合肥工業大學

宣城校区

课 程 设 计 报 告

课 程 设 计 名 称 微机原理与接口技术

课 程 设 计 题 目 简易电子琴、LED 16×16 点阵

院　　系 _____

专 业 班 级 _____

学 生 姓 名 _____

学　　号 _____

指 导 教 师 _____

实 验 地 点 计算中心 103

完 成 日 期 2019 年 12 月 28 日

目 录

设计 1：简易电子琴

1. 设计任务

掌握蜂鸣器的使用方法；掌握蜂鸣器的不同发音的方法。

2. 设计目标

借助可编程并行接口芯片 8255，通过 PC 机编程在 SUN ES86PCIU+实验仪平台上实现了一个简易的电子琴。

3. 设计原理

1) 对蜂鸣器输入不同频率的方波，会发出各个音阶的声音；
2) 通过编程设定或按键，由 8255 芯片控制发出不同频率的方波，即产生不同音阶的声音。

我们主要采用了可编程并行接口芯片 8255，通过 8255 的 PAD,使实验仪 F5 区的 1∽7 号键由低到高发出 1-7 的音阶。

设计有关的原理图如图 3.1 所示。

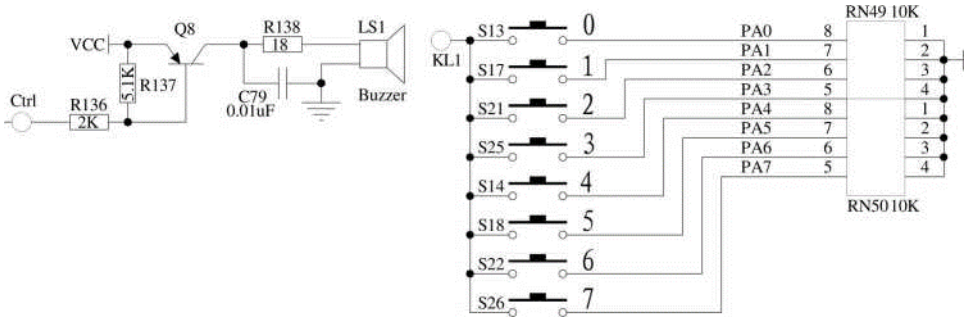

图 3. 1 简易电子琴实验原理图

主机连线说明如表 3.1 所示，连线的目的是编程或按键控制 F5 区的按键使 F8 区的蜂鸣器发出不同音阶的声音。

表 3. 1 简易电子琴连线说明

D3 区：CS、AO、Al	A3 区：CS1、AO、A1
D3 区：PC7	F8 区：Ctrl
D3 区：JP23 (PA 口)	F5 区：JP37 (A)
Cl 区：GND	F5 区：KL1

4. 设计实现

4.1. 硬件实现

我们按照原理图与实验连线说明在实验仪上连线，具体如图 4.1 所示。

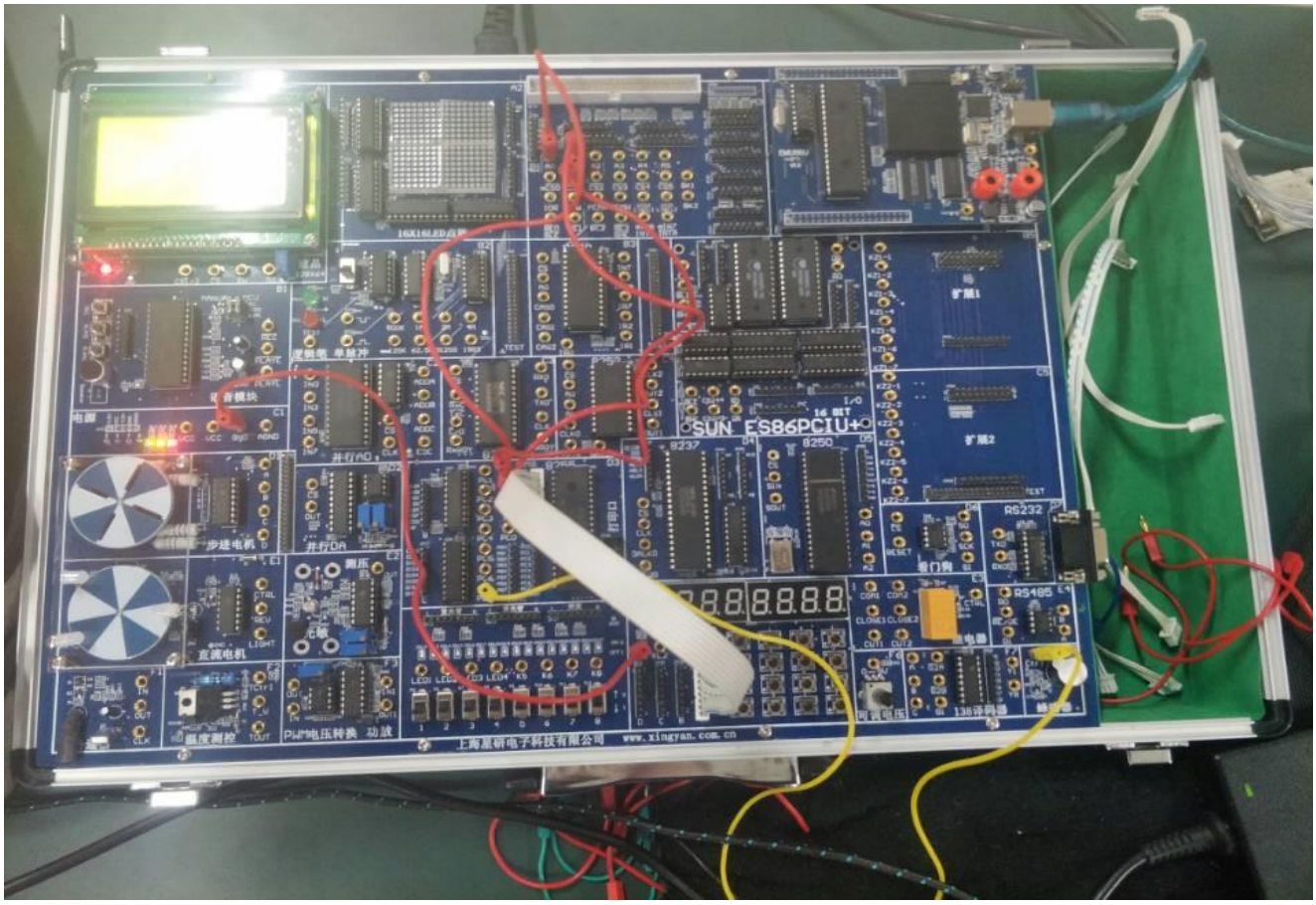

图 4. 1 实验仪连线

4.2. 软件实现

本次设计的系统流程图如图 4.2 所示。

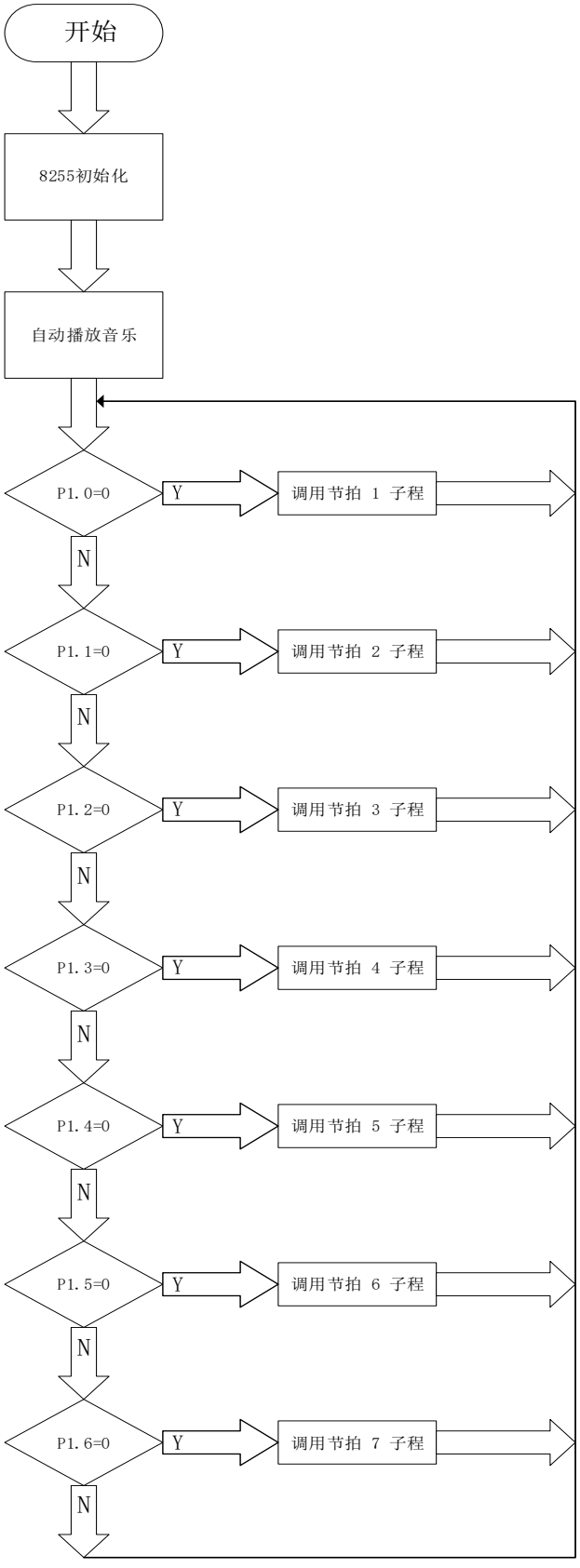

图 4.2 简易电子琴系统执行流程

本次设计所用代码放在附录 A 中以供查阅，接下来对代码做一些必要的说明。

程序首先定义了最终可执行文件的格式，由于本次实验的代码行数较少、访存空间较小，因此采用了 tiny 格式。接着设置控制口地址、输入口地址与输出口地址，需要注意的是，不同类型的实验仪的口地址不一定相同，在编程时应予以注意。程序所用的栈空间为 100（dw），数据段定义了由各函数组成的《两只老虎》的乐谱。程序的功能部分主要由以下七个部分组成：

1) 调用初始化 8255 芯片以及播放默认乐曲的函数。显然，为了 8255 芯片可以正常工作，我们需要根据实验的实际情况，编程决定 8255 芯片的方式选择控制字与工作方式。本部分主要由两个函数调用组成：

```
call init8255          ;8255  初始化
call demo                    ;播放一段音乐
```

2) 按键查询有关的一系列操作。我们的设计是先由 CPU 发送一段默认的乐谱給 8255 芯片，使之控制蜂鸣器演奏乐曲，再允许操作者做其他演奏。本部分主要由 start1~start7 等 7 个分支组成，其中 start1 完成有无按键按下的查询并检测所按键是否为 1，其余 6 个语句块完成的功能基本相同：查询对应的按键是否按下。

3) 自动播放乐曲的主函数。为了实现自动播放的功能，我们实现了一个函数 demo，主要定义了歌曲的节拍数并顺序发出各个音阶。值得一提的是，我们使用寄存器 CX 来保存节拍数，为了避免在函数调用时其值被覆盖，因此需要提前压入堆栈中保护。

4) 响应按键的一系列函数。这个部分主要由 music1~music7 等 7 个函数组成，它们的功能基本相似，区别仅在于为了能够区别不同的音阶而设置的延时，即蜂鸣器在一定时间内震荡的频率。为了保证音阶可以方便地被识别，我们应当取各震荡周期基本一致。

5) 响应自动播放的一系列函数。本部分代码结构与上一部分在形式上具有很大的相似性，主要由 m1~m7 等 7 个不同函数组成，所不同的是此部分代码的功能是响应自动播放。不得不说的是，各个函数为了使其对应的音阶更明显，通过寄存器 CX 设置 LOOP 循环的次数，即延时时间。另外，上部分与本部分的延时如何调整才能使音阶达到更好的效果需要

一定的音乐知识，并加以调试修改。

6) 初始化 8255 芯片与读写控制有关的函数。这个部分的代码比较简单，由 w_l、w_h 与 init8255 等三个函数构成，其中 w_l 定义向 PC 口的第 7 位写 0 时则蜂鸣器响，w_h 的功能相反，而 init8255 定义了 8255 芯片的方式选择控制字与具体采用的芯片口，PC 口第 7 位用于输出，PA 口用于输入。

7) 为了满足拍子的要求而构建的一些列延时函数。最后这部分的代码具体实现了各个延时函数，本部分的关键在于函数复用，即使用已定义的函数完成新的功能。

至此，本次实验所用代码的总体架构介绍完毕。

5. 设计结果

本实验的演示分为两个部分，首先是对 8255 芯片编程使之发布不同频率的方波，控制蜂鸣器发出不同的音阶，演奏一首完整的简易乐曲；其次是通过按控制板上的键盘，手动演奏乐曲。

我们经过不断地调试，使电子琴最终完整地演奏了乐曲《两只老虎》，音阶明晰、音色较好，整体仿真效果较好。

6. 设计心得与改进方法

在这次课程设计中，我们通过编程控制 8255 芯片，对电子琴主体部分的电路进行仿真设计，实现了一个具备自动播放与自主演奏功能的简易电子琴。在实验过程中，我们遇到了许多疑难，尤其是代码调试阶段，譬如：未正确设置端口地址导致程序无法载入 8255 芯片；如何调整曲谱的节拍，使之更加接近标准的演奏效果；如何在不采用外部中断源的情况下，利用软件产生长短适宜的间隔；等等。在经历了一系列的失败与调试后，我们最终达取得了想要的效

果。

必须承认的是，我们的实验方法还存在值得改进之处。为了产生更为精确的时间间隔，使乐曲拍子的仿真效果更好，可以考虑使用可编程计数器/定时器 8253 芯片，通过编程来产生不同频率的方波，将方波输入蜂鸣器来产生不同频率、不同声强的音阶；蜂鸣器同时受 8255 芯片控制，在控制口允许的情况下才发声。

设计 2：LED 16×16 点阵

1 设计任务

1) 熟悉 8255 的功能，了解点阵显示的原理及控制方法；
2) 学会使用 LED 点阵，通过编程显示不同字符。

2 设计目标

借助可编程并行接口芯片 8255，通过 PC 机编程在 SUN ES86PCIU+实验仪的 16×16 LED 点阵上自下而上循环显示“欢迎使用星研实验仪”字样。本设计的扩展实验为：修改程序与实验线路，在 LED 点阵上从左至右显示指定的字符串。

3 设计原理

本设计的实验原理图如图 3.1 所示。

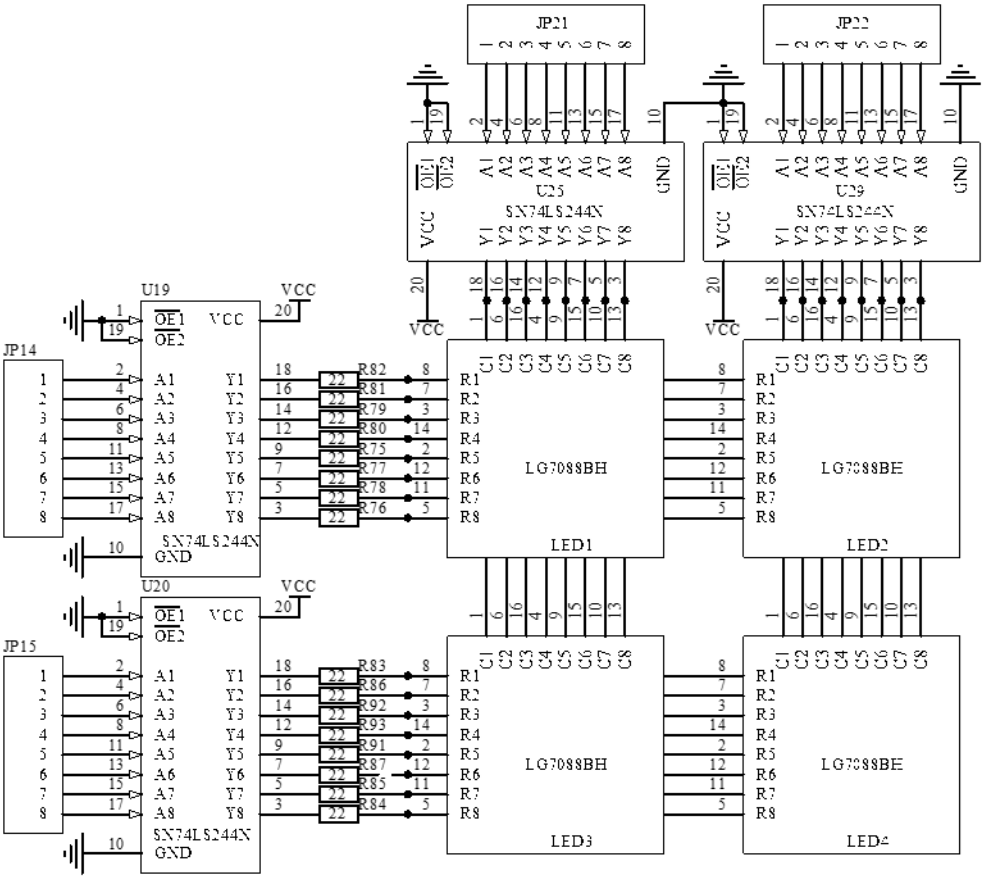

图 3. 1 LED 16×16 点阵实验原理图

下面对图 3.1 所示的各个区域与接口进行简单介绍。

3.1 A2 区

A2 区为 16×16 LED 实验电路。

JP14、JP15 组成 16 根行扫描线；JP21、JP22 组成 16 根列扫描线。

3.2 A3 区

A3 区是 CPU 总线与片选区。

3.2.1 CPU 总线

JP28：地址线 A0..A7；

JP32：地址线 A8..A15；

JP29：地址线 A0..A9；

JP33：地址线 A8..A17；　　根据数据总线宽度，选择合适的地址线。

JP41、JP42：数据总线 D0..D7；

JP39、JP40：数据总线 D8..D15；

JP47、JP48：数据总线 D16..D23；

JP45、JP46：数据总线 D24..D31；

控制线：IOR、IOW、MEMR、MEMW、HOLD、HLDA、BLE、BHE、INTR、INTA；

BK1、BK2：备用片选区。

3.2.2 片选区

片选情况如表 3.1 所示。

表 3. 1 A3 区片选情况

片选	地址范围	说明
mCS0	80000H～BFFFFH	存贮器芯片的片选，16 位数据总线
CS1	0270H～027FH	I/O 芯片的片选，8 位数据总线
CS2	0260H～026FH	
CS3	0250H～025FH	
CS4	0240H～024FH	
CS5	0230H～023FH	I/O 芯片的片选，16 位数据总线

4 设计实现

4.1 硬件实现

在实验仪上按照表 4.1 所示进行连线。

表 4. 1 LED 16×16 点阵主机连线

D3 区：CS（8255）、A0、A1	——	A3 区：CS1、A0、A1
D3 区：JP23(PA)、JP20(PB)	——	A2 区：JP21、JP22（列输出线）
B4 区：JP57(D0..D7)	——	A3 区：JP42(D0..D7)
B4 区：JP56(D8..D15)	——	A3 区：JP40(D8..D15)
B4(I/O)区：CS273、BLE、BHE	——	A3 区：CS5、BLE、BHE
B4(I/O)区：RD、WR	——	A3 区：IOR、IOW
B4(I/O)区：JP51、JP55	——	A2 区：JP14、JP15（行输出线）

连线的目的是为了保证控制口命令与地址能够正确地载入 8255 芯片中。

4.2 软件实现

本次设计的系统流程图如图 4.1 所示。

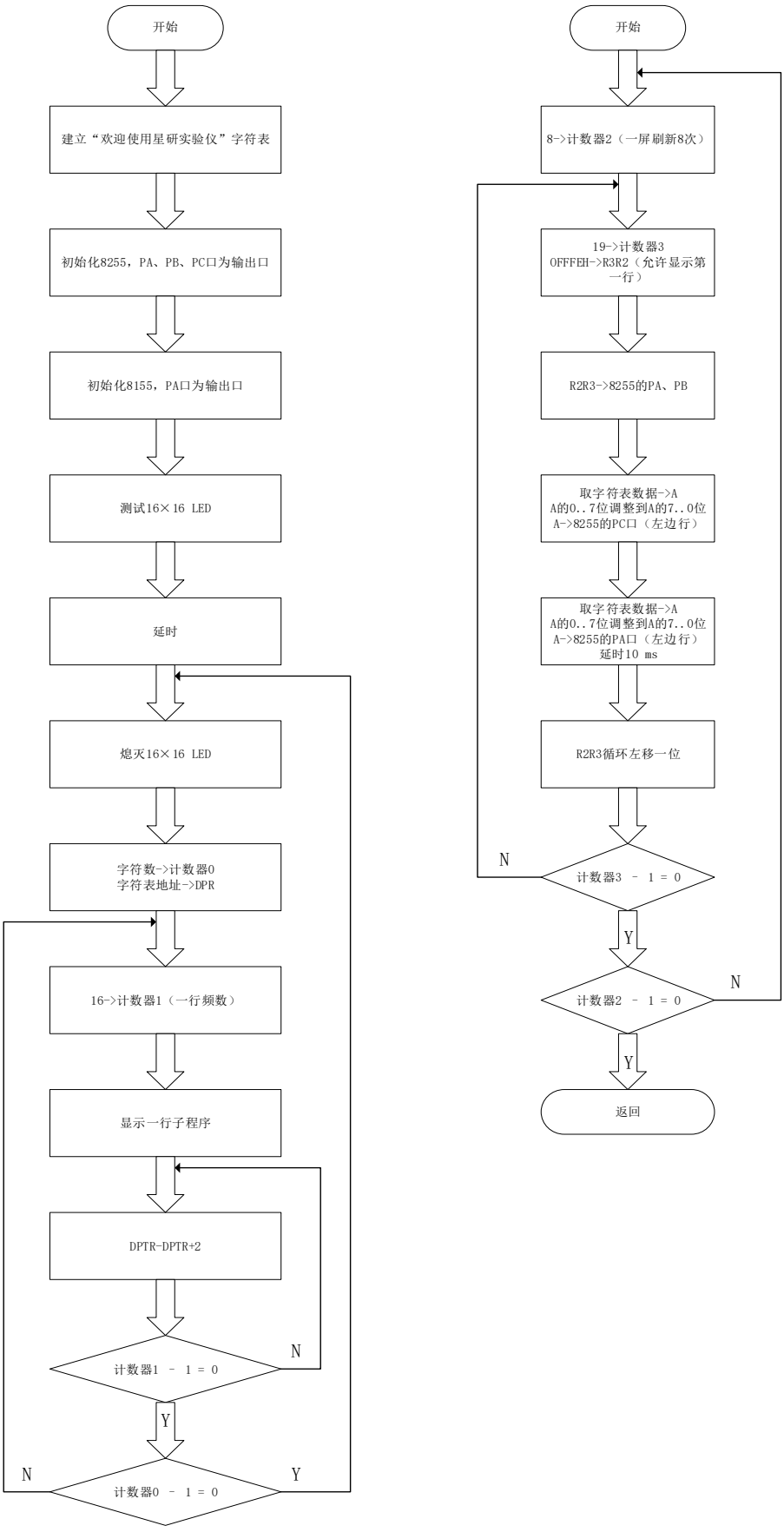

图 4.1 LED 16×16 点阵系统执行流程图

本次设计所用代码放在附录 B 中以供查阅，若需运行出样例结果，只需正确连线并执行程序即可。本节重点是分析如何重构程序，使之可以完成扩展实验，即：在 LED 点阵上从左至右显示指定的字符串。为了使分析更加自然与透彻，首先介绍与显示有关的器件与基本概念。

本设计所用实验仪的 16×16 LED 屏幕由二极管组成，当 AB 端电压差（ROW=1,LINE=0）大于一定值时二极管导通，LED 发亮。

所谓的字形码，每两个字节构成一行，输入列控制线 ROW1 和 ROW2，共 16 行。值得一提是，由于字形码自下而上滚动，需要把最后一个字流动完，所以需要一个 32 字节的全零数据，此即 NONE 为全零的作用。

下面开始正式分析整个示例程序。

4.2.1 示例代码分析

程序先定义各个 IO 口和控制口地址。

```
.MODEL TINY
ADDR_8255_PA  EQU    0270H    ;8255 PA口;列线控制端口1
ADDR_8255_PB  EQU    0271H    ;8255 PB口;列线控制端口2
ADDR_8255_C       EQU    273H        ;8255控制口
ADDR_273      EQU    230H        ;8155 PA口;行线控制端口
LINE          EQU    ADDR_273    ;行控制线
ROW1          EQU    ADDR_8255_PA ;列控制线1
ROW2          EQU    ADDR_8255_PB ;列控制线 2
```

其次，INIT_IO 函数往 8255 输入控制字，设置 8255 的 PA、PB 口为输出口；

再用测试函数 TEST_LED 测试 LED 是否可以全亮；

最后用 CLEAR 清屏，使 LED 全灭。

为了使分析不失细节性，接下来介绍程序的一些关键变量。

寄存器 CX1：将需要滚动的字符计数。

寄存器 SI：将 SI 原地址指针指向字形码数据首地址。

寄存器 CX2：将一个完整字符需要滚动的次数（16 屏）计数。

在本设计中，编程控制 LED 的刷新进而增减点阵的滚动速度。

现在分析已经到了显示一个完整字符的子程序。

寄存器 CX3：一屏刷新次数（控制滚动频率）。

寄存器 CX4：将显示一屏的全部行数（16）给计数器。

需要注意的是，将 IO 扩展口输入的最低位输入 0、其他位置 1 通过语句

LINE = FFFEH

实现之。

示例代码分析的最后，介绍实现的几个关键点。

1) 把一行的字形码送入 ROW1 和 ROW2：用 LODSB 读入当前字节码的一个字节到 PA 数据口（ROW1），再读入下一个字节到 PB 数据口（ROW2），若 LED 的列控制线引脚与 PA、PB 口的位号相反，则需要调用 ADJUST 函数将二进制数旋转 180°；
2) 将 LINE 数据循环左移（RCL）。

依次执行 1）与 2）共 16 次，每次 CX3 计数减一，显示一个完整的字形码。

当显示完一屏点阵后 SI+2，使得下一次显示的点阵为当前点阵向上滚动一行，并将 CX1 减一，若 CX1=0 则已将所有字形码滚动完毕，否则继续循环。当所有字符滚动完毕后用 JMP 指令跳转到显示第一个字符的代码，循环执行字符滚动。

4.2.2 重构代码分析

重构代码完成了设计的扩展，实现了字符串自上而下滚动到自左向右滚动的功能转变，同时修改显示字符串为“我爱❤微机”。

相较 4.2.1 节分析的示例代码，重构部分主要改变了显示一个完整字符的子程序代码，具体改动为：

1) 将 ROW2 的最低位输入 1，ROW2 的其他位以及 ROW1 所有位清 0，通过如下语句实现：

ROW2=01H,ROW1=00H

2) 把一行的字形码送入 LINE：用 LODSB 连续读入当前字节码的两个个字节到 LINE，若 LED 的行控制线引脚与 I/O 扩展口的位号相反，则需要调用 ADJUST 函数将二进制数逆置。
3) 将 LINE 数据循环左移（RCL）。

函数依次执行 2）与 3）共 16 次，每次 CX3 计数减一，显示一个完整的字

形码。

5 设计结果

按照图 3.1 所示原理图与表 4.1 所示主机接线示意，我们成功运行程序得到了示例结果。经过审慎的分析与大胆的实验，我们完成了本设计的扩展功能。具体结果如图 5.1 至图 5.4 所示。

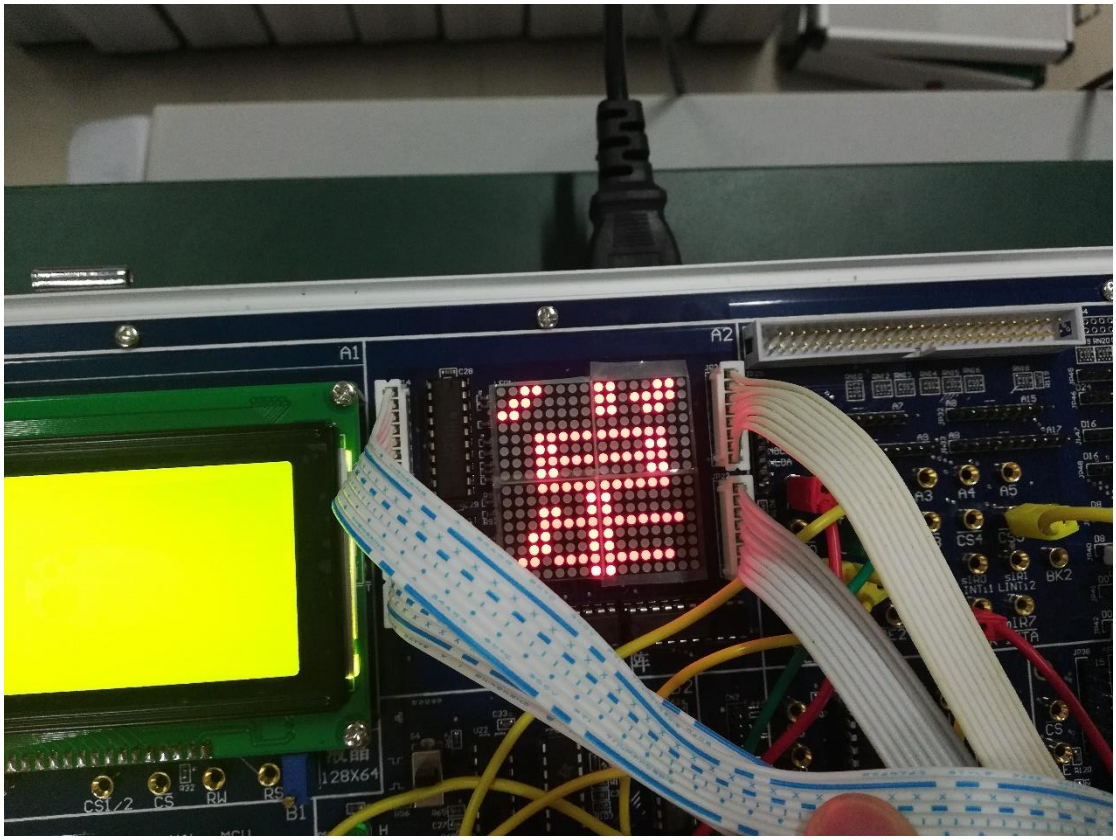

图 5. 1 示例代码运行结果

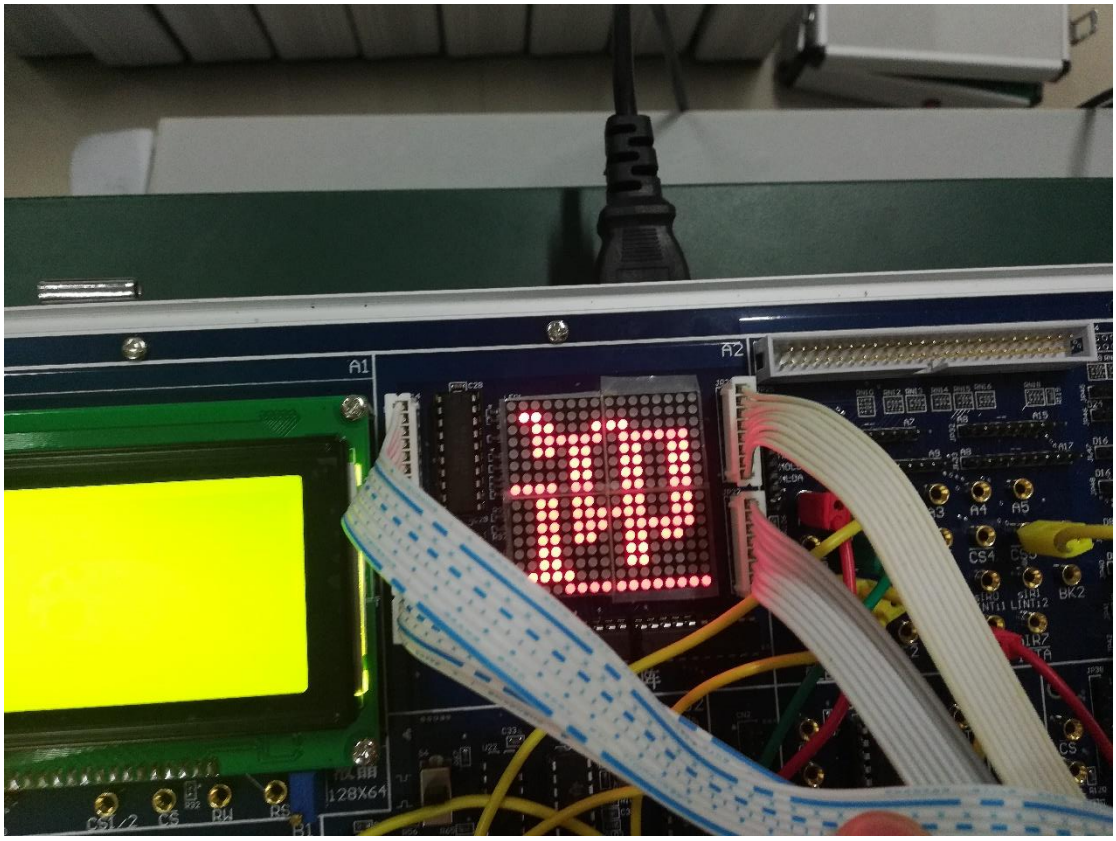

图 5.2 示例代码运行结果 2

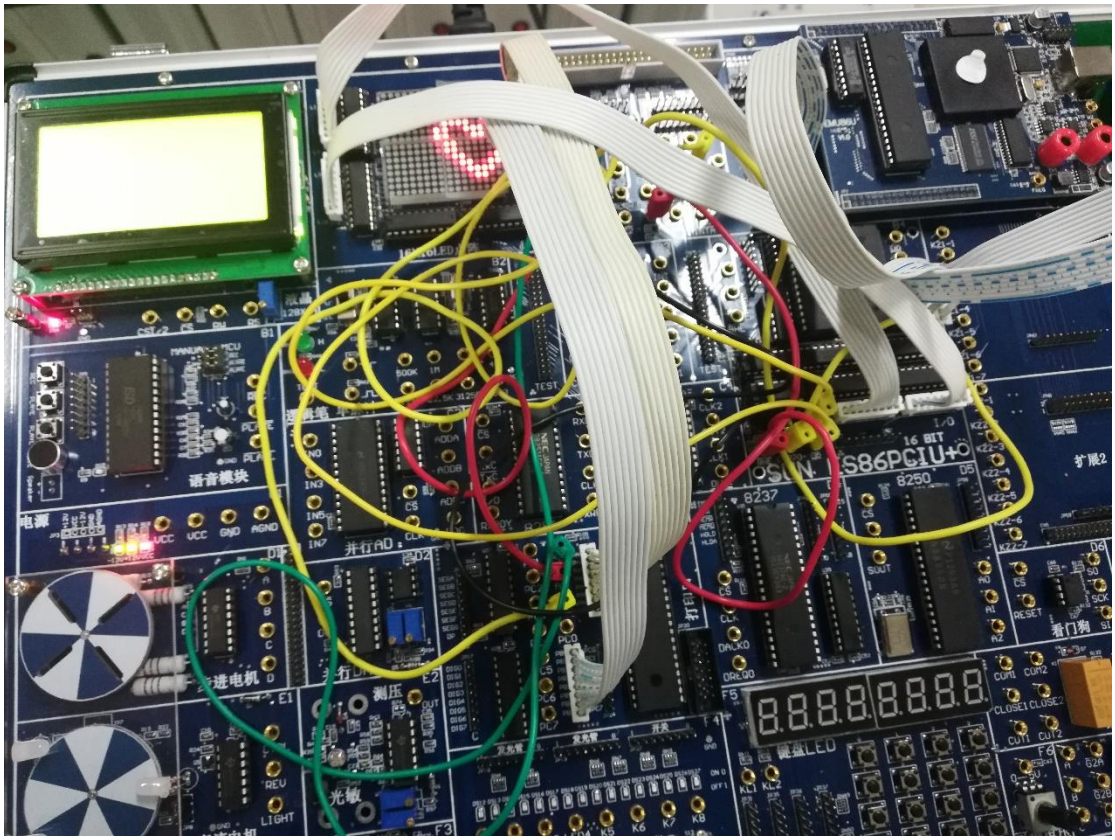

图 5.3 设计扩展结果 1

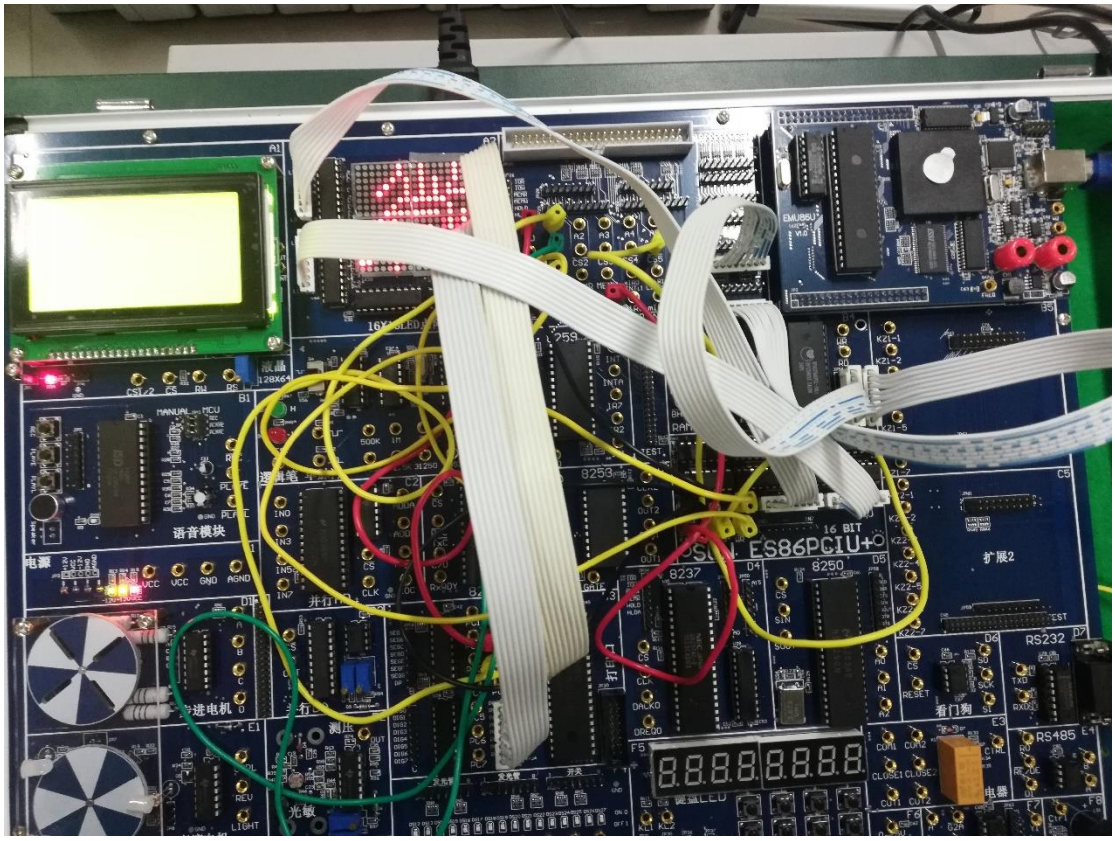

图 5.4 设计扩展结果 2

6 设计心得与改进方法

在本次课程设计中，我们经过反复的调试与大量的失败，最终成功得到了预期的实验结果。很大程度上本次设计的疑难都是在实现实验扩展时遇到的，由于汇编代码并非十分直观，而我们编写汇编程序的能力又较为稚嫩，调试工作初期进行得异常艰难。总之，得益于我们不畏失败、愈战愈勇的优良品质，以及参考网络上大量的参考资料，我们完满地实现了本次课程设计地既定目标。

本设计地改进方法可以从以下几方面考虑。

1) 精简代码、优化架构。由于我们编写汇编代码能力不够成熟，导致写了许多冗余的代码。

2) 使用可编程计数器/定时器 8253 芯片。通过编程来控制 8253 芯片产生不同频率的方波，可以更精确地增减字符串滚动的速度。

3) 高效地生成字形码。本设计所有的待显示字符串都是通过使用常量来固定字形码，进而在程序中调用。可以改进算法，使得用户只需输入给定的字符串，由程序自动将其转换为字形码，增强用户的使用体验。

附录 A

简易电子琴实现代码：

```
        .model tiny
c8255   equ  0273h
pa8255  equ 0270h
pc8255  equ 0272h
        .stack 100
        .data
music       dw   m1,q3,m2,q3,m3,q3,m1,q3,m1,q3,m2,q3,m3,q3,m1,q3
                 dw   m3,q2,m4,q2,m5,q3,m3,q2,m4,q2,m5,q3
                 dw   m5,q2,m6,q2,m5,q2,m4,q2,m3,q2,m1,q2,m5,q2,m6,q2,m5,q2
                 dw   m4,q2,m3,q2,m1,q2,m1,q2,m5,q2,m1,q2,m1,q2,m5,q2,m1

        .code
start:
        call init8255           ;8255 初始化
        call demo                   ;播放一段音乐
start1:
        mov dx, pa8255              ;按键查询
        in al, dx               ;读键值
        cmp al, 0ffh            ;judge if al equals 11111111b
        jz start1               ;无键
        xor al, 0ffh            ;有键
        test al, 1
        jz start2
        call music1             ;1 号键 ， 调 1 号键输出
        jmp start1

start2:
        test al, 2              ;2h = 00000010b
        jz start3
        call music2             ;2 号键
        jmp start1
start3:
        test al, 4              ;4h = 00000100b
        jz start4
        call music3             ;3 号键
        jmp start1
start4:
        test al, 8              ;8h = 00001000b
        jz start5
        call music4             ;4 号键
        jmp start1
```

```
start5:
        test al, 10h            ;10h = 00010000b
        jz start6
        call music5             ;5 号键
        jmp start1
start6:
        test al, 20h            ;20h = 00100000b
        jz start7
        call music6             ;6 号键
        jmp start1
start7:
        test al, 40h            ;40h = 01000000b
        jz start1
        call music7             ;7 号键
        jmp start1

demo    proc near
        mov cx, 80                  ;生日歌共 25 拍

        lea bx, music



demo10:
        push cx                     ;保护 cx 的值

        call [bx]               ;播放 bx 指向的音阶

        ;call [bx+50]
        ;call [si]


        ;call [bx]
        inc bx
        inc bx                      ;bx <- bx + 2，指向预定的下一个音阶
        ;inc si
        ;inc si


        pop cx                      ;恢复 cx 的值
        loop demo10                 ;继续播放
        ret
demo    endp
```

```
music1  proc near          ;节拍 1(手动按键时用 )
        call w_l           ;写 0, 蜂鸣器响
        call t10           ;延时 100us
        call t5            ;延时 50us
        call t5            ;延时 50us
        call w_h           ;写 1, 蜂鸣器不响
        call t10           ;延时
        call t5            ;
        call t5            ;
        ret
music1  endp

music2  proc near          ;节拍 2, 同上
        call w_l           ;写 0, 蜂鸣器响
        call t10
        call t5
        call t2
        call t1
        call w_h           ;写 1, 蜂鸣器不响
        call t10
        call t5
        call t2
        call t2
        ret
music2  endp

music3  proc near          ;节拍 3, 同上
        call w_l
        call t10
        call t5
        call t2
        call t1
        call w_h
        call t10
        call t5
        call t2
        ret
music3  endp

music4  proc near          ;节拍 4, 同上
```

```
        call w_l
        call t10
        call t5
        call t1
        call w_h
        call t10
        call t5
        call t1
        ret
music4  endp

music5  proc near               ;节拍 5，同上
        call w_l
        call t10
        call t5
        call w_h
        call t10
        call t5
        ret
music5  endp


music6  proc near               ;节拍 6，同上
        call w_l
        call t10
        call t2
        call t2
        call w_h
        call t10
        call t2
        call t2
        ret
music6  endp


music7  proc near               ;节拍 7，同上
        call w_l
        call t10
        call t2
        call t1
        call w_h
        call t10
        call t2
        call t1
```

```
        ret
music7  endp


;m1~m7每个节拍放音的时间基本相同，约为0.2s，通过修改函数体所call的函数体来改变每秒蜂鸣器震
;                               荡的次数，以此来体现放音时不同的音阶。

m1      proc near               ;节拍 1(自动放音时用 ，时间约 0.2s)
        mov cx, 1100                ;cx 控制的是下面两条loop指令执行的次
                                ;数，1100 共执行loop指令 550 次
m10:
        call w_l
        call t10
        call t10
        call t2
        loop m11
m11:
        call w_h
        call t10
        call t10
        call t1
        loop m10
        ret
m1      endp


m2      proc near               ;节拍 2，同上
        mov cx, 1150
m20:
        call w_l
        call t10
        call t5
        call t2
        call t2
        loop m21
m21:
        call w_h
        call t10
        call t5
        call t2
        call t2
        loop m20
        ret
```

```
m2      endp

;节拍 3, 同上
m3      proc near
        mov cx, 1200
m30:
        call w_l
        call t10
        call t5
        call t2
        call t1
        push ax
        pop ax
        nop
        nop
        loop m31
m31:
        call w_h
        call t10
        call t5
        call t2
        call t1
        loop m30
        ret
m3      endp

;节拍 4, 同上
m4      proc near
        mov cx, 1250
m40:
        call w_l
        call t10
        call t5
        call t2
        call t1
        loop m41
m41:
        call w_h
        call t10
        call t5
        call t2
        call t1
        loop m40
        ret
```

```
m4      endp

;节拍 5, 同上
m5      proc near
        mov cx, 1300
m50:
        call w_l
        call t10
        call t5
        call t2
        loop m51
m51:
        call w_h
        call t10
        call t5
        call t2
        loop m50
        ret
m5      endp

;节拍 6, 同上
m6      proc near
        mov cx, 1350
m60:
        call w_l
        call t10
        call t5
        call t1
        loop m61
m61:
        call w_h
        call t10
        call t5
        call t1
        loop m60
        ret
m6      endp

;节拍 7, 同上
m7      proc near
        mov cx, 1420
m70:
        call w_l
        call t10
```

```
            call t5
            loop m71
m71:
            call w_h
            call t10
            call t5
            loop m70
            ret
m7          endp

                              ;写  0(8255.pc.7=0)，蜂鸣器响
w_l         proc near
            mov dx, c8255
            mov al, 0eh
            out dx, al
            ret
w_l         endp

;写  1(8255.pc.7=1)
w_h         proc near
            mov dx, c8255
            mov al, 0fh
            out dx, al
            ret
w_h         endp

;8255  初始化
init8255 proc near
            mov dx, c8255
            mov al, 90h ;pc.7输出 ，pa输入
            out dx, al
            mov dx, c8255
            mov al, 0fh
            out dx, al
            ret
            init8255 endp

;延时  10us
t1          proc near
            ret
t1          endp

;延时  20us
t2          proc near
```

```
        call t1
        ret
t2      endp

;延时 50us
t5      proc near
        call t2
        call t2
        ret
t5      endp

;延时 100us
t10  proc near
        call t2
        call t2
        call t5
        ret
t10  endp

q1      proc near
        mov cx,300
loopq1: call t10
        loop loopq1
        ret
q1      endp

q2      proc near
        call q1
        call q1
        ret
q2      endp

q3      proc near
        call q2
        call q2
        ret
q3      endp
        end start
```

附录 B

LED 16×16 点阵（自下而上滚动）：

```
    .model    tiny
addr_8255_pa equ  0270h
addr_8255_pb equ  0271h
addr_8255_c  equ  273h
addr_273 equ  230h
line  equ  addr_273
row1      equ  addr_8255_pa
row2      equ  addr_8255_pb
          .stack     100
    .data
huan     db 00h,0c0h,00h,0c0h,0feh,0c0h,07h,0ffh,0c7h,86h,6fh,6ch,3ch,60h,18h,60h
db 1ch,60h,1ch,70h,36h,0f0h,36h,0d8h,61h,9ch,0c7h,0fh,3ch,06h,00h,00h
ying     db 60h,00h,31h,0c0h,3fh,7eh,36h,66h,06h,66h,06h,66h,0f6h,66h,36h,66h
db 37h,0e6h,37h,7eh,36h,6ch,30h,60h,30h,60h,78h,00h,0cfh,0ffh,00h,00h
shi      db 00h,00h,06h,30h,07h,30h,0fh,0ffh,0ch,30h,1fh,0ffh,3bh,33h,7bh,33h
db 1bh,0ffh,1bh,33h,19h,0b0h,18h,0e0h,18h,60h,18h,0fch,19h,8fh,1fh,03h
yong     db 00,0,1fh,0feh,18h,0c6h,18h,0c6h,18h,0c6h,1fh,0feh,018h,0c6h,18h,0c6h
db 18h,0c6h,1fh,0feh,18h,0c6h,18h,0c6h,30h,0c6h,30h,0c6h,60h,0deh,0c0h,0cch
xing     db 00h,00h,1fh,0fch,18h,0ch,1fh,0fch,18h,0ch,1fh,0fch,01h,80h,19h,80h
db 1fh,0feh,31h,80h,31h,80h,6fh,0fch,01h,80h,01h,80h,7fh,0ffh,00h,00h
yan      db 0,0,0ffh,0ffh,18h,0cch,18h,0cch,30h,0cch,30h,0cch,7fh,0ffh,7ch,0cch
         db 0fch,0cch,3ch,0cch,3ch,0cch,3dh,8ch,3dh,8ch,33h,0ch,06h,0ch,0ch,0ch
shi0     db 01h,80h,00h,0c0h,3fh,0ffh,3ch,06h,67h,0cch,06h,0c0h,0ch,0c0h,07h,0c0h
db 06h,0c0h,7fh,0ffh,00h,0c0h,01h,0e0h,03h,30h,06h,18h,1ch,1ch,70h,18h
yan0     db 00h,00h,0fch,60h,0ch,60h,6ch,0f0h,6ch,0d8h,6dh,8fh,6fh,0f8h,7eh,00h
db 06h,0c6h,07h,66h,3fh,0ech,0e7h,0ech,06h,18h,1fh,0ffh,0ch,00h,00h,00h
yi       db 0ch,0c0h,0ch,60h,18h,7ch,1bh,6ch,33h,0ch,73h,18h,0f1h,98h,31h,98h
         db 30h,0f0h,30h,0f0h,30h,60h,30h,0f0h,31h,98h,33h,0fh,3eh,06h,30h,00h
none     db 00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h
         db 00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h
    .code
    start:    mov      ax,@data
    mov ds,ax
    mov es,ax
    nop
    call  init_io
    call      test_led
    call      clear
chs_show:     mov      cx,9
    lea       si,huan
chs_1:    push     cx
```

```
        mov cx,16
chs_2:  call  disp_ch
        inc   si
        inc   si
        loop      chs_2
        pop   cx
        loop      chs_1
        jmp chs_show
disp_ch   proc near
        push          cx
        mov cx,8
disp_ch_1:  call  disp1
        loop      disp_ch_1
        pop       cx
        ret
disp_ch   endp
disp1     proc near
        push          si
        push          cx
        mov cx,16
        mov       bl,0feh
        mov bh,0ffh
repeat:         mov       dx,line
        mov       ax,bx
        out   dx,ax
        lodsb

        mov dx,row1
            out   dx,al
        lodsb

        mov dx,row2
        out       dx,al
        call  dl10ms
        call      clear
        stc
        rcl   bl,1
        rcl   bh,1
        loop repeat
        pop   cx
        pop   si
        ret
disp1     endp
init_io         proc near
```

```
        mov dx,addr_8255_c
        mov al,80h
        out  dx,al
        ret
init_io         endp
clearproc near
        mov al,0ffffh
        mov dx,line
        out  dx,ax
        mov al,0
        mov dx,row1
        out  dx,al
        mov dx,row2
        out  dx,al
        ret
clearendp
test_led    proc near
        mov dx,line
        xor  ax,ax
        out  dx,ax
        mov al,0ffh
        mov dx,row1
        out  dx,al
        mov dx,row2
        out  dx,al
        call        dl500ms
        call        dl500ms
        ret
test_led        endp

dl10ms    proc        near
        push            cx
        mov cx,133
        loop $
        pop         cx
        ret
dl10ms    endp
dl500ms         proc near
        push            cx
        mov         cx,0ffffh
        loop        $
        pop  cx
        ret
dl500ms  endp
```

```
        end  start
```

LED 16×16 点阵（自左向右滚动）：

```
    .model    tiny
addr_8255_pa equ  0270h
addr_8255_pb equ  0271h
addr_8255_c  equ  273h
addr_273 equ  230h
line  equ  addr_273
row1      equ  addr_8255_pa
row2      equ  addr_8255_pb
          .stack     100
    .data
wo db
00h,00h,04h,00h,0ch,07h,04h,01h,35h,86h,44h,58h,04h,60h,0ffh,90h,04h,08h,44h,84h,0c4h,40h,
7fh,0feh,24h,21h,24h,12h,24h,10h,04h,00h
ai db
00h,00h,0ch,00h,0ah,02h,89h,03h,0a9h,42h,99h,64h,89h,54h,49h,48h,79h,54h,4fh,62h,49h,0c1h,
59h,31h,69h,0ch,49h,02h,0dh,00h,02h,00h
wei db
00h,00h,08h,02h,1eh,03h,09h,0cch,0e8h,30h,1bh,0cch,04h,12h,3dh,09h,05h,7ch,0fdh,40h,05h,40
h,3dh,7ch,0c5h,02h,23h,0ffh,11h,00h,08h,80h
ji db
00h,00h,00h,0eh,00h,02h,20h,02h,7fh,0fch,20h,00h,20h,00h,20h,00h,3fh,0f8h,00h,04h,08h,82h,0
9h,01h,0ffh,0ffh,0bh,00h,08h,0c0h,08h,20h
xin db
00h,00h,00h,00h,00h,00h,03h,00h,07h,0c0h,0ch,0e0h,0ch,30h,06h,18h,03h,0ch,03h,0ch,06h,18h,
0ch,30h,0ch,0e0h,07h,0c0h,03h,00h,00h,00h

none     db 00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h
         db 00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h,00h
    .code
start:    mov ax,@data
    mov ds,ax
    mov es,ax
    nop
    call  init_io
    call  test_led
    call  clear
chs_show:
    mov cx,5
    lea  si,wo
chs_1:    push     cx
    mov cx,16
```

```
chs_2:    call  disp_ch
          inc   si
          inc   si
          loop  chs_2
          pop   cx
          loop  chs_1
          jmp   chs_show
disp_ch   proc  near
          push      cx
          mov   cx,20
disp_ch_1:
          call  disp1
          loop  disp_ch_1
          pop   cx
          ret
disp_ch   endp

disp1     proc  near
          push      si
          push      cx
          mov   cx,16

          mov   bl,01h
          mov   bh,00h

repeat:   mov   dx,row2
          mov       al,bl
          out   dx,al
          mov   dx,row1
          mov   al,bh
          out   dx,al


          lodsb
     call adjust
          mov   dl,al
          lodsb
          call  adjust
          mov   ah,dl
          xchg      al,ah
          xor   ax,0ffffh
          mov       dx,line
          out   dx,ax
          call  dl10ms
```

```
        call  clear
        clc
        rcl   bl,1
        rcl   bh,1
        loop repeat
        pop  cx
        pop  si
        ret
disp1       endp
init_io     proc near
        mov dx,addr_8255_c
        mov al,80h
        out  dx,al      ;
        ret
init_io         endp
clearproc near
        mov al,0ffffh
        mov dx,line
        out  dx,ax
        mov al,0
        mov dx,row1
        out  dx,al
        mov dx,row2
        out  dx,al

        ret
clearendp

test_led    proc near
        mov dx,line
        xor  ax,ax
        out  dx,ax
        mov al,0ffh
        mov dx,row1
        out  dx,al
        mov dx,row2
        out  dx,al

        call  dl500ms
        call  dl500ms
        ret
test_led    endp

adjust      proc        near
```

```
        push            cx
        mov cx,8
adjust1:  rcl   al,1
        xchg    al,ah
        rcr     al,1
        xchg    al,ah
        loop    adjust1
        mov     al,ah
        pop  cx
        ret
adjust      endp

dl10ms    proc near
        push    cx
        mov cx,133
        loop $
        pop  cx
        ret
dl10ms    endp
dl500ms   proc near
        push    cx
        mov cx,0ffffh
        loop $
        pop  cx
        ret
dl500ms   endp
            end   start
```