
**Column #80, December 2001 by Jon Williams:**

# Security Concerns

*In the past I was always a bit of jokey guy when starting this column. It's not that I've lost my ability to laugh and have fun, it's just that recent events have caused me – and many – to be a bit more serious and cautious in our approach to life.*

Security concerns have hit me, personally, twice in the last two days. Yesterday I got an e-mail from a guy who wanted to know how to use the Stamp to talk to a GPS unit so he could control the steering vanes on a large rocket. A rocket...or a guided missile intended to do harm? Since it's impossible to tell via e-mail, I politely declined his request for help. I love helping Stamp customers with their applications, but this one just made me too nervous about the possible negative consequences of his success.

Then, just this morning, it dawned on me that the airline ticket my friend purchased for me is in my stage name, not my legal name (there are too many guys in the business named Jon Williams, so I go by Jon McPhalen as an actor). I called the airline and will have to make a special trip to the airport this afternoon to "fix" the ticket. I have all the legal documentation required by the airline (I own my stage name), but FAA security specialists at the airport might not take the same point of view.

So, security is, indeed, a concern. My friend Chuck pointed this out and how he'd heard from Stamp users interested in building their own security systems. Our project this month is just that –

the beginnings of a very simple security system. Please keep in mind that my job here is to teach you Stamp programming and interfacing and that many of the applications I present are only suitable for training purposes. If you decide to build your own security system, proceed with extreme caution – as if your decided to build your own car. Your security is at stake.

Okay, enough serious stuff. Let's have some fun and build a project.

**Lots O' Pins On the –40**

I've made it pretty clear that my favorite Stamp is the BS2p; it's fast, got great features and even comes in a 40-pin variant. Since most of my projects are fairly small, I never needed the 40-pin package – until now. The BS2p-40 has 16 additional I/O pins. They're addressed somewhat differently that you might expect, although the scheme makes sense in context to the BASIC Stamp.

The first 16 pins are known as the main I/O pin group. The additional 16 pins are known as the auxiliary I/O group. Since the Stamp uses 16-bit variables, we will still deal with pins 0 – 15. What we need to do, then, is tell the Stamp which group of pins we're working with.

The main group is accessed with the keyword **MAINIO**. Any reference to I/O pins after this command will refer to the lower (main) group. The auxiliary group is accessed with the keyword **AUXIO**. After this command, I/O pin references will be to the upper (auxilliary) group.

There is one more command used by the BS2p-40: **IOTERM**. This command requires a parameter that specifies which I/O group to use (0 = main, 1 = auxiliary). **IOTERM** is useful for general-purpose subroutines like this:

```
Set_Pin_High:
  IOTERM (pinNum / 16)
  HIGH (pinNum // 16)
  RETURN
```

With this subroutine your could set pinNum to any value between 0 – 31 and the Stamp would set the physical pin high.

Since the BS2p-40 has a second set of I/O pins, it also has a second **Dirs** register, **Outs** register and **Ins** register. These are accessed by using **AUXIO** or **IOTERM** as I've described above.

### Scan 'Em, Danno

Pin polling is a new feature in the BS2p series that has caused a bit of confusion – especially for those wishing for true interrupts. Conceptually, it's actually fairly simple: When pin polling is enabled, the BS2p will look at specified input pins between each PBASIC instruction and can take some action. Here's what you can have the Stamp do:

Nothing
Set an output pin to specified state
Run another program
Wait (pause program) until interrupt condition occurs
Any combination of 2, 3 and 4

In our particular project, we're going to use the upper eight pins on the auxillary I/O port for our alarm inputs. What this means is we'll have to use the **AUXIO** command when defining our polled input pins. Once defined and enabled, we don't have to worry about which I/O set is active, pin polling looks at all phsycial pins meaning that our auxilliary input pins are still active for polling, even when we have the main I/O group selected.

### Can I See A Menu Please?

Since most alarm systems are menu driven we should probably follow suit. This program will use a multi-tiered menu system like we built back in ( ). The user interface will consist of four buttons and a two-line LCD. If you decide to improve the project with a larger keyboard, there are enough pins open to scan a 4x4 matrix.

**Figure 80.1: Button Menu Circuitry**

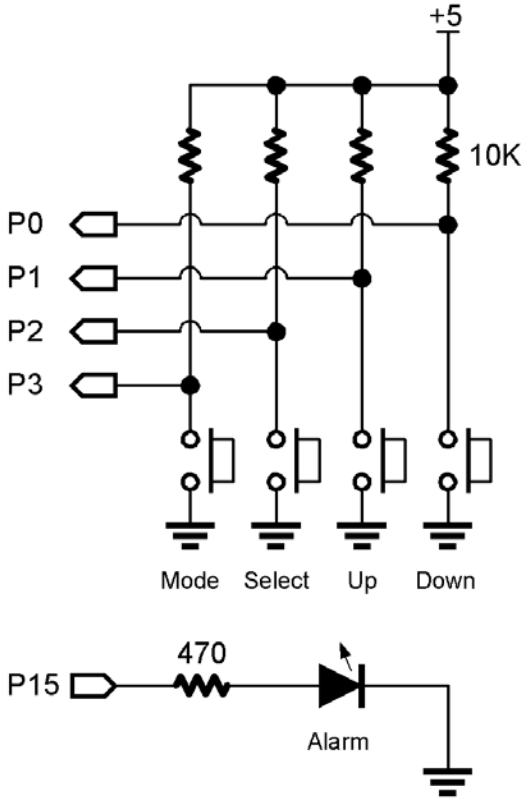
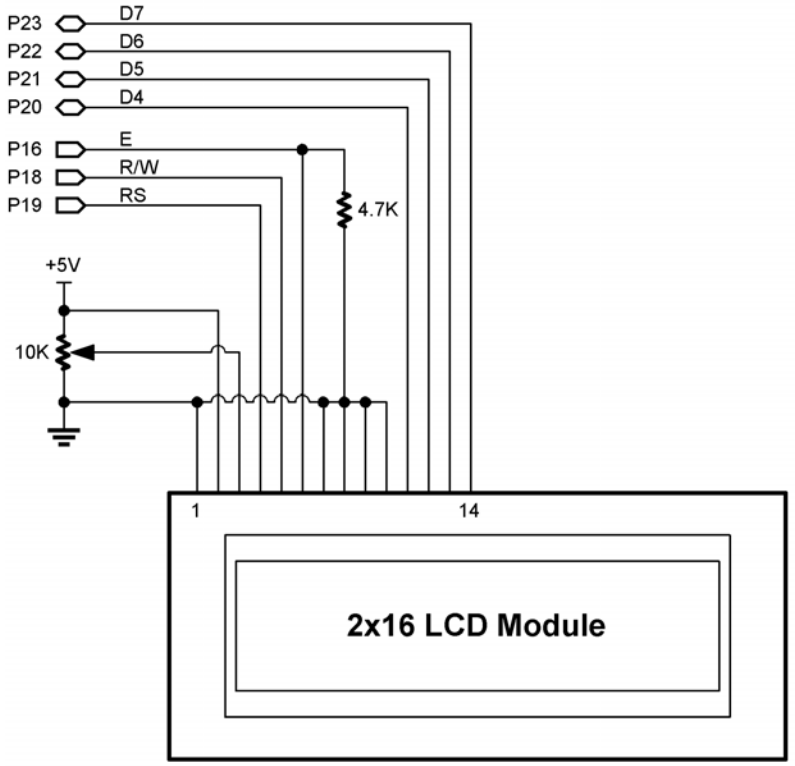


**Figure 80.2: LCD Setup**




Our menu, then, will look like this:

Alarm On
Alarm Off
View Alarms
Clear Alarms
Set Clock

Within the Set Clock  menu we will be able to set the hours, minutes and day.

**Figure 80.3: PCF8583 Setup**

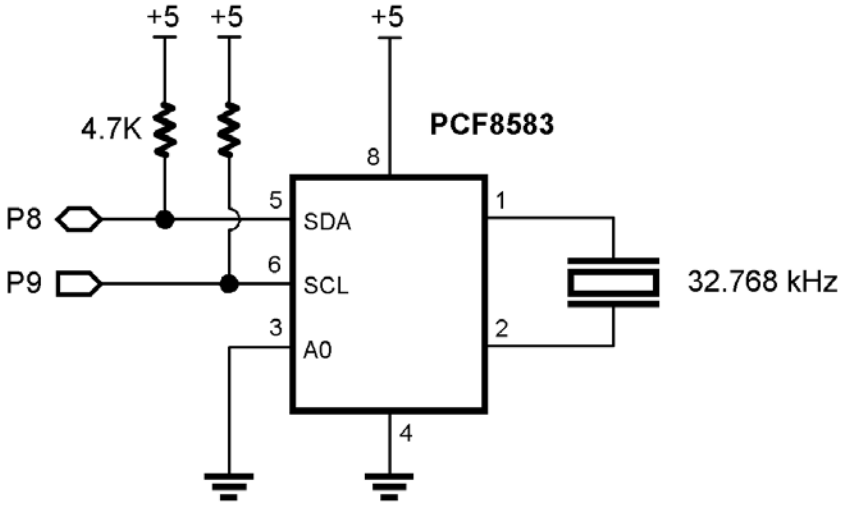
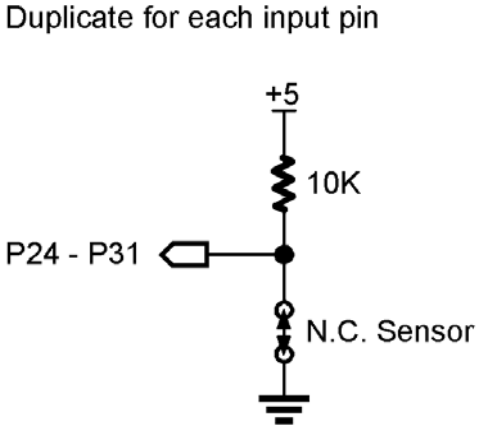



**Figure 80.4: Sensory Circuit**



```
' -----[ Title ]-------------------------------------------------------------
' Program Listing 80.1
' File...... ALARM.BSP
' Purpose... Simple Alarm System Using The BS2p-40
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started... 28 OCT 2001
' Updated... 06 NOV 2001

' { $STAMP BS2p }


' -----[ Program Description ]-------------------------------------------------
'
' This program uses a BS2p-40 to demonstrate polled pins and the use of the
' BS2p-40's extended I/O pins.  It is also a good demonstration of an LCD-based
' HMI (human machine interface) for Stamp projects.
'
' NOTE: THIS PROGRAM IS FOR EDUCATIONAL PURPOSES ONLY.
'       USE TO PROTECT PROPERTY AT YOUR OWN RISK.
'
' General connections:
' -- Main 0 - 8         User keys (keyboard, etc.)
' -- Main 9 - 15        I2C bus, alarm LED, misc I/O (future)
' -- Aux  0 - 8         LCD output
' -- Aux  9 - 15        Alarm inputs (pulled up, NC to ground, 1 = alarm)


' -----[ Revision History ]---------------------------------------------------
'


' -----[ I/O Definitions ]-----------------------------------------------------
'
AlarmIns        VAR     InH                     ' AUX  - high byte
LCDpin          CON     16                      ' AUX  - low byte
AlarmLED        CON     15                      ' MAIN - pin 15
I2Cpin          CON     8                       ' MAIN - pins 8 & 9
Buttons         VAR     InA                     ' MAIN - low nib


' -----[ Constants ]-----------------------------------------------------------
'
M Status        CON     0                       ' mode values
M Clear         CON     1
M System        CON     2
M Clock         CON     3

Modes           CON     4                       ' menu modes
TimeOut         CON     150                     ' mode timer (loop iterations)
```

```
NoCmd           CON     $00                     ' No command in LCDOUT
ClrLCD          CON     $01                     ' clear the LCD
CrsrHm          CON     $02                     ' move cursor to home position
CrsrLf          CON     $10                     ' move cursor left
CrsrRt          CON     $14                     ' move cursor right
DispLf          CON     $18                     ' shift displayed chars left
DispRt          CON     $1C                     ' shift displayed chars right
DDRam           CON     $80                     ' Display Data RAM control
CGRam           CON     $40                     ' Custom character RAM
Line1           CON     $80                     ' DDRAM address of line 1
Line2           CON     $C0                     ' DDRAM address of line 2

UpAr            CON     0                       ' up arrow char number
DnAr            CON     1

Wr8583          CON     %10100000               ' write to RTC
Rd8583          CON     %10100001               ' read from RTC


' -----[ Variables ]-------------------------------------------------------
'
hours           VAR     Byte
minutes         VAR     Byte
day             VAR     Nib                     ' 0 = Sunday, 6 = Saturday

rtcCtrl         VAR     Byte                    ' [0] control/status
rtcHuns         VAR     Byte                    ' [1] hundredths (bcd)
rtcSecs         VAR     Byte                    ' [2] seconds (bcd)
rtcMins         VAR     Byte                    ' [3] minutes (bcd)
rtcHrs          VAR     Byte                    ' [4] hours (bcd)
rtcYrDate       VAR     Byte                    ' [5] year & date (bcd+)
rtcMoDay        VAR     Byte                    ' [6] day & month (bcd+)

mode            VAR     Nib                     ' main menu level
modeTimer       VAR     Byte                    ' auto time-out to Status mode
level           VAR     Nib                     ' sub menu level

btns            VAR     Nib                     ' button input
btnMode         VAR     btns.Bit3
btnSelect       VAR     btns.Bit2
btnUp           VAR     btns.Bit1
btnDn           VAR     btns.Bit0

alarmStatus     VAR     Bit                     ' 0 = Off, 1 = On
blink           VAR     Bit                     ' cursor blink control

temp1           VAR     Byte                    ' general purpose vars
temp2           VAR     Byte


' -----[ EEPROM Data ]-----------------------------------------------------
```

```
'
UpArrow         DATA    $00,$00,$04,$0E,$1F,$00,$00,$00
DnArrow         DATA    $00,$00,$00,$1F,$0E,$04,$00,$00

Su              DATA    "SUN",0                 ' day names
Mo              DATA    "MON",0
Tu              DATA    "TUE",0
We              DATA    "WED",0
Th              DATA    "THU",0
Fr              DATA    "FRI",0
Sa              DATA    "SAT",0


' -----[ Initialization ]----------------------------------------------------
'
Initialize:
  ' setup alarm input pins
  AUXIO                                         ' point to auxilliary i/o pins
  POLLMODE 0                                    ' clear and disable polling
  POLLIN  8, 1                                  ' inputs look for high
  POLLIN  9, 1
  POLLIN 10, 1
  POLLIN 11, 1
  POLLIN 12, 1
  POLLIN 13, 1
  POLLIN 14, 1
  POLLIN 15, 1

  ' setup LCD
  PAUSE 500                                     ' let the LCD settle
  IOTERM (LCDpin / 16)                          ' point to LCD I/O  bank
  LCDCMD (LCDpin // 16), %00110000 : PAUSE 5    ' 8-bit mode
  LCDCMD (LCDpin // 16), %00110000 : PAUSE 0
  LCDCMD (LCDpin // 16), %00110000 : PAUSE 0
  LCDCMD (LCDpin // 16), %00100000 : PAUSE 0    ' 4-bit mode
  LCDCMD (LCDpin // 16), %00101000 : PAUSE 0    ' 2-line mode
  LCDCMD (LCDpin // 16), %00001100 : PAUSE 0    ' no crsr, no blink
  LCDCMD (LCDpin // 16), %00000110              ' inc crsr, no disp shift
  LCDCMD (LCDpin // 16), ClrLCD

  LCDCMD (LCDpin // 16), CGRam                  ' prepare to write CG data
  FOR temp2 = UpArrow TO (DnArrow + 7)          ' build 2 custom chars
    READ temp2, temp1                           ' get byte from EEPROM
    LCDOUT (LCDpin // 16), NoCmd, [temp1]       ' put into LCD CGRAM
  NEXT

  ' setup alarm output pin
  MAINIO                                        ' point to main i/o pins
  POLLOUT AlarmLED, 1

  ' setup keyboard
```

```
  MAINIO                                        ' point to lower group
  DirA = %0000                                  ' make pins input


' -----[ Main Code ]-------------------------------------------------------------
'
Main:
  GOSUB Scan Buttons                            ' get button inputs
  mode = mode + btnMode // Modes                ' update current mode

Check Mode Timer:
  modeTimer = (modeTimer + 1) * (1 - btnMode)   ' inc if Mode button not pressed
  IF (modeTimer < TimeOut) THEN Run_Mode        ' if not expried, run mode
  mode = M Status                               ' - otherwise, set to Status
  modeTimer = 0

Run Mode:
  BRANCH mode, [Show_Status, Clear_Alarms, Set_System, Clock_Set]

Loop Pad:                                       ' pad loop for button presses
  PAUSE 150
  GOTO Main
  END


' -----[ Main Menu Fuctions ]-----------------------------------------------
'
' -- called with BRANCH
' -- should end with GOTO Loop Pad

' *************
' System Status
' *************

Show Status:
  level = 0                                     ' reset sub-menu level
  GOSUB Get_Clock                               ' get and display current clock
  IOTERM (LCDpin / 16)
  LCDCMD (LCDpin // 16),%00001100               ' no crsr, no blink
  LCDOUT (LCDpin // 16), Line1, [HEX2 rtcHrs,":",HEX2 rtcMins,":",HEX2 rtcSecs]
  LCDOUT (LCDpin // 16), NoCmd, ["     "]

  LOOKUP day, [Su,Mo,Tu,We,Th,Fr,Sa], temp2     ' point to day name
  GOSUB Print Str                               ' print it on LCD
  BRANCH alarmStatus, [Is Off,Is On]            ' show system status

Is Off:
  LCDOUT (LCDpin // 16), Line2, [" * SYSTEM OFF * "]
  GOTO Show_Status_Done

Is_On:
```

```
  GET 131, temp2                                ' grab alarm bits
  IF (temp2 > 0) THEN Show Alarm Bits
  LCDOUT (LCDpin // 16), Line2, ["   All Clear    "]
  GOTO Show Status Done

Show_Alarm_Bits:
  ' alternate between latched event and current inputs every other second
  BRANCH rtcSecs.Bit0, [Show Latched,Show Current]

Show Latched:
  LCDOUT (LCDpin // 16), Line2, ["Alarms  ",BIN8 temp2]
  GOTO Show_Status_Done

Show Current:
  AUXIO
  LCDOUT (LCDpin // 16), Line2, ["Now     ",BIN8 AlarmIns]

Show_Status_Done:
  PAUSE (200 * btnMode)                         ' extra delay if first entry
  GOTO Loop Pad


' ***************************
' Clear Alarm
' -- will clear if Up pressed
' ***************************

Clear Alarms:
  GET 131, temp2                                ' read latched alarm bits
  IF (temp2 > 0) THEN Check_Clear               ' alarms enabled and active?
  mode = M_System                               ' skip to system set if no alarms
  GOTO Set System

Check Clear:
  IOTERM (LCDpin / 16)
  LCDOUT (LCDpin // 16), Line1, ["Clear Alarms?   "]
  LCDOUT (LCDpin // 16), Line2, ["  ",UpAr, " Yes   ",DnAr, " No  "]

No Clear:
  IF (btnDn = 0) THEN Yes Clear                 ' is Dn pressed?
  mode = M Status                               ' - return to Status mode
  GOTO Clear_Alarm_Done

Yes Clear:
  IF (btnUp = 0) THEN Clear Alarm Done          ' is Up pressed?
  LCDOUT (LCDpin // 16), Line1, ["Clearing Alarms "]
  LCDOUT (LCDpin // 16), Line2, [REP " "\16]

  POLLMODE 9                                    ' clear polling, save setup
  PAUSE 1000                                    ' pause to show message
  POLLMODE (alarmStatus + 9)                    ' reset if alarms = On
```

```
  mode = M_Status                                ' return to Status mode

Clear Alarm Done:
  PAUSE (200 * btnMode)                          ' extra delay if first entry
  GOTO Loop Pad


' ************************************
' Set System (enable alarms)
' -- Up = System On
' -- Dn = System Off
' -- will reset alarm if Up or Dn pressed
' ************************************

Set System:
  IOTERM (LCDpin / 16)
  LCDOUT (LCDpin // 16), Line2, ["  ",UpAr," On   ",DnAr," Off "]
  LCDOUT (LCDpin // 16), Line1, ["Set System "]
  BRANCH alarmStatus, [System_Off,System_On]

System Off:
  LCDOUT (LCDpin // 16), NoCmd, ["(OFF)"]
  GOTO Check System Set

System_On:
  LCDOUT (LCDpin // 16), NoCmd, [" (ON)"]

Check System Set:
  IF ((btns & %11) = 0) THEN Set System Done     ' skip if not Up or Dn
  alarmStatus = btnUp                            ' set new status
  POLLMODE (alarmStatus + 9)                     ' set latching POLLMODE
  mode = M Status                                ' back to Status mode

Set System Done:
  PAUSE (100 * btnMode)                          ' extra delay if first entry
  GOTO Loop_Pad


' *********
' Set Clock
' *********

Clock_Set:
  IOTERM (LCDpin / 16)
  LCDCMD (LCDpin // 16), %00001100               ' no cursor
  LCDOUT (LCDpin // 16), Line1, ["Set Clock       "]
  LCDOUT (LCDpin // 16), Line2, ["   ", DEC2 hours,":",DEC2 minutes,"  "]

  LOOKUP day, [Su,Mo,Tu,We,Th,Fr,Sa], temp2
  GOSUB Print Str
  LCDOUT (LCDpin // 16), NoCmd, ["   "]
```

```
  IF ((btns & %0111) = 0) THEN Check Level      ' check for press
  modeTimer = 0                                 ' clear mode timer if press

Check Level:
  level = level + btnSelect // 3                ' update level selection
  BRANCH level, [Hrs_Set,Mins_Set,Day_Set]      ' branch to sub-menu code
  GOTO Clock Set Done

Hrs Set:
  LCDCMD (LCDpin // 16), Line2 + 4              ' move to hours position
  GOSUB Blink_Cursor
  BRANCH btnUp, [Check_Hrs_Down]
  hours = hours + 1 // 24
  GOSUB Put Clock                               ' update RTC
  GOTO Clock Set Done

Check_Hrs_Down:
  BRANCH btnDn, [Clock_Set_Done]
  hours = hours + 23 // 24
  GOSUB Put Clock
  GOTO Clock Set Done

Mins_Set:
  LCDCMD (LCDpin // 16), Line2 + 7              ' move to minutes position
  GOSUB Blink Cursor
  BRANCH btnUp, [Check Mins Down]
  minutes = minutes + 1 // 60
  GOSUB Put Clock
  GOTO Clock_Set_Done

Check Mins Down:
  BRANCH btnDn, [Clock Set Done]
  minutes = minutes + 59 // 60
  GOSUB Put Clock
  GOTO Clock_Set_Done

Day Set:
  LCDCMD (LCDpin // 16), Line2 + 12             ' move to day position
  GOSUB Blink Cursor
  BRANCH btnUp, [Check Day Down]
  day = day + 1 // 7
  GOSUB Put_Clock
  GOTO Clock Set Done

Check Day Down:
  BRANCH btnDn, [Clock Set Done]
  day = day + 6 // 7
  GOSUB Put_Clock

Clock_Set_Done:
```

```
  PAUSE (200 * btnMode)                          ' extra delay if first entry
  GOTO Loop Pad


' -----[ Subroutines ]-------------------------------------------------------
'
Scan_Buttons:                                    ' debounce four buttons
  MAINIO                                         ' keys are connected to main I/O
  btns = %1111                                   ' assume pressed
  FOR temp2 = 1 TO 5                             ' scan 5 times
    btns = btns & ~Buttons                       ' check for press
    PAUSE 10                                     ' debounce delay
  NEXT
  RETURN


Put Clock:                                       ' send data to RTC
  rtcSecs = 0
  rtcMins.HighNib = minutes / 10                 ' convert regs to BCD
  rtcMins.LowNib = minutes // 10
  rtcHrs.HighNib = hours / 10
  rtcHrs.LowNib = hours // 10
  rtcMoDay = 1 | (day << 5)                      ' pack weekday in
  IOTERM (I2Cpin / 16)                           ' point to I2C bus I/O bank
  I2COUT (I2Cpin // 16), Wr8583, 2, [STR rtcSecs\5]
  RETURN


Get Clock:                                       ' read data from RTC
  IOTERM (I2Cpin / 16)
  I2CIN (I2Cpin // 16), Rd8583, 0, [STR rtcCtrl\7]
  minutes = (rtcMins.HighNib * 10) + rtcMins.LowNib
  hours = (rtcHrs.HighNib * 10) + rtcHrs.LowNib
  day = rtcMoDay >> 5
  RETURN


Print Str:                                       ' print zero-terminated string
  READ temp2, temp1                              ' read a character
  IF (temp1 = 0) THEN Print Done                 ' done?
  IOTERM (LCDpin / 16)
  LCDOUT LCDpin, NoCmd, [temp1]                  ' print the character
  temp2 = temp2 + 1                              ' point to next
  GOTO Print Str:                                ' go get it
Print Done:
  RETURN


Blink_Cursor:                                    ' blink every other loop
  temp2 = %00001100                              ' no cursor
  temp2.Bit1 = blink
```

```
  IOTERM (LCDpin / 16)
  LCDCMD (LCDpin // 16), temp2
  blink = ~blink                                ' invert blink control
  RETURN
```