指令周期	
			T_{osc}	TM
NOP	空操作:PC←PC+1	1	12	1

空操作指令不进行任何操作。执行空操作指令时,PC 加 1 指向下一条指令,占用 CPU 一个机器周期时间。NOP 指令常用于等待、延时等程序的微调。

3.3 输入口应用实例

3.3.1 信号灯电路

已知电路如图 3.2 所示(注：电路未全部画出)，当 P1.2～P1.4 为高电平时，所接指示灯熄灭，要求实现如下功能：

(1) J1 单独按下，红灯亮，其余灯灭。

(2) J2 单独按下，绿灯亮，其余灯灭。

(3) 其余情况，黄灯亮。

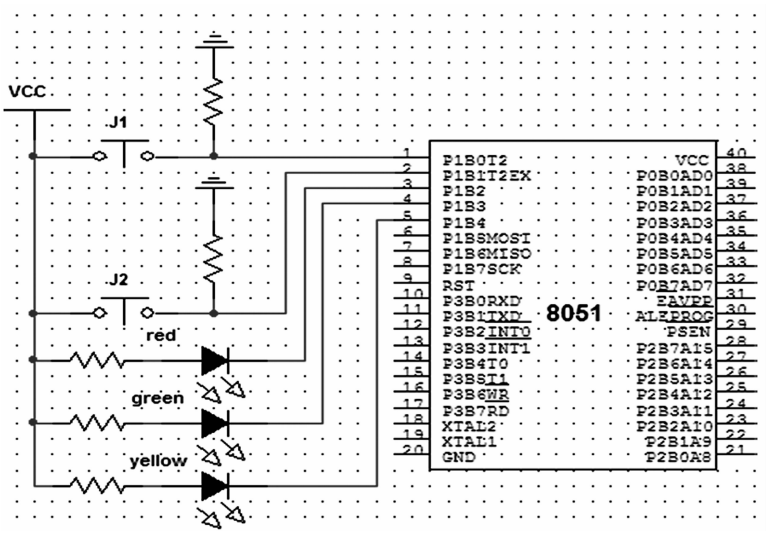

图 3.2 信号灯电路

解：将 P1.0、P1.1 的状态读到寄存器，再使用 CJNE 指令判断，则比较清晰。程序如下：(MOV C，bit 指令的用法请参考第 4 章的内容)。程序流程图如图 3.3 所示。

```
        ORG   0
        MOV   P1,#0FFH
START:  MOV   20H,#000000011B
        MOV   C,P1.0
        MOV   20H.0,C                ;读 P1.0 的状态
        MOV   C,P1.1
        MOV   20H.1,C                ;读 P1.1 的状态
        MOV   R0,20H
SL0:    CJNE  R0,#00000010B,SL1      ;P1.0、P1.1 均为 10,红灯亮
RED:    SETB  P1.2
        CLR   P1.3
        CLR   P1.4
        AJMP  START
SL1:    CJNE  R0,#00000001B,YELW     ;P1.0、P1.1 均为 01,绿灯亮
```

```
GREN:   SETB  P1.3
        CLR   P1.2
        CLR   P1.4
        AJMP  START
YELW:   SETB  P1.4                    ;P1.0、P1.1 均为 11 或 00,黄灯亮
        CLR   P1.2
        CLR   P1.3
        AJMP  START
        END
```

说明：0FFH＝FFH 中，之所以在 FFH 前加 0，是为了便于仿真软件识别。仿真软件要求凡是用英文字母开头的地址或数据，在英文字母前要加 0。

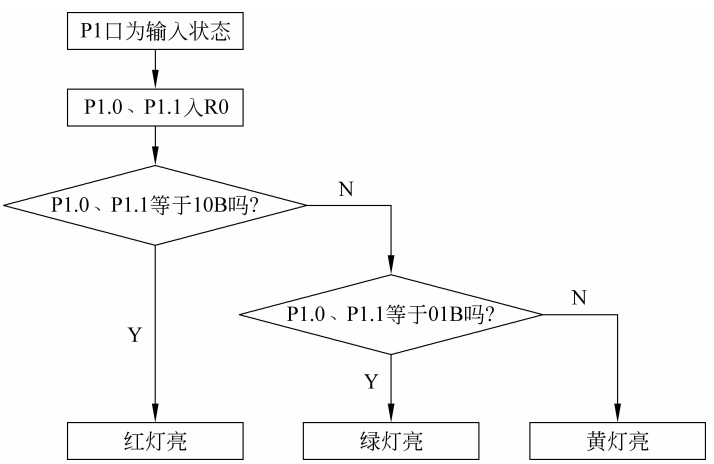

图 3.3 程序流程图

本例也可以使用 JB、JNB 指令，但是程序逻辑关系较复杂，程序清单如下：

```
        ORG   0;
        MOV   P1,#0FFH                ;红绿黄灯全灭,置 P1.0、P1.1 为输入状态
LOOP1:  JB    P1.0,LOOP2              ;P1.0=1,J1 按下,跳转 LOOP2,判断 J2
        JB    P1.1,GREEN              ;P1.0=0,J1 未按下,且 P1.1=1,J2 按下,则绿灯亮
YELLOW: CLR   P1.4                    ;黄灯亮
        SETB  P1.2                    ;红灯灭
        SETB  P1.3                    ;绿灯灭
        SJMP  LOOP1                   ;转循环 LOOP1,继续判断 J1、J2 开关状态
LOOP2:  JB    P1.1,YELLOW             ;P1.0=P1.1=1,J1、J2 均按下,则转黄灯亮
RED:    CLR   P1.2                    ;红灯亮
        SETB  P1.3                    ;绿灯灭
        SETB  P1.4                    ;黄灯灭
        SJMP  LOOP1                   ;转循环 LOOP1,继续判断 J1、J2 开关状态
GREEN:  CLR   P1.3                    ;绿灯亮
        SETB  P1.2                    ;红灯灭
        SETB  P1.4                    ;黄灯灭
        SJMP  LOOP1                   ;转循环 LOOP1,继续判断 J1、J2 开关状态
        END
```

3.3.2 BCD 译码器驱动

BCD 码～七段 LED 译码器(BCD-to-7-Segment Decoder)。

用途如下：BCD 码转换成 7 段 LED 数码管的驱动芯片，常用的有 7447 系列(见图 3.4)，如 7447、7448、7449 等，其中 7447 用以推动共阳极 7 段 LED 数码管，输出低电平，其真值表如表 3.8 所示。各脚功能如下。

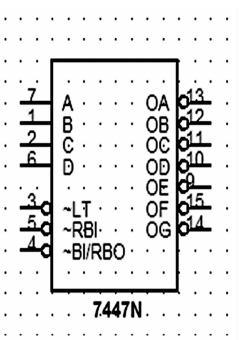

图 3.4 7447 驱动芯片

- D、C、B、A：BCD 码输入脚。
- a,b,c,d,e,f,g：7 段 LED 数码管输出脚。
- LT：测试脚。当输入低电平时，所连接的数码管全亮，正常显示时，该引脚应接高电平。
- RBI：正常显示时，接高电平；当 D、C、B、A 引脚输入为 0 时，若本引脚接低电平，则该数码管不显示。用于控制不需要显示的 0，如 016，最左边的 0 是不必显示的；9.850 中最右边的 0 也是没必要显示的。
- BI/RBO：正常显示时，应接高电平；若本引脚接低电平，则该数码管不显示。

表 3.8 7447 真值表

十进制数或功能	INPUT							OUTPUT								注释
	LTN	RBIN	D	C	B	A	BIN	A	B	C	D	E	F	G	RBON	
0	1	1	0	0	0	0	1	0	0	0	0	0	0	1	1	1
1	1	X	0	0	0	1	1	1	0	0	1	1	1	1	1	
2	1	X	0	0	1	0	1	0	0	1	0	0	1	0	1	
3	1	X	0	0	1	1	1	0	0	0	0	1	1	0	1	
4	1	X	0	1	0	0	1	1	0	0	1	1	0	0	1	
5	1	X	0	1	0	1	1	0	1	0	0	1	0	0	1	
6	1	X	0	1	1	0	1	1	1	0	0	0	0	0	1	
7	1	X	0	1	1	1	1	0	0	0	1	1	1	1	1	
8	1	X	1	0	0	0	1	0	0	0	0	0	0	0	1	
9	1	X	1	0	0	1	1	0	0	0	1	1	0	0	1	
10	1	X	1	0	1	0	1	1	1	1	0	0	1	0	1	
11	1	X	1	0	1	1	1	1	1	0	0	1	1	0	1	
12	1	X	1	1	0	0	1	1	0	1	1	1	0	0	1	
13	1	X	1	1	0	1	1	0	1	1	0	1	0	0	1	
14	1	X	1	1	1	0	1	1	1	1	0	0	0	0	1	
15	1	X	1	1	1	1	1	1	1	1	1	1	1	1	1	
BI	X	X	X	X	X	X	0	1	1	1	1	1	1	1	X	2
FBI	1	0	0	0	0	0	X	1	1	1	1	1	1	1	0	3
LT	0	X	X	X	X	X	1	0	0	0	0	0	0	0	1	4