Order this document
by AN1287/D

Motorola Semiconductor Application Note

AN1287

MC68HC708LN56 LCD Utilities

By Rick Cramer
CSIC Product Engineering
Austin, Texas

Introduction

A set of software utilities that causes the LCD module on the MC68HC708LN56 to function is described in this application note. Information about LCD software subroutines that, with minimal effort, can be called to write text to the display also is included here. Additionally, this information can be used as a basis to develop more complex graphical subroutines.

LCD Hardware General Information

The LCD module has of group of frontplanes and backplanes that intersect on the display to form pixels. The 40 frontplanes and 32 backplanes form 40 x 32 (or 1280) pixels. By implementing the LCD hardware in different configurations, these pixels can be arranged to form any type of display. When the hardware is arranged in a two-dimensional array, the pixels form a display of 40 x 32 dots. By turning on these pixels in a specific pattern, alphabet characters or special symbols can be formed. All the characters on a typical computer keyboard can be displayed by an array of pixels seven pixels high by five pixels wide, which enables the MC68HC708LN56 to display an 8 x 4

MOTOROLA

character array. However, by using different hardware implementation methods, a 16 x 2 character array also can be formed.

This application note contains information for a 16 x 2 character array, although the array can be modified easily to work for any configuration. With this type of hardware configuration, the LCD array has 32 possible character positions. The subroutines in this application note use the values of $00 through $1F to represent the position in the array that each character occupies. This is shown in **Figure 1**.

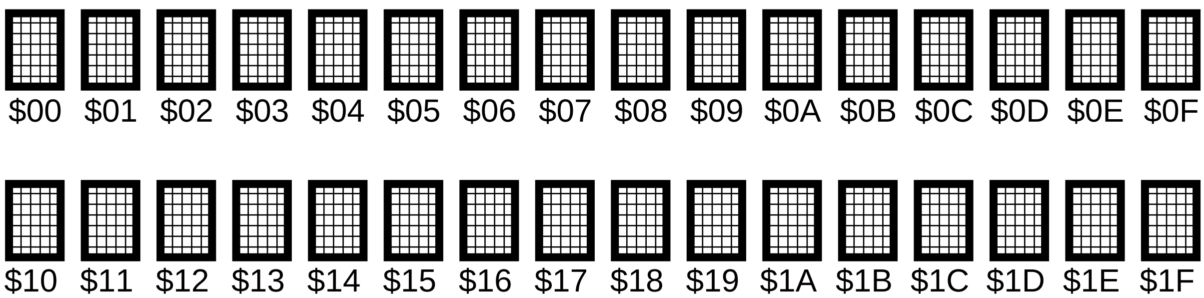


Figure 1. LCD Display Representation

The MC68HC708LN56 associates one RAM byte (8 bits) for each column in a character. So, a 5 x 7 character actually takes up 5 x 8 bits of RAM. Each column of every LCD character has a specific memory address associated with it. By writing to these addresses, as shown in **Figure 2**, the display can be made to exhibit information.

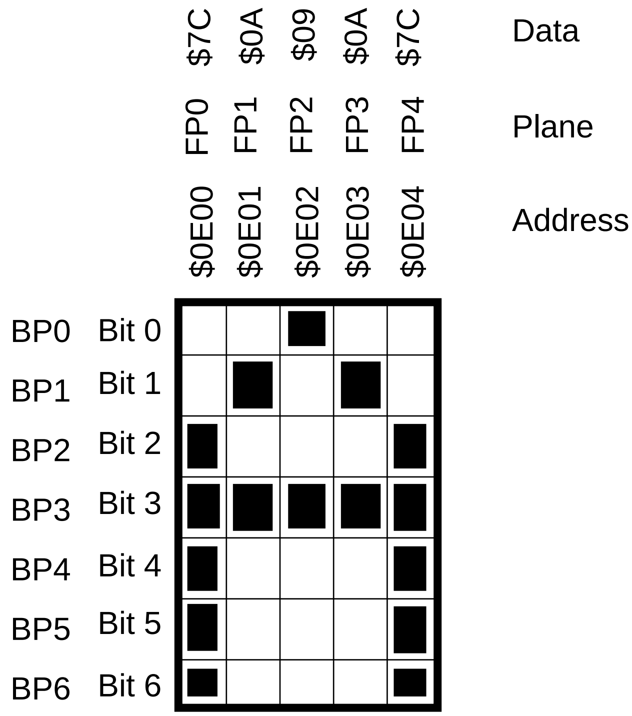


Figure 2. LCD Character Representation

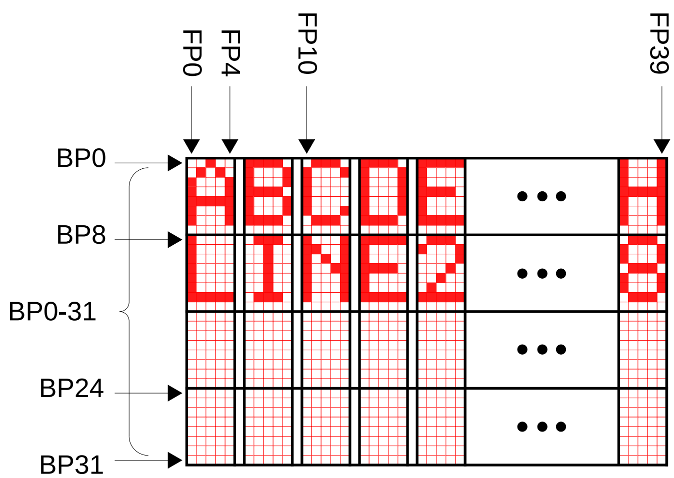


Figure 3. LCD Panel Dot Matrix Example

Subroutine Descriptions

The next sections list and describe the subroutines' functionality from the programmer's viewpoint, including a specific example of how the LCD works. A set of tested, working LCD subroutines also is provided for Motorola customers.

Because the input required to run the subroutines and the output returned from the subroutines are provided, the code is presented from the end-user's point of view. This means that the user does not have to understand the code to utilize the subroutines, which decreases software development cycle time. Setting a few parameters in a couple of tables and calling the subroutine are all that is necessary for using these subroutines.

General Description

The LCD utilities and their capabilities are listed here. They are described in more detail in following sections to provide a more thorough guide to their usage.

- The WR_STR (write string) subroutine writes an ASCII string to the LCD. It is used to write text messages or user prompts such as “Messages Waiting” or “Press Any Key."
- The BINTOASC (binary to ASCII) subroutine displays the hexadecimal equivalent of a binary character. This is more of a "programmer's friend" subroutine that can be used in code debugging. Its primary use is to display the value of data that is contained in a memory location.
- The WR_BIN (write binary) subroutine displays the ASCII representation of the data contained in the X register when called. In short, it is called to access the character ROM table directly. This is necessary for writing special user-designed graphic characters.
- The CLS (clear screen) subroutine clears all positions on the display.

- The INV (invert screen) subroutine turns all the on pixels to off and all the off pixels to on. It typically can be used for getting the user's attention.

Main

The main code section is designed as an example of what is needed to enable the LCD and to call the subroutines. The main code clears the display, writes text to the screen, then loops through memory, updating only sections of the screen and displaying the current address and the data contained at that address. The loop repeats after cycling through memory.

RAM Subroutines

The RAM subroutines are designed to be called by other subroutines. They are modified by the calling subroutine before they are called. The RAM subroutines contain this assembled data:

```
0050 C6 10 23       LDA     $1023
0053 81             RTS
0054 C7 45 67       STA     $4567
0057 81             RTS
```

Memory location $50 contains the opcode for LDA, $C6. Two locations contain the address where the data is to be loaded from: Location $53 contains the RTS opcode and location $54 contains the LDA opcode.

To change the address, the subroutine writes over the address portion of the RAM with the new address. This way the subroutines can read by reference memory locations in any memory page. That is, it is easy to cross reference addresses. To call these routines, simply load the H:X register with the pointer to the address you wish to read, store the H:X register in location $51, then jump to the subroutine. Upon return from the subroutine, the accumulator contains the value of the data at the specific memory location.

NOTE: *Care must be taken not to overwrite the two opcodes, LDA and RTS. If these locations are accidentally changed, the MCU could get hopelessly lost executing code that is invalid.*

Location $54 contains the opcode necessary to perform the store A (STA) function. This is set up to write data to any memory location on the MC68HC708LN56 memory map.

WR_BIN

The write binary (WR_BIN) subroutine displays the ASCII representation of the data contained in the X register. Before calling the WR_BIN subroutine, load the accumulator with the position on the display and load the X register with the data to be displayed. This subroutine starts by storing the data into RAM location MSG. Then it stores the delimiter character at MSG+1. This sets up the RAM message with a single-byte string. WR_BIN then calls the write-string subroutine with the position of the newly created message in RAM so the character can be written to the display.

BINTOASC

The BINTOASC subroutine displays the hexadecimal equivalent of a binary character. To call this routine, load the X register with the data to be displayed and load the accumulator with the position on the LCD where the first of two ASCII characters is to be placed. The subroutine works by separately writing each nibble of the binary data to two consecutive RAM locations. Then it calls the WR_STR subroutine. BINTOASC first filters out the upper nibble from the data, leaving the lower nibble. Then it loads the lower nibble into the X register. The BINASC table converts the binary data (from $00 to $0F) to its ASCII representation. This new ASCII data is stored in the MSG+1 memory location for later use. The subroutine then takes the original data and executes a nibble swap, placing the upper nibble in the lower nibble position. After clearing the upper nibble, the BINASC table is used to translate this binary data into its ASCII equivalent. This ASCII data is stored in the MSG RAM location. Once both nibbles have been converted, the delimiter character is placed at MSG+2, and the WR_STR subroutine is called with the pointer MSG. The position on the display is passed through to WR_STR unmodified.

WR_STR

The write string (WR_STR) subroutine writes an ASCII string to the LCD. It is called by loading the H:X register pair with the pointer to the string that is to be displayed and loading the accumulator with the position on the LCD display where the first character of the string is to be placed.

The string can be of any length as long as it ends with a delimiter. For this application note, the end of string delimiter is the close-brace (}) character. Writing a string that is longer than one line will cause it to wrap to the next line. A string that extends past the bottom of the display will be truncated at the last screen position. The WR_STR subroutine starts by putting the beginning memory location of the string off to RAM for later use. The WR_STR routine modifies the RAM subroutines to do indexed addressing by changing the opcode, then indexes through the string one character at a time. While indexing, WR_STR checks to see if the character is the delimiter. If the character is not the delimiter, WR_STR writes the character to the display using the WR_POS subroutine if it is not off the screen.

CLS

The clear screen (CLS) subroutine clears the LCD screen. Since the memory locations are all cleared, how the LCD is wired is not important. There are four blocks of LCD RAM starting at $0E00, $0E80, $0F00, and $0F80. Each is 30 bytes. The CLS subroutine indexes through each byte of all blocks and stores a $00 there. The $00 is the value associated with turning off all the dots in the matrix.

INV

The invert screen (INV) subroutine takes the data on the display and toggles each bit’s on/off state. INV works similarly to the CLS subroutine in that it indexes through the blocks. But this routine first reads the data already on the display, first, compliments it, then writes it back.

WR_POS

The write position (WR_POS) subroutine is designed to be called directly by other subroutines. This subroutine has two major functions: It uses the LCDLOC table to find the absolute memory location to write the character data, and it uses the CHARROM table to get the character pattern to write to the memory location. WR_POS starts by putting the first LCD address in the RAM subroutine containing the LDA opcode. Then it gets the absolute address of the first position of the LCD RAM for the designated position. It puts this address in the RAM subroutine containing the STA opcode. Once the addresses are stored, control is passed to the writeit or writeit2 subroutines where all five data bytes are written.

WRITEIT

The writeit subroutine is also designed to be called directly by other subroutines. It, in essence, is part of the WR_POS subroutine. This subroutine calls the RAM subroutines to load the character data from the CHARROM table and then calls the RAM subroutine again to store the data in the LCD RAM. Writeit then increments both addresses in the RAM subroutines and writes the data again. It does this five times, once each for the five bytes of data that represent the character.

WRITEIT2

The writeit2 subroutine is almost identical to the writeit subroutine, except that it writes the character data into the LCD RAM in the reverse order.

Tables

The LCD utilities use several tables which contain information that the subroutines use for positioning the characters on the LCD display.

LCDLOC

The LCDLOC table is the most important table. It relates LCD character position ($00 to $1F) to the absolute memory address in which the LCD characters reside. The first entry in the table contains information pertaining to position $00 of the LCD array. As shown in **Figure 1**, the display's upper lefthand corner is position $00 and the bottom righthand corner is position $1F. As shown in **Figure 2**, the LCD location $00 is wired to backplane 0 through backplane 6 along the side and frontplane 0 through frontplane 4 across the top. The MC68HC708LN56 specification s associate memory location $0E00 for FP0 and $0E01 for FP1 and $0E02 for FP2 and so on. This table requires the lowest memory location as its entry for each position. For instance, for position $00, the memory location $0E00 is entered into the table. Depending on how the LCD display hardware is configured, changes to this table could be necessary.

LCDBACK

The LCDBACK table is used to indicate if the writeit subroutine is to write a specific character on the LCD screen backward. This is necessary because of circuit board layout restrictions that may require some of the frontplanes to be wired in reverse order. This will enable the use of the same character table no matter how the display is wired.

CHARROM

The CHARROM table contains the data necessary to form all the ASCII characters. Since each letter is made of a 5 x 7 display, each ASCII character requires five bytes of data. The table is placed in order of its appearance in the ASCII character chart for easy cross reference. Most of the data in the CHARROM table has been developed by Nortel and has been used in this table with Nortel's permission.

Code Listings

```
*************************************************************
** LN56LCD.ASM                                             **
**              26 May 96 Rick Cramer                      **
**                                                         **
**  This program contains subroutines that will allow      **
**   easy access to the MC68HC708LN56's LCD Module.        **
**  Routines contained within will place ASCII characters  **
**   on the LCD screen.                                    **
*************************************************************


*************************************************************
*  Memory Map Equates                                       *
*************************************************************
RAM_Start      equu     $50   ; Location where RAM Starts
EPROM_Start    equ     $1E00 ; Location where EPROM Starts
LCDFL0         equ     $33   ; LCD Control and Status Requsters.
LCDFL1         equ     $34
LCDFL2         equ     $35
LCDFL3         equ     $36
LCDFL4         equ     $37
LCDCR          equ     $38
LCDCCR         equ     $39
LCDDIV         equ     $3a
LCDFR          equ     $3b
RESET          equ     $FFFE ; Reset Vectors are at $FFFE
MOR            equ     $1f   ; Mask Option Register


*************************************************************
*               RESET and Interrupt Vectors                 *
*                                                           *
* For any interrupts used, the ORG and FDB statement given  *
* below must be placed in the routine using the interrupt.  *
*                                                           *
*************************************************************
               org     RESET
               fdb      BEGIN
```

```
************************************************************
*  Varriables contained below are used by the LCD          *
*  Subroutines.                                            *
************************************************************
                org    RAM_Start

OPCD    RMB     1       ; LDA
HI      RMB     1       ; Hi Data
LO      RMB     1       ; Lo Data
OPCD2   RMB     1       ; RTS
OPCD3   RMB     1       ; STA
HI2     RMB     1       ; Hi Data
LO2     RMB     1       ; Lo Data
OPCD4   RMB     1       ; RTS
HI3     RMB     1      ; String Pointer
LO3     RMB     1
POS     RMB     1
POS2    RMB     1
DATA    RMB     1
TEMP    RMB     1
TEMP2   RMB     1
VARR    RMB     2
OFFSET  RMB     2
INVERT  RMB     1
BACK    RMB     1
STRPOS  RMB     1        ; Current Position in String
ERRCNT  RMB     1
MSG     RMB     20       ; Space for controller generated
messages.

************************************************************
*  BEGIN sets up the microcontroller for general use.      *
************************************************************
                org  EPROM_Start
BEGIN
mov #$01,MOR            ; turn off cop

************************************************************
*  MAIN subroutine is the main loop that shows how to call  *
*   subroutines. Also sets up microcontroller for LCD use.  *
************************************************************
MAIN:
       clra              ; The following section of code
                         ; writes RAM with executable code
                         ; that will be called by subroutines.
       clr    ERRCNT
       sta    INVERT
       lda     #$C6    ; Load A Extended Opcode
       sta     OPCD
       lda     #$81    ; RTS OpCode
       sta     OPCD2
```

```
        lda     #$C7    ; Store A Extended Opcode
        sta     OPCD3
        lda     #$81    ; RTS OpCode
        sta     OPCD4

*************************************************************
         jsr    CLS            ; Clear Screen Subroutine

*************************************************************
* The following section of code turns on the LCD and enables*
* it to run at a given bus frequency.                       *
*                                                           *
*  32-kHz OSCILLATOR code follows                           *
*       mov    #$01,LCDDIV    ; 32khzOSC                     *
*       mov    #$04,LCDFR     ; Frame Rate 62hz              *
*       mov    #$17,LCDCCR    ; Contrast Control             *
*       mov    #$C0,LCDCR     ; SUPV=1                       *
*                                                           *
*  4-Mhz OSCILLATOR code follows                            *
*       mov    #$9f,LCDDIV    ; 4Mhz OSC                     *
*       mov    #$04,LCDFR     ; 42h                          *
*       mov    #$17,LCDCCR    ; Contrast Control             *
*       mov    #$C0,LCDCR     ; SUPV=1                       *
*************************************************************
        mov    #$9f,LCDDIV    ; 4Mhz OSC
        mov    #$04,LCDFR     ; 42h
        mov    #$17,LCDCCR    ; VLL=7V
        mov    #$C0,LCDCR     ; SUPV=1

*************************************************************
* The X1, X2, and X2A section of code shows how to call     *
* the Write_String subroutine.                              *
*************************************************************
X1      ldhx   #ERR           ; H:X is a pointer to string
                              ; At Location #ERR
        lda    #$10           ; $10 is the LCD Screen
                              ; Position
        jsr    WR_STR         ; jump to subroutine & RTN.

12      ldhx   #ADDR          ; H:X is a pointer to string
                              ; At Location #ADDR
        lda    #$00           ; LCD Location $00
        jsr    WR_STR         ; jump to subroutine & RTN.

X2A     ldhx   #DATR          ; H:X is a pointer to string
                              ; At Location #DATA
        lda    #$0A           ; LCD Location $0A
        jsr    WR_STR         ; jump to subroutine & RTN.
```

```
*************************************************************
* The next section of code sets up a loop to cycle thru     *
*  the entire memory map starting at #BEGIN.                *
*************************************************************
Z1      ldhx   #BEGIN
        sthx   VARR

*************************************************************
* The X3, X4, and X4A section of code shows how to call     *
* the BINTOASC subroutine.                                  *
*  X3 and X4 show how to write a 2 byte address to the      *
*  screen.                                                  *
*************************************************************
X3      ldx    VARR           ; Load X reg with binary data
        lda    #$05           ; Load Acc with LCD Position
        jsr    BINTOASC       ; jump to subroutine & RTN.

X4      ldx    VARR+1         ; Load X reg with binary data
        lda    #$07           ; Load Acc with LCD Position
        jsr    BINTOASC       ; jump to subroutine & RTN.

X4A     ldx    ERRCNT         ; Load X reg with binary data
        lda    #$19           ; Load Acc with LCD Position
        jsr    BINTOASC       ; jump to subroutine & RTN.

*************************************************************
* The X5 Section of code shows how to call                  *
* the WR_BIN subroutine.                                    *
* X5 Writes the ASCII EQUIVALENT of binary data             *
* It can be used to write custom graphic characters.        *
*************************************************************
X5      ldx    VARR
        lda    #$0F
        jsr    WR_BIN

*************************************************************
* Following code increments the main loop address and jump  *
* back to update the address and data.                      *
*************************************************************
        inc    VARR+1
        bne    X3
        inc    VARR
        ldhx   VARR
        cphx   #ENDLOC
        bne    X3
        bra    Z1
********* END OF MAIN PROGRAM ***************************
********* BEGINNING OF SUBROUTINES **********************
```

```
**************************************************************
***** Binary write routine                                  **
*****  Enter with  X = Binary Number                        **
*****         and  A = Location on LCD                      **
**************************************************************

WR_BIN
       stx    MSG      ; Store Data in RAM
       sta    TEMP     ; Store loc in TEMP
       lda    #"}"     ; End MSG delimiter
       sta    MSG+1    ;
       ldhx   #MSG     ; Setup for subroutine call
       lda    TEMP     ; Get position from TEMP
       jsr    WR_STR   ; jump to subroutinr and return
       rts             ; return


**************************************************************
***** Binary to ASCII MSG                                   **
*****  Enter with  X = Binary Number                        **
*****         and  A = Location on LCD                      **
**************************************************************
BINTOASC
       sta    TEMP2    ; Store LOC for later use
       stx    TEMP     ; Store data for later use
       txa             ; Put data in A for use now
       and    #$0F     ; Use only lower nibble
       tax             ; Store it in X for indexing
       lda    BINASC,x ; ASCII data stored at #BINASC
       sta    MSG+1    ; store the lower nibble
       lda    TEMP     ; Get the data back
       nsa             ; put upper nibble in lower
       and    #$0F     ; use only lower nibble
       tax             ; store in X for indexing
       lda    BINASC,x ; ASCII data stored at #BINASC
       sta    MSG      ; store data off
       lda    #"}"     ; End of MSG delimiter
       sta    MSG+2
       lda    TEMP2    ; Get the LCD Location
       ldhx   #MSG     ; H:X = newly created Message
       jsr    WR_STR   ; write it to screen
       rts             ; return
```

```
************************************************************
********* Write String Subroutine *************************
** Writes an ASCII string to LCD.                         **
** H:X contains pointer to beginning of                   **
** string, string must end with a delimeter               **
** character }.                                           **
** If string goes past screen, subroutine                 **
** exits.                                                 **
** Call with H:X = String Pointer                         **
**       and   A = Position on LCD to start               **
************************************************************
WR_STR
       sthx   HI3           ; Store off Pointer to string
       sta    POS2          ; Location on LCD
       clr    STRPOS        ; Start at beginning
       clrh
NXT    lda    #$D6          ; LDA indexed (IX2) OPCODE
       sta    OPCD          ; store in RAM
       ldx    STRPOS        ; Get Current string position
       lda    HI3           ; Copy string beginning
       sta    HI            ; into RAM subroutine
       lda    LO3           ; do the low byte
       sta    LO            ;
       jsr    OPCD          ; execute RAM (LDA  $HI LO,x)
       cmp    #"}"          ; Is this data delimeter?
       beq    RT            ; return
       inc    STRPOS        ; Set up next character
       ldx    POS2          ; get LCD location
       inc    POS2          ; and increment for next char
       cpx    #$1f          ; Is Character off Sceen?
       bhi    RT            ; YES: Return
       JSR    WR_POS        ; NO:  Write Character
       bra    NXT           ; always do next character
RT     rts                  ; return

************************************************************
** CLS    Subroutine                                      **
** Clears LCD Screen                                      **
************************************************************
CLS     clrx           ; Clear Pointer
        lda  #$00      ; Data =$00 (BLANK)
LPX     sta  $0E00,x   ; First Bank
        sta  $0E80,x   ; Second Bank
        sta  $0F00,x   ; Third Bank
        sta  $0F80,x   ; Forth Bank
        incx           ; Next position in BANK
        cpx  #$29      ; Cleared all of them?
        bne  LPX       ; No:  Do the next
        RTS            ; Yes: Return
```

```
*************************************************************
** INV    Subroutine                                       **
** INVERTS LCD Screen                                      **
*************************************************************
INV     clrx           ; clear pointer
LPI
       lda  $0E00,x    ; get Bank1, char x
       coma            ; Invert data
       sta  $0E00,x    ; write it back
        lda  $0E80,x   ; get Bank2, char x
       coma            ; Invert data
        sta  $0E80,x   ; write it back
        lda  $0F00,x   ; get Bank3, char x
       coma            ; Invert data
        sta  $0F00,x   ; write it back
        lda  $0F80,x   ; Get Bank4, char x
       coma            ; Invert data
        sta  $0F80,x   ; write it back
        incx           ; Next character
        cpx  #$29      ; Done with bank?
        bne  LPI       ; No:  do the next
        RTS            ; yes: return

*************************************************************
** WR_POS Subroutine                                       **
** Writes ASCII data in A into LCD POS in X                **
*************************************************************
WR_POS:
           sta     DATA        ; store data for later use
           stx     POS         ; store POSition for later

** Setup ram subroutines to be called later
           lda     #$C6      ; Load A Extended Opcode
           sta     OPCD
           lda     #$81      ; RTS OpCode
           sta     OPCD2
           lda     #$C7      ; Store A Extended Opcode
           sta     OPCD3
           lda     #$81      ; RTS OpCode
           sta     OPCD4

** Get information about how to write this location
          lda    LCDBACK,x    ; Check if char POS is wired
                               backward
          sta    BACK         ; store data for later use

** Is character off screen?
           cpx     #$1f       ; Is X $1f (writing off screen)
           bhi     RETRN      ; branch if X > $1f to Return
```

```
** Find table memory location for this position
** by incrementing address POS amount of times
** The table contains ABSOLUTE memory locations for
** the lowest memory location in LCD position.
           ldhx    #LCDLOC   ; Beginning of LCD Character loc
                               Map
           sthx    HI        ; Store in Ram Subroutine
          clrh
           ldx     POS;      ; POS is $00-->$1f
LP0        beq     FINLOC    ; If POS = $00, $HI LO = LCDLOC
           jsr     INLOC     ; If not increment HI LO
           jsr     INLOC     ; Twice
           decx              ; Dec POS Counter
           bra     LP0
RETRN     rts

*** load Acc with pointer table address in RAM subroutine.
FINLOC
          jsr     OPCD    ; (LDA #$HI LO)
          sta     HI2     ; First byte of ABSOLUTE address
          jsr     INLOC   ; Get next byte in pointer table
          jsr     OPCD
          sta     LO2      ; Second byte of ABSOLUTE address

*** At this point the RAM subroutine contains the following:
*** OPCD2 STAq   $HI2 LO2
***      RTS
***
*** Now, check out the Data to write there.
GETDAT
           clrh
           clrx
           ldx     DATA   ; Load X with Data
           cpx     #$91   ; Past Character table?
           bls     VAL    ; No? Goto VAL
ERR2       nop
           RTS

** Find the beginning of character ROM data
VAL
          lda  #$05      ; 5 bytes for each char
           mul            ; X:A <--- A*X
           sta  OFFSET+1  ; # of locations from beginning
           stx  OFFSET    ; of char pattern.
           ldhx #CHARROM  ; Get loc for beginning of char
          sthx HI        ; pattern and store
```

```
** find first data byte of the current char to write
           lda HI+1        ; Get base address of char rom
           add  OFFSET+1   ; Add the offset
           sta  OFFSET+1   ; To find the first data byte
           lda  HI         ;of single character to write
          adc  OFFSET
          sta  OFFSET
          ldhx OFFSET
          sthx HI
          clrh
          clrx

***************************************************************
*** WriteIt subroutine takes beginning data byte and the    ***
***   next four and writes it onto the LCD screen.          ***
***************************************************************
WRITEIT
           lda     BACK;     ; If Char is wired backward, call
                               writeit2
           BNE     WRITEIT2  ; If LCD is wired backward

           jsr     OPCD;     ; LDA $HI LO    BYTE 1
           eor     INVERT    ; char data with INVERT
           jsr     OPCD3     ; STA $HI2 LO2
           jsr     INLOC     ; Increment Char Pattern Location
           jsr     INLOC2    ; Increment LCD ABSOLUTE location

           jsr     OPCD      ; LDA $HI LO    BYTE 2
          eor     INVERT
           jsr     OPCD3     ; STA $HI2 LO2
          jsr     INLOC
          jsr     INLOC2

           jsr     OPCD      ; LDA $HI LO    BYTE 3
          eor     INVERT
           jsr     OPCD3     ; STA $HI2 LO2
          jsr     INLOC
           sr     INLOC2

           jsr     OPCD      ; LDA $HI LO    BYTE 4
          eor     INVERT
           jsr     OPCD3     ; STA $HI2 LO2
          jsr     INLOC
          jsr     INLOC2

           jsr     OPCD      ; LDA $HI LO    BYTE 5
          eor     INVERT
           jsr     OPCD3     ; STA $HI2 LO2

           RTS
```

```
***************************************************************
*** WriteIt2 subroutine takes beginning data byte and the    **
***   next four and writes it onto the LCD screen, but it    **
***   does it backward.                                       **
***************************************************************

WRITEIT2
           jsr     INLOC2    ; increment 4 positions in the
           jsr     INLOC2    ; character table beacuse it
           jsr     INLOC2    ; will be writen backward
           jsr     INLOC2

           jsr     OPCD      ; LDA $HI LO    BYTE 1
           eor     INVERT    ; char data with INVERT
           jsr     OPCD3     ; STA $HI2 LO2
           jsr     INLOC     ; Increment Char Pattern Location
           jsr     DECLOC2   ; Decrement LCD ABSOLUTE location

           jsr     OPCD      ; LDA $HI LO    BYTE 2
           eor     INVERT
           jsr     OPCD3     ; STA $HI2 LO2
           jsr     INLOC
           jsr     DECLOC2

           jsr     OPCD      ; LDA $HI LO    BYTE 3
           eor     INVERT
           jsr     OPCD3     ; STA $HI2 LO2
           jsr     INLOC
           jsr     DECLOC2

           jsr     OPCD      ; LDA $HI LO    BYTE 4
           eor     INVERT
           jsr     OPCD3     ; STA $HI2 LO2
           jsr     INLOC
           jsr     DECLOC2

           jsr     OPCD      ; LDA $HI LO    BYTE 5
           eor     INVERT
           jsr     OPCD3     ; STA $HI2 LO2

           RTS
```

```
****************************************************************
** RAM Subroutine address Increment and Decrement             **
****************************************************************
DECLOC2     dec     LO2       ; Increment Low Address
            lda     #$ff      ; Did LO2 Dec thru a page?
            cmp     LO2
            bne     RRR4      ; If next page
            dec     HI2       ; Increment Page
RRR4        rts
******************************
INLOC       inc     LO        ; Increment Low Address
            bne     RRR       ; If next page
            inc     HI        ; Increment Page
RRR         rts
******************************
INLOC2      inc     LO2       ; Increment Low Address
            bne     RRR2      ; If next page
            inc     HI2       ; Increment Page
RRR2        rts


****************************************************************
** Beginning of Data                                          **
****************************************************************

** LCDLOC is a pointer table that points to memory locations
**  in the LCD RAM the represents the "beginning" location
**  of each character position.
LCDLOC:
            FDB     $0F00   ; Pos $00
            FDB     $0F05   ; Pos $01
            FDB     $0F0A   ; Pos $02
            FDB     $0F0F   ; Pos $03
            FDB     $0F14   ; Pos $04
            FDB     $0F19   ; Pos $05
            FDB     $0F1E   ; Pos $06
            FDB     $0F23   ; Pos $07

            FDB     $0E23   ; Pos $08
            FDB     $0E1E   ; Pos $09
            FDB     $0E19   ; Pos $0A
            FDB     $0E14   ; Pos $0B
            FDB     $0E0F   ; Pos $0C
            FDB     $0E0A   ; Pos $0D
            FDB     $0E05   ; Pos $0E
            FDB     $0E00   ; Pos $0F
```

```
            FDB     $0F80   ; Pos $10
            FDB     $0F85   ; Pos $11
            FDB     $0F8A   ; Pos $12
            FDB     $0F8F   ; Pos $13
            FDB     $0F94   ; Pos $14
            FDB     $0F99   ; Pos $15
            FDB     $0F9E   ; Pos $16
            FDB     $0FA3   ; Pos $17

            FDB     $0EA3   ; Pos $18
            FDB     $0E9E   ; Pos $19
            FDB     $0E99   ; Pos $1A
            FDB     $0E94   ; Pos $1B
            FDB     $0E8F   ; Pos $1C
            FDB     $0E8A   ; Pos $1D
            FDB     $0E85   ; Pos $1E
            FDB     $0E80   ; Pos $1F

*** LCDBACK is the data table of the individual characters
*** that are wired in backward.  This table allows
*** the subroutines to print backward characters backward
*** which makes them look the correct way when read.
LCDBACK
            FCB   $00,$00,$00,$00,$00,$00,$00,$00
            FCB   $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF
            FCB   $00,$00,$00,$00,$00,$00,$00,$00
            FCB   $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF

**** BINASC table converts binary numbers into their ASCII
     equivalent.
BINASC
            FCB   "0","1","2","3","4","5","6","7","8","9"
            FCB   "A","B","C","D","E","F"

** The following data is sample strings.

PAGE:
            fcb     'Page}'
ADDR:
            fcb     'ADR}'
DATR:
            fcb     'DAT}'
ERR:
            fcb     'ERRORS:}'
```

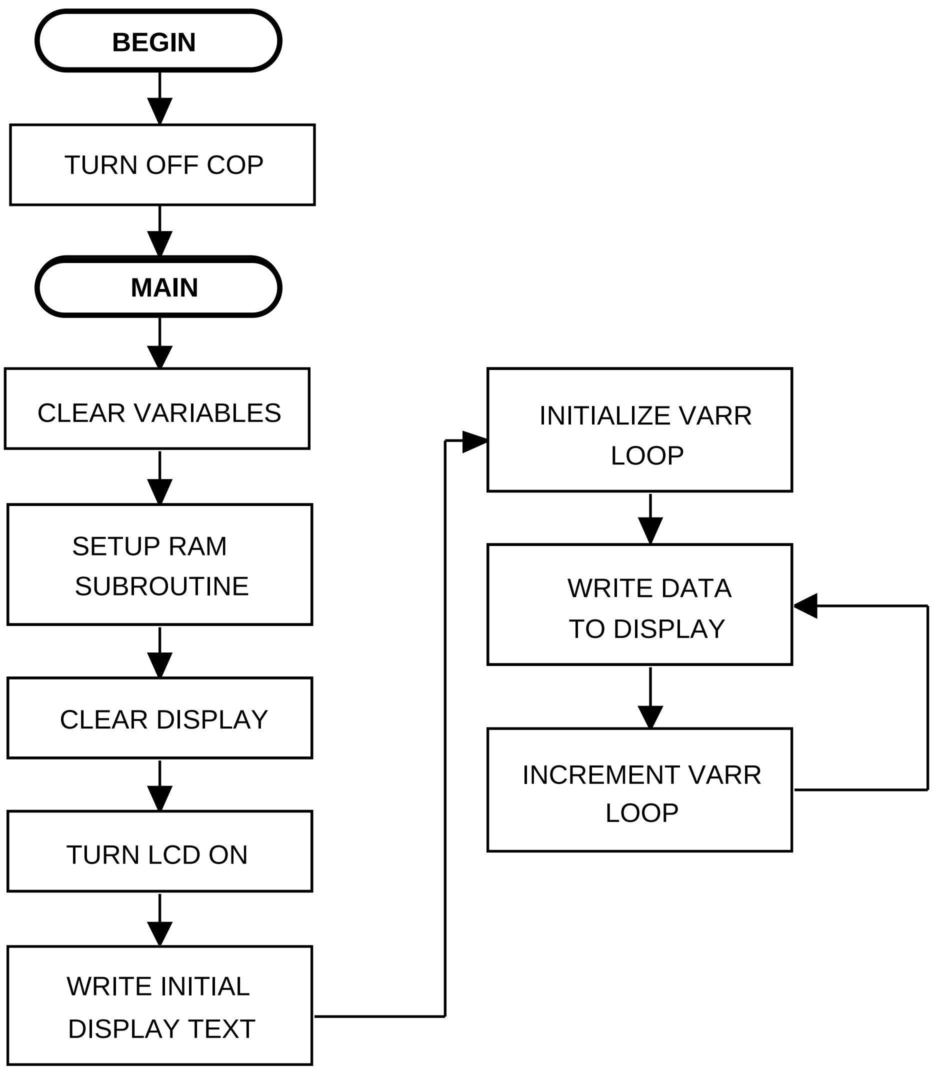
```
**  CHARROM table is the physical dot-matrix representation of
**  each character in the ASCII table (from $00-$7f)
**  It is arranged in order of its position in the
**  ASCII table, so that cross referencing is done easily.
**  Values higher than $7F are used for custom characters.
**  The Motorola logo has been included as an example.
**  Most of the data in the CHARROM table has been developed
**  at Nortel and has been used in this table with Nortel's
    permission.

CHARROM:
            FCB     $7F,$7F,$7F,$7F,$7F     ; 00
            FCB     $7F,$7F,$7F,$7F,$7F     ; 01
            FCB     $7F,$7F,$7F,$7F,$7F     ; 02
            FCB     $7F,$7F,$7F,$7F,$7F     ; 03
            FCB     $7F,$7F,$7F,$7F,$7F     ; 04
            FCB     $7F,$7F,$7F,$7F,$7F     ; 05
            FCB     $7F,$7F,$7F,$7F,$7F     ; 06
            FCB     $7F,$7F,$7F,$7F,$7F     ; 07
            FCB     $7F,$7F,$7F,$7F,$7F     ; 08
            FCB     $7F,$7F,$7F,$7F,$7F     ; 09
            FCB     $7F,$7F,$7F,$7F,$7F     ; 0A
            FCB     $7F,$7F,$7F,$7F,$7F     ; 0B
            FCB     $7F,$7F,$7F,$7F,$7F     ; 0C
            FCB     $7F,$7F,$7F,$7F,$7F     ; 0D
            FCB     $7F,$7F,$7F,$7F,$7F     ; 0E
            FCB     $7F,$7F,$7F,$7F,$7F     ; 0F
            FCB     $7F,$7F,$7F,$7F,$7F     ; 10
            FCB     $7F,$7F,$7F,$7F,$7F     ; 11
            FCB     $7F,$7F,$7F,$7F,$7F     ; 12
            FCB     $7F,$7F,$7F,$7F,$7F     ; 13
            FCB     $7F,$7F,$7F,$7F,$7F     ; 14
            FCB     $7F,$7F,$7F,$7F,$7F     ; 15
            FCB     $7F,$7F,$7F,$7F,$7F     ; 16
            FCB     $7F,$7F,$7F,$7F,$7F     ; 17
            FCB     $7F,$7F,$7F,$7F,$7F     ; 18
            FCB     $7F,$7F,$7F,$7F,$7F     ; 19
            FCB     $7F,$7F,$7F,$7F,$7F     ; 1A
            FCB     $7F,$7F,$7F,$7F,$7F     ; 1B
            FCB     $7F,$7F,$7F,$7F,$7F     ; 1C
            FCB     $7F,$7F,$7F,$7F,$7F     ; 1D
            FCB     $7F,$7F,$7F,$7F,$7F     ; 1E
            FCB     $7F,$7F,$7F,$7F,$7F     ; 1F
            FCB     $00,$00,$00,$00,$00     ; 20 <SPACE>
            FCB     $00,$00,$5F,$00,$00     ; 21 !
            FCB     $00,$06,$00,$06,$00     ; 22 "
            FCB     $14,$7F,$14,$7F,$14     ; 23 #
            FCB     $04,$2A,$6D,$2A,$10     ; 24 $
            FCB     $27,$16,$08,$34,$32     ; 25 %
            FCB     $20,$56,$49,$36,$50     ; 26 &
            FCB     $00,$03,$05,$00,$00     ; 27 `
```

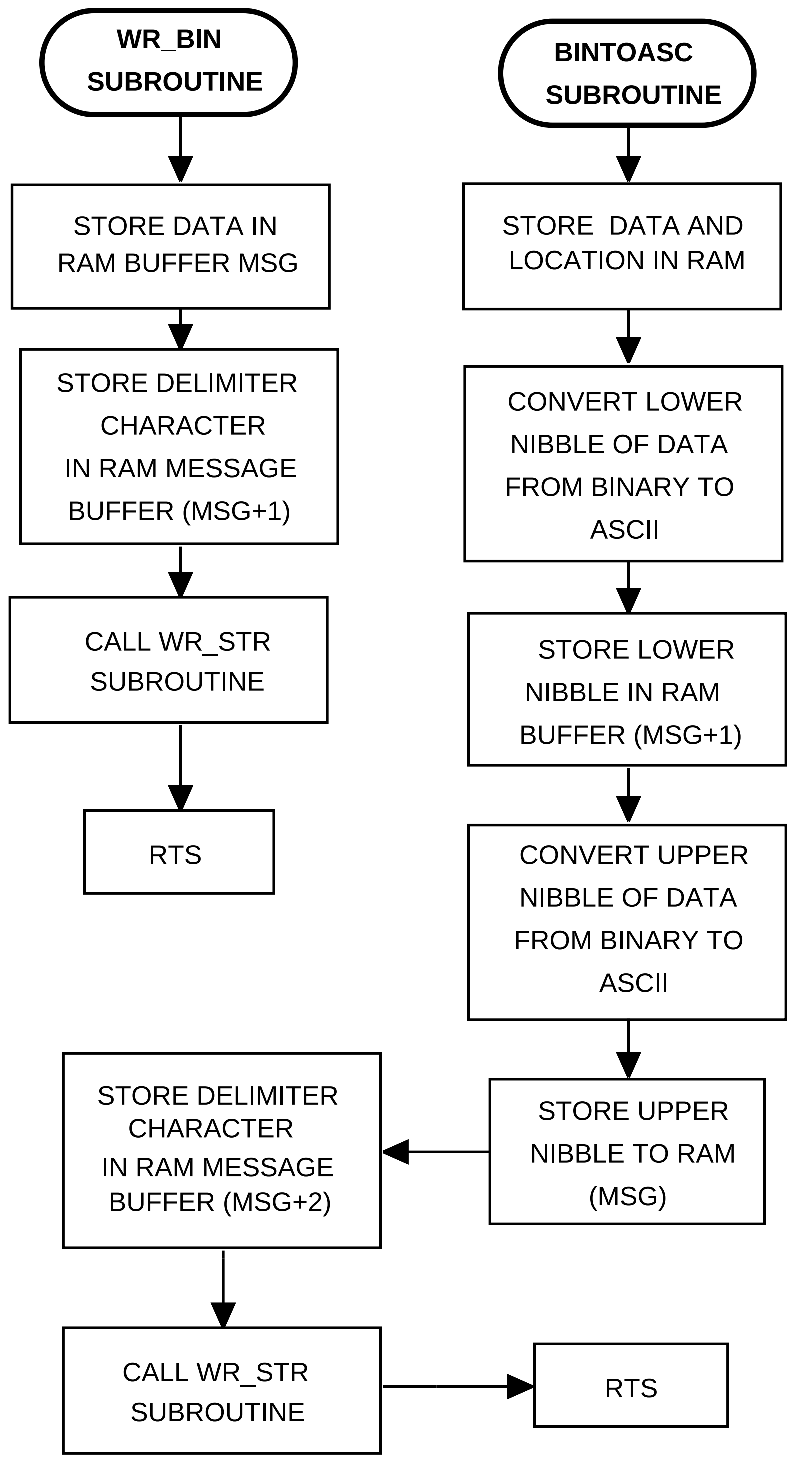
```
FCB     $00,$00,$1D,$22,$41     ; 28 (
FCB     $41,$22,$1D,$00,$00     ; 29 )
FCB     $14,$08,$3E,$08,$14     ; 2A *
FCB     $20,$56,$49,$36,$50     ; 2B +
FCB     $00,$50,$30,$00,$00     ; 2C ,
FCB     $08,$08,$08,$08,$08     ; 2D -
FCB     $00,$30,$30,$00,$00     ; 2E .
FCB     $20,$10,$08,$04,$02     ; 2F /
FCB     $3E,$51,$49,$45,$3E     ; 30 0
FCB     $00,$42,$7F,$40,$00     ; 31 1
FCB     $42,$61,$51,$49,$46     ; 32 2
FCB     $21,$41,$45,$4B,$31     ; 33 3
FCB     $18,$14,$12,$7F,$10     ; 34 4
FCB     $27,$45,$45,$45,$39     ; 35 5
FCB     $3C,$4A,$49,$49,$30     ; 36 6
FCB     $01,$71,$09,$05,$03     ; 37 7
FCB     $36,$49,$49,$49,$36     ; 38 8
FCB     $06,$49,$49,$29,$1E     ; 39 9
FCB     $00,$36,$36,$00,$00     ; 3A :
FCB     $00,$56,$36,$00,$00     ; 3B ;
FCB     $08,$14,$22,$41,$00     ; 3C <
FCB     $14,$14,$14,$14,$14     ; 3D =
FCB     $00,$41,$22,$14,$08     ; 3E >
FCB     $02,$01,$51,$09,$06     ; 3F ?
FCB     $3E,$41,$4D,$4D,$06     ; 40 @
FCB     $7E,$11,$11,$11,$7E     ; 41 A
FCB     $7F,$49,$49,$49,$36     ; 42 B
FCB     $3E,$41,$41,$41,$22     ; 43 C
FCB     $7F,$41,$41,$22,$1C     ; 44 D
FCB     $7F,$49,$49,$49,$41     ; 45 E
FCB     $7F,$09,$09,$09,$01     ; 46 F
FCB     $3E,$41,$49,$49,$7A     ; 47 G
FCB     $7F,$08,$08,$08,$7F     ; 48 H
FCB     $00,$41,$7F,$41,$00     ; 49 I
FCB     $20,$40,$41,$3F,$01     ; 4A J
FCB     $7F,$08,$14,$22,$41     ; 4B K
FCB     $7F,$40,$40,$40,$40     ; 4C L
FCB     $7F,$02,$0C,$02,$7F     ; 4D M
FCB     $7F,$04,$08,$10,$7F     ; 4E N
FCB     $3E,$41,$41,$41,$3E     ; 4F O
FCB     $7F,$09,$09,$09,$06     ; 50 P
FCB     $3E,$41,$51,$21,$5E     ; 51 Q
FCB     $7F,$09,$19,$29,$46     ; 52 R
FCB     $46,$49,$49,$49,$31     ; 53 S
FCB     $01,$01,$7F,$01,$01     ; 54 T
FCB     $3F,$40,$40,$40,$3F     ; 55 U
FCB     $1F,$20,$40,$20,$1F     ; 56 V
FCB     $3F,$40,$38,$40,$3F     ; 57 W
FCB     $63,$14,$08,$14,$63     ; 58 X
FCB     $07,$08,$70,$08,$07     ; 59 Y
FCB     $61,$51,$49,$45,$43     ; 5A Z
```

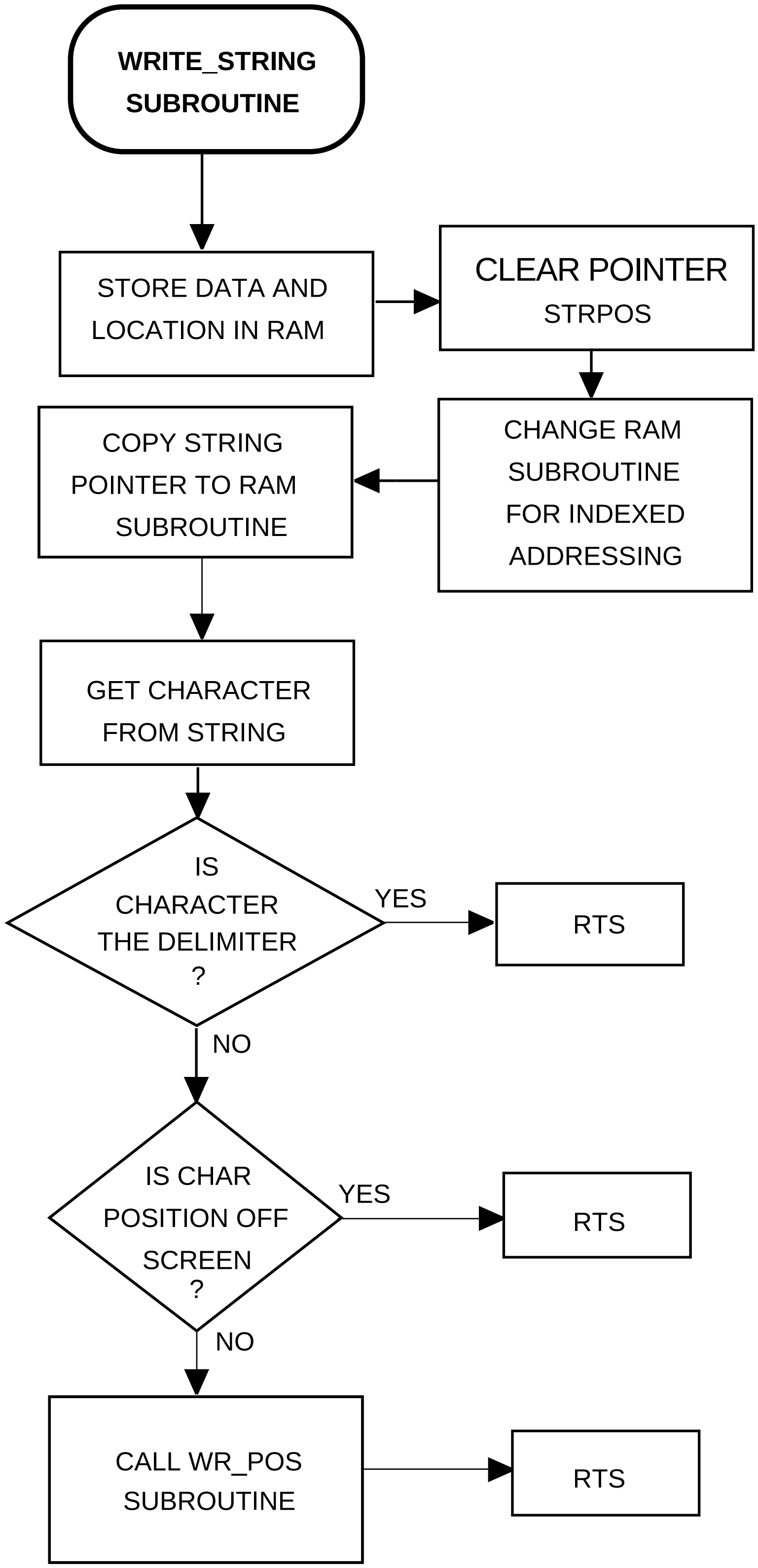
```
FCB     $00,$7F,$41,$41,$00     ; 5B [
FCB     $02,$04,$08,$10,$20     ; 5C \
FCB     $00,$41,$41,$7F,$00     ; 5D ]
FCB     $04,$02,$01,$02,$04     ; 5E ^
FCB     $02,$01,$51,$09,$06     ; 5F ?
FCB     $00,$00,$05,$03,$00     ; 60 `
FCB     $20,$54,$54,$54,$78     ; 61 a
FCB     $7F,$48,$44,$44,$38     ; 62 b
FCB     $38,$44,$44,$44,$20     ; 63 c
FCB     $38,$44,$44,$48,$7F     ; 64 d
FCB     $38,$54,$54,$54,$18     ; 65 e
FCB     $08,$7E,$09,$01,$02     ; 66 f
FCB     $04,$2A,$2A,$2A,$1C     ; 67 g
FCB     $7F,$08,$04,$04,$78     ; 68 h
FCB     $00,$44,$7D,$40,$00     ; 69 i
FCB     $20,$40,$44,$3D,$00     ; 6A j
FCB     $7F,$10,$28,$44,$00     ; 6B k
FCB     $00,$41,$7F,$40,$00     ; 6C l
FCB     $7C,$04,$18,$04,$78     ; 6D m
FCB     $7C,$08,$04,$04,$78     ; 6E n
FCB     $38,$44,$44,$44,$38     ; 6F o
FCB     $7C,$14,$14,$14,$08     ; 70 p
FCB     $08,$14,$14,$18,$7C     ; 71 q
FCB     $7C,$08,$04,$04,$08     ; 72 r
FCB     $48,$54,$54,$54,$20     ; 73 s
FCB     $04,$3F,$44,$40,$20     ; 74 t
FCB     $3C,$40,$40,$20,$7C     ; 75 u
FCB     $1C,$20,$40,$20,$1C     ; 76 v
FCB     $3C,$40,$30,$40,$3C     ; 77 w
FCB     $44,$28,$10,$28,$44     ; 78 x
FCB     $44,$64,$54,$4C,$44     ; 7A z
FCB     $7F,$7F,$7F,$7F,$7F     ; 7B
FCB     $7F,$7F,$7F,$7F,$7F     ; 7C
FCB     $7F,$7F,$7F,$7F,$7F     ; 7D
FCB     $1E,$48,$90,$50,$8E     ; 7E ~
FCB     $7F,$7F,$7F,$7F,$7F     ; 7F
FCB     $0C,$50,$50,$50,$3C     ; 79 y
FCB     $50,$08,$04,$60,$7A     ; 80 (Batwing)
FCB     $3C,$71,$41,$71,$3C     ; 81 (Batwing)
FCB     $7A,$60,$04,$08,$50     ; 82 (Batwing)
FCB     $05,$00,$17,$01,$20     ; 83 (Batwing)
FCB     $00,$40,$41,$40,$00     ; 84 (Batwing)
FCB     $20,$01,$17,$00,$05     ; 85 (Batwing)
FCB     $78,$10,$60,$10,$78     ; 86 M -TOP
FCB     $07,$00,$00,$00,$07     ; 87 M -BOT
FCB     $70,$08,$08,$08,$70     ; 88 O
FCB     $03,$04,$04,$04,$03     ; 89 O
FCB     $08,$08,$78,$08,$08     ; 8A T
FCB     $00,$00,$07,$00,$00     ; 8B T
FCB     $78,$48,$48,$48,$30     ; 8C R
FCB     $07,$00,$01,$02,$04     ; 8D R
```

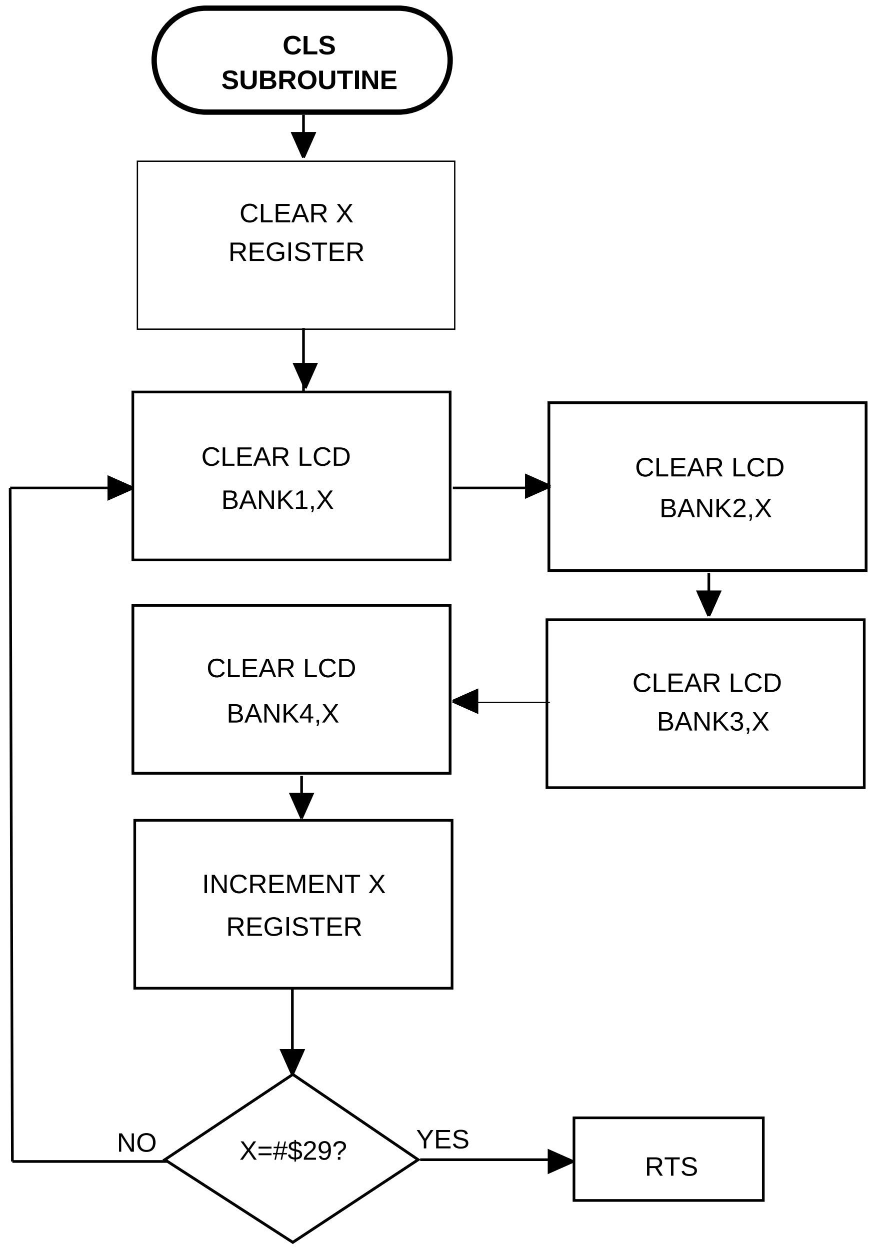
```
                FCB     $78,$00,$00,$00,$00     ; 8E L
                FCB     $07,$04,$04,$04,$04     ; 8F L
                FCB     $70,$08,$08,$08,$70     ; 90 A
                FCB     $07,$01,$01,$01,$07     ; 91 A
ENDLOC:
```

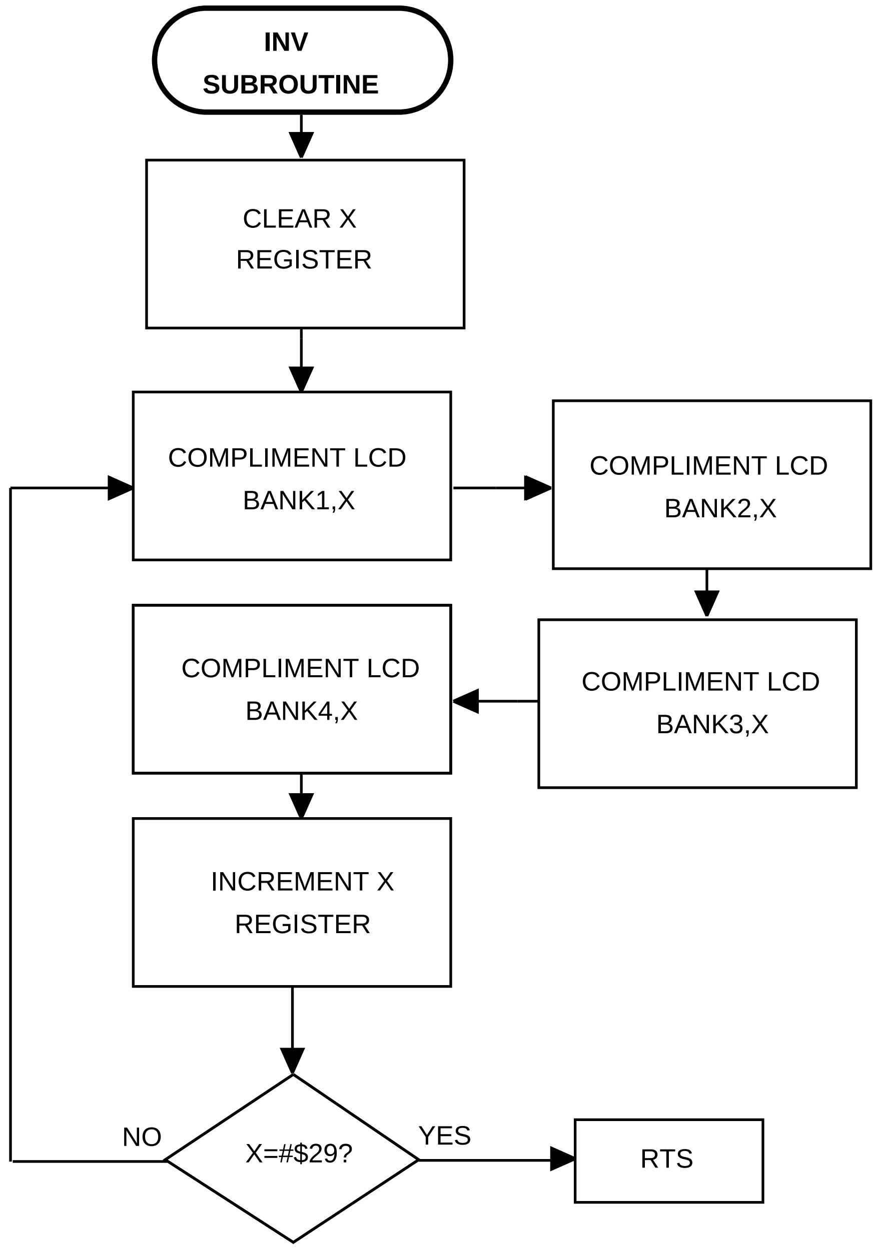
Flow Chart

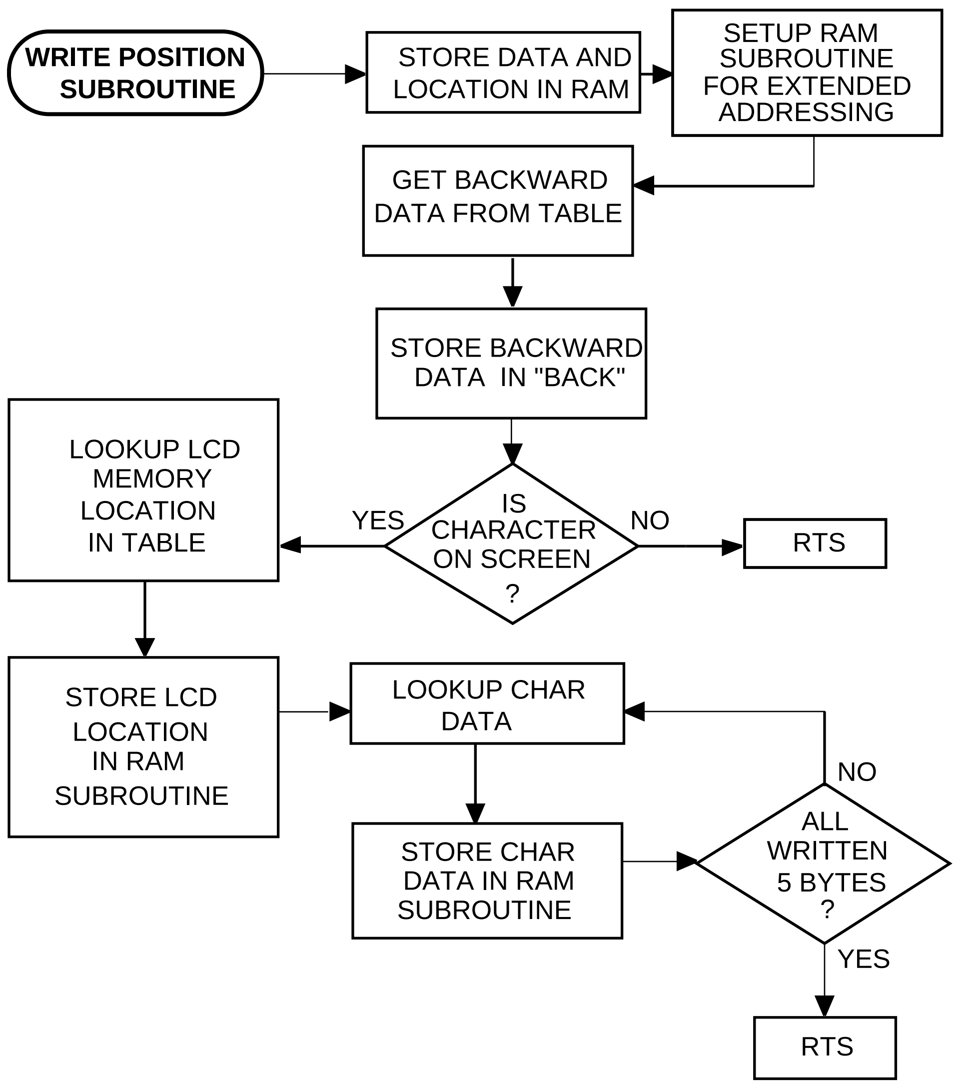

BEGIN
TURN OFF COP
MAIN
CLEAR VARIABLES
SETUP RAM SUBROUTINE
CLEAR DISPLAY
TURN LCD ON
WRITE INITIAL DISPLAY TEXT
INITIALIZE VARR LOOP
WRITE DATA TO DISPLAY
INCREMENT VARR LOOP


WR_BIN SUBROUTINE
STORE DATA IN RAM BUFFER MSG
STORE DELIMITER CHARACTER IN RAM MESSAGE BUFFER (MSG+1)
CALL WR_STR SUBROUTINE
RTS
BINTOASC SUBROUTINE
STORE DATA AND LOCATION IN RAM
CONVERT LOWER NIBBLE OF DATA FROM BINARY TO ASCII
STORE LOWER NIBBLE IN RAM BUFFER (MSG+1)
CONVERT UPPER NIBBLE OF DATA FROM BINARY TO ASCII
STORE UPPER NIBBLE TO RAM (MSG)
STORE DELIMITER CHARACTER IN RAM MESSAGE BUFFER (MSG+2)
CALL WR_STR SUBROUTINE
RTS


WRITE_STRING
SUBROUTINE
STORE DATA AND
LOCATION IN RAM
CLEAR POINTER
STRPOS
CHANGE RAM
SUBROUTINE
FOR INDEXED
ADDRESSING
COPY STRING
POINTER TO RAM
SUBROUTINE
GET CHARACTER
FROM STRING
IS
CHARACTER
THE DELIMITER
?
YES
RTS
NO
IS CHAR
POSITION OFF
SCREEN
?
YES
RTS
NO
CALL WR_POS
SUBROUTINE
RTS


CLS SUBROUTINE
CLEAR X REGISTER
CLEAR LCD BANK1,X
CLEAR LCD BANK2,X
CLEAR LCD BANK3,X
CLEAR LCD BANK4,X
INCREMENT X REGISTER
X=#$29?
NO
YES
RTS


INV
SUBROUTINE
CLEAR X
REGISTER
COMPLIMENT LCD
BANK1,X
COMPLIMENT LCD
BANK2,X
COMPLIMENT LCD
BANK3,X
COMPLIMENT LCD
BANK4,X
INCREMENT X
REGISTER
X=#$29?
NO
YES
RTS


WRITE POSITION SUBROUTINE
STORE DATA AND LOCATION IN RAM
SETUP RAM SUBROUTINE FOR EXTENDED ADDRESSING
GET BACKWARD DATA FROM TABLE
STORE BACKWARD DATA IN "BACK"
IS CHARACTER ON SCREEN ?
YES
NO
RTS
LOOKUP LCD MEMORY LOCATION IN TABLE
STORE LCD LOCATION IN RAM SUBROUTINE
LOOKUP CHAR DATA
STORE CHAR DATA IN RAM SUBROUTINE
ALL WRITTEN 5 BYTES ?
NO
YES
RTS

How to reach us:

USA/EUROPE/Locations Not Listed:

MOTOROLA LITERATURE DISTRIBUTION: P.O. Box 20912; Phoenix, Arizona 85036.1-800-441-2447 or 602-303-5454

MFAX: RMFAX0@email.sps.mot.com – TOUCHTONE (602) 244-6609

INTERNET: http://Design-NET.com

JAPAN: Nippon Motorola Ltd.; Tatsumi-SPD-JLDC, 6F Seibu-Butsuryu-Center, 3-14-2 Tatsumi Koto-Ku, Tokyo 135, Japan. 03-81-3521-8315

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park, 51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852-26629298

MOTOROLA