```
; University of Illinois at Chicago, Dept. of Electrical and Computer Engineering
; ECE 367 -Microprocessor-Based Design
; Semester: Spring 2013

; Experiment Title: Electronic Multi-Meter - A Voltage, Temperature, and Light Meter
; Experiment Description: This product is a voltage, temperature, and a light meter.
;                         It takes analog data from a transducer and converts it
to
;                         digital data used for output.  It can read voltages
from
;                         1 to 5, temperatures from 0C to 100C (Fahrenheit as
well),
;                         and light in 5 areas.  The transducer output is shown
at all
;                         times and can be switched from binary to decimal or
hex!
; Date: 4/4/2013
; Updated: 4/9/2013
; Version: 1
; Programmer: Mitchell Hedditch
; Lab Session: Tuesday 8AM-10:50AM
; Programming Notes:
;    1. STORE ALL VALUES IN VARIABLES (EVEN WHEN PASSING TO OTHER ROUTINES)
;    2. COMMENT ALL BLOCKS OF CODE
;    3. ORDER SUBROUTINES IN ORDER OF FIRST USED


; Define symbolic constants
REGBAS          EQU $0000                       ; REGISTER BLOCK STARTS AT $0000
PortA           EQU $0000                       ; PortA address (relative to Regbase
i.e. offset)
DDRA            EQU $0002                       ; PortA Data Direction control register
offset
PortM           EQU $0250                       ; PortM offset (actual address of
PortM)
DDRM            EQU $0252                       ; PortM Data Direction control register
offset
PortT           EQU $0240                       ; PortT offset (actual address of
PortT)
DDRT            EQU $0242                       ; Actual Data Direction Register for
PortT
PortE           EQU $0008                       ; PortE LABEL (XIRQ' INTERRUPT)
; TIMER SYMBOLIC CONSTANTS
TSCR1   EQU $0046                       ; TIMER SYSTEM CONTROL REGISTER - WITH FAST
FLAGS
TSCR2   EQU $004D                       ; TIMER SYSTEM CONTROL REGISTER 2 - NO FAST
FLAGS
TFLG1   EQU $004E                       ; TIMER INTERRUPT FLAG1 REGISTER
TFLG2           EQU $004F                       ; TIMER INTERRUPT FLAG2 REGISTER
TIOS    EQU $0040                       ; TIMER INTERRUPT OUTPUT COMPARE
TCNT    EQU $0044                       ; TIMER COUNTER REGISTER - 16 BIT, INPUT
CAPTURE/OUTPUT COMPARE REQUIRED
TC0     EQU $0050                       ; TIME I/O COMPARE SELECT 0 REGISTER TO
LOCATION $50 HEX
TC1     EQU $0052                       ; TIME I/O COMPARE SELECT 1 REGISTER TO
LOCATION $52 HEX
TIE             EQU $004C                       ; TIMER TCi INTERRUPT ENABLE REGISTER
; INTERRUPT CONSTANTS
IRQCR           EQU $001E                       ; IRQ CONTROL REGISTER ADDRESS LABEL
; SERIAL COMMUNICATION INTERFACE
SPCR1           EQU $00D8
SPCR2           EQU $00D9
SPIB            EQU $00DA
```

```
SPSR            EQU $00DB
SPDR            EQU $00DD
ENABLE   EQU $02                        ; LCD ENABLE at PM1
RCK      EQU $08          ; RCK connect to PM3
;UNKNOWN
INITRG          EQU $0011
INITRM          EQU $0010
PLLCTL   EQU $003A
; CLOCKS
CLKSEL   EQU $0039
CRGFLG   EQU $0037
SYNR     EQU $0034
REFDV    EQU $0035
COPCTL   EQU $003C                      ; COMPUTER OPERATING PROPERLY CONTROL LOCATION
; A/D CONVERSION
ATD0CTL2        EQU $0082                       ; A/D POWER
ATD0CTL3        EQU $0083                       ; # OF CONVERSIONS
ATD0CTL4        EQU $0084                       ; CLOCK SPEED FOR A/D
ATD0CTL5        EQU $0085                       ; START CONVERSION
ATD0STAT0       EQU $0086                       ; SEQUENCE COMPLETE FLAG
ATD0STAT1       EQU $008B                       ;
ATD0DR0         EQU $0090                       ; STORAGE FOR FIRST A/D CONVERSION
ATD0DR1         EQU $0092                       ; STORAGE FOR SECOND A/D CONVERSION
ATD0DR2         EQU $0094                       ; STORAGE FOR THIRD A/D CONVERSION
ATD0DR3         EQU $0096                       ; STORAGE FOR FOURTH A/D CONVERSION
; STANDARD VARIABLES
TEST            EQU $3800                       ; DEFINE LOCATION FOR TEST BYTE STORAGE
FOR DEBUGGING
DUMMY_2         EQU $3802                       ; DEFINES A DUMMY VARIABLE OF 1 BYTES
DUMMY_4         EQU $3803                       ; DEFINES A DUMMY VARIABLE OF 2 BYTES,
FOR TEMPORARY USAGE IN PROGRAM
SAVE_X          EQU $3805                       ; Defines location for the storage of
the X index register
SAVE_Y          EQU $3807                       ; Defines location for the storage of
the Y index register
; TIMER VARIABLES
TIME_COUNT      EQU $3809                       ; MEM ADDRESS TO STORE TIME FOR SECONDS
TMR_FLAG        EQU $3811                       ; DEFINES LOCATION FOR STORAGE OF TIMER
FLAG
                                                ; FLAG= 0->NOTHING; 1->TIMER FIRED
; FLAGS
INVALID_KEY     EQU $3812                       ; DEFINES LOCATION FOR STORAGE OF
INVALID KEY FLAG

;KEYPAD VARIABLES
NUM_FLAG        EQU $3813                       ; A FLAG THAT GOES TO 1 IF A KEY IS
PRESSED ON THE PAD
CUR_PAD_VAL     EQU $3814                       ; USED TO HOUSE THE VALUE FOR THE
CURRENT KEYPAD ITERATION
CUR_COLUMN      EQU $3816                       ; STORAGE LOCATION FOR VARIABLE OF
CURRENT COLUMN
; VARIABLES
MODE            EQU $3817                       ; HOLDS THE MODE VALUE FOR SYSTEM
(01=TEMP,02=VOLTAGE,03=LIGHT)
BASE_MODE       EQU $3818                       ; HOLDS THE MODE VALUE FOR SYSTEM
(01=BINARY,02=DECIMAL,03=HEX)
TEMP_MODE       EQU $3819                       ; HOLDS THE MODE VALUE FOR SYSTEM
(01=CENTIGRADE,02=FARENHEIT)
VALUE           EQU $3820                       ; VALUE HOLDS THE VALUE COMING FROM THE
A/D CONVERTER
VOLTAGE_VAL     EQU $3822                       ; VOLTAGE TABLE VALUE STORAGE
TEMP_VAL        EQU $3824                       ; TEMP TABLE VALUE STORAGE
LIGHT_VAL       EQU $3826                       ; LIGHT TABLE VALUE STORAGE
```

```
; BIT INDICATORS
RS      EQU $01         ; REGISTER SELECT (RS) AT PM0 (0=COMMAND, 1=DATA)

;*******************************************************************************
*******
; The ORG statment below is followed by variable definitions
; THIS IS THE BEGINNING SETUP CODE
;
        ORG     $3800   ; Beginning of RAM for Variables
;
; The main code begins here. Note the START Label
;
        ORG     $4000   ; Beginning of Flash EEPROM
START           LDS     #$3FC0  ; Top of the Stack
        SEI             ; Turn Off Interrupts
                MOVB    #$00, INITRG    ; I/O and Control Registers Start at $0000
        MOVB    #$39, INITRM    ; RAM ends at $3FFF
;
; We Need To Set Up The PLL So that the E-Clock = 24MHz
;
        BCLR CLKSEL,$80               ; disengage PLL from system
        BSET PLLCTL,$40               ; turn on PLL
        MOVB #$2,SYNR                 ; set PLL multiplier
        MOVB #$0,REFDV                ; set  PLL divider
        NOP             ; No OP
        NOP             ; NO OP
PLP     BRCLR CRGFLG,$08,PLP          ; while (!(crg.crgflg.bit.lock==1))
        BSET CLKSEL,$80               ; engage PLL

        CLI             ; TURN ON ALL INTERRUPTS
;
; End of  setup code. You will always need the above setup code for every experiment


;*******************************************************************************
******
; Begin Code
;*******************************************************************************
******
; Initialize the 68HC11

               LDY  #REGBAS                   ; Initialize register base address
                                              ; Note that Regbas = $0000 so now <Y> =
$0000

; INITIALIZE ALL SYSTEM PORTS/INTERRUPTS/DDRS/FLAGS/ETC
               JSR INIT                       ; INITIALIZE ALL OF OUR VARIABLES,
FLAGS, ETC.
               JSR InitLCD                    ; INITIALIZE THE LCD


;*********************************
;                               *
;      MAIN PROGRAM CODE IS HERE   *
;                               *
;*********************************
               JSR INTRODUCTION               ; DISPLAY SYSTEM INTRODUCTION
               JSR DIRECTIONS                 ; SHOW THE USER THE DIRECTIONS
               JSR DRAW_SCREEN                ; DRAW SCREEN FOR THE FIRST TIME
POLL:          MOVB #$00,INVALID_KEY          ; RESET INVALID KEY FLAG
               MOVB #$00,NUM_FLAG             ; CLEAR THE NUM FLAG TO WAIT FOR A NEW
KEY
               JSR GET_KEY                    ; CHECK THE KEYPAD FOR A PRESSED VALUE
```

```
                BRCLR NUM_FLAG,$01,NO_KEY     ; IF NO KEY HAS BEEN PRESSED THEN MOVE
ON THE THE NO_KEY LINE
                JSR CHANGE_MODE               ; CHECK TO SEE IF USER WANTS TO CHANGE
SYSTEM MODE
                JSR CHANGE_BASE_MODE          ; CHECK TO SEE IF USER WANTS TO CHANGE
NUMBER BASE MODE
                JSR CHANGE_TEMP_MODE          ; CHECK TO SEE IF USER WANTS TO CHANGE
TEMP OUTPUT UNITS
NO_KEY          BRCLR TMR_FLAG,$01,POLL       ; IF THE TIME FLAG ISN'T SET BRANCH
BACK TO POLL
                LDAA MODE                     ; LOAD THE CURRENT MODE
                CMPA #$01                     ; ARE WE IN TEMP MODE?
                BNE CV                        ; IF NOT CHECK VOLTAGE MODE
                JSR CALC_TEMP                 ; IF WE'RE IN TEMPERATURE MODE, THEN
CALCULATE THE TEMPERATURE
                BRA CONTINUE                  ; GO TO CONTINUE
CV              CMPA #$02                     ; ARE WE IN VOLTAGE MODE?
                BNE CL                        ; IF NOT, CHECK LIGHT MODE
                JSR CALC_VOLT                 ; IF WE'RE IN VOLTAGE MODE, THEN
CALCULATE THE VOLTAGE
                BRA CONTINUE                  ; GO TO CONTINUE
CL              CMPA #$03                     ; ARE WE IN LIGHT MODE?
                BNE CONTINUE                  ; IF NOT GO TO CONTINUE
                JSR CALC_LIGHT                ; IF WE'RE IN LIGHT MODE, THEN
CALCULATE THE LIGHT
CONTINUE        MOVB #$00,TMR_FLAG            ; CLEAR THE TIMER FLAG
                JSR DRAW_SCREEN               ; REDRAW HOME SCREEN (EVERY SECOND)
                BRA POLL                      ; GO BACK START PROCESSING AT POLL
AGAIN!



;*********************************
;                               *
;      SUBROUTINES BELOW        *
;                               *
;*********************************
; PROGRAM INITIALIZATION
INIT:           ; SETUP THE DATA DIRECTON REGISTERS AND INITIALIZE PORT A & PORT T
                MOVB #$F0,DDRT                ; SET PortT PINS 4-7 TO OUTBOUND AND
PINS 0-3 TO INBOUND
                MOVB #$00,PortT               ; SET ALL PortT PINS TO LOW

                ; SET UP SERIAL PROGRAM INTERFACE SYSTEM
                MOVB #$22,SPIB                ; SPI CLOCKS A 1/24 OF E-CLOCK
                MOVB #$3B,DDRM                ; SETUP PortM DATA DIRECTION
                MOVB #$50,SPCR1               ; ENABLE SPI AND SET MODE AS MASTER
                MOVB #$00,SPCR2               ; RESETS SPCR2 TO $00 (ALSO DOES AT
RESET)
                BSET PortM,RCK                ; SET RCK TO IDLE HIGH
                BCLR PortM, ENABLE            ; ENABLE to Idle LOW

                ; SET UP TIMER COUNT INFORMATION AND PRESCALE INITIALIZE THE COUNTER
                MOVB #$06,TSCR2               ; CONFIGURE PRESCALE FACTOR 64
                MOVB #$01,TIOS                ; ENABLE OC0 FOR OUTPUT COMPARE
                MOVB #$90,TSCR1               ; ENABLE TCNT & FAST FLAGS CLEAR
                MOVB #$01,TIE                 ; ENABLE TC1 INTERRUPT
                LDD TCNT                      ; FIRST GET CURRENT TCNT
                ADDD #3750                    ; INCREMENT TCNT COUNT BY 3750 AND
STORE INTO TC0
                STD TC0                       ; WE WILL HAVE A SUCCESSFUL COMPARE IN
375 CLICKS
```

```
                MOVB #$01,TFLG1                 ; OF TCNT. BETTER BE SURE FLAG C0F IS
CLEAR TO START

                ; INITIALIZE PROGRAM DEFINED VARIABLES
                MOVB #$00,CUR_PAD_VAL           ; INITIALIZE THE KEYPAD VALUE
                MOVB #$00,TMR_FLAG              ; INITIALIZE THE TIMER FLAG TO LOW
                MOVB #$00,TIME_COUNT            ; SET TIME_COUNT TO 0
                MOVB #$00,NUM_FLAG              ; SET NUM_FLAG TO 0 TO
                MOVB #$00,INVALID_KEY           ; RESET INVALID KEY FLAG
                MOVB #$01,MODE                  ; INITIALIZE MODE TO TEMPERATURE
                MOVB #$01,BASE_MODE             ; INITIALIZE BASE MODE TO DECIMAL
                MOVB #$01,TEMP_MODE             ; INITIALIZE TEMP MODE TO C
                MOVB #$00,TEMP_VAL              ; INITIALIZE THE VALUE OF TEMP
                MOVB #$00,VOLTAGE_VAL           ; INITIALIZE THE VALUE OF VOLTAGE
                MOVB #$00,LIGHT_VAL             ; INITIALIZE THE VALUE OF LIGHT
                MOVB #$00,VALUE                 ; INITIALIZE THE VALUE
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: TO RETRIEVE A PRESSED KEY FROM A MATRIX KEYBOARD, IF THIS ACTION HAPPENS,
SET A FLAG
InitLCD:        JSR delay3              ; WE NEED A SHORT DELAY HERE

        BCLR PortM,RS                   ; SEND A COMMAND
                LDAA #$30                 ; Could be $38 too, 2 LINES AND 5X7 MATRIX
        JSR LCD_INPUT                   ; OUTPUT CLEAR TO SIPO SERIALLY
                JSR delay3                ; need extra delay at startup
        LDAA #$30               ; Could be $38 too, 2 LINES AND 5X7 MATRIX
        JSR LCD_INPUT                   ; OUTPUT CLEAR TO SIPO SERIALLY
                JSR delay3                      ; WE NEED A SHORT DELAY HERE
        LDAA #$30                       ; Could be $38 too, 2 LINES AND 5X7 MATRIX
        JSR LCD_INPUT                   ; OUTPUT CLEAR TO SIPO SERIALLY
                JSR delay3                      ; GIVE US A DELAY AGAIN

                LDAA #$38                 ; Use 8 - words (command or data) and
        JSR LCD_INPUT                   ; OUTPUT CLEAR TO SIPO SERIALLY
                JSR delay3                      ; NEED SHORT DELAY TO WAIT FOR COMMAND
TO COMPLETE

                LDAA #$0C                 ; Turn on the display
        JSR LCD_INPUT                   ; OUTPUT CLEAR TO SIPO SERIALLY
                JSR delay3                      ; NEED SHORT DELAY TO WAIT FOR COMMAND
TO COMPLETE

                LDAA #$01                 ; clear the display and put the cursor
        JSR LCD_INPUT                   ; OUTPUT CLEAR TO SIPO SERIALLY
                JSR delay                 ; clear command needs more time
        JSR delay               ; to execute
        JSR delay                       ; NEED SHORT DELAY TO WAIT FOR COMMAND TO
COMPLETE
        RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: TO RETRIEVE A PRESSED KEY FROM A MATRIX KEYBOARD, IF THIS ACTION HAPPENS,
SET A FLAG
;          AND STORE THE VALUE
GET_KEY:        LDX #KP_VALUE                   ; LOAD X WITH MEM ADDRESS FOR KP_VALUE

                STX CUR_PAD_VAL                 ; STORE THE ADDRESS OF THE FIRST KEYPAD
VALUE
                LDX #ROW                        ; LOAD X WITH THE INITIAL VALUE AT THE
ROW ADDRESS
                LDY #COLUMN                     ; LOAD Y WITH THE INITIAL VALUE AT THE
```

```
COLUMN ADDRESS
                ; NOW WE BEGIN OUR LOOPING
NEXT_ROW        LDAA 1,X+                       ; LOAD ACCUM A WITH CURRENT ROW VALUE
POST INCREMENT
NEXT_COLUMN     LDAB 1,Y+                       ; LOAD ACCUM Y WITH CURRENT COLUMN
VALUE POST INCREMENT
                STAA PortT                      ; SET THE CURRENT ROW TO HIGH VALUE
                STAB CUR_COLUMN                 ; STORE THE CURRENT COLUMN VALUE
                PSHA                            ; PUSH ONTO THE STACK OR IT WILL BE
LOST
                PSHB                            ; PUSH B ONTO THE STACK OR IT WILL BE
LOST
                NOP                             ; WAIT SOME TIME FOR PIN TO GO HI
                NOP                             ; WAIT SOME TIME FOR PIN TO GO HI
                NOP                             ; WAIT SOME TIME FOR PIN TO GO HI
                ABA                             ; ADD B TO A TO GET ALL PINS THAT
SHOULD BE HIGH
                LDAB PortT                      ; LOAD THE VALUE IN PortT INTO ACCUM B
                CBA                             ; CHECK THE CURRENT BIT IN PortT TO OUR
CURRENT COLUMN
                BEQ KEY_PRESSED                 ; IF THE KEY IS PRESSED THEN MAKE IT
SO!
                LDD CUR_PAD_VAL                 ; LOAD THE CUR_PAD_VAL INTO D
                ADDD #1                         ; ADD 1 TO D
                STD CUR_PAD_VAL                 ; STORE D BACK INTO THE PAD VALUE
                PULB                            ; GET B BACK FROM THE STACK FIRST
                PULA                            ; NOW RESTORE A FROM THE STACK
                CPY #COLUMN+4                   ; CHECK TO SEE IF WE'RE AT THE END OF
THE COLUMNS
                BNE NEXT_COLUMN                 ; IF NOT, THEN GO BACK AND TRY NEXT
COLUMN
                LDY #COLUMN                     ; IF WE ARE THEN RESET THE COLUMNS
                CPX #ROW+4                      ; CHECK TO SEE IF WE'RE AT THE END OF
THE ROWS
                BNE NEXT_ROW                    ; IF WE'RE NOT AT END OF ROWS, GO TO
NEXT ROW
                RTS                             ; RETURN FROM THE SUBROUTINE IF WE'VE
PROCESS ALL ROWS AND COLUMNS
KEY_PRESSED     PULB                            ; GET B BACK FROM THE STACK FIRST
                PULA                            ; NOW RESTORE A FROM THE STACK
                MOVB #$01,NUM_FLAG              ; SET NUM_FLAG SINCE A NUMBER WAS
PRESSED
                JSR KEY_RELEASE                 ; NOW WE NEED TO WAIT UNTIL THE KEYS
ARE RELEASED
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
**********
; PURPOSE: WAIT UNTIL A PRESSED KEY IS RELEASED TO ELIMINATE BOUNCE AND DOUBLE
PRESSING
KEY_RELEASE:    MOVB #$F0,PortT                 ; SET ROWS 4,5,6,7 OF PortT TO HIGH
                NOP                             ; SHORT TIME WAITING FOR PINS TO GO
HIGH
                BRCLR PortT,$0F,FINISH          ; WHEN COLUMN 1-4 (PM0-PM3) IS CLEAR
THEN ALL KEYS
                                                ; HAVE BEEN RELEASED
                BRA KEY_RELEASE                 ; BRANCH BACK TO KEY RELEASE
FINISH          RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
**********
; PURPOSE: TO CHECK AND MAKE SURE WE HAVE A VALID NUMERIC KEY PRESSED
IS_NUMBER_KEY: LDX CUR_PAD_VAL                  ; GET THE CURRENT KEYPAD VALUE ADDRESS
                LDAA X                          ; LOAD THE KEYPAD VALUE ADDRESS
                CMPA #$09                       ; WAS THIS KEY AN INVALID KEY?
```

```
                BGT INVALID                     ; IF IT WAS THEN SET THE FLAG
                RTS                             ; IF NOT RETURN FROM SUBROUTINE
INVALID         MOVB #$01,INVALID_KEY           ; SET THE INVALID KEY FLAG

                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
CHANGE_MODE:
                LDX CUR_PAD_VAL                 ; GET THE CURRENT KEYPAD VALUE ADDRESS
                LDAA X                          ; LOAD THE KEYPAD VALUE ADDRESS
                CMPA #$0A                       ; DOES THE USER WANT TEMP MODE?
                BNE CM_VOLT                     ; IF NOT CHECK VOLTAGE
                LDAB #$01                       ; LOAD ACMB WITH $01
                STAB MODE                       ; STORE $01 IN MODE
CM_VOLT         CMPA #$0B                       ; WAS THIS KEY AN INVALID KEY?
                BNE CM_LIGHT                    ; IF NOT CHECK LIGHT MODE
                LDAB #$02                       ; LOAC ACMB WITH $02
                STAB MODE                       ; STORE $02 IN MODE
CM_LIGHT        CMPA #$0C                       ; WAS THIS KEY AN INVALID KEY?
                BNE CM_DONE                     ; IF NOT WE'RE DONE
                LDAB #$03                       ; LOAD ACMB WITH $03
                STAB MODE                       ; STORE $03 IN MODE
                JSR DRAW_SCREEN                 ; REDRAW THE LCD
CM_DONE         RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
CHANGE_BASE_MODE:
                LDX CUR_PAD_VAL                 ; GET THE CURRENT KEYPAD VALUE ADDRESS
                LDAA X                          ; LOAD THE KEYPAD VALUE ADDRESS
                CMPA #$0D                       ; DOES THE USER WANT TO CHANGE BASE
MODE?
                BNE CBM_DONE                    ; IF NOT THEN WE'RE DONE
                LDAA BASE_MODE                  ; LOAD THE CURRENT BASE MODE
                CMPA #$01                       ; ARE WE IN BINARY MODE?
                BNE CBM_DEC                     ; IF NOT CHECK DEC
                LDAB #$02                       ; LOAD ACMB WITH $02 FOR DEC BASE MODE
                STAB BASE_MODE                  ; STORE $02 IN BASE_MODE
CBM_DEC         CMPA #$02                       ; ARE WE IN DECIMAL MODE?
                BNE CBM_HEX                     ; IF NOT CHECK HEX MODE
                LDAB #$03                       ; LOAC ACMB WITH $03 (HEX MODE)
                STAB BASE_MODE                  ; STORE $03 IN MODE
CBM_HEX         CMPA #$03                       ; ARE WE IN HEX MODE?
                BNE CBM_DONE                    ; IF NOT WE'RE DONE
                LDAB #$01                       ; LOAD ACMB WITH $01 (BINARY)
                STAB BASE_MODE                  ; STORE $01 IN MODE
                JSR DRAW_SCREEN                 ; REDRAW THE LCD
CBM_DONE        RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
CALC_TEMP:      JSR GET_TEMP                    ; GET THE VOLTAGE FROM THE A/D
CONVERTER
                JSR CONVERT_TEMP                ; CONVERT THE VOLTAGE TO THE TABLE
VALUE
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
GET_TEMP:       LDAA #$00                       ; LOAD ACMA WITH 0
                BCLR PortM, RCK                 ; PULSE RCK
                NOP                             ; WAIT
                NOP                             ; WAIT
                BSET PortM, RCK                 ; COMMAND NOW AVAILABEL FOR LCD
                NOP                             ; WAIT
PSPI_EF:        BRCLR SPSR,$20,PSPI_EF          ; WAIT FOR REGISTER EMPTY FLAG (SPIEF)
```

```
                STAA SPDR                       ; OUTPUT COMMAND VIA SPI TO SIPO
PCKFLG1         BRCLR SPSR,$80,PCKFLG1          ; WAIT FOR SPI FLAG
                LDAB SPDR                       ; LOAD FROM SPI TO CLEAR FLAG
                STD VALUE                       ; STORE D IN VALUE VARIABLE
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
CONVERT_TEMP:
                LDD VALUE                       ; LOAD A/D VALUE INTO D
                CPD #57                         ; COMPARE TEMP VOLTAGE
                BGT CT_84                       ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0000                      ; LOAD X WITH ADDRESS OFFSET
                LBRA CT_DONE                    ; BRANCH TO DONE
CT_84           CPD #84                         ; COMPARE TEMP VOLTAGE
                BGT CT_113                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0001                      ; LOAD X WITH ADDRESS OFFSET
                LBRA CT_DONE                    ; BRANCH TO DONE
CT_113          CPD #113                        ; COMPARE TEMP VOLTAGE
                BGT CT_116                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0002                      ; LOAD X WITH ADDRESS OFFSET
                LBRA CT_DONE                    ; BRANCH TO DONE
CT_116          CPD #116                        ; COMPARE TEMP VOLTAGE
                BGT CT_119                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0003                      ; LOAD X WITH ADDRESS OFFSET
                LBRA CT_DONE                    ; BRANCH TO DONE
CT_119          CPD #119                        ; COMPARE TEMP VOLTAGE
                BGT CT_122                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0004                      ; LOAD X WITH ADDRESS OFFSET
                LBRA CT_DONE                    ; BRANCH TO DONE
CT_122          CPD #122                        ; COMPARE TEMP VOLTAGE
                BGT CT_125                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0005                      ; LOAD X WITH ADDRESS OFFSET
                LBRA CT_DONE                    ; BRANCH TO DONE
CT_125          CPD #125                        ; COMPARE TEMP VOLTAGE
                BGT CT_128                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0006                      ; LOAD X WITH ADDRESS OFFSET
                LBRA CT_DONE                    ; BRANCH TO DONE
CT_128          CPD #128                        ; COMPARE TEMP VOLTAGE
                BGT CT_131                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0007                      ; LOAD X WITH ADDRESS OFFSET
                BRA CT_DONE                     ; BRANCH TO DONE
CT_131          CPD #131                        ; COMPARE TEMP VOLTAGE
                BGT CT_134                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0008                      ; LOAD X WITH ADDRESS OFFSET
                BRA CT_DONE                     ; BRANCH TO DONE
CT_134          CPD #134                        ; COMPARE TEMP VOLTAGE
                BGT CT_137                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0009                      ; LOAD X WITH ADDRESS OFFSET
                BRA CT_DONE                     ; BRANCH TO DONE
CT_137          CPD #137                        ; COMPARE TEMP VOLTAGE
                BGT CT_139                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0010                      ; LOAD X WITH ADDRESS OFFSET
                BRA CT_DONE                     ; BRANCH TO DONE
CT_139          CPD #139                        ; COMPARE TEMP VOLTAGE
                BGT CT_142                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0011                      ; LOAD X WITH ADDRESS OFFSET
                BRA CT_DONE                     ; BRANCH TO DONE
CT_142          CPD #142                        ; COMPARE TEMP VOLTAGE
                BGT CT_168                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0012                      ; LOAD X WITH ADDRESS OFFSET
                BRA CT_DONE                     ; BRANCH TO DONE
CT_168          CPD #168                        ; COMPARE TEMP VOLTAGE
                BGT CT_190                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
```

```
                LDAB #0013                      ; LOAD X WITH ADDRESS OFFSET
                BRA CT_DONE                     ; BRANCH TO DONE
CT_190          CPD #190                        ; COMPARE TEMP VOLTAGE
                BGT CT_206                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0014                      ; LOAD X WITH ADDRESS OFFSET
                BRA CT_DONE                     ; BRANCH TO DONE
CT_206          CPD #206                        ; COMPARE TEMP VOLTAGE
                BGT CT_219                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0015                      ; LOAD X WITH ADDRESS OFFSET
                BRA CT_DONE                     ; BRANCH TO DONE
CT_219          CPD #219                        ; COMPARE TEMP VOLTAGE
                BGT CT_228                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0016                      ; LOAD X WITH ADDRESS OFFSET
                BRA CT_DONE                     ; BRANCH TO DONE
CT_228          CPD #228                        ; COMPARE TEMP VOLTAGE
                BGT CT_235                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0017                      ; LOAD X WITH ADDRESS OFFSET
                BRA CT_DONE                     ; BRANCH TO DONE
CT_235          CPD #235                        ; COMPARE TEMP VOLTAGE
                BGT CT_240                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDAB #0018                      ; LOAD X WITH ADDRESS OFFSET
                BRA CT_DONE                     ; BRANCH TO DONE
CT_240          LDAB #0019                      ; LOAD X WITH ADDRESS OFFSET
CT_DONE         LDAA TEMP_MODE                  ; LOAD THE CURRENT TEMP MODE
                CMPA #$01                       ; ARE WE IN CENTIGRADE MODE?
                BNE CT_F                        ; IF NOT CHECK F
                LDY #TEMPERATUREC               ; LOAD ADDRESS OF TEMPERATURE IN C
                BRA GO                          ; FINISH STORING TEMP
CT_F            LDY #TEMPERATUREF               ; LOAD ADDRESS OF TEMPERATURE IN F
GO              ABY                             ; ADD B TO ADDRESS OF TEMP
                STY TEMP_VAL                    ; STORE OUR TEMP VALUE
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
DISPLAY_TEMP:
                BSET PortM, RS                  ; PRINT CHARACTER TO LCD
                LDX #STRING23                   ; PRINT THE SPACES
                JSR PRINT_STRING                ; TO THE LCD
                LDAA #$20                       ; LOAD A BLANK CHARACTER
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                LDX TEMP_VAL                    ; LOAD ACMB WITH THE TEMP ADDRESS
                LDAB X                          ; LOAD THE KEYPAD VALUE ADDRESS
                LDAA #$00                       ; LOAD ACMA WITH 00
                LDX #100                        ; PLACE 10 IN X
                IDIV                            ; DIVIDE THE NUMBER BY 10
                XGDX                            ; MOVE THE REMAINDER TO X AND THE
QUOTIENT TO D
                TBA                             ; MOVE THE VALUE IN B TO A
                PSHX                            ; PUSH X ON THE STACK
                JSR KEY_TO_ASCII                ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                PULX                            ; PULL X OFF STACK
                XGDX                            ; EXCHANGE D WITH X
                LDX #10                         ; PLACE 10 IN X
                IDIV                            ; DIVIDE THE NUMBER BY 10
                XGDX                            ; MOVE THE REMAINDER TO X AND THE
QUOTIENT TO D
                TBA                             ; MOVE THE VALUE IN B TO A
                PSHX                            ; PUSH X ON THE STACK
                JSR KEY_TO_ASCII                ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                PULX                            ; PULL X OFF STACK
                XGDX                            ; EXCHANGE D WITH X
```

```
                TBA                             ; TRANSFER B INTO A
                JSR KEY_TO_ASCII                ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                LDAA TEMP_MODE                  ; LOAD THE CURRENT TEMP MODE
                CMPA #$01                       ; ARE WE IN CENTIGRADE MODE?
                BNE DT_F                        ; IF NOT CHECK F
                LDAA #$43                       ; LOAD ASCII C
                BRA DT_DONE                     ; FINISH STORING TEMP
DT_F            CMPA #$02                       ; ARE WE IN FARENHEIT MODE?
                LDAA #$46                       ; LOAD ASCII F
DT_DONE         JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
CHANGE_TEMP_MODE:
                LDX CUR_PAD_VAL                 ; GET THE CURRENT KEYPAD VALUE ADDRESS
                LDAA X                          ; LOAD THE KEYPAD VALUE ADDRESS
                CMPA #$0E                       ; DOES THE USER WANT TO CHANGE TEMP
MODE?
                BNE CTM_DONE                    ; IF NOT THEN WE'RE DONE
                LDAA TEMP_MODE                  ; LOAD THE CURRENT TEMP MODE
                CMPA #$01                       ; ARE WE IN CENTIGRADE MODE?
                BNE CTM_F                       ; IF NOT CHECK F
                LDAB #$02                       ; LOAD ACMB WITH $02 FOR FARENHEIT BASE
MODE
                STAB TEMP_MODE                  ; STORE $02 IN TEMP_MODE
CTM_F           CMPA #$02                       ; ARE WE IN FARENHEIT MODE?
                BNE CTM_DONE                    ; IF NOT WE'RE DONE
                LDAB #$01                       ; LOAC ACMB WITH $01 (CENTIGRADE MODE)
                STAB TEMP_MODE                  ; STORE $03 IN TEMP MODE
                JSR DRAW_SCREEN                 ; REDRAW THE LCD
CTM_DONE        RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
CALC_VOLT:      JSR GET_VOLTAGE                 ; GET THE VOLTAGE FROM THE A/D
CONVERTER
                JSR CONVERT_VOLTAGE             ; CONVERT THE VOLTAGE TO THE TABLE
VALUE
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
GET_VOLTAGE:    MOVB #$80,ATD0CTL2              ; TURN ON ATD POWER.  NO FAST FLAGS
                JSR delay                       ; WAIT 20USEC FOR ATD POWER TO
STABILIZE
                MOVB #$20,ATD0CTL3              ; DO 4 CONVERSIONS
                MOVB #$05,ATD0CTL4              ; 10-BIT AT 2MHz
                MOVB #$80,ATD0CTL5              ; START CONVERSION: RIGHT-JUSTIFIED,
UNSIGNED, SCAN OF CHANNEL 2
                BRCLR ATD0STAT0,$80,*           ; POLL THE SCF (SEQUENCE COMPLETE FLAG)
                LDD ATD0DR0                     ; LOAD THE FIRST A/D VALUE
                ADDD ATD0DR1                    ; AND ADD THEM ALL UP FOR AVERAGING
                ADDD ATD0DR2                    ; ADD THE THIRD
                ADDD ATD0DR3                    ; ADD THE FOURTH
                LSRD                            ; LET'S DIVIE THE TOTAL BY 4 (RIGHT
SHIFT TWICE)
                LSRD                            ; TO GET THE AVERAGE VALUE (IN D)
                STD VALUE                       ; STORE D IN VALUE VARIABLE
                MOVB #$00,ATD0CTL2              ; TURN OFF ATD POWER
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
CONVERT_VOLTAGE:
                LDD VALUE                       ; LOAD A/D VALUE INTO D
```

```
                LDX #40                         ; LOAD X WITH 40
                IDIV                            ; DIVIDE THE NUMBERS TO GET THE VALUE
OF OUR VOLTAGE
                XGDX                            ; EXCHANGE THE CONTENTS OF D & X
                ADDD #VOLTAGE                   ; ADD D TO ADDRESS OF VOLTAGE
                STD VOLTAGE_VAL                 ; STORE OUR VOLTAGE VALUE
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
DISPLAY_VOLTAGE:
                BSET PortM, RS                  ; PRINT CHARACTER TO LCD
                LDX #STRING23                   ; PRINT THE SPACES
                JSR PRINT_STRING                ; TO THE LCD
                LDX VOLTAGE_VAL                 ; LOAD ACMB WITH THE VOLTAGE ADDRESS
                LDAB X                          ; LOAD THE KEYPAD VALUE ADDRESS
                LDAA #$00                       ; LOAD ACMA WITH 00
                LDX #10                         ; PLACE 10 IN X
                IDIV                            ; DIVIDE THE NUMBER BY 10
                XGDX                            ; MOVE THE REMAINDER TO X AND THE
QUOTIENT TO D
                TBA                             ; MOVE THE VALUE IN B TO A
                PSHX                            ; PUSH X ON THE STACK
                JSR KEY_TO_ASCII                ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                LDAA #$2E                       ; PRINT A PERIOD
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                PULX                            ; PULL X OFF STACK
                XGDX                            ; EXCHANGE D WITH X
                TBA                             ; TRANSFER B INTO A
                JSR KEY_TO_ASCII                ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                LDAA #$56                       ; PRINT A 'V'
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
CALC_LIGHT:     JSR GET_LIGHT                   ; GET THE LIGHT FROM THE A/D CONVERTER
                JSR CONVERT_LIGHT               ; CONVERT THE VOLTAGE TO THE TABLE
VALUE
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
GET_LIGHT:      MOVB #$80,ATD0CTL2              ; TURN ON ATD POWER.  NO FAST FLAGS
                JSR delay                       ; WAIT 20USEC FOR ATD POWER TO
STABILIZE
                MOVB #$20,ATD0CTL3              ; DO 4 CONVERSIONS
                MOVB #$05,ATD0CTL4              ; 10-BIT AT 2MHz
                MOVB #$82,ATD0CTL5              ; START CONVERSION: RIGHT-JUSTIFIED,
UNSIGNED, SCAN OF CHANNEL 2
                BRCLR ATD0STAT0,$80,*           ; POLL THE SCF (SEQUENCE COMPLETE FLAG)
                LDD ATD0DR0                     ; LOAD THE FIRST A/D VALUE
                ADDD ATD0DR1                    ; AND ADD THEM ALL UP FOR AVERAGING
                ADDD ATD0DR2                    ; ADD THE THIRD
                ADDD ATD0DR3                    ; ADD THE FOURTH
                LSRD                            ; LET'S DIVIE THE TOTAL BY 4 (RIGHT
SHIFT TWICE)
                LSRD                            ; TO GET THE AVERAGE VALUE (IN D)
                STD VALUE                       ; STORE D IN VALUE VARIABLE
                MOVB #$00,ATD0CTL2              ; TURN OFF ATD POWER
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
CONVERT_LIGHT:
```

```
                LDD VALUE                       ; LOAD A/D VALUE INTO D
                CPD #49                         ; COMPARE LIGHT VALUE
                BGT CL_349                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDX #STRING25                   ; LOAD X WITH ADDRESS
                BRA CL_DONE                     ; BRANCH TO DONESTD LIGHT_VAL
CL_349          CPD #349                        ; COMPARE LIGHT VALUE
                BGT CL_699                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDX #STRING26                   ; LOAD X WITH ADDRESS
                BRA CL_DONE                     ; BRANCH TO DONESTD LIGHT_VAL
CL_699          CPD #699                        ; COMPARE LIGHT VALUE
                BGT CL_939                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDX #STRING27                   ; LOAD X WITH ADDRESS
                BRA CL_DONE                     ; BRANCH TO DONESTD LIGHT_VAL
CL_939          CPD #939                        ; COMPARE LIGHT VALUE
                BGT BRIGHT                      ; BRANCH IF WE'RE NOT HIGH ENOUGH
                LDX #STRING28                   ; LOAD X WITH ADDRESS
                BRA CL_DONE                     ; BRANCH TO DONESTD LIGHT_VAL
BRIGHT          LDX #STRING29                   ; LOAD X WITH ADDRESS
                BRA CL_DONE                     ; BRANCH TO DONESTD LIGHT_VAL
CL_DONE         STX LIGHT_VAL                   ; STORE OUR LIGHT VALUE
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
DISPLAY_LIGHT:
                BSET PortM, RS                  ; PRINT CHARACTER TO LCD
                LDX LIGHT_VAL                   ; LOAD INDEX X OF LIGHT VALUE
                JSR PRINT_STRING                ; PRINT STRING TO LCD
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
DRAW_SCREEN:    BCLR PortM, RS                  ; SEND A COMMAND TO LCD
                LDAA #$01                       ; CLEAR SCREEN COMMAND
                JSR LCD_INPUT                   ; SEND TO LCD
                LDAA #$02                       ; RETURN TO HOME COMMAND
                JSR LCD_INPUT                   ; SEND COMMAND

                JSR DISPLAY_BASE_MODE           ; GET THE NUMBER BASE AND DISPLAY IT
                JSR DISPLAY_VALUE               ; DRAW A/D OUTPUT

                BCLR PortM, RS                  ; SENT A COMMAND TO LCD
                LDAA #$C0                       ; GO TO SECOND LINE TO PRINT
                JSR LCD_INPUT                   ; SEND COMMAND

                JSR DISPLAY_MODE                ; DRAW THE SYSTEM MODE ON THE SCREEN
                JSR DISPLAY_MODE_VALUE          ; CHECK MODE, AND DISPLAY APPROPRIATE
VALUE

                BCLR PortM, RS                  ; SEND A COMMAND TO LCD
                LDAA #$0E                       ; LCD DISPLAY ON, CURSOR BLINKING
                JSR LCD_INPUT                   ; PRINT COMMAND TO LCD

                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
DISPLAY_MODE_VALUE:
                LDAA MODE                       ; LOAD THE CURRENT MODE IN ACMA
                CMPA #$01                       ; IS IT 1?
                BNE DMV_CV                      ; IF NO, CHECK VOLTAGE
                JSR DISPLAY_TEMP                ; IF WE'RE IN TEMPERATURE MODE, DISPLAY
THE TEMPERATURE
                BRA DMV_CONTINUE                ; LEAVE SUB
DMV_CV          CMPA #$02                       ; IS IT 2?
                BNE DMV_CL                      ; IF NO, CHECK LIGHT
```

```
                JSR DISPLAY_VOLTAGE           ; IF WE'RE IN VOLTAGE MODE, THEN
DISPLAY THE VOLTAGE
                BRA DMV_CONTINUE              ; LEAVE SUB
DMV_CL          CMPA #$03                     ; IS IT 3?
                BNE DMV_CONTINUE              ; IF NO, LEAVE
                JSR DISPLAY_LIGHT             ; IF WE'RE IN LIGHT MODE, THEN DISPLAY
THE LIGHT
DMV_CONTINUE    RTS                          ; RETURN FROM SUB
;*******************************************************************************
***********
DISPLAY_MODE:   BSET PortM, RS                ; LET'S PRINT TO LCD
                BRSET MODE,$03,DM_L           ; BRANCH TO DM_V IF MODE IS LIGHT

                BRSET MODE,$01,DM_T           ; BRANCH TO DM_T IF MODE IS TEMPERATURE
                BRSET MODE,$02,DM_V           ; BRANCH TO DM_V IF MODE IS VOLTAGE
DM_T            LDX #STRING17                 ; LOAD "1:" INTO DISPLAY
                JMP DM_DONE                   ; GO TO END OF SUB
DM_V            LDX #STRING18                 ; LOAD "1:" INTO DISPLAY
                JMP DM_DONE                   ; GO TO END OF SUB
DM_L            LDX #STRING19                 ; LOAD "1:" INTO DISPLAY
DM_DONE         JSR PRINT_STRING              ; GO TO PRINT_STRING SUB
                RTS                           ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
DISPLAY_BASE_MODE:
                BSET PortM, RS                ; LET'S PRINT TO LCD
                BRSET BASE_MODE,$03,DBM_H     ; BRANCH TO DM_H IF MODE IS HEX

                BRSET BASE_MODE,$01,DBM_B     ; BRANCH TO DM_B IF MODE IS BINARY
                BRSET BASE_MODE,$02,DBM_D     ; BRANCH TO DM_D IF MODE IS DECIMAL
DBM_B           LDX #STRING20                 ; LOAD "BIN" INTO DISPLAY
                JMP DBM_DONE                  ; GO TO END OF SUB
DBM_D           LDX #STRING21                 ; LOAD "DEC" INTO DISPLAY
                JMP DBM_DONE                  ; GO TO END OF SUB
DBM_H           LDX #STRING22                 ; LOAD "HEX" INTO DISPLAY
DBM_DONE        JSR PRINT_STRING              ; GO TO PRINT_STRING SUB
                RTS                           ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
DISPLAY_VALUE:
                LDAA BASE_MODE                ; LOAD THE CURRENT MODE IN ACMA
                CMPA #$01                     ; IS IT 1?
                BNE DV_CV                     ; IF NO, CHECK VOLTAGE
                JSR DISPLAY_BIN               ; IF WE'RE IN TEMPERATURE MODE, DISPLAY
THE TEMPERATURE
                BRA DV_CONTINUE               ; LEAVE SUB
DV_CV           CMPA #$02                     ; IS IT 2?
                BNE DV_CL                     ; IF NO, CHECK LIGHT
                JSR DISPLAY_DEC               ; IF WE'RE IN VOLTAGE MODE, THEN
DISPLAY THE VOLTAGE
                BRA DV_CONTINUE               ; LEAVE SUB
DV_CL           CMPA #$03                     ; IS IT 3?
                BNE DV_CONTINUE               ; IF NO, LEAVE
                JSR DISPLAY_HEX               ; IF WE'RE IN LIGHT MODE, THEN DISPLAY
THE LIGHT
DV_CONTINUE     RTS                           ; RETURN FROM SUB
;*******************************************************************************
***********
DISPLAY_DEC:    BSET PortM, RS                ; PRINT CHARACTER TO LCD
                LDD VALUE                     ; LOAD ACMD WITH THE MEASUREMENT VALUE
                LDX #1000                     ; PLACE 10 IN X
                IDIV                          ; DIVIDE THE NUMBER BY 10
                XGDX                          ; MOVE THE REMAINDER TO X AND THE
```

```
QUOTIENT TO D
                TBA                             ; MOVE THE VALUE IN B TO A
                PSHX                            ; PUSH X ON THE STACK
                JSR KEY_TO_ASCII                ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                PULX                            ; PULL X OFF STACK
                XGDX                            ; EXCHANGE D WITH X
                LDX #100                        ; PLACE 10 IN X
                IDIV                            ; DIVIDE THE NUMBER BY 10
                XGDX                            ; MOVE THE REMAINDER TO X AND THE
QUOTIENT TO D
                TBA                             ; MOVE THE VALUE IN B TO A
                PSHX                            ; PUSH X ON THE STACK
                JSR KEY_TO_ASCII                ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                PULX                            ; PULL X OFF STACK
                XGDX                            ; EXCHANGE D WITH X
                LDX #10                         ; PLACE 10 IN X
                IDIV                            ; DIVIDE THE NUMBER BY 10
                XGDX                            ; MOVE THE REMAINDER TO X AND THE
QUOTIENT TO D
                TBA                             ; MOVE THE VALUE IN B TO A
                PSHX                            ; PUSH X ON THE STACK
                JSR KEY_TO_ASCII                ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                PULX                            ; PULL X OFF STACK
                XGDX                            ; EXCHANGE D WITH X
                TBA                             ; TRANSFER B INTO A
                JSR KEY_TO_ASCII                ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
DISPLAY_HEX:    BSET PortM, RS                  ; PRINT CHARACTER TO LCD
                LDD VALUE                       ; LOAD ACMD WITH THE MEASUREMENT VALUE
                LDX #4096                       ; PLACE 10 IN X
                IDIV                            ; DIVIDE THE NUMBER BY 10
                XGDX                            ; MOVE THE REMAINDER TO X AND THE
QUOTIENT TO D
                TBA                             ; MOVE THE VALUE IN B TO A
                PSHX                            ; PUSH X ON THE STACK
                JSR KEY_TO_ASCII                ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                PULX                            ; PULL X OFF STACK
                XGDX                            ; EXCHANGE D WITH X
                LDX #256                        ; PLACE 10 IN X
                IDIV                            ; DIVIDE THE NUMBER BY 10
                XGDX                            ; MOVE THE REMAINDER TO X AND THE
QUOTIENT TO D
                TBA                             ; MOVE THE VALUE IN B TO A
                PSHX                            ; PUSH X ON THE STACK
                JSR KEY_TO_ASCII                ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                PULX                            ; PULL X OFF STACK
                XGDX                            ; EXCHANGE D WITH X
                LDX #16                         ; PLACE 10 IN X
                IDIV                            ; DIVIDE THE NUMBER BY 10
                XGDX                            ; MOVE THE REMAINDER TO X AND THE
QUOTIENT TO D
                TBA                             ; MOVE THE VALUE IN B TO A
                PSHX                            ; PUSH X ON THE STACK
                JSR KEY_TO_ASCII                ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
```

```
                PULX                            ; PULL X OFF STACK
                XGDX                            ; EXCHANGE D WITH X
                TBA                             ; TRANSFER B INTO A
                JSR KEY_TO_ASCII                ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
DISPLAY_BIN:
                BSET PortM, RS                  ; PRINT CHARACTER TO LCD
                LDY #$000A                      ; LOAD Y WITH 10
NEXT_BIT        LDD VALUE                       ; LOAD ACMB WITH THE MEASUREMENT VALUE
                TFR Y,X                         ; LOAD X WITH Y
                DEY                             ; DECREMENT Y
                DEX                             ; DECREMENT X
SHIFT_LOOP      CPY #0                          ; COMPARE Y TO ZERO
                BEQ BIT_TEST                    ; GO TO BIT TEST IF Y = 0
                LSRD                            ; LOGICAL SHIFT RIGHT D
                DEX                             ; DECREMENT X
                CPX #0                          ; COMPARE X TO 0
                BNE SHIFT_LOOP                  ; GO TO SHIFT_LOOP IF X NOT 0
BIT_TEST        BITB #$01                       ; BIT TEST ACMA $01
                BEQ BIT_ZERO                    ; IF IT'S NOT ONE GO TO BIT_ZERO
                LDAA #$01                       ; IF IT IS 1, STORE 01 IN ACMA
                BRA PRINT_BIN                   ; GO TO PRINT_BIN
BIT_ZERO        LDAA #$00                       ; LOAD ACMA WITH 0
PRINT_BIN       PSHY                            ; PUSH Y ONTO STACK
                JSR KEY_TO_ASCII                ; CONVERT VALUE IN ACMA TO ASCII
                JSR LCD_INPUT                   ; SEND CHARACTER TO LCD
                PULY                            ; PULL Y FROM STACK
                CPY #0                          ; COMPARE Y TO ZERO
                BNE NEXT_BIT                    ; IF IT'S NOT ZERO, GO TO NEXT BIT
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: TAKE THE VALUE ACMA AND UPDATE IT TO THE ASCII VALUE OF THAT CHARACTER
KEY_TO_ASCII:   TAB
                LDY #ASCII                      ; LOAD THE BEGINNING ADDRESS OF TABLE
INTO X
                ABY                             ; ADD B TO THE X INDEX
                LDAA Y                          ; LOAD THE ADDRESS OF INDEX X INTO
ACCUM A
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: LOAD A BIT INTO THE LCD (RS = 0 for commands OR RS = 1 FOR PRINT)
LCD_INPUT:
SPI_EF: BRCLR SPSR,$20,SPI_EF   ; WAIT FOR REGISTER EMPTY FLAG (SPIEF)
        STAA SPDR               ; OUTPUT COMMAND VIA SPI TO SIPO
CKFLG1          BRCLR SPSR,$80,CKFLG1           ; WAIT FOR SPI FLAG
        LDAA SPDR                       ; LOAD FROM SPI TO CLEAR FLAG
                NOP                             ; WAIT
                BCLR PortM, RCK                 ; PULSE RCK
        NOP                             ; WAIT
        NOP                             ; WAIT
                BSET PortM, RCK                 ; COMMAND NOW AVAILABEL FOR LCD
        NOP                             ; WAIT
        NOP             ; PROBABLY DON'T NEED TO WAIT
        NOP             ; BUT WE WILL, JUST IN CASE...
        BSET PortM, ENABLE      ; FIRE ENABLE
        NOP                     ; WE SHOULD WAIT AGAIN
        NOP                             ; UNTIL IT'S FINISHED
        BCLR PortM, ENABLE      ; ENABLE OFF
```

```
        JSR delay                   ; GIVE THE LCD TIME TO TAKE COMMAND IN
        RTS                         ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: PRINT A STRING TO THE LCD (USES LCD_INPUT)
PRINT_STRING:
Loop1   LDAA 0,X                ; LOAD A CHARACTER INTO ACMA
        BEQ Done1               ; QUIT IF WE REACH A $00
        JSR LCD_INPUT           ; AND OUTPUT THE CHARACTER
        INX             ; GO TO NEXT CHARACTER
        BRA Loop1                   ; PROCESS NEXT CHARACTER
Done1           RTS                         ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
;*******************************************************************************
***********
delay           LDY #8000                       ; COMMAND DELAY ROUTINE.  WAY TO
LONG. OVERKILL!
A2:     DEY             ; BUT WE DO NEED TO WAIT FOR THE LCD CONTROLLER
        BNE A2                  ; TO DO IT'S THING.  HOW MUCH TIME?
        RTS                         ; RETURN FROM SUBROUTINE

delay2          LDY #$F000                      ; LONG DELAY ROUTINE.  ADJUST AS
NEEDED.
        PSHA            ; SAVE ACMA
A3:     LDAA #$8F                   ; LONG DELAY LOAD ACMA WITH 8F (NESTED LOOP)
AB:     DECA                        ; DECREMENT A
        BNE AB                  ; BRANCH TO AB IF NOT EQUAL
        DEY                         ; DECREMENT Y
        BNE A3                  ; BRANCH TO A3 IF NOT EQUAL
        PULA            ; GET ACMA BACK
        RTS                         ; RETURN FROM SUBROUTINE

delay3          LDAA #$0F                       ; LOAD 15 (F) INTO ACMA
AA6:    LDY #$FFFF                   ; LOAD Y WITH FFFF (Blink Delay routine.)
A6:     DEY             ; DECREMENT Y
        BNE A6                      ; BRANCH TO A6 IF NOT EQUAL
        DECA                        ; DECREMENT A
        BNE AA6                 ; BRANCH TO AA6 IF NOT EQUAL
        RTS                         ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
DIRECTIONS:     BCLR PortM, RS              ; SEND A COMMAND TO LCD
                LDAA #$01                   ; CLEAR SCREEN COMMAND
                JSR LCD_INPUT               ; SEND TO LCD
                LDAA #$02                   ; RETURN TO HOME COMMAND
                JSR LCD_INPUT               ; SEND COMMAND
                BSET PortM, RS              ; LET'S PRINT TO LCD
                LDX #STRING3                ; PRINT DIRECTION
                JSR PRINT_STRING            ; GO TO PRINT_STRING SUB
                BCLR PortM, RS              ; SENT A COMMAND TO LCD
                LDAA #$C0                   ; GO TO SECOND LINE TO PRINT
                JSR LCD_INPUT               ; SEND COMMAND
                BSET PortM, RS              ; LET'S PRINT TO LCD
                LDX #STRING4                ; PRINT DIRECTION
                JSR PRINT_STRING            ; GO TO PRINT_STRING SUB TO PPRINT
                JSR delay2                  ; DELAY A BIT
                JSR delay2                  ; DELAY A BIT

                BCLR PortM, RS              ; SEND A COMMAND TO LCD
                LDAA #$01                   ; CLEAR SCREEN COMMAND
                JSR LCD_INPUT               ; SEND TO LCD
                LDAA #$02                   ; RETURN TO HOME COMMAND
```

```
JSR LCD_INPUT                ; SEND COMMAND
BSET PortM, RS               ; LET'S PRINT TO LCD
LDX #STRING5                 ; PRINT DIRECTION
JSR PRINT_STRING             ; GO TO PRINT_STRING SUB
BCLR PortM, RS               ; SENT A COMMAND TO LCD
LDAA #$C0                    ; GO TO SECOND LINE TO PRINT
JSR LCD_INPUT                ; SEND COMMAND
BSET PortM, RS               ; LET'S PRINT TO LCD
LDX #STRING6                 ; PRINT DIRECTION
JSR PRINT_STRING             ; GO TO PRINT_STRING SUB TO PPRINT
JSR delay2                   ; DELAY A BIT
JSR delay2                   ; DELAY A BIT

BCLR PortM, RS               ; SEND A COMMAND TO LCD
LDAA #$01                    ; CLEAR SCREEN COMMAND
JSR LCD_INPUT                ; SEND TO LCD
LDAA #$02                    ; RETURN TO HOME COMMAND
JSR LCD_INPUT                ; SEND COMMAND
BSET PortM, RS               ; LET'S PRINT TO LCD
LDX #STRING7                 ; PRINT DIRECTION
JSR PRINT_STRING             ; GO TO PRINT_STRING SUB
BCLR PortM, RS               ; SENT A COMMAND TO LCD
LDAA #$C0                    ; GO TO SECOND LINE TO PRINT
JSR LCD_INPUT                ; SEND COMMAND
BSET PortM, RS               ; LET'S PRINT TO LCD
LDX #STRING8                 ; PRINT DIRECTION
JSR PRINT_STRING             ; GO TO PRINT_STRING SUB TO PPRINT
JSR delay2                   ; DELAY A BIT
JSR delay2                   ; DELAY A BIT

BCLR PortM, RS               ; SEND A COMMAND TO LCD
LDAA #$01                    ; CLEAR SCREEN COMMAND
JSR LCD_INPUT                ; SEND TO LCD
LDAA #$02                    ; RETURN TO HOME COMMAND
JSR LCD_INPUT                ; SEND COMMAND
BSET PortM, RS               ; LET'S PRINT TO LCD
LDX #STRING9                 ; PRINT DIRECTION
JSR PRINT_STRING             ; GO TO PRINT_STRING SUB
BCLR PortM, RS               ; SENT A COMMAND TO LCD
LDAA #$C0                    ; GO TO SECOND LINE TO PRINT
JSR LCD_INPUT                ; SEND COMMAND
BSET PortM, RS               ; LET'S PRINT TO LCD
LDX #STRING10                ; PRINT DIRECTION
JSR PRINT_STRING             ; GO TO PRINT_STRING SUB TO PPRINT
JSR delay2                   ; DELAY A BIT
JSR delay2                   ; DELAY A BIT

BCLR PortM, RS               ; SEND A COMMAND TO LCD
LDAA #$01                    ; CLEAR SCREEN COMMAND
JSR LCD_INPUT                ; SEND TO LCD
LDAA #$02                    ; RETURN TO HOME COMMAND
JSR LCD_INPUT                ; SEND COMMAND
BSET PortM, RS               ; LET'S PRINT TO LCD
LDX #STRING11                ; PRINT DIRECTION
JSR PRINT_STRING             ; GO TO PRINT_STRING SUB
BCLR PortM, RS               ; SENT A COMMAND TO LCD
LDAA #$C0                    ; GO TO SECOND LINE TO PRINT
JSR LCD_INPUT                ; SEND COMMAND
BSET PortM, RS               ; LET'S PRINT TO LCD
LDX #STRING12                ; PRINT DIRECTION
JSR PRINT_STRING             ; GO TO PRINT_STRING SUB TO PPRINT
JSR delay2                   ; DELAY A BIT
JSR delay2                   ; DELAY A BIT
```

```
                BCLR PortM, RS                  ; SEND A COMMAND TO LCD
                LDAA #$01                       ; CLEAR SCREEN COMMAND
                JSR LCD_INPUT                   ; SEND TO LCD
                LDAA #$02                       ; RETURN TO HOME COMMAND
                JSR LCD_INPUT                   ; SEND COMMAND
                BSET PortM, RS                  ; LET'S PRINT TO LCD
                LDX #STRING13                   ; PRINT DIRECTION
                JSR PRINT_STRING                ; GO TO PRINT_STRING SUB
                BCLR PortM, RS                  ; SENT A COMMAND TO LCD
                LDAA #$C0                       ; GO TO SECOND LINE TO PRINT
                JSR LCD_INPUT                   ; SEND COMMAND
                BSET PortM, RS                  ; LET'S PRINT TO LCD
                LDX #STRING14                   ; PRINT DIRECTION
                JSR PRINT_STRING                ; GO TO PRINT_STRING SUB TO PPRINT
                JSR delay2                      ; DELAY A BIT
                JSR delay2                      ; DELAY A BIT

                BCLR PortM, RS                  ; SEND A COMMAND TO LCD
                LDAA #$01                       ; CLEAR SCREEN COMMAND
                JSR LCD_INPUT                   ; SEND TO LCD
                LDAA #$02                       ; RETURN TO HOME COMMAND
                JSR LCD_INPUT                   ; SEND COMMAND
                BSET PortM, RS                  ; LET'S PRINT TO LCD
                LDX #STRING15                   ; PRINT DIRECTION
                JSR PRINT_STRING                ; GO TO PRINT_STRING SUB
                BCLR PortM, RS                  ; SENT A COMMAND TO LCD
                LDAA #$C0                       ; GO TO SECOND LINE TO PRINT
                JSR LCD_INPUT                   ; SEND COMMAND
                BSET PortM, RS                  ; LET'S PRINT TO LCD
                LDX #STRING16                   ; PRINT DIRECTION
                JSR PRINT_STRING                ; GO TO PRINT_STRING SUB TO PPRINT
                JSR delay2                      ; DELAY A BIT
                JSR delay2                      ; DELAY A BIT

                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
INTRODUCTION:   BCLR PortM, RS                  ; SEND A COMMAND TO LCD
                LDAA #$01                       ; CLEAR SCREEN COMMAND
                JSR LCD_INPUT                   ; SEND TO LCD
                LDAA #$02                       ; RETURN TO HOME COMMAND
                JSR LCD_INPUT                   ; SEND COMMAND
                BSET PortM, RS                  ; LET'S PRINT TO LCD
                LDX #STRING1                    ; PRINT DIRECTION
                JSR PRINT_STRING                ; GO TO PRINT_STRING SUB
                BCLR PortM, RS                  ; SENT A COMMAND TO LCD
                LDAA #$C0                       ; GO TO SECOND LINE TO PRINT
                JSR LCD_INPUT                   ; SEND COMMAND
                BSET PortM, RS                  ; LET'S PRINT TO LCD
                LDX #STRING2                    ; PRINT DIRECTION
                JSR PRINT_STRING                ; GO TO PRINT_STRING SUB TO PPRINT
                JSR delay2                      ; DELAY A BIT
                JSR delay2                      ; DELAY A BIT
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; TC0 INTERRUPT SUBROUTINE
ISR_TC0:        LDD TC0                         ; INTERRUPT READS THE FLAG SO THIS
WRITE CLEARS THE FLAG
                ADDD #3750                      ; ADD THE EQUIVALENT .1 SECOND CNT TO
REGISTER D
                STD TC0                         ; UPDATE TC0 MEMORY TO NEW VALUE
```

```
                PSHA                            ; SAVE A ON THE STACK
                LDAA TIME_COUNT                 ; LOAD THE VALUE OF TIME_COUNT INTO A
                CMPA #100                       ; IF TIME_COUNT = 5 THEN WE HAVE 1
SECOND
                BNE TMR_UPDATE                  ; IF WE'RE NOT AT 5 YET, GOTO
TMR_UPDATE LINE
                MOVB #$01,TMR_FLAG              ; TURN ON OUR TIMER FLAG
                MOVB #$00,TIME_COUNT            ; RESET OUR TIMER COUNT BACK TO ZERO
                PULA                            ; PUL A BACK OFF THE STACK
PAUSED          RTI                             ; RETURN FROM THE INTERRUPT
TMR_UPDATE      ADDA #01                        ; INCREMENT THE VALUE IN A
                STAA TIME_COUNT                 ; STORE A BACK INTO TIME_COUNT
                PULA                            ; PULL A BACK OFF THE STACK
                RTI                             ; RETURN FROM THE INTERRUPT
;*******************************************************************************
*******
                ORG $FFEE                       ; VECTOR ADDRESS FOR TC0 INTERRUPT
                FDB ISR_TC0                     ; ISR_TIMER IS A LABEL FOR THE
INTERRUPT SUBROUTINE

;*******************************************************************************
******
; Have the Assembler put the solution data in the look-up table

                ORG $5500                       ; The look-up table is at $5500

TABLE:          DC.B $00, $01, $02, $03, $04    ; Define data table of mappings to each
of the
                DC.B $05, $06, $07, $08, $09    ; matrix keypad values.
                DC.B $0A, $0B, $0C, $0D, $0E    ; Memory locations correspond to their
values
                DC.B $0F                        ; i.e. $5500 = 0, $5501 = 1, etc

ROW:            DC.B $10, $20, $40, $80         ; PortT OUTPUT VALUES FOR MATRIX KEYPAD
ROWS
COLUMN:         DC.B $01, $02, $04, $08         ; PortM INPUT VALUES FOR MATRIX KEYPDA
COLUMNS

KP_VALUE:       DC.B $01, $02, $03, $0A         ; KEY VALUES FROM KEYPAD FOR ITERATING
THROUGH
                DC.B $04, $05, $06, $0B
                DC.B $07, $08, $09, $0C
                DC.B $00, $0F, $0E, $0D

ASCII:          DC.B $30, $31, $32, $33, $34    ; Define data table of mappings to each
of the
                DC.B $35, $36, $37, $38, $39    ; ascii values for the keypad
                DC.B $41, $42, $43, $44, $45    ; Memory locations correspond to their
values
                DC.B $46                        ; i.e. $5500 = 0, $5501 = 1, etc

VOLTAGE:        DC.B $00, $02, $04, $06, $08, $0A  ; DEFINE DATA TABLE MAPPING FOR
VOLTAGE MEASUREMENTS
                DC.B $0C, $0E, $10, $12, $14
                DC.B $16, $18, $1A, $1C, $1E
                DC.B $20, $22, $24, $26, $28
                DC.B $2A, $2C, $2E, $30, $32

TEMPERATUREC:   DC.B $00, $0A, $14, $15, $16    ; DEFINE DATA TABLE MAPPING FOR TEMP C
MEASUREMENTS
                DC.B $17, $18, $19, $1A, $1B
                DC.B $1C, $1D, $1E, $28, $32
                DC.B $3C, $46, $50, $5A, $64
```

```
TEMPERATUREF:  DC.B $20, $32, $44, $45, $47  ; DEFINE DATA TABLE MAPPING FOR TEMP F
MEASUREMENTS
               DC.B $49, $4B, $4D, $4E, $50
               DC.B $52, $54, $56, $68, $7A
               DC.B $8C, $9E, $B0, $C2, $D4

STRING1        FCC "TeVoLi 13 Meter"         ; CREATE A STRING FOR PAUSED
               DC.B $00
STRING2        FCC "    By CUI"              ; CREATE A STRING WITH THE RUN
               DC.B $00
STRING3        FCC "PRESS A TO"              ; CREATE A STRING WITH THE UP
               DC.B $00
STRING4        FCC "MEASURE TEMP"            ; CREATE A STRING WITH THE DOWN
               DC.B $00
STRING5        FCC "PRESS B TO"              ; CREATE A STRING FOR THE TIME LINE
               DC.B $00
STRING6        FCC "MEASURE VOLTAGE"         ; CREATE A STRING
               DC.B $00
STRING7        FCC "PRESS C TO"              ; CREATE A STRING
               DC.B $00
STRING8        FCC "MEASURE LIGHT"           ; CREATE A STRING
               DC.B $00
STRING9        FCC "PRESS D TO"              ; CREATE A STRING
               DC.B $00
STRING10       FCC "TOGGLE..."               ; CREATE A STRING
               DC.B $00
STRING11       FCC "BETWEEN DECIMAL,"        ; CREATE A STRING
               DC.B $00
STRING12       FCC "HEX & BINARY"            ; CREATE A STRING
               DC.B $00
STRING13       FCC "PRESS E TO"              ; CREATE A STRING
               DC.B $00
STRING14       FCC "TOGGLE..."               ; CREATE A STRING
               DC.B $00
STRING15       FCC "TEMP BETWEEN"            ; CREATE A STRING
               DC.B $00
STRING16       FCC "CELCIUS OR FAREN"        ; CREATE A STRING
               DC.B $00
STRING17       FCC "TEMP "                   ; CREATE A STRING
               DC.B $00
STRING18       FCC "VOLT "                   ; CREATE A STRING
               DC.B $00
STRING19       FCC "LGHT"                    ; CREATE A STRING
               DC.B $00
STRING20       FCC "BIN   "                  ; CREATE A STRING
               DC.B $00
STRING21       FCC "DEC   "                  ; CREATE A STRING
               DC.B $00
STRING22       FCC "HEX   "                  ; CREATE A STRING
               DC.B $00
STRING23       FCC "      "                  ; CREATE A STRING
               DC.B $00
STRING24       FCC " "                       ; CREATE A STRING FOR LIGHT MEASUREMENT
               DC.B $00
STRING25       FCC "        DARK"            ; CREATE A STRING
               DC.B $00
STRING26       FCC "  MEDIUM LOW"            ; CREATE A STRING
               DC.B $00
STRING27       FCC "      MEDIUM"            ; CREATE A STRING
               DC.B $00
STRING28       FCC " MEDIUM HIGH"            ; CREATE A STRING
               DC.B $00
```

```
STRING29        FCC "      BRIGHT"            ; CREATE A STRING
                DC.B $00
STRING30        FCC "C"                       ; CREATE A STRING
                DC.B $00
STRING31        FCC "F"                       ; CREATE A STRING
                DC.B $00

; End of code

; Define Power-On Reset Interrupt Vector - Required for all programs!

; AGAIN - OP CODES are at column 9
                ORG $FFFE                     ; $FFFE, $FFFF = Power-On Reset Int.
Vector Location
                FDB START                     ; Specify instruction to execute on
power up

                END                           ; (Optional) End of source code

; Labels start in the first column  (left most column = colunm 1)
; OP CODES are at column 9
; COMMENTS follow a ";" symbol
; Blank lines are allowed (Makes the code more readable)
```