Column #87 July 2002 by Jon Williams:

Multi-Bank Programming

If you work with BASIC Stamps long enough there will come a time when you either run out of space or wish you could change some part of your program (usually the user interface device) without impacting all the hard work you applied to your control code. Or both. Welcome to the club.

If space becomes the issue, that can be certainly solved with one of the multi-bank BASIC Stamps (BS2e, BS2sx or BS2p). But how do we take advantage of all those program banks? Well, there are a lot of ways, really. In this issue I'll show you a strategy that has worked for me an that you can apply to your own projects.

Plan Your Work, Work Your Plan

Yeah, yeah, I know I harp on it a bit, but I sincerely believe that we get into trouble with our projects when we don't plan them. You know the saying: "We don't plan to fail, we fail to plan." I think that's particularly the case when we start to work across program banks with the BS2e, BS2sx or BS2p. Since talk is theoretical talk is cheap, let's dive into a project and learn by doing.

Figure 87.1: Multi-Bank Programming Circuit

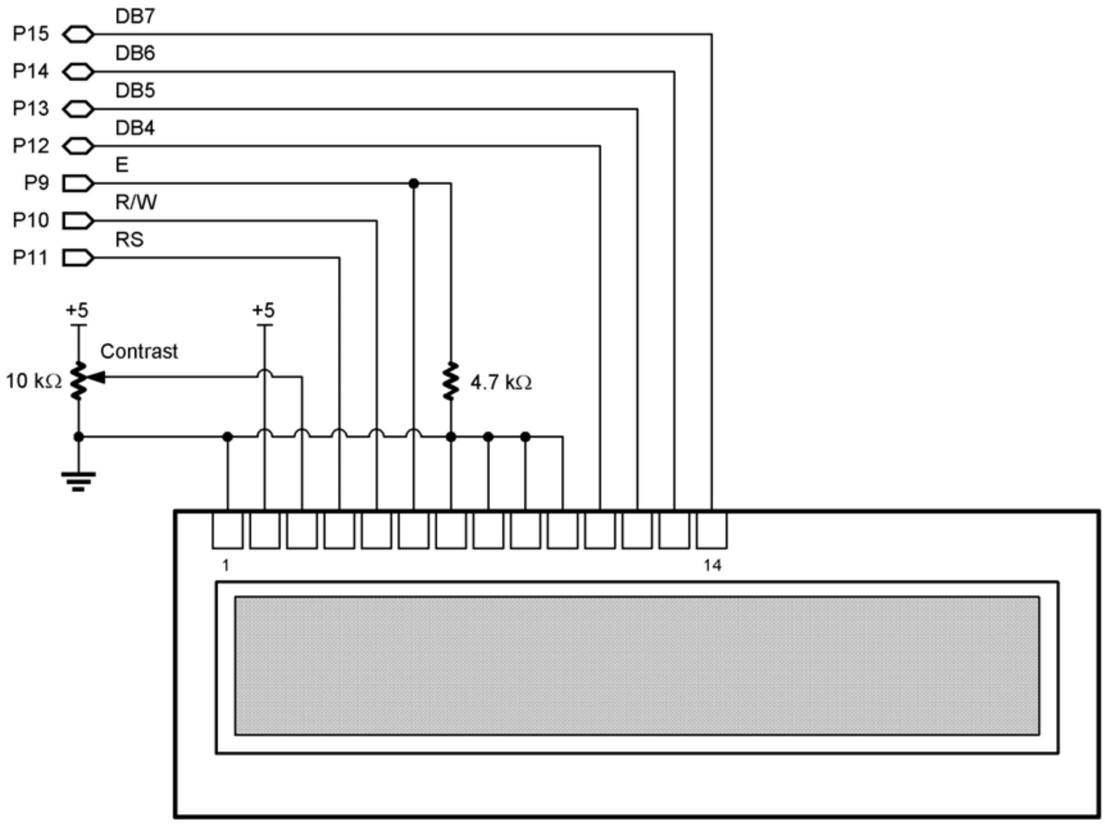
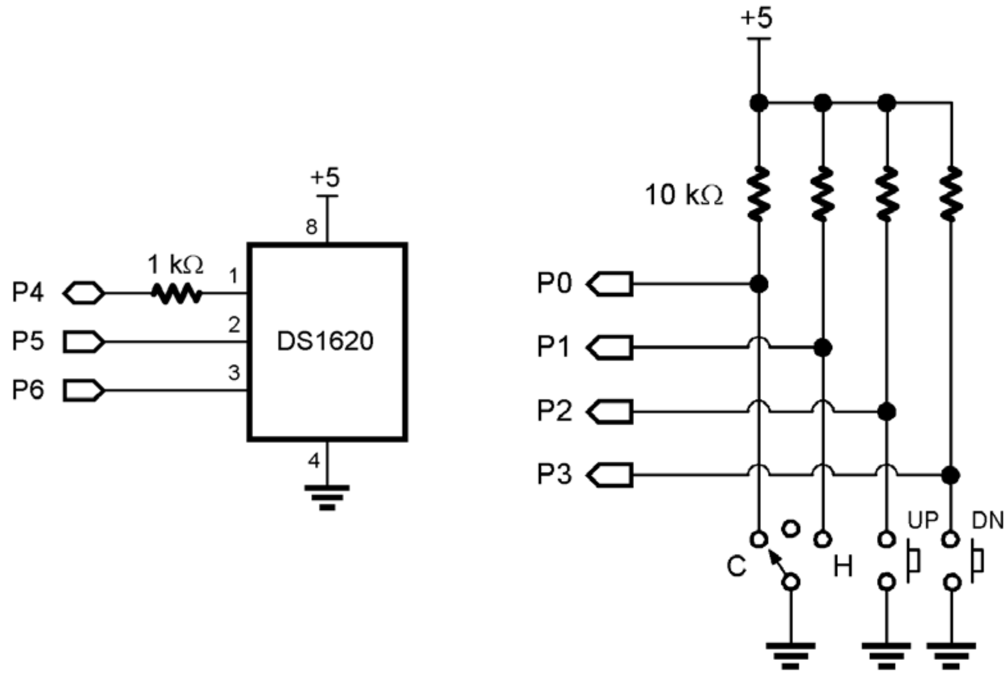


Figure 87.1: Multi-Bank Programming Circuit (continued)



Our project this month is a simple thermostat simulation. The goal is to manage the temperature and control code in one bank and the display output in another. Why? Well, this version will use a standard 2x16 LCD display. But what if, two months from now, we decide we want to use one of Scott Edwards' nifty graphics LCDs instead? By keeping the display code in a separate module, we don't have to tear-up the control code module to use it.

In the BS2e, BS2sx and BS2p there are three keywords that apply to the use of multiple program banks: PUT, GET and RUN. PUT will write a byte variable to a specific location in a shared RAM space called the Scratchpad. GET will retrieve a byte. RUN will execute the target program bank.

What were going to do is use the Scratchpad as mechanism to store program variables and to pass commands and data between program banks. Here's where some of the planning comes into play. Program design will also play a big role in making all of this work easily.

I've long advocated the use of a "task manager" approach to writing PBASIC programs. I like this style because it allows programs to become very flexible without overusing GOTO. In this case, it really helps because we can save our current task to the Scratchpad, go run code in another module, then come back and retrieve the task to run. It'll probably make more sense as we get into the code.

Let's define our program: The main module will monitor a temperature sensor (DS1620), a mode switch (Off, Cool, Heat) and a couple of buttons (Up and Down) to change the current setpoint. The external module will initialize the display device, clear the display device, show the temperature, the setpoint, the thermostat mode and whether or not the fan is running. What we'll find is that the main module will be completely unaware of the mechanics of displaying data – it will simply pass the command and/or data and rely on the external code to handle it. This aspect of the program design will let us change the display device and code later without affecting our main module.

Based on what we have so far, here's how we'll use the Scratchpad:

0	Bank 0 task
1	Bank 1 task (command)
2	thermostat mode (plus fan status)
3	temperature (low byte)
4	temperature (high byte)
5	setpoint (low byte)
6	setpoint (high byte)

As you can see, the start of our data "package" for the external module starts at address 2. We'll actually define this value as a constant so we can shift the package around if necessary to accommodate the use of more than one external module.

Cool It, Buddy

Okay, it's time to write some code. As you can see by the schematics, we're working with simple parts that we've all dealt with a thousand times (if you're new, don't worry, there's plenty of documentation available to explain how these parts work). As I pointed out earlier, we'll use a task manager approach to our design so we can save what we're doing when we access an external module. For the main program, we'll need to do the following tasks:

0	Initialize the display (external code)
1	Initialize the DS1620
2	Read the temperature
3	Get the setpoint
4	Update the display (external code)

Tasks 0 and 1 will only have to run once – the others will repeat through the run of the program. Now, you may be wondering why we don't define scanning the mode switch and buttons as a task. The reason is that we want this to happen all the time, so our design will allow us to do that between every iteration of tasks 2, 3 and 4.

Take a look at the Initialization section in Proram Listing 87.1. You'll notice that the first thing we do is read the Scratchpad for our current task and the stored setpoint. On power-up or reset, these values will be zero so the BRANCH command that follows will take us to Init_Screen. This section of code prepares us to launch the [external] code that initializes our display device (LCD). What we have to do before running the external module is save what we want to do when we get back. In this case, we'll want to initialize the DS1620 (task value of 1). In Scratchpad address 1 we'll tell the external module what to do. Then we run the external module. So let's go there.

Jump over to Program Listing 87.2. What you'll see is that this module simply holds a group of subroutines that deal with the display: initialize, clear and update. The routine to run is passed via the Scratchpad in location 1.

Our first task is to initialize the display. This is pretty common code as we're using a standard 2x16 LCD for this program. What you'll notice is that the end of the initialization section is allowed to drop through to the code that clears the display. This is necessary in case of a reset when the program has been running. Re-initializing the display does not automatically clear it. Once the display is cleared, the program exits back to the main code module (Program Listing 87.1).

Now when we return to the main module, the program starts all over again. This is why we save the current task and the setpoint in the Scratchpad – they will probably get destroyed because of the different variable definitions in the other program bank. This time through our task value is one, so the program will BRANCH to the [internal] code that initializes the DS1620. Again, this is code we've used before. It sets up the DS1620 to "free-run" and be accessed by an external CPU. When this is complete, we update our task variable and initialize the setpoint to a default value.

Now we're in the heart of the main control program. At the top is where we scan our mode switch and Up/Down buttons for the setpoint. This little loop of code is useful for debouncing multiple inputs. The tilde (~) operator inverts our active-low inputs to "1" when pressed or on to make the inputs easier to deal with in code. Once done, the mode value is isolated so we can pass it to the external module. The modulus operator (//) keeps the mode value in the range of 0 (off), 1 (cool) and 2 (heat).

The first [repeating] task is to get the current temperature and compare it to the setpoint. This code calls an internal subroutine to read the DS1620 and to convert its output (half degrees Celsius) to whole degrees Fahrenheit. The returned value is compared to the setpoint and, based on the current control mode, the fan control bit is set or cleared.

The end of this code updates the task variable and goes back to the top where we scan the inputs again then BRANCH to checking for a setpoint change. This is actually very simple code and demonstrates the usefulness of aliasing variables. If you look at the variables section, you'll see that the Up and Down bits have been aliased from the btnIns variable. As bits, these variables will have values or 0 (not pressed) or 1 (pressed).

The entry portion of this code actually looks to see if both buttons are being pressed at the same time. If not, it jumps to code that handles a possible setpoint change. If both buttons are pressed, the setpoint is reset to the default value. Most of the time, though, only one button will be pressed.

Let's say, for example, that our current setpoint is lower than the specified maximum. In this case, the value of the Up button will be added to the current setpoint. If pressed, this value will be one. If not, the value will be zero. The nice thing is that we don't have to use an IF-THEN construct to check if the button was pressed or not, we simply add the current button value. Pretty neat. But what if you wanted to increment or decrement by a different value, say five? No problem. Just change the code so it looks like this:

```
setpoint = setpoint + (btnUp * 5)
```

The same approach is used to check the down button and decrease the setpoint if it's pressed.

Now that we have the current temperature and setpoint, it's time to update the LCD. The task that handles this actually sets up everything so that it can run externally. In this task we'll store what we want to do when we get back, what external routine to run (display update) and the values used by the external code.

Notice that the fan control bit is added into the mode value and passed that way. Since the temperature and setpoint are stored as words, we have to use PUT twice to pass the value. This is required because PUT and GET only work with bytes. The technique of storing low-byte first is often referred to as "Little Endian" and is common practice.

Now we want to update the display, so let's jump back over to Program Listing 87.2. At this point, the command passed will cause the program to BRANCH to Update_LCD. Since this routine uses data passed from the main module, the first thing it has to do is use GET to retrieve the data from the Scratchpad.

With the data in hand, the temperature and setpoint values are printed using a subroutine called Print_Temperature. This code prints a three-digit, right justified (space padded) value. It assumes the value to be positive, so if you want to deal with negative values this code will have to be updated. It's not tough to do. Simply look at bit 15 of the tPrint value. If it's a one, the value is negative. In this case, you would print a "-" then use the ABS function to get the positive temperature value and print using the code as shown.

The next thing to do is print the current thermostat mode. The various mode strings are stored in DATA statements. LOOKUP is used to locate the first character of a string and a simple loop writes the characters to the LCD. The strings are terminated with zero so that the print loop knows when to stop. Also note that the strings are also padded with a leading space that will erase the fan running indicator when we change the mode.

The final step, then, is to display the fan status. In this demo, I took the lead from my own home thermostat that prints an asterisk when the fan is running. Once the fan status is displayed (or not), the program exits back to the control program and the process starts over again at reading the temperature.

That wasn't too tough, was it? Of course, we could have easily fit both this programs into one bank, but then updating the display portion would lead to us potentially damaging the control code. By using the external module to deal with the display, we free up variable and code space for control code and can change display types without worry.

Saving Everything ... Almost Everything

I am not a fan of -- and I actually discourage -- the use of internal variable names (like B0, W1, etc.), but there is a case here where it can be useful. Let's say, for example, that you need to save and retrieve a lot of variables when dealing with an external program module. Here's bit of code that will save everything to the Scratchpad except one byte:

```
Push_Vars:
  FOR B25 = 0 TO 24
    PUT (BankVarsStart + B25), B0(B25)
  NEXT
  RETURN
```

This routine uses B25 (last allocated byte in the variable RAM space) as a loop counter and takes advantage of the fact that the BASIC Stamp treats the variable RAM space as an array. So B0(0) is the first byte of variable RAM and B0(24) is the penultimate byte. The constant called BankVarsStart determines where the data is saved in the Scratchpad (be careful not to make it so high as to overrun the end of the Scratchpad). The only thing that doesn't get saved is B25 since it's used as the loop control. Of course, if things get really desperate, you could use 26 PUT statements to save the data. But that's not likely to be the case since the use of an external for subroutines generally frees up some variable space.

Retrieving data is just as easy:

```
Pop_Vars:
  FOR B25 = 0 TO 24
    GET (BankVarsStart + B25), B0(B25)
  NEXT
  RETURN
```

Go For It!

Okay, now that you've seen how easy using multiple program banks can be, it's time for you to use this technique in your own programs. It only takes a little bit of planning to organize the use the Scratchpad and a task-manager approach to your code so that you can direct the flow across modules. Remember to plan your work and work your plan and you won't have any trouble.

For those of you that have either of the Scott Edwards graphics displays, a good first project would be to create a module that is compatible with the code we've built here. Could be a lot of fun....

```
' =========================================================================
'
'   Program Listing 87.1
'   File...... Thermo_Demo.BSE
'   Purpose... Multi-bank Program Demo
'   Author.... Jon Williams
'   E-mail.... jwilliams@parallaxinc.com
'   Started...
'   Updated... 02 JUN 2002
'
'   {$STAMP BS2e, Thermo_LCD.BSE}
'
' =========================================================================


' -------------------------------------------------------------------------
' Program Description
' -------------------------------------------------------------------------
'
' The pupose of these programs is to demonstrate the multi-bank capability of
' the BS2e, BS2sx and BS2p.  The core program monitors a DS1620 and functions
' as a simple thermostat control.  Information from the program is displayed
' on an LCD that is controlled from a different program bank.
'
' Tasks:
'
'  0    Initialize LCD (code in bank 1)
'  1    Initialize DS1620
'  2    Read temperature
'  3    Get setpoint
'  4    Update LCD (code in bank 1)
'
' Tasks 0 and 1 run only once.


' -------------------------------------------------------------------------
' Revision History
' -------------------------------------------------------------------------


' -------------------------------------------------------------------------
' I/O Definitions
' -------------------------------------------------------------------------

Inputs          VAR     InA                     ' mode and temp change inputs
DQ              CON     4                       ' DS1620.1 (data I/O)
Clock           CON     5                       ' DS1620.2
Reset           CON     6                       ' DS1620.3
```

```
' -----------------------------------------------------------------------------
' Constants
' -----------------------------------------------------------------------------

RdTmp           CON     $AA                     ' read temperature
WrHi            CON     $01                     ' write TH (high temp)
WrLo            CON     $02                     ' write TL (low temp)
RdHi            CON     $A1                     ' read TH
RdLo            CON     $A2                     ' read TL
StartC          CON     $EE                     ' start conversion
StopC           CON     $22                     ' stop conversion
WrCfg           CON     $0C                     ' write config register
RdCfg           CON     $AC                     ' read config register

TskInitScr      CON     0                       ' program tasks
TskInitTmp      CON     1
TskTemp         CON     2
TskSetPoint     CON     3
TskScreen       CON     4

ScreenBank      CON     1                       ' bank that holds output code

ScrInit         CON     0                       ' initialize screen
ScrClear        CON     1                       ' clear screen
ScrUpdate       CON     2                       ' update screen

AcOff           CON     0                       ' A/C modes
AcCool          CON     1
AcHeat          CON     2

MinTemp         CON     0                       ' valid temp range
MaxTemp         CON     125
DefaultSP       CON     75                      ' default setpoint

Yes             CON     1
No              CON     0

DataStart       CON     2                       ' data block starts at loc 2


' -----------------------------------------------------------------------------
' Variables
' -----------------------------------------------------------------------------

task            VAR     Nib                     ' current task
loop            VAR     Nib                     ' loop counter
btnIns          VAR     Nib                     ' switch and button inputs
btnUp           VAR     btnIns.Bit2
btnDn           VAR     btnIns.Bit3
mode            VAR     Nib
fanCtrl         VAR     mode.Bit3               ' 1 = run fan
```

```
fan             VAR     bit
setpoint        VAR     Word                    ' temperature setpoint
tempIn          VAR     Word                    ' raw temp from DS1620
sign            VAR     tempIn.Bit8             ' 1 = negative temperature
tSign           VAR     Bit
tempC           VAR     Word
tempF           VAR     Word


' -------------------------------------------------------------------------------
' EEPROM Data
' -------------------------------------------------------------------------------



' -------------------------------------------------------------------------------
' Initialization
' -------------------------------------------------------------------------------

Initialize:
  GET 0, task                                   ' get current task
  GET (DataStart + 3), setpoint.LowByte         ' get last setpoint
  GET (DataStart + 4), setpoint.HighByte

  BRANCH task, [Init_Screen, Init_DS1620, Main, Main, Main]

Init_Screen:
  PUT 0, TskInitTmp                             ' store task for retrun
  PUT 1, ScrInit                                ' store task for external code
  RUN ScreenBank                                ' run external code

Init_DS1620:
  HIGH Reset                                    ' alert the DS1620
  SHIFTOUT DQ, Clock, LSBFirst, [WrCfg, %10]    ' use with CPU; free-run
  LOW Reset
  PAUSE 10
  HIGH Reset
  SHIFTOUT DQ, Clock, LSBFirst, [StartC]        ' start conversions
  LOW Reset

  task = TskTemp
  setpoint = DefaultSP


' -------------------------------------------------------------------------------
' Program Code
' -------------------------------------------------------------------------------

Main:
  btnIns = %1111                                ' enable all four inputs
  FOR loop = 1 TO 10
    btnIns = btnIns & ~Inputs                   ' test inputs
```

```
    PAUSE 5                                     ' delay between tests
  NEXT

  mode = (btnIns & %0011) // 3                  ' isolate mode switch bits

Task_Manager:
  BRANCH (task - 2), [Get_Temperature, Get_SetPoint, Update_Screen]
  GOTO Main


Get_Temperature:
  GOSUB Read_DS1620                             ' read current temperature
  fan = No                                      ' assume fan is off
  BRANCH mode, [Get_TempX, Check_Cool, Check_Heat]

Check_Cool:                                     ' check for cooling on
  IF (tempF <= setpoint) THEN Get_TempX
  fan = Yes
  GOTO Get_TempX

Check_Heat:                                     ' check for heating on
  IF (tempF >= setpoint) THEN Get_TempX
  fan = Yes

Get_TempX:
  task = TskSetPoint
  GOTO Main


Get_SetPoint:                                   ' check for both pressed
  IF ((btnIns >> 2) <> %11) THEN Check_Increase
  setpoint = DefaultSP
  GOTO SP_Done

Check_Increase:
  IF (setpoint = MaxTemp) THEN Check_Decrease
  setpoint = setpoint + btnUp

Check_Decrease:
  IF (setpoint = MinTemp) THEN SP_Done
  setpoint = setpoint - btnDn

SP_Done:
  PAUSE 100                                     ' delay between keys
  task = TskScreen
  GOTO Main


Update_Screen:
  PUT 0, TskTemp                                ' save next task
  PUT 1, ScrUpdate                              ' store task for external code
```

```
  fanCtrl = fan                                 ' pass fan control in mode
  PUT (DataStart + 0), mode                     ' store data packet
  PUT (DataStart + 1), tempF.LowByte
  PUT (DataStart + 2), tempF.HighByte
  PUT (DataStart + 3), setpoint.LowByte
  PUT (DataStart + 4), setpoint.HighByte
  RUN ScreenBank                                ' run external code


' -------------------------------------------------------------------------------
' Subroutines
' -------------------------------------------------------------------------------

Read_DS1620:
  HIGH Reset                                    ' alert the DS1620
  SHIFTOUT DQ, Clock, LSBFIRST, [RdTmp]         ' give command to read temp
  SHIFTIN DQ, Clock, LSBPRE, [tempIn\9]         ' read it in
  LOW Reset                                     ' release the DS1620

  tSign = sign                                  ' save sign bit
  tempIn = tempIn / 2                           ' round to whole degrees
  IF (tSign = 0) THEN No_Neg1
  tempIn = tempIn | $FF00                       ' extend sign bits for negative

No_Neg1:
  tempC = tempIn                                ' save Celsius value
  tempIn = tempIn */ $01CC                      ' multiply by 1.8
  IF (tSign = 0) THEN No_Neg2                   ' if negative, extend sign bits
  tempIn = tempIn | $FF00

No_Neg2:
  tempIn = tempIn + 32                          ' finish C -> F conversion
  tempF = tempIn                                ' save Fahrenheit value
  RETURN
```

```
' =========================================================================
'   Program Listing 87.2
'   File...... Thermo_LCD.BSE
'   Purpose... LCD output for THERMO DEMO.BSE
'   Author.... Jon Williams
'   E-mail.... jwilliams@parallaxinc.com
'   Started...
'   Updated... 02 JUN 2002
'
'   {$STAMP BS2e}
'
' =========================================================================


' -------------------------------------------------------------------------
' Program Description
' -------------------------------------------------------------------------

' This module provides LCD output for the THEMO DEMO program.  The main program
' will pass a task value using Scratchpad RAM location 1.
'
' Task Values:
'
' 0     Initialize LCD
' 1     Clear LCD
' 2     Update LCD
'
' For task 2, the following values are passed via the Scratchpad
'
' mode (off, cool, heat, cool-running, heat-running)
' temp.LowByte
' temp.HighByte
' setpoint.LowByte
' setpoint.HighByte


' -------------------------------------------------------------------------
' Revision History
' -------------------------------------------------------------------------


' -------------------------------------------------------------------------
' I/O Definitions
' -------------------------------------------------------------------------

E               CON     9                       ' LCD Enable pin  (1 = enabled)
RW              CON     10                      ' LCD read/write (0 = write)
RS              CON     11                      ' Register Select (1 = char)
LcdBus          VAR     OutD                    ' 4-bit LCD data bus
LcdBusDirs      VAR     DirD
```

```
' -----------------------------------------------------------------------------
' Constants
' -----------------------------------------------------------------------------

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
Line1           CON     $80                     ' DDRAM address of line 1
Line2           CON     $C0                     ' DDRAM address of line 2

LcdInit         CON     0                       ' initialize screen
LcdClear        CON     1                       ' clear screen
LcdUpdate       CON     2                       ' update screen

Yes             CON     1
No              CON     0

PgmBank         CON     0                       ' main program in bank 0
DataStart       CON     2                       ' data block starts at loc 2


' -----------------------------------------------------------------------------
' Variables
' -----------------------------------------------------------------------------

task            VAR     Nib
mode            VAR     Nib                     ' A/C control mode
running         VAR     mode.Bit3
temp            VAR     Word                    ' current temperature
setpoint        VAR     Word                    ' A/C setpoint
tPrint          VAR     Word                    ' temp to print
char            VAR     Byte                    ' character sent to LCD
index           VAR     Byte                    ' loop counter
eeAddr          VAR     Byte                    ' address of string char


' -----------------------------------------------------------------------------
' EEPROM Data
' -----------------------------------------------------------------------------

Msg_Off         DATA    "  OFF", 0
Msg_Cool        DATA    " COOL", 0
Msg_Heat        DATA    " HEAT", 0
```

```
' -------------------------------------------------------------------------------
' Initialization
' -------------------------------------------------------------------------------

Initialize:
  GET 1, task
  BRANCH task, [Init_LCD, Clear_LCD, Update_LCD]


' -------------------------------------------------------------------------------
' Program Code
' -------------------------------------------------------------------------------

Init_LCD:
  LOW E                                         ' initialize LCD pins
  LOW RW
  LOW RS
  LcdBusDirs = %1111                            ' make bus lines outputs

  PAUSE 500                                     ' let the LCD settle
  LCDbus = %0011                                ' 8-bit mode
  PULSOUT E, 1
  PAUSE 5
  PULSOUT E, 1
  PULSOUT E, 1
  LCDbus = %0010                                ' 4-bit mode
  PULSOUT E, 1
  char = %00101000                              ' multi-line mode
  GOSUB LCD_Command
  char = %00001100                              ' disp on, crsr off, blink off
  GOSUB LCD_Command
  char = %00000110                              ' inc crsr, no disp shift
  GOSUB LCD_Command


Clear_LCD:
  char = ClrLCD
  GOSUB LCD_Command
  GOTO Exit


Update_LCD:
  GET (DataStart + 0), mode                     ' retrieve data packet
  GET (DataStart + 1), temp.LowByte
  GET (DataStart + 2), temp.HighByte
  GET (DataStart + 3), setpoint.LowByte
  GET (DataStart + 4), setpoint.HighByte

  char = Line1 + 0                              ' print temperature
  GOSUB LCD_Command
  tPrint = temp
```

```
  GOSUB Print_Temperarature

  char = Line1 + 4                              ' print (setpoint)
  GOSUB LCD_Command
  char = "("
  GOSUB LCD_Write
  tPrint = setpoint
  GOSUB Print_Temperarature
  char = ")"
  GOSUB LCD_Write


Show_Mode:
  char = Line2 + 11                             ' show system mode
  GOSUB LCD_Command
  LOOKUP (mode & %0011), [Msg_Off, Msg_Cool, Msg_Heat], eeAddr

Print_Char:
  READ eeAddr, char
  IF (char = 0) THEN Show_Fan
  GOSUB LCD_Write
  eeAddr = eeAddr + 1
  GOTO Print_Char

Show_Fan:
  IF (running = No) THEN Exit
  char = Line2 + 11                             ' show fan status
  GOSUB LCD_Command
  char = "*"                                    ' show on
  GOSUB LCD_Write


Exit:
  RUN PgmBank


' -------------------------------------------------------------------------------
' Subroutines
' -------------------------------------------------------------------------------

Print_Temperarature:                            ' prints 3-digit, space padded
  char = " "                                    ' clear old digit
  GOSUB LCD_Write
  IF (tPrint < 100) THEN Print_T10
  char = CrsrLf
  GOSUB LCD_Command
  char = "0" + (tPrint DIG 2)                   ' convert 100's digit to ASCII
  GOSUB LCD_Write

Print_T10:
  char = " "
```

```
  GOSUB LCD_Write
  IF (tPrint < 10) THEN Print_T01
  char = CrsrLf
  GOSUB LCD_Command
  char = "0" + (tPrint DIG 1)                       ' convert 10's digit to ASCII
  GOSUB LCD_Write

Print_T01:
  char = "0" + (tPrint DIG 0)                       ' convert 1's digit to ASCII
  GOSUB LCD_Write
  RETURN


LCD_Command:
  LOW RS                                            ' enter command mode

LCD_Write:
  LCDbus = char.HighNib                             ' output high nibble
  PULSOUT E, 1                                      ' strobe the Enable line
  LCDbus = char.LowNib                              ' output low nibble
  PULSOUT E, 1
  HIGH RS                                           ' return to character mode
  RETURN
```