Column #62, June 2000 by Jon Williams:

Menus Made Easy

When it comes right down to it, I'm a very lucky guy. Really. I have a wonderful family, terrific friends, I live in one of the best cities in the world and I get to work with some really bright people. Like my friend, Will, for example. Now this guy is definitely one of the sharpest knives in the drawer. I love working with him; he inspires me on a daily basis.

Will and I work for a company that manufactures water-pumping stations for golf courses. Our big stations use off-the-shelf PLCs for control. The price of the PLC is easy to justify due to the sophistication of control required and the volume of stations we sell. But now that we're moving into the simpler municipal water market, the PLC is just a bit expensive.

That's no longer a problem for us – thanks to Will. He spent the last year designing a custom pumping station controller from the ground up. It's a real beauty and has been a big hit, inside the company and out. A very big reason, I believe, is the elegance and simplicity of its user interface.

I'll admit that I'm biased here. When it comes to user interfaces, I have been justifiably called anal-retentive. It's a fair criticism – I'm a nut when it comes to UI design. I'm a very big believer in UI standards, even if they're only loosely defined. Nothing throws me off about a piece of software more than a poorly designed or non-standard interface.

When it comes to the PC – especially in our "Windowed" world – designing to standard is pretty easy since there are a lot of good examples. There's even a set of written guidelines, called the CUA. But what do we do when it comes to industrial controllers?

I'm not suggesting that all industrial controls should have a common interface. What I am suggesting is that a simple and intuitive interface can be developed and applied to our Stamp projects. That's the goal here: apply Will's great UI design to the BASIC Stamp, creating a platform from which we can develop any number distinct control projects. And just as we're able to navigate any properly designed Windows program, we should be able to easily navigate any of our control projects that follow the standard we develop here.

Keeping It Simple

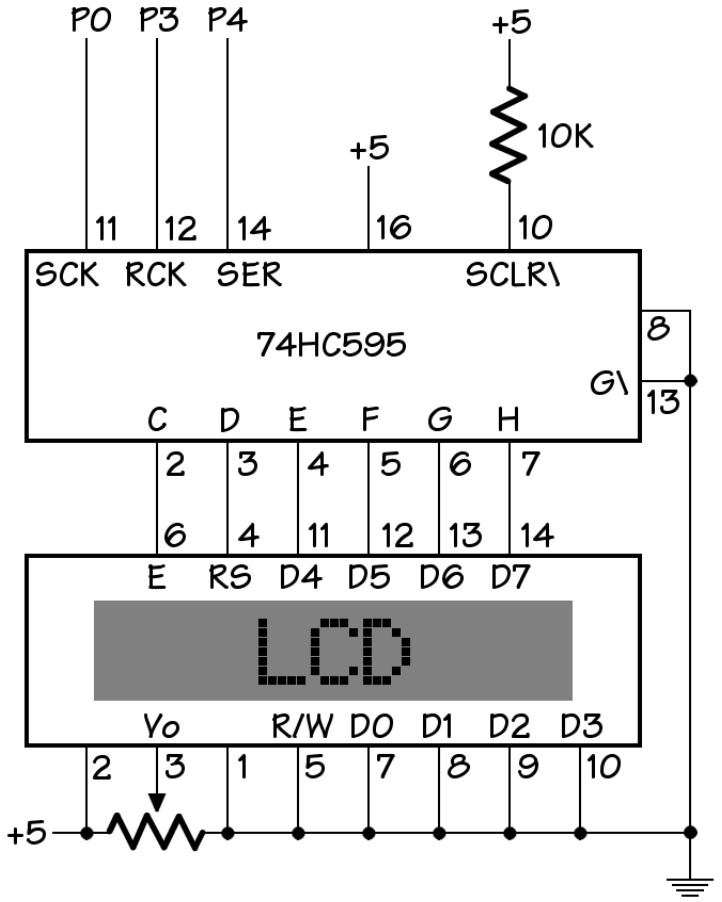
Yep, back to the KISS principal – keep it simple, silly. The user interface on Will's controller uses six buttons and a two-line LCD. With this simple interface, he created a multi-level menu system that is intuitive and easy to navigate (our design goal). So how do we duplicate that on a Stamp?

Using a conventional approach, connecting to six buttons would take six lines and connecting to the LCD (assuming 4-bit mode) would take another six; twelve total lines. Yikes – that doesn't leave much left to connect to the outside world. There's got to be another way.

And there is. Using SHIFTIN and SHIFTOUT, we can add a couple of fifty-cent shift registers to our project and reduce the I/O lines required for the interface to five. That's much better. The schematics for our demo project are shown in Figures 1 (LCD) and 2 (buttons).

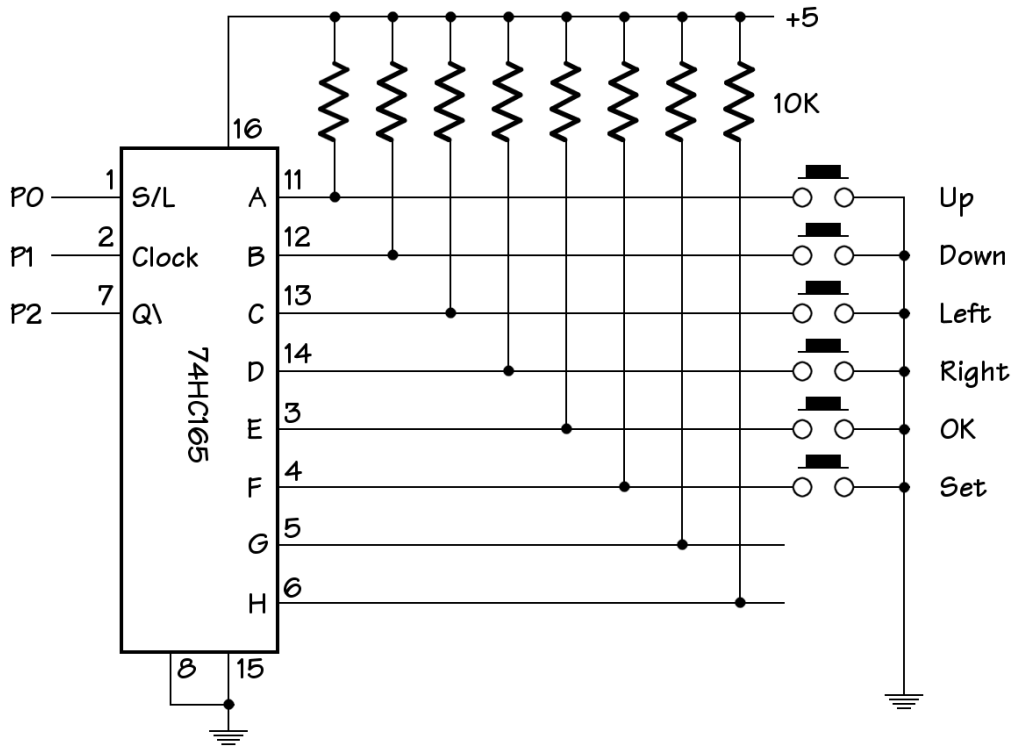
Since I've covered the use of the 74HC595 with LCDs in past articles, I'm not going to deal with it here except to say that with a little planning, you can easily cascade the additional 75HC595s to create more outputs. You'll need to connect the serial output (pin 9) of one 74HC595 to the serial input (pin 14) of the next. The Clock (pin 11) and Latch (pin 12) lines need to be tied together.

Figure 62.1: 74HC595 and LCD hookup



We'll use the 74HC595's compliment, the 74CH165 parallel-in/serial-out shift register to read our buttons. Since we're only using six inputs, the other two could be used as configuration switches, additional buttons, anything the project requires. And like the 74HC595, the 74HC165 can be cascaded if we ever need additional inputs.

Figure 62.2: 74HC165 and button hookups



Keyboard Debouncing

Debouncing one input with a Stamp is pretty easy with the BUTTON command, but what happens when we want to debounce six inputs and do it simultaneously? As it turns out, the solution is not particularly difficult and takes very little code. Take a look at Listing 1, down in the subroutines section. Look for the routine called GetKey.

GetKey returns inputs that have been held stable for about 25 milliseconds. That should be enough time to validate the button press and we can easily adjust the debounce timing if required. Here's how GetKey works: On entry to the routine we assume that all the buttons are pressed (this may seem odd, but will make sense in just a second). Then we scan the inputs and logically AND them with the current value. If a button has released due to contact-bounce, it will have a bit-value of zero. Zero ANDed with one is zero and will remain at zero through the remainder of the routine. Only a button that stays down

(bit value of 1) during the entire loop will return as a valid input. This technique can be used with nibbles, bytes or words – up to 16 inputs can be simultaneously debounced.

Sharp readers (that's all of you) are probably asking, "Wait, Jon, how can the inputs return a value of one when pressed if we've connect the buttons between the shift register inputs and ground?" Good catch. Look again at the schematic in Figure 62.2. We're using the inverted serial output from the 74HC165 to restore the positive logic for us. If we ever want to modify GetKey to deal with direct inputs, we would change the test line to look like this:

```
key = key & ~tempB
```

The tilde (~) in front of *tempB* inverts the bits for us.

In this program GetKey uses the SHIFTIN function to retrieve the buttons from the 75HC165. Before we can use SHIFTIN, however, we have to pulse the Shift/Load line from high-to-low, then back to high. This action "grabs" the buttons and holds them while we do the shifting. If any of the inputs change while we're shifting the data, we won't see it until the next scan.

Menu, Please

Last month we talked about project planning and that certainly applies here. In addition to any control functionality, we need to define our menu structure so that it makes sense to the user and is easy to navigate.

The goal of our demo program is to allow the user to set the time and day. To that end, we've set up three operational modes: display current time and day (mode 0), set time (mode 1) and set day (mode 2). Since setting the time is easier to do by individually setting the hours and the minutes, the set time mode has two levels. Note that zero is always used to indicate the topmost element in either structure. Mode 0, then, is our "normal" operational display. A level of zero indicates a menu display only. Once we get into actual value editing, we indicate the element to change with a non-zero level value.

Both *mode* and *level* are defined as nibbles, allow up to 15 menu items (beyond the normal display) and 15 levels within each menu. Our program is much simpler than that. Here's how the menu for our demo program is mapped:

mode *level*

0 : display time and day

1 : SET TIME
    1 : set hours
    2 : set minutes

2 : SET DAY
    1 : set day of week

Navigation Rules

With our menu structure in place, we need to define the rules by which we'll navigate through it. As we stated earlier, there are six buttons. Here's how they'll work:

Set	Enter menus or editing within a menu
OK	Move up one level
Up	Previous menu item or increment value
Down	Next menu item or decrement value
Right	Move to next field
Left	Move to previous field

As you can see, things get started by pressing the "Set" button. This will take us from our normal ("run") display into the menus. We will use "Up" and "Down" to select a specific menu item. With the desired menu item displayed, we'll press "Set" again. This will put us into edit mode (level one for the selected menu). We can change a value in edit mode by using "Up" and "Down." If there are multiple fields to edit for the selected menu item, we can move through the fields by pressing "Left" and "Right." Pressing "OK" in edit mode takes us back up to the menu so that we can select another. Finally, pressing "OK" while in the menu takes us back to the "run" display.

Putting It All Together

Okay, we know what we want to do and how our program should behave, so let's put it all together. We'll start, as always, by defining CONstants that will help make the code self-documenting. We use quite a few in this program and they really do help.

Operationally, we kick off the program by initializing the LCD and program variables. Since we only have eight bits available from the 75HC595, we'll use the 4-bit interface to the LCD as this mode requires only six lines. This LCD is also initialized to use multiple lines. The same initialization sequence will work with two- or four-line LCDs.

Like our exercise timer last month, this program runs in a continuous loop. Each pass through the loop scans the buttons then BRANCHes to the handler for the current mode and level. It is within the menus or edit code that we will process any button inputs. Let's follow the program from startup though setting the time. Along the way we'll try every possible button press so that the program is understood.

The program loop starts by scanning the buttons and placing the result in a variable named *key*. With *level* set to zero, the program BRANCHes to the line labeled Run_Mode. Since the flag variable *updtLCD* was initialize to Yes (1), the code drops through the IF...THEN and prints the time and day on line one of the LCD. Keep in mind that this is just a demonstration program and that the time and date are not automatically updated.

You might wonder why we go through the trouble to keep track of when the LCD needs to be updated. The reason is two-fold: we can save a little time by not writing to the LCD when there are no changes to be displayed and we keep the display "clean" as constant updates to the LCD can cause an annoying flash.

We've simplified the program by printing the time and day from subroutines (PrintTime and PrintDay). These subroutines allow us to print at the current cursor position of the LCD. PrintTime calls a neat little routine called LCDdec2. This routine is similar to the DEC2 modifier for DEBUG or SEROUT. Look closely at the code. Just above is an entry point called LCDhex2. This works like the HEX2 modifier. Both LCDdec2 and LCDhex2 set the base value for the working section of code, LCDnum2. This bit of code will print a two-digit number at the current cursor position of the LCD. Notice that we don't actually calculate the character to print (as we typically do), but instead, we calculate the character's position in a EEPROM table. Then we read it from the EEPROM and print it. This is how the same code can be used to print decimal or hex numbers. In fact, by setting the variable *base* to eight, we could print a two-digit octal number as well.

PrintDay also takes advantage of data stored in the EEPROM, in this case, zero-terminated strings. By storing our strings in the EEPROM, we can easily make changes – even change the language of our displays should we ever decide to internationalize the

project. The routine that puts the string on the LCD is called LCDprint. What we have to do is set the variable *addr* to the first character of the string to print. LCDprint will loop through the EEPROM from that point, printing the characters it reads until it encounters a zero. So we have to make sure that we end our strings with zero, otherwise we'll end up with a corrupted display.

Okay, the time and date is displayed and the program is waiting for an input. The only button that does anything from the top level is "Set," so let's press it. When we do, the *mode* variable is set to MNU_Tm (1) and *level* is cleared to zero. Since we're going to change to a new display, we tell the program by setting *updtLCD* to Yes. We exit the current action by jumping to LoopPad250. This label finishes the loop and gives us a 250-millisecond delay – enough time to release the button. In other cases, we'll use a 100-millisecond loop delay by jumping to LoopPad100.

On our next pass through the main loop we will BRANCH to line labeled Mode_Time. As with Mode_Run, we will update the display and wait for a valid button. Again, we'll use the routine LCDprint to send a string ("SET TIME") to the display. Pressing "OK" at this level causes us to return to the top. This is achieved by setting the variable *state* to RunMode. If you look carefully through the variable definitions, you'll see that our variables *mode* and *level* are actually aliased elements of *state*. Setting *state* to zero (RunMode) clears *mode* and *level* at the same time.

Let's return to the "SET TIME" menu and then press "Set." This causes us to enter the editing mode by setting *level* to SET_Hr (1). On the next pass through the program loop, we will end up at the label called Time_Hours. This bit of code will put the current time on line two and place a visible cursor under the hours value.

In hours editing mode, more buttons are used. Pressing "OK" clears *level* to zero and returns us to the menu where we can make another selection. We can change the hours value by pressing either "Up" or "Down." Both routines keep the hours value within range by using the modulus (//) operator. I find this technique easier (less code) and more user-friendly for interfaces like we're designing. Pressing "Up" or "Down" necessitates a display change so we'll set *updtLCD* to Yes.

With the hours set, we move to the minutes field by pressing the "Right" button. This causes *level* to be set to SET_Min (2), forcing the program to move to the minutes editing routines. As before, we indicate that we're editing by placing the cursor under the minutes value. Button processing is identical to setting the hours. Once we're satisfied, pressing "OK" twice will return us to our topmost display.

And we're done. The "run" display will now show the new time. Setting the day works the same, but only requires one edit level. In an operational program we would use our new to update a real-time-clock.

Wrap Up

Another one of those sharp guys I know in Dallas is Roger Arrick, the owner of Arrick Robotics (check out Roger's Stamp-controlled ARobot at www. robotics.com). Roger's e-mail tag line is, "It's Harder Than It Looks." That was the case with this menu system. Now, I don't want you to be put off by this, I'm just warning you to take your time with your menu design and program development, lest your project take off to la-la land. It is a bit of work and yet, I think you'll agree – and your customer's will agree -- that the result is well worth the effort. Happy Stamping.

```
' Nuts & Volts "Stamp Applications" - June 2000
' Program Listing 62.1

'=====================================================================
' Program... STAMPUI.BS2
' Purpose... Stamp User-Interface for general control applications
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
'=====================================================================


' ----[ Program Description ]-----------------------------------------
'
' This program demonstrates a multi-level menu system using a keypad input
' and LCD output.  Stamp pins are conserved by using shift registers for
' the keys and LCD.


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


' ----[ I/O Definitions ]---------------------------------------------
'
Clock   CON     0                           ' shared clock line
SL_165  CON     1                           ' shift/load of 74HC165
DI_165  CON     2                           ' serial data from 74HC165
L_595   CON     3                           ' 74HC595 output latch
DO_595  CON     4                           ' serial data to 74HC595


' ----[ Constants ]--------------------------------------------------
'
ClrLCD  CON     $01                         ' clear the LCD
CrsrHm  CON     $02                         ' move cursor home
CrsrLf  CON     $10                         ' move cursor left
CrsrRt  CON     $14                         ' move cursor right
DispLf  CON     $18                         ' shift chars left
DispRt  CON     $1C                         ' shift chars right

Crsr1   CON     %00001110                   ' underline cursor on
Crsr0   CON     %00001100                   ' underline cursor off

DDRam   CON     $80                         ' Display Data RAM control
CGRam   CON     $40                         ' Char Gen RAM control

Line1   CON     $00                         ' line 1, column 0
Line2   CON     $40
Line3   CON     $14
Line4   CON     $54
```

```
Key_Up  CON     %000001                         ' input keys
Key_Dn  CON     %000010
Key_Lf  CON     %000100
Key_Rt  CON     %001000
Key_OK  CON     %010000
Key_Set CON     %100000

RunMode CON     0                               ' menu displays
MNU_Tm  CON     1
MNU_Day CON     2

SET_Hr  CON     1                               ' setting hours
SET_Min CON     2                               ' setting minutes
SET_Day CON     1                               ' setting day

D_Sun   CON     0                               ' days of week
D_Mon   CON     1
D_Tue   CON     2
D_Wed   CON     3
D_Thu   CON     4
D_Fri   CON     5
D_Sat   CON     6

Yes     CON     1
No      CON     0


' ----[ Variables ]------------------------------------------------------
'
key     VAR     Byte                            ' key input
char    VAR     Byte                            ' character out to LCD
temp    VAR     Byte                            ' work variable for LCD
lcd_E   VAR     temp.Bit2                       ' LCD Enable pin
lcd_RS  VAR      temp.Bit3                      ' Reg Select (1 = char)
addr    VAR     Byte                            ' EE address for LCDprint
base    VAR     Byte                            ' base for display

hrs     VAR     Byte                            ' hours
mins    VAR     Byte                            ' minutes
day     VAR     Nib                             ' day of week, 0 to 6

state   VAR     Byte                            ' program state
mode    VAR     state.HighNib                   ' menu mode
level   VAR     state.LowNib                    ' edit level

tempW   VAR     Word                            ' general purpose word
temp1   VAR     tempW.HighByte
temp2   VAR     tempW.LowByte
tempB   VAR     Byte                            ' general purpose byte
loop    VAR     Byte                            ' loop counter
```

```
flags   VAR     Nib
updtLCD VAR     flags.Bit0                      ' update LCD flag



' ----[ EEPROM Data ]-------------------------------------------------
'
Digits  DATA    "0123456789ABCDEF"              ' digits for LCDnum2 sub

Days    DATA    "SUN", 0                        ' day strings
        DATA    "MON", 0
        DATA    "TUE", 0
        DATA    "WED", 0
        DATA    "THU", 0
        DATA    "FRI", 0
        DATA    "SAT", 0

LCD_ST  DATA    "SET TIME", 0                   ' menu strings
LCD_SD  DATA    "SET DAY", 0


' ----[ Initialization ]-----------------------------------------------
'
' Initialize the LCD (Hitachi HD44780 controller)
'
LCDinit:
        PAUSE 500                               ' let the LCD settle
        char = %0011                            ' 8-bit mode
        GOSUB LCDcmd
        PAUSE 5
        GOSUB LCDcmd
        GOSUB LCDcmd
        char = %0010                            ' put in 4-bit mode
        GOSUB LCDcmd
        char = %00101000                        ' 2-line mode
        GOSUB LCDcmd
        char = %00001100                        ' disp on, crsr off
        GOSUB LCDcmd
        char = %00000110                        ' inc crsr, no disp shift
        GOSUB LCDcmd
        char = ClrLCD
        GOSUB LCDcmd


Initialize:
        updtLCD = Yes                           ' refresh the LCD
        state = RunMode                         ' top menu

        hrs = 12
        mins = 34
```

```
        day = D_Sun


' ----[ Main ]----------------------------------------------------------
'
Main:
        GOSUB GetKey
        BRANCH mode, [Mode_Run, Mode_Time, Mode_Day]
        GOTO LoopPad100

' =========================
'  Run Display (top level)
' =========================

Mode_Run:
        IF updtLCD = No THEN Mode_Run2       ' no update, check key
        char = Crsr0                         ' clear cursor frome edit
        GOSUB LCDcmd
        char = ClrLCD                        ' clear the LCD
        GOSUB LCDcmd
        GOSUB PrintTime                      ' print the time
        char = DDRam + Line1 + 6             ' move to position 6
        GOSUB LCDcmd
        GOSUB PrintDay                       ' print the day
        updtLCD = No                         ' LCD updated

Mode_Run2:
        IF key <> Key_Set THEN LoopPad100    ' "Set" not pressed
        mode = MNU_Tm                        ' "Set" pressed, Time menu
        level = 0                            ' menu level
        updtLCD = Yes                        ' update the LCD
        GOTO LoopPad250                      ' allow key release


' ==============
'  Time Display
' ==============

Mode_Time:
        ' branch to current mode level
        BRANCH level, [Time_Menu, Time_Hours, Time_Mins]
        GOTO LoopPad100

Time_Menu:                                   ' display "SET TIME"
        IF updtLCD = No THEN Time_Menu2      ' update on if required
        char = Crsr0
        GOSUB LCDcmd
        char = ClrLCD
        GOSUB LCDcmd
        addr = LCD_ST
        GOSUB LCDprint
```

```
        updtLCD = No

Time_Menu2:
        IF key <> Key_OK THEN Time_Menu2a    ' check "OK"
        state = RunMode                      ' - pressed; up to top
        updtLCD = Yes
        GOTO LoopPad100

Time_Menu2a:
        IF key <> Key_Set THEN Time_Menu2b   ' check "Set"
        level = SET_Hr                       ' - pressed; set hours
        updtLCD = Yes
        GOTO LoopPad250

Time_Menu2b:
        IF key <> Key_Dn THEN LoopPad100     ' check "Down"
        mode = MNU_Day                       '  - move to day menu
        updtLCD = Yes
        GOTO LoopPad250

Time_Hours:                                  ' display hours with cursor
        IF updtLCD = No THEN Time_Hours1     ' - if refresh required
        char = Crsr0                         ' no cursor during refresh
        GOSUB LCDcmd
        char = DDRam + Line2                 ' time on Line 2
        GOSUB LCDcmd
        GOSUB PrintTime
        char = DDRam + Line2 + 1             ' cursor under hours
        GOSUB LCDcmd
        char = Crsr1
        GOSUB LCDcmd
        updtLCD = No

Time_Hours1:
        IF key <> Key_OK THEN Time_Hours1a   ' check "OK"
        level = 0                            ' - back to menu
        updtLCD = Yes
        GOTO LoopPad250

Time_Hours1a:
        IF key <> Key_Up THEN Time_Hours1b   ' check "Up"
        hrs = hrs + 1 // 24                  ' - increment with rollover
        updtLCD = Yes
        GOTO LoopPad250

Time_Hours1b:
        IF key <> Key_Dn THEN Time_Hours1c   ' check "Down"
        hrs = hrs + 23 // 24                 ' - dec with rollunder
        updtLCD = Yes
        GOTO LoopPad250
```

```
Time_Hours1c:
        IF key <> Key_Rt THEN LoopPad100     ' check "Right"
        level = SET_Min                      ' - set minutes
        updtLCD = Yes
        GOTO LoopPad100

Time_Mins:                                   ' display mins with cursor
        IF updtLCD = No THEN Time_Mins1      ' - if refresh required
        char = Crsr0
        GOSUB LCDcmd
        char = DDRam + Line2
        GOSUB LCDcmd
        GOSUB PrintTime
        char = DDRam + Line2 + 4             ' cursor under minutes
        GOSUB LCDcmd
        char = Crsr1
        GOSUB LCDcmd
        updtLCD = No
        GOTO LoopPad100

Time_Mins1:
        IF key <> Key_OK THEN Time_Mins1a    ' check "OK"
        level = 0                            ' - back to menu
        updtLCD = Yes
        GOTO LoopPad100

Time_Mins1a:
        IF key <> Key_Up THEN Time_Mins1b    ' check "Up"
        mins = mins + 1 // 60                ' - inc with rollover
        updtLCD = Yes
        GOTO LoopPad100

Time_Mins1b:
        IF key <> Key_Dn THEN Time_Mins1c    ' check "Down"
        mins = mins + 59 // 60               ' - dec with rollunder
        updtLCD = Yes
        GOTO LoopPad100

Time_Mins1c:
        IF key <> Key_Lf THEN LoopPad100     ' check "Left"
        level = SET_Hr                       ' - set hours
        updtLCD = Yes
        GOTO LoopPad100


' =============
'  Day Display
' =============

Mode_Day:
        ' branch to current mode level
```

```
        BRANCH level, [Day_Menu, Day_Set]
        GOTO LoopPad100

Day_Menu:                                   ' display "SET DAY"
        IF updtLCD = No THEN Day_Menu2      ' - if refresh required
        char = Crsr0
        GOSUB LCDcmd
        char = ClrLCD
        GOSUB LCDcmd
        addr = LCD_SD
        GOSUB LCDprint
        updtLCD = No

Day_Menu2:
        IF key <> Key_OK THEN Day_Menu2a    ' check "OK"
        state = RunMode                     ' - back to top
        updtLCD = Yes
        GOTO LoopPad100

Day_Menu2a:
        IF key <> Key_Set THEN Day_Menu2b   ' check "Set"
        level = SET_Day                     ' - set day
        updtLCD = Yes
        GOTO LoopPad250

Day_Menu2b:
        IF key <> Key_Up THEN LoopPad100    ' check "Up"
        mode = MNU_Tm                       ' - back to time menu
        level = 0
        updtLCD = Yes
        GOTO LoopPad100

Day_Set:
        IF updtLCD = No THEN Day_Set1
        char = Crsr0
        GOSUB LCDcmd
        char = DDRam+ Line2
        GOSUB LCDcmd
        GOSUB PrintDay
        char = DDRam + Line2
        GOSUB LCDcmd
        char = Crsr1
        GOSUB LCDcmd
        updtLCD = No
        GOTO LoopPad100

Day_Set1:
        IF key <> Key_OK THEN Day_Set1a     ' check "OK"
        level = 0                           ' - back up to menu
        updtLCD = Yes
        GOTO LoopPad100
```

```
Day_Set1a:
       IF key <> Key_Up THEN Day_Set1b      ' check "Up"
       day = day + 1 // 7                   ' - inc with rollover
       updtLCD = Yes
       GOTO LoopPad250

Day_Set1b:
       IF key <> Key_Dn THEN LoopPad100     ' check "Down"
       day = day + 6 // 7                   ' - dec with rollunder
       updtLCD = Yes
       GOTO LoopPad250

' ==================
'  End of Main Loop
' ==================


LoopPad250:                                 ' 250 ms pad
       PAUSE 150

LoopPad100:                                 ' 100 ms pad
       PAUSE 100
       GOTO Main


' ----[ Subroutines ]------------------------------------------------------
'

' Send command to the LCD
'
LCDcmd:
       lcd_RS = 0                           ' command mode
       GOTO LCDout


' Write ASCII char to LCD
'
LCDputc:
       lcd_RS = 1                           ' character mode
       GOTO LCDout


' send char to LCD
'
LCDout:
       temp.HIGHNIB = char.HIGHNIB          ' get high nibble
       lcd_E = 1
       SHIFTOUT DO_595, Clock, MSBFIRST, [temp]
       PULSOUT L_595, 1
       lcd_E = 0                            ' drop Enable line low
```

```
        SHIFTOUT DO_595, Clock, MSBFIRST, [temp]
        PULSOUT L_595, 1
        temp.HIGHNIB = char.LOWNIB          ' get low nibble
        lcd_E = 1
        SHIFTOUT DO_595, Clock, MSBFIRST, [temp]
        PULSOUT L_595, 1
        lcd_E = 0
        SHIFTOUT DO_595, Clock, MSBFIRST, [temp]
        PULSOUT L_595, 1
        RETURN


' send EE string to LCD
' - string starts at addr and ends with zero
'
LCDprint:
        READ addr, char                     ' get character from EE
        IF char = 0 THEN LCDprintX          ' if 0, we're done
        GOSUB LCDputc                       ' write the character
        addr = addr + 1                     ' point to next character
        GOTO LCDprint

LCDprintX:
        RETURN


' print 2-digit number on LCD
'
LCDdec2:
        base = 10                           ' display number as decimal
        GOTO LCDnum2

LCDhex2:
        base = 16                           ' display number as hex

LCDnum2:
        READ Digits + (tempB / base), char  ' high digit
        GOSUB LCDputc
        READ Digits + (tempB // base), char ' low digit
        GOSUB LCDputc
        RETURN


GetKey:
        key = %00111111                     ' assume all pressed
        FOR loop = 1 TO 5                   ' test five times
          LOW SL_165                        ' load data from keys
          PAUSE 1
          HIGH SL_165                       ' allow data to shift in

          SHIFTIN DI_165, CLOCK, MSBPRE, [tempB\8]
```

```
          key = key & ~tempB                ' test against new input
          PAUSE 5                           ' wait 5 ms between tests
        NEXT
        RETURN


PrintTime:
        tempB = hrs
        GOSUB LCDdec2
        char = ":"
        GOSUB LCDputc
        tempB = mins
        GOSUB LCDdec2
        RETURN


PrintDay:
        addr = Days + (day * 4)             ' point to day string
        GOSUB LCDprint                      ' print it
        RETURN
```