Column #49, May 1999 by Lon Glazner:

Dual Digital Power Supply — Part 1

Just a few days ago, I was back in the lab, rummaging around for parts for this Stamp Applications article. By delving deep into our company's hoard of electronic parts, I can usually drum up some cool stuff to work with. So, with a box full of components, I went to work laying out my schematic for the design. Since there was one component in particular I hadn't worked with before, I decided that it would be prudent to do a few tests on the component in question to get a feel for how it worked.

So I strolled off into the lab looking for a place to work. Now I should say that we're quite well-equipped for electronic design. But each lab station that I went to was either being used for test or verification of various projects. It seemed that the only thing I really needed was a power supply for my design. I could have made use of the 10 or 20 wall plugs that we have stashed away. But what I really wanted was a dual regulated power supply with a display. That's when the gears started turning in my head.

Component names and descriptions were flashing down the myriad of synapses in my slightly twisted engineer's brain. It's great when you stumble upon a project that you really want to build.

After pulling out some old data sheets, and looking over last month's Stamp Applications article, I felt like I had a handle on a digitally controlled, BASIC Stamp based, solution to my power supply woes. With reckless abandon, I deemed it necessary to mercilessly throttle two birds with one stone, or hand, or whatever. In other words, I would write my Stamp application about a power supply that I could use for other Stamp projects.

Defining the Design

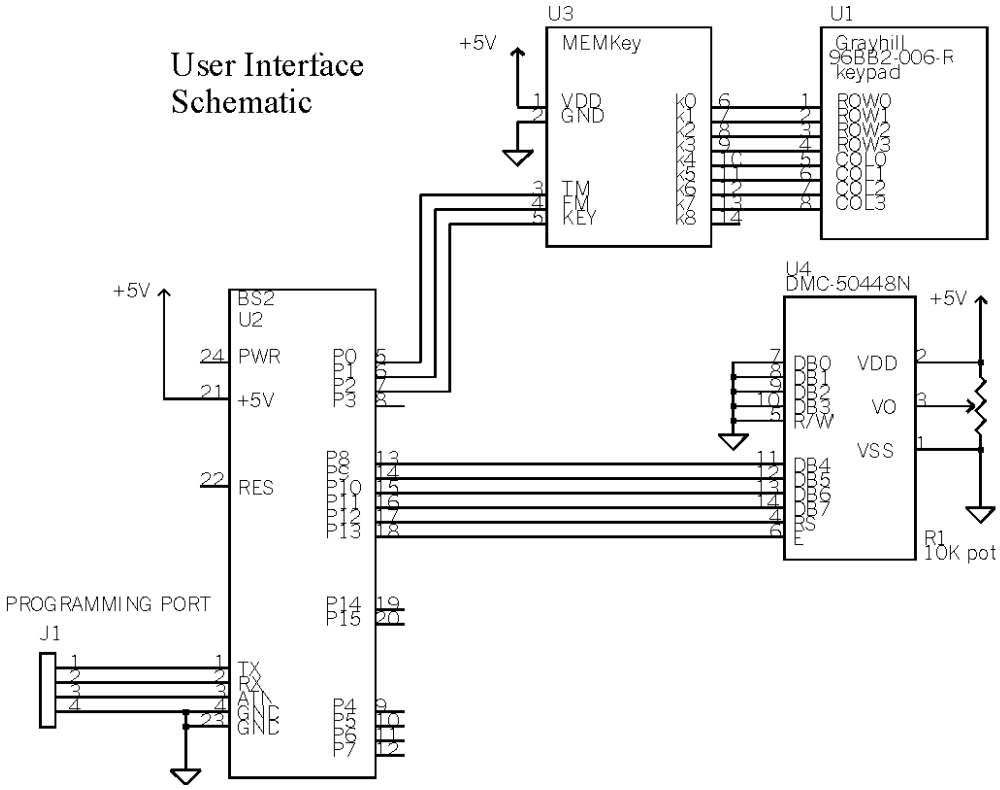
Quite often, a design can be chopped up into smaller pieces. I find it easier to stay focused on a design when I can see portions of the design completed. Dividing your designs up into segments also allows you to work on multiple projects without getting too bogged down. For this particular project, I'm going to break up the two major design blocks over a couple of articles. This month, I'll focus on designing a user interface with a minimal number of I/O lines. Next month, I'll define the power supply parameters and control circuitry.

For ease of use, I want the dual power supply to have a sophisticated user interface. Power supply settings should also be stored in non-volatile memory. Furthermore, I want the power supply settings to be entered with a keypad and not via a potentiometer. I'll use the Solutions Cubed product — the MEMKey serial keypad encoder — for the keypad to BS2 interface. This requires about the same number of I/O lines as a couple of potentiometers, and allows a significant increase in functionality. Both voltage levels will be displayed via a 2x8 LCD display utilizing a four-bit interface protocol.

This LCD interface has been covered extensively in previous Stamp Application articles. In fact, the code I'm using here was taken directly out of a previous author's Stamp Application.

For this project, I'm going to concentrate on the actual user-interface rather than the mechanics of LCD communication. It looks as though the four-bit LCD interface and the serial MEMKey interface will allow enough I/O lines for the power supply controls as I have them envisioned. But if I run short of I/O lines, I can always opt for a Scott Edwards Electronics serial LCD backpack. I don't think I'll need one this time around, but it's nice to know it's there if I do.

Figure 49.1: User Interface Schematic



Connecting the Parts

Interfacing a Stamp to a keypad can require quite a few resources. Sometimes, you can multiplex some of the I/O lines between your keypad and other electronics. But implementing typematic rates and key debounce can tie up your Stamp and use memory that might be necessary to complete the design.

To reduce the amount of resources that the Stamp needs to interface to a keypad, I'm going to use the Solutions Cubed MEMKey. This is a product that just rolled out of production around mid-March. There's a Stamp-friendly interface scheme built into the MEMKey, as well as non-volatile scratch pad memory. Many of the settings — including the row and column locations — are programmable via a 2400 baud 8N1 serial interface. The MEMKey's row and column configuration defaults to the low-cost Grayhill 96 series keypads, which are available from both Parallax and Digi-Key. The 2x8 LCD screen

(Optrex DMC-50448N) that I will use is also available from Digi-Key.

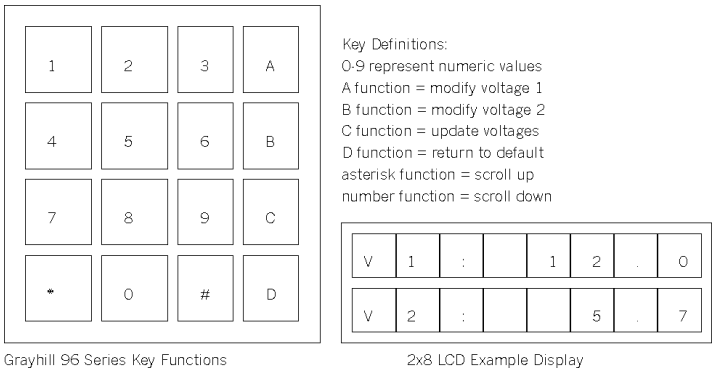
The MEMKey, keypad, and LCD screen will be connected as shown in Figure 49.1. Notice that I left six I/O lines open on my schematic for the power supply controls. At this point in the project, I have a complete schematic of the design, but it is only hand drawn. I didn't want to commit to the power supply portion of this circuit until I have spent a little more time doing design and testing. It's also easier to read the schematic for the user interface components when the power supply portion has been omitted.

As you can see from Figure 49.1, I have included a programming port for the Stamp, and have selected the BASIC Stamp 2 (BS2) for this design. The programming port makes good sense, and I try to include it on all of my Stamp designs that migrate to a printed circuit board (PCB).

One of the real strengths of the BASIC Stamp is the ability to be programmed over and over again. This coupled with the fact that BASIC Stamps have DEBUG commands built into them means that after-market modifications can be implemented quickly. A programming port for this design has other ramifications.

For instance, I may wish to have a power supply that can be controlled by a PC, or one that varies its output over set periods of time. If those needs arise, I can reprogram this hardware to do just that, because the programming port will be available.

Figure 49.2: Keypad and LCD example



Keypad and LCD Setup

One of the first things you should do when designing a user interface is define the functionality you expect to implement. A good way to get this started is to label your keypad keys with values and functions. Next, you can actually diagram your LCD display. Figure 49.2 is an example of the what I'm talking about.

In my mind's eye, I see a couple of functions that I would like to implement in this dual supply. For example, if a user wished to modify voltage 1 so that it was equal to 12.0V, they could do it in one of two ways. To select a numeric value, they would press the key marked A. Next, they could press key 1, 2, and 0. Then by pressing key C the display would be updated, and the power supply would output 12V at the voltage 1 terminal.

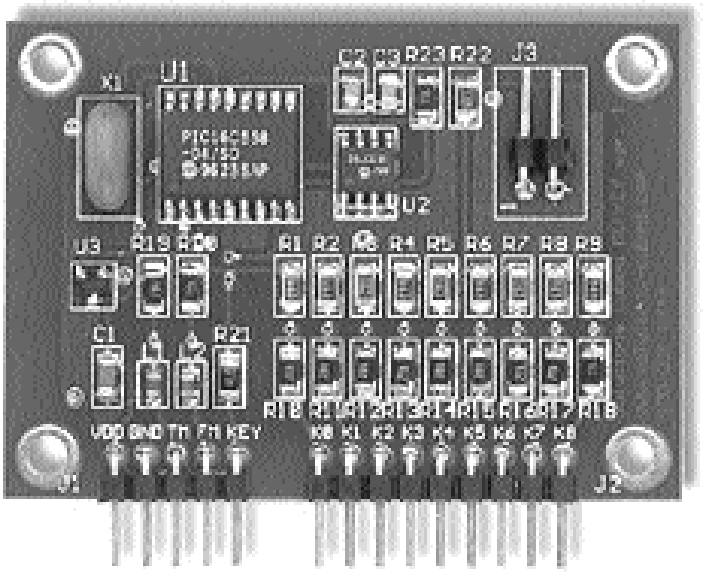
A second method of modifying the voltage would be to press the key marked A, and then scroll the voltage value up or down using the asterisk or pound key. When the correct voltage level was reached, the C key would update the voltage output. To modify the second voltage available in this dual supply, the key marked B would be used in place of the key marked A in both of the previous examples.

As far as the 2x8 LCD screen is concerned, there are three digits allotted for the voltage values. Leading zeros will not be displayed. There is room on the LCD to display the voltage with a resolution of 10mV. However, it's unlikely that a resolution greater than 100's of millivolts will be useful since this design will make use of eight bit A/Ds with an input range of zero to five volts.

You'll see as this process goes on that user interfaces are pretty code intensive. If we start running out of program room, we can always upgrade from the BS2 to the BS2SX which has quite a bit more program memory, as well as more program RAM.

One thing I want to be sure to do is to keep the code as simple as possible. I hate to read an application note where the writer implements cryptic programming techniques. Unless the application note is specifically about programming techniques, I think it's best to implement everything in the simplest code possible. As this project goes on, we may be forced to get tricky to make everything fit. If that's the case, I'll be sure to provide notes in the source code if not in the text of the article.

Figure 49.3: MemKey



Using the MEMKey

The MEMKey serial keypad encoder (Figure 49.3) can be used with keypad matrices up to 4x5 in size. The Grayhill 96BB2-006-R that I'm using here is an example of a 4x4 keypad. The MEMKey has two forms of serial interfaces. The first is PC/AT, which is one of the protocols that IBM style PCs use to communicate with keyboards. The PC/AT protocol is selected by removing the jumper J1 on the MEMKey. BASIC Stamp users will not want to make use of the PC/AT interface. That function is included for use by original equipment manufacturers (OEMs) of electronic devices and keypad suppliers.

The second interface is selected by leaving the jumper J1 in place (as the product is shipped). This jumper selects a serial communication protocol consisting of a 2400 baud data rate, eight data bits, no parity, and one stop bit (8N1). Like all Solutions Cubed Miniature Engineering Modules (Mini-Mods), this device can make use of a one- or two-wire serial interface. I'll stick with the two wire serial interface for simplicity.

The complete data sheet for the MEMKey can be downloaded in PDF format from www.solutions-cubed.com. As with all Mini-Mods, a Master is defined as the device accessing the Mini-Mod, in this case, the MEMKey. Our Master unit will be the BS2.

The MEMKey, when using its serial interface, can be configured for Automatic Mode, or Polled Mode. In Automatic Mode, the MEMKey sends key press data to the Master unit automatically. This can be useful when your Master unit is a PC or microcontroller with interrupt driven serial communication hardware built into it.

For a BS2, we should concentrate on Polled Mode. Polled Mode doesn't send key press data until the Master unit reads the keypad buffer. The MEMKey stores the key presses in an eight key FIFO (first-in first-out) buffer, for later retrieval by the Master. In both modes, the MEMKey KEY pin is set to a logic high when any key is pressed. For ease of use, the BS2 can simply check the KEY pin every now and again and, if it's high, it can perform a Read Key Buffer, and act on the data received.

User Interface Description

For this particular design, my BS2 doesn't have to be doing much other than waiting for key presses. Next month, when I add the power supply control circuitry, that will probably change. If you have a particular design that can't be waiting around for keys to be pressed you can periodically poll the KEY pin of the MEMKey for a logic high. Using this method, the key press buffer may be read at the convenience of the Master unit.

My user interface stores and displays ASCII characters. Next month, when the power circuitry is defined, there will have to be an ASCII-to-binary conversion. It's not likely to be a straight conversion. For that reason, I think it's best to save a little code space and heartache by keeping the user interface simple. All ASCII data is stored in the MEMKey's user EEPROM. Of the 64 bytes of EEPROM available, only 16 were consumed by this program. The user can modify the voltage portions the LCD display. I haven't implemented the voltage scroll-up or scroll-down commands yet (asterisk and pound keys). I wanted to wait until the power supply is better defined before tackling those functions.

The user interface, in its current state, allows user input to adjust the two voltage displays independently. The voltage range is currently limited to between 3.0 and 20.0, in 100mV increments. This can be easily modified in software. The reset function has also been implemented and resets both voltage displays to 5.0. All in all, the keypad and LCD interface appears to be pretty robust.

Believe it or not, one of the best ways to test a keypad based design is to press as many keys as you can as fast as possible. Press multiple keys at once, and be sure to press the numeric and function keys. If there's a hole in your programming, it's bound to turn up when you do this. A solid user interface will not be affected by this kind of mistreatment. If you provide for worst case possibilities, like someone laying a heavy book across your keypad, or sitting on it, then the design will probably survive the real world. And if you're not sure you can account for everything the world is going to throw at you, you can always include a reset button just for good measure.

In Closing

The user interface went together pretty smoothly. It's always nice to get that part of a design completed. Whenever I have a project that requires a user interface, I like to design that part of the project first. When you've got a user interface completed, you can actually use it as a debugging tool during the rest of the design.

The tricky part to a user interface is being able to foresee all of the weird things the end user is going to do to the device. When testing, you first need to be sure that if the user presses all of the right buttons, all of the right things happen. Second, you need to make sure that when a user presses all of the wrong buttons, none of the bad things happen.

This design isn't all that efficient as far as the code is concerned. There are a few big chunks of code that differ only with respect to the MEMKey user EEPROM that is accessed. These code blocks could be combined. I think the code itself is somewhat self-explanatory. This program took up about half of the BS2's EEPROM, and RAM. Since the display characters are stored in the MEMKey's EEPROM you could probably hone the RAM requirements down to just a few bytes.

I guess it's time to get back into the lab and figure out the power specifications for the power supply. I hope you found something useful in this article, and I'll see you again next month.

```
'Program Listing 49.1
'MAY99.BS2 - User interface code listing. This source code implements a
'user interface consisting of a 2x8 LCD screen operating in four bit mode,
'and a 4x4 keypad. The LCD is driven directly with a BASIC Stamp 2 while
'the keypad is encoded by a MEMKey serial keypad encoder. LCD display data
'is stored in the MEMKey's user accessible EEPROM.This code is phase one
'of a BS2 based power supply. Many of the limits imposed on the display
'values can be modified during the second phase of the project which will
'define the regulated DC output of the system.
'
' LCD constants
RS      CON     12              ' Register Select (1 = char)
E       CON     13              ' LCD Enable pin (1 = enabled)
'
'LCD control characters
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
'
' LCD Variables
Char    VAR     Byte            ' Character sent to LCD
'
' MEMKey pin assignments
TM      CON     0               ' To Master
FM      CON     1               ' From Master
KEY     CON     2               ' Key press notification pin
'
' MEMKey variables
Index   VAR     Byte            ' For next loop variable
KeyVal  VAR     Byte            ' Storage for key values
B_1     VAR     Byte            ' Variable storage byte
B_2     VAR     Byte            ' Variable storage byte
B_3     VAR     Byte            ' Variable storage byte
B_4     VAR     Byte            ' Variable storage byte
B_5     VAR     Byte            ' Variable storage byte
B_6     VAR     Byte            ' Variable storage byte
B_7     VAR     Byte            ' Variable storage byte
B_8     VAR     Byte            ' Variable storage byte
B_9     VAR     Byte            ' Variable storage byte
'
' MEMKey constants
Baud    CON     396             ' Baud rate = 2400
PConfig CON     $0E             ' Program configuration command
Config  CON     $00             ' Disable typematic, disable auto
PDBounce CON    $04             ' Program debounce command
DBounce CON     $0A             ' Set debounce for 25ms
```

```
Peeprom CON     $08             ' Program user EEPROM command
Reeprom CON     $09             ' Read user EEPROM command
Pkeyval CON     $0A             ' Program key value command
Rkeyval CON     $0B             ' Read key value command
Default CON     $11             ' Resets MEMKey to default values
Rbuffer CON     $00             ' Read key in buffer
'
'*****************************************************************************
'
' This routine initializes the BASIC Stamp, LCD, and MEMKey.

' Initialize the BS2
BS2_ini:
  DirH  = %01111111             ' set pins 8-15 direction
  OutH  = %00000000             ' clear the pins
  DirL  = %00000010             ' set pins 0-7 direction
  OutL  = %00000010

' Initialize the LCD (Hitachi HD44780 controller)
'
LCD_ini:
  OutC = %0011                  ' 8-bit mode
  PULSOUT E, 1
  PAUSE 5
  PULSOUT E, 1
  PULSOUT E, 1
  OutC = %0010                  ' 4-bit mode
  PULSOUT E, 1
  Char = 40                     ' Set for 2 line operation
  GOSUB LCDcmd
  Char = 12                     ' Shift cursor right
  GOSUB LCDcmd
  Char = 6                      ' Increment DDRAM after write
  GOSUB LCDcmd
  Char = 1                      ' Clear LCD screen
  GOSUB LCDcmd

'
' Initialize the MEMKey
MEMKey_ini:
  HIGH  FM                      ' Make sure FM is high
  PAUSE 2000                    ' Let the system power settle
  SEROUT FM,Baud,[PConfig,Config]
' Configure MEMKey for Polled Mode
  PAUSE 15                      ' Pause 15ms for EEPROM access
  SEROUT FM,Baud,[PDBounce,DBounce]
' Program debounce value
  PAUSE 15                      ' Pause 15ms for EEPROM access
'
  GOSUB DisplayLCD
'*****************************************************************************
```

```
MainProgram:
  GOSUB KeyFind               'Poll for a key press
  GOSUB ModeSelect
'If key is pressed start user interface interaction
GOTO   MainProgram
'
'
'****************************************************************************
' -----[ Subroutines ]-------------------------------------------------------
'****************************************************************************
' LCD commands, such as address pointer, are sent via LCDcmd, characters
'are sent with the LCDwr routine. This routine and LCD initialization
'routines taken from a previous author's Stamp Applications code listing.
'It was Jon Williams who wrote the original code in Nuts & Volts Volume
'18, Number 9 (September 1997 pg. 41).
'
LCDcmd:
  LOW RS                      ' Enter command mode
                              ' then write the character
LCDwr:
  OutC = Char.HIGHNIB         ' Output high nibble
  PULSOUT E, 1                ' Strobe the Enable line
  OutC = Char.LOWNIB          ' Output low nibble
  PULSOUT E, 1
  HIGH RS                     ' Return to character mode
RETURN
'****************************************************************************
' All display data including voltage levels are stored in the MEMkey's
'EEPROM in locations $00-$0F. Whenever the display is updated each
'character is read from EEPROM and then sent to the LCD. The leading zeros
'for the voltage levels on both line one and line two are displayed as
'ASCII " "(space). This makes the display look a little nicer. If this
'display update routine is too slow for your design you can just update
'the voltage levels.
'
DisplayLCD:
Line1:
  Char =      $80             ' Display line one
  GOSUB LCDcmd
  For Index = $00 to $07
    SEROUT FM,Baud,[Reeprom,Index] ' Read EEPROM command
    SERIN TM,Baud,[B_1]       ' Display value is read
    Char = B_1
    If Index <> $04 then Continue1
' Test for leading zero
    If Char <> $30 then Continue1 ' If ASCII zero then
    Char = " "                ' replace with blank space
Continue1:
    GOSUB LCDwr
  Next
```

```
Line2:
  Char = $C0                  ' Display line two
  GOSUB LCDcmd
  For Index = $08 to $0F
  SEROUT FM,Baud,[Reeprom,Index]' Read EEPROM comand
  SERIN TM,Baud,[B_1]         ' Display value is read
  Char = B_1
  If Index <> $0C then Continue2
' Test for leading zero
  If Char <> $30 then Continue2 ' If ASCII zero then
  Char = " "                  ' replace with blank space

Continue2:
GOSUB LCDwr
Next
RETURN
'*****************************************************************************
' The Reset subroutine returns the MEMKey to its initial settings. It also
' resets the LCD display values. Upon initial power up of this design
' a "GOSUB Reset" should be placed prior to entering the MainProgram code
' space. After the EEPROM has been initialized, the "GOSUB Reset" may be
' commented out or deleted. By doing this the last values displayed, prior
' to a power down, will be the values loaded on power up.
'
Reset:
  SEROUT FM,Baud,[Default]    ' Reset MEMKey to default settings
  PAUSE 200
  SEROUT FM,Baud,[PConfig,Config]
' Configure MEMKey for Polled Mode
  PAUSE 15                    ' Pause 15ms for EEPROM access
  SEROUT FM,Baud,[PDBounce,DBounce]  ' Program debounce value
  PAUSE 15                    ' Pause 15ms for EEPROM access
                              ' Update key values and display values
  For Index = $00 to $0F
    LOOKUP
Index,[$00,$01,$02,$03,$04,$05,$06,$07,$08,$09,$0A,$0B,$0C,$0D,$0E,$0F],B_
1
    LOOKUP
Index,["1","2","3","A","4","5","6","B","7","8","9","C","*","0","#","D"],B_
2
    LOOKUP Index,["V","1",":"," ","0","5",".","0","V","2",":","
","0","5",".","0"],B_3
    SEROUT FM,Baud,[Peeprom,B_1,B_3]
    PAUSE       15
    SEROUT FM,Baud,[Pkeyval,B_1,B_2]
    PAUSE       15
  Next
RETURN
'*****************************************************************************
' ModeSelect determines which voltage is being adjusted, or if a reset
' command has been implemented. The scroll-up and scroll-down modes have
```

```
' not yet been implemented. The scroll functions
' can be defined once the voltage control method has been proved in the
'lab.

ModeSelect:
  If KeyVal = "A" then AdjustV1      ' Adjust voltage one has been
selected
  If KeyVal = "B" then AdjustV2      ' Adjust voltage two has been
selected
  If KeyVal = "D" then ResetSupply ' A reset command has been entered
RETURN         ' Key other than A,B, or D was pressed and ignored

AdjustV1:
  SEROUT FM,Baud,[Peeprom,$03,">"]   ' Load a ">" into EEPROM
  PAUSE 15
  GOSUB DisplayLCD     ' Display the ">" next to the voltage being adjusted
  B_2 = $04            ' Start EEPROM address at voltage values
AdjV1Continue:
  GOSUB KeyFind        ' Wait for the next keypress
  debug ishex2 B_2,cr
  If KeyVal = "C" then AdjV1Done
' If a "C" was pressed then were done adjusting
  If KeyVal > "9" then AdjV1Over
' Check to see if A through D key was pressed by accident
  If KeyVal = "#" then AdjV1Over
' Check to see if pound key was preseed
  If KeyVal = "*" then AdjV1Over
' Check to see if asterisk key was pressed
  SEROUT FM,Baud,[Peeprom,B_2,KeyVal]
' Store numeric 0-9 in current EEPROM address
  debug ishex2 B_2,cr
  PAUSE 15
GOTO AdjV1Again
AdjV1Over:
  SEROUT FM,Baud,[Peeprom,B_2,"0"]
' If keypress was "A","B", or "D" then return a "0"
  PAUSE 15
AdjV1Again:
  GOSUB DisplayLCD           ' Update display with the latest key press
  B_2 = B_2 + 1              ' Increment EEPROM address pointer
  If B_2 = $06 then AdjV1Again
' If EEPROM pointer is pointed at "." then increment again
  If B_2 = $08 then AdjustV1
' If line2 value is pointed at then reset EEPROM pointer
GOTO   AdjV1Continue                 '
AdjV1Done:
  GOSUB Limits               ' Make sure values are within limits
  RETURN
AdjustV2:
  SEROUT FM,Baud,[Peeprom,$0B,">"]
' Comments for the second line are the same except all
```

```
  PAUSE 15                    ' EEPROM pointers are 8 greater than line 1
  GOSUB DisplayLCD
  B_2 = $0C
AdjV2Continue:
  GOSUB KeyFind
  If KeyVal = "C" then AdjV2Done
  If KeyVal > "9" then AdjV2Over
  If KeyVal = "#" then AdjV2Over
  If KeyVal = "*" then AdjV2Over
  SEROUT FM,Baud,[Peeprom,B_2,KeyVal]
  PAUSE 15
GOTO AdjV2Again
AdjV2Over:
  SEROUT FM,Baud,[Peeprom,B_2,"0"]
  PAUSE 15
AdjV2Again:
  GOSUB DisplayLCD
  B_2 = B_2 + 1
  If B_2 = $0E then AdjV2Again
  If B_2 = $10 then AdjustV2
  GOTO AdjV2Continue
AdjV2Done:
  GOSUB Limits
RETURN
ResetSupply:
  GOSUB Reset                 ' Implement reset command
  GOSUB DisplayLCD            ' Update display
RETURN
'****************************************************************************
' Keyfind looks for a logic high on the KEY pin(IN2). If one is present
' then the Read Key Buffer command for the MEMKey is implemented. There is
' enough RAM allotted in the SERIN command to read a maximum
' buffer size of 8 bytes. It is likely that only one key value will be
' returned. The SERIN "escape clause" has been set to 50ms to ensure that
' the serial communication does not hang up the program. Once this
' subroutine is entered it will not exit until a key has been pressed.
' When it does exit the key value will be loaded into the KeyVal variable.
'
KeyFind:
  If IN2 <> 1 Then KeyFind    ' Check the KEY pin for a logic high
  SEROUT FM,Baud,[Rbuffer]    ' Read buffer command
  SERIN TM,Baud,50,DoneBuffer,[KeyVal,B_3,B_4,B_5,B_6,B_7,B_8,B_9]
DoneBuffer:
  SEROUT FM,Baud,[Rbuffer]
' If there is a fast key press don't accept it
  PAUSE 40
  If IN2 <> 0 then DoneBuffer' Wait for KEY pin to go to logic low
RETURN
'****************************************************************************
' The Limits subroutine checks entered values for out of range or mis-
keyed entries. If either
```

```
' voltage input is greater than 20.0 or less than 03.0 then the values are
'forced to either the minimum or maximum values accepted. Prior to this
'routine being exited the ">" character that was loaded next to the
'adjusted voltage is replaced with a space character.
'
Limits:
       SEROUT FM,Baud,[Reeprom,$04]
' Read tens character of voltage on line 1
       SERIN TM,Baud,[B_1]
       If B_1 > "1" then ZeroOnesV1
' Check to see if tens is greater than "1"
       If B_1 = "0" then TestOnesV1
' Check to see if tens is a zero
       GOTO   NextLimit
ZeroOnesV1:
       SEROUT FM,Baud,[Peeprom,$04,"2"]
' If tens is greater than "1" then force display
       PAUSE  15                    ' to "20.0". The "." is not adjusted
       SEROUT FM,Baud,[Peeprom,$05,"0"]
       PAUSE  15
       SEROUT FM,Baud,[Peeprom,$07,"0"]
       PAUSE  15
       GOTO NextLimit
TestOnesV1:
       SEROUT FM,Baud,[Reeprom,$05]
' If tens is "0" the read ones character
       SERIN TM,Baud,[B_1]
       If B_1 > "2" then NextLimit
' Check to see if ones is less than "3"
       SEROUT FM,Baud,[Peeprom,$05,"3"]
' If so then force display to "03.0"
       PAUSE  15
       SEROUT FM,Baud,[Peeprom,$07,"0"]
       PAUSE  15
NextLimit:
       SEROUT FM,Baud,[Reeprom,$0C]
' Test voltage on line 2 as line one was tested
       SERIN TM,Baud,[B_1]
       If B_1 > "1" then ZeroOnesV2
       If B_1 = "0" then TestOnesV2
       GOTO   DoneLimit
ZeroOnesV2:
       SEROUT FM,Baud,[Peeprom,$0C,"2"]
       PAUSE  15
       SEROUT FM,Baud,[Peeprom,$0D,"0"]
       PAUSE  15
       SEROUT FM,Baud,[Peeprom,$0F,"0"]
       PAUSE  15
       GOTO   DoneLimit
TestOnesV2:
       SEROUT FM,Baud,[Reeprom,$0D]
```

```
        SERIN TM,Baud,[B_1]
        If B_1 > "2" then DoneLimit
        SEROUT FM,Baud,[Peeprom,$0D,"3"]
        PAUSE  15
        SEROUT FM,Baud,[Peeprom,$0F,"0"]
        PAUSE  15
DoneLimit:
        SEROUT FM,Baud,[Peeprom,$03," "]
' Make sure to replace either ">" with a " "(space)
        PAUSE  15
        SEROUT FM,Baud,[Peeprom,$0B," "]
        PAUSE  15
        GOSUB  DisplayLCD
        RETURN
'****************************************************************************
END:
```