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

; Experiment Title: Programmable Electronic Combination Lock
; Experiment Description: This experiment is for an electronic combination lock that
can be
;                         fully programmed using a static administrator
password.  The
;                         lock uses 4 digits and can be reprogrammed at any
time.
; Date: 3/9/13
; Updated: 3/11/13
; Version: 1
; Programmer: Mitchell Hedditch
; Lab Session: Tuesday 8AM-10:50AM


; 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
; SERIAL COMMUNICATION INTERFACE
SPCR1           EQU $00D8
SPCR2           EQU $00D9
SPIB            EQU $00DA
SPSR            EQU $00DB
SPDR            EQU $00DD
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

TEST            EQU $3800                   ; DEFINE LOCATION FOR TEST BYTE STORAGE
FOR DEBUGGING
SAVE_X          EQU $3802                   ; Defines location for the storage of
the X index register
SAVE_Y          EQU $3804                   ; Defines location for the storage of
the Y index register
CUR_COLUMN      EQU $3806                   ; STORAGE LOCATION FOR VARIABLE OF
CURRENT COLUMN
SYS_MODE        EQU $3808                   ; STORAGE LOCATION FOR SYSTEM MODE
                                            ; $0A=LOCKED;$EE=OPEN;$0F=PROGRAM
TMR_FLAG        EQU $3810                   ; DEFINES LOCATION FOR STORAGE OF TIMER
FLAG
                                            ; FLAG= 0->NOTHING; 1->TIMER FIRED
NUM_FLAG        EQU $3812                   ; FLAG FOR KEYPAD BUTTON PRESSED

TIME_COUNT      EQU $3814                   ; MEM ADDRESS TO STORE TIME FOR SECONDS
CUR_PAD_VAL     EQU $3816                   ; USED TO HOUSE THE VALUE FOR THE
CURRENT KEYPAD ITERATION
INPUT1          EQU $3818                   ; INPUT 1 FOR PASSWORD FROM USER
INPUT2          EQU $3820                   ; INPUT 2 FOR PASSWORD FROM USER
INPUT3          EQU $3822                   ; INPUT 3 FOR PASSWORD FROM USER
INPUT4          EQU $3824                   ; INPUT 4 FOR PASSWORD FROM USER
PC1             EQU $3826                   ; STORED LOCK PASSWORD CHARACTER 1
PC2             EQU $3828                   ; STORED LOCK PASSWORD CHARACTER 2
PC3             EQU $3830                   ; STORED LOCK PASSWORD CHARACTER 3
PC4             EQU $3832                   ; STORED LOCK PASSWORD CHARACTER 4
ADMIN_LOCK      EQU $3834                   ; THE TELLS THE SYSTEM WHETHER THE
PROGRAM MODE IS UNLOCKED


;*****************************************************************************
*******
; 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
               SEI                             ; TURN OFF INTERRUPTS

               JSR INIT                        ; INITIALIZE ALL OF OUR VARIABLES,
FLAGS, ETC.

               ; ALL VARIABLES ARE INITIALIZED SO WE'RE READY FOR INTERRUPTS
               CLI                             ; TURN ON INTERRUPTS
;*********************************
;
;      MAIN PROGRAM CODE IS HERE
;
;*********************************
POLL:          JSR GET_KEY                     ; CHECK THE KEYPAD FOR A PRESSED VALUE
               BRCLR NUM_FLAG,$01,CONTINUE     ; IF NO KEY HAS BEEN PRESSED THEN MOVE
ON THE THE NO_KEY LINE
               JSR INPUT_KEY                   ; IF A KEY HAS BEEN PRESSED THEN LOAD
THE NEW NUMBER
               MOVB #$00,NUM_FLAG              ; CLEAR THE NUM FLAG TO WAIT FOR A NEW
KEY
               CMPB #$0C                       ; COMPARE B TO 0C TO SEE IF USER WANTS
CLEAR
               BNE MODE_CHANGE                 ; IF NOT, GO TO MODE_CHANGE
               JSR CLEAR_KEYS                  ; IF C PRESSED, CLEAR ALL OF THE KEY
USER INPUTS
               BRA CONTINUE                    ; NOW LET'S GO BACK AND POLL THE KEYS
AGAIN
MODE_CHANGE    JSR SYSTEM_ACTION               ; CHECK TO SEE IF WE NEED TO CHANGE
MODES
               LDAA SYS_MODE                   ; GET THE CURRENT SYSTEM MODE
               CMPA #$0A                       ; ARE WE IN LOCK MODE?
               BNE PROG_MODE                   ; IF NOT IN LOCK MODE, CHECK PROGRAM
MODE
               JSR LOCKED                      ; GOTO THE LOCKED SUBROUTINE
               BRA CONTINUE                    ; GO BACK AND WAIT FOR ANOTHER KEY
PRESS
PROG_MODE      CMPA #$0F                       ; ARE WE IN PROGRAM MODE?
               BNE CONTINUE                    ; IF NOT, THEN GO BACK AND WAIT FOR
ANOTHER KEY
               JSR PROGRAM                     ; IF WE'RE IN PROGRAM MODE GO TO
PROGRAM SUBROUTINE
CONTINUE       BRA POLL                        ; GO BACK START PROCESSING AT POLL
AGAIN!

;*******************************************************************************
***********
; 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
```

```
                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

                MOVB #$00,TMR_FLAG              ; INITIALIZE THE TIMER FLAG TO LOW
                LDD #$0000                      ; INITIALIZE THE COUNT TO 0
                MOVB #$00,TIME_COUNT            ; SET TIME_COUNT TO 0
                MOVB #$00,NUM_FLAG              ; SET NUM_FLAG TO 0 TO
                MOVB #$0A,SYS_MODE              ; INITIALIZE THE SYSTEM IN PROGRAM MODE
                MOVB #$01,ADMIN_LOCK            ; INITIALIZE THE ADMIN PROGRAM MODE AS
LOCKED
                MOVB #$FF,INPUT1                ; INITIALIZE INPUT1 TO FF SINCE THAT
BUTTON CAN NEVER BE PRESSED
                MOVB #$FF,INPUT2                ; INITIALIZE INPUT2 TO FF SINCE THAT
BUTTON CAN NEVER BE PRESSED
                MOVB #$FF,INPUT3                ; INITIALIZE INPUT3 TO FF SINCE THAT
BUTTON CAN NEVER BE PRESSED
                MOVB #$FF,INPUT4                ; INITIALIZE INPUT4 TO FF SINCE THAT
BUTTON CAN NEVER BE PRESSED

                LDAA #$00                       ; SET THE LEFT LED BLANK
                JSR SERIAL_OUT                  ; OUTPUT THE DATA SERIALLY
                JSR UPDT_LCD1                   ; UPDATE LCD1 DISPLAY
                LDAA #$68                       ; MAKE SURE RIGHT LED IS "L"
                JSR SERIAL_OUT                  ; OUTPUT THE DATA SERIALLY
                JSR UPDT_LCD2                   ; UPDATE LCD2 DISPLAY
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; TIMER INITIALIZATION
INIT_TMR:       ; 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
                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:        BCLR PortM,$03                  ; TURN THE LATCH ENABLES OFF!!
                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 (PM2-PM5) IS CLEAR
THEN ALL KEYS
                                                ; HAVE BEEN RELEASED
                BRA KEY_RELEASE                 ; BRANCH BACK TO KEY RELEASE
FINISH          RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: THIS SUBROUTINE IS USED TO LOAD A NEW DIGIT INTO THE LED AND THE COUNT
VALUE
INPUT_KEY:      LDY CUR_PAD_VAL                 ; LOAD THE EFFECTIVE ADDRESS INTO Y
(NEW VALUE)
                LDAB Y                          ; LOAD A WITH THE ADDRESS IN Y
                RTS                             ; RETURN FROM SUBROUTINE
```

```
;*******************************************************************************
***********
; PURPOSE:
SERIAL_OUT:
SPI_EF          BRCLR SPSR,$20,SPI_EF          ; WAIT FOR REGISTER EMPTY FLAG
                STAA SPDR                      ; OUTPUT COMMAND VIA SPI TO SIPO FROM
ACCUM A
CKFLG1          BRCLR SPSR,$80,CKFLG1          ; WAIT FOR THE SPI FLAG
                LDAA SPDR                      ; AUTOMATIC SPI FLAG CLEAR - YOU MUST
DO THIS
                NOP                            ; WAIT
                BCLR PortM,RCK                 ; PULSE RCK
                NOP                            ; WAIT
                NOP                            ; WAIT AGAIN
                BSET PortM,RCK                 ; DATA NOW AVAILABLE AT 74HC595 OUTPUT
                RTS                            ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
UPDT_LCD1:
                BSET PortM,$01                 ; ENABLE LCD1 LATCH
                NOP                            ; WAIT
                NOP                            ; WAIT
                BCLR PortM,$03                 ; DISABLE THE LATCHES
                RTS                            ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
; PURPOSE: TAKE THE VALUE IN THE Y INDEX AND DISPLAY IT IN THE ONES LCD
UPDT_LCD2:
                BSET PortM,$02                 ; ENABLE LCD2 LATCH
                NOP                            ; WAIT
                NOP                            ; WAIT
                BCLR PortM,$03                 ; DISABLE THE LATCHES
                RTS                            ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
SYSTEM_ACTION: ;CHECK FOR LOCK MODE
                CMPB #$0A                      ; DID THE USER PRESS A?
                BNE PROG                       ; IF NOT, THEN CHECK PROGRAM MODE
                STAB SYS_MODE                  ; IF SO, PUT US IN LOCK MODE
                MOVB #$01,ADMIN_LOCK           ; MAKE SURE ADMIN MODE IS LOCKED
                LDAA #$00                      ; CLEAR THE LEFT LCD DISPLAY
                JSR SERIAL_OUT                 ; SERIAL DATA TO SIPO
                JSR UPDT_LCD1                  ; OUTPUT SIPO TO LCD1 (LEFT)
                LDAA #$68                      ; LOAD THE RIGHT DISPLAY WITH AN "L"
                BRA NEW_MODE                   ; BRANCH TO NEW_MODE LINE
                ;CHECK FOR PROGRAM MODE
PROG            CMPB #$0F                      ; DID THE USER PRESS "F"?
                BNE NO_MODE                    ; IF NOT, THEN NO MODE CHANGE GO TO
NO_MODE
                STAB SYS_MODE                  ; IF SO, PUT US IN PROGRAM MODE
                LDAA #$04                      ; PUT A LINE AT THE TOP OF THE LEFT
DISPLAY TO INDICATE ADMIN
                JSR SERIAL_OUT                 ; SEND ACCUM A TO SIPO
                JSR UPDT_LCD1                  ; OUTPUT SIPO DATA TO LCD1
                LDAA #$3E                      ; SEND A "P" FOR RIGHT DISPLAY
NEW_MODE        JSR SERIAL_OUT                 ; OUTPUT SERIAL DATA TO SIPO FROM ACCUM
A
                JSR UPDT_LCD2                  ; MOVE DATA FROM SIPO TO LCD2 (RIGHT)
                JSR CLEAR_KEYS                 ; CLEAR ALL USER INPUT KEY VALUES
NO_MODE         RTS                            ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
LOCKED:         CMPB #$0E                      ; DID THE USER PRESS E?
```

```
                BNE ENTRY                       ; IF NOT THEN ENTER THE NEW KEY VALUE
                LDAA INPUT1                     ; LOAD USER INPUT1 INTO ACCUM A
                CMPA PC1                        ; COMPARE TO FIRST PASSWORD CHARACTER
                BNE INCORRECT                   ; IF NOT EQUAL GO TO INCORRECT
                LDAA INPUT2                     ; LOAD USER INPUT2 INTO ACCUM A
                CMPA PC2                        ; COMPARE TO SECOND PASSWORD CHARACTER
                BNE INCORRECT                   ; IF NOT EQUAL GO TO INCORRECT
                LDAA INPUT3                     ; LOAD USER INPUT3 INTO ACCUM A
                CMPA PC3                        ; COMPARE TO THIRD PASSWORD CHARACTER
                BNE INCORRECT                   ; IF NOT EQUAL GO TO INCORRECT
                LDAA INPUT4                     ; LOAD USER INPUT4 INTO ACCUM A
                CMPA PC4                        ; COMPARE TO FOURTH PASSWORD CHARACTER
                BNE INCORRECT                   ; IF NOT EQUAL GO TO INCORRECT
CORRECT         LDAA #$EE                       ; LOAD LCD VALUE "O" INTO ACCUM A
                STAA SYS_MODE                   ; CHANGE SYS_MODE INTO OPEN "EE" VALUE
                JSR SERIAL_OUT                  ; OUTPUT ACCUM A TO SIPO SERIALLY
                JSR UPDT_LCD2                   ; UPDATE RIGHT LCD TO "O"
NEW             LDAA #$00                       ; CLEAR VALUE FOR LCD1 INTO ACCUM A
                JSR SERIAL_OUT                  ; OUTPUT ACCUM A TO SIPO SERIALLY
                JSR UPDT_LCD1                   ; UPDATE LEFT LCD TO BLANK
DONE            RTS                             ; RETURN FROM SUBROUTINE
INCORRECT       JSR CLEAR_KEYS                  ; CLEAR USER ENTERED KEYS IF PASSWORD
INCORRECT
                LDAA #$00                       ; CLEAR VALUE FOR LCD1 INTO ACCUM A
                JSR SERIAL_OUT                  ; OUTPUT ACCUM A TO SIPO SERIALLY
                JSR UPDT_LCD1                   ; UPDATE LEFT LCD TO BLANK
                RTS                             ; RETURN FROM SUBROUTINE
ENTRY           JSR LOAD_INPUTS                 ; LETS GO AND LOAD USER INPUT
                RTS                             ; RETURN FROM SUBROUTINE
;*******************************************************************************
**********
PROGRAM:        BRCLR ADMIN_LOCK,$01,A_UNLOCKED; IF WE'RE ADMIN UNLOCKED, THEN GOTO
A_UNLOCKED
                LDAA #$04                       ; WE'RE LOCKED, DISPLAY A LINE AT THE
TOP
                JSR SERIAL_OUT                  ; OUTPUT ACCUM A TO SIPO SERIALLY
                JSR UPDT_LCD1                   ; OUTPUT SIPO DATA TO LCD1
                CMPB #$0E                       ; DID THE USER PRESS E?
                BNE NOT_READY                   ; IF NOT THEN ENTER THE NEW KEY VALUE
                JSR ADMIN_ENTRY                 ; IF USER PRESSED E, THEN CHECK THEIR
ADMIN_PASSWORD
                RTS                             ; RETURN FROM SUBROUTINE
A_UNLOCKED      CMPB #$0E                       ; DID THE USER PRESS E?
                BEQ NEW_PW                      ; IF NOT THEN ENTER THE NEW KEY VALUE
                JSR LOAD_INPUTS                 ; LOAD THE USERS LATEST INPUT
                LDAA #$00                       ; LOAD 0 INTO ACCUM A
                LDX #TABLE                      ; LOAD TABLE VALUE INTO X
                ABX                             ; ADD B TO X AND PLACE IT IN X
                LDAA X                          ; LOAD X INTO ACCUM (VALUE OF KEY)
                JSR SERIAL_OUT                  ; OUTPUT KEY VALUE TO SIPO SERIALLY
                JSR UPDT_LCD1                   ; UPDATE LCD1 WITH SIPO VALUE
                RTS                             ; RETURN TO SUBROUTINE
NEW_PW          LDAA INPUT1                     ; IF WE'RE HERE, LOAD INPUT1 INTO ACCUM
A
                STAA PC1                        ; STORE FIRST NEW PW CHARACTER
                LDAA INPUT2                     ; LOAD INPUT2 INTO ACCUM A
                STAA PC2                        ; STORE SECOND NEW PW CHARACTER
                LDAA INPUT3                     ; LOAD INPUT3 INTO A
                STAA PC3                        ; STORE PW CHARACTER 3
                LDAA INPUT4                     ; LOAD INPUT4 INTO A
                STAA PC4                        ; STORE PW CHARACTER 4
                MOVB #$01,ADMIN_LOCK            ; RELOCK ADMIN PROGRAM MODE
                LDAA #$10                       ; LOAD ACCUM A WITH 10 (DASH FOR LCD)
```

```
                JSR SERIAL_OUT              ; SERIALLY OUTPUT DATA TO SIPO
                JSR UPDT_LCD1               ; UPDATE LCD1 WITH SIPO
                RTS                         ; RETURN FROM SUBROUTINE
NOT_READY       JSR LOAD_INPUTS             ; LOAD USER INPUT IF WE'RE HERE
                RTS                         ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
LOAD_INPUTS:
                CMPB #09                    ; DID THE USER PRESS A NON-NUMERIC
CHARACTER?
                BGT OVER_9                  ; THEN EXIT THE SUBROUTINE, WE ONLY
WANT NUMBERS
I1              LDAA INPUT1                 ; LOAD THE CURRENT VALUE OF THE FIRST
PW CHARACTER
                CMPA #$FF                   ; COMPARE IT TO OUR DEFAULT
                BNE I2                      ; IF IT'S NOT DEFAULT IT'S ALREADY
USED, GOTO NEXT CHARACTER
                STAB INPUT1                 ; STORE THE VALUE INTO THE FIRST USER
INPUT
                LDAA #$80                   ; LOAD ACCUM A WITH FIRST LED VALUE FOR
INPUT PROGRESS
                BRA DISP_UD                 ; BRANCH TO DISPLAY UPDATE TO DISPLAY
LED VALUE
I2              LDAA INPUT2                 ; LOAD THE CURRENT VALUE OF THE SECOND
PW CHARACTER
                CMPA #$FF                   ; COMPARE IT TO OUR DEFAULT
                BNE I3                      ; IF IT'S NOT DEFAULT IT'S ALREADY
USED, GOTO NEXT CHARACTER
                STAB INPUT2                 ; STORE THE VALUE INTO THE SECOND USER
INPUT
                LDAA #$C0                   ; LOAD ACCUM A WITH SECOND LED VALUE
FOR INPUT PROGRESS
                BRA DISP_UD                 ; BRANCH TO DISPLAY UPDATE TO DISPLAY
LED VALUE
I3              LDAA INPUT3                 ; LOAD THE CURRENT VALUE OF THE THIRD
PW CHARACTER
                CMPA #$FF                   ; COMPARE IT TO OUR DEFAULT
                BNE I4                      ; IF IT'S NOT DEFAULT IT'S ALREADY
USED, GOTO NEXT CHARACTER
                STAB INPUT3                 ; STORE THE VALUE INTO THE THIRD USER
INPUT
                LDAA #$E0                   ; LOAD ACCUM A WITH THIRD LED VALUE FOR
INPUT PROGRESS
                BRA DISP_UD                 ; BRANCH TO DISPLAY UPDATE TO DISPLAY
LED VALUE
I4              LDAA INPUT4                 ; LOAD THE CURRENT VALUE OF THE FOURTH
PW CHARACTER
                CMPA #$FF                   ; COMPARE IT TO OUR DEFAULT
                BNE TOO_MANY                ; IF WE'VE GOTTEN HERE, THE USER
ENTERED TOO MANY VALUES
                STAB INPUT4                 ; STORE THE USER VALUE INTO THE FOURTH
USER INPUT VAR
                LDAA #$F0                   ; LOAD ACCUM A WITH FOURTH LED VALUE
FOR INPUT PROGRESS
DISP_UD         JSR SERIAL_OUT              ; SEND ACCUM A THROUGH SERIAL OUTPUT TO
SIPO
                JSR UPDT_LCD1               ; UPDATE LEFT LCD (LCD1) WITH SIPO DATA

OVER_9          RTS                         ; RETURN FROM SUBROUTINE
TOO_MANY        JSR CLEAR_KEYS              ; CLEAR USER ENTERED VALUES
                RTS                         ; RETURN FROM SUBROUTINE
;*******************************************************************************
***********
```

```
ADMIN_ENTRY:    LDX #MASTER                     ; LOAD X WITH THE FIRST LOCATION OF OUR
MASTER PW
                LDAA INPUT1                     ; LOAD ACCUM A WITH FIRST USER INPUT
VALUE
                LDAB X                          ; LOAD ACCUM B WITH VALUE OF FIRST
ADMIN-PW CHARACTER
                CBA                             ; COMPARE THE VALUES
                BNE DONT_UNLOCK                 ; IF THEY'RE NOT THE SAME GO TO
DONT_UNLOCK!
                LDX #MASTER+1                   ; LOAD X WITH THE SECOND LOCATION OF
OUR MASTER PW
                LDAA INPUT2                     ; LOAD ACCUM A WITH SECOND USER INPUT
VALUE
                LDAB X                          ; LOAD ACCUM B WITH VALUE OF SECOND
ADMIN-PW CHARACTER
                CBA                             ; COMPARE THE VALUES
                BNE DONT_UNLOCK                 ; IF THEY'RE NOT THE SAME GO TO
DONT_UNLOCK!
                LDX #MASTER+2                   ; LOAD X WITH THE THIRD LOCATION OF OUR
MASTER PW
                LDAA INPUT3                     ; LOAD ACCUM A WITH THE THIRD USER
INPUT VALUE
                LDAB X                          ; LOAD ACCUM B WITH VALUE OF THIRD
ADMIN-PW CHARACTER
                CBA                             ; COMPARE THE VALUES
                BNE DONT_UNLOCK                 ; IF THEY'RE NOT THE SAME GO TO
DONT_UNLOCK!
                LDX #MASTER+3                   ; LOAD X WITH THE FOURTH LOCATION OF
OUR MASTER PW
                LDAA INPUT4                     ; LOAD ACCUM A WITH THE FOURTH USER
INPUT VALUE
                LDAB X                          ; LOAD ACCUM B WITH THE VALUE OF THIRD
ADMIN-PW CHARACTER
                CBA                             ; COMPARE THE VALUES
                BNE DONT_UNLOCK                 ; IF THEY'RE NOT THE SAME GO TO
DONT_UNLOCK!
                MOVB #$00,ADMIN_LOCK            ; IF WE'VE MADE IT HERE, UNLOCK AND
LET'EM IN!
DONT_UNLOCK     JSR CLEAR_KEYS                  ; CLEAR ALL OF THE USER ENTERED KEY
VALUES
                RTS                             ; RETURN FROM THE SUBROUTINE
;*******************************************************************************
***********
CLEAR_KEYS:     MOVB #$FF,INPUT1                ; CLEAR INPUT1 VALUE
                MOVB #$FF,INPUT2                ; CLEAR INPUT2 VALUE
                MOVB #$FF,INPUT3                ; CLEAR INPUT3 VALUE
                MOVB #$FF,INPUT4                ; CLEAR INPUT4 VALUE
                LDAA #$00                       ; MAKE SURE TO CLEAR THE LEFT DISPLAY
                JSR SERIAL_OUT                  ; OUTPUT CLEAR TO SIPO SERIALLY
                JSR UPDT_LCD1                   ; OUTPUT SIPO CLEAR TO LCD1
                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 = 100 THEN WE HAVE 1
SECOND
```

```
                BNE TMR_UPDATE                  ; IF WE'RE NOT AT 100 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
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 $5000

TABLE:          DC.B $EE, $82, $76, $D6, $9A    ; Define data table of mappings to each
of the
                DC.B $DC, $FC, $86, $FE, $DE    ; segments of the 7-segment LED
displays
                DC.B $BE, $F8, $6C, $F2, $7C    ; Memory locations correspond to their
values
                DC.B $3C                        ; 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

MASTER:         DC.B $01, $09, $08, $02         ; DATA TABLE FOR MASTER PASSWORD WHICH
IS 1982

; 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)
```