INTRODUCTION

It is often necessary for small microprocessor-based devices to display status information to the user. If the quantity of information is small, light emitting diodes (LEDs) can give a simple status display. However, LEDs are not practical if the volume of information gets too large. Segmented LED displays allow more information to be displayed but, LEDs consume power, shortening the life of battery-operated devices. Liquid crystal display (LCD) modules are a small, light, low-power alternative. LCD displays can be purchased that display nearly anything by using a dot matrix or a segmented display of discrete areas of liquid crystal.

The drawback to dot matrix LCD is the cost of the dedicated controller chip that is usually required to drive the LCD glass. While LCDs are not difficult to control, they are very unforgiving. They require constant attention because a voltage change across the liquid crystal can destroy the crystal structure and ruin the display. Also, typical dot-matrix display panels require a large number of signal inputs. This makes it impractical to use dot-matrix LCD displays without a dedicated driver chip. This chip can be built directly into the same case as the glass, and this combination is then called an LCD module.

Note: Application Note AP96Z8X1400, Interfacing LCDs to the Z8 (found in the Z8 Application Note Handbook), describes how LCD modules can be used with the Zilog Z8 family of microcontrollers.

LCD displays are also made in a segmented fashion, however. This type of display uses segments of liquid crystal to form characters and enunciators, in the same manner used by LED-segmented displays. Since the total number of controlled segments is lower than with dot matrix-type displays, the Z8 can take direct control of the LCD glass. This application note describes how the designer can interface directly to a simple LCD using the Z86X3X and Z86X4X family of microcontrollers.

THEORY OF OPERATION

LCD Basics

A liquid crystal display is manufactured by layering polarizing liquid crystal between two plates of glass and a polarizer. (See Figure 1.) When a voltage potential is developed across the liquid crystal, the crystalline matrix twists. The effect is that the voltage controls a polarizing filter, alternately blocking and transmitting light.

Applying the control voltage for too long a period of time causes the matrix of the liquid crystal to permanently twist, ruining the polarizing effect. To prevent this problem, the...
LCD must be pulsed—first in one direction, then in the other. The shifting effect is neutralized. The voltage is alternated quickly enough (typically 50 to 100 Hz) that the eye does not perceive the ON segment as flickering.

Traditional LCD panels were built with one backplane of glass acting as a common conductor for all the segments. Another glass plate had a conductor for each segment brought out to the edge of the panel for connection to the outside. The signals to drive this type of display are illustrated in Figure 2.

As the number of segments to be driven increases, the number of pins required on the driver chip increases proportionally.

**Multiple Backplanes**

In order to reduce the number of control lines required, for large segment counts, modern LCD display panels are usually built with more than one backplane. This is done by splitting the backplane glass into several conductors and connecting more than one segment to each control pin. Then, by placing a signal on the common pins as well as the segment pins, the segments can be toggled independently.

In this example, segment A (the top of the character) would be ON and segment G (the bar across the middle) would be OFF. The two common planes drive an alternating signal with periods of zero between each high and low drive pulse and the planes out of phase with each other. The common signal pin is then driven with the data for both pins, the data for common 1, data for common 2, inverted data for common 1 and inverted data for common 2. Figure 3c shows this sequence.

The resulting waveforms at each segment are shown at the bottom of Figure 3b. The Root Mean Squared (RMS) value of the signal on segment A is larger than the initial voltage of the liquid crystal so it appears dark while the RMS voltage across segment G is below the threshold so the segment is clear.

It is important to keep each segment toggling quickly enough to prevent noticeable flicker. The common planes must toggle twice as fast in a two-plane configuration, four times as fast in a four plane, and so forth. Obviously, as the number of backplanes goes up, the speed of the driving processor must also increase. This sets up a trade off between speed of the controller and complexity of the glass on one side and pin count on the other.
HARDWARE IMPLEMENTATION

The Application
To demonstrate the method for using the Z8 to directly drive an LCD glass, this application note implements a small, battery-operated travel alarm clock. The clock design has a single alarm setting with a snooze feature and an audible alarm.

The LCD Glass
LCD glass for this type of application is typically custom manufactured in volume. This gives the user flexibility in selecting an initial threshold voltage that matches the chosen power supply as well as explicitly defines the appearance of the segments and the number of backplanes.

For purposes of this note, the circuit is designed around an LCD that was left over from another project. It has a threshold voltage of about 1.2 volts, 13 segment lines and 2 backplanes. Using a supply voltage that can range from 3.0 volts down to 2.0 volts, the worst-case RMS voltage across an OFF segment is calculated as:

\[ V_{off} = \sqrt{\frac{(1.5^2 + 0^2)}{2}} = 1.06V \]

which is below the threshold so the segment stays clear. An ON segment's worst-case RMS voltage is calculated as:

\[ V_{on} = \sqrt{\frac{(1^2 + 2^2)}{2}} = 158\text{V} \]

which is well above the threshold. Thus, a 3-volt lithium button-cell supply works nicely.

Driving The Backplanes
Since microprocessors normally do not deal in negative output voltages, the center line of the plots is usually half the supply voltage referenced to the ground of the chip. The positive drive level is the chip supply and the negative drive level is ground. The liquid crystal is insensitive to the DC component common to both the backplane and segment lines, only the difference between them matters. This can be accomplished using the binary drive of the Z8 as shown in Figure 4.

The common signals can be generated, phase by phase, simply by driving the three port pins to the correct state. For example, to generate phase zero, pin P35 would be driven HIGH and pins P34 and P37 driven LOW. The common 1 voltage is then LOW while the common 2 voltage is set by the resistive divider from Voh to Vol. The OFF state can be accomplished by driving all three pins LOW and driving all the segments LOW. Since the common mode DC component is ignored by the LCD glass, this is a safe state. The segment drivers can simply be connected directly to port pin outputs since they only need to drive a HIGH or a LOW.

This method allows up to 3 backplanes (an uncommon but feasible number) times 24 segment lines for a total of 72 segments, including enunciators. This maximum case leaves only four inputs (port 3, lower nibble) and no free outputs. An alternative is available if more segments are required. Figure 5 shows how port 2 can be used as the common driver using fixed resistor dividers.

Using this method allows up to 8 back planes times 20 segment lines for a total of 160 segments. This configuration is speed limited, however, and the OFF state continues to draw current through the dividers unless
external circuitry is added to deactivate the power. There is also a penalty in software complexity if not all of port 2 is used for plane drivers, and the extra pins are used for outputs.

The circuit described in this application note uses the first method of generating the common backplane signals. The number of segments available from a two backplane solution is sufficient. (In fact, the Z86L33, 28-pin device is enough.) The complete schematic is shown on the next page in Figure 6.

The User Interface

Aside from the LCD glass itself, the user interface consists of a Piezo buzzer to generate the alarm sound, an optional LED backlight, five buttons used for setting the time, setting the alarm, the snooze bar and the backlight, and the power switch.

The power switch does not actually deactivate the power since the Z8 must keep running to update the real-time clock. Instead, the power switch is an input to the Z8. When the switch is in the OFF position, the Z8 shuts off the LCD, and ignores the buttons. The reduced software load lets the Z8 be in HALT mode a higher percentage of the time, saving current. The switch is a break-before-make slider built into the clam-shell style case. When the case is closed, it automatically turns the switch OFF.

The LED backlight for the LCD is actually not directly driven by the button input. The Z8 MCU has control of the LEDs, allowing it to be disabled when the power is OFF and allowing the light to stay on for a few seconds after the button is released. The backlight is optional because it draws significant power from the batteries. A second battery can easily be added to supply the backlight. This battery could also be a higher voltage to further improve battery life or allow a different type of backlight.

The Piezo buzzer is driven by a hardware timer using timer-out mode. This minimizes the software requirement. The buzzer is tied between Vcc and the P36 pin with a small resistor in series to reduce the in-rush current when the pin toggles.

To protect the circuit from a reversed battery condition, a diode is placed from the ground to the Vcc pin of the Z8. If the battery is inserted incorrectly, the diode prevents the Vcc from going more than 0.7 volts below ground, quickly discharging and destroying the battery. The more common method of placing the diode in line between the battery and the Vcc pin has the drawback of reducing the Vcc voltage at any given battery voltage. Often, the battery still has some energy left when the Vcc gets too low to work correctly.
Figure 8. LCD Direct Drive Demo Circuit
SOFTWARE IMPLEMENTATION

The LCD Driver

The heart of the application is the LCD direct drive software, of course. The LCD drive is based on a timer interrupt that runs every 10 ms (100-Hz plane drive frequency.) This timer interrupt must occur on time since any deviation causes a net DC voltage to be applied to the liquid crystal. For this reason, the timer interrupt always has priority over the other sections of the code. Also, since math can cause a variable execution length, all the math for the LCD service is performed in advance. Immediately after the timer interrupt is acknowledged, the new data is copied out to the port pins. The service routine then sets about calculating the data for the next interrupt. This ensures that the only variable in the placement of edges at the LCD pins is the interrupt latency.

The LCD and real time clock are driven by timer T1. When a timer interrupt is issued, the contents of the registers are copied to the ports. The Z8 then performs the math required to set up the next phase.

The current phase is set by the values of P37 and an offset holding register, PHASE_PTR. The value of PHASE_PTR switches from 1 to 0 at each cycle, and points to the data to be sent to plane 1 or plane 2. As described in the first section, the value of P37 causes inversion on the common planes at alternating cycles.

The common plane voltages are generated by using the XOR function to flip the appropriate pins for each cycle. The current value of the port 3 outputs are stored in an image register to ensure that the XOR function reads valid data levels and to allow the next plane state to be set up on the prior cycle. The next state is simply created by taking the XOR of the current value with a number that represents the pins that should flip for this cycle. The number is then updated to change the pins that flip for the next cycle. Because the pins in question are P34, P35 and P37, the magic numbers are 0x30 and 0x80, alternately. The easiest way to flip the number between 0x30 and 0x80 is by alternately adding 0x50 (80 decimal) and 0xB0 (-80 decimal.) Storing the adder value in a register results in the sign flipping for each cycle just by taking its two’s complement (COM and then INC.)

The current value of the port 3 outputs are stored in an image register to ensure that the XOR function reads valid data levels and to allow the next plane state to be set up on the prior cycle. The next state is simply created by taking the XOR of the current value with a number that represents the pins that should flip for this cycle. The number is then updated to change the pins that flip for the next cycle. Because the pins in question are P34, P35 and P37, the magic numbers are 0x30 and 0x80, alternately. The easiest way to flip the number between 0x30 and 0x80 is by alternately adding 0x50 (80 decimal) and 0xB0 (-80 decimal.) Storing the adder value in a register results in the sign flipping for each cycle just by taking its two’s complement (COM and then INC.)

LCD Data Manipulation

To display any given combination of segments, all the software has to do is load the set of data registers with the correct numbers. Because the data is being inverted for two out of the four cycles, it is important that the data be put into the registers at the correct time. To ensure this, the UPDATE_DISPLAY routine takes data from a set of holding registers and waits for the interrupt service to tick as many times as necessary until the LCD is back to phase 0. Then the data is copied into the data registers. This means the programmer has only to LOAD the holding registers and CALL the update routine.

For simplicity, the data in the program is stored as a binary (or BCD) value. It is then easy to use that number as an offset into a lookup table of seven segment display characters. The difficulty comes from having two backplanes for the LCD. The three-and-a-half digits of the clock are split across the two planes. No two characters have their segments split up in exactly the same way. So, some manipulation is required to format the seven-segment data correctly for the LCD. The UPDATE_HOURS and UPDATE_MINUTES routines handle this chore. It would also be possible to create lookup tables for each digit separately, preformatted for the LCD, and then use a series of OR operations in the correct order to get all the data bytes required. This would use some extra ROM for the lookup tables but would run faster. It may be an option to consider in a speed-limited application, especially if ROM space is not at a premium.

The Real Time Clock

Real Time Clock (RTC) applications are fairly common today with most appliances having clocks on the front panel. While it is possible to use a dedicated clock chip for the time keeping, it is often cheaper and easier to do it in software.

Note: Zilog Z8 Application Note number AP96Z8X1100 (found in the Z8 Application Note Handbook) describes two methods for generating an RTC in software on the Z8 MCU family. This application is similar to the crystal method shown there.

One interesting item in the RTC code is the use of the DISP_HOURS and DISP_MINS pointers. In order to simplify switching the display from the current time to the alarm setting or the snooze-timer setting, these registers point to the actual location of the time to display. The pointer is used to make a copy of the time registers prior to doing the manipulation needed before it can be displayed.

Button Inputs

All of the buttons except the backlight button are interrupt driven. The button routines are set up such that they can be interrupted by the LCD timer. In fact, the 10 ms LCD timer is used to create a 50 ms debounce delay after a button is pressed. Three of the four buttons have at least two modes of operation. The Alarm button serves to make the clock display the alarm time setting. It also toggles the alarm ON and OFF and shuts off the alarm buzzer. The snooze bar, similarly, causes the clock to display the snooze time-out setting and causes the alarm to go into snooze mode if pressed while the buzzer is sounding. The Advance button does nothing by itself but, when pressed
while one of the other three buttons is held, causes the displayed time to increment. If the Advance button is depressed continuously, the rate of change accelerates. The exception, the Clock Set button, serves only to set the clock time and only functions in conjunction with the Advance button.

The backlight button is not interrupt driven. It is sampled once each 10-ms period, after the LCD is updated. If pressed, the LED backlight is activated and a two-second timer is loaded. If the button is still down at the next sampling, the counter is reloaded. When the button is released, the counter starts decrementing. When it reaches zero, the LEDs are extinguished.

The complete software listing is appended below.
Direct Drive of LCD Displays

; LCD Pin:  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15
;   PL0: COM1  x  COL D2  E2  G2  C2  E3  G3  C3  E4  G4  C4  D4  Z
;   PL1:  x COM2  P  BC1 F2  A2  B2  F3  AD3 B3  F4  A4  B4  BEL AL
;
;
;---------------------------------------------------------------------
;
; Defines

LCD_group    .EQU    %00
PHASE_PTR    .EQU    R4
SWITCH_PLANE .EQU    R5
NEXT_PLANE   .EQU    R6
P0_DATA1     .EQU    R8
P0_DATA2     .EQU    R9
P1_DATA1     .EQU    R10
P1_DATA2     .EQU    R11
P2_DATA1     .EQU    R12
P2_DATA2     .EQU    R13
GL_LIGHT_CNT .EQU    %0E
P3_COPY      .EQU    R15
GL_P3_COPY   .EQU    %0F

CLK_group    .EQU    %10
P0D1_NEXT    .EQU    R0
P0D2_NEXT    .EQU    R1
P1D1_NEXT    .EQU    R2
P1D2_NEXT    .EQU    R3
P2D1_NEXT    .EQU    R4
P2D2_NEXT    .EQU    R5
DISP_HOURS   .EQU    R6
DISP_MINS    .EQU    R7
HOURS        .EQU    R8
GL_HOURS     .EQU    CLK_group+8
BLANK_HOURS  .EQU    %28          ; CAUTION!
MINUTES      .EQU    R9
GL_MINUTES   .EQU    CLK_group+9
BLANK_MINS   .EQU    %29          ; CAUTION!
HALF_SECS    .EQU    R10
HUNDRETHS    .EQU    R11
ALARM_HOURS  .EQU    R12
GL_A_HOURS   .EQU    CLK_group+12
ALARM_MINS   .EQU    R13
GL_A_MINS    .EQU    CLK_group+13
SNOOZE_MINS  .EQU    R14
CLK_STATUS   .EQU    R15
GL_CLK_STATUS .EQU    CLK_group+15

; CLK_STATUS bit masks
TIME_SET .EQU 00000001B
AM_PM .EQU 00000010B
ALARM_AM_PM .EQU 00000100B
SETTING .EQU 00001000B
POWER .EQU 00010000B
SNOOZE .EQU 00100000B
ALARMING .EQU 01000000B
ALARM_ON .EQU 10000000B

WORK_group .EQU %20
SCRATCH0 .EQU WORK_group
SCRATCH1 .EQU WORK_group+1
SCRATCH2 .EQU WORK_group+2
SCRATCH3 .EQU WORK_group+3
DEBOUNCE_CNT .EQU WORK_group+4
ADVANCE_CNT .EQU WORK_group+5
ALARM_TIME .EQU WORK_group+6
SNOOZE_TIME .EQU WORK_group+7
; BLANK_HOURS .EQU R8
; BLANK_MINS .EQU R9
PTR_HI .EQU R10
PTR_LO .EQU R11
TAB_PTR .EQU RR10
HOLD1 .EQU WORK_group+12
HOLD2 .EQU WORK_group+13
HOLD3 .EQU WORK_group+14
HOLD4 .EQU WORK_group+15

; Enunciator bit masks
PM_ON .EQU 00000001B
COLON_BLINK .EQU 00000001B
BEL_ON .EQU 00001000B
Z_ON .EQU 00010000B
AL_ON .EQU 00010000B
ONE_ON .EQU 00000010B

; Bit mask for backlight control
LIGHT_BIT .EQU 00100000B

; Interrupt masks
ALL-buttons .EQU 00101111B
NO_BUTTONS .EQU 00100000B
ADV_ONLY .EQU 00100010B
SET_ONLY .EQU 00100100B
CLR_BUTTONS .EQU 11110000B

; Extended register file defines
PCON .EQU %00
SMR .EQU %0B
Direct Drive of LCD Displays

WDTMR .EQU %0F

; Interrupt vector table
.ORG %00
.WORD ALARM_BUTTON ; IRQ0 (P32) ; Alarm button
.WORD ADV_BUTTON ; IRQ1 (P33) ; Advance button
.WORD SET_BUTTON ; IRQ2 (P31) ; Time set button
.WORD SNOOZE_BAR ; IRQ3 (P30) ; Snooze bar
.WORD Init ; IRQ4 (T0) ; Just carrier, no IRQ
.WORD T1_SERVICE ; IRQ5 (T1) ; Master clock/LCD timer

; Start main program
.ORG %0C

; Initialize the part
Init:           DI
.WORD %310F ; SRP #%0F ; Config SMR/PCON/WDTMR
LD      WDTMR   #%13    ; Min Current
LD      SMR     #%22    ; ; Div by 1 mode
LD      PCON    #%16    ; ; Low EMI (%06 for L43)
SRP     #%00
LD      P01M    #%04    ; ; Set P0,1=out, Int Stack
LD      P2M     #%C0    ; ; Set P2=out, P26,7 = in
LD      P3M     #%01    ; Set P3=IO, Pull P2
LD      IPR     #%04    ; LCD>Snz>Tim>Alm>Adv>T0
LD      IMR     #SET_ONLY ; Enable T1 and TimeSet
CLR     SPH
LD      SPL     #%F0    ; Init Stack Pointer
CALL    CLK_INIT
CALL    LCD_INIT
EI                      ; Init IRQ
CLR     IRQ             ; P31/2 falling edge Ints
SRP     #CLK_group

MAIN:           TM      CLK_STATUS #POWER ; Is power on or off?
                JR      Z       POWER_OFF
POWER_ON:       TM      P2      #%80    ; Has switch moved?
                JR      NZ      NO_CHANGE
TURN_OFF:       CALL    DEBOUNCE        ; Discard pending buttons
                AND     CLK_STATUS #^C(Power) ; Clear power bit
                AND     CLK_STATUS #^C(ALARMING) ; Shut off alarm
                AND     TMR     #%FC    ; Shut off buzzer
                CALL    LCD_INIT ; Shut off LCD
                JR      NO_CHANGE
POWER_OFF:      TM      P2      #%80    ; Has switch moved?
                JR      Z       NO_CHANGE
TURN_ON:        CALL    DEBOUNCE        ; Discard pending buttons
                LD      IMR     #ALL_BUTTONS ; Back to normal
                OR      CLK_STATUS #POWER ; Set power bit
TM CLK_STATUS #TIME_SET ; Is the time set?
JR NZ NO_CHANGE
LD IMR #SET_ONLY ; T1 and TimeSet Only

NO_CHANGE:
EI
NOP
HALT ; Stay in HALT until timer ticks
NOP
JR MAIN

CLK_INIT:
LD PRE0 #%05 ; No prescale, Mod-N mode
LD T0 #34 ; (decimal) Generate 3.7kHz tone
SRP #CLK_group
LD HOURS #%12 ; Start at midnight
CLR MINUTES
LD HALF_SECS #120 ; (decimal)
LD HUNDRETHS #50 ; (decimal)
LD ALARM_HOURS HOURS ; Alarm time = midnight
CLR ALARM_MINS
LD CLK_STATUS #00100000B ; AM, alarm off, snooze off
LD DISP_HOURS #GL_HOURS ; Display current time
LD DISP_MINS #GL_MINUTES ; (Note: these are POINTERS)
CLR P0D1_NEXT ; Clear display and all enunciators
CLR P0D2_NEXT
CLR P1D1_NEXT
CLR P1D2_NEXT
CLR P2D1_NEXT
CLR P2D2_NEXT
CLR GL_LIGHT_CNT ; Make sure light is off
LD BLANK_HOURS #%FF
LD BLANK_MINS #%FF
LD SNOOZE_TIME #%05 ; Set minutes to snooze
LD ALARM_TIME #%05 ; Set longest alarm time
CALL UPDATE_HOURS ; Load "Next" registers
CALL UPDATE_MINS
RET

LCD_INIT:
SRP #LCD_group
AND TMR #%F3 ; Stop T1
CLR R0 ; Clear display
CLR R1
CLR R2
CLR R3
LD R4 #%00 ; Phase 0 is next
LD R5 #%30
LD R6 #%50
LD P3_COPY #%10 ; Initialize to Phase 3
CALL UPDATE_DISP;                  ; Stuff working registers
LD PRE1  #%2B                      ; 10 prescale, contin mode
   LD T1   #250                    ; (decimal) 100Hz Timer
LD TMR  #%4C                       ; Start T1 / Tout0 mode
RET

; Key debounce. Used by all four key routines.
DEBOUNCE:                     
   DI
   LD IMR  #NO_BUTTONS            ; Ignore further button IRQs
   EI
   CLR DEBOUNCE_CNT
DBNCE_LOOP:                   
   CP DEBOUNCE_CNT #%05          ; Wait 50 mS
   JR NE DBNCE_LOOP
   DI
   AND IRQ  #CLR_BUTTONS         ; Discard any buttons pending
RET

SET_BUTTON:                   
   CALL DEBOUNCE
   TM CLK_STATUS #ALARMING       ; Is the alarm ringing?
   JR Z SET_TIME
   EI
   JR SET_LOOP                   ; Wait for button release
SET_TIME:                     
   OR CLK_STATUS # (TIME_SET + SETTING)
   LD DISP_HOURS #GL_HOURS       ; Make sure time is displayed
   LD DISP_MINS #GL_MINUTES
   AND P2D1_NEXT #^C(Z_ON)       ; Turn “Z” off
   AND P2D2_NEXT #^C(AL_ON)      ; Turn “AL” off
   LD IMR #ADV_ONLY             ; Only allow ADV button or T1 IRQs
   EI
   SET_LOOP:                    
   TCM P3  #00000010B            ; Wait for button release
   JR NZ SET_LOOP
   CALL DEBOUNCE
   AND CLK_STATUS #^C(SETTING)
   LD IMR #ALL_BUTTONS          ; Back to normal
   IRET

ALARM_BUTTON:                 
   CALL DEBOUNCE
   TM CLK_STATUS #ALARM_ON       ; Is the alarm on?
   JR Z SET_ALARM
   AND P2D2_NEXT #^C(BEL_ON)     ; Turn off bell indicator
   AND P2D1_NEXT #^C(Z_ON)       ; Turn off snooze indicator
   AND CLK_STATUS #01111111B     ; Turn alarm off
   AND CLK_STATUS #^C(ALARMING)  ; Shut off alarm if ringing
   OR CLK_STATUS #SNOOZE         ; Turn off snooze mode
   AND TMR #%FC                  ; Silence alarm
   EI
Zilog Direct Drive of LCD Displays

JR ALM_LOOP

SET_ALARM:
OR CLK_STATUS #SETTING
LD DISP_HOURS #GL_A_HOURS ; Display alarm time
LD DISP_MINS #GL_A_MINS
OR P2D2_NEXT #AL_ON ; Turn "AL" on
OR P2D2_NEXT #BEL_ON ; Turn on bell indicator
OR CLK_STATUS #ALARM_ON ; Turn alarm on
LD IMR #ADV_ONLY ; Only allow ADV button or T1 IRQs
EI

ALM_LOOP:
TCM P3 #00000100B ; Wait for button release
JR NZ ALM_LOOP
CALL DEBOUNCE
LD DISP_HOURS #GL_HOURS ; Display current time
LD DISP_MINS #GL_MINUTES
AND P2D2_NEXT #^C(AL_ON) ; Turn "AL" off
AND CLK_STATUS #^C(SETTING)
LD IMR #ALL_BUTTONS ; Back to normal
IRET

SNOOZE_BAR:
CALL DEBOUNCE
OR P2D1_NEXT #Z_ON ; Turn "Z" indicator on
TM CLK_STATUS #ALARMING ; Is the alarm ringing?
JR NZ START_SNOOZE
LD DISP_HOURS #BLANK_HOURS ; Display snooze timer
LD DISP_MINS #SNOOZE_TIME
OR CLK_STATUS #SETTING ; Setting mode
LD IMR #ADV_ONLY ; Allow T1 or ADV key IRQs
EI
JR SNOOZE_LOOP

START_SNOOZE:
AND CLK_STATUS #^C(SNOOZE) ; Set snoozing mode
LD SNOOZE_MINS SNOOZE_TIME ; Init snooze counter
AND TMR #%FC ; Silence alarm
EI

SNOOZE_LOOP:
TCM P3 #00000001B ; Wait for button release
JR NZ SNOOZE_LOOP
CALL DEBOUNCE
LD DISP_HOURS #GL_HOURS ; Display time
LD DISP_MINS #GL_MINUTES
TM CLK_STATUS #SNOOZE ; Snooze mode?
JR Z SNOOZE_DONE
AND P2D1_NEXT #^C(Z_ON) ; Turn "Z" off
AND CLK_STATUS #^C(SETTING) ; Done setting
SNOOZE_DONE:
LD IMR #ALL_BUTTONS ; Back to normal
IRET
Direct Drive of LCD Displays

ADV_BUTTON: PUSH IMR
CALL DEBOUNCE
EI
TM CLK_STATUS #ALARMING
JR NZ ADV_DONE ; Do nothing if alarm ringing
TM CLK_STATUS #SETTING
JR Z ADV_DONE ; We're not in set mode, exit
LD SCRATCH0 P3
COM SCRATCH0
AND SCRATCH0 #%0F

ADV_ALARM: CP SCRATCH0 #%0C ; ADV and ALARM buttons only?
JR ADV_CLOCK

ADV_TIME: CP SCRATCH0 #%0A ; ADV and SET buttons only?
JR NE ADV_SNOOZE

ADV_CLOCK: CLR ADVANCE_CNT
CLR SCRATCH2

ADV_LOOP: CP ADVANCE_CNT SCRATCH2 ; Wait for half second tick
JR EQ ADV_LOOP
LD SCRATCH2 ADVANCE_CNT ; Save current counter

ADVANCE: TM P3 SCRATCH0 ; Either button released?
JR NZ ADV_DONE

ADD @DISP_MINS #%01 ; Add a minute (BCD)
DA @DISP_MINS ; Fix BCD
CP @DISP_MINS #%60 ; Roll minutes?
JR NE ADV_UPDT
CLR @DISP_MINS ; Reset minutes
ADD @DISP_HOURS #%01 ; Add an hour (BCD)
DA @DISP_HOURS ; Fix BCD
CP @DISP_HOURS #%12 ; Roll hours?
JR LE NO_ROLL_ADV
LD @DISP_HOURS #%01 ; Roll the hours

NO_ROLL_ADV: JR NE ADV_UPDT
XOR CLK_STATUS SCRATCH0 ; Toggle appropriate AM_PM
XOR CLK_STATUS #SETTING ; Fix SETTING bit

ADV_UPDT: CALL UPDATE_CLK ; (Leaves INTs disabled)
EI
CP ADVANCE_CNT #%10 ; Go into fast mode (>15 INCs)
JR ULT ADV_LOOP

FASTMODE: CALL DEBOUNCE ; (Just to wait the 50mS)
EI
LD ADVANCE_CNT #%10 ; (To prevent roll over to 00h)
JR ADVANCE
ADV_SNOOZE: CP SCRATCH0 #%09 ; ADV and SNOOZE buttons only?
JR NE ADV_DONE

ADV_SNOOZE1: CLR ADVANCE_CNT

ADV_SNZ_LOOP: CP ADVANCE_CNT #%000 ' ; Wait for half second tick
JR Z ADV_SNZ_LOOP
ADD SNOOZE_TIME #%01 ; Inc snooze limit (BCD)
DA SNOOZE_TIME
CP SNOOZE_TIME #%31 ; 30 mins max
JR NE NO_ROLL_SNZ
LD SNOOZE_TIME #%01

NO_ROLL_SNZ: CALL UPDATE_CLK ; Leaves Ints disabled
EI
TM P3 #%09 ; Either button released?
JR Z ADV_SNOOZE1

ADV_DONE: TM P3 #00001000B ; Wait for button release
JR Z ADV_DONE
CALL DEBOUNCE
POP IMR
IRET

T1_SERVICE: ; This INT takes care of the LCD refresh, it must be on
; time to prevent DC offset.
SRP #LCD_group
TM GL_CLK_STATUS #POWER ; Do not update LCD if power is off
JR Z LCD_OFF
LD R0 %08(R4) ; R8,9 hold P0’s D1 and D2 resp.
LD R1 %0A(R4) ; R10,11 “ P1 “ “ “ “
LD R2 %0C(R4) ; R12,13 “ P2 “ “ “ “
LD R3 P3_COPY

CHK_LIGHT: TM P2 #%40 ; Test P26 input
JR NZ LCD_OFF
LD GL_LIGHT_CNT #%04 ; Force light for 1 sec

LCD_OFF: XOR P3_COPY R5 ; Update Plane outputs
XOR R4 #%01 ; Switch D pointer
JR NZ SKIPCOMP ; Only invert every other time
COM R8 ; Invert the data for next phase
COM R9

; COM R10
; COM R11
XOR R12 #%1F; (Only lower 5 bits of P2 used)
XOR R13 #%1F

SKIPCOMP: ADD R5 R6 ; Update Plane modifier
COM R6 ; Switch sign (+50h / -50h)
INC R6
IRET

CLK_TICK:   SRP  #CLK_group
; each 10mS
INC  DEBOUNCE_CNT
DJNZ  HUNDRETHS CLK_EXIT  ; Count 100ths of a second
LD  HUNDRETHS #50  ; (decimal)
; each half second
AND  P2D1_NEXT #^C(LIGHT_BIT)
AND  P2D2_NEXT #^C(LIGHT_BIT)  ; Turn off the backlight
SRA  GL_LIGHT_CNT
JR  Z  LIGHT_OFF
OR  P2D1_NEXT #LIGHT_BIT
OR  P2D2_NEXT #LIGHT_BIT  ; Turn on the backlight
LIGHT_OFF:
INC  ADVANCE_CNT
TM  CLK_STATUS #TIME_SET  ; See if the time is set
JR  NZ  TICK1
XOR  DISP_HOURS #%30  ; Blink the display...
XOR  DISP_MINS #%30
JR  UPDATE_CLK  ; and don’t increment time
TICK1:
XOR  P0D1_NEXT #COLON_BLINK  ; Toggle colon bit
TCM  CLK_STATUS #(ALARMING + SNOOZE)  ; Alarming & not snooze?
JR  NZ  NOT_ALARMING
XOR  TMR  #00000010B  ; Toggle the buzzer on/off
NOT_ALARMING:
DJNZ  HALF_SeCS UPDATE_CLK  ; Count half seconds
LD  HALF_SeCS #120  ; (decimal)
; each minute
ADD  MINUTES #%01  ; BCD so add, not inc
DA  MINUTES
CP  MINUTES #%60  ; BCD
JR  NE  CHK_ALARM
CLR  MINUTES
; each hour
ADD  HOURS #%01  ; BCD so add, not inc
DA  HOURS
CP  HOURS #%12  ; BCD
JR  LE  NOT_NOON
LD  HOURS #%01
NOT_NOON:JR  NE  CHK_ALARM
XOR  CLK_STATUS  #AM_PM
; each minute
CHK_ALARM:  TM  CLK_STATUS #POWER  ; Skip alarm if power off
JR  Z  UPDATE_CLK
CALL  CHECK_ALARM
; each half second
UPDATE_CLK: CALL  UPDATE_HOURS
CALL  UPDATE_MINS
CALL UPDATE_DISP ; Write the new time data

CLK_EXIT:
RET

CHECK_ALARM:
TM CLK_STATUS #ALARM_ON ; Alarm on?
JR Z ALARM_DONE
TM CLK_STATUS #ALARMING ; Alarm ringing?
JR Z CHECK_TIME
TCM CLK_STATUS #SNOOZE ; Snooze bar counter running?
JR Z NOT_SNOOZING
DJNZ SNOOZE_MINS ALARM_DONE ; Subtract one snooze minute
AND P2D1_NEXT #^C(Z_ON) ; Turn “Z” off
LD ALARM_TIME #%05 ; Reset max alarm time
JR SND_ALARM ; “Get up you bum!”

CHECK_TIME:
LD HOLD4 CLK_STATUS
RR HOLD4 ; Align alarm AM/PM with time AM/PM
XOR HOLD4 CLK_STATUS ; Compare
AND HOLD4 #AM_PM ; Ignore other bits
JR NZ ALARM_DONE ; Same?
CP HOURS ALARM_HOURS
JR NE ALARM_DONE
CP MINUTES ALARM_MINS
JR NE ALARM_DONE

SND_ALARM:
OR TMR #%03 ; Sound the alarm
OR CLK_STATUS #(ALARMING + SNOOZE) ; Set ALARMING and SNOOZE bits
ALARM_DONE:
RET

NOT_SNOOZING:
DEC ALARM_TIME
JR NZ ALARM_DONE
AND TMR #%FC ; Shut off buzzer
AND CLK_STATUS #^C(ALARMING) ; Not alarming
RET

UPDATE_HOURS:
AND P0D1_NEXT #%E1 ; Blank out the hours digits
AND P0D2_NEXT #%E0
TCM @DISP_HOURS #%FF ; See if hours should be blank
JR Z END_UPD_HOURS
LD HOLD4 CLK_STATUS
CP DISP_HOURS #GL_A_HOURS
JR NE NO_SHIFT
RR HOLD4

NO_SHIFT:
TM HOLD4 #AM_PM ; See if it’s AM or PM
JR Z ITS_AM
OR P0D2_NEXT #PM_ON ; Turn on the PM enunciator

ITS_AM:
LD HOLD4 @DISP_HOURS
TM HOLD4 #%F0 ; See if there’s a leading one
OR      P0D2_NEXT #ONE_ON ; Turn on the leading one

NO_ONE:
CALL    GET_DISP
RL      HOLD4 ; Line up bit positions
OR      P0D1_NEXT HOLD4
RL      HOLD3
RL      HOLD3
OR      P0D2_NEXT HOLD3

END_UPD_HOURS:  RET

UPDATE_MINS:
AND     P0D1_NEXT #%1F ; Blank out the minutes digits
AND     P0D2_NEXT #%1F
AND     P2D1_NEXT #%F0
AND     P2D2_NEXT #%F8
TCM     @DISP_MINS #%FF ; See if minutes should be blank
JR      Z       END_UPD_MINS
LD      HOLD4   @DISP_MINS
CALL    GET_DISP
RCF
RRC     HOLD4           ; Move D bit into carry
JR      NC      D_NOT_SET
OR      HOLD4   #%08    ; Put D bit into new position

D_NOT_SET:
OR      P2D1_NEXT HOLD4
OR      P2D2_NEXT HOLD3
AND     HOLD2   #%0E    ; Drop off D segment bit
SWAP    HOLD2           ; Align nibbles
OR      P0D1_NEXT HOLD2
RL      HOLD1           ; Align bits
SWAP    HOLD1
OR      P0D2_NEXT HOLD1

END_UPD_MINS:   RET

UPDATE_DISP:
EI                      ; Make sure LCD can interrupt
DI
TM      GL_P3_COPY #%80 ; Wait for true data state
JR      NZ      UPDATE_DISP
LD      %08     %10     ; New data for P0 D1
LD      %09     %11     ; P0 D2
LD      %0A     %12     ; P1 D1
LD      %0B     %13     ; P1 D2
LD      %0C     %14     ; P2 D1
LD      %0D     %15     ; P2 D2
RET

GET_DISP: ; Takes a packed BCD byte in R15 and returns with
; the corresponding digit nibbles in RR12 and RR14
PUSH    RP
Zilog Direct Drive of LCD Displays

SRP     #WORK_group
LD      R14  R15     ; Packed BCD, 2 digits to display
SWAP    R15
AND     R14  #%0F
AND     R15  #%0F
LD      R10  #^HB(TABLE)
LD      R11  #^LB(TABLE)
ADD     R11  R15     ; Add digit as table offset
ADC     R10  #%00     ; (carry into upper byte)
LDC     R12  @RR10   ; 10s digit code into R12
LD      R13  R12
SWAP    R12
AND     R12  #%0F
AND     R13  #%0F
LD      R10  #^HB(TABLE)
LD      R11  #^LB(TABLE)
ADD     R11  R14     ; Add digit as table offset
ADC     R10  #%00     ; (carry into upper byte)
LDC     R15  @RR10   ; 1s code into R15
LD      R14  R15
SWAP    R14
AND     R14  #%0F
AND     R15  #%0F
POP     RP
RET

TABLE:          ; Digit lookup table
    .BYTE  01111011B       ; 0
    .BYTE  01001000B       ; 1
    .BYTE  01100111B       ; 2
    .BYTE  01101101B       ; 3
    .BYTE  01011100B       ; 4
    .BYTE  00111101B       ; 5
    .BYTE  00111111B       ; 6
    .BYTE  01101000B       ; 7
    .BYTE  01111111B       ; 8
    .BYTE  01111100B       ; 9
    .BYTE  01111110B       ; A
    .BYTE  00011111B       ; b
    .BYTE  00110011B       ; C
    .BYTE  01001111B       ; d
    .BYTE  00110111B       ; E
    .BYTE  00110110B       ; F

.END