8051单片机

C语言

软件设计的艺术

赖麒文 编著

科学出版社 文魁资讯股份有限公司

8051 单片机 C 语言
软件设计的艺术

赖麒文　编著

科学出版社

2002

内 容 简 介

本书主要介绍了 8051 单片机 C 语言软件设计的思维与解决方法。本书每一章都是一个精彩的例子，范例说明深入浅出。重点介绍软件的设计流程、软件的构思和解决方法。在实例中说明模块化程序设计的各种指令的应用，使用户可以更有效地学习。

本书适合于从事 8051 单片机应用设计的人员参考使用。

图字：01-2001-4800 号

图书在版编目（CIP）数据

8051 单片机 C 语言软件设计的艺术/赖麒文编著.—北京：科学出版社，2002

ISBN 7-03-010410-2

Ⅰ.8… Ⅱ.赖… Ⅲ.C 语言—软件设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2002）第 028788 号

科学出版社 出版

北京东黄城根北街16号

邮政编码：100717

http://www.sciencep.com

新蕾印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2002 年 6 月第 一 版 开本：720×1000 1/16

2002 年 6 月第一次印刷 印张：42

印数：1—5 000 字数：836 000

定价：58.00 元

（如有印装质量问题，我社负责调换〈路通〉）

导　读

第 1 章：介绍如何输出方波信号，使喇叭发出声音的方法，包括发出“哔”声的函数和分别传递一个、二个及三个自变量的“哔”声函数，以及利用定时器产生方波信号而令喇叭发出“哔”声，并叙述音阶与频率的关系，以此作为演奏音乐的基础。

第 2 章：演奏音乐的程序由 main( )函数开始，将其所有函数定义在一个 main.c 的模块内，并分别以各种指令结构来循序渐进地介绍软件构建的思维与解决方法。

第 3 章：以模块化的设计方式将单独的一个 main.c 模块细分为 main.c 模块、initial.c 模块、delay.c 模块、music.c 模块以及其对应的包括文件，可以使程序易于了解，节省开发时间。而且，用范例来说明各种应用方法，以使读者建立整体思维，并进行有效的学习。

第 4 章：详细介绍如何利用定时器的中断方法来产生音阶的频率，并由 I/O 输出此方波信号而驱动喇叭发出正确的音阶。当连续产生各音符的音调频率时，则形成演奏音乐，并渐进式地说明什么样的设计方法是最好的。

第 5 章：音符的形成有两个要素：音调及音长，当音调以定时器中断方法来产生，音长是否也可以由定时器来产生呢？本章介绍如何利用 timer0 及 timer1 两个定时器中断方法来演奏音乐，并特别说明当音长计时中断时间太短时所造成的影响以及解决的方法。

第 6 章：说明音乐中“移调”的概念，分别以查表法和计算法来举例说明 D 大调、降 E 大调、F 大调、G 大调、降 A 大调、降 B 大调。并以 TACT 开关的按键动作来阐述移调的功能，而以外部中断的方法来达到音乐演奏中实时移调的功能。

第 7 章：介绍如何以按键开关来选曲，以“哔”声和 LED 闪烁方式作为选曲的提示动作，并以下列技巧来说明按键的处理方法：开关持续按着的重复动作、开关持续按着也只动作一次、消除按键弹跳波的程序规划、持续按键以延时方式来继续执行动作，及持续按键以定时器计时方式来继续执行动作。同时，通过此方式来培养读者软件设计的能力并使读者养成慎密的思维方式。

第 8 章：以 9 个按键开关分别代表 1～9 首的按键选曲，并介绍如何以 I/O 的方式、SCAN 的方式以及 ADC 的方式来检测按键动作，以及当微电脑 I/O 不敷使用时的解决方法。

第 9 章：介绍如何以 DIP 指拨开关作为选曲的方法，当利用 2P 的 DIP 开关时，具各自独立或相邻 I/O 作为选曲输入的软件构思与解决方法；当利用 4P 的 DIP 开关

可选曲 1~15 首曲目，其低 4 位或高 4 位作为选曲输入及起始键输入的软件构思与解决方法。

第 10 章：介绍当按下选曲键或数字键以七段显示器立即显示的方法，分别是以一个 I/O 端口直接驱动显示器，以 IC7447 来驱动显示器，以及以串行移位缓存器 IC4094 扩充为输出端口作为 IC7447 BCD 码的输入而驱动显示器。

第 11 章：分别以单曲循环、顺序播放、随机选曲及播放简介的演奏功能来说明如何加强系统的附加价值，以及随机选曲功能如何能以开关弹跳波的方式来达成。

第 12 章：介绍如何以 ADC 方式检测更多的按键，而将 LED 以更简单的 I/O 来控制的方法，并在演奏曲目时能立即显示出曲目编号；最后介绍如何将功能模式及曲目编号存储起来，以便重新开机后能立即显示最后输入的曲目编号，并详细介绍记忆 IC 及 IIC BUS 的控制与应用、其软件构建的思维与解决方法。

目　录

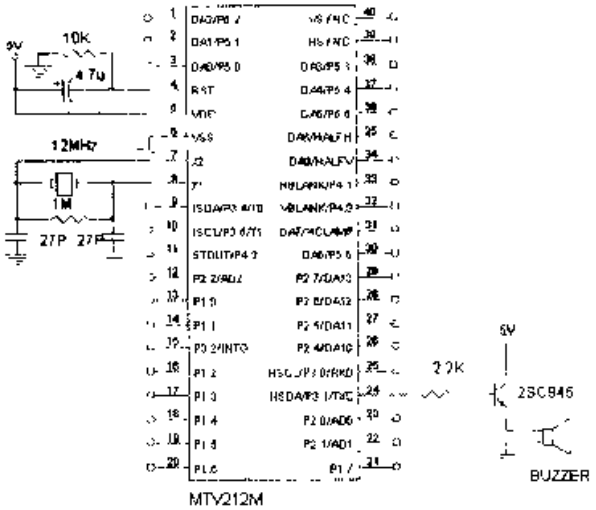
第 1 章　声音基础

1-1　Beep 无自变量

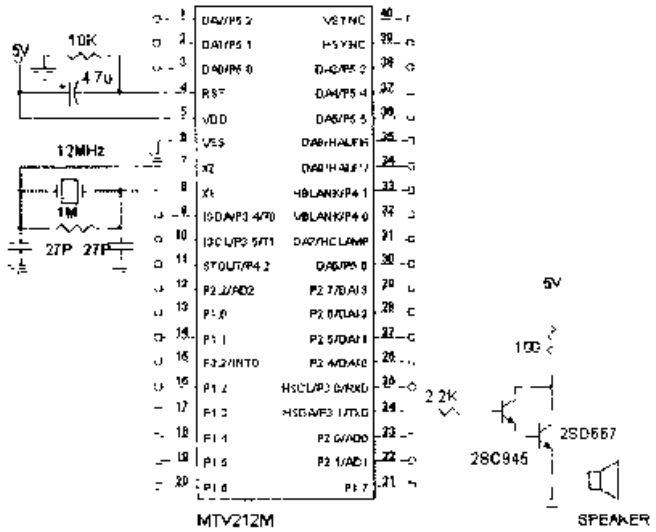
目的：通过 CPU 的 I/O 引脚以软件方式使喇叭发出声音。

电路图：驱动喇叭的线路，为了使其流过电流而控制喇叭的声音大小，可分为以下三种方法。

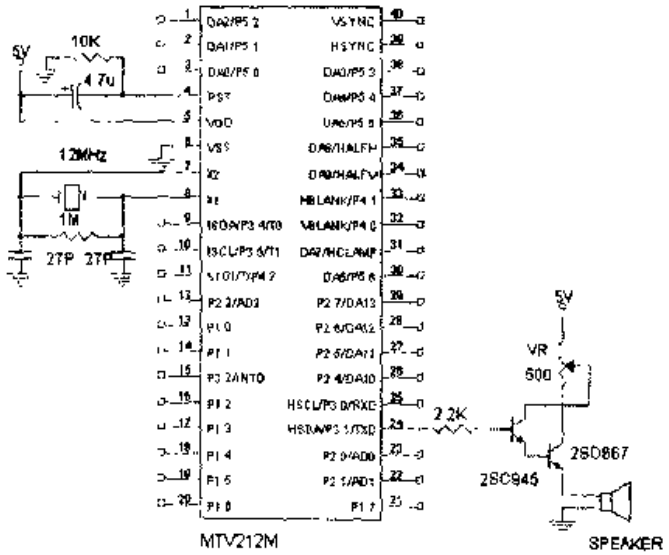
1. 晶体管方式：



2. 达灵顿大电流方式：



3. 可调电阻控制声音大小方式：



原理：其实原理很简单，只需令I/O引脚不断地产生方波，则晶体管就不断地ON/OFF，其集电极电流也就不断地流经喇叭或截止不流过喇叭，而使得喇叭发出声音。也就是控制方波的频率，则喇叭就会发出此频率的声音，而声音的长短只要控制此方波频率的时间长短即可。

软件构建的思维与解决方法

测试喇叭发出声音的软件程序，为了进行音乐演奏，必须将喇叭通过软件方法来发声。除了必须将CPU进行初始化外（只要以微电脑进行控制的，都必须进行CPU的初始化，以稳定CPU状态的功能），在main( )主程序中每秒钟发出一声“哔”并一直重复下去，而延迟程序有三层for循环并通过自变量来达到可任意延迟多少时间。而Beep( )函数的实现方法：即将I/O先设定为“1”电位，此时晶体管导通，电流将流过喇叭，并以for循环作短暂延迟后，再将I/O清除为“0”电位，此时晶体管为截止状态，电流将不会流过喇叭，同样作短暂延迟，这样作17次就会发出“哔”声了。而在程序前面必须包括define.h文件，以便可以使用自定义类型来代替数据类型的声明，例如：Bit、Bool、Byte、Word、Long，以及内存类型的转换字符，如：DATA、IDATA、PDATA、XDATA、RDATA等，而为了调用其他函数，则应在程序前面先进行函数的声明，程序如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
```

```
void PowerOnInit(void);
void DelayX10ms(Word count);
void Beep(void);

/*****************************/
void Main( void )
{
    PowerOnInit();

    while( 1 )
    {
       Beep( );
       DelayX10ms(100);
    }
}

/*****************************/
void PowerOnInit(void)
{
    IE  = 0;              //disable all interrupt
    PSW = 0;              //bank 0

    IP  = 0x0b;      //hi priority:int0,timer0,timer1

    TMOD= 0x11;           //set timer1,timer0 mode

    TR0 = 0;         //stop timer0
    TR1 = 0;              //stop timer1
    IT0 = 1;              //set int0:falling eage trigger

    EX0 = 0;              //disable int0   interrupt
    ET1 = 0;              //disable timer1 interrupt
    ET0 = 0;              //disable timer0 interrupt

    EA  = 1;              //enable all interrupt gate
}

/*****************************/
void DelayX10ms(Word count)
{
    Word i,j,k;

    for(i=0; i<count; i++)
       for(j=0; j<10; j++)
```

```
        for(k=0; k<120; k++)
          ;
}

/*****************************/
void Beep(void)
{
   Byte  j,k;

   for(j=0; j<17; j++)
   {
     P3_1 = 1;
     for(k=0; k<200; k++)
       ;
     P3_1 = 0;
     for(k=0; k<200; k++)
       ;
   }
}
```

包括文件 define.h 的内容主要为自定义类型，bit 数据类型可以使用 Bit 或 Bool 声明，unsigned char 数据类型可以使用 Byte 声明，unsigned int 数据类型可以使用 Word 声明，unsigned long 数据类型可以用 Long 声明，这样有利于缩短文字，同时能望文生义。并定义常数 DATA 以内存类型 data 来转换；常数 IDATA 以内存类型 idata 来转换；常数 PDATA 以内存类型 pdata 来转换；常数 XDATA 以内存类型 xdata 来转换；常数 RDATA 以内存类型 code 来转换，其定义如下：

```
#ifndef   _DEFINE_H
#define   _DEFINE_H

//declare
typedef   bit              Bit;
typedef   bit              Bool;
typedef   unsigned char    Byte;
typedef   unsigned int     Word;
typedef   unsigned long    Long;

#define   DATA          data
#define   IDATA         idata
#define   PDATA         pdata
#define   XDATA         xdata
#define   RDATA         code

#encif
```

包括文件 mtv212m.h 的内容是 8052 CPU 各缓存器地址的定义和缓存器中的每一个位地址定义以及 I/O 的符号定义，将每一个 I/O 引脚以符号来定义。例如：P1_0 代表 I/O P1.0、P2_3 代表 I/O P2.3、P3_5 代表 I/O P3.5 等，其定义如下：

```
/*---8052 sfr address declare---*/
sfr  ACC      = 0xE0;
sfr  B        = 0xF0;
sfr  PSW      = 0xD0;
sfr  SP       = 0x81;
sfr  DPL      = 0x82;
sfr  DPH      = 0x83;
sfr  P0       = 0x80;
sfr  P1       = 0x90;
sfr  P2       = 0xA0;
sfr  P3       = 0xB0;
sfr  IE       = 0xA8;
sfr  IP       = 0xB8;
sfr  PCON     = 0x87;
sfr  TCON     = 0x88;
sfr  TMOD     = 0x89;
sfr  TL0      = 0x8A;
sfr  TL1      = 0x8B;
sfr  TH0      = 0x8C;
sfr  TH1      = 0x8D;
sfr  T2CON    = 0xC8;
sfr  RCAP2L   = 0xCA;
sfr  RCAP2H   = 0xCB;
sfr  TL2      = 0xCC;
sfr  TH2      = 0xCD;
sfr  SCON     = 0x98;
sfr  SBUF     = 0x99;

/*---PSW---*/
sbit CY       = 0xD7;
sbit AC       = 0xD6;
sbit F0       = 0xD5;
sbit RS1      = 0xD4;
sbit RS0      = 0xD3;
sbit OV       = 0xD2;
sbit P        = 0xD0;
/*---TCON---*/
sbit TF1      = 0x8F;
sbit TR1      = 0x8E;
sbit TF0      = 0x8D;
```

```
sbit TR0        = 0x8C;
sbit IE1        = 0x8B;
sbit IT1        = 0x8A;
sbit IE0        = 0x89;
sbit IT0        = 0x88;
/*---T2CON---*/
sbit TF2        = 0xCF;
sbit EXF2       = 0xCE;
sbit RCLK       = 0xCD;
sbit TCLK       = 0xCC;
sbit EXEN2      = 0xCB;
sbit TR2        = 0xCA;
sbit CT2        = 0xC9;
sbit CPRL2      = 0xC8;
/*---PCON---*/
sbit SMOD       = 0x8e;
sbit GF1        = 0x8A;
sbit GF0        = 0x89;
sbit PD         = 0x88;
sbit IDL        = 0x87;
/*---SCON---*/
sbit SM0        = 0x9F;
sbit SM1        = 0x9E;
sbit SM2        = 0x9D;
sbit REN        = 0x9C;
sbit TB8        = 0x9B;
sbit RB8        = 0x9A;
sbit TI         = 0x99;
sbit RI         = 0x98;
/*---IE---*/
sbit EA         = 0xAF;
sbit ET2        = 0xAD;
sbit ES         = 0xAC;
sbit ET1        = 0xAB;
sbit EX1        = 0xAA;
sbit ET0        = 0xA9;
sbit EX0        = 0xA8;
/*---IP---*/
sbit PT2        = 0xBD;
sbit PS         = 0xBC;
sbit PT1        = 0xBB;
sbit PX1        = 0xBA;
sbit PT0        = 0xB9;
sbit PX0        = 0xB8;
```

```
/*---P0---*/
sbit P0_0      = P0 ^ 0;
sbit P0_1      = P0 ^ 1;
sbit P0_2      = P0 ^ 2;
sbit P0_3      = P0 ^ 3;
sbit P0_4      = P0 ^ 4;
sbit P0_5 = P0 ^ 5;
sbit P0_6      = P0 ^ 6;
sbit P0_7      = P0 ^ 7;

/*---P1---*/
sbit P1_0 = P1 ^ 0;
sbit P1_1 = P1 ^ 1;
sbit P1_2 = P1 ^ 2;
sbit P1_3      = P1 ^ 3;
sbit P1_4      = P1 ^ 4;
sbit P1_5 = P1 ^ 5;
sbit P1_6      = P1 ^ 6;
sbit P1_7      = P1 ^ 7;

/*---P2---*/
sbit P2_0 = P2 ^ 0;
sbit P2_1 = P2 ^ 1;
sbit P2_2      = P2 ^ 2;
sbit P2_3 = P2 ^ 3;
sbit P2_4 = P2 ^ 4;
sbit P2_5      = P2 ^ 5;
sbit P2_6      = P2 ^ 6;
sbit P2_7      = P2 ^ 7;

/*---P3---*/
sbit P3_0 = P3 ^ 0;
sbit P3_1      = P3 ^ 1;
sbit P3_2      = P3 ^ 2;
sbit P3_3      = P3 ^ 3;
sbit P3_4      = P3 ^ 4;
sbit P3_5      = P3 ^ 5;
sbit P3_6      = P3 ^ 6;
sbit P3_7      = P3 ^ 7;
```

1-2　Beep 自变量 X1

目的：通过传递声音的频率，自变量可弹性地控制喇叭“哔”声的高低。

软件构建的思维与解决方法

利用传入自变量的方式来取代程序中固定的数值，将使程序本身更具弹性，更适合作各种不同的应用。例如：要产生三种不同的声调来表示三种不同的状况提示时，利用不同的自变量内容即可达到要求，而无需再重新编写程序，这样可大大缩小整体的程序内存空间，所以利用自变量的传递确实是很好且实用的方法。在Beep()函数中的tone变量，即是此函数用来接收调用函数的传递值，经过运算处理后形成产生I/O P3.1“1”电位及“0”电位的时间，也就是方波的频率参数，当tone内容不同时则方波的频率也会不同，则发出声音的音调也会不同，从而达到要求，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"

void PowerOnInit(void);
void DelayX10ms(Word count);
void Beep(Byte tone);

/*****************************/
void Main( void )
{
    PowerOnInit();

    while( 1 )
    {
       Beep(5);
       DelayX10ms(100);
    }
}

/*****************************/
void PowerOnInit(void)
{
    IE  = 0;              //disable all interrupt
    PSW = 0;              //bank 0

    IP  = 0x0b;           //hi priority:int0,timer0,timer1

    TMOD= 0x11;           //set timer1,timer0 mode
```

```
    TR0 = 0;              //stop timer0
    TR1 = 0;              //stop timer1
    IT0 = 1;              //set int0:falling eage trigger

    EX0 = 0;              //disable int0   interrupt
    ET1 = 0;              //disable timer1 interrupt
    ET0 = 0;              //disable timer0 interrupt

    EA  = 1;              //enable all interrupt gate
}

/*****************************/
void DelayX10ms(Word count)
{
    Word i,j,k;

    for(i=0; i<count; i++)
       for(j=0; j<10; j++)
          for(k=0; k<120; k++)
             ;
}

/*****************************/
void Beep(Byte tone)
{
    Byte  j,k,spfreq;

    spfreq  = (1000/tone)/2;

    for(j=0; j<17; j++)
    {
       P3_1 = 1;
       for(k=0; k<spfreq; k++)
          ;
       P3_1 = 0;
       for(k=0; k<spfreq; k++)
          ;
    }
}
```

1-3 Beep 自变量 X2

目的：通过传递声音的持续时间和频率自变量，可弹性控制喇叭发出“哔”声的长短和音调的高低。

软件构建的思维与解决方法

Beep( )函数，除了利用自变量控制频率以改变其音调外，也可利用自变量控制其发出“哔”声的时间长短，则 Beep( )函数将可发出急促或深长的“哔”声，而 soundlong 变量即是此变量，当 soundlong 数值小时则发出急促短暂的声音，当 soundlong 数值大时，则可发出深长的声音，使得 Beep( )函数的功能更趋于完整，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"

void PowerOnInit(void);
void DelayX10ms(Word count);
void Beep(Byte soundlong,Byte tone);

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      Beep(17,10);
      DelayX10ms(100);
   }
}

/*****************************/
void PowerOnInit(void)
{
   IE  = 0;              //disable all interrupt
   PSW = 0;              //bank 0
```

```
    IP  = 0x0b;          //hi priority:int0,timer0,timer1

    TMOD= 0x11;          //set timer1,timer0 mode

    TR0 = 0;             //stop timer0
    TR1 = 0;             //stop timer1
    IT0 = 1;             //set int0:falling eage trigger

    EX0 = 0;             //disable int0   interrupt
    ET1 = 0;             //disable timer1 interrupt
    ET0 = 0;             //disable timer0 interrupt

    EA  = 1;             //enable all interrupt gate
}

/*****************************/
void DelayX10ms(Word count)
{
    Word i,j,k;

    for(i=0; i<ccunt; i++)
       for(j=0; j<10; j++)
          for(k=0; k<120; k++)
            ;
}

/*****************************/
void Beep(Byte soundlong,Byte tone)
{
    Byte  j,k,spfreq;

    spfreq  = (1000/tone)/2;

    for(j=0; j<scundlong; j++)
    {
       P3_1 = 1;
       for(k=0; k<spfreq; k++)
          ;
       P3_1 = 0;
       for(k=0; k<spfreq; k++)
          ;
    }
}
```

1-4 Beep自变量X3

目的：通过传递发出“哔”声个数的自变量，可任意控制其声响次数，以达到各种不同的提示效果。

软件构建的思维与解决方法

令喇叭发出“哔”声的目的，除了作为音乐演奏的基本函数外，也可将其应用于系统中各种不同状态的提示。例如：当输入按键正确时，发出一个“哔”声；当不允许输入按键或按键输入错误时，则发出两个“哔”声；当系统发生严重异常时，则发出急促的5个“哔”声等。因此，将发出“哔”声的Beep( )函数，其个数、音长及音调都以自变量来控制其“哔”声的类型，反而更具弹性而且更具实用价值。Beep( )函数中count变量即是控制发出“哔”声次数的自变量，souldlong变量是控制“哔”声时间长短的自变量，tone变量是控制“哔”声音调高低的自变量，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"

void PowerOnInit(void);
void DelayX10ms(Word count);
void Beep(Byte count,Byte soundlong,Byte tone);

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      Beep(1,17,10);
      DelayX10ms(100);
   }
}

/*****************************/
void PowerOnInit(void)
```

```
{
    IE  = 0;              //disable all interrupt
    PSW = 0;              //bank 0

    IP  = 0x0b;           //hi priority:int0,timer0,timer1

    TMOD= 0x11;           //set timer1,timer0 mode

    TR0 = 0;              //stop timer0
    TR1 = 0;              //stop timer1
    IT0 = 1;              //set int0:falling eage trigger

    EX0 = 0;              //disable int0   interrupt
    ET1 = 0;              //disable timer1 interrupt
    ET0 = 0;              //disable timer0 interrupt

    EA  = 1;              //enable all interrupt gate
}

/*****************************/
void DelayX10ms(Word count)
{
    Word i,j,k;

    for(i=0; i<count; i++)
       for(j=0; j<10; j++)
          for(k=0; k<120; k++)
             ;
}

/*****************************/
void Beep(Byte count,Byte soundlong,Byte tone)
{
    Byte  i,j,k,spfreq;

    spfreq  = (1000/tone)/2;

    for(i=0; i<count; i++)
    {
       for(j=0; j<soundlong; j++)
       {
          P3_1 = 1;
          for(k=0; k<spfreq; k++)
             ;
          P3_1 = 0;
```

```
        for(k=0; k<spfreq; k++)
          ;
      }
      DelayX10ms(12);
    }
}
```

汇编程序如下：

```
$NOMOD51

NAME    MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN           SEGMENT CODE
?PR?PowerOnInit?MAIN    SEGMENT CODE
?PR?_DelayX10ms?MAIN    SEGMENT CODE
?DT?_DelayX10ms?MAIN    SEGMENT DATA OVERLAYABLE
?PR?_Beep?MAIN          SEGMENT CODE
?DT?_Beep?MAIN          SEGMENT DATA OVERLAYABLE
EXTRN   CODE (?C_STARTUP)
EXTRN   CODE (?C?UIDIV)
PUBLIC  _Beep
PUBLIC  _DelayX10ms
PUBLIC  PowerOnInit
PUBLIC  Main

RSEG  ?DT?_DelayX10ms?MAIN
?_DelayX10ms?BYTE:
    count?240:  DS  2

RSEG  ?DT?_Beep?MAIN
?_Beep?BYTE:
    count?344:  DS  1
 soundlong?345:  DS  1
ORG  2
   spfreq?350:  DS  1

RSEG  ?PR?Main?MAIN
USING   0
Main:
LCALL   PowerOnInit
?C0001:
MOV     R7,#01H
```

```
MOV       R5,#011H
MOV       R3,#0AH
LCALL     _Beep
MOV       R7,#064H
MOV       R6,#00H
LCALL     _DelayX10ms
SJMP      ?C0001
RET

RSEG  ?PR?PowerOnInit?MAIN
USING     0
PowerOnInit:
CLR       A
MOV       IE,A
MOV       PSW,A
MOV       IP,#0BH
MOV       TMOD,#011H
CLR       TR0
CLR       TR1
SETB      IT0
CLR       EX0
CLR       ET1
CLR       ET0
SETB      EA
RET

RSEG  ?PR?_DelayX10ms?MAIN
USING     0
_DelayX10ms:
MOV       count?240,R6
MOV       count?240+01H,R7
CLR       A
MOV       R7,A
MOV       R6,A
?C0005:
CLR       C
MOV       A,R7
SUBB      A,count?240+01H
MOV       A,R6
SUBB      A,count?240
JNC       ?C0014
CLR       A
MOV       R5,A
MOV       R4,A
?C0008:
```

```
        CLR     A
        MOV     R3,A
        MOV     R2,A
?C0011:
        INC     R3
        CJNE    R3,#00H,?C0028
        INC     R2
?C0028:
        MOV     A,R3
        XRL     A,#078H
        ORL     A,R2
        JNZ     ?C0011
?C0010:
        INC     R5
        CJNE    R5,#00H,?C0029
        INC     R4
?C0029:
        MOV     A,R5
        XRL     A,#0AH
        ORL     A,R4
        JNZ     ?C0008
?C0007:
        INC     R7
        CJNE    R7,#00H,?C0030
        INC     R6
?C0030:
        SJMP    ?C0005
?C0014:
        RET

        RSEG  ?PR?_Beep?MAIN
        USING   0
_Beep:
        MOV     count?344,R7
        MOV     soundlong?345,R5
        MOV     A,R3
        MOV     R5,A
        MOV     R4,#00H
        MOV     R6,#03H
        MOV     R7,#0E8H
        LCALL   ?C?UIDIV
        MOV     A,R6
        CLR     C
        RRC     A
        MOV     A,R7
```

```
RRC     A
MOV     spfreq?350,A
CLR     A
MOV     R1,A
?C0015:
MOV     A,R1
CLR     C
SUBB    A,count?344
JNC     ?C0027
CLR     A
MOV     R7,A
?C0018:
MOV     A,R7
CLR     C
SUBB    A,soundlong?345
JNC     ?C0019
SETB    P3_1
CLR     A
MOV     R6,A
?C0021:
MOV     A,R6
CLR     C
SUBB    A,spfreq?350
JNC     ?C0022
INC     R6
SJMP    ?C0021
?C0022:
CLR     P3_1
CLR     A
MOV     R6,A
?C0024:
MOV     A,R6
CLR     C
SUBB    A,spfreq?350
JNC     ?C0020
INC     R6
SJMP    ?C0024
?C0020:
INC     R7
SJMP    ?C0018
?C0019:
MOV     R7,#0CH
MOV     R6,#00H
LCALL   _DelayX10ms
INC     R1
```

```
SJMP      ?C0015
?C0027:
RET

END
```

1-5 Beep 定时器

目的：利用定时器方式来产生方波频率，从而控制喇叭发出“哔”声的音调。

软件构建的思维与解决方法

前面介绍了令喇叭发出“哔”声的频率，是通过 tone 参数来产生“1”电位及“0”电位的时间，其时间是以 for 循环进行延迟的，即先设置 I/O 为“1”电位后，以 for 循环延迟 tone 参数的时间，再令 I/O 为“0”电位后，再以 for 循环延迟 tone 参数的时间，然而 tone 参数经过转换后，再加上 for 循环的指令执行周期，其产生“哔”声的频率并非完全正确。如果利用定时器，将 tone 音调变量转换为周期变量 Period，并存入定时器 1 的缓存器 TH1 和 TL1，每经过 Preiod 周期的时间则定时器 1 将产生中断，在中断函数内令 I/O P3.1 输出为反相，则可产生正确频率方波，而在 PowerOnInit( )函数内，必须将定时器 1 的中断使能，即设置 ET1 位才能正确动作。此种以定时器来产生方波频率的方法其准确度非常高，也是音乐演奏的基础，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/

#include  "define.h"

#include  "mtv212m.h"

Word  IDATA   Period;

void PowerOnInit(void);
void DelayX10ms(Word count);
void Beep(Byte count,Byte soundlong,Byte tone);

/*****************************/
void Main( void )
{
```

```
    PowerOnInit();

    while( 1 )
    {
       Beep(1,17,1);
       DelayX10ms(100);
    }
}

/*****************************/
void PowerOnInit(void)
{
    IE  = 0;              //disable all interrupt
    PSW = 0;              //bank 0

    IP  = 0x0b;           //hi priority:int0,timer0,timer1

    TMOD= 0x11;           //set timer1,timer0 mode

    TR0 = 0;              //stop timer0
    TR1 = 0;              //stop timer1
    IT0 = 1;              //set int0:falling eage trigger

    EX0 = 0;              //disable int0 interrupt
    ET1 = 1;              //enable timer1 interrupt
    ET0 = 0;              //disable timer0 interrupt

    EA  = 1;              //enable all interrupt gate
}

/*****************************/
void DelayX10ms(Word count)
{
    Word i,j,k;

    for(i=0; i<count; i++)
       for(j=0; j<10; j++)
          for(k=0; k<120; k++)
             ;
}

/*****************************/
void Beep(Byte count,Byte soundlong,Byte tone)
{
    Byte  i,j,k;
```

```
    Period  = (1000/tone)/2;
    TL1 = (65536 - Period) & 0xff;
    TH1 = (65536 - Period) >> 8;

    for(i=0; i<count; i++)
    {
      TR1 = 1;
      //wait timer1 interrupt
      for(j=0; j<soundlong; j++)
      {
        for(k=0; k<250; k++)
          ;
        for(k=0; k<250; k++)
          ;
      }
      //stop timer1:soundtone
      TR1 = 0;

      DelayX10ms(12);
    }
}

/*****************************/
//timer1,execute service route
/*****************************/
void  Timer1ISR ( void ) interrupt 3 using 2
{
    Word temp;

    temp = 65536 - Period;
    P3_1 = !P3_1;

    TL1 = temp & 0xff;
    TH1 = temp >> 8;
    TF1 = 0;
}
```

汇编程序如下：

```
$NOMOD51

NAME MAIN
```

```
$INCLUDE (mtv212m.inc)

?PR?Main?MAIN            SEGMENT CODE
?PR?PowerOnInit?MAIN     SEGMENT CODE
?PR?_DelayX10ms?MAIN     SEGMENT CODE
?DT?_DelayX10ms?MAIN     SEGMENT DATA OVERLAYABLE
?PR?_Beep?MAIN           SEGMENT CODE
?DT?_Beep?MAIN           SEGMENT DATA OVERLAYABLE
?PR?Timer1ISR?MAIN       SEGMENT CODE
?ID?MAIN                 SEGMENT IDATA
 EXTRN    CODE (?C_STARTUP)
 EXTRN    CODE (?C?UIDIV)
 EXTRN    CODE (?C?SLSHR)
 PUBLIC   Period
 PUBLIC   Timer1ISR
 PUBLIC   _Beep
 PUBLIC   _DelayX10ms
 PUBLIC   PowerOnInit
 PUBLIC   Main

 RSEG  ?DT?_DelayX10ms?MAIN
?_DelayX10ms?BYTE:
     count?240:   DS   2

 RSEG  ?DT?_Beep?MAIN
?_Beep?BYTE:
     count?344:   DS   1
  soundlong?345:   DS   1

 RSEG  ?ID?MAIN
       Period:   DS   2

 RSEG  ?PR?Main?MAIN
 USING    0
Main:
 LCALL    PowerOnInit
?C0001:
 MOV      R7,#01H
 MOV      R5,#011H
 MOV      R3,#01H
 LCALL    _Beep
 MOV      R7,#064H
 MOV      R6,#00H
 LCALL    _DelayX10ms
 SJMP     ?C0001
```

```
RET

RSEG  ?PR?PowerOnInit?MAIN
USING    0
PowerOnInit:
CLR      A
MOV      IE,A
MOV      PSW,A
MOV      IP,#0BH
MOV      TMOD,#011H
CLR      TR0
CLR      TR1
SETB     IT0
CLR      EX0
SETB     ET1
CLR      ET0
SETB     EA
RET

RSEG  ?PR?_DelayX10ms?MAIN
USING    0
_DelayX10ms:
MOV      count?240,R6
MOV      count?240+01H,R7
CLR      A
MOV      R7,A
MOV      R6,A
?C0005:
CLR      C
MOV      A,R7
SUBB     A,count?240+01H
MOV      A,R6
SUBB     A,count?240
JNC      ?C0014
CLR      A
MOV      R5,A
MOV      R4,A
?C0008:
CLR      A
MOV      R3,A
MOV      R2,A
?C0011:
INC      R3
CJNE     R3,#00H,?C0029
INC      R2
```

```
?C0029:
 MOV       A,R3
 XRL       A,#078H
 ORL       A,R2
 JNZ       ?C0011
?C0010:
 INC       R5
 CJNE      R5,#00H,?C0030
 INC       R4
?C0030:
 MOV       A,R5
 XRL       A,#0AH
 ORL       A,R4
 JNZ       ?C0008
?C0007:
 INC       R7
 CJNE      R7,#00H,?C0031
 INC       R6
?C0031:
 SJMP      ?C0005
?C0014:
 RET

 RSEG  ?PR?_Beep?MAIN
 USING     0
_Beep:
 MOV       count?344,R7
 MOV       soundlong?345,R5
 MOV       A,R3
 MOV       R5,A
 MOV       R4,#00H
 MOV       R6,#03H
 MOV       R7,#0E8H
 LCALL     ?C?UIDIV
 MOV       A,R6
 CLR       C
 RRC       A
 MOV       R6,A
 MOV       A,R7
 RRC       A
 MOV       R7,A
 MOV       R0,#Period
 MOV       A,R6
 MOV       @R0,A
 INC       R0
```

```
MOV     A,R7
MOV     @R0,A
CLR     A
MOV     R4,A
MOV     R5,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
SUBB    A,R3
MOV     R7,A
CLR     A
SUBB    A,R2
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
SUBB    A,#00H
MOV     R4,A
MOV     A,R7
CLR     A
MOV     TL1,R7
DEC     R0
MOV     A,@R0
MOV     R6,A
INC     R0
MOV     A,@R0
MOV     R7,A
CLR     A
MOV     R4,A
MOV     R5,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
SUBB    A,R3
MOV     R7,A
CLR     A
```

```
SUBB    A,R2
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
SUBB    A,#00H
MOV     R4,A
MOV     R0,#08H
LCALL   ?C?SLSHR
MOV     TH1,R7
CLR     A
MOV     R1,A
?C0015:
MOV     A,R1
CLR     C
SUBB    A,count?344
JNC     ?C0027
SETB    TR1
CLR     A
MOV     R7,A
?C0018:
MOV     A,R7
CLR     C
SUBB    A,soundlong?345
JNC     ?C0019
CLR     A
MOV     R6,A
?C0021:
INC     R6
CJNE    R6,#0FAH,?C0021
?C0022:
CLR     A
MOV     R6,A
?C0024:
INC     R6
CJNE    R6,#0FAH,?C0024
?C0020:
INC     R7
SJMP    ?C0018
?C0019:
CLR     TR1
MOV     R7,#0CH
MOV     R6,#00H
LCALL   _DelayX10ms
```

```
        INC     R1
        SJMP    ?C0015
?C0027:
        RET

CSEG AT  0001BH
        LJMP Timer1ISR

        RSEG  ?PR?Timer1ISR?MAIN
        USING   2
Timer1ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     PSW,#010H
        MOV     R0,#Period
        MOV     A,@R0
        MOV     R6,A
        INC     R0
        MOV     A,@R0
        MOV     R7,A
        CLR     C
        CLR     A
        SUBB    A,R7
        MOV     R7,A
        CLR     A
        SUBB    A,R6
        MOV     R6,A
        CPL     P3_1
        XCH     A,R5
        MOV     A,R7
        XCH     A,R5
        MOV     A,R5
        MOV     TL1,A
        MOV     A,R6
        MOV     TH1,A
        CLR     TF1
        POP     PSW
        POP     ACC
        RETI

        END
```

1-6　音乐演奏基础

当有了 Beep 声的概念之后，即可以设计更复杂的音乐演奏了，因为音乐是由一连串的音符构成的，曲目中只是每个音符的音长和频率不同而已。只要将每个音符的音长和音调事先计算出来并建表即可，当程序不断地将音长和音调的数值取出后，并利用 for 循环产生方波频率和时间，则喇叭就会发出声音而形成音乐演奏了。然而，要如何建立每个音符的音长和频率呢？在音乐中 4/4 表示为每小节有 4 拍，并以四分音符为一拍，因此，在程序中音长可以将四分音符定为 12，而演奏速度可设为 12 个循环，占用一个字节，但可根据需求而改变；而每个音阶的频率也必须知道才能换算为音调数值而建成音调表，其音名和频率对照表如下：

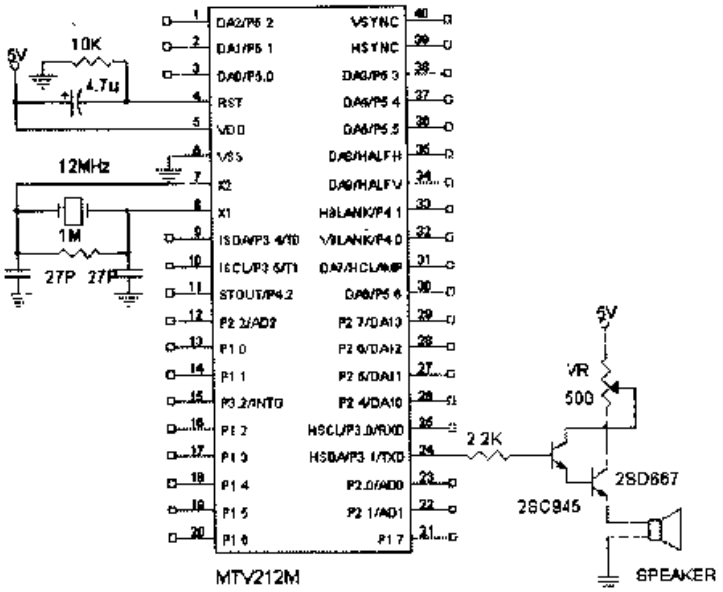
音名	频率	音名	频率	音名	频率	音名	频率
C2	65	A3	220	F#5	740	D#7	2489
C#2	69	A#3	233	G5	784	E7	2637
D2	73	B3	247	G#5	831	F7	2794
D#2	78	C4	262	A5	880	F#7	2960
E2	82	C#4	277	A#5	932	G7	3136
F2	87	D4	294	B5	988	G#7	3322
F#2	93	D#4	311	C6	1047	A7	3520
G2	98	E4	330	C#6	1109	A#7	3729
G#2	104	F4	349	D6	1175	B7	3951
A2	110	F#4	370	D#6	1245	C8	4186
A#2	116	G4	392	E6	1319	C#8	4435
B2	123	G#4	415	F6	1397	D8	4699
C3	131	A4	440	F#6	1480	D#8	4978
C#3	139	A#4	466	G6	1568	E8	5274
D3	147	B4	494	G#6	1661	F8	5587
D#3	156	中央 C5	523	A6	1760	F#8	5919
E3	165	C#5	554	A#6	1865	G8	6271
F3	175	D5	587	B6	1976	G#8	6645
F#3	185	D#5	622	C7	2093	A8	7040
G3	196	E5	659	C#7	2217	A#8	7459
G#3	208	F5	698	D7	2349	B8	7902

有了这些信息，即可进行音乐演奏了。

第 2 章　单一程序音乐演奏

目的：利用 main.c 单一模块并通过各种指令结构来进行音乐演奏，以说明软件构建的思维与解决方法。

电路图：



原理：要让 CPU 可以正确地执行软件的动作，有以下三点：

- CPU 要供应电源+5V 和接地。
- CPU 要有重置信号。
- CPU 要能正确、稳定地产生脉冲。

图上 MTV212M 的第六支脚为接地，第五支脚为+5V 的输入脚，而第四支脚为利用 RC 所组成的简单重置信号。当开机提供+5V 时，则第四支脚将产生短暂时间的正脉冲信号作为 CPU 的重置信号，CPU 重置信号完成后，将由程序中 Main( )函数开始执行。而第七、八支脚即是脉冲的输入脚，作为执行指令，定时器、计数器、外部中断的时间周期的依据，使用 12MHz 石英振荡器，以及 1MΩ 电阻和 27PF 的陶瓷电容可以很简单且稳定地产生脉冲，而 I/O 脚 P3.1 是产生音阶频率的输出脚，通过达灵顿晶体可驱动较大电流的喇叭输出，其中 VR 500Ω 可调整喇叭的输出音量。综上所述，如此简单的电路构成，即可达到音乐演奏的目的，也借以说明如何培养软件构建的思维与解决方法。

2-1　if 指令

演奏“三轮车”音乐的软件程序完全由一个 Main( )函数所组成，包含初始化程序、音乐演奏程序以及延迟程序，为了达到调用这些程序的目的，必须将函数定义在程序的前面。程序中因为用到 8051 缓存器及自定义数据类型，因此，在程序开始也必须包括相对应的包括文件，如：define.h 和 mtv212m.h，而 Music( )音乐函数以 if 指令结构来依次加载每一音符的音长和音调，通过三层 for 循环来达到演奏的目的，分述如下：

1. Main( )函数：程序的进入点，除了作初始化的动作外，以一个 while(1)反复循环来调用 Music( )音乐演奏函数，每演奏完毕并延迟 2 秒后继续演奏，如此不断地循环。

2. PowerOnInit( )函数：包含初始化 CPU 的缓存器：中断缓存器（IE）、缓存器库 R0～R7（PSW）、中断优先次序缓存器（IP）、计时计数模式缓存器（TMOD）、启动或停止定时器（TR0、TR1）、外部中断边缘或准位触发（ITO）、使能或除能外部中断、定时器 0 或定时器 1（EX0、ET0、ET1）以及全部中断闸的控制（EA）等，以及初始化 I/O：即将喇叭的输出状态清除为 0，即不导通状态。

3. DelayX10ms( )函数：通过自变量 count 来达到弹性延迟时间的函数，而真正的延迟时间为 count 乘以 10ms，以占用 2 个字节的局域变量 i、j、k 来作为三层 for 循环的等待。

4. Music( )函数：以反复循环来不断演奏每一个音符，当演奏至最后一个音符时（以 if 指令来判断），则跳离此反复循环回到原调用程序，局域变量 i 要初始化为 0，并作为音长表和音调表的下标，以取得音符的音长和音调。通过 for 循环来完成实际输出至喇叭方波频率和方波的时间以演奏出音阶，每当演奏一个音符，则下标 i 变量就累加 1，并一直到最后的音符，由 Main( )函数组成的所有程序如下：

```
/*******************************/
/* include files               */
/*******************************/

#include  "define.h"

#include  "mtv212m.h"

void PowerOnInit(void);
void InitialCpu(void);
void InitialCpuIO(void);
void DelayX10ms(Word count);
```

```
void Music(void);

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
     Music( );
     DelayX10ms(100);
   }
}

/*****************************/
void PowerOnInit(void)
{
   InitialCpu();

   InitialCpuIO();
}

void InitialCpu(void)
{
   IE  = 0;              //disable all interrupt
   PSW = 0;              //bank 0

   IP  = 0x0b;           //hi priority:int0,timer0,timer1

   TMOD= 0x11;           //set timer1,timer0 mode

   TR0 = 0;              //stop timer0
   TR1 = 0;              //stop timer1
   IT0 = 1;              //set int0:falling eage trigger

   EX0 = 0;              //disable int0   interrupt
   ET1 = 0;              //disable timer1 interrupt
   ET0 = 0;              //disable timer0 interrupt

   EA  = 1;              //enable all interrupt gate
}

void InitialCpuIO(void)
{
   P3_1 = 0;             //speaker=off
```

```
}

/*****************************/
void DelayX10ms(Word count)
{
   Word i,j,k;

   for(i=0; i<count; i++)
      for(j=0; j<10; j++)
         for(k=0; k<120; k++)
            ;
}

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
   6,6,9,3,6,6,12,
   6,6,6,6,6,6,12,
   6,6,9,3,6,6,9 ,3,
   6,3,3,6,3,3,6 ,6,9
};
Byte RDATA SOUNDTONE[ ] =
{
   120,120,106,95,80,80,95,
   80 ,80 ,71 ,63,60,60,80,
   60 ,60 ,71 ,80,95,71,80,95 ,
   120,106,95 ,80,71,80,95,106,120
};

void Music(void)
{
   Byte  i=0,j,k;
   Byte  SoundLong,SoundTone;
   Word  m;

   while( 1 )
   {
      //sound count
      if (i==31) return;
      SoundLong=SOUNDLONG[i];
      SoundTone=SOUNDTONE[i];
      i++;

      //sound long
      for(j=0; j<SoundLong; j++)
```

```
        {
          //tempo
          for(k=0; k<12; k++)
          {
            //sound tone
            P3_1 = 1;
            for(m=0; m<SoundTone; m++)
              ;
            P3_1 = 0;
            for(m=0; m<SoundTone; m++)
              ;
          }
        }
    }
}
```

汇编程序如下：

```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN            SEGMENT CODE
?PR?PowerOnInit?MAIN     SEGMENT CODE
?PR?InitialCpu?MAIN      SEGMENT CODE
?PR?InitialCpuIO?MAIN    SEGMENT CODE
?PR?_DelayX10ms?MAIN     SEGMENT CODE
?DT?_DelayX10ms?MAIN     SEGMENT DATA OVERLAYABLE
?PR?Music?MAIN           SEGMENT CODE
?DT?Music?MAIN           SEGMENT DATA OVERLAYABLE
?CO?MAIN                 SEGMENT CODE
 EXTRN    CODE (?C_STARTUP)
 PUBLIC   SOUNDTONE
 PUBLIC   SOUNDLONG
 PUBLIC   Music
 PUBLIC   _DelayX10ms
 PUBLIC   InitialCpuIO
 PUBLIC   InitialCpu
 PUBLIC   PowerOnInit
 PUBLIC   Main

RSEG  ?DT?_DelayX10ms?MAIN
?_DelayX10ms?BYTE:
```

```
    count?440:   DS   2

RSEG  ?DT?Music?MAIN
?Music?BYTE:
  SoundLong?547:   DS   1
  SoundTone?548:   DS   1

RSEG  ?CO?MAIN
SOUNDLONG:
DB   006H,006H,009H,003H,006H
DB   006H,00CH,006H,006H,006H
DB   006H,006H,006H,00CH,006H
DB   006H,009H,003H,006H,006H
DB   009H,003H,006H,003H,003H
DB   006H,003H,003H,006H,006H
DB   009H

SOUNDTONE:
DB   078H,078H,06AH,05FH,050H
DB   050H,05FH,050H,050H,047H
DB   03FH,03CH,03CH,050H,03CH
DB   03CH,047H,050H,05FH,047H
DB   050H,05FH,078H,06AH,05FH
DB   050H,047H,050H,05FH,06AH
DB   078H

RSEG  ?PR?Main?MAIN
USING    0
Main:
LCALL    PowerOnInit
?C0001:
LCALL    Music
MOV      R7,#064H
MOV      R6,#00H
LCALL    _DelayX10ms
SJMP     ?C0001
RET

RSEG  ?PR?PowerOnInit?MAIN
USING    0
PowerOnInit:
LCALL    InitialCpu
LCALL    InitialCpuIO
RET
```

```
RSEG  ?PR?InitialCpu?MAIN
USING    0
InitialCpu:
CLR      A
MOV      IE,A
MOV      PSW,A
MOV      IP,#0BH
MOV      TMOD,#011H
CLR      TR0
CLR      TR1
SETB     IT0
CLR      EX0
CLR      ET1
CLR      ET0
SETB     EA
RET

RSEG  ?PR?InitialCpuIO?MAIN
USING    0
InitialCpuIO:
CLR      P3_1
RET

RSEG  ?PR?_DelayX10ms?MAIN
USING    0
_DelayX10ms:
MOV      count?440,R6
MOV      count?440+01H,R7
CLR      A
MOV      R7,A
MOV      R6,A
?C0007:
CLR      C
MOV      A,R7
SUBB     A,count?440+01H
MOV      A,R6
SUBB     A,count?440
JNC      ?C0016
CLR      A
MOV      R5,A
MOV      R4,A
?C0010:
CLR      A
MOV      R3,A
MOV      R2,A
```

```
?C0013:
 INC       R3
 CJNE      R3,#00H,?C0033
 INC       R2
?C0033:
 MOV       A,R3
 XRL       A,#078H
 ORL       A,R2
 JNZ       ?C0013
?C0012:
 INC       R5
 CJNE      R5,#00H,?C0034
 INC       R4
?C0034:
 MOV       A,R5
 XRL       A,#0AH
 ORL       A,R4
 JNZ       ?C0010
?C0009:
 INC       R7
 CJNE      R7,#00H,?C0035
 INC       R6
?C0035:
 SJMP      ?C0007
?C0016:
 RET

 RSEG  ?PR?Music?MAIN
 USING     0
Music:
 CLR       A
 MOV       R1,A
?C0017:
 MOV       A,R1
 XRL       A,#01FH
 JZ   ?C0020
?C0019:
 MOV       A,R1
 MOV       DPTR,#SOUNDLONG
 MOVC      A,@A+DPTR
 MOV       SoundLong?547,A
 MOV       A,R1
 MOV       DPTR,#SOUNDTONE
 MOVC      A,@A+DPTR
 MOV       SoundTone?548,A
```

```
INC     R1
CLR     A
MOV     R7,A
?C0021:
MOV     A,R7
CLR     C
SUBB    A,SoundLong?547
JNC     ?C0017
CLR     A
MOV     R6,A
?C0024:
SETB    P3_1
CLR     A
MOV     R5,A
MOV     R4,A
?C0027:
CLR     C
MOV     A,R5
SUBB    A,SoundTone?548
MOV     A,R4
SUBB    A,#00H
JNC     ?C0028
INC     R5
CJNE    R5,#00H,?C0036
INC     R4
?C0036:
SJMP    ?C0027
?C0028:
CLR     P3_1
CLR     A
MOV     R4,A
MOV     R5,A
?C0030:
CLR     C
MOV     A,R5
SUBB    A,SoundTone?548
MOV     A,R4
SUBB    A,#00H
JNC     ?C0026
INC     R5
CJNE    R5,#00H,?C0037
INC     R4
?C0037:
SJMP    ?C0030
?C0026:
```

```
INC     R6
CJNE    R6,#0CH,?C0024
?C0023:
INC     R7
SJMP    ?C0021
?C0020:
RET

END
```

2-2　if 宏定义 Speaker

将输出至喇叭的 I/O 以宏指令定义，以方便当硬件电路改变时，而软件只要修改此定义的 I/O 引脚即可。这样，可以避免在程序中相关的部分都必须作修改的麻烦，同时避免出错，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#define  SPEAKER   P3_1

void PowerOnInit(void);
void InitialCpu(void);
void InitialCpuIO(void);
void DelayX10ms(Word count);
void Music(void);

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      Music( );
      DelayX10ms(100);
   }
}

/*****************************/
```

```
void PowerOnInit(void)
{
   InitialCpu();

   InitialCpuIO();
}

void InitialCpu(void)
{
   IE  = 0;              //disable all interrupt
   PSW = 0;              //bank 0

   IP  = 0x0b;           //hi priority:int0,timer0,timer1

   TMOD= 0x11;           //set timer1,timer0 mode

   TR0 = 0;              //stop timer0
   TR1 = 0;              //stop timer1
   IT0 = 1;              //set int0:falling eage trigger

   EX0 = 0;              //disable int0   interrupt
   ET1 = 0;              //disable timer1 interrupt
   ET0 = 0;              //disable timer0 interrupt

   EA  = 1;              //enable all interrupt gate
}

void InitialCpuIO(void)
{
   SPEAKER = 0;
}

/*******************************/
void DelayX10ms(Word count)
{
   Word i,j,k;

   for(i=0; i<count; i++)
      for(j=0; j<10; j++)
         for(k=0; k<120; k++)
            ;
}

/*******************************/
Byte RDATA SOUNDLONG[ ] =
```

```
{
    6,6,9,3,6,6,12,
    6,6,6,6,6,6,12,
    6,6,9,3,6,6,9 ,3,
    6,3,3,6,3,3,6 ,6,9
};
Byte RDATA SOUNDTONE[ ] =
{
    120,120,106,95,80,80,95,
    80 ,80 ,71 ,63,60,60,80,
    60 ,60 ,71 ,80,95,71,80,95 ,
    120,106,95 ,80,71,80,95,106,120
};

void Music(void)
{
    Byte  i=0,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    while( 1 )
    {
       //sound count
       if (i==31) return;
       SoundLong=SOUNDLONG[i];
       SoundTone=SOUNDTONE[i];
       i++;

       //sound long
       for(j=0; j<SoundLong; j++)
       {
          //tempo
          for(k=0; k<12; k++)
          {
             //sound tone
             SPEAKER = 1;
             for(m=0; m<SoundTone; m++)
                ;
             SPEAKER = 0;
             for(m=0; m<SoundTone; m++)
                ;
          }
       }
    }
}
```

2-3 for 循环

演奏音乐函数用 for 指令来完成，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"

#define  SPEAKER   P3_1

void PowerOnInit(void);
void InitialCpu(void);
void InitialCpuIO(void);
void DelayX10ms(Word count);
void Music(void);

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      Music( );
      DelayX10ms(100);
   }
}

/*****************************/
void PowerOnInit(void)
{
   InitialCpu();

   InitialCpuIO();
}

void InitialCpu(void)
{
   IE  = 0;              //disable all interrupt
   PSW = 0;              //bank 0
```

```
    IP  = 0x0b;          //hi priority:int0,timer0,timer1

    TMOD= 0x11;          //set timer1,timer0 mode

    TR0 = 0;             //stop timer0
    TR1 = 0;             //stop timer1
    IT0 = 1;             //set int0:falling eage trigger

    EX0 = 0;             //disable int0   interrupt
    ET1 = 0;             //disable timer1 interrupt
    ET0 = 0;             //disable timer0 interrupt

    EA  = 1;             //enable all interrupt gate
}

void InitialCpuIO(void)
{
    SPEAKER = 0;
}

/*******************************/
void DelayX10ms(Word count)
{
    Word i,j,k;

    for(i=0; i<count; i++)
       for(j=0; j<10; j++)
          for(k=0; k<120; k++)
             ;
}

/*******************************/
Byte RDATA SOUNDLONG[ ] =
{
    6,6,9,3,6,6,12,
    6,6,6,6,6,6,12,
    6,6,9,3,6,6,9 ,3,
    6,3,3,6,3,3,6 ,6,9
};
Byte RDATA SOUNDTONE[ ] =
{
    120,120,106,95,80,80,95,
    80 ,80 ,71 ,63,60,60,80,
    60 ,60 ,71 ,80,95,71,80,95 ,
    120,106,95 ,80,71,80,95,106,120
```

```
};

void Music(void)
{
   Byte  i,j,k;
   Byte  SoundLong,SoundTone;
   Word  m;

   //sound count
   for(i=0; i<31; i++)
   {
      SoundLong=SOUNDLONG[i];
      SoundTone=SOUNDTONE[i];

      //sound long
      for(j=0; j<SoundLong; j++)
      {
            //tempo
         for(k=0; k<12; k++)
         {
            //sound tone
            SPEAKER = 1;
            for(m=0; m<SoundTone; m++)
               ;
            SPEAKER = 0;
            for(m=0; m<SoundTone; m++)
               ;
         }
      }
   }
}
```

其汇编程序如下：

```
Music:
 CLR       A
 MOV       R7,A
?C0017:
 MOV       A,R7
 MOV       DPTR,#SOUNDLONG
 MOVC      A,@A+DPTR
 MOV       SoundLong?547,A
 MOV       A,R7
 MOV       DPTR,#SOUNDTONE
 MOVC      A,@A+DPTR
```

```
MOV     SoundTone?548,A
CLR     A
MOV     R1,A
?C0020:
MOV     A,R1
CLR     C
SUBB    A,SoundLong?547
JNC     ?C0019
CLR     A
MOV     R6,A
?C0023:
SETB    P3_1
CLR     A
MOV     R5,A
MOV     R4,A
?C0026:
CLR     C
MOV     A,R5
SUBB    A,SoundTone?548
MOV     A,R4
SUBB    A,#00H
JNC     ?C0027
INC     R5
CJNE    R5,#00H,?C0036
INC     R4
?C0036:
SJMP    ?C0026
?C0027:
CLR     P3_1
CLR     A
MOV     R4,A
MOV     R5,A
?C0029:
CLR     C
MOV     A,R5
SUBB    A,SoundTone?548
MOV     A,R4
SUBB    A,#00H
JNC     ?C0025
INC     R5
CJNE    R5,#00H,?C0037
INC     R4
?C0037:
SJMP    ?C0029
?C0025:
```

```
INC     R6
CJNE    R6,#0CH,?C0023
?C0022:
INC     R1
SJMP    ?C0020
?C0019:
INC     R7
CJNE    R7,#01FH,?C0017
?C0032:
RET
```

2-4 do...while 循环

演奏音乐函数以 do....while 指令来完成，每执行一个音符则下标 i 变量就加 1，并以 0X00 作为结束音符的数值，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"

#define  SPEAKER   P3_1

void PowerOnInit(void);
void InitialCpu(void);
void InitialCpuIO(void);
void DelayX10ms(Word count);
void Music(void);

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      Music( );
      DelayX10ms(100);
   }
}

/*****************************/
```

```
void PowerOnInit(void)
{
    InitialCpu();

    InitialCpuIO();
}

void InitialCpu(void)
{
    IE  = 0;              //disable all interrupt
    PSW = 0;              //bank 0

    IP  = 0x0b;           //hi priority:int0,timer0,timer1

    TMOD= 0x11;           //set timer1,timer0 mode

    TR0 = 0;              //stop timer0
    TR1 = 0;              //stop timer1
    IT0 = 1;              //set int0:falling eage trigger

    EX0 = 0;              //disable int0   interrupt
    ET1 = 0;              //disable timer1 interrupt
    ET0 = 0;              //disable timer0 interrupt

    EA  = 1;              //enable all interrupt gate
}

void InitialCpuIO(void)
{
    SPEAKER = 0;
}

/*******************************/
void DelayX10ms(Word count)
{
    Word i,j,k;

    for(i=0; i<count; i++)
      for(j=0; j<10; j++)
        for(k=0; k<120; k++)
          ;
}

/*******************************/
Byte RDATA SOUNDLONG[ ] =
```

```
{
    6,6,9,3,6,6,12,
    6,6,6,6,6,6,12,
    6,6,9,3,6,6,9 ,3,
    6,3,3,6,3,3,6 ,6,9,
    0      //end
};
Byte RDATA SOUNDTONE[ ] =
{
    120,120,106,95,80,80,95,
    80 ,80 ,71 ,63,60,60,80,
    60 ,60 ,71 ,80,95,71,80,95 ,
    120,106,95 ,80,71,80,95,106,120,
    0 //end
};

void Music(void)
{
    Byte  i=0,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    do
    {
       SoundLong=SOUNDLONG[i];
       SoundTone=SOUNDTONE[i];
       i++;

       //sound long
       for(j=0; j<SoundLong; j++)
       {
            //tempo
          for(k=0; k<12; k++)
          {
             //sound tone
             SPEAKER = 1;
             for(m=0; m<SoundTone; m++)
                ;
             SPEAKER = 0;
             for(m=0; m<SoundTone; m++)
                ;
          }
       }

    } while ( (SOUNDLONG[i]!=0x00)||(SOUNDTONE[i]!=0x00) );
```

```
}
```

其汇编程序如下：

```
Music:
 CLR      A
 MOV      R1,A
?C0019:
 MOV      A,R1
 MOV      DPTR,#SOUNDLONG
 MOVC     A,@A+DPTR
 MOV      SoundLong?547,A
 MOV      A,R1
 MOV      DPTR,#SOUNDTONE
 MOVC     A,@A+DPTR
 MOV      SoundTone?548,A
 INC      R1
 CLR      A
 MOV      R7,A
?C0020:
 MOV      A,R7
 CLR      C
 SUBB     A,SoundLong?547
 JNC      ?C0017
 CLR      A
 MOV      R6,A
?C0023:
 SETB     P3_1
 CLR      A
 MOV      R5,A
 MOV      R4,A
?C0026:
 CLR      C
 MOV      A,R5
 SUBB     A,SoundTone?548
 MOV      A,R4
 SUBB     A,#00H
 JNC      ?C0027
 INC      R5
 CJNE     R5,#00H,?C0036
 INC      R4
?C0036:
 SJMP     ?C0026
?C0027:
 CLR      P3_1
```

```
CLR     A
MOV     R4,A
MOV     R5,A
?C0029:
CLR     C
MOV     A,R5
SUBB    A,SoundTone?548
MOV     A,R4
SUBB    A,#00H
JNC     ?C0025
INC     R5
CJNE    R5,#00H,?C0037
INC     R4
?C0037:
SJMP    ?C0029
?C0025:
INC     R6
CJNE    R6,#0CH,?C0023
?C0022:
INC     R7
SJMP    ?C0020
?C0017:
MOV     A,R1
MOV     DPTR,#SOUNDLONG
MOVC    A,@A+DPTR
JNZ     ?C0019
MOV     A,R1
MOV     DPTR,#SOUNDTONE
MOVC    A,@A+DPTR
JNZ     ?C0019
RET
```

2-5 while循环

演奏音乐函数以while先判断后执行的语句完成，如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#define  SPEAKER   P3_1
```

```
void PowerOnInit(void);
void InitialCpu(void);
void InitialCpuIO(void);
void DelayX10ms(Word count);
void Music(void);

/*****************************/
void Main( void )
{
    PowerOnInit();

    while( 1 )
    {
      Music( );
      DelayX10ms(100);
    }
}

/*****************************/
void PowerOnInit(void)
{
    InitialCpu();

    InitialCpuIO();
}

void InitialCpu(void)
{
    IE  = 0;              //disable all interrupt
    PSW = 0;              //bank 0

    IP  = 0x0b;           //hi priority:int0,timer0,timer1

    TMOD= 0x11;           //set timer1,timer0 mode

    TR0 = 0;              //stop timer0
    TR1 = 0;              //stop timer1
    IT0 = 1;              //set int0:falling eage trigger

    EX0 = 0;              //disable int0   interrupt
    ET1 = 0;              //disable timer1 interrupt
    ET0 = 0;              //disable timer0 interrupt

    EA  = 1;              //enable all interrupt gate
}
```

```
void InitialCpuIO(void)
{
   SPEAKER = 0;
}

/*****************************/
void DelayX10ms(Word count)
{
   Word i,j,k;

   for(i=0; i<count; i++)
      for(j=0; j<10; j++)
         for(k=0; k<120; k++)
            ;
}

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
   6,6,9,3,6,6,12,
   6,6,6,6,6,6,12,
   6,6,9,3,6,6,9 ,3,
   6,3,3,6,3,3,6 ,6,9,
   0 //end
};
Byte RDATA SOUNDTONE[ ] =
{
   120,120,106,95,80,80,95,
   80 ,80 ,71 ,63,60,60,80,
   60 ,60 ,71 ,80,95,71,80,95 ,
   120,106,95 ,80,71,80,95,106,120,
   0 //end
};

void Music(void)
{
   Byte  i=0,j,k;
   Byte  SoundLong,SoundTone;
   Word  m;

   while ( (SOUNDLONG[i]!=0x00)||(SOUNDTONE[i]!=0x00) )
   {
      SoundLong=SOUNDLONG[i];
      SoundTone=SOUNDTONE[i];
```

```
        i++;

        //sound long
        for(j=0; j<SoundLong; j++)
        {
              //tempo
           for(k=0; k<12; k++)
           {
              //sound tone
              SPEAKER = 1;
              for(m=0; m<SoundTone; m++)
                 ;
              SPEAKER = 0;
              for(m=0; m<SoundTone; m++)
                 ;
           }
        }
     }
}
```

其汇编程序如下：

```
Music:
 CLR       A
 MOV       R1,A
?C0017:
 MOV       A,R1
 MOV       DPTR,#SOUNDLONG
 MOVC      A,@A+DPTR
 MOV       R6,A
 JNZ       ?C0019
 MOV       A,R1
 MOV       DPTR,#SOUNDTONE
 MOVC      A,@A+DPTR
 JZ   ?C0032
?C0019:
 MOV       SoundLong?547,R6
 MOV       A,R1
 MOV       DPTR,#SOUNDTONE
 MOVC      A,@A+DPTR
 MOV       SoundTone?548,A
 INC       R1
 CLR       A
 MOV       R7,A
?C0020:
```

```
MOV     A,R7
CLR     C
SUBB    A,SoundLong?547
JNC     ?C0017
CLR     A
MOV     R6,A
?C0023:
SETB    P3_1
CLR     A
MOV     R5,A
MOV     R4,A
?C0026:
CLR     C
MOV     A,R5
SUBB    A,SoundTone?548
MOV     A,R4
SUBB    A,#00H
JNC     ?C0027
INC     R5
CJNE    R5,#00H,?C0036
INC     R4
?C0036:
SJMP    ?C0026
?C0027:
CLR     P3_1
CLR     A
MOV     R4,A
MOV     R5,A
?C0029:
CLR     C
MOV     A,R5
SUBB    A,SoundTone?548
MOV     A,R4
SUBB    A,#00H
JNC     ?C0025
INC     R5
CJNE    R5,#00H,?C0037
INC     R4
?C0037:
SJMP    ?C0029
?C0025:
INC     R6
CJNE    R6,#0CH,?C0023
?C0022:
INC     R7
```

```
SJMP      ?C0020
?C0032:
RET
```

2-6　do...while 结束符号 0X00

结束符号 0X00 写在音长表和音调表的第一个字节以表示不演奏，但演奏音乐函数却以 do....while 指令来完成，将产生错误而会一直演奏下去，因为 do...while 是先执行后判断的指令，应该以 while 先判断后执行的指令才能正确，其 do...while 的程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#define  SPEAKER   P3_1

void PowerOnInit(void);
void InitialCpu(void);
void InitialCpuIO(void);
void DelayX10ms(Word count);
void Music1(void);
void Music2(void);

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      Music1( );
      DelayX10ms(100);
      Music2( );
      DelayX10ms(100);
   }
}

/*****************************/
void PowerOnInit(void)
{
```

```
    InitialCpu();

    InitialCpuIO();
}

void InitialCpu(void)
{
    IE  = 0;              //disable all interrupt
    PSW = 0;              //bank 0

    IP  = 0x0b;           //hi priority:int0,timer0,timer1

    TMOD= 0x11;           //set timer1,timer0 mode

    TR0 = 0;              //stop timer0
    TR1 = 0;              //stop timer1
    IT0 = 1;              //set int0:falling eage trigger

    EX0 = 0;              //disable int0   interrupt
    ET1 = 0;              //disable timer1 interrupt
    ET0 = 0;              //disable timer0 interrupt

    EA  = 1;              //enable all interrupt gate
}

void InitialCpuIO(void)
{
    SPEAKER = 0;
}

/*****************************/
void DelayX10ms(Word count)
{
    Word i,j,k;

    for(i=0; i<count; i++)
      for(j=0; j<10; j++)
        for(k=0; k<120; k++)
          ;
}

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
    0,      //end
```

```
    6,6,9,3,6,6,12,
    6,6,6,6,6,6,12,
    6,6,9,3,6,6,9 ,3,
    6,3,3,6,3,3,6 ,6,9,
    0      //end
};
Byte RDATA SOUNDTONE[ ] =
{
    0,     //end
    120,120,106,95,80,80,95,
    80 ,80 ,71 ,63,60,60,80,
    60 ,60 ,71 ,80,95,71,80,95 ,
    120,106,95 ,80,71,80,95,106,120,
    0      //end
};

//first do then error
void Music1(void)
{
    Byte  i=0,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    do
    {
      SoundLong=SOUNDLONG[i];
      SoundTone=SOUNDTONE[i];
      i++;

      //sound long
      for(j=0; j<SoundLong; j++)
      {
          //tempo
          for(k=0; k<12; k++)
        {
          //sound tone
          for(m=0; m<SoundTone; m++)
            P3_1 = 0;
          for(m=0; m<SoundTone; m++)
            P3_1 = 1;
        }
      }

    } while ( (SOUNDLONG[i]!=0x00)||(SOUNDTONE[i]!=0x00) );
}
```

而 while 的指令结构，其程序如下：

```
/*****************************/
void Music2(void)
{
    Byte  i=0,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    while ( (SOUNDLONG[i]!=0x00)||(SOUNDTONE[i]!=0x00) )
    {
       SoundLong=SOUNDLONG[i];
       SoundTone=SOUNDTONE[i];
       i++;

       //sound long
       for(j=0; j<SoundLong; j++)
       {
            //tempo
          for(k=0; k<12; k++)
          {
             //sound tone
             P3_1 = 0;
             for(m=0; m<SoundTone; m++)
                ;
             P3_1 = 1;
             for(m=0; m<SoundTone; m++)
                ;
          }
       }
    }
}
```

2-7 指针法 for 方式

演奏音乐函数利用指针方式来取得音长表和音调表的内容，其演奏音符以 for 循环来完成，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
```

```
#include  "mtv212m.h"
#define  SPEAKER   P3_1

void PowerOnInit(void);
void InitialCpu(void);
void InitialCpuIO(void);
void DelayX10ms(Word count);
void Music(void);

/*******************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      Music( );
      DelayX10ms(100);
   }
}

/*******************************/
void PowerOnInit(void)
{
   InitialCpu();

   InitialCpuIO();
}

void InitialCpu(void)
{
   IE  = 0;              //disable all interrupt
   PSW = 0;              //bank 0

   IP  = 0x0b;           //hi priority:int0,timer0,timer1

   TMOD= 0x11;           //set timer1,timer0 mode

   TR0 = 0;              //stop timer0
   TR1 = 0;              //stop timer1
   IT0 = 1;              //set int0:falling eage trigger

   EX0 = 0;              //disable int0   interrupt
   ET1 = 0;              //disable timer1 interrupt
   ET0 = 0;              //disable timer0 interrupt
```

```
    EA  = 1;              //enable all interrupt gate
}

void InitialCpuIO(void)
{
    SPEAKER = 0;
}

/*******************************/
vcid DelayX10ms(Word count)
{
    Word i,j,k;

    for(i=0; i<count; i++)
       for(j=0; j<10; j++)
          for(k=0; k<120; k++)
             ;
}

/*******************************/
Byte RDATA SOUNDLONG[ ] =
{
    6,6,9,3,6,6,12,
    6,6,6,6,6,6,12,
    6,6,9,3,6,6,9 ,3,
    6,3,3,6,3,3,6 ,6,9
};
Byte RDATA SOUNDTONE[ ] =
{
    120,120,106,95,80,80,95,
    80 ,80 ,71 ,63,60,60,80,
    60 ,60 ,71 ,80,95,71,80,95 ,
    120,106,95 ,80,71,80,95,106,120
};

void Music(void)
{
    Byte  i,j,k;
    Byte  SoundLong,SoundTone;
    Byte  *DataPointer1,*DataPointer2;
    Word  m;

    DataPointer1=SOUNDLONG;
    DataPointer2=SOUNDTONE;
```

```
    //sound count
    for(i=0; i<31; i++)
    {
       SoundLong=*(DataPointer1+i);
       SoundTone=*(DataPointer2+i);

       //sound long
       for(j=0; j<SoundLong; j++)
       {
            //tempo
          for(k=0; k<12; k++)
          {
             //sound tone
             SPEAKER = 1;
             for(m=0; m<SoundTone; m++)
                ;
             SPEAKER = 0;
             for(m=0; m<SoundTone; m++)
                ;
          }
       }
    }
}
```

其汇编程序如下：

```
Music:
 MOV       DataPointer1?549,#0FFH
 MOV       DataPointer1?549+01H,#HIGH (SOUNDLONG)
 MOV       DataPointer1?549+02H,#LOW (SOUNDLONG)
 MOV       DataPointer2?550,#0FFH
 MOV       DataPointer2?550+01H,#HIGH (SOUNDTONE)
 MOV       DataPointer2?550+02H,#LOW (SOUNDTONE)
 CLR       A
 MOV       i?544,A
?C0017:
 MOV       R3,DataPointer1?549
 MOV       R2,DataPointer1?549+01H
 MOV       R1,DataPointer1?549+02H
 MOV       DPL,i?544
 MOV       DPH,#0CH
 LCALL     ?C?CLDOPTR
 MOV       SoundLong?547,A
 MOV       R3,DataPointer2?550
```

```
MOV     R2,DataPointer2?550+01H
MOV     R1,DataPointer2?550+02H
MOV     DPL,i?544
MOV     DPH,#00H
LCALL   ?C?CLDOPTR
MOV     R1,A
CLR     A
MOV     R7,A
?C0020:
MOV     A,R7
CLR     C
SUBB    A,SoundLong?547
JNC     ?C0019
CLR     A
MOV     R6,A
?C0023:
SETB    P3_1
CLR     A
MOV     R5,A
MOV     R4,A
?C0026:
MOV     A,R1
MOV     R3,A
CLR     C
MOV     A,R5
SUBB    A,R3
MOV     A,R4
SUBB    A,#00H
JNC     ?C0027
INC     R5
CJNE    R5,#00H,?C0036
INC     R4
?C0036:
SJMP    ?C0026
?C0027:
CLR     P3_1
CLR     A
MOV     R4,A
MOV     R5,A
?C0029:
MOV     A,R1
MOV     R3,A
CLR     C
MOV     A,R5
SUBB    A,R3
```

```
MOV       A,R4
SUBB      A,#00H
JNC       ?C0025
INC       R5
CJNE      R5,#00H,?C0037
INC       R4
?C0037:
SJMP      ?C0029
?C0025:
INC       R6
CJNE      R6,#0CH,?C0023
?C0022:
INC       R7
SJMP      ?C0020
?C0019:
INC       i?544
MOV       A,i?544
XRL       A,#01FH
JNZ       ?C0017,
?C0032:
RET
```

2-8　指针法 while 方式

演奏音乐函数除了利用指针方式，也以结束符号 0X00 来作为判断音符是否结束的依据，而不需要用 for 循环计算出每一首曲目音符个数的麻烦，如下所示：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#define  SPEAKER   P3_1

void PowerOnInit(void);
void InitialCpu(void);
void InitialCpuIO(void);
void DelayX10ms(Word count);
void Music(void);

/*****************************/
void Main( void )
{
```

```
    PowerOnInit();

    while( 1 )
    {
      Music( );
      DelayX10ms(100);
    }
}

/*****************************/
void PowerOnInit(void)
{
    InitialCpu();

    InitialCpuIO();
}

void InitialCpu(void)
{
    IE  = 0;              //disable all interrupt
    PSW = 0;              //bank 0

    IP  = 0x0b;           //hi priority:int0,timer0,timer1

    TMOD= 0x11;           //set timer1,timer0 mode

    TR0 = 0;              //stop timer0
    TR1 = 0;              //stop timer1
    IT0 = 1;              //set int0:falling eage trigger

    EX0 = 0;              //disable int0  interrupt
    ET1 = 0;              //disable timer1 interrupt
    ET0 = 0;              //disable timer0 interrupt

    EA  = 1;              //enable all interrupt gate
}

void InitialCpuIO(void)
{
    SPEAKER = 0;
}

/*****************************/
void DelayX10ms(Word count)
{
```

```
    Word i,j,k;

    for(i=0; i<count; i++)
       for(j=0; j<10; j++)
          for(k=0; k<120; k++)
             ;
}

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
    6,6,9,3,6,6,12,
    6,6,6,6,6,6,12,
    6,6,9,3,6,6,9 ,3,
    6,3,3,6,3,3,6 ,6,9,
    0 //end
};
Byte RDATA SOUNDTONE[ ] =
{
    120,120,106,95,80,80,95,
    80 ,80 ,71 ,63,60,60,80,
    60 ,60 ,71 ,80,95,71,80,95 ,
    120,106,95 ,80,71,80,95,106,120,
    0 //end
};

void Music(void)
{
    Byte  j,k;
    Byte  SoundLong,SoundTone;
    Byte  *DataPointer1,*DataPointer2;
    Word  m;

    DataPointer1=SOUNDLONG;
    DataPointer2=SOUNDTONE;

    //sound count
    while( (*DataPointer1!=0x00)||(*DataPointer2!=0x00) )
    {
       SoundLong=*DataPointer1;
       SoundTone=*DataPointer2;
       DataPointer1++;
       DataPointer2++;

       //sound long
```

```
        for(j=0; j<SoundLong; j++)
        {
              //tempo
           for(k=0; k<12; k++)
           {
              //sound tone
              SPEAKER = 1;
              for(m=0; m<SoundTone; m++)
                 ;
              SPEAKER = 0;
              for(m=0; m<SoundTone; m++)
                 ;
           }
        }
     }
}
```

其汇编程序如下：

```
Music:
 MOV       DataPointer1?548,#0FFH
 MOV       DataPointer1?548+01H,#HIGH (SOUNDLONG)
 MOV       DataPointer1?548+02H,#LOW (SOUNDLONG)
 MOV       DataPointer2?549,#0FFH
 MOV       DataPointer2?549+01H,#HIGH (SOUNDTONE)
 MOV       DataPointer2?549+02H,#LOW (SOUNDTONE)
?C0017:
 MOV       R3,DataPointer1?548
 MOV       R2,DataPointer1?548+01H
 MOV       R1,DataPointer1?548+02H
 LCALL     ?C?CLDPTR
 MOV       R7,A
 JNZ       ?C0019
 MOV       R3,DataPointer2?549
 MOV       R2,DataPointer2?549+01H
 MOV       R1,DataPointer2?549+02H
 LCALL     ?C?CLDPTR
 JZ   ?C0032
?C0019:
 MOV       SoundLong?546,R7
 MOV       R3,DataPointer2?549
 MOV       R2,DataPointer2?549+01H
 MOV       R1,DataPointer2?549+02H
 LCALL     ?C?CLDPTR
 MOV       SoundTone?547,A
```

```
MOV     A,#01H
ADD     A,DataPointer1?548+02H
MOV     DataPointer1?548+02H,A
CLR     A
ADDC    A,DataPointer1?548+01H
MOV     DataPointer1?548+01H,A
MOV     A,#01H
ADD     A,DataPointer2?549+02H
MOV     DataPointer2?549+02H,A
CLR     A
ADDC    A,DataPointer2?549+01H
MOV     DataPointer2?549+01H,A
CLR     A
MOV     j?544,A
?C0020:
MOV     A,j?544
CLR     C
SUBB    A,SoundLong?546
JNC     ?C0017
CLR     A
MOV     R7,A
?C0023:
SETB    P3_1
CLR     A
MOV     R5,A
MOV     R4,A
?C0026:
MOV     A,SoundTone?547
MOV     R3,A
CLR     C
MOV     A,R5
SUBB    A,R3
MOV     A,R4
SUBB    A,#00H
JNC     ?C0027
INC     R5
CJNE    R5,#00H,?C0036
INC     R4
?C0036:
SJMP    ?C0026
?C0027:
CLR     P3_1
CLR     A
MOV     R4,A
MOV     R5,A
```

```
?C0029:
MOV       A,SoundTone?547
MOV       R3,A
CLR       C
MOV       A,R5
SUBB      A,R3
MOV       A,R4
SUBB      A,#00H
JNC       ?C0025
INC       R5
CJNE      R5,#00H,?C0037
INC       R4
?C0037:
SJMP      ?C0029
?C0025:
INC       R7
CJNE      R7,#0CH,?C0023
?C0022:
INC       j?544
SJMP      ?C0020
?C0032:
RET
```

2-9 2个字节的音调表

音调表以2个字节来表示音符的频率，因此，可以演奏出很低的频率音符，以适合各种曲目的应用，其中音调表中的第一个字节为高字节，第二个字节为低字节，而程序中SoundTone变量也必须将高、低字节作运算后才能写入，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#define  SPEAKER   P3_1

void PowerOnInit(void);
void InitialCpu(void);
void InitialCpuIO(void);
void DelayX10ms(Word count);
void Music(void);
```

```
/*******************************/
void Main( void )
{
    PowerOnInit();

    while( 1 )
    {
      Music( );
      DelayX10ms(100);
    }
}

/*******************************/
void PowerOnInit(void)
{
    InitialCpu();

    InitialCpuIO();
}

void InitialCpu(void)
{
    IE  = 0;              //disable all interrupt
    PSW = 0;              //bank 0

    IP  = 0x0b;           //hi priority:int0,timer0,timer1

    TMOD= 0x11;           //set timer1,timer0 mode

    TR0 = 0;              //stop timer0
    TR1 = 0;              //stop timer1
    IT0 = 1;              //set int0:falling eage trigger

    EX0 = 0;              //disable int0   interrupt
    ET1 = 0;              //disable timer1 interrupt
    ET0 = 0;              //disable timer0 interrupt

    EA  = 1;              //enable all interrupt gate
}

void InitialCpuIO(void)
{
    SPEAKER = 0;
}
```

```
/*****************************/
void DelayX10ms(Word count)
{
   Word i,j,k;

   for(i=0; i<count; i++)
      for(j=0; j<10; j++)
         for(k=0; k<120; k++)
            ;
}

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
   6,6,9,3,6,6,12,
   6,6,6,6,6,6,12,
   6,6,9,3,6,6,9 ,3,
   6,3,3,6,3,3,6 ,6,9
};
Byte RDATA SOUNDTONE[ ] =
{
   0,120,0,120,0,106,0,95,0,80,0,80,0,95,
   0,80 ,0,80 ,0,71 ,0,63,0,60,0,60,0,80,
   0,60 ,0,60 ,0,71 ,0,80,0,95,0,71,0,80,0,95 ,
   0,120,0,106,0,95 ,0,80,0,71,0,80,0,95,0,106,0,120
};

void Music(void)
{
   Byte  i,j,k,SoundLong;
   Word  m,SoundTone;

   //sound count
   for(i=0; i<31; i++)
   {
      SoundLong = SOUNDLONG[i];
      SoundTone = SOUNDTONE[i*2] << 8 ;
      SoundTone += SOUNDTONE[i*2+1];

      //sound long
      for(j=0; j<SoundLong; j++)
      {
            //tempo
         for(k=0; k<12; k++)
```

```
            {
               //sound tone
               SPEAKER = 1;
               for(m=0; m<SoundTone; m++)
                  ;
               SPEAKER = 0;
               for(m=0; m<SoundTone; m++)
                  ;
            }
         }
      }
}
```

其汇编程序如下：

```
SOUNDLONG:
 DB   006H,006H,009H,003H,006H
 DB   006H,00CH,006H,006H,006H
 DB   006H,006H,006H,00CH,006H
 DB   006H,009H,003H,006H,006H
 DB   009H,003H,006H,003H,003H
 DB   006H,003H,003H,006H,006H
 DB   009H

SOUNDTONE:
 DB   000H,078H,000H,078H
 DB   000H,06AH,000H,05FH
 DB   000H,050H,000H,050H
 DB   000H,05FH,000H,050H
 DB   000H,050H,000H,047H
 DB   000H,03FH,000H,03CH
 DB   000H,03CH,000H,050H
 DB   000H,03CH,000H,03CH
 DB   000H,047H,000H,050H
 DB   000H,05FH,000H,047H
 DB   000H,050H,000H,05FH
 DB   000H,078H,000H,06AH
 DB   000H,05FH,000H,050H
 DB   000H,047H,000H,050H
 DB   000H,05FH,000H,06AH
 DB   000H,078H

Music:
 CLR       A
 MOV       R5,A
```

```
?C0017:
 MOV       A,R5
 MOV       DPTR,#SOUNDLONG
 MOVC      A,@A+DPTR
 MOV       R4,A
 MOV       A,R5
 ADD       A,ACC
 ADD       A,#LOW (SOUNDTONE)
 MOV       DPL,A
 CLR       A
 ADDC      A,#HIGH (SOUNDTONE)
 MOV       DPH,A
 CLR       A
 MOVC      A,@A+DPTR
 MOV       SoundTone?549+01H,#00H
 MOV       SoundTone?549,A
 MOV       A,R5
 ADD       A,ACC
 ADD       A,#LOW (SOUNDTONE+01H)
 MOV       DPL,A
 CLR       A
 ADDC      A,#HIGH (SOUNDTONE+01H)
 MOV       DPH,A
 CLR       A
 MOVC      A,@A+DPTR
 ADD       A,SoundTone?549+01H
 MOV       SoundTone?549+01H,A
 CLR       A
 ADDC      A,SoundTone?549
 MOV       SoundTone?549,A
 CLR       A
 MOV       R7,A
?C0020:
 MOV       A,R7
 CLR       C
 SUBB      A,R4
 JNC       ?C0019
 CLR       A
 MOV       R6,A
?C0023:
 SETB      P3_1
 CLR       A
 MOV       R3,A
 MOV       R2,A
?C0026:
```

```
CLR     C
MOV     A,R3
SUBB    A,SoundTone?549+01H
MOV     A,R2
SUBB    A,SoundTone?549
JNC     ?C0027
INC     R3
CJNE    R3,#00H,?C0036
INC     R2
?C0036:
SJMP    ?C0026
?C0027:
CLR     P3_1
CLR     A
MOV     R2,A
MOV     R3,A
?C0029:
CLR     C
MOV     A,R3
SUBB    A,SoundTone?549+01H
MOV     A,R2
SUBB    A,SoundTone?549
JNC     ?C0025
INC     R3
CJNE    R3,#00H,?C0037
INC     R2
?C0037:
SJMP    ?C0029
?C0025:
INC     R6
CJNE    R6,#0CH,?C0023
?C0022:
INC     R7
SJMP    ?C0020
?C0019:
INC     R5
MOV     A,R5
XRL     A,#01FH
JNZ     ?C0017
?C0032:
RET
```

2-10 类型的音调表

将音调表声明成int类型，则当下标i变量增加1，将略过两个字节而取得正确的音调数值，程序也无需将高低字节运算后才能写入至SoundTone变量，而将取得音调表的数值直接写入即可，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#define  SPEAKER   P3_1

void PowerOnInit(void);
void InitialCpu(void);
void InitialCpuIO(void);
void DelayX10ms(Word count);
void Music(void);

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      Music( );
      DelayX10ms(100);
   }
}

/*****************************/
void PowerOnInit(void)
{
   InitialCpu();

   InitialCpuIO();
}

void InitialCpu(void)
{
   IE  = 0;              //disable all interrupt
   PSW = 0;              //bank 0
```

```
    IP  = 0x0b;          //hi priority:int0,timer0,timer1

    TMOD= 0x11;          //set timer1,timer0 mode

    TR0 = 0;             //stop timer0
    TR1 = 0;             //stop timer1
    IT0 = 1;             //set int0:falling eage trigger

    EX0 = 0;             //disable int0  interrupt
    ET1 = 0;             //disable timer1 interrupt
    ET0 = 0;             //disable timer0 interrupt

    EA  = 1;             //enable all interrupt gate
}

void InitialCpuIO(void)
{
    SPEAKER = 0;
}

/*****************************/
void DelayX10ms(Word count)
{
    Word i,j,k;

    for(i=0; i<count; i++)
      for(j=0; j<10; j++)
        for(k=0; k<120; k++)
          ;
}

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
    6,6,9,3,6,6,12,
    6,6,6,6,6,6,12,
    6,6,9,3,6,6,9 ,3,
    6,3,3,6,3,3,6 ,6,9
};
Word RDATA SOUNDTONE[ ] =
{
    120,120,106,95,80,80,95,
    80 ,80 ,71 ,63,60,60,80,
    60 ,60 ,71 ,80,95,71,80,95 ,
```

```
    120,106,95 ,80,71,80,95,106,120
};

void Music(void)
{
    Byte  i,j,k,SoundLong;
    Word  m,SoundTone;

    //sound count
    for(i=0; i<31; i++)
    {
       SoundLong = SOUNDLONG[i];
       SoundTone = SOUNDTONE[i];
       //sound long
       for(j=0; j<SoundLong; j++)
       {
            //tempo
         for(k=0; k<12; k++)
         {
            //sound tone
            SPEAKER = 1;
            for(m=0; m<SoundTone; m++)
               ;
            SPEAKER = 0;
            for(m=0; m<SoundTone; m++)
               ;
         }
       }
    }
}
```

其汇编程序如下：

```
SOUNDLONG:
 DB   006H,006H,009H,003H,006H
 DB   006H,00CH,006H,006H,006H
 DB   006H,006H,006H,00CH,006H
 DB   006H,009H,003H,006H,006H
 DB   009H,003H,006H,003H,003H
 DB   006H,003H,003H,006H,006H
 DB   009H

SOUNDTONE:
 DW   00078H,00078H,0006AH,0005FH
 DW   00050H,00050H,0005FH,00050H
```

```
DW   00050H,00047H,0003FH,0003CH
DW   0003CH,00050H,0003CH,0003CH
DW   00047H,00050H,0005FH,00047H
DW   00050H,0005FH,00078H,0006AH
DW   0005FH,00050H,00047H,00050H
DW   0005FH,0006AH,00078H

Music:
CLR      A
MOV      R7,A
?C0017:
MOV      A,R7
MOV      DPTR,#SOUNDLONG
MOVC     A,@A+DPTR
MOV      R6,A
MOV      A,R7
ADD      A,ACC
ADD      A,#LOW (SOUNDTONE)
MOV      DPL,A
CLR      A
ADDC     A,#HIGH (SOUNDTONE)
MOV      DPH,A
CLR      A
MOVC     A,@A+DPTR
MOV      R4,A
MOV      A,#01H
MOVC     A,@A+DPTR
MOV      SoundTone?549,R4
MOV      SoundTone?549+01H,A
CLR      A
MOV      R5,A
?C0020:
MOV      A,R5
CLR      C
SUBB     A,R6
JNC      ?C0019
CLR      A
MOV      R4,A
?C0023:
SETB     P3_1
CLR      A
MOV      R3,A
MOV      R2,A
?C0026:
CLR      C
```

```
MOV     A,R3
SUBB    A,SoundTone?549+01H
MOV     A,R2
SUBB    A,SoundTone?549
JNC     ?C0027
INC     R3
CJNE    R3,#00H,?C0036
INC     R2
?C0036:
SJMP    ?C0026
?C0027:
CLR     P3_1
CLR     A
MOV     R2,A
MOV     R3,A
?C0029:
CLR     C
MOV     A,R3
SUBB    A,SoundTone?549+01H
MOV     A,R2
SUBB    A,SoundTone?549
JNC     ?C0025
INC     R3
CJNE    R3,#00H,?C0037
INC     R2
?C0037:
SJMP    ?C0029
?C0025:
INC     R4
CJNE    R4,#0CH,?C0023
?C0022:
INC     R5
SJMP    ?C0020
?C0019:
INC     R7
CJNE    R7,#01FH,?C0017
?C0032:
RET
```

2-11 变量选曲 switch 和 for 方式

利用自变量作为演奏曲目的编号，可弹性地选择想要演奏的歌曲，而音长表和

音调表也必须将曲目编号的所有音符连续地编写下去，只有曲目编号不同时，其 for 循环的起始值和条件式需累积计算下去，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#define  SPEAKER   P3_1

Byte  IDATA   MusicNum;

void PowerOnInit(void);
void InitialCpu(void);
void InitialCpuIO(void);
void DelayX10ms(Word count);
void Music(Byte number);

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      Music(2);
      DelayX10ms(100);
      Music(1);
      DelayX10ms(100);
   }
}

/*****************************/
void PowerOnInit(void)
{
   InitialCpu();

   InitialCpuIO();
}

void InitialCpu(void)
{
   IE  = 0;              //disable all interrupt
   PSW = 0;              //bank 0
```

```
    IP  = 0x0b;          //hi priority:int0,timer0,timer1

    TMOD= 0x11;          //set timer1,timer0 mode

    TR0 = 0;             //stop timer0
    TR1 = 0;             //stop timer1
    IT0 = 1;             //set int0:falling eage trigger

    EX0 = 0;             //disable int0  interrupt
    ET1 = 0;             //disable timer1 interrupt
    ET0 = 0;             //disable timer0 interrupt

    EA  = 1;             //enable all interrupt gate
}

void InitialCpuIO(void)
{
    SPEAKER = 0;
}

/*******************************/
void DelayX10ms(Word count)
{
    Word i,j,k;

    for(i=0; i<count; i++)
      for(j=0; j<10; j++)
        for(k=0; k<120; k++)
          ;
}

/*******************************/
Byte RDATA SOUNDLONG[ ] =
{
    6 ,6 ,9 ,3 ,6 ,6 ,12,
    6 ,6 ,6 ,6 ,6 ,6 ,12,
    6 ,6 ,9 ,3 ,6 ,6 ,9 ,3 ,
    6 ,3 ,3 ,6 ,3 ,3 ,6 ,6 ,9 ,

    12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
    12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
    12,6 ,6 ,6 ,6 ,6 ,6 ,
    6 ,12,6 ,24,12,12,12,6 ,6 ,
    6 ,12,6 ,24
};
```

```
Byte RDATA SOUNDTONE[ ] =
{
    120,120,106,95 ,80 ,80 ,95,
    80 ,80 ,71 ,63 ,60 ,60 ,80,
    60 ,60 ,71 ,80 ,95 ,71 ,80,95 ,
    120,106,95 ,80 ,71 ,80 ,95,106,120,

    71,80,95,71 ,80,95,71 ,71 ,80 ,71,
    71,80,95,71 ,80,95,106,106,120,106,
    95,95,80,71 ,60,71,80 ,
    95,95,80,120,95,95,95 ,95 ,95 ,
    71,71,80,71
};

void Music(Byte number)
{
    Byte  i,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    MusicNum=number;

    switch( MusicNum )
    {
     case 1 :
        for(i=0; i<31; i++)
        {
           SoundLong=SOUNDLONG[i];
           SoundTone=SOUNDTONE[i];

           //sound long
           for(j=0; j<SoundLong; j++)
           {
              //tempo
              for(k=0; k<12; k++)
              {
                 //sound tone
                 SPEAKER = 1;
                 for(m=0; m<SoundTone; m++)
                    ;
                 SPEAKER = 0;
                 for(m=0; m<SoundTone; m++)
                    ;
              }
           }
```

```
        }
        break;
    case 2 :
        for(i=31; i<71; i++)
        {
            SoundLong=SOUNDLONG[i];
            SoundTone=SOUNDTONE[i];

            //sound long
            for(j=0; j<SoundLong; j++)
            {
                //tempo
                for(k=0; k<12; k++)
                {
                    //sound tone
                    SPEAKER = 1;
                    for(m=0; m<SoundTone; m++)
                        ;
                    SPEAKER = 0;
                    for(m=0; m<SoundTone; m++)
                        ;
                }
            }
        }
        break;
    default :
        break;
    }
}
```

其汇编程序如下：

```
SOUNDLONG:
DB   006H,006H,009H,003H,006H
DB   006H,00CH,006H,006H,006H
DB   006H,006H,006H,00CH,006H
DB   006H,009H,003H,006H,006H
DB   009H,003H,006H,003H,003H
DB   006H,003H,003H,006H,006H
DB   009H
DB   00CH,006H,006H,00CH,006H
DB   006H,006H,00CH,006H,018H
DB   00CH,006H,006H,00CH,006H
DB   006H,006H,00CH,006H,018H
DB   00CH,006H,006H,006H,006H
```

```
 DB   006H,006H,006H,00CH,006H
 DB   018H,00CH,00CH,00CH,006H
 DB   006H,006H,00CH,006H,018H

SOUNDTONE:
 DB   078H,078H,06AH,05FH,050H
 DB   050H,05FH,050H,050H,047H
 DB   03FH,03CH,03CH,050H,03CH
 DB   03CH,047H,050H,05FH,047H
 DB   050H,05FH,078H,06AH,05FH
 DB   050H,047H,050H,05FH,06AH
 DB   078H
 DB   047H,050H,05FH,047H,050H
 DB   05FH,047H,047H,050H,047H
 DB   047H,050H,05FH,047H,050H
 DB   05FH,06AH,06AH,078H,06AH
 DB   05FH,05FH,050H,047H,03CH
 DB   047H,050H,05FH,05FH,050H
 DB   078H,05FH,05FH,05FH,05FH
 DB   05FH,047H,047H,050H,047H

_Music:
 MOV       R0,#MusicNum
 MOV       A,R7
 MOV       @R0,A
 ADD       A,#0FEH
 JZ   ?C0034
 INC       A
 JZ   $ + 5H
 LJMP      ?C0051
?C0018:
 CLR       A
 MOV       R3,A
?C0019:
 MOV       A,R3
 MOV       DPTR,#SOUNDLONG
 MOVC      A,@A+DPTR
 MOV       SoundLong?548,A
 MOV       A,R3
 MOV       DPTR,#SOUNDTONE
 MOVC      A,@A+DPTR
 MOV       SoundTone?549,A
 CLR       A
 MOV       R1,A
?C0022:
```

```
MOV     A,R1
CLR     C
SUBB    A,SoundLong?548
JNC     ?C0021
CLR     A
MOV     R2,A
?C0025:
SETB    P3_1
CLR     A
MOV     R4,A
MOV     R5,A
?C0028:
CLR     C
MOV     A,R5
SUBB    A,SoundTone?549
MOV     A,R4
SUBB    A,#00H
JNC     ?C0029
INC     R5
CJNE    R5,#00H,?C0055
INC     R4
?C0055:
SJMP    ?C0028
?C0029:
CLR     P3_1
CLR     A
MOV     R4,A
MOV     R5,A
?C0031:
CLR     C
MOV     A,R5
SUBB    A,SoundTone?549
MOV     A,R4
SUBB    A,#00H
JNC     ?C0027
INC     R5
CJNE    R5,#00H,?C0056
INC     R4
?C0056:
SJMP    ?C0031
?C0027:
INC     R2
CJNE    R2,#0CH,?C0025
?C0024:
INC     R1
```

```
 SJMP     ?C0022
?C0021:
 INC      R3
 CJNE     R3,#01FH,?C0019
 RET
?C0034:
 MOV      R3,#01FH
?C0035:
 MOV      A,R3
 MOV      DPTR,#SOUNDLONG
 MOVC     A,@A+DPTR
 MOV      SoundLong?548,A
 MOV      A,R3
 MOV      DPTR,#SOUNDTONE
 MOVC     A,@A+DPTR
 MOV      SoundTone?549,A
 CLR      A
 MOV      R1,A
?C0038:
 MOV      A,R1
 CLR      C
 SUBB     A,SoundLong?548
 JNC      ?C0037
 CLR      A
 MOV      R2,A
?C0041:
 SETB     P3_1
 CLR      A
 MOV      R4,A
 MOV      R5,A
?C0044:
 CLR      C
 MOV      A,R5
 SUBB     A,SoundTone?549
 MOV      A,R4
 SUBB     A,#00H
 JNC      ?C0045
 INC      R5
 CJNE     R5,#00H,?C0057
 INC      R4
?C0057:
 SJMP     ?C0044
?C0045:
 CLR      P3_1
 CLR      A
```

```
MOV       R4,A
MOV       R5,A
?C0047:
CLR       C
MOV       A,R5
SUBB      A,SoundTone?549
MOV       A,R4
SUBB      A,#00H
JNC       ?C0043
INC       R5
CJNE      R5,#00H,?C0058
INC       R4
?C0058:
SJMP      ?C0047
?C0043:
INC       R2
CJNE      R2,#0CH,?C0041
?C0040:
INC       R1
SJMP      ?C0038
?C0037:
INC       R3
CJNE      R3,#047H,?C0035
RET
?C0051:
RET
```

2-12 变量选曲 while 方式

利用 while 方式需要计算出每一首曲目的起始地址，以取得音长表和音调表的数值。首先以自变量来判断曲目中所有音符的音长和音调，以曲目中结束符号 0X00 来作为地址的依据，当取得的音长和音调数值不是 0X00，则索引地址 i 变量就不断地累加 1，当发现是 0X00，则索引地址也要跳过此结束符号而再加 1，则此索引地址就变成了实际要演奏音符的起始地址了，以取得正确的音长和音调数值，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#define  SPEAKER   P3_1
```

```
void PowerOnInit(void);
void InitialCpu(void);
void InitialCpuIO(void);
void DelayX10ms(Word count);
void Music(Byte number);

/*****************************/
void Main( void )
{
   PowerOnInit( );

   while( 1 )
   {
      Music(2);
      DelayX10ms(100);
      Music(1);
      DelayX10ms(100);
   }
}

/*****************************/
void PowerOnInit(void)
{
   InitialCpu( );

   InitialCpuIO( );
}

void InitialCpu(void)
{
   IE  = 0;              //disable all interrupt
   PSW = 0;              //bank 0

   IP  = 0x0b;           //hi priority:int0,timer0,timer1

   TMOD= 0x11;           //set timer1,timer0 mode

   TR0 = 0;              //stop timer0
   TR1 = 0;              /stop timer1
   IT0 = 1;              //set int0:falling eage trigger

   EX0 = 0;              //disable int0   interrupt
   ET1 = 0;              //disable timer1 interrupt
   ET0 = 0;              //disable timer0 interrupt
```

```
    EA  = 1;              //enable all interrupt gate
}

void InitialCpuIO(void)
{
    SPEAKER = 0;
}

/*******************************/
void DelayX10ms(Word count)
{
    Word i,j,k;

    for(i=0; i<count; i++)
      for(j=0; j<10; j++)
        for(k=0; k<120; k++)
          ;
}

/*******************************/
Byte RDATA SOUNDLONG[ ] =
{
    6 ,6 ,9 ,3 ,6 ,6 ,12,
    6 ,6 ,6 ,6 ,6 ,6 ,12,
    6 ,6 ,9 ,3 ,6 ,6 ,9 ,3 ,
    6 ,3 ,3 ,6 ,3 ,3 ,6 ,6 ,9,
    0 ,    //end
    12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
    12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
    12,6 ,6 ,6 ,6 ,6 ,6 ,
    6 ,12,6 ,24,12,12,12,6 ,6 ,
    6 ,12,6 ,24,
    0 //end
};
Byte RDATA SOUNDTONE[ ] =
{
    120,120,106,95 ,80 ,80 ,95,
    80 ,80 ,71 ,63 ,60 ,60 ,80,
    60 ,60 ,71 ,80 ,95 ,71 ,80,95 ,
    120,106,95 ,80 ,71 ,80 ,95,106,120,
    0,     //end
    71,80,95,71 ,80,95,71 ,71 ,80 ,71,
    71,80,95,71 ,80,95,106,106,120,106,
    95,95,80,71 ,60,71,80 ,
```

```
    95,95,80,120,95,95,95 ,95 ,95 ,
    71,71,80,71 ,
    0 //end
};

void Music(Byte number)
{
    Byte  k,n;
    Byte  SoundLong,SoundTone;
    Word  i=0,j=0,m;

    for(k=0; k<(number-1); k++)
    {
       while( SOUNDLONG[i] != 0x00 )
       {
          i++;
       }
       i++;
    }
    for(k=0; k<(number-1); k++)
    {
       while( SOUNDTONE[j] != 0x00 )
       {
          j++;
       }
       j++;
    }

    do
    {
       SoundLong=SOUNDLONG[i];
       SoundTone=SOUNDTONE[j];
       i++;
       j++;

       //sound long
       for(n=0; n<SoundLong; n++)
       {
            //tempo
          for(k=0; k<12; k++)
          {
             SPEAKER = 1;
             for(m=0; m<SoundTone; m++)
                ;
             SPEAKER = 0;
```

```
            for(m=0; m<SoundTone; m++)
               ;
         }
      }

   } while ( (SOUNDLONG[i]!=0x00)||(SOUNDTONE[j]!=0x00) );
}
```

其汇编程序如下：

```
_Music:
 CLR       A
 MOV       i?549,A
 MOV       i?549+01H,A
 MOV       j?550,A
 MOV       j?550+01H,A
 MOV       R6,A
?C0017:
 MOV       A,R7
 DEC       A
 MOV       R5,A
 MOV       A,R6
 CLR       C
 SUBB      A,R5
 JNC       ?C0018
?C0020:
 MOV       A,i?549+01H
 MOV       DPTR,#SOUNDLONG
 MOVC      A,@A+DPTR
 JZ    ?C0021
 INC       i?549+01H
 MOV       A,i?549+01H
 JNZ       ?C0046
 INC       i?549
?C0046:
 SJMP      ?C0020
?C0021:
 INC       i?549+01H
 MOV       A,i?549+01H
 JNZ       ?C0047
 INC       i?549
?C0047:
 INC       R6
 SJMP      ?C0017
?C0018:
```

```
CLR     A
MOV     R6,A
?C0022:
MOV     A,R7
DEC     A
MOV     R5,A
MOV     A,R6
CLR     C
SUBB    A,R5
JNC     ?C0029
?C0025:
MOV     A,j?550+01H
MOV     DPTR,#SOUNDTONE
MOVC    A,@A+DPTR
JZ  ?C0026
INC     j?550+01H
MOV     A,j?550+01H
JNZ     ?C0048
INC     j?550
?C0048:
SJMP    ?C0025
?C0026:
INC     j?550+01H
MOV     A,j?550+01H
JNZ     ?C0049
INC     j?550
?C0049:
INC     R6
SJMP    ?C0022
?C0029:
MOV     A,i?549+01H
MOV     DPTR,#SOUNDLONG
MOVC    A,@A+DPTR
MOV     SoundLong?547,A
MOV     A,j?550+01H
MOV     DPTR,#SOUNDTONE
MOVC    A,@A+DPTR
MOV     R1,A
INC     i?549+01H
MOV     A,i?549+01H
JNZ     ?C0050
INC     i?549
?C0050:
INC     j?550+01H
MOV     A,j?550+01H
```

```
        JNZ     ?C0051
        INC     j?550
?C0051:
        CLR     A
        MOV     R7,A
?C0030:
        MOV     A,R7
        CLR     C
        SUBB    A,SoundLong?547
        JNC     ?C0027
        CLR     A
        MOV     R6,A
?C0033:
        SETB    P3_1
        CLR     A
        MOV     R5,A
        MOV     R4,A
?C0036:
        MOV     A,R1
        MOV     R3,A
        CLR     C
        MOV     A,R5
        SUBB    A,R3
        MOV     A,R4
        SUBB    A,#00H
        JNC     ?C0037
        INC     R5
        CJNE    R5,#00H,?C0052
        INC     R4
?C0052:
        SJMP    ?C0036
?C0037:
        CLR     P3_1
        CLR     A
        MOV     R4,A
        MOV     R5,A
?C0039:
        MOV     A,R1
        MOV     R3,A
        CLR     C
        MOV     A,R5
        SUBB    A,R3
        MOV     A,R4
        SUBB    A,#00H
        JNC     ?C0035
```

```
INC       R5
CJNE      R5,#00H,?C0053
INC       R4
?C0053:
SJMP      ?C0039
?C0035:
INC       R6
CJNE      R6,#0CH,?C0033
?C0032:
INC       R7
SJMP      ?C0030
?C0027:
MOV       A,i?549+01H
MOV       DPTR,#SOUNDLONG
MOVC      A,@A+DPTR
JNZ       ?C0029
MOV       A,j?550+01H
MOV       DPTR,#SOUNDTONE
MOVC      A,@A+DPTR
JNZ       ?C0029
RET
```

2-13　变量选曲 if...else 方式

利用 Music( )函数以 if 指令来判断自变量，并将自变量作为曲目编号，当曲目编号为 1 则执行 Music1( )函数，当曲目编号为 2 则执行 Music2( )函数，依此类推，使得程序结构变得清晰易懂，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#define  SPEAKER   P3_1

void PowerOnInit(void);
void InitialCpu(void);
void InitialCpuIO(void);
void DelayX10ms(Word count);
void Music(Byte number);
void Music1(void);
void Music2(void);
```

```
/*******************************/
void Main( void )
{
   PowerOnInit( );

   while( 1 )
   {
      Music(2);
      DelayX10ms(100);
      Music(1);
      DelayX10ms(100);
   }
}

/*******************************/
void PowerOnInit(void)
{
   InitialCpu( );

   InitialCpuIO( );
}

void InitialCpu(void)
{
   IE  = 0;              //disable all interrupt
   PSW = 0;              //bank 0

   IP  = 0x0b;           //hi priority:int0,timer0,timer1

   TMOD= 0x11;           //set timer1,timer0 mode

   TR0 = 0;              //stop timer0
   TR1 = 0;              //stop timer1
   IT0 = 1;              //set int0:falling eage trigger

   EX0 = 0;              //disable int0  interrupt
   ET1 = 0;              //disable timer1 interrupt
   ET0 = 0;              //disable timer0 interrupt

   EA  = 1;              //enable all interrupt gate
}

void InitialCpuIO(void)
{
```

```
    SPEAKER = 0;
}

/*****************************/
void DelayX10ms(Word count)
{
    Word i,j,k;

    for(i=0; i<count; i++)
       for(j=0; j<10; j++)
          for(k=0; k<120; k++)
             ;
}

/*****************************/
void Music(Byte number)
{
    if ( number==1 )
       Music1( );
    else if ( number==2 )
       Music2( );
}

/*****************************/
Byte RDATA SOUNDLONG1[ ] =
{
    6 ,6 ,9 ,3 ,6 ,6 ,12,
    6 ,6 ,6 ,6 ,6 ,6 ,12,
    6 ,6 ,9 ,3 ,6 ,6 ,9 ,3 ,
    6 ,3 ,3 ,6 ,3 ,3 ,6 ,6 ,9
};
Byte RDATA SOUNDTONE1[ ] =
{
    120,120,106,95 ,80 ,80 ,95,
    80 ,80 ,71 ,63 ,60 ,60 ,80,
    60 ,60 ,71 ,80 ,95 ,71 ,80,95 ,
    120,106,95 ,80 ,71 ,80 ,95,106,120
};

void Music1(void)
{
    Byte  i,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;
```

```
    for(i=0; i<31; i++)
    {
       SoundLong=SOUNDLONG1[i];
       SoundTone=SOUNDTONE1[i];

       //sound long
       for(j=0; j<SoundLong; j++)
       {
          //tempo
          for(k=0; k<12; k++)
          {
             //sound tone
             SPEAKER = 1;
             for(m=0; m<SoundTone; m++)
                ;
             SPEAKER = 0;
             for(m=0; m<SoundTone; m++)
                ;
          }
       }
    }
}

/*****************************/
Byte RDATA SOUNDLONG2[ ] =
{
   12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
   12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
   12,6 ,6 ,6 ,6 ,6 ,6 ,
   6 ,12,6 ,24,12,12,12,6 ,6 ,
   6 ,12,6 ,24
};
Byte RDATA SOUNDTONE2[ ] =
{
   71,80,95,71 ,80,95,71 ,71 ,80 ,71,
   71,80,95,71 ,80,95,106,106,120,106,
   95,95,80,71 ,60,71,80 ,
   95,95,80,120,95,95,95 ,95 ,95 ,
   71,71,80,71
};

void Music2(void)
{
   Byte  i,j,k;
   Byte  SoundLong,SoundTone;
```

```
    Word  m;

    for(i=0; i<40; i++)
    {
       SoundLong=SOUNDLONG2[i];
       SoundTone=SOUNDTONE2[i];

       //sound long
       for(j=0; j<SoundLong; j++)
       {
          //tempo
          for(k=0; k<12; k++)
          {
             //sound tone
             SPEAKER = 1;
             for(m=0; m<SoundTone; m++)
                ;
             SPEAKER = 0;
             for(m=0; m<SoundTone; m++)
                ;
          }
       }
    }
}
```

其汇编程序如下：

```
_Music:
MOV       number?544,R7
MOV       A,number?544
CJNE      A,#01H,?C0017
LCALL     Music1
RET
?C0017:
MOV       A,number?544
CJNE      A,#02H,?C0020
LCALL     Music2
?C0020:
RET

Music1:
CLR       A
MOV       R7,A
?C0021:
MOV       A,R7
```

```
MOV     DPTR,#SOUNDLONG1
MOVC    A,@A+DPTR
MOV     SoundLong?648,A
MOV     A,R7
MOV     DPTR,#SOUNDTONE1
MOVC    A,@A+DPTR
MOV     SoundTone?649,A
CLR     A
MOV     R1,A
?C0024:
MOV     A,R1
CLR     C
SUBB    A,SoundLong?648
JNC     ?C0023
CLR     A
MOV     R6,A
?C0027:
SETB    P3_1
CLR     A
MOV     R5,A
MOV     R4,A
?C0030:
CLR     C
MOV     A,R5
SUBB    A,SoundTone?649
MOV     A,R4
SUBB    A,#00H
JNC     ?C0031
INC     R5
CJNE    R5,#00H,?C0056
INC     R4
?C0056:
SJMP    ?C0030
?C0031:
CLR     P3_1
CLR     A
MOV     R4,A
MOV     R5,A
?C0033:
CLR     C
MOV     A,R5
SUBB    A,SoundTone?649
MOV     A,R4
SUBB    A,#00H
JNC     ?C0029
```

```
 INC      R5
 CJNE     R5,#00H,?C0057
 INC      R4
?C0057:
 SJMP     ?C0033
?C0029:
 INC      R6
 CJNE     R6,#0CH,?C0027
?C0026:
 INC      R1
 SJMP     ?C0024
?C0023:
 INC      R7
 CJNE     R7,#01FH,?C0021
?C0036:
 RET

Music2:
 CLR      A
 MOV      R7,A
?C0037:
 MOV      A,R7
 MOV      DPTR,#SOUNDLONG2
 MOVC     A,@A+DPTR
 MOV      SoundLong?754,A
 MOV      A,R7
 MOV      DPTR,#SOUNDTONE2
 MOVC     A,@A+DPTR
 MOV      SoundTone?755,A
 CLR      A
 MOV      R1,A
?C0040:
 MOV      A,R1
 CLR      C
 SUBB     A,SoundLong?754
 JNC      ?C0039
 CLR      A
 MOV      R6,A
?C0043:
 SETB     P3_1
 CLR      A
 MOV      R5,A
 MOV      R4,A
?C0046:
 CLR      C
```

```
MOV     A,R5
SUBB    A,SoundTone?755
MOV     A,R4
SUBB    A,#00H
JNC     ?C0047
INC     R5
CJNE    R5,#00H,?C0058
INC     R4
?C0058:
SJMP    ?C0046
?C0047:
CLR     P3_1
CLR     A
MOV     R4,A
MOV     R5,A
?C0049:
CLR     C
MOV     A,R5
SUBB    A,SoundTone?755
MOV     A,R4
SUBB    A,#00H
JNC     ?C0045
INC     R5
CJNE    R5,#00H,?C0059
INC     R4
?C0059:
SJMP    ?C0049
?C0045:
INC     R6
CJNE    R6,#0CH,?C0043
?C0042:
INC     R1
SJMP    ?C0040
?C0039:
INC     R7
CJNE    R7,#028H,?C0037
?C0052:
RET
```

2-14 变量选曲 switch...case 方式

当曲目编号很多时，用 if...else 指令结构也会变得较混乱，如果以 switch...case

的条件式作为判断，就会使程序更容易阅读，而且条理分明，如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#define  SPEAKER   P3_1

void PowerOnInit(void);
void InitialCpu(void);
void InitialCpuIO(void);
void DelayX10ms(Word count);
void Music(Byte number);
void Music1(void);
void Music2(void);

/*******************************/
void Main( void )
{
   PowerOnInit( );

   while( 1 )
   {
      Music(2);
      DelayX10ms(100);
      Music(1);
      DelayX10ms(100);
   }
}

/*******************************/
void PowerOnInit(void)
{
   InitialCpu( );

   InitialCpuIO( );
}

void InitialCpu(void)
{
   IE  = 0;              //disable all interrupt
   PSW = 0;              //bank 0

   IP  = 0x0b;           //hi priority:int0,timer0,timer1
```

```
    TMOD= 0x11;          //set timer1,timer0 mode

    TR0 = 0;             //stop timer0
    TR1 = 0;             //stop timer1
    IT0 = 1;             //set int0:falling eage trigger

    EX0 = 0;             //disable int0   interrupt
    ET1 = 0;             //disable timer1 interrupt
    ET0 = 0;             //disable timer0 interrupt

    EA  = 1;             //enable all interrupt gate
}

void InitialCpuIO(void)
{
    SPEAKER = 0;
}

/*****************************/
void DelayX10ms(Word count)
{
    Word i,j,k;

    for(i=0; i<count; i++)
       for(j=0; j<10; j++)
          for(k=0; k<120; k++)
             ;
}

/*****************************/
void Music(Byte number)
{
    switch( number )
    {
     case 1 :
        Music1( );
        break;
     case 2 :
        Music2( );
        break;
     default :
        break;
    }
}
```

```
/*****************************/
Byte RDATA SOUNDLONG1[ ] =
{
    6 ,6 ,9 ,3 ,6 ,6 ,12,
    6 ,6 ,6 ,6 ,6 ,6 ,12,
    6 ,6 ,9 ,3 ,6 ,6 ,9 ,3 ,
    6 ,3 ,3 ,6 ,3 ,3 ,6 ,6 ,9
};
Byte RDATA SOUNDTONE1[ ] =
{
    120,120,106,95 ,80 ,80 ,95,
    80 ,80 ,71 ,63 ,60 ,60 ,80,
    60 ,60 ,71 ,80 ,95 ,71 ,80,95 ,
    120,106,95 ,80 ,71 ,80 ,95,106,120
};

void Music1(void)
{
    Byte  i,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    for(i=0; i<31; i++)
    {
      SoundLong=SOUNDLONG1[i];
      SoundTone=SOUNDTONE1[i];

      //sound long
      for(j=0; j<SoundLong; j++)
      {
        //tempo
        for(k=0; k<12; k++)
        {
          //sound tone
          SPEAKER = 1;
          for(m=0; m<SoundTone; m++)
              ;
          SPEAKER = 0;
          for(m=0; m<SoundTone; m++)
              ;
        }
      }
    }
}
```

```
/*****************************/
Byte RDATA SOUNDLONG2[ ] =
{
   12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
   12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
   12,6 ,6 ,6 ,6 ,6 ,6 ,
   6 ,12,6 ,24,12,12,12,6 ,6 ,
   6 ,12,6 ,24
};
Byte RDATA SOUNDTONE2[ ] =
{
   71,80,95,71 ,80,95,71 ,71 ,80 ,71,
   71,80,95,71 ,80,95,106,106,120,106,
   95,95,80,71 ,60,71,80 ,
   95,95,80,120,95,95,95 ,95 ,95 ,
   71,71,80,71
};

void Music2(void)
{
   Byte  i,j,k;
   Byte  SoundLong,SoundTone;
   Word  m;

   for(i=0; i<40; i++)
   {
      SoundLong=SOUNDLONG2[i];
      SoundTone=SOUNDTONE2[i];

      //sound long
      for(j=0; j<SoundLong; j++)
      {
         //tempo
         for(k=0; k<12; k++)
         {
            //sound tone
            SPEAKER = 1;
            for(m=0; m<SoundTone; m++)
               ;
            SPEAKER = 0;
            for(m=0; m<SoundTone; m++)
               ;
         }
      }
```

```
    }
}
```

其汇编程序如下：

```
_Music:
 MOV       A,R7
 ADD       A,#0FEH
 JZ    ?C0019
 INC       A
 JNZ       ?C0021
?C0018:
 LCALL     Music1
 RET
?C0019:
 LCALL     Music2
?C0021:
 RET

Music1:
 CLR       A
 MOV       R7,A
?C0022:
 MOV       A,R7
 MOV       DPTR,#SOUNDLONG1
 MOVC      A,@A+DPTR
 MOV       SoundLong?648,A
 MOV       A,R7
 MOV       DPTR,#SOUNDTONE1
 MOVC      A,@A+DPTR
 MOV       SoundTone?649,A
 CLR       A
 MOV       R1,A
?C0025:
 MOV       A,R1
 CLR       C
 SUBB      A,SoundLong?648
 JNC       ?C0024
 CLR       A
 MOV       R6,A
?C0028:
 SETB      P3_1
 CLR       A
 MOV       R5,A
 MOV       R4,A
```

```
?C0031:
 CLR      C
 MOV      A,R5
 SUBB     A,SoundTone?649
 MOV      A,R4
 SUBB     A,#00H
 JNC      ?C0032
 INC      R5
 CJNE     R5,#0CH,?C0057
 INC      R4
?C0057:
 SJMP     ?C0031
?C0032:
 CLR      P3_1
 CLR      A
 MOV      R4,A
 MOV      R5,A
?C0034:
 CLR      C
 MOV      A,R5
 SUBB     A,SoundTone?649
 MOV      A,R4
 SUBB     A,#00H
 JNC      ?C0030
 INC      R5
 CJNE     R5,#00H,?C0058
 INC      R4
?C0058:
 SJMP     ?C0034
?C0030:
 INC      R6
 CJNE     R6,#0CH,?C0028
?C0027:
 INC      R1
 SJMP     ?C0025
?C0024:
 INC      R7
 CJNE     R7,#01FH,?C0022
?C0037:
 RET

Music2:
 CLR      A
 MOV      R7,A
?C0038:
```

```
MOV     A,R7
MOV     DPTR,#SOUNDLONG2
MOVC    A,@A+DPTR
MOV     SoundLong?754,A
MOV     A,R7
MOV     DPTR,#SOUNDTONE2
MOVC    A,@A+DPTR
MOV     SoundTone?755,A
CLR     A
MOV     R1,A
?C0041:
MOV     A,R1
CLR     C
SUBB    A,SoundLong?754
JNC     ?C0040
CLR     A
MOV     R6,A
?C0044:
SETB    P3_1
CLR     A
MOV     R5,A
MOV     R4,A
?C0047:
CLR     C
MOV     A,R5
SUBB    A,SoundTone?755
MOV     A,R4
SUBB    A,#00H
JNC     ?C0048
INC     R5
CJNE    R5,#00H,?C0059
INC     R4
?C0059:
SJMP    ?C0047
?C0048:
CLR     P3_1
CLR     A
MOV     R4,A
MOV     R5,A
?C0050:
CLR     C
MOV     A,R5
SUBB    A,SoundTone?755
MOV     A,R4
SUBB    A,#00H
```

```
JNC       ?C0046
INC       R5
CJNE      R5,#00H,?C0060
INC       R4
?C0060:
SJMP      ?C0050
?C0046:
INC       R6
CJNE      R6,#0CH,?C0044
?C0043:
INC       R1
SJMP      ?C0041
?C0040:
INC       R7
CJNE      R7,#028H,?C0038
?C0053:
RET
```

2-15 变量选曲指针法

利用指针变量来取得每一首曲目实际要演奏的起始地址，必须以音长表和音调表作为指针变量的地址，再判断指针变量内容是否为结束符号 0X00，以作为指针变量累加 1 的依据，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#define  SPEAKER   P3_1

void PowerOnInit(void);
void InitialCpu(void);
void InitialCpuIO(void);
void DelayX10ms(Word count);
void Music(Byte number);

/*****************************/
void Main( void )
{
   PowerOnInit( );
```

```
    while( 1 )
    {
      Music(2);
      DelayX10ms(100);
      Music(1);
      DelayX10ms(100);
    }
}

/*****************************/
void PowerOnInit(void)
{
    InitialCpu( );

    InitialCpuIO( );
}

void InitialCpu(void)
{
    IE  = 0;              //disable all interrupt
    PSW = 0;              //bank 0

    IP  = 0x0b;           //hi priority:int0,timer0,timer1

    TMOD= 0x11;           //set timer1,timer0 mode

    TR0 = 0;              //stop timer0
    TR1 = 0;              //stop timer1
    IT0 = 1;              //set int0:falling eage trigger

    EX0 = 0;              //disable int0   interrupt
    ET1 = 0;              //disable timer1 interrupt
    ET0 = 0;              //disable timer0 interrupt

    EA  = 1;              //enable all interrupt gate
}

void InitialCpuIO(void)
{
    SPEAKER = 0;
}

/*****************************/
void DelayX10ms(Word count)
{
```

```
    Word i,j,k;

    for(i=0; i<count; i++)
       for(j=0; j<10; j++)
          for(k=0; k<120; k++)
             ;
}

/*******************************/
Byte RDATA SOUNDLONG[ ] =
{
    6 ,6 ,9 ,3 ,6 ,6 ,12,
    6 ,6 ,6 ,6 ,6 ,6 ,12,
    6 ,6 ,9 ,3 ,6 ,6 ,9 ,3 ,
    6 ,3 ,3 ,6 ,3 ,3 ,6 ,6 ,9,
    0 ,    //end
    12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
    12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
    12,6 ,6 ,6 ,6 ,6 ,6 ,
    6 ,12,6 ,24,12,12,12,6 ,6 ,
    6 ,12,6 ,24,
    0 //end
};
Byte RDATA SOUNDTONE[ ] =
{
    120,120,106,95 ,80 ,80 ,95,
    80 ,80 ,71 ,63 ,60 ,60 ,80,
    60 ,60 ,71 ,80 ,95 ,71 ,80,95 ,
    120,106,95 ,80 ,71 ,80 ,95,106,120,
    0,     //end
    71,80,95,71 ,80,95,71 ,71 ,80 ,71,
    71,80,95,71 ,80,95,106,106,120,106,
    95,95,80,71 ,60,71,80 ,
    95,95,80,120,95,95,95 ,95 ,95 ,
    71,71,80,71 ,
    0 //end
};

void Music(Byte number)
{
    Byte  i,j,k;
    Byte  SoundLong,SoundTone;
    Byte  *DataPointer1,*DataPointer2;
    Word  m;
```

```
    DataPointer1=SOUNDLONG;
    DataPointer2=SOUNDTONE;

    for(i=0; i<(number-1); i++)
    {
       while( *DataPointer1!=0x00 )
       {
          DataPointer1++;
       }
       DataPointer1++;
    }
    for(i=0; i<(number-1); i++)
    {
       while( *DataPointer2!=0x00 )
       {
          DataPointer2++;
       }
       DataPointer2++;
    }

    //sound count
    while( (*DataPointer1!=0x00)||(*DataPointer2!=0x00) )
    {
       SoundLong=*DataPointer1;
       SoundTone=*DataPointer2;
       DataPointer1++;
       DataPointer2++;

       //sound long
       for(j=0; j<SoundLong; j++)
       {
            //tempo
          for(k=0; k<12; k++)
          {
             //sound tone
             SPEAKER = 1;
             for(m=0; m<SoundTone; m++)
                ;
             SPEAKER = 0;
             for(m=0; m<SoundTone; m++)
                ;
          }
       }
    }
}
```

其汇编程序如下：

```
_Music:
MOV       DataPointer1?550,#0FFH
MOV       DataPointer1?550+01H,#HIGH (SOUNDLONG)
MOV       DataPointer1?550+02H,#LOW (SOUNDLONG)
MOV       DataPointer2?551,#0FFH
MOV       DataPointer2?551+01H,#HIGH (SOUNDTONE)
MOV       DataPointer2?551+02H,#LOW (SOUNDTONE)
CLR       A
MOV       R6,A
?C0017:
MOV       A,R7
DEC       A
MOV       R5,A
MOV       A,R6
CLR       C
SUBB      A,R5
JNC       ?C0018
?C0020:
MOV       R3,DataPointer1?550
MOV       R2,DataPointer1?550+01H
MOV       R1,DataPointer1?550+02H
LCALL     ?C?CLDPTR
JZ   ?C0021
MOV       A,#01H
ADD       A,DataPointer1?550+02H
MOV       DataPointer1?550+02H,A
CLR       A
ADDC      A,DataPointer1?550+01H
MOV       DataPointer1?550+01H,A
SJMP      ?C0020
?C0021:
MOV       A,#01H
ADD       A,DataPointer1?550+02H
MOV       DataPointer1?550+02H,A
CLR       A
ADDC      A,DataPointer1?550+01H
MOV       DataPointer1?550+01H,A
INC       R6
SJMP      ?C0017
?C0018:
CLR       A
MOV       R6,A
?C0022:
```

```
MOV     A,R7
DEC     A
MOV     R5,A
MOV     A,R6
CLR     C
SUBB    A,R5
JNC     ?C0027
?C0025:
MOV     R3,DataPointer2?551
MOV     R2,DataPointer2?551+01H
MOV     R1,DataPointer2?551+02H
LCALL   ?C?CLDPTR
JZ  ?C0026
MOV     A,#01H
ADD     A,DataPointer2?551+02H
MOV     DataPointer2?551+02H,A
CLR     A
ADDC    A,DataPointer2?551+01H
MOV     DataPointer2?551+01H,A
SJMP    ?C0025
?C0026:
MOV     A,#01H
ADD     A,DataPointer2?551+02H
MOV     DataPointer2?551+02H,A
CLR     A
ADDC    A,DataPointer2?551+01H
MOV     DataPointer2?551+01H,A
INC     R6
SJMP    ?C0022
?C0027:
MOV     R3,DataPointer1?550
MOV     R2,DataPointer1?550+01H
MOV     R1,DataPointer1?550+02H
LCALL   ?C?CLDPTR
MOV     R7,A
JNZ     ?C0029
MOV     R3,DataPointer2?551
MOV     R2,DataPointer2?551+01H
MOV     R1,DataPointer2?551+02H
LCALL   ?C?CLDPTR
JZ  ?C0042
?C0029:
MOV     SoundLong?548,R7
MOV     R3,DataPointer2?551
MOV     R2,DataPointer2?551+01H
```

```
MOV     R1,DataPointer2?551+02H
LCALL   ?C?CLDPTR
MOV     SoundTone?549,A
MOV     A,#01H
ADD     A,DataPointer1?550+02H
MOV     DataPointer1?550+02H,A
CLR     A
ADDC    A,DataPointer1?550+01H
MOV     DataPointer1?550+01H,A
MOV     A,#01H
ADD     A,DataPointer2?551+02H
MOV     DataPointer2?551+02H,A
CLR     A
ADDC    A,DataPointer2?551+01H
MOV     DataPointer2?551+01H,A
CLR     A
MOV     j?546,A
?C0030:
MOV     A,j?546
CLR     C
SUBB    A,SoundLong?548
JNC     ?C0027
CLR     A
MOV     R7,A
?C0033:
SETB    P3_1
CLR     A
MOV     R5,A
MOV     R4,A
?C0036:
MOV     A,SoundTone?549
MOV     R3,A
CLR     C
MOV     A,R5
SUBB    A,R3
MOV     A,R4
SUBB    A,#00H
JNC     ?C0037
INC     R5
CJNE    R5,#00H,?C0046
INC     R4
?C0046:
SJMP    ?C0036
?C0037:
CLR     P3_1
```

```
CLR     A
MOV     R4,A
MOV     R5,A
?C0039:
MOV     A,SoundTone?549
MOV     R3,A
CLR     C
MOV     A,R5
SUBB    A,R3
MOV     A,R4
SUBB    A,#00H
JNC     ?C0035
INC     R5
CJNE    R5,#00H,?C0047
INC     R4
?C0047:
SJMP    ?C0039
?C0035:
INC     R7
CJNE    R7,#0CH,?C0033
?C0032:
INC     j?546
SJMP    ?C0030
?C0042:
RET
```

第 3 章　模块化程序音乐演奏

目的：利用模块化程序设计理念来构建软件，并以各种指令结构进行音乐演奏，以阐述软件的思维和解决方法。

模块化：第 2 章将全部的函数，以一个 main.c 模块来完成，通过 Main( )函数来调用初始化动作，延迟时间动作以及演奏音乐的函数而构成演奏音乐的目的。如果是小型化的程序，其功能需求简单，将这样的程序编写在单一的模块之内，还不至于构成混乱而不易维护。但如果发展大型程序，则此种方式将不敷使用，建议读者不管程序大型或小型程序，养成模块化程序设计概念将有助于日后软件设计能力的提高，本章节即是说明如何将第 2 章的全部程序以模块化的方式来进行设计。

首先，可规划的模块如下：

1. global.c：全局变量的模块，演奏音乐的程序无需使用到全局变量，但仍然进行规划，以便日后使用。

2. main.c：程序进入点模块，但第一个动作便是作初始化的动作。

3. initial.c：初始化动作的模块，包含初始化 CPU 缓存器和初始化 CPU 的 I/O。

4. delay.c：延迟时间的模块，可作为任何时间的延迟。

5. music.c：演奏音乐的模块，每一音符的音长和音调在此模块建表，以便产生方波时间和频率。

以及对应的包括文件：

1. global.h：全局变量的外部声明，以便其他模块使用。

2. initial.h：initial.c 模块的包括文件，即将 initial.c 模块下的所有函数声明成外部，以便其他模块调用。

3. delay.h：delay.c 模块的包括文件，也是将 delay.c 模块下的所有函数声明成外部，以便其他模块调用。

4. music.h：music.c 模块的包括文件，除了将 music.c 模块下的所有函数声明成外部，以便其他模块调用外，应将喇叭的输出 I/O 定义为 P3.1，如果硬件电路设计为 P1.1 作为驱动喇叭的 I/O，则须将此 SPEAKER 定义为 P1.1。

3-1　if 指令

当发展一套软件时，最好分解成若干模块，而模块是任一具独立功能的单位，可分为主程序、子程序、函数等，而当一个模块里的程序太长时，也可再细分为 2

个模块。模块与模块之间所有的联系，可通过自变量来传递，一个模块通常在程序前端会将每个模块对应的包括文件利用#include 指令包括进来，以及包括宏指令定义文件、CPU 缓存器定义文件等，以使本模块可以调用其他模块的函数、存取全局变量、宏指令使用以及常数定义、I/O 符号定义等，下面分别介绍。

1. main.c：主程序模块，并调用音乐演奏函数，以 while(1)反复循环，当每演奏完一首曲目并延迟一秒钟后继续演奏，在程序前端分别包括了 define.h 文件、mtv212m.h 文件、global.h 文件、initial.h 文件、delay.h 文件以及 music.h 文件，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"


/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
     Music( );
     DelayX10ms(100);
   }
}
```

2. intial.c：初始化的模块，当 CPU 送上电源后进行 CPU 本身及外围电路的设定、清除、脉冲信号以及缓存器的动作，与程序前端一样，包括 define.h 文件、mtv212.h 文件、initial.h 文件、delay.h 文件以及 music.h 文件，以便本模块的应用。模块中包含了 CPU 缓存器初始化（即 Initialcpu( )函数）和 CPU I/O 的初始化（InitialCpuIO( )函数），其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
```

```
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
void PowerOnInit(void)
{
   InitialCpu();

   InitialCpuIO();
}

/*****************************/
void InitialCpu( void )
{
   IE  = 0;              //disable all interrupt
   PSW = 0;              //bank 0

   IP  = 0x0b;           //hi priority:int0,timer0,timer1

   TMOD= 0x11;           //set timer1,timer0 mode

   TR0 = 0;              //stop timer0
   TR1 = 0;              //stop timer1
   IT0 = 1;              //set int0:falling eage trigger

   EX0 = 0;              //disable int0 interrupt
   ET1 = 0;              //disable timer1 interrupt
   ET0 = 0;              //disable timer0 interrupt

   EA  = 1;              //enable all interrupt gate
}

/*****************************/
void InitialCpuIO(void)
{
   SPEAKER = 0;
}
```

3. delay.c：延迟时间的模块，利用 for 循环使得 CPU 一直等待，当等待时间结束后才返回原调用程序。在其他模块调用延迟函数时，须注意 CPU 将只等待，而无法检测到外界的变化，如果是中断发生，才能将 CPU 转移至中断函数去处理，待中

断函数处理完毕又会回到延迟程序继续等待。要留意此时的 R0～R7 缓存器是否已改变而影响延迟函数的时间，其中程序前端的包括文件意义如上所示，本 delay.c 模块的程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
void DelayX10ms(Word count)
{
   Word i,j,k;

   for(i=0; i<count; i++)
      for(j=0; j<10; j++)
         for(k=0; k<120; k++)
            ;
}
```

4. music.c：音乐演奏的模块，利用 if 指令来顺序演奏每一个音符，当音符到达最后一个时就返回原调用程序。其中，将音符的音长和音调数值依序从表中取出，通过 for 循环产生音阶频率和 for 循环来持续音阶的时间，而程序前端的包括文件意义如前面的说明，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
   6,6,9,3,6,6,12,
```

```
    6,6,6,6,6,6,12,
    6,6,9,3,6,6,9 ,3,
    6,3,3,6,3,3,6 ,6,9
};
Byte RDATA SOUNDTONE[ ] =
{
    120,120,106,95,80,80,95,
    80 ,80 ,71 ,63,60,60,80,
    60 ,60 ,71 ,80,95,71,80,95 ,
    120,106,95 ,80,71,80,95,106,120
};

void Music(void)
{
    Byte  i=0,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    while( 1 )
    {
      //sound count
      if (i==31) return;
      SoundLong=SOUNDLONG[i];
      SoundTone=SOUNDTONE[i];
      i++;

      //sound long
      for(j=0; j<SoundLong; j++)
      {
        //tempo
        for(k=0; k<12; k++)
        {
          //sound tone
          SPEAKER = 1;
          for(m=0; m<SoundTone; m++)
            ;
          SPEAKER = 0;
          for(m=0; m<SoundTone; m++)
            ;
        }
      }
    }
}
```

而包括文件的内容如下：

（1）initial.h，如下：

```
#ifndef   _INITIAL_H
#define   _INITIAL_H

extern void PowerOnInit(void);
extern void InitialCpu(void);
extern void InitialCpuIO(void);

#endif
```

（2）delay.h，如下：

```
#ifndef   _DELAY_H
#define   _DELAY_H

extern void DelayX10ms(Word count);

#endif
```

（3）music.h

其中喇叭输出的I/O定义为P3.1，如下：

```
#ifndef   _MUSIC_H
#define   _MUSIC_H

#define  SPEAKER   P3_1

extern Byte RDATA SOUNDLONG[ ];
extern Byte RDATA SOUNDTONE[ ];
extern void Music(void);

#endif
```

3-2　for循环

音乐演奏函数以for指令来完成，而演奏曲目为“三轮车”共有31个音符，所以，for循环中的条件式i变量必须小于31，要计算出曲目的音符个数很麻烦，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
```

```
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
    6,6,9,3,6,6,12,
    6,6,6,6,6,6,12,
    6,6,9,3,6,6,9 ,3,
    6,3,3,6,3,3,6 ,6,9
};
Byte RDATA SOUNDTONE[ ] =
{
    120,120,106,95,80,80,95,
    80 ,80 ,71 ,63,60,60,80,
    60 ,60 ,71 ,80,95,71,80,95 ,
    120,106,95 ,80,71,80,95,106,120
};

void Music(void)
{
    Byte  i,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    //sound count
    for(i=0; i<31; i++)
    {
       SoundLong=SOUNDLONG[i];
       SoundTone=SOUNDTONE[i];

       //sound long
       for(j=0; j<SoundLong; j++)
       {
            //tempo
         for(k=0; k<12; k++)
         {
            //sound tone
            SPEAKER = 1;
            for(m=0; m<SoundTone; m++)
               ;
            SPEAKER = 0;
            for(m=0; m<SoundTone; m++)
```

```
                ;
            }
        }
    }
}
```

3-3　do...while 循环

先执行后判断的指令结构，每当演奏完一个音符，即将 i 变量加 1，并取得下一音符的音长和音调数值，在音符演奏完毕才进行是否已经是最后一个音符的判断，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
    6,6,9,3,6,6,12,
    6,6,6,6,6,6,12,
    6,6,9,3,6,6,9 ,3,
    6,3,3,6,3,3,6 ,6,9,
    0      //end
};
Byte RDATA SOUNDTONE[ ] =
{
    120,120,106,95,80,80,95,
    80 ,80 ,71 ,63,60,60,80,
    60 ,60 ,71 ,80,95,71,80,95 ,
    120,106,95 ,80,71,80,95,106,120,
    0 //end
};

void Music(void)
{
```

```
    Byte  i=0,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    do
    {
       SoundLong=SOUNDLONG[i];
       SoundTone=SOUNDTONE[i];
       i++;

       //sound long
       for(j=0; j<SoundLong; j++)
       {
            //tempo
          for(k=0; k<12; k++)
          {
             //sound tone
             for(m=0; m<SoundTone; m++)
                SPEAKER = 1;
             for(m=0; m<SoundTone; m++)
                SPEAKER = 0;
          }
       }

    } while ( (SOUNDLONG[i]!=0x00)||(SOUNDTONE[i]!=0x00) );
}
```

3-4 while 循环

先判断后执行的指令结构，先判断目前数值是否为最后的结束符号，是，则表示所有音符已经演奏完毕，否，则表示此音符要继续演奏。同样，在演奏完一个音符须将下一个音符的地址通过 i 变量加 1 而取得音长和音调的数值，此种结构比 do...while 的结构更清晰明了，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
```

```
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
    6,6,9,3,6,6,12,
    6,6,6,6,6,6,12,
    6,6,9,3,6,6,9 ,3,
    6,3,3,6,3,3,6 ,6,9,
    0 //end
};
Byte RDATA SOUNDTONE[ ] =
{
    120,120,106,95,80,80,95,
    80 ,80 ,71 ,63,60,60,80,
    60 ,60 ,71 ,80,95,71,80,95 ,
    120,106,95 ,80,71,80,95,106,120,
    0 //end
};

void Music(void)
{
    Byte  i=0,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    while ( (SOUNDLONG[i]!=0x00)||(SOUNDTONE[i]!=0x00) )
    {
       SoundLong=SOUNDLONG[i];
       SoundTone=SOUNDTONE[i];
       i++;

       //sound long
       for(j=0; j<SoundLong; j++)
       {
            //tempo
         for(k=0; k<12; k++)
         {
            //sound tone
            SPEAKER = 1;
            for(m=0; m<SoundTone; m++)
               ;
            SPEAKER = 0;
            for(m=0; m<SoundTone; m++)
               ;
```

```
        }
      }
    }
}
```

3-5 do...while 结束符号 0X00

当以 do...while 的指令结构作为音乐演奏的函数，而将结束符号 0X00 写在音长表和音调表的第一个字节，表示不演奏之意，则将产生错误，为什么呢？因为 do...while 指令是先执行后判断的指令，当 i=0，第一个字节即结束符号 0X00 写入至音长和音调变量内，此时 i 加 1 后变成 i=1，当音符演奏完毕再判断音长表和音调表是否为结束符号 0X00，因为 i=1，当然已经不是结束符号，而会一直演奏下去，与“不演奏”相违背，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
   0,     //end
   6,6,9,3,6,6,12,
   6,6,6,6,6,6,12,
   6,6,9,3,6,6,9 ,3,
   6,3,3,6,3,3,6 ,6,9,
   0 //end
};
Byte RDATA SOUNDTONE[ ] =
{
   0,     //end
   120,120,106,95,80,80,95,
   80 ,80 ,71 ,63,60,60,80,
   60 ,60 ,71 ,80,95,71,80,95 ,
   120,106,95 ,80,71,80,95,106,120,
   0 //end
```

```
};

void Music(void)
{
    Byte  i=0,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    do
    {
       SoundLong=SOUNDLONG[i];
       SoundTone=SOUNDTONE[i];
       i++;

       //sound long
       for(j=0; j<SoundLong; j++)
       {
            //tempo
         for(k=0; k<12; k++)
         {
            //sound tone
            for(m=0; m<SoundTone; m++)
              SPEAKER = 1;
            for(m=0; m<SoundTone; m++)
              SPEAKER = 0;
         }
       }

    } while ( (SOUNDLONG[i]!=0x00)||(SOUNDTONE[i]!=0x00) );
}
```

而用 while 指令就不会产生错误，所以说，以 while 指令结构优于以 do...while 的指令结构，原因在此，以 while 指令其程序写法如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*******************************/
```

```
Byte RDATA SOUNDLONG[ ] =
{
    0,     //end
    6,6,9,3,6,6,12,
    6,6,6,6,6,6,12,
    6,6,9,3,6,6,9 ,3,
    6,3,3,6,3,3,6 ,6,9,
    0 //end
};
Byte RDATA SOUNDTONE[ ] =
{
    0,     //end
    120,120,106,95,80,80,95,
    80 ,80 ,71 ,63,60,60,80,
    60 ,60 ,71 ,80,95,71,80,95 ,
    120,106,95 ,80,71,80,95,106,120,
    0 //end
};

/*****************************/
void Music1(void)
{
    Byte  i=0,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    while ( (SOUNDLONG[i]!=0x00)||(SOUNDTONE[i]!=0x00) )
    {
       SoundLong=SOUNDLONG[i];
       SoundTone=SOUNDTONE[i];
       i++;

       //sound long
       for(j=0; j<SoundLong; j++)
       {
            //tempo
         for(k=0; k<12; k++)
         {
            //sound tone
            SPEAKER = 1;
            for(m=0; m<SoundTone; m++)
               ;
            SPEAKER = 0;
            for(m=0; m<SoundTone; m++)
               ;
```

```
        }
      }
    }
}
```

3-6　指针法 for 方式

演奏音乐函数通过指针变量*datapointer1 和*datapointer2 来取得音长表和音调表的数值，而以 for 循环来演奏一个音符的方法，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
    6,6,9,3,6,6,12,
    6,6,6,6,6,6,12,
    6,6,9,3,6,6,9 ,3,
    6,3,3,6,3,3,6 ,6,9
};
Byte RDATA SOUNDTONE[ ] =
{
    120,120,106,95,80,80,95,
    80 ,80 ,71 ,63,60,60,80,
    60 ,60 ,71 ,80,95,71,80,95 ,
    120,106,95 ,80,71,80,95,106,120
};

void Music(void)
{
    Byte  i,j,k;
    Byte  SoundLcng,SoundTone;
    Byte  *DataPcinter1,*DataPointer2;
    Word  m;
```

```
    DataPointer1=SOUNDLONG;
    DataPointer2=SOUNDTONE;

    //sound count
    for(i=0; i<31; i++)
    {
       SoundLong=*(DataPointer1+i);
       SoundTone=*(DataPointer2+i);

       //sound long
       for(j=0; j<SoundLong; j++)
       {
            //tempo
          for(k=0; k<12; k++)
          {
             //sound tone
             SPEAKER = 1;
             for(m=0; m<SoundTone; m++)
                ;
             SPEAKER = 0;
             for(m=0; m<SoundTone; m++)
                ;
          }
       }
    }
}
```

3-7 指针法 while 方式

演奏音乐函数除了利用指针方式，并加上结束符号 0X00 可避免每首曲目都要计算此音符个数的麻烦，也可以以 while 指令结构来取代 do...while 的指令结构，为何要这样，还记得原因吗？程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
```

```
/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
    6,6,9,3,6,6,12,
    6,6,6,6,6,6,12,
    6,6,9,3,6,6,9 ,3,
    6,3,3,6,3,3,6 ,6,9,
    0 //end
};
Byte RDATA SOUNDTONE[ ] =
{
    120,120,106,95,80,80,95,
    80 ,80 ,71 ,63,60,60,80,
    60 ,60 ,71 ,80,95,71,80,95 ,
    120,106,95 ,80,71,80,95,106,120,
    0 //end
};

void Music(void)
{
    Byte  j,k;
    Byte  SoundLong,SoundTone;
    Byte  *DataPointer1,*DataPointer2;
    Word  m;

    DataPointer1=SOUNDLONG;
    DataPointer2=SOUNDTONE;

    //sound count
    while( (*DataPointer1!=0x00)||(*DataPointer2!=0x00) )
    {
      SoundLong=*DataPointer1;
      SoundTone=*DataPointer2;
      DataPointer1++;
      DataPointer2++;

      //sound long
      for(j=0; j<SoundLong; j++)
      {
           //tempo
        for(k=0; k<12; k++)
        {
          //sound tone
          SPEAKER = 1;
```

```
                for(m=0; m<SoundTone; m++)
                  ;
                SPEAKER = 0;
                for(m=0; m<SoundTone; m++)
                  ;
            }
          }
        }
    }
```

3-8 2 个字节的音调表

以 2 个字节来表示音调，虽然可以演奏出较低频率的音符，但音调表的内容编写过于复杂与冗长，而且程序也必须将高、低字节运算后才写入至音调变量，所以，并不是一个很好的程序设计。较完整实用的程序应该将音调表声明为 int 类型，如后面的介绍，但仍列出其程序，如下所示：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*******************************/
Byte RDATA SOUNDLONG[ ] =
{
   6,6,9,3,6,6,12,
   6,6,6,6,6,6,12,
   6,6,9,3,6,6,9 ,3,
   6,3,3,6,3,3,6 ,6,9
};
Byte RDATA SOUNDTONE[ ] =
{
   0,120,0,120,0,106,0,95,0,80,0,80,0,95,
   0,80 ,0,80 ,0,71 ,0,63,0,60,0,60,0,80,
   0,60 ,0,60 ,0,71 ,0,80,0,95,0,71,0,80,0,95 ,
   0,120,0,106,0,95 ,0,80,0,71,0,80,0,95,0,106,0,120
};
```

```
void Music(void)
{
    Byte  i,j,k,SoundLong;
    Word  m,SoundTone;

    //sound count
    for(i=0; i<31; i++)
    {
       SoundLong = SOUNDLONG[i];
       SoundTone = SOUNDTONE[i*2] << 8 ;
       SoundTone += SOUNDTONE[i*2+1];

       //sound long
       for(j=0; j<SoundLong; j--)
       {
            //tempo
          for(k=0; k<12; k++)
          {
             //sound tone
             SPEAKER = 1;
             for(m=0; m<SoundTone; m++)
                ;
             SPEAKER = 0;
             for(m=0; m<SoundTone; m++)
                ;
          }
       }
    }
}
```

3-9　int 类型的音调表

将音调表声明成 int 类型，即 2 个字节的类型，不但可以演奏出频率很低的音符，而且程序结构清晰，容易阅读。程序也无需将高、低字节运算后才能写入至音调变量内，由下标 i 变量每增加一个单位，则音调表将自动略过 2 个字节而取得下一音符的音调，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
```

```
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
   6,6,9,3,6,6,12,
   6,6,6,6,6,6,12,
   6,6,9,3,6,6,9 ,3,
   6,3,3,6,3,3,6 ,6,9
};
Word RDATA SOUNDTONE[ ] =
{
   120,120,106,95,80,80,95,
   80 ,80 ,71 ,63,60,60,80,
   60 ,60 ,71 ,80,95,71,80,95 ,
   120,106,95 ,80,71,80,95,106,120
};

void Music(void)
{
   Byte  i,j,k,SoundLong;
   Word  m,SoundTone;

   //sound count
   for(i=0; i<31; i++)
   {
      SoundLong = SOUNDLONG[i];
      SoundTone = SOUNDTONE[i];
      //sound long
      for(j=0; j<SoundLong; j++)
      {
           //tempo
        for(k=0; k<12; k++)
        {
          //sound tone
          SPEAKER = 1;
          for(m=0; m<SoundTone; m++)
            ;
          SPEAKER = 0;
          for(m=0; m<SoundTone; m++)
            ;
```

```
        }
      }
    }
}
```

3-10　变量选曲 switch 和 for 方式

以自变量作为曲目编号，可以大大缩短每一曲目程序编写的麻烦，但配合 for 循环仍必须计算出每一曲目的音符个数，较为麻烦，为说明软件构建的思维，仍列出程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
   6 ,6 ,9 ,3 ,6 ,6 ,12,
   6 ,6 ,6 ,6 ,6 ,6 ,12,
   6 ,6 ,9 ,3 ,6 ,6 ,9 ,3 ,
   6 ,3 ,3 ,6 ,3 ,3 ,6 ,6 ,9 ,

   12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
   12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
   12,6 ,6 ,6 ,6 ,6 ,6 ,
   6 ,12,6 ,24,12,12,12,6 ,6 ,
   6 ,12,6 ,24
};
Byte RDATA SOUNDTONE[ ] =
{
   120,120,106,95 ,80 ,80 ,95,
   80 ,80 ,71 ,63 ,60 ,60 ,80,
   60 ,60 ,71 ,80 ,95 ,71 ,80,95 ,
   120,106,95 ,80 ,71 ,80 ,95,106,120,
```

```
    71,80,95,71 ,80,95,71 ,71 ,80 ,71,
    71,80,95,71 ,80,95,106,106,120,106,
    95,95,80,71 ,60,71,80 ,
    95,95,80,120,95,95,95 ,95 ,95 ,
    71,71,80,71
};

void Music(Byte number)
{
    Byte  i,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    MusicNum=number;

    switch( MusicNum )
    {
     case 1 :
       for(i=0; i<31; i++)
       {
          SoundLong=SOUNDLONG[i];
          SoundTone=SOUNDTONE[i];

          //sound long
          for(j=0; j<SoundLong; j++)
          {
             //tempo
             for(k=0; k<12; k++)
             {
                //sound tone
                SPEAKER = 1;
                for(m=0; m<SoundTone; m++)
                   ;
                SPEAKER = 0;
                for(m=0; m<SoundTone; m++)
                   ;
             }
          }
       }
       break;
     case 2 :
       for(i=31; i<71; i++)
       {
          SoundLong=SOUNDLONG[i];
          SoundTone=SOUNDTONE[i];
```

```
            //sound long
            for(j=0; j<SoundLong; j++)
            {
               //tempo
               for(k=0; k<12; k++)
               {
                  //sound tone
                  SPEAKER = 1;
                  for(m=0; m<SoundTone; m++)
                     ;
                  SPEAKER = 0;
                  for(m=0; m<SoundTone; m++)
                     ;
               }
            }
         }
         break;
      default :
         break;
      }
}
```

3-11　变量选曲 while 方式

如果有 2 首曲目，则须将每首曲目的音长和音调连续地建表，在音符结束时并补上 0X00 的结束符号。因此，在程序中以曲目编号，即自变量，来计算出其音长表和音调表的起始地址，只要判断结束符号是否到了，否，则将地址一直加 1 即可，当起始地址已经计算出，则开始演奏以此起始地址而取得音长和音调数值，每演奏完一个音符，则地址就加 1，直到曲目的结束符号出现为止，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
```

```
Byte RDATA SOUNDLONG[ ] =
{
   6 ,6 ,9 ,3 ,6 ,6 ,12,
   6 ,6 ,6 ,6 ,6 ,6 ,12,
   6 ,6 ,9 ,3 ,6 ,6 ,9 ,3 ,
   6 ,3 ,3 ,6 ,3 ,3 ,6 ,6 ,9,
   0 ,    //end
   12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
   12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
   12,6 ,6 ,6 ,6 ,6 ,6 ,
   6 ,12,6 ,24,12,12,12,6 ,6 ,
   6 ,12,6 ,24,
   0 //end
};
Byte RDATA SOUNDTONE[ ] =
{
   120,120,106,95 ,80 ,80 ,95,
   80 ,80 ,71 ,63 ,60 ,60 ,80,
   60 ,60 ,71 ,80 ,95 ,71 ,80,95 ,
   120,106,95 ,80 ,71 ,80 ,95,106,120,
   0,     //end
   71,80,95,71 ,80,95,71 ,71 ,80 ,71,
   71,80,95,71 ,80,95,106,106,120,106,
   95,95,80,71 ,60,71,80 ,
   95,95,80,120,95,95,95 ,95 ,95 ,
   71,71,80,71 ,
   0 //end
};

void Music(Byte number)
{
   Byte  k,n;
   Byte  SoundLong,SoundTone;
   Word  i=0,j=0,m;

   for(k=0; k<(number-1); k++)
   {
     while( SOUNDLONG[i] != 0x00 )
     {
       i++;
     }
     i++;
   }
   for(k=0; k<(number-1); k++)
   {
```

```
        while( SOUNDTONE[j] != 0x00 )
        {
          j++;
        }
        j++;
    }

    do
    {
        SoundLong=SOUNDLONG[i];
        SoundTone=SOUNDTONE[j];
        i++;
        j++;

        //sound long
        for(n=0; n<SoundLong; n++)
        {
              //tempo
          for(k=0; k<12; k++)
          {
            SPEAKER = 1;
            for(m=0; m<SoundTone; m++)
              ;
            SPEAKER = 0;
            for(m=0; m<SoundTone; m++)
              ;
          }
        }

    } while ( (SOUNDLONG[i]!=0x00)||(SOUNDTONE[j]!=0x00) );
}
```

3-12　变量选曲 if...else 方式

利用 Music( )函数以 if...else 来个别判断自变量而调用真正的演奏音乐函数，其结构简单明了、容易阅读，适合于曲目编号不是很多的时候，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
```

```
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
void Music(Byte number)
{
   if ( number==1 )
     Music1( );
   else if ( number==2 )
     Music2( );
}

/*****************************/
Byte RDATA SOUNDLONG1[ ] =
{
   6 ,6 ,9 ,3 ,6 ,6 ,12,
   6 ,6 ,6 ,6 ,6 ,6 ,12,
   6 ,6 ,9 ,3 ,6 ,6 ,9 ,3 ,
   6 ,3 ,3 ,6 ,3 ,3 ,6 ,6 ,9
};
Byte RDATA SOUNDTONE1[ ] =
{
   120,120,106,95 ,80 ,80 ,95,
   80 ,80 ,71 ,63 ,60 ,60 ,80,
   60 ,60 ,71 ,80 ,95 ,71 ,80,95 ,
   120,106,95 ,80 ,71 ,80 ,95,106,120
};

void Music1(void)
{
   Byte  i,j,k;
   Byte  SoundLong,SoundTone;
   Word  m;

   for(i=0; i<31; i++)
   {
     SoundLong=SOUNDLONG1[i];
     SoundTone=SOUNDTONE1[i];

     //sound long
     for(j=0; j<SoundLong; j++)
     {
       //tempo
       for(k=0; k<12; k++)
```

```
        {
          //sound tone
          SPEAKER = 1;
          for(m=0; m<SoundTone; m++)
              ;
          SPEAKER = 0;
          for(m=0; m<SoundTone; m++)
              ;
        }
      }
    }
}

/*****************************/
Byte RDATA SOUNDLONG2[ ] =
{
    12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
    12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
    12,6 ,6 ,6 ,6 ,6 ,6 ,
    6 ,12,6 ,24,12,12,12,6 ,6 ,
    6 ,12,6 ,24
};
Byte RDATA SOUNDTONE2[ ] =
{
    71,80,95,71 ,80,95,71 ,71 ,80 ,71,
    71,80,95,71 ,80,95,106,106,120,106,
    95,95,80,71 ,60,71,80 ,
    95,95,80,120,95,95,95 ,95 ,95 ,
    71,71,80,71
};

void Music2(void)
{
    Byte  i,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    for(i=0; i<40; i++)
    {
      SoundLong=SOUNDLONG2[i];
      SoundTone=SOUNDTONE2[i];

      //sound long
      for(j=0; j<SoundLong; j++)
      {
```

```
            //tempo
            for(k=0; k<12; k++)
            {
               //sound tone
               SPEAKER = 1;
               for(m=0; m<SoundTone; m++)
                  ;
               SPEAKER = 0;
               for(m=0; m<SoundTone; m++)
                  ;
            }
         }
      }
   }
```

3-13 变量选曲 switch...case 方式

当曲目编号很多时，以 switch...case 的指令结构反而比 if...else 的指结构来得更条理分明，更容易阅读，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
void Music(Byte number)
{
   switch( number )
   {
    case 1 :
       Music1( );
       break;
    case 2 :
       Music2( );
       break;
    default :
       break;
```

```
    }
}

/*****************************/
Byte RDATA SOUNDLONG1[ ] =
{
    6 ,6 ,9 ,3 ,6 ,6 ,12,
    6 ,6 ,6 ,6 ,6 ,6 ,12,
    6 ,6 ,9 ,3 ,6 ,6 ,9 ,3 ,
    6 ,3 ,3 ,6 ,3 ,3 ,6 ,6 ,9
};
Byte RDATA SOUNDTONE1[ ] =
{
    120,120,106,95 ,80 ,80 ,95,
    80 ,80 ,71 ,63 ,60 ,60 ,80,
    60 ,60 ,71 ,80 ,95 ,71 ,80,95 ,
    120,106,95 ,80 ,71 ,80 ,95,106,120
};

void Music1(void)
{
    Byte  i,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    for(i=0; i<31; i++)
    {
      SoundLong=SOUNDLONG1[i];
      SoundTone=SOUNDTONE1[i];

      //sound long
      for(j=0; j<SoundLong; j++)
      {
        //tempo
        for(k=0; k<12; k++)
        {
          //sound tone
          SPEAKER = 1;
          for(m=0; m<SoundTone; m++)
              ;
          SPEAKER = 0;
          for(m=0; m<SoundTone; m++)
              ;
        }
      }
```

```
    }
}

/*****************************/
Byte RDATA SOUNDLONG2[ ] =
{
    12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
    12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
    12,6 ,6 ,6 ,6 ,6 ,6 ,
    6 ,12,6 ,24,12,12,12,6 ,6 ,
    6 ,12,6 ,24
};
Byte RDATA SOUNDTONE2[ ] =
{
    71,80,95,71 ,80,95,71 ,71 ,80 ,71,
    71,80,95,71 ,80,95,106,106,120,106,
    95,95,80,71 ,60,71,80 ,
    95,95,80,120,95,95,95 ,95 ,95 ,
    71,71,80,71
};

void Music2(void)
{
    Byte  i,j,k;
    Byte  SoundLong,SoundTone;
    Word  m;

    for(i=0; i<40; i++)
    {
       SoundLong=SOUNDLONG2[i];
       SoundTone=SOUNDTONE2[i];

       //sound long
       for(j=0; j<SoundLong; j++)
       {
          //tempo
          for(k=0; k<12; k++)
          {
             //sound tone
             SPEAKER = 1;
             for(m=0; m<SoundTone; m++)
                ;
             SPEAKER = 0;
             for(m=0; m<SoundTone; m++)
                ;
```

```
        }
      }
    }
}
```

3-14　变量选曲指针法

首先将音长表和音调表的起始地址分别存入至 datapointer1 和 datapointer2 指针变量内，并根据自变量来分别跳过每一曲目所有的音长和音调数值，并以此起始地址作为演奏音符的取表地址而依次演奏每一音符，当指针变量的内容等于结束符号 0X00，即表示演奏完毕，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
    6 ,6 ,9 ,3 ,6 ,6 ,12,
    6 ,6 ,6 ,6 ,6 ,6 ,12,
    6 ,6 ,9 ,3 ,6 ,6 ,9 ,3 ,
    6 ,3 ,3 ,6 ,3 ,3 ,6 ,6 ,9,
    0 ,    //end
    12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
    12,6 ,6 ,12,6 ,6 ,6 ,12,6 ,24,
    12,6 ,6 ,6 ,6 ,6 ,6 ,
    6 ,12,6 ,24,12,12,12,6 ,6 ,
    6 ,12,6 ,24,
    0 //end
};
Byte RDATA SOUNDTONE[ ] =
{
    120,120,106,95 ,80 ,80 ,95,
    80 ,80 ,71 ,63 ,60 ,60 ,80,
    60 ,60 ,71 ,80 ,95 ,71 ,80,95 ,
```

```
    120,106,95 ,80 ,71 ,80 ,95,106,120,
    0,     //end
    71,80,95,71 ,80,95,71 ,71 ,80 ,71,
    71,80,95,71 ,80,95,106,106,120,106,
    95,95,80,71 ,60,71,80 ,
    95,95,80,120,95,95,95 ,95 ,95 ,
    71,71,80,71 ,
    0 //end
};

void Music(Byte number)
{
    Byte  i,j,k;
    Byte  SoundLong,SoundTone;
    Byte  *DataPointer1,*DataPointer2;
    Word  m;

    DataPointer1=SOUNDLONG;
    DataPointer2=SOUNDTONE;

    for(i=0; i<(number-1); i++)
    {
       while( *DataPointer1!=0x00 )
       {
          DataPointer1++;
       }
       DataPointer1++;
    }
    for(i=0; i<(number-1); i++)
    {
       while( *DataPointer2!=0x00 )
       {
          DataPointer2++;
       }
       DataPointer2++;
    }

    //sound count
    while( (*DataPointer1!=0x00)||(*DataPointer2!=0x00) )
    {
       SoundLong=*DataPointer1;
       SoundTone=*DataPointer2;
       DataPointer1++;
       DataPointer2++;
```

```
        //sound long
        for(j=0; j<SoundLong; j++)
        {
              //tempo
           for(k=0; k<12; k++)
           {
              //sound tone
              SPEAKER = 1;
              for(m=0; m<SoundTone; m++)
                 ;
              SPEAKER = 0;
              for(m=0; m<SoundTone; m++)
                 ;
           }
        }
    }
}
```

第 4 章　音调定时器

4-1　计算法

目的：利用 Timer1 来产生每一音阶“0”和“1”电位的时间，并以每一音阶的频率计算出时间，以驱动喇叭产生音乐。

模块化：利用定时器 1 来计算时间是否结束，以音阶的频率来换算半周期的时间，即“0”电位和“1”电位的时间相同，当“0”电位的计时时间结束即产生计时中断，并输出“1”电位，当“1”电位的计时时间结束也产生计时中断并输出“0”电位，而形成一个方波，即利用计时中断的方法来产生每一音阶的方波频率，可规划为一个名叫 timer.c 的模块，因而，可将下列模块文件规划为一个项目文件，如下：

1. global.c：声明全局变量的数据类型及内存类型。

2. main.c：主程序，程序的进入点，为处理初始化的动作及不断调用演奏音乐的循环。

3. initial.c：开机后处理初始化的工作，包括 CPU 缓存器的初始设定及 CPU I/O 的初始值设定。

4. delay.c：以自变量来传递并最少延迟 10ms 的程序，即令 CPU 不断跑空循环达到延迟时间的模块。

5. music.c：每一音阶的音长和频率的内容，并将频率换算成时间，以作为定时器 1 实际要中断的时间依据的模块。

6. timer.c：定时器 1 中断的模块，用以产生每一音阶的方波，以便发出正确音阶的声音。

4-1-1　软件构建的思维

1. global.c：需要一个变量用以存储计算每一音阶频率的时间，变量名称为 period，即周期。

2. global.h：将全局变量声明成外部，以便其他模块使用。

3. main.c：进行初始化的函数调用和音乐演奏的函数调用。

4. initial.c：实际要执行初始化动作的模块，包含 CPU 缓存器的设定及 CPU I/O 的设定。

5. initial.h：initial.c 模块中所有函数的外部声明。

6. delay.c：延迟函数的模块，在 main( )函数中每 1 秒钟又重复演奏音乐。

7. delay.h：delay.c 模块中所有函数的外部声明。

8. music.c：音乐中每一个音阶的音长和频率，通过取表方式计算出相对的时间，并将时间存入至 Timer1 的缓存器内，及起动 Timer1 以令计时开始，待计时时间结束，用计时中断程序来输出方波，以推动晶体管令喇叭发出正确的音阶。

9. music.h：music.c 模块中的所有函数的外部声明，以及每一音阶的音长表和频率表的外部声明，并定义推动晶体管的喇叭输出为 I/O 3.1。

10. timer.c：用以将每一音阶中的半周期时间加载至 timer1 缓存器内，当半周期的时间到，将现阶段的输出反相，即将“0”电位转态为“1”电位，将“1”电位转态为“0”电位，而形成一个方波，当连续产生数十个方波后，即可令喇叭发出这个频率的声音了。

4-1-2　参数的意义说明与使用时机

Period：每一音阶频率的半周期时间

数据类型：占用 2 个字节的无符号整数类型（unsigned int）。

内存类型：变量地址介于 0X00～0Xff 之间，使用间接寻址模式（idata）。

写入：（1）以频率为主的音调表取出后乘以 2 的倒数并乘上 10^6 即表示此频率半周期的时间。

（2）将此时间被 65536 减去之后（定时器 1 为上数型）将高字节存入 TH1，低字节存入 TL1，以作为定时器 1 的中断时间。

4-1-3　软件的解决方法

1. global.c：将全局变量集合在此模块中以便统一管理，而 Period 全局变量的声明如下：

Word IDATA Period;

2. global.h：全局变量的外部声明包括文件，程序中只要包括“global.h”则其他模块也可存取全局变量，Period 全局变量的外部声明如下：

extern Word IDATA Period;

3. main.c：调用的函数为 PowerOnInit( )函数、Music( )音乐函数及 DelayX10ms( )函数，而利用重复循环 while(1)的叙述来达到每 1 秒钟就重复演奏的目的，程序写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      Music( );
      DelayX10ms(200);
   }
}
```

4. initial.c：实际执行初始化的动作，包含 InitialCpu( )函数及 InitialCpuIO( )函数，初始化 CPU 缓存器要设定：中断缓存器除能(IE)、选择缓存器库(PSW)、中断优先缓存器(IP)、计时计数模式设定缓存器(TMOD)、停止起动定时器 0 和定时器 1，设定外部中断 0 为低电频触发、只使能定时器 1、除能定时器 0 和外部中断 0，也要打开中断闸，程序中也要包括各模块文件的外部函数声明，外部全局变量的声明，CPU 8051 缓存器的地址定义，以及自定义类型的声明，程序写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
void PowerOnInit(void)
{
   InitialCpu();
```

```
    InitialCpuIO();
}

/*******************************/
void InitialCpu( void )
{
    IE  = 0;              //disable all interrupt
    PSW = 0;              //bank 0

    IP  = 0x0b;           //hi priority:int0,timer0,timer1

    TMOD= 0x11;           //set timer1,timer0 mode

    TR0 = 0;              //stop timer0
    TR1 = 0;              //stop timer1
    IT0 = 1;              //set int0:falling eage trigger

    EX0 = 0;              //disable int0 interrupt
    ET1 = 1;              //enable timer1 interrupt
    ET0 = 0;              //disable timer0 interrupt

    EA  = 1;              //enable all interrupt gate
}

/*******************************/
void InitialCpuIO(void)
{
    SPEAKER = 0;
}
```

5. initial.h：initial.c 的外部函数声明，其程序写法如下：

```
#ifndef   _INITIAL_H
#define   _INITIAL_H

extern void PowerOnInit(void);
extern void InitialCpu(void);
extern void InitialCpuIO(void);

#endif
```

6. delay.c：main( )函数中以每 1 秒钟就重复演奏，因而需要调用延迟函数，其程序写如下。

```
/*******************************/
```

```
/* include files              */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
void DelayX10ms(Word count)
{
   Word i,j,k;

   for(i=0; i<count; i++)
      for(j=0; j<10; j++)
         for(k=0; k<120; k++)
            ;
}
```

7. delay.h：delay.c 模块中 DelayX10ms( )函数的外部声明，写法如下：

```
#ifndef   _DELAY_H
#define   _DELAY_H

extern void DelayX10ms(Word count);

#endif
```

8. music.c：先将歌曲中每一音符的音长及频率建表，以使于程序中利用程序内存组来读取内容后并计算出其实际半周期的时间，并存入至定时器 11 的缓存器内，高字节存入 TH1，低字节存入 TL1，并开始激活定时器 1，则此时 I/O 脚 SPEAKER 端将因为计时时间（音符的半周期长）结束而不断地输出“1”、“0”的变化，以发出声音。而声音的长短以音长表为基准，并以三个循环的延迟后，停止计时，即不再产生计时中断，也不再输出方波，因此，也不再发出声音，即发出此音符的时间已经到了，就准备下一音符的音长。频率的取表再计算出半周期的时间后，令定时器 1 的缓存器内容为此音符的半周期，开始计时（即开始产生方波）并延迟发出此音符的时间后停止，一直到全部音符演奏完毕为止，其程序的写法如下：

```
/*****************************/

/* include files              */
/*****************************/
#include  "define.h"
```

```
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
//soundlong
Byte RDATA SOUNDLONG[ ] =
{
    4,4,6,2,
    4,4,8,
    4,4,4,4,
    4,4,8,

    4,4,4,2,
    4,4,6,2,
    4,2,2,4,2,2,
    4,4,8
};

//soundtone frequence
Word RDATA SOUNDTONE[ ] =
{
    523   ,523 ,587,659,
    784   ,784 ,659,
    784   ,784 ,880,988,
    1047,1047,784,
    1047,1047,880,784,
    659   ,880 ,784,659,
    523   ,587 ,659,784,880,784,
    659   ,587 ,523
};

void Music(void)
{
    Byte  i,j,k;
    Word  m;

    for(i=0; i<31; i++)
    {
      Period=1000000/(SOUNDTONE[i]*2);
      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;
```

```
        //wait timer1 interrupt
        for(j=0; j<SOUNDLONG[i]; j++)  //soundlong
           for(k=0; k<8; k++)          //tempo
              for(m=0; m<1080; m++)
                 ;
        //stop timer1:soundtone
        TR1 = 0;
     }
  }
```

9. music.h：音长表、频率表的外部声明，music( )函数的外部声明及推动喇叭的 I/O 脚定义，如果要改变喇叭的 I/O 脚位为 P1.0，可将#define SPEAKER P3_0 改成 #define SPEAKER P1_0，其程序写法如下：

```
#ifndef   _MUSIC_H
#define   _MUSIC_H

#define  SPEAKER   P3_1

extern Byte RDATA SOUNDLONG[ ];
extern Word RDATA SOUNDTONE[ ];
extern void Music(void);

#endif
```

10. timer.c：时间变量 Period 代表着每一音符半周期的时间，但由于 8051 CPU 的定时器 1 是上数型，也就是每经过 1us 则由 TH1、TL1 所组成的 16 位缓存器就会加 1（当 CPU 的振荡晶体是使用 12MHz）。而 16 位的最大数值为 65536，而当计数至 65536 时，再经过 1us 则 TH1、TL1 将变为 0X0000，并产生定时器 1 的中断，因此，将 65536-period 后重新再存入 TH1、TL1 以便下一次同一时间的中断，并将推动喇叭的 I/O 输出转态，其程序写法如下：

```
/*****************************/

/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
```

```
/*****************************/
//timer1,execute service route
/*****************************/
void  Timer1ISR ( void ) interrupt 3 using 2
{
    Word temp;

    temp = 65536 - Period;
    SPEAKER = !SPEAKER;

    TL1 = temp & 0xff;
    TH1 = temp >> 8;
    TF1 = 0;
}
```

汇编语言程序如下：

1. global.a51

```
$NOMOD51

NAME GLOBAL

$INCLUDE (mtv212m.inc)

?ID?GLOBAL          SEGMENT IDATA
PUBLIC    Period

RSEG  ?ID?GLOBAL
        Period:   DS   2

END
```

2. main.a51

```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN       SEGMENT CODE
EXTRN     CODE (PowerOnInit)
EXTRN     CODE (_DelayX10ms)
EXTRN     CODE (Music)
```

```
EXTRN    CODE (?C_STARTUP)
PUBLIC   Main

RSEG  ?PR?Main?MAIN
USING    0
Main:
LCALL    PowerOnInit
?C0001:
LCALL    Music
MOV      R7,#0C8H
MOV      R6,#00H
LCALL    _DelayX10ms
SJMP     ?C0001
RET

END
```

3. initial.a51

```
$NOMOD51

NAME INITIAL

$INCLUDE (mtv212m.inc)

?PR?PowerOnInit?INITIAL          SEGMENT CODE
?PR?InitialCpu?INITIAL           SEGMENT CODE
?PR?InitialCpuIO?INITIAL         SEGMENT CODE
PUBLIC   InitialCpuIO
PUBLIC   InitialCpu
PUBLIC   PowerOnInit

RSEG  ?PR?PowerOnInit?INITIAL
USING    0
PowerOnInit:
LCALL    InitialCpu
LCALL    InitialCpuIO
RET

RSEG  ?PR?InitialCpu?INITIAL
USING    0
InitialCpu:
CLR      A
MOV      IE,A
MOV      PSW,A
```

```
MOV       IP,#0BH
MOV       TMOD,#011H
CLR       TR0
CLR       TR1
SETB      IT0
CLR       EX0
SETB      ET1
CLR       ET0
SETB      EA
RET

RSEG  ?PR?InitialCpuIO?INITIAL
USING     0
InitialCpuIO:
CLR       P3_1
RET

END
```

4. delay.a51

```
$NOMOD51

NAME DELAY

$INCLUDE (mtv212m.inc)

?PR?_DelayX10ms?DELAY        SEGMENT CODE
?DT?_DelayX10ms?DELAY        SEGMENT DATA OVERLAYABLE
PUBLIC     DelayX10ms

RSEG  ?DT?_DelayX10ms?DELAY
?_DelayX10ms?BYTE:
     count?040:   DS   2

RSEG  ?PR?_DelayX10ms?DELAY
USING     0
_DelayX10ms:
MOV       count?040,R6
MOV       count?040+01H,R7
CLR       A
MOV       R7,A
MOV       R6,A
?C0001:
CLR       C
```

```
MOV     A,R7
SUBB    A,count?040+01H
MOV     A,R6
SUBB    A,count?040
JNC     ?C0010
CLR     A
MOV     R5,A
MOV     R4,A
?C0004:
CLR     A
MOV     R3,A
MOV     R2,A
?C0007:
INC     R3
CJNE    R3,#00H,?C0011
INC     R2
?C0011:
MOV     A,R3
XRL     A,#078H
ORL     A,R2
JNZ     ?C0007
?C0006:
INC     R5
CJNE    R5,#00H,?C0012
INC     R4
?C0012:
MOV     A,R5
XRL     A,#0AH
ORL     A,R4
JNZ     ?C0004
?C0003:
INC     R7
CJNE    R7,#00H,?C0013
INC     R6
?C0013:
SJMP    ?C0001
?C0010:
RET
```

5. music.a51

```
$NOMOD51

NAME MUSIC
```

```
$INCLUDE (mtv212m.inc)

?PR?Music?MUSIC          SEGMENT CODE
?DT?Music?MUSIC          SEGMENT DATA OVERLAYABLE
?CO?MUSIC                SEGMENT CODE
 EXTRN     IDATA (Period)
 EXTRN     CODE (?C?SLDIV)
 EXTRN     CODE (?C?SLSHR)
 PUBLIC    SOUNDTONE
 PUBLIC    SOUNDLONG
 PUBLIC    Music

 RSEG  ?DT?Music?MUSIC
?Music?BYTE:
        i?040:   DS   1

 RSEG  ?CO?MUSIC
SOUNDLONG:
 DB   004H,004H,006H,002H,004H
 DB   004H,008H,004H,004H,004H
 DB   004H,004H,004H,008H,004H
 DB   004H,004H,002H,004H,004H
 DB   006H,002H,004H,002H,002H
 DB   004H,002H,002H,004H,004H
 DB   008H

SOUNDTONE:
 DW   0020BH,0020BH,0024BH,00293H
 DW   00310H,00310H,00293H,00310H
 DW   00310H,00370H,003DCH,00417H
 DW   00417H,00310H,00417H,00417H
 DW   00370H,00310H,00293H,00370H
 DW   00310H,00293H,0020BH,0024BH
 DW   00293H,00310H,00370H,00310H
 DW   00293H,0024BH,0020BH

 RSEG  ?PR?Music?MUSIC
 USING    0
Music:
 CLR      A
 MOV      i?040,A
?C0001:
 MOV      A,i?040
 ADD      A,ACC
 ADD      A,#LOW (SOUNDTONE)
```

```
MOV     DPL,A
CLR     A
ADDC    A,#HIGH (SOUNDTONE)
MOV     DPH,A
MOV     A,#01H
MOVC    A,@A+DPTR
ADD     A,ACC
MOV     R7,A
CLR     A
MOVC    A,@A+DPTR
RLC     A
MOV     R6,A
CLR     A
MOV     R4,A
MOV     R5,A
MOV     R0,A
MOV     R1,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
MOV     R7,#040H
MOV     R6,#042H
MOV     R5,#0FH
LCALL   ?C?SLDIV
MOV     R0,#Period
MOV     A,R6
MOV     @R0,A
INC     R0
MOV     A,R7
MOV     @R0,A
CLR     A
MOV     R4,A
MOV     R5,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
SUBB    A,R3
MOV     R7,A
```

```
CLR     A
SUBB    A,R2
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
SUBB    A,#00H
MOV     R4,A
MOV     A,R7
CLR     A
MOV     TL1,R7
DEC     R0
MOV     A,@R0
MOV     R6,A
INC     R0
MOV     A,@R0
MOV     R7,A
CLR     A
MOV     R4,A
MOV     R5,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
SUBB    A,R3
MOV     R7,A
CLR     A
SUBB    A,R2
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
SUBB    A,#00H
MOV     R4,A
MOV     R0,#08H
LCALL   ?C?SLSHR
MOV     TH1,R7
SETB    TR1
CLR     A
MOV     R7,A
```

```
?C0004:
 MOV       A,i?040
 MOV       DPTR,#SOUNDLONG
 MOVC      A,@A+DPTR
 MOV       R6,A
 MOV       A,R7
 CLR       C
 SUBB      A,R6
 JNC       ?C0005
 CLR       A
 MOV       R6,A
?C0007:
 CLR       A
 MOV       R5,A
 MOV       R4,A
?C0010:
 INC       R5
 CJNE      R5,#00H,?C0014
 INC       R4
?C0014:
 CJNE      R4,#04H,?C0010
 CJNE      R5,#038H,?C0010
?C0009:
 INC       R6
 CJNE      R6,#08H,?C0007
?C0006:
 INC       R7
 SJMP      ?C0004
?C0005:
 CLR       TR1
 INC       i?040
 MOV       A,i?040
 XRL       A,#01FH
 JZ    $ + 5H
 LJMP      ?C0001
?C0013:
 RET

 END
```

6. timer.a51

```
$NOMOD51

NAME TIMER
```

```
$INCLUDE (mtv212m.inc)

?PR?Timer1ISR?TIMER    SEGMENT CODE
 EXTRN    IDATA (Period)
 PUBLIC   Timer1ISR

CSEG AT   0001BH
 LJMP Timer1ISR

 RSEG  ?PR?Timer1ISR?TIMER
 USING    2
Timer1ISR:
 PUSH     ACC
 PUSH     PSW
 MOV      PSW,#010H
 MOV      R0,#Period
 MOV      A,@R0
 MOV      R6,A
 INC      R0
 MOV      A,@R0
 MOV      R7,A
 CLR      C
 CLR      A
 SUBB     A,R7
 MOV      R7,A
 CLR      A
 SUBB     A,R6
 MOV      R6,A
 CPL      P3_1
 XCH      A,R5
 MOV      A,R7
 XCH      A,R5
 MOV      A,R5
 MOV      TL1,A
 MOV      A,R6
 MOV      TH1,A
 CLR      TF1
 POP      PSW
 POP      ACC
RETI

END
```

4-2 宏指令的英文字

目的：将音长表的内容以宏指令定义英文字表示，而音调表的内容也以宏指令定义英文字直接计算出半周期的时间，利用前置处理器先计算出每一音符的半周期时间，程序中直接取表而无需以频率再计算出半周期时间，以节省 CCPU 的运算速度。

4-2-1 软件构建的思维

在 Music( )函数中需要将每一音符的半周期时间存入至定时器 1 的缓存器内（TH1、TL1），以便利用此时间来产生定时器 1 的中断而令推动喇叭的输出反相，每一次的中断就输出反相，即当目前的输出是“0”电位，则中断后立即输出“1”电位。此时，继续计时半周期的时间，当时间结束又再中断则将“1”电位又变成输出“0”电位，即表示“1”电位总共输出了半周期的时间，而以每一音符的音长来控制输出“0”电位及输出“1”电位的个数，当音长越长则发出声音越久，音长越短则发出的声音就越短，将每一音符按此方法达到演奏的目的。

1. music.c

（1）音长表的内容以大写英文字母来表示，由于大写英文字可能会和其他的定义冲突，因此，在英文字之前加上底线以便区隔和阅读，而以望文生义的英文字表示音符的拍长比较明了。例如：_WHOLE 为全音符、_HALF 为二分音符、_QUARTER 为四分音符、_EIGHTH 为八分音符、_SIXTEENTH 为十六分音符、_THIRTYSECOND 为三十二音符等，而最后一个字为小写英文 d 表示为附点的意思，例如：_QUARTERd 为附点四分音符等。

（2）音调表的内容以唱名表示，也以底线为开头，如为低音则在第一个英文字之前先写上数字 1、2、3，数字越大表示越低音。例：_1DO 表示为低音 DO(1)，_2DO 表示次低音 DO(1)，比_1DO 低 8 度音，_1DO 比_DO 也低 8 度音；如为高音则在最后英文字之后写上数字 1、2、3，数字越大表示越高音，例：_DO1 表示为高音 DO(1)，_DO2 表示次高音 DO(1)，_DO2 比_DO1 高 8 度音，_DO1 比_DO 也高 8 度音，音阶唱名的表示分别为_DO(1)、_RE(2)、_MI(3)、_FA(4)、_SO(5)、_LA(6)、_TI(7)。

（3）在音调表中先行利用宏指令计算出半周期的时间，在程序中直接取表后存入 TH1、TL1 缓存器内，并开始起动计器以发生声音，以音长表来等待发出声音的长短，待延迟循环结束后即表示此音符演奏完毕并停止计时，依此方法即可将全部音符演奏完毕。

2. music.h

演奏音乐函数的包括文件定义，包含音符：全音符、二分音符、四分音符、八分音符、十六分音符、三十二分音符的定义以及附点音符的定义；音阶：从音阶 2 个低 8 度音的 DO(_2DO)至 3 个高 8 度音的 TI(_TI3)之间音阶频率的半周期时间的计算，以及外部函数的声明，并将喇叭的输出 I/O 引脚也一并进行定义。

4-2-2　软件的解决方法

1. music.c

音长表中以底线为开头的英文字，代表着音符的拍长，当经过前置处理器处理后，会换算为实际的数值，但仍以英文字来表示较易了解每一音符的拍长。实际上乐谱中每一音符也以拍长作记号，例如：四分音符、附点八分音符等，而音调表中以底线为开头的英文字，代表着每一音符的频率，经过前置处理器处理后会计算出每一音符实际的半周期时间，将这些时间以取表方式取出后，将低字节赋给 TL1，高字节赋给 TH1。由于是上数型定时器，因此要利用 65536 减去周期变量 period 并向右移 8 个位，即取出高字节并存入 TH1，当 65536 减去 period 变量并和 0Xff 作逻辑 AND，即得出低字节并存入 TL1，而 TR1=1 为激活定时器 1，以音调表为基准而且当作完三层循环后，清除 TR1，即表示此一音符演奏完成，通过 for 循环来继续演奏下一个音符。此首曲目为“三轮车”，总共有 31 个音符，当 31 个音符演奏完毕后，则返回 main( )函数，继续做下一语句，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
    _EIGHTH,_EIGHTH   ,_EIGHTHd  ,_SIXTEENTH,
    _EIGHTH,_EIGHTH   ,_QUARTER  ,
    _EIGHTH,_EIGHTH   ,_EIGHTH   ,_EIGHTH   ,
    _EIGHTH,_EIGHTH   ,_QUARTER  ,
```

```
    _EIGHTH,_EIGHTH   ,_EIGHTHd  ,_SIXTEENTH,
    _EIGHTH,_EIGHTH   ,_EIGHTHd  ,_SIXTEENTH,
    _EIGHTH,_SIXTEENTH,_SIXTEENTH,_EIGHTH,_SIXTEENTH,
    _SIXTEENTH,
    _EIGHTH,_EIGHTH   ,_QUARTER
};

Word RDATA SOUNDTONE[ ] =
{
    _DO    ,_DO ,_RE,_MI,
    _SO ,_SO ,_MI,
    _SO ,_SO ,_LA,_TI,
    _DO1,_DO1,_SO,
    _DO1,_DO1,_LA,_SO,
    _MI    ,_LA ,_SO,_MI,
    _DO ,_RE ,_MI,_SO,_LA,_SO,
    _MI    ,_RE ,_DO
};

void Music(void)
{
    Byte  i,j,k;
    Word  m;

    for(i=0; i<31; i++)
    {
      Period=SOUNDTONE[i];
      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      //wait timer1 interrupt
      for(j=0; j<SOUNDLONG[i]; j++)        //soundlong
        for(k=0; k<TEMPO; k++)             //tempo
          for(m=0; m<1080; m++)
            ;
      //stop timer1:soundtone
      TR1 = 0;
    }
}
```

2. music.h

在程序前端必须包括此文件，否则所有以底线为开头的英文字将无法找到对应的数值而造成错误。music.h 必须定义音符拍长与音阶的半周期，喇叭输出定义为P3.0，也可以任意修改I/O的引脚，但需配合硬件线路，当然所有在music.c模块下

的函数也必须在对应的包括文件作外部声明，在此为 music.h，将扩展名.c 改为.h 即为该模块文件的包括文件。包括文件的第一行语句#ifndef _-MUSIC_H 表示为_MUSIC_H 符号未被定义则开始做第二行以下的语句，而第二行的语句#define _MUSIC_H 表示为_MUSIC_H 已经定义了，则往后在其他模块即使使用 include "music.h"，由于_MUSIC_H 已经被定义过了，因此就不会再做 music.h 中第二行以下的语句了，其中音阶最后一个英文字为小写 r，表示升半音。

包括文件的写法如下：

```
#ifndef   _MUSIC_H
#define   _MUSIC_H

#define  SPEAKER              P3_1

//以四分音符为一拍
#define  TEMPO                8
#define  _WHOLE               TEMPO*4        //全音符
#define  _WHOLEd              TEMPO*6        //附点全音符
#define  _HALF                TEMPO*2        //二分音符
#define  _HALFd               TEMPO*3        //附点二分音符
#define  _QUARTER             TEMPO*1        //四分音符
#define  _QUARTERd            TEMPO*3/2      //附点四分音符
#define  _EIGHTH              TEMPO*1/2      //八分音符
#define  _EIGHTHd             TEMPO*3/4      //附点八分音符
#define  _SIXTEENTH           TEMPO*1/4      //十六分音符
#define  _SIXTEENTHd          TEMPO*3/8          //附点十六分音符
#define  _THIRTYSECOND  TEMPO*1/8      //三十二分音符

#define  _2DO       1000000/(131*2)
#define  _2DOr      1000000/(139*2)
#define  _2RE       1000000/(147*2)
#define  _2REr      1000000/(156*2)
#define  _2MI       1000000/(165*2)
#define  _2FA       1000000/(175*2)
#define  _2FAr      1000000/(185*2)
#define  _2SO       1000000/(196*2)
#define  _2SOr      1000000/(208*2)
#define  _2LA       1000000/(220*2)
#define  _2LAr      1000000/(233*2)
#define  _2TI       1000000/(247*2)
#define  _1DO       1000000/(262*2)
#define  _1DOr      1000000/(277*2)
#define  _1RE       1000000/(294*2)
#define  _1REr      1000000/(311*2)
```

```
#define  _1MI      1000000/(330*2)
#define  _1FA      1000000/(349*2)
#define  _1FAr     1000000/(370*2)
#define  _1SO      1000000/(392*2)
#define  _1SOr     1000000/(415*2)
#define  _1LA      1000000/(440*2)
#define  _1LAr     1000000/(466*2)
#define  _1TI      1000000/(494*2)
#define  _DO       1000000/(523*2)
#define  _DOr      1000000/(554*2)
#define  _RE       1000000/(587*2)
#define  _REr      1000000/(622*2)
#define  _MI       1000000/(659*2)
#define  _FA       1000000/(698*2)
#define  _FAr      1000000/(740*2)
#define  _SO       1000000/(784*2)
#define  _SOr      1000000/(831*2)
#define  _LA       1000000/(880*2)
#define  _LAr      1000000/(932*2)
#define  _TI       1000000/(988*2)
#define  _DO1      1000000/(1047*2)
#define  _DO1r     1000000/(1109*2)
#define  _RE1      1000000/(1175*2)
#define  _RE1r     1000000/(1245*2)
#define  _MI1      1000000/(1319*2)
#define  _FA1      1000000/(1397*2)
#define  _FA1r     1000000/(1480*2)
#define  _SO1      1000000/(1568*2)
#define  _SO1r     1000000/(1661*2)
#define  _LA1      1000000/(1760*2)
#define  _LA1r     1000000/(1865*2)
#define  _TI1      1000000/(1976*2)
#define  _DO2      1000000/(2093*2)
#define  _DO2r     1000000/(2217*2)
#define  _RE2      1000000/(2349*2)
#define  _RE2r     1000000/(2489*2)
#define  _MI2      1000000/(2637*2)
#define  _FA2      1000000/(2794*2)
#define  _FA2r     1000000/(2960*2)
#define  _SO2      1000000/(3136*2)
#define  _SO2r     1000000/(3322*2)
#define  _LA2      1000000/(3520*2)
#define  _LA2r     1000000/(3729*2)
#define  _TI2      1000000/(3951*2)
#define  _DO3      1000000/(4186*2)
```

```
#define  _DO3r      1000000/(4435*2)
#define  _RE3       1000000/(4699*2)
#define  _RE3r      1000000/(4978*2)
#define  _MI3       1000000/(5274*2)
#define  _FA3       1000000/(5587*2)
#define  _FA3r      1000000/(5919*2)
#define  _SO3       1000000/(6271*2)
#define  _SO3r      1000000/(6645*2)
#define  _LA3       1000000/(7040*2)
#define  _LA3r      1000000/(7459*2)
#define  _TI3       1000000/(7902*2)

extern Byte RDATA SOUNDLONG[ ];
extern Word RDATA SOUNDTONE[ ];
extern void Music(void);

#endif
```

音乐演奏函数的汇编程序如下：

```
$NOMOD51

NAME MUSIC

$INCLUDE (mtv212m.inc)

?PR?Music?MUSIC          SEGMENT CODE
?DT?Music?MUSIC          SEGMENT DATA OVERLAYABLE
?CO?MUSIC                SEGMENT CODE
 EXTRN     IDATA (Period)
 EXTRN     CODE (?C?SLSHR)
 PUBLIC    SOUNDTONE
 PUBLIC    SOUNDLONG
 PUBLIC    Music

 RSEG  ?DT?Music?MUSIC
?Music?BYTE:
        i?040:   DS   1

 RSEG  ?CO?MUSIC
SOUNDLONG:
 DB   004H,004H,006H,002H,004H
 DB   004H,008H,004H,004H,004H
 DB   004H,004H,004H,008H,004H
 DB   004H,006H,002H,004H,004H
```

```
DB   006H,002H,004H,002H,002H
DB   004H,002H,002H,004H,004H
DB   008H

SOUNDTONE:
DW   003BCH,003BCH,00353H,002F6H
DW   0027DH,0027DH,002F6H,0027DH
DW   0027DH,00238H,001FAH,001DDH
DW   001DDH,0027DH,001DDH,001DDH
DW   00238H,0027DH,002F6H,00238H
DW   0027DH,002F6H,003BCH,00353H
DW   002F6H,0027DH,00238H,0027DH
DW   002F6H,00353H,003BCH

RSEG  ?PR?Music?MUSIC
USING    0
Music:
CLR      A
MOV      i?040,A
?C0001:
MOV      A,i?040
ADD      A,ACC
ADD      A,#LOW (SOUNDTONE)
MOV      DPL,A
CLR      A
ADDC     A,#HIGH (SOUNDTONE)
MOV      DPH,A
CLR      A
MOVC     A,@A+DPTR
MOV      R6,A
MOV      A,#01H
MOVC     A,@A+DPTR
MOV      R7,A
MOV      R0,#Period
MOV      A,R6
MOV      @R0,A
INC      R0
MOV      A,R7
MOV      @R0,A

CLR      A
MOV      R4,A
MOV      R5,A
XCH      A,R2
MOV      A,R6
```

```
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
SUBB    A,R3
MOV     R7,A
CLR     A
SUBB    A,R2
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
SUBB    A,#00H
MOV     R4,A
MOV     A,R7
CLR     A
MOV     TL1,R7

DEC     R0
MOV     A,@R0
MOV     R6,A
INC     R0
MOV     A,@R0
MOV     R7,A
CLR     A
MOV     R4,A
MOV     R5,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
SUBB    A,R3
MOV     R7,A
CLR     A
SUBB    A,R2
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
```

```
SUBB    A,#00H
MOV     R4,A
MOV     R0,#08H
LCALL   ?C?SLSHR
MOV     TH1,R7
SETB    TR1

CLR     A
MOV     R7,A
?C0004:
MOV     A,i?040
MOV     DPTR,#SOUNDLONG
MOVC    A,@A+DPTR
MOV     R6,A
MOV     A,R7
CLR     C
SUBB    A,R6
JNC     ?C0005

CLR     A
MOV     R6,A
?C0007:
CLR     A
MOV     R5,A
MOV     R4,A
?C0010:
INC     R5
CJNE    R5,#00H,?C0014
INC     R4
?C0014:
CJNE    R4,#04H,?C0010
CJNE    R5,#038H,?C0010
?C0009:
INC     R6
CJNE    R6,#08H,?C0007
?C0006:
INC     R7
SJMP    ?C0004
?C0005:
CLR     TR1
INC     i?040
MOV     A,i?040
XRL     A,#01FH
JZ   $ + 5H
LJMP    ?C0001
```

```
?C0013:
RET


END
```

4-3　宏指令法的数字

目的：由于音长表中以常数定义的英文字太冗长，每当输入一个音符则需要键入很多个英文字，太浪费时间了，而且也由于英文字太多易造成阅读上的困难，因而改变为以底线为开头的数字来表示音符的长短。

4-3-1　软件构建的思维

音符中四分音符则须键入_QUARTER，八分音符须键入_EIGHTH、十六音符则须键入_SIXTEENTH 等英文字，你不觉得英文字太多，如果一首歌有前奏、二段、间奏及尾奏，或许可以有 200 个以上的音符。一个音符宏定义的英文字太冗长，会敲到手酸而且不易维护，有什么方法可以代表音符的意义，文字又简短的，以这样的思维是不是就很容易想到用数字来表示。_1 表示为全音符，_2 表示为二分音符，_4 表示为四分音符，_8 表示为八分音符，_16 表示为十六分音符，_32 表示为三十二分音符，_1d 为附点全音符，_2d 为附点二分音符，_4d 为附点四分音符，依此类推。

4-3-2　软件的解决方法

1. music.c

程序中音长表的内容变成以底线为开头加上数字，如果有小写英文字 d 表示为附点音符，如下所示：

```
/*******************************/

Byte RDATA SOUNDLONG[ ] =
{
    _8,_8 ,_8d,_16,
    _8,_8 ,_4 ,
    _8,_8 ,_8 ,_8 ,
    _8,_8 ,_4 ,
```

```
    _8,_8 ,_8d,_16,
    _8,_8 ,_8d,_16,
    _8,_16,_16,_8 ,_16,_16,
    _8,_8 ,_4
};
```

2. mucis.h

包含音符宏指令的定义，也包含附点音符的定义，其中常数 TEMPO 为演奏速度的意思。数值越大则演奏速度就越慢，因为音符的长短实际上是以 TEMPO 乘上拍数，并作为程序延迟的循环。而以四分音符为一拍，则二分音符为二拍，其_2 的实际数值为_2=TEMPO*2，而_4d 为附点四分音符有 1.5 拍，则其实际数值为_4d=TEMPO*3/2，而_32 为三十二分音符有 1/8 拍，其实际数值为_32=TEMPO*1/8，因而，定义 TEMPO 最小数值为 8，而且必须是 8 的倍数，以便可以整除。

```
//以四分音符为一拍
#define  TEMPO 8
#define  _1        TEMPO*4       //全音符
#define  _1d       TEMPO*6       //附点全音符
#define  _2        TEMPO*2       //二分音符
#define  _2d       TEMPO*3       //附点二分音符
#define  _4        TEMPO*1       //四分音符
#define  _4d       TEMPO*3/2     //附点四分音符
#define  _8        TEMPO*1/2     //八分音符
#define  _8d       TEMPO*3/4     //附点八分音符
#define  _16       TEMPO*1/4     //十六分音符
#define  _16d      TEMPO*3/8     //附点十六分音符
#define  _32       TEMPO*1/8     //三十二分音符
```

4-4 自动转换类型

目的：音长一样以定时器 1 来计算，取代以三层循环的延迟，这样每一音符的音长会比较准确，但需注意数据类型的限制。

软件构建的思维与解决方法

在程序中执行运算时注意变量的最大数值，当运算结果超过最大数值则会舍去超过的字节，只留下在范围内的字节作为结果值，那么，一定会产生错误。以 Word（unsigned int）类型声明的变量只占用两个字节，其最大数值为 0Xffff，当运算超

过 0Xffff 则将会舍去第三位组以上的数值，结果势必会形成错误，应特别留意。例如：声明 Word 变量为 SouldLongCount 代表此一音符半周期时间的音长个数，当执行 SoundLongCount=((Tempo*SOUNDLONG[i]*1000)/8/Period 的运算，看似没问题，实际上在计算 Tempo*SOUNDLONG[i]*1000 会很容易超过两个字节的最大数值 0Xffff，当超过后仍只取第一和第二字节继续作除 8 和除 Period 变量的运算，你想最后的结果值 SoundLongCount 变量会正确吗？

容易出错的参数范例如下，您知道错在那儿？如何修正吗？正确的写法请参考下一章节。

```
void Music(void)
{
    Byte i;

    Tempo = MODERATO;

    for(i=0; i<31; i++)
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE[i];
      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      //is error:运算数值超过 unsigned int 类型的最大值(0~65536)
      //Tempo*SOUNDLONG[i]*1000)=0x201ac0 -->0x1ac0
      //Period=956,so SoundLongCount=(0x1ac0/8)/956=0
      SoundLongCount=((Tempo*SOUNDLONG[i]*1000)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
    }
}
```

4-5　音长中断法 for 循环

目的：利用定时器 1 来作为每一音符音长控制，以便得到更准确的时间。

4-5-1　软件构建的思维

以取表音长的方式，并作为延迟音长循环的参数，固然可行，但其弹性及准确

度并不如以定时器的方式来得有效率和可靠，程序中以 Timer1 来产生音符的半周期时间，即利用此半周期时间来产生中断，以一个变量来计算出音长的时间到底是半周期时间的几倍，而在计时中断程序中，每中断一次就令此变量不断地减 1，当变量的数值为 0，就不再减 1，也表示音长的时间到了，利用此技巧，即可达到以定时器的方法来控制每一音符的演奏时间了。

1. global.c

分别要增加 Tempo 变量，以代表演奏速度，想要改变演奏速度，只要在程序中改变程序即可，而 SoundLongCount 即代表此音符的音长是此音符半周期时间的倍数。

2. globale.h

新增变量的外部声明，以利于存取这两个新增变量的模块使用，如果不作外部声明，则编译时将会出现错误。

3. music.c

由于 SoundLongCount 是两个字节的无符号数据类型（unsigned int），程序在运算中为避免超过其声明的最大字节，必须将数值 1000 强迫转态为 long 数据类型，long 数据类型为占用 4 个字节，其范围为－2147483648～2147483647 即十六进制 -80000000～0X7fffffff，而表达式 Tempo*SOUNDLONG[i]*1000L 就不会被舍去第 3 字节以后的数值，则 SoundLongCount 变量才能正确。

4. music.h

将演奏速度也以宏指令定义，当想要改变演奏曲目的快慢时，以 Tempo 变量设定演奏速度的常数即可，程序中自动以此变量值计算出改变后的音长，程序设计千万不要修正某一个条件后就大费周章地修改很多的模块，最好以设定某个数值，或类似简单的方法来作为程序结构的基础。

5. timer.c

中断程序必须将 SoundLongCount 不断地减 1，即每中断一次就减 1，但如果 SoundLongCount 已经等于 0 了，再减 1 不就成为 0Xffff，而会造成错误，因此，要减 1 之前必须事先判断是否已经等于 0 了，当不等于 0 才可以不断地减 1，这样的思维才正确。

4-5-2 参数的意义说明与使用的时机

1. Tempo：演奏速度的参数

- 数据类型：占用两个位的无符号整数类型(unsigned int)。
- 内存类型：变量地址介于 0X00～0Xff 之间，使用间接寻址模式(idata)。

- 写入：将各种不同的演奏速度定义成常数，如果要修正演奏的速度，可利用此变量直接设定演奏速度常数。
- 计算：计算出每一个音符在此演奏速度下所持续时间的一个变量。

2. Period: 音符的半周期时间

- 数据类型：占用二个位的无符号整数类型(unsigned int)。
- 内存类型：变量地址介于 0X00～0Xff 之间，使用间接寻址模式(idata)。
- 写入：半周期时间的计算，利用宏先定义完成，直接取出音调表的内容即为半周期的时间。
- 计算：
  （1）将此变量内容以上数型的计算方式(65536-Period)将高字节写入至 TH1，低字节写入至 TL1，以作为定时器 1 中断的时间间隔，也就是输出“0”电位及“1”电位的时间。
  （2）音长的时间换算，是利用有多少次的中断来判断音长的时间是否结束，而中断时间就是以 Period 变量的内容为基准，因此，将音长时间除以 Period 即为中断的次数，也就是音符演奏的结束依据。
  （3）TH1、TL1 的数值必须由 0Xffff→0X0000 才能产生中断，也就是每当进入 Timer1 中断程序，则 TH1、TL1 皆为 0X00，但由于中断程序也需要 CPU 的运算而占用数 us，看中断程序的处理内容为何。而 TH1、TL1 有不一样的时间，则在中断程序结束前必须将 period 内容重新加载至 TH1、TL1 并清除溢位标志 TF1，以确定下一次相同的中断时间。

3. SoundLongCount: 以 Period 变量内容为中断时间的相对音长的个数

- 数据类型：占用二个位的无符号整数类型。
- 内存类型：变量地址介于 0X00～0Xff 之间，使用间接寻址模式(idata)。
- 计算：
  （1）计算出此音符音长要有多少次的中断时间，其计算式为

  SoundLongCount =((Tempo*SOUNDLONG[i]*1000L)/8)/period

  其中，Tempo：为演奏速度的变量。

  SOUNDLONG[i]：每个音符的音长时间，以四分音符为一拍，则三十二分音符为 1/8 拍，因此预先乘以 8，即以三十二分音符的数值为 1。

  1000L：将 ms 的时间乘以 1000 换算为 us，因为 Period 变量内容的时间单位是 us，英文字 L 是将此表达式的乘积以 long 类型来存储，因为计算式会以最长的数据类型作为最长的数据类型来运算。

除以 8：因音符的宏定义已事先乘以 8，所以要再除以 8，如此，音符的时间才会正确。

除以 Period：Period 为中断时间，将音长时间除以中断时间就成为中断的次数，也就是当达到这些次数表示发出此一音符的时间终止了。

（2）每中断一次则此变量必须减 1，直到 0 为止。

- 判断：

（1）在执行音乐函数必须等待音长的时间是否结束，如果未结束则继续等待，如果结束，程序继续执行便立即停止定时器 1 的计时动作。

（2）在中断程序中必须先判断 SoundLongCount 是否为 0，如果为 0 则不再减 1，不为 0 则每中断 1 次就减 1。

4-5-3 软件的解决方法

新增变量必须在 global.c 模块中进行声明，而利用 global.h 包括文件作外部变量的声明，音长、音阶的定义也需先定义，以便程序直接取表。为使演奏速度较有弹性，直接以 Tempo 变量来设定为演奏速度常数，再根据此 Tempo 变量作运算，以改变不同演奏速度下音符的音长，所以各种演奏速度：缓板、慢板、稍慢板、行板、中板、快板、稍快板、急板等，都必须在 music.h 先行定义，各模块的解决方法如下：

1. global.c

全局变量的数据类型，内存类型及变量名称的声明，声明的写法如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"

/*******************************/
// Global Data
/*******************************/

//music
Word  IDATA   Tempo;
Word  IDATA   Period;
Word  IDATA   SoundLongCount;
```

2. global.h

Tempo、Period、SoundLongCount 全局变量的外部声明，以便在其他模块存取、判断、计算，循环或作为自变量之用，其外部声明的写法如下：

```
#ifndef   _GLOBAL_H
#define   _GLOBAL_H

//music
extern Word  IDATA   Tempo;
extern Word  IDATA   Period;
extern Word  IDATA   SoundLongCount;

#endif
```

3. music.c

以 Tempo 等于演奏速度的常数，只要任意改变常数值，则相对的音符其时间也会相对地修正，而先定义好每一个音阶的半周期时间，再直接取表会较简捷，并存入至 TH1、TL1 作为定时器 1 的中断时间，激活定时器后，计算出要中断的次数，程序便一直等待中断次数是否到了。当中断次数到达了 SoundLongCount 变量内容时，表示此一音符演奏完成，便立即停止计时，如此，以 for 循环反复执行 31 次即可将整首曲目演奏完毕，程序写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
   _8,_8 ,_8d,_16,
   _8,_8 ,_4 ,
   _8,_8 ,_8 ,_8 ,
   _8,_8 ,_4 ,

   _8,_8 ,_8d,_16,
   _8,_8 ,_8d,_16,
```

```
    _8,_16,_16,_8 ,_16,_16,
    _8,_8 ,_4
};

Word RDATA SOUNDTONE[ ] =
{
    _DO ,_DO ,_RE,_MI,
    _SO ,_SO ,_MI,
    _SO ,_SO ,_LA,_TI,
    _DO1,_DO1,_SO,
    _DO1,_DO1,_LA,_SO,
    _MI ,_LA ,_SO,_MI,
    _DO ,_RE ,_MI,_SO,_LA,_SO,
    _MI ,_RE ,_DO
};

void Music(void)
{
    Byte i;

    Tempo = MODERATO;

    for(i=0; i<31; i++)
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE[i];
      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
    }
}
```

4. music.h

定义每一音符、音阶的数值，作为程序中音长表和音调表的实际数值，以便让程序以数组方式取表，由于 Tempo 是直接以演奏速度设定，因此，也必须定义各种演奏速度的数值，音乐中演奏速度是每一分钟有几拍，但为换算音长为 ms 的时间单位，改为每一拍有多少的 ms，其宏定义如下：

```
//演奏速度 = ms/拍
```

```
#define  LARGHISSIMO    60000/40      //极缓板
#define  LARGO          60000/50      //最缓板
#define  LARGHETTO      60000/60      //甚缓板
#define  GRAVE          60000/63      //稍缓板
#define  LENTO          60000/66      //缓板
#define  ADAGIO         60000/70      //慢板
#define  ADAGIETTO      60000/80      //稍慢板
#define  ANDANTE        60000/92      //行板
#define  ANDANTINO      60000/103//小行板
#define  MODERATO       60000/114//中板
#define  ALLEGRETTO     60000/130//稍快板
#define  ALLEGRO        60000/144//快板
#define  VIVACE         60000/155//甚快板
#define  VELOCE         60000/166//稍急板
#define  PRESTO         60000/177//急板
#define  PRESTISSIMO    60000/188//最急板
```

5. timer.c

音长的次数是以半周期时间为除数而换算出来的，而中断时间即是半周期的时间。也就是 Period 变量的内容，所以在中断程序中必须先判断 SoundLongCount 变量是否已经为 0 了，不为 0 才能中断一次就减 1，而输出波形每中断一次就转态一次，并需将半周期时间重新加载 TH1、TL1 内，而且必须清除溢位标志 TF1。8051 内部的硬件也会在执行中断程序后自动清除，但仍以软件再作一次，以确保下一次仍可继续中断，因为有些中断设备其中断溢位标志并不会由硬件自动清除，其程序写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
//timer1,execute service route
void  Timer1ISR ( void ) interrupt 3 using 2
{
   Word temp;

   if ( SoundLongCount != 0 )
```

```
        SoundLongCount--;

    temp = 65536 - Period;
    SPEAKER = !SPEAKER;

    TL1 = temp & 0xff;
    TH1 = temp >> 8;
    //TF1 = 0;
}
```

其汇编程序如下：

1. global.a51

```
$NOMOD51

NAME GLOBAL

$INCLUDE (mtv212m.inc)

?ID?GLOBAL              SEGMENT IDATA
 PUBLIC   SoundLongCount
 PUBLIC   Period
 PUBLIC   Tempo

 RSEG  ?ID?GLOBAL
        Tempo:   DS   2
       Period:   DS   2
 SoundLongCount:   DS   2

 END
```

2. music.a51

```
$NOMOD51

NAME MUSIC

$INCLUDE (mtv212m.inc)

?PR?Music?MUSIC         SEGMENT CODE
?DT?Music?MUSIC         SEGMENT DATA OVERLAYABLE
?CO?MUSIC               SEGMENT CODE
 EXTRN    IDATA (Tempo)
 EXTRN    IDATA (Period)
 EXTRN    IDATA (SoundLongCount)
```

```
EXTRN     CODE (?C?SLSHR)
EXTRN     CODE (?C?IMUL)
EXTRN     CODE (?C?LMUL)
EXTRN     CODE (?C?SLDIV)
PUBLIC    SOUNDTONE
PUBLIC    SOUNDLONG
PUBLIC    Music

RSEG  ?DT?Music?MUSIC
?Music?BYTE:
        i?040:   DS   1

RSEG  ?CO?MUSIC
SOUNDLONG:
DB    004H,004H,006H,002H,004H
DB    004H,008H,004H,004H,004H
DB    004H,004H,004H,008H,004H
DB    004H,006H,002H,004H,004H
DB    006H,002H,004H,002H,002H
DB    004H,002H,002H,004H,004H
DB    008H

SOUNDTONE:
DW    003BCH,003BCH,00353H,002F6H
DW    0027DH,0027DH,002F6H,0027DH
DW    0027DH,00238H,001FAH,001DDH
DW    001DDH,0027DH,001DDH,001DDH
DW    00238H,0027DH,002F6H,00238H
DW    0027DH,002F6H,003BCH,00353H
DW    002F6H,0027DH,00238H,0027DH
DW    002F6H,00353H,003BCH

RSEG  ?PR?Music?MUSIC
USING     0
Music:
MOV       R0,#Tempo
MOV       @R0,#02H
INC       R0
MOV       @R0,#0EH
CLR       A
MOV       i?040,A
?C0001:
MOV       A,i?040
ADD       A,ACC
ADD       A,#LOW (SOUNDTONE)
```

```
MOV     DPL,A
CLR     A
ADDC    A,#HIGH (SOUNDTONE)
MOV     DPH,A
CLR     A
MOVC    A,@A+DPTR
MOV     R6,A
MOV     A,#01H
MOVC    A,@A+DPTR
MOV     R7,A
MOV     R0,#Period
MOV     A,R6
MOV     @R0,A
INC     R0
MOV     A,R7
MOV     @R0,A

CLR     A
MOV     R4,A
MOV     R5,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
SUBB    A,R3
MOV     R7,A
CLR     A
SUBB    A,R2
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
SUBB    A,#00H
MOV     R4,A
MOV     A,R7
CLR     A
MOV     TL1,R7

DEC     R0
MOV     A,@R0
MOV     R6,A
```

```
INC     R0
MOV     A,@R0
MOV     R7,A
CLR     A
MOV     R4,A
MOV     R5,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
SUBB    A,R3
MOV     R7,A
CLR     A
SUBB    A,R2
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
SUBB    A,#00H
MOV     R4,A
MOV     R0,#08H
LCALL   ?C?SLSHR
MOV     TH1,R7
SETB    TR1

MOV     A,i?040
MOV     DPTR,#SOUNDLONG
MOVC    A,@A+DPTR
MOV     R7,A
MOV     R6,#00H
MOV     R0,#Tempo
MOV     A,@R0
MOV     R4,A
INC     R0
MOV     A,@R0
MOV     R5,A
LCALL   ?C?IMUL
CLR     A
MOV     R4,A
MOV     R5,A
MOV     R3,#0E8H
```

```
MOV     R2,#03H
MOV     R1,A
MOV     R0,A
LCALL   ?C?LMUL
MOV     R3,#08H
MOV     R2,#00H
MOV     R1,#00H
MOV     R0,#00H
LCALL   ?C?SLDIV
MOV     A,R4
PUSH    ACC
MOV     A,R5
PUSH    ACC
MOV     A,R6
PUSH    ACC
MOV     A,R7
PUSH    ACC
MOV     R0,#Period
MOV     A,@R0
MOV     R6,A
INC     R0
MOV     A,@R0
MOV     R7,A
CLR     A
MOV     R4,A
MOV     R5,A
MOV     R0,A
MOV     R1,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
POP     ACC
MOV     R7,A
POP     ACC
MOV     R6,A
POP     ACC
MOV     R5,A
POP     ACC
MOV     R4,A
LCALL   ?C?SLDIV
MOV     R0,#SoundLongCount
MOV     A,R6
```

```
MOV       @R0,A
INC       R0
MOV       A,R7
MOV       @R0,A
?C0004:
MOV       R0,#SoundLongCount+01H
MOV       A,@R0
DEC       R0
ORL       A,@R0
JNZ       ?C0004
?C0005:
CLR       TR1
INC       i?040
MOV       A,i?040
XRL       A,#01FH
JZ   $ + 5H
LJMP      ?C0001
?C0006:
RET

END
```

3. timer.a51

```
$NOMOD51

NAME TIMER

$INCLUDE (mtv212m.inc)

?PR?Timer1ISR?TIMER    SEGMENT CODE
 EXTRN     IDATA (Period)
 EXTRN     IDATA (SoundLongCount)
 PUBLIC    Timer1ISR

CSEG AT   0001BH
 LJMP Timer1ISR

 RSEG  ?PR?Timer1ISR?TIMER
 USING     2
Timer1ISR:
 PUSH      ACC
 PUSH      PSW
 MOV       PSW,#010H
 MOV       R0,#SoundLongCount+01H
```

```
        MOV     A,@R0
        DEC     R0
        ORL     A,@R0
        JZ   ?C0001
        INC     R0
        MOV     A,@R0
        DEC     @R0
        JNZ     ?C0001
        DEC     R0
        DEC     @R0
?C0001:
        MOV     R0,#Period
        MOV     A,@R0
        MOV     R6,A
        INC     R0
        MOV     A,@R0
        MOV     R7,A
        CLR     C
        CLR     A
        SUBB    A,R7
        MOV     R7,A
        CLR     A
        SUBB    A,R6
        MOV     R6,A
        CPL     P3_1
        XCH     A,R5
        MOV     A,R7
        XCH     A,R5
        MOV     A,R5
        MOV     TL1,A
        MOV     A,R6
        MOV     TH1,A
        CLR     TF1
        POP     PSW
        POP     ACC
        RETI

        END
```

4-6 音长中断法 while 循环

目的： 以 while 循环来演奏每一个音符，因而可省略计算曲目到底有多少个音

符的麻烦。

4-6-1　软件构建的思维

循环结构除了 for 循环以外，还有 do...while 及 while、switch...case 等结构，当使用 for 循环，需要计算出一首歌中有多少个音符，并将所有音符演奏完毕即可，如果使用 while 循环的结构，就无需计算出音符的个数，而在音长表和音调表中最后以一个结束符号表示即可，但结束符号的数值不可和其他常数的数值重复，则程序才可判断出是否已经演奏到最后一个音符而结束演奏函数。

4-6-2　软件的解决方法

Music( )函数中先判断是否到了最后的音符(音长表和音调表都为 0X00)，如果不是则开始从表中第 0 个地址的数据开始取表，而且在每次演奏必须从第 0 个地址开始作索引，其初始值必须为 0，每取出一个音符的数据：音长和音调，并完成演奏后必须指到下一个地址以便取出下一个音符，因此，在演奏完一个音符必须将索引地址 i 加 1，一直演奏至结束符号为止。由于每一个音符的音长和音调都有数据，所以可将结束符号定义为 0X00，程序写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
    _8,_8 ,_8d,_16,
    _8,_8 ,_4 ,
    _8,_8 ,_8 ,_8 ,
    _8,_8 ,_4 ,

    _8,_8 ,_8d,_16,
    _8,_8 ,_8d,_16,
    _8,_16,_16,_8 ,_16,_16,
```

```
    _8,_8 ,_4 ,
    0x00
};

Word RDATA SOUNDTONE[ ] =
{
    _DO ,_DO ,_RE,_MI,
    _SO ,_SO ,_MI,
    _SO ,_SO ,_LA,_TI,
    _DO1,_DO1,_SO,
    _DO1,_DO1,_LA,_SO,
    _MI ,_LA ,_SO,_MI,
    _DO ,_RE ,_MI,_SO,_LA,_SO,
    _MI ,_RE ,_DO,
    0x00
};

void Music(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG[i]!=0x00)||(SOUNDTONE[i]!=0x00) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

4-7 音长中断法EOF结束符号

目的：将结束符号由宏指令定义，以便望文生义。

4-7-1　软件构建的思维

结束符号在表中也是一个数值，如果是 Byte 类型则数值为 0X00～0Xff，一般取数值的最前端或最末端，即 0X00 或 0Xff，如果是 word 类型则数值为 0X0000～0Xffff，一般是以 0X0000 或 0Xffff 作为结束符号的内容，但如果以英文字来表示结束符号较易望文生义、看懂程序。因此，以_EOF 来表示音符的结束，即结束待号，EOF 是 End of File 的缩写，是文件结束的意思，正好可以引申为音符的结束。

4-7-2　软件的解决方法

在音长表和音调表中最后一个数据内容，以_EOF 来取代 0X00 的数值，程序中判断是否结束符号的指令，必须将 0X00 改成_EOF，重要的是必须定义_EOF，在 music.h 的包括文件要增加此宏指令的定义，其内容一样为 0X00。如果定义 0Xff 则可能与音长表或音调表的内容冲突，因为程序中是判断音长表和音调表都要是结束符号才停止程序执行，虽然定义 0Xff 相冲突的机会不大，但如果定义 0X00 则不会冲突，你会选择哪一个呢？程序写法如下：

```
music.h
#define  _EOF       0x00

music.c
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG[ ] =
{
    _8,_8 ,_8d,_16,
    _8,_8 ,_4 ,
    _8,_8 ,_8 ,_8 ,
    _8,_8 ,_4 ,

    _8,_8 ,_8d,_16,
    _8,_8 ,_8d,_16,
```

```
    _8,_16,_16,_8 ,_16,_16,
    _8,_8 ,_4 ,
    _EOF
};

Word RDATA SOUNDTONE[ ] =
{
    _DO ,_DO ,_RE,_MI,
    _SO ,_SO ,_MI,
    _SO ,_SO ,_LA,_TI,
    _DO1,_DO1,_SO,
    _DO1,_DO1,_LA,_SO,
    _MI ,_LA ,_SO,_MI,
    _DO ,_RE ,_MI,_SO,_LA,_SO,
    _MI ,_RE ,_DO,
    _EOF
};

void Music(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG[i]!=_EOF)||(SOUNDTONE[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

第 5 章　音长与音调定时器

5-1　1ms 定时器 0 的中断

目的：将音长的时间由 Timer0 控制，即利用 Timer0 计时中断来决定音符的声音是否结束，但 Timer0 的中断时间不可以太短，以免 CPU 的负荷过重。

5-1-1　软件构建的思维

每一音符的频率由 Timer1 进行控制，利用中断方法令其输出此音符方波的频率，而决定此方波输出的时间长短则由 Timer0 控制，控制音符频率的 Timer1 大约是数百微秒就中断一次，如果 Timer0 是以 1000μs 作为中断时间，则势必造成 Timer0 的中断次数很频繁，易影响 Timer1 输出的频率不准确，甚至变声，本节就是在说明此现象。正确的方法与说明请参考下一节，也请读者注意此现象：当有两个以上的中断需求时，需考虑是否会互相影响，软件要如何解决？方法如何？下面进行介绍。

5-1-2　软件的设计

1. initial.c：设定定时器 0，TH0、TL0 为 1ms 的内容参数为上数型的计时模式，并使能定时器 0 的中断位，程序设计如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
void PowerOnInit(void)
{
   InitialCpu();
```

```
    InitialCpuIO();
}

/*******************************/
void InitialCpu( void )
{
    IE  = 0;          //disable all interrupt
    PSW = 0;          //bank 0

    IP  = 0x09;       //hi priority:int0,timer1

    TMOD= 0x11;       //set timer1,timer0 mode

    TR0 = 0;          //stop timer0,soundlong
    TR1 = 0;          //stop timer1,soundtone
    IT0 = 1;          //set int0:falling eage trigger

    //CLOCK_1MS=(65536 - 1000)
    TL0 = CLOCK_1MS & 0xff;
    TH0 = CLOCK_1MS >> 8;

    EX0 = 0;          //disable int0 interrupt
    ET1 = 1;          //enable timer1 interrupt
    ET0 = 1;          //enable timer0 interrupt

    EA  = 1;          //enable all interrupt gate
}

/*******************************/
void InitialCpuIO(void)
{
    SPEAKER = 0;
}
```

2. music.c：Timer0 的中断时间是以 1ms 为单位，所以演奏速度及音符的乘积也必须以 ms 为单位，除以 8 是音符在定义时已经多乘了 8，最后再除以 1，表示有多少次 1ms 的中断，程序设计如下：

```
void Music(void)

{
    Byte i;

    Tempo = MODERATO;
```

```
    for(i=0; i<31; i++)
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE[i];
      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      //start timer0=1mS,generate soundlong
      TR0 = 1;
      SoundLongCount=(Tempo*SOUNDLONG[i]/8)/1;
      while ( SoundLongCount != 0 );
      TR0 = 0;      //stop timer0:soundtone

      TR1 = 0;      //stop timer1:soundtone
    }
}
```

3. timer.c：将 SoundLongCount 次数的判断式移至 Timer0 中断函数来处理，并在每 1ms 中断后，需将 1ms 的数值再加载至 TH0、TL0 缓存器内，程序设计如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
//timer0,execute service route
/*****************************/
void  Timer0ISR ( void ) interrupt 1 using 3
{
    //every 1ms,then coumter-1,if coumter=0 then timer ok
    if ( SoundLongCount != 0 )
      SoundLongCount--;

    TL0 = CLOCK_1MS & 0xff;
    TH0 = CLOCK_1MS >> 8;
    TF0 = 0;
}
```

```
/*****************************/
//timer1,execute service route
/*****************************/
void  Timer1ISR ( void ) interrupt 3 using 2
{
   unsigned int temp;

   temp = 65536 - Period;
   SPEAKER = !SPEAKER;

   TL1 = temp & 0xff;
   TH1 = temp >> 8;
   TF1 = 0;
}
```

4. define.h：必须定义1ms的数据内容，由于Crystal是使用12MHz，通过内部硬件电路分频（除12），经过1us则TH0、TL0的内容就会增加1，而Timer0是上数型定时器，所以1ms的内容实际上应为65536－1000，其定义的方法为：

```
#define CLOCK_BASE 1
#define CLOCK_1MS   (65536-1000*CLOCK_BASE)
```

5-2 10ms定时器0的中断

目的：音长的时间控制改由Timer0每10ms就中断的方法。

5-2-1 软件构建的思维

由于10ms的时间大大地大于音调数百微秒的时间，所以不会影响输出频率品质。

例如：曲目中的音符DO，其频率为523Hz，换算为半周期时间为1000000/(523×2)=956us，则10ms/956us=10.46倍，表示连续中断Timer1十次才中断Timer0一次，所以并不会影响其输出频率声音的品质，因此这样的构思可行。

1. initial.c：将10ms的数值存入至TH0、TL0缓存器内，要设定Timer0及Timer1为定时器模式，并使能Timer0及Timer1的中断位，而喇叭的驱动输出为NPN型晶体管，因此将I/O清除为0使其不动作。

2. music.c：计算音长中断的次数，其基底要以10ms为参数，而以ms为单位。

3. timer.c：Timer0的中断函数在每处理完中断程序后，须将10ms的参数内容

再加载至 TH0、TL0 内，而 SoundLongCount 判断式及递减表达式也要由 Timer1 中断函数移至 Timer0 中断函数去处理。

4. define.h：必须定义 10ms 的参数数值，以宏指令#define 即可完成。

5-2-2　软件的解决方法

Timer1 控制音调而以 Timer0 控制音长，以两个定时器来区分音调和音长的控制参数，音调的中断很频繁，而音长的中断时间要大于音调的中断时间，大约大 10 倍左右。但如果太大，例如：100ms 才中断一次，则将舍去个位数及十位数的音长时间，易形成实际音长的时间误差，选择以 10ms 作为 Timer0 音长的中断时间。

1. initial.c：初始化模块以设定定时器模式为主，即将 10ms 存入至 TH0、TL0 内，程序写法如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*******************************/
void PowerOnInit(void)
{
   InitialCpu();

   InitialCpuIO();
}

/*******************************/
void InitialCpu( void )
{
   IE  = 0;          //disable all interrupt
   PSW = 0;          //bank 0

   IP  = 0x09;       //hi priority:int0,timer1

   TMOD= 0x11;       //set timer1,timer0 mode

   TR0 = 0;          //stop timer0,soundlong
```

```
    TR1 = 0;          //stop timer1,soundtone
    IT0 = 1;          //set int0:falling eage trigger

    //CLOCK_10MS=(65536 - 10000)
    TL0 = CLOCK_10MS & 0xff;
    TH0 = CLOCK_10MS >> 8;

    EX0 = 0;          //disable int0 interrupt
    ET1 = 1;          //enable timer1 interrupt
    ET0 = 1;          //enable timer0 interrupt

    EA  = 1;          //enable all interrupt gate
}

/*****************************/
void InitialCpuIO(void)
{
    SPEAKER = 0;
}
```

2. music.c：Music( )函数计算，Tempo 变量为设定演奏的速度，其演奏速度值在 music.h 包括文件中定义，SoundLongCount 个数的表达式，由于是以 ms 为单位，所以乘积的式子就无需乘上 1000L 了，而除数变成 10，因为以 10ms 为中断一次，程序写法如下：

```
void Music(void)
{
    Byte i;


    Tempo = MODERATO;


    for(i=0; i<31; i++)
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE[i];
      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      //start timer0=10mS interrupt,generate soundlong
      TR0 = 1;
      SoundLongCount=(Tempo*SOUNDLONG[i]/8)/10;
      while ( SoundLongCount != 0 );
      TR0 = 0;        //stop timer0:soundtone
```

```
        TR1 = 0;      //stop timer1:soundtone
    }
}
```

3. music.h：为 music.c 模块的包括文件，包含定义演奏速度的数值（一拍有多少 ms）；音符的数值（以三十二分音符为最小单位，又以四分音符为一拍实际上三十二分音符应该是 1/8 拍，所以预先乘以 8，使其数值变为 1），因此，music.c 模块中的 Music( )函数的 SoundLongCount 变量的表达式要再除以 8；半音阶和全音阶的频率换算为半周期的时间，其中 r 为升半音，演奏速度的定义如下：

```
//演奏速度 = ms/拍
#define  LARGHISSIMO      60000/40      //极缓板
#define  LARGO            60000/50      //最缓板
#define  LARGHETTO        60000/60      //甚缓板
#define  GRAVE            60000/63      //稍缓板
#define  LENTO            60000/66      //缓板
#define  ADAGIO           60000/70      //慢板
#define  ADAGIETTO        60000/80      //稍慢板
#define  ANDANTE          60000/92      //行板
#define  ANDANTINO        60000/103     //小行板
#define  MODERATO         60000/114     //中板
#define  ALLEGRETTO       60000/130     //稍快板
#define  ALLEGRO          60000/144     //快板
#define  VIVACE           60000/155     //甚快板
#define  VELOCE           60000/166     //稍急板
#define  PRESTO           60000/177     //急板
#define  PRESTISSIMO      60000/188     //最急板
```

4. timer.c：定时器 0 的中断时间是以 10ms 为单位，程序中 TH0、TL0 要设定为 10ms，高字节存入 TH0（将 10ms 右移 8bit），低字节存入 TL0（将 10ms 和 0Xff 作 AND 运算），Timer1 一样是产生各音符频率的中断函数定时器，欲将 I/O 的输出反相，只要在其前面加一个“!”的运算符即可，程序写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
```

```
/*******************************/
//timer0,execute service route
/*******************************/
void  Timer0ISR ( void ) interrupt 1 using 3
{
   //every 10ms,then coumter-1,if coumter=0 then timer ck
   if ( SoundLongCount != 0 )
     SoundLongCount--;

   TL0 = CLOCK_10MS & 0xff;
   TH0 = CLOCK_10MS >> 8;
   TF0 = 0;
}

/*******************************/
//timer1,execute service route
/*******************************/
void  Timer1ISR ( void ) interrupt 3 using 2
{
   unsigned int temp;

   temp = 65536 - Period;
   SPEAKER = !SPEAKER;

   TL1 = temp & 0xff;
   TH1 = temp >> 8;
   TF1 = 0;
}
```

5. define.h：除了自定义类型的定义，内存类型的字符串转换外，也要将 Timer0 中断时间的参数 10ms 定义为实际的数值，CLOCK_BASE 定义为 1，是因为 CPU 的振荡频率为 12MHz，经过内部分频（除 12 等于 1us）作为定时器的增量时间，即每经过 1us，TH0、TL0 的内容就会增加 1，因而 CLOCK_BASE 要定义为 1，其定义如下：

```
#ifndef   _DEFINE_H
#define   _DEFINE_H

//declare
typedef   bit              Bit;
typedef   bit              Bool;
typedef   unsigned char    Byte;
typedef   unsigned int     Word;
typedef   unsigned long    Long;
```

```
#define   DATA          data
#define   IDATA         idata
#define   PDATA         pdata
#define   XDATA         xdata
#define   RDATA         code

#define   CLOCK_BASE    1
#define   CLOCK_10MS    (65536 - 10000*CLOCK_BASE)

#endif
```

汇编程序如下：

1. initial.a51

```
$NOMOD51

NAME INITIAL

$INCLUDE (mtv212m.inc)

?PR?PowerOnInit?INITIAL          SEGMENT CODE
?PR?InitialCpu?INITIAL           SEGMENT CODE
?PR?InitialCpuIO?INITIAL         SEGMENT CODE
 PUBLIC   InitialCpuIO
 PUBLIC   InitialCpu
 PUBLIC   PowerOnInit

 RSEG  ?PR?PowerOnInit?INITIAL
 USING    0
PowerOnInit:
 LCALL    InitialCpu
 LCALL    InitialCpuIO
 RET

 RSEG  ?PR?InitialCpu?INITIAL
 USING    0
InitialCpu:
 CLR      A
 MOV      IE,A
 MOV      PSW,A
 MOV      IP,#09H
 MOV      TMOD,#011H
 CLR      TR0
```

```
CLR     TR1
SETB    IT0
MOV     TL0,#0F0H
MOV     TH0,#0D8H
CLR     EX0
SETB    ET1
SETB    ET0
SETB    EA
RET

RSEG  ?PR?InitialCpuIO?INITIAL
USING   0
InitialCpuIO:
CLR     P3_1
RET

END
```

2. music.a51

```
$NOMOD51

NAME MUSIC

$INCLUDE (mtv212m.inc)

?PR?Music?MUSIC          SEGMENT CODE
?DT?Music?MUSIC          SEGMENT DATA OVERLAYABLE
?CO?MUSIC                SEGMENT CODE
EXTRN   IDATA (Tempo)
EXTRN   IDATA (Period)
EXTRN   IDATA (SoundLongCount)
EXTRN   CODE (?C?SLSHR)
EXTRN   CODE (?C?IMUL)
EXTRN   CODE (?C?UIDIV)
PUBLIC  SOUNDTONE
PUBLIC  SOUNDLONG
PUBLIC  Music

RSEG  ?DT?Music?MUSIC
?Music?BYTE:
        i?040:   DS   1

RSEG  ?CO?MUSIC
SOUNDLONG:
```

```
 DB   004H,004H,006H,002H,004H
 DB   004H,008H,004H,004H,004H
 DB   004H,004H,004H,008H,004H
 DB   004H,006H,002H,004H,004H
 DB   006H,002H,004H,002H,002H
 DB   004H,002H,002H,004H,004H
 DB   008H

SOUNDTONE:
 DW   003BCH,003BCH,00353H,002F6H
 DW   0027DH,0027DH,002F6H,0027DH
 DW   0027DH,00238H,001FAH,001DDH
 DW   001DDH,0027DH,001DDH,001DDH
 DW   00238H,0027DH,002F6H,00238H
 DW   0027DH,002F6H,003BCH,00353H
 DW   002F6H,0027DH,00238H,0027DH
 DW   002F6H,00353H,003BCH

 RSEG  ?PR?Music?MUSIC
 USING    0
Music:
 MOV      R0,#Tempo
 MOV      @R0,#02H
 INC      R0
 MOV      @R0,#0EH
 CLR      A
 MOV      i?040,A
?C0001:
 MOV      A,i?040
 ADD      A,ACC
 ADD      A,#LOW (SOUNDTONE)
 MOV      DPL,A
 CLR      A
 ADDC     A,#HIGH (SOUNDTONE)
 MOV      DPH,A
 CLR      A
 MOVC     A,@A+DPTR
 MOV      R6,A
 MOV      A,#01H
 MOVC     A,@A+DPTR
 MOV      R7,A
 MOV      R0,#Period
 MOV      A,R6
 MOV      @R0,A
 INC      R0
```

```
MOV     A,R7
MOV     @R0,A

CLR     A
MOV     R4,A
MOV     R5,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
SUBB    A,R3
MOV     R7,A
CLR     A
SUBB    A,R2
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
SUBB    A,#00H
MOV     R4,A
MOV     A,R7
CLR     A
MOV     TL1,R7

DEC     R0
MOV     A,@R0
MOV     R6,A
INC     R0
MOV     A,@R0
MOV     R7,A
CLR     A
MOV     R4,A
MOV     R5,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
SUBB    A,R3
```

```
MOV     R7,A
CLR     A
SUBB    A,R2
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
SUBB    A,#00H
MOV     R4,A
MOV     R0,#08H
LCALL   ?C?SLSHR
MOV     TH1,R7
SETB    TR1
SETB    TR0

MOV     A,i?040
MOV     DPTR,#SOUNDLONG
MOVC    A,@A+DPTR
MOV     R7,A
MOV     R6,#00H
MOV     R0,#Tempo
MOV     A,@R0
MOV     R4,A
INC     R0
MOV     A,@R0
MOV     R5,A
LCALL   ?C?IMUL
MOV     A,R7
MOV     R0,#03H
?C0007:
XCH     A,R6
CLR     C
RRC     A
XCH     A,R6
RRC     A
DJNZ    R0,?C0007
MOV     R7,A
MOV     R4,#00H
MOV     R5,#0AH
LCALL   ?C?UIDIV
MOV     R0,#SoundLongCount
MOV     A,R6
MOV     @R0,A
INC     R0
```

```
MOV       A,R7
MOV       @R0,A
?C0004:
MOV       R0,#SoundLongCount+01H
MOV       A,@R0
DEC       R0
ORL       A,@R0
JNZ       ?C0004
?C0005:
CLR       TR0
CLR       TR1
INC       i?040
MOV       A,i?040
XRL       A,#01FH
JZ    $ + 5H
LJMP      ?C0001
?C0006:
RET

END
```

3. timer.a51

```
$NOMOD51

NAME TIMER

$INCLUDE (mtv212m.inc)

?PR?Timer0ISR?TIMER     SEGMENT CODE
?PR?Timer1ISR?TIMER     SEGMENT CODE
EXTRN     IDATA (Period)
EXTRN     IDATA (SoundLongCount)
PUBLIC    Timer1ISR
PUBLIC    Timer0ISR

CSEG AT   0000BH
LJMP Timer0ISR

RSEG  ?PR?Timer0ISR?TIMER
USING     3
Timer0ISR:
PUSH      ACC
PUSH      PSW
MOV       PSW,#018H
```

```
        MOV     R0,#SoundLongCount+01H
        MOV     A,@R0
        DEC     R0
        ORL     A,@R0
        JZ   ?C0001
        INC     R0
        MOV     A,@R0
        DEC     @R0
        JNZ     ?C0001
        DEC     R0
        DEC     @R0
?C0001:
        MOV     TL0,#0F0H
        MOV     TH0,#0D8H
        CLR     TF0
        POP     PSW
        POP     ACC
        RETI

CSEG AT   0001BH
        LJMP Timer1ISR

        RSEG  ?PR?Timer1ISR?TIMER
        USING    2
Timer1ISR:
        PUSH    ACC
        PUSH    PSW
        MOV     PSW,#010H
        MOV     R0,#Period
        MOV     A,@R0
        MOV     R6,A
        INC     R0
        MOV     A,@R0
        MOV     R7,A
        CLR     C
        CLR     A
        SUBB    A,R7
        MOV     R7,A
        CLR     A
        SUBB    A,R6
        MOV     R6,A

        CPL     P3_1

        XCH     A,R5
```

```
        MOV     A,R7
        XCH     A,R5
        MOV     A,R5
        MOV     TL1,A
        MOV     A,R6
        MOV     TH1,A
        CLR     TF1
        POP     PSW
        POP     ACC
        RETI

        END
```

第6章 移　　调

6-1 基本概念

一般的歌曲，有2/4、3/4、4/4等拍子类型，但不管有几拍，它们都是在C大调演奏的。如果是C大调，则其和钢琴键盘的音名的位置完全一样，即音名C就唱Do，音名D就唱Re、音名E唱Mi，音名F就唱Fa，音名G就唱So，音名A就唱La，音名B就唱Ti等。但是，每一首曲目并不一定要用C大调演奏，有人的声音高亢，如果用C大调来唱就会显得太低沉了，就可以改唱D大调、F大调、G大调等。如果改唱D大调，就要将唱名Do升到音名D的位置，也就是键盘音名D的位置不再是唱Re，而要改唱为Do了，因此，键盘上音名C、D、E、F、G、A、B七个音中的每一个音都可以拿来唱成Do，现在以D大调来进行说明。

音名D的位置本来是要唱Re，但是在D大调下，必须以Do来唱，而音名E就要唱为Re，音名F的位置必须升高半音（即黑键F#音）才能发出正确的Mi，音名G就要改唱为Fa，音名A就要改唱为So，音名B就要改唱为La了，而音名C的位置必须升高半音（即黑键C#音）才能发出正确的Ti，此种改变唱法就称之为移调。

下表为“移调快速记谱法”，可以很快地记住各种调子的音名、唱名和简谱，如下：

唱名	Do	Re	Mi	Fa	So	La	Ti
简谱	1	2	3	4	5	6	7
C调	C	D	E	F	G	A	B
D调	D	E	F#	G	A	B	C#
E调	E	F#	G#	A	B	C	D#
E^b调	E^b	F	G	A^b	B^b	C	D
F调	F	G	A	B^b	C	D	E
G调	G	A	B	C	D	E	F#
A调	A	B	C#	D	E	F#	G#
A^b调	A^b	B^b	C	D^b	E^b	F	G
B调	B	C#	D#	E	F#	G#	A#
B^b调	B^b	C	D	E^b	F	G	A

6-2 D大调

目的：利用查表法将C大调的音符转换成D大调来演奏。

模块化：欲设计一套小系统或大系统的功能软件，首先要将系统中的软件给予分类，或是将大程序区分为多个更小的程序，以便于设计、维护、阅读，日后修改也会变得很方便，此种构想的思维称之为将程序模块化。日后在keil_C编译器内只要新增一个项目文件并包含所有源文件的模块文件(*.C)，即可进行编译和连接操作。可以如下思考，有哪些函数可以独立成一个模块？在相同模块内可以包含哪些函数？在设计软件的同时就要以模块化的规划来进行，以节省日后发展程序的时程，此范例中可以规划的模块如下：

1. global.c：包含所有全局变量、数组变量、外部数据存储器变量(pdata、xdata)等的声明，以便其他模块使用。

2. main.c：程序进入点模块或主程序模块，将CPU送上电源，则CPU将从Main( )函数开始执行，包含初始化的动作和音乐演奏函数的调用。

3. initail.c：初始化动作的模块，举凡CPU内部缓存器设定、CPU I/O的设定或清除，变量的初始化、外部数据存储器的初始化以及外围硬件电路的初始化；可能作复位，使能、除能的动作等，开机后，为使系统正常操作所作的处理动作，都应包含在initial.c初始化模块内。

4. delay.c：以循环作为延迟时间的模块，不管延迟时间是数us、几十us、几ms或几十ms，都须包含在模块内。如果是以计时方式作为延迟，为区分其不同，请不要将计时延迟程序写在此模块内，应该写在timer.c模块内比较理想。

5. timer.c：以中断为请求的模块，将CPU的中断处理程序写在此模块内，包含外部中断、计数器中断、定时器中断等，如果是串行中断，可以独立出来而写在别的模块内，或是其他特殊功能的中断，例如：iic 2bi通信协议的中断，另外独立出来较妥当。

6. music.c：产生每一个音符频率和音长的模块，包含曲目中每一个音符的音长表和音调表，即演奏音乐的模块。

以及各模块相对应的包括文件及其他模块如下：

1. global.h：为模块global.c的对应包括文件，主要将global.c内的变量作外部声明，则在其他模块的程序才能存取此变量。

2. initial.h：为模块initial.c的对应包括文件，主要将initial.c模块内的函数声明为外部函数，这样的话，则其他模块才可以调用initial.c内的函数，以及利用宏指令定义常数等。

3. delay.h：为模块delay.c的对应包括文件，主要将delay.c模块内的函数声明为

外部函数，以便其他模块调用使用。

4. music.h：为模块 music.c 的对应包括文件，除了将 music.c 模块内的函数声明为外部函数外，也定义音乐中的演奏速度、音符的音长、半音阶频率的转换等。

5. define.h：自定义数据类型、内存类型、定时器数值、宏函数、结构体等模块，一切额外的功能定义都可利用此模块来完成。

6. mtv212m.h：CPU 8051 内的特殊功缓存器（SFR）的地址定义，每一个缓存器位的地址定义及 I/O 端口位的地址定义等，当使用不同型号的微处理器，凡是 CPU 内的缓存器地址都可在此模块内定义，以便存取。

6-2-1　查表法

1. 软件构建的思维

利用程序建表的方法，须先将 C 大调的音符转换为 D 大调的音符，但要如何转换呢？首先要知道移调的意义，移调的意思就是将音符的频率平移，例如：C 大调平移至 D 大调，即以 D 大调来演奏，也就是原本弹奏 C 大调的 Do，则须改弹 Re，即平移一度音，也称为 D 大调的 Do，简单的意思为 C 大调的 Re 当作 Do，C 大调的 Mi 当作 Re，C 大调的#Fa 当作 Mi，依此类推。而演奏音乐的函数以 while 循环为结构，所以需要有结束符号，以符号_EOF 来表示，因此在音长表和音调表须增加一个字节为结束符号，现分述如下：

（1）main.c：同一曲目先演奏 C 大调完毕并停二秒后再演奏 D 大调，以分别和感觉 C 大调和 D 大调各音阶的不同，其中 Music1( )函数为 C 大调，Music2( )函数为 D 大调。

（2）music.c：演奏 C 大调后再演奏 D 大调，因此要分别建立 C 大调的音长表、音调表及建立 D 大调的音长表和音调表。而演奏函数也各自独立，这样会比较清晰明了。

（3）music.h：C 大调音乐演奏函数，就必须包含函数的外部声明，及音长表、音调表的程序内存数组外部声明，同样的，D 大调的音乐演奏函数，也需要作外部声明及音长表、音调表的外部声明。

2. 软件的解决方法

D 大调的演奏，程序会改变的模块有 main.c 模块、music.c 模块及 music.h 包括文件，如下的解决方法：

（1）main.c：初始化的动作处理完成后，再进行音乐的演奏，其中 Music1( )函数为 C 大调的演奏，演奏完毕后间隔二秒再演奏 D 大调又间隔二秒，如此一直反

复，程序写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      Music1( );
      DelayX10ms(200);
      Music2( );
      DelayX10ms(200);
   }
}
```

（2）music.c：其中 C 大调的音长表和音调表，如前面章节所述，此处不再列出。

而 D 大调的音符其音长部分并不会改变，只会改变音阶，程序中也以 while 循环为结构。唯一需要注意的是，取表的名称已变为 D 大调的音长表和音调表的名称了，其中 i++的意义为每发出一个音符后，要将音长表和音调表的数组地址往后一个单位，以便取出下一个音符的音长和音调，初始值 i=0，即表示从数组中的第 0 个地址开始取表，即从第一个音符开始发声，程序写法如下：

```
/*****************************/
Byte RDATA SOUNDLONG2[ ] =
{
   _8,_8 ,_8d,_16,
   _8,_8 ,_4 ,
   _8,_8 ,_8 ,_8 ,
   _8,_8 ,_4 ,

   _8,_8 ,_8d,_16,
   _8,_8 ,_8d,_16,
```

```
    _8,_16,_16,_8 ,_16,_16,
    _8,_8 ,_4 ,
    _EOF
};

Word RDATA SOUNDTONE2[ ] =
{
    _RE ,_RE ,_MI ,_FAr ,
    _LA ,_LA ,_FAr,
    _LA ,_LA ,_TI ,_DO1r,
    _RE1,_RE1,_LA ,
    _RE1,_RE1,_TI ,_LA  ,
    _FAr,_TI ,_LA ,_FAr ,
    _RE ,_MI ,_FAr,_LA  ,_TI,_LA,
    _FAr,_MI ,_RE ,
    _EOF
};

void Music2(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG2[i]!=_EOF)||(SOUNDTONE2[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE2[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG2[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

（3）music.h：music.c 模块多了 D 大调的函数，也需要声明为外部，只列出函数的外部声明部分，和前面相同的部分：喇叭的 I/O，演奏速度，音符拍长，音阶频率转换不再重复列出，如下：

```
extern Byte RDATA SOUNDLONG1[ ];
extern Word RDATA SOUNDTONE1[ ];
extern void Music1(void);
extern Byte RDATA SOUNDLONG2[ ];
extern Word RDATA SOUNDTONE2[ ];
extern void Music2(void);
```

汇编程序如下：

1. main.a51

```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN       SEGMENT CODE
 EXTRN    CODE (PowerOnInit)
 EXTRN    CODE (_DelayX10ms)
 EXTRN    CODE (Music1)
 EXTRN    CODE (Music2)
 EXTRN    CODE (?C_STARTUP)
 PUBLIC   Main

 RSEG  ?PR?Main?MAIN
 USING    0
Main:
 LCALL    PowerOnInit
?C0001:
 LCALL    Music1
 MOV      R7,#0C8H
 MOV      R6,#00H
 LCALL    _DelayX10ms
 LCALL    Music2
 MOV      R7,#0C8H
 MOV      R6,#00H
 LCALL    _DelayX10ms
 SJMP     ?C0001
 RET

 END
```

2. music.a51

```
$NOMOD51

NAME MUSIC

$INCLUDE (mtv212m.inc)

?PR?Music1?MUSIC        SEGMENT CODE
?DT?Music1?MUSIC        SEGMENT DATA OVERLAYABLE
?PR?Music2?MUSIC        SEGMENT CODE
?DT?Music2?MUSIC        SEGMENT DATA OVERLAYABLE
?CO?MUSIC               SEGMENT CODE
 EXTRN    IDATA (Tempo)
 EXTRN    IDATA (Period)
 EXTRN    IDATA (SoundLongCount)
 EXTRN    CODE (?C?SLSHR)
 EXTRN    CODE (?C?IMUL)
 EXTRN    CODE (?C?LMUL)
 EXTRN    CODE (?C?SLDIV)
 PUBLIC   SOUNDTONE2
 PUBLIC   SOUNDLONG2
 PUBLIC   SOUNDTONE1
 PUBLIC   SOUNDLONG1
 PUBLIC   Music2
 PUBLIC   Music1

 RSEG  ?DT?Music1?MUSIC
?Music1?BYTE:
        i?040:   DS   1

 RSEG  ?DT?Music2?MUSIC
?Music2?BYTE:
        i?141:   DS   1

 RSEG  ?CO?MUSIC
SOUNDLONG1:
 DB   004H,004H,006H,002H,004H
 DB   004H,008H,004H,004H,004H
 DB   004H,004H,004H,008H,004H
 DB   004H,006H,002H,004H,004H
 DB   006H,002H,004H,002H,002H
 DB   004H,002H,002H,004H,004H
 DB   008H,000H

SOUNDTONE1:
 DW   003BCH,003BCH,00353H,002F6H
```

```
DW   0027DH,0027DH,002F6H,0027DH
DW   0027DH,00238H,001FAH,001DDH
DW   001DDH,0027DH,001DDH,001DDH
DW   00238H,0027DH,002F6H,00238H
DW   0027DH,002F6H,003BCH,00353H
DW   002F6H,0027DH,00238H,0027DH
DW   002F6H,00353H,003BCH,00000H

SOUNDLONG2:
DB   004H,004H,006H,002H,004H
DB   004H,008H,004H,004H,004H
DB   004H,004H,004H,008H,004H
DB   004H,006H,002H,004H,004H
DB   006H,002H,004H,002H,002H
DB   004H,002H,002H,004H,004H
DB   008H,000H

SOUNDTONE2:
DW   00353H,00353H,002F6H,002A3H
DW   00238H,00238H,002A3H,00238H
DW   00238H,001FAH,001C2H,001A9H
DW   001A9H,00238H,001A9H,001A9H
DW   001FAH,00238H,002A3H,001FAH
DW   00238H,002A3H,00353H,002F6H
DW   002A3H,00238H,001FAH,00238H
DW   002A3H,002F6H,00353H,00000H

RSEG  ?PR?Music1?MUSIC
USING    0
Music1:
CLR      A
MOV      i?040,A
MOV      R0,#Tempo
MOV      @R0,#02H
INC      R0
MOV      @R0,#0EH
?C0001:
MOV      A,i?040
MOV      DPTR,#SOUNDLONG1
MOVC     A,@A+DPTR
JNZ      ?C0003
MOV      A,i?040
ADD      A,ACC
ADD      A,#LOW (SOUNDTONE1)
MOV      DPL,A
```

```
CLR     A
ADDC    A,#HIGH (SOUNDTONE1)
MOV     DPH,A
MOV     A,#01H
MOVC    A,@A+DPTR
JNZ     ?C0013
MOVC    A,@A+DPTR
?C0013:
JNZ     $ + 5H
LJMP    ?C0006
?C0003:
MOV     A,i?040
ADD     A,ACC
ADD     A,#LOW (SOUNDTONE1)
MOV     DPL,A
CLR     A
ADDC    A,#HIGH (SOUNDTONE1)
MOV     DPH,A
CLR     A
MOVC    A,@A+DPTR
MOV     R6,A
MOV     A,#01H
MOVC    A,@A+DPTR
MOV     R7,A
MOV     R0,#Period
MOV     A,R6
MOV     @R0,A
INC     R0
MOV     A,R7
MOV     @R0,A

CLR     A
MOV     R4,A
MOV     R5,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
SUBB    A,R3
MOV     R7,A
CLR     A
SUBB    A,R2
```

```
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
SUBB    A,#00H
MOV     R4,A
MOV     A,R7
CLR     A
MOV     TL1,R7

DEC     R0
MOV     A,@R0
MOV     R6,A
INC     R0
MOV     A,@R0
MOV     R7,A
CLR     A
MOV     R4,A
MOV     R5,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
SUBB    A,R3
MOV     R7,A
CLR     A
SUBB    A,R2
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
SUBB    A,#00H
MOV     R4,A
MOV     R0,#08H
LCALL   ?C?SLSHR
MOV     TH1,R7
SETB    TR1

MOV     A,i?040
MOV     DPTR,#SOUNDLONG1
```

```
MOVC    A,@A+DPTR
MOV     R7,A
MOV     R6,#00H
MOV     R0,#Tempo
MOV     A,@R0
MOV     R4,A
INC     R0
MOV     A,@R0
MOV     R5,A
LCALL   ?C?IMUL
CLR     A
MOV     R4,A
MOV     R5,A
MOV     R3,#0E8H
MOV     R2,#03H
MOV     R1,A
MOV     R0,A
LCALL   ?C?LMUL
MOV     R3,#08H
MOV     R2,#00H
MOV     R1,#00H
MOV     R0,#00H
LCALL   ?C?SLDIV
MOV     A,R4
PUSH    ACC
MOV     A,R5
PUSH    ACC
MOV     A,R6
PUSH    ACC
MOV     A,R7
PUSH    ACC
MOV     R0,#Period
MOV     A,@R0
MOV     R6,A
INC     R0
MOV     A,@R0
MOV     R7,A
CLR     A
MOV     R4,A
MOV     R5,A
MOV     R0,A
MOV     R1,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
```

```
XCH     A,R3
MOV     A,R7
XCH     A,R3
POP     ACC
MOV     R7,A
POP     ACC
MOV     R6,A
POP     ACC
MOV     R5,A
POP     ACC
MOV     R4,A
LCALL   ?C?SLDIV
MOV     R0,#SoundLongCount
MOV     A,R6
MOV     @R0,A
INC     R0
MOV     A,R7
MOV     @R0,A
?C0004:
MOV     R0,#SoundLongCount+01H
MOV     A,@R0
DEC     R0
ORL     A,@R0
JNZ     ?C0004
?C0005:
CLR     TR1
INC     i?040
LJMP    ?C0001
?C0006:
RET

RSEG  ?PR?Music2?MUSIC
USING   0
Music2:
CLR     A
MOV     i?141,A
MOV     R0,#Tempo
MOV     @R0,#02H
INC     R0
MOV     @R0,#0EH
?C0007:
MOV     A,i?141
MOV     DPTR,#SOUNDLONG2
MOVC    A,@A+DPTR
JNZ     ?C0009
```

```
MOV     A,i?141
ADD     A,ACC
ADD     A,#LOW (SOUNDTONE2)
MOV     DPL,A
CLR     A
ADDC    A,#HIGH (SOUNDTONE2)
MOV     DPH,A
MOV     A,#01H
MOVC    A,@A+DPTR
JNZ     ?C0014
MOVC    A,@A+DPTR
?C0014:
JNZ     $ + 5H
LJMP    ?C0012
?C0009:
MOV     A,i?141
ADD     A,ACC
ADD     A,#LOW (SOUNDTONE2)
MOV     DPL,A
CLR     A
ADDC    A,#HIGH (SOUNDTONE2)
MOV     DPH,A
CLR     A
MOVC    A,@A+DPTR
MOV     R6,A
MOV     A,#01H
MOVC    A,@A+DPTR
MOV     R7,A
MOV     R0,#Period
MOV     A,R6
MOV     @R0,A
INC     R0
MOV     A,R7
MOV     @R0,A

CLR     A
MOV     R4,A
MOV     R5,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
```

```
SUBB    A,R3
MOV     R7,A
CLR     A
SUBB    A,R2
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
SUBB    A,#00H
MOV     R4,A
MOV     A,R7
CLR     A
MOV     TL1,R7

DEC     R0
MOV     A,@R0
MOV     R6,A
INC     R0
MOV     A,@R0
MOV     R7,A
CLR     A
MOV     R4,A
MOV     R5,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
SUBB    A,R3
MOV     R7,A
CLR     A
SUBB    A,R2
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
SUBB    A,#00H
MOV     R4,A
MOV     R0,#08H
LCALL   ?C?SLSHR
MOV     TH1,R7
```

```
SETB    TR1

MOV     A,i?141
MOV     DPTR,#SOUNDLONG2
MOVC    A,@A+DPTR
MOV     R7,A
MOV     R6,#00H
MOV     R0,#Tempo
MOV     A,@R0
MOV     R4,A
INC     R0
MOV     A,@R0
MOV     R5,A
LCALL   ?C?IMUL
CLR     A
MOV     R4,A
MOV     R5,A
MOV     R3,#0E8H
MOV     R2,#03H
MOV     R1,A
MOV     R0,A
LCALL   ?C?LMUL
MOV     R3,#08H
MOV     R2,#00H
MOV     R1,#00H
MOV     R0,#00H
LCALL   ?C?SLDIV
MOV     A,R4
PUSH    ACC
MOV     A,R5
PUSH    ACC
MOV     A,R6
PUSH    ACC
MOV     A,R7
PUSH    ACC
MOV     R0,#Period
MOV     A,@R0
MOV     R6,A
INC     R0
MOV     A,@R0
MOV     R7,A
CLR     A
MOV     R4,A
MOV     R5,A
```

```
MOV     R0,A
MOV     R1,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
POP     ACC
MOV     R7,A
POP     ACC
MOV     R6,A
POP     ACC
MOV     R5,A
POP     ACC
MOV     R4,A
LCALL   ?C?SLDIV
MOV     R0,#SoundLongCount
MOV     A,R6
MOV     @R0,A
INC     R0
MOV     A,R7
MOV     @R0,A
?C0010:
MOV     R0,#SoundLongCount+01H
MOV     A,@R0
DEC     R0
ORL     A,@R0
JNZ     ?C0010
?C0011:
CLR     TR1
INC     i?141
LJMP    ?C0007
?C0012:
RET

END
```

6-2-2 计算法

目的：将C大调的音符以D大调来演奏。

1. 软件构建的思维

音乐演奏中的程序技巧为取表法，而在 music.h 包括文件中预先定义每个音阶的频率，当取表后即可将数据存入 TH1、TL1 定时器内，而完成音符的频率操作，将 C 大调转换成 D 大调的音符操作，以取表方式较为简单、方便。只是每个音符都需要改变而且容易出错，而计算法主要先算出各音符的相对地址后，再加上移调的地址再取表，即得到移调后实际要演奏的频率了，因而可省去每一音符的换算，只要建立各音符的表序正确，则此方法并不会出错，分述如下：

（1）global.c：增加一个变量 Key，表示移调的数值。例如：C 大调，符号 DO，数值为 0，D 大调，符号为 Re，数值为 2 等，即每移调半音其数值增加 1。

（2）global.h：key 变量的外部声明，以便 Music( )函数的存取。

（3）main.c：此计算法将列出 C 大调的音长表和音调表，借着搜寻每一音符的地址再平移多个地址即能得到转换后正确的音符频率，而演奏函数仍只有 Music( ) 函数。

（4）music.c：必须建立每一音阶的表序，而以半音阶来建立，因为移调后常会出现半音阶的音符，而每个音符都可以在此表序内找到。以低 2 个 8 度音的 Do 至高 3 个 8 度的 Ti 作为寻找音符地址的建表范围，而音乐演奏函数 Music( )必须先找出欲演奏的音符在表序中的地址，将此地址加上移调的数值，作为索引地址并从表序中取出此地址的频率值，即变成为实际演奏的频率了，以 while 循环来比较欲演奏音符的频率和表序中各音阶的频率，当频率不相同时则继续往下一个地址作比较，当频率相同时则立即退出不再比较，则此地址即是音符在表序中的地址了，按此方法，即可将曲目中每个音符都平移成多个音阶的地址，即达到移调的目的。

（5）music.h：要建立一个各音符的表序，即各音符在此表中的相对地址，因此，也要将此表声明为外部数组。

2. 参数的意义说明与使用时机

- Key：移调的数值，以半音阶为单位。
- 数据类型：占用一个字节的无符号数字元类型(unsigned char)。
- 内存类型：变量地址介于 0X00～0Xff 之间，使用间接寻址模式(idata)。
- 写入：在演奏函数中，将移调的数值写入至此变量，以户作为要存取实际频率的地址之一，移调中的数值以半音阶为单位，其 C 大调(DO)的数值为 0，D 大调(RE)的数值 2，降 E 大调(Mlf)的数值为 3，F 大调(FA)的数值为 5，G 大调(SO)的数值为 7，降 A 大调的数值为 8，降 B 大调(Tlf)的数值为-2 等，其数值代表平移几个半音的意思。
- 下标：在表序中找到各音符的地址后也要加上此变量移调的数值，再作为下

标的索引地址，从表序中取出实际要演奏的频率，才有达到平移几个半音的功效。

3. 软件的解决方法

最重要的是要建立一个半音阶的表序，而程序中将原来音符的半周期时间转换为移调后的半周期时间，并需要知道平移几个半音，因此，要定义转调的数值内容，在程序前端设定 Key 变量值，以任意改变要移哪一个调，是为较弹性的设计方式，各模块改变的内容，现分述如下：

（1）global.c：新增一个 key 变量，用以写入移调的数值，在 D 大调中此数值为 2，并利用此模块来声明其数据类型及内存类型，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"

/*****************************/
// Global Data
/*****************************/

//music
Byte      IDATA   Key;
Word      IDATA   Tempo;
Word      IDATA   Period;
Word      IDATA   SoundLongCount;
```

（2）global.h：key 变量要利用此包括文件作外部声明，如下：

```
#ifndef   _GLOBAL_H
#define   _GLOBAL_H

//music
extern Byte    IDATA   Key;
extern Word    IDATA   Tempo;
extern Word    IDATA   Period;
extern Word    IDATA   SoundLongCount;

#endif
```

（3）main.c：利用此计算法只需要一个音乐函数，及建立一个半音阶音符的表序即可，而音乐演奏函数仍以 Music( )函数命名，其写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
void Main( void )
{
    PowerOnInit();

    while( 1 )
    {
       Music( );
       DelayX10ms(200);
    }
}
```

（4）music.c：各音阶的表序以半音阶顺序填入表中，地址从 0 开始计算，数组名为 SOUNDNUMBER 并占用两个字节的数据类型，而演奏音乐程序 Music( )的初始值 i 为代表曲目中的音符，所以要从 0 开始，变量 j 为目前要演奏的这个音符在 SOUNDNUMBER 数组中所存在的地址，所以只要演奏每个音符则 j 变量就要初始化为 0，不能只在声明数据类型初始化为 0 而已，否则，会出错。而演奏之初必须将 Tempo 变量写入演奏速度，key 变量写入移调的数值后，将音符的半周期(Period 变量)去比较 SOUNDNUMBER 表序，而找出其对应的地址（j 变量），取出 j 变量加上 key 变量为表序数组下标的内容，即为移调后半周期时间，并存入 Period 变量内，将每个音符依此方法即可演奏移调的音乐了，程序写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG[ ] =
```

```
{
    _8,_8 ,_8d,_16,
    _8,_8 ,_4 ,
    _8,_8 ,_8 ,_8 ,
    _8,_8 ,_4 ,

    _8,_8 ,_8d,_16,
    _8,_8 ,_8d,_16,
    _8,_16,_16,_8 ,_16,_16,
    _8,_8 ,_4 ,
    _EOF
};

Word RDATA SOUNDTONE[ ] =
{
    _DO ,_DO ,_RE,_MI,
    _SO ,_SO ,_MI,
    _SO ,_SO ,_LA,_TI,
    _DO1,_DO1,_SO,
    _DO1,_DO1,_LA,_SO,
    _MI ,_LA ,_SO,_MI,
    _DO ,_RE ,_MI,_SO,_LA,_SO,
    _MI ,_RE ,_DO,
    _EOF
};

Word RDATA SOUNDNUMBER[ ] =
{
   _2DO ,_2DOr,_2RE ,_2REr,_2MI ,_2FA ,_2FAr,_2SO ,
   _2SOr,_2LA ,_2LAr,_2TI ,_1DO ,_1DOr,_1RE ,_1REr,
   _1MI ,_1FA ,_1FAr,_1SO ,_1SOr,_1LA ,_1LAr,_1TI ,
   _DO ,_DOr ,_RE ,_REr ,_MI ,_FA ,_FAr ,_SO ,
   _SOr ,_LA ,_LAr ,_TI ,_DO1 ,_DO1r,_RE1 ,_RE1r,
   _MI1 ,_FA1 ,_FA1r,_SO1 ,_SO1r,_LA1 ,_LA1r,_TI1 ,
   _DO2 ,_DO2r,_RE2 ,_RE2r,_MI2 ,_FA2 ,_FA2r,_SO2 ,
   _SO2r,_LA2 ,_LA2r,_TI2 ,_DO3 ,_DO3r,_RE3 ,_RE3r,
   _MI3 ,_FA3 ,_FA3r,_SO3 ,_SO3r,_LA3 ,_LA3r,_TI3 ,
   _EOF
};

/*****************************/
void Music(void)
{
    Byte i=0,j;
```

```
    Tempo = MODERATO;
    Key   = RE;      //D 大调

    while ( (SOUNDLONG[i]!=_EOF)||(SOUNDTONE[i]!=_EOF) )
    {
      Period=SOUNDTONE[i];
      j=0;
      while ( (SOUNDNUMBER[j]!=_EOF) )
      {
           if (Period==SOUNDNUMBER[j])
             break;
        j++;
      }
      Period=SOUNDNUMBER[j+Key];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

（5）music.h：在 music.h 包括文件内也要有移调数值的定义，以便写入 Key 变量才能正确取出移调后各音符半周期的时间，其中符号 Do 为 C 大调，Re 为 D 大调，MIf 为降 E 大调，Fa 为 F 大调，So 为 G 大调，LAf 为降 A 大调，TIf 为降 B 大调，其定义如下：

```
//移调
#define  DO        0
#define  RE        2
#define  MIf       3
#define  FA        5
#define  SO        7
#define  LAf       8
#define  TIf       -2
```

汇编程序如下：

1. global.a51

```
$NOMOD51

NAME GLOBAL

$INCLUDE (mtv212m.inc)

?ID?GLOBAL              SEGMENT IDATA
 PUBLIC   SoundLongCount
 PUBLIC   Period
 PUBLIC   Tempo
 PUBLIC   Key

 RSEG  ?ID?GLOBAL
          Key:   DS   1
        Tempo:   DS   2
       Period:   DS   2
 SoundLongCount:   DS   2

 END
```

2. main.a51

```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN           SEGMENT CODE
 EXTRN    CODE (PowerOnInit)
 EXTRN    CODE (_DelayX10ms)
 EXTRN    CODE (Music)
 EXTRN    CODE (?C_STARTUP)
 PUBLIC   Main

 RSEG  ?PR?Main?MAIN
 USING    0
Main:
 LCALL    PowerOnInit
?C0001:
 LCALL    Music
 MOV      R7,#0C8H
 MOV      R6,#00H
 LCALL    _DelayX10ms
 SJMP     ?C0001
```

```
RET

END
```

3. music.a51

```
$NOMOD51

NAME MUSIC

$INCLUDE (mtv212m.inc)

?PR?Music?MUSIC          SEGMENT CODE
?DT?Music?MUSIC          SEGMENT DATA OVERLAYABLE
?CO?MUSIC                SEGMENT CODE
 EXTRN     IDATA (Key)
 EXTRN     IDATA (Tempo)
 EXTRN     IDATA (Period)
 EXTRN     IDATA (SoundLongCount)
 EXTRN     CODE (?C?SLSHR)
 EXTRN     CODE (?C?IMUL)
 EXTRN     CODE (?C?LMUL)
 EXTRN     CODE (?C?SLDIV)
 PUBLIC    SOUNDNUMBER
 PUBLIC    SOUNDTONE
 PUBLIC    SOUNDLONG
 PUBLIC    Music

 RSEG  ?DT?Music?MUSIC
?Music?BYTE:
        i?040:   DS   1

 RSEG  ?CO?MUSIC
SOUNDLONG:
 DB   004H,004H,006H,002H,004H
 DB   004H,008H,004H,004H,004H
 DB   004H,004H,004H,008H,004H
 DB   004H,006H,002H,004H,004H
 DB   006H,002H,004H,002H,002H
 DB   004H,002H,002H,004H,004H
 DB   008H,000H

SOUNDTONE:
 DW   003BCH,003BCH,00353H,002F6H
 DW   0027DH,0027DH,002F6H,0027DH
```

```
 DW   0027DH,00238H,001FAH,001DDH
 DW   001DDH,0027DH,001DDH,001DDH
 DW   00238H,0027DH,002F6H,00238H
 DW   0027DH,002F6H,003BCH,00353H
 DW   002F6H,0027DH,00238H,0027DH
 DW   002F6H,00353H,003BCH,00000H

SOUNDNUMBER:
 DW   00EE8H,00E3DH,00D49H,00C85H
 DW   00BD6H,00B29H,00A8EH,009F7H
 DW   00963H,008E0H,00861H,007E8H
 DW   00774H,0070DH,006A4H,00647H
 DW   005EBH,00598H,00547H,004FBH
 DW   004B4H,00470H,00430H,003F4H
 DW   003BCH,00386H,00353H,00323H
 DW   002F6H,002CCH,002A3H,0027DH
 DW   00259H,00238H,00218H,001FAH
 DW   001DDH,001C2H,001A9H,00191H
 DW   0017BH,00165H,00151H,0013EH
 DW   0012DH,0011CH,0010CH,000FDH
 DW   000EEH,000E1H,000D4H,000C8H
 DW   000BDH,000B2H,000A8H,0009FH
 DW   00096H,0008EH,00086H,0007EH
 DW   00077H,00070H,0006AH,00064H
 DW   0005EH,00059H,00054H,0004FH
 DW   0004BH,00047H,00043H,0003FH
 DW   00000H

 RSEG  ?PR?Music?MUSIC
 USING    0
Music:
 CLR      A
 MOV      i?040,A
 MOV      R0,#Tempo
 MOV      @R0,#02H
 INC      R0
 MOV      @R0,#0EH
 MOV      R0,#Key
 MOV      @R0,#02H
?C0001:
 MOV      A,i?040
 MOV      DPTR,#SOUNDLONG
 MOVC     A,@A+DPTR
 JNZ      ?C0003
 MOV      A,i?040
```

```
ADD     A,ACC
ADD     A,#LOW (SOUNDTONE)
MOV     DPL,A
CLR     A
ADDC    A,#HIGH (SOUNDTONE)
MOV     DPH,A
MOV     A,#01H
MOVC    A,@A+DPTR
JNZ     ?C0010
MOVC    A,@A+DPTR
?C0010:
JNZ     $ + 5H
LJMP    ?C0009
?C0003:
MOV     A,i?040
ADD     A,ACC
ADD     A,#LOW (SOUNDTONE)
MOV     DPL,A
CLR     A
ADDC    A,#HIGH (SOUNDTONE)
MOV     DPH,A
CLR     A
MOVC    A,@A+DPTR
MOV     R7,A
MOV     A,#01H
MOVC    A,@A+DPTR
MOV     R0,#Period
XCH     A,R7
MOV     @R0,A
INC     R0
MOV     A,R7
MOV     @R0,A
CLR     A
MOV     R7,A
?C0004:
MOV     A,R7
ADD     A,ACC
ADD     A,#LOW (SOUNDNUMBER)
MOV     DPL,A
CLR     A
ADDC    A,#HIGH (SOUNDNUMBER)
MOV     DPH,A
CLR     A
MOVC    A,@A+DPTR
MOV     R4,A
```

```
MOV       A,#01H
MOVC      A,@A+DPTR
MOV       R5,A
ORL       A,R4
JZ   ?C0005
MOV       R0,#Period+01H
MOV       A,@R0
XRL       A,R5
JNZ       ?C0011
DEC       R0
MOV       A,@R0
XRL       A,R4
?C0011:
JZ   ?C0005
?C0006:
INC       R7
SJMP      ?C0004
?C0005:
MOV       R0,#Key
MOV       A,@R0
ADD       A,R7
ADD       A,ACC
ADD       A,#LOW (SOUNDNUMBER)
MOV       DPL,A
CLR       A
ADDC      A,#HIGH (SOUNDNUMBER)
MOV       DPH,A
CLR       A
MOVC      A,@A+DPTR
MOV       R6,A
MOV       A,#01H
MOVC      A,@A+DPTR
MOV       R7,A
MOV       R0,#Period
MOV       A,R6
MOV       @R0,A
INC       R0
MOV       A,R7
MOV       @R0,A

CLR       A
MOV       R4,A
MOV       R5,A
XCH       A,R2
MOV       A,R6
```

```
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
SUBB    A,R3
MOV     R7,A
CLR     A
SUBB    A,R2
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
SUBB    A,#00H
MOV     R4,A
MOV     A,R7
CLR     A
MOV     TL1,R7

DEC     R0
MOV     A,@R0
MOV     R6,A
INC     R0
MOV     A,@R0
MOV     R7,A
CLR     A
MOV     R4,A
MOV     R5,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
CLR     C
SUBB    A,R3
MOV     R7,A
CLR     A
SUBB    A,R2
MOV     R6,A
MOV     A,#01H
SUBB    A,#00H
MOV     R5,A
CLR     A
```

```
SUBB    A,#00H
MOV     R4,A
MOV     R0,#08H
LCALL   ?C?SLSHR
MOV     TH1,R7

SETB    TR1

MOV     A,i?040
MOV     DPTR,#SOUNDLONG
MOVC    A,@A+DPTR
MOV     R7,A
MOV     R6,#00H
MOV     R0,#Tempo
MOV     A,@R0
MOV     R4,A
INC     R0
MOV     A,@R0
MOV     R5,A
LCALL   ?C?IMUL
CLR     A
MOV     R4,A
MOV     R5,A
MOV     R3,#0E8H
MOV     R2,#03H
MOV     R1,A
MOV     R0,A
LCALL   ?C?LMUL
MOV     R3,#08H
MOV     R2,#00H
MOV     R1,#00H
MOV     R0,#00H
LCALL   ?C?SLDIV
MOV     A,R4
PUSH    ACC
MOV     A,R5
PUSH    ACC
MOV     A,R6
PUSH    ACC
MOV     A,R7
PUSH    ACC
MOV     R0,#Period
MOV     A,@R0
MOV     R6,A
INC     R0
```

```
MOV     A,@R0
MOV     R7,A
CLR     A
MOV     R4,A
MOV     R5,A
MOV     R0,A
MOV     R1,A
XCH     A,R2
MOV     A,R6
XCH     A,R2
XCH     A,R3
MOV     A,R7
XCH     A,R3
POP     ACC
MOV     R7,A
POP     ACC
MOV     R6,A
POP     ACC
MOV     R5,A
POP     ACC
MOV     R4,A
LCALL   ?C?SLDIV
MOV     R0,#SoundLongCount
MOV     A,R6
MOV     @R0,A
INC     R0
MOV     A,R7
MOV     @R0,A
?C0007:
MOV     R0,#SoundLongCount+01H
MOV     A,@R0
DEC     R0
ORL     A,@R0
JNZ     ?C0007
?C0008:
CLR     TR1
INC     i?040
LJMP    ?C0001
?C0009:
RET

END
```

6-3 降 E 大调...降 B 大调

目的：将 C 大调的曲目改由降 E 大调～降 B 大调来演奏。

软件构建的思维与解决方法：

任何的移调，如果是直接取表，则音调表的内容必须是已经移调后的音阶，需要事先经过人工转换再键入表中，较容易出错，而且浪费时间，因为每一首曲目的每一个音符都需要进行转换，所以，取表法并不是很好的方法。读者是否愿意思考一下，还有什么方法可以不用劳心又劳力呢？想要使软件设计能力更上一层楼，请尽量让脑筋活动活动，如果你能想得到方法，则这种思维就会烙印在脑海里，就会永远属于你了。如果使用计算法：先确定每个音符在音表中的地址后，再加上移调的数值（即平移），跟据此地址取出音阶表序中的实际音阶，经过此程序，则原先的音符已经被转换成移调后的音符了，此种方法不会浪费时间也不会出错，是否比取表法好呢？在上一节 D 大调已经叙述，此处不再赘述，今列出降 E 大调、F 大调、G 大调、降 A 大调和降 B 大调取表法程序和计算法的说明。

6-3-1 降 E 大调

- 查表法：

```
/*****************************/
Byte RDATA SOUNDLONG2[ ] =
{
    _8,_8 ,_8d,_16,
    _8,_8 ,_4 ,
    _8,_8 ,_8 ,_8 ,
    _8,_8 ,_4 ,

    _8,_8 ,_8d,_16,
    _8,_8 ,_8d,_16,
    _8,_16,_16,_8 ,_16,_16,
    _8,_8 ,_4 ,
    _EOF
};

Word RDATA SOUNDTONE2[ ] =
{
    _MIf ,_MIf ,_FA ,_SO ,
    _TIf ,_TIf ,_SO ,
    _TIf ,_TIf ,_DO1,_RE1,
```

```
    _MI1f,_MI1f,_TIf,
    _MI1f,_MI1f,_DO1,_TIf,
    _SO ,_DO1 ,_TIf,_SO ,
    _MIf ,_FA ,_SO ,_TIf,_DO1,_TIf,
    _SO ,_FA ,_MIf,
    _EOF
};

void Music2(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG2[i]!=_EOF)||(SOUNDTONE2[i]!=_EOF) )
    {
       //start timer1,generate soundtone
       Period=SOUNDTONE2[i];

       TL1 = (65536 - Period) & 0xff;
       TH1 = (65536 - Period) >> 8;
       TR1 = 1;

       SoundLongCount=((Tempo*SOUNDLONG2[i]*1000L)/8)/Period;
       while ( SoundLongCount != 0 );

       TR1 = 0;      //stop timer1:soundtone
       i++;
    }
}
```

- 计算法：将 Music( )函数中的 key 变量，直接设定为 MIf 常数，即 key=MIf，其余和 D 大调都一样。
- 查表法：

```
/*****************************/

Byte RDATA SOUNDLONG2[ ] =
{
    _8,_8 ,_8d,_16,
    _8,_8 ,_4 ,
    _8,_8 ,_8 ,_8 ,
    _8,_8 ,_4 ,

    _8,_8 ,_8d,_16,
    _8,_8 ,_8d,_16,
```

```
    _8,_16,_16,_8,_16,_16,
    _8,_8 ,_4 ,
    _EOF
};

Word RDATA SOUNDTONE2[ ] =
{
    _FA ,_FA ,_SO ,_LA ,
    _DO1,_DO1,_LA ,
    _DO1,_DO1,_RE1,_MI1,
    _FA1,_FA1,_DO1,
    _FA1,_FA1,_RE1,_DO1,
    _LA ,_RE1,_DO1,_LA ,
    _FA ,_SO ,_LA ,_DO1,_RE1,_DO1,
    _LA ,_SO ,_FA ,
    _EOF
};

void Music2(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG2[i]!=_EOF)||(SOUNDTONE2[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE2[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG2[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

- 计算法：将 Music( )函数中的 key 变量，直接设定为 FA 常数，即 key=FA，其余和 D 大调一样。

6-3-2 G大调

- 查表法:

```
/*****************************/

Byte RDATA SOUNDLONG2[ ] =
{
    _8,_8 ,_8d,_16,
    _8,_8 ,_4 ,
    _8,_8 ,_8 ,_8 ,
    _8,_8 ,_4 ,

    _8,_8 ,_8d,_16,
    _8,_8 ,_8d,_16,
    _8,_16,_16,_8 ,_16,_16,
    _8,_8 ,_4 ,
    _EOF
};

Word RDATA SOUNDTONE2[ ] =
{
    _SO ,_SO ,_LA ,_TI ,
    _RE1,_RE1,_TI ,
    _RE1,_RE1,_MI1,_FA1,
    _SO1,_SO1,_RE1,
    _SO1,_SO1,_MI1,_RE1,
    _TI ,_MI1,_RE1,_TI ,
    _SO ,_LA ,_TI ,_RE1,_MI1,_RE1,
    _TI ,_LA ,_SO ,
    _EOF
};

void Music2(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG2[i]!=_EOF)||(SOUNDTONE2[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE2[i];

      TL1 = (65536 - Period) & 0xff;
```

```
        TH1 = (65536 - Period) >> 8;
        TR1 = 1;

        SoundLongCount=((Tempo*SOUNDLONG2[i]*1000L)/8)/Period;
        while ( SoundLongCount != 0 );

        TR1 = 0;      //stop timer1:soundtone
        i++;
    }
}
```

- 计算法：将 Music( )函数中的 key 变量，直接设定为 SO 常数，即 key=SO，其余和 D 大调都一样。

6-3-3 降 A 大调

- 查表法：

```
/*****************************/
Byte RDATA SOUNDLONG2[ ] =
{
    _8,_8 ,_8d,_16,
    _8,_8 ,_4 ,
    _8,_8 ,_8 ,_8 ,
    _8,_8 ,_4 ,

    _8,_8 ,_8d,_16,
    _8,_8 ,_8d,_16,
    _8,_16,_16,_8 ,_16,_16,
    _8,_8 ,_4 ,
    _EOF
};

Word RDATA SOUNDTONE2[ ] =
{
    _LAf ,_LAf ,_TIf ,_DO1,
    _MI1f,_MI1f,_DO1 ,
    _MI1f,_MI1f,_FA1 ,_SO1,
    _LA1f,_LA1f,_MI1f,
    _LA1f,_LA1f,_FA1 ,_MI1f,
    _DO1 ,_FA1 ,_MI1f,_DO1 ,
    _LAf ,_TIf ,_DO1 ,_MI1f,_FA1,_MI1f,
    _DO1 ,_TIf ,_LAf ,
    _EOF
};
```

```
void Music2(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG2[i]!=_EOF)||(SOUNDTONE2[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE2[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG2[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

- 计算法：将 Music( )函数中的 key 变量，直接设定为 LAf 常数，即 key=LAf，其余和 D 大调都一样。

6-3-4 降 B 大调

- 查表法：

```
/*****************************/

Byte RDATA SOUNDLONG2[ ] =
{
    _8,_8 ,_8d,_16,
    _8,_8 ,_4 ,
    _8,_8 ,_8 ,_8 ,
    _8,_8 ,_4 ,

    _8,_8 ,_8d,_16,
    _8,_8 ,_8d,_16,
    _8,_16,_16,_8 ,_16,_16,
    _8,_8 ,_4 ,
    _EOF
```

```
};

Word RDATA SOUNDTONE2[ ] =
{
    _1TIf,_1TIf,_DO ,_RE,
    _FA ,_FA ,_RE ,
    _FA ,_FA ,_SO ,_LA,
    _TIf ,_TIf ,_FA ,
    _TIf ,_TIf ,_SO ,_FA,
    _RE ,_SO ,_FA ,_RE,
    _1TIf,_DO ,_RE ,_FA,_SO,_FA,
    _RE ,_DO ,_1TIf,
    _EOF
};

void Music2(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG2[i]!=_EOF)||(SOUNDTONE2[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE2[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG2[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

- 计算法：将Music( )函数中的key变量，直接设定为key=TIf，实际上并不会演奏降B大调，而会变成降E大调，原因是：

①在music.h包括文件中TIf等于－2，即负数，不想定义数值为10，是因为调子太高了，所以往下降8度音，以C大调(Do)数值为0，所以降B大调(TIf)其数值应为－2，即以DO为起始往下二个半度音。

②在global.c模块中，声明Key变量为Byte类型，即占用一个字节的无符号字

符类型，即正整数。

③因此，在 Music( )函数设定 Key=TIf，实际上 Key 变量并不是等于－2 而会变成+3，因为 Key 变量的数据类型在 global.c 模块已被声明为正整数的缘故。

所以，只要将 global.c 模块中的 Key 变量声明为：

```
Byte IDATA key;改成 char IDATA key;
```

而 global.h 包括文件也要修正如下：

```
e.xtern char IDATA key;
```

经过以上修改，则才能正确演奏降 B 大调。

6-4 C 大调...降 B 大调顺序演奏

目的：从 C 大调开始顺序演奏至降 B 大调。

6-4-1 软件构建的思维

将一首曲目从 C 大调顺序演奏，即先演奏完 C 大调后，再继续演奏 D 大调、降 E 大调、F 大调、G 大调、降 A 大调最后再演奏降 B 大调，如此不断地反复；由于每一个调子其调子数值都不一样，也没有规则可循，因而，每当转调时将调子数值以取表方式存入至 key 变量内，即当演奏 C 大调，则 key 变量的内容要等于 0，演奏 D 大调则 key 变量的内容要等于 2，演奏 ^{b}E 大调，key=3，演奏 F 大调，key=5，演奏 G 大调，key=7，演奏 ^{b}A 大调，key=8，演奏 ^{b}B 大调，key=-2，而 key 的数据类型必须为有符号数的字符类型，其范围为－128～＋127 之间，并在 main( )函数中以 for 循环分别来演奏 C 大调～^{b}B 大调，分述如下：

1. global.c：Tempo、Period、SoundLongCount 变量以 unsigned int 类型声明，并占用二个字节，因为这些变量的内容都会超过 0Xff，而 key 变量必须以 char 类型进行声明，否则，当演奏至 ^{b}B 大调会变成 ^{b}E 大调。

2. global.h：将变量都声明成外部，以便各模块取用方便，如果不作外部声明，则有取用此变量的模块，在编译时将出现“undefined identifier”，即未定义符号的信息。

3. main.c：除了需作初始化动作外，以 for 循环来处理 C 大调～^{b}B 大调的调子数值，即 key 变量的内容在此 main( )函数设定，则 Music( )函数就无需设定 key 变量了。

4. music.c：调子数值的建表写在此模块内，数组名为 KEYTABLE 其内容须事先定义好，而以符号 DO、RE、MIf、FA、SO、LAf、TIf 来取代数值 0、2、3、5、

7、8、—2较易阅读。

5. music.h：需要增加一个数组调子数值表的外部声明，否则无法存取数组的内容

6-4-2 参数的意义说明与使用时机

- Key：调子的数值，以平移取用转换后的音符。
- 数据类型：占用一个字节的有符号字符类型(char)。
- 内存类型：变量地址介于0X00～0Xff之间，使用间接寻址模式(idata)。
- 写入：在main( )函数内，利用for循环将调子的数值以取表方式(KEYTABLE表)写入，即当i=0，则写入C大调的数值，i=1则写入D大调的数值，i=2则写入^{b}E大调的数值等。
- 下标：从音符表序中再平移调子数值，则此地址的音阶即变成实际要演奏的音阶了，利用key变量作为音符表序的下标值，可以很容易地达到功效。

6-4-3 软件的解决方法

在Music( )函数里其程序结构只要改变key变量的内容，则演奏出来的音乐是其相对应的调子，因此，要移调演奏就会变得很简单，只要再思索着如何令key变量随着不同调子其key变量内容也随之作修正就可以了。当然取表法是最方便最简单的，取表法必须先知道表上的数据才可以，所以，在main( )函数以for循环来顺序取调子数值并存入key内，则整体软件结构变得很明了清晰，现分述如下：

1. global.c：将key变量声明成char字符类型，其取值范围为－128～+127，因而，当演奏至^{b}B大调，则key=—2也不会出错，global.c模块如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"

/*******************************/
// Global Data
/*******************************/

//music
char      IDATA   Key;
Word      IDATA   Tempo;
```

```
Word      IDATA   Period;
Word      IDATA   SoundLongCount;
```

2. global.h：key 变量及其他变量也需要声明成外部变量，其方法如下：

```
#ifndef   _GLOBAL_H
#define   _GLOBAL_H

//music
extern char   IDATA   Key;
extern Word   IDATA   Tempo;
extern Word   IDATA   Period;
extern Word   IDATA   SoundLongCount;

#endif
```

3. main.c：由 C 大调至 ^{b}B 大调总共有 7 个调子，所以 for 循环局域变量 i=0～6，并以 i 变量作为调子数值表 KEYTABLE 的数组下标，并取出数值存入 key 变量内，以为 Music( )函数使用，程序写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
void Main( void )
{
   Byte i;

   PowerOnInit();

   while( 1 )
   {
      for(i=0; i<7; i++)
      {
         Key=KEYTABLE[i];
         Music( );
         DelayX10ms(200);
      }
   }
}
```

4. music.c：Music( )函数中的 key 变量设定改由 Main( )函数中写入，其余程序部分和音长表、音调表、音阶表都和前面章节一样，唯一不同的是需要一个内含调子数值的建表，以便 Main( )函数的取用而写入至 key 变量内，调子数值表如下：

```
//TIf=-2
char RDATA KEYTABLE[ ] =
{
    DO,RE,MIf,FA,SO,LAf,TIf
};
```

5. music.h：由于在 music.c 的模块新增一个名称为 KEYTABLE 的程序内存数组，所以在其对应的 music.h 包括文件中也要作数组的外部声明，如下：

```
extern char RDATA KEYTABLE[];
```

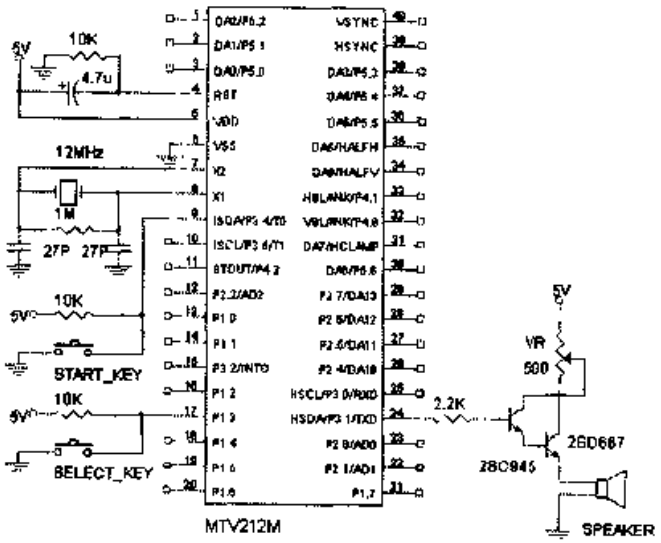
其中，

- **extern**：为外部声明的指令。
- **char**：为此数组的内容其数据类型为有符号数的字符，取值范围为－128～+127 之间
- **RDATA**：在 define.h 包括文件中定义 RDATA，用 code 来取代，表示此数组的内存类型为程序内存。
- **KEYTABLE[]**：表示为数组字符，KEYTABLE 为其数组名。

6-5 Tact 开关循环调整

目的：以一个按键作循环调整的移调功能。

电路图：



原理：以 SELECT_KEY 来作为移调键，平常 I/O P1.3 接一个提升电阻 10kΩ至 +5V，所以其正常电位为“1”电位，当按下 SELECT_KEY，则 P1.3 的输入电位转变为“0”电位，因而，只要侦测 I/O P1.3 的电位是“0”电位，表示按下 SELECT_KEY，则令喇只发出一声作为提示，并将计数变量 KeyTone 累加一，当移调调整完成后，按下 START_KEY 则可开始演奏移调后的音乐，由于平常 I/O P3.4 接 10kΩ的提升电阻至+5V，其输入电位为“1”电位，当按下 START_KEY，P3.4 的电位变成“0”电位，则可根据此电位的转变而侦测出 START_KEY 是否按下，当 P3.4=“1”，表示 START_KEY 未按下，P3.4=“0”表示 START_KEY 已经按下，即可以开始演奏音乐了，而驱动喇叭使用达灵顿晶体组成，以便有足够的电流来驱动 5W/8Ω的喇叭。当 P31 =“1”电位则晶体导通，喇叭流过电流，当 P3.1=“0”电位则晶体截止喇叭未流入电流，如此当 I/O P3.1 输出“1”、“0”，则喇叭就会发出方波频率的声音。

模块化：借着按键的动作来移调，另一个按键则作为起始键，当选择的调子完成后，按下起始键，则音乐开始演奏，因此可增加一个为处理按键动作的模块，称为 input.c。而每按键一次，则调子就升高一阶，即从 C 大调～^{b}B 大调循环调整，则喇叭响叫一声，以作为提示表示已经移调一次了。此喇叭响叫的函数也可独立为一个模块，称为 beep.c，而 input.c 和 beep.c 的模块，其对应的包括文件为 input.h 和 beep.h，则总共可规划的模块如下：

1. global.c：全局变量声明的模块。

2. main.c：主程序模块，其中有为程序的进入点函数，即 Main( )函数，为处理起始键和“移调键”动作。

3. initial.c：程序初始化模块，用以初始化 CPU 缓存器，I/O 以及清除程序中的全局变量。

4. delay.c：所有关于以循环为主结构的延迟函数的模块，一般为延迟 50us、1ms 和 10ms。

5. beep.c：当按下键时，喇叭发出声响，当作提示的模块。

6. initpu.c：处理按键的模块，将按键动作存入变量内，以表示不同的按键输入。

7. timer.c：定时器 1 产生 I/O 引脚的方波输出，用以驱动喇叭的模块。

8. music.c：取得各音符频率藉以演奏音乐的模块。

以及下列的包括文件：

1. global.h：全局变量的外部声明。

2. initial.h：初始化模块函数的外部声明。

3. delay.h：延迟模块函数的外部声明。

4. beep.h：喇叭声响模块函数的外部声明。

5. input.h：按键输入模块函数的外部声明，定义移调键和起始键的 I/O 引脚以及当按下键的常数定义。

6. music.h：音乐演奏模块函数、数组的外部声明，喇叭输出的 I/O 引脚，演奏速度、音符的拍长数值、移调的调子数值、半音阶的半周期时间。

7. define.h：自定义数据类型及内存类型的替换。

8. mtv212m.h：8051 CPU SFR 地址定义，位的地址定义及 I/O 地址定义，以便程序存取。

6-5-1 参数的意义说明与使用时机

1. FgSelectKey：避免持续按下移调键就一直动作的标志变量

- 数据类型：占用一个位的数据类型(bit)。
- 内存类型：标志变量地址介于 0X20～0X2f 的其中一个位，使用位寻址模式。
- 设定：当按下 SELECT_KEY（移调键），则设定此标志变量为 1，以防止当一直按下 SELECT_KEY 而不放开也只作按键一次的动作。
- 清除：当 I/O P1.3=“1”电位即 SELECT_KEY 未按下，则必须清除此变量，将按下 SELECT_KEY 的设定可以重新归零，以便下一次按键的动作。
- 判断：处理按键动作之前，需先判断是否可处理，当标志为“0”，表示已经释放键了，则可以处理按键功能；当标志为“1”，表示持续按键中，则不可以再次执行按键功能。

2. KeyData：按键的数据值

- 数据类型：占用一个字节的无符号字符类型(unsigned char)。
- 内存类型：变量地址介于 0X00～0Xff 之间，使用间接寻址模式(idata)。
- 写入：当按下“移调键”，则写入移调键数值，当按下起始键，则写入起始键的数值，否则，写入“未按下键”的数值，一般为 0X00，三者的数值要不相同。
- 判断：在主程序中判断变量值是否等于起始键值，以作为起始键的按键输入的依据，并判断变量值是否等于“移调键”值，当相同时，表示按下“移调键”了。

3. KeyTone：音调的数值

C 大调的数值为 0，D 大调的数值为 1，^{b}E 大调的数值为 2、F 大调的数值为 3、G 大调的数值为 4、^{b}A 大调的数值为 5、^{b}B 大调的数值为 6。

- 数据类型：占用一个字节的无符号字符类型(unsigned char)。
- 内存类型：变量地址介于 0X00～0Xff 之间，使用间接寻址模式(idata)。
- 清除：

  （1）开机后在 initial.c 模块下先清除，表示以 C 大调为内定调。

  （2）以 SELECT_KEY 作为移调的调整，当移调至 ^{b}B 大调而又再按 SELECT_KEY 一次则需要从 C 大调开始，即清除此变量内容。
- 写入：每按一次 SELECT_KEY，则必须将此变量累加一，以指到下一个调子。
- 判断：当调整至 bB 大调，其数值为 6，又再调整一次，则变量内容变为 7，实际上必须恢复为 C 大调（数值为 0），因此，要判断变量内容，当其大于 6 时则归零。

6-5-2　软件构建的思维与解决方法

以 KeyTone 变量作为调子数值表(KEYTABLE)的数组下标，来取代前面章节 for 循环的 i 变量，当按下 SELECT_KEY，则 KeyTone 变量就会存入不同的数值以表示不同的调子，而增加一个 input.c 模块来侦测按键的输入，当按下键，则要立即处理对应的功能："移调键"，则将 KeyTone 变量累增 1，"起始键"，则根据 KeyTone 取出调子数值表的音阶平移地址的数据，并存入 key 变量内，现分述如下：

1. global.c：新增 FgSelectKey 标志变量，KeyData 及 KeyTone 变量，必须分别声明其数据类型和内存类型。

2. global.h：新增的变量都要作外部声明，此 global.h 包括文件即是作此功能。

3. main.c：初始化的动作执行一次后，则利用 while(1)的指令作永远的反复循环，形成不断的侦测，是否按下 START_KEY 或 SELECT_KEY，也就是按下起始键就根据 KeyTone 变量取出调子数值并存入 Key 变量内，在 Music( )函数又依据 key 变量的内容作平移地址再取出实际的音阶频率而存入 TH1、TL1 缓存器内；而按下"移调键"除了响叫一声作为提示，必将 KeyTone 累加 1，并判断是否大于 ^{b}B 大调的数值（^{b}B 大调的数值等于 6），如是，则须从 C 大调开始调整（C 大调的数值等于 0），其程序内容如下：

```
/*****************************/
/* include files             */
```

```
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      //detect start key
      InputStart( );
      if ( KeyData == START_KEY )
      {
         Key=KEYTABLE[KeyTone];
         Music( );
         DelayX10ms(200);
      }

      //sel KeyTone
      InputKey( );
      if ( KeyData == SELECT_KEY )
      {
         if (FgSelectKey==0)
         {
            FgSelectKey = 1;
            Beep(1,17,10);

            KeyTone++;
            if ( KeyTone>6 ) //DO,RE,MIf,FA,SO,LAf,TIf
               KeyTone=0;
         }
      }
   }
}
```

4. initial.c：在此模块下新增一个叫做 InitialVariable( )的函数，即初始化变量的意思，由于 KeyTone 变量关系着调子，此变量内容错误则演奏调子势必不对，当不

按下“移调键”时，则以C大调来演奏，则KeyTone要等于0，当内定值调子是D大调时，则在InitialVariable( )函数内的KeyTone要等于1，即修改此初始化的KeyTone，就成为内定调子的依据了，程序如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"

/*******************************/
void PowerOnInit(void)
{
    InitialCpu();

    InitialCpuIO();

    InitialVariable();
}

/*******************************/
void InitialCpu( void )
{
    IE  = 0;          //disable all interrupt
    PSW = 0;          //bank 0

    IP  = 0x0b;//hi priority:int0,timer0,timer1

    TMOD= 0x11;       //set timer1,timer0 mode

    TR0 = 0;      //stop timer0
    TR1 = 0;          //stop timer1
    IT0 = 1;    //set int0:falling eage trigger

    EX0 = 0;          //disable int0 interrupt
    ET1 = 1;          //enable timer1 interrupt
    ET0 = 0;          //disable timer0 interrupt
```

```
    EA  = 1;          //enable all interrupt gate
}

/*****************************/
void InitialCpuIO(void)
{
    SPEAKER = 0;
}

/*****************************/
void InitialVariable(void)
{
    KeyTone=0;
}
```

5. initial.h：必须将InitialVariable( )函数声明为外部函数，如下：

```
#ifndef   _INITIAL_H
#define   _INITIAL_H

extern void PowerOnInit(void);
extern void InitialCpu(void);
extern void InitialCpuIO(void);
extern void InitialVariable(void);

#endif
```

6. beep.c：按下“移调键”，以“哔”声作为输入的提示，“哔”声的方法为：将SPEAKER的I/O产生数个方波，其中，SPEAKER定义为I/O P3.1，即控制喇叭响叫的I/O；spferg为方波的半周期时间，即声音的频率；soundlong，用以产生方波的个数，为发出声音的长短；count为发出几个“哔”声的自变量，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"

/*****************************/
```

```
void Beep(Word count,Byte soundlong,Byte tone)
{
   Word  i,j,k,spfreq;

   spfreq  = (1000/tone)/2; //Khz unit

   for(i=0; i<ccunt; i++)
   {
      for(j=0; j<soundlong; j++)
      {
            SPEAKER = 1;
         for(k=0; k<spfreq; k++)
            ;
         SPEAKER = 0;
         for(k=0; k<spfreq; k++)
            ;
      }
      DelayX10ms(12);
   }
}
```

7. beep.h：将 beep.c 模块下的函数声明为外部，如下：

```
#ifndef   _BEEP_H
#define   _BEEP_H

extern void Beep(Word count,Byte soundlong,Byte tone);

#endif
```

8. input.c：将“移调键”和起始键的侦测函数写在此模块内，以便给 Main( )函数调用，当“移调键”的 I/O 为“0”电位，表示按下此键则 KeyData 变量内容写入 SELECT_KEY，否则，写入 0X00，且清除此按键标志，FgSelectKey=0；当起始键的 I/O 为“0”电位，也表示按下了起始键，则 KeyData 变量写入 START_KEY 的数值，否则，也清除为 0X00，其程序写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
```

```
#include  "input.h"
#include  "beep.h"

/*****************************/
void InputKey(void)
{
   if ( SELECT_IO==0 )
     KeyData=SELECT_KEY;
   else
   {
     KeyData=NO_KEY;
     FgSelectKey = 0;
   }
}

/*****************************/
void InputStart(void)
{
   if ( START_IO==0 )
     KeyData=START_KEY;
   else
     KeyData=NO_KEY;
}
```

9. input.h：将 input.c 模块的函数声明成外部，并定义按键的 I/O 和按键数值，如下：

```
#ifndef   _INPUT_H
#define   _INPUT_H

#define  START_IO       P3_4
#define  SELECT_IO      P1_3

#define  NO_KEY              0
#define  START_KEY      1
#define  SELECT_KEY     2

extern void InputKey(void);
extern void InputStart(void);

#endif
```

汇编程序如下：

1. global.a51

```
$NOMOD51

NAME GLOBAL

$INCLUDE (mtv212m.inc)

?BI?GLOBAL          SEGMENT BIT
?ID?GLOBAL          SEGMENT IDATA
 PUBLIC   SoundLongCount
 PUBLIC   Period
 PUBLIC   Tempo
 PUBLIC   Key
 PUBLIC   KeyTone
 PUBLIC   KeyData
 PUBLIC   FgSelectKey

 RSEG  ?BI?GLOBAL
    FgSelectKey:   DBIT   1

 RSEG  ?ID?GLOBAL
       KeyData:   DS   1
       KeyTone:   DS   1
          Key:   DS   1
        Tempo:   DS   2
       Period:   DS   2
 SoundLongCount:   DS   2

 END
```

2. main.a51

```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN       SEGMENT CODE
EXTRN     BIT (FgSelectKey)
EXTRN     IDATA (KeyData)
EXTRN     IDATA (KeyTone)
EXTRN     IDATA (Key)
EXTRN     CODE (PowerOnInit)
EXTRN     CODE (_DelayX10ms)
EXTRN     CODE (KEYTABLE)
```

```
EXTRN     CODE (Music)
EXTRN     CODE (InputKey)
EXTRN     CODE (InputStart)
EXTRN     CODE (_Beep)
EXTRN     CODE (?C_STARTUP)
PUBLIC    Main

RSEG  ?PR?Main?MAIN
USING     0
Main:
LCALL     PowerOnInit
?C0001:
LCALL     InputStart
MOV       R0,#KeyData
MOV       A,@R0
CJNE      A,#01H,?C0003
MOV       R0,#KeyTone
MOV       A,@R0
MOV       DPTR,#KEYTABLE
MOVC      A,@A+DPTR
MOV       R0,#Key
MOV       @R0,A
LCALL     Music
MOV       R7,#0C8H
MOV       R6,#00H
LCALL     _DelayX10ms
?C0003:
LCALL     InputKey
MOV       R0,#KeyData
MOV       A,@R0
CJNE      A,#02H,?C0001
JB   FgSelectKey,?C0001
SETB      FgSelectKey
MOV       R7,#01H
MOV       R6,#00H
MOV       R5,#011H
MOV       R3,#0AH
LCALL     _Beep
MOV       R0,#KeyTone
INC       @R0
MOV       A,@R0
SETB      C
SUBB      A,#06H
JC   ?C0001
CLR       A
```

```
MOV       @R0,A
SJMP      ?C0001
RET

END
```

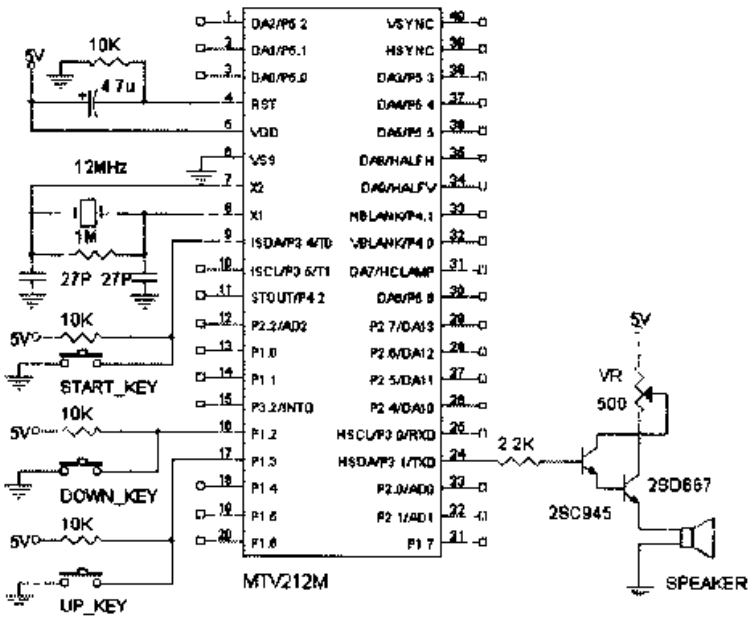
3. input.a51

```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)

 RSEG  ?PR?InputKey?INPUT
 USING    0
InputKey:
 JB   P1_3,?C0001
 MOV       R0,#KeyData
 MOV       @R0,#02H
 RET
?C0001:
 CLR       A
 MOV       R0,#KeyData
 MOV       @R0,A
 CLR       FgSelectKey
?C0003:
 RET

 RSEG  ?PR?InputStart?INPUT
 USING    0
InputStart:
 JB   P3_4,?C0004
 MOV       R0,#KeyData
 MOV       @R0,#01H
 RET
?C0004:
 CLR       A
 MOV       R0,#KeyData
 MOV       @R0,A
?C0006:
 RET

 END
```

6-6 Tact 开关升降调

目的： 利用两个按键来作升降调的调整。

电路图：



原理： START_KEY（起始键）、DOWN_KEY（降调）、UP_KEY（升调）都是由 CPU 的 I/O 作输入，由于每一支 I/O 都以 10KΩ提升至+5V，所以其平常的电位为“1”电位，每一支 I/O 也都串接一个开关至地，当开关按下去，则 I/O 的电位会被接地，所以其输入的电位为“0”电位，依此特性来判断 I/O 是否为“0”电位，就知道是否按下键了。而将升调和降调分别以不同的开关作侦测，当 DOWN_KEY 被按下表示往下调整一个调，当 UP_KEY 被按下表示往上调整一个调子，而可明显区分升降调的操作。

6-6-1 软件构建的思维

按键侦测程序也要同时侦测升调键或降调键，以 if...else 的条件判断指令来顺序判断各 I/O 是否为“0”电位，并将适当的值写入 KeyData 变量中。其升调键数值和降调键数值要各自独立才能区分出来，当按下“升调键”时，则 KeyTone 为累加 1，而大于 6，则须清除为 0；当按下“降调键”时，则 KeyTone 为递减 1，而当等于 0，再按下“降调键”，须成为 6，现分述如下：

1. globa.c：FgSelectKey 变量和 KeyData 变量的使用时机改变了，处理 UP_KEY 和 DOWN_KEY 的持续按键情形和加载至 KeyDate 的数值不同，才能区分到底是按了 UP_KEY 还是 DOWN_KEY。

2. main.c：处理起始键输入的动作程序和前面章节一样，都是跟据 KeyTone 来取得调子数值(KEYTABLE)后存入 Key 变量内，并开始演奏，并依据按下升降键来改变 KeyTone 的数值，也就是按键动作只是要处理 KeyTone 的变化，最后来影响实际演奏的调子而已。

3. input.c：按键侦测程序要分别判断是否按下了“升调键”？是否按下了“降调键”？如果都没有才将键值清除为 0，以及将按键标志清除为 0。

4. input.h：由两个按键分别控制升调和降调，因此，要个别定义 I/O 地址和不同的键值。

6-6-2　参数的意义说明和使用时机

1. FgSelectKey：按键标志，不管是升调键或降调键。

- 数据类型：占用一个位的数据类型(bit)。
- 内存类型：变量地址介于 0X20～0X2f 的其中一个位，使用位寻址模式。
- 设定：在“升调键”和“降调键”按下时都设置此标志，以避免不放开键而一直动作中。
- 判断：要作升调和降调的动作，需事先判断此标志为“0”，才允许动作。

2. KeyData：按键的数值，以区分出到底哪一个键被按下了。

- 数据类型：占用一个字节的无符号字符类型(unsigned char)。
- 内存类型：变量地址介于 0X00～0Xff 之间，使用位寻址模式(idata)。
- 写入：当按下“升调键”则写入升调键的数值，当按下“降调键”则需写入降调键的数值，当两个都未按下才写入 0X00 的数值。
- 判断：由于按不同键会写入不同的数值，所以才要个别判断键值，当 KeyData 等于 UP_KEY，表示按下“升调键”，当 KeyData 等于 DOWN_KEY，表示按下“降调键”。

6-6-3　软件的解决方法

以两个 I/O 来作为两个按键的输入，并依次判断这两个 I/O 是否为“0”电位，以 input.c 模块作为侦测按键的函数，而在 Main( )函数调用，并立即判断是否按下

键而处理 KeyTone 的增量或减量，现分述如下。

1. main.c

（1）调用侦测 START_KEY 的函数：InputStar( )，当按下起始键，则立即执行此键功能：加载调子数值并演奏。

（2）调用侦测“升调键”和“降调键”的函数：InputKey( )，要依序判断是否按下了“升调键”，如是，则将 KeyTone 累加 1，如果按下了“降调键”，则将 KeyTone 递减 1。

（3）并以 while(1)不断地做外界按键的侦测，当处理完按键动作或演奏完成，则又继续进行侦测，只有按下 START_KEY 才会开始演奏，也就是当演奏完成也必须再按 START_KEY 才会又演奏，程序写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      //detect start key
      InputStart( );
      if ( KeyData == START_KEY )
      {
         Key=KEYTABLE[KeyTone];
         Music( );
         DelayX10ms(200);
      }

      //sel KeyTone
      InputKey( );
      if ( KeyData == UP_KEY )
```

```
        {
          if (FgSelectKey==0)
          {
            FgSelectKey = 1;
            Beep(1,17,10);

            //KeyTone=0~6
            KeyTone++;
            if ( KeyTone>6 ) //DO,RE,MIf,FA,SO,LAf,TIf
              KeyTone=0;
          }
        }
        else if ( KeyData == DOWN_KEY )
        {
          if (FgSelectKey==0)
          {
            FgSelectKey = 1;
            Beep(1,17,10);

            //KeyTone=0~6
            if ( KeyTone==0 )      //DO,RE,MIf,FA,SO,LAf,TIf
              KeyTone=7;
            KeyTone--;
          }
        }
    }
}
```

2. input.c：按键侦测函数 InputKey( )，除了要判断 UP_KEY 的 I/O，也要判断 DOWN_KEY 的 I/O，当 I/O 为“0”电位表示此键被按下了，而将 KeyData 存入对应的键值，而当 START_KEY I/O 为“0”电位，也表示按下起始键了，也将起始键值存入 KeyData 变量内，其程序写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
```

```
/*****************************/
void InputKey(void)
{
   if ( UP_IO==0 )
     KeyData=UP_KEY;
   else if ( DOWN_IO==0 )
     KeyData=DOWN_KEY;
   else
   {
     KeyData=NO_KEY;
     FgSelectKey = 0;
   }
}

/*****************************/
void InputStart(void)
{
   if ( START_IO==0 )
     KeyData=START_KEY;
   else
     KeyData=NO_KEY;
}
```

3. input.h：分别定义起始键 I/O 为 P3.4，“降调键” I/O 为 P1.2，“升调键” I/O 为 P1.3 以及各键的数值：未按下键的数值为 0，按下起始键的数值为 1，按下“降调键”的数值为 2，按下“升调键”的数值为 3，各键值的数值可任意修改，只要都不相同即可，一般以数值 1 开始定义，而未按下键的数值一般以 0X00 或 0Xff 为标准，包括文件的定义如下：

```
#ifndef   _INPUT_H
#define   _INPUT_H

#define  START_IO       P3_4
#define  DOWN_IO        P1_2
#define  UP_IO          P1_3

#define  NO_KEY              0
#define  START_KEY      1
#define  DOWN_KEY       2
#define  UP_KEY              3

extern void InputKey(void);
extern void InputStart(void);

#endif
```

汇编程序如下：

1. main.a51

```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN       SEGMENT CODE
 EXTRN     BIT (FgSelectKey)
 EXTRN     IDATA (KeyData)
 EXTRN     IDATA (KeyTone)
 EXTRN     IDATA (Key)
 EXTRN     CODE (PowerOnInit)
 EXTRN     CODE (_DelayX10ms)
 EXTRN     CODE (KEYTABLE)
 EXTRN     CODE (Music)
 EXTRN     CODE (InputKey)
 EXTRN     CODE (InputStart)
 EXTRN     CODE (_Beep)
 EXTRN     CODE (?C_STARTUP)
 PUBLIC    Main

 RSEG  ?PR?Main?MAIN
 USING     0
Main:
 LCALL     PowerOnInit
?C0001:
 LCALL     InputStart
 MOV       R0,#KeyData
 MOV       A,@R0
 CJNE      A,#01H,?C0003
 MOV       R0,#KeyTone
 MOV       A,@R0
 MOV       DPTR,#KEYTABLE
 MOVC      A,@A+DPTR
 MOV       R0,#Key
 MOV       @R0,A
 LCALL     Music
 MOV       R7,#0C8H
 MOV       R6,#00H
 LCALL     _DelayX10ms
?C0003:
```

```
LCALL     InputKey
MOV       R0,#KeyData
MOV       A,@R0
CJNE      A,#03H,?C0004
JB   FgSelectKey,?C0001
SETB      FgSelectKey
MOV       R7,#01H
MOV       R6,#00H
MOV       R5,#011H
MOV       R3,#0AH
LCALL     _Beep
MOV       R0,#KeyTone
INC       @R0
MOV       A,@R0
SETB      C
SUBB      A,#06H
JC   ?C0001
CLR       A
MOV       @R0,A
SJMP      ?C0001
?C0004:
MOV       R0,#KeyData
MOV       A,@R0
CJNE      A,#02H,?C0001
JB   FgSelectKey,?C0001
SETB      FgSelectKey
MOV       R7,#01H
MOV       R6,#00H
MOV       R5,#011H
MOV       R3,#0AH
LCALL     _Beep
MOV       R0,#KeyTone
MOV       A,@R0
JNZ       ?C0010
MOV       @R0,#07H
?C0010:
MOV       R0,#KeyTone
DEC       @R0
SJMP      ?C0001
RET

END
```

2. input.a51

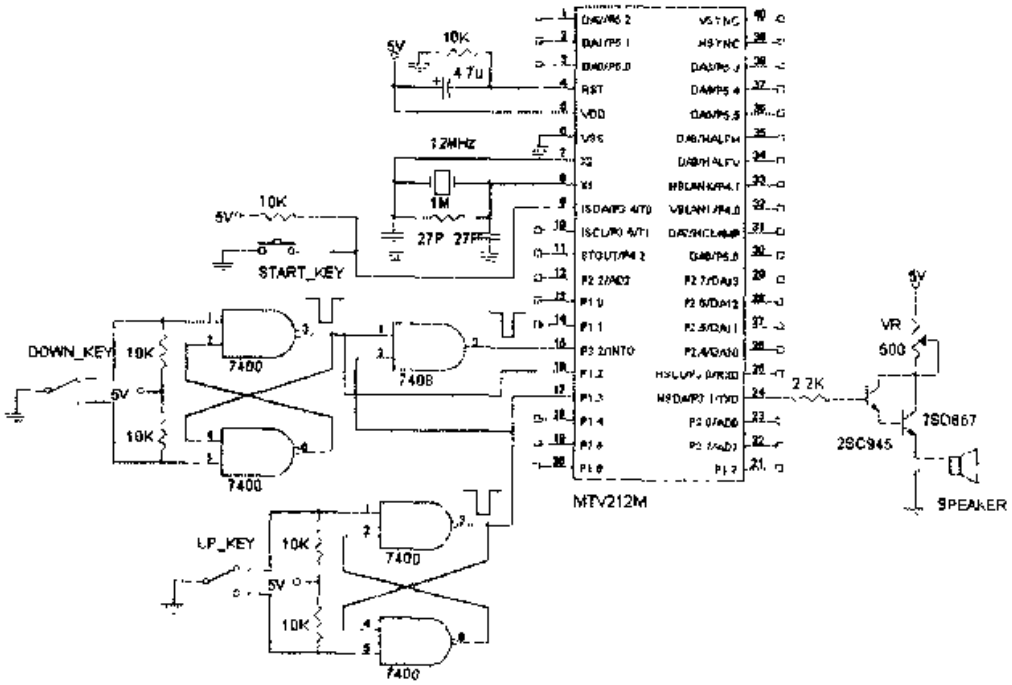
```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT      SEGMENT CODE
?PR?InputStart?INPUT        SEGMENT CODE
 EXTRN     BIT (FgSelectKey)
 EXTRN     IDATA (KeyData)
 PUBLIC    InputStart
 PUBLIC    InputKey

 RSEG  ?PR?InputKey?INPUT
 USING     0
InputKey:
 JB   P1_3,?C0001
 MOV       R0,#KeyData
 MOV       @R0,#03H
 RET
?C0001:
 JB   P1_2,?C0003
 MOV       R0,#KeyData
 MOV       @R0,#02H
 RET
?C0003:
 CLR       A
 MOV       R0,#KeyData
 MOV       @R0,A
 CLR       FgSelectKey
?C0005:
 RET

 RSEG  ?PR?InputStart?INPUT
 USING     0
InputStart:
 JB   P3_4,?C0006
 MOV       R0,#KeyData
 MOV       @R0,#01H
 RET
?C0006:
 CLR       A
 MOV       R0,#KeyData
 MOV       @R0,A
?C0008:
```

```
RET

END
```

6-7 中断法的移调

目的：在演奏中可随时作移调功能，并立即以移调后音符演奏。

电路图：



原理：按键的弹跳波先经过处理，以消除弹跳波后再输入至中断引脚，平常 DOWN_KEY 和 UP_KEY 经过两个 7400 NAND 闸处理后，其输出为“1”电位，再分别接至 AND 闸，以得到 into 的引脚也是“1”电位，当任何一个开关按下又释放后，会产生一个负脉冲而令 CPU 产生中断，则在外部中断程序判断 I/O P1.2 及 P1.3 是否为“0”电位，当 P1.2= “0” 则表示由 DOWN_KEY 产生中断，当 P1.3= “0” 时，则表示由 UP_KEY 来产生中断，由此可知是按了“降调键”还是“升调键”。

6-7-1 软件构建的思维

由于演奏函数并没有一直侦测 I/O 移调的动作（如果以 I/O 方式侦测是否移调，势必会占用 CPU 时间，而且程序结构较为混乱），因而，要以中断的方法来演奏。

如果按了“降调键”或“升调键”则可立即执行中断程序，而令 KeyTone 变量立即更新，在演奏函数中每一个音符也都会以 KeyTone 作调子数值的更新，而达到按键后立即演奏新调子的功能，现分述如下：

1. music.c：为了可以实时移调，且立即演奏移调后的曲目音符，则 key 变量的写入不能在 Main( )函数中做，要移到读取每个音符的回路内，那么，当执行中断程序而改变 KeyTone，则 Key 变量也会随着 KeyTone 变化而改变，则读取音符的半周期时间也会跟着变化，达到立即移调演奏的目的。

2. timer.c：要增加一个外部中断的函数，此函数最主要的目的是要判断到底是按下 UP_KEY 产生的中断，还是按下 DOWN_KEY 所产生的中断，而能适当修正 KeyTone 变量的内容。

3. input.h：虽然是利用中断的方法，但仍须借着判断 I/O 是否为“0”电位来区分“升调键”或“降调键”，因此，还是要定义个别的 I/O 引脚，但只需定义起始键的数值即可，“升调键”数值和“降调键”数值已经不是在 input.c 模块侦测 I/O 而写入至 KeyData 内，而是在中断程序直接侦测 I/O 而立即改变 KeyTone 的方式，所以无需再定义数值了。

6-7-2　软件的解决方法

侦测“升调键”和“降调键”不需要独立侦测了，在 input.c 的模块下就无需编写程序了，Main( )函数也无须调用了，主要是因为硬件电路已经设计成当按下“升调键”或“降调键”都立即产生中断，借着在中断程序判断到底是哪一个键并立即改变 KeyTone 的数值，而在演奏函数中读取音符前都更新 Key 变量就能达到所求了，兹分述如下：

1. music.c：演奏音符是利用 while 循环来顺序读取，而在读取音符之前就更新目前的调子，以作为平移音符的依据，即将 Main( )函数的 key=KEYTABLE[KeyTone] 的指令移至 Music( )函数的 while 循环下的第一个指令，程序如下：

```
/*****************************/
void Music(void)
{
   Byte i=0,j;

   Tempo = MODERATO;

   while ( (SOUNDLONG[i]!=_EOF)||(SOUNDTONE[i]!=_EOF) )
   {
      Key=KEYTABLE[KeyTone];
      Period=SOUNDTONE[i];
```

```
        j=0;
        while ( (SOUNDNUMBER[j]!=_EOF) )
        {
              if (Period==SOUNDNUMBER[j])
                break;
           j++;
        }
        Period=SOUNDNUMBER[j+Key];

        TL1 = (65536 - Period) & 0xff;
        TH1 = (65536 - Period) >> 8;
        TR1 = 1;

        SoundLongCount=((Tempo*SOUNDLONG[i]*1000L)/8)/Period;
        while ( SoundLongCount != 0 );

        TR1 = 0;      //stop timer1:soundtone
        i++;
    }
}
```

2. timer.c：中断的好处就是随时输入就能实时反应，想要有实时的需求功能，就要想到利用中断的方法，还要思索着硬件电路该如何配合？如何设计？而中断函数该如何编写？要处理的项目是什么等？有了这些思维才能逐步的去完成目标。此范例主要是判断I/O的变化，就能知道到底按了“升调键”或“降调键”而改变KeyTone的内容，为何能够如此，因为硬件电路是这样设计呀！如果不将输入信号拉至 I/O做侦测，那又如何判定到底是按了“升调键”或按了“降调键”呢？程序写法如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*******************************/
//timer1,execute service route
/*******************************/
void  Timer1ISR ( void ) interrupt 3 using 2
```

```
{
    Word temp;

    if ( SoundLongCount != 0 )
      SoundLongCount--;

    temp = 65536 - Period;
    SPEAKER = !SPEAKER;

    TL1 = temp & 0xff;
    TH1 = temp >> 8;
    TF1 = 0;
}

/*****************************/
//INT0 interrupt:
//INT0 Pin:"1"->"0"->"1" pulse
/*****************************/
void  INT0ISR (void) interrupt 0 using 3
{
    EX0 = 0;                        //disable int0

    //sel KeyTone
    if ( UP_IO==0 )
    {
      //KeyTone=0~6
      KeyTone++;
      if ( KeyTone>6 )        //DO,RE,MIf,FA,SO,LAf,TIf
        KeyTone=0;
    }
    else if ( DOWN_IO==0 )
    {
      //KeyTone=0~6
      if ( KeyTone==0 )       //DO,RE,MIf,FA,SO,LAf,TIf
        KeyTone=7;
      KeyTone--;
    }

    EX0 = 1;                        //enable next interrupt
    IE0 = 0;                        //set TCON.1 flag
}
```

3. input.h：“移调键”I/O 和“降调键”I/O 以及起始键 I/O 都需要定义，数值只定义起始键的数值即可，并定义其数值为 1，未按下起始键，其数值为 0，包括文件的定义如下：

```
#ifndef   _INPUT_H
#define   _INPUT_H

#define  START_IO       P3_4
#define  DOWN_IO        P1_2
#define  UP_IO          P1_3

#define  NO_KEY              0
#define  START_KEY      1

extern void InputStart(void);

#endif
```

汇编程序如下：

1. timer.a51

```
$NOMOD51

NAME TIMER

$INCLUDE (mtv212m.inc)

?PR?Timer1ISR?TIMER     SEGMENT CODE
?PR?INT0ISR?TIMER       SEGMENT CODE
 EXTRN     IDATA (KeyTone)
 EXTRN     IDATA (Period)
 EXTRN     IDATA (SoundLongCount)
 PUBLIC    INT0ISR
 PUBLIC    Timer1ISR

CSEG AT   0001BH
 LJMP Timer1ISR

 RSEG  ?PR?Timer1ISR?TIMER
 USING     2
Timer1ISR:
 PUSH      ACC
 PUSH      PSW
 MOV       PSW,#010H
 MOV       R0,#SoundLongCount+01H
 MOV       A,@R0
 DEC       R0
 ORL       A,@R0
 JZ   ?C0001
```

```
INC     R0
MOV     A,@R0
DEC     @R0
JNZ     ?C0009
DEC     R0
DEC     @R0
?C0009:
?C0001:

MOV     R0,#Period
MOV     A,@R0
MOV     R6,A
INC     R0
MOV     A,@R0
MOV     R7,A
CLR     C
CLR     A
SUBB    A,R7
MOV     R7,A
CLR     A
SUBB    A,R6
MOV     R6,A

CPL     P3_1
XCH     A,R5
MOV     A,R7
XCH     A,R5
MOV     A,R5
MOV     TL1,A
MOV     A,R6
MOV     TH1,A
CLR     TF1
POP     PSW
POP     ACC
RETI

CSEG AT  00003H
 LJMP INT0ISR

 RSEG  ?PR?INT0ISR?TIMER
 USING    3
INT0ISR:
 PUSH    ACC
 PUSH    PSW
 MOV     PSW,#018H
```

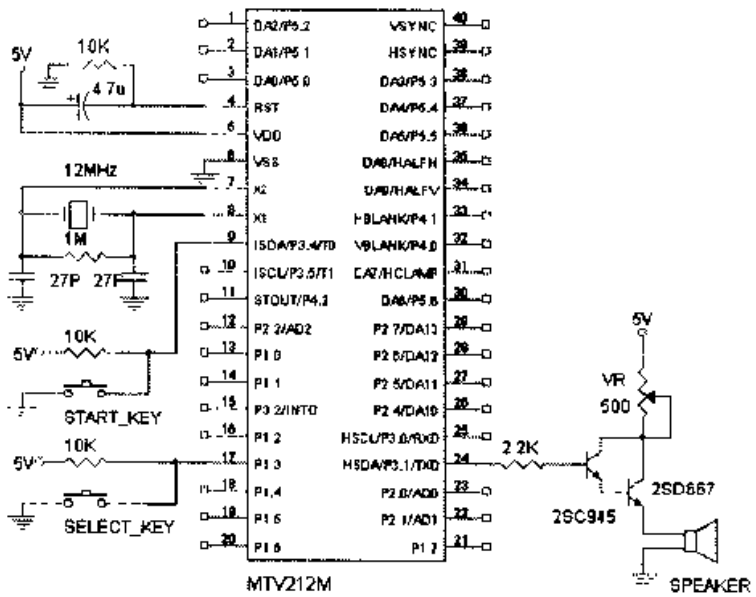
```
CLR     EX0
JB   P1_3,?C0003
MOV     R0,#KeyTone
INC     @R0
MOV     A,@R0
SETB    C
SUBB    A,#06H
JC   ?C0005
MOV     @R0,#00H
SJMP    ?C0005
?C0003:
JB   P1_2,?C0005
MOV     R0,#KeyTone
MOV     A,@R0
JNZ     ?C0007
MOV     @R0,#07H
?C0007:
MOV     R0,#KeyTone
DEC     @R0
?C0005:
SETB    EX0
CLR     IE0
POP     PSW
POP     ACC
RETI

END
```

第 7 章　按键开关选曲

7-1　以 Beep 作为按键输入提示

目的：以 Tact 开关来选曲，按一下就选择下一首曲目，并发出“哔”声作为提示，曲目范围为 1~9 首。

电路图：



原理：电路非常简单，利用 Tact 开关来选曲，每当 SELECT_KEY 按下，则 I/O P1.3 的输入为“0”电位，而平常未按下，其输入电位为“1”电位。当动作发生变化（按下键），其输入的信号也会随之变化，因此，可以很容易地侦测出是否按下键，按一下就将曲目编号加 1，范围从 1～9 循环，当选曲完成按下 START_KEY，则 P3.4 的 I/O 输入变成“0”电位，当侦测到 P3.4= “0”，就依据曲目编号而进行演奏。由前述章节至此，可以很清楚地知道 CPU I/O 的输入侦测或输出动作，是非常实用的，而且程序的写法不管是设定、清除或判断也很简单。请各位读者要学会位寻址的方法和标志的观念、使用时机，以 bit 为数据类型的变量声明，则此变量称为标志变量，标志变量只有两种状态，“0”，即标志清除为 0，“1”，即标志设定为“1”，要思索着什么时候设定？什么时候清除？什么时候要判断此标志？判断条件成立或不成立时，要存取此标志吗？此

标志影响层面为何等。

模块化：以Tact开关作为选曲的输入动作，每按一下就将选曲的曲目编号加1，当已经选到第九首，则再按一下开关，需从第一首开始，因而要增加曲目编号为1~9的音乐模块，作为实际演奏音乐的曲目，名称分别为：music1.c、mucis2.c、music3.c、music4.c、music5.c、music6.c、music7.c、music8.c、music9.c等，总共可规划的模块如下：

1. global.c：声明全局变量，数组、指针的模块。
2. main.c：主程序模块、提供程序进入主函数：Main( )函数。
3. initial.c：开机后初始化动作的模块。
4. delay.c：时间延迟的模块。
5. timer.c：由定时器1中断所产生音阶频率的模块。
6. input.c：侦测起始键和选曲键的按键输入模块。
7. beep.c：每按输入选曲键的喇叭响叫模块。
8. music.c：以条件式判断的方式来决定演奏哪一首曲目标模块，为演奏函数的主程序模块。
9. music1.c：曲目1的音乐演奏模块：三轮车。
10. music2.c：曲目2的音乐演奏模块：红彩妹妹。
11. music3.c：曲目3的音乐演奏模块：只要我长大。
12. music4.c：曲目4的音乐演奏模块：小星星。
13. music5.c：曲目5的音乐演奏模块：捉泥鳅。
14. music6.c：曲目6的音乐演奏模块：送别。
15. music7.c：曲目7的音乐演奏模块：蒙古牧歌。
16. music8.c：曲目8的音乐演奏模块：祝您生日快乐。
17. music9.c：曲目9的音乐演奏模块：妹妹背着洋娃娃。

以及下列的包括文件：

1. global.h：将gloabal.c模块下所有的变量、数组或指针作外部声明。
2. initial.h：initial.c模块下的函数外部声明。
3. delay.h：delay.c模块下的函数外部声明。
4. input.h：input.c模块下的函数外部声明和I/O的定义及常数定义。
5. beep.h：beep.c模块下的函数外部声明。
6. music.h：music.c模块下的函数外部声明。
7. define.h：自定数据类型（Bit、Bool、Byte、Word、Long）和内存类型的定义。
8. mtv212m.h：8051 CPU SFR地址声明及I/O地址声明。

7-1-1　开关持续按着会重复动作

1. 软件构建的思维

在 Main( )函数下不断地侦测是否按了起始键和是否按了选曲键，当按了起始键，便根据曲目编号进行演奏，当按了选曲键，则将曲目编号累加 1，按一下为演奏第一首，按两下为演奏第二首，按三下为演奏第三首，所曲目编号必须初始为 0。又由于总共有九首曲目，因此要有依据曲目编号而执行以曲目函数的结构或主程序，此为 Music( )函数的产生，现分述如下。

（1）global.c：整个软件结构，需要新增一个叫做曲目编号的变量，当选曲键调整时，这个变量要跟着变；当按下起始键则需依据此变量演奏曲目，你说，这个变量是不是程序中很重要的参数，是不是要把它研究透彻一点，是不是要多去思考它的使用时机呢？此变量名称规划为 MusicNumber，即曲目编号之意，变量名称尽量取可以望文生义，一看到这个英文字就能知道它的意义，它在程序中所占的份量，千万不要取名称为阿狗、阿猫、AAA、BBB，到时候真会忘了我是谁了？到了你要修改程序、增加新功能或有 BUG 需要处理，或到了别人来接你的工作时，就会很麻烦。

（2）global.h：新增变量要声明为外部变量。

（3）main.c：侦测按键的动作必须一直循环侦测，包含起始键和选曲键，以及按键后的命令功能。

（4）initial.c：曲目编号需先清除为 0，则按选曲键一次，才会从第一首曲目开始演奏。

（5）initial.h：初始化变量的函数必须在包括文件内作外部函数声明。

（6）input.c：只侦测起始键 I/O 是否变为“0”电位，选曲键 I/O 是否也变为“0”电位，并以一个变量来写入按键输入的状态，以便其他模块判断此变量的内容，就可以知道外界按键是否按下。

（7）input.h：定义 I/O 的引脚，以英文字来表示此 I/O 所提供的功能为何较易明了，即直接将 I/O 的引脚写在程序上，以及定义数值和函数外部声明。

（8）music.c：演奏哪一首曲目上层函数的模块，将所有曲目的函数调用集中于此，较易明了。

（9）music.h：每一首曲目函数需要作外部声明。

2. 参数的意义说明与使用时机

MusicNumber：曲目编号

- 数据类型：占用一个字节的无符号字符类型(unsigned char)。

- 内存类型：变量地址介于 0X00～0Xff 之间，使用间接寻址模式(idata)。
- 清除：为了要在第一次按下选曲键，就能从第一首开始演奏，则必须在 initial.c 模块内清除为 0。
- 判断：当选曲至第九首曲目时，再按一次选曲键必须变为第一首，即必须先判断此参数是否大于 9，当条件成立则写入 1。
- 写入：曲目编号大于 9 时，才执行写入为 1 的数值。
- 自变量：按下起始键，则将此参数作为自变量传递给音乐演奏函数，才能正确地演奏出所选择的曲目，曲目编号变量的内容所代表的歌曲名称如下：

Music Number	歌曲名称
1	三轮车
2	红彩妹妹
3	只要我长大
4	小星星
5	捉泥鳅
6	送别
7	蒙古牧歌
8	祝您生日快乐
9	妹妹背着洋娃娃

3. 软件的解决方法

最主要的是处理 MusicNumber 曲目编号的变量，在每按下选曲键，必须将 MusicNumber 变量加 1。当达到最大值，也要作重置处理，而于按下起始键，并根据曲目编号进行演奏；而在 music.c 模块下的 Music( )函数已经不再是单一曲目的演奏函数，而是必须借着 number 自变量来执行到底要演奏哪一首曲目的处理函数了，现分述如下。

（1）global.c：声明 MusicNumber 变量的数据类型和内存类型，如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"

/*******************************/
// Global Data
/*******************************/
```

```
//key
Byte      IDATA   KeyData;

//music
Byte      IDATA   MusicNumber;
Word      IDATA   Tempo;
Word      IDATA   Period;
Word      IDATA   SoundLongCount;
```

（2）global.h：MusicNumber 变量的外部声明，如下：

```
#ifndef   _GLOBAL_H
#define   _GLOBAL_H

//key
extern Byte    IDATA   KeyData;

//music
extern Byte    IDATA   MusicNumber;
extern Word  IDATA   Tempo;
extern Word  IDATA   Period;
extern Word  IDATA   SoundLongCount;

#endif
```

（3）main.c：按下起始键和选曲键的功能处理程序，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
```

```
        //detect start key
        InputStart( );
        if ( KeyData == START_KEY )
        {
          Music(MusicNumber);
          DelayX10ms(200);
        }

        //sel play no#
        InputKey( );
        if ( KeyData == SELECT_KEY )
        {
          Beep(1,17,10);

          MusicNumber++;
          if ( MusicNumber>9 )
            MusicNumber=1;
        }
      }
    }
```

（4）initial.c：于初始变量函数内需要将 MusicNumber 变量清除，当按选曲键，才会从第一首开始，函数如下：

```
/*****************************/
void InitialVariable(void)
{
    KeyData     = 0;

    MusicNumber= 0;
}
```

（5）initial.h：initial.c 模块下所有函数的外部声明，如下：

```
#ifndef   _INITIAL_H
#define   _INITIAL_H

extern void PowerOnInit(void);
extern void InitialCpu(void);
extern void InitialCpuIO(void);
extern void InitialVariable(void);

#endif
```

（6）input.c：只单纯地侦测 I/O，包含起始键和选曲键，当按下键，则 I/O 的输入信号为“0”电位，所以程序判断当 I/O=“0”时，KeyData=键值，否则，Keydata=0，

程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"

/*****************************/
void InputKey(void)
{
   if ( SELECT_IO==0 )
     KeyData=SELECT_KEY;
   else
     KeyData=NO_KEY;
}

/*****************************/
void InputStart(void)
{
   if ( START_IO==0 )
     KeyData=START_KEY;
   else
     KeyData=NO_KEY;
}
```

（7）input.h：起始键定义为 I/O P3.4，选曲键则定义为 I/O P1.3，要配合硬件电路而定义，键值则不可重复，如下方法：

```
#ifndef   _INPUT_H
#define   _INPUT_H

#define  START_IO       P3_4
#define  SELECT_IO      P1_3

#define  NO_KEY              0
#define  START_KEY      1
#define  SELECT_KEY     2
```

```
extern void InputKey(void);
extern void InputStart(void);

#endif
```

（8）music.c：以 switch...case 的条件循环，而以自变量当作条件判断式，使得程序结构一目了然，又可达到依据曲目编号而演奏该曲目的功能，因为 Main( ) 函数会将 MusicNumber 变量内容传递给 Music( )函数中的 number 自变量。当开机后，MusicNumber 会初始化为 0，当未按下选曲键就按下起始键时，则 Music( ) 函数的自变量 number 也会为 0，将执行 default 指令之后的语句，即执行 break 指令而离开 switch...case 循环返回到 Main( )函数，即不演奏，而曲目编号为 0，本来就没有歌曲，更不应该演奏，整个程序流程、算法是合乎逻辑的，其程序写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
void Music(Byte number)
{
   switch( number )
   {
    case 1 :
       Music1( );
       break;
    case 2 :
       Music2( );
       break;
    case 3 :
       Music3( );
       break;
    case 4 :
       Music4( );
       break;
    case 5 :
       Music5( );
       break;
```

```
        case 6 :
            Music6( );
            break;
        case 7 :
            Music7( );
            break;
        case 8 :
            Music8( );
            break;
        case 9 :
            Music9( );
            break;
        default :
            break;
    }
}
```

（9）music.h：除了演奏音乐的处理函数 Music( )外，也需将个别曲目的演奏函数作外部声明，如下：

```
extern void Music(Byte number);
extern void Music1(void);
extern void Music2(void);
extern void Music3(void);
extern void Music4(void);
extern void Music5(void);
extern void Music6(void);
extern void Music7(void);
extern void Music8(void);
extern void Music9(void);
```

（10）mucic1.c：曲目 1 的音乐函数，其程序如下：

```
/*****************************/

/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
unsigned char code SOUNDLONG1[ ] =
{
    _8, _8 ,_8d,_16,
```

```
    _8, _8 ,_4 ,
    _8, _8 ,_8 ,_8 ,
    _8, _8 ,_4 ,
    _8, _8 ,_8d,_16,
    _8, _8 ,_8d,_16,
    _8, _16,_16,_8 ,_16,_16,
    _8, _8 ,_4 ,
    _EOF
};

unsigned int code SOUNDTONE1[ ] =
{
    _DO ,_DO ,_RE,_MI,
    _SO ,_SO ,_MI,
    _SO ,_SO ,_LA,_TI,
    _DO1,_DO1,_SO,
    _DO1,_DO1,_LA,_SO,
    _MI ,_LA ,_SO,_MI,
    _DO ,_RE ,_MI,_SO,_LA,_SO,
    _MI ,_RE ,_DO,
    _EOF
};

void Music1(void)
{
    unsigned char i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG1[i]!=_EOF)||(SOUNDTONE1[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE1[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG1[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

（11）mucic2.c：曲目 2 的音乐函数，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG2[ ] =
{
    _4 ,_8 ,_8 ,_4 ,_8 ,_8 ,
    _8 ,_4 ,_8 ,_2 ,
    _4 ,_8 ,_8 ,_4 ,_8 ,_8 ,
    _8 ,_4 ,_8 ,_2 ,
    _4 ,_8 ,_8 ,_8 ,_8 ,_8 ,_8 ,
    _8 ,_4 ,_8 ,_2 ,
    _4 ,_4 ,_4 ,_8 ,_8 ,
    _8 ,_4 ,_8 ,_2 ,
    _EOF
};

Word RDATA SOUNDTONE2[ ] =
{
    _LA ,_SO ,_MI ,_LA ,_SO ,_MI ,
    _LA ,_LA ,_SO ,_LA ,
    _LA ,_SO ,_MI ,_LA ,_SO ,_MI ,
    _RE ,_RE ,_DO ,_RE ,
    _MI ,_MI ,_SO ,_LA ,_DO1,_LA ,_SO ,
    _MI ,_MI ,_SO ,_DO ,
    _MI ,_MI ,_MI ,_MI ,_MI ,
    _1LA,_1LA,_1SO,_1LA,
    _EOF
};

void Music2(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG2[i]!=_EOF)||(SOUNDTONE2[i]!=_EOF) )
    {
```

```
        //start timer1,generate soundtone
        Period=SOUNDTONE2[i];

        TL1 = (65536 - Period) & 0xff;
        TH1 = (65536 - Period) >> 8;
        TR1 = 1;

        SoundLongCount=((Tempo*SOUNDLONG2[i]*1000L)/8)/Period;
        while ( SoundLongCount != 0 );

        TR1 = 0;      //stop timer1:soundtone
        i++;
    }
}
```

（12）mucic3.c：曲目3的音乐函数，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG3[ ] =
{
    _8 ,_8 ,_8 ,_8 ,_8 ,_8 ,_4 ,
    _8d,_16,_8 ,_8 ,_2 ,
    _4 ,_4 ,_8 ,_8 ,_4 ,
    _4 ,_4 ,_8 ,_8 ,_4 ,
    _8 ,_8 ,_8 ,_8 ,_8 ,_8 ,_3 ,_8 ,
    _8d,_16,_8 ,_8 ,_8 ,_8 ,_4 ,
    _8 ,_16,_16,_8 ,_8 ,_2 ,
    _8 ,_16,_16,_8 ,_8 ,_2 ,
    _EOF
};

Word RDATA SOUNDTONE3[ ] =
{
    _DO1,_DO1,_SO ,_SO,_LA,_LA,_SO,
    _MI ,_SO ,_DO1,_LA,_SO,
    _LA ,_SO ,_MI ,_LA ,_SO ,
    _MI ,_SO ,_MI ,_RE ,_DO ,
```

```
    _MI ,_MI ,_SO ,_SO ,_LA ,_LA ,_SO ,_SO ,
    _RE1,_DO1,_LA ,_SO ,_LA ,_DO1,_SO ,
    _LA ,_LA ,_SO ,_LA ,_DO1,_SO ,
    _LA ,_LA ,_SO ,_LA ,_RE1,_DO1,
    _EOF
};

void Music3(void)
{
    Byte i=0;

    Tempo = MODERATO;

    While ( (SOUNDLONG3[i]!=_EOF)||(SOUNDTONE3[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE3[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG3[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

（13）mucic4.c：曲目 4 的音乐函数，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include "define.h"
#include "mtv212m.h"
#include "global.h"
#include "music.h"

/*****************************/
Byte RDATA SOUNDLONG4[ ] =
{
    _4 ,_4 ,_4 ,_4 ,_4 ,_4 ,_2 ,
    _4 ,_4 ,_4 ,_4 ,_4 ,_4 ,_2 ,
```

```
    _4 ,_4 ,_4 ,_4 ,_4 ,_4 ,_2 ,
    _4 ,_4 ,_4 ,_4 ,_4 ,_4 ,_2 ,
    _4 ,_4 ,_4 ,_4 ,_4 ,_4 ,_2 ,
    _4 ,_4 ,_4 ,_4 ,_4 ,_4 ,_2 ,
    _EOF
};

Word RDATA SOUNDTONE4[ ] =
{
    _DO ,_DO ,_SO ,_SO ,_LA ,_LA ,_SO ,
    _FA ,_FA ,_MI ,_MI ,_RE ,_RE ,_DO ,
    _SO ,_SO ,_FA ,_FA ,_MI ,_MI ,_RE ,
    _SO ,_SO ,_FA ,_FA ,_MI ,_MI ,_RE ,
    _DO ,_DO ,_SO ,_SO ,_LA ,_LA ,_SO ,
    _FA ,_FA ,_MI ,_MI ,_RE ,_RE ,_DO ,
    _EOF
};

void Music4(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG4[i]!=_EOF)||(SOUNDTONE4[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE4[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG4[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

（14）mucic5.c：曲目 5 的音乐函数，其程序如下：

```
/*****************************/
/* include files             */
```

```
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG5[ ] =
{
    _4 ,_8d,_16,_8 ,_8 ,_4 ,
    _8 ,_8 ,_8 ,_8 ,_4 ,
    _4 ,_8d,_16,_8 ,_8 ,_4 ,
    _8 ,_8 ,_8 ,_8 ,_4 ,
    _4 ,_8d,_16,_8 ,_8 ,_4 ,
    _8 ,_16,_16,_8 ,_8 ,_4 ,
    _4 ,_8d,_16,_8 ,_8 ,_8 ,_8 ,
    _8 ,_16,_16,_8 ,_8 ,_4 ,
    _4 ,_8 ,_8 ,_8d,_16,_4 ,
    _8 ,_16,_16,_8 ,_8 ,_8d,_16,_4 ,
    _4 ,_8d,_16,_8 ,_8 ,_8 ,_8 ,
    _8 ,_16,_16,_8 ,_8 ,_4 ,
    _EOF
};

Word RDATA SOUNDTONE5[ ] =
{
    _LA ,_LA ,_SO ,_LA ,_SO ,_MI ,
    _SO ,_MI ,_MI ,_RE ,_MI ,
    _RE ,_RE ,_DO ,_RE ,_RE ,_SO ,
    _SO ,_MI ,_MI ,_RE ,_MI ,
    _LA ,_LA ,_SO ,_LA ,_SO ,_MI ,
    _FA ,_FA ,_FA ,_MI ,_RE ,_MI ,
    _SO ,_SO ,_SO ,_SO ,_SO ,_SO ,_TI ,
    _LA ,_LA ,_LA ,_LA ,_SO ,_LA ,
    _DO1,_DO1,_DO1,_TI ,_LA ,_SO ,
    _LA ,_LA ,_LA ,_LA ,_SO ,_LA ,_SO ,_MI ,
    _SO ,_SO ,_SO ,_SO ,_SO ,_SO ,_TI ,
    _LA ,_LA ,_LA ,_LA ,_SO ,_LA ,
    _EOF
};

void Music5(void)
{
    Byte i=0;
```

```
    Tempo = MODERATO;

    while ( (SOUNDLONG5[i]!=_EOF)||(SOUNDTONE5[i]!=_EOF) )
    {
       //start timer1,generate soundtone
       Period=SOUNDTONE5[i];

       TL1 = (65536 - Period) & 0xff;
       TH1 = (65536 - Period) >> 8;
       TR1 = 1;

       SoundLongCount=((Tempo*SOUNDLONG5[i]*1000L)/8)/Period;
       while ( SoundLongCount != 0 );

       TR1 = 0;      //stop timer1:soundtone
       i++;
    }
}
```

（15）mucic6.c：曲目 6 的音乐函数，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG6[ ] =
{
    _4 ,_8 ,_8 ,_2 ,_4 ,_4 ,_2 ,
    _4 ,_8 ,_8 ,_4 ,_8 ,_8 ,_2d,
    _4 ,_8 ,_8 ,_4d,_8 ,_4 ,_4 ,_2 ,
    _4 ,_8 ,_8 ,_4d,_8 ,_2d,
    _4 ,_4 ,_2 ,_4 ,_8 ,_8 ,_2 ,
    _8 ,_8 ,_8 ,_8 ,_8 ,_8 ,_8 ,_8 ,_2 ,
    _4 ,_8 ,_8 ,_4d,_8 ,_4 ,_4 ,_2 ,
    _4 ,_8 ,_8 ,_4d,_8 ,_2d,
    _EOF
};

Word RDATA SOUNDTONE6[ ] =
{
```

```
    _SO ,_MI ,_SO ,_DO1,_LA ,_DO1,_SO,
    _SO ,_DO ,_RE ,_MI ,_RE ,_DO ,_RE ,
    _SO ,_MI ,_SO ,_DO1,_TI ,_LA ,_DO1,_SO,
    _SO ,_RE ,_MI ,_FA ,_1TI,_DO ,
    _LA ,_DO1,_DO1,_TI ,_LA ,_TI ,_DO1,
    _LA ,_TI ,_DO1,_LA ,_LA ,_SO ,_MI ,_DO ,_RE ,
    _SO ,_MI ,_DO ,_DO1,_TI ,_LA ,_DO1,_SO ,
    _SO ,_RE ,_MI ,_FA ,_1TI,_DO ,
    _EOF
};

void Music6(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG6[i]!=_EOF)||(SOUNDTONE6[i]!=_EOF) )
    {
       //start timer1,generate soundtone
       Period=SOUNDTONE6[i];

       TL1 = (65536 - Period) & 0xff;
       TH1 = (65536 - Period) >> 8;
       TR1 = 1;

       SoundLongCount=((Tempo*SOUNDLONG6[i]*1000L)/8)/Period;
       while ( SoundLongCount != 0 );

       TR1 = 0;      //stop timer1:soundtone
       i++;
    }
}
```

（16）mucic7.c：曲目 7 的音乐函数，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
```

```
Byte RDATA SOUNDLONG7[ ] =
{
    _4 ,_4 ,_8 ,_8 ,_8 ,_8 ,_4d,_8 ,_2 ,
    _4 ,_4 ,_8 ,_8 ,_8 ,_8 ,_4d,_8 ,_2 ,
    _4d,_8 ,_8 ,_8 ,_8 ,_8 ,_4 ,_8 ,_8 ,_8 ,_4 ,_8 ,
    _8 ,_8 ,_8 ,_8 ,_8 ,_8 ,_8 ,_8 ,_4d,_8 ,_2 ,
    _4d,_8 ,_8 ,_8 ,_8 ,_8 ,_4 ,_8 ,_8 ,_8 ,_4 ,_8 ,
    _8 ,_8 ,_8 ,_8 ,_8 ,_8 ,_8 ,_8 ,_4d,_8 ,_2 ,
    _EOF
};

Word RDATA SOUNDTONE7[ ] =
{
    _TI ,_TI ,_TI ,_LA ,_FAr,_LA ,_TI ,_RE1,_TI ,
    _LA ,_LA ,_RE1,_TI ,_LA ,_FAr,_MI ,_FAr,_MI ,
    _RE1,_TI ,_LA ,_TI ,_LA ,_FAr,_MI ,_LA ,_FAr,_MI ,_RE ,_TI ,
    _RE ,_TI ,_RE ,_FAr,_MI ,_FAr,_RE ,_TI ,_LA ,_TI ,_LA ,
    _RE1,_TI ,_LA ,_TI ,_LA ,_FAr,_MI ,_LA ,_FAr,_MI ,_RE ,_TI ,
    _RE ,_TI ,_RE ,_FAr,_MI ,_FAr,_RE ,_TI ,_LA ,_TI ,_LA ,
    _EOF
};

void Music7(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG7[i]!=_EOF)||(SOUNDTONE7[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE7[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG7[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

（17）mucic8.c：曲目 8 的音乐函数，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG8[ ] =
{
    _8d,_16,_4 ,_4 ,_4 ,_2 ,
    _8d,_16,_4 ,_4 ,_4 ,_2 ,
    _8d,_16,_4 ,_4 ,_4 ,_4 ,_4 ,
    _8d,_16,_4 ,_4 ,_4 ,_2 ,
    _EOF
};

Word RDATA SOUNDTONE8[ ] =
{
    _RE ,_RE ,_MI ,_RE ,_SO ,_FAr,
    _RE ,_RE ,_MI ,_RE ,_LA ,_SO ,
    _RE ,_RE ,_RE1,_TI ,_SO ,_FAr, _MI ,
    _DO1,_DO1,_TI ,_SO ,_LA ,_SO ,
    _EOF
};

void Music8(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG8[i]!=_EOF)||(SOUNDTONE8[i]!=_EOF) )
    {
       //start timer1,generate soundtone
       Period=SOUNDTONE8[i];

       TL1 = (65536 - Period) & 0xff;
       TH1 = (65536 - Period) >> 8;
       TR1 = 1;

       SoundLongCount=((Tempo*SOUNDLONG8[i]*1000L)/8)/Period;
```

```
        while ( SoundLongCount != 0 );

        TR1 = 0;      //stop timer1:soundtone
        i++;
    }
}
```

（18）mucic9.c：曲目9的音乐函数，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG9[ ] =
{
    _4d,_8 ,_4 ,_4 ,_4 ,_4 ,_2 ,
    _4d,_8 ,_4 ,_4 ,_4 ,_4 ,_2 ,
    _4d,_8 ,_4 ,_4 ,_4 ,_4 ,_2 ,
    _4d,_8 ,_4 ,_4 ,_4 ,_4 ,_2 ,
    _EOF          \
};

Word RDATA SOUNDTONE9[ ] =
{
    _RE1,_RE1,_TI ,_LA ,_TI ,_LA ,_SO ,
    _TI ,_LA ,_SO ,_MI ,_RE ,_MI ,_RE ,
    _MI ,_MI ,_SO ,_MI ,_SO ,_LA ,_TI ,
    _LA ,_LA ,_RE1,_RE1,_LA ,_TI ,_SO ,
    _EOF
};

void Music9(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG9[i]!=_EOF)||(SOUNDTONE9[i]!=_EOF) )
    {
       //start timer1,generate soundtone
```

```
        Period=SOUNDTONE9[i];

        TL1 = (65536 - Period) & 0xff;
        TH1 = (65536 - Period) >> 8;
        TR1 = 1;

        SoundLongCount=((Tempo*SOUNDLONG9[i]*1000L)/8)/Period;
        while ( SoundLongCount != 0 );

        TR1 = 0;      //stop timer1:soundtone
        i++;
    }
}
```

汇编程序如下：

1. global.a51

```
$NOMOD51

NAME GLOBAL

$INCLUDE (mtv212m.inc)

?ID?GLOBAL          SEGMENT IDATA
PUBLIC    SoundLongCount
PUBLIC    Period
PUBLIC    Tempo
PUBLIC    MusicNumber
PUBLIC    KeyData

RSEG  ?ID?GLOBAL
      KeyData:   DS   1
   MusicNumber:   DS   1
        Tempo:   DS   2
       Period:   DS   2
SoundLongCount:   DS   2

END
```

2. main.a51

```
$NOMOD51

NAME MAIN
```

```
$INCLUDE (mtv212m.inc)

?PR?Main?MAIN       SEGMENT CODE
 EXTRN     IDATA (KeyData)
 EXTRN     IDATA (MusicNumber)
 EXTRN     CODE (PowerOnInit)
 EXTRN     CODE (_DelayX10ms)
 EXTRN     CODE (_Music)
 EXTRN     CODE (InputKey)
 EXTRN     CODE (InputStart)
 EXTRN     CODE (_Beep)
 EXTRN     CODE (?C_STARTUP)
 PUBLIC    Main

 RSEG  ?PR?Main?MAIN
 USING     0
Main:
 LCALL     PowerOnInit
?C0001:
 LCALL     InputStart
 MOV       R0,#KeyData
 MOV       A,@R0
 CJNE      A,#01H,?C0003
 MOV       R0,#MusicNumber
 MOV       A,@R0
 MOV       R7,A
 LCALL     _Music
 MOV       R7,#0C8H
 MOV       R6,#00H
 LCALL     _DelayX10ms
?C0003:
 LCALL     InputKey
 MOV       R0,#KeyData
 MOV       A,@R0
 CJNE      A,#02H,?C0001
 MOV       R7,#01H
 MOV       R6,#00H
 MOV       R5,#011H
 MOV       R3,#0AH
 LCALL     _Beep
 MOV       R0,#MusicNumber
 INC       @R0
 MOV       A,@R0
 SETB      C
```

```
SUBB    A,#09H
JC   ?C0001
MOV     @R0,#01H
SJMP    ?C0001
RET

END
```

3. input.a51

```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT      SEGMENT CODE
?PR?InputStart?INPUT        SEGMENT CODE
EXTRN   IDATA (KeyData)
PUBLIC  InputStart
PUBLIC  InputKey

RSEG  ?PR?InputKey?INPUT
USING   0
InputKey:
JB   P1_3,?C0001
MOV     R0,#KeyData
MOV     @R0,#02H
RET
?C0001:
CLR     A
MOV     R0,#KeyData
MOV     @R0,A
?C0003:
RET

RSEG  ?PR?InputStart?INPUT
USING   0
InputStart:
JB   P3_4,?C0004
MOV     R0,#KeyData
MOV     @R0,#01H
RET
?C0004:
CLR     A
```

```
MOV       R0,#KeyData
MOV       @R0,A
?C0006:
RET

END
```

4. music.a51

```
$NOMOD51

NAME MUSIC

$INCLUDE (mtv212m.inc)

?PR?_Music?MUSIC        SEGMENT CODE
EXTRN     CODE (Music1)
EXTRN     CODE (Music2)
EXTRN     CODE (Music3)
EXTRN     CODE (Music4)
EXTRN     CODE (Music5)
EXTRN     CODE (Music6)
EXTRN     CODE (Music7)
EXTRN     CODE (Music8)
EXTRN     CODE (Music9)
EXTRN     CODE (?C?CCASE)
PUBLIC    _Music

RSEG  ?PR?_Music?MUSIC
USING     0
_Music:
MOV       A,R7
LCALL     ?C?CCASE
DW   ?C0002
DB   01H
DW   ?C0003
DB   02H
DW   ?C0004
DB   03H
DW   ?C0005
DB   04H
DW   ?C0006
DB   05H
DW   ?C0007
DB   06H
```

```
DW   ?C0008
DB   07H
DW   ?C0009
DB   08H
DW   ?C0010
DB   09H
DW   00H
DW   ?C0012

?C0002:
 LCALL    Music1
 RET
?C0003:
 LCALL    Music2
 RET
?C0004:
 LCALL    Music3
 RET
?C0005:
 LCALL    Music4
 RET
?C0006:
 LCALL    Music5
 RET
?C0007:
 LCALL    Music6
 RET
?C0008:
 LCALL    Music7
 RET
?C0009:
 LCALL    Music8
 RET
?C0010:
 LCALL    Music9
?C0012:
 RET

 END
```

7-1-2 开关持续按着也只动作一次

1. 软件构建的思维与解决方法

如上述Main( )函数，当持续按下SELECT_KEY（选曲键）不放，则会不断地发出“哔”声，表示一直动作，MusicNumber变量也一直累加1，当大于9又从1开始累加，容易造成动作不确实，选曲不正确之虑，然而要如何避免这个现象呢？其实很简单，只要增加一个标志变量即可，方法如下：

（1）在Main( )函数内，当按下SELECT_KEY的条件式成立时，先判断标志变量FgSelectKey是否等于“0”，当FgSelectKey等于“0”，才能真正开始，并且一定要将FgSelectKey设定为“1”，则当动作执行完毕，由于又继续按着键，所以也会继续执行按键成立下的叙述，然而，FgSelectKey已经设定为“1”了，所以就不会将动作再做一遍了。简单地说，先判断标志变量是否为“0”，是，则必须设定为“1”，再进行处理；否则，不处理动作直接离开，程序写法如下：

```
/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      //detect start key
      InputStart( );
      if ( KeyData == START_KEY )
      {
         Music(MusicNumber);
         DelayX10ms(200);
      }

      //sel play no#
      InputKey( );
      if ( KeyData == SELECT_KEY )
      {
         if (FgSelectKey==0)
         {
            FgSelectKey = 1;
            Beep(1,17,10);

            MusicNumber++;
            if ( MusicNumber>9 )
```

```
                MusicNumber=1;
            }
        }
    }
}
```

（2）当按下选曲键，则 FgSelectKey 将会设定为“1”，然而，要如何清除？清除的时机是什么？这样才能为下一次按键又能动作而继续选曲。不错，这样的思维是对的，当按下键，则标志设定为“1”，一直压着也一直为“1”，只有放开键，则标志也必须立即变为“0”，则下一次按键才能正确动作，还记得 Main( )函数一直侦测按键函数吧。因此，只要在侦测选曲键 Inputkey( )的函数，当释放键（即选曲键 I/O=“1”电位时），就将 FgSelectKey 清除为“0”即可，程序如下：

```
/*****************************/
void InputKey(void)
{
   if ( SELECT_IO==0 )
     KeyData=SELECT_KEY;
   else
   {
     KeyData=NO_KEY;
     FgSelectKey = 0;
   }
}
```

（3）由于新增一个标志变量 FgSelectKey，所以在 global.c 模块中要作声明，如下：

```
Bool  FgSelectKey;
```

（4）在 global.h 包括文件下也要作外部声明，如下：

```
extern  Bool  FgSelectKey;
```

汇编程序如下：

1. main.a51

```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN       SEGMENT CODE
EXTRN     BIT (FgSelectKey)
EXTRN     IDATA (KeyData)
EXTRN     IDATA (MusicNumber)
```

```
EXTRN    CODE (PowerOnInit)
EXTRN    CODE (_DelayX10ms)
EXTRN    CODE (_Music)
EXTRN    CODE (InputKey)
EXTRN    CODE (InputStart)
EXTRN    CODE (_Beep)
EXTRN    CODE (?C_STARTUP)
PUBLIC   Main

RSEG  ?PR?Main?MAIN
USING    0
Main:
LCALL    PowerOnInit
?C0001:
LCALL    InputStart
MOV      R0,#KeyData
MOV      A,@R0
CJNE     A,#01H,?C0003
MOV      R0,#MusicNumber
MOV      A,@R0
MOV      R7,A
LCALL    _Music
MOV      R7,#0C8H
MOV      R6,#00H
LCALL    _DelayX10ms
?C0003:
LCALL    InputKey
MOV      R0,#KeyData
MOV      A,@R0
CJNE     A,#02H,?C0001
JB   FgSelectKey,?C0001
SETB     FgSelectKey
MOV      R7,#01H
MOV      R6,#00H
MOV      R5,#011H
MOV      R3,#0AH
LCALL    _Beep
MOV      R0,#MusicNumber
INC      @R0
MOV      A,@R0
SETB     C
SUBB     A,#09H
JC   ?C0001
MOV      @R0,#01H
SJMP     ?C0001
```

```
RET

END
```

2. input.a51

```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT      SEGMENT CODE
?PR?InputStart?INPUT        SEGMENT CODE
EXTRN     BIT (FgSelectKey)
EXTRN     IDATA (KeyData)
PUBLIC    InputStart
PUBLIC    InputKey

RSEG  ?PR?InputKey?INPUT
USING     0
InputKey:
JB   P1_3,?C0001
MOV       R0,#KeyData
MOV       @R0,#02H
RET
?C0001:
CLR       A
MOV       R0,#KeyData
MOV       @R0,A
CLR       FgSelectKey
?C0003:
RET

RSEG  ?PR?InputStart?INPUT
USING     0
InputStart:
JB   P3_4,?C0004
MOV       R0,#KeyData
MOV       @R0,#01H
RET
?C0004:
CLR       A
MOV       R0,#KeyData
MOV       @R0,A
```

```
?C0006:
RET

END
```

7-1-3 清除按键弹跳波的程序规划

1. 软件构建的思维与解决方法

上述 Main( )函数内，利用 FgSelectKey 标志变量可达到当按下键只做一次动作的目的，程序的结构、算法都正确无庸置疑，但由于当按下键都会有几毫秒的弹跳波，上述程序于按下键的第一个“0”电位就已经动作了而不管紧接而来的弹跳波，如何令按键动作稳定后（即等待弹跳波之后的信号），才执行按键功能？方法如下：

（1）按下键大约等待 10ms 的时间，以让弹跳波过去之后，才开始侦测 FgSelectKey 标志变量，等待 10ms 的方式为一按下键，就先设定 ScanKeyCounter 等于 10，而每将 ScankeyCounter 减 1 就延迟 1ms，当 ScanKeyCounter 等于 0 就已经作 10 次，而每一次延迟 1ms，所以作 10 次就已经延迟 10ms 了，当改变 ScanKeyCounter 的设定值即可修正要延迟多长的弹跳波，而 FgSelectKey 的清除时机一样是当释放选曲键时，并将清除 FgSelectKey 写在同一个 if...else 判断式的结构内，当 KeyBuffer 不等于 SELECT_KEY，就令 KeyBuffer 等于 NO_KEY（数值为 0X00）并清除 FgSelectkey。

（2）新增一个变量 KeyBuffer，用以判断是否第一次按键，当第一次按下键，则 KeyBuffer 等于 KeyData（按键值），如果一直压着，则 KeyBuffer 就会一直等于 KeyData，程序就会执行侦测 ScanKeyCounter 延迟弹跳波的循环，当释放键，则 KeyBuffer 要将其清除，这样，下一次按键才会又可以延迟弹跳波的目的。程序写法如下：

```
/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      //detect start key
      InputStart( );
      if ( KeyData == START_KEY )
      {
         Music(MusicNumber);
```

```
            DelayX10ms(200);
        }

        //sel play no#
        InputKey( );
        if ( KeyBuffer != KeyData )
        {
            KeyBuffer = KeyData;
            ScanKeyCounter = 10;
        }
        else if( ScanKeyCounter != 0 )
        {
            ScanKeyCounter--;
            DelayXlms(1);
        }
        else if ( KeyBuffer == SELECT_KEY )
        {
            if (FgSelectKey==0)
            {
                FgSelectKey = 1;
                Beep(1,17,10);

                MusicNumber++;
                if ( MusicNumber>9 )
                    MusicNumber=1;
            }
        }
        else
        {
            KeyBuffer   = NO_KEY;
            FgSelectKey = 0;
        }
    }
}
```

汇编程序如下：

1. main.a51

```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)
```

```
?PR?Main?MAIN       SEGMENT CODE
EXTRN     BIT (FgSelectKey)
EXTRN     IDATA (KeyData)
EXTRN     IDATA (KeyBuffer)
EXTRN     IDATA (ScanKeyCounter)
EXTRN     IDATA (MusicNumber)
EXTRN     CODE (PowerOnInit)
EXTRN     CODE (_DelayX10ms)
EXTRN     CODE (_DelayX1ms)
EXTRN     CODE (_Music)
EXTRN     CODE (InputKey)
EXTRN     CODE (InputStart)
EXTRN     CODE (_Beep)
EXTRN     CODE (?C_STARTUP)
PUBLIC    Main

RSEG  ?PR?Main?MAIN
USING     0
Main:
LCALL     PowerOnInit
?C0001:
LCALL     InputStart
MOV       R0,#KeyData
MOV       A,@R0
CJNE      A,#01H,?C0003
MOV       R0,#MusicNumber
MOV       A,@R0
MOV       R7,A
LCALL     _Music
MOV       R7,#0C8H
MOV       R6,#0CH
LCALL     _DelayX10ms
?C0003:
LCALL     InputKey
MOV       R0,#KeyData
MOV       A,@R0
MOV       R7,A
MOV       R0,#KeyBuffer
XRL       A,@R0
JZ    ?C0004
MOV       A,R7
MOV       @R0,A
MOV       R0,#ScanKeyCounter
MOV       @R0,#0AH
SJMP      ?C0001
```

```
?C0004:
 MOV       R0,#ScanKeyCounter
 MOV       A,@R0
 JZ   ?C0006
 DEC       @R0
 MOV       R7,#01H
 MOV       R6,#00H
 LCALL     _DelayX1ms
 SJMP      ?C0001
?C0006:
 MOV       R0,#KeyBuffer
 MOV       A,@R0
 CJNE      A,#02H,?C0008
 JB   FgSelectKey,?C0001
 SETB      FgSelectKey
 MOV       R7,#01H
 MOV       R6,#00H
 MOV       R5,#011H
 MOV       R3,#0AH
 LCALL     _Beep
 MOV       R0,#MusicNumber
 INC       @R0
 MOV       A,@R0
 SETB      C
 SUBB      A,#09H
 JC   ?C0001
 MOV       @R0,#01H
 SJMP      ?C0001
?C0008:
 CLR       A
 MOV       R0,#KeyBuffer
 MOV       @R0,A
 CLR       FgSelectKey
 SJMP      ?C0001
 RET

 END
```

7-1-4　消除弹跳波的方法独立成一个函数

1. 软件构建的思维与解决方法

在上述Main( )函数内的消除弹跳波方法移至input.c模块的InputHandler( )函数，

而在 Main( )函数只需调用 InputHandler( )函数即可。这样的话，在 Main( )函数里的程序结构会条理分明、容易阅读，如果要做任何有关于选曲键功能的修改、增加或删除操作，都可在 InputHandler( )函数中完成，函数名称也较名符其实，日后也较易维护、处理弹跳波的程序结构，需修改的地方如下：

1. 在 input.c 模块下新增 InputHandler( )函数，所以在 input.h 包括文件要做函数外部声明。

2. Main( )函数在反复循环内要调用 InputHandler( )函数。

3. 将上述章节在 Main( )函数内的弹跳波处理程序移至 InputHandler( )函数内。

7-1-5 持续按着键，以延时方式来继续执行动作

软件构建的思维与解决方法

上述 InputHandler( )函数利用 FgSelectKey 标志变量，按键后虽然一直压着仍只做一次的动作（MusicNumber 只增加 1，也只“哔”一声），如果持续按着键，隔一秒也要继续做，即 Music Number 每隔一秒要继续加 1，喇叭每隔一秒也要响叫一声，但要如何实现呢？其实也很简单：

（1）不能再判断 FgSelectKey=“0”才能做，做完后又设定 FgSelectKey 了，否则，压 100 年还是只做一次。

（2）最简单的是在按下选曲键的回路内增加延迟一秒的函数，则每当按下键，所有动作都处理完毕，最后延迟一秒后，也必须跳离函数，则 Main( )函数又继续调用（Main( )函数必须不断的侦测按键函数，以便有按键输入则立即反应），此时，KeyBuffer 一定会等于 SELECT_KEY（因为按键还未放开），则又继续响叫一声，MusicNumber 又增加 1，形成每间隔一秒就执行动作一次的功能。程序的写法如下：

```
/*****************************/
void InputHandler(void)
{
   //sel play no#
   InputKey( );

   if ( KeyBuffer != KeyData )
   {
      KeyBuffer = KeyData;
      ScanKeyCounter = 10;
   }
   else if( ScanKeyCounter != 0 )
   {
```

```
        ScanKeyCounter--;
        DelayX1ms(1);
    }
    else if ( KeyBuffer == SELECT_KEY )
    {
        Beep(1,17,10);

        MusicNumber++;
        if ( MusicNumber>9 )
            MusicNumber=1;

        DelayX10ms(100);
    }
    else
        KeyBuffer   = NO_KEY;
}
```

汇编程序如下：

1. input.a51

```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)

?PR?InputHandler?INPUT      SEGMENT CODE
?PR?InputKey?INPUT          SEGMENT CODE
?PR?InputStart?INPUT            SEGMENT CODE
EXTRN     IDATA (KeyData)
EXTRN     IDATA (KeyBuffer)
EXTRN     IDATA (ScanKeyCounter)
EXTRN     IDATA (MusicNumber)
EXTRN     CODE (_DelayX10ms)
EXTRN     CODE (_DelayX1ms)
EXTRN     CODE (_Beep)
PUBLIC    InputStart
PUBLIC    InputKey
PUBLIC    InputHandler

RSEG  ?PR?InputHandler?INPUT
USING     0
InputHandler:
LCALL     InputKey
```

```
MOV     R0,#KeyData
MOV     A,@R0
MOV     R7,A
MOV     R0,#KeyBuffer
XRL     A,@R0
JZ   ?C0001
MOV     A,R7
MOV     @R0,A
MOV     R0,#ScanKeyCounter
MOV     @R0,#0AH
RET
?C0001:
MOV     R0,#ScanKeyCounter
MOV     A,@R0
JZ   ?C0003
DEC     @R0
MOV     R7,#01H
MOV     R6,#00H
LCALL   _DelayX1ms
RET
?C0003:
MOV     R0,#KeyBuffer
MOV     A,@R0
CJNE    A,#02H,?C0005
MOV     R7,#01H
MOV     R6,#00H
MOV     R5,#011H
MOV     R3,#0AH
LCALL   _Beep
MOV     R0,#MusicNumber
INC     @R0
MOV     A,@R0
SETB    C
SUBB    A,#09H
JC   ?C0006
MOV     @R0,#01H
?C0006:
MOV     R7,#064H
MOV     R6,#00H
LCALL   _DelayX10ms
RET
?C0005:
CLR     A
MOV     R0,#KeyBuffer
MOV     @R0,A
```

```
?C0008:
 RET
 RSEG  ?PR?InputKey?INPUT
 USING    0
InputKey:
 JB   P1_3,?C0009
 MOV      R0,#KeyData
 MOV      @R0,#02H
 RET
?C0009:
 CLR      A
 MOV      R0,#KeyData
 MOV      @R0,A
?C0011:
 RET

 RSEG  ?PR?InputStart?INPUT
 USING    0
InputStart:
 JB   P3_4,?C0012
 MOV      R0,#KeyData
 MOV      @R0,#01H
 RET
?C0012:
 CLR      A
 MOV      R0,#KeyData
 MOV      @R0,A
?C0014:
 RET

 END
```

7-1-6　持续按着键，以 Timer 计时方式来继续执行动作

软件构建的思维与解决方法

上面以 DELAY 方式来继续执行按键动作，但也只能以固定时间来执行动作。按下第一次要久一点（例如：1 秒）才要继续执行，一秒之后以 0.5 秒就继续执行，则利用 DELAY 方式势必无法达到需求，而 DELAY 方式最大的缺点将形成扫描 Main( )函数的循环需要很久才会重复侦测一次。例如：以间隔一秒才执行按键动作一次，由于按键之后需 DELAY 一秒后才会回到 Main( )函数，因此，实际 Main( )函数变成一秒钟才扫描一次，失去实时侦测外界反应而实时处理的能力，也容易无

法捕捉外界变化而造成错误的状况，那要如何改善呢？可实时变化、可实时反应、又可持续按键而且能继续执行按键的功能呢？想到了吗？没错，就是利用定时器中断的方式，具体方法如下：

（1）一定要加入 FgSelectKey 标志变量的判断、设定和清除的时机，否则，会很快地又执行按键的功能，而失去计时中断的目的。

（2）以 KeyCountTimer 在第一次按下键设定为一秒，由于 FgSelectKey 标志已经被设定了，所以会执行一秒的时间是否到了的判断式，当时间到了则计时变量改成 0.5 秒就执行动作一次。

（3）由于 InputHandler( )函数，当按下选曲键的处理程序中，并没有延迟函数的调用，所以很快又会回到主程序 Main( )函数，继续侦测外界的输入变化，达到实时侦测实时反应的能力。其程序写法如下：

```
/*****************************/
void InputHandler(void)
{
   //sel play no#
   InputKey( );

   if ( KeyBuffer != KeyData )
   {
      KeyBuffer = KeyData;
      ScanKeyCounter = 10;
   }
   else if( ScanKeyCounter != 0 )
   {
      ScanKeyCounter--;
      DelayX1ms(1);
   }
   else if ( KeyBuffer == SELECT_KEY )
   {
      if (FgSelectKey==0)
      {
         FgSelectKey = 1;
         Beep(1,17,10);

         MusicNumber++;
         if ( MusicNumber>9 )
            MusicNumber=1;

         KeyCountTimer = TIME_1SEC;
      }
      else if ( KeyCountTimer==0 )
```

```
        {
          KeyCountTimer = TIME_500MS;

          Beep(1,17,10);

          MusicNumber++;
          if ( MusicNumber>9 )
            MusicNumber=1;
        }
      }
      else
      {
        KeyBuffer   = NO_KEY;
        FgSelectKey = 0;
      }
    }
```

（4）由于是利用定时器的中断来设定时间，而定时器 1 已经作为音阶频率产生之用，所以便用定时器 0 以每 40ms 作为中断一次的最小时间单位，并在定时器 0 中断函数内，每 40ms 将计时变量减 1，即 KeyCountTimer 减 1，程序写法如下：

```
/*****************************/

//timer0,execute service route
/*****************************/
void  Timer0ISR ( void ) interrupt 1 using 3
{
    TL0 = CLOCK_40MS & 0xff;
    TH0 = CLOCK_40MS >> 8;
    TF0 = 0;

    //every 40ms,then KeyCountTimer-1
    if ( KeyCountTimer != 0 )
      KeyCountTimer--;
}
```

（5）定时器 0 以 40ms 中断一次，则 TH0、TL0 在初始化模块下的 InitialCpu( ) 函数要先将 40ms 的内容写入至 TH0、TL0 缓存器内，程序如下：

```
/*****************************/
void InitialCpu( void )
{
    IE  = 0;        //disable all interrupt
    PSW = 0;        //bank 0

    IP  = 0x0b;//hi priority:int0,timer0,timer1
```

```
    TMOD= 0x11;     //set timer1,timer0 mode

    TR0 = 1;        //start timer0:system time units
    TR1 = 0;        //stop timer1 :soundtone

    //CLOCK_40MS=(65536 - 40000)
    TL0 = CLOCK_40MS & 0xff;
    TH0 = CLOCK_40MS >> 8;

    EX0 = 0;        //disable int0 interrupt
    ET1 = 1;        //enable timer1 interrupt
    ET0 = 1;        //enable timer0 interrupt

    EA  = 1;        //enable all interrupt gate
}
```

（6）由于有使用到1秒、0.5秒、40ms的时间参数，因此，也要在define.h包括文件内定义常数，以方便程序阅读，如下：

```
#ifndef   _DEFINE_H
#define   _DEFINE_H

//declare
typedef   bit              Bit;
typedef   bit              Bool;
typedef   unsigned char    Byte;
typedef   unsigned int     Word;
typedef   unsigned long    Long;

#define   DATA             data
#define   IDATA            idata
#define   PDATA            pdata
#define   XDATA            xdata
#define   RDATA            code

#define   TIME_BASE        40
#define   TIME_1SEC        ( 1000/TIME_BASE)
#define   TIME_500MS       (  500/TIME_BASE)
#define   CLOCK_BASE       1
#define   CLOCK_40MS       (65536 - 40000*CLOCK_BASE)

#endif
```

汇编程序如下：

1. input.a51

```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)

?PR?InputHandler?INPUT      SEGMENT CODE
?PR?InputKey?INPUT          SEGMENT CODE
?PR?InputStart?INPUT        SEGMENT CODE
 EXTRN     BIT (FgSelectKey)
 EXTRN     IDATA (KeyData)
 EXTRN     IDATA (KeyBuffer)
 EXTRN     IDATA (ScanKeyCounter)
 EXTRN     IDATA (KeyCountTimer)
 EXTRN     IDATA (MusicNumber)
 EXTRN     CODE (_DelayX1ms)
 EXTRN     CODE (_Beep)
 PUBLIC    InputStart
 PUBLIC    InputKey
 PUBLIC    InputHandler

 RSEG  ?PR?InputHandler?INPUT
 USING     0
InputHandler:
 LCALL     InputKey
 MOV       R0,#KeyData
 MOV       A,@R0
 MOV       R7,A
 MOV       R0,#KeyBuffer
 XRL       A,@R0
 JZ   ?C0001
 MOV       A,R7
 MOV       @R0,A
 MOV       R0,#ScanKeyCounter
 MOV       @R0,#0AH
 RET
?C0001:
 MOV       R0,#ScanKeyCounter
 MOV       A,@R0
 JZ   ?C0003
 DEC       @R0
 MOV       R7,#01H
 MOV       R6,#00H
```

```
LCALL   _DelayX1ms
RET
?C0003:
MOV     R0,#KeyBuffer
MOV     A,@R0
XRL     A,#02H
JNZ     ?C0005
JB   FgSelectKey,?C0006
SETB    FgSelectKey
MOV     R7,#01H
MOV     R6,A
MOV     R5,#011H
MOV     R3,#0AH
LCALL   _Beep
MOV     R0,#MusicNumber
INC     @R0
MOV     A,@R0
SETB    C
SUBB    A,#09H
JC   ?C0007
MOV     @R0,#01H
?C0007:
MOV     R0,#KeyCountTimer
MOV     @R0,#019H
RET
?C0006:
MOV     R0,#KeyCountTimer
MOV     A,@R0
JNZ     ?C0012
MOV     @R0,#0CH
MOV     R7,#01H
MOV     R6,A
MOV     R5,#011H
MOV     R3,#0AH
LCALL   _Beep
MOV     R0,#MusicNumber
INC     @R0
MOV     A,@R0
SETB    C
SUBB    A,#09H
JC   ?C0012
MOV     @R0,#01H
RET
?C0005:
CLR     A
```

```
MOV       R0,#KeyBuffer
MOV       @R0,A
CLR       FgSelectKey
?C0012:
RET

RSEG  ?PR?InputKey?INPUT
USING     0
InputKey:
JB   P1_3,?C0013
MOV       R0,#KeyData
MOV       @R0,#02H
RET
?C0013:
CLR       A
MOV       R0,#KeyData
MOV       @R0,A
?C0015:
RET

RSEG  ?PR?InputStart?INPUT
USING     0
InputStart:
JB   P3_4,?C0016
MOV       R0,#KeyData
MOV       @R0,#01H
RET
?C0016:
CLR       A
MOV       R0,#KeyData
MOV       @R0,A
?C0018:
RET

END
```

2. timer.a51

```
$NOMOD51

NAME TIMER

$INCLUDE (mtv212m.inc)

?PR?Timer0ISR?TIMER          SEGMENT CODE
```

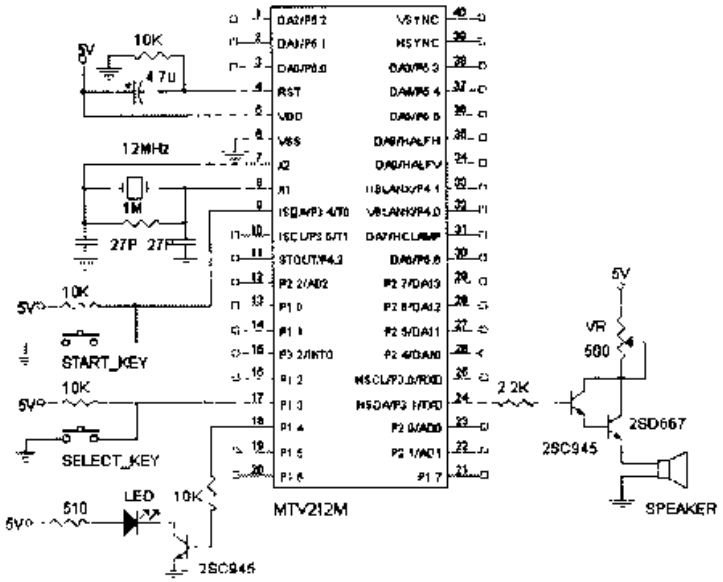
```
?PR?Timer1ISR?TIMER          SEGMENT CODE
 EXTRN     IDATA (KeyCountTimer)
 EXTRN     IDATA (Period)
 EXTRN     IDATA (SoundLongCount)
 PUBLIC    Timer1ISR
 PUBLIC    Timer0ISR

CSEG AT   0000BH
 LJMP Timer0ISR

 RSEG  ?PR?Timer0ISR?TIMER
 USING     3
Timer0ISR:
 PUSH      ACC
 PUSH      PSW
 MOV       PSW,#018H
 MOV       TL0,#0C0H
 MOV       TH0,#063H
 CLR       TF0
 MOV       R0,#KeyCountTimer
 MOV       A,@R0
 JZ    ?C0002
 DEC       @R0
?C0002:
 POP       PSW
 POP       ACC
 RETI

CSEG AT   0001BH
 LJMP Timer1ISR


 RSEG  ?PR?Timer1ISR?TIMER
 USING     2
Timer1ISR:
 PUSH      ACC
 PUSH      PSW
 MOV       PSW,#010H
 MOV       R0,#SoundLongCount+01H
 MOV       A,@R0
 DEC       R0
 ORL       A,@R0
 JZ    ?C0003
 INC       R0
 MOV       A,@R0
```

```
DEC       @R0
JNZ       ?C0003
DEC       R0
DEC       @R0
?C0003:
MOV       R0,#Period
MOV       A,@R0
MOV       R6,A
INC       R0
MOV       A,@R0
MOV       R7,A
CLR       C
CLR       A
SUBB A,R7
MOV       R7,A
CLR       A
SUBB      A,R6
MOV       R6,A

CPL       P3_1

XCH       A,R5
MOV       A,R7
XCH       A,R5
MOV       A,R5
MOV       TL1,A
MOV       A,R6
MOV       TH1,A
CLR       TF1
POP       PSW
POP       ACC
RETI

END
```

7-2　以 LED 闪烁作为按键输入提示

目的：每当按下选曲键时，将 LED 闪一下，以便识别输入的动作。

电路图：



原理： 多加了一支 I/O P1.4 来控制 LED 亮或熄，当 I/O 输出“1”电位，晶体管 2SC945 导通，则电流将流经 LED 至地，使得 LED 亮。而 510Ω的电阻为限流电阻，为控制 LED 亮的程度，电阻大，则流经 LED 的电流小，则 LED 较暗；电阻小，则流经 LED 的电流就会提高，所以 LED 会较亮。而当 I/O P1.4 输出为“0”电位，晶体管为截止状态，所以 LED 熄灭，其余起始键、选曲键和 7-1 节一样，此处不再赘述。

模块化： 在 7-1 节中的 beep.c 模块改成 led.c 模块，而 led.c 模块内容包含 led 设定、清除和闪烁的函数，及其对应的包括文件 led.h 便是将 led.c 模块的函数作外部声明，其余和 7-1 节一样。

软件构建的思维与解决方法

Main( )函数将 7-1 节中的调用 beep( )函数转换成 LedFlash( )函数，其余程序结构和 7-1 节一样，也就是将按下键，其提示由“哔”声变成 LED 闪烁，因此，有关于此的程序都应作修正，如下：

（1）main.c：当按下 SELECT_KEY 的条件成立下，要调用 LED 闪烁函数，程序如下：

```
/*******************************/
void Main( void )
{
   PowerOnInit();
```

```
    while( 1 )
    {
      //detect start key
      InputStart( );
      if ( KeyData == START_KEY )
      {
        Music(MusicNumber);
        DelayX10ms(200);
      }

      //sel play no#
      InputKey( );
      if ( KeyData == SELECT_KEY )
      {
        LedFlash(1,50,50);

        MusicNumber++;
        if ( MusicNumber>9 )
          MusicNumber=1;
      }
    }
}
```

（2）led.c：新增一个 led.c 的模块，要包含 LED 闪烁的函数，其中又将 LED 亮和 LED 熄也写成函数，以便阅读和易于改变 I/O 地址，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "led.h"

/*****************************/
void LedFlash(Byte count,Byte ontime,Byte offtime)
{
   Byte  i;

   for(i=0; i<count; i++)
   {
```

```
        LedOn( );
        DelayX10ms(ontime );
        LedOff( );
        DelayX10ms(offtime);
    }
}

/*****************************/
void LedOn(void)
{
    LED_IO=1;
}
void LedOff(void)
{
    LED_IO=0;
}
```

（3）led.h：除了将 led.c 模块下的函数作外部声明外，必须定义控制 LED 输出的 I/O，如下：

```
#ifndef   _LED_H
#define   _LED_H

#define  LED_IO          P1_4

extern void LedFlash(Byte count,Byte ontime,Byte offtime);
extern void LedOn(void);
extern void LedOff(void);

#endif
```

7-3 以“哔”声和 LED 闪烁顺序动作以作为按键输入提示

目的：每当按下选曲键时，除了“哔”声后 LED 也会闪烁一下，以便识别输入的动作。

模块化：由于按键后要发出“哔”声和 LED 闪烁，因此 beep.c 模块和 led.c 模块都必须加入至项目内，而其对应的包括文件 beep.h 和 led.h，在程序前端都必须用 #include 指令包括进来。

软件构建的思维与解决方法

将提示由单纯的“哔”声或“LED 闪烁”转换成发出“哔”声后再令 LED 闪烁，因此，其函数调用的顺序为 beep( )函数后再调用 LedFlash( )函数，如果为先 LED 闪烁再发出“哔”声，则函数调用的顺序就要改为：先调用 LedFlash( )函数后再调用 beep( )函数，在整个程序要修改的部分只有 Main( )函数了，程序写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "led.h"

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      //detect start key
      InputStart( );
      if ( KeyData == START_KEY )
      {
         Music(MusicNumber);
         DelayX10ms(200);
      }

      //sel play no#
      InputKey( );
      if ( KeyData == SELECT_KEY )
      {
         Beep(1,17,10);
         LedFlash(1,50,50);

         MusicNumber++;
         if ( MusicNumber>9 )
```

```
                MusicNumber=1;
            }
        }
    }
```

7-4 以“哔”声和 LED 闪烁同时动作作为按键输入提示

目的：每当按下选曲键时，以“哔”声和 LED 闪烁同时动作，以便识别输入的动作。

软件构建的思维与解决方法

由于发出“哔”声和 LED 闪烁要同时操作，因此，将 LED 的认定、清除写在“哔”声函数内较容易，也不会失去原来的程序结构，如果将“哔”声写在 LED 闪烁函数内，真不知要如何达成？而且大费周章反而会将程序结构变得更复杂，如果是你，你将会如何设计呢？解决方法如下：

1. main.c

按下选曲键的条件成立后，便调用“哔声和 LED 闪烁”的函数，名称为 BeepLed( ) 函数，其程序写法如下：

```
/*****************************/
void Main( void )
{
    PowerOnInit();

    while( 1 )
    {
      //detect start key
      InputStart( );
      if ( KeyData == START_KEY )
      {
        Music(MusicNumber);
        DelayX10ms(200);
      }

      //sel play no#
      InputKey( );
```

```
        if ( KeyData == SELECT_KEY )
        {
           BeepLed(1,17,10);

           MusicNumber++;
           if ( MusicNumber>9 )
              MusicNumber=1;
        }
     }
}
```

2. beep.c

原 beep( )函数的结构不变，在每次要开始发出“哔”声之前先设定 Led 亮，直到“哔”声结束后，再令 LED 熄，形成每发出“哔”声 LED 就快速闪烁一下，如果调用 Ledoff( )函数，写在 DelayX10ms( )函数之后，则 LED 将连续发出 count 自变量的“哔”声才会熄灭，其程序写法如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "led.h"

/*****************************/
void BeepLed(Word count,Byte soundlong,Byte tone)
{
   Word  i,j,k,spfreq;

   spfreq  = (1000/tone)/2; //Khz unit

   for(i=0; i<count; i++)
   {
      LedOn( );
      for(j=0; j<soundlong; j++)
      {
           SPEAKER = 1;
        for(k=0; k<spfreq; k++)
```

```
            ;
        SPEAKER = 0;
        for(k=0; k<spfreq; k++)
            ;
    }
    LedOff( );
    DelayX10ms(12);
  }
}
```

3. led.c

led.c模块只由简单的LED亮或熄的函数组成，以便给beep.c模块下的BeepLed( )函数调用，其程序如下：

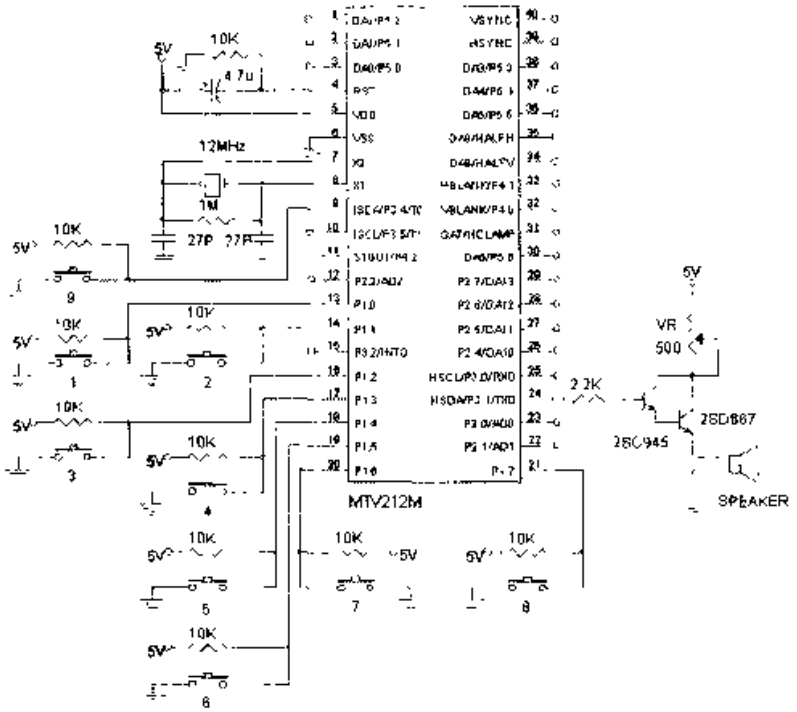
```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "led.h"

/*****************************/
void LedOn(void)
{
   LED_IO=1;
}
void LedOff(void)
{
   LED_IO=0;
}
```

第 8 章　九个按键开关的 1～9 首选曲

8-1　I/O 一对一的方式

目的：编号 1～9 的按键由个别的 I/O 作为选曲键，当按下键立即作演奏。

电路图：



原理：曲目编号从 1～9，因而需要 9 个按键，分别代表曲目编号为 1～9 的按键，当按下数值 1 的按键，即表示演奏第一首曲目，按下数值 2 的按键，即表示演奏第二首，依此类推，而 9 个按键的接法都是将 10kΩ的电阻提升至+5V，而开关一端接点接地，而另一端则和电阻的另一端共同接至 I/O 的输入引脚，当按下数值键则 I/O 为“0”电位，未按下数值键则 I/O 一直保持在“1”电位。

8-1-1　软件构建的思维

只要按下数值键，则立即根据据此数值作演奏，因而，在 Main( )函数的反复循环内，只需判断当数值不等于 0（平常未按下任意键，其数值必为 0）就执行演奏函

数，而在侦测按键的函数必须顺序地侦测数值 1～9 的按键，当按下键，则将键值写入变量内，否则必须将数值 0X00 写入变量内，现分述如下。

1. main.c

要有不断侦测按键的结构，以便当按键之后，可以立即接受按键值而立即演奏的功能，当按键之后，需要将键值存入曲目编号变量内，也依此变量当作自变量传入至演奏音乐函数。

2. input.c

以 if...else 的指令结构为基础，使用 if 的条件判断式来逐一测试每一个按键的 I/O，当 I/O 输入信号变为“0”电位，表示已按下键，则应将键值存入至 KeyData 变量内，如果 1～9 按键的 I/O 都为“1”电位，则须令 KeyData 变量存入 0X00 的数值，以作为 Main( )函数判断是否按键的依据。

3. input.h

每一个按键都应定义其输入 I/O 的引脚及其键值，键值也都要不一样才可以。

8-1-2 参数的意义说明与使用时机

KeyData：按键后其对应的数值。

- 数据类型：占用一个字节的无符号字符类型(unsigned char)。
- 内存类型：变量地址介于 0X00～0Xff 之间，使用间接寻址模式(idata)。
- 清除：在初始化模块内执行，使其侦测按键前，其数值就为 0X00。其实，也可以不用先初始化，因为在侦测按键函数，当都未按下键时，则变量内容就是为 0X00。
- 写入：按下键则须将对应的数值写入，以作为曲目编号，如果所有曲目键都未按下，须写入 0X00，以便区别。
- 判断：在 Main( )函数反复循环内，调用按键侦测程序后，立即判断是否有键按下？有，则将键值写入至曲目编号变量，否则，继续反复做循环，即继续做按键侦测。

8-1-3 软件的解决方法

由 I/O 控制按键的输入是最基本、最简单也是最实用的一种方式。如果按键很多，则将使得 CPU 的 I/O 引脚不敷使用，如果 CPU 的 I/O 有多余的话，则可作为按键输入当然最好。在设计之初，请多花一点时间作规划、作评量，其硬件结构要如何？再跟据硬件而作整体性考虑的软件规划，以一个 I/O 对应一个按键的方式，要能够很迅速地侦测按键的输入以及正确地写入其按键数值即可，分述如下：

1. main.c：主程序主要是不断地侦测按键程序，当输入按键与未输入按键要能够清楚地区分出来。以一个 if 指令即可达成，另外，必须将按键值(KeyData)写入至曲目编号变量内(MusicNumber)，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      //sel play no#
      InputKey( );
      if ( KeyData !=NO_KEY )
      {
         MusicNumber=KeyData;
         Music(MusicNumber);
         DelayX10ms(100);
      }
   }
}
```

2. input.c：input.c 模块只需要一个函数，即侦测按键的函数：inputkey( )，并顺序从按键 1 一直侦测到按键 9，当按下键则将键值写入 KeyData 变量内，如果都没有按下键，才将数值 0X00 写入至 KeyData 变量，而 NO_KEY 常数即为 0X00，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
```

```
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*****************************/
void InputKey(void)
{
    if ( NUMBER1_IO==0 )
      KeyData=NUMBER1_KEY;
    else if ( NUMBER2_IO==0 )
      KeyData=NUMBER2_KEY;
    else if ( NUMBER3_IO==0 )
      KeyData=NUMBER3_KEY;
    else if ( NUMBER4_IO==0 )
      KeyData=NUMBER4_KEY;
    else if ( NUMBER5_IO==0 )
      KeyData=NUMBER5_KEY;
    else if ( NUMBER6_IO==0 )
      KeyData=NUMBER6_KEY;
    else if ( NUMBER7_IO==0 )
      KeyData=NUMBER7_KEY;
    else if ( NUMBER8_IO==0 )
      KeyData=NUMBER8_KEY;
    else if ( NUMBER9_IO==0 )
      KeyData=NUMBER9_KEY;
    else
      KeyData=NO_KEY;
}
```

3. input.h：定义按键 1～按键 9 的 I/O 地址及其按键数值，分别如下：

按键名	I/O	按键数值
按键 1	P1.0	1
按键 2	P1.1	2
按键 3	P1.2	3
按键 4	P1.3	4
按键 5	P1.4	5
按键 6	P1.5	6
按键 7	P1.6	7
按键 8	P1.7	8
按键 9	P3.4	9

而按键数值就是等于曲目编号，包括文件如下：

```
#ifndef   _INPUT_H
#define   _INPUT_H

#define  NUMBER1_IO     P1_0
#define  NUMBER2_IO     P1_1
#define  NUMBER3_IO     P1_2
#define  NUMBER4_IO     P1_3
#define  NUMBER5_IO     P1_4
#define  NUMBER6_IO     P1_5
#define  NUMBER7_IO     P1_6
#define  NUMBER8_IO     P1_7
#define  NUMBER9_IO     P3_4

#define  NO_KEY              0
#define  NUMBER1_KEY    1
#define  NUMBER2_KEY    2
#define  NUMBER3_KEY    3
#define  NUMBER4_KEY    4
#define  NUMBER5_KEY    5
#define  NUMBER6_KEY    6
#define  NUMBER7_KEY    7
#define  NUMBER8_KEY    8
#define  NUMBER9_KEY    9

extern void InputKey(void);

#endif
```

汇编程序如下：

1. main.a51

```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN                SEGMENT CODE
EXTRN     IDATA (KeyData)
EXTRN     IDATA (MusicNumber)
EXTRN     CODE (PowerOnInit)
EXTRN     CODE (_DelayX10ms)
EXTRN     CODE (_Music)
EXTRN     CODE (InputKey)
```

```
EXTRN     CODE (?C_STARTUP)
PUBLIC    Main

RSEG  ?PR?Main?MAIN
USING     0
Main:
LCALL     PowerOnInit
?C0001:
LCALL     InputKey
MOV       R0,#KeyData
MOV       A,@R0
MOV       R7,A
JZ    ?C0001
MOV       R0,#MusicNumber
MOV       @R0,A
LCALL     _Music
MOV       R7,#064H
MOV       R6,#00H
LCALL     _DelayX10ms
SJMP      ?C0001
RET

END
```

2. input.a51

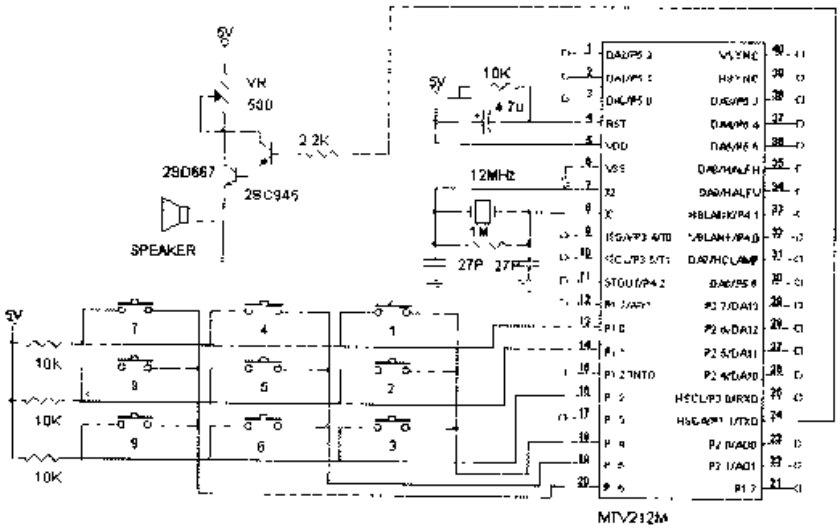
```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT          SEGMENT CODE
EXTRN     IDATA (KeyData)
PUBLIC    InputKey

RSEG  ?PR?InputKey?INPUT
USING     0
InputKey:
JB    P1_0,?C0001
MOV       R0,#KeyData
MOV       @R0,#01H
RET
?C0001:
JB    P1_1,?C0003
MOV       R0,#KeyData
```

```
MOV       @R0,#02H
RET
?C0003:
JB   P1_2,?C0005
MOV       R0,#KeyData
MOV       @R0,#03H
RET
?C0005:
JB   P1_3,?C0007
MOV       R0,#KeyData
MOV       @R0,#04H
RET
?C0007:
JB   P1_4,?C0009
MOV       R0,#KeyData
MOV       @R0,#05H
RET
?C0009:
JB   P1_5,?C0011
MOV       R0,#KeyData
MOV       @R0,#06H
RET
?C0011:
JB   P1_6,?C0013
MOV       R0,#KeyData
MOV       @R0,#07H
RET
?C0013:
JB   P1_7,?C0015
MOV       R0,#KeyData
MOV       @R0,#08H
RET
?C0015:
JB   P3_4,?C0017
MOV       R0,#KeyData
MOV       @R0,#09H
RET
?C0017:
CLR       A
MOV       R0,#KeyData
MOV       @R0,A
?C0019:
RET

END
```

8-2 SCAN 一对一的方式

目的： 以扫描键盘的方式来侦测编号 1～9 的选曲键。

电路图：



原理： 以矩阵的排列方式可以用较少的 I/O 侦测出较多的键盘个数，相对于一个按键就用一个 I/O 来侦测的方式，可大量节省 CPU I/O 的个数。例如：3×3 的矩阵，利用 3+3=6 个 I/O 可侦测出 3×3=9 个按键，4×4 的矩阵，利用 4+4=8 个 I/O 可侦测出 4×4=16 个按键，而扫描的原理如下：

1. 串接提升电阻 10kΩ 至+5V 的 I/O 引脚作为输入，例：P1.0、P1.1、P1.2 I/O 脚为输入引脚，P1.0 负责侦测数字键 1、4、7；P1.1 负责侦测数字键 2、5、8；P1.2 负责侦测数字键 3、6、9。

2. 接于“行”上的 I/O 脚作为输出（横线为列，纵线为行），也就是所谓的扫描线，例如：I/O P1.4、P1.5、P1.6、P1.4 I/O 脚负责数字键 1、2、3；P1.5 负责数字键 4、5、6；P1.6 负责数字键 7、8、9。

3. 由于 P1.0、P1.1、P1.2 接上提升电阻至+5V，其平常输入电位为“1”电位，所以扫描线的输出要为“0”电位，以便当按键压下去可将输入端由原来的“1”电位转变成“0”电位，从而侦测出哪一键被压下去。

4. 例如：当扫描线 P1.4 输出为“0”电位，而扫描线 P1.5 和 P1.6 都输出“1”电位（扫描线输出“1”电位，表示不作此行上键盘的侦测），则此时去侦测 I/O P1.0、P1.1 和 P1.2，当 I/O P1.0=“0”电位表示数字键 1 已被按下，I/O P1.1=“0”电位表示数字键 2 已被按下，而 I/O P1.2=“0”电位表示数字键 3 已被按下。

5. 同理，当扫描线 P1.5 输出为“0”电位，而扫描线 P1.4 和 P1.6 都输出“1”

电位，则当 P1.0=“0”电位表示数字键 4 被按下，P1.1=“0”电位表示数字键 5 被按下，P1.2=“0”电位表示数字键 6 被按下；当 P1.6 输出为“0”电位，而 P1.4 和 P1.5 都输出“1”电位，表示要侦测数字键 7、8、9 是否被按下，当 P1.0=“0”则数字键 7 被按下，P1.1=“0”，则数字键 8 被按下，P1.2=“0”，则数字键 9 被按下。

6. 依此方法，顺序地令扫描线使能（输出“0”电位），再分别侦测输入的 I/O 引脚，即可以轻易地判断数字键 1~9 了，即使将矩阵的范围扩大(8×8)，其侦测方法也是一样。

8-2-1　软件构建的思维

侦测按键最直接、最简单的是利用 CPU I/O 的方法，这样必须浪费较多的 I/O 引脚。而使用矩阵扫描可改善直接利用 I/O 方法的缺点，只是要注意扫描线输出脚和输入脚之间的关系，此范例主要是将侦测按键函数 InputKey( )的程序改写成扫描方式，其定义的数字键值和键值变量都无需改变，其他模块变量的存取和函数的调用都不变，分述如下：

1. initial.c：I/O P1.0、P1.1、P1.2 作为扫描键盘的输入脚，则先设定都为“1”电位，以便当扫描线输出“0”电位而同时也按下键就可以侦测出来。

2. input.c：在此模块下的按键侦测函数即是利用扫描方式，分别从扫描线 1～3 依序输出“0”电位，即可将数字 1～数字 9 顺序的侦测是否有按下，而在主程序 main( )函数不断地调用此按键侦测程序即可。

3. input.h：将扫描线 I/O 以宏定义之，以便程序阅读或日后修改方便，如果日后要修改只需改变扫描线的 I/O 定义，而无需在程序中或其他模块中一一作修正。

8-2-2　参数的意义说明与使用时机

- KeyData：根据扫描线动作下的按键数值变量。
- 数据类型：占用一个字节的无符号字符类型(unsigned char)。
- 内存类型：变量地址介于 0X00～0Xff 之间，使用间接寻址模式(idata)。
- 清除：

① 在 initial.c 模块下先初始化为 0。

② 扫描按键的侦测函数在程序一开始就先将其清除，表示未按下任意键，其按键变量为 0X00。

- 写入：当数字键 1～数字键 9 被侦测出按下时，则写入相对应的数值，数字键 1 写入 0X01，数字键 2 写入 0X02，数字键 3 写入 0X03，依此类推至数

字键 9 写入 0X09。

- 判断：当 KeyData 不等于 0X00，即表示数字键 1～数字键 9 其中一个按键已被按下，并将此按键值存入曲目编号变量。

8-2-3 软件的解决方法

扫描键盘当 I/O 作为输入时，要先行设定为“1”电位，并在 initial.c 模块下做此动作，而当按下键必须将此键值存入曲目编号变量 MusicNumber，并传入至音乐演奏 Music( )函数内，而行扫描线以定义 I/O 的方式，以便日后修改容易及程序阅读，需将完整的测试扫描键盘 Inputkey( )函数的正确性，现分述如下：

1. initial.c：将 I/O P1.0、P1.1、P1.2 设定为“1”电位，可个别设定或以 P1 端口作输出，当 P1.0、P1.1 及 P1.2 为“1”电位，则 P1 端口的数值为 1+2+4=7，在 InitialCpuIO( )函数内，其程序如下：

```
/*****************************/
void InitialCpuIO(void)
{
   P1=0x07;        //input:P1.0~P1.2
   SPEAKER = 0;
}
```

2. input.c：Inputkey( )侦测矩阵按键函数，其扫描按键的方法如下：

（1）先清除按键数值变量。

（2）令扫描线 1 使能，即 P1.4 输出“0”电位后，读取 I/O P1.0～P1.2 的内容。

（3）由于 I/O P1.0～P1.2 是“0”动作，所以反相后并屏蔽 bit3～bit7，写入至暂存变量 keytmp 内。

（4）利用 switch...case 指令来依序判断数字键，当 keytmp=1，即 P1.0=“0”，表示数字键 1 按下；keytmp=2，即 P1.1=“0”，表示数字键 2 按下；keytmp=4，即 P1.2=“0”表示数字键 3 按下。

（5）依此方法令扫描线 2 使能；即 P1.5 输出“0”电位及扫描线 3 使能，即 P1.6 输出“0”电位，而分别判断数字键 4～9。

（6）数字键 1～9 全部侦测完毕，则令扫描线 3（P1.6）恢复为“1”电位，使得平常 P1.4、P1.5、P1.6 都保持在“1”电位，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
```

```
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*****************************/
void InputKey(void)
{
   Byte  keytmp;

   KeyData = NO_KEY;

   COLUMN1_PIN = 0;     //scan line1:"0" active
   COLUMN2_PIN = 1;
   COLUMN3_PIN = 1;
   keytmp = ~(INPUT_PORT) & 0x07;    //P1.0~P1.2
   switch( keytmp )
   {
    case 1 :
      KeyData = NUMBER1_KEY;
      break;
    case 2 :
      KeyData = NUMBER2_KEY;
      break;
    case 4 :
      KeyData = NUMBER3_KEY;
      break;
    default :
      break;
   }

   COLUMN1_PIN = 1;
   COLUMN2_PIN = 0;     //scan line2:"0" active
   COLUMN3_PIN = 1;
   keytmp = ~(P1) & 0x07;
   switch( keytmp )
   {
    case 1 :
      KeyData = NUMBER4_KEY;
      break;
    case 2 :
      KeyData = NUMBER5_KEY;
      break;
    case 4 :
```

```
            KeyData = NUMBER6_KEY;
            break;
         default :
            break;
        }

        COLUMN1_PIN = 1;
        COLUMN2_PIN = 1;
        COLUMN3_PIN = 0;     //scan line3:"0" active
        keytmp = ~(P1) & 0x07;
        switch( keytmp )
        {
         case 1 :
            KeyData = NUMBER7_KEY;
            break;
         case 2 :
            KeyData = NUMBER8_KEY;
            break;
         case 4 :
            KeyData = NUMBER9_KEY;
            break;
         default :
            break;
        }
        COLUMN3_PIN = 1;     //scan line3:not active
    }
```

3. input.h：输入引脚是使用 P1.0、P1.1、P1.2，所以输入端口要定义为 P1，另外扫描线的输出定义 P1.4、P1.5 和 P1.6，以及按下数字键 1～9 的数值和定义 NO_KEY 未按下键的数值为 0X00，如下：

```
#ifndef   _INPUT_H
#define   _INPUT_H

#define  INPUT_PORT          P1
#define  COLUMN1_PIN     P1_4
#define  COLUMN2_PIN     P1_5
#define  COLUMN3_PIN     P1_6

#define  NO_KEY              0
#define  NUMBER1_KEY     1
#define  NUMBER2_KEY     2
#define  NUMBER3_KEY     3
#define  NUMBER4_KEY     4
#define  NUMBER5_KEY     5
```

```
#define  NUMBER6_KEY    6
#define  NUMBER7_KEY    7
#define  NUMBER8_KEY    8
#define  NUMBER9_KEY    9

extern void InputKey(void);

#endif
```

汇编程序如下：

1. input.a51

```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT          SEGMENT CODE
 EXTRN    IDATA (KeyData)
 PUBLIC   InputKey

 RSEG  ?PR?InputKey?INPUT
 USING    0
InputKey:
 CLR      A
 MOV      R0,#KeyData
 MOV      @R0,A
 CLR      P1_4
 SETB     P1_5
 SETB     P1_6
 MOV      A,P1
 CPL      A
 ANL      A,#07H
 MOV      R7,A
 ADD      A,#0FEH
 JZ   ?C0003
 ADD      A,#0FEH
 JZ   ?C0004
 ADD      A,#03H
 JNZ      ?C0001
?C0002:
 MOV      R0,#KeyData
 MOV      @R0,#01H
```

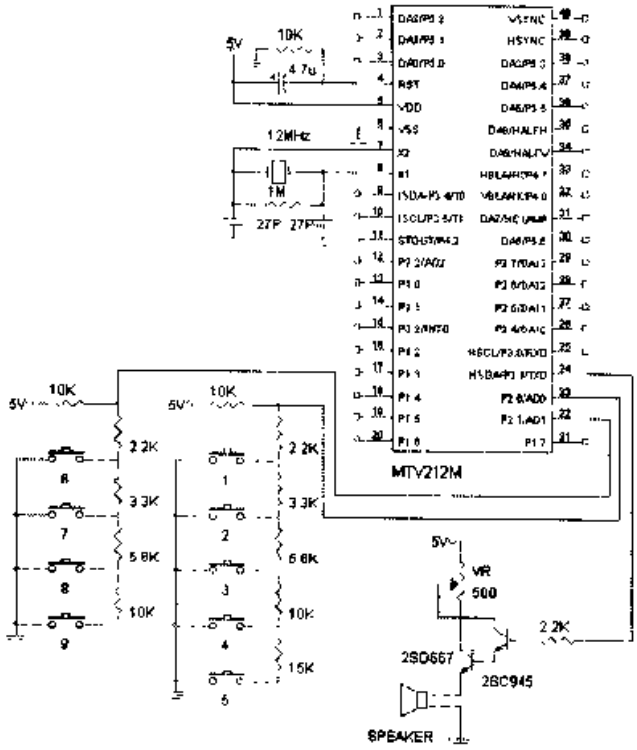
```
SJMP    ?C0001
?C0003:
MOV     R0,#KeyData
MOV     @R0,#02H
SJMP    ?C0001
?C0004:
MOV     R0,#KeyData
MOV     @R0,#03H
?C0001:
SETB    P1_4
CLR     P1_5
SETB    P1_6
MOV     A,P1
CPL     A
ANL     A,#07H
MOV     R7,A
ADD     A,#0FEH
JZ  ?C0008
ADD     A,#0FEH
JZ  ?C0009
ADD     A,#03H
JNZ     ?C0006
?C0007:
MOV     R0,#KeyData
MOV     @R0,#04H
SJMP    ?C0006
?C0008:
MOV     R0,#KeyData
MOV     @R0,#05H
SJMP    ?C0006
?C0009:
MOV     R0,#KeyData
MOV     @R0,#06H
?C0006:
SETB    P1_4
SETB    P1_5
CLR     P1_6
MOV     A,P1
CPL     A
ANL     A,#07H
MOV     R7,A
ADD     A,#0FEH
JZ  ?C0013
ADD     A,#0FEH
JZ  ?C0014
```

```
ADD       A,#03H
JNZ       ?C0011
?C0012:
MOV       R0,#KeyData
MOV       @R0,#07H
SJMP      ?C0011
?C0013:
MOV       R0,#KeyData
MOV       @R0,#08H
SJMP      ?C0011
?C0014:
MOV       R0,#KeyData
MOV       @R0,#09H
?C0011:
SETB      P1_6
RET

END
```

8-3　ADC 一对一的方式

目的：以 ADC 方式来侦测编号 1~9 的选曲键。

电路图：



原理： 利用电阻分压而取得不同直流电压值并输入到模拟至数字转换器内（ADC），以判别哪一个数字键被按下，MTV212MN32 或 MTV212MN64 的 CPU，其 ADC 的分辨率为 6 bit，其转换后的数值范围介于 0~63 之间，共 64 阶，其 ADC 输入的最大电压为+5V，所以其分辨率为+5V/64 阶=78.125mv，因此，当输入至 ADC 的直流准位不同，其转换后数值也跟着不同。而每一个按键按下后会被接地，形成电阻分压的回路，其电压的分布尽量平均分配，以便侦测，以此达到判定的依据。

当数字键 1 被按下时，其输入至 AD0 的电压为：

5V×(2.2K/(10K+2.2K))=0.901V，其对应数值为 901/78.125=11

当数字键 2 被按下时，其输入至 AD0 的电压为：

5V×((2.2K+3.3K)/(10K+2.2K+3.3K))=1.774V，其对应数值为 1774/78.125=22

当数字键 3 被按下时，其输入至 AD0 的电压为：

5V×((2.2K+3.3K+5.6K)/(10K+2.2K+3.3K+5.6K))=2.630V，其对应的数值为 2630/78.125=33

当数字键 4 被按下时，其输入至 AD0 的电压为：

5V×((2.2K+3.3K+5.6K+10K)/(10K+2.2K+3.3K+5.6K+10K))=3.392V，其对应的数为 3392/78.125=43

当数字键 5 被按下时，其输入至 AD1 的电压为：

5V×((2.2K+3.3K+5.6K+10K+15K)/(10K+2.2K+3.3K+5.6K+10K+15K))=3.915V，其对应的数值为 3915/78.125=50

当数字键 6 被按下时，其输入至 AD1 的电压为：

5V×(2.2K/(10K+2.2K))=0.901V，其对应数值为 901/78.125=11

当数字键 7 被按下时，其输入至 AD0 的电压为：

5V×((2.2K+3.3K)/(10K+2.2K+3.3K))=1.774V，其对应数值为 1774/78.125=22

当数字键 8 被按下时，其输入至 AD1 的电压为：

5V×((2.2K+3.3K+5.6K)/(10K+2.2K+3.3K+5.6K))=2.630V，其对应数值为 2630/78.125=33

当数字键 9 被按下时，其输入至 AD1 的电压为：

5V×((2.2K+3.3K+5.6K+10K)/(10K+2.2K+3.3K+5.6K+10K))=3.392V，其对应数值为 3392/78.125=43

数字键 1～5 由 AD0 作输入，而其转换后的数值也不相同，所以可以被侦测出来，而数字键 6～9 其转换的数值虽然和数字键 1～4 相同，但其由 AD1 作输入，所以还是可以被侦测出来，而由 ADC 作输入只需 2 个输入端即可侦测出 9 个按键，其实，也可以只有一个 ADC 作输入，而侦测出 9 个按键，但分压电阻必须重新配

过，其每个按键的直流电压值应尽量平均即可。

8-3-1　软件构建的思维

要利用 mtv212MN CPU 内建的 ADC 输入，ADC 控制缓存器为外部数据存储器，其地址为 0X10，所以必须在 global.c 模块下以 pdata 的数据类型进行声明，并指定地址为 0X10，而其他的外部数据记忆也一并作声明，并在 initial.c 模块作初始化的设定，而侦测函数必须改写以 ADC 方式作侦测是否按键，当侦测到按键后其写入按键数值的变量 KeyData 无需改变，只是在 Inputkey( )侦测函数其写入的时机不同而已，现分述如下：

1. global.c：mtv212MN CPU 外部数据存储器要作声明，并以_at_指令指定其地址。

2. global.h：外部数据存储器也要以 extern 作外部声明，以便其他模块存取。

3. initial.c：外部数据存储器也要作初始化的设定。

4. delay.c：增加一个 DelayX1ms( )函数，以便给 ADC 侦测函数使用，当选择 AD0 或 AD1 的输入时，要等待一点点时间，以让 ADC 确实转换完成后再判断其数值。

5. delay.h：新增的函数作外部声明。

6. input.c：ADC 侦测函数规划至此模块下，而侦测函数名称不变，一样是 InputKey( )函数，所以其他模块下的调用函数也可以不用改变，而按键数值变量一样是 KeyData，则任何存取、判断此变量的模块都不用改变，只是程序内容以判断 ADC 转换后的数值大小来决定哪一键被按下，当侦测完毕也必须将 ADC 缓存器除能。

7. input.h：单纯的定义按键 1～9 的数值及作函数的外部声明。

8-3-2　变量的意义与使用时机

- XFR_ADC：外部数据存储器，地址为 0X10。
- 数据类型：占用一个字节的无符号字符类型(unsigned char)。
- 内存类型：变量地址介于 0X00～0Xff 之间，使用外部数据存储器(pdata)。
- 写入：

（1）在 initial.c 模块下作初始化设定。

（2）侦测函数时 ADC 频道的输入切换，侦测数字键 1～5 要切至 AD0，侦测数字键 6～9 则要切至 AD1。

- 读取：转换后的数值也是存入至此缓存器，其数值范围为 0X00～0X3f。

- 清除：侦测完毕应除能 ADC 缓存器，即将其清除。

8-3-3 软件的解决方法

基本上按键的侦测是以输入至 ADC 的直流电压为依据，因此，不管一个 ADC 输入端负责几个按键，都应尽量保持平均分配的电压输入。例如：侦测四个按键，可设计成按下第一个按键，其输入电压为 1V，按下第二个按键其输入电压为 2V，按下第三个按键其输入电压为 3V，按下第四个按键其输入电压为 4V 等；如果要侦测八个按键，其每个按键输入时的电压间隔为 0.5V，即 0.5V、1V、1.5V、2V、2.5V、3V、3.5V、4V 共八个按键。另外，使用两个以上的 ADC 输入，必须先选择 ADC 输入端并使能 ADC 开始转换后，待转换完成才能读取其数值作判断，现分述如下：

1. global.c：除了声明 ADC 缓存器外，也一并将外部数据存储器进行声明，以便读者参考。请注意地址都介于 0X00～0Xff 之间，所以使用 pdata 作声明，并使用 _at_ 指令指定其绝对地址，如果不了解之处，请参考“8051 单片机 C 语言开发环境实务与设计”一书的第 11 章。如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"

/*****************************/
// Global Data
/*****************************/

//key
Byte      IDATA   KeyData;

//music
Byte      IDATA   MusicNumber;
Word      IDATA   Tempo;
Word      IDATA   Period;
Word      IDATA   SoundLongCount;

pdata unsigned char XFR_ADC          _at_        0x10;
pdata unsigned char XFR_WDT          _at_        0x18;
pdata unsigned char XFR_PADMOD1      _at_        0x30;
pdata unsigned char XFR_PADMOD2      _at_        0x31;
pdata unsigned char XFR_PADMOD3      _at_        0x32;
pdata unsigned char XFR_OPTION1      _at_        0x33;
```

```
pdata unsigned char XFR_OPTION2      _at_          0x34;
pdata unsigned char XFR_XBANK        _at_          0x35;
```

2. global.h：外部数据存储器也需要作外部声明，如下：

```
#ifndef   _GLOBAL_H
#define   _GLOBAL_H

//key
extern Byte   IDATA   KeyData;

//music
extern Byte   IDATA   MusicNumber;
extern Word   IDATA   Tempo;
extern Word   IDATA   Period;
extern Word   IDATA   SoundLongCount;

extern pdata unsigned char XFR_ADC;
extern pdata unsigned char XFR_WDT;
extern pdata unsigned char XFR_PADMOD1;
extern pdata unsigned char XFR_PADMOD2;
extern pdata unsigned char XFR_PADMOD3;
extern pdata unsigned char XFR_OPTION1;
extern pdata unsigned char XFR_OPTION2;
extern pdata unsigned char XFR_XBANK;

#endif
```

3. initial.c：在 InitialCpu( )函数将外部数据存储器缓存器作初始化设定，要根据 CPU 数据手册和硬件需求进行设定，如下：

```
/*****************************/

void InitialCpu( void )
{
    IE  = 0;                        //disable all interrupt
    PSW = 0;                        //bank 0

    IP = 0x0b;                      //hi priority:int0,timer0,timer1

    TMOD= 0x11;                     //set timer1,timer0 mode

    TR0 = 0;                        //stop timer0
    TR1 = 0;                        //stop timer1
    IT0 = 1;                        //set int0:falling eage trigger
```

```
    EX0 = 0;                        //disable int0 interrupt
    ET1 = 1;                        //enable timer1 interrupt
    ET0 = 0;                        //disable timer0 interrupt

    //Initial XFR's and DAC's for MTV212MNXX
    XFR_ADC=0x80;             //enable ADC,not select ch input
    XFR_WDT=0x40;             //clear watch dog timer,2s interval

    XFR_PADMOD1     =0x07;    //set adc0,adc1,adc2,i/o:P2.4~P2.7
    XFR_PADMOD2     =0x00;    //set dac0~dac6
    XFR_PADMOD3     =0x00;    //set dac,h,v
    XFR_OPTION1     =0xc0;    //94k pwm=1,div253=1
    XFR_OPTION2     =0x00;    //7 bit slave address for iic
    XFR_XBANK       =0x00;    //ram bank0

    EA  = 1;                        //enable all interrupt gate
}
```

4. delay.c：新增 DelayX1ms( )函数，以便给 InputKey( )侦测 ADC 输入的按键函数使用，用以等待 ADC 转换完成而作的延迟，如果不作短暂延迟，则将无法读取到正确的数值而造成错误，函数如下：

```
/*****************************/
void DelayX1ms(Word count)
{
    Word i,j;

    for(i=0; i<count; i++)
       for(j=0; j<120; j++)
          ;
}
```

5. delay.h：将 DelayX1ms( )函数作外部声明，如下：

```
#ifndef   _DELAY_H
#define   _DELAY_H

extern void DelayX10ms(Word count);
extern void DelayX1ms(Word count);

#endif
```

6. input.c：利用 ADC 方式作按键的侦测输入，九个按键也可以只使用一个 ADC 作输入（当 CPU 内建的 ADC 只有一个时），其电阻值要重新配置，而无需浪费一个 I/O 脚，其程序结构简单，条理分明，是最有效率的设计方法。另外 ADC 还有其

他的应用，例如：利用可调电阻作为时间调整等，本范例是利用 ADC 作为按键输入的侦测，而且是使用两个 ADC，分别是 AD0（动作数值为 0X81），AD1（动作数值为 0X82），需等待转换完成（程序中 DelayX1ms(2)的用意），才能读取数值并依据数值进行判断，哪一个按键被按下并将按键数值写入 KeyData 变量内．最后应使得 ADC 不再动作（除能 ADC），程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*****************************/
void InputKey(void)
{
   Byte  keytmp;

   KeyData  = NO_KEY;              //NO_KEY=0

   XFR_ADC=0x81;                   //adc channel=1
   DelayX1ms(2);
   keytmp = XFR_ADC & 0x3f;        //6 bit max.=0x3f
   if ( keytmp < (12+4) )          KeyData = NUMBER1_KEY;
   else if ( keytmp < (23+4) )     KeyData = NUMBER2_KEY;
   else if ( keytmp < (34+4) )     KeyData = NUMBER3_KEY;
   else if ( keytmp < (44+4) )     KeyData = NUMBER4_KEY;
   else if ( keytmp < (50+4) )     KeyData = NUMBER5_KEY;

   XFR_ADC=0x82;                   //adc channel=2
   DelayX1ms(2);
   keytmp = XFR_ADC & 0x3f;
   if ( keytmp < (12+4) )          KeyData = NUMBER6_KEY;
   else if ( keytmp < (23+4) )     KeyData = NUMBER7_KEY;
   else if ( keytmp < (34+4) )     KeyData = NUMBER8_KEY;
   else if ( keytmp < (44+4) )     KeyData = NUMBER9_KEY;

   XFR_ADC=0x00;                   //disable adc
}
```

7. input.h：按键侦测由 ADC 输入，都没有使用到 I/O 作输入或输出，所以无需定义 I/O 引脚，只用定义按键值和侦测函数的外部声明，如下：

```
#ifndef  _INPUT_H
#define  _INPUT_H

#define  NO_KEY              0
#define  NUMBER1_KEY    1
#define  NUMBER2_KEY    2
#define  NUMBER3_KEY    3
#define  NUMBER4_KEY    4
#define  NUMBER5_KEY    5
#define  NUMBER6_KEY    6
#define  NUMBER7_KEY    7
#define  NUMBER8_KEY    8
#define  NUMBER9_KEY    9

extern void InputKey(void);

#endif
```

汇编程序如下：

1. global.a51

```
$NOMOD51

NAME GLOBAL

$INCLUDE (mtv212m.inc)

?ID?GLOBAL              SEGMENT IDATA
 PUBLIC   XFR_X3ANK
 PUBLIC   XFR_OPTION2
 PUBLIC   XFR_OPTION1
 PUBLIC   XFR_PADMOD3
 PUBLIC   XFR_PADMOD2
 PUBLIC   XFR_PADMOD1
 PUBLIC   XFR_WDT
 PUBLIC   XFR_ADC
 PUBLIC   SoundLongCount
 PUBLIC   Period
 PUBLIC   Tempo
 PUBLIC   MusicNumber
 PUBLIC   KeyData
```

```
XSEG  AT  010H
      XFR_ADC:   DS   1

XSEG  AT  018H
      XFR_WDT:   DS   1

XSEG  AT  030H
   XFR_PADMOD1:   DS   1

XSEG  AT  031H
   XFR_PADMOD2:   DS   1

XSEG  AT  032H
   XFR_PADMOD3:   DS   1

XSEG  AT  033H
   XFR_OPTION1:   DS   1

XSEG  AT  034H
   XFR_OPTION2:   DS   1

XSEG  AT  035H
     XFR_XBANK:   DS   1

RSEG  ?ID?GLOBAL
      KeyData:   DS   1
   MusicNumber:   DS   1
        Tempo:   DS   2
       Period:   DS   2
 SoundLongCount:   DS   2

END
```

2. initial.a51

```
$NOMOD51

NAME INITIAL

$INCLUDE (mtv212m.inc)

?PR?PowerOnInit?INITIAL          SEGMENT CODE
?PR?InitialCpu?INITIAL           SEGMENT CODE
?PR?InitialCpuIO?INITIAL         SEGMENT CODE
?PR?InitialVariable?INITIAL      SEGMENT CODE
```

```
EXTRN     IDATA (KeyData)
EXTRN     IDATA (MusicNumber)
EXTRN     XDATA (XFR_ADC)
EXTRN     XDATA (XFR_WDT)
EXTRN     XDATA (XFR_PADMOD1)
EXTRN     XDATA (XFR_PADMOD2)
EXTRN     XDATA (XFR_PADMOD3)
EXTRN     XDATA (XFR_OPTION1)
EXTRN     XDATA (XFR_OPTION2)
EXTRN     XDATA (XFR_XBANK)
PUBLIC    InitialVariable
PUBLIC    InitialCpuIO
PUBLIC    InitialCpu
PUBLIC    PowerOnInit

RSEG  ?PR?PowerOnInit?INITIAL
USING     0
PowerOnInit:
LCALL     InitialCpu
LCALL     InitialCpuIO
LCALL     InitialVariable
RET

RSEG  ?PR?InitialCpu?INITIAL
USING     0
InitialCpu:
CLR       A
MOV       IE,A
MOV       PSW,A
MOV       IP,#0BH
MOV       TMOD,#011H
CLR       TR0
CLR       TR1
SETB      IT0
CLR       EX0
SETB      ET1
CLR       ET0
MOV       R0,#LOW (XFR_ADC)
MOV       A,#080H
MOVX      @R0,A
MOV       R0,#LOW (XFR_WDT)
MOV       A,#040H
MOVX      @R0,A
MOV       R0,#LOW (XFR_PADMOD1)
MOV       A,#07H
```

```
MOVX    @R0,A
CLR     A
MOV     R0,#LOW (XFR_PADMOD2)
MOVX    @R0,A
MOV     R0,#LOW (XFR_PADMOD3)
MOVX    @R0,A
MOV     R0,#LOW (XFR_OPTION1)
MOV     A,#0C0H
MOVX    @R0,A
CLR     A
MOV     R0,#LOW (XFR_OPTION2)
MOVX    @R0,A
MOV     R0,#LOW (XFR_XBANK)
MOVX    @R0,A
SETB    EA
RET

RSEG  ?PR?InitialCpuIO?INITIAL
USING   0
InitialCpuIO:
CLR     P3_1
RET

RSEG  ?PR?InitialVariable?INITIAL
USING   0
InitialVariable:
CLR     A
MOV     R0,#KeyData
MOV     @R0,A
MOV     R0,#MusicNumber
MOV     @R0,A
RET

END
```

3. input.a51

```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT      SEGMENT CODE
?DT?InputKey?INPUT      SEGMENT DATA OVERLAYABLE
```

```
EXTRN     IDATA (KeyData)
EXTRN     XDATA (XFR_ADC)
EXTRN     CODE (_DelayXlms)
PUBLIC    InputKey

RSEG  ?DT?InputKey?INPUT
?InputKey?BYTE:
    keytmp?040:   DS   1

RSEG  ?PR?InputKey?INPUT
USING     0
InputKey:
CLR       A
MOV       R0,#KeyData
MOV       @R0,A
MOV       R0,#LOW (XFR_ADC)
MOV       A,#081H
MOVX      @R0,A
MOV       R7,#02H
MOV       R6,#00H
LCALL     _DelayXlms
MOV       R0,#LOW (XFR_ADC)
MOVX      A,@R0
ANL       A,#03FH
MOV       keytmp?040,A
CLR       C
SUBB      A,#010H
JNC       ?C0001
MOV       R0,#KeyData
MOV       @R0,#01H
SJMP      ?C0002
?C0001:
MOV       A,keytmp?040
CLR       C
SUBB      A,#01BH
JNC       ?C0003
MOV       R0,#KeyData
MOV       @R0,#02H
SJMP      ?C0002
?C0003:
MOV       A,keytmp?040
CLR       C
SUBB      A,#026H
JNC       ?C0005
MOV       R0,#KeyData
```

```
    MOV     @R0,#03H
    SJMP    ?C0002
?C0005:
    MOV     A,keytmp?040
    CLR     C
    SUBB    A,#030H
    JNC     ?C00C7
    MOV     R0,#KeyData
    MOV     @R0,#04H
    SJMP    ?C0002
?C0007:
    MOV     A,keytmp?040
    CLR     C
    SUBB    A,#036H
    JNC     ?C0002
    MOV     R0,#KeyData
    MOV     @R0,#05H
?C0002:
    MOV     R0,#LOW (XFR_ADC)
    MOV     A,#082H
    MOVX    @R0,A
    MOV     R7,#02H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#LOW (XFR_ADC)
    MOVX    A,@R0
    ANL     A,#03FH
    MOV     keytmp?040,A
    CLR     C
    SUBB    A,#010H
    JNC     ?C0010
    MOV     R0,#KeyData
    MOV     @R0,#06H
    SJMP    ?C0011
?C0010:
    MOV     A,keytmp?040
    CLR     C
    SUBB    A,#01BH
    JNC     ?C0012
    MOV     R0,#KeyData
    MOV     @R0,#07H
    SJMP    ?C0011
?C0012:
    MOV     A,keytmp?040
    CLR     C
```

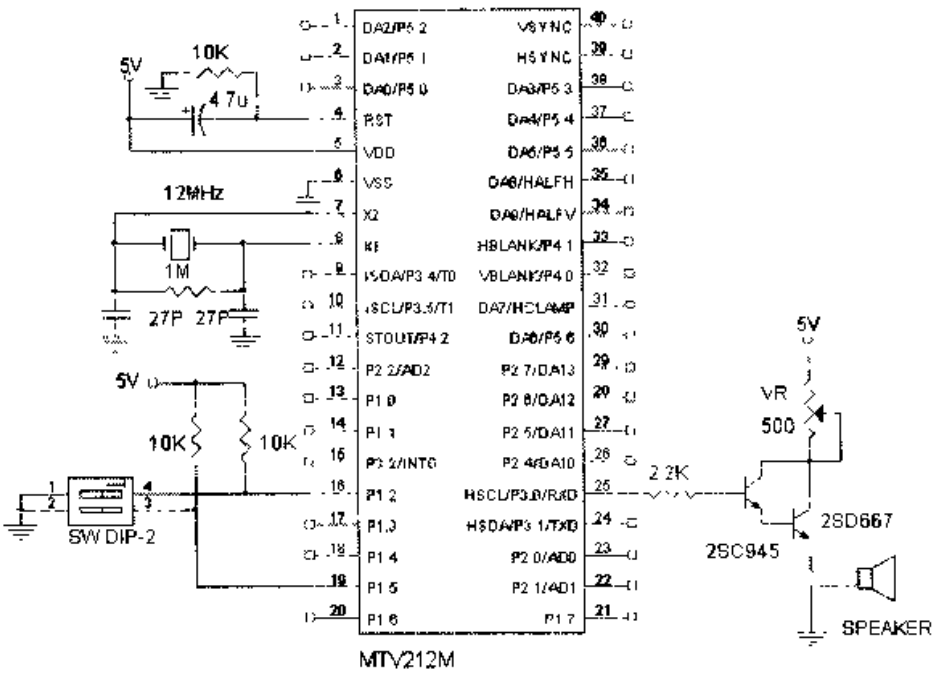
```
SUBB    A,#026H
JNC     ?C0014
MOV     R0,#KeyData
MOV     @R0,#08H
SJMP    ?C0011
?C0014:
MOV     A,keytmp?040
CLR     C
SUBB    A,#030H
JNC     ?C0011
MOV     R0,#KeyData
MOV     @R0,#09H
?C0011:
CLR     A
MOV     R0,#LOW (XFR_ADC)
MOVX    @R0,A
RET

END
```

第 9 章　DIP 指拨开关选曲

9-1　DIPX2:1～3 曲目

目的：利用 2P 的指拨开关作为选曲 1～3。

电路图：



原理：指拨开关和 Tact 开关（按下接点导通，放开接点不导通），都是作为开关，由 I/O 脚输入并要接提升电阻至+5V 的一种输入设备。指拨开关是并排的开关，其接点有 2P、4P、8P 甚至更多，体积小，开关个数多，开关上下拨动容易是其优点，常作为输入选项功能使用，一般也是使用“0”动作，I/O P1.2 和 P1.5 由于提升电阻的关系，平常都是“1”电位，当开关拨至“ON”位置，即上下接点导通，其输入变为“0”电位（因为指拨开关另一接点接地所致），则侦测 P1.2 及 P1.5 是否“0”电位，就知道指拨开关第一支脚及第二支脚是否拨至“ON”位置。

9-1-1 各自独立的 I/O 作为输入

1. 软件构建的思维

当指拨开关分别输入不同的 I/O 脚，则需个别侦测 I/O 脚是否为“0”电位，两个输入共有 2^2=4 个状态，即 0、1、2、3 分别代表曲目编号，所以程序必须先判断二支 I/O 脚是否都为“0”电位，表示为编号 3，当第一支 I/O 为“0”电位，第二支 I/O 为“1”电位，表示为编号 1；当第一支 I/O 为“1”电位，第二支 I/O 为“0”电位，表示为编号 2，其 I/O 电位与数值、曲目的关系如下：

P1.2	P1.5	数值	曲目编号
1	1	0	0
0	1	1	1
1	0	2	2
0	0	3	3

现分述如下：

（1）music.c：2P 的指拨开关只能选曲 1、2 或 3，则 Music( )函数 switch 指令下的条件值也只是 1、2 或 3，分别调用音乐函数 1、2、3 而已。

（2）music.h：外部函数声明只有 Music1( )、Music2( )和 Music3( )函数。

（3）input.c：指拨开关分别侦测 I/O，并将按键数值写入。

（4）input.h：定义指拨开关 I/O，按键数值和函数外部声明。

2. 软件的解决方法

虽然只能选曲 3 首，在 Music( )音乐演奏函数中，仍然以 switch...case 结构并以判断自变量作为演奏曲目的依据，程序一样的条理分明，而当 DIP1_IO 和 DIP2_IO 都被拨至“ON”，其输入的两个 I/O 都会是“0”电位，为选第 3 曲之意，所以应先判断，再按顺序判断个别的 I/O 才是正确的，现分述如下：

（1）music.c：其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
```

```
#include  "music.h"

/*****************************/
void Music(Byte number)
{
    switch( number )
    {
     case 1 :
        Music1( );
        break;
     case 2 :
        Music2( );
        break;
     case 3 :
        Music3( );
        break;
     default :
        break;

    }

}
```

（2）music.h：函数的外部声明，只剩下 Music( )函数及曲目 1、2、3 的函数，如下：

```
extern void Music(Byte number);
extern void Music1(void);
extern void Music2(void);
extern void Music3(void);
```

（3）input.c：当两个 DIP 开关都未拨至“ON”位置，其按键数值为 0X00，因而，InputKey( )函数一执行，在程序前端立即清除 KeyData 即是此意，再以 if...else 条件结构来判断 DIP 开关是否拨至“ON”位置，如果是，则将按键数值写入 KeyData 变量内，如下：

```
/*****************************/

/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
```

```
#include  "input.h"

/*****************************/
void InputKey(void)
{
   KeyData = NO_KEY;    //NO_KEY=0x00
   if ( (DIP1_IO==0) && (DIP2_IO==0) )
     KeyData = NUMBER3_KEY;
   else if ( DIP1_IO ==0 )
     KeyData = NUMBER1_KEY;
   else if ( DIP2_IO ==0 )
     KeyData = NUMBER2_KEY;
}
```

（4）input.h：分别定义 DIP1_IO 为 P1.2，DIP2_IO 为 P1.5，及键值和侦测函数 InputKey( )的外部声明，如下：

```
#ifndef   _INPUT_H
#define   _INPUT_H

#define  DIP1_IO            P1_2
#define  DIP2_IO            P1_5

#define  NO_KEY             0
#define  NUMBER1_KEY    1
#define  NUMBER2_KEY    2
#define  NUMBER3_KEY    3

extern void InputKey(void);

#endif
```

汇编程序如下：

1. iuput.a51

```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT          SEGMENT CODE
EXTRN     IDATA (KeyData)
PUBLIC    InputKey
```

```
RSEG  ?PR?InputKey?INPUT
USING    0
InputKey:
CLR      A
MOV      R0,#KeyData
MOV      @R0,A
JB   P1_2,?C0001
JB   P1_5,?C0001
MOV      @R0,#03H
RET
?C0001:
JB   P1_2,?C0003
MOV      R0,#KeyData
MOV      @R0,#01H
RET
?C0003:
JB   P1_5,?C0006
MOV      R0,#KeyData
MOV      @R0,#02H
?C0006:
RET

END
```

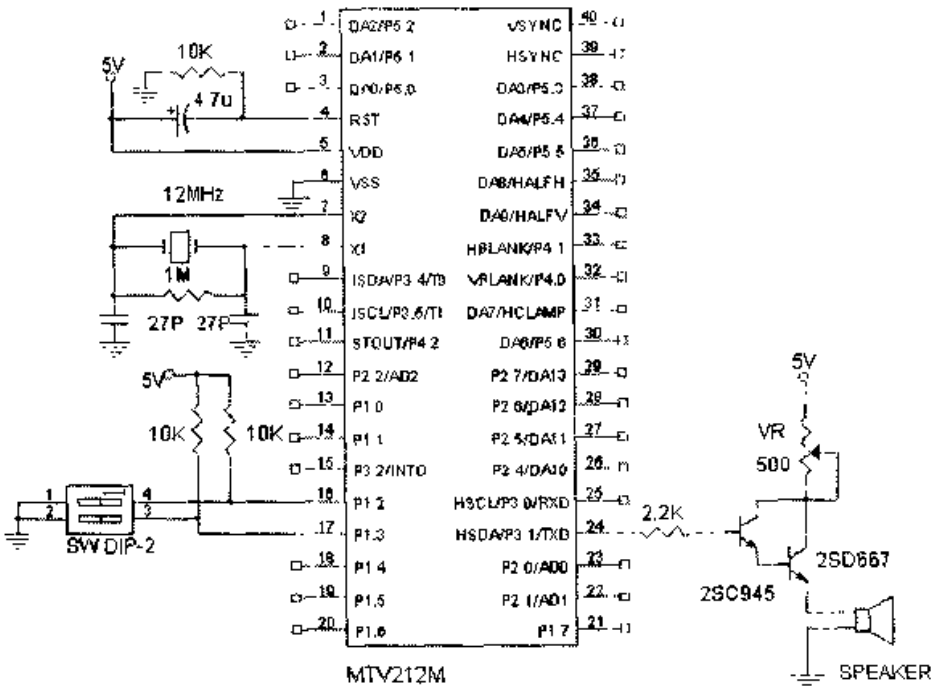
2. music.a51

```
$NOMOD51

NAME MUSIC

$INCLUDE (mtv212m.inc)

?PR?_Music?MUSIC              SEGMENT CODE
EXTRN    CODE (Music1)
EXTRN    CODE (Music2)
EXTRN    CODE (Music3)
PUBLIC   _Music

RSEG  ?PR?_Music?MUSIC
USING    0
_Music:
MOV      A,R7
ADD      A,#0FEH
JZ   ?C0003
DEC      A
```

```
    JZ   ?C0004
    ADD       A,#02H
    JNZ       ?C0006
?C0002:
    LCALL     Music1
    RET
?C0003:
    LCALL     Music2
    RET
?C0004:
    LCALL     Music3
?C0006:
    RET

    END
```

9-1-2 相邻的 I/O 作为输入

电路图：



软件构建的思维和解决方法

（1）input.c：以相邻的 I/O 作为输入，除了可单独进行侦测之外，也可以利用读取端口的方式，将 8bit 全部读入，反相后并屏蔽未输入 I/O 的位，再向右移位。例如：利用 P1.2 和 P1.3 作相邻 I/O 输入，则 P1.2 和 P1.3 的位为 bit2 和 bit3，其 bit2

的数值为 4，bit3 的数值为 8，两者相加为 0X0C，所以当读取 P1 端口之后还要和数值 0X0C 作 AND 运算，并向右移位 2 个 bit（将 bit0 和 bit1 移出去），再依据此暂时变量 Keytmp 进行判断，当 keytmp=1，表示 P1.2=“0”电位，DIP1 拨至“ON”位置；当 keytmp=2，表示 P1.3=“0”电位，DIP2 拨至“ON”的位置了；当 keytmp=3，表示 P1.2 和 P1.3 同时都为“0”电位，DIP1 和 DIP2 都拨至“ON”位置，也由于指拨开关拨至“ON”位置，其输入 I/O 脚变为“0”电位，称为“0”动作，因此，在读取 P1 端口时必须反相，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*****************************/
void InputKey(void)
{
    Byte  keytmp;

    keytmp = ~(INPUT_PORT) & 0x0c;    //P1.2~P1.3
    keytmp >>= 2;

    KeyData = NO_KEY;
    if ( keytmp ==1 )
       KeyData = NUMBER1_KEY;
    else if ( keytmp ==2 )
       KeyData = NUMBER2_KEY;
    else if ( keytmp ==3 )
       KeyData = NUMBER3_KEY;
}
```

（2）input.h：由于输入的 I/O 为 P1.2 和 P1.3，其输入端口为 P1，因此要定义，如下：

```
#ifndef   _INPUT_H
#define   _INPUT_H

#define  INPUT_PORT          P1
```

```
#define  NO_KEY              0
#define  NUMBER1_KEY    1
#define  NUMBER2_KEY    2
#define  NUMBER3_KEY    3

extern void InputKey(void);

#endif
```

汇编程序如下：

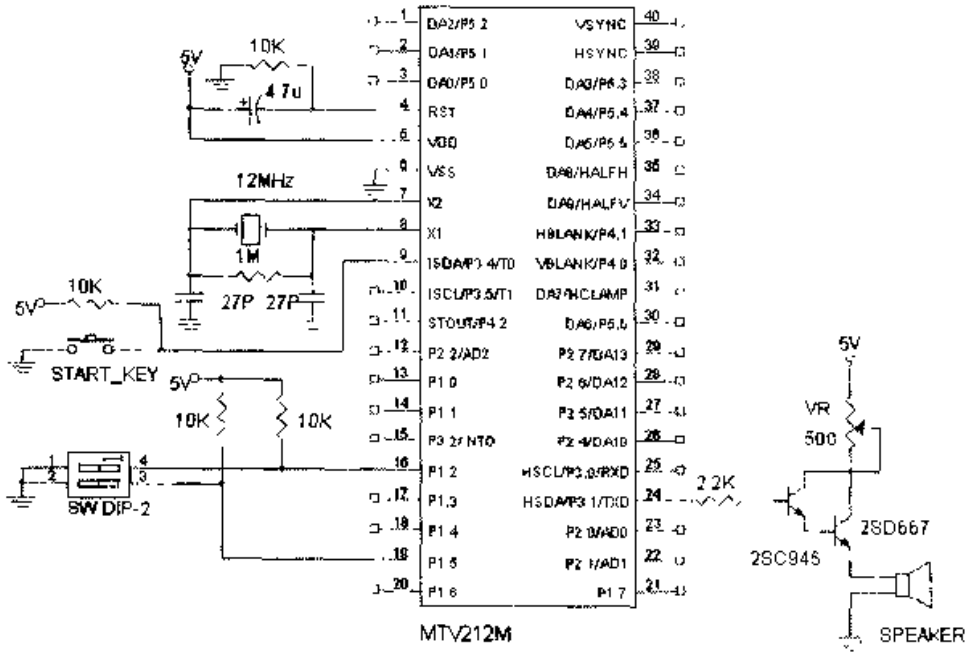
input.a51

```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT          SEGMENT CODE
EXTRN    IDATA (KeyData)
PUBLIC   InputKey

RSEG  ?PR?InputKey?INPUT
USING    0
InputKey:
MOV      A,P1
CPL      A
ANL      A,#0CH
MOV      R7,A
RRC      A
RRC      A
ANL      A,#03FH
MOV      R7,A
CLR      A
MOV      R0,#KeyData
MOV      @R0,A
CJNE     R7,#01H,?C0001
MOV      @R0,#01H
RET
?C0001:
CJNE     R7,#02H,?C0003
MOV      R0,#KeyData
MOV      @R0,#02H
RET
```

```
?C0003:
 CJNE      R7,#03H,?C0006
 MOV       R0,#KeyData
 MOV       @R0,#03H
?C0006:
 RET

 END
```

9-1-3　各自独立的 I/O 作为输入必须按下起始键才进行演奏

电路图：



软件构建的思维和解决方法

当 DIP 开关选择好曲目编号时，当按下起始键才进行演奏音乐功能，而演奏完毕或在演奏过程中都可任意拨动 DIP 开关，即改变下一次的演奏曲目，也必须再按下起始键才会继续演奏。整个流程为：当外界所有的条件输入或设定完成时，再命令执行动作，此起始键即是执行命令的一种输入状态，为了达此目的，其解决方法如下：

（1）main.c：在反复循环内，除了不断侦测是否按键外（ImputKey( )函数），在执行音乐演奏的函数前，应判断是否按下起始键，是，才执行，否则不执行音乐演奏函数，而继续侦测 DIP 开关和起始键，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      //sel play no#
      InputKey( );

      if ( START_IO==0 )
      {
         MusicNumber=KeyData;
         Music(MusicNumber);
         DelayX10ms(100);
      }
   }
}
```

（2）input.h：除了定义DIP_IO为P1.2，DIP2_IO为P1.5外，必须再定义侦测起始键的I/O为P3.4，以及拨动DIP1开关的数值为1，拨动DIP2开关的数值为2，同时拨动DIP1和DIP2开关的数值为3，如下：

```
#ifndef   _INPUT_H
#define   _INPUT_H

#define  START_IO       P3_4
#define  DIP1_IO        P1_2
#define  DIP2_IO        P1_5

#define  NO_KEY         0
#define  NUMBER1_KEY    1
#define  NUMBER2_KEY    2
#define  NUMBER3_KEY    3
```

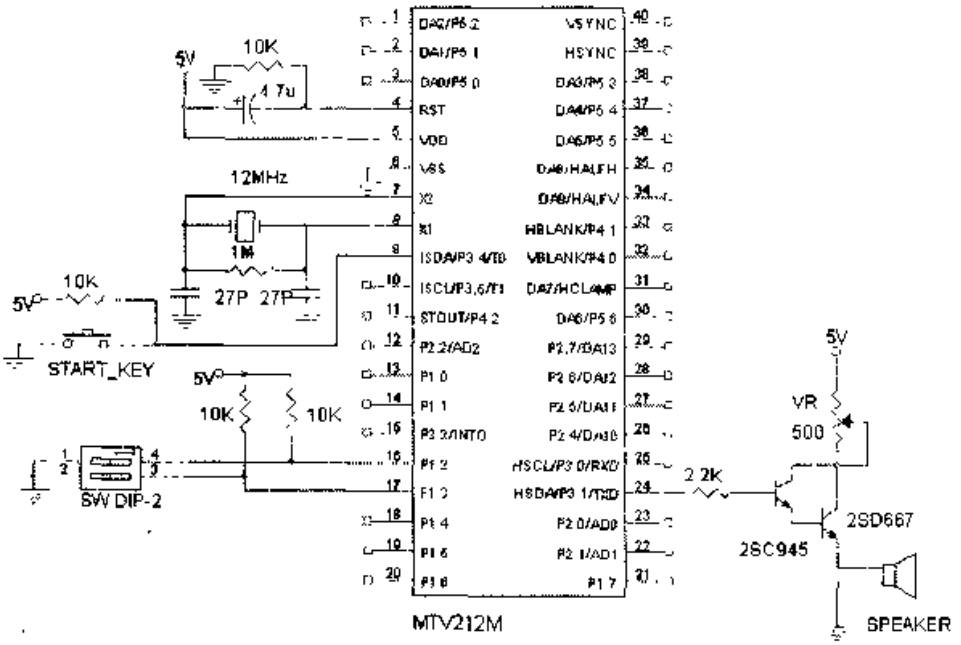
```
extern void InputKey(void);

#endif
```

汇编程序如下：

```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN                SEGMENT CODE
 EXTRN    IDATA (KeyData)
 EXTRN    IDATA (MusicNumber)
 EXTRN    CODE (PowerOnInit)
 EXTRN    CODE (_DelayX10ms)
 EXTRN    CODE (_Music)
 EXTRN    CODE (InputKey)
 EXTRN    CODE (?C_STARTUP)
 PUBLIC   Main

 RSEG  ?PR?Main?MAIN
 USING    0
Main:
 LCALL    PowerOnInit
?C0001:
 LCALL    InputKey
 JB   P3_4,?C0001
 MOV      R0,#KeyData
 MOV      A,@R0
 MOV      R7,A
 MOV      R0,#MusicNumber
 MOV      @R0,A
 LCALL    _Music
 MOV      R7,#064H
 MOV      R6,#00H
 LCALL    _DelayX10ms
 SJMP     ?C0001
 RET

 END
```

9-1-4 相邻的 I/O 作输入，须按下起始键才进行演奏

电路图：



软件构建的思维与解决方法

只要是相邻的I/O作为输入，不管是几支I/O作为输入，其侦测I/O是否动作（“0”动作）都可用读取端口反相后再屏蔽其他位和向右移位的方式来作为开关动作的数值，当开关决定之后，按下起始键才进行演奏，则在 Main( )函数多加一个起始键是否按下的判断，以及 InputKey( )函数以读取端口的方式侦测，其程序如下：

（1）main.c：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*******************************/
void Main( void )
{
```

```
    PowerOnInit();

    while( 1 )
    {
      //sel play no#
      InputKey( );

      if ( START_IO==0 )
      {
        MusicNumber=KeyData;
        Music(MusicNumber);
        DelayX10ms(100);
      }
    }
}
```

（2）input.c：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*****************************/
void InputKey(void)
{
    Byte  keytmp;

    keytmp = ~(INPUT_PORT) & 0x0c;    //P1.2~P1.3
    keytmp >>= 2;

    KeyData = NO_KEY;
    if ( keytmp ==1 )
      KeyData = NUMBER1_KEY;
    else if ( keytmp ==2 )
      KeyData = NUMBER2_KEY;
    else if ( keytmp ==3 )
      KeyData = NUMBER3_KEY;
}
```

（3）input.h：

```
#ifndef   _INPUT_H
#define   _INPUT_H

#define  START_IO       P3_4
#define  INPUT_PORT          P1

#define  NO_KEY              0
#define  NUMBER1_KEY    1
#define  NUMBER2_KEY    2
#define  NUMBER3_KEY    3

extern void InputKey(void);

#endif
```

汇编程序如下：

1. main.a51

```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN                SEGMENT CODE
 EXTRN     IDATA (KeyData)
 EXTRN     IDATA (MusicNumber)
 EXTRN     CODE (PowerOnInit)
 EXTRN     CODE (_DelayX10ms)
 EXTRN     CODE (_Music)
 EXTRN     CODE (InputKey)
 EXTRN     CODE (?C_STARTUP)
 PUBLIC    Main

 RSEG  ?PR?Main?MAIN
 USING     0
Main:
 LCALL     PowerOnInit
?C0001:
 LCALL     InputKey
 JB   P3_4,?C0001
 MOV       R0,#KeyData
 MOV       A,@R0
```

```
MOV       R7,A
MOV       R0,#MusicNumber
MOV       @R0,A
LCALL     _Music
MOV       R7,#064H
MOV       R6,#00H
LCALL     _DelayX10ms
SJMP      ?C0001
RET

END
```

2. input.a51

```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT          SECMENT CODE
EXTRN     IDATA (KeyData)
PUBLIC    InputKey

RSEG  ?PR?InputKey?INPUT
USING     0
InputKey:
MOV       A,P1
CPL       A
ANL       A,#0CH
MOV       R7,A
RRC       A
RRC       A
ANL       A,#03FH
MOV       R7,A
CLR       A
MOV       R0,#KeyData
MOV       @R0,A
CJNE      R7,#01H,?C0001
MOV       @R0,#01H
RET
?C0001:
CJNE      R7,#02H,?C0003
MOV       R0,#KeyData
MOV       @R0,#02H
```

```
RET
?C0003:
CJNE    R7,#03H,?C0006
MOV     R0,#KeyData
MOV     @R0,#03H
?C0006:
RET

END
```

9-1-5 未选曲时，按下起始键也不执行音乐演奏函数

软件构建的思维与解决方法

当指拨开关有任意拨动至“ON”位置后，按下起始键则根据数值进行曲目演奏，如果指拨开关都未拨动而在“OFF”位置，即使按下起始键也不调用音乐演奏函数，主要是利用 KeyData 变量判断是否不为 0X00。当不为 0X00，表示有 DIP 开关输入，当为 0X00，表示开关都不动作，即未选曲，因而，不执行音乐演奏函数，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
     //sel play no#
     InputKey( );

     if ( START_IO==0 )
     {
```

```
        if ( KeyData != NO_KEY )
        {
          MusicNumber=KeyData;
          Music(MusicNumber);
          DelayX10ms(100);
        }
      }
    }
}
```

汇编程序如下：

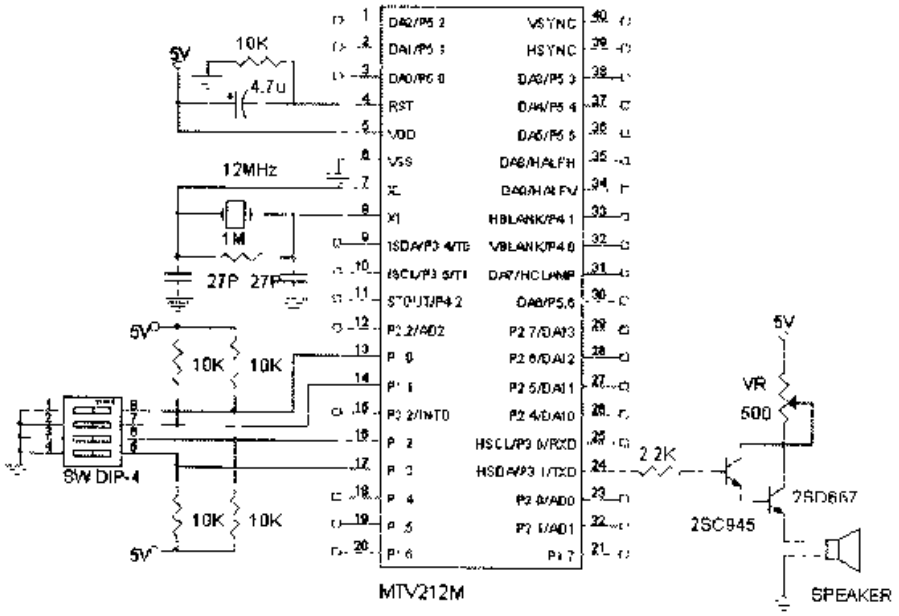
```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN                SEGMENT CODE
 EXTRN    IDATA (KeyData)
 EXTRN    IDATA (MusicNumber)
 EXTRN    CODE (PowerOnInit)
 EXTRN    CODE (_DelayX10ms)
 EXTRN    CODE (_Music)
 EXTRN    CODE (InputKey)
 EXTRN    CODE (?C_STARTUP)
 PUBLIC   Main

 RSEG  ?PR?Main?MAIN
 USING    0
Main:
 LCALL    PowerOnInit
?C0001:
 LCALL    InputKey
 JB   P3_4,?C0001
 MOV      R0,#KeyData
 MOV      A,@R0
 MOV      R7,A
 JZ   ?C0001
 MOV      R0,#MusicNumber
 MOV      @R0,A
 LCALL    _Music
 MOV      R7,#064H
 MOV      R6,#00H
 LCALL    _DelayX10ms
```

```
SJMP    ?C0001
RET

END
```

9-2 DIPX4：1～15 曲目

目的：以 4P 的指拔开关作为选曲，共 1～15 曲。

电路图：



原理：以 4P 的指拔开关作为输入，总共有 2^4=16 种变化，数值为 0～15，代表曲目编号 0～15，其中曲目编号 0 将不动作。当开关拨至“ON”位置，则输入的 I/O 将转变，为“0”电位，而以相邻 I/O P1.0～P1.3 作为开关的四个输入，则侦测函数的程序设计会更简单，因此，硬件电路才这样设计。

9-2-1 以低 4 位作为开关选曲输入

1. 软件构建的思维

指拔开关由 P1.0～P1.3 输入，则当读取 P1 端口反相后再屏蔽高 4 位，则形成开关数值也是曲目编号了，软件变得简单易懂，所以为避免麻烦，硬件设计在规划时应考虑软件的便捷性及复杂度。而当不作选曲，即开关都未拨至“ON”位置，以

不调用演奏函数为标准，另外，由于是 4P 的开关，可选曲 1～15，则音乐演奏函数 switch...case 中的条件值要从 1～15 进行判断，分述如下：

（1）main.c：多加了 KeyData 变量（开关的数值）是否不等于 0X00 的判断，以跳过当开关都不拨至“ON”位置的音乐演奏。

（2）input.c：以一个局域变量来暂存开关的读入值，经过屏蔽 bit4～bit7 位后再写入 KeyData 变量，其实，也可以不需要局域变量 Keytmp，直接读入值并运算后写入至 KeyData 变量内，由于输入是“0”电位动作，所以读入端口要反相。

（3）input.h：要定义输入端口 P1 和开关都未拨至“ON”位置的数值为 0X00。

（4）music.c：总共有 1～15 首曲目，所以有 Music1( )～Music15( )函数。

（5）music.h：Music1( )～Music15( )函数的外部声明。

2. 软件的解决方法

有了模块化的概念，又有了以上的构思，则要以程序的方法来实现功能，其实，也是很简单的。至此，相信各位读者应该建立了软件设计的思维和问题的解决方法吧！要达到此目标的解决方法如下：

（1）main.c：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      //sel play no#
      InputKey( );
      if ( KeyData !=NO_KEY )
      {
         MusicNumber=KeyData;
         Music(MusicNumber);
```

```
            DelayX10ms(100);
        }
    }
}
```

（2）input.c：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*****************************/
void InputKey(void)
{
    Byte  keytmp;

    keytmp = ~(INPUT_PORT) & 0x0f;    //P1.0~P1.3

    KeyData = keytmp;
}
```

（3）input.h：输入端口要设定为P1，因为分别输入的I/O为P1.0～P1.3，如下：

```
#ifndef   _INPUT_H
#define   _INPUT_H

#define  INPUT_PORT          P1

#define  NO_KEY              0

extern void InputKey(void);

#endif
```

（4）music.c：以自变量number作为演奏的曲目编号，总共有15首，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
```

```
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
void Music(Byte number)
{
    switch( number )
    {
     case 1 :
        Music1( );
        break;
     case 2 :
        Music2( );
        break;
     case 3 :
        Music3( );
        break;
     case 4 :
        Music4( );
        break;
     case 5 :
        Music5( );
        break;
     case 6 :
        Music6( );
        break;
     case 7 :
        Music7( );
        break;
     case 8 :
        Music8( );
        break;
     case 9 :
        Music9( );
        break;
     case 10 :
        Music10( );
        break;
     case 11 :
        Music11( );
        break;
     case 12 :
```

```
            Music12( );
            break;
        case 13 :
            Music13( );
            break;
        case 14 :
            Music14( );
            break;
        case 15 :
            Music15( );
            break;
        default :
            break;
       }
}
```

（5）music.h：除了音乐演奏的选曲函数Music( )外，以及这15首的函数都要作外部函数声明，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*****************************/
void Music(Byte number)
{
    switch( number )
    {
     case 1 :
        Music1( );
        break;
     case 2 :
        Music2( );
        break;
     case 3 :
        Music3( );
        break;
     case 4 :
        Music4( );
```

```
            break;
        case 5 :
            Music5( );
            break;
        case 6 :
            Music6( );
            break;
        case 7 :
            Music7( );
            break;
        case 8 :
            Music8( );
            break;
        case 9 :
            Music9( );
            break;
        case 10 :
            Music10( );
            break;
        case 11 :
            Music11( );
            break;
        case 12 :
            Music12( );
            break;
        case 13 :
            Music13( );
            break;
        case 14 :
            Music14( );
            break;
        case 15 :
            Music15( );
            break;
        default :
            break;
    }
}
```

而音乐演奏函数模块music10.c～music15.c，其曲名和程序如下：

Music10.c 曲名：平安夜

```
/*****************************/
/* include files             */
```

```
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG10[ ] =
{
    _4d,_8,_4 ,_2d,_4d,_8 ,_4 ,_2d,
    _2 ,_4,_2d,_2 ,_4 ,_2d,
    _2 ,_4,_4d,_8 ,_4 ,_4d,_8 ,_4 ,_2d,
    _2 ,_4,_4d,_8 ,_4 ,_4d,_8 ,_4 ,_2d,
    _2 ,_4,_4d,_8 ,_4 ,_2d,_2d,
    _4 ,_4,_4 ,_4d,_8 ,_4 ,_2d,
    _EOF
};

Word RDATA SOUNDTONE10[ ] =
{
    _SO ,_LA ,_SO ,_MI ,_SO ,_LA ,_SO ,_MI,
    _RE1,_RE1,_TI ,_DO1,_DO1,_SO ,
    _LA ,_LA ,_DO1,_TI ,_LA ,_SO ,_LA ,_SO,_MI,
    _LA ,_LA ,_DO1,_TI ,_LA ,_SO ,_LA ,_SO,_MI,
    _RE1,_RE1,_FA1,_RE1,_TI ,_DO1,_MI1,
    _DO1,_SO ,_MI ,_SO ,_FA ,_RE ,_DO ,
    _EOF
};

void Music10(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG10[i]!=_EOF)||(SOUNDTONE10[i]!=_EOF) )
    {
       //start timer1,generate soundtone
       Period=SOUNDTONE10[i];

       TL1 = (65536 - Period) & 0xff;
       TH1 = (65536 - Period) >> 8;
       TR1 = 1;

       SoundLongCount=((Tempo*SOUNDLONG10[i]*1000L)/8)/Period;
```

```
        while ( SoundLongCount != 0 );

        TR1 = 0;      //stop timer1:soundtone
        i++;
    }
}
```

Music11.c　曲名：雨的旋律

```
/*****************************/
/* include files             */
/*****************************/
#include "define.h"
#include "mtv212m.h"
#include "global.h"
#include "music.h"

/*****************************/
Byte RDATA SOUNDLONG11[ ] =
{
    _8,_8,_8,_8,_8,_8,_8 ,_8,_4 ,_8 ,_8,_2 ,
    _8,_8,_8,_8,_8,_8,_8 ,_8,_2d,_8 ,
    _8,_8,_8,_8,_8,_8,_8 ,_8,_4 ,_8 ,_6,_4d,_8,
    _8,_8,_8,_8,_4,_4,_2d,
    _4,_4,_8,_8,_8,_8,_8 ,_4,_8 ,_4d,_8,
    _8,_8,_8,_8,_8,_8,_8 ,_8,
    _8 ,_4 ,_8,_4d,_8,
    _4,_8,_8,_8,_8,_8,_8 ,
    _4,_8,_8,_8,_8,_4,_1 ,
    _EOF
};

Word RDATA SOUNDTONE11[ ] =
{
    _LA,_SO,_SO ,_MI ,_MI ,_RE ,_RE,_DO,_RE,_DO,_DO,_DO,
    _LA,_SO,_SO ,_MI ,_MI ,_RE ,_MI,_SO,_SO,_DO,
    _LA,_SO,_SO ,_MI ,_MI ,_RE ,_RE,_DO,_RE,_DO,_DO,_DO,_1LA,
    _DO,_RE,_DO ,_RE ,_1LA,_1TI,_DO,
    _LA,_LA,_LA ,_SOr,_LA ,_DO1,_TI,_SO,_SO,_SO,_MI,
    _FA,_MI,_FA ,_MI ,_FA ,_MI ,_FA,_LA,
    _SO,_MI,_MI ,_MI ,_1SO,
    _DO,_DO,_1TI,_DO ,_1TI,_DO ,_MI,
    _RE,_DO,_1LA,_1LA,_1SO,_1LA,_DO,
    _EOF
};
```

```
void Music11(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG11[i]!=_EOF)||(SOUNDTONE11[i]!=_EOF) )
    {
       //start timer1,generate soundtone
       Period=SOUNDTONE11[i];

       TL1 = (65536 - Period) & 0xff;
       TH1 = (65536 - Period) >> 8;
       TR1 = 1;

       SoundLongCount=((Tempo*SOUNDLONG11[i]*1000L)/8)/Period;
       while ( SoundLongCount != 0 );

       TR1 = 0;      //stop timer1:soundtone
       i++;
    }
}
```

Music12.c 曲名：Say Yes My Boy

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG12[ ] =
{
    _8,_8,_2d,_4,_4 ,_4,_1 ,
    _8,_8,_8 ,_8,_8 ,_8,_8 ,_8 ,

    _8,_8,_4 ,_8,_8 ,_4,
    _8,_8,_4 ,_4,_4 ,_8,_8 ,_8 ,_8,
    _8,_8,_4 ,_8,_8 ,_4,
    _8,_8,_4 ,_4,_4 ,_8,_8 ,_2d,
    _4,_8,_8 ,_4,_4d,_8,_2 ,
```

```
    _8,_8,_4 ,_8,_8 ,_4,_2d,
    _4,_8,_8 ,_8,_8 ,_4,_8 ,_8 ,_2,
    _8,_8,_8 ,_4,_8 ,_8,_8 ,_2 ,
    _EOF
};

Word RDATA SOUNDTONE12[ ] =
{
    _LA ,_DO1,_DO1,_TI ,_DO1,_RE1,_DO1,
    _SO ,_SO ,_FA ,_FA ,_MI ,_MI ,_RE ,_RE ,

    _MI ,_SO ,_SO ,_LA ,_SO ,_SO ,
    _LA ,_DO1,_DO1,_RE1,_TI ,_TI ,_LA ,_SO ,_FA,
    _MI ,_SO ,_SO ,_LA ,_SO ,_SO ,
    _LA ,_DO1,_DO1,_TI ,_DO1,_RE1,_DO1,_DO1,
    _DO1,_DO1,_DO1,_LA ,_TI ,_MI ,_MI ,
    _FA ,_MI ,_RE ,_MI ,_FA ,_LA ,_SO ,
    _LA ,_LA ,_LA ,_TI ,_DO1,_TI ,_SO ,_LA ,_TI,
    _LA ,_LA ,_LA ,_TI ,_DO1,_MI1,_RE1,_RE1,
    _EOF
};

void Music12(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG12[i]!=_EOF)||(SOUNDTONE12[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE12[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG12[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

Music13.c 曲名：你怎么说

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG13[ ] =
{
    _4d,_8,_4 ,_8 ,_8 ,_8 ,_8 ,_8 ,_2 ,
    _8 ,_8,_8 ,_8 ,_16,_16,_8 ,_8 ,_2 ,
    _8 ,_8,_8 ,_8 ,_8 ,_8 ,_8 ,_4d,_8 ,_2 ,
    _8 ,_8,_8 ,_8 ,_8 ,_8 ,_8 ,_2 ,
    _4d,_8,_4 ,_8 ,_8 ,_8 ,_8 ,_8 ,_8 ,_2 ,
    _8 ,_8,_8 ,_8 ,_8 ,_16,_16,_8 ,_8 ,_2 ,
    _8 ,_8,_4d,_8 ,_4 ,_8 ,_8 ,
    _8 ,_8,_16,_16,_16,_16,_2,
    _8 ,_8,_8 ,_8 ,_8 ,_8 ,_8 ,_16,_16,_2 ,
    _8 ,_8,_8 ,_8 ,_8 ,_8 ,_8 ,_2 ,
    _EOF
};

Word RDATA SOUNDTONE13[ ] =
{
    _DO ,_RE ,_DO,_1LA,_1SO,_1MI,_1SO,_1LA,_DO ,_1SO,
    _1SO,_1LA,_DO,_RE ,_MI ,_RE ,_DO ,_RE ,_MI ,
    _SO ,_MI ,_RE,_MI ,_MI ,_1SO,_1LA,_DO ,_MI ,_RE ,
    _1SO,_SO ,_MI,_RE ,_1SO,_MI ,_RE ,_DO ,
    _DO ,_RE ,_MI,_MI ,_SO ,_MI ,_RE ,_DO ,_1LA,_1SO,
    _DO ,_DO ,_RE,_MI ,_1MI,_1SO,_1MI,_1RE,_1MI,_1SO,
    _1SO,_1LA,_DO,_DO ,_DO ,_RE ,_SO ,
    _MI ,_SO ,_MI,_SO ,_MI ,_RE ,_DO ,
    _RE ,_RE ,_DO,_RE ,_MI ,_RE ,_DO ,_RE ,_MI ,_SO ,
    _1SO,_SO ,_MI,_RE ,_1SO,_MI ,_RE ,_DO ,
    _EOF
};

void Music13(void)
{
    Byte i=0;
```

```
    Tempo = MODERATO;

    while ( (SOUNDLONG13[i]!=_EOF)||(SOUNDTONE13[i]!=_EOF) )
    {
       //start timer1,generate soundtone
       Period=SOUNDTONE13[i];

       TL1 = (65536 - Period) & 0xff;
       TH1 = (65536 - Period) >> 8;
       TR1 = 1;

       SoundLongCount=((Tempo*SOUNDLONG13[i]*1000L)/8)/Period;
       while ( SoundLongCount != 0 );

       TR1 = 0;      //stop timer1:soundtone
       i++;
    }
}
```

Music14.c　曲名：庙会

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG14[ ] =
{
    _4,_4,_4 ,_4 ,_8d,_16,_8,_8,_2,
    _4,_4,_8 , 4d,_2 ,
    _4,_4,_4 ,_4 ,_4 ,_8 ,_8,_2,
    _4,_4,_8 ,_4d,_2 ,
    _8,_8,_4d,_8 ,_8 ,_8 ,_8,_8,_2,
    _8,_8,_4 ,_4 ,_8 ,_8 ,_2,
    _8,_8,_4 ,_4 ,_8 ,_8 ,_8,_8,_2,
    _8,_8,_4 ,_4 ,_2d,
    _EOF
};

Word RDATA SOUNDTONE14[ ] =
{
```

```
    _SO ,_SO ,_MI ,_RE,_MI,_MI,_MI ,_MI ,_1LA,
    _DO ,_1LA,_1LA,_DO,_RE,
    _SO ,_SO ,_MI ,_RE,_MI,_MI,_1LA,_1LA,
    _DO ,_1LA,_1LA,_DO,_RE,
    _1LA,_DO ,_RE ,_MI,_SO,_LA,_SO ,_SO ,_SO,
    _SO ,_LA ,_SO ,_SO,_MI,_MI,_MI ,
    _SO ,_LA ,_SO ,_MI,_RE,_MI,_RE ,_RE ,_RE,
    _RE ,_MI ,_SO ,_SO,_DO,
    _EOF
};

void Music14(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG14[i]!=_EOF)||(SOUNDTONE14[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE14[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG14[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

Music15.c 曲名：花非花

```
/*****************************/
/* include files             */
/*****************************/
#include "define.h"
#include "mtv212m.h"
#include "global.h"
#include "music.h"

/*****************************/
```

```
Byte RDATA SOUNDLONG15[ ] =
{
    _2,_4,_4,_2 ,_2,_2,_4,_4,_2,_2,
    _2,_4,_4,_2d,_4,_2,_4,_4,_2,
    _2,_4,_4,_2 ,_2,_2,_4,_4,_2,
    _2,_4,_4,_2 ,_4,_4,_2,_4,_4,_2,
    _EOF
};

Word RDATA SOUNDTONE15[ ] =
{
    _SO ,_LA,_SO ,_SO,_MI,_DO1,_RE1,_DO1,_DO1,_LA,
    _SO ,_SO,_DO1,_LA,_SO,_MI ,_RE ,_DO ,_RE ,
    _RE ,_MI,_SO ,_LA,_SO,_SO ,_RE1,_DO1,_LA ,
    _DO1,_LA,_DO1,_SO,_MI,_SO ,_1LA,_RE ,_MI ,_DO,
    _EOF
};

void Music15(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG15[i]!=_EOF)||(SOUNDTONE15[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE15[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG15[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

汇编程序如下：

1. main.a51

```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN                SEGMENT CODE
 EXTRN    IDATA (KeyData)
 EXTRN    IDATA (MusicNumber)
 EXTRN    CODE (PowerOnInit)
 EXTRN    CODE (_DelayX10ms)
 EXTRN    CODE (_Music)
 EXTRN    CODE (InputKey)
 EXTRN    CODE (?C_STARTUP)
 PUBLIC   Main

 RSEG  ?PR?Main?MAIN
 USING    0
Main:
 LCALL    PowerOnInit
?C0001:
 LCALL    InputKey
 MOV      R0,#KeyData
 MOV      A,@R0
 MOV      R7,A
 JZ   ?C0001
 MOV      R0,#MusicNumber
 MOV      @R0,A
 LCALL    _Music
 MOV      R7,#064H
 MOV      R6,#00H
 LCALL    _DelayX10ms
 SJMP     ?C0001
 RET

 END
```

2. input.a51

```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)
```

```
?PR?InputKey?INPUT          SEGMENT CODE
EXTRN     IDATA (KeyData)
PUBLIC    InputKey

RSEG  ?PR?InputKey?INPUT
USING     0
InputKey:
MOV       A,P1
CPL       A
ANL       A,#0FH
MOV       R0,#KeyData
MOV       @R0,A
RET

END
```

3. music.a51

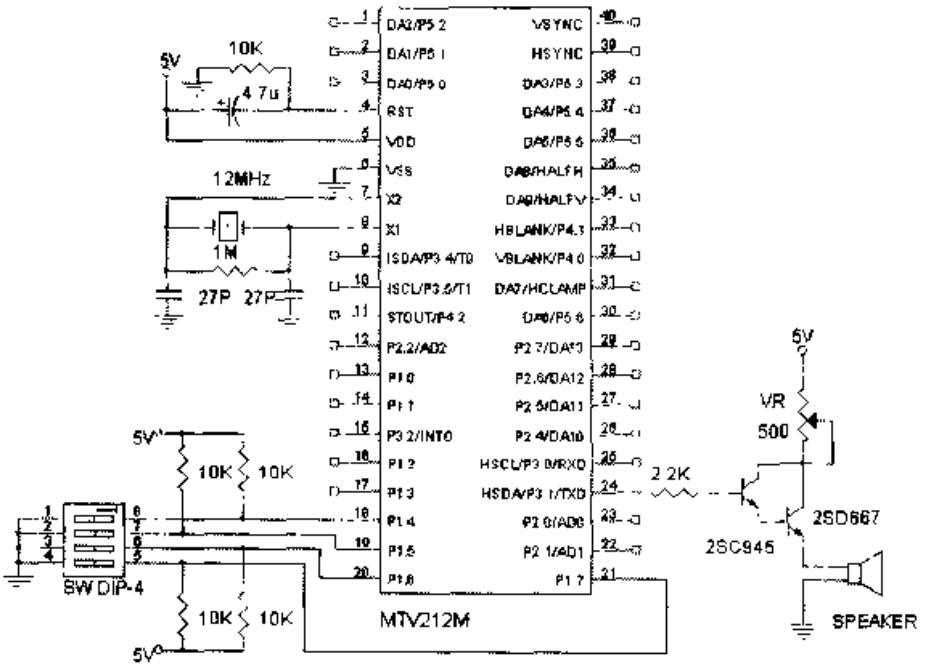
```
$NOMOD51

NAME MUSIC

$INCLUDE (mtv212n.inc)

?PR?_Music?MUSIC            SEGMENT CODE
EXTRN     CODE (Music1)
EXTRN     CODE (Music2)
EXTRN     CODE (Music3)
EXTRN     CODE (Music4)
EXTRN     CODE (Music5)
EXTRN     CODE (Music6)
EXTRN     CODE (Music7)
EXTRN     CODE (Music8)
EXTRN     CODE (Music9)
EXTRN     CODE (Music10)
EXTRN     CODE (Music11)
EXTRN     CODE (Music12)
EXTRN     CODE (Music13)
EXTRN     CODE (Music14)
EXTRN     CODE (Music15)
EXTRN     CODE (?C?CCASE)
PUBLIC    _Music

RSEG  ?PR?_Music?MUSIC
USING     0
```

```
_Music:
MOV       A,R7
LCALL     ?C?CCASE
DW    ?C0002
DB    01H
DW    ?C0003
DB    02H
DW    ?C0004
DB    03H
DW    ?C0005
DB    04H
DW    ?C0006
DB    05H
DW    ?C0007
DB    06H
DW    ?C0008
DB    07H
DW    ?C0009
DB    08H
DW    ?C0010
DB    09H
DW    ?C0011
DB    0AH
DW    ?C0012
DB    0BH
DW    ?C0013
DB    0CH
DW    ?C0014
DB    0DH
DW    ?C0015
DB    0EH
DW    ?C0016
DB    0FH
DW    00H
DW    ?C0018
?C0002:
LCALL     Music1
RET
?C0003:
LCALL     Music2
RET
?C0004:
LCALL     Music3
RET
?C0005:
```

```
    LCALL   Music4
    RET
?C0006:
    LCALL   Music5
    RET
?C0007:
    LCALL   Music6
    RET
?C0008:
    LCALL   Music7
    RET
?C0009:
    LCALL   Music8
    RET
?C0010:
    LCALL   Music9
    RET
?C0011:
    LCALL   Music10
    RET
?C0012:
    LCALL   Music11
    RET
?C0013:
    LCALL   Music12
    RET
?C0014:
    LCALL   Music13
    RET
?C0015:
    LCALL   Music14
    RET
?C0016:
    LCALL   Music15
?C0018:
    RET

    END
```

9-2-2　以高 4 位作为开关的选曲输入

电路图：



软件构建的思维与解决方法

由低4位转变为高4位作为指拨开关的4个输入，所有的模块、函数、变量都无需改变，只要将input.c模块下的侦测函数Inputkey( )进行改变而已。这是使用模块化结构和宏指令定义I/O地址或数值的最大好处，而无需个别去修改程序，由于是由P1.4～P1.7输入，也是“0”电位动作，所以当读取P1端口需反相，再和0Xf0作AND运算，以屏蔽低四位，而且向右移四个位，以使数值介于0～15之间，程序如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*******************************/
void InputKey(void)
{
   Byte  keytmp;
```

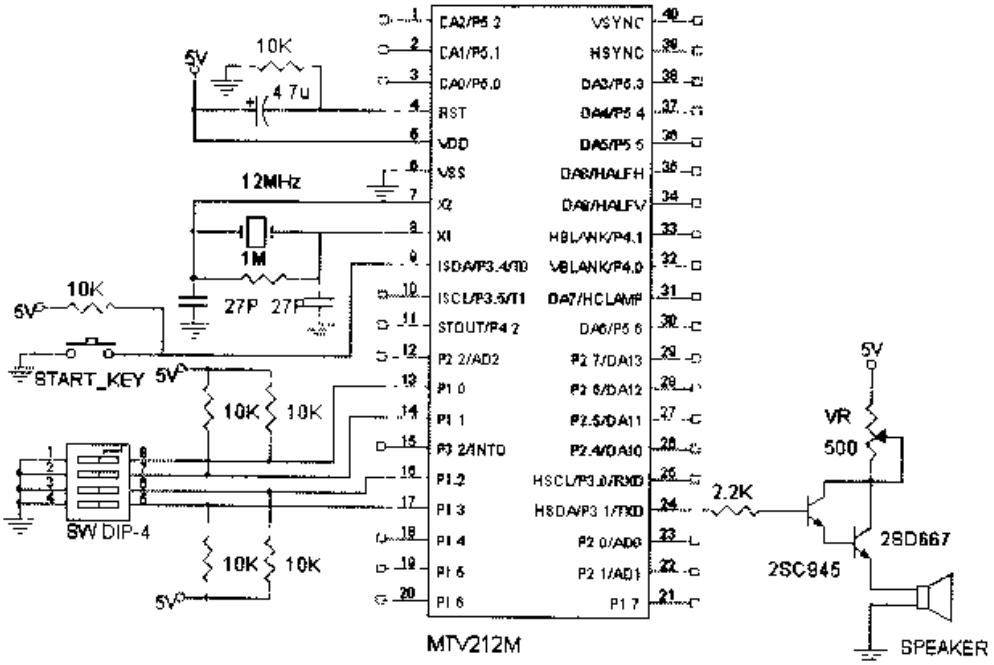
```
    keytmp = ~(INPUT_PORT) & 0xf0;    //P1.4~P1.7
    keytmp >>= 4;

    KeyData = keytmp;
}
```

汇编程序如下：

```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT          SEGMENT CODE
EXTRN    IDATA (KeyData)
PUBLIC   InputKey

RSEG  ?PR?InputKey?INPUT
USING    0
InputKey:
MOV      A,P1
CPL      A
ANL      A,#0F0H
MOV      R7,A
SWAP     A
ANL      A,#0FH
MOV      R7,A
MOV      R0,#KeyData
MOV      @R0,A
RET

END
```

9-2-3　以低 4 位作为输入应按下起始键才执行

电路图：



软件构建的思维与解决方法

要按下起始键才能执行演奏函数，则需定义起始键的 I/O 地址，并在 Main( )函数中不断地判断起始键是否按下，当按下时，其 I/O 引脚为“0”电位，而侦测开关输入的函数不再多使用一个局域变量了，将读入值作运算后直接设定给输入变量 KeyData 即可，如下：

（1）main.c：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*******************************/
void Main( void )
{
   PowerOnInit();
```

```
    while( 1 )
    {
      //sel play no#
      InputKey( );

      if ( START_IO==0 )
      {
        MusicNumber=KeyData;
        Music(MusicNumber);
        DelayX10ms(100);
      }
    }
}
```

（2）input.c

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*****************************/
void InputKey(void)
{
   KeyData = ~(INPUT_PORT) & 0x0f;   //P1.0~P1.3
}
```

（3）input.h：定义起始键 I/O 为 P3.4 如下：

```
#ifndef   _INPUT_H
#define   _INPUT_H

#define  INPUT_PORT          P1
#define  START_IO       P3_4
#define  NO_KEY              0

extern void InputKey(void);

#endif
```

汇编程序如下：

1. main.a51

```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN                SEGMENT CODE
 EXTRN    IDATA (KeyData)
 EXTRN    IDATA (MusicNumber)
 EXTRN    CODE (PowerOnInit)
 EXTRN    CODE (_DelayX10ms)
 EXTRN    CODE (_Music)
 EXTRN    CODE (InputKey)
 EXTRN    CODE (?C_STARTUP)
 PUBLIC   Main

 RSEG  ?PR?Main?MAIN
 USING    0
Main:
 LCALL    PowerOnInit
?C0001:
 LCALL    InputKey
 JB   P3_4,?C0001
 MOV      R0,#KeyData
 MOV      A,@R0
 MOV      R7,A
 MOV      R0,#MusicNumber
 MOV      @R0,A
 LCALL    _Music
 MOV      R7,#064H
 MOV      R6,#00H
 LCALL    _DelayX10ms
 SJMP     ?C0001
 RET

 END
```

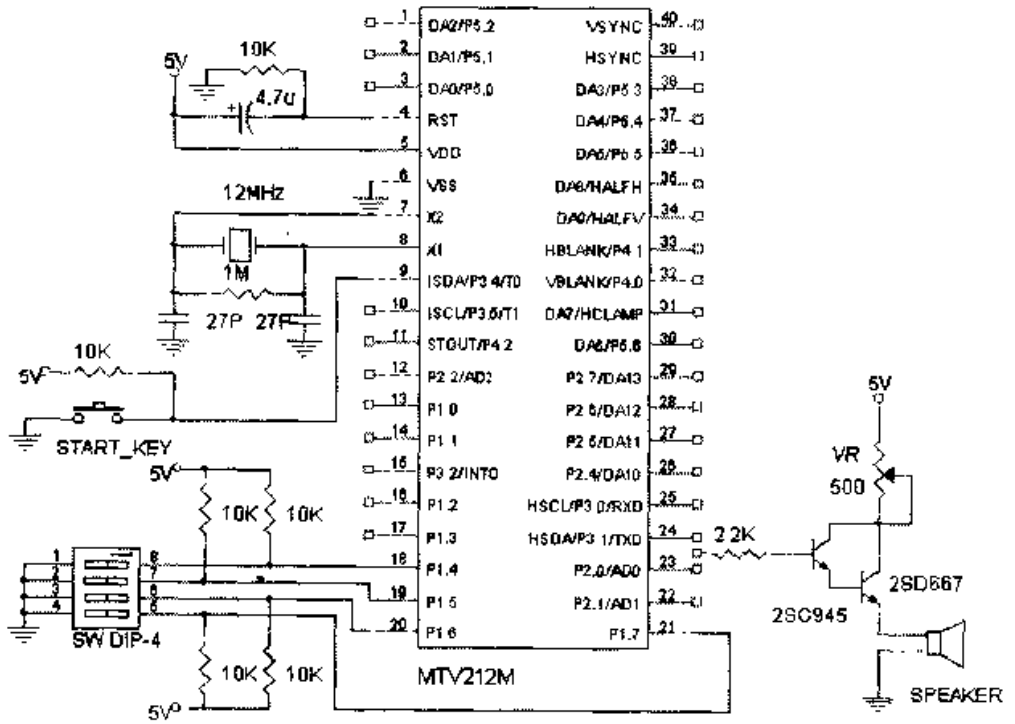
2. input.a51

```
$NOMOD51
```

```
NAME INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT          SEGMENT CODE
 EXTRN     IDATA (KeyData)
 PUBLIC    InputKey

 RSEG  ?PR?InputKey?INPUT
 USING     0
InputKey:
 MOV       A,P1
 CPL       A
 ANL       A,#0FH
 MOV       R0,#KeyData
 MOV       @R0,A
 RET

 END
```

9-2-4　以高 4 位作为输入应按下起始键才执行

电路图：



软件构建的思维与解决方法

硬件电路将开关由原来的 P1.0～P1.3，改变为 I/O P1.4～P1.7 作为输入，原来的起始键一样由 P3.4 作输入，则软件只改变侦测指拨开关的函数，即 InputKey( ) 函数，并将读入值作运算后直接赋给 KeyData 变量，其余模块下的函数都无需改变，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*****************************/
void InputKey(void)
{
   KeyData = ( ~(INPUT_PORT)&0xf0 ) >> 4;     //P1.4~P1.7
}
```

汇编程序如下：

1. input.a51

```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT          SEGMENT CODE
EXTRN     IDATA (KeyData)
PUBLIC    InputKey

RSEG  ?PR?InputKey?INPUT
USING     0
InputKey:
MOV       A,P1
CPL       A
ANL       A,#0F0H
MOV       R7,A
```

```
SWAP    A
ANL     A,#0FH
MOV     R0,#KeyData
MOV     @R0,A
RET

END
```

9-2-5　以高 4 位作为输入，如未选曲，即使按下起始键也不执行

软件构建的思维与解决方法

当四个指拨开关都未拨至“ON”位置，其 P1.4～P1.7 的输入信号为“1”电位，经过反相并和 0Xf0 作 AND 运算，其最后 KeyData 变量的数值将为 0X00，而在 Main( ) 函数内，除了判断 START_IO 等于“0”电位外，必须再判断 KeyData 不能等于 0X00，当两者条件都成立才能执行音乐函数 Music( )，而 NO_KEY 的定义必须为 0X00 才可以，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      //sel play no#
      InputKey( );

      if ( START_IO==0 )
      {
         if ( KeyData != NO_KEY )
         {
```

```
            MusicNumber=KeyData;
            Music(MusicNumber);
            DelayX10ms(100);
          }
        }
    }
}
```

汇编程序如下：

1. main.a51

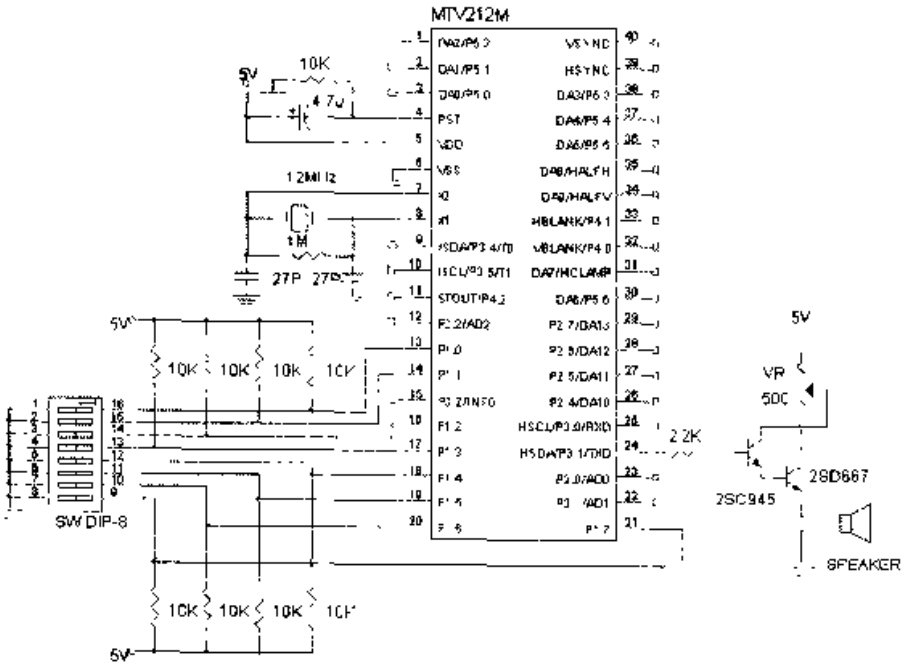
```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN                SEGMENT CODE
 EXTRN    IDATA (KeyData)
 EXTRN    IDATA (MusicNumber)
 EXTRN    CODE (PowerOnInit)
 EXTRN    CODE (_DelayX10ms)
 EXTRN    CODE (_Music)
 EXTRN    CODE (InputKey)
 EXTRN    CODE (?C_STARTUP)
 PUBLIC   Main

 RSEG  ?PR?Main?MAIN
 USING    0
Main:
 LCALL    PowerOnInit
?C0001:
 LCALL    InputKey
 JB   P3_4,?C0001
 MOV      R0,#KeyData
 MOV      A,@R0
 MOV      R7,A
 JZ   ?C0001
 MOV      R0,#MusicNumber
 MOV      @R0,A
 LCALL    _Music
 MOV      R7,#064H
 MOV      R6,#00H
 LCALL    _DelayX10ms
```

```
SJMP      ?C0001
RET

END
```

9-3　DIPX8：1～255 曲

目的：以 8P 的指拨开关作为选曲，共 1～255 曲。

电路图：



原理：八个输入的指拨开关正好是一个端口，八支 I/O，由于具有八个输入，而每一个开关的状态为“0”或“1”，即两种状态，因此，总共有 2^8=256 种不同的组合，其数值为 0～255。平常由于提升电阻 10KΩ所致，使得 P1.0～P1.7 都为“1”电位，当任何一个开关拨至“ON”位置，则其输入的 I/O 脚将转变为“0”电位，以此电位的转变即可侦测出其输入的数值了。

9-3-1　根据 DIP 开关立即执行音乐演奏函数

1. 软件构建的思维

八个输入正好使用一个端口，在 InputKey( )侦测函数中，读取端口的内容就无

需作 AND 和向右移位的运算，只要读取 P1 并反相后，即可写入至 KeyData 变量内。而音乐演奏函数 Music( )的条件值照理应该要写到 255，即可演奏 1～255 首曲目，但本范例主要是阐述软件构建的思维与解决方法，因此，只示范曲目 1～31 首而已，但其原理、构思可扩充至 255 首，现分述如下：

（1）input.c：直接读取 P1 端口即可。

（2）music.c：因为只有曲目 1～31 首，则 switch...case 的条件值只列出至 31，如果要增加曲目至 50 首，则要修正下列事项：

- Music( )函数 case 条件值要从 1 写至 50，则也要调用对应的演奏音乐函数，例如：case1，要调用 music1( )函数，case11，要调用 music11( )函数，case21，则要调用 music21( )函数，依此类推。
- Music.h 包括文件要增加 music32( )函数～music50( )函数的外部声明。
- 要编写 music32.c～music50.c 的音乐演奏模块。
- 在 keil_c 开发环境下，要将项目文件（例如：Demo.Prj）添加 music32.c～music50.c 的模块，并重新编译和连接。

（3）music.h：music1( )～music31( )函数作外部声明

2. 软件的解决方法

（1）input.c：直接读取 P1 端口，INPUT_PORT 定义为 P1 端口，如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*******************************/
void InputKey(void)
{
   KeyData = ~(INPUT_PORT);      //P1.0~P1.7
}
```

（2）music.c：调用音乐演奏函数 music1( )～music31( )的写法如下：

```
/*******************************/
/* include files               */
/*******************************/
```

```
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"

/*******************************/
void Music(Byte number)
{
    switch( number )
    {
     case 1 :
        Music1( );
        break;
     case 2 :
        Music2( );
        break;
     case 3 :
        Music3( );
        break;
     case 4 :
        Music4( );
        break;
     case 5 :
        Music5( );
        break;
     case 6 :
        Music6( );
        break;
     case 7 :
        Music7( );
        break;
     case 8 :
        Music8( );
        break;
     case 9 :
        Music9( );
        break;
     case 10 :
        Music10( );
        break;
     case 11 :
        Music11( );
        break;
```

```
case 12 :
   Music12( );
   break;
case 13 :
   Music13( );
   break;
case 14 :
   Music14( );
   break;
case 15 :
   Music15( );
   break;
case 16 :
   Music16( );
   break;
case 17 :
   Music17( );
   break;
case 18 :
   Music18( );
   break;
case 19 :
   Music19( );
   break;
case 20 :
   Music20( );
   break;
case 21 :
   Music21( );
   break;
case 22 :
   Music22( );
   break;
case 23 :
   Music23( );
   break;
case 24 :
   Music24( );
   break;
case 25 :
   Music25( );
   break;
case 26 :
   Music26( );
   break;
```

```
    case 27 :
       Music27( );
       break;
    case 28 :
       Music28( );
       break;
    case 29 :
       Music29( );
       break;
    case 30 :
       Music30( );
       break;
    case 31 :
       Music31( );
       break;
    default :
       break;
   }
}
```

（3）music.h：music( )、music1( )～music31( )函数的外部声明，如下：

```
extern void Music(Byte number);
extern void Music1(void);
extern void Music2(void);
extern void Music3(void);
extern void Music4(void);
extern void Music5(void);
extern void Music6(void);
extern void Music7(void);
extern void Music8(void);
extern void Music9(void);
extern void Music10(void);
extern void Music11(void);
extern void Music12(void);
extern void Music13(void);
extern void Music14(void);
extern void Music15(void);
extern void Music16(void);
extern void Music17(void);
extern void Music18(void);
extern void Music19(void);
extern void Music20(void);
extern void Music21(void);
extern void Music22(void);
```

```
extern void Music23(void);
extern void Music24(void);
extern void Music25(void);
extern void Music26(void);
extern void Music27(void);
extern void Music28(void);
extern void Music29(void);
extern void Music30(void);
extern void Music31(void);
```

而音乐演奏模块 music16.c～music31.c 其曲名和程序如下：

Music16.c　曲名：再会吧!原野

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG16[ ] =
{
    _4,_8 ,_8 ,_8 ,_8 ,_8 ,_8 ,_4,_4,_2,
    _4,_4 ,_4 ,_8 ,_8 ,_4 ,_8 ,_8,_2,
    _4,_8 ,_8 ,_4 ,_8 ,_8 ,_4 ,_8,_8,_2,
    _4,_8 ,_8 ,_4d,_8 ,_8 ,_8 ,_8,_8,_2,
    _4,_2d,_4d,_8 ,_8 ,_4d,_4 ,_8,_8,_4,_4,_1,
    _EOF
};

Word RDATA SOUNDTONE16[ ] =
{
    _DO ,_DO ,_RE ,_MI ,_RE ,_DO ,_1SO,_1LA,_DO ,_1SO,
    _DO ,_RE ,_MI ,_RE ,_DO ,_RE ,_RE ,_MI ,_RE ,
    _1SO,_1SO,_1SO,_DO ,_1SO,_1SO,_1LA,_1LA,_1LA,_RE ,
    _1TI,_1TI,_1TI,_1TI,_1LA,_1SO,_1SO,_1LA,_1TI,_DO ,
    _MI ,_SO ,_LA ,_LA ,_SO ,_MI ,_1SO,_1SO,_1SO,_1TI,_RE,_DO,
    _EOF
};

void Music16(void)
{
```

```
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG16[i]!=_EOF)||(SOUNDTONE16[i]!=_EOF) )
    {
       //start timer1,generate soundtone
       Period=SOUNDTONE16[i];

       TL1 = (65536 - Period) & 0xff;
       TH1 = (65536 - Period) >> 8;
       TR1 = 1;

       SoundLongCount=((Tempo*SOUNDLONG16[i]*1000L)/8)/Period;
       while ( SoundLongCount != 0 );

       TR1 = 0;      //stop timer1:soundtone
       i++;
    }
}
```

Music17.c　曲名：秋蝉

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG17[ ] =
{
    _4,_4,_4,_4,_4,_4,_1,
    _4,_4,_4,_4,_4,_4,_1,
    _4,_4,_4,_4,_4,_4,_4,_4,_4,_1,
    _2,_4,_4,_4,_4,_4,_4,_4,_1,

    _4,_4,_4,_4,_4,_4,_1,
    _4,_4,_4,_4,_4,_4,_1,
    _4,_4,_4,_4,_4,_4,_4,_4,_4,_1,
    _2,_4,_4,_4,_4,_4,_4,_4,_1,

    _4,_8,_8,_4,_2,_4,_8,_8,_4,_2,
```

```
    _4,_4,_4,_4,_4,_4,_2,_4,_2,
    _2,_4,_2,_4,_2,_4,_2,_4,
    _2,_4,_4,_4,_4,_4,_4,_4,_2,
    _2,_4,_4,_4,_4,_4,_4,_4,_2,
    _EOF
};

Word RDATA SOUNDTONE17[ ] =
{
    _SO ,_SO,_SO,_LA,_DO1,_LA,_SO ,
    _MI ,_MI,_RE,_MI,_RE ,_DO,_1LA,
    _1SO,_SO,_SO,_LA,_SO ,_SO,_MI ,_RE,_MI,_MI,
    _RE ,_RE,_RE,_MI,_RE ,_DO,_1LA,_DO,_DO,

    _SO ,_SO,_SO,_LA,_DO1,_LA,_SO ,
    _MI ,_MI,_RE,_MI,_RE ,_DO,_1LA,
    _1SO,_SO,_SO,_LA,_SO ,_SO,_MI ,_RE,_MI,_MI,
    _RE ,_RE,_RE,_MI,_RE ,_DO,_1LA,_DO,_DO,

    _DO ,_RE,_RE,_MI,_1LA,_DO,_RE ,_RE,_MI,_1SO,
    _DO ,_RE,_MI,_FA,_MI ,_FA,_LA ,_SO,_SO,
    _SO ,_MI,_SO,_MI,_DO1,_LA,_DO1,_LA,
    _SO ,_SO,_SO,_LA,_SO ,_FA,_SO ,_FA,_MI,
    _SO ,_SO,_SO,_LA,_SO ,_FA,_SO ,_FA,_MI,
    _EOF
};

void Music17(void)
{
   Byte i=0;

   Tempo = MODERATO;

   while ( (SOUNDLONG17[i]!=_EOF)||(SOUNDTONE17[i]!=_EOF) )
   {
      //start timer1,generate soundtone
      Period=SOUNDTONE17[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG17[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );
```

```
        TR1 = 0;      //stop timer1:soundtone
        i++;
    }
}
```

Music18.c　曲名：恭喜!恭喜!

```
/*****************************/
/* include files              */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG18[ ] =
{
    _8d,_16,_8,_8,_8,_8,_8,_8,
    _8d,_16,_8,_8,_8,_8,_8,

    _8 ,_8 ,_8,_8,_8,_8,_4,
    _8 ,_8 ,_8,_8,_8,_8,_4,
    _8 ,_8 ,_8,_8,_8,_8,_4,
    _8 ,_8 ,_8,_8,_4,_4,
    _8d,_16,_8,_8,_8,_8,_8,_8,
    _8d,_16,_8,_8,_8,_8,_4,
    _EOF
};

Word RDATA SOUNDTONE18[ ] =
{
    _RE ,_MI ,_DO ,_MI ,_1TI,_MI ,_1LA,_MI,
    _RE ,_MI ,_DO ,_MI ,_1TI,_MI ,_1LA,

    _1LA,_1TI,_DO ,_RE ,_FA ,_MI ,_MI ,
    _MI ,_LA ,_LA ,_MI ,_MI ,_RE ,_RE ,
    _RE ,_FA ,_MI ,_RE ,_RE ,_DO ,_DO ,
    _DO ,_1TI,_1LA,_1SOr,_1LA,_1LA,
    _RE ,_MI ,_DO ,_MI ,_1TI,_MI ,_1LA,_MI,
    _RE ,_MI ,_DO ,_MI ,_1TI,_MI ,_1LA,
    _EOF
};

void Music18(void)
```

```
{
    Byte i=0;

    Tempo = ANDANTINO;

    while ( (SOUNDLONG18[i]!=_EOF)||(SOUNDTONE18[i]!=_EOF) )
    {
       //start timer1,generate soundtone
       Period=SOUNDTONE18[i];

       TL1 = (65536 - Period) & 0xff;
       TH1 = (65536 - Period) >> 8;
       TR1 = 1;

       SoundLongCount=((Tempo*SOUNDLONG18[i]*1000L)/8)/Period;
       while ( SoundLongCount != 0 );

       TR1 = 0;      //stop timer1:soundtone
       i++;
    }
}
```

Music19.c 曲名：看我!听我

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG19[ ] =
{
    _2d,_4,_2d,_2d,_4 ,_2d,_8,
    _4 ,_4,_8 ,_4d,_2d,_8 ,
    _4 ,_4,_8 ,_4d,_2d,_4 ,_1,

    _2d,_4,_2d,_2d,_4 ,_2d,_8,
    _4 ,_4,_8 ,_4d,_2d,_8 ,
    _4 ,_4,_8 ,_4d,_1 ,

    _4 ,_4,_2 ,_8 ,_4d,_2 ,_4,_4,
    _4 ,_4,_8 ,_4d,_2 ,_4 ,_4,
```

```
    _2 ,_8,_4d,_2 ,_4 ,_4 ,
    _4 ,_4,_8 ,_4d,_2 ,_4 ,_4,_2,
    _8 ,_4 ,_4,_8 ,_4d,_2 ,
    _EOF
};

Word RDATA SOUNDTONE19[ ] =
{
    _SO ,_LA ,_MI ,_SO ,_LA ,_MI ,_MI ,
    _RE ,_DO ,_1LA,_DO ,_MI ,_MI ,
    _RE ,_DO ,_1LA,_DO ,_RE ,_DO ,_RE ,

    _SO ,_LA ,_MI ,_SO ,_LA ,_MI ,_MI ,
    _RE ,_DO ,_1LA,_DO ,_MI ,_MI ,
    _RE ,_DO ,_1LA,_1SO,_DO ,

    _DO ,_1TI,_1LA,_1LA,_1TI,_1LA,_1LA,_1TI,
    _DO ,_MI ,_RE ,_SO ,_MI ,_DO ,_1TI,
    _1LA,_1LA,_1TI,_1LA,_1LA,_1TI,
    _DO ,_MI ,_RE ,_MI ,_RE ,_SO ,_LA ,_MI,
    _DO ,_RE ,_DO ,_1LA,_1SO,_DO ,
    _EOF
};

void Music19(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG19[i]!=_EOF)||(SOUNDTONE19[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE19[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG19[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
```

```
}
```

Music20.c 曲名：月朦胧鸟朦胧

```
/*****************************/
/* include files             */
/*****************************/
#include "define.h"
#include "mtv212m.h"
#include "global.h"
#include "music.h"

/*****************************/
Byte RDATA SOUNDLONG20[ ] =
{
    _2,_4,_2,_4,_2,_4,_2d,
    _2,_4,_4,_4,_4,_1,
    _2,_4,_2,_8,_8,_2,_8 ,_8,_2d,
    _2,_4,_2,_4,_1,
    _EOF
};

Word RDATA SOUNDTONE20[ ] =
{
    _1SO,_DO,_MI,_DO,_1LA,_DO,_1SO,
    _MI ,_MI,_MI,_RE,_DO ,_RE,
    _SO ,_SO,_SO,_LA,_SO ,_MI,_RE ,_DO,_1LA,
    _SO ,_SO,_MI,_RE,_DO ,
    _EOF
};

void Music20(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG20[i]!=_EOF)||(SOUNDTONE20[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE20[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;
```

```
        SoundLongCount=((Tempo*SOUNDLONG20[i]*1000L)/8)/Period;
        while ( SoundLongCount != 0 );

        TR1 = 0;      //stop timer1:soundtone
        i++;
    }
}
```

Music21.c　曲名：四季谣

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG21[ ] =
{
    _4 ,_4,_4d,_8 ,_8 ,_8,_8,_8,_2,
    _4 ,_8,_8 ,_8 ,_8 ,_8,_8,_8,_8,_4,_4,

    _4 ,_4,_4d,_8 ,_8 ,_8,_8,_8,_2,
    _4 ,_4,_4d,_8 ,_4 ,_8,_8,_2,
    _4 ,_8,_8 ,_4 ,_4 ,_8,_8,_8,_8,_2,
    _4 ,_8,_8 ,_4 ,_8 ,_8,_2,
    _4 ,_8,_8 ,_4 ,_8 ,_8,_8,_8,_4,
    _4 ,_8,_8 ,_4d,_8 ,_4,_8,_8,_2,
    _4d,_8,_4d,_8 ,_4d,_8,_8,_8,_8,_8,_2,
    _EOF
};

Word RDATA SOUNDTONE21[ ] =
{
    _1SO,_DO ,_DO ,_RE ,_MI ,_RE ,_DO,_RE ,_MI ,
    _SO ,_LA ,_SO ,_LA ,_SO ,_MI ,_RE,_RE ,_MI ,_DO ,_DO,

    _1SO,_DO ,_DO ,_RE ,_MI ,_RE ,_DO,_RE ,_MI ,
    _1SO,_MI ,_MI ,_FA ,_MI , RE ,_DO,_RE ,
    _DO ,_RE ,_MI ,_RE ,_DO ,_RE ,_DO,_1TI,_1LA,_1SO,
    _1MI,_1SO,_1LA,_1SO, RE ,_1LA,_DO,
    _SO ,_RE ,_MI ,_SO ,_SO ,_SO ,_RE,_MI ,_RE ,
```

```
    _DO ,_1LA,_DO ,_1SO,_1LA,_1SO,_RE,_MI ,_DO ,
    _SO ,_SO ,_LA ,_LA ,_SO ,_LA ,_SO,_MI ,_RE ,_MI ,_SO,
    _EOF
};

void Music21(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG21[i]!=_EOF)||(SOUNDTONE21[i]!=_EOF) )
    {
       //start timer1,generate soundtone
       Period=SOUNDTONE21[i];

       TL1 = (65536 - Period) & 0xff;
       TH1 = (65536 - Period) >> 8;
       TR1 = 1;

       SoundLongCount=((Tempo*SOUNDLONG21[i]*1000L)/8)/Period;
       while ( SoundLongCount != 0 );

       TR1 = 0;      //stop timer1:soundtone
       i++;
    }
}
```

Music22.c 曲名：童年

```
/*****************************/
/* include files             */
/*****************************/
#include "define.h"
#include "mtv212m.h"
#include "global.h"
#include "music.h"

/*****************************/
Byte RDATA SOUNDLONG22[ ] =
{
  _8,_4,_8,_4,_4,_8,_4,_8,_8, 8,_4,
  _8,_4,_8,_8,_8,_8,_8,_2,

  _8,_8,_4,_8,_8,_8,_8,_8,_4,_8,_8,_8,
```

```
  _4,_8,_8,_8,_8,_8,_8,_2,
  _8,_8,_4,_8,_8,_8,_8,_8,_4,_8,_8,_8,
  _4,_8,_8,_8,_8,_8,_8,_2,
  _8,_4,_8,_4,_8,_8,_8,_4,_4,_8,_8,_8,
  _8,_8,_8,_8,_8,_8,_8,_8,_2,
  _8,_4,_8,_4,_4,_8,_4,_8,_8,_8,_4,
  _8,_4,_8,_8,_8,_8,_8,_2,
  _8,_4,_8,_4,_4,_8,_4,_8,_8,_8,_4,
  _8,_4,_8,_8,_8,_8,_8,_2,
  _EOF
};

Word RDATA SOUNDTONE22[ ] =
{
  _MI1,_MI1,_MI1,_MI1,_RE1,_DO1,_DO1,_DO1,_RE1,_DO1,_LA,
  _SO ,_SO ,_SO ,_LA ,_SO ,_RE1,_MI1,_DO1,

  _MI ,_SO ,_SO ,_MI ,_SO ,_LA ,_LA ,_TI ,_LA ,_LA ,_LA,_SO ,
  _DO1,_DO1,_DO1,_DO1,_LA ,_DO1,_LA ,_SO ,
  _MI ,_SO ,_SO ,_MI ,_SO ,_LA ,_LA ,_TI ,_LA ,_LA ,_LA,_SO ,
  _DO1,_DO1,_DO1,_DO1,_LA ,_LA ,_DO1,_RE1,
  _SO1,_SO1,_SO1,_SO1,_MI1,_RE1,_DO1,_DO1,_LA ,_DO1,_LA,_DO1,
  _RE1,_RE1,_RE1,_RE1,_RE1,_DO1,_MI1,_RE1,_RE1,
  _MI1,_MI1,_MI1,_MI1,_RE1,_DO1,_DO1,_DO1,_RE1,_DO1,_LA,
  _SO ,_SO ,_SO ,_LA ,_SO ,_RE1,_MI1,_DO1,
  _MI1,_MI1,_MI1,_MI1,_RE1,_DO1,_DO1,_DO1,_RE1,_DO1,_LA,
  _SO ,_SO ,_SO ,_LA ,_SO ,_RE1,_MI1,_DO1,
  _EOF
};

void Music22(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG22[i]!=_EOF)||(SOUNDTONE22[i]!=_EOF) )
    {
       //start timer1,generate soundtone
       Period=SOUNDTONE22[i];

       TL1 = (65536 - Period) & 0xff;
       TH1 = (65536 - Period) >> 8;
       TR1 = 1;
```

```
        SoundLongCount=((Tempo*SOUNDLONG22[i]*1000L)/8)/Period;
        while ( SoundLongCount != 0 );

        TR1 = 0;      //stop timer1:soundtone
        i++;
    }
}
```

Music23.c 曲名：农村曲

```
/*****************************/
/* include files             */
/*****************************/
#include "define.h"
#include "mtv212m.h"
#include "global.h"
#include "music.h"

/*****************************/
Byte RDATA SOUNDLONG23[ ] =
{
    _8,_8 ,_4 ,_8,_8,_4,
    _4,_16,_16,_8,_8,_16,_16,_4,
    _4,_8d,_16,_16,_16,_8,_4,
    _8,_4,_8,_4 ,_16,_16,_16,_16,_4,
    _8,_8,_8,_8,_2,
    _8,_8,_8,_8,_2,
    _8,_8,_4,_8,_4,_8,
    _4,_8,_8,_4,_16,_16,_16,_16,_2,
    _EOF
};

Word RDATA SOUNDTONE23[ ] =
{
    _MI1,_LA ,_DO1,_RE1,_SO1,_MI1,
    _SO1,_LA1,_SO1,_MI1,_RE1,_RE1,_MI1,_SO1,
    _LA ,_DO1,_MI1,_RE1,_DO1,_LA ,_DO1,
    _SO ,_SO ,_LA ,_DO1,_DO1,_SO ,_LA ,_DO1,_SO,
    _DO1,_DO1,_DO1,_RE1,_MI1,
    _MI1,_SO1,_DO2,_LA1,_SO1,
    _DO2,_LA1,_SO1,_MI1,_RE1,_MI1,
    _SO1,_SO1,_LA ,_DO1,_RE1,_DO1,_RE1,_MI1,_DO1,
    _EOF
};
```

```
void Music23(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG23[i]!=_EOF)||(SOUNDTONE23[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE23[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG23[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

Music24.c　曲名：踏雪寻梅

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG24[ ] =
{
    _4 ,_4,_4 ,_8,_8 ,_2 ,
    _4 ,_4,_4 ,_8,_8 ,_2d,_8,_8,
    _2d,_4,_4 ,_4,_2 ,
    _4 ,_4,_4d,_8,_2 ,
    _4 ,_4,_2 ,_4,_4 ,_2 ,
    _4 ,_4,_2 ,_4,_4 ,_2 ,
    _4 ,_4,_4 ,_2,_4d,_8 ,_4,_4,_2,
    _4 ,_8,_8 ,_4,_4 ,_2,_2,
    _4 ,_4,_4d,_8,_2 ,
```

```
    _EOF
};

Word RDATA SOUNDTONE24[ ] =
{
    _MI ,_SO,_SO,_DO ,_RE ,_MI ,
    _MI ,_LA,_SO,_DO ,_RE ,_MI ,_MI,_SO,
    _DO1,_TI,_MI,_LA ,_SO ,
    _1SO,_MI,_RE,_DO ,_DO ,
    _MI ,_SO,_SO,_RE ,_SO ,_SO ,
    _MI ,_SO,_SO,_DO ,_DO1,_DO1,
    _DO ,_MI,_SO,_DO1,_TI ,_LA ,_MI,_LA,_SO,
    _SO ,_DO,_RE,_MI ,_FA ,_SO ,_SO,
    _1SO,_MI,_RE,_DO ,_DO ,
    _EOF
};

void Music24(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG24[i]!=_EOF)||(SOUNDTONE24[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE24[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG24[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

Music25.c 曲名：郊游

```
/*****************************/
/* include files             */
/*****************************/
```

```
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG25[ ] =
{
    _4 ,_4,_4,_4,_4,
    _4d,_8,_4,_4,_4,_4,_4,
    _4 ,_4,_4,_4,_4,
    _4 ,_8,_8,_4,_4,_2,
    _EOF
};

Word RDATA SOUNDTONE25[ ] =
{
    _RE1,_RE1,_RE1,_TI,_RE1,
    _RE1,_TI ,_SO ,_TI,_LA ,_SO,_RE,
    _MI ,_SO ,_RE ,_SO,_TI ,
    _RE1,_SO ,_LA ,_TI,_LA ,_SO,
    _EOF
};

void Music25(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG25[i]!=_EOF)||(SOUNDTONE25[i]!=_EOF) )
    {
       //start timer1,generate soundtone
       Period=SOUNDTONE25[i];

       TL1 = (65536 - Period) & 0xff;
       TH1 = (65536 - Period) >> 8;
       TR1 = 1;

       SoundLongCount=((Tempo*SOUNDLONG25[i]*1000L)/8)/Period;
       while ( SoundLongCount != 0 );

       TR1 = 0;      //stop timer1:soundtone
       i++;
    }
```

```
}
```

Music26.c 曲名：青海青

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG26[ ] =
{
    _8,_8  ,_8d,_16,_16,_16,_16,_16,_4,
    _8,_16 ,_16,_8 ,_16,_16,_16,_16,_16,_16,_4,
    _8,_16 ,_16,_8d,_16,_16,_16,_16,_16,_4,
    _16,_16,_16,_16,_16,_16,_16,_16,_2,
    _EOF
};

Word RDATA SOUNDTONE26[ ] =
{
    _RE,_RE,_LA,_SO ,_LA ,_DO1,_LA ,_SO ,_LA,
    _SO,_FA,_SO,_LA ,_DO1,_DO1,_RE1,_DO1,_LA,_SO,_LA,
    _SO,_SO,_LA,_DO1,_LA ,_SO ,_LA ,_SO ,_FA,_RE,
    _FA,_FA,_FA,_FA ,_MI ,_MI ,_DO ,_DO ,_RE,
    _EOF
};

void Music26(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG26[i]!=_EOF)||(SOUNDTONE26[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE26[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;
```

```
        SoundLongCount=((Tempo*SOUNDLONG26[i]*1000L)/8)/Period;
        while ( SoundLongCount != 0 );

        TR1 = 0;      //stop timer1:soundtone
        i++;
    }
}
```

Music27.c　曲名：造飞机

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG27[ ] =
{
    _4,_8,_8,_4,_8 ,_8,
    _8,_8,_8,_8,_4d,
    _4,_8,_8,_4,_8 ,_8,
    _8,_8,_8,_8,_4d,
    _4,_8,_8,_8,_8 ,_4,
    _8,_8,_8,_8,_4d,
    _4,_8,_8,_4,_8 ,_8,
    _8,_8,_8,_8,_4d,
    _4,_8,_8,_4,_8 ,_8,
    _8,_8,_8,_8,_4d,
    _EOF
};

Word RDATA SOUNDTONE27[ ] =
{
    _DO1,_LA ,_TIf,_DO1,_LA ,_TIf,
    _DO1,_DO1,_RE1,_RE1,_DO1,
    _LA ,_SO ,_FA ,_LA ,_SO ,_FA ,
    _RE ,_RE ,_FA ,_FA ,_DO ,
    _RE ,_FA ,_FA ,_DO ,_DO ,_FA ,
    _SO ,_SO ,_FA ,_SO ,_LA ,
    _DO1,_LA ,_TIf,_DO1,_LA ,_TIf,
    _DO1,_DO1,_RE1,_RE1,_DO1,
```

```
    _RE1,_RE1,_DO1,_LA ,_SO ,_FA ,
    _SO ,_SO ,_DO1,_DO1,_FA ,
    _EOF
};

void Music27(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG27[i]!=_EOF)||(SOUNDTONE27[i]!=_EOF) )
    {
       //start timer1,generate soundtone
       Period=SOUNDTONE27[i];

       TL1 = (65536 - Period) & 0xff;
       TH1 = (65536 - Period) >> 8;
       TR1 = 1;

       SoundLongCount=((Tempo*SOUNDLONG27[i]*1000L)/8)/Period;
       while ( SoundLongCount != 0 );

       TR1 = 0;      //stop timer1:soundtone
       i++;
    }
}
```

Music28.c 曲名：小毛驴

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG28[ ] =
{
    _8,_8,_8,_8,_8 ,_8 ,_8 ,_8 ,
    _8,_8,_8,_8,_2 ,
    _8,_8,_8,_8,_8 ,_8 ,_8 ,_8 ,
    _8,_8,_8,_8,_4 ,_8 ,
```

```
    _8,_8,_8,_8,_8 ,_8 ,_8 ,_8 ,
    _8,_8,_8,_8,_2 ,
    _8,_8,_8,_8,_16,_16,_16,_16,_8,_8,
    _8,_8,_8,_8,_2,
    _EOF
};

Word RDATA SOUNDTONE28[ ] =
{
    _DO,_DO,_DO,_MI ,_SO,_SO,_SO,_SO,
    _LA,_LA,_LA,_DO1,_SO,
    _FA,_FA,_FA,_LA ,_MI,_MI,_MI,_MI,
    _RE,_RE,_RE,_RE ,_SO,_SO,
    _DO,_DO,_DO,_MI ,_SO,_SO,_SO,_SO,
    _LA,_LA,_LA,_DO1,_SO,
    _FA,_FA,_FA,_LA ,_MI,_MI,_MI,_MI,_MI,_MI,
    _RE,_RE,_RE,_MI ,_DO,
    _EOF
};

void Music28(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG28[i]!=_EOF)||(SOUNDTONE28[i]!=_EOF) )
    {
       //start timer1,generate soundtone
       Period=SOUNDTONE28[i];

       TL1 = (65536 - Period) & 0xff;
       TH1 = (65536 - Period) >> 8;
       TR1 = 1;

       SoundLongCount=((Tempo*SOUNDLONG28[i]*1000L)/8)/Period;
       while ( SoundLongCount != 0 );

       TR1 = 0;      //stop timer1:soundtone
       i++;
    }
}
```

Music29.c 曲名：我爱乡村

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG29[ ] =
{
    _4,_8,_8,_4,_4,
    _8,_8,_8,_8,_2,
    _4,_8,_8,_8,_8,_4,
    _8,_8,_8,_8,_2,
    _4,_8,_8,_8,_8,_4,
    _8,_8,_8,_8,_2,
    _4,_8,_8,_4,_4,
    _8,_8,_8,_8,_2,
    _EOF
};

Word RDATA SOUNDTONE29[ ] =
{
    _MI ,_DO ,_MI,_LA,_SO,
    _MI ,_MI ,_RE,_DO,_RE,
    _MI ,_DO ,_MI,_LA,_LA,_SO ,
    _MI ,_MI ,_RE,_RE,_DO,
    _RE ,_MI ,_RE,_DO,_DO,_1LA,
    _1SO,_1SO,_DO,_RE,_MI,
    _MI ,_DO ,_MI,_LA,_SO,
    _MI ,_MI ,_RE,_RE,_DO,
    _EOF
};

void Music29(void)
{
   Byte i=0;

   Tempo = MODERATO;

   while ( (SOUNDLONG29[i]!=_EOF)||(SOUNDTONE29[i]!=_EOF) )
   {
      //start timer1,generate soundtone
      Period=SOUNDTONE29[i];
```

```
        TL1 = (65536 - Period) & 0xff;
        TH1 = (65536 - Period) >> 8;
        TR1 = 1;

        SoundLongCount=((Tempo*SOUNDLONG29[i]*1000L)/8)/Period;
        while ( SoundLongCount != 0 );

        TR1 = 0;      //stop timer1:soundtone
        i++;
    }
}
```

Music30.c　曲名：蝴蝶

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG30[ ] =
{
    _4,_8,_8,_4,_4,_8,_8,_8,_8,_2,
    _4,_8,_8,_4,_4,_8,_8,_8,_8,_2,
    _4,_8,_8,_4,_4,_4,_8,_8,_2,
    _4,_8,_8,_4,_4,_4,_8,_8,_2,
    _EOF
};

Word RDATA SOUNDTONE30[ ] =
{
    _DO ,_DO,_RE ,_MI,_MI,_RE ,_DO,_RE,_MI,_DO,
    _MI ,_MI,_FA ,_SO,_SO,_FA ,_MI,_FA,_SO,_MI,
    _DO1,_TI,_LA ,_SO,_MI,_DO1,_TI,_LA,_SO,
    _LA ,_TI,_DO1,_SO,_MI,_SO ,_FA,_RE,_DO,
    _EOF
};

void Music30(void)
{
    Byte i=0;
```

```
    Tempo = MODERATO;

    while ( (SOUNDLONG30[i]!=_EOF)||(SOUNDTONE30[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE30[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG30[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

Music31.c 曲名：鱼儿水中游

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "music.h"

/*****************************/
Byte RDATA SOUNDLONG31[ ] =
{
    _4,_8,_4,_8,_8,_8,_8,_4d,
    _8,_8,_8,_8,_8,_8,_4,_8 ,_4d,
    _4,_8,_4,_8,_2,_4,_8,_4 ,_8 ,_2,
    _4,_8,_4,_4,_8,_4,
    _8,_8,_8,_8,_8,_8,_4,_8 ,_4 ,
    _EOF
};

Word RDATA SOUNDTONE31[ ] =
{
    _DO1,_DO1,_SO ,_SO ,_MI ,_FA ,_LA ,_SO ,
    _SO ,_LA ,_SO ,_FA ,_SO ,_FA ,_MI ,_DO ,_RE ,
```

```
    _DO ,_DO ,_DO ,_LA ,_SO ,_LA ,_SO ,_MI ,_DO ,_RE ,
    _DO ,_MI ,_SO ,_DO1,_TI ,_LA ,
    _SO ,_LA ,_SO ,_MI ,_RE ,_DO ,_RE ,_SO ,_DO ,
    _EOF
};

● void Music31(void)
{
    Byte i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG31[i]!=_EOF)||(SOUNDTONE31[i]!=_EOF) )
    {
      //start timer1,generate soundtone
      Period=SOUNDTONE31[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG31[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;      //stop timer1:soundtone
      i++;
    }
}
```

汇编程序如下：

1. input.a51

```
$NOMOD51

NAME INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT          SEGMENT CODE
 EXTRN    IDATA (KeyData)
 PUBLIC   InputKey

 RSEG  ?PR?InputKey?INPUT
```

```
USING    0
InputKey:
MOV      A,P1
CPL      A
MOV      R0,#KeyData
MOV      @R0,A
RET

END
```

2. music.a51

```
$NOMOD51

NAME MUSIC

$INCLUDE (mtv212m.inc)

?PR?_Music?MUSIC              SEGMENT CODE
EXTRN    CODE (Music1)
EXTRN    CODE (Music2)
EXTRN    CODE (Music3)
EXTRN    CODE (Music4)
EXTRN    CODE (Music5)
EXTRN    CODE (Music6)
EXTRN    CODE (Music7)
EXTRN    CODE (Music8)
EXTRN    CODE (Music9)
EXTRN    CODE (Music10)
EXTRN    CODE (Music11)
EXTRN    CODE (Music12)
EXTRN    CODE (Music13)
EXTRN    CODE (Music14)
EXTRN    CODE (Music15)
EXTRN    CODE (Music16)
EXTRN    CODE (Music17)
EXTRN    CODE (Music18)
EXTRN    CODE (Music19)
EXTRN    CODE (Music20)
EXTRN    CODE (Music21)
EXTRN    CODE (Music22)
EXTRN    CODE (Music23)
EXTRN    CODE (Music24)
EXTRN    CODE (Music25)
EXTRN    CODE (Music26)
```

```
EXTRN     CODE (Music27)
EXTRN     CODE (Music28)
EXTRN     CODE (Music29)
EXTRN     CODE (Music30)
EXTRN     CODE (Music31)
EXTRN     CODE (?C?CCASE)
PUBLIC    _Music

RSEG  ?PR?_Music?MUSIC
USING     0
_Music:
MOV       A,R7
LCALL     ?C?CCASE
DW   ?C0002
DB   01H
DW   ?C0003
DB   02H
DW   ?C0004
DB   03H
DW   ?C0005
DB   04H
DW   ?C0006
DB   05H
DW   ?C0007
DB   06H
DW   ?C0008
DB   07H
DW   ?C0009
DB   08H
DW   ?C0010
DB   09H
DW   ?C0011
DB   0AH
DW   ?C0012
DB   0BH
DW   ?C0013
DB   0CH
DW   ?C0014
DB   0DH
DW   ?C0015
DB   0EH
DW   ?C0016
DB   0FH
DW   ?C0017
DB   010H
```

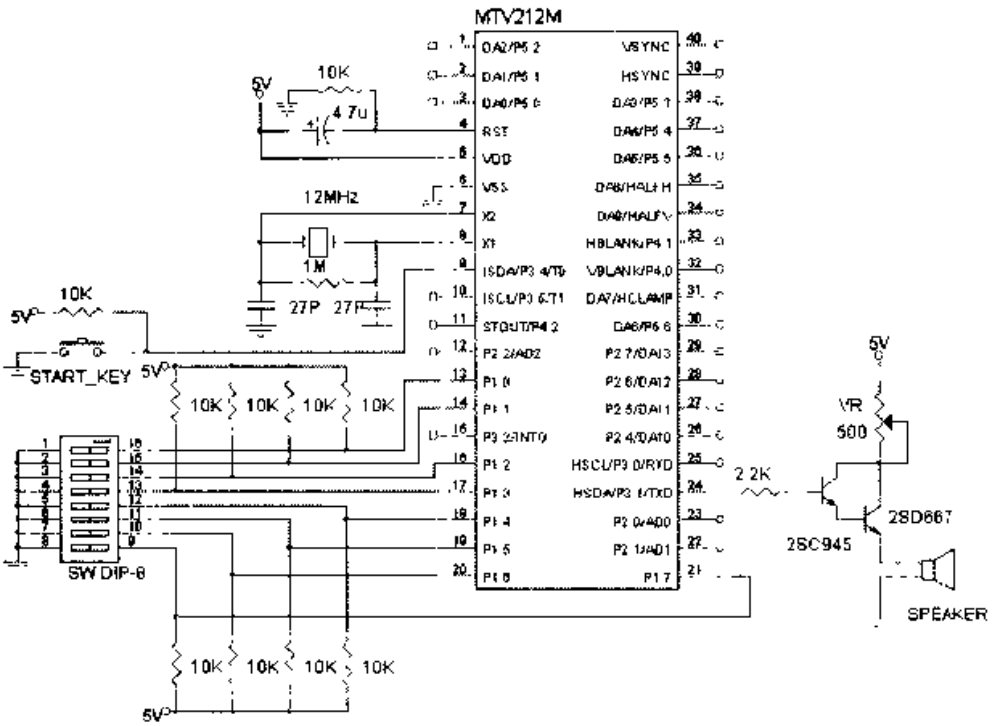
```
DW   ?C0018
DB   011H
DW   ?C0019
DB   012H
DW   ?C0020
DB   013H
DW   ?C0021
DB   014H
DW   ?C0022
DB   015H
DW   ?C0023
DB   016H
DW   ?C0024
DB   017H
DW   ?C0025
DB   018H
DW   ?C0026
DB   019H
DW   ?C0027
DB   01AH
DW   ?C0028
DB   01BH
DW   ?C0029
DB   01CH
DW   ?C0030
DB   01DH
DW   ?C0031
DB   01EH
DW   ?C0032
DB   01FH
DW   00H
DW   ?C0034

?C0002:
LCALL    Music1
RET
?C0003:
LCALL    Music2
RET
?C0004:
LCALL    Music3
RET
?C0005:
LCALL    Music4
RET
```

```
?C0006:
 LCALL    Music5
 RET
?C0007:
 LCALL    Music6
 RET
?C0008:
 LCALL    Music7
 RET
?C0009:
 LCALL    Music8
 RET
?C0010:
 LCALL    Music9
 RET
?C0011:
 LCALL    Music10
 RET
?C0012:
 LCALL    Music11
 RET
?C0013:
 LCALL    Music12
 RET
?C0014:
 LCALL    Music13
 RET
?C0015:
 LCALL    Music14
 RET
?C0016:
 LCALL    Music15
 RET
?C0017:
 LCALL    Music16
 RET
?C0018:
 LCALL    Music17
 RET
?C0019:
 LCALL    Music18
 RET
?C0020:
 LCALL    Music19
 RET
```

```
?C0021:
 LCALL    Music20
 RET
?C0022:
 LCALL    Music21
 RET
?C0023:
 LCALL    Music22
 RET
?C0024:
 LCALL    Music23
 RET
?C0025:
 LCALL    Music24
 RET
?C0026:
 LCALL    Music25
 RET
?C0027:
 LCALL    Music26
 RET
?C0028:
 LCALL    Music27
 RET
?C0029:
 LCALL    Music28
 RET
?C0030:
 LCALL    Music29
 RET
?C0031:
 LCALL    Music30
 RET
?C0032:
 LCALL    Music31
?C0034:
 RET

 END
```

9-3-2 按下起始键才执行音乐演奏函数

电路图：



软件构建的思维与解决方法

其实，只需在 Main( )函数增加起始键的判断，当 I/O=“0”表示按下起始键才能调用 Music( )函数，并在 input.h 包括文件下定义 START_IO 为 P3.4 即可，程序如下：

（1）main.c：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
```

```
    {
      //sel play no#
      InputKey( );

      if ( START_IO==0 )
      {
        MusicNumber=KeyData;
        Music(MusicNumber);
        DelayX10ms(100);
      }
    }
}
```

（2）input.h：

```
#ifndef   _INPUT_H
#define   _INPUT_H

#define  INPUT_PORT          P1
#define  START_IO       P3_4

extern void InputKey(void);

#endif
```

汇编程序如下：

main.a51

```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN                SEGMENT CODE
EXTRN     IDATA (KeyData)
EXTRN     IDATA (MusicNumber)
EXTRN     CODE (PowerOnInit)
EXTRN     CODE (_DelayX10ms)
EXTRN     CODE (_Music)
EXTRN     CODE (InputKey)
EXTRN     CODE (?C_STARTUP)
PUBLIC    Main

RSEG  ?PR?Main?MAIN
```

```
 USING    0
Main:
 LCALL    PowerOnInit
?C0001:
 LCALL    InputKey
 JB   P3_4,?C0001
 MOV      R0,#KeyData
 MOV      A,@R0
 MOV      R7,A
 MOV      R0,#MusicNumber
 MOV      @R0,A
 LCALL    _Music
 MOV      R7,#064H
 MOV      R6,#00H
 LCALL    _DelayX10ms
 SJMP     ?C0001
 RET

 END
```

9-3-3　未选曲时，按下起始键也不执行音乐演奏函数

软件构建的思维与解决方法

也就是当按下起始键，则 8P 的指拨开关，至少有一个要拨至“ON”位置，才能调用音乐演奏函数，即 Music( )函数，又由于音乐演奏函数只能演奏曲目 1～31，当开关输入第 64 首而按下起始键，则在 Main( )函数一样会调用 Music( )函数，只执行 Music( )函数，由于不符合 1～31 的条件值，将执行 default 语句下的 break 指令，而再次返回原调用程序，形成不演奏音乐，和未输入曲目（KeyData 将等于 0）而不演奏音乐的原因不同，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
```

```
/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      //sel play no#
      InputKey( );

      if ( START_IO==0 )
      {
         if ( KeyData != NO_KEY )
         {
            MusicNumber=KeyData;
            Music(MusicNumber);
            DelayX10ms(100);
         }
      }
   }
}
```

由于要按下起始键以及开关不能为0X00，其条件为逻辑AND，因此，可将上述由两个if条件判断式所构成的程序段，改成只需由一个if条件判断式来完成，即两个条件式需同时成立，才能调用Music( )函数，如下：

```
/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      //sel play no#
      InputKey( );

      if ( (START_IO==0)&&(KeyData != NO_KEY) )
      {
         MusicNumber=KeyData;
         Music(MusicNumber);
         DelayX10ms(100);
      }
   }
}
```

汇编程序如下：

1. main.a51

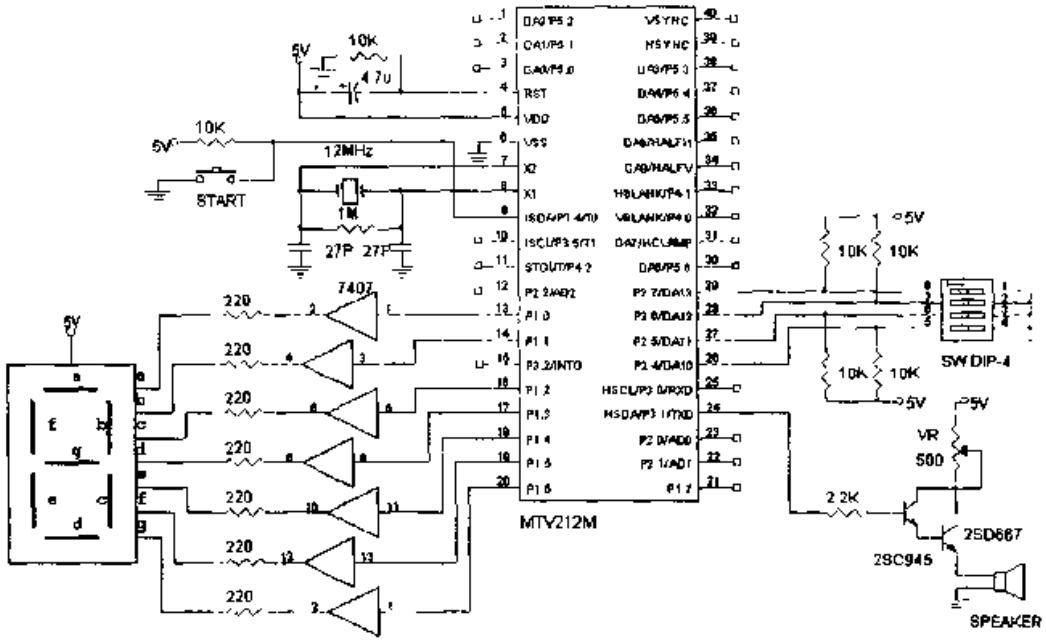
```
$NOMOD51

NAME MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN                SEGMENT CODE
 EXTRN    IDATA (KeyData)
 EXTRN    IDATA (MusicNumber)
 EXTRN    CODE (PowerOnInit)
 EXTRN    CODE (_DelayX10ms)
 EXTRN    CODE (_Music)
 EXTRN    CODE (InputKey)
 EXTRN    CODE (?C_STARTUP)
 PUBLIC   Main

 RSEG  ?PR?Main?MAIN
 USING    0
Main:
 LCALL    PowerOnInit
?C0001:
 LCALL    InputKey
 JB   P3_4,?C0001
 MOV      R0,#KeyData
 MOV      A,@R0
 MOV      R7,A
 JZ   ?C0001
 MOV      R0,#MusicNumber
 MOV      @R0,A
 LCALL    _Music
 MOV      R7,#064H
 MOV      R6,#00H
 LCALL    _DelayX10ms
 SJMP     ?C0001
 RET

 END
```

第 10 章　七段显示器

10-1　PortX1 显示器 X1

目的： 利用 I/O 端口直接推动显示器。

电路图：



原理： 七段显示器由七个 LED 组成，借着点亮每一段的 LED 就可以显示出数字 0～9。而每一段的 LED 则由 CPU 的 I/O 所控制，当 I/O 输出为“1”电位，则 LED 熄灭，当 I/O 输出“0”电位，则 LED 亮。为了使 LED 流入大电流而可以显示亮一点，因此以 IC7407 作为缓冲器，其中电阻 220Ω为限流电阻，可根据显示器的明暗程度作适当调整。由于 I/O P1.0 控制七段显示器的 a 段，I/O P1.1 控制 b 段，I/O P1.2 控制 c 段，I/O P1.3 控制 d 段，I/O P1.4 控制 e 段，I/O P1.5 控制 f 段，I/O P1.6 控制 g 段，也都是“0”动作，而 I/O P1.7 不控制为浮接状态，则设定一直为“1”电位输出，因而，当欲显示数字“0”，则 a、b、c、d、e、f 段都需点亮，即相对应的 I/O 为“0”电位，则 P1 端口的输出数值为 0XC0。同理，欲显示数字“1”，则 b、c 段点亮，P1 端口的输出应为 0Xf9，依此方法，将欲显示数字的数值先计算出来，在程序中取表，即可显示对应的数字了，其显示数字和 P1 端口的输出如下表所示。

数字	P1 端口输出
0	0XC0
1	0Xf9
2	0Xa4
3	0Xb0
4	0X99
5	0X92
6	0X82
7	0Xf8
8	0X80
9	0X09

10-1-1　DIP 选曲 1～9

模块化：将 DIP 指拨开关所输入的曲目编号，通过七段显示器显示出来，则其显示器的控制输出函数可独立为一模块，其名称为 7 Segled.c 模块，相对应的包括文件为 7 Segled.h，则新增的模块其功能如下：

（1）7Segled.c：七段显示器的输出控制模块，包含数字 0～9 的转换表。

（2）7Segled.h：7Segled.c 模块所对应的包括文件，除了定义驱动七段显示器为 P1 端口之外，也要将函数和转换表作外部声明。

软件构建的思维与解决方法

由 I/O P2.4～P2.7 作为 DIP 指拨开关的选曲输入，由于是四个开关，所以可选曲 1～15 首，在程序中当输入值大于 9，则认为输入 0，表示输入错误而不演奏。当 DIP 开关输入时则对应的 I/O 将为“0”电位，如果 DIP 开关不输入则其 I/O 将为“1”电位，因此，可以容易地侦测 DIP 开关的输入状态而能演奏曲目，现分述如下：

（1）initial.c：在开机时将 P2.4～P2.7 的 DIP 选曲输入设置为输入状态，即将 I/O 输出为“1”电位即可，并将七段显示器归零，而拨动指拨开关可以立即显示，其选曲编号和 DIP 输入的数值先初始化为 0，程序如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
```

```
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*****************************/
void PowerOnInit(void)
{
    InitialCpu();

    InitialCpuIO();

    InitialVariable();
}

/*****************************/
void InitialCpu( void )
{
    IE  = 0;          //disable all interrupt
    PSW = 0;          //bank 0

    IP  = 0x0b;       //hi priority:int0,timer0,timer1

    TMOD= 0x11;       //set timer1,timer0 mode

    TR0 = 0;          //stop timer0
    TR1 = 0;          //stop timer1
    IT0 = 1;          //set int0:falling eage trigger

    EX0 = 0;          //disable int0 interrupt
    ET1 = 1;          //enable timer1 interrupt
    ET0 = 0;          //disable timer0 interrupt

    EA  = 1;          //enable all interrupt gate
}

/*****************************/
void InitialCpuIO(void)
{
    //display "0"
    _7SEGDISP_PORT1=DISP7SEG_TABLE[0];
    P2=0xf0;      //input:P2.4~P2.7
    SPEAKER = 0;
```

```
}

/*******************************/
void InitialVariable(void)
{
   KeyData    = 0;

   MusicNumber= 0;
}
```

（2）input.c：DIP 选曲是由 I/O P2.4～P2.7 作为输入，而且是“0”动作，因而应将 P2 端口反相后，并和数值 0Xf0 作逻辑 AND 后，才能存入至 KeyData 变量内，并将此数值右移四个位，其数值将介于 0～15 之间，则正好可作为选曲的编号。实际上，是因为想要将此 DIP 开关作为选曲输入 0～15，而利用此软件方法（右移四位）才能达到所求的缘故，由于选曲编号只有 1～9 首，因而，当输入大于 9 时，则以 0 取代而不演奏，并将此输入由七段显示器显示其数字，其程序如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*******************************/
void InputKey(void)
{
   KeyData = ~(INPUT_PORT) & 0xf0; //P2.4~P2.7

   KeyData>>=4;
   if (KeyData > 9)
     KeyData=0;

   Disp7Seg(KeyData);
}
```

（3）input.h：input.c 模块下的包括文件，要定义 DIP 开关的输入端口，起始演奏的 I/O 输入，以及 DIP 未输入时的数值为 0，包括文件如下：

```
#ifndef     _INPUT_H
#define     _INPUT_H

#define  INPUT_PORT      P2  //P2.4~P2.7
#define  START_IO        P3_4
#define  NO_KEY          0

extern void InputKey(void);

#endif
```

（4）7Segled.c：将 DIP 开关的输入通过 P1 端口直接输出至显示器，侦测 DIP 的输入数值，则以此数值取出对应输出至七段显示器的 P1 端口输出数值，则用此取表方式可以很方便地将数字显示出来，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*****************************/
void Disp7Seg(Byte value)
{
   _7SEGDISP_PORT1=DISP7SEG_TABLE[value];
}

/*****************************/
//display 0~9
Byte RDATA DISP7SEG_TABLE[ ] =
{
   0xc0,0xf9,0xa4,0xb0,0x99,
   0x92,0x82,0xf8,0x80,0x90
};
```

（5）7Segled.h：7Segled.c 模块下的包括文件，要定义输出至七段显示器的 I/O 端口为 P1 端口，函数外部声明以及数字转换表，如下：

```
#ifndef     _7SEGLED_H
#define     _7SEGLED_H

#define  _7SEGDISP_PORT1     P1

extern Byte RDATA DISP7SEG_TABLE[ ];
extern void Disp7Seg(Byte value);

#endif
```

汇编程序如下：

1. initial.a51

```
$NOMOD51

NAME    INITIAL

$INCLUDE (mtv212m.inc)

?PR?PowerOnInit?INITIAL          SEGMENT CODE
?PR?InitialCpu?INITIAL           SEGMENT CODE
?PR?InitialCpuIO?INITIAL         SEGMENT CODE
?PR?InitialVariable?INITIAL          SEGMENT CODE
    EXTRN   IDATA (KeyData)
    EXTRN   IDATA (MusicNumber)
    EXTRN   CODE (DISP7SEG_TABLE)
    PUBLIC  InitialVariable
    PUBLIC  InitialCpuIO
    PUBLIC  InitialCpu
    PUBLIC  PowerOnInit

    RSEG  ?PR?PowerOnInit?INITIAL
    USING   0
PowerOnInit:
    LCALL   InitialCpu
    LCALL   InitialCpuIO
    LCALL   InitialVariable
    RET

    RSEG  ?PR?InitialCpu?INITIAL
    USING   0
InitialCpu:
    CLR     A
    MOV     IE,A
```

```
    MOV     PSW,A
    MOV     IP,#0BH
    MOV     TMOD,#011H
    CLR     TR0
    CLR     TR1
    SETB    IT0
    CLR     EX0
    SETB    ET1
    CLR     ET0
    SETB    EA
    RET

    RSEG  ?PR?InitialCpuIO?INITIAL
    USING   0
InitialCpuIO:
    MOV     DPTR,#DISP7SEG_TABLE
    CLR     A
    MOVC    A,@A+DPTR
    MOV     P1,A
    MOV     P2,#0F0H
    CLR     P3_1
    RET

    RSEG  ?PR?InitialVariable?INITIAL
    USING   0
InitialVariable:
    CLR     A
    MOV     R0,#KeyData
    MOV     @R0,A
    MOV     R0,#MusicNumber
    MOV     @R0,A
    RET

    END
```

2. input.a51

```
$NOMOD51

NAME    INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT   SEGMENT CODE
    EXTRN   IDATA (KeyData)
```

```
    EXTRN   CODE (_Disp7Seg)
    PUBLIC  InputKey

    RSEG  ?PR?InputKey?INPUT
    USING   0
InputKey:
    MOV     A,P2
    CPL     A
    ANL     A,#0F0H
    MOV     R0,#KeyData
    MOV     @R0,A
    MOV     R7,A
    SWAP    A
    ANL     A,#0FH
    MOV     @R0,A
    SETB    C
    SUBB    A,#09H
    JC      ?C0001
    CLR     A
    MOV     @R0,A
?C0001:
    MOV     R0,#KeyData
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Disp7Seg
    RET

    END
```

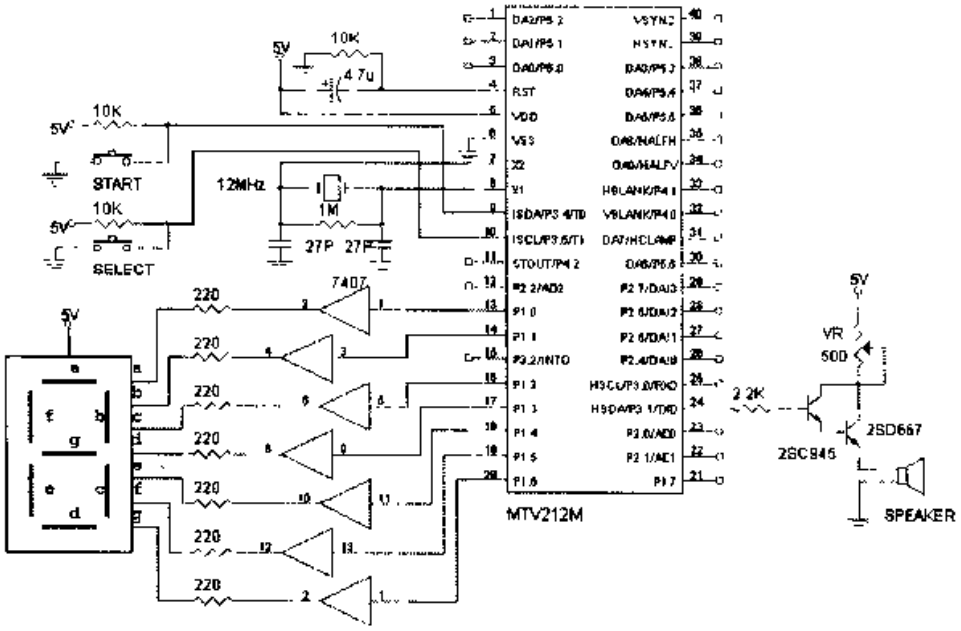
3. 3.7segled.a51

```
$NOMOD51

NAME    _7SEGLED

$INCLUDE (mtv212m.inc)

?PR?_Disp7Seg?7SEGLED          SEGMENT CODE
?CO?7SEGLED          SEGMENT CODE
    PUBLIC  DISP7SEG_TABLE
    PUBLIC  _Disp7Seg

    RSEG  ?CO?7SEGLED
DISP7SEG_TABLE:
    DB  0C0H,0F9H,0A4H,0B0H,099H
```

```
    DB   092H,082H,0F8H,080H,090H

    RSEG  ?PR?_Disp7Seg?7SEGLED
    USING   0
_Disp7Seg:
    MOV     A,R7
    MOV     DPTR,#DISP7SEG_TABLE
    MOVC    A,@A+DPTR
    MOV     P1,A
    RET

    END
```

10-1-2 TACT 选曲：1～9

电路图：



软件构建的思维与解决方法

（1）input.c：以 TACT 开关来选曲，当 TACT 开关按键一下，喇叭响叫一声作为提示，表示曲目编号累加 1，再按键一下又继续将曲目编号累加 1，当超过曲目编号 9，则应从曲目编号 1 开始循环，这样，以一个 TACT 开关作为选曲输入，应注意下列几点：

- 曲目编号的选择要一直循环，当到达上限值时，再按键一次要从下限值开始设定。

- 避免当按键一直按着而会一直累加选曲的情形，而要很清楚地按一次键，喇叭只响叫一声，曲目编号只增加一。
- 每按键一次，则七段显示器也要立即显示出目前曲目编号，以方便使用者明白欲设定的曲目来演奏。
- 当曲目编号设定完毕，应再按下“起始键”才能进行曲目演奏。

其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*****************************/
void InputHandler(void)
{
   //sel play no#
   InputKey( );

   if ( KeyBuffer != KeyData )
   {
      KeyBuffer = KeyData;
      ScanKeyCounter = 10;
   }
   else if( ScanKeyCounter != 0 )
   {
      ScanKeyCounter--;
      DelayX1ms(1);
   }
   else if ( KeyBuffer == SELECT_KEY )
   {
      if (FgSelectKey==0)
      {
         FgSelectKey = 1;
         Beep(1,17,10);
```

```
            MusicNumber++;
            if ( MusicNumber>9 )
               MusicNumber=1;
            Disp7Seg(MusicNumber);
         }
      }
      else
      {
         KeyBuffer   = NO_KEY;
         FgSelectKey = 0;
      }
}

/*****************************/
void InputKey(void)
{
   if ( SELECT_IO==0 )
      KeyData=SELECT_KEY;
   else
      KeyData=NO_KEY;
}

void InputStart(void)
{
   if ( START_IO==0 )
      KeyData=START_KEY;
   else
      KeyData=NO_KEY;
}
```

（2）input.h：input.c 模块的包括文件，应定义选曲键和起始演奏键的 I/O 及数值，其内容如下：

```
#ifndef     _INPUT_H
#define     _INPUT_H

#define  START_IO       P3_4
#define  SELECT_IO      P3_5

#define  NO_KEY         0
#define  START_KEY      1
#define  SELECT_KEY 2

extern void InputHandler(void);
extern void InputKey(void);
```

```
extern void InputStart(void);

#endif
```

汇编程序如下：

1. input.a51

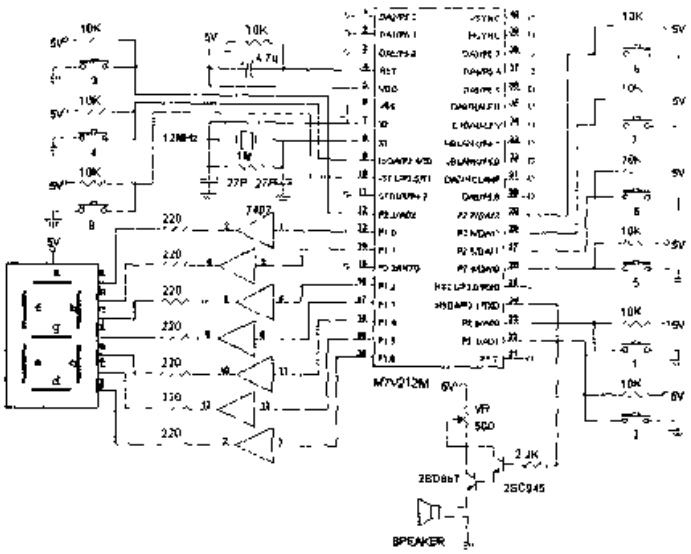
```
$NOMOD51

NAME    INPUT

$INCLUDE (mtv212m.inc)

?PR?InputHandler?INPUT          SEGMENT CODE
?PR?InputKey?INPUT              SEGMENT CODE
?PR?InputStart?INPUT SEGMENT CODE
    EXTRN   BIT (FgSelectKey)
    EXTRN   IDATA (KeyData)
    EXTRN   IDATA (KeyBuffer)
    EXTRN   IDATA (ScanKeyCounter)
    EXTRN   IDATA (MusicNumber)
    EXTRN   CODE (_DelayX1ms)
    EXTRN   CODE (_Beep)
    EXTRN   CODE (_Disp7Seg)
    PUBLIC  InputStart
    PUBLIC  InputKey
    PUBLIC  InputHandler

    RSEG  ?PR?InputHandler?INPUT
    USING   0
InputHandler:
    LCALL   InputKey
    MOV     R0,#KeyData
    MOV     A,@R0
    MOV     R7,A
    MOV     R0,#KeyBuffer
    XRL     A,@R0
    JZ      ?C0001
    MOV     A,R7
    MOV     @R0,A
    MOV     R0,#ScanKeyCounter
    MOV     @R0,#0AH
    RET
?C0001:
```

```
    MOV     R0,#ScanKeyCounter
    MOV     A,@R0
    JZ      ?C0003
    DEC     @R0
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    RET
?C0003:
    MOV     R0,#KeyBuffer
    MOV     A,@R0
    XRL     A,#02H
    JNZ     ?C0005
    JB      FgSelectKey,?C0009
    SETB    FgSelectKey
    MOV     R7,#01H
    MOV     R6,A
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep
    MOV     R0,#MusicNumber
    INC     @R0
    MOV     A,@R0
    SETB    C
    SUBB    A,#09H
    JC      ?C0007
    MOV     @R0,#01H
?C0007:
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Disp7Seg
    RET
?C0005:
    CLR     A
    MOV     R0,#KeyBuffer
    MOV     @R0,A
    CLR     FgSelectKey
?C0009:
    RET

    RSEG  ?PR?InputKey?INPUT
    USING   0
InputKey:
    JB      P3_5,?C0010
```

```
    MOV     R0,#KeyData
    MOV     @R0,#02H
    RET
?C0010:
    CLR     A
    MOV     R0,#KeyData
    MOV     @R0,A
?C0012:
    RET

    RSEG  ?PR?InputStart?INPUT
    USING   0
InputStart:
    JB      P3_4,?C0013
    MOV     R0,#KeyData
    MOV     @R0,#01H
    RET
?C0013:
    CLR     A
    MOV     R0,#KeyData
    MOV     @R0,A
?C0015:
    RET

    END
```

10-1-3　I/O 选曲：1～9

电路图：



软件构建的思维与解决方法

（1）input.c：将每一支 I/O 对应一个曲目编号，是最简单最容易的，其缺点是必须浪费很多的 I/O 引脚。有 1～9 首曲目则必需要使用九支 I/O，如 I/O P2.0 对应曲目 1，P2.1 对应曲目 2，P2.2 对应曲目 3 等，I/O 对应的曲目可由 input.h 包括文件来定义，当硬件有改变时，使得程序只要修改包括文件而不需要改变程序的设计。如果要演奏曲目编号 5，则应按下编号为 5 的按键，要演奏曲目编号为 7 的曲目，则应按下编号为 7 的按键，在程序中应依序判断其对应的 I/O 是否为“0”电位即能达到此目的，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*****************************/
void InputKey(void)
{
   if ( NUMBER1_IO==0 )
     KeyData=NUMBER1_KEY;
   else if ( NUMBER2_IO==0 )
     KeyData=NUMBER2_KEY;
   else if ( NUMBER3_IO==0 )
     KeyData=NUMBER3_KEY;
   else if ( NUMBER4_IO==0 )
     KeyData=NUMBER4_KEY;
   else if ( NUMBER5_IO==0 )
     KeyData=NUMBER5_KEY;
   else if ( NUMBER6_IO==0 )
     KeyData=NUMBER6_KEY;
   else if ( NUMBER7_IO==0 )
     KeyData=NUMBER7_KEY;
   else if ( NUMBER8_IO==0 )
     KeyData=NUMBER8_KEY;
   else if ( NUMBER9_IO==0 )
```

```
        KeyData=NUMBER9_KEY;
    else
        KeyData=NO_KEY;
}
```

（2）input.h：要定义每一支 I/O 及其曲目编号的内容，如下：

```
#ifndef     _INPUT_H
#define     _INPUT_H

#define  NUMBER1_IO P2_0
#define  NUMBER2_IO P2_1
#define  NUMBER3_IO P2_2
#define  NUMBER4_IO P3_4
#define  NUMBER5_IO P2_4
#define  NUMBER6_IO P2_5
#define  NUMBER7_IO P2_6
#define  NUMBER8_IO P2_7
#define  NUMBER9_IO P3_5

#define  NO_KEY          0
#define  NUMBER1_KEY1
#define  NUMBER2_KEY2
#define  NUMBER3_KEY3
#define  NUMBER4_KEY4
#define  NUMBER5_KEY5
#define  NUMBER6_KEY6
#define  NUMBER7_KEY7
#define  NUMBER8_KEY8
#define  NUMBER9_KEY9

extern void InputKey(void);

#endif
```

汇编程序如下：

1. input.a51

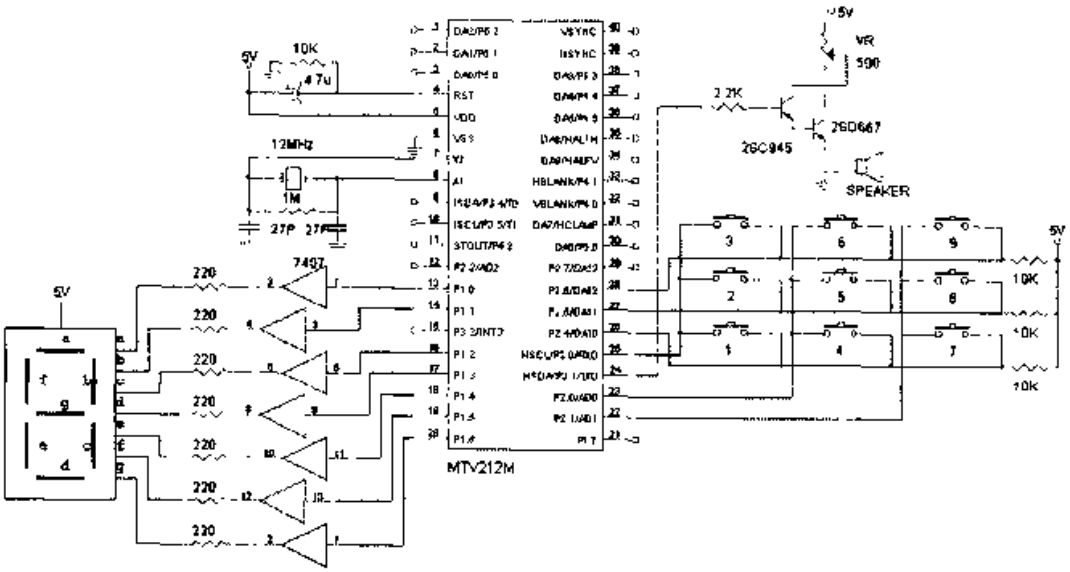
```
$NOMOD51

NAME    INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT   SEGMENT CODE
```

```
    EXTRN   IDATA (KeyData)
    PUBLIC  InputKey

    RSEG  ?PR?InputKey?INPUT
    USING   0
InputKey:
    JB      P2_0,?C0001
    MOV     R0,#KeyData
    MOV     @R0,#01H
    RET
?C0001:
    JB      P2_1,?C0003
    MOV     R0,#KeyData
    MOV     @R0,#02H
    RET
?C0003:
    JB      P2_2,?C0005
    MOV     R0,#KeyData
    MOV     @R0,#03H
    RET
?C0005:
    JB      P3_4,?C0007
    MOV     R0,#KeyData
    MOV     @R0,#04H
    RET
?C0007:
    JB      P2_4,?C0009
    MOV     R0,#KeyData
    MOV     @R0,#05H
    RET
?C0009:
    JB      P2_5,?C0011
    MOV     R0,#KeyData
    MOV     @R0,#06H
    RET
?C0011:
    JB      P2_6,?C0013
    MOV     R0,#KeyData
    MOV     @R0,#07H
    RET
?C0013:
    JB      P2_7,?C0015
    MOV     R0,#KeyData
    MOV     @R0,#08H
    RET
```

```
?C0015:
    JB      P3_5,?C0017
    MOV     R0,#KeyData
    MOV     @R0,#09H
    RET
?C0017:
    CLR     A
    MOV     R0,#KeyData
    MOV     @R0,A
?C0019:
    RET

    END
```

10-1-4　SCAN 选曲：1～9

电路图：



软件构建的思维与解决方法

（1）input.c：以矩阵扫描方式可以用较少的 I/O 脚来侦测更多的按键输入，因而可节省 CPU 的 I/O 引脚。其每个按键一样的对应曲目编号，有九个曲目需九个按键，则应以 3×3=9 的矩阵排列，分别有三支 I/O 作为输入，三支 I/O 作为行扫描线的输出，总共需要 3+3=6 支 I/O 就可以了。其原理也只是依序令行扫描线动作（“0”电位动作）再个别侦测此行的按键是否按下，当按下时其对应 I/O 的输入电位将变为“0”电位，因而，可轻易地进行判断。程序中以 switch...case 指令结构可以更清

楚地表示程序目的，如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*******************************/
void InputKey(void)
{
   Byte  keytmp;

   KeyData = NO_KEY;

   //scan line1:"0" active
   COLUMN1_PIN = 0;
   COLUMN2_PIN = 1;
   COLUMN3_PIN = 1;
   keytmp = ~(INPUT_PORT) & 0x70;   //P2.4~P2.6
   keytmp >>=4;
   switch( keytmp )
   {
    case 1 :
      KeyData = NUMBER1_KEY;
      break;
    case 2 :
      KeyData = NUMBER2_KEY;
      break;
    case 4 :
      KeyData = NUMBER3_KEY;
      break;
    default :
      break;
   }

   //scan line2:"0" active
   COLUMN1_PIN = 1;
```

```
    COLUMN2_PIN = 0;
    COLUMN3_PIN = 1;
    keytmp = ~(INPUT_PORT) & 0x70;   //P2.4~P2.6
    keytmp >>=4;
    switch( keytmp )
    {
     case 1 :
        KeyData = NUMBER4_KEY;
        break;
     case 2 :
        KeyData = NUMBER5_KEY;
        break;
     case 4 :
        KeyData = NUMBER6_KEY;
        break;
     default :
        break;
    }

    //scan line3:"0" active
    COLUMN1_PIN = 1;
    COLUMN2_PIN = 1;
    COLUMN3_PIN = 0;
    keytmp = ~(INPUT_PORT) & 0x70;   //P2.4~P2.6
    keytmp >>=4;
    switch( keytmp )
    {
     case 1 :
        KeyData = NUMBER7_KEY;
        break;
     case 2 :
        KeyData = NUMBER8_KEY;
        break;
     case 4 :
        KeyData = NUMBER9_KEY;
        break;
     default :
        break;
    }

    //scan line3:not active
    COLUMN3_PIN = 1;
}
```

（2）input.h：包括文件要定义每个按键的曲目编号，以及行扫描线的 I/O、列输入端口(P2.4～P2.6)，如下：

```
#ifndef     _INPUT_H
#define     _INPUT_H

#define  INPUT_PORT P2  //P2.4~P2.6
#define  COLUMN1_PINP3_0
#define  COLUMN2_PINP2_0
#define  COLUMN3_PINP2_1

#define  NO_KEY         0
#define  NUMBER1_KEY1
#define  NUMBER2_KEY2
#define  NUMBER3_KEY3
#define  NUMBER4_KEY4
#define  NUMBER5_KEY5
#define  NUMBER6_KEY6
#define  NUMBER7_KEY7
#define  NUMBER8_KEY8
#define  NUMBER9_KEY9

extern void InputKey(void);

#endif
```

汇编程序如下：

1. input.a51

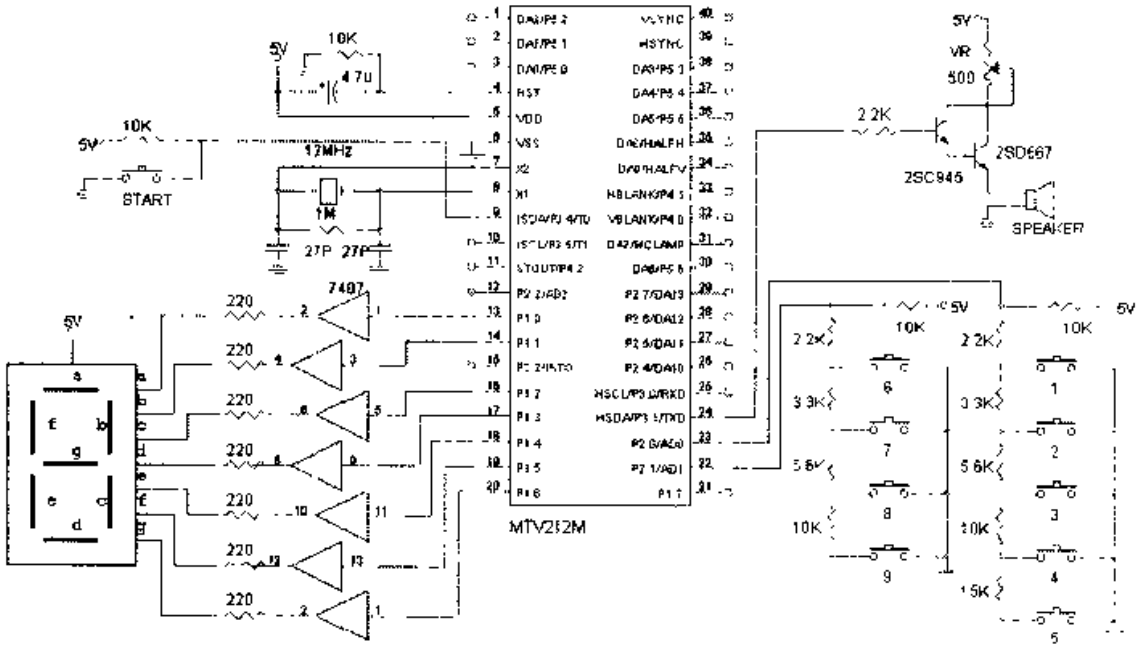
```
$NOMOD51

NAME    INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT   SEGMENT CODE
    EXTRN   IDATA (KeyData)
    PUBLIC  InputKey

    RSEG  ?PR?InputKey?INPUT
    USING   0
InputKey:
    CLR     A
    MOV     R0,#KeyData
```

```
    MOV     @R0,A
    CLR     P3_0
    SETB    P2_0
    SETB    P2_1
    MOV     A,P2
    CPL     A
    ANL     A,#070H
    MOV     R7,A
    SWAP    A
    ANL     A,#0FH
    MOV     R7,A
    ADD     A,#0FEH
    JZ      ?C0003
    ADD     A,#0FEH
    JZ      ?C0004
    ADD     A,#03H
    JNZ     ?C0001
?C0002:
    MOV     R0,#KeyData
    MOV     @R0,#01H
    SJMP    ?C0001
?C0003:
    MOV     R0,#KeyData
    MOV     @R0,#02H
    SJMP    ?C0001
?C0004:
    MOV     R0,#KeyData
    MOV     @R0,#03H
?C0001:
    SETB    P3_0
    CLR     P2_0
    SETB    P2_1
    MOV     A,P2
    CPL     A
    ANL     A,#070H
    MOV     R7,A
    SWAP    A
    ANL     A,#0FH
    MOV     R7,A
    ADD     A,#0FEH
    JZ      ?C0008
    ADD     A,#0FEH
    JZ      ?C0009
    ADD     A,#03H
    JNZ     ?C0006
```

```
?C0007:
    MOV     R0,#KeyData
    MOV     @R0,#04H
    SJMP    ?C0006
?C0008:
    MOV     R0,#KeyData
    MOV     @R0,#05H
    SJMP    ?C0006
?C0009:
    MOV     R0,#KeyData
    MOV     @R0,#06H
?C0006:
    SETB    P3_0
    SETB    P2_0
    CLR     P2_1
    MOV     A,P2
    CPL     A
    ANL     A,#070H
    MOV     R7,A
    SWAP    A
    ANL     A,#0FH
    MOV     R7,A
    ADD     A,#0FEH
    JZ      ?C0013
    ADD     A,#0FEH
    JZ      ?C0014
    ADD     A,#03H
    JNZ     ?C0011
?C0012:
    MOV     R0,#KeyData
    MOV     @R0,#07H
    SJMP    ?C0011
?C0013:
    MOV     R0,#KeyData
    MOV     @R0,#08H
    SJMP    ?C0011
?C0014:
    MOV     R0,#KeyData
    MOV     @R0,#09H
?C0011:
    SETB    P2_1
    RET

    END
```

10-1-5　ADC 选曲：1～9

电路图：



软件构建的思维与解决方法

利用 ADC 方式作为按键的输入侦测，每一个按键对应一个曲目编号，因此，总共有九个按键。ADC 是利用当按下键时经过电阻分压后其直流电压的不同，转换后的数字数值也是不同，而能判断出到底是哪一个曲目编号被选取的方式。有些 ADC 的电路已经内含至 CPU 内，而以外部数据存储器的方式来寻址，因而，在 initial.c 模式应先初始化，在全局变量模块 global.c 中也应以 pdata 的数据模式来定义 ADC 缓存器的地址，在 global.h 包括文件则应外部声明，如下：

（1）global.c：如

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"

/*******************************/
// Global Data
/*******************************/

//key
Byte  IDATA     KeyData;
```

```
//music
Byte  IDATA     MusicNumber;
Word  IDATA     Tempo;
Word  IDATA     Period;
Word  IDATA     SoundLongCount;

pdata unsigned char XFR_ADC          _at_    0x10;
pdata unsigned char XFR_WDT          _at_    0x18;
pdata unsigned char XFR_PADMOD1      _at_    0x30;
pdata unsigned char XFR_PADMOD2      _at_    0x31;
pdata unsigned char XFR_PADMOD3      _at_    0x32;
pdata unsigned char XFR_OPTION1      _at_    0x33;
pdata unsigned char XFR_OPTION2      _at_    0x34;
pdata unsigned char XFR_XBANK        _at_    0x35;
```

（2）global.h：如

```
#ifndef _GLOBAL_H
#define _GLOBAL_H

//key
extern Byte  IDATA   KeyData;

//music
extern Byte  IDATA   MusicNumber;
extern Word  IDATA   Tempo;
extern Word  IDATA   Period;
extern Word  IDATA   SoundLongCount;

extern pdata unsigned char XFR_ADC;
extern pdata unsigned char XFR_WDT;
extern pdata unsigned char XFR_PADMOD1;
extern pdata unsigned char XFR_PADMOD2;
extern pdata unsigned char XFR_PADMOD3;
extern pdata unsigned char XFR_OPTION1;
extern pdata unsigned char XFR_OPTION2;
extern pdata unsigned char XFR_XBANK;

#endif
```

（3）initial.c：如

```
/*****************************/
void InitialCpu( void )
{
```

```
    IE  = 0;                //disable all interrupt
    PSW = 0;                //bank 0

    IP  = 0x0b;             //hi priority:int0,timer0,timer1

    TMOD= 0x11;             //set timer1,timer0 mode

    TR0 = 0;                //stop timer0
    TR1 = 0;                //stop timer1
    IT0 = 1;                //set int0:falling eage trigger

    EX0 = 0;                //disable int0 interrupt
    ET1 = 1;                //enable timer1 interrupt
    ET0 = 0;                //disable timer0 interrupt

    //Initial XFR's and DAC's for MTV212MNXX
    XFR_ADC  =0x80;  //enable ADC,not select ch input
    XFR_WDT  =0x40;  //clear watch dog timer,2s interval

    XFR_PADMOD1  =0x07;  //set adc0,adc1,adc2,i/o:P2.4~P2.7
    XFR_PADMOD2  =0x00;  //set dac0~dac6
    XFR_PADMOD3  =0x00;  //set dac,h,v
    XFR_OPTION1  =0xc0;  //94k pwm=1,div253=1
    XFR_OPTION2  =0x00;  //7 bit slave address for iic
    XFR_XBANK    =0x00;  //ram bank0

    EA  = 1;                //enable all interrupt gate
}
```

利用 ADC 方式来侦测按键的软件方法如下：

① 先选取 ADC 缓存器的信道并使能。

② 等待少许时间，以确定转换完成。

③ 将转换后的数值屏蔽没用的位并暂存至变量内。

④ 根据此变量来判断输入的 I/O 而取得曲目编号。

⑤ 继续致能另一个 ADC 信道再判断其他的按键输入。

其 input.c 模块的程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
```

```
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*****************************/
void InputKey(void)
{
   Byte  keytmp;

   KeyData  = NO_KEY;        //NO_KEY=0

   XFR_ADC=0x81;         //adc channel=1
   DelayX1ms(2);
   keytmp = XFR_ADC & 0x3f;     //6 bit  2^6=0x3f
   if ( keytmp < (12+4) )       KeyData = NUMBER1_KEY;
   else if ( keytmp < (23+4) )  KeyData = NUMBER2_KEY;
   else if ( keytmp < (34+4) )  KeyData = NUMBER3_KEY;
   else if ( keytmp < (44+4) )  KeyData = NUMBER4_KEY;
   else if ( keytmp < (50+4) )  KeyData = NUMBER5_KEY;

   XFR_ADC=0x82;         //adc channel=2
   DelayX1ms(2);
   keytmp = XFR_ADC & 0x3f;
   if ( keytmp < (12+4) )       KeyData = NUMBER6_KEY;
   else if ( keytmp < (23+4) )  KeyData = NUMBER7_KEY;
   else if ( keytmp < (34+4) )  KeyData = NUMBER8_KEY;
   else if ( keytmp < (44+4) )  KeyData = NUMBER9_KEY;

   XFR_ADC=0x00;         //disable adc
}
```

汇编程序如下：

1. input.a51

```
$NOMOD51

NAME    INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT      SEGMENT CODE
?DT?InputKey?INPUT      SEGMENT DATA OVERLAYABLE
```

```
    EXTRN   IDATA (KeyData)
    EXTRN   XDATA (XFR_ADC)
    EXTRN   CODE (_DelayX1ms)
    PUBLIC  InputKey

    RSEG  ?DT?InputKey?INPUT
?InputKey?BYTE:
    keytmp?040:   DS   1

    RSEG  ?PR?InputKey?INPUT
    USING   0
InputKey:
    CLR     A
    MOV     R0,#KeyData
    MOV     @R0,A
    MOV     R0,#LOW (XFR_ADC)
    MOV     A,#081H
    MOVX    @R0,A
    MOV     R7,#02H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#LOW (XFR_ADC)
    MOVX    A,@R0
    ANL     A,#03FH
    MOV     keytmp?040,A
    CLR     C
    SUBB    A,#010H
    JNC     ?C0001
    MOV     R0,#KeyData
    MOV     @R0,#01H
    SJMP    ?C0002
?C0001:
    MOV     A,keytmp?040
    CLR     C
    SUBB    A,#01BH
    JNC     ?C0003
    MOV     R0,#KeyData
    MOV     @R0,#02H
    SJMP    ?C0002
?C0003:
    MOV     A,keytmp?040
    CLR     C
    SUBB    A,#026H
    JNC     ?C0005
    MOV     R0,#KeyData
```

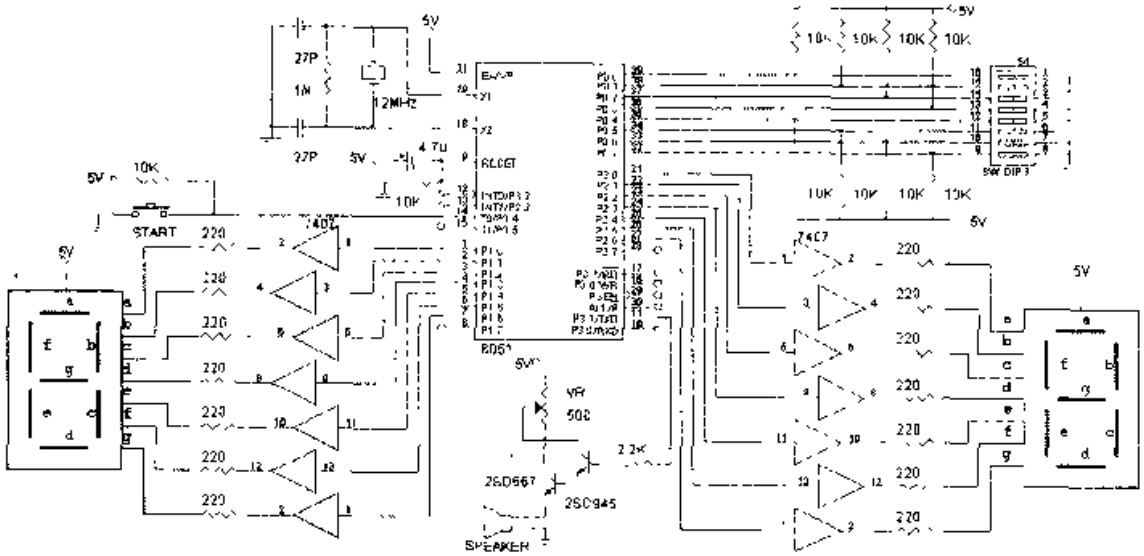
```
        MOV     @R0,#03H
        SJMP    ?C0002
?C0005:
        MOV     A,keytmp?040
        CLR     C
        SUBB    A,#030H
        JNC     ?C0007
        MOV     R0,#KeyData
        MOV     @R0,#04H
        SJMP    ?C0002
?C0007:
        MOV     A,keytmp?040
        CLR     C
        SUBB    A,#036H
        JNC     ?C0002
        MOV     R0,#KeyData
        MOV     @R0,#05H
?C0002:
        MOV     R0,#LOW (XFR_ADC)
        MOV     A,#082H
        MOVX    @R0,A
        MOV     R7,#02H
        MOV     R6,#00H
        LCALL   _DelayX1ms
        MOV     R0,#LOW (XFR_ADC)
        MOVX    A,@R0
        ANL     A,#03FH
        MOV     keytmp?040,A
        CLR     C
        SUBB    A,#010H
        JNC     ?C0010
        MOV     R0,#KeyData
        MOV     @R0,#06H
        SJMP    ?C0011
?C0010:
        MOV     A,keytmp?040
        CLR     C
        SUBB    A,#01BH
        JNC     ?C0012
        MOV     R0,#KeyData
        MOV     @R0,#07H
        SJMP    ?C0011
?C0012:
        MOV     A,keytmp?040
        CLR     C
```

```
    SUBB    A,#026H
    JNC     ?C0014
    MOV     R0,#KeyData
    MOV     @R0,#08H
    SJMP    ?C0011
?C0014:
    MOV     A,keytmp?040
    CLR     C
    SUBB    A,#030H
    JNC     ?C0011
    MOV     R0,#KeyData
    MOV     @R0,#09H
?C0011:
    CLR     A
    MOV     R0,#LOW (XFR_ADC)
    MOVX    @R0,A
    RET

    END
```

10-2　PortX2 显示器 X2

目的：利用两个 I/O 端口来分别驱动两个显示器。

电路图：



原理：利用标准 8051 的 P1 端口和 P2 端口分别驱动两个七段显示器，其驱动方式也是将数字转换成数值而从 I/O 端口直接输出即可。P1 端口控制个位数的显示器，而 P2 端口则控制十位数的显示器，在程序中应事先分

离出个位数和十位数，再分别由P1端口和P2端口输出。两个显示器其输入范围为1～99，而以八个DIP开关作为P0端口的选曲输入，在此只示范曲目1～31首，因而，当DIP开关输入大于31时，则以0取代表示输入错误而不演奏。

10-2-1 DIP选曲：1～99

模块化：以8P的DIP指拨开关作为选曲的输入，其可选择的范围为1～255首曲目，但为简单起见只提供1～31首曲目，实际上也可以增加音乐演奏模块至255首曲目。因此，在开发环境上的项目文件必须新增这些模块，而增加的音乐演奏的曲目，其模块分别为：

music10.c、music11.c、music12.c、music13.c、music14.c、music15.c、music16.c、music17.c、music18.c、music19.c、music20.c、music21.c、music22.c、music23.c、music24.c、music25.c、music26.c、music27.c、music28.c、music29.c、music30.c、music31.c等

软件构建的思维与解决方法

8051 P0端口作为八个DIP开关的选曲输入，八个DIP开关输入其最大数值为255，而两个七段显示器以十进制表示其最大数值为99，也就是可以选1～99首。然而，为说明方便，仅以31首曲目作为示范，因而，在Inputkey()函数中，当侦测到DIP开关大于31时，则以0取代，表示输入错误，如果有99首曲目，则程序中31的数值将改变为数值99，现分述如下：

（1）initial.c：有两个七段显示器，要分别初始化为0，即开机后显示器将显示为“00”，程序如下：

```
/*******************************/
void InitialCpuIO(void)
{
   //display "00"
   _7SEGDISP_PORT2=DISP7SEG_TABLE[0];
   _7SEGDISP_PORT1=DISP7SEG_TABLE[0];
   SPEAKER = 0;
}
```

（2）input.c：需设定INPUT_PORT为P0端口，因硬件上DIP开关由P0端口作为输入，八个DIP开关为“0”电位动作，则当撷取输入值时应先反相。程序中也可以不需要和0Xff作AND闸，由于0Xff换算成二进制位为11111111b，每一个位都为“1”，无法屏蔽任何位，所以程序中可不必写，但必须判断当大于31的曲目

时，将变为数值 0，并立即由显示器显示出来，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*****************************/
void InputKey(void)
{
   KeyData = ~(INPUT_PORT) & 0xff; //P0.0~P0.7

   if (KeyData > 31)
      KeyData=0;

   Disp7Seg(KeyData);
}
```

（3）input.h：包括文件也要定义输入端口为 P0，起始演奏键为 P3.4，未输入的数值为 0X00，以及函数外部声明，如下：

```
#ifndef     _INPUT_H
#define     _INPUT_H

#define  INPUT_PORT     P0
#define  START_IO       P3_4
#define  NO_KEY         0

extern void InputKey(void);

#endif
```

（4）7Segled.c：将两位数的输入数值除以 10，则将计算出十位数的数值，而将原来的数值减去十位数乘以 10 则会变成个位数了，例如：数值 28 除以 10 后，则 number2=2 即是十位数，将 28－(2×10)=8 即是个位数，为变量 number1 的内容了，用此方法分离出个位数和十位数，再将个位数由 P1 端口输出，十位数由 P2 端口输出，就可以正确地显示出目前的曲目编号了，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "7segled.h"

/*****************************/
void Disp7Seg(Byte value)
{
   Byte number2,number1;

   number2=value/10;                    //十位数
   number1=(value-=number2*10);//个位数

   _7SEGDISP_PORT2=DISP7SEG_TABLE[number2];
   _7SEGDISP_PORT1=DISP7SEG_TABLE[number1];
}

/*****************************/
//display 0~9:"0"active
Byte RDATA DISP7SEG_TABLE[ ] =
{
   0xc0,0xf9,0xa4,0xb0,0x99,
   0x92,0x82,0xf8,0x80,0x90
};
```

（5）**7Segled.h**：定义个位数显示器由 P1 驱动，十位数显示器由 P2 驱动，以及函数外部声明，数字与显示器转换表的外部声明，如下：

```
#ifndef     _7SEGLED_H
#define     _7SEGLED_H

#define  _7SEGDISP_PORT1    P1
#define  _7SEGDISP_PORT2    P2

extern Byte RDATA DISP7SEG_TABLE[ ];
extern void Disp7Seg(Byte value);

#endif
```

汇编程序如下：

1. input.a51

```
$NOMOD51

NAME    INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT              SEGMENT CODE
    EXTRN   IDATA (KeyData)
    EXTRN   CODE (_Disp7Seg)
    PUBLIC  InputKey

    RSEG  ?PR?InputKey?INPUT
    USING   0
InputKey:
    MOV     A,P0
    CPL     A
    MOV     R0,#KeyData
    MOV     @R0,A
    SETB    C
    SUBB    A,#01FH
    JC      ?C0001
    CLR     A
    MOV     @R0,A
?C0001:
    MOV     R0,#KeyData
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Disp7Seg
    RET

    END
```

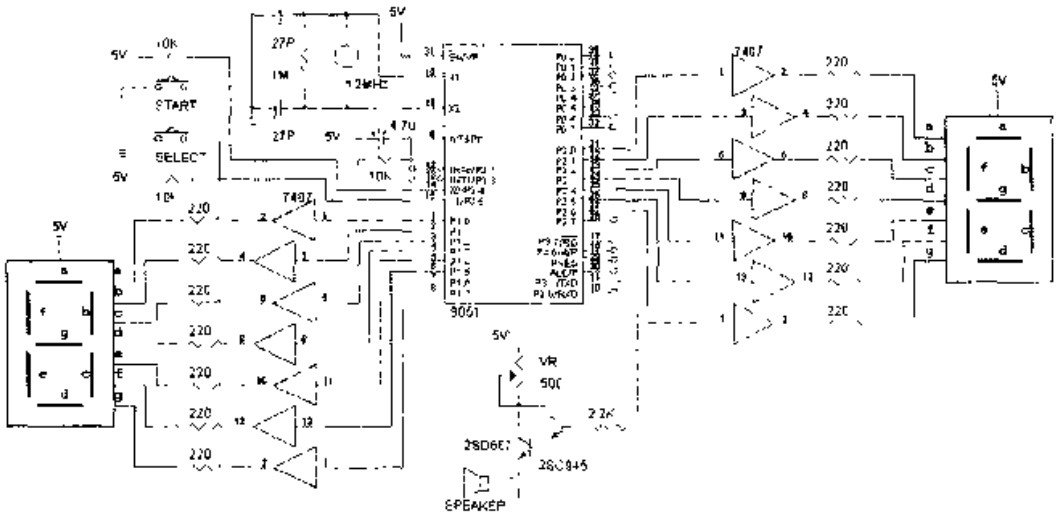
2. 2.7segled.a51

```
$NOMOD51

NAME    _7SEGLED

$INCLUDE (mtv212m.inc)

?PR?_Disp7Seg?7SEGLED           SEGMENT CODE
```

```
?CO?7SEGLED                    SEGMENT CODE
    PUBLIC  DISP7SEG_TABLE
    PUBLIC  _Disp7Seg

    RSEG  ?CO?7SEGLED
DISP7SEG_TABLE:
    DB  0C0H,0F9H,0A4H,0B0H,099H
    DB  092H,082H,0F8H,080H,090H

    RSEG  ?PR?_Disp7Seg?7SEGLED
    USING   0
_Disp7Seg:
    MOV     A,R7
    MOV     B,#0AH
    DIV     AB
    MOV     R6,A
    MOV     B,#0AH
    MUL     AB
    MOV     R5,A
    CLR     C
    MOV     A,R7
    SUBB    A,R5
    MOV     R5,A
    MOV     R7,A
    MOV     A,R6
    MOV     DPTR,#DISP7SEG_TABLE
    MOVC    A,@A+DPTR
    MOV     P2,A
    XCH     A,R7
    MOV     A,R5
    XCH     A,R7
    MOV     A,R7
    MOVC    A,@A+DPTR
    MOV     P1,A
    RET

    END
```

10-2-2 TACT 选：1～99

电路图：



软件构建的思维与解决方法

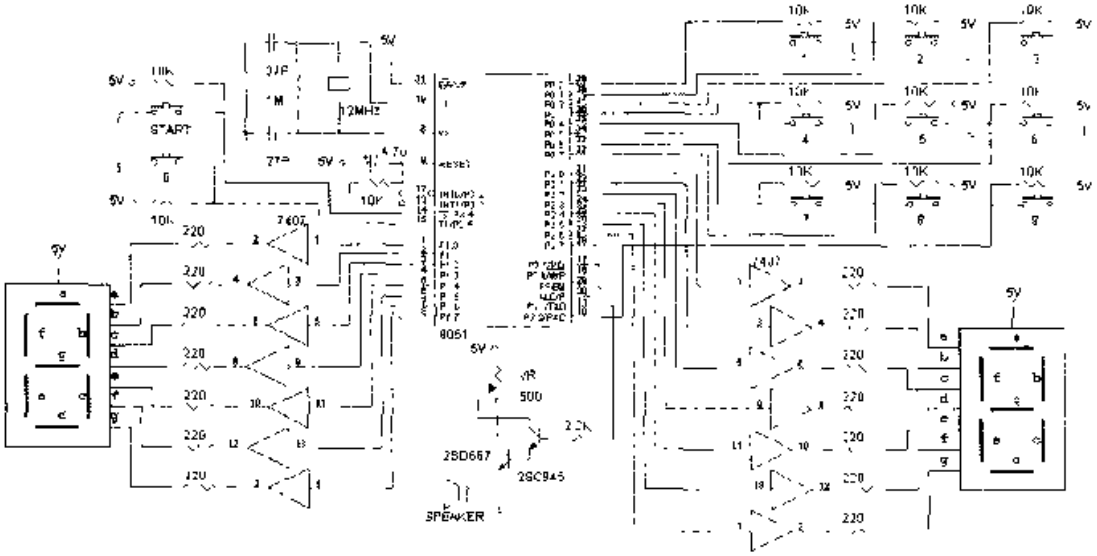
（1）input.c：当按键按一下则曲目就增加 1，一直到曲目编号为 99，此时再将按键按一下，则曲目编号将从 1 开始。每按一下则显示器立即显示按键次数，也就是曲目的编号，其余不再赘述，请参考前面章节，程序如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*******************************/
void InputHandler(void)
{
    //sel play no#
    InputKey( );

    if ( KeyBuffer != KeyData )
    {
       KeyBuffer = KeyData;
       ScanKeyCounter = 10;
    }
```

```
    else if( ScanKeyCounter != 0 )
    {
       ScanKeyCounter--;
       DelayX1ms(1);
    }
    else if ( KeyBuffer == SELECT_KEY )
    {
       if (FgSelectKey==0)
       {
          FgSelectKey = 1;
          Beep(1,17,10);

          MusicNumber++;
          if ( MusicNumber>99 )
             MusicNumber=1;
          Disp7Seg(MusicNumber);
       }
    }
    else
    {
       KeyBuffer   = NO_KEY;
       FgSelectKey = 0;
    }
}

/*****************************/
void InputKey(void)
{
    if ( SELECT_IO==0 )
       KeyData=SELECT_KEY;
    else
       KeyData=NO_KEY;
}

void InputStart(void)
{
    if ( START_IO==0 )
       KeyData=START_KEY;
    else
       KeyData=NO_KEY;
}
```

10-2-3　I/O 选曲：1～99

电路图：



软件构建的思维与解决方法

（1）input.c：有 10 个按键分别表示数字 0～9，当按下数字键一次时表示为个位数并立即显示出所按的数字，而按下数字键第二次时，表示第一次所按的数值变成为十位数，而现在所按的数字成为个位数，因而，应将十位数及个位数计算出曲目编号并将曲目编号显示出来。此时再按下数字键，因为无法表示三位数，而以喇叭响叫两声作为错误输入的提示，按下起始键演奏后才能重新选曲，即将按键次数 Number 归零，如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*******************************/
void InputHandler(void)
```

```
{
    //sel play no#
    InputKey( );

    if ( KeyBuffer != KeyData )
    {
       KeyBuffer = KeyData;
       ScanKeyCounter = 10;
    }
    else if( ScanKeyCounter != 0 )
    {
       ScanKeyCounter--;
       DelayX1ms(1);
    }
    else if ( KeyBuffer != NO_KEY )
    {
       if ( FgKeyFlag==0 )
       {
          FgKeyFlag = 1;

          Number++;
          switch ( Number )
          {
           case 1 :              //1 位数
              Beep(1,17,10);
              MusicNumber=KeyBuffer;
              Disp7Seg1(MusicNumber);
              break;
           case 2 :              //2 位数
              Beep(1,17,10);
              MusicNumber=MusicNumber*10+KeyBuffer;
              Disp7Seg2(MusicNumber);
              break;
           default :             //2 位数以上
              Beep(2,17,10);
              break;
          }
       }
    }
    else
    {
       KeyBuffer = NO_KEY;
       FgKeyFlag = 0;
    }
}
```

```
/*******************************/
void InputKey(void)
{
    if ( NUMBER1_IO==0 )
       KeyData=NUMBER1_KEY;
    else if ( NUMBER2_IO==0 )
       KeyData=NUMBER2_KEY;
    else if ( NUMBER3_IO==0 )
       KeyData=NUMBER3_KEY;
    else if ( NUMBER4_IO==0 )
       KeyData=NUMBER4_KEY;
    else if ( NUMBER5_IO==0 )
       KeyData=NUMBER5_KEY;
    else if ( NUMBER6_IO==0 )
       KeyData=NUMBER6_KEY;
    else if ( NUMBER7_IO==0 )
       KeyData=NUMBER7_KEY;
    else if ( NUMBER8_IO==0 )
       KeyData=NUMBER8_KEY;
    else if ( NUMBER9_IO==0 )
       KeyData=NUMBER9_KEY;
    else if ( NUMBER0_IO==0 )
       KeyData=NUMBER0_KEY;
    else
       KeyData=NO_KEY;
}

/*******************************/
void InputStart(void)
{
    if ( START_IO==0 )
    {
       Number=0;
       KeyData=START_KEY;
    }
    else
       KeyData=NO_KEY;
}
```

（2）input.h：数字 0～9 的 I/O 定义，如下：

```
#ifndef     _INPUT_H
#define     _INPUT_H
```

```
#define  START_IO        P3_4

#define  NUMBER1_IO P0_0
#define  NUMBER2_IO P0_1
#define  NUMBER3_IO P0_2
#define  NUMBER4_IO P0_3
#define  NUMBER5_IO P0_4
#define  NUMBER6_IO P0_5
#define  NUMBER7_IO P0_6
#define  NUMBER8_IO P0_7
#define  NUMBER9_IO P3_0
#define  NUMBER0_IO P3_5


#define  NO_KEY          0xff
#define  START_KEY       1

#define  NUMBER1_KEY1
#define  NUMBER2_KEY2
#define  NUMBER3_KEY3
#define  NUMBER4_KEY4
#define  NUMBER5_KEY5
#define  NUMBER6_KEY6
#define  NUMBER7_KEY7
#define  NUMBER8_KEY8
#define  NUMBER9_KEY9
#define  NUMBER0_KEY0

extern void InputHandler(void);
extern void InputKey(void);
extern void InputStart(void);

#endif
```

（3）7Segled.c：七段显示器的函数模块，当第一次按下数字键表示为个位数，则调用 Disp7Seg1( )函数而将十位数的显示器清除为 0，而第二次按下数字键表示选取两位数曲目编号，则调用 Disp7Seg2( )函数而将个位数及十位数分离出来，个位数由 P1 端口显示，十位数由 P2 显示，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
```

```
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "7segled.h"

/*****************************/
void Disp7Seg1(Byte value)
{
    _7SEGDISP_PORT2=DISP7SEG_TABLE[0];
    _7SEGDISP_PORT1=DISP7SEG_TABLE[value];
}

/*****************************/
void Disp7Seg2(Byte value)
{
    Byte number2,number1;

    number2=value/10;              //十位数
    number1=(value-=number2*10); //个位数

    _7SEGDISP_PORT2=DISP7SEG_TABLE[number2];
    _7SEGDISP_PORT1=DISP7SEG_TABLE[number1];
}

/*****************************/
//display 0~9:"0"active
Byte RDATA DISP7SEG_TABLE[ ] =
{
    0xc0,0xf9,0xa4,0xb0,0x99,
    0x92,0x82,0xf8,0x80,0x90
};
```

汇编程序如下：

1. input.a51

```
$NOMOD51

NAME    INPUT

$INCLUDE (mtv212m.inc)

?PR?InputHandler?INPUT          SEGMENT CODE
?PR?InputKey?INPUT              SEGMENT CODE
```

```
?PR?InputStart?INPUT                    SEGMENT CODE
    EXTRN   BIT (FgKeyFlag)
    EXTRN   IDATA (KeyData)
    EXTRN   IDATA (KeyBuffer)
    EXTRN   IDATA (Number?)
    EXTRN   IDATA (ScanKeyCounter)
    EXTRN   IDATA (MusicNumber)
    EXTRN   CODE (_DelayX1ms)
    EXTRN   CODE (_Beep)
    EXTRN   CODE (_Disp7Seg1)
    EXTRN   CODE (_Disp7Seg2)
    PUBLIC  InputStart
    PUBLIC  InputKey
    PUBLIC  InputHandler

    RSEG  ?PR?InputHandler?INPUT
    USING   0
InputHandler:
    LCALL   InputKey
    MOV     R0,#KeyData
    MOV     A,@R0
    MOV     R7,A
    MOV     R0,#KeyBuffer
    XRL     A,@R0
    JZ      ?C0001
    MOV     A,R7
    MOV     @R0,A
    MOV     R0,#ScanKeyCounter
    MOV     @R0,#0AH
    RET
?C0001:
    MOV     R0,#ScanKeyCounter
    MOV     A,@R0
    JZ      ?C0003
    DEC     @R0
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    RET
?C0003:
    MOV     R0,#KeyBuffer
    MOV     A,@R0
    CPL     A
    JZ      ?C0005
    JB      FgKeyFlag,?C0012
```

```
    SETB    FgKeyFlag
    MOV     R0,#Number?
    INC     @R0
    MOV     A,@R0
    ADD     A,#0FEH
    JZ      ?C0009
    INC     A
    JNZ     ?C0010
?C0008:
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep
    MOV     R0,#KeyBuffer
    MOV     A,@R0
    MOV     R7,A
    MOV     R0,#MusicNumber
    MOV     @R0,A
    LCALL   _Disp7Seg1
    RET
?C0009:
    MOV     R7,#01H
    MOV     R6,#C0H
    MOV     R5,#C11H
    MOV     R3,#CAH
    LCALL   _Beep
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     B,#0AH
    MUL     AB
    MOV     R0,#KeyBuffer
    ADD     A,@R0
    MOV     R7,A
    MOV     R0,#MusicNumber
    MOV     @R0,A
    LCALL   _Disp7Seg2
    RET
?C0010:
    MOV     R7,#02H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep
    RET
```

```
?C0005:
    MOV     R0,#KeyBuffer
    MOV     @R0,#0FFH
    CLR     FgKeyFlag
?C0012:
    RET

    RSEG  ?PR?InputKey?INPUT
    USING   0
InputKey:
    JB      P0_0,?C0013
    MOV     R0,#KeyData
    MOV     @R0,#01H
    RET
?C0013:
    JB      P0_1,?C0015
    MOV     R0,#KeyData
    MOV     @R0,#02H
    RET
?C0015:
    JB      P0_2,?C0017
    MOV     R0,#KeyData
    MOV     @R0,#03H
    RET
?C0017:
    JB      P0_3,?C0019
    MOV     R0,#KeyData
    MOV     @R0,#04H
    RET
?C0019:
    JB      P0_4,?C0021
    MOV     R0,#KeyData
    MOV     @R0,#05H
    RET
?C0021:
    JB      P0_5,?C0023
    MOV     R0,#KeyData
    MOV     @R0,#06H
    RET
?C0023:
    JB      P0_6,?C0025
    MOV     R0,#KeyData
    MOV     @R0,#07H
    RET
?C0025:
```

```
    JB      P0_7,?C0027
    MOV     R0,#KeyData
    MOV     @R0,#08H
    RET
?C0027:
    JB      P3_0,?C0029
    MOV     R0,#KeyData
    MOV     @R0,#09H
    RET
?C0029:
    JB      P3_5,?C0031
    CLR     A
    MOV     R0,#KeyData
    MOV     @R0,A
    RET
?C0031:
    MOV     R0,#KeyData
    MOV     @R0,#0FFH
?C0033:
    RET

    RSEG  ?PR?InputStart?INPUT
    USING   0
InputStart:
    JB      P3_4,?C0034
    CLR     A
    MOV     R0,#Number?
    MOV     @R0,A
    MOV     R0,#KeyData
    MOV     @R0,#01H
    RET
?C0034:
    MOV     R0,#KeyData
    MOV     @R0,#0FFH
?C0036:
    RET

    END
```

2. 2.7segled.a51

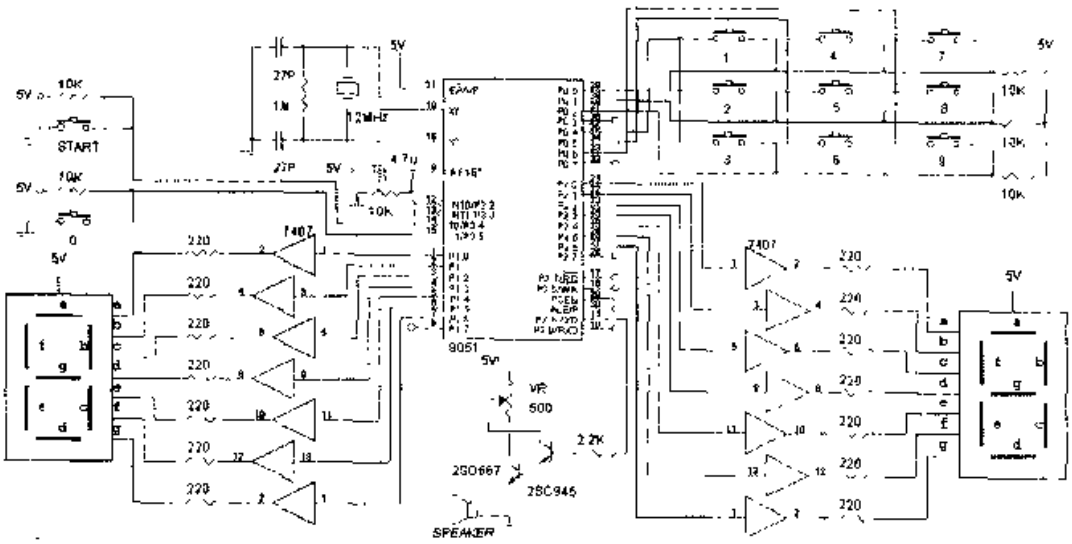
```
$NOMOD51

NAME    _7SEGLED
```

```
$INCLUDE (mtv212m.inc)

?PR?_Disp7Seg1?7SEGLED      SEGMENT CODE
?PR?_Disp7Seg2?7SEGLED          SEGMENT CODE
?CO?7SEGLED                 SEGMENT CODE
    PUBLIC  DISP7SEG_TABLE
    PUBLIC  _Disp7Seg2
    PUBLIC  _Disp7Seg1

    RSEG  ?CO?7SEGLED
DISP7SEG_TABLE:
    DB  0C0H,0F9H,0A4H,0B0H,099H
    DB  092H,082H,0F8H,080H,090H

    RSEG  ?PR?_Disp7Seg1?7SEGLED
    USING   0
_Disp7Seg1:
    MOV     DPTR,#DISP7SEG_TABLE
    CLR     A
    MOVC    A,@A+DPTR
    MOV     P2,A
    MOV     A,R7
    MOVC    A,@A+DPTR
    MOV     P1,A
    RET

    RSEG  ?PR?_Disp7Seg2?7SEGLED
    USING   0
_Disp7Seg2:
    MOV     A,R7
    MOV     B,#0AH
    DIV     AB
    MOV     R6,A
    MOV     B,#0AH
    MUL     AB
    MOV     R5,A
    CLR     C
    MOV     A,R7
    SUBB    A,R5
    MOV     R5,A
    MOV     R7,A
    MOV     A,R6
    MOV     DPTR,#DISP7SEG_TABLE
    MOVC    A,@A+DPTR
    MOV     P2,A
```

```
XCH     A,R7
MOV     A,R5
XCH     A,R7
MOV     A,R7
MOVC    A,@A+DPTR
MOV     P1,A
RET

END
```

10-2-4　SCAN 选曲：1～99

电路图：



软件构建的思维与解决方法

（1）input.c：行扫描线分别由 P0.4～P0.6 输出，而且是“0”电位动作，当行扫描线 1 为“0”电位，则侦测数字 1～3，行扫描线 2 为“0”电位则侦测数字 4～6，行扫描线 3 为“0”电位则侦测数字 7～9，而数字 0 则由 P3.5 作侦测，当 P3.5=“0”电位则表示数字 0 被按下了，再根据是第一次按下键或第二次按下键而分别执行 Disp7Seg1( )函数和 Disp7Seg2( )函数。当第二次按下键，应将第一次按下键的数值 MusicNumber 乘以 10 再加上目前的按键值 KeyBuffer 作为曲目编号并进行显示，程序如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
```

```
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*****************************/
void InputHandler(void)
{
   //sel play no#
   InputKey( );

   if ( KeyBuffer != KeyData )
   {
      KeyBuffer = KeyData;
      ScanKeyCounter = 10;
   }
   else if( ScanKeyCounter != 0 )
   {
      ScanKeyCounter--;
      DelayX1ms(1);
   }
   else if ( KeyBuffer != NO_KEY )
   {
      if ( FgKeyFlag==0 )
      {
         FgKeyFlag = 1;

         Number++;
         switch ( Number )
         {
          case 1 :              //1位数
             Beep(1,17,10);
             MusicNumber=KeyBuffer;
             Disp7Seg1(MusicNumber);
             break;
          case 2 :              //2位数
             Beep(1,17,10);
             MusicNumber=MusicNumber*10+KeyBuffer;
             Disp7Seg2(MusicNumber);
             break;
          default :             //2位数以上
```

```
            Beep(2,17,10);
            break;
          }
        }
    }
    else
    {
      KeyBuffer = NO_KEY;
      FgKeyFlag = 0;
    }
}

/*****************************/
void InputKey(void)
{
    Byte  keytmp;

    KeyData = NO_KEY;

    //scan line1:"0" active
    COLUMN1_PIN = 0;              //P0_4
    COLUMN2_PIN = 1;              //P0_5
    COLUMN3_PIN = 1;              //P0_6
    keytmp = ~(INPUT_PORT) & 0x07;   //P0.0~P0.2
    switch( keytmp )
    {
     case 1 :
       KeyData = NUMBER1_KEY;
       break;
     case 2 :
       KeyData = NUMBER2_KEY;
       break;
     case 4 :
       KeyData = NUMBER3_KEY;
       break;
     default :
       break;
    }

    //scan line2:"0" active
    COLUMN1_PIN = 1;
    COLUMN2_PIN = 0;
    COLUMN3_PIN = 1;
    keytmp = ~(INPUT_PORT) & 0x07;   //P0.0~P0.2
    switch( keytmp )
```

```
    {
     case 1 :
        KeyData = NUMBER4_KEY;
        break;
     case 2 :
        KeyData = NUMBER5_KEY;
        break;
     case 4 :
        KeyData = NUMBER6_KEY;
        break;
     default :
        break;
    }

    //scan line3:"0" active
    COLUMN1_PIN = 1;
    COLUMN2_PIN = 1;
    COLUMN3_PIN = 0;
    keytmp = ~(INPUT_PORT) & 0x07;   //P0.0~P0.2
    switch( keytmp )
    {
     case 1 :
        KeyData = NUMBER7_KEY;
        break;
     case 2 :
        KeyData = NUMBER8_KEY;
        break;
     case 4 :
        KeyData = NUMBER9_KEY;
        break;
     default :
        break;
    }

    //scan line3:not active
    COLUMN3_PIN = 1;

    if ( NUMBER0_IO==0 )
      KeyData = NUMBER0_KEY;
}

/*****************************/
void InputStart(void)
{
    if ( START_IO==0 )
```

```
    {
      Number=0;
      KeyData=START_KEY;
    }
    else
      KeyData=NO_KEY;
}
```

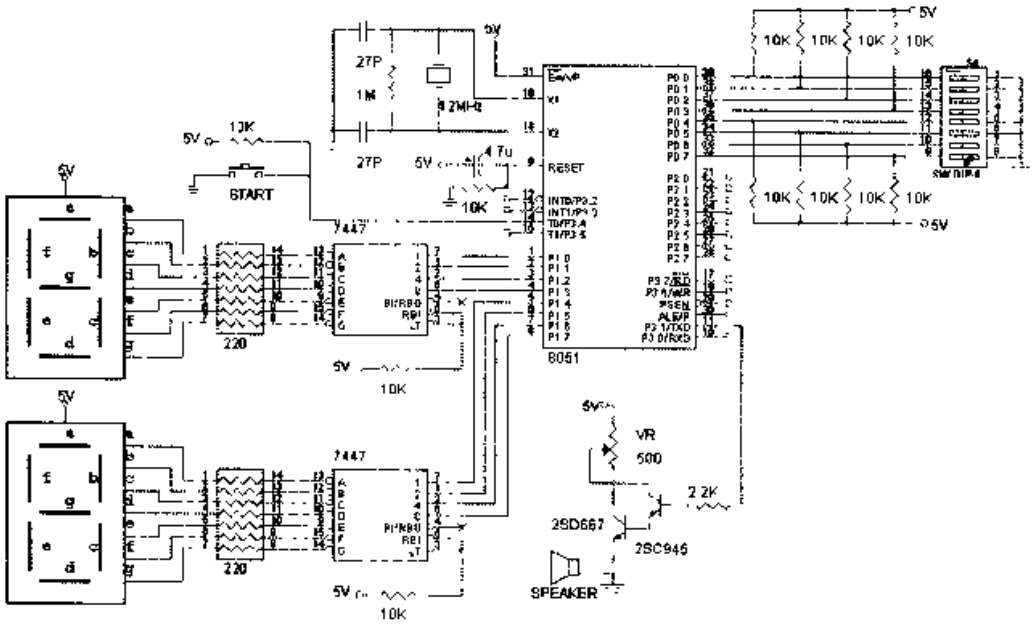
（2）input.h：由于有数字键 0，其数值为 0X00，因此，在定义未按下键的数值要改为 0Xff，否则，将无法侦测出数字 0，并定义侦测数字键 0 的 I/O 为 P3.5，其他 I/O 的定义及键值，函数外部声明等，如下：

```
#ifndef     _INPUT_H
#define     _INPUT_H

#define  START_IO       P3_4
#define  NUMBER0_IO P3_5

#define  INPUT_PORT P   0    //P0.0~P0.2
#define  COLUMN1_PINP0_4
#define  COLUMN2_PINP0_5
#define  COLUMN3_PINP0_6

#define  NO_KEY         0xff
#define  START_KEY      1

#define  NUMBER1_KEY1
#define  NUMBER2_KEY2
#define  NUMBER3_KEY3
#define  NUMBER4_KEY4
#define  NUMBER5_KEY5
#define  NUMBER6_KEY6
#define  NUMBER7_KEY7
#define  NUMBER8_KEY8
#define  NUMBER9_KEY9
#define  NUMBER0_KEY0

extern void InputHandler(void);
extern void InputKey(void);
extern void InputStart(void);

#endif
```

10-3 7447 显示器 X2

目的：利用 IC 7447 BCD 至七段显示器的译码器直接驱动，可节省一个 I/O 端口。

电路图：



原理：IC 7447 是专门用来推动七段显示器的译码 IC，当输入 BCD 码时则其输出将会译码出七段显示器的电位。例如：当输入为 BCD 码 0001，即十进制的 0X01 时，则 B、C 引脚（即第 12、11 脚）将输出为“0”电位，其余 A、D、E、F、G 引脚输出为“1”电位，则七段显示器将点亮 b 和 c 段，而呈现出数字“1”的字形出来，其余 BCD 码的译码如下表所示：

BCD	数字	A	B	C	D	E	F	G	数值	
00000	0	0	0	0	0	0	1	1	0XC0	
0001	1	1	0	0	1	1	1	1	0Xf9	
0010	2	0	0	1	0	0	1	0	0Xa4	
0011	3	0	0	0	0	1	1	0	0Xb0	
0100	4	1	0	0	1	1	0	0	0X99	
0101	5	0	1	0	0	1	0	0	0X92	
0110	6	0	1	0	0	0	0	0	0X82	
0111	7	0	0	0	1	1	1	1	0Xf8	
1000	8	0	0	0	0	0	0	0	0X80	
1001	9	0	0	0	0	1	0	0	0X90	

而显示器的个位数利用 P1 端口的 P1.0～P1.3 来输出个位数 BCD 码，显示器的十位数则利用 P1 端口的 P1.4～P1.7 来输出十位数的 BCD 码，因而，只要使用一个端口即可控制两个七段显示器而可以选 1～99 首。如果利用 I/O 端口直接驱动，两个七段显示器则需两个 I/O 端口，而可以大大减少 I/O 端口的使用。

10-3-1　DIP 选曲：1～99

软件构建的思维与解决方法

直接由 IC7447 推动，所以只要一个 I/O 端口输出 BCD 码即可，其中 P1.0～P1.3 控制个位数的显示器，而 P1.4～P1.7 则控制十位数的显示器，也不需要将数字 0～9 作取表的转换了，程序的写法也变得较容易、清楚，现分述如下：

（1）initial.c：初始化两个七段显示器使其显示为“00”，程序只需令 P1 端口输出为 0X00 即可，如下：

```
/*****************************/
void InitialCpuIO(void)
{
   //display "00"
   _7SEGDISP_PORT1=0x00;
   SPEAKER = 0;
}
```

（2）input.c：DIP 开关输入的选曲，将 P0 端口反相后再撷取至 KeyData 变量内，当大于 99 则存入 0X00，以表示超过显示范围，因两位数的七段显示器其最大值为 99，并立即显示，如下：

```
/*****************************/
void InputKey(void)
{
   KeyData = ~(INPUT_PORT) & 0xff; //P0.0~P0.7

   if (KeyData > 99)
     KeyData=0;

   Disp7Seg(KeyData);
}
```

（3）7 Segled.c：七段显示器的控制方式通过 P1 端口输出 BCD 码，再通过 IC7447 自动译码成七段显示器的数值，而十位数的 BCD 码由 P1.4～P1.7 输出，所以应十位数 number 2 左移四位再加上个位数 number 1 的值，作为 P1 端口输出，才能正确地显示出实际的数字，程序如下：

```
/*****************************/
void Disp7Seg(Byte value)
{
    Byte number2,number1;

    number2=value/10;              //十位数
    number1=(value-=number2*10); //个位数

    _7SEGDISP_PORT1=(number2<<4)+number1;
}
```

（4）7segled.h：Dis7Seg( )函数的外部声明以及定义驱动七段显示器为 P1 端口，如下：

```
#ifndef     _7SEGLED_H
#define     _7SEGLED_H

#define  _7SEGDISP_PORT1     P1

extern void Disp7Seg(Byte value);

#endif
```

汇编程序如下：

1. input.a51

```
$NOMOD51

NAME    INPUT

$INCLUDE (mtv212m.inc)

?PR?InputKey?INPUT          SEGMENT CODE
    EXTRN   IDATA (KeyData)
    EXTRN   CODE (_Disp7Seg)
    PUBLIC  InputKey

    RSEG  ?PR?InputKey?INPUT
    USING   0
InputKey:
    MOV     A,P0
    CPL     A
    MOV     R0,#KeyData
    MOV     @R0,A
    SETB    C
    SUBB    A,#063H
```

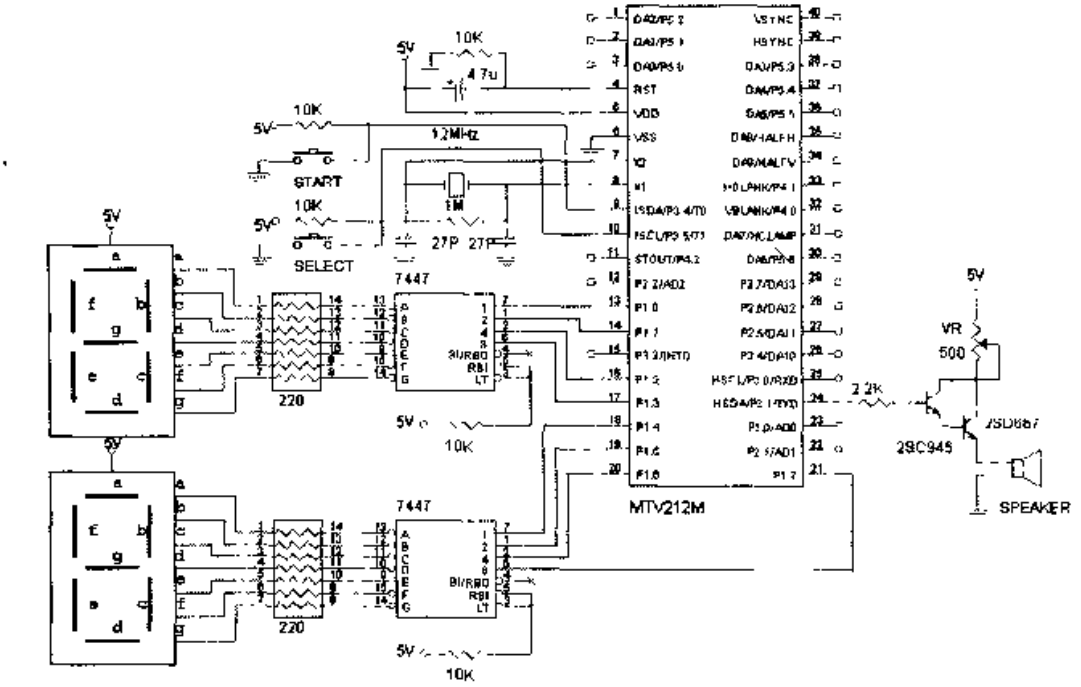
```
    JC      ?C0001
    CLR     A
    MOV     @R0,A
?C0001:
    MOV     R0,#KeyData
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Disp7Seg
    RET

    END
```

2. 2.7segled.a51

```
$NOMOD51

NAME    _7SEGLED

$INCLUDE (mtv212m.inc)

?PR?_Disp7Seg?7SEGLED       SEGMENT CODE
    PUBLIC  _Disp7Seg

    RSEG  ?PR?_Disp7Seg?7SEGLED
    USING   0
_Disp7Seg:
    MOV     A,R7
    MOV     B,#0AH
    DIV     AB
    MOV     R6,A
    MOV     B,#0AH
    MUL     AB
    MOV     R5,A
    CLR     C
    MOV     A,R7
    SUBB    A,R5
    MOV     R5,A
    MOV     R7,A
    MOV     A,R6
    SWAP    A
    ANL     A,#0F0H
    ADD     A,R5
    MOV     P1,A
    RET

    END
```

10-3-2 TACT 选曲：1～99

电路图：



软件构建的思维与解决方法

（1）input.c：先侦测是否按下选曲键，如果是，则每按下一次就将曲目编号加 1，并在 initial.c 模块下，令曲目编号初始化为 0X00，即使一直按着也只是加 1 而已。利用判断 FgSelectKey 标志是否等于“0”即可达到，每按一次则喇叭响叫一声，当超过第 99 首时，则从第一首开始选曲，而当按下“起始键”才开始进行演奏，其程序如下：

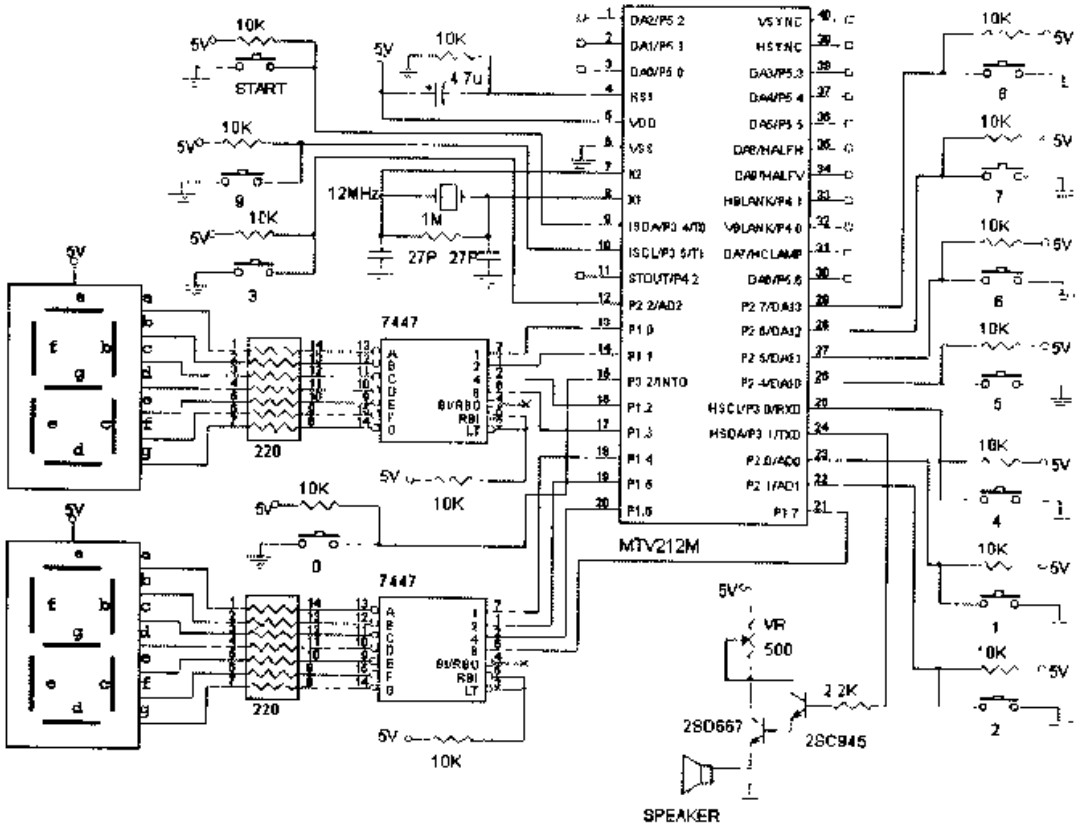
```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"
```

```
/*****************************/
void InputHandler(void)
{
    //sel play no#
    InputKey( );

    if ( KeyBuffer != KeyData )
    {
       KeyBuffer = KeyData;
       ScanKeyCounter = 10;
    }
    else if( ScanKeyCounter != 0 )
    {
       ScanKeyCounter--;
       DelayX1ms(1);
    }
    else if ( KeyBuffer == SELECT_KEY )
    {
       if (FgSelectKey==0)
       {
          FgSelectKey = 1;
          Beep(1,17,10);

          MusicNumber++;
          if ( MusicNumber>99 )
             MusicNumber=1;
          Disp7Seg(MusicNumber);
       }
    }
    else
    {
       KeyBuffer   = NO_KEY;
       FgSelectKey = 0;
    }
}

/*****************************/
void InputKey(void)
{
    if ( SELECT_IO==0 )
       KeyData=SELECT_KEY;
    else
       KeyData=NO_KEY;
}
```

```
void InputStart(void)
{
   if ( START_IO==0 )
     KeyData=START_KEY;
   else
     KeyData=NO_KEY;
}
```

10-3-3 I/O 选曲：1～99

电路图：



软件构建的思维与解决方法

（1）input.c：一个数字对应一个 I/O 的方式，会很浪费 I/O 引脚，但程序只要依照数字的对应 I/O 一直判断下来，当 I/O 为“0”电位表示已按下键并写入按键的键值即可，而将数字显示在显示器上，其算法及程序结构不再重复，其程序如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
```

```
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*****************************/
void InputHandler(void)
{
    //sel play no#
    InputKey( );

    if ( KeyBuffer != KeyData )
    {
       KeyBuffer = KeyData;
       ScanKeyCounter = 10;
    }
    else if( ScanKeyCounter != 0 )
    {
       ScanKeyCounter--;
       DelayX1ms(1);
    }
    else if ( KeyBuffer != NO_KEY )
    {
       if ( FgKeyFlag==0 )
       {
          FgKeyFlag = 1;

          KeyNumber++;
          switch ( KeyNumber )
          {
           case 1 :              //1 位数
              Beep(1,17,10);
              MusicNumber=KeyBuffer;
              Disp7Seg1(MusicNumber);
              break;
           case 2 :              //2 位数
              Beep(1,17,10);
              MusicNumber=MusicNumber*10+KeyBuffer;
              Disp7Seg2(MusicNumber);
              break;
           default :             //2 位数以上
```

```
            Beep(2,17,10);
            break;
          }
        }
      }
      else
      {
        KeyBuffer = NO_KEY;
        FgKeyFlag = 0;
      }
}

/*****************************/
void InputKey(void)
{
    if ( NUMBER1_IO==0 )
      KeyData=NUMBER1_KEY;
    else if ( NUMBER2_IO==0 )
      KeyData=NUMBER2_KEY;
    else if ( NUMBER3_IO==0 )
      KeyData=NUMBER3_KEY;
    else if ( NUMBER4_IO==0 )
      KeyData=NUMBER4_KEY;
    else if ( NUMBER5_IO==0 )
      KeyData=NUMBER5_KEY;
    else if ( NUMBER6_IO==0 )
      KeyData=NUMBER6_KEY;
    else if ( NUMBER7_IO==0 )
      KeyData=NUMBER7_KEY;
    else if ( NUMBER8_IO==0 )
      KeyData=NUMBER8_KEY;
    else if ( NUMBER9_IO==0 )
      KeyData=NUMBER9_KEY;
    else if ( NUMBER0_IO==0 )
      KeyData=NUMBER0_KEY;
    else
      KeyData=NO_KEY;
}

/*****************************/
void InputStart(void)
{
    if ( START_IO==0 )
    {
      KeyNumber=0;
```

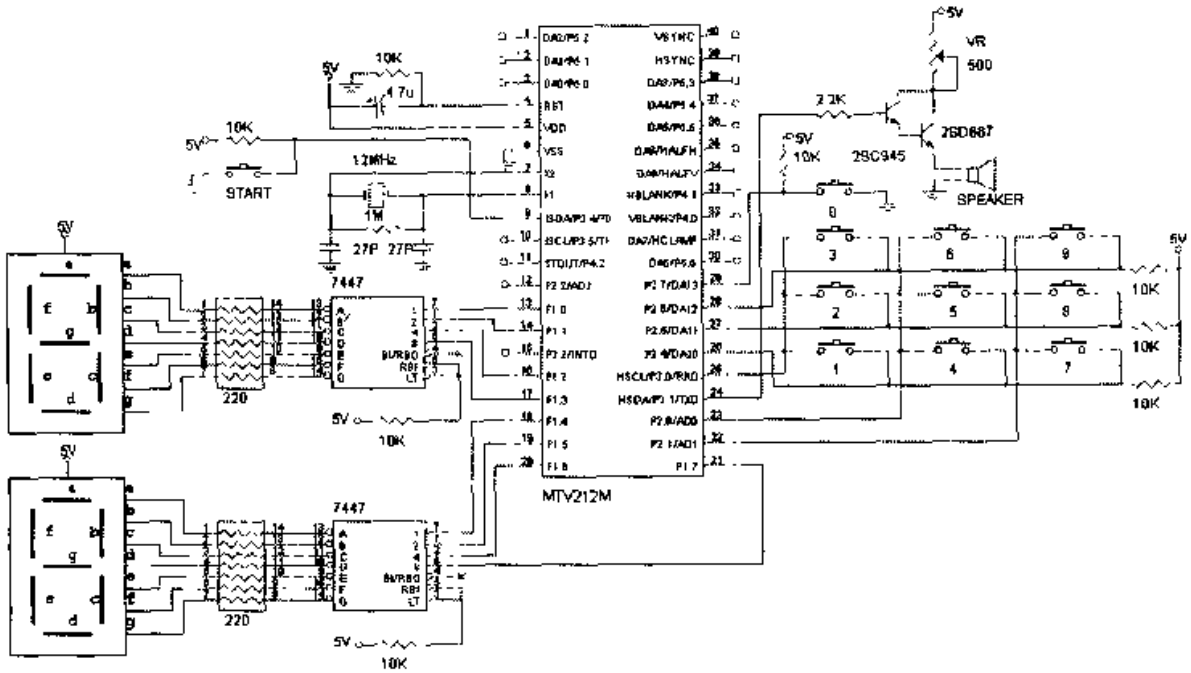
```
        KeyData=START_KEY;
    }
    else
        KeyData=NO_KEY;
}
```

（2）input.h：其包括文件的内容要注意 NO_KEY 的定义修正为 0Xff，不能再定义为 0X00 了，因子值 0X00 已被数字键 NUMBER0_KEY 所定义，即数字键 0 的定义内容如下：

```
#ifndef     _INPUT_H
#define     _INPUT_H

#define  START_IO       P3_4

#define  NUMBER1_IO P2_0
#define  NUMBER2_IO P2_1
#define  NUMBER3_IO P2_2
#define  NUMBER4_IO P3_0
#define  NUMBER5_IO P2_4
#define  NUMBER6_IO P2_5
#define  NUMBER7_IO P2_6
#define  NUMBER8_IO P2_7
#define  NUMBER9_IO P3_5
#define  NUMBER0_IO P3_2

#define  NO_KEY          0xff
#define  START_KEY       1

#define  NUMBER1_KEY1
#define  NUMBER2_KEY2
#define  NUMBER3_KEY3
#define  NUMBER4_KEY4
#define  NUMBER5_KEY5
#define  NUMBER6_KEY6
#define  NUMBER7_KEY7
#define  NUMBER8_KEY8
#define  NUMBER9_KEY9
#define  NUMBER0_KEY0

extern void InputHandler(void);
extern void InputKey(void);
extern void InputStart(void);

#endif
```

10-3-4 SCAN 选曲：1～99

电路图：



软件构建的思维与解决方法

（1）7Segled.c：每按下数字键 0～9 后，则立即由显示器显示出此数字，当第一次按下数字代表个位数，则在两个显示器的右边显示，而在两个显示器的左边表示为十位数则显示 0；当继续按下第二次的数字键，则右边个位数的数字将移至左边变成十位数，而所按下的数字则显示在左边为个位数，例如：第一次按下 2，则两个显示器将显示“02”，再按第二次数字为 7，则两个显示器将显示为“27”，即使再按下数字键，喇叭将响叫二声作为错误输入提示，控制显示器的输出，其程序如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "7segled.h"
```

```
/*****************************/
void Disp7Seg1(Byte value)
{
    _7SEGDISP_PORT1=0x00+value;
}

/*****************************/
void Disp7Seg2(Byte value)
{
    Byte number2,number1;

    number2=value/10;                    //十位数
    number1=(value-=number2*10);         //个位数

    _7SEGDISP_PORT1=(number2<<4)+number1;
}
```

（2）**input.c**：矩阵式扫描侦测函数以及按键之后显示器输出的调用函数，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*****************************/
void InputHandler(void)
{
    //sel play no#
    InputKey( );

    if ( KeyBuffer != KeyData )
    {
       KeyBuffer = KeyData;
       ScanKeyCounter = 10;
    }
    else if( ScanKeyCounter != 0 )
```

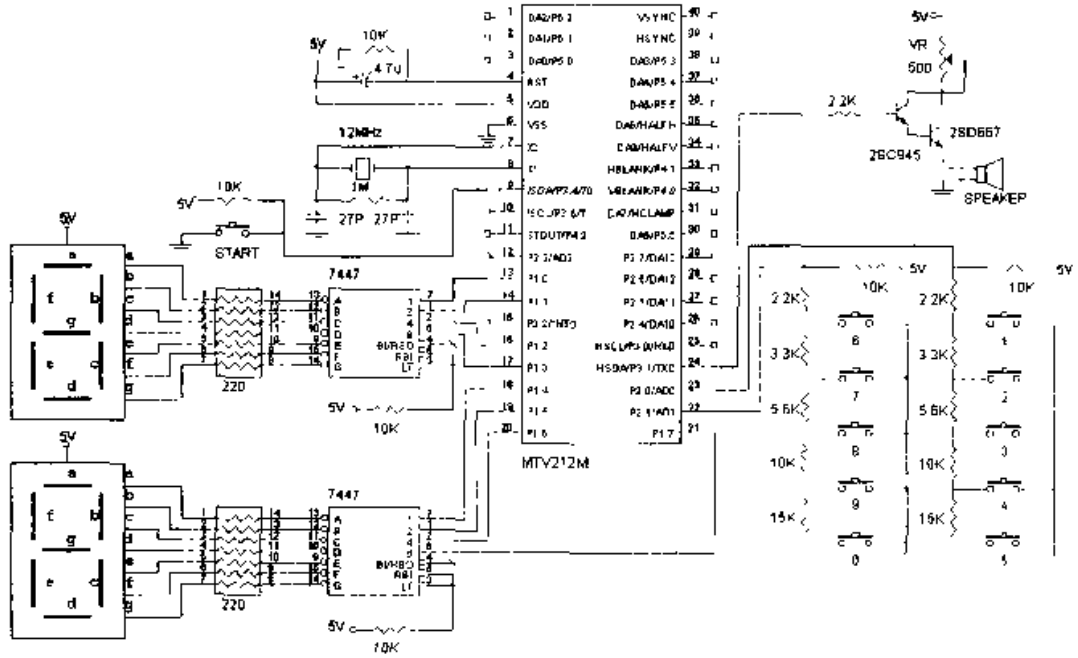
```
    {
      ScanKeyCounter--;
      DelayX1ms(1);
    }
    else if ( KeyBuffer != NO_KEY )
    {
      if ( FgKeyFlag==0 )
      {
        FqKeyFlag = 1;

        KeyNumber++;
        switch ( KeyNumber )
        {
         case 1 :              //1 位数
           Beep(1,17,10);
           MusicNumber=KeyBuffer;
           Disp7Seg1(MusicNumber);
           break;
         case 2 :              //2 位数
           Beep(1,17,10);
           MusicNumber=MusicNumber*10+KeyBuffer;
           Disp7Seg2(MusicNumber);
           break;
         default :             //2 位数以上
           Beep(2,17,10);
           break;
        }
      }
    }
    else
    {
      KeyBuffer = NO_KEY;
      FgKeyFlag = 0;
    }
}

/*****************************/
void InputKey(void)
{
    Byte  keytmp;

    KeyData = NO_KEY;

    //scan line1:"0" active
    COLUMN1_PIN = 0;              //P3.0
```

```
COLUMN2_PIN = 1;              //P2.0
COLUMN3_PIN = 1;              //P2.1
keytmp = ~(INPUT_PORT) & 0x70;   //P2.4~P2.6
keytmp >>=4;
switch( keytmp )
{
 case 1 :
    KeyData = NUMBER1_KEY;
    break;
 case 2 :
    KeyData = NUMBER2_KEY;
    break;
 case 4 :
    KeyData = NUMBER3_KEY;
    break;
 default :
    break;
}

//scan line2:"0" active
COLUMN1_PIN = 1;
COLUMN2_PIN = 0;
COLUMN3_PIN = 1;
keytmp = ~(INPUT_PORT) & 0x70;   //P2.4~P2.6
keytmp >>=4;
switch( keytmp )
{
 case 1 :
    KeyData = NUMBER4_KEY;
    break;
 case 2 :
    KeyData = NUMBER5_KEY;
    break;
 case 4 :
    KeyData = NUMBER6_KEY;
    break;
 default :
    break;
}

//scan line3:"0" active
COLUMN1_PIN = 1;
COLUMN2_PIN = 1;
COLUMN3_PIN = 0;
keytmp = ~(INPUT_PORT) & 0x70;   //P2.4~P2.6
```

```
        keytmp >>=4;
        switch( keytmp )
        {
         case 1 :
            KeyData = NUMBER7_KEY;
            break;
         case 2 :
            KeyData = NUMBER8_KEY;
            break;
         case 4 :
            KeyData = NUMBER9_KEY;
            break;
         default :
            break;
        }

        //scan line3:not active
        COLUMN3_PIN = 1;

        if ( NUMBER0_IO==0 )
           KeyData=NUMBER0_KEY;
    }

    /*****************************/
    void InputStart(void)
    {
        if ( START_IO==0 )
        {
           KeyNumber=0;
           KeyData=START_KEY;
        }
        else
           KeyData=NO_KEY;
    }
```

10-3-5 ADC选曲：1~99

电路图：



软件构建的思维与解决方法

既然是以外部数据存储器的形式才能存取 ADC 缓存器，initial.c、global.c 及 global.h 都作初始化及缓存器的寻址，此处不再重述，然而 main.c 模块以函数调用使程序结构更清晰，分述如下。

（1）main.c：不断地询问按键是否按下，因为在主程序中有做此不断询问的动作，因此，当外界按下数字键，就能被侦测到，而能将所按下的数字立即由显示器显示出来。另外，必须轮询是否按下起始演奏键，当按下时则立即演奏刚才所输入的曲目编号，因为在 main( )函数通过 while(1)反复循环才能达到此功能，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*****************************/
```

```
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      //sel play no#
      InputHandler( );

      //detect start key
      InputStart( );
      if ( KeyData == START_KEY )
      {
         Music(MusicNumber);
         DelayX10ms(200);
      }
   }
}
```

（2）input.c：以ADC侦测按键的函数InputKey( )，接收到按键后的处理函数InputHander( )以及按下第一次、第二次显示器的输出控制、调用函数。程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*****************************/
void InputHandler(void)
{
   //sel play no#
   InputKey( );

   if ( KeyBuffer != KeyData )
   {
      KeyBuffer = KeyData;
      ScanKeyCounter = 10;
```

```
    }
    else if( ScanKeyCounter != 0 )
    {
       ScanKeyCounter--;
       DelayX1ms(1);
    }
    else if ( KeyBuffer != NO_KEY )
    {
       if ( FgKeyFlag==0 )
       {
          FgKeyFlag = 1;

          KeyNumber++;
          switch ( KeyNumber )
          {
           case 1 :              //1 位数
              Beep(1,17,10);
              MusicNumber=KeyBuffer;
              Disp7Seg1(MusicNumber);
              break;
           case 2 :              //2 位数
              Beep(1,17,10);
              MusicNumber=MusicNumber*10+KeyBuffer;
              Disp7Seg2(MusicNumber);
              break;
           default :             //2 位数以上
              Beep(2,17,10);
              break;
          }
       }
    }
    else
    {
       KeyBuffer = NO_KEY;
       FgKeyFlag = 0;
    }
}

/*****************************/
void InputKey(void)
{
    Byte  keytmp;

    KeyData  = NO_KEY;            //NO_KEY=0
```

```
    XFR_ADC=0x81;                    //adc channel=1
    DelayX1ms(2);
    keytmp = XFR_ADC & 0x3f;         //6 bit  2^6=0x3f
    if ( keytmp < (12+4) )           KeyData = NUMBER1_KEY;
    else if ( keytmp < (23+4) )          KeyData = NUMBER2_KEY;
    else if ( keytmp < (34+4) )          KeyData = NUMBER3_KEY;
    else if ( keytmp < (44+4) )          KeyData = NUMBER4_KEY;
    else if ( keytmp < (50+4) )          KeyData = NUMBER5_KEY;

    XFR_ADC=0x82;                    //adc channel=2
    DelayX1ms(2);
    keytmp = XFR_ADC & 0x3f;
    if ( keytmp < (12+4) )           KeyData = NUMBER6_KEY;
    else if ( keytmp < (23+4) )          KeyData = NUMBER7_KEY;
    else if ( keytmp < (34+4) )          KeyData = NUMBER8_KEY;
    else if ( keytmp < (44+4) )          KeyData = NUMBER9_KEY;
    else if ( keytmp < (50+4) )          KeyData = NUMBER0_KEY;

    XFR_ADC=0x00;                    //disable adc
}

/*****************************/
void InputStart(void)
{
    if ( START_IO==0 )
    {
      KeyNumber=0;
      KeyData=START_KEY;
    }
    else
      KeyData=NO_KEY;
}
```

10-3-6 连续按键选曲：1～99

目的： 可连续一直输入曲目编号，改善只能输入两次的缺点。

1. 软件构建的思维

第一次按下的数字作为个位数的曲目编号，而第二次按下的数字成为个位数而第一次按下的数字变成十位数，当再按下第三次、第四次以上时则无法接收所输入的数值，形成只能输入后却不能修改曲目编号的缺点。因此，如果可一直接受输入按键，而最后一次输入为个位数，前一次为十位数，如此反复地将保留最后两次所

输入的曲目编号，是否更弹性、更合理？选曲更方便呢？

2. 参数的意义说明与使用时机

LastBuffer：

- 数据类型：占用一个字节的无符号字符类型(unsigned char)。
- 内存类型：变量地址介于 0X00～0Xff 之间，使用间接寻址模式(idata)。
- 清除：在初始化模块前先清除为 0X00，表示十位数目前为 0。
- 写入：当按下键输出至显示器之后，则应将目前所按下的数值写入此变量，以作为下一次输入数值的十位数。
- 计算：每一次的按键，应将十位数先乘以数值 10 后再加上此按键数值，才是目前的曲目编号，也就是保留最后两次的输入值。之前一次为十位数存入 LastBuffer 变量内，最后一次（即目前所按下的数值）为个位数存入 KeyBuffer 变量，这样，才能达到可连续输入曲目编号的目的。

3. 软件的解决方法

（1）input.c：即使持续按着键也只动作一次的功能，是利用 FgKeyFlag 标志先行判断是否为“0”才能执行的方法，否则，虽然按下数字却会同时显示在十位数和个位数而失去意义。每当按下键后必须将前一次的按键值作为十位数，而目前所得到的数值当作是个位数，而先行运算后就是曲目编号并输出至显示器上。最重要的是，要将目前按键值存入至十位数变量内，即 LastBuffer 变量，程序如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"

/*******************************/
void InputHandler(void)
{
   //sel play no#
   InputKey( );
```

```
    if ( KeyBuffer != KeyData )
    {
       KeyBuffer = KeyData;
       ScanKeyCounter = 10;
    }
    else if( ScanKeyCounter != 0 )
    {
       ScanKeyCounter--;
       DelayX1ms(1);
    }
    else if ( KeyBuffer != NO_KEY )
    {
       if ( FgKeyFlag==0 )
       {
          FgKeyFlag = 1;

          Beep(1,17,10);
          MusicNumber=LastBuffer*10+KeyBuffer;
          Disp7Seg(MusicNumber);
          LastBuffer=KeyBuffer;
       }
    }
    else
    {
       KeyBuffer = NO_KEY;
       FgKeyFlag = 0;
    }
}

/*****************************/
void InputKey(void)
{
    Byte  keytmp;

    KeyData  = NO_KEY;        //NO_KEY=0

    XFR_ADC=0x81;             //adc channel=1
    DelayX1ms(2);
    keytmp = XFR_ADC & 0x3f;     //6 bit  2^6=0x3f
    if ( keytmp < (12+4) )       KeyData = NUMBER1_KEY;
    else if ( keytmp < (23+4) )      KeyData = NUMBER2_KEY;
    else if ( keytmp < (34+4) )      KeyData = NUMBER3_KEY;
    else if ( keytmp < (44+4) )      KeyData = NUMBER4_KEY;
    else if ( keytmp < (50+4) )      KeyData = NUMBER5_KEY;
```

```
    XFR_ADC=0x82;                    //adc channel=2
    DelayX1ms(2);
    keytmp = XFR_ADC & 0x3f;
    if ( keytmp < (12+4) )           KeyData = NUMBER6_KEY;
    else if ( keytmp < (23+4) )          KeyData = NUMBER7_KEY;
    else if ( keytmp < (34+4) )          KeyData = NUMBER8_KEY;
    else if ( keytmp < (44+4) )          KeyData = NUMBER9_KEY;
    else if ( keytmp < (50+4) )          KeyData = NUMBER0_KEY;

    XFR_ADC=0x00;                    //disable adc
}

/*****************************/
void InputStart(void)
{
    if ( START_IO==0 )
      KeyData=START_KEY;
    else
      KeyData=NO_KEY;
}
```

其中 InputKey( )函数为利用 ADC 的方式来侦测数字 0～9 是否按下，请参考前面章节。

（2）7Segled.c：显示器的输出函数就不必以个位数或十位数来分别处理了，个位数及十位数要一起处理才行，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "7segled.h"

/*****************************/
void Disp7Seg(Byte value)
{
    Byte number2,number1;

    number2=value/10;                //十位数
    number1=(value-=number2*10);     //个位数
```

```
    _7SEGDISP_PORT1=(number2<<4)+number1;
}
```

汇编程序如下：

1. input.a51

```
$NOMOD51

NAME    INPUT

$INCLUDE (mtv212m.inc)

?PR?InputHandler?INPUT      SEGMENT CODE
?PR?InputKey?INPUT          SEGMENT CODE
?DT?InputKey?INPUT          SEGMENT DATA OVERLAYABLE
?PR?InputStart?INPUT SEGMENT CODE
    EXTRN   BIT (FgKeyFlag)
    EXTRN   IDATA (KeyData)
    EXTRN   IDATA (KeyBuffer)
    EXTRN   IDATA (LastBuffer)
    EXTRN   IDATA (ScanKeyCounter)
    EXTRN   IDATA (MusicNumber)
    EXTRN   XDATA (XFR_ADC)
    EXTRN   CODE (_DelayX1ms)
    EXTRN   CODE (_Beep)
    EXTRN   CODE (_Disp7Seg)
    PUBLIC  InputStart
    PUBLIC  InputKey
    PUBLIC  InputHandler

    RSEG  ?DT?InputKey?INPUT
?InputKey?BYTE:
    keytmp?140:   DS   1

    RSEG  ?PR?InputHandler?INPUT
    USING   0
InputHandler:
    LCALL   InputKey
    MOV     R0,#KeyData
    MOV     A,@R0
    MOV     R7,A
    MOV     R0,#KeyBuffer
    XRL     A,@R0
    JZ      ?C0001
```

```
    MOV     A,R7
    MOV     @R0,A
    MOV     R0,#ScanKeyCounter
    MOV     @R0,#0AH
    RET
?C0001:
    MOV     R0,#ScanKeyCounter
    MOV     A,@R0
    JZ      ?C0003
    DEC     @R0
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    RET
?C0003:
    MOV     R0,#KeyBuffer
    MOV     A,@R0
    CPL     A
    JZ      ?C0005
    JB      FgKeyFlag,?C0008
    SETB    FgKeyFlag
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep
    MOV     R0,#LastBuffer
    MOV     A,@R0
    MOV     B,#0AH
    MUL     AB
    MOV     R0,#KeyBuffer
    ADD     A,@R0
    MOV     R7,A
    MOV     R0,#MusicNumber
    MOV     @R0,A
    LCALL   _Disp7Seg
    MOV     R0,#KeyBuffer
    MOV     A,@R0
    MOV     R0,#LastBuffer
    MOV     @R0,A
    RET
?C0005:
    MOV     R0,#KeyBuffer
    MOV     @R0,#0FFH
    CLR     FgKeyFlag
```

```
?C0008:
    RET

    RSEG  ?PR?InputKey?INPUT
    USING   0
InputKey:
    MOV     R0,#KeyData
    MOV     @R0,#0FFH
    MOV     R0,#LOW (XFR_ADC)
    MOV     A,#081H
    MOVX    @R0,A
    MOV     R7,#02H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#LOW (XFR_ADC)
    MOVX    A,@R0
    ANL     A,#03FH
    MOV     keytmp?140,A
    CLR     C
    SUBB    A,#010H
    JNC     ?C0009
    MOV     R0,#KeyData
    MOV     @R0,#01H
    SJMP    ?C0010
?C0009:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#01BH
    JNC     ?C0011
    MOV     R0,#KeyData
    MOV     @R0,#02H
    SJMP    ?C0010
?C0011:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#026H
    JNC     ?C0013
    MOV     R0,#KeyData
    MOV     @R0,#03H
    SJMP    ?C0010
?C0013:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#030H
    JNC     ?C0015
```

```
    MOV     R0,#KeyData
    MOV     @R0,#04H
    SJMP    ?C0010
?C0015:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#036H
    JNC     ?C0010
    MOV     R0,#KeyData
    MOV     @R0,#05H
?C0C10:
    MOV     R0,#LOW (XFR_ADC)
    MOV     A,#082H
    MOVX    @R0,A
    MOV     R7,#02H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#LOW (XFR_ADC)
    MOVX    A,@R0
    ANL     A,#03FH
    MOV     keytmp?140,A
    CLR     C
    SUBB    A,#010H
    JNC     ?C0018
    MOV     R0,#KeyData
    MOV     @R0,#06H
    SJMP    ?C0019
?C0018:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#01BH
    JNC     ?C0020
    MOV     R0,#KeyData
    MOV     @R0,#07H
    SJMP    ?C0019
?C0020:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#026H
    JNC     ?C0022
    MOV     R0,#KeyData
    MOV     @R0,#08H
    SJMP    ?C0019
?C0022:
    MOV     A,keytmp?140
```

```
    CLR     C
    SUBB    A,#030H
    JNC     ?C0024
    MOV     R0,#KeyData
    MOV     @R0,#09H
    SJMP    ?C0019
?C0024:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#036H
    JNC     ?C0019
    CLR     A
    MOV     R0,#KeyData
    MOV     @R0,A
?C0019:
    CLR     A
    MOV     R0,#LOW (XFR_ADC)
    MOVX    @R0,A
    RET

    RSEG  ?PR?InputStart?INPUT
    USING   0
InputStart:
    JB      P3_4,?C0028
    MOV     R0,#KeyData
    MOV     @R0,#01H
    RET
?C0028:
    MOV     R0,#KeyData
    MOV     @R0,#0FFH
?C0030:
    RET

    END
```

2. 7segled.a51

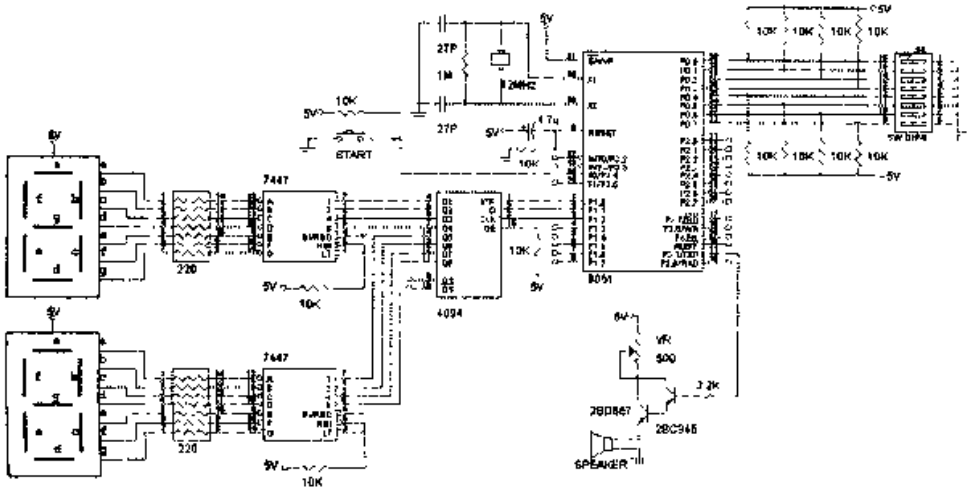
```
$NOMOD51

NAME    _7SEGLED

$INCLUDE (mtv212m.inc)

?PR?_Disp7Seg?7SEGLED       SEGMENT CODE
    PUBLIC  _Disp7Seg
```

```
    RSEG  ?PR?_Disp7Seg?7SEGLED
    USING   0
_Disp7Seg:
    MOV     A,R7
    MOV     B,#0AH
    DIV     AB
    MOV     R6,A
    MOV     B,#0AH
    MUL     AB
    MOV     R5,A
    CLR     C
    MOV     A,R7
    SUBB    A,R5
    MOV     R5,A
    MOV     R7,A
    MOV     A,R6
    SWAP    A
    ANL     A,#0F0H
    ADD     A,R5
    MOV     P1,A
    RET

    END
```

10-4　4094 显示器 X2

目的: 利用串行移位缓存器 IC4094 作为 I/O 端口的扩充，而控制显示器的应用。

电路图:



原理： 以 I/O 端口输出 BCD 码至 IC7447 再驱动两个显示器，也需要八支 I/O 引脚，如果系统较大致使 CPU 的 I/O 引脚不敷使用，则可将 BCD 码由 IC7447 作为输出，而其 BCD 码的数据可通过串行数据由 I/O P1.1，P1.2 来传输，当将数据传输至 IC7447 闩锁内，再送 STROBE "0" → "1" → "0" 的脉冲即可令数据输出至 IC7447 Q1～Q8 的引脚上，而 Q1～Q4 作为个位数的 BCD 码，Q5～Q8 作为十位数的 BCD 码，则可令七段显示器显示出目前所输入的曲目编号，却只有使用三支 I/O 引脚。如果欲扩充八个输出，则也仅需要一支 I/O 引脚控制另外一颗 IC4094 的 STROBE 信号而已。这样，可节省很多的 I/O 脚，其程序只是稍为复杂而已，即可具有更多的控制和更多的功能。

10-4-1 DIP 选曲：1～99

模块化： 将推动 IC7447 的 BCD 码原本由 P1 端口直接输出，但为节省 I/O 端口而改用 IC4094 的输出 Q1～Q8 作为 BCD 码的输入，因而可将 IC4094 串行数据的输出控制独立为一模块，称之为 ic4094.c 模块，相对应的包括文件为 ic4094.h 则新增的模块其功能如下：

- ic4094.c：串行数据的 clock，data 通信协议控制以及 strobe 信号的控制。
- ic4094.h：ic4094.c 模块的函数外部声明以及分别将 STROBE、DATA、CLOCK 信号定义为 I/O P1.0、P1.1 及 P1.2。

软件构建的思维与解决方法

由 IC4094 的输出作为 IC7447 的 BCD 码输入，所以必须将所输入的选曲编号通过串行的通信协议传输至 ic4094 并使其输出至 Q1～Q8 的引脚上，其原理只是将 8 位数据以一个位一个位传入即可，最后将数据转换成 8 位并输出至 Q1～Q8，然后只需再送 "0" → "1" → "0" 的脉冲至 STROBE 引脚而已，现分述如下：

（1）initial.c：开机后要将两个七段显示器输出为 "00"，其程序如下：

```
/*****************************/
void InitialCpuIO(void)
{
   //display "00"
   Output4094(0x00);
   SPEAKER = 0;
}
```

（2）7Segled.c：七段显示器的输出控制，除了将个位数、十位数分离出来后并

需将十位数移至 bit4～bit7，因为 ic4094 的 Q1～Q4 控制个位数的显示器，而 Q5～Q8 则控制十位数的显示器，最后再调用 OutPut4094( )函数，将此曲目编号的数值由 ic4094 的 Q1～Q8 输出，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"
#include  "ic4094.h"

/*****************************/
void Disp7Seg(Byte value)
{
    Byte number2,number1;

    number2=value/10;                  //十位数
    number1=(value-=number2*10);       //个位数

    Output4094((number2<<4)+number1);
}
```

（3）ic4094.c：将曲目编号通过 ic4094 的 clk、data 的串行传输及 Strobe 信号的并排输出至 Q1～Q8 引脚，可分为下列函数：

- Output4094( )函数：将数据传输至闩锁内以及并排输出至 Q1～Q8 的引脚
- DataTx8bit( )函数：将数据以串行通信协议传输至闩锁内，其传输以高位先传送，即 data 先设定 bit7 的内容，再送一个“0”→“1”→“0”的脉冲至 clk 引脚，然后再将 bit6 的内容输出至 data 线上，并输出脉冲至 clk 线上，这样，反复至 bit0 传送完毕为止，总共传送 8 次。
- Strobe( )函数：即输出“0”→“1”→“0”的脉冲至 STR 引脚上，则在闩锁内的数据将立即并排输出至 Q1～Q8 的引脚上。
- clkDelay( )函数：为了符合 clk、data 以及 strobe 脉冲的时序而作的延迟函数，以 for 语句来执行空语句。

此模块的程序如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"
#include  "ic4094.h"

/*******************************/
void Output4094(Byte value)
{
   DataTx8bit(value);
   Strobe( );
}

void DataTx8bit(Byte value)
{
   Byte  i;

   for(i=0; i<8; i++)
   {
      if ( value & 0x80 )
         DATA_PIN = 1;        //P1.1=1
      else
         DATA_PIN = 0;
      value <<= 1;

      ClkDelay( );

      CLK_PIN = 1;            //clk="0"->"1"->"0"
      ClkDelay( );

      CLK_PIN = 0;            //P1.2=0
      ClkDelay( );
   }
}

void Strobe(void)
{
```

```
    STROBE_PIN = 0;          //P1.0="0"->"1"->"0"
    ClkDelay( );

    STROBE_PIN = 1;
    ClkDelay( );

    STROBE_PIN = 0;
    ClkDelay( );
}

//50uS
void ClkDelay(void)
{
    Byte i;

    for(i=0; i<6; i++)
      ;
}
```

（4）ic4094.h：ic4094.c 模块的包括文件，包含定义 SRTOBE_PIN 为 I/O P1.0，DATA_PIN 为 I/O P1.1，CLK_PIN 为 I/O P1.2 以及 ic4094.c 模块下函数的外部声明，其内容如下：

```
#ifndef     _IC4094_H
#define     _IC4094_H

#define  STROBE_PIN     P1_0
#define  DATA_PIN       P1_1
#define  CLK_PIN    P1_2

extern void  Output4094(Byte value);
extern void  DataTx8bit(Byte value);
extern void  Strobe(void);
extern void  ClkDelay(void);

#endif
```

汇编程序如下：

1. 7segled.a51

```
$NOMOD51

NAME    _7SEGLED

$INCLUDE (mtv212m.inc)
```

```
?PR?_Disp7Seg?7SEGLED   SEGMENT CODE
    EXTRN   CODE (_Output4094)
    PUBLIC  _Disp7Seg

    RSEG  ?PR?_Disp7Seg?7SEGLED
    USING   0
_Disp7Seg:
    MOV     A,R7
    MOV     B,#0AH
    DIV     AB
    MOV     R6,A
    MOV     B,#0AH
    MUL     AB
    MOV     R5,A
    CLR     C
    MOV     A,R7
    SUBB    A,R5
    MOV     R5,A
    MOV     R7,A
    MOV     A,R6
    SWAP    A
    ANL     A,#0F0H
    ADD     A,R5
    MOV     R7,A
    LCALL   _Output4094
    RET

    END
```

2. ic4094.a51

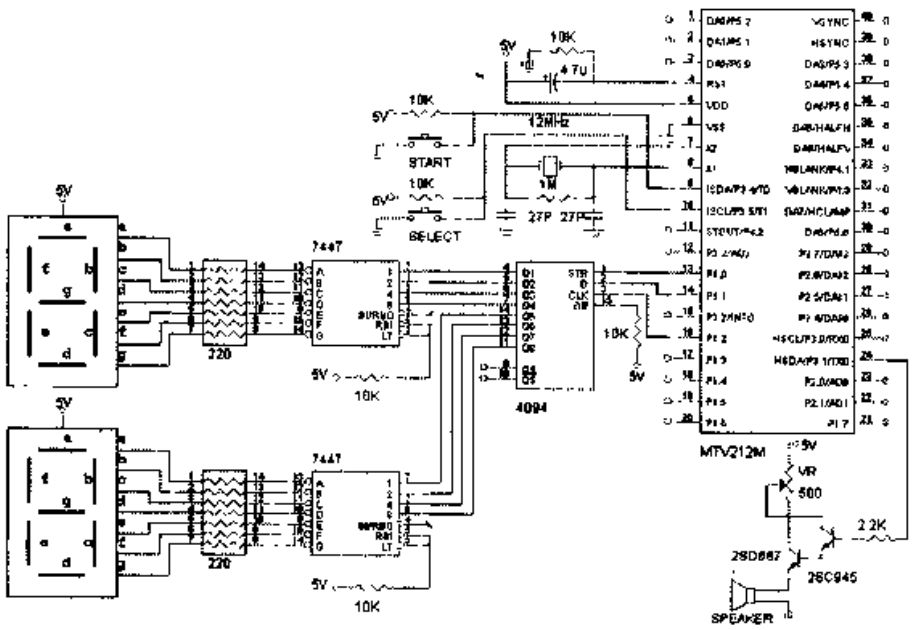
```
$NOMOD51

NAME    IC4094

$INCLUDE (mtv212m.inc)

?PR?_Output4094?IC4094      SEGMENT CODE
?PR?_DataTx8bit?IC4094      SEGMENT CODE
?DT?_DataTx8bit?IC4094      SEGMENT DATA OVERLAYABLE
?PR?Strobe?IC4094           SEGMENT CODE
?PR?ClkDelay?IC4094         SEGMENT CODE
    PUBLIC  ClkDelay
    PUBLIC  Strobe
```

```
    PUBLIC  _DataTx8bit
    PUBLIC  _Output4094

    RSEG  ?DT?_DataTx8bit?IC4094
?_DataTx8bit?BYTE:
     value?141:   DS   1
    ORG  1
        i?142:   DS   1

    RSEG  ?PR?_Output4094?IC4094
    USING   0
_Output4094:
    LCALL   _DataTx8bit
    LCALL   Strobe
    RET

    RSEG  ?PR?_DataTx8bit?IC4094
    USING   0
_DataTx8bit:
    MOV     value?141,R7
    CLR     A
    MOV     i?142,A
?C0002:
    MOV     A,value?141
    JNB     ACC.7,?C0005
    SETB    P1_1
    SJMP    ?C0006
?C0005:
    CLR     P1_1
?C0006:
    MOV     A,value?141
    ADD     A,ACC
    MOV     value?141,A
    LCALL   ClkDelay
    SETB    P1_2
    LCALL   ClkDelay
    CLR     P1_2
    LCALL   ClkDelay
    INC     i?142
    MOV     A,i?142
    CLR     C
    SUBB    A,#08H
    JC      ?C0002
?C0007:
    RET
```

```
    RSEG  ?PR?Strobe?IC4094
    USING   0
Strobe:
    CLR     P1_0
    LCALL   ClkDelay
    SETB    P1_0
    LCALL   ClkDelay
    CLR     P1_0
    LCALL   ClkDelay
    RET

    RSEG  ?PR?ClkDelay?IC4094
    USING   0
ClkDelay:
    CLR     A
    MOV     R7,A
?C0009:
    INC     R7
    CJNE    R7,#06H,?C0009
?C0012:
    RET

    END
```

10-4-2 TACT 选曲：1～99

电路图：



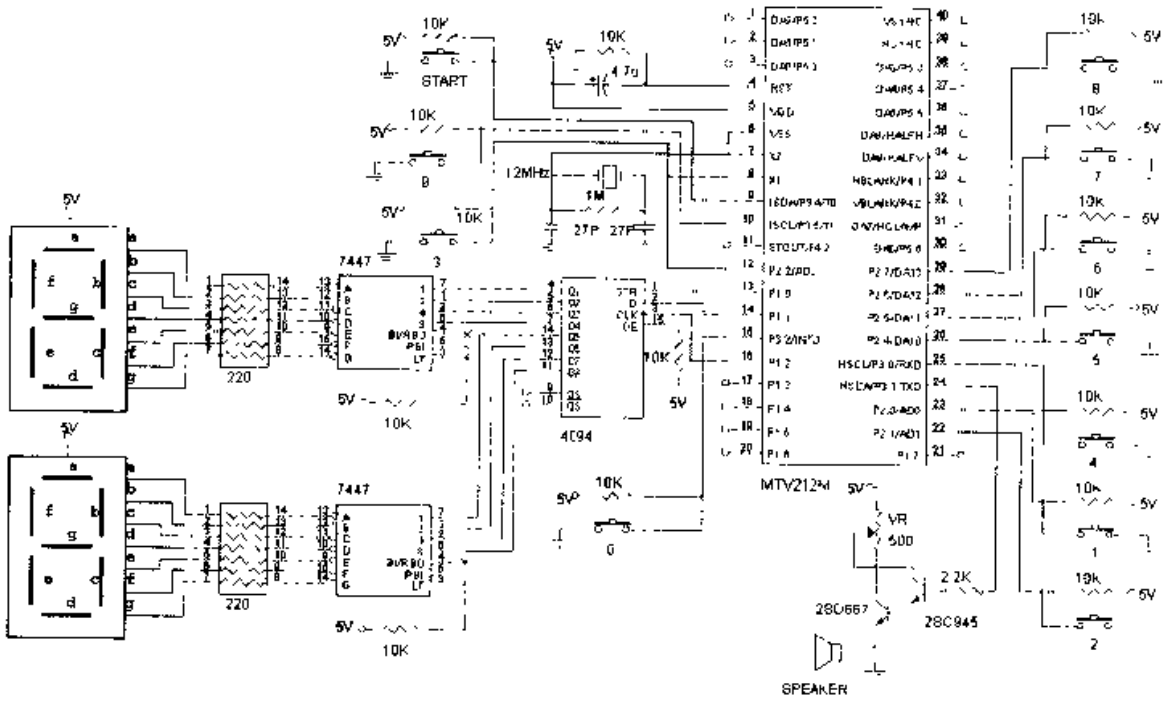
请参考 10-3 节的 TACT 选曲中的 input.c 模块，其控制显示器的输出函数如下：

```
/*****************************/
void Disp7Seg(Byte value)
{
   Byte number2,number1;

   number2=value/10;                    //十位数
   number1=(value-=number2*10);         //个位数

   Output4094((number2<<4)+number1);
}
```

完整的程序设计及开发环境的项目设定。

10-4-3　I/O 选曲：1～99

电路图：



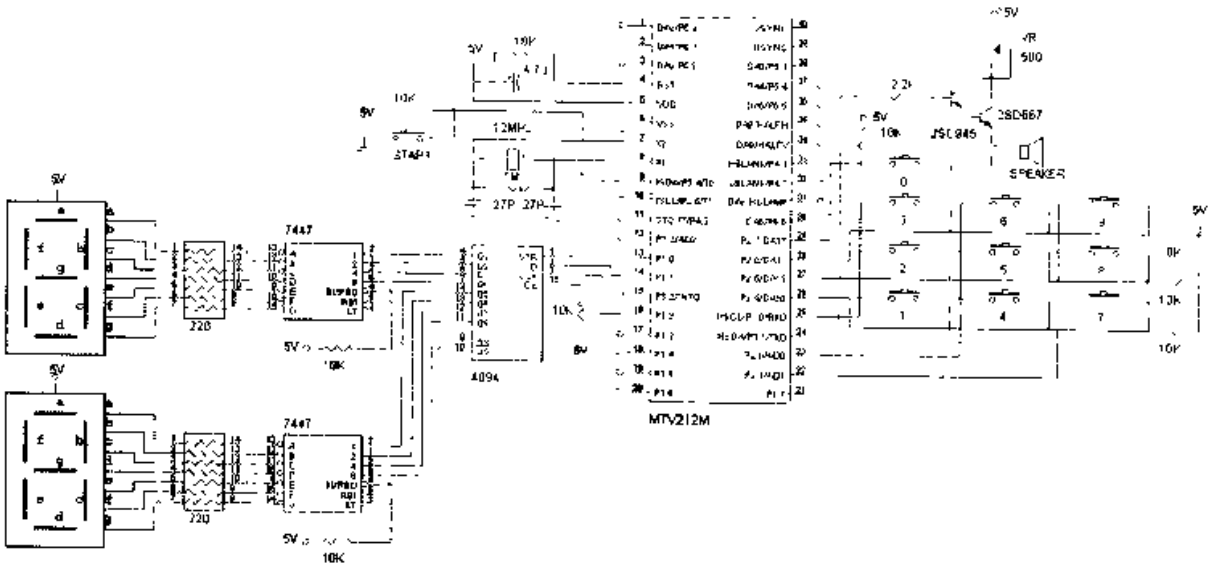
请参考 10-3 节的 I/O 选曲中的 input.c 模块，其控制显示器的输出函数如下：

```
/*****************************/
void Disp7Seg1(Byte value)           //  位数
{
   Output4094(0x00+value);
}
```

```
/*****************************/
void Disp7Seg2(Byte value)          //二位数
{
    Byte number2,number1;

    number2=value/10;               //十位数
    number1=(value-=number2*10);    //个位数

    Output4094((number2<<4)+number1);
}
```

10-4-4 SCAN 选曲：1～99

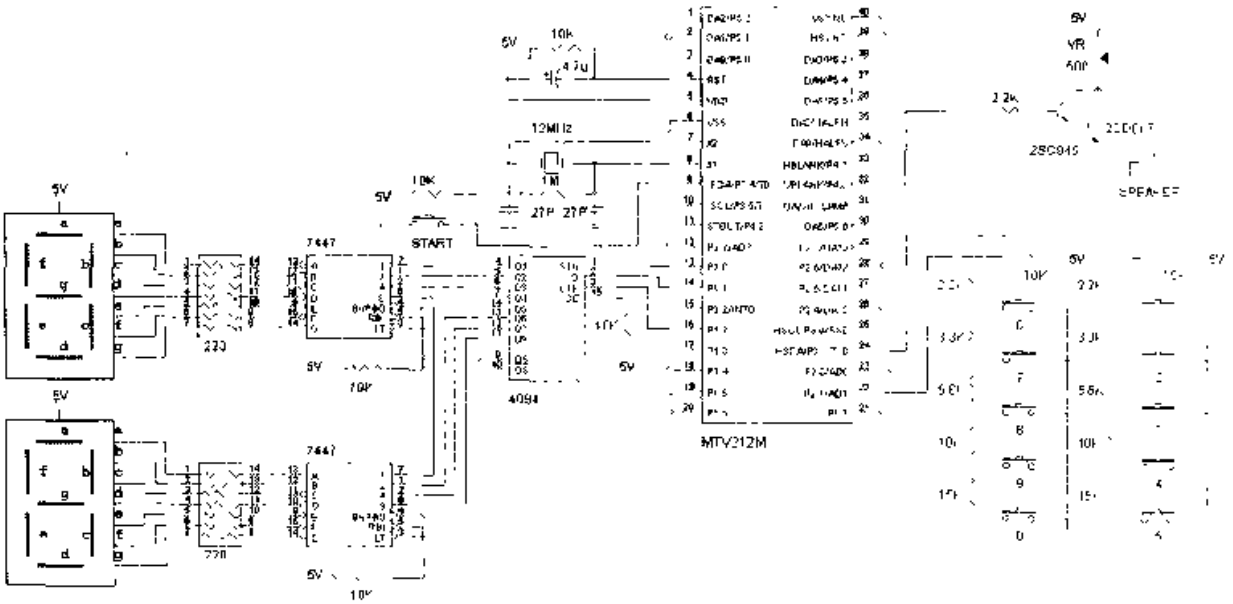
电路图：



请参考 10-3 节的 SCAN 选曲中的 input.c 模块，其控制显示器的输出函数，如 I/O 选曲：1～99 所示。

10-4-5 ADC 选曲：1～99

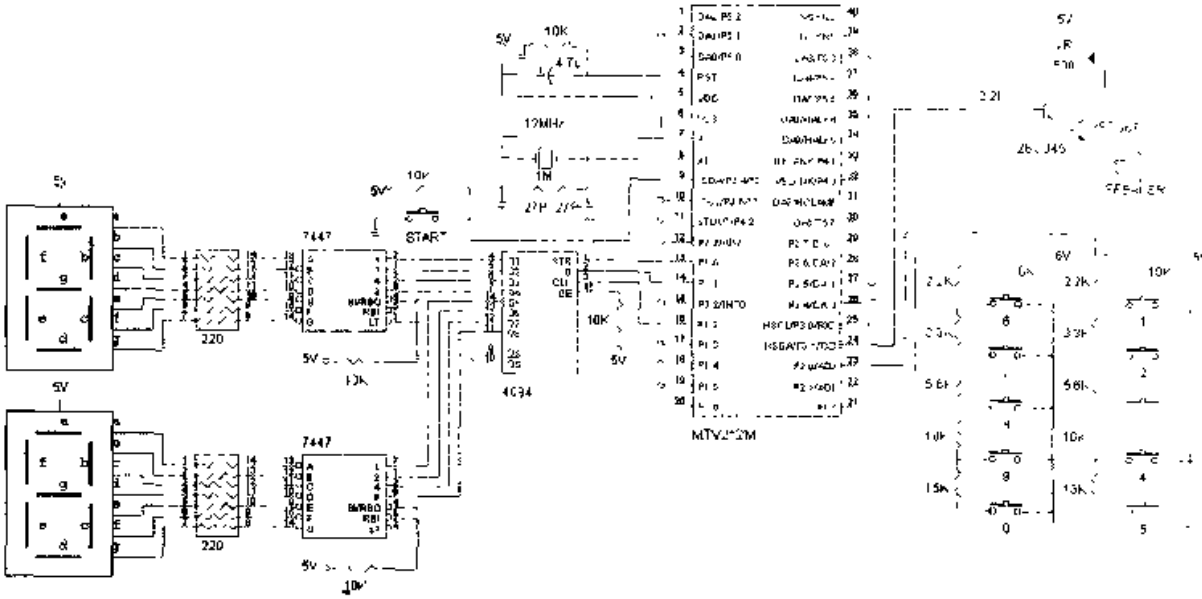
电路图：



请参考 10-3 节的 ADC 选曲中的 input.c 模块，其控制显示器的输出函数，如 I/O 选曲：1～99 所示。

10-4-6　连续按键选曲：1～99

电路图：



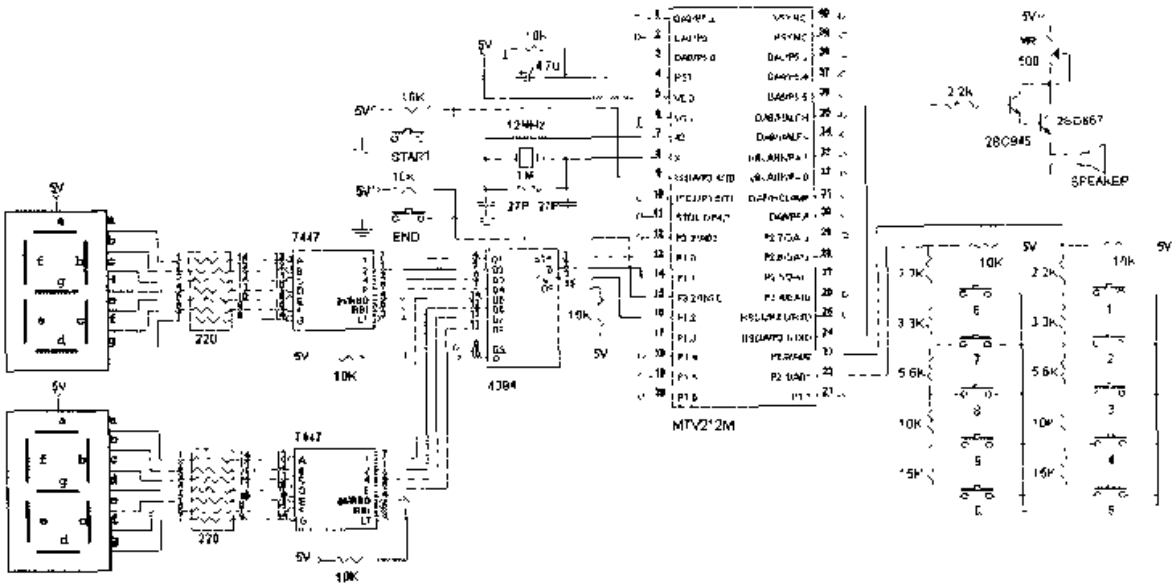
请参考 10-3 节的连续按键选曲中的 input.c 模块，其控制显示器的输出函数，如 TACT 选曲：1～99 所示。

第 11 章　功能模式

11-1　单曲循环

目的：将所选取的曲目编号，可以一直演奏下去，直到按下“停止键”。

电路图：



原理：按键输入可以连续操作，但不管如何只会保留最后两次的数值输入作为曲目编号的设定，在程序中可利用 while(1)反复循环不断地执行音乐演奏。在演奏中只有利用中断的停止键输入才能中止此演奏，因为即使在演奏音乐的函数里，其音符也是利用 timer1 作方波频率的输出而驱动喇叭发出声音。然而，以外部中断作为最高优先权的处理，外部中断引脚(into)一旦接收到低电频的脉冲，则立即执行外部中断程序。在中断程序中只要设定一个标志变量，并在音乐演奏函数及主程序中去侦测此标志变量是否为“1”，如果是，则跳离反复循环而达到停止演奏的功能。

11-1-1　参数的意义说明与使用时机

FgEndMusic：按下停止键的音乐演奏中止标志。

- 数据类型：占用一个位的数据类型(bit)。
- 内存类型：变量地址介于 0X20～0X2f 的其中一个位，使用位寻址模式。

- 清除“0”：

（1）在初始化模块事先将其清除。

（2）当按下起始演奏键，必须将其清除，以确定正在演奏中的状态。

- 设定“1”：按下停止演奏键必须设定，以表示要停止演奏
- 判断：

（1）在每一个音乐模块中必须判断是否为“1”，如果为“1”，表示要停止演奏，则不再取出下一个音符的音长和音调数值，并立即退出音乐演奏函数而回到主程序。

（2）于主程序的反复循环也需要判断，当此标志是“1”，则也要退出反复循环不再继续演奏，而继续等待下一次的起始键。

11-1-2　软件构建的思维与解决方法

1. maic.c：当按下起始键，则应将停止演奏标志 FgEndMusic 清除，以表示正在演奏。而为了不断地重复演奏曲目，可以用 while(1)指令来达到目的，只要停止演奏的侦测，其中 if 指令就是。当条件式成立则执行 break 指令，此指令即可跳出 while(1)所构成的反复循环，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"
#include  "ic4094.h"

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
     //sel play no#
```

```
        InputHandler( );

        //detect start key
        InputStart( );
        if ( KeyData == START_KEY )
        {
           FgEndMusic =0;      //int0  :END_KEY

           while( 1 )
           {
              Music(MusicNumber);
              if (FgEndMusic==1) break;
              DelayX10ms(100);
           }
        }
     }
  }
```

2. timer.c：音符的频率一样是利用 timer1 来输出方波，通过驱动喇叭而发出声音，而外部中断程序其目的只有一样，就是设定停止演奏标志，即令 FgEndMusic=1 而已，如果不借着一个停止键利用外部中断的技巧，一旦演奏音乐则无法使其停止，除非关机，因而，此标志的思维、观念、技巧就变得很重要了，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"
#include  "ic4094.h"

/*****************************/
//INT0 interrupt:
//INT0 Pin:"1"->"0"->"1" pulse
/*****************************/
void  INT0ISR ( void ) interrupt 0 using 3
{
   EX0 = 0;             //disable int0
```

```
    FgEndMusic=1;    //end play music

    EX0 = 1;              //enable next interrupt
    IE0 = 0;              //set TCON.1 flag
}

/*******************************/
//timer1,execute service route
/*******************************/
void  Timer1ISR ( void ) interrupt 3 using 2
{
    Word temp;

    if ( SoundLongCount != 0 )
      SoundLongCount--;

    temp = 65536 - Period;
    SPEAKER = !SPEAKER;

    TL1 = temp & 0xff;
    TH1 = temp >> 8;
    TF1 = 0;
}
```

汇编程序如下：

1. main.a51

```
$NOMOD51

NAME    MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN       SEGMENT CODE
    EXTRN   BIT (FgEndMusic)
    EXTRN   IDATA (KeyData)
    EXTRN   IDATA (MusicNumber)
    EXTRN   CODE (PowerOnInit)
    EXTRN   CODE (_DelayX10ms)
    EXTRN   CODE (_Music)
    EXTRN   CODE (InputHandler)
    EXTRN   CODE (InputStart)
    EXTRN   CODE (?C_STARTUP)
    PUBLIC  Main
```

```
    RSEG  ?PR?Main?MAIN
    USING   0
Main:
    LCALL   PowerOnInit
?C0001:
    LCALL   InputHandler
    LCALL   InputStart
    MOV     R0,#KeyData
    MOV     A,@R0
    CJNE    A,#01H,?C0001
    CLR     FgEndMusic
?C0004:
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Music
    JB      FgEndMusic,?C0001
?C0006:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0004
    RET

    END
```

2. timer.a51

```
$NOMOD51

NAME    TIMER

$INCLUDE (mtv212m.inc)

?PR?INT0ISR?TIMER   SEGMENT CODE
?PR?Timer1ISR?TIMER     SEGMENT CODE
    EXTRN   BIT (FgEndMusic)
    EXTRN   IDATA (Period)
    EXTRN   IDATA (SoundLongCount)
    PUBLIC  Timer1ISR
    PUBLIC  INT0ISR

CSEG    AT  00003H
    LJMP    INT0ISR
```

```
    RSEG  ?PR?INT0ISR?TIMER
    USING   3
INT0ISR:
    CLR     EX0
    SETB    FgEndMusic
    SETB    EX0
    CLR     IE0
    RETI

CSEG    AT  0001BH
    LJMP    Timer1ISR

    RSEG  ?PR?Timer1ISR?TIMER
    USING   2
Timer1ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     PSW,#010H
    MOV     R0,#SoundLongCount+01H
    MOV     A,@R0
    DEC     R0
    ORL     A,@R0
    JZ      ?C0002
    INC     R0
    MOV     A,@R0
    DEC     @R0
    JNZ     ?C0002
    DEC     R0
    DEC     @R0
?C0002:
    MOV     R0,#Period
    MOV     A,@R0
    MOV     R6,A
    INC     R0
    MOV     A,@R0
    MOV     R7,A
    CLR     C
    CLR     A
    SUBB    A,R7
    MOV     R7,A
    CLR     A
    SUBB    A,R6
    MOV     R6,A
    CPL     P3_1
```

```
XCH     A,R5
MOV     A,R7
XCH     A,R5
MOV     A,R5
MOV     TL1,A
MOV     A,R6
MOV     TH1,A
CLR     TF1
POP     PSW
POP     ACC
RETI

END
```

11-2 顺序播放

目的：将所选取的曲目编号演奏完毕后，紧接着下一曲目继续演奏，当演奏至 31 首曲目结束后，以第 1 首曲目开始演奏，如此反复。

软件构建的思维与解决方法

侦测起始演奏键在主程序中判断，其程序结构较为流畅，每当演奏完一首曲目后，则将曲目编号加 1 后继续演奏，但还需要一个 while(1)循环，应将曲目编号运算后再进行演奏。程序内容写在主程序内会较简捷，因为重点为曲目编号的变量，在每演奏完毕需加 1，而演奏至最后一首需重头（即数值 0X01）开始演奏，而在内层的反复循环一样要判断停止演奏标志是否为“1”，在按下起始键后则此标志必须清除，以保证按下停止键后才停止演奏，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"
```

```
#include  "ic4094.h"

/*****************************/
void Main( void )
{
    PowerOnInit();

    while( 1 )
    {
       //sel play no#
       InputHandler( );

       //detect start key
       InputStart( );
       if ( KeyData == START_KEY )
       {
          FgEndMusic =0;      //int0  :END_KEY

          while( 1 )
          {
             Music(MusicNumber);
             if (FgEndMusic==1) break;

             MusicNumber++;
             if ( MusicNumber > LASTMUSIC )
                MusicNumber=1;
             DelayX10ms(100);
          }
       }
    }
}
```

汇编程序如下：

1. main.a51

```
$NOMOD51

NAME    MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN       SEGMENT CODE
    EXTRN   BIT (FgEndMusic)
    EXTRN   IDATA (KeyData)
```

```
    EXTRN   IDATA (MusicNumber)
    EXTRN   CODE (PowerOnInit)
    EXTRN   CODE (_DelayX10ms)
    EXTRN   CODE (_Music)
    EXTRN   CODE (InputHandler)
    EXTRN   CODE (InputStart)
    EXTRN   CODE (?C_STARTUP)
    PUBLIC  Main

    RSEG  ?PR?Main?MAIN
    USING   0
Main:
    LCALL   PowerOnInit
?C0001:
    LCALL   InputHandler
    LCALL   InputStart
    MOV     R0,#KeyData
    MOV     A,@R0
    CJNE    A,#01H,?C0001
    CLR     FgEndMusic
?C0004:
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Music
    JB      FgEndMusic,?C0001
?C0006:
    MOV     R0,#MusicNumber
    INC     @R0
    MOV     A,@R0
    SETB    C
    SUBB    A,#01FH
    JC      ?C0007
    MOV     @R0,#01H
?C0007:
    MOV     R7,#064H
    MOV     R6,#C0H
    LCALL   _DelayX10ms
    SJMP    ?C0004
    RET

    END
```

11-3　随机选曲

目的：当曲目演奏完毕后，可由微电脑随意地选曲并自动演奏。

11-3-1　rand( )函数

软件构建的思维与解决方法

在 keil_C 的函数库里有一个名称为 stdlib.h 的包括文件，此包括文件有一个 rand( )函数，为随机函数，当调用此随机函数将回复 0～32767 的数值，因此，只要在曲目演奏完毕后再调用此随机函数，即又可产生一个介于 0～32767 的数值。然而须注意的是，产生的随机值应介于 1～31 首的曲目（因为最后的曲目编号 LASTMUSIC=31 所致），而且每次产生的随机值不可以和上一次一样，现分述如下。

（1）main.c：在 main( )函数之前应先编写#include <stdlib.h>的指令，即将 stdlib 的函数库包括进来，才可调用 rand( )函数，否则，compiler 将出现错误。rand( )函数的返回值为 0～32767，因此，将返回值去除以 100 的余数，即变成 0～99，再将此值判断是否介于 1～31 之间而且不能和上一次的随机值重复，否则，不断地再调用随机值，直到上述两个条件都成立，才将此随机值当作曲目编号而进行演奏。演奏完毕继续取随机值，如此反复，直到按下停止键后停止，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"
#include  "ic4094.h"
#include  <stdlib.h>

/*****************************/
void Main( void )
{
   int lastvalue,currentvalue;
```

```
    PowerOnInit();

    while( 1 )
    {
      //sel play no#
      InputHandler( );

      //detect start key
      InputStart( );
      if ( KeyData == START_KEY )
      {
        FgEndMusic =0;          //int0  :END_KEY

        while( 1 )
        {
          currentvalue=rand(); //产生一个随机数
          currentvalue%=100;   //取最后两位数 0~99

          //currentvalue=1~LASTMUSIC
          if ( (currentvalue<=LASTMUSIC)&&(currentvalue>0) )
          {
           //和上次不一样
           if (currentvalue!=lastvalue)
           {
             MusicNumber=currentvalue;
              Music(MusicNumber);
              lastvalue=currentvalue;
              if (FgEndMusic==1) break;
              DelayX10ms(100);
            }
          }
        }
      }
    }
}
```

执行此程序时确实可以产生随机数，看似没问题，功能也正常，可以达到所求，实际上在每次执行时，其所产生的随机数值的顺序、数值都一样，如同取表一样，不是真正的随机选曲。

11-3-2　srand( )函数

软件构建的思维与解决方法

stdlib 函数库另外有一个 srand( )函数，其函数功能为设定种子数值，以便产生一个随机数，也就是当种子不同，rand( )随机函数所返回的数值便不同，因此，如果将 srand( )函数和 rand( )函数同时应用是否就可以在每次执行时都产生不同的随机数值，现说明如下：

（1）main.c：在程序执行完初始化的动作之后，立即执行 srand( )函数，而 srand( )函数的种子数值由 rand( )函数产生，所以种子数值是随机数，紧接着在 while(1)反复循环调用 rand( )函数，并取出 0～99 的数值，但此数值须介于 1～31 曲目，及不同于上一次的随机数才能进行演奏，程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"
#include  "ic4094.h"
#include  <stdlib.h>

/*****************************/
void Main( void )
{
   int lastvalue,currentvalue;

   PowerOnInit();

   srand(rand());                    //random seed

   while( 1 )
   {
     //sel play no#
     InputHandler( );
```

```
        //detect start key
        InputStart( );
        if ( KeyData == START_KEY )
        {
          FgEndMusic =0;          //int0  :END_KEY

          while( 1 )
          {
            currentvalue=rand(); //产生一个随机数
            currentvalue%=100;   //取最后两位数 0~99

            //currentvalue=1~LASTMUSIC
            if ( (currentvalue<=LASTMUSIC)&&(currentvalue>0) )
            {
              //和上次不一样
              if (currentvalue!=lastvalue)
              {
                MusicNumber=currentvalue;
                Music(MusicNumber);
                lastvalue=currentvalue;
                if (FgEndMusic==1) break;
                DelayX10ms(100);
              }
            }

          }
        }
      }
    }
```

在程序一开始即设定一个随机函数的种子数值，确实和上述所产生的顺序、数值都不一样，就如同取另外一份表的数值而已。程序在重新执行时，其顺序和数值又是一样，因为只要一重新执行，其种子数值是由rand( )函数产生，而rand( )函数再重新执行就会产生一样的数值，变成每次执行其种子数值相同，其后的rand( )函数当然就产生一样的顺序、数值了。但是要如何产生随机数呢？不能在每次开机时产生不一样的数值，则就不能算是真正的随机选曲了，各位读者，学习软件设计，除了将功能以程序的方式呈现出来之外，更重要的是学习如何思考！想到了吗，其实也很简单：利用按键的弹跳波作为种子数值，在每次按键时则累加弹跳波的数值，此数值又当成最新的种子数值，当种子数每次都不一样，则rand( )函数的随机数当然就不一样了。

11-3-3　开关弹跳波

软件构建的思维与解决方法

要进行音乐演奏必须按下起始键才能执行，正好可以利用按下此开关来产生弹跳波的个数而作为随机函数的种子数。而为了计算此弹跳波的次数，必须以计数中断来侦测，其开关的硬件线路只能以简单的提升电阻设计，即正常为“1”电位，当按下开关则开始产生弹跳的负脉冲，其负脉冲的个数本身也是未定数，正好符合随机函数，其种子的数值最好也是随机数，现分述如下：

（1）timer.c：当按下起始键则开始累加脉冲数，其变量为 PluseCount，而定时器 0 规划为计数器的模式，每当一有负脉冲则立即产生计数中断，在计数中断函数内则将此 Pluse Count 加 1，如果有八个负脉冲则产生八次中断，于是 PluseCount 的内容变为 8。在下一次又按下起始键又继续累加这次负脉冲的个数，如果这次负脉冲的个数为 5，则 PulseCount 内容形成 8+5=13，此数值 13 又形成最新的种子数，因而可以产生不同的随机数。当按下起始键，则设定 FgStartMusic 标志作为 Main( ) 函数执行演奏的判断根据，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"
#include  "ic4094.h"

/*****************************/
//INT0 interrupt:
//INT0 Pin:"1"->"0"->"1" pulse
/*****************************/
void  INT0ISR ( void ) interrupt 0 using 3
{
   EX0 = 0;              //disable int0

   FgEndMusic  =1;       //end play music
   FgStartMusic=0;
```

```
    EX0 = 1;              //enable next interrupt
    IE0 = 0;              //set TCON.1 flag
}

/*****************************/
//COUNT0 interrupt:
//T0 Pin:"1"->"0"->"1" pulse
/*****************************/
void  Timer0ISR ( void ) interrupt 1 using 3
{
    PulseCount++;

    FgStartMusic=1;

    TL0 = 0xff;
    TH0 = 0xff;
    TF0 = 0;
}

/*****************************/
//timer1,execute service route
/*****************************/
void  Timer1ISR ( void ) interrupt 3 using 2
{
    Word temp;

    if ( SoundLongCount != 0 )
       SoundLongCount--;

    temp = 65536 - Period;
    SPEAKER = !SPEAKER;

    TL1 = temp & 0xff;
    TH1 = temp >> 8;
    TF1 = 0;
}
```

(2)按下起始键则在计数中断函数中会设定 FgStartMusic 标志，当主程序侦测此标志为 1，则表示要开始演奏，在执行演奏的程序处理必须将此标志清除，以便下一次起始键的操作。Delay 0.5 秒的用意是要让计数中断函数确实地计算出弹跳波的次数，并将此次数作为 srand( )函数的种子数，而在后面调用 rand( )函数而能得到随机数，并进行此随机数的判断而进行演奏。由于重新执行程序后，每当按下起始

键，所引发的弹跳波其次数并不固定，而且又是累加此弹跳波的次数作为种子数，因此，可以真正达到随机选曲的功能，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"
#include  "ic4094.h"
#include  <stdlib.h>

/*****************************/
void Main( void )
{
   int lastvalue,currentvalue;

   PowerOnInit();

   while( 1 )
   {
     //sel play no#
     InputHandler( );

     //detect start key
     if (FgStartMusic)
     {
        FgEndMusic  =0;      //int0  :END_KEY

         FgStartMusic=0;
         DelayX10ms(50);
         srand(PulseCount);

        while( 1 )
        {
          currentvalue=rand(); //产生一个随机数
          currentvalue%=100;   //取最后两位数 0~99

          //currentvalue=1~LASTMUSIC
```

```
            if ( (currentvalue<=LASTMUSIC)&&(currentvalue>0) )
            {
               //和上次不一样
               if (currentvalue!=lastvalue)
               {
                  MusicNumber=currentvalue;
                  Music(MusicNumber);
                  lastvalue=currentvalue;
                  if (FgEndMusic==1) break;
                  DelayX10ms(100);
               }
            }
         }
      }
   }
}
```

汇编程序如下：

1. main.a51

```
$NOMOD51

NAME    MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN       SEGMENT CODE
?DT?Main?MAIN       SEGMENT DATA OVERLAYABLE
    EXTRN   BIT (FgEndMusic)
    EXTRN   BIT (FgStartMusic)
    EXTRN   IDATA (PulseCount)
    EXTRN   IDATA (MusicNumber)
    EXTRN   CODE (PowerOnInit)
    EXTRN   CODE (_DelayX10ms)
    EXTRN   CODE (_Music)
    EXTRN   CODE (InputHandler)
    EXTRN   CODE (rand)
    EXTRN   CODE (_srand)
    EXTRN   CODE (?C_STARTUP)
    EXTRN   CODE (?C?SIDIV)
    PUBLIC  Main

    RSEG  ?DT?Main?MAIN
?Main?BYTE:
```

```
  lastvalue?040:   DS   2
currentvalue?041:   DS   2

    RSEG  ?PR?Main?MAIN
    USING   0
Main:
    LCALL   PowerOnInit
?C0001:
    LCALL   InputHandler
    JNB     FgStartMusic,?C0001
    CLR     FgEndMusic
    CLR     FgStartMusic
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    MOV     R0,#PulseCount
    MOV     A,@R0
    MOV     R6,A
    INC     R0
    MOV     A,@R0
    MOV     R7,A
    LCALL   _srand
?C0004:
    LCALL   rand
    MOV     currentvalue?041,R6
    MOV     currentvalue?041+01H,R7
    MOV     R4,#00H
    MOV     R5,#064H
    LCALL   ?C?SIDIV
    MOV     currentvalue?041,R4
    MOV     currentvalue?041+01H,R5
    SETB    C
    MOV     A,currentvalue?041+01H
    SUBB    A,#01FH
    MOV     A,currentvalue?041
    XRL     A,#080H
    SUBB    A,#080H
    JNC     ?C0004
    SETB    C
    MOV     A,currentvalue?041+01H
    SUBB    A,#00H
    MOV     A,currentvalue?041
    XRL     A,#080H
    SUBB    A,#080H
    JC      ?C0004
```

```
    MOV     A,currentvalue?041+01H
    XRL     A,lastvalue?040+01H
    JNZ     ?C0010
    MOV     A,currentvalue?041
    XRL     A,lastvalue?040
?C0010:
    JZ      ?C0004
    MOV     R7,currentvalue?041+01H
    MOV     R0,#MusicNumber
    MOV     A,R7
    MOV     @R0,A
    LCALL   _Music
    MOV     lastvalue?040,currentvalue?041
    MOV     lastvalue?040+01H,currentvalue?041+01H
    JB      FgEndMusic,?C0001
?C0008:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0004
    RET

    END
```

2. timer.a51

```
$NOMOD51

NAME    TIMER

$INCLUDE (mtv212m.inc)

?PR?INT0ISR?TIMER   SEGMENT CODE
?PR?Timer0ISR?TIMER     SEGMENT CODE
?PR?Timer1ISR?TIMER     SEGMENT CODE
    EXTRN   BIT (FgEndMusic)
    EXTRN   BIT (FgStartMusic)
    EXTRN   IDATA (PulseCount)
    EXTRN   IDATA (Period)
    EXTRN   IDATA (SoundLongCount)
    PUBLIC  Timer1ISR
    PUBLIC  Timer0ISR
    PUBLIC  INT0ISR

CSEG    AT  00003H
```

```
    LJMP    INT0ISR

    RSEG  ?PR?INT0ISR?TIMER
    USING   3
INT0ISR:
    CLR     EX0
    SETB    FgEndMusic
    CLR     FgStartMusic
    SETB    EX0
    CLR     IE0
    RETI

CSEG    AT  000BH
    LJMP    Timer0ISR

    RSEG  ?PR?Timer0ISR?TIMER
    USING   3
Timer0ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     PSW,#018H
    MOV     R0,#PulseCount+01H
    INC     @R0
    MOV     A,@R0
    JNZ     ?C0005
    DEC     R0
    INC     @R0
?C0005:
    SETB    FgStartMusic
    MOV     TL0,#0FFH
    MOV     TH0,#0FFH
    CLR     TF0
    POP     PSW
    POP     ACC
    RETI

CSEG    AT  001BH
    LJMP    Timer1ISR

    RSEG  ?PR?Timer1ISR?TIMER
    USING   2
Timer1ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     PSW,#010H
```

```
    MOV     R0,#SoundLongCount+01H
    MOV     A,@R0
    DEC     R0
    ORL     A,@R0
    JZ      ?C0003
    INC     R0
    MOV     A,@R0
    DEC     @R0
    JNZ     ?C0003
    DEC     R0
    DEC     @R0
?C0003:
    MOV     R0,#Period
    MOV     A,@R0
    MOV     R6,A
    INC     R0
    MOV     A,@R0
    MOV     R7,A
    CLR     C
    CLR     A
    SUBB    A,R7
    MOV     R7,A
    CLR     A
    SUBB    A,R6
    MOV     R6,A
    CPL     P3_1
    XCH     A,R5
    MOV     A,R7
    XCH     A,R5
    MOV     A,R5
    MOV     TL1,A
    MOV     A,R6
    MOV     TH1,A
    CLR     TF1
    POP     PSW
    POP     ACC
    RETI

    END
```

11-4 播放简介

目的： 在所输入的曲目开始演奏五秒钟后，继续演奏下一曲目，而所演奏的时

间都是五秒钟，即有试听之意。

软件构建的思维与解决方法

从演奏曲目开始第五秒钟后结束，此时间的计时由 timer0 担任计时工作，因此，模式缓存器(TMOD)必须规划为 timer0 和 timer1。timer1 一直作为音符的频率计时而以产生方波的形式，不再重述，而规划 timer0 为 40ms 就中断一次，当到达了五秒的次数则设定 FgNextMusic 标志，以表示要演奏下一曲目，现分述如下：

（1）initial.c：按键选曲是以 ADC 方式来侦测数字键是否按下，timer1 功能为输出波形，timer0 功能为计时五秒，into 功能为停止演奏，这些都是利用中断方式，因此，应将对应的中断使能，以便能发挥作用，除了将 TMOD 模式缓存器规划为 timer0、timer1 模式外，timer0 的 TH0、TL0 要设定为 40ms，以便每 40ms 就中断一次的，其初始化设定如下：

```
/*****************************/
void InitialCpu( void )
{
   IE  = 0;                  //disable all interrupt
   PSW = 0;                  //bank 0

   IP  = 0x0b;               //hi priority:int0,timer0,timer1

   TMOD= 0x11;               //set timer1,timer0 mode

   TR0 = 0;                  //stop timer0
   TR1 = 0;                  //stop timer1
   IT0 = 1;                  //set int0:falling eage trigger

   //CLOCK_40MS=(65536 - 40000)
   TL0 = CLOCK_40MS & 0xff;
   TH0 = CLOCK_40MS >> 8;

   EX0 = 1;                  //enable int0 interrupt
   ET1 = 1;                  //enable timer1 interrupt
   ET0 = 1;                  //enable timer0 interrupt

   //Initial XFR's and DAC's for MTV212MNXX
   XFR_ADC  =0x80;       //enable ADC,not select ch input
   XFR_WDT  =0x40;       //clear watch dog timer,2s interval

   XFR_PADMOD1  =0x07;  //set adc0,adc1,adc2,i/o:P2.4~P2.7
   XFR_PADMOD2  =0x00;  //set dac0~dac6
   XFR_PADMOD3  =0x00;  //set dac,h,v
```

```
    XFR_OPTION1  =0xc0;  //94k pwm=1,div253=1
    XFR_OPTION2  =0x00;  //7 bit slave address for iic
    XFR_XBANK    =0x00;  //ram bank0

    EA  = 1;             //enable all interrupt gate
}
```

（2）timer.c：定时器 0 每 40ms 就中断一次，并在每次中断就将 MusicLength 加 1，而五秒实际上就等于中断 500/40=125 次，也就是当 MusicLength 变量内容等于 125 时，表示演奏的时间已经到了五秒，即需演奏下一曲目。而在中断函数中应设定 FgNextMusic 标志，以便主程序及音乐演奏函数得以判断而跳离，其中断函数的程序如下：

```
/*****************************/
//timer0,execute service route:播放简介
/*****************************/
void  Timer0ISR ( void ) interrupt 1 using 3
{
    MusicLength++;
    if ( MusicLength == TIME_5SEC )
       FgNextMusic=1;

    TL0 = CLOCK_40MS & 0xff;
    TH0 = CLOCK_40MS >> 8;
    TF0 = 0;
}
```

（3）main.c：起始演奏键恢复成以 I/O 的方式并对其进行判断之，当按下起始键，则 KeyData 变量将写入 START_KEY 的数值，而且应将演奏下一曲目的标志 FgNextMusic 清除，其计时也要从 0 开始，即 MusicLength=0，并设定计时开始，也开始演奏，即 Music( )函数待时间终了得在计时中断函数设定 FgNextMusic 标志，使得 Music( )函数立即退出而达到中止演奏的目的，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
```

```
#include  "7segled.h"
#include  "ic4094.h"

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
      //sel play no#
      InputHandler( );

      //detect start key
      InputStart( );
      if ( KeyData == START_KEY )
      {
         FgEndMusic =0;      //int0  :END_KEY

         while( 1 )
         {
            FgNextMusic=0;   //timer0:FgNextMusic=1
            MusicLength=0;
            TR0 = 1;
            Music(MusicNumber);
            TR0 = 0;

            if (FgEndMusic==1) break;

            MusicNumber++;
            if ( MusicNumber > LASTMUSIC )
               MusicNumber=1;
            DelayX10ms(100);
         }
      }
   }
}
```

（4）music1.c：在音乐演奏的曲目中，要有停止演奏及演奏下一曲目标志的判断。一旦发生，要立即退出此音乐演奏的循环，分别为当 FgEndMusic=“1”及 FgNextMusic=“1”，要返回至主程序，今列举 music1()函数，其他的音乐演奏曲目都是这样，如下：

```
void Music1(void)
{
```

```
    unsigned char i=0;

    Tempo = MODERATO;

    while ( (SOUNDLONG1[i]!=_EOF)||(SOUNDTONE1[i]!=_EOF) )
    {
      if (FgEndMusic ==1) return;
      if (FgNextMusic==1) return;

      //start timer1,generate soundtone
      Period=SOUNDTONE1[i];

      TL1 = (65536 - Period) & 0xff;
      TH1 = (65536 - Period) >> 8;
      TR1 = 1;

      SoundLongCount=((Tempo*SOUNDLONG1[i]*1000L)/8)/Period;
      while ( SoundLongCount != 0 );

      TR1 = 0;       //stop timer1:soundtone
      i++;
    }
}
```

汇编程序如下：

1. main.a51

```
$NOMOD51

NAME    MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN       SEGMENT CODE
    EXTRN   BIT (FgEndMusic)
    EXTRN   BIT (FgNextMusic)
    EXTRN   IDATA (MusicLength)
    EXTRN   IDATA (KeyData)
    EXTRN   IDATA (MusicNumber)
    EXTRN   CODE (PowerOnInit)
    EXTRN   CODE (_DelayX10ms)
    EXTRN   CODE (_Music)
    EXTRN   CODE (InputHandler)
    EXTRN   CODE (InputStart)
```

```
    EXTRN   CODE (?C_STARTUP)
    PUBLIC  Main

    RSEG  ?PR?Main?MAIN
    USING   0
Main:
    LCALL   PowerOnInit
?C0001:
    LCALL   InputHandler
    LCALL   InputStart
    MOV     R0,#KeyData
    MOV     A,@R0
    CJNE    A,#01H,?C0001
    CLR     FgEndMusic
?C0004:
    CLR     FgNextMusic
    CLR     A
    MOV     R0,#MusicLength
    MOV     @R0,A
    SETB    TR0
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Music
    CLR     TR0
    JB      FgEndMusic,?C0001
?C0006:
    MOV     R0,#MusicNumber
    INC     @R0
    MOV     A,@R0
    SETB    C
    SUBB    A,#01FH
    JC      ?C0007
    MOV     @R0,#01H
?C0007:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0004
    RET

    END
```

2. timer.a51

```
$NOMOD51

NAME    TIMER

$INCLUDE (mtv212m.inc)

?PR?INT0ISR?TIMER   SEGMENT CODE
?PR?Timer0ISR?TIMER     SEGMENT CODE
?PR?Timer1ISR?TIMER     SEGMENT CODE
    EXTRN   BIT (FgEndMusic)
    EXTRN   BIT (FgNextMusic)
    EXTRN   IDATA (MusicLength)
    EXTRN   IDATA (Period)
    EXTRN   IDATA (SoundLongCount)
    PUBLIC  Timer1ISR
    PUBLIC  Timer0ISR
    PUBLIC  INT0ISR

CSEG    AT  00003H
    LJMP    INT0ISR

    RSEG  ?PR?INT0ISR?TIMER
    USING   3
INT0ISR:
    CLR     EX0
    SETB    FgEndMusic
    SETB    EX0
    CLR     IE0
    RETI

CSEG    AT  0000BH
    LJMP    Timer0ISR

    RSEG  ?PR?Timer0ISR?TIMER
    USING   3
Timer0ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     PSW,#018H
    MOV     R0,#MusicLength
    INC     @R0
    MOV     A,@R0
    CJNE    A,#07DH,?C0002
    SETB    FgNextMusic
?C0002:
```

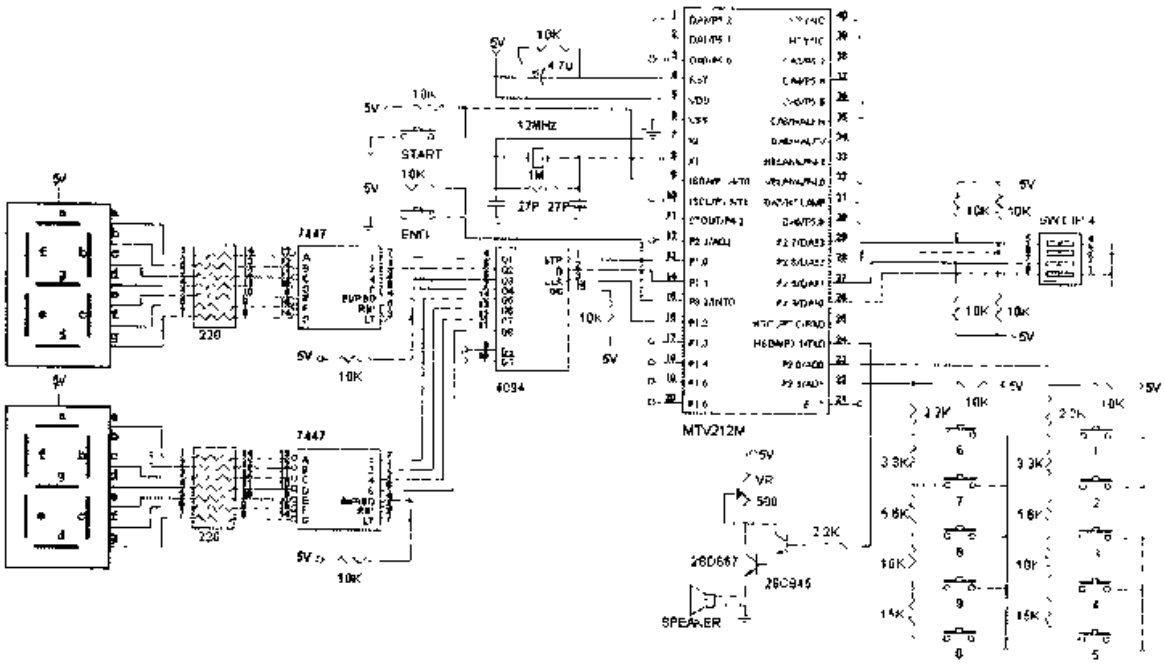
```
    MOV     TL0,#0C0H
    MOV     TH0,#063H
    CLR     TF0
    POP     PSW
    POP     ACC
    RETI

CSEG    AT  0001BH
    LJMP    Timer1ISR

    RSEG  ?PR?Timer1ISR?TIMER
    USING   2
Timer1ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     PSW,#010H
    MOV     R0,#SoundLongCount+01H
    MOV     A,@R0
    DEC     R0
    ORL     A,@R0
    JZ      ?C0004
    INC     R0
    MOV     A,@R0
    DEC     @R0
    JNZ     ?C0004
    DEC     R0
    DEC     @R0
?C0004:
    MOV     R0,#Period
    MOV     A,@R0
    MOV     R6,A
    INC     R0
    MOV     A,@R0
    MOV     R7,A
    CLR     C
    CLR     A
    SUBB    A,R7
    MOV     R7,A
    CLR     A
    SUBB    A,R6
    MOV     R6,A
    CPL     P3_1
    XCH     A,R5
    MOV     A,R7
    XCH     A,R5
```

```
MOV     A,R5
MOV     TL1,A
MOV     A,R6
MOV     TH1,A
CLR     TF1
POP     PSW
POP     ACC
RETI

END
```

11-5 功能选择 DIP 方式

目的：以 DIP 指拨开关来选取功能模式，分别为单曲播放、单曲循环、顺序播放、随机选曲及播放简介等功能。

电路图：



原理：以四个 DIP 开关分别来表示四个不同的功能模式，当 DIP 都不动作，即四个 I/O 都为“1”电位时，则表示为单曲播放，也就是只播放所输入按键的曲目，当 P2.4=“0”为单曲循环功能，P2.5=“0”为顺序播放功能，P2.6=“0”为随机选曲功能，P2.7=“0”为播放简介功能，而 DIP1 作为 I/O P2.4 的输入，DIP2 作为 I/O P2.5 的输入，DIP3 作为 I/O P2.6 的输入，DIP4 作为 I/O 2.7 的输入，以 DIP 指拨开关作为功能模式的选择是最常见、最简单的。

软件构建的思维与解决方法

以 DIP 开关作为输入，所以要有侦测此功能模式的函数，而将此四个 DIP 的状态存入，在主程序中必须根据此状态而执行不同的功能。即当 P2.4=“0”，要执行单曲循环功能；P2.5=“0”，要执行顺序播放功能；P2.6=“0”，要执行随机选曲功能；P2.7=“0”，要执行播放简介功能。而 timer0 要根据不同模式而作计时或计数的规划，当模式为随机选曲功能则 timer 0 要规划为计数模式，以计数弹跳波个数作为随机函数的种子数值，当模式为播放简介功能，则 timer0 要规划为计时模式，以计时五秒后就要演奏下一曲目的功能，现分述如下：

（1）input.c：DIP 开关的侦测函数名称为 Fuctionkey( )函数，其 DIP 的动作是由 I/O P2.4～P2.7 所输入，其读入的数值需屏蔽低的四个位后，再右移四个位而存入至 FunctionData 功能模式的状态变量内，如下：

```
/*****************************/
void FunctionKey(void)
{
   Byte  keytmp;

   keytmp = ~(DIP_PORT) & 0xf0;//P2.4~P2.7

   FunctionData = keytmp>>4;
}
```

（2）input.h：起始键的 I/O 输入为 P3.4 或是随机选曲功能的 T0 计数器输入，因为 mtv212m CPU 此引脚可作为 I/O P3.4 或 T0 计数器的输入，而 DIP_PORT 为 P2.4～P2.7，所以需设定 P2 端口。另外，每一按键依序设定其数值为 0～9，以及新增 Functionkey( )函数的外部声明，如下：

```
#ifndef     _INPUT_H
#define     _INPUT_H

#define  START_IO        P3_4
#define  DIP_PORT        P2   //P2.4~P1.7

#define  NO_KEY          0xff
#define  START_KEY       1

#define  NUMBER0_KEY0
#define  NUMBER1_KEY1
#define  NUMBER2_KEY2
#define  NUMBER3_KEY3
#define  NUMBER4_KEY4
```

```
#define  NUMBER5_KEY5
#define  NUMBER6_KEY6
#define  NUMBER7_KEY7
#define  NUMBER8_KEY8
#define  NUMBER9_KEY9

extern void InputHandler(void);
extern void InputKey(void);
extern void InputStart(void);
extern void FunctionKey(void);

#endif
```

（3）timer.c：timer0 有两个用途：

- 当功能模式为随机选曲功能，即 P2.6=“0”电位，FuncionData=4，要作为计数器，当每一个负脉冲产生则立即中断而计数负脉冲的个数，以作为随机函数的种子数。
- 当功能模式为播放简介功能，即 P2.7=“0”电位，FunctionData=8，要作为定时器，每 40ms 就产生一次计时中断，并累加变量值是否到了五秒的时间，使得主程序得以演奏下一首曲目，如此反复。其程序如下：

```
/*****************************/
//随机:count0 及播放简介:timer0
/*****************************/
void  Timer0ISR ( void ) interrupt 1 using 3
{
   if (FunctionData==4)      //随机选曲
   {
      PulseCount++;

      FgStartMusic=1;
      KeyData=START_KEY;

      TL0 = 0xff;
      TH0 = 0xff;
      TF0 = 0;
   }
   else if (FunctionData==8 )   //播放简介
   {

      MusicLength++;
      if ( MusicLength == TIME_5SEC )
         FgNextMusic=1;
```

```
        TL0 = CLOCK_40MS & 0xff;
        TH0 = CLOCK_40MS >> 8;
        TF0 = 0;
    }
}
```

（4）main.c：将四种功能模式由 DIP 开关来决定目前的功能模式是什么？因此，要先取得 DIP 开关的状态，并根据此状态而作不同的执行功能，此 DIP 开关的状态为 FuncionData 变量，程序以 switch...case 的指令结构来判断 FunctionData 的内容。当 FunctionData=0X01 则执行单曲循环功能，FunctionData=0X02，则执行顺序播放功能，FunctionData=0X04 则执行随机选曲功能，FuntctionData=0X08 则执行播放简介功能，当 DIP 开关都未拨动而保持在原来位置，则 FunctionData 将为 0X00，则执行单曲播放一次的功能，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"
#include  "ic4094.h"
#include  <stdlib.h>

/*****************************/
void Main( void )
{
   int lastvalue,currentvalue;

   PowerOnInit();

   while( 1 )
   {
      //sel play no#
      InputHandler( );

      //function item
      FunctionKey( );
```

```
//detect start key
InputStart( );
if ( KeyData == START_KEY )
{
   FgEndMusic =0;                    //int0  :END_KEY

   switch( FunctionData )
   {
    case 0 :                         //单曲播放
      Music(MusicNumber);
      if (FgEndMusic==1) break;
      DelayX10ms(100);
      break;
    case 1 :                         //单曲循环
      while( 1 )
      {
        Music(MusicNumber);
        if (FgEndMusic==1) break;
        DelayX10ms(100);
      }
      break;
    case 2 :                         //顺序播放
      while( 1 )
      {
        Music(MusicNumber);
        if (FgEndMusic==1) break;

        MusicNumber++;
        if ( MusicNumber > LASTMUSIC )
          MusicNumber=1;
        DelayX10ms(100);
      }
      break;
    case 4 :                         //随机选曲
       FgStartMusic=0;
       DelayX10ms(50);
       srand(PulseCount);

      while( 1 )
      {
        currentvalue=rand();  //产生一个随机数
        currentvalue%=100;    //取最后两位数 0~99

        //currentvalue=1~LASTMUSIC
        if ( (currentvalue<=LASTMUSIC)&&(currentvalue>0) )
```

```
                {
                  //和上次不一样
                  if (currentvalue!=lastvalue)
                  {
                    MusicNumber=currentvalue;
                    Music(MusicNumber);
                    lastvalue=currentvalue;
                    if (FgEndMusic==1) break;
                    DelayX10ms(100);
                  }
                }
              }
              break;
            case 8 :                    //播放简介
              while( 1 )
              {
                FgNextMusic=0;          //timer0:FgNextMusic=1
               MusicLength=0;
               TR0 = 1;
                Music(MusicNumber);
                TR0 = 0;

                if (FgEndMusic==1) break;

                MusicNumber++;
                if ( MusicNumber > LASTMUSIC )
                  MusicNumber=1;
                DelayX10ms(100);
              }
              break;
            default :
              break;
          }
        }
      }
    }
}
```

汇编程序列表如下：

1. **main.a51**

```
$NOMOD51

NAME    MAIN
```

```
$INCLUDE (mtv212m.inc)

?PR?Main?MAIN           SEGMENT CODE
?DT?Main?MAIN       SEGMENT DATA OVERLAYABLE
    EXTRN   BIT (FgEndMusic)
    EXTRN   BIT (FgNextMusic)
    EXTRN   BIT (FgStartMusic)
    EXTRN   IDATA (MusicLength)
    EXTRN   IDATA (PulseCount)
    EXTRN   IDATA (KeyData)
    EXTRN   IDATA (FunctionData)
    EXTRN   IDATA (MusicNumber)
    EXTRN   CODE (PowerOnInit)
    EXTRN   CODE (_DelayX10ms)
    EXTRN   CODE (_Music)
    EXTRN   CODE (InputHandler)
    EXTRN   CODE (InputStart)
    EXTRN   CODE (FunctionKey)
    EXTRN   CODE (rand)
    EXTRN   CODE (_srand)
    EXTRN   CODE (?C_STARTUP)
    EXTRN   CODE (?C?SIDIV)
    PUBLIC  Main

    RSEG  ?DT?Main?MAIN
?Main?BYTE:
  lastvalue?040:   DS   2
currentvalue?041:   DS   2

    RSEG  ?PR?Main?MAIN
    USING   0
Main:
    LCALL   PowerOnInit
?C0001:
    LCALL   InputHandler
    LCALL   FunctionKey
    LCALL   InputStart
    MOV     R0,#KeyData
    MOV     A,@R0
    CJNE    A,#01H,?C0001
    CLR     FgEndMusic
    MOV     R0,#FunctionData
    MOV     A,@R0
    DEC     A
    JZ      ?C0008
```

```
    DEC     A
    JZ      ?C0012
    ADD     A,#0FEH
    JZ      ?C0016
    ADD     A,#0FCH
    JNZ     $ + 5H
    LJMP    ?C0023
    ADD     A,#08H
    JNZ     ?C0001
?C0005:
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Music
    JB      FgEndMusic,?C0001
?C0006:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0001
?C0008:
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Music
    JB      FgEndMusic,?C0001
?C0010:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0008
?C0012:
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Music
    JB      FgEndMusic,?C0001
?C0014:
    MOV     R0,#MusicNumber
    INC     @R0
    MOV     A,@R0
    SETB    C
    SUBB    A,#01FH
    JC      ?C0015
    MOV     @R0,#01H
```

```
?C0015:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0012
?C0016:
    CLR     FgStartMusic
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    MOV     R0,#PulseCount
    MOV     A,@R0
    MOV     R6,A
    INC     R0
    MOV     A,@R0
    MOV     R7,A
    LCALL   _srand
?C0017:
    LCALL   rand
    MOV     currentvalue?041,R6
    MOV     currentvalue?041+01H,R7
    MOV     R4,#00H
    MOV     R5,#064H
    LCALL   ?C?SIDIV
    MOV     currentvalue?041,R4
    MOV     currentvalue?041+01H,R5
    SETB    C
    MOV     A,currentvalue?041+01H
    SUBB    A,#01FH
    MOV     A,currentvalue?041
    XRL     A,#080H
    SUBB    A,#080H
    JNC     ?C0017
    SETB    C
    MOV     A,currentvalue?041+01H
    SUBB    A,#00H
    MOV     A,currentvalue?041
    XRL     A,#080H
    SUBB    A,#080H
    JC      ?C0017
    MOV     A,currentvalue?041+01H
    XRL     A,lastvalue?040+01H
    JNZ     ?C0029
    MOV     A,currentvalue?041
    XRL     A,lastvalue?040
```

```
?C0029:
    JZ      ?C0017
    MOV     R7,currentvalue?041+01H
    MOV     R0,#MusicNumber
    MOV     A,R7
    MOV     @R0,A
    LCALL   _Music
    MOV     lastvalue?040,currentvalue?041
    MOV     lastvalue?040+01H,currentvalue?041+01H
    JNB     FgEndMusic,$ + 6H
    LJMP    ?C0001
?C0021:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0017
?C0023:
    CLR     FgNextMusic
    CLR     A
    MOV     R0,#MusicLength
    MOV     @R0,A
    SETB    TR0
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Music
    CLR     TR0
    JNB     FgEndMusic,$ + 6H
    LJMP    ?C0001
?C0025:
    MOV     R0,#MusicNumber
    INC     @R0
    MOV     A,@R0
    SETB    C
    SUBB    A,#01FH
    JC      ?C0026
    MOV     @R0,#01H
?C0026:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0023
    RET

    END
```

2. timer.a51

```
$NOMOD51

NAME    TIMER

$INCLUDE (mtv212m.inc)

?PR?INT0ISR?TIMER       SEGMENT CODE
?PR?Timer0ISR?TIMER     SEGMENT CODE
?PR?Timer1ISR?TIMER     SEGMENT CODE
    EXTRN   BIT (FgEndMusic)
    EXTRN   BIT (FgNextMusic)
    EXTRN   BIT (FgStartMusic)
    EXTRN   IDATA (MusicLength)
    EXTRN   IDATA (PulseCount)
    EXTRN   IDATA (KeyData)
    EXTRN   IDATA (FunctionData)
    EXTRN   IDATA (Period)
    EXTRN   IDATA (SoundLongCount)
    PUBLIC  Timer1ISR
    PUBLIC  Timer0ISR
    PUBLIC  INT0ISR

CSEG    AT  00003H
    LJMP    INT0ISR

    RSEG  ?PR?INT0ISR?TIMER
    USING   3
INT0ISR:
    CLR     EX0
    SETB    FgEndMusic
    SETB    EX0
    CLR     IE0
    RETI

CSEG    AT  0000BH
    LJMP    Timer0ISR

    RSEG  ?PR?Timer0ISR?TIMER
    USING   3
Timer0ISR:
    PUSH    ACC
    PUSH    PSW
```

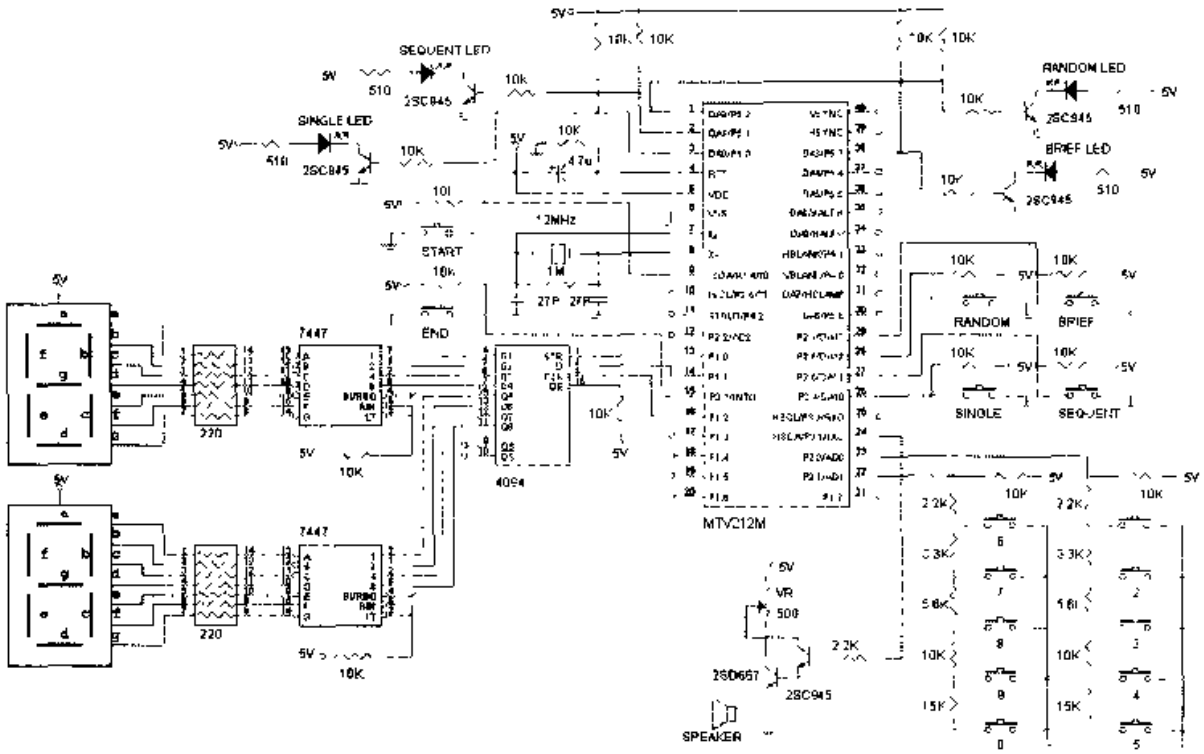
```
    MOV     PSW,#018H
    MOV     R0,#FunctionData
    MOV     A,@R0
    MOV     R7,A
    CJNE    A,#04H,?C0002
    MOV     R0,#PulseCount+01H
    INC     @R0
    MOV     A,@R0
    JNZ     ?C0009
    DEC     R0
    INC     @R0
?C0009:
    SETB    FgStartMusic
    MOV     R0,#KeyData
    MOV     @R0,#01H
    MOV     TL0,#0FFH
    MOV     TH0,#0FFH
    CLR     TF0
    SJMP    ?C0006
?C0002:
    MOV     A,R7
    CJNE    A,#08H,?C0006
    MOV     R0,#MusicLength
    INC     @R0
    MOV     A,@R0
    CJNE    A,#07DH,?C0005
    SETB    FgNextMusic
?C0005:
    MOV     TL0,#0C0H
    MOV     TH0,#063H
    CLR     TF0
?C0006:
    POP     PSW
    POP     ACC
    RETI

CSEG    AT  0001BH
    LJMP    Timer1ISR

    RSEG  ?PR?Timer1ISR?TIMER
    USING   2
Timer1ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     PSW,#010H
```

```
        MOV     R0,#SoundLongCount+01H
        MOV     A,@R0
        DEC     R0
        ORL     A,@R0
        JZ      ?C0007
        INC     R0
        MOV     A,@R0
        DEC     @R0
        JNZ     ?C0007
        DEC     R0
        DEC     @R0
    ?C0007:
        MOV     R0,#Period
        MOV     A,@R0
        MOV     R6,A
        INC     R0
        MOV     A,@R0
        MOV     R7,A
        CLR     C
        CLR     A
        SUBB    A,R7
        MOV     R7,A
        CLR     A
        SUBB    A,R6
        MOV     R6,A
        CPL     P3_1
        XCH     A,R5
        MOV     A,R7
        XCH     A,R5
        MOV     A,R5
        MOV     TL1,A
        MOV     A,R6
        MOV     TH1,A
        CLR     TF1
        POP     PSW
        POP     ACC
        RETI

        END
```

11-6 功能选择 TACT 和 LED 方式

目的：以 TACT 开关来选择功能模式并以 DAC 作为对应功能模式 LED 的输出。

电路图：



原理：以 P2.4～P2.7 作为 TACT 开关的功能模式输入，并将对应的 LED 点亮，但又由于 I/O 的输出不敷使用，而利使用 DAC0～DAC3 的 PWM 输出作为驱动 LED 的点亮与否，当将 PWM 输出设定为数值 0Xff 时，其输出为“1”电位，即+5V，当将 PWM 输出设定为数值 0X00 时，其输出为“0”电位，即 0V，因此，可作为 I/O 的输出使用。

软件构建的思维与解决方法

以 LED 来显示目前功能模式的状态，易于明了但也多需要四个 I/O 以驱动 LED，由于 I/O 不够用，所以将 PWM 的输出作为 I/O 使用。在初始化时应将四个 LED 清除。而功能模式是以 TACT 开关选择，即当 TACT 开关按一下，则 LED 亮，再按一下则 LED 熄，当换成不同功能模式的选择时，则应将其他的 LED 都熄灭，只点亮所选择功能模式的 LED 而已，将此较复杂的处理程序以 FunctionHandler( )函数进行，而执行功能模式的音乐演奏处理程序则以 StartHandler( )函数进行。这样，Main( )函数只需调用这些函数即可，从而变得较清晰明了，现分述如下：

（1）initial.c：以 I/O P2.4～P2.7 作为输入，因此，将 P2 端口输出为 0Xf0，即 P2.4～P2.7 都设定为“1”电位，而四个功能模式的 LED 都使其熄灭，PWM 的输出应为 0X00，如下：

```
/*******************************/
```

```
void InitialCpuIO(void)
{
   //display "00"
   Output4094(0x00);

   //P2.4~P2.7:input
   P2=0xf0;

   //all led off
   SINGLE_LED =0x00;        //DAC0
   SEQUENT_LED=0x00;        //DAC1
   RANDOM_LED =0x00;        //DAC2
   BRIEF_LED  =0x00;        //DAC3

   SPEAKER = 0;
}
```

（2）main.c：将曲目编号的输入、功能模式 LED 的亮或熄，以及按下起始演奏键功能模式的执行动作，都改成以函数并写在 input.c 模块下，使得 Main( )函数变得清晰明了，如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"
#include  "ic4094.h"

/*****************************/
void Main( void )
{
   PowerOnInit();

   while( 1 )
   {
     //sel play no#
     InputHandler( );
```

```
        //function item
        FunctionHandler( );

        //detect start key
        StartHandler( );
    }
}
```

（3）input.c：包含按键曲目编号的显示器处理函数、利用 ADC 方式的按键侦测函数、起始演奏键的侦测函数，以及执行功能模式音乐演奏处理函数。在前面章节中说明，此处不再重复，另外增加的函数如下：

- **Functionkey( )函数**：以 P2.4～P2.7 作为功能模式的选择，当按下 TACT 开关，则对应的数值存入至 FunctionData 变量内，即按下单曲循环键，则 FunctionData=10，按下顺序播放键，则 FunctionData=11，按下随机选曲键，则 FunctionData=12，按下播放简介键，则 FunctionData=13，其程序如下：

```
void FunctionKey(void)
{
    if ( SINGLE_IO==0 )
      FunctionData=SINGLE_KEY;
    else if ( SEQUENT_IO==0 )
      FunctionData=SEQUENT_KEY;
    else if ( RANDOM_IO==0 )
      FunctionData=RANDOM_KEY;
    else if ( BRIEF_IO==0 )
      FunctionData=BRIEF_KEY;
    else
      FunctionData=NO_KEY;
}
```

- **FunctionHandler( )函数**：功能模式的 LED 处理程序以及设定或清除对应的功能模式标志。其中每按下开关一次则 LED 亮，再按一次 LED 又变成熄灭，如此反复的循环，如果其中一种功能模式设定了则 LED 亮，但后来又设定另外一种功能模式时，则原先功能模式的 LED 将熄灭，而目前所设定功能模式的 LED 将点亮，即使持续压着键也只动作一次，而当放开键，则将持续按键标志清除，分别将 FgSingleKey、FgSequentKey、FgRandmoKey 及 FgBriefKey 标志都清除，其程序如下：

```
/*****************************/
void FunctionHandler(void)
{
    //sel play no#
    FunctionKey( );
```

```
if ( KeyBuffer2 != FunctionData )
{
   KeyBuffer2 = FunctionData;
   ScanKeyCounter = 10;
}
else if( ScanKeyCounter != 0 )
{
   ScanKeyCounter--;
   DelayX1ms(1);
}
else if ( KeyBuffer2 != NO_KEY )
{
   switch ( KeyBuffer2 )
   {
    case SINGLE_KEY :
      if ( FgSingleKey==0 )
      {
         FgSingleKey = 1;
         Beep(1,17,10);
         FgSingleFlag= !FgSingleFlag;
         if ( FgSingleFlag==1 )
            SINGLE_LED=0xff;
         else
            SINGLE_LED=0x00;

         //other key off
         FgSequentFlag=0;
         FgRandomFlag =0;
         FgBriefFlag  =0;
         SEQUENT_LED =0x00;
         RANDOM_LED =0x00;
         BRIEF_LED   =0x00;
      }
      break;
    case SEQUENT_KEY :
      if ( FgSequentKey==0 )
      {
         FgSequentKey = 1;
         Beep(1,17,10);
         FgSequentFlag= !FgSequentFlag;
         if ( FgSequentFlag==1 )
            SEQUENT_LED=0xff;
         else
            SEQUENT_LED=0x00;
```

```
        FgSingleFlag =0;
        FgRandomFlag =0;
        FgBriefFlag  =0;
        SINGLE_LED  =0x00;
        RANDOM_LED =0x00;
        BRIEF_LED   =0x00;
      }
      break;
    case RANDOM_KEY :
      if ( FgRandomKey==0 )
      {
        FgRandomKey = 1;
        Beep(1,17,10);

        TMOD= 0x15;       //set timer1,count0 mode
        TL0 = 0xff;       //count 1 times
        TH0 = 0xff;
        TR0 = 1;          //start count0

        FgRandomFlag= !FgRandomFlag;
        if ( FgRandomFlag==1 )
          RANDOM_LED=0xff;
        else
          RANDOM_LED=0x00;

        FgSingleFlag =0;
        FgSequentFlag=0;
        FgBriefFlag  =0;
        SINGLE_LED  =0x00;
        SEQUENT_LED =0x00;
        BRIEF_LED   =0x00;
      }
      break;
    case BRIEF_KEY :
      if ( FgBriefKey==0 )
      {
        FgBriefKey = 1;
        Beep(1,17,10);

        TMOD= 0x11;       //set timer1,timer0 mode
        //CLOCK_40MS=(65536 - 40000)
        TL0 = CLOCK_40MS & 0xff;
        TH0 = CLOCK_40MS >> 8;
```

```
                FgBriefFlag= !FgBriefFlag;
                if ( FgBriefFlag==1 )
                  BRIEF_LED=0xff;
                else
                  BRIEF_LED=0x00;

                FgSingleFlag =0;
                FgSequentFlag=0;
                FgRandomFlag =0;
                SINGLE_LED  =0x00;
                SEQUENT_LED =0x00;
                RANDOM_LED =0x00;
              }
              break;
            default :
              break;
          }
        }
        else
        {
          KeyBuffer2  = NO_KEY;

          FgSingleKey = 0;
          FgSequentKey= 0;
          FgRandomKey = 0;
          FgBriefKey  = 0;
        }
    }
```

（4）input.h：功能模式的 I/O 定义为 P2.4～P2.7，以及对应的数值为 10～13 和新增的函数外部声明，如下：

```
#ifndef     _INPUT_H
#define     _INPUT_H

#define  START_IO        P3_4
#define  SINGLE_IO       P2_4
#define  SEQUENT_IO P2_5
#define  RANDOM_IO       P2_6
#define  BRIEF_IO        P2_7

#define  NO_KEY          0xff
#define  START_KEY       1

#define  NUMBER0_KEY0
```

```
#define  NUMBER1_KEY1
#define  NUMBER2_KEY2
#define  NUMBER3_KEY3
#define  NUMBER4_KEY4
#define  NUMBER5_KEY5
#define  NUMBER6_KEY6
#define  NUMBER7_KEY7
#define  NUMBER8_KEY8
#define  NUMBER9_KEY9
#define  SINGLE_KEY 10
#define  SEQUENT_KEY11
#define  RANDOM_KEY 12
#define  BRIEF_KEY      13

extern void InputHandler(void);
extern void InputKey(void);
extern void FunctionHandler(void);
extern void FunctionKey(void);
extern void StartHandler(void);
extern void InputStart(void);

#endif
```

汇编程序如下：

1. main.a51

```
$NOMOD51

NAME    MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN       SEGMENT CODE
    EXTRN   CODE (PowerOnInit)
    EXTRN   CODE (InputHandler)
    EXTRN   CODE (FunctionHandler)
    EXTRN   CODE (StartHandler)
    EXTRN   CODE (?C_STARTUP)
    PUBLIC  Main

    RSEG  ?PR?Main?MAIN
    USING   0
Main:
    LCALL   PowerOnInit
```

```
?C0001:
    LCALL   InputHandler
    LCALL   FunctionHandler
    LCALL   StartHandler
    SJMP    ?C0001
    RET

    END
```

2. input.a51

```
$NOMOD51

NAME    INPUT

$INCLUDE (mtv212m.inc)

?PR?InputHandler?INPUT      SEGMENT CODE
?PR?InputKey?INPUT          SEGMENT CODE
?DT?InputKey?INPUT          SEGMENT DATA OVERLAYABLE
?PR?FunctionHandler?INPUT   SEGMENT CODE
?PR?FunctionKey?INPUT       SEGMENT CODE
?PR?StartHandler?INPUT      SEGMENT CODE
?DT?StartHandler?INPUT      SEGMENT DATA OVERLAYABLE
?PR?InputStart?INPUT            SEGMENT CODE
    EXTRN   BIT (FgEndMusic)
    EXTRN   BIT (FgNextMusic)
    EXTRN   BIT (FgStartMusic)
    EXTRN   IDATA (MusicLength)
    EXTRN   IDATA (PulseCount)
    EXTRN   BIT (FgSingleKey)
    EXTRN   BIT (FgSequentKey)
    EXTRN   BIT (FgRandomKey)
    EXTRN   BIT (FgBriefKey)
    EXTRN   BIT (FgSingleFlag)
    EXTRN   BIT (FgSequentFlag)
    EXTRN   BIT (FgRandomFlag)
    EXTRN   BIT (FgBriefFlag)
    EXTRN   BIT (FgKeyFlag)
    EXTRN   IDATA (KeyData)
    EXTRN   IDATA (KeyBuffer1)
    EXTRN   IDATA (KeyBuffer2)
    EXTRN   IDATA (LastBuffer)
    EXTRN   IDATA (ScanKeyCounter)
    EXTRN   IDATA (FunctionData)
```

```
    EXTRN   IDATA (MusicNumber)
    EXTRN   XDATA (SINGLE_LED)
    EXTRN   XDATA (SEQUENT_LED)
    EXTRN   XDATA (RANDOM_LED)
    EXTRN   XDATA (BRIEF_LED)
    EXTRN   XDATA (XFR_ADC)
    EXTRN   CODE (_DelayX10ms)
    EXTRN   CODE (_DelayX1ms)
    EXTRN   CODE (_Music)
    EXTRN   CODE (_Beep)
    EXTRN   CODE (_Disp7Seg)
    EXTRN   CODE (rand)
    EXTRN   CODE (_srand)
    EXTRN   CODE (?C?SIDIV)
    PUBLIC  InputStart
    PUBLIC  StartHandler
    PUBLIC  FunctionKey
    PUBLIC  FunctionHandler
    PUBLIC  InputKey
    PUBLIC  InputHandler

    RSEG  ?DT?InputKey?INPUT
?InputKey?BYTE:
    keytmp?140:   DS   1

    RSEG  ?DT?StartHandler?INPUT
?StartHandler?BYTE:
  lastvalue?441:   DS   2
currentvalue?442:   DS   2

    RSEG  ?PR?InputHandler?INPUT
    USING   0
InputHandler:
    LCALL   InputKey
    MOV     R0,#KeyData
    MOV     A,@R0
    MOV     R7,A
    MOV     R0,#KeyBuffer1
    XRL     A,@R0
    JZ      ?C0001
    MOV     A,R7
    MOV     @R0,A
    MOV     R0,#ScanKeyCounter
    MOV     @R0,#0AH
    RET
```

```
?C0001:
    MOV     R0,#ScanKeyCounter
    MOV     A,@R0
    JZ      ?C0003
    DEC     @R0
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    RET
?C0003:
    MOV     R0,#KeyBuffer1
    MOV     A,@R0
    CPL     A
    JZ      ?C0005
    JB      FgKeyFlag,?C0008
    SETB    FgKeyFlag
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep
    MOV     R0,#LastBuffer
    MOV     A,@R0
    MOV     B,#0AH
    MUL     AB
    MOV     R0,#KeyBuffer1
    ADD     A,@R0
    MOV     R7,A
    MOV     R0,#MusicNumber
    MOV     @R0,A
    LCALL   _Disp7Seg
    MOV     R0,#KeyBuffer1
    MOV     A,@R0
    MOV     R0,#LastBuffer
    MOV     @R0,A
    RET
?C0005:
    MOV     R0,#KeyBuffer1
    MOV     @R0,#0FFH
    CLR     FgKeyFlag
?C0008:
    RET

    RSEG  ?PR?InputKey?INPUT
    USING   0
```

```
InputKey:
    MOV     R0,#KeyData
    MOV     @R0,#0FFH
    MOV     R0,#LOW (XFR_ADC)
    MOV     A,#081H
    MOVX    @R0,A
    MOV     R7,#02H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#LOW (XFR_ADC)
    MOVX    A,@R0
    ANL     A,#03FH
    MOV     keytmp?140,A
    CLR     C
    SUBB    A,#010H
    JNC     ?C0009
    MOV     R0,#KeyData
    MOV     @R0,#01H
    SJMP    ?C0010
?C0009:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#01BH
    JNC     ?C0011
    MOV     R0,#KeyData
    MOV     @R0,#02H
    SJMP    ?C0010
?C0011:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#026H
    JNC     ?C0013
    MOV     R0,#KeyData
    MOV     @R0,#03H
    SJMP    ?C0010
?C0013:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#030H
    JNC     ?C0015
    MOV     R0,#KeyData
    MOV     @R0,#04H
    SJMP    ?C0010
?C0015:
    MOV     A,keytmp?140
```

```
    CLR     C
    SUBB    A,#036H
    JNC     ?C0010
    MOV     R0,#KeyData
    MOV     @R0,#05H
?C0010:
    MOV     R0,#LOW (XFR_ADC)
    MOV     A,#082H
    MOVX    @R0,A
    MOV     R7,#02H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#LOW (XFR_ADC)
    MOVX    A,@R0
    ANL     A,#03FH
    MOV     keytmp?140,A
    CLR     C
    SUBB    A,#010H
    JNC     ?C0018
    MOV     R0,#KeyData
    MOV     @R0,#06H
    SJMP    ?C0019
?C0018:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#01BH
    JNC     ?C0020
    MOV     R0,#KeyData
    MOV     @R0,#07H
    SJMP    ?C0019
?C0020:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#026H
    JNC     ?C0022
    MOV     R0,#KeyData
    MOV     @R0,#08H
    SJMP    ?C0019
?C0022:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#030H
    JNC     ?C0024
    MOV     R0,#KeyData
    MOV     @R0,#09H
```

```
    SJMP    ?C0019
?C0024:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#036H
    JNC     ?C0019
    CLR     A
    MOV     R0,#KeyData
    MOV     @R0,A
?C0019:
    CLR     A
    MOV     R0,#LOW (XFR_ADC)
    MOVX    @R0,A
    RET

    RSEG  ?PR?FunctionHandler?INPUT
    USING   0
FunctionHandler:
    LCALL   FunctionKey
    MOV     R0,#FunctionData
    MOV     A,@R0
    MOV     R7,A
    MOV     R0,#KeyBuffer2
    XRL     A,@R0
    JZ      ?C0028
    MOV     A,R7
    MOV     @R0,A
    MOV     R0,#ScanKeyCounter
    MOV     @R0,#0AH
    RET
?C0028:
    MOV     R0,#ScanKeyCounter
    MOV     A,@R0
    JZ      ?C0030
    DEC     @R0
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    RET
?C0030:
    MOV     R0,#KeyBuffer2
    MOV     A,@R0
    MOV     R7,A
    CPL     A
    JNZ     $ + 5H
```

```
    LJMP    ?C0032
    MOV     A,R7
    ADD     A,#0F5H
    JZ      ?C0038
    DEC     A
    JZ      ?C0042
    DEC     A
    JNZ     $ + 5H
    LJMP    ?C0046
    ADD     A,#03H
    JZ      $ + 5H
    LJMP    ?C0052
?C0034:
    JNB     FgSingleKey,$ + 6H
    LJMP    ?C0052
    SETB    FgSingleKey
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep
    CPL     FgSingleFlag
    JNB     FgSingleFlag,?C0036
    MOV     R0,#LOW (SINGLE_LED)
    MOV     A,#0FFH
    MOVX    @R0,A
    SJMP    ?C0037
?C0036:
    CLR     A
    MOV     R0,#LOW (SINGLE_LED)
    MOVX    @R0,A
?C0037:
    CLR     FgSequentFlag
    CLR     FgRandomFlag
    CLR     FgBriefFlag
    CLR     A
    MOV     R0,#LOW (SEQUENT_LED)
    MOVX    @R0,A
    MOV     R0,#LOW (RANDOM_LED)
    MOVX    @R0,A
    MOV     R0,#LOW (BRIEF_LED)
    MOVX    @R0,A
    RET
?C0038:
    JNB     FgSequentKey,$ + 6H
```

```
    LJMP    ?C0052
    SETB    FgSequentKey
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep
    CPL     FgSequentFlag
    JNB     FgSequentFlag,?C0040
    MOV     R0,#LOW (SEQUENT_LED)
    MOV     A,#0FFH
    MOVX    @R0,A
    SJMP    ?C0041
?C0040:
    CLR     A
    MOV     R0,#LOW (SEQUENT_LED)
    MOVX    @R0,A
?C0041:
    CLR     FgSingleFlag
    CLR     FgRandomFlag
    CLR     FgBriefFlag
    CLR     A
    MOV     R0,#LOW (SINGLE_LED)
    MOVX    @R0,A
    MOV     R0,#LOW (RANDOM_LED)
    MOVX    @R0,A
    MOV     R0,#LOW (BRIEF_LED)
    MOVX    @R0,A
    RET
?C0042:
    JNB     FgRandomKey,$ + 6H
    LJMP    ?C0052
    SETB    FgRandomKey
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep
    MOV     TMOD,#015H
    MOV     TL0,#0FFH
    MOV     TH0,#0FFH
    SETB    TR0
    CPL     FgRandomFlag
    JNB     FgRandomFlag,?C0044
    MOV     R0,#LOW (RANDOM_LED)
```

```
    MOV     A,#0FFH
    MOVX    @R0,A
    SJMP    ?C0045
?C0044:
    CLR     A
    MOV     R0,#LOW (RANDOM_LED)
    MOVX    @R0,A
?C0045:
    CLR     FgSingleFlag
    CLR     FgSequentFlag
    CLR     FgBriefFlag
    CLR     A
    MOV     R0,#LOW (SINGLE_LED)
    MOVX    @R0,A
    MOV     R0,#LOW (SEQUENT_LED)
    MOVX    @R0,A
    MOV     R0,#LOW (BRIEF_LED)
    MOVX    @R0,A
    RET
?C0046:
    JB      FgBriefKey,?C0052
    SETB    FgBriefKey
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep
    MOV     TMOD,#011H
    MOV     TL0,#0C0H
    MOV     TH0,#063H
    CPL     FgBriefFlag
    JNB     FgBriefFlag,?C0048
    MOV     R0,#LOW (BRIEF_LED)
    MOV     A,#0FFH
    MOVX    @R0,A
    SJMP    ?C0049
?C0048:
    CLR     A
    MOV     R0,#LOW (BRIEF_LED)
    MOVX    @R0,A
?C0049:
    CLR     FgSingleFlag
    CLR     FgSequentFlag
    CLR     FgRandomFlag
    CLR     A
```

```
    MOV     R0,#LOW (SINGLE_LED)
    MOVX    @R0,A
    MOV     R0,#LOW (SEQUENT_LED)
    MOVX    @R0,A
    MOV     R0,#LOW (RANDOM_LED)
    MOVX    @R0,A
    RET
?C0032:
    MOV     R0,#KeyBuffer2
    MOV     @R0,#0FFH
    CLR     FgSingleKey
    CLR     FgSequentKey
    CLR     FgRandomKey
    CLR     FgBriefKey
?C0052:
    RET

    RSEG  ?PR?FunctionKey?INPUT
    USING   0
FunctionKey:
    JB      P2_4,?C0053
    MOV     R0,#FunctionData
    MOV     @R0,#0AH
    RET
?C0053:
    JB      P2_5,?C0055
    MOV     R0,#FunctionData
    MOV     @R0,#0BH
    RET
?C0055:
    JB      P2_6,?C0057
    MOV     R0,#FunctionData
    MOV     @R0,#0CH
    RET
?C0057:
    JB      P2_7,?C0059
    MOV     R0,#FunctionData
    MOV     @R0,#0DH
    RET
?C0059:
    MOV     R0,#FunctionData
    MOV     @R0,#0FFH
?C0061:
    RET
```

```
    RSEG  ?PR?StartHandler?INPUT
    USING   0
StartHandler:
    LCALL   InputStart
    MOV     R0,#KeyData
    MOV     A,@R0
    XRL     A,#01H
    JZ      $ + 5H
    LJMP    ?C0067
    CLR     FgEndMusic
    JNB     FgSingleFlag,?C0063
?C0064:
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Music
    JNB     FgEndMusic,$ + 6H
    LJMP    ?C0067
?C0066:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0064
?C0063:
    JNB     FgSequentFlag,?C0069
?C0070:
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Music
    JNB     FgEndMusic,$ + 6H
    LJMP    ?C0067
?C0072:
    MOV     R0,#MusicNumber
    INC     @R0
    MOV     A,@R0
    SETB    C
    SUBB    A,#01FH
    JC      ?C0073
    MOV     @R0,#01H
?C0073:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0070
```

```
?C0069:
    JNB     FgRandomFlag,?C0075
    CLR     FgStartMusic
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    MOV     R0,#PulseCount
    MOV     A,@R0
    MOV     R6,A
    INC     R0
    MOV     A,@R0
    MOV     R7,A
    LCALL   _srand
?C0076:
    LCALL   rand
    MOV     currentvalue?442,R6
    MOV     currentvalue?442+01H,R7
    MOV     R4,#C0H
    MOV     R5,#064H
    LCALL   ?C?SIDIV
    MOV     currentvalue?442,R4
    MOV     currentvalue?442+01H,R5
    SETB    C
    MOV     A,currentvalue?442+01H
    SUBB    A,#01FH
    MOV     A,currentvalue?442
    XRL     A,#080H
    SUBB    A,#080H
    JNC     ?C0076
    SETB    C
    MOV     A,currentvalue?442+01H
    SUBB    A,#00H
    MOV     A,currentvalue?442
    XRL     A,#080H
    SUBB    A,#080H
    JC      ?C0076
    MOV     A,currentvalue?442+01H
    XRL     A,lastvalue?441+01H
    JNZ     ?C0092
    MOV     A,currentvalue?442
    XRL     A,lastvalue?441
?C0092:
    JZ      ?C0076
    MOV     R7,currentvalue?442+01H
    MOV     R0,#MusicNumber
```

```
    MOV     A,R7
    MOV     @R0,A
    LCALL   _Music
    MOV     lastvalue?441,currentvalue?442
    MOV     lastvalue?441+01H,currentvalue?442+01H
    JB      FgEndMusic,?C0067
?C0080:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0076
?C0075:
    JNB     FgBriefFlag,?C0082
?C0083:
    CLR     FgNextMusic
    CLR     A
    MOV     R0,#MusicLength
    MOV     @R0,A
    SETB    TR0
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Music
    CLR     TR0
    JB      FgEndMusic,?C0067
?C0085:
    MOV     R0,#MusicNumber
    INC     @R0
    MOV     A,@R0
    SETB    C
    SUBB    A,#01FH
    JC      ?C0086
    MOV     @R0,#01H
?C0086:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0083
?C0082:
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Music
    JB      FgEndMusic,?C0067
?C0088:
```

```
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
?C0067:
    RET

    RSEG  ?PR?InputStart?INPUT
    USING   0
InputStart:
    JB      P3_4,?C0089
    MOV     R0,#KeyData
    MOV     @R0,#01H
    RET
?C0089:
    MOV     R0,#KeyData
    MOV     @R0,#0FFH
?C0091:
    RET

    END
```

第 12 章　完整的设计组合

12-1　ADC 按键功能选择

目的：四种功能模式的按键操作也是由 ADC 来侦测，就可以节省 4 支 I/O 引脚，而将 P2.4～P2.7 的 I/O 脚作为每一功能模式的 LED 输出显示状态，则 DAC0～DAC3 PWM 输出作为 LED 显示的动作就可以不用了，因此，SINGLE_LED、SEQUENT_LED、RANDOM_LED、BRIEF_LED 的定义由外部数据存储器、地址从 0X20～0X23 的 PWM 输出改为 CPU 内部的 I/O 定义，分别为 P2.4～P2.7。也就是说将原本四个功能模式的按键输入动作，由 I/O P2.4～P2.7 改成由 AD2 作为输入，而每一个功能模式使能动作的 LED 显示由原来 DAC0～DAC3 PWM 输出（当 DAC 设定 0Xff，则引脚输出“1”电位，当 DAC 清除为 0X00，则引脚输出“0”电位）改成由 I/O P2.4～P2.7 作为输出。虽然硬件的修改不大，只是将侦测功能模式的按键稍为变了一点点，但软件仍需根据需要而进行修改。

12-1-1　软件构建的思维

1. global.c

（1）以 pdata 定义的外部数据存储器 PWM 输出要删除。

（2）按键变量要统一为 KeyBuffer。

理由：

（1）因为 PWM 输出作为 LED 显示改为由 P2.4～P2.7 控制了。

（2）函数被合并，只需使用一个全局变量即可。

2. global.h

PWM 输出的外部声明也要删除及修改全局变量 KeyBuffer。

理由：LED 的输出控制不再由 PWM 控制了，而改成由 I/O P2.4～P2.7 控制，因而，这 4 个 PWM 就不会再使用到了。

3. main.c

将 FunctionHandler( )函数删除。

理由：FunctionHandler( )函数是处理由 I/O P2.4～P2.7 的功能模式的输入及其相

对应的 LED 显示或熄灭的动作，如今已经将这些处理的功能移至 InputHandler( )函数进行处理了，因此要将此 FunctionHandler( )函数删除。

4. input.c

（1）将侦测功能模式的使能动作统一在 ADC 的按键处理程序，因此在 InputKey( )函数中要增加 AD2 的四个功能模式的侦测。

（2）在 InputHandler( )函数里，当侦测到是数字键 0～9，则作数字键的显示功能，否则，表示是功能模式的输入按键，并做对应的 LED 显示或熄灭动作，因此，四个功能模式的按键数值要大于 9。

理由：

（1）数字键 0～9 及功能模式键都是由 ADC 进行转换，因此，可在 ADC 侦测函数(Input Key)中同时侦测。

（2）功能模式键的动作处理与数字键 0～9 的处理也是共同写在 InputHandler( )函数中，会很容易明了当按下数字键 0～9 是做这个回路，而按下功能模式键是由这个回路作处理。再配合 switch...case 的指令结构，使得程序结构条理分明，而且可以共享一个按键变量 KeyBuffer，所以 global.c 及 global.h 都要修改全局变量为 KeyBuffer。

5. input.h

（1）定义四个功能模式 LED 的 I/O 引脚及定义四个功能模式的输入键值，都要大于 9 且不可重复。

（2）删除 FunctionHandler( )函数及 Functionkey( )函数的外部声明。

理由：这两个函数的程序内容都已经被其他函数合并了，已经没有实质函数的定义，当然就不需要作函数外部声明了。

6. initial.c

新增一个 Initial4094( )函数，作为 ic4094 的初始化动作，因而在 PowerOnInit( )函数要调用此函数。

理由：ic4094 的 data pin、clh pin、Strobe pin 及八个输出在一开机时就都清除为 0，以确保当要显示数字时的正确性。

7. initial.h

新增一行 Initial4094( )函数的外部声明。

理由：任一模块下，只要有新增函数，则其对应的包括文件就要增加函数的外部声明，以便日后扩充程序结构而能给其他模块来进行函数调用。

12-1-2 参数的意义说明与使用时机

1. timer.c

FgEndMusic：终止演奏音乐的标志。

- 数据类型：占用一个位的数据类型(bit)。
- 内存类型：变量地址介于 0X20～0X2f 的其中一个位，使用位寻址模式。
- 设定“1”：当按下结束键后，由于需要立即停止音乐演奏，以硬件作为外部中断的输入，请注意是因为有这样的思维所以才规划设计硬件电路以结束键作为外部中断，而不是因为硬件电路这样设计才软件配合。在外部中断程序设定此标志，表示按下结束键，希望可以结束演奏了。
- 清除“0”

  （1）程序一开始进行，则立即在 initial.c 模块下的 initialVariable( )函数中清除。

  （2）只要一按下开始键，须先清除为 0，以避免因上一次按下结束键，而永远设定为“1”而无法演奏音乐。
- 判断：

  （1）在每一首曲目的函数，每一个音符演奏之前都会判断是否要结束演奏而离开此演奏回路。

  （2）每一个功能模式的回路中都要判断此标志，以便结束此功能返回 Main( )函数，而继续等待按下开始键，做不同功能模式的操作。

FgNextMusic：播放简介功能的下一首标志，即演奏时间到的标志而准备演奏下一首曲目。

- 数据类型：占用一个位的数据类型(bit)。
- 内存类型：变量地址介于 0X20～0X2f 的其中一个位，使用位寻址模式。
- 设定“1”：以 Timer0 作为定时器，当播放简介功能的演奏时间（设定为 5 秒）终止时设定，以每 40ms 就将 MusicLength 变量加 1。当判断 MusicLength 变量内容等于 TIME_5SEC 常数的内容，就表示五秒的时间已经到了，则立即设定 FgNextMusic=“1”，表示要换下一首曲目，而达到此播放简介功能的目的。从设定的曲目开始，每演奏五秒钟就换下一首，如此一直重复直到按下结束键为止。
- 清除“0”

  （1）按下开始键。

  （2）播放简介功能模下，在调用演奏音乐之前必须先清除，即清除上一首五秒时间到的设定，否则，只能演奏完一首五秒后，在 Timer0ISR( )函数中

会设定 FgNextMusic 标志变量。由于 FgNextMusic 永远为“1”，因此再调用 Music( )函数，在实际演奏曲目的函数内，因判断到 FgNextMusic=“1”，即返回到原调用程序，而造成无法继续演奏下一首的 BUG。

- 判断：由模块 Music1.c～Music31.c，即实际演奏音乐的函数：Music1( )函数～Music31( )函数内，在发出每一个音符前会先判断此标志是否为“1”，是，则不演奏立即返回原调用程序，否，继续发出每一个音符。

针对 FgEndMusic 和 FgNextMusic 两个标志变量在程序中设定、清除、判断所造成的影响，现分述如下。

（1）将 FgEndMusic=0 及 FgNextMusic=0 写在 Music( )函数的进入点。

想法：在开始演奏音乐的函数前先清除之，达到按下结束键立即终止演奏的目的。

盲点：

① 已经在演奏中，因想要结束而按下结束键，则 Music( )函数会立即退出，紧接着判断 FgEndMusic=1 条件的成立，因而返回至 Main( )函数而达到此结束键的功能，所以正常演奏下此流程正常不会出错。

② 当 Music.c 模块下的 Music( )函数，其自变量并没有符合所有 case 的条件值则会执行 default 语句，即执行 break 指令，而立即返回原调用程序。此软件的动作非常快速，在几十个 us 的时间即可完成，紧接着就判断 FgEndMusic 是否为“1”，由于在 Music( )函数中就已经清除 FgEndMusic 变量了，因而此条件不会成立，继续将 MusicNumber 递增并延迟一秒后，一直反复演奏歌曲，即使在延迟一秒的回路内按下结束键，则 FgEndMusic 变量被设定为“1”，但只要一执行 Music( )函数又被清除为 0 了，因而无法返回原调用程序，程序就一直在此功能模式的回路内绕圈子而形成死机，除非程序在执行 Music( )函数，利用这几十 us 的时间按下结束键，才不会死机，但这种机会是微乎其微。

（2）将 FgEndMusic=0 及 FgNextMusic=0 写在按下开始键之后的语句中，而在播放简介功能模式不设定 FgNextMusic=0。

想法：按下“开始键”，就清除这两个标志，以便结束键的设定而能达到要求。

盲点：在播放简介功能中，每演奏完五秒后就会在 Timer0ISR( )函数中设定 FgNextMusic，如果在演奏下一首之前不在播放简介的回路中将 FgNextMusic 标志清除掉，则下一首将无法正确演奏而只能演奏所设定的曲目而已。

FgstartMusic：随机选曲以开始键作为 COUNTER0 中断的输入，以弹跳波的个数作为随机函数的种子数而产生随机曲目，以此标志变量作为按下起始键的根据。

- 数据类型：占用一个位的数据类型(bit)。

- 内存类型：变量地址介于 0X20～0X2f 的其中一个位，使用位寻址模式。
- 设定“1”：当按下开始键，表示要开始演奏了。
- 清除“0”：①initial.c 的 Initial Variable( )函数里清除。

②在随机选曲的功能模式的回路内

此标志的用意被在 Timer0ISR( )函数内的 KeyData=START_KEY 的指令所取代了。主要是当按下开始键，则在随机选曲的功能模式也要立即动作即可，由于在 StartHandler( )函数中会判断 KeyData 变量是否等于 START_KEY 而作各功能模式的动作，因此，无需判断 FgStartMusic 标志了。

MusicLength：播放简介的 5 秒时间变量。

- 数据类型：占用一个字节的无符号字符类型(unsigned char)。
- 内存类型：变量地址介于 0X00～0Xff 之间，使用间接寻址模式(idata)。
- 写入：在 Timer0ISR( )函数内，每经过 40ms 就一直加 1，为了是要判断时间是否到了 5 秒了。
- 清除：在执行播放简介功能模式时，起动定时器 0 必须先清除，以确保时间从 0 开始计时至 5 秒。
- 判断：在 Timer0ISR( )函数中，如果此时间变量的内容等于 5 秒，则设定播放简介的演奏时间到了，可以进行下一首的演奏。

PluseCount：在随机选曲功能模式下，当按下开始键的弹跳波个数，以此个数作为随机函数种子，当随机函数的种子数不一样，其产生的随机数就会不一样，Keil_C 函数库有 rand( )函数及 srand( )函数，因此可利用它来产生随机数而达到目的。

- 数据类型：占用一个字节的无符号字符类型(unsigned char)。
- 内存类型：变量地址介于 0X00～0Xff 之间，使用间接寻址模式(idata)。
- 写入：以 T0 引脚作为外部脉冲的中断输入，利用中断特性可计数弹跳波的个数，即每产生一个负脉冲就累加 1，而且每按下开始键就一直累加，可令任意时候不断地按开始键也会使得此变量内容都不一样。
- 清除：只有在开机后，InitialVariable( )函数将其清除。
- 自变量：作为 srand( )随机种子函数的自变量内容，可令随机函数 rand( )产生不同的随机数。

2. input.c

FgSinglekey：持续按下单曲循环键，也只会动作一次。

- 数据类型：占用一个位的数据类型(bit)。
- 内存类型：变量地址介于 0X20～0X2f 的其中一个位，使用位寻址模式。

- 设定 1：当按下单曲循环键时，要进行设定，以避免持续按键就一直反复动作。
- 清除 0：当没有按下任意键时，要进行清除，否则，下一次再按键时也不会再动作了。
- 判断：按下单曲循环键后，必须先判断是否为 0 才能做动作，否则就退出回路以达到持续按键也只动作一次的功能。

FgSequentKey：持续按下顺序播放键，也只会动作一次。

- 数据类型：占用一个位的数据类型(bit)。
- 内存类型：变量地址介于 0X20～0X2f 的其中一个位，使用位寻址模式。
- 设定“1”：按下顺序播放键时，以避免持续按键就一直反复动作。
- 清除“0”：只要没有按下任意键，即表示 STAND BY 状态就清除。
- 判断：执行顺序播放功能模式的第一个动作，即先判断标志为“0”才能进行动作。

FgRandomkey：持续按下随机选曲键，也只会动作一次。

- 数据类型：占用一个位的数据类型(bit)。
- 内存类型：变量地址介于 0X20～0X2f 的其中一个位，使用位寻址模式。
- 设定“1”：当按下随机选曲键时，以避免持续按键就一直反复动作。
- 清除“0”：当没有按下任意键时要清除，否则，下一次再按随机选曲键也不会再动作了。
- 判断：按下随机选曲键后，必须先判断是否为“0”才能做动作，否则就退出回路，以达到持续按键也只动作一次的功能。

FigBriefKey：持续按下播放简介键，也只会动作一次。

- 数据类型：占用一个位的数据类型(bit)。
- 内存类型：变量地址介于 0X20～0X2f 的其中一个位，使用位寻址模式。
- 设定“1”：当按下播放简介键时，以避免持续按键就一直反复动作。
- 清除“0”：当没有按下任意键时要清除，否则，下一次再按播放简介键也不会再动作了。
- 判断：按下“播放简介键”后，必须先判断是否为“0”才能做动作，否则就退出回路，以达到持续按键也只动作一次的功能。

FgSingleFlag：单曲循环动作的标志，当按下单曲循环键标志设定，表示此功能将被执行，再按一次此标志清除之，以解除此功能模式。

- 数据类型：占用一个位的数据类型(bit)。
- 内存类型：变量地址介于 0X20～0X2f 的其中一个位，使用位寻址模式。
- 清除“0”：当按下顺序播放键、随机选曲键及播放简介键要将此单曲循环

的功能模式的标志清除，也就是当按下此功能键时，则必将其他功能键的标志全部清除。

- 运算：每当按下单曲循环键，则将此标志作反相运算，形成当按下第一次则此标志会被设定，再按下第二次时，则此标志反而会清除，按第三次又会被设定，按第四次将变成清除，这样，反复地循环设定、清除。
- 判断：判断当此标志为“1”，则将对应的 LED 点亮，如果为“0”，则将 LED 熄灭。

FgSequentFlag：顺序播放动作的标志，当按下顺序播放键，此标志进行设定，表示此功能将被执行，再按一次此标志清除之，以解除此功能模式。

- 数据类型：占用一个位的数据类型(bit)。
- 内存类型：变量地址介于 0X20～0X2f 的其中一个位，使用位寻址模式。
- 清除“0”：当按下单曲播放键、随机选曲键及播放简介键，要将此顺序播放的功能模式的标志清除，也就是当按下此功能键时，则应将其他功能键的标志全部清除。
- 运算：每当按下“顺序播放键”则将此标志作反相运算，形成当按下第一次则此标志会被设定，再按下第二次时，则此标志反而会清除，按第三次又会被设定，按第四次将变成清除，如此，反复的循环设定、清除。
- 判断：判断当此标志为“1”，则将对应的 LED 点亮，如果为“0”则将 LED 熄灭。

FgRandomFlag：随机选曲动作的标志，当按下随机选曲键此标志设定，表示此功能将被执行，再按一次此标志清除，以解除此功能模式。

- 数据类型：占用一个位的数据类型(bit)。
- 内存类型：变量地址介于 0X20～0X2f 的其中一个位，使用位寻址模式。
- 清除“0”：当按下顺序播放键、单曲循环键及播放简介键要将此随机选曲的功能模式的标志清除，也就是当按下此功能键时，则必须将其他功能键的标志全部清除。
- 运算：每当按下随机播放键，则将此标志作反相运算，形成当按下第一次则此标志会被设定，再按下第二次时，则此标志反而会清除，按第三次又会被设定，按第四次将变成清除，这样反复的循环设定、清除。
- 判断：判断当此标志为“1”则将对应的 LED 点亮，如果为“0”则将 LED 熄灭。

FgBriefFlag：播放简动作的标志，当按下播放简介键此标志设定，表示此功能将被执行，再按一次此标志清除，以解除此功能模式。

- 数据类型：占用一个位的数据类型(bit)。

- 内存类型：变量地址介于 0X20～0X2f 的其中一个位，使用位寻址模式。
- 清除“0”：当按下顺序播放键、随机选曲键及单曲循环键，将此播放简介的功能模式的标志清除，也就是当按下此功能键时，则必须将其他功能键的标志全部清除。
- 运算：每当按下播放简介键，则将此标志作反相运算，形成当按下第一次则此标志会被设定，再按下第二次时，则此标志反而会清除，按第三次又会被设定，按第四次将变成清除，这样，反复地循环设定、清除。
- 判断：判断当此标志为“1”则将对应的 LED 点亮，如果为“0”，则将 LED 熄灭。

12-1-3　软件的解决方法

1. global.c

```
pdata unsigned char SIGLE_LED       _at_       0X20;
pdata unsigned char SEQUENT_LED   _at_      0X21;
pdata unsigned char RANDOM_LED      _at_       0X22;
pdata unsigned char BRIEF_LED       _at_       0X23;
```

将上面部分全部删除，因为上述第一到四列的意义，是将 SINGLE_LED、SEQUENT_LED、RANDOM_LED、BRIEF_LED 符号定义外部数据存储器，并占用一个字节，其内容为 0～255，地址是 0X20～0X23，也就是 DAC0～DAC3，即 CPU 引脚的第 3、2、1 及 38 支脚。

2. global.h

由于 global.c 外部数据存储器的定义要删除，因此，其对应的包括文件 global.h 中的外部声明也要删除，也就是将下列四列全部删处。

```
extern pdata unsigned char SINGLE_LED;
extern pdata unsigned char SEQENT_LED;
extern pdata unsigned char PANDOM_LED;
extern pdata unsigned char BRIEF_LED;
```

3. main.c

功能模式由 ADC 侦测并一起在 InputHandler( )函数中进行判断，已经不是由单纯的 I/O P2.4～P2.7 做功能模式的判断，因而，在 Main( )函数的 FuntionHandler( ) 函数调用就可以被省略了。

4. input.c

将 FunctionHandler( )函数的按键侦测功能一起写在 InputHandler( )函数里，因而，按键变量可以共享为一个，即 KeyBuffer1、KeyBuffer2 可以统一使用 KeyBuffer 的变量。当全局变量增加（修改或删除）时，在整个软件结构下，必须做以下的动作。

（1）在 global.c 的模块下，新增一个变量，例如：

- 标志变量 FgTest，是在 Key.c 的模块下被用到，则在 global.c 需要增加 Bool FgTest;的语句，而写在 Key 模块下以方便阅读及容易了解此变量的使用状态。
- 占用一个字节的变量 KeyTest1 也是在 Key 模块下使用，则在 global.c 下需增加 Byte IDATA KeyTest1;的语句，也是写在 Key 模块下，其中 Byte 是指 unsigned char 的内存类型声明。
- 占用二个字节的变量 KeyTest2，则在 global.c 需要增加 Word IDATA KeyTest2，则在 global.c 必须增加 Word IDATA KeyTest2; 的语句，其中 Word 是指 unsigned int 的内存类型声明。
- 占用十个字节的数组变量 KeyTest3，则在 global.c 需要增加 Byte IDATA KeyTest3[10]; 的语句，也就是程序中可任意存取 KeyTest3[0]～KeyTest3[9] 的变量，其内容范围为 0～255，IDATA 为间接寻址法，记忆空间为 0Xff 的地址。

（2）在 global.h 模块下做外部声明，以利于其他模块存取，上述的例子如下：

```
extern Bool FgTest;
extern Byte IDATA       KeyTest1;
extern Word IDATA       KeyTest2;
extern Byte IDATA       KeyTest3[10];
```

也就是在变量数据类型的前端多写了一个关键词：extern 即可，其意义为此变量的声明并不是在本模块之内，但当本模块要存取此变量时，则必须声明为外部变量才可以。

（3）initial.c 模块下的 InitialVariable( )函数要设定此变量的内容，或清除为 0。一般为清除，以确保变量的起始值从 0 开始。

（4）要思考此变量要在哪里读取？在哪里写入及在哪里判断？什么时候要清除？哪一种状况要设定数值等。

在 input.c 的模块下除了要判断 0～9 的按键数值外，也要个别判断四种功能模式键值，以 switch...case 的指令结构来作为功能模式键输入的动作，会变得简单明了而且程序结构清晰完整，当侦测到功能模式键输入时，则需将相对应的 LED 点亮，

以表示此模式动作中，并设定其对应的标志变量以作为当按下起始键的实际执行动作的根据。而其他模式的标志变量清除之，及其他的 LED 也清除使其熄灭，也就是只要一种功能模式使能，则其他的功能模式则除能，程序如下：

```
/*****************************/
void InputHandler(void)
{
   //sel play no#
   InputKey( );

   if ( KeyBuffer != KeyData )
   {
      KeyBuffer = KeyData;
      ScanKeyCounter = 10;
   }
   else if( ScanKeyCounter != 0 )
   {
      ScanKeyCounter--;
      DelayX1ms(1);
   }
   else if ( KeyBuffer != NO_KEY )
   {
      //KeyBuffer=0~9
      if ( KeyBuffer<=9 )
      {
         if ( FgKeyFlag==0 )
         {
            FgKeyFlag = 1;
            Beep(1,17,10);

            MusicNumber=LastBuffer*10+KeyBuffer;
            Disp7Seg(MusicNumber);
            LastBuffer=KeyBuffer;
         }
      }
      //KeyBuffer=10~13
      else
      {
         switch ( KeyBuffer )
         {
          case SINGLE_KEY :
             if ( FgSingleKey==0 )
             {
                FgSingleKey = 1;
```

```
        Beep(1,17,10);
        FgSingleFlag= !FgSingleFlag;
        if ( FgSingleFlag==1 )
          SINGLE_LED=1;
        else
          SINGLE_LED=0;

        //other key off
        FgSequentFlag =0;
        FgRandomFlag=0;
        FgBriefFlag=0;
        SEQUENT_LED=0;
        RANDOM_LED=0;
        BRIEF_LED=0;
      }
      break;
    case SEQUENT_KEY :
      if ( FgSequentKey==0 )
      {
        FgSequentKey = 1;
        Beep(1,17,10);
        FgSequentFlag= !FgSequentFlag;
        if ( FgSequentFlag==1 )
          SEQUENT_LED=1;
        else
          SEQUENT_LED=0;

        FgSingleFlag=0;
        FgRandomFlag=0;
        FgBriefFlag=0;
        SINGLE_LED=0;
        RANDOM_LED=0;
        BRIEF_LED=0;
      }
      break;
    case RANDOM_KEY :
      if ( FgRandomKey==0 )
      {
        FgRandomKey = 1;
        Beep(1,17,10);

        TMOD= 0x15;          //set timer1,count0 mode
        TR0 = 1;             //start count0
        TL0 = 0xff;          //count 1 times
        TH0 = 0xff;
```

```
            FgRandomFlag= !FgRandomFlag;
            if ( FgRandomFlag==1 )
              RANDOM_LED=1;
            else
              RANDOM_LED=0;

            FgSingleFlag=0;
            FgSequentFlag=0;
            FgBriefFlag=0;
            SINGLE_LED=0;
            SEQUENT_LED=0;
            BRIEF_LED=0;
          }
          break;
        case BRIEF_KEY :
          if ( FgBriefKey==0 )
          {
            FgBriefKey = 1;
            Beep(1,17,10);

            TMOD= 0x11;        //set timer1,timer0 mode
            //CLOCK_40MS=(65536 - 40000)
            TL0 = CLOCK_40MS & 0xff;
            TH0 = CLOCK_40MS >> 8;

            FgBriefFlag= !FgBriefFlag;
            if ( FgBriefFlag==1 )
              BRIEF_LED=1;
            else
              BRIEF_LED=0;

            FgSingleFlag=0;
            FgSequentFlag=0;
            FgRandomFlag=0;
            SINGLE_LED=0;
            SEQUENT_LED=0;
            RANDOM_LED=0;
          }
          break;
        default :
          break;
      }
    }
  }
```

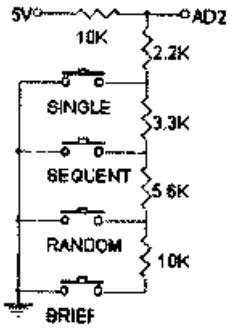
```
        //KeyBuffer=NO_KEY
        else
        {
           KeyBuffer = NO_KEY;
           FgKeyFlag = 0;

           FgSingleKey= 0;
           FgSequentKey= 0;
           FgRandomKey= 0;
           FgBriefKey= 0;
        }
    }
```

因为，功能模式键由 AD2 作为输入，由硬件电路的电阻分压通过由 ADC 转换可得知其转换数值为 SIGLE_KEY 等于 12。

由硬件电路得知当按下 SINGLE_KEY，则输入至 AD2 的直流电压为 5V×(2.2K/(10K+2.2K))=0.9V，由于此 AD2 的分辨率为 6bit，其最大数值为 63，也就是电压为 5/63=79.36mv，因此当按下 SINGLE_KEY 其直流电压为 0.9V，其转换数值将会为 0.9V/79.36mv=11.34，取其整数等于 11，因为暂存转换后变量声明为一个 Byte 的无符号字符变量，即 Byte 记忆类型声明。

同理，当按下 SEQUENT_KEY，则输入至 AD2 的直流电压为 5V×((2.2K+3.3K/(10K+2.2K+3.3K))=1.774V，其转换数值为 1.774V/79.36mv=22.3，取其整数等于 22，当按下 RANDOM_KEY，则输入至 AD2 的直流电压为，5V×((2.2K+3.3K+5.6K)/(10K+2.2K+3.3K+5.6K))=2.63V，其转换数值为 2.63V/79.36mv=33.1，取其整数等于 33，而按下 BRIEF_KEY，则输入至 AD2 的直流电压为 5V×((2.2K+3.3K+5.6K+10K)/(10K+2.2K+3.3K+5.6K+10K))=3.392V，其转换数值为 3.392V/79.36mv=42.7，取其整数并四舍五入等于 43。

上述是根据理论的方法当按下键值而求出转换后的数值，但实际上电阻零件会有误差，因而造成电阻分压时会有一点点的差别，但并不会太大，程序设计时要以实际电路所量测的电压为基准，其判断键值也要以实际的为准，才更容易正确。按键检测电路如下：



程序如下：

```
void InputKey(void)
{
    Byte  keytmp;

    KeyData  = NO_KEY;        //NO_KEY=0

    XFR_ADC=0x81;             //adc channel=1
    DelayX1ms(2);
    keytmp = XFR_ADC & 0x3f;     //6 bit  2^6=0x3f
    if ( keytmp < (12+4) )       KeyData = NUMBER1_KEY;
    else if ( keytmp < (23+4) )      KeyData = NUMBER2_KEY;
    else if ( keytmp < (34+4) )      KeyData = NUMBER3_KEY;
    else if ( keytmp < (44+4) )      KeyData = NUMBER4_KEY;
    else if ( keytmp < (50+4) )      KeyData = NUMBER5_KEY;

    XFR_ADC=0x82;         //adc channel=2
    DelayX1ms(2);
    keytmp = XFR_ADC & 0x3f;
    if ( keytmp < (12+4) )       KeyData = NUMBER6_KEY;
    else if ( keytmp < (23+4) )      KeyData = NUMBER7_KEY;
    else if ( keytmp < (34+4) )      KeyData = NUMBER8_KEY;
    else if ( keytmp < (44+4) )      KeyData = NUMBER9_KEY;
    else if ( keytmp < (50+4) )      KeyData = NUMBER0_KEY;

    XFR_ADC=0x84;         //adc channel=3
    DelayX1ms(2);
    keytmp = XFR_ADC & 0x3f;
    if ( keytmp < (12+4) )       KeyData = SINGLE_KEY;
    else if ( keytmp < (23+4) )      KeyData = SEQUENT_KEY;
    else if ( keytmp < (34+4) )      KeyData = RANDOM_KEY;
    else if ( keytmp < (44+4) )      KeyData = BRIEF_KEY;

    XFR_ADC=0x00;         //disable adc
}
```

5. input.h

在 input.c 模块下已经将功能模式的按键侦测程序改由 InputHandler( )函数侦测了，因此要将由 P2.4~P2.7 作为功能模式的按键输入的 FuncitonHandler( )函数和 FunctionKey( )函数删除，也无需作外部函数声明了。

但当功能模式设定时其对应的 LED 要动作，即以 SINGLE_KEY 对应 SINGLE_LED，而由 I/O P2.4 作为输出。当 SINGLE_KEY 按下，I/O P2.4 输出“1”

电位则 LED 亮，再按一次 SINGLE_KEY，I/O P2.4 输出“0”电位则 LED 熄，同理，当按下 SEQUENT_KEY 则对应的 SEQUENT_LED 由 I/O P2.5 输出“1”电位，则 LED 亮，再按一次 SEQUENT_LED 由 I/O P2.5 输出“0”电位，则 LED 熄灭；按下 RANDOM_KEY，则对应的 RANDOM_LED 由 I/O P2.6 输出“1”电位，则 LED 亮，再按一次 RANDOM_KEY，则 I/O P2.6 输出“0”电位，则 LED 熄灭；按下 BRIEF_KEY，则对应的 BRIEF_LED 由 I/O P2.7 输出“1”电位，则 LED 亮，再按一次 BRIEF_LED 则 I/O P2.7 输出“0”电位，则 LED 熄灭。因而，必须要在此定义 I/O 如下：

```
#define SINGLE_LED P2_4
#define SEQUENT_LED P2_5
#define RANDOM_LED P2_6
#cefine BRIEF _LED P2_7
```

定义功能模式的键值如下：

```
#define SINGLE_KEY 10
#define SEQUENT_KEY 11
#define RANDOM_KEY 12
#define BRIEF_KEY 13
```

由于数字键 0～9 及功能模式键是写在同一个函数下，因此定义键值必须独立，不能和数字键重复，否则程序会误判，因而功能模式键值从 10 开始定义并递增。而当没有按下键，其定义的数值不可以为 0X00，因为已经有数字键 0 了，如果定义为 0X00，则按下数字键 0，程序会误判为 NO_KEY 表示，没有按下，则数字键 0 永远不会被接受，因此，定义未按下任意键的数值为 0Xff，以便区隔，input.h 的包括文件如下：

```
#ifndef  _INPUT_H
#define  _INPUT_H

#define  START_IO        P3_4
#define  SINGLE_LED      P2_4
#define  SEQUENT_LEDP2_5
#define  RANDOM_LED P2_6
#define  BRIEF_LED       P2_7

#define  NO_KEY          0xff
#define  START_KEY       1

#define  NUMBER0_KEY0
#define  NUMBER1_KEY1
```

```
#define  NUMBER2_KEY2
#define  NUMBER3_KEY3
#define  NUMBER4_KEY4
#define  NUMBER5_KEY5
#define  NUMBER6_KEY6
#define  NUMBER7_KEY7
#define  NUMBER8_KEY8
#define  NUMBER9_KEY9
#define  SINGLE_KEY      10
#define  SEQUENT_KEY11
#define  RANDOM_KEY 12
#define  BRIEF_KEY       13

extern void InputHandler(void);
extern void InputKey(void);
extern void StartHandler(void);
extern void InputStart(void);

#endif
```

其中 P3_4 就是 I/O P3.4，P2_4 就是 I/O P2.4，P2_5 就是 I/O P2.5，P2_6 就是 I/O P2.6，P2_7 就是 I/O P2.7 在 mtv212m.h 包括文件都有定义，请参照。

6. initial.c

开机送电后，要先将 ic4094 的八个输出清除，并将 clock pin、data pin 和 stroke pin 先清除为 0，以确保七段显示器可以正常显示出数字，而四个功能模式的 LED 要清除为 0，七段显示器也要清除为“00”，其程序如下：

```
/*******************************/
void Initial4094(void)
{
   DATA_PIN   =0;    //P1.1,IC4094 data
   CLK_PIN    =0;    //P1.2,IC4094 clock
   STROBE_PIN =0;    //P1.0,IC4094 strobe
   Output4094(0x00);    //clear IC4094 output
}

/*******************************/
void InitialCpuIO(void)
{
   SPEAKER=0;

   //display "00"
   Disp7Seg(0x00);
```

```
    //i/o:P2.4~P2.7,led on/off
    SINGLE_LED=0;
    SEQUENT_LED=0;
    RANDOM_LED=0;
    BRIEF_LED=0;
}
```

7. initial.h

新增一个清除 ic4094 的函数，函数名称为 Initial4094( )，因而在包括文件里也要声明此函数为外部函数，以便其他模块调用。虽然本范例 Initial4094( )函数并未给其他模块的函数调用，但仍要养成习惯：只要在模块中有新增函数，则要将此新增函数在相对应的包括文件内做外部函数的声明。如果有删除或使用条件式编译指令使其函数不经过编译，则在包括文件内也须删除此外部声明函数或使其变成注释（使用双斜线//，则编译器就不会编译了）。

汇编程序如下：

1. global.a51

```
$NOMOD51

NAME    GLOBAL

$INCLUDE (mtv212m.inc)

?BI?GLOBAL          SEGMENT BIT
?ID?GLOBAL          SEGMENT IDATA
    PUBLIC  XFR_XBANK
    PUBLIC  XFR_OPTION2
    PUBLIC  XFR_OPTION1
    PUBLIC  XFR_PADMOD3
    PUBLIC  XFR_PADMOD2
    PUBLIC  XFR_PADMOD1
    PUBLIC  XFR_WDT
    PUBLIC  XFR_ADC
    PUBLIC  SoundLongCount
    PUBLIC  Period
    PUBLIC  Tempo
    PUBLIC  MusicNumber
    PUBLIC  ScanKeyCounter
    PUBLIC  LastBuffer
    PUBLIC  KeyBuffer
    PUBLIC  KeyData
```

```
 PUBLIC  FgKeyFlag
 PUBLIC  FgBriefFlag
 PUBLIC  FgRandomFlag
 PUBLIC  FgSequentFlag
 PUBLIC  FgSingleFlag
 PUBLIC  FgBriefKey
 PUBLIC  FgRandomKey
 PUBLIC  FgSequentKey
 PUBLIC  FgSingleKey
 PUBLIC  PulseCount
 PUBLIC  MusicLength
 PUBLIC  FgStartMusic
 PUBLIC  FgNextMusic
 PUBLIC  FgEndMusic

 XSEG  AT  010H
    XFR_ADC:   DS   1

 XSEG  AT  018H
    XFR_WDT:   DS   1

 XSEG  AT  030H
XFR_PADMOD1:   DS   1

 XSEG  AT  031H
XFR_PADMOD2:   DS   1

 XSEG  AT  032H
XFR_PADMOD3:   DS   1

 XSEG  AT  033H
XFR_OPTION1:   DS   1

 XSEG  AT  034H
XFR_OPTION2:   DS   1

 XSEG  AT  035H
  XFR_XBANK:   DS   1

 RSEG  ?BI?GLOBAL
  FgEndMusic:   DBIT   1
 FgNextMusic:   DBIT   1
FgStartMusic:   DBIT   1
 FgSingleKey:   DBIT   1
FgSequentKey:   DBIT   1
```

```
    FgRandomKey:   DBIT   1
     FgBriefKey:   DBIT   1
   FgSingleFlag:   DBIT   1
  FgSequentFlag:   DBIT   1
   FgRandomFlag:   DBIT   1
    FgBriefFlag:   DBIT   1
      FgKeyFlag:   DBIT   1

    RSEG  ?ID?GLOBAL
   MusicLength:   DS   1
    PulseCount:   DS   2
       KeyData:   DS   1
     KeyBuffer:   DS   1
    LastBuffer:   DS   1
 ScanKeyCounter:   DS   1
   MusicNumber:   DS   1
         Tempo:   DS   2
        Period:   DS   2
 SoundLongCount:   DS   2

    END
```

2. main.a51

```
$NOMOD51

NAME    MAIN

$INCLUDE (mtv212m.inc)

?PR?Main?MAIN       SEGMENT CODE
    EXTRN   CODE (PowerOnInit)
    EXTRN   CODE (InputHandler)
    EXTRN   CODE (StartHandler)
    EXTRN   CODE (?C_STARTUP)
    PUBLIC  Main

    RSEG  ?PR?Main?MAIN
    USING   0
Main:
    LCALL   PowerOnInit
?C0001:
    LCALL   InputHandler
    LCALL   StartHandler
    SJMP    ?C0001
```

```
    RET

    END
```

3. initial.a51

```
$NOMOD51

NAME    INITIAL

$INCLUDE (mtv212m.inc)

?PR?PowerOnInit?INITIAL                 SEGMENT CODE
?PR?InitialCpu?INITIAL                  SEGMENT CODE
?PR?Initial4094?INITIAL                 SEGMENT CODE
?PR?InitialCpuIO?INITIAL                SEGMENT CODE
?PR?InitialVariable?INITIAL             SEGMENT CODE
    EXTRN   BIT (FgEndMusic)
    EXTRN   BIT (FgStartMusic)
    EXTRN   IDATA (PulseCount)
    EXTRN   IDATA (KeyData)
    EXTRN   IDATA (LastBuffer)
    EXTRN   XDATA (XFR_ADC)
    EXTRN   XDATA (XFR_WDT)
    EXTRN   XDATA (XFR_PADMOD1)
    EXTRN   XDATA (XFR_PADMOD2)
    EXTRN   XDATA (XFR_PADMOD3)
    EXTRN   XDATA (XFR_OPTION1)
    EXTRN   XDATA (XFR_OPTION2)
    EXTRN   XDATA (XFR_XBANK)
    EXTRN   CODE (_Disp7Seg)
    EXTRN   CODE (_Output4094)
    PUBLIC  InitialVariable
    PUBLIC  InitialCpuIO
    PUBLIC  Initial4094
    PUBLIC  InitialCpu
    PUBLIC  PowerOnInit

    RSEG  ?PR?PowerOnInit?INITIAL
    USING   0
PowerOnInit:
    LCALL   InitialCpu
    LCALL   Initial4094
    LCALL   InitialCpuIO
    LCALL   InitialVariable
```

```
    RET

    RSEG  ?PR?InitialCpu?INITIAL
    USING   0
InitialCpu:
    CLR     A
    MOV     IE,A
    MOV     PSW,A
    MOV     IP,#0BH
    MOV     TMOD,#011H
    CLR     TR0
    CLR     TR1
    SETB    IT0
    MOV     TL0,#0C0H
    MOV     TH0,#063H
    SETB    EX0
    SETB    ET1
    SETB    ET0
    MOV     R0,#LOW (XFR_ADC)
    MOV     A,#080H
    MOVX    @R0,A
    MOV     R0,#LOW (XFR_WDT)
    MOV     A,#040H
    MOVX    @R0,A
    MOV     R0,#LOW (XFR_PADMOD1)
    MOV     A,#07H
    MOVX    @R0,A
    CLR     A
    MOV     R0,#LOW (XFR_PADMOD2)
    MOVX    @R0,A
    MOV     R0,#LOW (XFR_PADMOD3)
    MOVX    @R0,A
    MOV     R0,#LOW (XFR_OPTION1)
    MOV     A,#0C0H
    MOVX    @R0,A
    CLR     A
    MOV     R0,#LOW (XFR_OPTION2)
    MOVX    @R0,A
    MOV     R0,#LOW (XFR_XBANK)
    MOVX    @R0,A
    SETB    EA
    RET

    RSEG  ?PR?Initial4094?INITIAL
    USING   0
```

```
Initial4094:
    CLR     P1_1
    CLR     P1_2
    CLR     P1_0
    CLR     A
    MOV     R7,A
    LCALL   _Output4094
    RET

    RSEG  ?PR?InitialCpuIO?INITIAL
    USING   0
InitialCpuIO:
    CLR     P3_1
    CLR     A
    MOV     R7,A
    LCALL   _Disp7Seg
    CLR     P2_4
    CLR     P2_5
    CLR     P2_6
    CLR     P2_7
    RET

    RSEG  ?PR?InitialVariable?INITIAL
    USING   0
InitialVariable:
    MOV     R0,#KeyData
    MOV     @R0,#0FFH
    CLR     A
    MOV     R0,#LastBuffer
    MOV     @R0,A
    CLR     FgEndMusic
    CLR     FgStartMusic
    MOV     R0,#PulseCount
    MOV     @R0,A
    INC     R0
    MOV     @R0,A
    RET

    END
```

4. input.a51

```
$NOMOD51

NAME    INPUT
```

```
$INCLUDE (mtv212m.inc)

?PR?InputHandler?INPUT                    SEGMENT CODE
?PR?InputKey?INPUT                        SEGMENT CODE
?DT?InputKey?INPUT                        SEGMENT DATA OVERLAYABLE
?PR?StartHandler?INPUT                    SEGMENT CODE
?DT?StartHandler?INPUT                    SEGMENT DATA OVERLAYABLE
?PR?InputStart?INPUT SEGMENT CODE
    EXTRN   BIT (FgEndMusic)
    EXTRN   BIT (FgNextMusic)
    EXTRN   BIT (FgStartMusic)
    EXTRN   IDATA (MusicLength)
    EXTRN   IDATA (PulseCount)
    EXTRN   BIT (FgSingleKey)
    EXTRN   BIT (FgSequentKey)
    EXTRN   BIT (FgRandomKey)
    EXTRN   BIT (FgBriefKey)
    EXTRN   BIT (FgSingleFlag)
    EXTRN   BIT (FgSequentFlag)
    EXTRN   BIT (FgRandomFlag)
    EXTRN   BIT (FgBriefFlag)
    EXTRN   BIT (FgKeyFlag)
    EXTRN   IDATA (KeyData)
    EXTRN   IDATA (KeyBuffer)
    EXTRN   IDATA (LastBuffer)
    EXTRN   IDATA (ScanKeyCounter)
    EXTRN   IDATA (MusicNumber)
    EXTRN   XDATA (XFR_ADC)
    EXTRN   CODE (_DelayX10ms)
    EXTRN   CODE (_DelayX1ms)
    EXTRN   CODE (_Music)
    EXTRN   CODE (_Beep)
    EXTRN   CODE (_Disp7Seg)
    EXTRN   CODE (rand)
    EXTRN   CODE (_srand)
    EXTRN   CODE (?C?SIDIV)
    PUBLIC  InputStart
    PUBLIC  StartHandler
    PUBLIC  InputKey
    PUBLIC  InputHandler

    RSEG  ?DT?InputKey?INPUT
?InputKey?BYTE:
    keytmp?140:   DS   1
```

```
    RSEG  ?DT?StartHandler?INPUT
?StartHandler?BYTE:
  lastvalue?241:   DS   2
currentvalue?242:   DS   2

    RSEG  ?PR?InputHandler?INPUT
    USING   0
InputHandler:
    LCALL   InputKey
    MOV     R0,#KeyData
    MOV     A,@R0
    MOV     R7,A
    MOV     R0,#KeyBuffer
    XRL     A,@R0
    JZ      ?C0001
    MOV     A,R7
    MOV     @R0,A
    MOV     R0,#ScanKeyCounter
    MOV     @R0,#0AH
    RET
?C0001:
    MOV     R0,#ScanKeyCounter
    MOV     A,@R0
    JZ      ?C0003
    DEC     @R0
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    RET
?C0003:
    MOV     R0,#KeyBuffer
    MOV     A,@R0
    MOV     R7,A
    CPL     A
    JNZ     $ + 5H
    LJMP    ?C0005
    MOV     A,R7
    SETB    C
    SUBB    A,#09H
    JNC     ?C0006
    JNB     FgKeyFlag,$ + 6H
    LJMP    ?C0028
    SETB    FgKeyFlag
    MOV     R7,#01H
```

```
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep
    MOV     R0,#LastBuffer
    MOV     A,@R0
    MOV     B,#0AH
    MUL     AB
    MOV     R0,#KeyBuffer
    ADD     A,@R0
    MOV     R7,A
    MOV     R0,#MusicNumber
    MOV     @R0,A
    LCALL   _Disp7Seg
    MOV     R0,#KeyBuffer
    MOV     A,@R0
    MOV     R0,#LastBuffer
    MOV     @R0,A
    RET
?C0006:
    MOV     R0,#KeyBuffer
    MOV     A,@R0
    ADD     A,#0F5H
    JZ      ?C0014
    DEC     A
    JZ      ?C0018
    DEC     A
    JNZ     $ + 5H
    LJMP    ?C0022
    ADD     A,#03H
    JZ      $ + 5H
    LJMP    ?C0028
?C0010:
    JNB     FgSingleKey,$ + 6H
    LJMP    ?C0028
    SETB    FgSingleKey
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep
    CPL     FgSingleFlag
    JNB     FgSingleFlag,?C0012
    SETB    P2_4
    SJMP    ?C0013
```

```
?C0C12:
    CLR     P2_4
?C0013:
    CLR     FgSequentFlag
    CLR     FgRandomFlag
    CLR     FgBriefFlag
    CLR     P2_5
    CLR     P2_6
    CLR     P2_7
    RET
?C0014:
    JNB     FgSequentKey,$ + 6H
    LJMP    ?C0028
    SETB    FgSequentKey
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep
    CPL     FgSequentFlag
    JNB     FgSequentFlag,?C0016
    SETB    P2_5
    SJMP    ?C0017
?C0016:
    CLR     P2_5
?C0017:
    CLR     FgSingleFlag
    CLR     FgRandomFlag
    CLR     FgBriefFlag
    CLR     P2_4
    CLR     P2_6
    CLR     P2_7
    RET
?C0018:
    JB      FgRandomKey,?C0028
    SETB    FgRandomKey
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep
    MOV     TMOD,#015H
    SETB    TR0
    MOV     TL0,#0FFH
    MOV     TH0,#0FFH
```

```
    CPL     FgRandomFlag
    JNB     FgRandomFlag,?C0020
    SETB    P2_6
    SJMP    ?C0021
?C0020:
    CLR     P2_6
?C0021:
    CLR     FgSingleFlag
    CLR     FgSequentFlag
    CLR     FgBriefFlag
    CLR     P2_4
    CLR     P2_5
    CLR     P2_7
    RET
?C0022:
    JB      FgBriefKey,?C0028
    SETB    FgBriefKey
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep
    MOV     TMOD,#011H
    MOV     TL0,#0C0H
    MOV     TH0,#063H
    CPL     FgBriefFlag
    JNB     FgBriefFlag,?C0024
    SETB    P2_7
    SJMP    ?C0025
?C0024:
    CLR     P2_7
?C0025:
    CLR     FgSingleFlag
    CLR     FgSequentFlag
    CLR     FgRandomFlag
    CLR     P2_4
    CLR     P2_5
    CLR     P2_6
    RET
?C0005:
    MOV     R0,#KeyBuffer
    MOV     @R0,#0FFH
    CLR     FgKeyFlag
    CLR     FgSingleKey
    CLR     FgSequentKey
```

```
    CLR     FgRandomKey
    CLR     FgBriefKey
?C0028:
    RET

    RSEG  ?PR?InputKey?INPUT
    USING   0
InputKey:
    MOV     R0,#KeyData
    MOV     @R0,#0FFH
    MOV     R0,#LOW (XFR_ADC)
    MOV     A,#081H
    MOVX    @R0,A
    MOV     R7,#02H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#LOW (XFR_ADC)
    MOVX    A,@R0
    ANL     A,#03FH
    MOV     keytmp?140,A
    CLR     C
    SUBB    A,#010H
    JNC     ?C0029
    MOV     R0,#KeyData
    MOV     @R0,#01H
    SJMP    ?C0030
?C0029:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#01BH
    JNC     ?C0031
    MOV     R0,#KeyData
    MOV     @R0,#02H
    SJMP    ?C0030
?C0031:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#026H
    JNC     ?C0033
    MOV     R0,#KeyData
    MOV     @R0,#03H
    SJMP    ?C0030
?C0033:
    MOV     A,keytmp?140
    CLR     C
```

```
        SUBB    A,#030H
        JNC     ?C0035
        MOV     R0,#KeyData
        MOV     @R0,#04H
        SJMP    ?C0030
    ?C0035:
        MOV     A,keytmp?140
        CLR     C
        SUBB    A,#036H
        JNC     ?C0030
        MOV     R0,#KeyData
        MOV     @R0,#05H
    ?C0030:
        MOV     R0,#LOW (XFR_ADC)
        MOV     A,#082H
        MOVX    @R0,A
        MOV     R7,#02H
        MOV     R6,#00H
        LCALL   _DelayXlms
        MOV     R0,#LOW (XFR_ADC)
        MOVX    A,@R0
        ANL     A,#03FH
        MOV     keytmp?140,A
        CLR     C
        SUBB    A,#010H
        JNC     ?C0038
        MOV     R0,#KeyData
        MOV     @R0,#06H
        SJMP    ?C0039
    ?C0038:
        MOV     A,keytmp?140
        CLR     C
        SUBB    A,#01BH
        JNC     ?C0040
        MOV     R0,#KeyData
        MOV     @R0,#07H
        SJMP    ?C0039
    ?C0040:
        MOV     A,keytmp?140
        CLR     C
        SUBB    A,#026H
        JNC     ?C0042
        MOV     R0,#KeyData
        MOV     @R0,#08H
        SJMP    ?C0039
```

```
?C0042:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#030H
    JNC     ?C0044
    MOV     R0,#KeyData
    MOV     @R0,#09H
    SJMP    ?C0039
?C0044:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#036H
    JNC     ?C0039
    CLR     A
    MOV     R0,#KeyData
    MOV     @R0,A
?C0039:
    MOV     R0,#LOW (XFR_ADC)
    MOV     A,#084H
    MOVX    @R0,A
    MOV     R7,#02H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#LOW (XFR_ADC)
    MOVX    A,@R0
    ANL     A,#03FH
    MOV     keytmp?140,A
    CLR     C
    SUBB    A,#010H
    JNC     ?C0047
    MOV     R0,#KeyData
    MOV     @R0,#0AH
    SJMP    ?C0048
?C0047:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#01BH
    JNC     ?C0049
    MOV     R0,#KeyData
    MOV     @R0,#0BH
    SJMP    ?C0048
?C0049:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#026H
```

```
    JNC     ?C0051
    MOV     R0,#KeyData
    MOV     @R0,#0CH
    SJMP    ?C0048
?C0051:
    MOV     A,keytmp?140
    CLR     C
    SUBB    A,#030H
    JNC     ?C0048
    MOV     R0,#KeyData
    MOV     @R0,#0DH
?C0048:
    CLR     A
    MOV     R0,#LOW (XFR_ADC)
    MOVX    @R0,A
    RET

    RSEG  ?PR?StartHandler?INPUT
    USING   0
StartHandler:
    LCALL   InputStart
    MOV     R0,#KeyData
    MOV     A,@R0
    XRL     A,#01H
    JZ      $ + 5H
    LJMP    ?C0060
    CLR     FgEndMusic
    CLR     FgNextMusic
    JNB     FgSingleFlag,?C0056
?C0057:
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Music
    JNB     FgEndMusic,$ + 6H
    LJMP    ?C0060
?C0059:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0057

?C0056:
    JNB     FgSequentFlag,?C0062
?C0063:
```

```
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Music
    JNB     FgEndMusic,$ + 6H
    LJMP    ?C0060
?C0065:
    MOV     R0,#MusicNumber
    INC     @R0
    MOV     A,@R0
    SETB    C
    SUBB    A,#01FH
    JC      ?C0066
    MOV     @R0,#01H
?C0066:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0063

?C0062:
    JNB     FgRandomFlag,?C0068
    CLR     FgStartMusic
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    MOV     R0,#PulseCount
    MOV     A,@R0
    MOV     R6,A
    INC     R0
    MOV     A,@R0
    MOV     R7,A
    LCALL   _srand
?C0069:
    LCALL   rand
    MOV     currentvalue?242,R6
    MOV     currentvalue?242+01H,R7
    MOV     R4,#00H
    MOV     R5,#064H
    LCALL   ?C?SIDIV
    MOV     currentvalue?242,R4
    MOV     currentvalue?242+01H,R5

    SETB    C
    MOV     A,currentvalue?242+01H
```

```
        SUBB    A,#01FH
        MOV     A,currentvalue?242
        XRL     A,#080H
        SUBB    A,#080H
        JNC     ?C0069
        SETB    C
        MOV     A,currentvalue?242+01H
        SUBB    A,#00H
        MOV     A,currentvalue?242
        XRL     A,#080H
        SUBB    A,#080H
        JC      ?C0069

        MOV     A,currentvalue?242+01H
        XRL     A,lastvalue?241+01H
        JNZ     ?C0085
        MOV     A,currentvalue?242
        XRL     A,lastvalue?241
?C0085:
        JZ      ?C0069
        MOV     R7,currentvalue?242+01H
        MOV     R0,#MusicNumber
        MOV     A,R7
        MOV     @R0,A
        LCALL   _Music
        MOV     lastvalue?241,currentvalue?242
        MOV     lastvalue?241+01H,currentvalue?242+01H
        JB      FgEndMusic,?C0060
?C0073:
        MOV     R7,#064H
        MOV     R6,#00H
        LCALL   _DelayX10ms
        SJMP    ?C0069

?C0068:
        JNB     FgBriefFlag,?C0075
?C0076:
        CLR     FgNextMusic
        CLR     A
        MOV     R0,#MusicLength
        MOV     @R0,A
        SETB    TR0
        MOV     R0,#MusicNumber
        MOV     A,@R0
        MOV     R7,A
```

```
    LCALL   _Music
    CLR     TR0
    JB      FgEndMusic,?C0060
?C0078:
    MOV     R0,#MusicNumber
    INC     @R0
    MOV     A,@R0
    SETB    C
    SUBB    A,#01FH
    JC      ?C0079
    MOV     @R0,#C1H
?C0079:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0076

?C0075:
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Music
    JB      FgEndMusic,?C0060
?C0081:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
?C0060:
    RET

    RSEG  ?PR?InputStart?INPUT
    USING   0
InputStart:
    JB      P3_4,?C0082
    MOV     R0,#KeyData
    MOV     @R0,#01H
    RET
?C0082:
    MOV     R0,#KeyData
    MOV     @R0,#0FFH
?C0084:
    RET

    END
```

12-2 实时显示曲目编号

目的：按数字键 0~9 作为选曲并将曲目编号显示出来外，也将四种功能模式中的演奏曲目实时显示出来，也就是显示器的编号就是目前的演奏曲目。

12-2-1 软件构建的思维

input.c：在 StartHandler( )函数里功能模式的回路内，在调用 Music( )函数之前先将曲目编号由显示器显示出来即可，分别在单曲循环回路、顺序播放回路、随机选曲回路、播放简介回路及单曲播放一次的回路内将曲目编号显示出来。除了单曲播放一次及单曲循环的功能模式，其曲目编号就是按键输入的号码（每当按键一次就立即显示出按下的数字键并作为曲目编号的个位数，而前一次的按键码会变成十位数，依次类推，可以不停地一直作按键选曲的操作），并会一直保留其编号，除非再重新作按键选曲。而顺序播放功能，在演奏完一首曲目后，就会持续演奏下一首；随机选曲功能，利用计算机产生随机数作为演奏的曲目编号；播放简介功能，就是每一首曲目都是演奏五秒钟，在演奏完毕并自动地将曲目编号加 1 以演奏下一首，这些功能模式的曲目编号会改变，如果显示器一直保留手动按键选曲的编号而并不随着演奏曲目作更新，会失去显示器的意义，也是一个缺点，因此，才需要将显示器做立即更新功能。

12-2-2 软件的解决方法

在 StartHandler( )函数内，当按下开始键后，以 if...else 的指令结构来描述每一种功能模式的动作程序，只要在每一功能模式回路内，在执行 Music(MusicNumber)函数之前，先调用 Disp7Seg(MusicNumber)函数即可，其中 MusicNumber 为代表曲目编号的全局变量，而 Disp7Seg (MusicNumber)函数，为将目前的曲目编号由七段显示器显示的函数，当显示后才进行演奏，即先执行 Dis7Seg(MusicNumber)函数，再执行 Music(MusicNumber)才能达到要求，范例如下：

```
/*****************************/
void StartHandler(void)
{
   int lastvalue,currentvalue;

   //detect start key
   InputStart( );
```

```
if ( KeyData == START_KEY )
{
   FgEndMusic =0;              //int0:end play music
   FgNextMusic=0;              //timer0:FgNextMusic=1

   if ( FgSingleFlag )              //单曲循环
   {
      while( 1 )
      {
         Disp7Seg(MusicNumber);
         Music(MusicNumber);
         if (FgEndMusic==1) return;
         DelayX10ms(100);
      }
   }
   else if ( FgSequentFlag )       //顺序播放
   {
      while( 1 )
      {
         Disp7Seg(MusicNumber);
         Music(MusicNumber);
         if (FgEndMusic==1) return;

         MusicNumber++;
         if ( MusicNumber > LASTMUSIC )
            MusicNumber=1;
         DelayX10ms(100);
      }
   }
   else if ( FgRandomFlag )        //随机选曲
   {
       FgStartMusic=0;
       DelayX10ms(50);
       srand(PulseCount);      //产生一个随机数种子

     while( 1 )
     {
        currentvalue=rand(); //产生一个随机数
        currentvalue%=100;   //取最后两位数 0~99

        //currentvalue=1~LASTMUSIC
        if ( (currentvalue<=LASTMUSIC)&&(currentvalue>0) )
        {
           //和上次不一样
```

```
            if (currentvalue!=lastvalue)
            {
              MusicNumber=currentvalue;
              Disp7Seg(MusicNumber);
              Music(MusicNumber);
              lastvalue=currentvalue;
              if (FgEndMusic==1) return;
              DelayX10ms(100);
            }
          }
        }
      }
      else if ( FgBriefFlag )              //播放简介
      {
        while( 1 )
        {
          Disp7Seg(MusicNumber);
          FgNextMusic=0;          //timer0:FgNextMusic=1
          MusicLength=0;
          TR0 = 1;
          Music(MusicNumber);
          TR0 = 0;

          if (FgEndMusic==1) return;

          MusicNumber++;
          if ( MusicNumber > LASTMUSIC )
            MusicNumber=1;
          DelayX10ms(100);
        }
      }
      else                                 //单曲播放一次
      {
        Disp7Seg(MusicNumber);
        Music(MusicNumber);
        if (FgEndMusic==1) return;
        DelayX10ms(100);
      }
    }
}
```

12-3　具有记忆功能的装置

目的：将手动按键的曲目编号及功能模式记忆起来，则开机后立即会显示出曲目编号及最后的功能模式。

模块化：因具有记忆功能，因此增加了 eeprom.c 的模块，以便将曲目编号和功能模式存入至 eeprom 内，而 eeprom 的通信模式为 iic bus，因而需要增加 iic.c 的模块，以便将所有 iic bus 的通信协议的软件程序全部都写在此模块内。

12-3-1　软件构建的思维

1. global.c

使用记忆 IC，零件名称为 24C08，总共有 4 页，1024 个字节，其通信协议为 iic bus 的方式，因此增加 ByteCnt 变量和 SlvAdr 的变量，以及传送数据的数组变量 TrmBuf，其占用 10 个字节。

2. iic.c

由于 IC24c08 的控制是使用 iic bus 的通信协议，因此，要设计出 iic bus 的函数库：传送数据和接收数据的函数，以及基本的函数：起始信号函数、传送字节函数、停止信号函数及时序延迟函数等。

3. eeprom.c

既然要将曲目编号及功能模式存入至记忆 IC 内，所以必须写入数据至 24C08 的函数，以及当开机后要将存入至 24C08 的数据立即读出并显示出来，所以需要有从 24C08 读出数据的函数，并规划连续读出或写入多个字节的方式，使得程序结构更为简捷。而功能模式在程序中是以标志变量来表示，仍以标志变量的内容“0”或“1”存入至 24C08 并占用一个字节，在程序中以 Bool 和 Byte 作型别转换即可。

4. initial.c

（1）以两个字节的数据来判别记忆 ic24C08 是否已经存有数据，由于 24C08 制 1024 个字节地址的数据都是空的，内容为 0Xff，因此在第一次开机可将各个变量的内容写入至 24C08 内，并在最后的两个地址内作上版本标记。例如：第 1023byte 地址的内容为 0X12，第 1024 byte 地址的内容为 0X01，则只要每一次开机，检查最后两个 byte 其数据是否为 0X1201，如果条件成立，表示已经将内定值存入至 24C08

了，就无需再重新存入，否则，在往后按键选曲的编号又会被内定值覆盖掉，这样就无法达到记忆最后一曲的功能了。如果条件不成立，表示是第一次开机，就需要将内定值存入至 24C08 了。

（2）开机后必须将最后存入至 24C08 的曲目编号及功能模式的标志值立即读出，并存入至对应变量及标志变量内，以便其他模块程序的应用，由于从 24C08 读出的数据是暂存至数组变量 TrmBuf 内，是占用一个字节的字符数据类型，以 Bool 类型转换后才存入相对应的标志变量内。

12-3-2 变量的意义说明与使用时机

1. ByteCnt：iic bus 通信协议传送至 24C08 的字节个数

- 数据类型：占用一个字节的无符号字符类型(unsigned char)。
- 内存类型：变量地址介于 0X00～0Xff 之间，使用间接寻址模式(idata)。
- 写入：当要传送或接收数据字节个数时。
- 循环：

（1）传送一个字节函数的循环判断式，当未达到传送的个数时就继续传送，一直到全部的数据都传送完毕为止。

（2）在接收数据循环的判断式，如果未达到接收个数则循环继续做，即继续接收数据，每接收完一笔数据就存入至数组变量内，直到全部的数据都接收完毕为止。

- 判断：当接收完最后一笔数据时，要将确认信号送“1”电位，即 Data Pin 输出“1”电位给 24C08 做确认，表示已经接收完毕，如未接收完毕则需将 Data Pin 送“0”电位，即确认信号 ack=“0”传递给 24C08，以表示还要继续接收。

2. SlvAdr：记忆 ic 24C08 的装置地址，每一个 iic bus 的 IC 都有一个固定的地址，称之为 Slave Address 或 Device Address。

- 数据类型：占用一个字节的无符号字符类型(unsigned char)。
- 内存类型：变量地址介于 0X00～0Xff 之间，使用间接寻址模式(idata)。
- 写入：当将数据存入至 24C08 或由 24C08 读出数据，由于 24C08 的格式必须先寻址 bank 再寻址实际的地址，因此必须将地址转换成 bank 和 subAddress，例如：

当 Address<256 时，则 SlvAdr=A0，SubAddress=Address

Address<512 时，则 SlvAdr=A2，SubAddress=Address－256

Address<768 时，则 SlvAdr=A4，SubAddress=Address－512

Address<1024 时，则 SlvAdr=A6，SubAddress=Address－768

也就是 bank0，其 SlvAdr=A0，地址区间 0X00～0Xff

bank1，其 SlvAdr=A2，地址区间 0X00～0X1ff

bank2，其 SlvAdr=A4，地址区间 0X00～0X2ff

bank3，其 SlvAdr=A6，地址区间 0X00～0X3ff

- 自变量：

（1）写入数据至 24C08 必须寻址 bank，也就是 24C08 的 Slave Address 分别为 0XA0、0XA2、0XA4 及 0XA6，因此，将 SlvAdr 变量内容即 24C08 的 Slave Address 以自变量的方式传送给 24C08，紧接着再传送实际地址，才能将数据正确的写入至 24C08 内。

（2）要接收数据时，必须将 Slave Address 变换成读取模式，即将 SlvAdr 加 1 后传送给 24C08，则 24C08 就会开始将数据以一个 bit 紧接着一个 bit 的类型出现在 data pin（高位先出现），以方便微电脑读取 24C08 地址内的数据。

3. TrmBuf[10]

- 数据类型：占用一个字节的无号数字元类型(unsigned char)，其数组总共占用 10 个字节。
- 内存类型：地址介于 0X00～0Xff，使用间接寻址模式(idata)。
- 写入：

（1）eeprom 24C08 第一次开机将内定值写入后，也将版本代号写入至 eeprom 最后的两个地址内。

（2）将程序内存的内定值依序暂存至 TrmBuf 数组内后，并写入至 24C08 的地址内，如果内定值的个数超过 10 个，则需分两次写入至 24C08 或是将 TrmBuf 数组变量的下标值 10 改变为实际要写入至 24C08 内定值的个数。

（3）将最后输入的曲目编号暂存以便写入至 24C08 内。

（4）由 24C08 读取到的数据，依次暂存至数组内，再将数组内容存入至实际的变量内。

（5）要写入 24C08 的数据也是通过先依序暂存至数组内，再将数组内的数据传送至 24C08 做储存记忆，即使 24C08 来供应电源，其地址内的数据也不会消失，因而可作为记忆之用。

（6）要读取 24C08 的数据，除了须寻址 Slave Address 外，也需寻址要从哪一个地址开始读取，即须将索引地址先暂存至数组 0 内再传送至 24C08。

（7）将一串数据写入至 24C08，实际上以每次写入一个 byte 会比较稳定，也不会有跨列的 miss，TrmBuf[0]存入待写入的地址，而 TrmBuf[1]则为写入的数据，由于 TrmBuf[1]会改变，因此需要暂存其值及写入完毕后的回复。

- 判断：

（1）第一次开机，先从 24C08 读取最后两个字节的数据。判断是否已经将内定值写入至 24C08 了。

（2）将数组变量的内容数据传送给 24C08 后，必须判断是否传送成功。

12-3-3 软件的解决方法

1. global.c

ByteCut、SlvAdr 及 TrmBuf 数组变量都是占用一个字节的无符号字符变量，其地址范围介于 0X00～0Xff 并使用间接寻址法，写法如下：

```
Byte IDATA TrmBuf[10];
Byte IDATA TrmCnt;
Byte IDATA SlvAdr;
```

2. global.h

将全局变量也利用相对应的包括文件来声明成外部变量，以便其他模块的写入、读取或判断之用，写法如下：

```
extern Byte IDATA TrmBuf[10];
extern Byte IDATA ByteCnt;
extern Byte IDATA SlvAdr;
```

3. iic.c

由于利用 iic bus 的通信协议，将数据写入至 24C08 或从 24C08 的地址读取数据，因而必须增加 iic.c 的模块，其构成的函数如下：

（1）IICDelay( )：iic bus 时序延迟函数，用以确保 iic bus 的 clock 和 data 可以正确运作，并声明成静态函数，以隔离其他模块的取用，也就是只能在 iic.c 的模块下才能调用 IICDelay( )函数，其程序写法如下：

```
/*****************************/
static void IICDelay(void)
{
   _nop_();
   _nop_();
   _nop_();
   _nop_();
}
```

（2）SendStop( )：iic bus 时序停止信号的函数，当利用 iic bus 传送或接收数据完毕时，则必须以停止信号来结束通信。程序写法如下：

```
/*****************************/
static void SendStop(void)
{
    SDA_PIN = 0;
    SCL_PIN = 0;
    IICDelay();

    SCL_PIN = 1;
    IICDelay();

    SDA_PIN = 1;
    IICDelay();
}
```

其中 SDA_PIN 为 I/O P3.5，SCL_PIN 为 I/O P3.0。

（3）GoMaster( )：iic bus 时序传送起始信号和 Slave Address，在开始进行 iic bus 的通信，其 scl 和 sda pin 都必须是“1”电位，只要有任何一个信号是“0”电位，则必须结束通信，否则，即使做通信传送则通信结果也会不正确，当传送成功则回复 SUCCESS（定义为“0”）信号至原调用程序，程序写法如下：

```
/*****************************/
static Bool GoMaster(Byte slave_addr)
{
    if ( (SCL_PIN==0) || (SDA_PIN==0) )
       return FAILURE;

    SDA_PIN = 0;
    IICDelay();

    SCL_PIN = 0;
    IICDelay();

    SendByte(slave_addr);

    return SUCCESS;
}
```

（4）SendByte( )：iic bus 传送一个字节数据的函数，此函数是最基本 iic bus 的传送方法，传送时以最高位的数据先传送，并判断要传送数据的每一个位，当数据位为“1”则 sda 引脚就送“1”电位，当数据位为“0”，则 sda 引脚就送“0”电位，此时 scl 作“1”→“0”的变化则可将此位传送至 slave 端（24C08），如此反复做八次即可传送一个字节的数据，每将此数据位的内容（“0”或“1”）送至 sda pin 时，则待传送的变量数据也需左移一个位以便继续传送下一个位，待八个位传送完

毕，也要读取确认信号并将确认信号回复至原调用程序以便判断传送是否正确，而确认信号会反应在sda pin上，所以只要读取sda pin的信号“1”或“0”，就知道确认信号是“1”或“0”了，当确认信号为“0”表示传送成功，为“1”表示传送失败，程序写法如下：

```
/*****************************/
static Bool SendByte(Byte bval)
{
   Byte  i;
   Bool  no_ack = 0;

   for(i=0; i<8; i++)
   {
      if ( bval & 0x80 )   SDA_PIN = 1;
      else                 SDA_PIN = 0;
      bval <<= 1;
      IICDelay();

      SCL_PIN = 1;
      IICDelay();

      SCL_PIN = 0;
      IICDelay();
   }

   SDA_PIN = 1;
   IICDelay();

   SCL_PIN = 1;
   IICDelay();

   if ( SDA_PIN == 1 )  no_ack = 1;

   SCL_PIN = 0;
   IICDelay();

   return  no_ack;
}
```

（5）RcvData( )：iic bus 接收一连串数据的函数，接收数据的个数以变量ByteCnt来表示，而接收到数据依序暂存至TrmBuf数组中，也就是第一个接收的数据存至Trmbuf[0]中，第二个接收到的数据存至TrmBuf[1]中，依次类推。要接收数据必须将Slave Address加一变成读取模式才可以，每一个字节也是从最高位开始将

“0”或“1”反应到 data pin 上，每接收到一个位就将数据左移一次，如此反复八次后再将确认信号“0”电位送至 data pin 上，以表示接收成功，当接收完最后一个字节的数据时，须将确认信号“1”电位送至 data pin，以表示接收完毕，当侦测到 data pin=“0”电位，表示数据位为“0”，data pin=“1”电位则表示数据位为“1”，每接收到一个位，则 scl pin 必须做“1”→“0”的变化，以符合 iic bus 的通信协议，才能正确地接收到每一位。每一字节的第九位为确认位，sel 也需要做“1”→“0”的变化才能送至 slave 端(24C08)，如果 master 端为 MCU，slave 端为 24C08，则当 MCU 将数据写入至 24C08 时，则确认信号由 24C08 送出；MCU 做判断；当 MCU 读取 24C08 内的数据时，则确认信号由 MCU 送出，程序写法如下：

```
/*****************************/
Bool RcvData(void)
{
   Byte  i,j,bval;

   if ( GoMaster(SlvAdr+1) != SUCCESS )
   {
      SendStop();
      return  FAILURE;
   }

   for(i=0; i<ByteCnt; i++)
   {
      bval = 0;
      for(j=0; j<8; j++)
      {
         SCL_PIN = 1;
         IICDelay();

         bval = bval << 1;
         if ( SDA_PIN )  bval |= 0x01;

         SCL_PIN = 0;
         IICDelay();
      }
      if ( i==(ByteCnt-1) ) SDA_PIN = 1;
      else                  SDA_PIN = 0;
      IICDelay();

      SCL_PIN = 1;
      IICDelay();

      SCL_PIN = 0;
```

```
        IICDelay();

        SDA_PIN = 1;
        IICDelay();

        TrmBuf[i] = bval;
    }

    SendStop();

    return  SUCCESS;
}
```

（6）SendEepromData()：写入一串数据至 eeprom 24C08 内，当 iic bus 传送数据有错误发生时，则必须先传送“停止信号”后才能从“起始信号”开始重新传送，也必须做 TimeOut 的处理，即连续传送 10 次，如果都不成功则退出 iic bus 的通信，如果不做 TimeOut 处理，系统中有 iic bus 的组件，组件如果损坏或未正确插入电路板上，则将死机，因程序一直在重新传送的回路内绕圈子，而 TimeOut 处理则可避免此现象。eeprom 24C08 做写入命令时，则下一次的写入命令必须延迟 10ms 以上才能正确地写入，因此，当 iic bus 有异常需重新传送时，则需等待 10ms 以上，程序写法如下：

```
/*****************************/
Bool SendEepromData(void)
{
    Byte  i,j;
    Bool  no_ack = 1;

    for( i=0; i<10; i++)
    {
        if ( GoMaster(SlvAdr) != SUCCESS )
        {
            SendStop();
            DelayX1ms (10);
            continue;
        }
        no_ack = 0;
        for(j=0; j<ByteCnt; j++)
        {
            if ( SendByte( TrmBuf[j] ) != SUCCESS )
            {
                no_ack = 1;
                break;
```

```
            }
        }
        SendStop();
        if ( no_ack==0 )  break;
        DelayX1ms (10);
    }
    return  no_ack;
}
```

（7）SendData( )：iic bus 传送一串数据的函数，可适用于非 eeprom 记忆 IC 的 slave 组件，因为每写入 eeprom 则需等待 10ms 才能再写入。例如：要连续写入 10 个数据至 24C08 内，实际上为确保跨列的错误和麻烦，而采取一个字节的写入模式，即先寻址后再将一个数据写入至此地址上，因此 10 个数据必须做 10 次写入模式的动作，每一次须延迟 10ms，10 个数据须延迟 10X10=100ms，为了其他 device 的写入动作（除了 eeprom，其他 device 的写入动作不须延迟 10ms 就能继续写入）而将 SendEepromData( )函数 10ms 延迟删除掉而成为 SendData( )函数，而当欲读取 eeprom 24C08 的地址数据，必须先传送欲从哪一个地址开始读取后才能开始读取，因为紧接着是读取模式，所以传送读取地址不需要延迟 10ms，就可以使用 SendData( )函数以节省 10ms 的时间了，程序写法如下：

```
/*****************************/
Bool SendData(void)
{
    Byte  i,j;
    Bool  no_ack = 1;

    for( i=0; i<10; i++)
    {
        if ( GoMaster(SlvAdr) != SUCCESS )
        {
            SendStop();
            continue;
        }
        no_ack = 0;
        for(j=0; j<ByteCnt; j++)
        {
            if ( SendByte( TrmBuf[j] ) != SUCCESS )
            {
                no_ack = 1;
                break;
            }
        }
        SendStop();
```

```
        if ( no_ack==0 )  break;
    }
    return  no_ack;
}
```

程序中以SUCCESS或FAILURE常数来回复iic bus的通信是否成功，因而，两个常数需要进行定义，而GoMaster( )函数、SendByte( )函数、SendStop( )函数、IIC Delay( )函数为iic.c模块中才能使用的函数，因而声明成静态函数，以隔绝其他模块，在IICDelay( )函数中使用了_nop_( )的指令，则必须在程序的开头利用include指令来包括相对应的包括文件，_nop_( )函数的包括文件为intrins.h，如上所述，在程序前端写法如下：

```
#include  <intrins.h>

/*******************************/
#define   SUCCESS    0
#define   FAILURE    1

static Bool GoMaster(Byte slave_addr);
static Bool SendByte(Byte bval);
static void SendStop(void);
static void IICDelay(void);
```

4. iic.h：iic.c模块的包括文件，包含定义iic bus scl pin和data pin的I/O引脚(scl pin的I/O引脚为P3.0，sda pin的I/O引脚为P3.5)及将SendData( )函数，SendEeprom Data( )函数、RcvData( )函数声明成外部函数，以便其他模块调用。包括文件的写法如下：

```
#ifndef  _IIC_H
#define  _IIC_H

#define  SCL_PIN P3_0
#define  SDA_PIN P3_5

extern Bool SendData(void);
extern Bool SendEepromData(void);
extern Bool RcvData(void);

#endif
```

其中行号01及行号11为条件式编译指令。

行号01：#ifndef，标名：表示当标名未被定义则开始执行以下的语句。

行号11：#endif，表示结束条件式编译指令。

5. eeprom.c：为了将最后按键选曲的曲目编号及设定的功能模式可以记忆，便在开机后即能显示曲目编号及功能模式的 LED 点亮；因而，多增加 eeprom.c 的模块，以作为 ic24C08 的写入和读取数据之用，其构成的函数如下：

（1）EepromWrite( )函数：将一串数据写入至 eeprom 24C08 的函数，利用地址 0～1023 作为要写入数据的开始地址，并以自变量来传递开始地址及欲写入数据个数，而程序必须将地址 0～255 换算为 bank0，slave 地址为 A0；地址 256～511 换算为 bank1，slave 地址为 A2，地址须减去 256 作为实际地址；地址 512～768 换算为 bank2，slave 地址为 A4，地址须减去 512 作为实际地址；地址 768～1023 换算为 bank3，slave 地址为 A6，地址必须减去 768 作为实际地址，并依序将数组变量 TrmBuf[1]至 TrmBuf[n]写入至 24C08 内，其中 n 为欲写入的个数，虽然是写入一串数据至 24C08 内，程序仍以一个字节的写入模式并以 for 循环来判断是否将全部数据写入完毕，以确保写入数据的正确性，原来数组 TrmBuf[1]的内容会因为执行 EepromWrite( )函数而改变，因而先备份此原始的 TrmBuf[1]内容，待写入完成欲返回调用程序前再还原成原来的 TrmBuf[1]即可，程序写法如下：

```
/*******************************/
void EepromWrite(Word SubAddress, Byte count)
{

   Byte i,FirstValue;

   if ( SubAddress < 256 )
   {
      SlvAdr = EEPROM_PAGE0_SLVADR;.
   }
   else if ( SubAddress < 512 )
   {
      SlvAdr = EEPROM_PAGE1_SLVADR;
      SubAddress = SubAddress - 256;
   }
   else if ( SubAddress < 768 )
   {
      SlvAdr = EEPROM_PAGE2_SLVADR;
      SubAddress = SubAddress - 512;
   }
   else
   {
      SlvAdr = EEPROM_PAGE3_SLVADR;
      SubAddress = SubAddress - 768;
   }
```

```
    FirstValue = TrmBuf[1];
    for( i=1; i<=count; i++)
    {
       TrmBuf[0] = (Byte)SubAddress;
       TrmBuf[1] = TrmBuf[i];
       ByteCnt = 2;
       SendEepromData();
       SubAddress++;
    }
    TrmBuf[1] = FirstValue;
}
```

（2）EepromRead()：由 eeprom 24C08 读取一串数据并依序暂存至下标从 0 开始的 TrmBuf 数组变量内，也以地址 0～1023 及接收数据的个数作为自变量，须将地址适当地转换为 slave 地址 A0、A2、A4 或 A6，以取得正确的数据，于读取之前须先送出读取的起始地址后，才真正的开始读取以自变量 count 为个数的数据，程序写法如下：

```
/*******************************/
void EepromReao(Word SubAddress,Byte count)
{
    if ( SubAddress < 256 )
    {
       SlvAdr = EEPROM_PAGE0_SLVADR;
    }
    else if ( SubAddress < 512 )
    {
       SlvAdr = EEPROM_PAGE1_SLVADR;
       SubAddress = SubAddress - 256;
    }
    else if ( SubAddress < 768 )
    {
       SlvAdr = EEPROM_PAGE2_SLVADR;
       SubAddress = SubAddress - 512;
    }
    else
    {
       SlvAdr = EEPROM_PAGE3_SLVADR;
       SubAddress = SubAddress - 768;
    }

    TrmBuf[0] = (Byte)SubAddress;
    ByteCnt   = 0x01;
    SendData();
```

```
    ByteCnt   = count;
    RcvData();
}
```

（3）EepromCommonStore( )：将单曲循环功能标志、顺序播放功能标志、随机选曲功能标志以及播放简介功能标志顺序地存入以 FgModeFlag_ADR 为起始地址的 eeprom 内，以便开机可以立即显示最后的功能模式为何，其程序如下：

```
/*****************************/
void  EepromCommonStore(void)
{
    TrmBuf[ 1] = (Byte)FgSingleFlag;
    TrmBuf[ 2] = (Byte)FgSequentFlag;
    TrmBuf[ 3] = (Byte)FgRandomFlag;
    TrmBuf[ 4] = (Byte)FgBriefFlag;

    EepromWrite(FgModeFlag_ADDR,4);
}
```

（4）EepromCommonRead( )：将曲目编号、单曲循环功能标志、顺序播放功能标志、随机选曲功能标志以及播放简介功能标志依序从 eeprom 内 EE1_START_ADDR 的地址读出来，并分别存入至数组变量 TrmBuf[0]～TrmBuf[4]，其中需将 TrmBuf[1]～TrmBuf[4]强迫转换为 bit 声明类型，分别再存入 FgSingleFlag、FgSequentFlag、FgRandomFlag 以及 FgBriefFlag 的标志变量内，以便日后重新开机后可立即显示上一次开机的曲目编号和功能模式的状态，而其对应的 LED 也会点亮，程序如下：

```
/*****************************/
void  EepromCommonRead(void)
{
    EepromRead( EE1_START_ADDR,5 );

    MusicNumber   = TrmBuf[0];
    FgSingleFlag  = (Bool)TrmBuf[1];
    FgSequentFlag = (Bool)TrmBuf[2];
    FgRandomFlag  = (Bool)TrmBuf[3];
    FgBriefFlag   = (Bool)TrmBuf[4];
}
```

6. eeprom.h：eeprom.c 模块的包括文件，除了将 eeprom.c 模块下的所有函数声明为外部之外，也必须定义曲目编号存入至 eeprom 的地址，而单曲循环功能标志、顺序播放功能标志、随机选曲功能标志以及播放简介功能标志也应定义其 eeprom 内的地址、还有其他的定义符号：

VERSION_NO：eeprom 内容的版本编号，当检查到版本编号不一样时则重新加载内定值至 eeprom 内。

ROM_SIZE：eeprom 可写入的地址空间。

VERSION_NO_ADDR：存入版本编号的地址，为 eeprom 地址内的最后 2 个字节。

EEPROM_PAGE0_SLVADR:eeprom bank0 的 slave 地址为 0XA0

EEPROM_PAGE1_SLVADR:eeprom bank1 的 slave 地址为 0XA2

EEPROM_PAGE2_SLVADR:eeprom bank2 的 slave 地址为 0XA4

EEPROM_PAGE3_SLVADR:eeprom bank3 的 slave 地址为 0XA6

EE4_START_ADDR SLVADR:eeprom bank3 的实际地址编号为 768

EE3_START_ADDR SLVADR:eeprom bank2 的实际地址编号为 512

EE2_START_ADDR SLVADR:eeprom bank1 的实际地址编号为 256

EE1_START_ADDR SLVADR:eeprom bank0 的实际地址编号为 0

其包括文件的内容如下所示：

```
#ifndef _EEPROM_H
#define _EEPROM_H

#define VERSION_NO              0x1201
#define ROM_SIZE                1024
#define VERSION_NO_ADDR         ROM_SIZE-2

#define EEPROM_PAGE0_SLVADR     0xA0
#define EEPROM_PAGE1_SLVADR     0xA2
#define EEPROM_PAGE2_SLVADR     0xA4
#define EEPROM_PAGE3_SLVADR     0xA6

#define EE4_START_ADDR              768
#define EE3_START_ADDR              512
#define EE2_START_ADDR              256
#define EE1_START_ADDR              0

#define MusicNumber_ADDR            EE1_START_ADDR+0x00

#define FgModeFlag_ADDR             EE1_START_ADDR+0x01
#define FgSingleFlag_ADDR           EE1_START_ADDR+0x01
#define FgSequentFlag_ADDR          EE1_START_ADDR+0x02
#define FgRandomFlag_ADDR           EE1_START_ADDR+0x03
#define FgBriefFlag_ADDR                EE1_START_ADDR+0x04

extern void EepromCommonStore(void);
```

```
extern void EepromCommonRead(void);
extern void EepromRead(Word,Byte );
extern void EepromWrite(Word,Byte);

#endif
```

7.initial.c：每次开机除了 CPU 缓存器作初始化及 CPU I/O 的初始化外，由于要具备记忆功能，因此，要将 eeprom 的版本编号读出，以判断 eeprom 是否有内容了。当 eeprom 的内容都为空白，则在第一次开机后将自动填入内定值的内容，并写入版本编号表示已经作好内定值的设定了，如果判别是正确的版本编号则不再重新写入内定值，以避免以前所调整或设定的内容又被改变，除此之外，也要读出最后设定的曲目编号和功能模块，以便开机后即能显示目前的状态。而显示器是通过 ic4094 来驱动，因此，先行将 ic4094 作初始化的动作，以便能正确的显示曲目编号，最后需将所读到的功能模式标志，也将其相对应的 LED 点亮，和读到的曲目编号借着 ic4094 而将其由七段显示器显示出来，而读到的曲目编号必须将最后输入的数字换算出来而写入至 LastBuffer 变量内，以便继续作数字输入时，可立即移位至十位数而正确地显示数字，例如：当读到的曲目编号为 12，则 LastBuffer 变量的内容将为 2，此时又输入 4，则七段显示器将显示出“24”，如果按下“起始键”则会演奏第 24 首的歌曲，其程序如下：

```
/*****************************/
/* include files             */
/*****************************/
#include  "define.h"
#include  "mtv212m.h"
#include  "global.h"
#include  "initial.h"
#include  "delay.h"
#include  "music.h"
#include  "input.h"
#include  "beep.h"
#include  "7segled.h"
#include  "ic4094.h"
#include  "eeprom.h"
#include  "iic.h"

Byte RDATA EepromDefault_table[] =
{
  0x01,       //MusicNumber
  0x00,       //FgSingleFlag
  0x00,       //FgSequentFlag
  0x00,       //FgRandomFlag
```

```
    0x00        //FgBriefFlag
};

/*****************************/
void PowerOnInit(void)
{
    InitialCpu();

    InitialCpuIO();

    EepromRead(VERSION_NO_ADDR,2);
    if ( (TrmBuf[0]!=HIBYTE(VERSION_NO)) || (TrmBuf[1] != LOBYTE
(VERSION_NO)) )
      InitialEeprom();

    EepromCommonRead();

    Initial4094();

    RecallStatus();

    InitialVariable();
}

/*****************************/
void InitialCpu( void )
{
    IE  = 0;          //disable all interrupt
    PSW = 0;          //bank 0

    IP  = 0x0b;       //hi priority:int0,timer0,timer1

    TMOD= 0x11;       //set timer1,timer0 mode

    TR0 = 0;          //stop timer0
    TR1 = 0;          //stop timer1
    IT0 = 1;          //set int0:falling eage trigger

    //CLOCK_40MS=(65536 - 40000)
    TL0 = CLOCK_40MS & 0xff;
    TH0 = CLOCK_40MS >> 8;

    EX0 = 1;          //enable int0 interrupt
    ET1 = 1;          //enable timer1 interrupt
    ET0 = 1;          //enable timer0 interrupt
```

```
    //Initial XFR's and DAC's for MTV212MNXX
    XFR_ADC  =0x80;       //enable ADC,not select ch input
    XFR_WDT  =0x40;       //clear watch dog timer,2s interval

    XFR_PADMOD1  =0x07;   //set adc0,adc1,adc2,i/o:P2.4~P2.7
    XFR_PADMOD2  =0x00;   //set dac0~dac6
    XFR_PADMOD3  =0x00;   //set dac,h,v
    XFR_OPTION1  =0xc0;   //94k pwm=1,div253=1
    XFR_OPTION2  =0x00;   //7 bit slave address for iic
    XFR_XBANK    =0x00;   //ram bank0

    EA  = 1;              //enable all interrupt gate
}

/*****************************/
void InitialCpuIO(void)
{
    SPEAKER    =0;
}

/*****************************/
void InitialEeprom(void)
{
    Byte  i;

    for( i=0; i<5; i++)
      TrmBuf[i+1] = EepromDefault_table[i];
    EepromWrite( EE1_START_ADDR,5 );

    TrmBuf[1] = HIBYTE( VERSION_NO );
    TrmBuf[2] = LOBYTE( VERSION_NO );
    EepromWrite( VERSION_NO_ADDR,2 );
}

/*****************************/
void Initial4094(void)
{
    DATA_PIN   =0;        //P1.1,IC4094 data
    CLK_PIN    =0;        //P1.2,IC4094 clock
    STROBE_PIN =0;        //P1.0,IC4094 strobe
    Output4094(0x00);     //clear IC4094 output
}

/*****************************/
void RecallStatus(void)
```

```
{
    //display MusicNumber
    Disp7Seg(MusicNumber);

    //i/o:P2.4~P2.7,led on/off
    SINGLE_LED =0;
    SEQUENT_LED=0;
    RANDOM_LED =0;
    BRIEF_LED  =0;

    if ( FgSingleFlag==1 )
       SINGLE_LED=1;
    else
       SINGLE_LED=0;

    if ( FgSequentFlag==1 )
       SEQUENT_LED=1;
    else
       SEQUENT_LED=0;

    if ( FgRandomFlag==1 )
       RANDOM_LED=1;
    else
       RANDOM_LED=0;

    if ( FgBriefFlag==1 )
       BRIEF_LED=1;
    else
       BRIEF_LED=0;
}

/*****************************/
void InitialVariable(void)
{
    Byte  i;

    KeyData      = NO_KEY;
    FgEndMusic   = 0;
    FgStartMusic = 0;                    //stop play music

    i=MusicNumber/10;                  //十位数
    LastBuffer=(MusicNumber-=i*10);  //个位数
}
```

8. initial.h：initial.c 模块的包括文件，即将 initial.c 模块下的所有函数都声明为

外部函数，如下：

```
#ifndef _INITIAL_H
#define _INITIAL_H

extern void PowerOnInit(void);
extern void InitialCpu(void);
extern void InitialCpuIO(void);
extern void InitialEeprom(void);
extern void Initial4094(void);
extern void RecallStatus(void);
extern void InitialVariable(void);

#endif
```

汇编程序如下：

1. global.a51

```
$NOMOD51

NAME    GLOBAL

$INCLUDE (mtv212n.inc)

?ID?GLOBAL          SEGMENT  IDATA
?BI?GLOBAL          SEGMENT  BIT
    PUBLIC  XFR_XBANK
    PUBLIC  XFR_OPTION2
    PUBLIC  XFR_OPTION1
    PUBLIC  XFR_PADMOD3
    PUBLIC  XFR_PADMOD2
    PUBLIC  XFR_PADMOD1
    PUBLIC  XFR_WDT
    PUBLIC  XFR_ADC
    PUBLIC  SoundLongCount
    PUBLIC  Period
    PUBLIC  Tempo
    PUBLIC  MusicNumber
    PUBLIC  ScanKeyCounter
    PUBLIC  LastBuffer
    PUBLIC  KeyBuffer
    PUBLIC  KeyData
    PUBLIC  FgKeyFlag
    PUBLIC  FgBriefFlag
    PUBLIC  FgRandomFlag
    PUBLIC  FgSequentFlag
```

```
PUBLIC  FgSingleFlag
PUBLIC  FgBriefKey
PUBLIC  FgRandomKey
PUBLIC  FgSequentKey
PUBLIC  FgSingleKey
PUBLIC  PulseCount
PUBLIC  MusicLength
PUBLIC  FgStartMusic
PUBLIC  FgNextMusic
PUBLIC  FgEndMusic
PUBLIC  SlvAdr
PUBLIC  ByteCnt
PUBLIC  TrmBuf

XSEG  AT  010H
   XFR_ADC:   DS   1

XSEG  AT  018H
   XFR_WDT:   DS   1

XSEG  AT  030H
XFR_PADMOD1:  DS   1

XSEG  AT  031H
XFR_PADMOD2:  DS   1

XSEG  AT  032H
XFR_PADMOD3:  DS   1

XSEG  AT  033H
XFR_OPTION1:   DS   1

XSEG  AT  034H
XFR_OPTION2:   DS   1

XSEG  AT  035H
XFR_XBANK:     DS   1

RSEG  ?ID?GLOBAL
     TrmBuf:      DS   10
    ByteCnt:      DS   1
     SlvAdr:      DS   1
MusicLength:      DS   1
 PulseCount:      DS   2
    KeyData:      DS   1
```

```
      KeyBuffer:     DS   1
     LastBuffer:     DS   1
ScanKeyCounter:      DS   1
   MusicNumber:      DS   1
         Tempo:      DS   2
        Period:      DS   2
 SoundLongCount:     DS   2

    RSEG  ?BI?GLOBAL
     FgEndMusic:     DBIT   1
    FgNextMusic:     DBIT   1
   FgStartMusic:     DBIT   1
    FgSingleKey:     DBIT   1
   FgSequentKey:     DBIT   1
    FgRandomKey:     DBIT   1
     FgBriefKey:     DBIT   1
   FgSingleFlag:     DBIT   1
  FgSequentFlag:     DBIT   1
   FgRandomFlag:     DBIT   1
    FgBriefFlag:     DBIT   1
      FgKeyFlag:     DBIT   1

    END
```

2. initial.a51

```
$NOMOD51

NAME    INITIAL

$INCLUDE (mtv212m.inc)

?PR?PowerOnInit?INITIAL          SEGMENT CODE
?PR?InitialCpu?INITIAL           SEGMENT CODE
?PR?InitialCpuIO?INITIAL         SEGMENT CODE
?PR?InitialEeprom?INITIAL        SEGMENT CODE
?PR?Initial4094?INITIAL          SEGMENT CODE
?PR?RecallStatus?INITIAL         SEGMENT CODE
?PR?InitialVariable?INITIAL      SEGMENT CODE
?CO?INITIAL                   SEGMENT CODE
    EXTRN   IDATA (TrmBuf)
    EXTRN   BIT (FgEndMusic)
    EXTRN   BIT (FgStartMusic)
    EXTRN   BIT (FgSingleFlag)
    EXTRN   BIT (FgSequentFlag)
```

```
    EXTRN   BIT (FgRandomFlag)
    EXTRN   BIT (FgBriefFlag)
    EXTRN   IDATA (KeyData)
    EXTRN   IDATA (LastBuffer)
    EXTRN   IDATA (MusicNumber)
    EXTRN   XDATA (XFR_ADC)
    EXTRN   XDATA (XFR_WDT)
    EXTRN   XDATA (XFR_PADMOD1)
    EXTRN   XDATA (XFR_PADMOD2)
    EXTRN   XDATA (XFR_PADMOD3)
    EXTRN   XDATA (XFR_OPTION1)
    EXTRN   XDATA (XFR_OPTION2)
    EXTRN   XDATA (XFR_XBANK)
    EXTRN   CODE (_Disp7Seg)
    EXTRN   CODE (_Output4094)
    EXTRN   CODE (EepromCommonRead)
    EXTRN   CODE (_EepromRead)
    EXTRN   CODE (_EepromWrite)
    PUBLIC  EepromDefault_table
    PUBLIC  InitialVariable
    PUBLIC  RecallStatus
    PUBLIC  Initial4094
    PUBLIC  InitialEeprom
    PUBLIC  InitialCpuIO
    PUBLIC  InitialCpu
    PUBLIC  PowerOnInit

    RSEG  ?CO?INITIAL
EepromDefault_table:
    DB  001H
    DB  000H
    DB  000H
    DB  000H
    DB  000H

    RSEG  ?PR?PowerOnInit?INITIAL
    USING   0
PowerOnInit:
    LCALL   InitialCpu
    LCALL   InitialCpuIO
    MOV     R7,#0FEH
    MOV     R6,#03H
    MOV     R5,#02H
    LCALL   _EepromRead
    MOV     R0,#TrmBuf
```

```
    MOV     A,@R0
    CJNE    A,#012H,?C0002
    INC     R0
    MOV     A,@R0
    XRL     A,#01H
    JZ      ?C0001
?C0002:
    LCALL   InitialEeprom
?C0001:
    LCALL   EepromCommonRead
    LCALL   Initial4094
    LCALL   RecallStatus
    LCALL   InitialVariable
    RET

    RSEG  ?PR?InitialCpu?INITIAL
    USING   0
InitialCpu:
    CLR     A
    MOV     IE,A
    MOV     PSW,A
    MOV     IP,#0BH
    MOV     TMOD,#011H
    CLR     TR0
    CLR     TR1
    SETB    IT0
    MOV     TL0,#0C0H
    MOV     TH0,#063H
    SETB    EX0
    SETB    ET1
    SETB    ET0
    MOV     R0,#LOW (XFR_ADC)
    MOV     A,#080H
    MOVX    @R0,A
    MOV     R0,#LOW (XFR_WDT)
    MOV     A,#040H
    MOVX    @R0,A
    MOV     R0,#LOW (XFR_PADMOD1)
    MOV     A,#07H
    MOVX    @R0,A
    CLR     A
    MOV     R0,#LOW (XFR_PADMOD2)
    MOVX    @R0,A
    MOV     R0,#LOW (XFR_PADMOD3)
    MOVX    @R0,A
```

```
    MOV     R0,#LOW (XFR_OPTION1)
    MOV     A,#0C0H
    MOVX    @R0,A
    CLR     A
    MOV     R0,#LOW (XFR_OPTION2)
    MOVX    @R0,A
    MOV     R0,#LOW (XFR_XBANK)
    MOVX    @R0,A
    SETB    EA
    RET

    RSEG  ?PR?InitialCpuIO?INITIAL
    USING   0
InitialCpuIO:
    CLR     P3_1
    RET

    RSEG  ?PR?InitialEeprom?INITIAL
    USING   0
InitialEeprom:
    CLR     A
    MOV     R7,A
?C0006:
    MOV     A,R7
    MOV     DPTR,#EepromDefault_table
    MOVC    A,@A+DPTR
    MOV     R6,A
    MOV     A,#TrmBuf+01H
    ADD     A,R7
    MOV     R0,A
    MOV     A,R6
    MOV     @R0,A
    INC     R7
    CJNE    R7,#05H,?C0006
?C0007:
    CLR     A
    MOV     R7,A
    MOV     R6,A
    MOV     R5,#05H
    LCALL   _EepromWrite
    MOV     R0,#TrmBuf+01H
    MOV     @R0,#012H
    INC     R0
    MOV     @R0,#01H
    MOV     R7,#0FEH
```

```
    MOV     R6,#03H
    MOV     R5,#02H
    LCALL   _EepromWrite
    RET

    RSEG  ?PR?Initial4094?INITIAL
    USING   0
Initial4094:
    CLR     P1_1
    CLR     P1_2
    CLR     P1_0
    CLR     A
    MOV     R7,A
    LCALL   _Output4094
    RET

    RSEG  ?PR?RecallStatus?INITIAL
    USING   0
RecallStatus:
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    LCALL   _Disp7Seg
    CLR     P2_4
    CLR     P2_5
    CLR     P2_6
    CLR     P2_7
    JNB     FgSingleFlag,?C0011
    SETB    P2_4
    SJMP    ?C0012
?C0011:
    CLR     P2_4
?C0012:
    JNB     FgSequentFlag,?C0013
    SETB    P2_5
    SJMP    ?C0014
?C0013:
    CLR     P2_5
?C0014:
    JNB     FgRandomFlag,?C0015
    SETB    P2_6
    SJMP    ?C0016
?C0015:
    CLR     P2_6
?C0016:
```

```
    JNB     FgBriefFlag,?C0017
    SETB    P2_7
    RET
?C0017:
    CLR     P2_7
?C0019:
    RET

    RSEG  ?PR?InitialVariable?INITIAL
    USING   0
InitialVariable:
    MOV     R0,#KeyData
    MOV     @R0,#0FFH
    CLR     FgEndMusic
    CLR     FgStartMusic
    MOV     R0,#MusicNumber
    MOV     A,@R0
    MOV     R7,A
    MOV     B,#0AH
    DIV     AB
    MOV     B,#0AH
    MUL     AB
    MOV     R6,A
    CLR     C
    MOV     A,R7
    SUBB    A,R6
    MOV     @R0,A
    MOV     R0,#LastBuffer
    MOV     @R0,A
    RET

    END
```

3. eeprom.a51

```
$NOMOD51

NAME    EEPROM

$INCLUDE (mtv212m.inc)

?PR?EepromCommonRead?EEPROM     SEGMENT CODE
?PR?EepromCommonStore?EEPROM    SEGMENT CODE
?PR?_EepromRead?EEPROM          SEGMENT CODE
?DT?_EepromRead?EEPROM          SEGMENT DATA OVERLAYABLE
```

```
?PR?_EepromWrite?EEPROM          SEGMENT CODE
?DT?_EepromWrite?EEPROM          SEGMENT DATA OVERLAYABLE
    EXTRN   IDATA (TrmBuf)
    EXTRN   IDATA (ByteCnt)
    EXTRN   IDATA (SlvAdr)
    EXTRN   BIT (FgSingleFlag)
    EXTRN   BIT (FgSequentFlag)
    EXTRN   BIT (FgRandomFlag)
    EXTRN   BIT (FgBriefFlag)
    EXTRN   IDATA (MusicNumber)
    EXTRN   CODE (SendData)
    EXTRN   CODE (SendEepromData)
    EXTRN   CODE (RcvData)
    PUBLIC  _EepromWrite
    PUBLIC  _EepromRead
    PUBLIC  EepromCommonStore
    PUBLIC  EepromCommonRead

    RSEG  ?DT?_EepromRead?EEPROM
?_EepromRead?BYTE:
     count?241:   DS   1

    RSEG  ?DT?_EepromWrite?EEPROM
?_EepromWrite?BYTE:
 SubAddress?342:   DS   2
     count?343:   DS   1
    ORG  3
         i?344:   DS   1
 FirstValue?345:   DS   1

    RSEG  ?PR?EepromCommonRead?EEPROM
    USING   0
EepromCommonRead:
    CLR     A
    MOV     R7,A
    MOV     R6,A
    MOV     R5,#05H
    LCALL   _EepromRead
    MOV     R0,#TrmBuf
    MOV     A,@R0
    MOV     R0,#MusicNumber
    MOV     @R0,A
    MOV     R0,#TrmBuf+01H
    MOV     A,@R0
    ADD     A,#0FFH
```

```
        MOV     FgSingleFlag,C
        INC     R0
        MOV     A,@R0
        ADD     A,#0FFH
        MOV     FgSequentFlag,C
        INC     R0
        MOV     A,@R0
        ADD     A,#0FFH
        MOV     FgRandomFlag,C
        INC     R0
        MOV     A,@R0
        ADD     A,#0FFH
        MOV     FgBriefFlag,C
        RET

        RSEG  ?PR?EepromCommonStore?EEPROM
        USING   0
EepromCommonStore:
        MOV     C,FgSingleFlag
        CLR     A
        RLC     A
        MOV     R0,#TrmBuf+01H
        MOV     @R0,A
        MOV     C,FgSequentFlag
        CLR     A
        RLC     A
        INC     R0
        MOV     @R0,A
        MOV     C,FgRandomFlag
        CLR     A
        RLC     A
        INC     R0
        MOV     @R0,A
        MOV     C,FgBriefFlag
        CLR     A
        RLC     A
        INC     R0
        MOV     @R0,A
        MOV     R7,#01H
        MOV     R6,#00H
        MOV     R5,#04H
        LCALL   _EepromWrite
        RET

        RSEG  ?PR?_EepromRead?EEPROM
```

```
    USING   0
_EepromRead:
    MOV     count?241,R5
    CLR     C
    MOV     A,R6
    SUBB    A,#01H
    JNC     ?C0003
    MOV     R0,#SlvAdr
    MOV     @R0,#0A0H
    SJMP    ?C0004
?C0003:
    CLR     C
    MOV     A,R6
    SUBB    A,#02H
    JNC     ?C0005
    MOV     R0,#SlvAdr
    MOV     @R0,#0A2H
    CLR     A
    ADD     A,R7
    MOV     R7,A
    MOV     A,#0FFH
    ADDC    A,R6
    MOV     R6,A
    SJMP    ?C0004
?C0005:
    CLR     C
    MOV     A,R6
    SUBB    A,#03H
    JNC     ?C0007
    MOV     R0,#SlvAdr
    MOV     @R0,#0A4H
    CLR     A
    ADD     A,R7
    MOV     R7,A
    MOV     A,#0FEH
    ADDC    A,R6
    MOV     R6,A
    SJMP    ?C0004
?C0007:
    MOV     R0,#SlvAdr
    MOV     @R0,#0A6H
    CLR     A
    ADD     A,R7
    MOV     R7,A
    MOV     A,#0FDH
```

```
        ADDC    A,R6
        MOV     R6,A
    ?C0004:
        MOV     R0,#TrmBuf
        MOV     A,R7
        MOV     @R0,A
        MOV     R0,#ByteCnt
        MOV     @R0,#01H
        LCALL   SendData
        MOV     R0,#ByteCnt
        MOV     @R0,count?241
        LCALL   RcvData
        RET

        RSEG  ?PR?_EepromWrite?EEPROM
        USING   0
    _EepromWrite:
        MOV     SubAddress?342,R6
        MOV     SubAddress?342+01H,R7
        MOV     count?343,R5
        CLR     C
        MOV     A,SubAddress?342
        SUBB    A,#01H
        JNC     ?C0010
        MOV     R0,#SlvAdr
        MOV     @R0,#0A0H
        SJMP    ?C0011
    ?C0010:
        CLR     C
        MOV     A,SubAddress?342
        SUBB    A,#02H
        JNC     ?C0012
        MOV     R0,#SlvAdr
        MOV     @R0,#0A2H
        CLR     A
        ADD     A,SubAddress?342+01H
        MOV     SubAddress?342+01H,A
        MOV     A,#0FFH
        ADDC    A,SubAddress?342
        MOV     SubAddress?342,A
        SJMP    ?C0011
    ?C0012:
        CLR     C
        MOV     A,SubAddress?342
        SUBB    A,#03H
```

```
    JNC     ?C0014
    MOV     R0,#SlvAdr
    MOV     @R0,#0A4H
    CLR     A
    ADD     A,SubAddress?342+01H
    MOV     SubAddress?342+01H,A
    MOV     A,#0FEH
    ADDC    A,SubAddress?342
    MOV     SubAddress?342,A
    SJMP    ?C0011
?C0014:
    MOV     R0,#SlvAdr
    MOV     @R0,#0A6H
    CLR     A
    ADD     A,SubAddress?342+01H
    MOV     SubAddress?342+01H,A
    MOV     A,#0FDH
    ADDC    A,SubAddress?342
    MOV     SubAddress?342,A
?C0011:
    MOV     R0,#TrmBuf+01H
    MOV     A,@R0
    MOV     FirstValue?345,A
    MOV     i?344,#01H
?C0016:
    MOV     A,i?344
    SETB    C
    SUBB    A,count?343
    JNC     ?C0017
    MOV     R0,#TrmBuf
    MOV     A,SubAddress?342+01H
    MOV     @R0,A
    MOV     A,#TrmBuf
    ADD     A,i?344
    MOV     R0,A
    MOV     A,@R0
    MOV     R0,#TrmBuf+01H
    MOV     @R0,A
    MOV     R0,#ByteCnt
    MOV     @R0,#02H
    LCALL   SendEepromData
    INC     SubAddress?342+01H
    MOV     A,SubAddress?342+01H
    JNZ     ?C0020
    INC     SubAddress?342
```

```
?C0020:
    INC     i?344
    SJMP    ?C0016
?C0017:
    MOV     R0,#TrmBuf+01H
    MOV     @R0,FirstValue?345
    RET

    END
```

4. iic.a51

```
$NOMOD51

NAME    IIC

$INCLUDE (mtv212m.inc)

?PR?SendData?IIC          SEGMENT CODE
?DT?SendData?IIC          SEGMENT DATA OVERLAYABLE
?BI?SendData?IIC          SEGMENT BIT OVERLAYABLE
?PR?SendEepromData?IIC        SEGMENT CODE
?DT?SendEepromData?IIC    SEGMENT DATA OVERLAYABLE
?BI?SendEepromData?IIC        SEGMENT BIT OVERLAYABLE
?PR?RcvData?IIC           SEGMENT CODE
?DT?RcvData?IIC           SEGMENT DATA OVERLAYABLE
?PR?_GoMaster?IIC         SEGMENT CODE
?DT?_GoMaster?IIC         SEGMENT DATA OVERLAYABLE
?PR?_SendByte?IIC         SEGMENT CODE
?DT?_SendByte?IIC         SEGMENT DATA OVERLAYABLE
?BI?_SendByte?IIC         SEGMENT BIT OVERLAYABLE
?PR?SendStop?IIC          SEGMENT CODE
?PR?IICDelay?IIC          SEGMENT CODE
    EXTRN   IDATA (TrmBuf)
    EXTRN   IDATA (ByteCnt)
    EXTRN   IDATA (SlvAdr)
    EXTRN   CODE (_DelayX1ms)
    PUBLIC  RcvData
    PUBLIC  SendEepromData
    PUBLIC  SendData

    RSEG  ?DT?SendData?IIC
?SendData?BYTE:
        i?040:      DS   1
        j?041:      DS   1
```

```
    RSEG  ?BI?SendData?IIC
?SendData?BIT:
    no_ack?042:     DBIT   1

    RSEG  ?DT?SendEepromData?IIC
?SendEepromData?BYTE:
        i?143:      DS   1
        j?144:      DS   1

    RSEG  ?BI?SendEepromData?IIC
?SendEepromData?BIT:
    no_ack?145:     DBIT   1

    RSEG  ?DT?RcvData?IIC
?RcvData?BYTE:
        i?246:      DS   1
        j?247:      DS   1
      bval?248:     DS   1

    RSEG  ?DT?_GoMaster?IIC
?_GoMaster?BYTE:
 slave_addr?349:    DS   1

    RSEG  ?DT?_SendByte?IIC
?_SendByte?BYTE:
      bval?450:     DS   1
    ORG  1
        i?451:      DS   1

    RSEG  ?BI?_SendByte?IIC
?_SendByte?BIT:
    no_ack?452:     DBIT   1

    RSEG  ?PR?SendData?IIC
    USING   0
SendData:
    SETB    no_ack?042
    CLR     A
    MOV     i?040,A
?C0001:
    MOV     R0,#SlvAdr
    MOV     A,@R0
    MOV     R7,A
    LCALL   _GoMaster
```

```
    JNC     ?C0004
    LCALL   SendStop
    SJMP    ?C0003
?C0004:
    CLR     no_ack?042
    CLR     A
    MOV     j?041,A
?C0005:
    MOV     A,j?041
    CLR     C
    MOV     R0,#ByteCnt
    SUBB    A,@R0
    JNC     ?C0006
    MOV     A,#TrmBuf
    ADD     A,j?041
    MOV     R0,A
    MOV     A,@R0
    MOV     R7,A
    LCALL   _SendByte
    JNC     ?C0007
    SETB    no_ack?042
    SJMP    ?C0006
?C0007:
    INC     j?041
    SJMP    ?C0005
?C0006:
    LCALL   SendStop
    JNB     no_ack?042,?C0002
?C0003:
    INC     i?040
    MOV     A,i?040
    CLR     C
    SUBB    A,#0AH
    JC      ?C0001
?C0002:
    MOV     C,no_ack?042
?C0010:
    RET

    RSEG  ?PR?SendEepromData?IIC
    USING   0
SendEepromData:
    SETB    no_ack?145
    CLR     A
    MOV     i?143,A
```

```
?C0011:
    MOV     R0,#SlvAdr
    MOV     A,@R0
    MOV     R7,A
    LCALL   _GoMaster
    JNC     ?C0014
    LCALL   SendStop
    MOV     R7,#0AH
    MOV     R6,#00H
    LCALL   _DelayX1ms
    SJMP    ?C0013
?C0014:
    CLR     no_ack?145
    CLR     A
    MOV     j?144,A
?C0015:
    MOV     A,j?144
    CLR     C
    MOV     R0,#ByteCnt
    SUBB    A,@R0
    JNC     ?C0016
    MOV     A,#TrmBuf
    ADD     A,j?144
    MOV     R0,A
    MOV     A,@R0
    MOV     R7,A
    LCALL   _SendByte
    JNC     ?C0017
    SETB    no_ack?145
    SJMP    ?C0016
?C0017:
    INC     j?144
    SJMP    ?C0015
?C0016:
    LCALL   SendStop
    JNB     no_ack?145,?C0012
?C0019:
    MOV     R7,#0AH
    MOV     R6,#00H
    LCALL   _DelayX1ms
?C0013:
    INC     i?143
    MOV     A,i?143
    CLR     C
    SUBB    A,#0AH
```

```
    JC      ?C0011
?C0012:
    MOV     C,no_ack?145
?C0020:
    RET

    RSEG  ?PR?RcvData?IIC
    USING   0
RcvData:
    MOV     R0,#SlvAdr
    MOV     A,@R0
    INC     A
    MOV     R7,A
    LCALL   _GoMaster
    JNC     ?C0021
    LCALL   SendStop
    SETB    C
    RET
?C0021:
    CLR     A
    MOV     i?246,A
?C0023:
    MOV     A,i?246
    CLR     C
    MOV     R0,#ByteCnt
    SUBB    A,@R0
    JNC     ?C0024
    CLR     A
    MOV     bval?248,A
    MOV     j?247,A
?C0026:
    SETB    P3_0
    LCALL   IICDelay
    MOV     A,bval?248
    ADD     A,ACC
    MOV     bval?248,A
    JNB     P3_5,?C0029
    ORL     bval?248,#01H
?C0029:
    CLR     P3_0
    LCALL   IICDelay
    INC     j?247
    MOV     A,j?247
    CLR     C
    SUBB    A,#08H
```

```
    JC      ?C0026
?C0027:
    MOV     R0,#ByteCnt
    MOV     A,@R0
    DEC     A
    XRL     A,i?246
    JNZ     ?C0030
    SETB    P3_5
    SJMP    ?C0031
?C0030:
    CLR     P3_5
?C0031:
    LCALL   IICDelay
    SETB    P3_0
    LCALL   IICDelay
    CLR     P3_0
    LCALL   IICDelay
    SETB    P3_5
    LCALL   IICDelay
    MOV     A,#TrmBuf
    ADD     A,i?246
    MOV     R0,A
    MOV     @R0,bval?248
    INC     i?246
    SJMP    ?C0023
?C0024:
    LCALL   SendStop
    CLR     C
?C0022:
    RET

    RSEG  ?PR?_GoMaster?IIC
    USING   0
_GoMaster:
    MOV     slave_addr?349,R7
    JNB     P3_0,?C0033
    JB      P3_5,?C0032
?C0033:
    SETB    C
    RET
?C0032:
    CLR     P3_5
    LCALL   IICDelay
    CLR     P3_0
    LCALL   IICDelay
```

```
    MOV     R7,slave_addr?349
    LCALL   _SendByte
    CLR     C
?C0034:
    RET

    RSEG  ?PR?_SendByte?IIC
    USING   0
_SendByte:
    MOV     bval?450,R7
    CLR     no_ack?452
    CLR     A
    MOV     i?451,A
?C0035:
    MOV     A,bval?450
    JNB     ACC.7,?C0038
    SETB    P3_5
    SJMP    ?C0039
?C0038:
    CLR     P3_5
?C0039:
    MOV     A,bval?450
    ADD     A,ACC
    MOV     bval?450,A
    LCALL   IICDelay
    SETB    P3_0
    LCALL   IICDelay
    CLR     P3_0
    LCALL   IICDelay
    INC     i?451
    MOV     A,i?451
    CLR     C
    SUBB    A,#08H
    JC      ?C0035
?C0036:
    SETB    P3_5
    LCALL   IICDelay
    SETB    P3_0
    LCALL   IICDelay
    JNB     P3_5,?C0040
    SETB    no_ack?452
?C0040:
    CLR     P3_0
    LCALL   IICDelay
    MOV     C,no_ack?452
```

```
?C0041:
    RET

    RSEG  ?PR?SendStop?IIC
    USING   0
SendStop:
    CLR     P3_5
    CLR     P3_0
    LCALL   IICDelay
    SETB    P3_0
    LCALL   IICDelay
    SETB    P3_5
    LCALL   IICDelay
    RET

    RSEG  ?PR?IICDelay?IIC
    USING   0
IICDelay:
    NOP
    NOP
    NOP
    NOP
    RET

    END
```

12-4　随播随显示曲目编号

目的：将播放中的曲目编号随时显示出来，以便清楚地表示目前播放的状态。

软件构建的思维与解决方法

功能模式有单曲循环、顺序播放、随机选曲以及播放简介等，当输入数字键则显示器将会立即显示出所输入的数字，一旦进入功能模式，则显示器不再更新目前所播放的曲目，不是会感觉怪怪的，为何现在所播放的曲目并不是显示器上的编号呢？例如，一旦设定顺序播放功能或随机选曲功能或播放简介功能，当播完所设定的曲目编号后，则将会改变演奏的曲目，因此，才想到应该要将显示器、更新目前所播放的曲目，这样，是否才更合理呢？但如何实现这个构思呢？其实也很简单，只要将所有调用 Music(MusicNumber)函数中的语句内，在 Music(MusicNumber)函数

的前面再调用 Disp7Seg(MusicNumber)函数即可，也就是显示出目前的曲编号后，才进行音乐的演奏，在 input.c 模块下的 StartHander( )函数其程序如下：

```
/*****************************/
void StartHandler(void)
{
   int lastvalue,currentvalue;

   //detect start key
   InputStart( );

   if ( KeyData == START_KEY )
   {
      FgEndMusic =0;              //int0:end play music
      FgNextMusic=0;              //timer0:FgNextMusic=1

      if ( FgSingleFlag )         //单曲循环
      {
         while( 1 )
         {
            Disp7Seg(MusicNumber);
            TrmBuf[ 1] = MusicNumber;
            EepromWrite(MusicNumber_ADDR,1);

            Music(MusicNumber);
            if (FgEndMusic==1) return;
            DelayX10ms(100);
         }
      }
      else if ( FgSequentFlag )      //顺序播放
      {
         while( 1 )
         {
            Disp7Seg(MusicNumber);
            TrmBuf[ 1] = MusicNumber;
            EepromWrite(MusicNumber_ADDR,1);

            Music(MusicNumber);
            if (FgEndMusic==1) return;

            MusicNumber++;
            if ( MusicNumber > LASTMUSIC )
               MusicNumber=1;
            DelayX10ms(100);
         }
```

```
}
else if ( FgRandomFlag )        //随机选曲
{
    FgStartMusic=0;
    DelayX10ms(50);
    srand(PulseCount);

  while( 1 )
  {
    currentvalue=rand(); //产生一个随机数
    currentvalue%=100;   //取最后两位数 0~99

    //currentvalue=1~LASTMUSIC
    if ( (currentvalue<=LASTMUSIC)&&(currentvalue>0) )
    {
      //和上次不一样
      if (currentvalue!=lastvalue)
      {
        MusicNumber=currentvalue;
        Disp7Seg(MusicNumber);
        TrmBuf[ 1] = MusicNumber;
        EepromWrite(MusicNumber_ADDR,1);

        Music(MusicNumber);
        lastvalue=currentvalue;
        if (FgEndMusic==1) return;
        DelayX10ms(100);
      }
    }
  }
}
else if ( FgBriefFlag )         //播放简介
{
  while( 1 )
  {
    Disp7Seg(MusicNumber);
    TrmBuf[ 1] = MusicNumber;
    EepromWrite(MusicNumber_ADDR,1);

    FgNextMusic=0;          //timer0:FgNextMusic=1
    MusicLength=0;
    TR0 = 1;
    Music(MusicNumber);
    TR0 = 0;
```

```
            if (FgEndMusic==1) return;

            MusicNumber++;
            if ( MusicNumber > LASTMUSIC )
              MusicNumber=1;
            DelayX10ms(100);
          }
        }
        else                              //单曲播放一次
        {
          Disp7Seg(MusicNumber);
          TrmBuf[ 1] = MusicNumber;
          EepromWrite(MusicNumber_ADDR,1);

          Music(MusicNumber);
          if (FgEndMusic==1) return;
          DelayX10ms(100);
        }
      }
    }
```