第 十 二 章

输入/输出端口应用

12

HT48CX0 系列微控制器的输入/输出端口的使用相当简单，可利用端口的控制寄存器来控制端口的输入输出，更可以做到位的输入输出控制，当端口的控制寄存器设定好了输入输出後，就可以视情况需要在程序中做输入输出控制，更可以在程序执行中更改端口的输入输出，不过往往不建议使用者如此使用，因为通常设定为输入或输出後，电路采取固定模式，除非有经过特殊设计为双向输入输出端口，否则硬件线路不适合作为既输入又输出的端口。因此把握这几个要点即可开始设计程序。

流水灯

本单元是模拟流水灯的功能，让 LED 一个接著一个来回显示。使用 PA0~PA7 推动 LED，使 LED 左右移动，每个 I/O 端口的输出有一个 LED 串联一个 240Ω电阻，排成一排，即可完成初步电路。此一范例是使用一个 HT48C10 的 20Pin 接脚型号，所占的器件空间相当於一个 TTL 晶片的电路，但其功能却等於几个 TTL 的组合。

电路设计

I/O 端口 PA0~PA7 设置为输出，每个 I/O 引脚驱动一个 LED 串联一个 240Ω电阻，而使用 RRC、RLC 的指令来使 LED 左右移动，如图所示(图 11.1)。

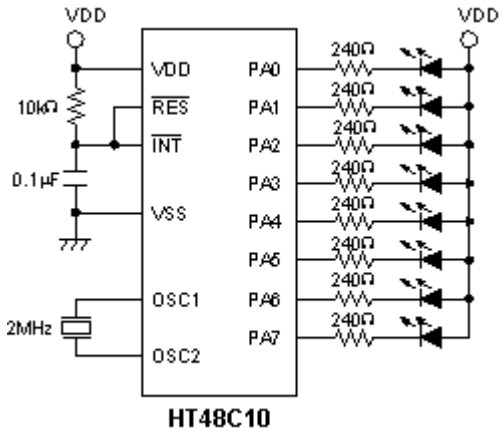

图 11.1

程序

```
#include ht48c10.inc
;------------------------------------------------------------
data .section 'data'           ;== 数据节区 ==
count1     db  ?               ;延时计数变量 1
count2     db  ?               ;延时计数变量 2
lamp       db  ?               ;灯光临时变量
;------------------------------------------------------------
code .section at 0 'code'      ;== 程序节区 ==
    org    00H                 ;
    jmp    start               ;
    org    04h                 ;外部中断入口地址
    reti                       ;
    org    08h                 ;定时 / 计数器 0 中断入口地址
    reti                       ;
    org    0ch                 ;定时 / 计数器 1 中断入口地址
    reti                       ;

start:                         ;
    clr    intc                ;程序一开始执行时先
```

```
    clr     tmrc                ;设定寄存器起始值
    clr     tmr                 ;以确保程序可以正确执行
    set     pac                 ;
    set     pbc                 ;I/O Port 设定为输入模式
    set     pcc                 ;
main:
    mov     a,0         ;(1)    ;
    mov     pac,a               ;设定 port A 为输出端口
    mov     pa,a                ;portA 写入 0(灯全亮)
    mov     a,0feh      ;(2)    ;1111 1110 (1:暗,0:亮)
    mov     lamp,a              ;设定灯状态初值
llamp:                          ;灯状态左移循环
    mov     a,lamp              ;载入灯状态
    mov     pa,a                ;输出灯状态到 port A
    call    delay       ;(3)    ;延时
    set     c                   ;;灯状态值左移与
    rlc     lamp        ;(4)    ;; 进位标志(fill LSB 1)
    sz      c                   ;是否所有灯处理完?
    jmp     llamp       ;(5)    ;;否. 继续灯状态左移循环
    rrc     lamp        ;(6)    ;;是. 恢复灯状态
rlamp:                          ;灯状态右移循环
    mov     a,lamp              ;载入灯状态
    mov     pa,a                ;输出灯状态到 port A
    call    delay               ;延时
    set     c                   ;;灯状态值右移与
    rrc     lamp        ;(7)    ;; 进位标志(fill MSB 1)
    sz      c                   ;是否所有灯处理完?
    jmp     rlamp               ;;否. 继续灯状态右移循环
    rlc     lamp                ;;是. 恢复灯状态
    jmp     llamp       ;(8)    ;重复灯状态左移循环
```

```
delay  proc                    ;延时子程序
    mov    a,2fh               ;;载入计数值
    mov    count1,a            ;;
    mov    count2,a            ;;
d1:
    sdz    count2              ;递减延时计数变量 2
    jmp    d1                  ;
    sdz    count1              ;递减延时计数变量 1
    jmp    d1                  ;
    ret
delay endp

    end
```

程序说明

程序起始首先(1)设定端口的输入输出状态，由于我们在本单元只有作输出端口，因此将端口状态控制器设定为 0，而且由于流水灯在同一时间内只有一个灯亮，所以我们先设定位 0 亮(2)，再经过一段延迟时间(3)，使得我们眼睛可以看到，完了後再经由 RLC 指令把亮灯作左移(4)，同样的再来经过一段延迟，(5)再重复 RLC 指令把亮灯作左移经过一段延迟，直到判断到亮灯的位经移位到进位标志，此时则表示 8 个位已经都显示过了，因此开始要改变亮灯移位的方向，再来就是(6)利用 RRC 指令把进位标志位移出到位 7，同样的经过一段延迟，再重复 RRC 指令把亮灯作右移(7)经过一段延迟，直到判断到亮灯的位已经移位到进位标志(8)，此时则表示 8 个位已经右移显示过了，如此再从前段左移部份开始重复整段程序，即完成流水灯。

红绿灯

本单元使用红绿黄色的 LED 模拟十字路口红绿灯的功能。首先 R1(红灯)及 G2(绿灯)亮，延迟一段时间後绿灯闪四下接著 Y1(黄灯)亮，延迟一段时间後 G1(绿灯) 及 R2(红灯)亮，如此循环不停。如同我们时常所看到的交通标志红绿灯。程序中可设置红灯和绿灯亮以及闪烁的时间长短不同。

电路设计

电路使用 PA0~PA2 及 PA4~PA6 分别驱动二组红绿黄的 LED，以模拟红绿灯的显示动作，其主要的动作都是由程序来控制，相当的简单明了，如图 11.2 所示，为红绿灯的电路图。

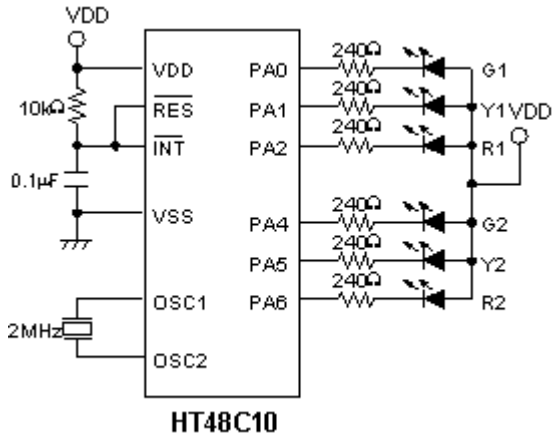

图 11.2

程序

```
#include ht48c10.inc
;------------------------------------------------------------
data .section 'data'           ;== 数据节区 ==
count1 db  ?                   ;延时计数变量 1
count2 db  ?                   ;延时计数变量 2
count3 db  ?                   ;延时计数变量 3
flash  db  ?                   ;闪烁临时变量
rglight    db ?                ;灯光临时变量
;------------------------------------------------------------
```

```
code .section at 0 'code'      ;== 程序节区 ==
    org    00H                 ;
    jmp    start               ;
    org    04h                 ;外部中断入口地址
    reti                       ;
    org    08h                 ;定时 / 计数器 0 中断入口地址
    reti                       ;
    org    0ch                 ;定时 / 计数器 1 中断入口地址
    reti                       ;

start:                         ;
    clr    intc                ;程序一开始执行时先
    clr    tmrc                ;设定寄存器起始值
    clr    tmr                 ;以确保程序可以正确执行
    set    pac                 ;
    set    pbc                 ;I/O Port 设定为输入模式
    set    pcc                 ;
main:
    mov    a,0          ;(1)   ;
    mov    pac,a               ;设定 port A 为输出端口
    mov    pa,a                ;port A 写入 0 (灯全亮)

loop:                          ;亮灯循环
    mov    a,0                 ;;载入查表指针
    mov    tblp,a              ;;
    tabrdl rglight      ;(2)   ;查表, 载入灯状态
    mov    a,rglight    ;(3)   ;;输出灯状态到 port A
    mov    pa,a                ;;
    call   delayl       ;(4)   ;延时 "长"时间
    inc    tblp         ;(5)   ;
```

```
        mov     a,7                     ;载入闪烁数值
        mov     flash,a                 ;暗亮暗亮暗亮暗(计 7 次)
    flash1:                             ;flash1 循环
        tabrdl  rglight                 ;载入灯状态
        mov     a,rglight               ;;
        mov     pa,a                    ;;输出灯状态到 port A
        call    delays          ;(6)    ;延时 "短"时间
        inc     tblp                    ;
        sdz     flash                   ;是否闪烁该结束?
        jmp     flash1                  ;否. 继续闪烁
        tabrdl rglight                  ;(是. 往下执行) 载入灯状态
        mov     a,rglight               ;;输出灯状态到 port A
        mov     pa,a                    ;;
        call    delaym          ;(7)    ;延时 "中"时间
        inc     tblp                    ;
;----------------------------;
        tabrdl rglight                  ;载入灯状态
        mov     a,rglight               ;;输出灯状态到 port A
        mov     pa,a                    ;;
        call    delayl                  ;延时 "长"时间
        inc     tblp                    ;

        mov     a,7                     ;;载入闪烁数值
        mov     flash,a                 ;;暗亮暗亮暗亮暗(计 7 次)
    flash2:                             ;flash2 循环
        tabrdl rglight                  ;载入灯状态
        mov     a,rglight               ;;输出灯状态到 port A
        mov     pa,a                    ;
        call    delays                  ;延时 "短"时间
        inc     tblp                    ;
```

```
    sdz     flash               ;是否闪烁该结束?
    jmp     flash2              ;否. 继续闪烁
    tabrdl rglight              ;(是. 往下执行) 载入灯状态
    mov     a,rglight           ;;输出灯状态到 port A
    mov     pa,a                ;;
    call    delaym              ;延时 "中"时间
    jmp     loop                ;重复 loop

delayl  proc                    ;"长"时间延时子程序
    mov     a,0fh               ;;载入计数值
    mov     count1,a            ;;
    mov     count2,a            ;;
    mov     count3,a            ;;
d1:
    sdz     count3              ;;递减延时计数变量 3
    jmp     d1
    sdz     count2              ;;递减延时计数变量 2
    jmp     d1
    sdz     count1              ;;递减延时计数变量 1
    jmp     d1
    ret
delayl  endp

delaym  proc                    ;"中"时间延时子程序
    mov     a,07h               ;;载入计数值
    mov     count1,a            ;;
    mov     a,0ffh              ;;
    mov     count2,a            ;;
    mov     count3,a            ;;
d2:
```

```
    sdz    count3              ;;递减延时计数变量 3
    jmp    d2
    sdz    count2              ;;递减延时计数变量 2
    jmp    d2
    sdz    count1              ;;递减延时计数变量 1
    jmp    d2
    ret
delaym  endp

delays  proc                   ;"短"时间延时子程序
    mov    a,0ffh              ;;载入计数值
    mov    count1,a            ;;
    mov    count2,a            ;;
d3:
    sdz    count2              ;;递减延时计数变量 2
    jmp    d3
    sdz    count1              ;;递减延时计数变量 1
    jmp    d3
    ret
delays  endp

    org    300h                ;数据表
                               ; 红黄绿 红黄绿
    dc     0EBh                ;1110 1011 G R
    dc     0FBh                ;1111 1011 O R
    dc     0EBh                ;1110 1011 G R
    dc     0FBh                ;1111 1011 O R
    dc     0EBh                ;1110 1011 G R
    dc     0FBh                ;1111 1011 O R
    dc     0EBh                ;1110 1011 G R
```

```
        dc      0FBh                    ;1111 1011 O R
        dc      0DBh                    ;1101 1011 Y R
        dc      0BEh                    ;1011 1110 R G
        dc      0BFh                    ;1011 1111 R O
        dc      0BEh                    ;1011 1110 R G
        dc      0BFh                    ;1011 1111 R O
        dc      0BEh                    ;1011 1110 R G
        dc      0BFh                    ;1011 1111 R O
        dc      0BEh                    ;1011 1110 R G
        dc      0BFh                    ;1011 1111 R O
        dc      0BDh                    ;1011 1101 R Y

        end
```

程序说明

程序起始首先(1)设定端口的输入输出状态，由于我们在本单元只作输出端口，因此将端口状态控制器设定为 0，由于红绿灯的亮灯状态是相当固定的，所以我们可以利用查表指令来完成(2)，将数据先存在列表区。由于我们是使用最後一页的查表指令，所以将列表区设定在300h 的地址(程序最大到 3FFh)。再来就是(5)一连串将数据表指标寄存器加一，(2)查表得到显示数据，再将数据输出至端口(3)。最大的不同就是延迟时间，由于绿灯亮、闪灯以及亮黄灯的时间各不相同，故需要不相同的延迟时间程序，而在程序中就有分(4)delayl、(7)delaym、(6)delays 三种不同时间延迟程序。

键盘扫描

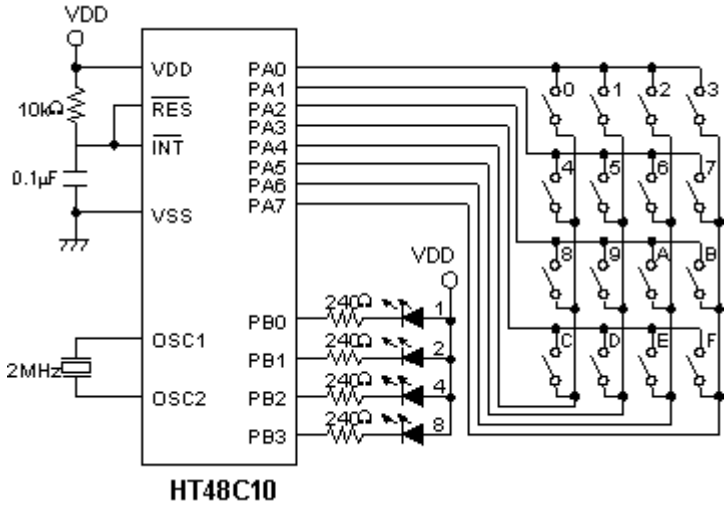

图 11.3

本单元使用一个 4×4 按键矩阵，输入按键後，由四个 LED 显示 16 进制码，判别并显示是哪一个按键被按下。如果有按键按下时，LED 即将此键所代表的 16 进制码以 4 Bits 显示 0000~1111 值。按键扫描程序在扫描按键时，如果两键同时按下扫描程序将以最先扫描到的键为主，而且当按键按下未放开时，并不允许下一按键输入。相同原理可扩充按键数目，并且可自行定义各键的意义。

电路设计

PA0~PA3 设置为输出，PA4~PA7 设置为输入(Mask Option 需设为 Pull-High)，来构成 4×4 按键矩阵。按键的排列方式可以由程序来设置并加以定义，并利用查表的方式来定义各个按键的键号。而 PB0~PB3 设置为输出，用以显示 4 Bits 所组成的 16 进制码，每一数字都代表一个按键，如图 11.3 所示。

程序

```
#include ht48c10.inc
;------------------------------------------------------------
data .section 'data'                ;== 数据节区 ==
temp        db  ?                   ;数据临时变量
disp        db  ?                   ;显示临时变量
count1      db  ?                   ;延时计数变量
mask        db  ?                   ;掩膜临时变量
```

```
    matrix     db  ?                        ;矩阵临时变量

;------------------------------------------------------------
    code .section at 0 'code'               ;== 程序节区 ==
        org    00h                          ;
        jmp    start                        ;
        org    04h                          ;外部中断入口地址
        reti                                ;
        org    08h                          ;定时/计数器 0 中断入口地址
        reti                                ;
        org    0ch                          ;定时/计数器 1 中断入口地址
        reti                                ;

    start:                                  ;
        clr    intc                         ;程序一开始执行时先
        clr    tmrc                         ;设定寄存器起始值
        clr    tmr                          ;以确保程序可以正确执行
        set    pac                          ;
        set    pbc                          ;I/O Port 设定为输入模式
        set    pcc                          ;
    main:
        set    pac                ;(1)      ;设定 port A 为输入端口
        clr    pbc                          ;设定 port B 为输出端口
        clr    pa                           ;清除 port A (latch=0)
        set    pb                           ;关闭 LEDs

    keyloop:                                ;
        mov    a,0feh     ;(2)              ;扫描第一列按键
        mov    matrix,a                     ;储存扫描值
        mov    pac,a                        ;设定 pa.0 为输出端口
```

```
mov     a,pa                    ;读取输入状态
cpl     acc                     ;
and     a,0f0h                  ;
sz      acc                     ;有无按键?
jmp     get_key                 ;有．去取按键的资料

mov     a,0fdh          ;(2)    ;无．扫描第二列按键
mov     matrix,a                ;储存扫描值
mov     pac,a                   ;设定 pa.1 为输出端口
mov     a,pa                    ;读取输入状态
cpl     acc                     ;
and     a,0f0h                  ;
sz      acc                     ;有无按键?
jmp     get_key                 ;有．去取按键的资料

mov     a,0fbh          ;(2)    ;无．扫描第三列按键
mov     matrix,a                ;储存扫描值
mov     pac,a                   ;设定 pa.2 为输出端口
mov     a,pa                    ;读取输入状态
cpl     acc                     ;
and     a,0f0h                  ;
sz      acc                     ;有无按键?
jmp     get_key                 ;有．去取按键的资料

mov     a,0f7h          ;(2)    ;无．扫描第四列按键
mov     matrix,a                ;储存扫描值
mov     pac,a                   ;设定 pa.3 为输出端口
mov     a,pa                    ;读取输入状态
cpl     acc                     ;
and     a,0f0h                  ;
```

```
    sz      acc                     ;有无按键?
    jmp     get_key                 ;有. 去取按键的资料
    jmp     keyloop                 ;重覆 keyloop

get_key:                            ;取得按键资料
    call    delays                  ;debounce
    mov     a,pa                    ;测试 port A
    or      a,0fh                   ;
    cpl     acc                     ;
    sz      acc                     ;有无按键被按下?
    Jmp     go_on                   ;有. 到 go_on (某键被按下)
    Jmp     keyloop                 ;无. 回到 keyloop 继续扫描

go_on:
    call    key_in          ;(3)    ;计算数据表的索引值
    tabrdl  disp            ;(10)   ;载入显示数据
    mov     a,disp                  ;;输出数据到 port B
    mov     pb,a            ;(11)   ;;
    jmp     keyloop                 ;重覆 keyloop

key_in proc                         ;取得按键数值
    mov     a,pa                    ;;储存 port A 状态
    mov     temp,a          ;(4)    ;;

get_release:                        ;等按键放开
    mov     a,pa                    ;;测试 port A 状态
    cpl     acc                     ;;
    and     a,0f0h                  ;;
    sz      acc             ;(6)    ;是否放开?
    jmp     get_release             ;否. 继续等待
```

```
    mov     a,0fh                   ;是. 计算按键数值
    andm    a,matrix        ;(7)    ;屏蔽扫描值的低 4 位
    Mmv     a,0                     ;将数据表的索引放入 a 寄存器
get_row:                            ;计算列的数值
    rrc     matrix                  ;透过 carry 检查每个 bit 以取得列的
                                    ;数值
    snz     status.0                ;
    jmp     get_next                ;若是有值跳到 get_next
    clr     c                       ;
    add     a,4h            ;(8)    ;数据表索引+4 (一列有 4 个按键)
    jmp     get_row                 ;继续计算
get_next:                           ;
    mov     tblp,a                  ;储存数据表索引於 TBLP 临时变量
    mov     a,0efh                  ;
    mov     mask,a                  ;mask = 0111 1111, 一次检查一个
                                    ;bit
    mov     a,0fh                   ;
    orm     a,temp                  ;temp = XXXX 1111, X 为输入值
get_column:                         ;计算行的数值
    mov     a,temp                  ;载入 temp
    xor     a,mask                  ;;测试行的数值
    snz     z                       ;;
    jmp     index                   ;否. 测试下一行
    ret                             ;是. 返回 (TBLP)
index:                              ;下一行
    inc     tblp            ;(9)    ;数据表索引+1
    set     c                       ;
    rlc     mask                    ;屏蔽值左移 (LSB=1)
    jmp     get_column              ;重复 get_column
```

```
key_in endp

delays proc                         ;延时子程序
    mov    a,0ffh                   ;载入计数值
    mov    count1,a                 ;
d1:
    sdz    count1                   ;递减 count1
    jmp    d1
    ret
delays  endp

    org 300h                        ;显示数据表
    dc     0fh,0eh,0dh,0ch          ;key0, key1, key2, key3
    dc     0bh,0ah,09h,08h          ;key4, key5, key6, key7
    dc     07h,06h,05h,04h          ;key8, key9, keyA, keyB
    dc     03h,02h,01h,00h          ;keyC, keyD, keyE, keyF

    end
```

程序说明

本单元由于要使用到输出及输入，所以端口状态控制器要把(1)输入的部份 (端口 A 的位 4~7) 设定为 1、输出的部份 (端口 A 的位 0~3 及端口 B) 设定为 0，再来就是整个判断有无按键输入的循环，而这个循环所要做的是一行一行判断有无按键按下。由于硬件按键是采取网状连接，因此可以从程序中可看出，换行判断有无按键差别只是不一样的值(2)输出而已，假如有输入按键的话，则会跳开判断有无按键循环，再去调用判断按键的子程序(3)。判断按键的子程序一进入，会先把输入端口取得的值存於数据临时寄存器(4)，然後再延迟一段时间(5)去除按键抖动 (bounce)，再来判断按键有无放开(6)，如果没有则等待按键放开後才继续执行下面程序。再来则是由换行判断有无按键循环中的数据，先行判断出那一行(7)，每跳一行则表列数据区的指标地址就必须位移 4 个地址(8)，取得行的表列数据区的指标地址後，就紧接著判断是此行的那一个按键，每判断一个键则列表数据区的指标地址又必须位移 1 个地址(9)，取得按键所代表的显示值的表列数据区指标地址後，就结束判断按键的子程序返回。再来则是利用取得的指标地址，以(10)查表指令从表列数据区取得按键所代表的显示值，输出至端口 B(11)所接的 LED 显示，即完成简单的键盘扫描。

液晶显示模块 (Liquid Crystal Module)

本单元介绍利用八位微控制器规划液晶显示模块 DV16100NRB，这块模块是单一行 16 个字元的显示器，其内部是由一颗 Hitachi 的 HD44780 所驱动及控制，因此只要依照 LCM 所规定的 timing，利用八位微控制器来产生 LCM 所要求的控制时序即可利用 LCM 显示字元，详细的时序及命令码仍需使用者自行参考厂家的原始数据为主。

由于 LCM 的操作方式可以为 4 位或八位模式，当操作於 4 位模式，传送一个字元或命令时需两次的传送才可形成一完整的命令或字元，而八位模式传送时只需一次传送即可，但多占用 4 位的 I/O 线。本节将利用编译器命令 #define，if ，else 及 endif 来编译程序。

电路设计

PB0~PB7 为数据汇流排设置成 I/O 端口，PC0~PC2 为 LCM 控制线设置成输出，读者可视需要自行修改。

程序

```
;===========
;= LCM.inc =
;===========
;请依所选用的微处理器设定适当的参数值
ifndef HEADER_HT48C30
    #define HEADER_HT48C30
    #include ht48c30.inc
endif

;------------------------------------------------------------
;for DV-16100NRB
LCM_CLS          EQU    01H        ; LCM 指令码
CURSOR_HOME      EQU    02H
CURSOR_SR        EQU    14H
```

```
CURSOR_SL         EQU    10H
INCDD_CG_SHF_C    EQU    06H
TURN_ON_DISP      EQU    0FH
LCD_ON_CSR_OFF    EQU    0CH

;------------------------------------------------------------
LCM_DATA          EQU    pb             ;端口 B 为 LCM 数据端口
LCM_DATA_CTRL     EQU    pbc            ;
LCM_CTRL          EQU    pc             ;端口 C 为 LCM 控制端口
LCM_CTRL_CTRL     EQU    pcc            ;

;------------------------------------------------------------
;LCM 显示命令以及控制信号
E                 EQU    0              ;控制信号脚位
RW                EQU    1              ;
RS                EQU    2              ;

;============
;= main.asm =
;============
#define HEADER_HT48C30
#include ht48c30.inc
#include lcm.inc

;------------------------------------------------------------
;#define four_bit                  ;定义 four_bit 时为 4-bit 读写模式
extern  busy_chk:near               ;说明外部定义函数
extern  delay:near                  ;
extern  write_char:near             ;
extern  snd_cmd:near                ;
```

```
;------------------------------------------------------------
data .section 'data'                ;== 数据节区 ==
counter0  db  ?
counter1  db  ?
msg       db  ?
tmp       db  ?

;------------------------------------------------------------
code .section at 0 'code'           ;== 程序节区 ==
   org    00H                       ;
   jmp    start                     ;
   org    04h                       ;外部中断地址
   reti                             ;
   org    08h                       ;定时 / 计数器 0 中断地址
   reti                             ;
   org    0ch                       ;定时 / 计数器 1 中断地址
   reti                             ;

start:                              ;
   clr    intc                      ;程序一开始执行时先
   clr    tmrc                      ;设定寄存器启始值
   clr    tmr                       ;以确保程序可以正确执行
   set    pac                       ;I/O Port 设定为输入模式
   set    pbc                       ;
   set    pcc                       ;
main:
   clr LCM_DATA_CTRL                ;设定 LCM 数据控制端口为输出端口
   clr LCM_CTRL_CTRL                ;设定 LCM 控制端口为输出端口
   clr LCM_DATA
```

```
    clr LCM_CTRL
DISPLAY_INIT:
ifdef  four_bit                    ;
    mov    a,20h                   ;4-bit 读取模式
else
    mov    a,30h                   ;8-bit 读取模式
endif
    mov    LCM_DATA,a
    set    LCM_CTRL.E              ;初始设定 LCM 功能
    clr    LCM_CTRL.E              ;
LCM_DELAY:                         ;延迟 4.5ms，等待 LCM 初始设定完成
    mov    a,0ffh                  ;
    mov    counter1,a
    mov    counter0,a
lp0:
    sdz    counter1
    jmp    lp0
    sdz    counter0
    jmp    lp0
CMD_SEQ:
ifdef four_bit                     ;
    mov    a,28h                   ;4-bit 读写模式，双列(1st pass)
else                               ;28h:双列，20h:单列
    mov    a,38h                   ;8-bit 读写模式，双列
endif                              ;38h:双列，30h:单列
    mov    LCM_DATA,a              ;写入指令
    set    LCM_CTRL.E
    clr    LCM_CTRL.E
ifdef four_bit
    mov    a,80h                   ;4-bit 读写模式的低位组(2nd pass)
```

```
    mov     LCM_DATA,a              ;写入指令
    set     LCM_CTRL.E              ;
    clr     LCM_CTRL.E              ;
endif
    call    busy_chk                ;检查 busy 信号
    mov     a,LCM_CLS               ;清除显示
    call    snd_cmd                 ;
    call    busy_chk                ;检查 busy 信号
    mov     a,TURN_ON_DISP          ;开启显示
    call    snd_cmd                 ;
    call    busy_chk                ;检查 busy 信号
    mov     a,INCDD_CG_SHF_C        ;;设定为自动递增模式
    call    snd_cmd                 ;;DD ram 地址自动加一，且游标右移
;-------------------------------------------------------------
    ;inc    addr of DD ram & shift
    ;the    cursor to the right at
    ;the    time of write to DD/CG
    ;RAM.
;-------------------------------------------------------------
    call    busy_chk                ;检查 busy 信号
    mov     a,LCM_CLS               ;清除 LCM 萤幕
    call    snd_cmd                 ;
    call    busy_chk                ;检查 busy 信号
    mov     a,CURSOR_HOME           ;LCM 游标归位
    call    snd_cmd                 ;
    call    busy_chk                ;检查 busy 信号
    clr     tblp                    ;清除查表指针

;显示"HOLTEK 8 bit mC"
agn:
```

```
    tabrdl msg                  ;载入讯息数据
    mov    a,msg                ;
    mov    tmp,a                ;
    mov    a,24h                ;结束值=24h
    xorm   a,msg                ;
    sz     msg                  ;是否显示完一行?
    jmp    agn1                 ;否，再显示下一字元
    jmp    secn_line            ;是，显示下一行
agn1:
    call   busy_chk             ;检查 busy 信号
    mov    a,tmp                ;
    call   write_char           ;写入字元到 LCM
    inc    tblp                 ;下一字元
    jmp    agn                  ;
secn_line:
    inc    tblp                 ;下一字元
    call   busy_chk             ;检查 busy 信号
    mov    a,0c0h               ;移动游标到第二行
    call   snd_cmd              ;(第一行由 00h 开始，第二行由 40h
    call   busy_chk             ; 开始)
snd_line:
    tabrdl msg                  ;载入讯息数据
    mov    a,msg                ;
    mov    tmp,a                ;
    mov    a,24h                ;
    xorm   a,msg                ;
    sz     msg                  ;是否显示完一行?
    jmp    snd_lin1             ;否，再显示下一字元
    mov    a,LCD_ON_CSR_OFF     ;是，隐藏游标
    call   snd_cmd
```

```
        jmp lp                          ;
    snd_lin1:
        mov     a,tmp                   ;
        call    write_char              ;写入字元到 LCM
        call    busy_chk                ;检查 busy 信号
        inc     tblp                    ;
        jmp     snd_line                ;
    lp:
        jmp     lp                      ;程序结束

;------------------------------------------------------------
    holtek_tbl .section at 700h 'code';table at last page
    htk_tbl:                            ; 字串 "HOLTEK 8 bit uC"
        dc 0048h,004fh,004ch,0054h,0045h,004bh,0020h,0038h,0024h
        dc 0020h,0062h,0069h,0074h,0020h,0075h,0043h,0024h

    end                                 ;module

    ;===========
    ;= LCM.asm =
    ;===========
    include lcm.inc

    ;------------------------------------------------------------
    ;#define four_bit                   ;定义四位读取模式

    public  busy_chk                    ;公共函数
    public  delay                       ;
    public  write_char                  ;
    public  snd_cmd                     ;
```

```
;------------------------------------------------------------
dataLCM .section 'data'
dtmp      db  ?
dtmp2  db  ?

;------------------------------------------------------------
codeLCM .section 'code'
;传送指令到 LCM
snd_cmd:                          ;写入指令到 LCM
ifdef  four_bit                   ;
   mov    dtmp,a                  ;对於四位读取模式
   and    a,0f0h                  ;先写入 high nibble(1st pass)
endif
   mov    LCM_DATA,a              ;
   clr    LCM_CTRL.RW             ;RW=0
   clr    LCM_CTRL.RS             ;RS=0
   set    LCM_CTRL.E              ;high
   clr    LCM_CTRL.E              ;low  (产生写入信号)
ifdef  four_bit
   swapa  dtmp                    ;写入 low nibble (2nd pass)
   and    a,0f0h
   mov    LCM_DATA,a              ;latch command
   set    LCM_CTRL.E              ;high
   clr    LCM_CTRL.E              ;low  (产生写入信号)
endif
   ret
;检测 busy 信号
busy_chk:                         ;检测 busy 信号
   clr    LCM_CTRL.E              ;
```

```
    set     LCM_DATA_CTRL           ;设定 LCM 数据端口为输入端口
    clr     LCM_CTRL.RS             ;RS=0
    set     LCM_CTRL.RW             ;RW=1
    set     LCM_CTRL.E              ;
    mov     a,LCM_DATA              ;读入 busy 信号
    clr     LCM_CTRL.E              ;
ifdef  four_bit
    and     a,0f0h                  ;读取 high byte(1st pass)
    mov     dtmp,a                  ;
    set     LCM_CTRL.E              ;
    swapa   LCM_DATA                ;
    clr     LCM_CTRL.E              ;
    and     a,0fh                   ;读取 low byte(2nd pass)
    or      a,dtmp                  ;
endif
    sz      acc.7                   ;是否 busy?
    jmp     busy_chk                ;是，则再检查
    clr     LCM_CTRL.RW             ;
    clr     LCM_DATA_CTRL           ;设定 LCM 数据端口为输出端口
    ret

;写入字元
write_char:                         ;写入字元至 LCM
ifdef  four_bit                     ;4-bit 读写模式(1st pass)
    mov     dtmp,a
    and     a,0f0h                  ;将 high nibble 送至 LCM
endif                               ;
    mov     LCM_DATA,a              ;
    clr     LCM_CTRL.RW             ;RW=0
    set     LCM_CTRL.RS             ;RS=1 (写入指令)
```

```
    set     LCM_CTRL.E              ;high
    clr     LCM_CTRL.E              ;low  (产生写入信号)
ifdef  four_bit
    swapa   dtmp                    ;4-bit 读写模式(2nd pass)
    and     a,0f0h                  ;将 low byte 送至 LCM
    mov     LCM_DATA,a              ;
    set     LCM_CTRL.E              ;high
    clr     LCM_CTRL.E              ;low  (产生写入信号)
endif
    ret

;延迟子程序
delay:                              ;延迟子程序
    mov     dtmp,a
    drep:
    sdz     dtmp2                   ;
    jmp     drep
    sdz     dtmp                    ;
    jmp     drep
    ret

end                                 ;程序模组结束
```

程序说明

程序首先将一些含括文件(*.inc 文件)包含进来，并定义 LCM 数据端口为 PB 而定义 LCM 控制线为 PC 同时并宣告子程序为外部模组，编译器命令(define)用来定义参数以便後来程序可以较方便的编译成 4bit 模式或 8bit 模式。

LCM 在电源接通时将会自动的执行内部 RESET 及初始化的动作，但一般多会利用软件去控制 LCM 的初始化，程序从 start 起开始一连串的对 LCM 进行初始化，根据 HD44780 的数据所示，当作完第一笔规划後至少需等待 4.5 毫秒的後才能对 LCM 作其他的规划 LCM_DELAY 的作用就是如此，LCM 没有完成初始化前是无法检查 LCM 是否在 BUSY 状态.当我们对 LCM 做规

划或下命令时依照 HD44780 所定义的命令码，LCM.INC 中即定义了一些常用的命令。

LCM 在写命令的前一定要检查 LCM 是否在 BUSY 状态，而 BUSY_CHK 就是检查 LCM 是否在 BUSY。当程序由 BUSY_CHK 返回时即代表 LCM 已不 BUSY 可以送数据给 LCM 了。将要显示的 ASCII 码放在程序码的最後一页,利用查表法将要显示的码存入 ACC 後,调用 WRITE_CHAR 即可完成显示。

以下列出 RS，R/W 及 E 三支控制信号的关系。

RS	RW	E	Operation
0	0		Write instruction code
0	1		Read busy flag & address counter
1	0		Write data
1	1		Read data

利用 I/O 端口做串行端口的应用

本单元提供一个模拟串行端口工作的程序，使用者可利用此程序来做简单的通讯，通讯格式为 8 个数据位,无同位检查(Non Parity)，1 个停止位.当使用者要使用时，只要将此文件加入 Project，再来就是定义您自己用来做为传送的 Pin 脚和接收的 Pin 脚以及 Baudrate 与系统频率所对照出的参数即可。

以下是使用者使用步骤及注意事项：

1.必须先定义一个文件 DFSERIPT.INC 於使用者工作目录,或是开发系统的 INCLUDE 子目录中(例：\HT-IDE\INCLUDE\)。

其内容有：

1.Baudreatconst XXX，XXX 值可由表格对照取得或者由公式计算出

(去掉小数点以下取整数值)。

2.TXPIN ---------- 定义 Transmit 从那个 I/O Pin。

3.RXPIN ---------- 定义 Receive 从那个 I/O Pin。

2、串行端口子程序会占用 4 个 RAM 位置、2 个 I/O Pin 及 49 个程序记忆体地址，程序中必须自行定义 TXPIN 为输出，RXPIN 为输入。

3、Call Receive 的前必须先确定 RXPIN 处於高准位的停止位状态。调用这二个子程序都会改变 Carry flag。

4、有使用到的 Routine，程序前面必须加上说明，如下：

```
EXTERN    TRANSMIT: NEAR
EXTERN    RECEIVE: NEAR
```

5、必须在 Project 内，主程序的下加入 LIBSER.ASM。

Baudrate 参数 baudrateconst 的计算公式如下：

Baudrateconst = (Fsys÷baudrate÷12)－3 (去掉小数点后的数取整数)

备注：baudrateconst 最好在大於 7 、小於 256 的范围内。

baudrateconst 的数值愈大误差愈小。

如下表可直接对照：

baudrate\Fsys	4MHz	2MHz	1MHz
9600	31	14	X
7200	43	20	8
4800	66	31	14
3600	89	43	20
2400	135	66	31
2000	163	80	38
1800	182	89	43
1200	X	135	66

X：表示无法使用，请降低系统频率 (Fsys) 或 baudrate 值。

使用者需定义的 DFSERIPT.INC 档案内容：

<例>

baudrate 3600，系统频率 4MHz 时的定义，PA.3 为 Transmit 的 Pin 脚，PA.2 为 Receive 的 Pin 脚。

```
BAUDRATECONST EQU 89          ; 由查表或公式计算得。
TXPIN   EQU   PA.3            ; 定义作为 Transmit 的 Pin 脚。
RXPIN   EQU   PA.2            ; 定义作为 Receive 的 Pin 脚。
```

程序

```
;================
;= Dfseript.inc =
;================
BAUDRATECONST EQU 66              ;波特率参数(4MHz, 4800
                                  ;bits/second)(请参考上表的说明)
TXPIN EQU PA.3                    ;传送脚
RXPIN EQU PA.2                    ;接收脚

;============
;= Main.asm =
;============
#define HEADER_HT48C70
#include ht48c70.inc
#include dfseript.inc

;#define TRANSMIT_MODE            ;定义 TRANSMIT_MODE,使工作於传送
;模式(内定为接收模式)
extern  transmit:near             ;外部定义函数
extern  receive:near

;------------------------------------------------------------
data .section 'data'              ;== 数据节区 ==
transmit_data db     ?            ;
receive_data  db     ?
counter       db     ?

;------------------------------------------------------------
code .section at 0 'code'         ;== 程序节区 ==
   org    00H                     ;
```

```
    jmp     start                   ;
    org     04h                     ;外部中断入口地址
    reti                            ;
    org     08h                     ;定时 / 计数器 0 中断入口地址
    reti                            ;
    org     0ch                     ;定时 / 计数器 1 中断入口地址
    reti                            ;

start:                              ;
    clr     intc                    ;程序一开始执行时先
    clr     tmr0c                   ;设定寄存器起始值
    clr     tmr0h                   ;以确保程序可以正确执行
    clr     tmr0l                   ;
    clr     tmr1c                   ;
    clr     tmr1h                   ;
    clr     tmr1l                   ;
    set     pac                     ;I/O Port 设定为输入模式
    set     pbc                     ;
    set     pcc                     ;
    set     pdc                     ;
    set     pec                     ;
    set     pfc                     ;
    set     pgc                     ;
main:
    set     pac.2                   ;设定接收脚位为输入端口
    clr     pac.3                   ;设定传送脚位为输出端口

loop:
ifdef       TRANSMIT_MODE           ;传送模式
    mov     a,32                    ;传送 32 个位组
```

```
    mov     counter,a
    clr     tblp                    ;清除查表指标以传送存於表中的数据
again:
    tabrdl transmit_data            ;载入欲传送的数据
    mov     a,transmit_data
    call    transmit                ;调用传送函数
    inc     tblp                    ;指向下一个欲传送的数据

    sdz     counter                 ;计数器递减并检查是否传送完毕?
    jmp     again                   ;否，传送下一个位组
    jmp     loop                    ;是，重覆再传送
else
    mov     a,40h                   ;接收模式
    mov     mp0,a                   ;mp0 指向接收数据区(40h~5Fh)
    mov     a,32                    ;将接收 32 笔
    mov     counter,a               ;
again:
    call    receive                 ;调用接收函式以接收一位组
    mov     r0,a                    ;将接收到的数据存於 mp0 所指到的资
                                    ;料区
    inc     mp0                     ;数据区指标指向下一个可存地址
    sdz     counter                 ;计数器递减并检查是否接收完毕?
    jmp     again                   ;否，再接收下一笔数据
    jmp     $                       ;是，程序结束
endif

test .section at 1f00h 'code'
 test_table:                        ;待传送的 32 个位组的数据
    dc 012h,034h,056h,078h,09ah,0bch,0deh,0f0h,011h,
       022h,033h,044h,055h,066h,077h,088h
```

```
    dc 099h,0aah,0bbh,0cch,0ddh,0eeh,0ffh,000h,055h,
       0aah, 055h,0aah,055h,0aah,055h,0aah
    end

;==============
;= Serial.asm =
;==============
;serial port library
ifndef HEADER_HT48C70              ;根据所使用的微控制器选用标头档
#define HEADER_HT48C70
#include ht48c70.inc
endif
#include dfseript.inc

public  transmit                   ;宣告公用程序
public  receive                    ;

baudrate  equ baudrateconst;
tx  equ txpin                      ;
rx  equ rxpin                      ;

sdata .section 'data'              ;
count     db  ?                    ;传送位数寄存器
txreg     db  ?                    ;传送数据寄存器
rcreg     db  ?                    ;接收数据寄存器
delay     db  ?                    ;延迟寄存器

serial .section 'code'             ;
transmit proc                      ;透过 Acc 传送一位组
    mov    txreg,a                 ;
```

```
    mov    a,baudrate          ;
    mov    delay,a             ;
    clr    tx                  ;传送启动位'0'
    mov    a,9                 ;共用传送 8 个位
    mov    count,a             ;
txdelay:                       ;
    sdz    delay               ;延迟以符合波特率
    jmp    txdelay             ;
    mov    a,baudrate          ;
    mov    delay,a             ;
    sdz    count               ;是否传送完毕?
    jmp    sendbit             ;否，再传送下一位
    jmp    endtx               ;是，结束
sendbit:
    rrc    txreg               ;将欲传送数据右移入 carry flag
    snz    c                   ;是否要传送'1'
    jmp    lobit               ;
    set    tx                  ;是，则传送'1'
    jmp    txdelay             ;
lobit:                         ;
    clr    tx                  ;否，传送'0'
    jmp    txdelay             ;
endtx:                         ;
    nop                        ;
    nop                        ;
    set    tx                  ;传送停止位'1'
t1:                            ;
    sdz    delay               ;为调整时间而稍微延迟
    jmp    t1                  ;
    mov    a,baudrate          ;
```

```
    mov     delay,a                 ;
t2:                                 ;
    sdz     delay                   ;
    jmp     t2                      ;
    ret                             ;
transmit    endp                    ;

receive proc                        ;接收一个位组
    sz      rx                      ;检查起始位是否为 0
    jmp     receive                 ;否，则再检测
    mov     a,9                     ;是，准备接收八个位的数据
    mov     count,a                 ;
    mov     a,baudrate+1            ;延迟以符合波特率
    mov     delay,a                 ;(注意：+1 用以调整时间以符合
rxdelay:                            ;       波特率)
    sdz     delay                   ;
    jmp     rxdelay                 ;
    mov     a,baudrate+1            ;载入延迟计数器
    mov     delay,a                 ;(注意：+1 用以调整时间)
    sdz     count                   ;是否接收完 8 位?
    jmp     rxbit                   ;否，再接收下一位
    mov     a,rcreg                 ;是，将收到的 8 位数据存入 Acc
    ret                             ;
rxbit:                              ;
    set     c                       ;设定 c=1
    snz     rx                      ;是否接收到'1'?
    clr     c                       ;否，设定 c=0
    rrc     rcreg                   ;将收到的数据存入 rcreg
    jmp     rxdelay                 ;
receive endp                        ;
```

```
end
```

程序说明

整个程序最主要的就是 baudrate 参数，因为要配合 baudrate 的速度，所以传送数据或者接收数据都要速度配合，因此由我们替您计算出配合 baudrate 要何种参数的公式，以及常用的 baudrate 与系统频率的对照表格。由于此一范例并非是利用计数/定时来作为 baudrate 时间的计数，因此有某种程度的误差，但根据我们以 8051 来测试的结果，表格上的数值都正确无误，事实上，baudrate 的参数其数值愈大误差愈小，因此您可以调整 baudrate 与系统频率把 baudrate 的参数值提高，提供您做为参考。

第 十 三 章

中断及定时/计数器应用

13

运用定时/计数器来作为中断，可以应用在很多方面，例如在程序中需要一个时间性的讯号或者是需要在某一段固定或不固定时间後做某件事，就可以利用定时/计数器来作时间的计数，在此同时主程序可以先做其它事，等时间计数到後就会执行中断，让使用者去做特定的事，等中断子程序完了之後又会跳回原程序段继续执行。

本公司 HT48CX0 系列之计数/定时器，可分为 16 位元及 8 位元，皆属於上述定时器，因此您预计要的计数，必须先把它转为二的补数，再载入 16 位元或 8 位元定时器即可。另外定时/计数器可分三种定时/计数模式，有事件计数模式、时间计时模式及脉冲宽度测量模式，其中事件计数模式即是由外部触发信号输入，而时间计时模式是以内部振荡频率为基准。

电子琴

本单元说明如何使用扫描程序扫描按键输入，同时将按键转换成所定义的键，再转换为相对应的声音频率，把频率值放入定时/计数器的计数暂存器，等定时/计数器计数到後会跳到中断常式中执行，当中断子程序执行完後定时/计数器的计数暂存器值会重新载入，因此我们利用这一特性来发音，每种频率的声音其周期长短皆不同，我们在每一次中断後，中断子程序内的程序会改变端口的状态，使其形成某一个频率的波输出，即可形成声音讯号，只要我们再接个放大电路，输出接喇叭就可以完成此一例子。发音的程序主要都是使用定时/计数器来做计数，控制输出频率，而频率值必须要经过换算，并且定时/计数器是采取往上计数的，因此预载入计数暂存器值必须作补数。

电路设计

PA0~PA7 设置为输入，同时用电阻接电源使其电位为 High，当按键按下时其电位为 Low ，此一设计只是为了程序编写容易。PB0 设置为输出来驱动喇叭，在作 High/Low 转换时形成声音脉波，使喇叭发出声音，如图 12.1 所示。

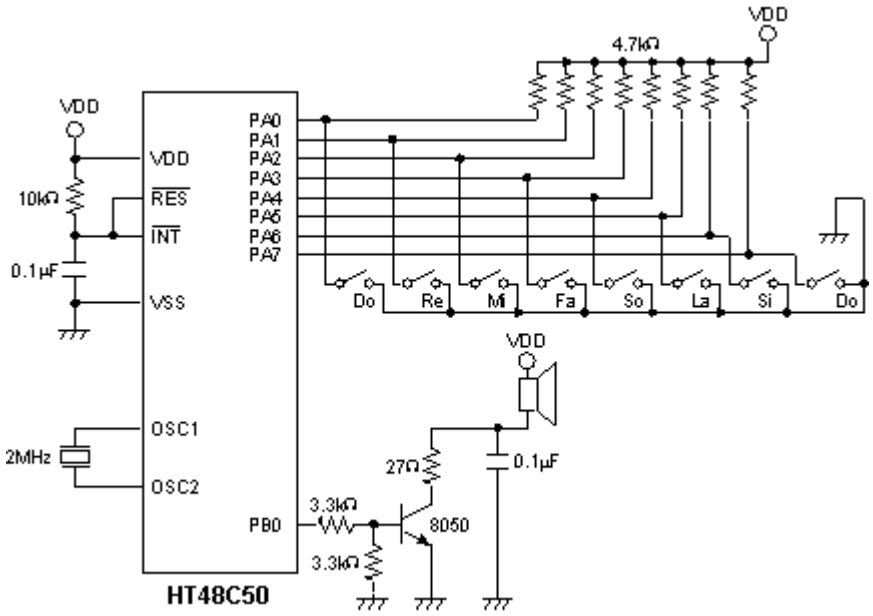

图 12.1

程序

```
#include ht48c50.inc

;------------------------------------------------------------

data .section 'data'                ;== 数据节区 ==

temp       db  ?                    ;数据临时变量

sound  db  ?                        ;声音频率临时变量

;------------------------------------------------------------

code .section at 0 'code'           ;== 程序节区 ==

   org    00h                       ;

   jmp    start                     ;
```

```
    org     04h                     ;外部中断入口地址
    reti                            ;
    org     08h                     ;定时 / 计数器 0 中断入口地址
    cpl     pb              ;(6)    ;产生方波
    reti                            ;
    org     0ch                     ;定时 / 计数器 1 中断入口地址
    reti                            ;

start:                              ;程序一开始执行时先
    clr     intc                    ;设定暂存器启始值
    clr     tmr0c                   ;以确保程序可以正确执行
    clr     tmr0h                   ;
    clr     tmr0l                   ;
    clr     tmr1c                   ;
    clr     tmr1                    ;
    set     pac                     ;I/O Port 设定为输入模式
    set     pbc                     ;
    set     pcc                     ;
    set     pdc                     ;
main:
    set     pac             ;(1)    ;设定 port A 为输入口
    clr     pbc                     ;设定 port B 为输出口
    clr     pb                      ;
keyloop:                    ;(2)    ;
    mov     a,pa                    ;测试是否按下输入键
    cpl     acc                     ;
    sz      acc                     ;
```

```
    call      whichkey              ;若是，则找出按下那一个输入键
    jmp    keyloop                  ;若否，则重测是否按下输入键

whichkey  proc             ;(3)    ;找出按下的输入键
    mov    temp,a                   ;保留 Acc 值
    mov    a,0                      ;清除查表指针
    mov    tblp,a                   ;
    clr    c                        ;c=0 (由 carry flag 查出是那一个键被
                                    ;   按下)
keynext:                            ;
    rrc    temp                     ;
    sz     status.0                 ;是否 carry? (若是，则表示有键被按下)
    jmp    timerset                 ;是，则输出声音
    inc    tblp            ;(4)    ;否，将查表指标指向下一位
    inc    tblp                     ;(2 bytes/key)
    jmp    keynext                  ;检查下一个按键
timerset:                           ;设定定时器以产生声音
    mov    a,5                      ;启动定时器 0 中断
    mov    intc,a                   ;
    mov    a,80h                    ;设定定时器 0 为内部计数模式
    mov    tmr0c,a                  ;
    tabrdl sound            ;(5)    ;载入声音频率(2 bytes)
    mov    a,sound                  ;读取 low byte
    mov    tmr0l,a                  ;
    inc    tblp                     ;
    tabrdl sound            ;(6)    ;读取 high byte
    mov    a,sound                  ;
```

```
    mov     tmr0h,a                 ;
    set     tmr0c.4                 ;启动定时器 0
key_halt:                   ;(7)    ;查看输入键是否持续被按下
    mov     a,pa                    ;读取 Port A
    cpl     acc                     ;
    sz      acc                     ;输入键是否按着的?
    jmp     key_halt                ;是，再检查
    clr     tmr0c.4         ;(8)    ;否,定时器停止计数,同时也停止输出声音
    clr     pb                      ;
    ret                             ;
whichkey endp                       ;

    org     0f00h                   ;声音频率表
    dc      21h,0feh,58h,0feh
    dc      84h,0feh,99h,0feh
    dc      0c1h,0feh,0e3h,0feh
    dc      02h,0ffh,11h,0ffh

       end
```

程序说明

程序起始先设定(1)端口 A 为输入端口，端口 B 为输出端口，把端口 A 控制暂存器设定为 1，把端口 B 控制暂存器设定为 0，後面紧接就是一个取得按键之循环(2)，如果有按键输入才跳出此循环，否则一直执行循环等待按键输入。待有按键输入後就跳至判断那一个按键的副程序(3)去，而判断按键只是为了指到在列表区所储存的声音频率值之位置(4)，在判断按键的同时把数据表指针暂存器的指针作调整，使其能在判断出不同的按键时指到不同的位置取得不同声音频率值，再来则是分别把声音频率值之高位元和低位元(5)，放在定时/计数器的高位元和低位元，利用计数完毕产生中断，执行中断常式，每执行一次就把原来端口 B 的

值取反(6)，如此即可由端口 B 取得一个音阶频率的脉波，只要接上一个简单的放大器即可输出声音。

而在判断按键及取得声音频率值之後，就紧接着判断按键是否放开(7)，因为定时/计数器的计数值一经载入後，除非把它关掉不然它计数完毕後，其计数值会重新载入，因此端口 B 就会一直有脉波输出，所以判断按键是否放开就是决定当按键放开後，关掉定时/计数器(8)不让它计数而无输出声音，等下一按键输入时再打开，如此电子琴即完成。

时钟

本单元程序是应用定时/计数器的 16 Bits 计数器产生中断方式来作计时，由於必须配合硬件电路使用的振荡频率来计算计数值，因此本单元例子使用振荡频率 400KHz 作系统输入，再来就是定时/计数器 0 为一 16 Bits 计数器，故其最大值可达到 65536，依振荡频率为 400KHz 来计算，最大可计时到每 0.65536 S 中断一次，还不到一秒钟，所以要达到一秒钟必须要中断多次才可以，因此我们设定每 0.5 秒钟定时器中断一次，中断两次即是一秒钟，如此时钟的定时器即可开始动作。本单元例子乃利用四个七段显示器来显示 24 小时进制的时和分，还有两个按键可调整时与分。

电路设计

PA0~PA7 设置输出端口，其 PA0~PA3 作为 4Bits 时间的数字码输出，PA4~PA7 输出至电晶体，控制电晶体成为开关，控制那一个七段显示器要显示，一直交替扫描，PB0、PB1 作为调整时跟分输入键，如图 12.2 所示。

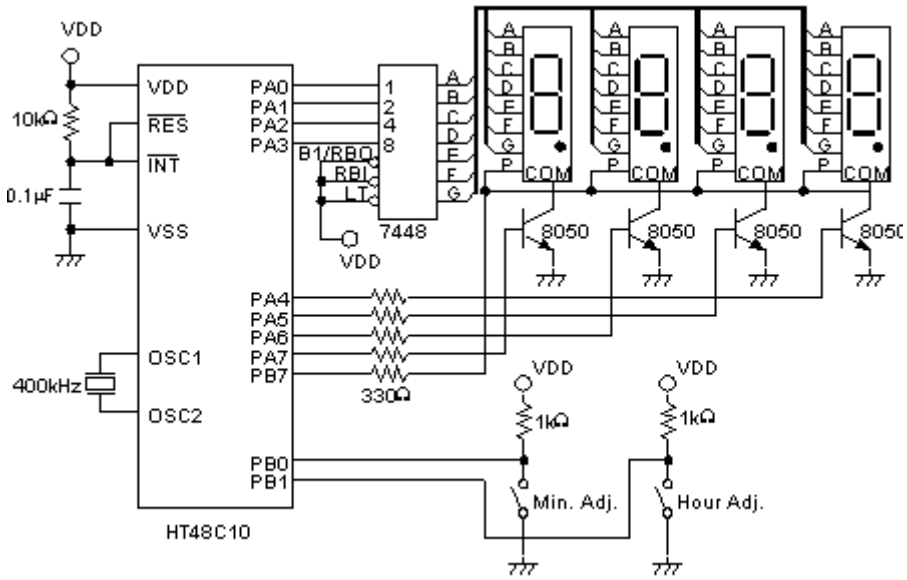

图 12.2

程序

```
#include ht48c50.inc

;------------------------------------------------------------
data .section 'data'            ;== 数据节区 ==
second db  ?                    ;秒
minl       db  ?                ;分钟的 low byte
minh   db  ?                    ;分钟的 high byte
hourl  db  ?                    ;小时的 low byte
hourh  db  ?                    ;小时的 high byte
count1 db  ?                    ;延迟计数变量
mask   db  ?                    ;掩膜变量
disp       db  ?                ;显示记录变量
```

```
;------------------------------------------------------------
code .section at 0 'code'      ;== 程序节区 ==
    org    00h                 ;
    jmp    start               ;
    org    04h                 ;外部中断入口地址
    reti                       ;
    org    08h                 ;定时 / 计数器 0 中断入口地址
    inc    second              ;秒数加 1(一个单位是 0.5 秒)
    cpl    pb                  ;每 0.5 秒闪烁 '点' 一次
    reti                       ;
    org    0ch                 ;定时 / 计数器 1 中断入口地址
    reti                       ;

start:                         ;
    clr    intc                ;程序一开始执行时先
    clr    tmr0c               ;设定暂存器启始值
    clr    tmr0h               ;以确保程序可以正确执行
    clr    tmr0l               ;
    clr    tmr1c               ;
    clr    tmr1                ;
    set    pac                 ;I/O Port 设定为输入模式
    set    pbc                 ;
    set    pcc                 ;
    set    pdc                 ;
main:
    clr    pac           ;(1)  ;设定 port A 为输出端口
    mov    a,7fh               ;设定 port B 为输入端口
```

```
    mov     pbc,a                   ;除了 pb.7
    clr     pb              ;(2)    ;设定变量启始值
    clr     pa                      ;
    clr     minl                    ;
    clr     minh                    ;
    clr     hourl                   ;
    clr     hourh                   ;
    clr     second                  ;
    mov     a,05h                   ;启动定时器 0 中断
    mov     intc,a                  ;
    mov     a,80h                   ;设定定时器 0 为内部计数模式
    mov     tmr0c,a                 ;
    mov     a,0b0h          ;(5)    ;设定定时器 0，使之每 0.5 秒产生中断
    mov     tmr0l,a                 ;设定 low byte
    mov     a,3ch                   ;
    mov     tmr0h,a                 ;设定 high byte
    set     tmr0c.4                 ;启动定时器 0
loop:                       ;(3)    ;
    mov     a,0                     ;清除查表指针
    mov     tblp,a                  ;
    mov     a,minl                  ;载入显示数据
    mov     disp,a                  ;将分钟的 low byte
    call    show_clock              ; 显示於第四个七段显示器
    inc     tblp                    ;
    mov     a,minh                  ;载入显示数据
    mov     disp,a                  ;将分钟的 high byte
    call    show_clock              ; 显示於第三个七段显示器
```

```
    inc     tblp                ;
    mov     a,hourl             ;载入显示数据
    mov     disp,a              ;将小时 low byte
    call    show_clock          ; 显示於第二个七段显示器
    inc     tblp                ;
    mov     a,hourh             ;载入显示数据
    mov     disp,a              ;将小时 high byte
    call    show_clock          ; 显示於第一个七段显示器
    jmp     loop                ;重覆显示动作


;------------------------------------------------------------
cal_number proc                 ;
    inc     minl                ;分钟 low byte 加一
    mov     a,minl              ;
    sub     a,0ah               ;
    sz      acc                 ;是否大於 10 分钟?
    Ret                         ;  若否，则结束此副程序
    clr     minl                ;  若是，则将分钟(low byte)清为 0
    inc     minh                ;        并将分钟(high byte)加一
    mov     a,minh              ;
    sub     a,06h               ;
    sz      acc                 ;是否大於 60 分钟?
    Ret                         ;  若否，则结束此副程序
    clr     minh                ;  若是，则将分钟(high byte)清为 0
    mov     a,hourh             ;
    sub     a,02h               ;
    sz      acc                 ;是否大於 20 小时?
```

```
    jmp     h_20                ;  若否，则跳至 h_20
    inc     hourl               ;  若是，将小时 low byte 加一
    mov     a,hourl             ;
    sub     a,04h               ;
    sz      acc                 ;是否大於 24 小时?
    Ret                         ;  若否，结束此副程序
    clr     hourl               ;  若是，将小时 low byte 和 high byte
                                ;        都清为 0
    clr     hourh               ;
    ret                         ;结束此副程序
h_20:                           ;
    inc     hourl               ;小时 low byte 加一
    mov     a,hourl             ;
    sub     a,0ah               ;
    sz      acc                 ;是否大於 10 小时?
    ret                         ;  若否，则结束此副程序
    clr     hourl               ;  若是，则将小时 low byte 清为 0
    inc     hourh               ;  并将小时 high byte 加一
    ret                         ;结束此副程序
cal_number endp                 ;

;------------------------------------------------------------
show_clock proc                 ;时钟显示副程序
    mov     a,1fh               ;设定显示循环次数
    mov     count1,a            ;
    tabrdl mask                 ;查表取得显示位数
    mov     a,mask              ;
```

```
    or      a,disp                  ;
    mov     pa,a                    ;显示时间数字於 mask 所指定位置
d1:                                 ;
    snz     pb.0            ;(4)    ;分钟输入键是否被按下?
    jmp     min_inc                 ;  若是，则跳至 min_inc 将分钟加 1
    snz     pb.1            ;(4)    ;  若否，则检查小时输入键是否被按下?
    jmp     hour_inc                ;    若是，则跳至 hour_inc 将小时加 1
    mov     a,second                ;    若否，检查秒数是否大於 60 秒
    clr     c                       ;由於程序是以 0.5 秒为计算单位，
    sub     a,78h           ;(6)    ;故检查秒数是否等於 78h=120
    sz      acc                     ;
    jmp     scan_next               ;  若否，则再扫描
    clr     second                  ;  若是，则将秒数设为 0
    call    cal_number      ;(7)    ;同时呼叫副程序将时间增加
                                    ; 1 分钟
    ret                             ;
scan_next:                          ;
    sdz     count1                  ; 递减计数器并检查循环是否结束了
    jmp     d1                      ;  若否，再扫描按键
    ret                             ;  若是，则结束此副程序
min_inc:                            ;
    call      delays                ;延迟以等按下之按键被放开
    snz     pb.0                      ;是否分钟输入键被放开
    jmp     min_inc                   ;  若否，则再侦测
    call      inc_min                 ;  若是，则将分钟加一
    clr     second                    ;清除秒变量
    clr     tmr0c.4                   ;停止定时器 0
```

```
    mov     a,0b0h              ;(5)    ;以便重新载入计数值
    mov     tmr0l,a                     ;
    mov     a,3ch                       ;每 0.5 秒产生一次中断
    mov     tmr0h,a                     ;
    set     tmr0c.4                     ;重新启动定时器 0
    ret                                 ;
hour_inc:                               ;
    call        delays                  ;延迟以等按下之按键被放开
    snz     pb.1                        ;是否小时输入键被放开
    jmp     hour_inc                    ;  若否，则再侦测
    call        inc_hour                ;  若是，则将小时加一
    clr     second                      ;清除秒变量
    clr     tmr0c.4                     ;停止定时器 0
    mov     a,0b0h              ;(5)    ;以便重新载入计数值
    mov     tmr0l,a                     ;
    mov     a,3ch                       ;每 0.5 秒产生一次中断
    mov     tmr0h,a                     ;
    set     tmr0c.4                     ;重新启动定时器 0
    ret                                 ;
show_clock endp                         ;

;------------------------------------------------------------
    org 0f00h                           ;显示位数表
    dc      10h,20h,40h,80h

;------------------------------------------------------------
delays          proc                    ;
```

```
    mov     a,7fh                   ;载入延迟计数变量值
    mov     count1,a                ;
d2:                                 ;
    sdz     count1                  ;逐次递减
    jmp     d2                      ;
    ret                             ;
delays        endp                  ;

;------------------------------------------------------------
inc_hour      proc                  ;
    mov     a,hourh             ;
    sub     a,02h               ;
    sz      acc                 ;是否大於 20 小时?
    jmp     h_201               ;  若否，则跳至 h_201
    inc     hourl               ;  若是，将小时 low byte 加一
    mov     a,hourl             ;
    sub     a,04h               ;
    sz      acc                 ;是否大於 24 小时?
    Ret                         ;  若否，结束此副程序
    clr     hourl               ;  若是，将小时 low byte 和 high byte
                                ;        都清为 0
    clr     hourh               ;
    ret                         ;结束此副程序
h_201:                          ;
    inc     hourl               ;小时的 low byte 加一
    mov     a,hourl             ;
    sub     a,0ah               ;
```

```
    sz      acc                 ;是否大於 10 小时?
    Ret                         ;  若否，则结束此副程序
    clr     hourl               ;  若是，则将小时 low byte 清为 0
    inc     hourh               ;          并将小时 high byte 加一
ret                             ;
inc_hour    endp                ;

;-------------------------------------------------------
inc_min       proc              ;
    inc     minl                ;分钟的 low byte 加一
    mov     a,minl              ;
    sub     a,0ah               ;
    sz      acc                 ;是否大於 10 分钟?
    Ret                         ;  若否，则结束此副程序
    clr     minl                ;  若是，则将分钟(low byte)清为 0
    inc     minh                ;        并将分钟(high byte)加一
    mov     a,minh              ;
    sub     a,06h               ;
    sz      acc                 ;是否大於 60 分钟?
    Ret                         ;  若否，则结束此副程序
    clr     minh                ;  若是，则将分钟(high byte)清为 0
    ret                         ;
inc_min       endp              ;
    end
```

程序说明

程序起始先(1)设定端口 A 为输出端口，端口 B 位元 7 外其余为输入口，把端口 A 控制暂存器设定为 FFH，把端口 B 控制暂存器设定为 7FH，再来就是(2)一连串清除记忆体内值、中断控制设定以及计数/定时器和控制器设定後，就是一个显示时间兼取得按键之循环(3)，如果有调整时或调整分键输入，则呼叫调整时或调整分副程序(4)，作加一及进位调整动作，如果是调整分则加至 59 後变为 00 而且小时的部份不会改变，如果是调整时则加至 23 後变为 00 而且分的部份不会改变，否则一直执行显示时间之循环。

由於我们是设定每 0.5 秒计数/计数器中断一次(5)，所以当中断达到 120 次时即为 1 分钟，当在执行中断常式时，会对一个秒数存放器作加一的动作，我们在显示时间之循环只是一直去扫描时间的数字，再来就是去判断秒数存放器是否已经达到中断 120 次(6)，如果是的话就作加一分的动作，以及最重要的时间进位调整(7)，每 60 分进位 1 小时，23 时进位後是 0 时等。再则程序大部份皆是在作扫描时间的数字，由於只有用一个端口作 4 位元输出数字码，因此把一个端口就分成一半，4 个位元作数字码输出，4 个位元作位数扫描，所以实际情形是在某一段时间内只有一个位数显示，但是由於扫描显示速度够快，因此可以看到 4 位数同时在亮。

第 十 四 章

并行端口介面应用

14

本单元所要介绍的是并行接口介面应用。由於并行传输速度必定是比串行传输快，但其所需要的传输线比较多，而且信号的电压基准是一般的 TTL 基准，因此最好是在几公尺范围内，超过此范围就有可能因为信号线过长，导致信号在传输过程中，因为信号线过长阻抗过大，而传输的信号会错掉，因此并行接口介面通常适合使用於短距离数据传输。由於其所需要的传输线比较多，而且必须作握手信号以控制数据的进出，因此很适合用我们公司 HT48CX0 系列芯片来作控制。

ROM 模拟器

首先，ROM 模拟器是使用在需要监控程序的控制系统上，由於在开发展系统之初，并不会真正烧写 ROM，一般我们常常会使用 EPROM 来代替，因为其特性可利用紫外线，把原有的程序清除掉重新再烧写，方便设计者在发展系统之初，能够测试之用，但是有一缺点就是，EPROM 必须要使用紫外线清除，而且清除的时间必须够久才可完成。为了弥补这一缺点，我们就可以使用 ROM 模拟器。它是利用打印机介面作并行传输，把程序载入到 ROM 模拟器的 SRAM 上，其花费的时间相当的短，可以在开发程序的初期节省相当多的时间。这一范例是使用一个 28 Pin 接脚型号的 HT48C30 ，如图所示(图 13.1)。

电路设计

I/O 口 PA0~PA7、PB0~PB7、PC0~PC3 皆设置为输出，端口 A 作为低位字节地址线，端口 B

作为高位字节地址线，端口 C 作为握手信号控制线。数据线部份乃是利用 74HC374 以及 /STROBE 来锁存数据，再把数据写到 SRAM 里去。使用者必须另外接一个转接线路，把 SRAM 的地址线、数据线和控制线连接到一个 ROM 的 SOCKET，这样即可使用。

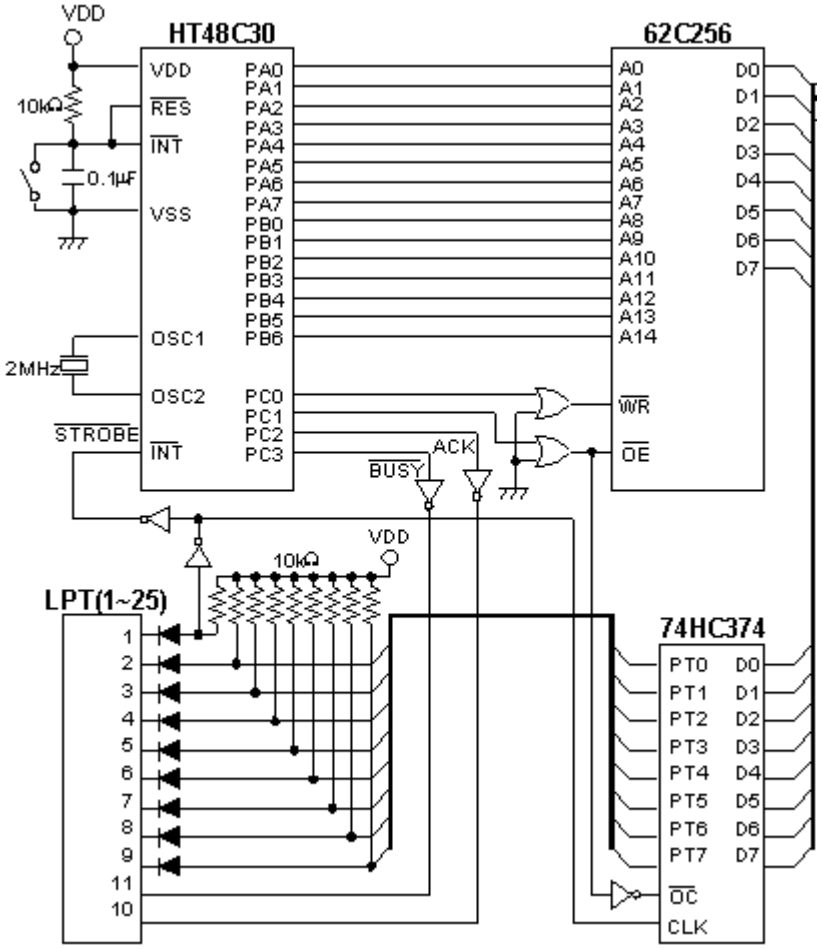

图 13.1

由於程序的控制必须根据控制信号时序图，才可以从打印机端口接收正确的数据，因此整个程序是对照控制信号时序来编写。

程序

```
#include  ht48300.inc

;------------------------------------------------------------

data   .section 'data'          ;数据设定区

addrl     db  ?                 ;低位字节地址
```

```
addrh     db  ?                  ;高位字节地址
timer_ov  db  ?                  ;定时计数器
;-------------------------------------------------------------
code   .section at 0 'code'      ;程序区
   org    00h                    ;各向量位址设定
   jmp    start
   org    04h                    ;外部中断入口地址
   jmp    int_sub                ;跳至外部中断服务程序
   org    08h                    ;定时器中断入口地址
   jmp    timer_sub              ;跳至定时器中断服务程序
int_sub:                         ;外部中断服务子程序
   mov    a,00000011b            ;将 a 设为(busy=0,ack=0,out=1,wr=1)
   mov    pc,a          ;(5)     ;输出端口 C
   reti                          ;由中断服务子程序返回
timer_sub:                       ;定时器中断服务子程序
   inc    timer_ov               ;定时/计数器加一
   mov    a,timer_ov             ;将定时/计数器内值送 a
   xor    a,20h                  ;把 a 内值与 20h 作逻辑异或操作
   sz     acc                    ;判断定时/计数器是否等於 20h
   jmp    timer_nov              ;如果不是则跳至 timer_nov 处
   mov    a,00001011b            ;如果是则 a 设为(00001011)
   reti                ;(4)     ;由定时器中断服务子程序返回
timer_nov:                       ;timer_nov 处
   clr    acc                    ;清除 ACC
   reti                          ;由定时器中断服务子程序返回
start:                           ;
   mov    a,07h          ;(1)    ;设定 a 为 07h (外部中断允许、时间定时器中
```

```
                                    ;断允许)
    mov     intc,a                  ;将 a 放至中断控制器
    mov     a,80h                   ;设定 a 为 80h (时间计时器，内部定时模式)，
    mov     tmrc,a                  ;将 a 放至时间计时控制器。
    clr     pcc                     ;将端口 C 设定为输出端
top:                        ;(13)   ;top 处
    set     pac             ;(2)    ;将端口 A 设定为输入端口
    set     pbc                     ;将端口 B 设定为输入端口
    clr     addrl                   ;清除低位字节地址址
    clr     addrh                   ;清除高位字节地址址
    mov     a,00001001b             ;将 a 设为(busy=1,ack=0,out=0,wr=1)
    mov     pc,a            ;(3)    ;输出至端口 C
    clr     acc                     ;清除 ACC
store:                              ;store 处
    snz     acc.0           ;(4)    ;判断有无中断进入，假如有会使 acc.0 不
    jmp     store                   ;为 0，而跳离此循环
    clr     pac             ;(5)    ;设定端口 A 为输出端口
    clr     pbc                     ;设定端口 B 为输出端口
next:                               ;next 处
    mov     a,addrl         ;(6)    ;将低位字节地址送 a
    mov     pa,a                    ;输出至端口 A
    mov     a,addrh                 ;将高位字节地址送 a
    mov     pb,a                    ;输出至端口 B
    inc     addrl                   ;将低位字节地址加一
    sz      addrl                   ;判断低位字节地址是否为 0
    jmp     no_inc                  ;如果不是则跳至 no_inc 处
    mov     a,1                     ;如果是则高位地址加一作进位
```

```
    addm    a,addrh             ;
no_inc:                         ;no_inc 处
    mov     a,00000010b         ;将 a 设为(busy=0,ack=0,out=1,wr=0)
    mov     pc,a        ;(7)    ;输出至端口 C
    mov     a,10h               ;设定延迟时间
    call      delays    ;(8)    ;调用延迟子程序
    mov     a,00000011b         ;将 a 设为(busy=0,ack=0,out=1,wr=1)
    mov     pc,a                ;输出端口 C
    mov     a,7                 ;设定延迟时间
    call      delays            ;调用延迟子程序
    mov     a,00000111b         ;将 a 设为(busy=0,ack=1,out=1,wr=1)
    mov     pc,a                ;输出至端口 C
    mov     a,5                 ;设定延迟时间
    call      delays            ;调用延迟子程序
    mov     a,00000011b         ;将 a 设为(busy=0,ack=0,out=1,wr=1)
    mov     pc,a        ;(9)    ;输出端口 C
    mov     a,5                 ;设定延迟时间
    call      delays            ;调用延迟子程序
    clr     tmr                 ;清除计时暂存器
    set     tmrc.4      ;(10)   ;设定定时控制器开始计时
    mov     a,00001011b         ;将 a 设为(busy=1,ack=0,out=1,wr=1)
    mov     pc,a        ;(11)   ;输出至端口 C
    clr     acc                 ;清除 ACC
store1:                         ;store1 处
    snz     acc.0       ;(4)    ;判断是否外部中断或者定时到时进入中断
    jmp     store1              ;如果有外部中断或定时器到则 a 不为 0 而跳出此
循环
```

```
    clr     tmrc.4                  ;设定定时控制器停止计时
    snz     acc.3           ;(12)   ;判断是外部中断还是定时器到时中断
    jmp     next                    ;外部中断的话 acc.3=0
    jmp     top                     ;定时到时中断 acc.3=1
delays      proc                    ;延迟子程序
d1:                                 ;d1 处
    sdz     acc                     ;作 acc 减一的动作，并判断 acc 是否为 0
    jmp     d1                      ;acc 不为 0 的话，跳至 d1 处
    ret                             ;返回调用延迟子程序处
delays      endp                    ;
    end                             ;
```

程序说明

程序起始先(1)设定中断控制，允许外部中断及定时计数中断，再把定时计数模式设定为内部定时模式，及将端口 C 设定为输出端口，端口 A 和端口 B 先行设定为输入端口(2)，清除低位字节地址及高位字节地址，(3)再将控制信号初始设定为 BUSY=0，/ACK=1，OUT=0，/WR=1 输出至端口 C，清除 ACC 等待中断输入，因为只要有中断信号输入就会改变 ACC 值(4)，中断服务子程序内会将 ACC 值改变为 BUSY=1，/ACK=1，OUT=1，/WR=1 输出至端口 C，由於 LPT 端口的 1 号引脚 /STROBE 直接接到外部中断 /INT 去，所以只要有 /STROBE 进来就会去执行中断服务子程序，但是必须根据 BUSY 信号决定是否再次送出 /STROBE，因此我们必须等收到数据而且处理好了後再送出 BUSY 信号。

经由 LPT 端口送出的 /STROBE 信号表示数据已送出，因此我们将端口 A 和端口 B 设定为输出端口(5)，再把低位字节地址及高位字节地址分别由端口 A 及端口 B 输出(6)，然後再把地址值加一，而在送出/STROBE 信号的同时，由於它经过一非门输入至 74HC374 的 CLK，而 LPT 端口的第 2~9 脚数据线是接到 74HC374 数据线，因此在此同时 LPT 端口的数据会锁存于 74HC374 的输出线上，所以程序接着送出 /WR 信号(7)，把数据写到 SRAM 上，延迟一段时间後返回(8)，再将 /ACK 信号送出确认(9)，延迟一段时间後返回，这样就写入一笔数据，然后清除 BUSY 信号 (11)，允许下一笔数据进入，即 /STROBE 信号进来，同时设定定时计数器开始计数(10)。

此时有两种情形会跳出侦测 ACC 是否为 0 的循环(4)，只要有外部中断或者是定时计数器计数到时中断，都会使得 ACC 不为 0，而且两种中断对 ACC 的值改变有所不同，这可以由 ACC.3 判断出来(12)，如果是外部中断输入则跳至 NEXT 处接下一笔数据，如果是定时计数器计数到时中断，表示已有一段时间没有数据送出，此时就认为数据已送完，再来将

OUT 信号送出，表示 SRAM 在输出状态可以被外部使用，并且再来的一个动作是将原来设定为地址输出端口的 A 口和 B 端口，更改设定为输入端口(13)，因为地址线是接在一起，如果还是设定为输出端口，将会把 SRAM 的地址线状态弄乱掉，只要把端口 A 和端口 B 设定为输入端口，就不会去干扰到 SRAM 上的地址线状态。

附 錄 A

指令集速查表

A

助记符	运算过程	影响标志位
算术运算		

ADD A,[m]	ACC←ACC+[m]	Z,C,AC,OV
ADDM A,[m]	[m]←ACC+[m]	Z,C,AC,OV
ADD A,X	ACC←ACC+X	Z,C,AC,OV
ADC A,[m]	ACC←ACC+[m]+C	Z,C,AC,OV
ADCM A,[m]	[m]←ACC+[m]+C	Z,C,AC,OV
SUB A,X	ACC←ACC-X	Z,C,AC,OV
SUB A,[m]	ACC←ACC-[m]	Z,C,AC,OV
SUBM A,[m]	[m]←ACC-[m]	Z,C,AC,OV
SBC A,[m]	ACC←ACC-[m]-C	Z,C,AC,OV
SBCM A,[m]	[m]←ACC-[m]-C	Z,C,AC,OV
DAA [m]	if ACC.3~ACC.0>9 OR AC=1 then ACC.3~ACC.0←ACC.3~ACC.0+6	C
	if ACC.7~ACC.4>9 or C=1 then ACC.7~ACC.4 ← ACC.7~ACC.4+6;C=1	
	[m]←ACC	Z
	ACC←[m]+1	Z
	[m]←[m]+1	Z
	ACC←[m]-1	Z
	[m]←[m]-1	
INCA [m]		
INC [m]		
DECA [m]		
DEC [m]		

逻辑运算		
AND A,[m]		Z
OR A,[m]		Z
XOR A,[m]		Z
ANDM A,[m]		Z
ORM A,[m]		Z
XORM A,[m]		Z
AND A,X		Z
OR A,X		Z
XOR A,X		Z
CPL [m]		Z
CPLA [m]		
递增和递减		
INCA [m]	递增数据存储器,结果放入累加器	Z
INC [m]	递增数据存储器	Z
DECA [m]	递减数据存储器,结果放入累加器	Z
DEC [m]	递减数据存储器	Z
移位		
RRA [m]	将数据存储器向右移位,结果放入累加器	无
RR [m]	将数据存储器向右移位	无
RRCA [m]	带进位将数据存储器向右移位,结果放入累加器	C
RRC [m]	带进位将数据存储器向右移位	C
RLA [m]	将数据存储器向左移位,结果放入累加器	无
RL [m]	将数据存储器向左移位	无
RLCA [m]	带进位将数据存储器向左移位,结果放入累加器	C
RLC [m]	带进位将数据存储器向左移位	C
数据传送		
MOV A,[m]	传送数据存储器至累加器	无
MOV [m],A	传送累加器至数据存储器	无
MOV A,X	传送立即数至累加器	无
位运算		
CLR [m],I	清除数据存储器的位	无
SET [m],I	置位数据存储器的位	无

转移		
JMP addr	无条件跳转	无
SZ [m]	若数据存储器为零,则跳过下一条指令	无
SZA [m]	将数据存储器传送至累加器,若内容为零,则跳过下一条指令	无
SZ [m],I	若数据存储器的第 i 位为零,则跳过下一条指令	无
SNZ [m],i	若数据存储器的第 i 位不为零,则跳过下一条指令	无
SIZ [m]	递增数据存储器,若结果为零, 则跳过下一条指令	无
SDZ [m]	递减数据存储器,若结果为零, 则跳过下一条指令	无
SIZA [m]	递增数据存储器将结果放入累加器,若结果为零,则跳过下一条指令	无
SDZA [m]	递减数据存储器将结果放入累加器,若结果为零,则跳过下一条指令	无
CALL addr	子程序调用	无
RET	从子程序返回	无
RET A,X	从子程序返回,并将立即数放入累加器	无
RETI	从中断返回	无
读表指令		
TABRDC [m]	读取本页的 ROM 码至数据存储器和 TBLH	无
TABRDL [m]	读取末页的 ROM 码至数据存储器和 TBLH	无
其它指令		
NOP	空指令	无
CLR [m]	清除数据存储器	无
SET [m]	置位数据存储器	无
CLR WDT	清除看门狗定时器	TO,PD
CLR WDT1	预先清除看门狗定时器	TO*,PD*
CLR WDT2	预先清除看门狗定时器	TO*,PD*
SWAP [m]	交换数据存储器的高低字节	无
SWAPA [m]	交换数据存储器的高低字节,结果放入累加器	无
HALT	进入暂停省电模式	TO,PD

附注:

x: 8 位立即数

m: 7 位数据存储器地址

A: 累加器

i：0~7 位

addr：11 位程序存储器地址

√：影响标志位

—：不影响标志位

*：执行状态可能会影响标志位