KegLock(DK)

Final Project Report
December 14, 2003
E155

Kim Shultz and Damian Small

Abstract

Kegs in North Dorm are a constant point of contention between the dorm and the administration. The main problem is that kegs remain tapped and available to anyone who might wander through the dorm, even when residents are not around to monitor usage. Our system implements access control for kegs, with such useful features as: automatic shut off alarm, time-limited access code, unlimited access code, master configuration code, settable/ resettable codes. An emergency reset button is protected by physical security measures. The system uses a keypad and LCD to provide an easy to understand, user-friendly interface, which enables residential keg access.

Introduction

This project is intended to help North Dorm to comply with the wishes of the administration on student alcohol use. There have been problems in North with the regulation of kegs. The administration has repeatedly asked North to un-tap kegs each day and to try to curb underage drinking. The KegLock is a proposed solution to these problems. The digital lock has a combination that allows a single beer to be poured, a combination that can be given only to people over 21. There is another code that can turn the valve on for the entire night if the dorm is throwing a party. These codes are settable and can also be erased to prevent access to the keg. Whichever code is used, the keg will be closed at 6 AM, to prevent it from being tapped during the day.

The KegLock has three combinations, all of which can be changed. The master code will allow the user to set or reset the codes, as well as set the time. The continuous code will open the valve until 6 AM. The single-use code will open the valve for a variable length of time, currently set for 11 seconds. The single-use and continuous use codes can be erased, to prevent access to a keg.

The physical components of the KegLock are shown in Figure 1. The PIC controls the finite-state machine for the system. The valve is opened by current from the Darlington transistor, which is used as a switch controlled by the PIC. The LCD is also controlled by the PIC, which sends control and data signals. The keypad is used for user-input, which is stored in the PIC. The clock chip is used to keep time. The PIC can write the time to the clock chip or read the time. An alarm on the clock chip is used to activate the system reset at 6 AM. The system can be externally reset at anytime by pressing the reset button, at which point all of the codes will be erased and the user prompted for entry of a new master code.

The entire system is powered by a 12 V DC adapter. The adaptor connects to the DC power jack on the MicroP’s board. The power regulator on the board provides 5 V to most of the components, but it is bypassed for the valve and the backlight to the LCD.

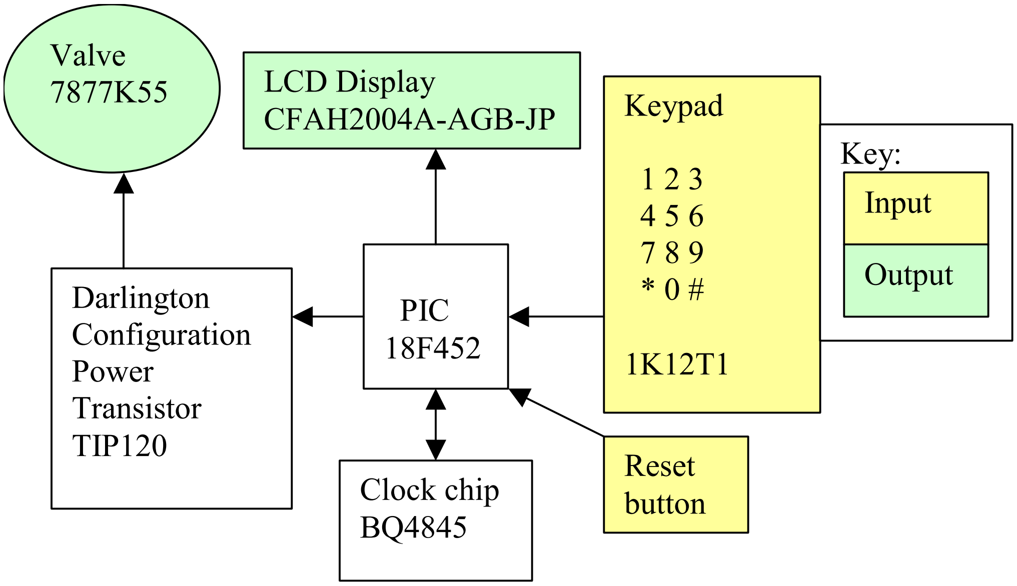


Figure 1: Physical Components of the KegLock

New Hardware

Valve

The main function of our project is to control a valve, so one of the most important pieces of hardware that we used was an electrically controllable valve. We chose a normally-closed solenoid valve, which is opened by applying a 12V, .54A signal. In order to supply such a large amount of power, we used a TIP120, Darlington configuration power transistor. The collector of the transistor was hooked up to the valve, and the emitter to ground. Then, with a 1K resistor in series to limit base current, the PIC can effectively control the valve with a 5V signal, and less than 25mA to the base of the transistor.

Clock Chip

Our system also needs to keep track of the time, both to display to the user on the main entry screen, and to generate the 6am alarm. To perform these functions, we chose a Real Time Clock chip, the bq5854 Parallel RTC. This chip can keep track of the date, time, an alarm, and generate interrupts based on a variety of conditions. The RTC also has inputs for a battery backup, allowing it to keep track of time even when the system is shut off. In order to operate, the RTC requires a 32.768KHz crystal that provides the timing reference for the counter inside the chip.

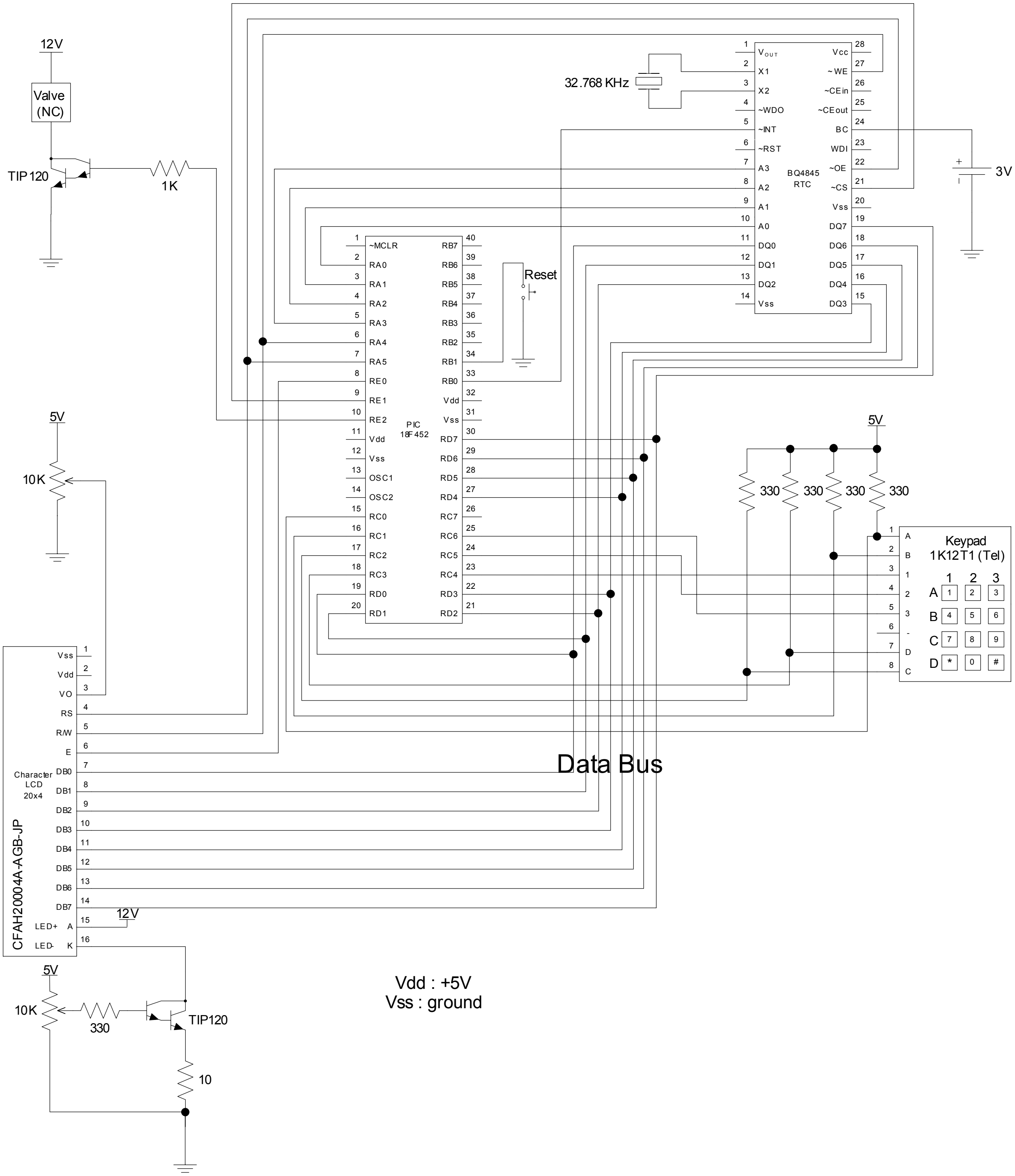
The RTC is organized much like any external memory, with read and write functions, an 8bit data bus, and a chip enable system allowing multiplexing of the main data bus. It contains 16 8-bit registers that allow read/write access to the time, alarm, and configuration information, which makes it easy to control.

Every time the system is reset, the PIC re-writes the configuration information to the RTC to make sure that it is correct, and checks to see if an interrupt has been generated while the system was shut down. The RTC notifies the PIC of alarms by driving the ~INT line low. This line is connected to the RB0 pin on the PIC, which is configured to generate an internal interrupt on the negative edge, and uses a large pull-up resistor to keep the line high when it is not being driven by an external component. When an interrupt is detected, the PIC clears the valve state, closing it, and also erases the single use code.

In normal operation the PIC polls the RTC around every 50ms and reads the time. The PIC then formats the time data and displays it on the LCD, allowing the user to see what time it is.

Schematics

The schematic of our system is below (Figure 2).



Vdd : +5V
Vss : ground

Figure 2: System Schematic

On the top left, our valve is controlled by a Darlington transistor configuration. A 1k resistor limits base current from the PIC. Top right is our RTC (the clock chip). It uses a 32.768kHz crystal to keep accurate time, and a 3V backup battery to enable operation through power-failures. The PIC in the center acts as the brains of the operation, controlling each of the peripheral components and keeping track of the state. A reset button triggers the reset interrupt. In the lower left is the LCD, which provides feedback to the user. The data bus is shared between the LCD and the RTC. The LCD has two adjustments, for contrast and back light intensity. The contrast adjustment is controlled by a 10k POT, while the back light is controlled by an adjustable current supply, which uses a Darlington transistor configuration with emitter degradation and a 10k POT. The keypad is on the right, and uses a standard matrix polling scheme with 330 ohm pull-up resistors on the rows.

Microcontroller Design

The PIC acts as a finite-state machine to control the system. A partial state machine of the system can be seen in Appendix B. The PIC controls the valve, LCD, and polls the keypad. It gives data and control signals to the LCD and the clock chip (when setting the time). It also outputs a control signal to the Darlington power transistor used to turn on the valve. The PIC gets input from the keypad by polling the columns. When the PIC is in a state where it needs input from the keypad, it pulls each of the columns low in turn and waits for a low input from one of the rows, which are weakly held high through resistors. The clock chip sends time information to the PIC. If the reset button is pressed, the PIC will go to a reset state.

Interface with the LCD

The LCD display is one of the most complicated components of our project. Because the LCD has its own controller onboard, it can be written to using commands, as documented in it's data sheet. The controller does have specifications with regard to signal timing, but for the most part the PIC is slow enough so that these timings do not matter. Also, the PIC has a read command which returns a busy flag when an operation is being performed. The one area where timing is important is in the initialization stage. The LCD has a set initialization procedure that requires minimum delays between steps, and the busy flag is not operational for this initialization. In order to account for these delays, we have used simple delay loops, which use the cycle time of the PIC processor to delay for a set amount of time.

Once the LCD has been initialized, we have written a function that copies a block of data from the PIC to the data memory on the LCD. This subroutine takes as arguments the data block and register, and copies the next 80 bytes, starting from that location. Copying the data from the PIC to the LCD is simple: first we copy the next byte, then we check if we're done, and if not we check the LCD busy flag until it is finished processing the byte and loop. The one complicated part is that the display data is stored in an interlaced fashion: first line 1, then line 3, then line 2, and finally line 4. This means that we have to pre-interlace the data before it is copied to the LCD. We do this by using a temporary block of data, stored at the start of data block 1. By manually copying over line 1, then advancing the pointer to where line 3 would start, and copying line 2, etc. we pre-interlace the data, allowing the display screens to be stored in an easily human-readable format.

Our display can re-write the data block in around 7ms, but the actual LCD screen takes about 150ms to respond, giving us an effective refresh rate of around 7Hz

Initialize memory

Because our LCD display routines allow us access display pages stored in data memory, we need to initialize the data memory on any reset. Our data is stored in code, as DB data bytes. The DB command allows us to store the data in either hex form, or, conveniently enough, in ASCII strings. It so happens that the ROM character code page stored in the LCD controller is very close to the standard ASCII specification. This means that for all of the common characters (0-9, a-z, A-Z, and most symbols) the byte codes match exactly, and we can input the display screens into code directly. In the code, we have one set of constants to hold the start of the data. The end of the data is detected automatically when a 0x00 code is hit (0x20 is the code for a space, so there is no need for a 0x00 in a display screen). The routine for copying memory from flash is simple: just a loop with a check for a 0x00 byte, using the post-increment features of the table read pointer, and the FSR0 pointer.

The combinations for the lock must also be initialized on reset. When codes are written or erased, they are stored in the EEPROM on the PIC. When the system is reset (other than the 6 AM reset), the codes are read from the EEPROM and stored in the appropriate file registers.

Interface with the keypad

When the system is prepared for input from the keypad, it polls the keypad. The rows of the keypad are tied to power through 47kΩ resistors. Each column is pulled low in turn. A 5ms delay loop is used to ensure that the signal is not bouncing. If a low value is detected on any of the rows, the system recognizes that a key has been pressed. Based on the row and column that are low, the system decodes the input and saves it in a file register called "inputDigit." * and # are stored as E and F, respectively. The key is saved in the lower 4 bits of the register. The upper 4 bits are 0. The system will continue to poll if no low row is found.

Code operation

<u>Code input</u>

To input a code, FSR0 starts pointing at the first input file register. The key pressed is read as described in "Keypad input." If the key is a number, that number is stored in the file register and the pointer moves to the next register. If the user attempts to enter more digits than belong in the code, the system displays an error message and clears the input file registers. If the user presses the "*" key, the input file registers clear. Once the user presses "#," used as an enter key, the length of the code is checked. If it is not 4 or 6 digits, an error message is displayed and the input registers cleared. If it is a valid length, the code is accepted.

<u>Code recognition</u>

Compare 6 input digits to each code. The codes are stored in a series of file select registers. To compare codes, we use file select registers. Starting with the first digit of the input code and the code to be compared, we compare those digits, and if a match is found, we compare the next digits. If no match is found, we compare the next code until each code is compared. The single-use code acts like a 6-digit code with the two most significant bits equal to A. If a match is found, the starting address of that code is stored in a file register called "match." If no match is found, match contains 0.

<u>Code set and reset</u>

When the user decides to set a code, the starting address of the code to be set is stored in a register called "codeSet." The system then prompts the user for input. The user presses keys on the keypad and the system recognizes them as described under "Interface with the keypad." The code is stored in the input file registers as described in "Code input." Once the code has been entered, its length is checked. If it is not the proper length, the user is prompted to reenter the code. If it is the proper length, the code is stored in the appropriate file registers using FSR's to copy one digit at a time. The user is then prompted to reenter the code. The new input is compared to the stored input. If they do not match, the input is cleared and the user prompted to try again. If they do match, the new code is saved. If the user chooses to reset the code, "A" will be stored as every digit of the code, to indicate that there is no valid code (because there is no "A" input from the keypad). When a new code is saved, it is copied to EEPROM memory, for use if the system power is turned off.

Master code operation

When the master code has been entered and verified, the code branches to the master menu. This menu presents the option to change all of the codes and the time. Input from the user is parsed, and the appropriate sub-menu is displayed. Codes are changed using the change code functionality described above. The time change has not been implemented yet.

Continuous-use code operation

When the continuous code has been entered, the current state of the valve is checked. The valve state is stored in a register, and has states for closed, open-single, and open-continuous. Any other value of the 8-bit register is an error, and usually detecting an open-single state is as well (except inside the 'single use code entered' function). If the valve state is as expected, the valve is toggled from closed to open-continuous or from open-continuous to closed as appropriate. Also a display screen is displayed for a second, alerting the user of the state of the valve.

Single-use code operation

When the single-use code is entered, the state of the valve is checked for errors. If no errors are found, and the valve is not in the open-continuous state, the valve is open and a display screen is shown to the user with a progress bar which denotes the amount of time left until the valve will be closed. The progress bar fills one square a second, and when the bar is full the valve is shut off, and the program jumps back to the main prompt.

6 AM reset

At 6 AM, the alarm on the clock chip will send an interrupt signal to the PIC. The systems will leave whatever state it is in and enter the 6 AM reset state. The single-use code is reset and the valve, if on, is turned off. The LCD displays a message informing the user what is happening. If the system is powered off at 6 AM, the reset will occur when the system regains power.

Results

We met all of the specifications in the proposal. The one change is that instead of the single-use code opening the valve for 20 seconds, it opens it for an adjustable length of time, currently set to 11 seconds. Moisture considerations will be addressed by sealing the circuitry in a box with a desiccant. Installation into a refrigerator has not yet taken place, but will occur early next semester. The hardest part of the project was physical construction.

References

MGR1513-ND Datasheet ftp://Key:mat@ftp.ambrit.co.uk/technicalspecs/1000_low.pdf
CFAH2004A-AGB-JP Datasheet
http://www.crystalfontz.com/products/2004a-color/CFAH2004AAGBJP.pdf
BQ4845P-A4 Datasheet http://www-s.ti.com/sc/ds/bq4845.pdf
TIP120 Datasheet http://www.fairchildsemi.com/ds/TI/TIP120.pdf

Parts List

Part	Source	Vendor Part #	Price
Keypad	DigiKey	MGR1513-ND	65.00
LCD	Crystalfontz	CFAH2004A-AGB-JP	25.21
RTC	DigiKey	BQ4845P-A4	5.02
32.768KHz Crystal	DigiKey	SE3201-ND	0.27
3V Battery	DigiKey	P192-ND	1.68
Heatsink	DigiKey	294-1067-ND	1.63
Solenoid Valve	McMaster-Carr	7877K55	18.12
TIP120	Prof. Harris' Lab	TIP120	-
3W Resistor	Prof. Harris' Lab	-	-

Appendix A: Instruction Manual for KegLock(DK)

Codes and their functions:

Master Code: Allows all codes to be set, and allows the single-use and continuous use codes to be reset (so no code is stored for them). Also allows the time to be set. The master code is 6 digits long.

Continuous Code: Opens the valve. The valve will stay open until the continuous code is reentered or until 6 AM. The continuous code is 6 digits long.

Single-Use Code: Opens the valve for 11 seconds. Will not work if the valve is already open from the continuous code. The single-use code is 4 digits long.

Initialization:

Press the red reset button contained within the circuit box. The LCD will prompt for mater code entry. Enter the desired 6-digit master code, followed by #. Re-enter the code as prompted. If the codes do not match, or the wrong length of code is entered, the LCD will display an error message and code entry with start over. Once the same 6-digit code has been entered twice, that code will be stored as the master code. This process can be used if the master code is forgotten.

Code entry:

To enter a code, press the 4 or 6 digits of the code, followed by the # key. If the code entered does not match a code stored in memory, an error message will display. Also, if a code that is not 4 or 6 digits long is entered, an error message will display.

Changing or resetting codes:

To change or reset a code, enter the master code. The master menu will display, with the options of changing the master code (1), continuous code (2), single-use code (3), or time (4). Press the appropriate key for the code to be changed. If the continuous or single-use code is selected, the LCD will display an option of either setting or resetting the code selected. Press * to reset the code or # to set the code. There is no option of resetting the master code, so if the master code is selected, the system will automatically enter code entry mode. If reset is selected, the code will be erased. If set is selected, the LCD will prompt for code entry. Enter the code twice to set the new code. If the codes match and are the appropriate length, the code entered will be saved.

Setting the time:

To change the time, enter the master code. Select option 4, time. Enter the time in 24 hour format, followed by #. If every digit of the time is not set, the digits not set will be zero.

Appendix B: FSM for selected functions

Key:

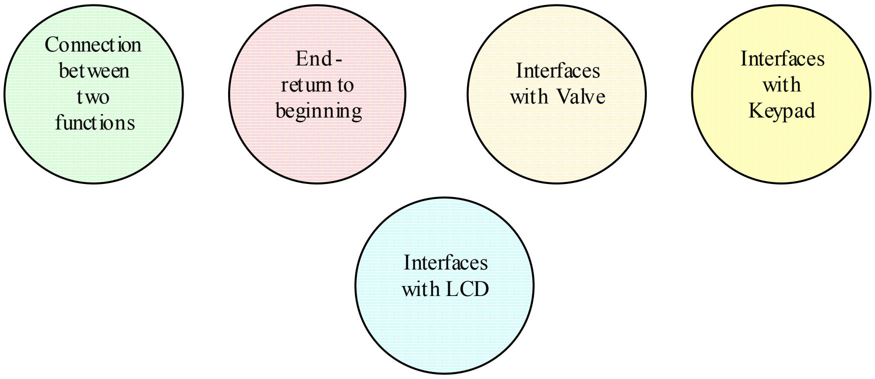


Figure 3 Recognize Codes

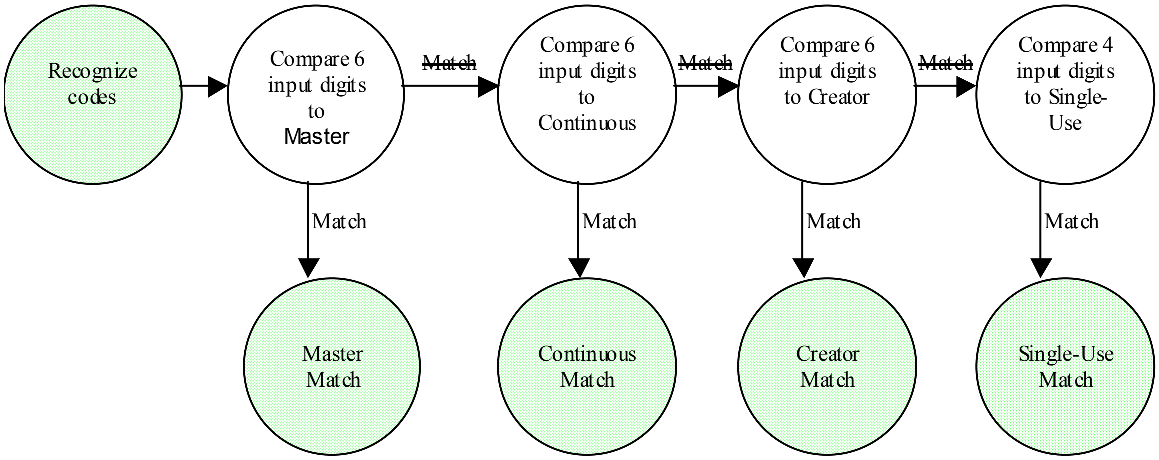

Recognize codes
Compare 6 input digits to Master
~~Match~~
Compare 6 input digits to Continuous
~~Match~~
Compare 6 input digits to Creator
~~Match~~
Compare 4 input digits to Single-Use
Match
Match
Match
Match
Master Match
Continuous Match
Creator Match
Single-Use Match

Figure 4 Operation when Continuous Use Code Entered

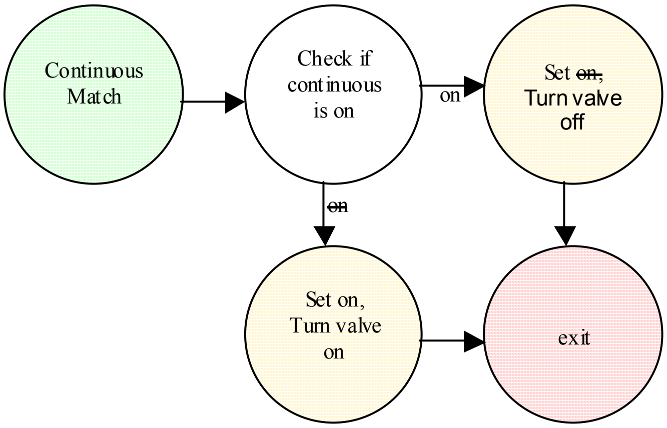


Figure 5 Operation when Master Code Entered

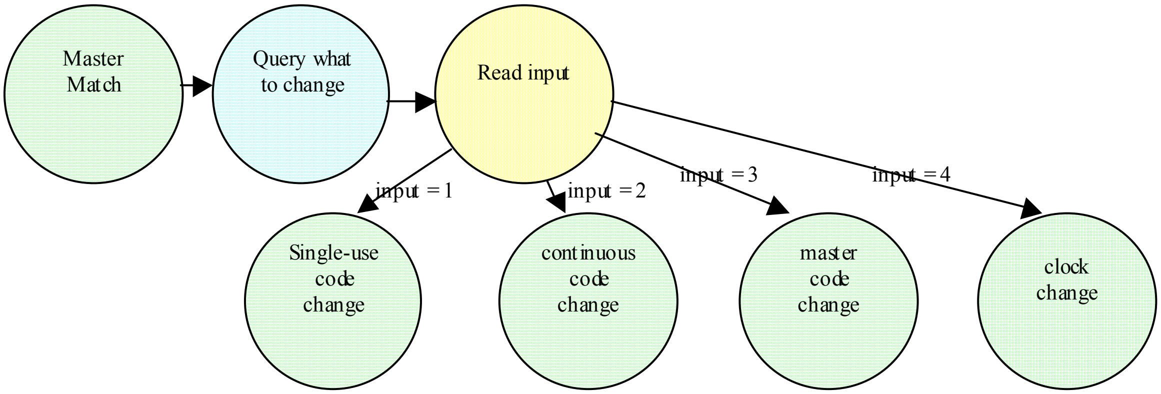


Figure 6 Operation when Single Use Code Entered

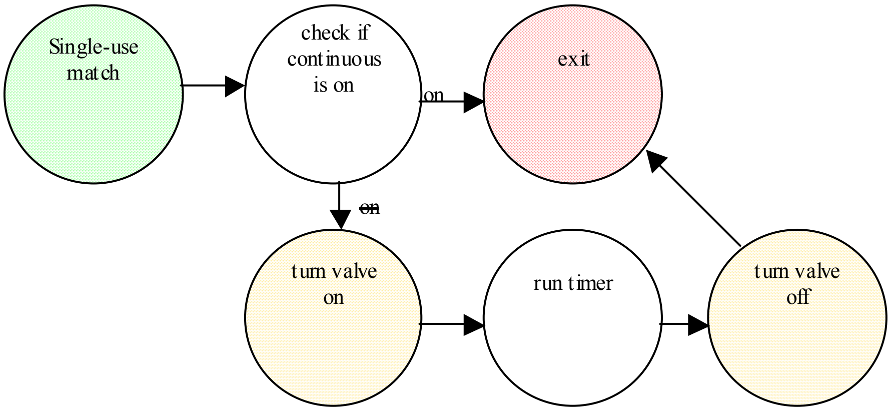
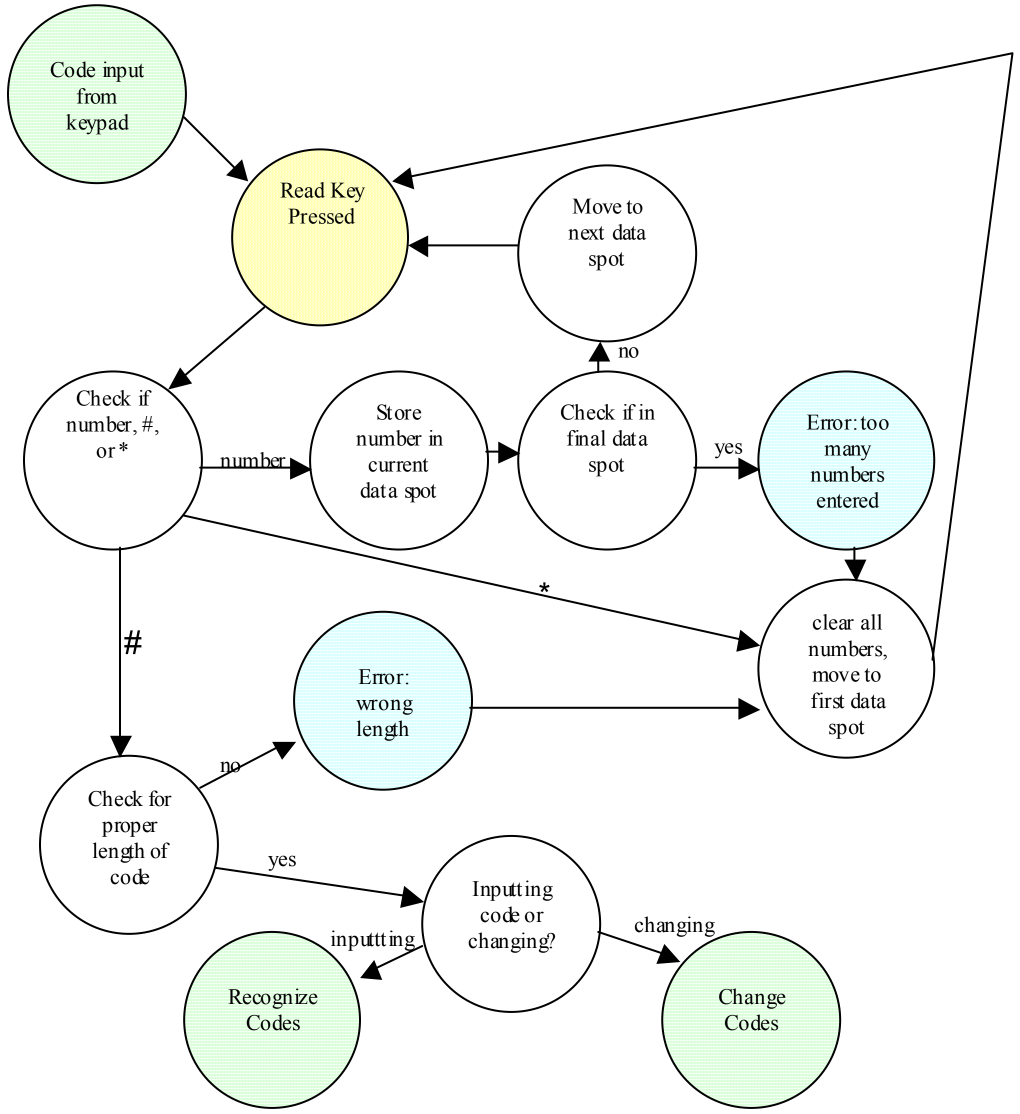


Figure 7 Code input from keypad



Code input from keypad
Read Key Pressed
Move to next data spot
no
Check if number, #, or *
number
Store number in current data spot
Check if in final data spot
yes
Error: too many numbers entered
*
#
clear all numbers, move to first data spot
Error: wrong length
no
Check for proper length of code
yes
Inputting code or changing?
inputtting
changing
Recognize Codes
Change Codes

Appendix C: PIC Code

```
; codefile.asm
; written 11/19/2003 by Damian_small@hmc.edu
; various psuedo codes for the MicroP's project

; Use the 18F452 PIC Microprocessor
       LIST p=18F452
       include "p18f452.inc"

; Constants file for the electronic lock project
       include "elconstants.inc"

       org 0x00
       bra            Initialize

       org 0x08
; high priority interrupt 36 commands till 0x50 maybe
       ; we should clear the stack. CLEAR THE STACK
       btfsc   INTCON,1               ; check 6am
       bra            SixAmInterrupt
       btfsc   INTCON3,0              ; check reset
       bra            ResetInterrupt
       ; otherwise, uh.... we're screwed
       bra            FatalError

ResetInterrupt:
       bcf            INTCON3,0              ; clear interrupt flag
       movlw   MCSMCDB
       movwf   WRLCDBREG
       movlw   MCSMCDR
       movwf   WRLCDDREG              ; setup display
       movlw   TIMEOFFV
       movwf   WRTDISPREG             ; disable time display
;      reset single
       movlw   singleCode1                   ; set the single code
       movwf   codeSet
       call    clearCode
;      reset continuous
       movlw   continCode1            ; set the continuous code
       movwf   codeSet
       call    clearCode
;      reset master   ( don't really need to do this, set already does)
       movlw   masterCode1            ; set the master code
       movwf   codeSet
       call    changeCodes
       bsf            INTCON, 7      ; enable high priority interrupts
       bra            Initialize             ; go to the main loop
SixAmInterrupt
       call    ClearAlarm             ; clear interrupt in clock chip
       bcf            INTCON,1               ; clear interrupt flag
       movlw   SIXAMB
       movwf   WRLCDBREG
       movlw   SIXAMR
       movwf   WRLCDDREG              ; setup display
; do important stuff
;      close valve
       bcf            LATE,2                 ; reset valve
       clrf    VALVEIND               ; reset indicator
;      reset single
       movlw   singleCode1                   ; set the single code
       movwf   codeSet
       call    clearCode

       call    WriteDisplay
       call    DelaySecond
       call    DelaySecond
       call    DelaySecond
       call    DelaySecond
       bsf            INTCON, 7      ; enable high priority interrupts
       bra            Initialize            ; go to the main loop

       org    0x80
; INITIALIZE
Initialize:
       ; setup io pins
       clrf   LATA
       clrf   TRISA  ; setup port A : output
       setf   TRISB  ; setup port B : input
       movlw  0x0F   ; set the 4 MSB's of B to output, 4 LSB's to input
       movwf  TRISC  ; setup port B : input/output
       clrf   LATD
       clrf   TRISD  ; setup port D : output
       bcf    LATE, 0
       bsf            LATE, 1 ; set port E, bit 1: ~CE for the clock
       bcf            LATE, 2
       clrf   TRISE  ; setup port E : output

       ;setup interrupts
       bcf            INTCON2,7      ; enable portB pull ups
       bsf            INTCON,4       ; enable INT0
       bcf            INTCON2,6      ; INT0 is on falling edge
;      bsf            blah blah      ; INT0 is always high priority
       bcf            INTCON,1       ; clear INT0 flag (maybe comment out)
       bsf            INTCON3,3      ; enable INT1
       bcf            INTCON2,5      ; INT1 is on falling edge
       bsf            INTCON3,6      ; INT1 high priority
       bcf            INTCON3,0      ; clear INT1 flag
       bsf            RCON, 7        ; enable interrupt priority
;      BSF            INTCON, 6      ; enable low priority interrupts
       bsf            INTCON, 7      ; enable high priority interrupts

       call   DelaySecond ; need to delay 1 second for clock chip

       clrf   VALVEIND       ; setup valve
       call   InitDisplay    ; initializeDisplay
       ; initialize keys
       call   InitClock      ; initialize clock chip
       call   CopyCode       ; initialize the codes from memory

       ; other setup
       movlw  maxlit
       movwf  maxreg         ; move the literal max into the file resiter for it
;start

Main:
idleState:
       movlw  MAININPUTB
       movwf  WRLCDBREG
       movlw  MAININPUTR
       movwf  WRLCDDREG              ; setup display
       movlw  TIMEONV
       movwf  WRTDISPREG             ; enable time display
       movlw  0x06                   ; put 6 in the wreg
       movwf  lengthCode             ; put it in lengthCode
       call   codeInput
       call   compareCodes
       movlw  0x00
       cpfsgt match                  ; compare match to zero
       bra    errorNoMatch   ; if it is zero, error
       movlw  singleCode1
       cpfseq match                  ; check if the single-use code matches
       bra    checkCon               ; if not, compare to continuous code
       bra    SingleCodeEntered      ; if matches, go to single-use action
checkCon:
       movlw  continCode1
       cpfseq match                  ; check if the continuous use code matches
       bra    MasterCodeEntered             ; if not,must be master or creator
       bra    ContinuousCodeEntered ; if matches, go to continuous action
errorNoMatch:
       ; display message saying that the entered code is no good
```

```
        movlw  ERRINPUTB
        movwf  WRLCDBREG
        movlw  ERRINPUTR
        movwf  WRLCDDREG             ; setup display
        call   WriteDisplay
        call   DelaySecond
        call   DelaySecond           ; write the display, then delay 2 seconds
        bra    idleState

; test
;Main
;       bra SingleCodeEntered
;       bra test4
;       bra Main

; test #4
test4:
        ; display enter code screen
        movlw  0x10
        movwf  WRLCDBREG
        movlw  0x00
        movwf  WRLCDDREG             ; setup initial display copy from flash
        ; update time

        movlw  0x2E
        movwf  WRTDISPREG
        call   WriteDisplay   ; set display
test4loop:
        call   DisplayTime           ; get time
        call   RefreshDisplay ; update display
        bra test4loop


; test #3
test3:
; REDACTED
        bra test3

; test #2
        movlw  0x04   ; bank 4
        movwf  WRLCDBREG
        movlw  0x00   ; start of bank 4
        movwf  WRLCDDREG

        lfsr   2,0x400
testloop2a
        movlw  0x20
        movwf  POSTINC2
        movlw  0x4F   ; end of line '4'
        cpfsgt FSR2L
        bra           testloop2a

        lfsr   2,0x400
testloop2b
        movlw  0x2A   ; * character
        movwf  INDF2
        call   WriteDisplay
        call   WriteDisplay
        call   WriteDisplay
        call   WriteDisplay
        call   WriteDisplay
        call   WriteDisplay
        call   WriteDisplay
        call   WriteDisplay
        movlw  0x20
        movwf  POSTINC2
        movlw  0x50
        cpfslt FSR2L
        clrf   FSR2L
        bra    testloop2b
```

```
; creatorMenu (MAIN CODE PATH)
CreatorCodeEntered:
       bra            MasterCodeEntered     ; go to the master menu

; masterMenu  (MAIN CODE PATH)
MasterCodeEntered:
;write display
       movlw   MCMAINDB
       movwf   WRLCDBREG
       movlw   MCMAINDR
       movwf   WRLCDDREG
       movlw   TIMEOFFV
       movwf   WRTDISPREG            ; disable time display
       call    WriteDisplay   ; set display
;get key
       call    keyInput
MCcheckSMC:
;      if (set mastercode) goto setMasterCodes
       movlw   MCSMCK
       cpfseq  inputDigit
       bra            MCcheckSCC
       bra            MCsetMasterCode
MCcheckSCC:
;      if (set continuoutsCode) goto setcontinuousCode
       movlw   MCSCCK
       cpfseq  inputDigit
       bra            MCcheckSSC
       bra            MCsetContinuousCode
MCcheckSSC:
;      if (set singleCode) goto setSingleCode
       movlw   MCSSCK
       cpfseq  inputDigit
       bra            MCcheckSTK
       bra            MCsetSingleCode
MCcheckSTK:
;      if (set time) goto setTime
       movlw   MCSTK
       cpfseq  inputDigit
       bra            MCcheckExit
       bra            MCsetTime
MCcheckExit:
;      if (exit) goto main prompt
       movlw   MCEXIT
       cpfseq  inputDigit
       bra            MCunknownKey
       bra            Main
MCunknownKey:
;      else goto masterMenu
       bra            MasterCodeEntered
MCsetMasterCode:
       movlw   MCSMCDB
       movwf   WRLCDBREG
       movlw   MCSMCDR
       movwf   WRLCDDREG
       call    WriteDisplay          ; set display
       movlw   masterCode1           ; set the master code
       movwf   codeSet
       call    changeCodes
       bra            MasterCodeEntered     ; return to master menu
MCsetContinuousCode:
       movlw   continCode1           ; set the continuous code
       movwf   codeSet
       ; ask to set or reset
       movlw   MCSRCCDB
       movwf   WRLCDBREG
       movlw   MCSRCCDR
       movwf   WRLCDDREG
       call    WriteDisplay          ; set display
MCSetResetCon:
       ;get key
```

```
        call    keyInput
        movlw   star
        cpfseq  inputDigit
        bra             MCCheckPoundC
        bra     MCResetCon                   ; reset code
MCCheckPoundC:
        movlw   pound
        cpfseq  inputDigit
        bra             MCSetResetCon        ; get another key
        bra             MCSetCon
MCResetCon:
        call    clearCode
        bra             MasterCodeEntered    ; return to master menu
MCSetCon:
        movlw   MCSCCDB
        movwf   WRLCDBREG
        movlw   MCSCCDR
        movwf   WRLCDDREG
        call    WriteDisplay          ; set display
        call    changeCodes
        bra             MasterCodeEntered    ; return to master menu
MCsetSingleCode:
        movlw   singleCode1                  ; set the single code
        movwf   codeSet
        ; ask to set or reset
        movlw   MCSRSCDB
        movwf   WRLCDBREG
        movlw   MCSRSCDR
        movwf   WRLCDDREG
        call    WriteDisplay          ; set display
MCSetResetSin:
        ;get key
        call    keyInput
        movlw   star
        cpfseq  inputDigit
        bra             MCCheckPoundS
        bra     MCResetCon                   ; reset code
MCCheckPoundS:
        movlw   pound
        cpfseq  inputDigit
        bra             MCSetResetSin        ; get another key
        bra             MCSetSin
MCResetSin:
        call    clearCode
        bra             MasterCodeEntered    ; return to master menu
MCSetSin:
        movlw   MCSSCDB
        movwf   WRLCDBREG
        movlw   MCSSCDR
        movwf   WRLCDDREG
        call    WriteDisplay          ; set display
        call    changeCodes
        bra             MasterCodeEntered    ; return to master menu
MCsetTime:
        movlw   MCSTDB
        movwf   WRLCDBREG
        movlw   MCSTDR
        movwf   WRLCDDREG
        call    WriteDisplay          ; set display
        movlw   0x00
        lfsr    FSR1, SETTINREG              ; set FSR1 to the start of the time
        movwf   POSTINC1
        movwf   POSTINC1
        movwf   POSTINC1
        movwf   POSTINC1
        movwf   POSTINC1
        movwf   POSTINC1                     ; clear the time input
        lfsr    FSR1, SETTINREG              ; set FSR1 to the start of the time
        movlw   SETTIMEC                     ; load the prompt character
        lfsr    FSR2, SETTDISPREG     ; set FSR2 to the start of the time display
        movwf   INDF2                        ; write the prompt to the screen
        call    RefreshDisplay
HoursTen:
        call    keyInput
        movlw   star
        cpfseq  inputDigit
        bra             HTCheckStar
        bra     MCsetTime                       ; clear input
HTCheckStar:
        movlw   pound
        cpfseq  inputDigit
        bra             HTCheckDigit
        bra             TimeEntered                     ; set the time
HTCheckDigit:
        movlw   0x03
        cpfslt  inputDigit
        bra             HoursTen                        ; inputDigit >= 3
        movf    inputDigit,0
        movwf   INDF1                           ; save input
        addlw   0x30                            ; convert to ascii
        movwf   POSTINC2                        ; write the input to the screen
        movlw   SETTIMEC
        movwf   INDF2                           ; write the prompt to the next space
        call    RefreshDisplay
HoursOne:
        call    keyInput
        movlw   star
        cpfseq  inputDigit
        bra             HOCheckStar
        bra     MCsetTime                       ; clear input
HOCheckStar:
        movlw   pound
        cpfseq  inputDigit
        bra             HOCheckDigit
        bra             TimeEntered                     ; set the time
HOCheckDigit:
        movlw   0x01
        cpfsgt  INDF1                           ; check if the HourTen is < 2
        bra             HoursGood                       ; hourTen < 2
        movlw   0x04
        cpfslt  inputDigit
        bra             HoursOne                        ; hoursOne >= 4
HoursGood:
        movf    POSTINC1, 0                     ; increment IND1
        movf    inputDigit,0
        movwf   POSTINC1                        ; save input
        addlw   0x30                            ; convert to ascii
        movwf   POSTINC2                        ; write the input to the screen
        movf    POSTINC2,0                      ; increment screen (skip ':')
        movlw   SETTIMEC
        movwf   INDF2                           ; write the prompt to the next space
        call    RefreshDisplay
MinutesTen:
        call    keyInput
        movlw   star
        cpfseq  inputDigit
        bra             MTCheckStar
        bra     MCsetTime                       ; clear input
MTCheckStar:
        movlw   pound
        cpfseq  inputDigit
        bra             MTCheckDigit
        bra             TimeEntered                     ; set the time
MTCheckDigit:
        movlw   0x06
        cpfslt  inputDigit
        bra             MinutesTen                      ; MinutesTen >= 6
        movf    inputDigit,0
        movwf   POSTINC1                        ; save input
        addlw   0x30                            ; convert to ascii
        movwf   POSTINC2                        ; write the input to the screen
        movlw   SETTIMEC
        movwf   INDF2                           ; write the prompt to the next space
        call    RefreshDisplay
MinutesOne:
        call    keyInput
        movlw   star
        cpfseq  inputDigit
        bra             MOCheckStar
        bra     MCsetTime                       ; clear input
MOCheckStar:
        movlw   pound
        cpfseq  inputDigit
        bra             MOCheckDigit
        bra             TimeEntered                     ; set the time
MOCheckDigit:
        movlw   0x0A
        cpfslt  inputDigit
        bra             MinutesOne                      ; MinutesOne >= 10
        movf    inputDigit,0
        movwf   POSTINC1                        ; save input
        addlw   0x30                            ; convert to ascii
        movwf   POSTINC2                        ; write the input to the screen
        movf    POSTINC2,0                      ; increment screen (skip '.')
        movlw   SETTIMEC
        movwf   INDF2                           ; write the prompt to the next space
        call    RefreshDisplay
SecondsTen:
        call    keyInput
        movlw   star
        cpfseq  inputDigit
        bra             STCheckStar
        bra     MCsetTime                       ; clear input
STCheckStar:
        movlw   pound
        cpfseq  inputDigit
        bra             STCheckDigit
        bra             TimeEntered                     ; set the time
STCheckDigit:
        movlw   0x06
        cpfslt  inputDigit
        bra             SecondsTen                      ; SecondsTen >= 6
        movf    inputDigit,0
        movwf   POSTINC1                        ; save input
        addlw   0x30                            ; convert to ascii
        movwf   POSTINC2                        ; write the input to the screen
        movlw   SETTIMEC
        movwf   INDF2                           ; write the prompt to the next space
        call    RefreshDisplay
SecondsOne:
        call    keyInput
        movlw   star
        cpfseq  inputDigit
        bra             SOCheckStar
        bra     MCsetTime                       ; clear input
SOCheckStar:
        movlw   pound
        cpfseq  inputDigit
        bra             SOCheckDigit
        bra             TimeEntered                     ; set the time
SOCheckDigit:
        movlw   0x0A
        cpfslt  inputDigit
        bra             SecondsOne                      ; SecondsOne >= 10
        movf    inputDigit,0
        movwf   INDF1                   ; save input
        addlw   0x30                            ; convert to ascii
        movwf   INDF2                   ; write the input to the screen
        movlw   SETTIMEC
        call    RefreshDisplay
        bra             SecondsOne              ; loop forever

TimeEntered:
       call    SetTime
       bra            MasterCodeEntered     ; return to master menu

; setTime
;display prompt
;loop:
;get keys
;      if (number && valid) // check if the number matches the
;current range (date, time, etc.)
;              enter number
;              advance to next digit
;      if (back && not at first digit)
;              go back
;      if (enter)
;              set time
;              goto masterMenu
;loop


; singleCode (MAIN CODE PATH)
; first do some error checking
SingleCodeEntered:
       movlw   VALVEOFF
       cpfseq  VALVEIND              ; check if valve off
       bra            SCcheck2
       bra            SCstartSingle
SCcheck2:
       movlw   VALVECONT
       cpfseq  VALVEIND              ; check if continuous already on
       bra            SCcheck3
       bra            SCcontinuousOn
SCcheck3:
       movlw   VALVESINGLE
       cpfseq  VALVEIND              ; check if single already on (error)
       bra            SCerrorUnknown ; error: unknown state
       bra            SCerrorSingle  ; error: single
SCerrorUnknown:
SCerrorSingle:
       bcf            LATE,2                 ; reset valve
       clrf    VALVEIND              ; reset indicator
SCcontinuousOn:
       ; display something?
       bra            Main
; now turn on the valve
SCstartSingle:
       movlw   SCDISPB
       movwf   WRLCDBREG
       movlw   SCDISPR
       movwf   WRLCDDREG             ; setup initial display copy from flash
       movlw   SCDISPPBB
       movwf   FSR2H                 ; setup progress bar FSR2H
       movlw   SCDISPPBS
       movwf   FSR2L                 ; setup progress bar FSR2L
       call    WriteDisplay   ; set display

       movlw   VALVESINGLE
       movwf   VALVEIND              ; set the valve indicator
       bsf            LATE,2                 ; turn valve on
;begin timing, time is length of progress bar +1
; so if a bar starts at 0x10 and ends at 0x10, then time is 2 seconds
; because the length is 1, +1 second
; This is NOT highly accurate: delay second will delay a second, plus
; you have the time to update the display
       call    DelaySecond
SCtimeloop:
       movlw   SCDISPPBC             ; load the progress bar character
       movwf   POSTINC2              ; write the bar character
       call    RefreshDisplay ; update the display
       call    DelaySecond
       movlw   SCDISPPBE
       cpfsgt  FSR2L                 ; check if at end of progress bar
```

```
        bra    SCtimeloop
; clean up and return to main loop
        bcf            LATE,2                 ; turn valve off
        clrf   VALVEIND               ; set the valve indicator to off
        bra    Main                   ;goto main prompt


; continuousCode (MAIN CODE PATH)
; first do some error checking
ContinuousCodeEntered:
        movlw  VALVEOFF
        cpfseq VALVEIND               ; check if valve off
        bra            CCcheck2
        bra            CCstartContinuous
CCcheck2:
        movlw  VALVECONT
        cpfseq VALVEIND               ; check if continuous already on
        bra            CCcheck3
        bra            CCcontinuousOn
CCcheck3:
        movlw  VALVESINGLE
        cpfseq VALVEIND               ; check if single already on (error)
        bra            CCerrorUnknown ; error: unknown state
        bra            CCerrorSingle  ; error: single
CCerrorUnknown:
CCerrorSingle:
        bcf            LATE,2                 ; reset valve
        clrf   VALVEIND               ; reset indicator
        bra            Main
CCcontinuousOn:                              ; turn off the valve
        bcf            LATE,2                 ; reset valve
        clrf   VALVEIND               ; reset indicator
        movlw  CCOFFDISPB
        movwf  WRLCDBREG
        movlw  CCOFFDISPR
        movwf  WRLCDDREG
        call   WriteDisplay   ; set display
        call   DelaySecond
        call   DelaySecond
        call   DelaySecond
        call   DelaySecond
        bra            Main
CCstartContinuous:                    ; turn on the valve
        movlw  VALVECONT
        movwf  VALVEIND               ; set the indicator
        bsf            LATE,2                 ; turn on the valve
        movlw  CCONDISPB
        movwf  WRLCDBREG
        movlw  CCONDISPR
        movwf  WRLCDDREG
        call   WriteDisplay   ; set display
        call   DelaySecond
        call   DelaySecond
        call   DelaySecond
        call   DelaySecond
        bra    Main


FatalError
        movlw  FATALB
        movwf  WRLCDBREG
        movlw  FATALR
        movwf  WRLCDDREG              ; setup display
        call   WriteDisplay
fataloop:
        nop
        nop
        nop
        bra            fataloop

; include subroutines
        ; Kim's subroutines
        include "codeControl.inc"
        include "keypadControl.inc"

        ; Damian's subroutines
        include "displayControl.inc"
        include "timerControl.inc"

        include "displays.inc"

        end
```

```
; codeControl.inc
; written 12/4/03 by Damian_small@hmc.edu
; contains subroutines for controlling codes, originally
; written by Kim_Shultz@hmc.edu
; codeInput, compareCodes, changeCodes, clearCodes, startup

; startup
; written 12/5 by kim_shultz@hmc.edu
; copy the codes from the eeprom

; SUBROUTINE
CopyCode:
        clrf FSR0H
        movlw masterCode1
        movwf FSR0L                   ; write the master code
        movwf EEADR                   ; read the master code
        movlw 0x06
        movwf count                   ; put 6 in the count register
        call readLoop
        movlw continCode1
        movwf FSR0L                   ; write the contin code
        movwf EEADR                   ; read the contin code
        movlw 0x06
        movwf count                   ; put 6 in the count register
        call readLoop
        movlw singleCode1
        movwf FSR0L                   ; write the single code
        movwf EEADR                   ; read the single code
        movlw 0x06
        movwf count                   ; put 6 in the count register
        call readLoop
        bra readCreator

readLoop
        bcf EECON1, EEPGD     ; point to DATA memory
        bcf EECON1, CFGS      ; access program FLASH of data EEPROM memory
        bsf EECON1, RD        ; EEPROM read
        movff EEDATA, POSTINC0 ; put the data from memory into the file register
        incf EEADR                    ; read from the next mem location
        decfsz count          ; decrement count
        bra readLoop          ; if count above 0, repeat
        return

readCreator
        movlw creatorCode1
        movwf FSR0L                   ; write the creator code
        movlw 0xXX
        movwf POSTINC0
        movlw 0xXX
        movwf POSTINC0
        movlw 0xXX
        movwf POSTINC0
        movlw 0xXX
        movwf POSTINC0
        movlw 0xXX
        movwf POSTINC0
        movlw 0xXX
        movwf POSTINC0
        return



; codeInput.inc
; written 11/22/03 by kim_shultz@hmc.edu
; get input from user and store in input registers
; NOTE: writeDisplay and DsiplayTime use FSR0, so this subroutine uses
; FSR1 and FSR2
codeInput
        ; for codeInput
        clrf   FSR1H                  ; ensure the high bits of FSR1 are 0
```

```
clearInput                                   ; clear the input registers
        movff  WRLCDBREG, TEMPREGB
        movff  WRLCDDREG, TEMPREGR   ; save the display
        call   WriteDisplay
        call   DisplayTime
        call   RefreshDisplay ; display the cleared input with the time
        movlw  inputCode1
        movwf  FSR1L                 ; put the pointer at the beginning
        lfsr   FSR2, CODESTART ; put the code field into FSR2L (to display *'s)
        movlw  0x06                  ; put the number of registers in the wreg
        movwf  count                 ; move it to the count register
        movlw  0x0A                  ; put A in the wreg

clearLoop
        movwf  POSTINC1              ; put A in the input data spot
        decfsz count                 ; decrement count, skip if zero
        bra    clearLoop             ; repeat and clear the next spot

        movlw  inputCode1
        movwf  FSR1L                 ; put the pointer at the beginning

getKey
        call   keyInput              ; get a key input from the user

        movlw  0x0A                  ; put A in the wreg
        cpfsgt inputDigit
        bra            numInput              ; if input<A, then it is a number
        movlw  pound
        cpfseq inputDigit
        bra    clearInput            ; if input is neither a number or #,
                                                    ; only other valid input is *
        bra    poundInput            ; if input is #, branch

numInput
        movf   lengthCode,0   ; put the length of the code in the wreg
        cpfslt count                 ; compare the length to be inputted
                                                    ; to the length that has been inputted
        bra    errorTooMany   ; if count is not less than the input length,
                                                    ; too many have been inputted
        movff  inputDigit, POSTINC1 ; put the input in the inputCode file register
                                                    ; point to the next file register
        incf   count                 ; increment the count register
        movlw  CODECHAR
        movwf  POSTINC2              ; display the code character on the LCD
        bra    getKey                ; get the next input key

poundInput
        movf   lengthCode,0   ; put the length of code looking for in the wreg
        cpfslt count                 ; compare the length to be inputted
                                                    ; to the length that has been inputted
        bra    codeEntered           ; if count is not less than the input length
        movlw  0x04                  ; put 4 in the wreg
        cpfseq count                 ; compare the number of inputted digits to 4
        bra    errorWrongNum  ; if not equal to 4 (or 6, from above), then error
        clrf   lengthCode            ; clear lengthCode to tell that 4 digits entered

codeEntered
        return                               ; return when code has been entered

errorTooMany
errorWrongNum
        ; display in bank 13 0xC0
        ; need a message here
        ; should say "You entered the wrong number of digits. Please try again"
        movlw  ERRLENB
        movwf  WRLCDBREG
        movlw  ERRLENR
        movwf  WRLCDDREG             ; setup display
        call   WriteDisplay
        call   DelaySecond
        call   DelaySecond           ; write the display, then delay 2 seconds
        movff  TEMPREGB, WRLCDBREG
        movff  TEMPREGR, WRLCDDREG    ; restore the display
        bra    clearInput             ; clear the input spots


; compareCodes.asm
; written 11/18/03 by kim_shultz@hmc.edu
; recognize inputted codes
compareCodes
; initialize variables

        clrf FSR0H            ; ensure the high bits of FRS0 are 0's
        clrf FSR1H            ; ensure the high bits of FRS1 are 0's
        clrf match            ; clear match

compareMaster
        movlw inputCode1      ; store the address of the input code in FSR0
        movwf FSR0L
        movlw masterCode1     ; store the address of the master code in FSR1
        movwf FSR1L
        clrf count            ; put 0 in the count register
loopMaster
        incf count            ; increment count
        movf POSTINC0,0               ; put the input digit in the wreg,
                                             ; point to the next input digit
        cpfseq POSTINC1               ; compare to the master digit, point to the next digit
        bra compareContin     ; if the digits do not match,
                                             ; compare to the continuous code
        movlw 0x06                    ; put six in the wreg
        cpfseq count          ; compare the count to six (in the wreg)
        bra loopMaster        ; if the loop has not been iterated six times, repeat
                                               ; if the loop has found six matches, the codes match
        movlw masterCode1     ; put the master match flag in the wreg
        movwf match                   ; put the master match flag into match
        bra matchDone

compareContin
        movlw inputCode1      ; store the address of the input code in FSR0
        movwf FSR0L
        movlw continCode1     ; store the address of the continuous code in FSR1
        movwf FSR1L
        clrf count                    ; put 0 in the count register
loopContin
        incf count                    ; increment count
        movf POSTINC0,0               ; put input digit in wreg, point to next input digit
        cpfseq POSTINC1               ; compare to the continuous digit, point to next digit
        bra compareCreator    ; if the digits do not match, compare to creator code
        movlw 0x06                    ; put six in the wreg
        cpfseq count          ; compare the count to six (in the wreg)
        bra loopContin        ; if the loop has not been iterated six times, repeat
                                               ; if the loop has found six matches, the codes match
        movlw continCode1     ; put the continuous match flag in the wreg
        movwf match                   ; put the continuous match flag into match
        bra matchDone

compareCreator
        movlw inputCode1      ; store the address of the input code in FSR0
        movwf FSR0L
        movlw creatorCode1    ; store the address of the creator code in FSR1
        movwf FSR1L
        clrf count                    ; put 0 in the count register
loopCreator
        incf count                    ; increment count
        movf POSTINC0,0               ; put the input digit in the wreg, point to the next input digit
        cpfseq POSTINC1               ; compare to the creator digit, point to the next digit
        bra compareSingle     ; if the digits do not match, compare to the single-use code
        movlw 0x06                    ; put six in the wreg
        cpfseq count          ; compare the count to six (in the wreg)
        bra loopCreator               ; if the loop has not been iterated six times, repeat
                                               ; if the loop has found six matches, the codes match
        movlw creatorCode1    ; put the creator match flag in the wreg
        movwf match                   ; put the creator match flag into match
        bra matchDone

compareSingle
        movlw inputCode1      ; store the address of the input code in FSR0
        movwf FSR0L
        movlw singleCode1     ; store the address of the single-use code in FSR1
        movwf FSR1L
        clrf count                    ; put 0 in the count register
loopSingle
        incf count                    ; increment count
        movf POSTINC0,0               ; put the input digit in the wreg, point to the next input digit
        cpfseq POSTINC1               ; compare to the single-use digit, point to the next digit
        bra matchDone         ; if the digits do not match, no codes match
        movlw 0x06                    ; put six in the wreg
        cpfseq count          ; compare the count to six (in the wreg)
                                             ; the last two comparisons are to ensure that only 4
digits have been entered
        bra loopSingle        ; if the loop has not been iterated six times, repeat
                                             ; if the loop has found six matches, the codes match
        movlw singleCode1     ; put the continuous match flag in the wreg
        movwf match                   ; put the continuous match flag into match

matchDone
        return


; changeCodes.inc
; written 11/18/03 by kim_shultz@hmc.edu
; change the codes
changeCodes
        movff  WRLCDBREG, TEMPREGB
        movff  WRLCDDREG, TEMPREGR    ; save the display
        movlw 0x04                    ; put the length of the single-use code in the wreg
        movwf lengthCode      ; put it in lengthCode
        movlw singleCode1     ; put the address of the single-use code in the wreg
        cpfseq codeSet        ; check if the single-use code is being input
        bra longCode          ; if it is not, the lengthCode needs to be 6
lengthDone
        call codeInput        ; get code input
        movlw 0x00                    ; put 0 in the wreg
        cpfsgt lengthCode     ; check if lengthCode is 0
        bra errorWrongNumS    ; if == zero, then 4 digits were entered for a 6 code
        call compareCodes     ; check if the inputted code matches an already inputted code

        movlw 0x00                    ; put 0x00 in the wreg
        cpfsgt match          ; if no match has been found, set the code
        bra setTheCode
        movf match,0          ; put the code matched in the wreg
        cpfseq codeSet
        bra errorMatch        ; if the code matches a different code, this code cannot be set
;       bra doneSetting               ; if the code matched is the code being set, the code does not
need to be set
        bra setTheCode        ; if the code matched is the code being set, we still need to set the
code again

longCode
        movlw 0x06                    ; put the length of the long codes in the wreg
        movwf lengthCode      ; put it in lengthCode
        bra lengthDone

setTheCode
        movlw singleCode1     ; put the single-use code address in the wreg
        cpfseq codeSet        ; if the single use code is being set, continue
        bra setInit                   ; otherwise, copy the code to the appropriate location
        movlw 0x0A                    ; put 0x0A into the wreg
        movwf inputCode5
        movwf inputCode6      ; set the last two bits of the inputted code to A

setInit
        clrf FSR0H                    ; ensure the high bits of FRS0 are 0's
        clrf FSR1H                    ; ensure the high bits of FRS1 are 0's
        movlw inputCode1       ; put the address of the inputted code in the wreg
        movwf FSR0L                   ; put the address of the inputted code in FRS0
        movf codeSet,0         ; put the address of the code to set in the wreg
        movwf FSR1L                   ; put the address of the code to set in FSR1
        movlw 0x06                    ; put 6 in the wreg
        clrf count                    ; clear the count register

setLoop
        movff POSTINC0,POSTINC1       ; move the inputted digit to the proper location
        incf count                    ; increment the count
        cpfseq count           ; if the loop has run 6 times, finish
        bra setLoop                   ; if the loop has not yet run 6 times, exit
        bra doneSetting

errorMatch
        ; need display to say that that code is already taken
        ; display in bank 13 0x70
        movlw   ERRMATCHB
        movwf   WRLCDBREG
        movlw   ERRMATCHR
        movwf   WRLCDDREG             ; setup display
        call    WriteDisplay
        call    DelaySecond
        call    DelaySecond           ; write the display, then delay 2 seconds
        movff   TEMPREGB, WRLCDBREG
        movff   TEMPREGR, WRLCDDREG   ; restore the display
        bra     changeCodes

doneSetting
        clrf match                    ; remove the match flag, if anything matched
repeatInput
        ; get the user to re-input the code
        ; need display to tell the user that
        ; display in bank 13 0x20
        movlw   ENTERAGAINB
        movwf   WRLCDBREG
        movlw   ENTERAGAINR
        movwf   WRLCDDREG             ; setup display
        call    WriteDisplay
        call    DelaySecond
        call    DelaySecond           ; write the display, then delay 2 seconds
        movff   TEMPREGB, WRLCDBREG
        movff   TEMPREGR, WRLCDDREG   ; restore the display

        movlw   0x06                  ; put 6 in the wreg
        movwf   lengthCode            ; put it in lengthCode
        call codeInput         ; get input again
        call compareCodes      ; compare the new input to the codes
        movf codeSet,0         ; put the code to be set in the wreg
        cpfseq match           ; see if the new code matches the code being changed
        bra misMatch           ; if they don't match, error

        ; copy to permanent memory
copyToMem:
        clrf FSR0H
        movf codeSet,0
        movwf EEADR                   ; put the address of the code being set in the data address spot
        addlw 0x06                    ; find the location after the code spot
        movwf count                   ; store it in the count register
        movff codeSet, FSR0L; put the address of the code in FSR0
        bcf EECON1, CFGS       ; access program flash or data EEPROM memory
        bcf EECON1, EEPGD      ; point to DATA memory
        bcf INTCON, GIE               ; disable interrupts
        bsf EECON1, WREN       ; enable writes
memLoop:
        movf    POSTINC0,0
        movwf   EEDATA         ; put the data in EEDATA
        movlw   0x55           ; the following is from the data sheet
        movwf   EECON2         ; write 0x55
        movlw   0xAA
        movwf   EECON2        ; write 0xAA
        bsf     EECON1, WR    ; set the write bit to begin write
memWait:
        btfsc EECON1, WR      ; wait for write to complete
        bra memWait

        incf EEADR                    ; point to the next address in memory
        movf count,0          ; put the count contents in the wreg
        cpfseq FSR0L                  ; compare to the current mem address
        bra memLoop                   ; if not the same, repeat

doneWriting:
        bcf EECON1, WREN      ; disable writes
        clrf codeSet          ; to code has been set, so remove the flag saying to change that code
        bsf INTCON,GIE        ; reenable interrupts
        return

misMatch
        ; need to display an error message
        ; repeat the code changing process
        movlw   MISMATCHB
        movwf   WRLCDBREG
        movlw   MISMATCHR
        movwf   WRLCDDREG             ; setup display
        call    WriteDisplay
        call    DelaySecond
        call    DelaySecond           ; write the display, then delay 2 seconds
        movff   TEMPREGB, WRLCDBREG
        movff   TEMPREGR, WRLCDDREG   ; restore the display
        bra     changeCodes

errorWrongNumS
        ; display in bank 13 0xC0
        ; should say "You entered the wrong number of digits. Please try again"
        movlw   ERRLENB
        movwf   WRLCDBREG
        movlw   ERRLENR
        movwf   WRLCDDREG             ; setup display
        call    WriteDisplay
        call    DelaySecond
        call    DelaySecond           ; write the display, then delay 2 seconds
        movff   TEMPREGB, WRLCDBREG
        movff   TEMPREGR, WRLCDDREG   ; restore the display
        bra     changeCodes           ; start changing codes again


; clearCodes
; written 12/5 by kim_shultz@hmc.edu
; clear a code

; code to be cleared should have its address stored in codeSet

clearCode
        clrf FSR0H
        movf codeSet,0 ; put the address of the code to be reset in the wreg
        movwf FSR0L           ; point FSR0 at the code
        movlw 0x06            ; put 6 in the wreg
        movwf count           ; put 6 in the count register
        movlw 0x0A            ; put 0x0A in the wreg
clearCodeLoop
        movwf POSTINC0 ; put 0x0A in the current address, point to next address
        decfsz count   ; decrement the count register
        bra clearCodeLoop     ; if not 0, repeat

        call copyToMem
        return
```

```
; displayControl.inc
; written 12/02/2003 by Damian_small@hmc.edu
; display control routines for the MicroP's project

; SUBROUTINE CheckBF
; checks the display flag, and waits
; until the display is ready for the next instruction
CheckBF:
        setf    TRISD
        movlw   LCDCHECKBFA
        movwf   LATA
;       bsf            LATE,1  ; disable chip
        bsf            LATE,0  ; enable LCD
cbfloop:
        btfsc   PORTD, 7
        bra            cbfloop
        bcf            LATE,0
        clrf    LATA
        clrf    TRISD
        return

; SUBROUTINE WriteDisplay
; NOTE: the difference between WriteDisplay and RefreshDisplay
; is that RefreshDisplay does not recopy the data from flash
; memory, allowing the display to be changed in data memory.
; WriteDisplay automatically calls RefreshDisplay.
; uses FSR0
WriteDisplay:
        ; copy data to copy location
        lfsr    0,WRLCDTEMP    ; start of storage location in FSR0
        movf    WRLCDBREG,0    ; set temporary storage destination
;       movwf   FSR1H
        clrf    TBLPTRU
        movwf   TBLPTRH
        movf    WRLCDDREG,0
        movwf   TBLPTRL        ; setup table read pointer
;       movwf   FSR1L          ; load FSR1 with the location of the data
        ; copy over data: line 1
        movlw   0x13    ; end of line1
wrlcdcopy1:
;       movff   POSTINC1, POSTINC0 ; copy data to temp store
        tblrd*+                                    ; read table pointer, postinc
        movff   TABLAT, POSTINC0       ; copy the table data to data memory, postinc
        cpfsgt  FSR0L
        bra            wrlcdcopy1
        movlw   0x28
        movwf   FSR0L   ; go to the '3rd' line
        movlw   0x3B    ; end of line '3'
wrlcdcopy2:
;       movff   POSTINC1, POSTINC0 ; copy data to temp store
        tblrd*+                        ; read table pointer, postinc
        movff   TABLAT, POSTINC0       ; copy the table data to data memory, postinc
        cpfsgt  FSR0L
        bra            wrlcdcopy2
        movlw   0x14
        movwf   FSR0L   ; go to the '2nd' line
        movlw   0x27    ; end of line '2'
wrlcdcopy3:
;       movff   POSTINC1, POSTINC0 ; copy data to temp store
        tblrd*+                        ; read table pointer, postinc
        movff   TABLAT, POSTINC0       ; copy the table data to data memory, postinc
        cpfsgt  FSR0L
        bra            wrlcdcopy3
        movlw   0x3C
        movwf   FSR0L   ; go to the '4th' line
        movlw   0x4F    ; end of line '4'
wrlcdcopy4:
;       movff   POSTINC1, POSTINC0 ; copy data to temp store
        tblrd*+                        ; read table pointer, postinc
        movff   TABLAT, POSTINC0       ; copy the table data to data memory, postinc
        cpfsgt  FSR0L
        bra            wrlcdcopy4

; SUBROUTINE RefreshDisplay
; refreshes the LCD from the temporary data memory location
; uses FSR0
RefreshDisplay:
        ; begin init display
        call CheckBF
        movlw  LCDRETURN      ; return the cursor to the home position
        movwf  LATD
        bsf            LATE,0
        bcf            LATE,0
        lfsr   0,WRLCDTEMP    ; start of storage location in FSR0
wrloop:                       ; write data to LCD
        call   CheckBF
        movlw  LCDDATAWRA
        movwf  LATA
        movff  POSTINC0, LATD
        bsf            LATE,0  ; enable LCD
        bcf            LATE,0
        movlw  0x4F    ; end of line 4
        cpfsgt FSR0L
        bra            wrloop
        return

; SUBROUTINE InitDisplay
; initializes the display
InitDisplay:
        setf   WREG
        ; wait 15 ms
delay1:
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz WREG
        bra            delay1
        ; first init
        movlw  LCDINIT1
        movwf  LATD
        bsf            LATE,0
        bcf            LATE,0
        ; wait 4.1ms
        setf   WREG
delay2:
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz WREG
        bra            delay2
        ; second init
        movlw  LCDINIT1
        movwf  LATD
        bsf            LATE,0
        bcf            LATE,0
        ; wait 100us
        setf   WREG
delay3:
        decfsz WREG
        bra            delay3
        ; third init
        movlw  LCDINIT1
        movwf  LATD
        bsf            LATE,0  ; enable LCD
        bcf            LATE,0
; use check BF from now on
        call   CheckBF
        ; fourth init
        movlw  LCDINIT2
        movwf  LATD
        bsf            LATE,0  ; enable LCD
        bcf            LATE,0
        call   CheckBF
        ; fifth init
        movlw  LCDINIT3
        movwf  LATD
        bsf            LATE,0  ; enable LCD
        bcf            LATE,0
        call   CheckBF
        ; sixth init
        movlw  LCDINIT4
        movwf  LATD
        bsf            LATE,0  ; enable LCD
        bcf            LATE,0
        call   CheckBF
        return
```

```
; keypadControl originally keyInput.inc
; written 11/23/03 by kim_shultz@hmc.edu
; poll the keypad to get input

; returns the key pressed in file register 0x0C
; stores digits 0-9 in hex

; bits 0-3 of PORTC are row inputs A through D
; bits 4-6 of PORTC are column outputs 1-3

keyInput
       ; poll the first column
       call    DisplayTime
       call    RefreshDisplay ; this delays ~40ms, so no delay needed
       movlw MSB1low
       movwf PORTC                    ; pull column 1 low
;;     call pollDelay         ; delay to avoid bounce
       btfss PORTC,0          ; check if row A is high
       bra rowAcol1           ; if not, row A col 1 is the key pressed
       btfss PORTC,1          ; check if row B is high
       bra rowBcol1           ; if not, row B col 1 is the key pressed
       btfss PORTC,2          ; check if row C is high
       bra rowCcol1           ; if not, row C col 1 is the key pressed
       btfss PORTC,3          ; check if row D is high
       bra rowDcol1           ; if not, row D col 1 is the key pressed
       ; poll the second column
       movlw MSB2low
       movwf PORTC                    ; pull column 2 low
;;     call pollDelay         ; delay to avoid bounce
       btfss PORTC,0          ; check if row A is high
       bra rowAcol2           ; if not, row A col 2 is the key pressed
       btfss PORTC,1          ; check if row B is high
       bra rowBcol2           ; if not, row B col 2 is the key pressed
       btfss PORTC,2          ; check if row C is high
       bra rowCcol2           ; if not, row C col 2 is the key pressed
       btfss PORTC,3          ; check if row D is high
       bra rowDcol2           ; if not, row D col 2 is the key pressed
       ; poll the third column
       movlw MSB3low
       movwf PORTC                    ; pull column 3 low
;;     call pollDelay         ; delay to avoid bounce
       btfss PORTC,0          ; check if row A is high
       bra rowAcol3           ; if not, row A col 3 is the key pressed
       btfss PORTC,1          ; check if row B is high
       bra rowBcol3           ; if not, row B col 3 is the key pressed
       btfss PORTC,2          ; check if row C is high
       bra rowCcol3           ; if not, row C col 3 is the key pressed
       btfss PORTC,3          ; check if row D is high
       bra rowDcol3           ; if not, row D col 3 is the key pressed

       bra keyInput           ; if all rows are high, repeat polling

rowAcol1
       movlw 0x01                     ; row A column 1 is 1
       movwf inputDigit
       bra releaseRowA
rowBcol1
       movlw 0x04                     ; row B column 1 is 4
       movwf inputDigit
       bra releaseRowB
rowCcol1
       movlw 0x07                     ; row C column 1 is 7
       movwf inputDigit
       bra releaseRowC
rowDcol1
       movlw star                     ; row D column 1 is *
       movwf inputDigit
       bra releaseRowD
rowAcol2
       movlw 0x02                     ; row A column 2 is 2
       movwf inputDigit
```

```
        bra releaseRowA
rowBcol2
        movlw 0x05                   ; row B column 2 is 5
        movwf inputDigit
        bra releaseRowB
        return
rowCcol2
        movlw 0x08                   ; row C column 2 is 8
        movwf inputDigit
        bra releaseRowC
rowDcol2
        movlw 0x00                   ; row D column 2 is 0
        movwf inputDigit
        bra releaseRowD
rowAcol3
        movlw 0x03                   ; row A column 3 is 3
        movwf inputDigit
        bra releaseRowA
rowBcol3
        movlw 0x06                   ; row B column 3 is 6
        movwf inputDigit
        bra releaseRowB
rowCcol3
        movlw 0x09                   ; row C column 3 is 9
        movwf inputDigit
        bra releaseRowC
rowDcol3
        movlw pound                  ; row D column 3 is #
        movwf inputDigit
        bra releaseRowD

releaseRowA
        btfss PORTC, 0        ; check if the key has been released
        bra releaseRowA              ; if not, repeat
        return                       ; if it has, finish
releaseRowB
        btfss PORTC, 1        ; check if the key has been released
        bra releaseRowB              ; if not, repeat
        return                       ; if it has, finish
releaseRowC
        btfss PORTC, 2        ; check if the key has been released
        bra releaseRowC              ; if not, repeat
        return                       ; if it has, finish
releaseRowD
        btfss PORTC, 3        ; check if the key has been released
        bra releaseRowD              ; if not, repeat
        return                       ; if it has, finish

; uses displayTime to delay ~40ms (with display write)
; so no delay loop needed
;pollDelay
        ; delay ~5 ms to avoid bounce
;       movlw 0                                     ; set the wreg to 0
;loopPoll                                           ; loop for delay
;              addlw 1                      ; increment the wreg
;              cpfseq maxreg  ; if the loop has been iterated max times, exit loop
;              bra loopPoll          ; if not, repeat the loop
;       return
; timerControl.inc
; written 12/02/2003 by Damian_small@hmc.edu
; timer control routines for the MicroP's project

; SUBROUTINE InitClock
; initializes the clock
InitClock:
       ; set the three configuration bytes
;      setf   TRISD
;      movlw  CCFLAGSA
;      bcf           LATE,1
;      bsf           LATE,1        ; read to clock chip

       ; write rate bits
       clrf   TRISD         ; set port D to output
       movlw  CCRATESA
       movwf  LATA          ; set write rates reg
       movlw  CCRATES
       movwf  PORTD
       bcf           LATE,1
       bsf           LATE,1        ; write to clock chip
       ; write interrupt enable flags
       movlw  CCIEFLAGSA
       movwf  LATA          ; set write rates reg
       movlw  CCIEFLAGS
       movwf  PORTD
       bcf           LATE,1
       bsf           LATE,1        ; write to clock chip
       ; write control flags
       movlw  CCCONTROLA
       movwf  LATA          ; set write rates reg
       movlw  CCCONTROL
       movwf  PORTD
       bcf           LATE,1
       bsf           LATE,1        ; write to clock chip
       ; write the alarm time
       movlw  CCALARM1A
       movwf  LATA
       movlw  CCALARM1
       movwf  PORTD
       bcf           LATE,1
       bsf           LATE,1        ; write to clock chip
       movlw  CCALARM2A
       movwf  LATA
       movlw  CCALARM2
       movwf  PORTD
       bcf           LATE,1
       bsf           LATE,1        ; write to clock chip
       movlw  CCALARM3A
       movwf  LATA
       movlw  CCALARM3
       movwf  PORTD
       bcf           LATE,1
       bsf           LATE,1        ; write to clock chip
       movlw  CCALARM4A
       movwf  LATA
       movlw  CCALARM4
       movwf  PORTD
       bcf           LATE,1
       bsf           LATE,1        ; write to clock chip

       call   ClearAlarm    ; make sure the alarm is cleared
       return

; SUBROUTINE DisplayTime
; this subroutine queries the clock chip for
; the time and parses the result, then writes
; the parsed result to the display
DisplayTime:
       ; setup indirection
       movlw  0x01
```

```
        movwf   FSR0H
        movf    WRTDISPREG,0
        movwf   FSR0L

        setf    TRISD         ; set port D to input
        movlw   GETIMEA3
        movwf   LATA          ; set timer chip to hours
        bcf            LATE,1         ; ~~enable clock chip (~CE = 0)

;query clock for time
;parse time
;write time to display
        ; HOURS
        movf    PORTD,0       ; get hours data
        swapf   WREG          ; swap nibbles
        andlw   TENMASK       ; isolate tens digit
        addlw   ZEROASCII     ; convert to ASCII character
        movwf   POSTINC0      ; write ten hours
        movf    PORTD,0       ; get hours data again
        andlw   ONEMASK       ; isolate ones digit
        addlw   ZEROASCII     ; convert to ASCII character
        movwf   POSTINC0      ; write one hours
        movlw   HOURCHAR
        movwf   POSTINC0      ; write hour/minute char

        ; MINUTES
        movlw   GETIMEA2
        movwf   LATA          ; set timer chip to minutes
        movf    PORTD,0       ; get minutes data
        swapf   WREG          ; swap nibbles
        andlw   TENMASK       ; isolate tens digit
        addlw   ZEROASCII     ; convert to ASCII character
        movwf   POSTINC0      ; write ten minutes
        movf    PORTD,0       ; get minutes data again
        andlw   ONEMASK       ; isolate ones digit
        addlw   ZEROASCII     ; convert to ASCII character
        movwf   POSTINC0      ; write one minutes
        movlw   MINCHAR
        movwf   POSTINC0      ; write minute/seconds char

        ; SECONDS
        movlw   GETIMEA1
        movwf   LATA          ; set timer chip to seconds
        movf    PORTD,0       ; get seconds data
        swapf   WREG          ; swap nibbles
        andlw   TENMASK       ; isolate tens digit
        addlw   ZEROASCII     ; convert to ASCII character
        movwf   POSTINC0      ; write ten seconds
        movf    PORTD,0       ; get seconds data again
        andlw   ONEMASK       ; isolate ones digit
        addlw   ZEROASCII     ; convert to ASCII character
        movwf   POSTINC0      ; write one seconds

        bsf            LATE,1         ; ~enable clock chip (~CE = 1)
        clrf    TRISD         ; set port D to output

        return

; SUBROUTINE SetTime
SetTime:
; ok, or together hours, minutes, seconds
        ; bleh
        lfsr    FSR0, SETTINREG       ; set FSR0 to the start of the time data
        clrf    TRISD         ; set port D to output
        movlw   CCCONTROLA
        movwf   LATA          ; set write uti register
        movlw   CCCONTROLS
        movwf   PORTD
        bcf            LATE,1
        bsf            LATE,1         ; write to clock chip uti=1
        movlw   SETIMEA3
        movwf   LATA            ; set write hours
        movf    POSTINC0,0
        swapf   WREG                    ; swap nibbles
        iorwf   POSTINC0,0              ; or with hours
        movwf   PORTD
        bcf             LATE,1
        bsf             LATE,1          ; write to clock chip hours

        movlw   SETIMEA2
        movwf   LATA            ; set write minutes
        movf    POSTINC0,0
        swapf   WREG                    ; swap nibbles
        iorwf   POSTINC0,0              ; or with minutes
        movwf   PORTD
        bcf             LATE,1
        bsf             LATE,1          ; write to clock chip minutes

        movlw   SETIMEA1
        movwf   LATA            ; set write seconds
        movf    POSTINC0,0
        swapf   WREG                    ; swap nibbles
        iorwf   POSTINC0,0              ; or with seconds
        movwf   PORTD
        bcf             LATE,1
        bsf             LATE,1          ; write to clock chip seconds

        movlw   CCCONTROLA
        movwf   LATA            ; set write uti register
        movlw   CCCONTROL
        movwf   PORTD
        bcf             LATE,1
        bsf             LATE,1          ; write to clock chip uti=0
        return

; SUBROUTINE ClearAlarm
; clears the alarm bit on the clock chip
ClearAlarm:
        setf    TRISD           ; set port D to input
        movlw   GETALARMA
        movwf   LATA            ; set timer chip to hours
        bcf             LATE,1          ; ~~enable clock chip (~CE = 0)
        bsf             LATE,1          ; ~enable clock chip (~CE = 1)
        clrf    TRISD           ; set port D to output
        return



; allocate variables
outermax300     EQU     0x4B            ; 300ms delay
innermaxlit EQU 0xF8                    ; number of times to iterate inner loop is 248
outermaxlit EQU 0xFA                    ; number of times to iterate outer loop is 250
innermaxreg EQU 0x11                    ; reserve file register for innermax
outermaxreg     EQU 0x12                        ; reserve file register for outermax
counter EQU 0x10                                        ; reserve address 0x10 for the counter
zero EQU 0x00                                   ; define a zero constant
timetodelay     EQU     0x13                    ; store the number of seconds to delay here

Delay300ms:
        movlw innermaxlit
        movwf innermaxreg                       ; move the literal innermax into the file
        movlw outermax300
        movwf outermaxreg                       ; repeat for the next entry in the table
        bra     timer

; SUBROUTINE DelaySecond
DelaySecond:
; initialization
        movlw innermaxlit
        movwf innermaxreg                       ; move the literal innermax into the file
        movlw outermaxlit
        movwf outermaxreg                    ; repeat for the next entry in the table

; delay 1 second
timer
        movlw 0                                      ; set the wreg to 0
        movwf counter                        ; put 0 into count
loopouter                                            ; outer loop
        movlw 0                              ; put 0 in the wreg
loopinner                                            ; inner loop
        addlw 1                       ; increment the wreg
        nop
        nop
        nop
        nop
        cpfseq innermaxreg    ; if the loop has been iterated innermax times, exit
        bra loopinner         ; if not, repeat the inner loop
        movf counter, 0                      ; move the count value into the wreg for easy use
        addlw 1                              ; increment the wreg
        movwf counter                        ; put the incremented value back into the count
        cpfseq outermaxreg            ; if the outer loop has been iterated outermax
                                                     ; times, exit loop
        bra loopouter                 ; if not, repeat the outer loop
        return                                       ; return after 1 second delay
; implement code here
; delay one second
        return
```

```
; elconstants.inc
; written 11/24/2003 by Damian_Small@hmc.edu
; constants for the electronic code MicroP's project

; INTERRUPTS
SIXAMB         EQU 0x12
SIXAMR         EQU     0x80   ; six am display

; "STUFF"
FATALB         EQU     0x12
FATALR         EQU     0x30   ; fatal error display

; VALVE
; allocate variables, constants
; valve is e:2, 0 = off, 1 = on
VALVEIND       EQU 0x7F       ; register that holds valve state:
VALVEOFF       EQU     0x00   ; value for valve off  (clrf used in code)
VALVESINGLE    EQU     0xF0   ; value for valve on (single)
VALVECONT      EQU 0xFF       ; value for valve on (continuous)

; single use constants
SCDISPB        EQU     0x14   ; bank of single display
SCDISPR        EQU     0xB0   ; register of single display - default B0
SCDISPPBB      EQU     0x01   ; bank of progress bar data
                                             ; (same as bank of dispdatad)
SCDISPPBS      EQU     0x41   ; single progress bar start register
SCDISPPBE      EQU     0x4A   ; single progress bar end register
                                             ; NOTE: total length +1 is also seconds
                                             ; REMEMBER: lines are interlaced!!!
SCDISPPBC      EQU     0xFF   ; single progress bar character

; continuous code constants
CCONDISPB      EQU     0x14   ; bank of continuous on display
CCONDISPR      EQU 0x10       ; register of continous on display
CCOFFDISPB     EQU     0x14   ; bank of continuous off display
CCOFFDISPR     EQU 0x60       ; register of continous off display

; master menu constants
MCMAINDB       EQU     0x10   ; bank of master main menu display
MCMAINDR       EQU     0x50   ; register of master main menu display
MCSMCDB        EQU     0x10   ; bank of master set master code display
MCSMCDR        EQU     0xA0   ; register of master set master code display
MCSRCCDB       EQU     0x15   ; bank of master set/reset continuous code display
MCSRCCDR       EQU     0x00   ; register of master set/reset continuous code display
MCSCCDB        EQU     0x10   ; bank of master set continuous code display
MCSCCDR        EQU     0xF0   ; register of master set continuous code display
MCSRSCDB       EQU     0x15   ; bank of master set/reset single code display
MCSRSCDR       EQU     0x50   ; register of master set/reset single code display
MCSSCDB        EQU     0x11   ; bank of master set single code display
MCSSCDR        EQU     0x40   ; register of master set single code display
MCSTDB         EQU     0x11   ; bank of master set time display
MCSTDR         EQU     0x90   ; register of master set time display

MCSMCK         EQU     0x01   ; key for set master code
MCSCCK         EQU     0x02   ; key for set continuous code
MCSSCK         EQU     0x03   ; key for set single code
MCSTK          EQU     0x04   ; key for set time
MCEXIT         EQU     0x00   ; key for exit master menu

; LCD
; allocate variables, constants
DISPDATAU      EQU     0x00   ; start of display screen data: upper byte
DISPDATAH      EQU     0x10   ; start of display screen data: high byte
DISPDATAL      EQU     0x00   ; start of display screen data: low byte
DISPDATAD      EQU     0x100  ; destination in data memory  (12 bytes)
; note: the end of data is denoted by a 0x00 byte, use 0x20 for space

LCDINIT1       EQU     0x38   ; First LCD initialization data 'N,F'
LCDINIT2       EQU     0x0C   ; LCD initialization data 'Display on'
LCDINIT3       EQU     0x01   ; LCD initialization data 'Clear Display'
LCDINIT4       EQU     0x06   ; LCD initialization data 'I/D, S'
```

```
LCDRETURN      EQU    0x02    ; command to return cursor to home position
LCDCHECKBFA    EQU    0x10    ; check BF port A data
LCDDATAWRA     EQU    0x20    ; write data port A data

WRLCDTEMP      EQU 0x100      ; location of temp data to write
WRLCDBREG      EQU    0x40    ; where the bank address is stored for the
                                             ; write display subroutine
WRLCDDREG      EQU    0x41    ; the start of the diplay data in the specified
                                             ; bank

; Clock Chip
ZEROASCII      EQU    0x30    ; ASCII for 0 (offset for characters)
TENMASK        EQU    0x07    ; mask for 10's digit numerals
ONEMASK        EQU    0x0F    ; mask for 1's digit numerals
HOURCHAR       EQU    0x3A    ; character between hours and minutes
MINCHAR        EQU    0x2E    ; character between minutes and seconds

CCFLAGSA       EQU    0x1D    ; reads AF, PF, PWRF, BVF flags
CCRATESA       EQU 0x2B
CCRATES        EQU 0x0E       ; sets the WD[0:2] and RS[0:3] bits on the clock chip
CCIEFLAGSA     EQU    0x2C
CCIEFLAGS      EQU    0x08    ; sets the interrupt enable flags
CCCONTROLA     EQU    0x2E
CCCONTROL      EQU    0x06    ; sets UTI, ~STOP, 24/12, DSE control flags

CCALARM1A      EQU    0x21
CCALARM1       EQU    0x00
CCALARM2A      EQU    0x23
CCALARM2       EQU    0x00    ; minutes
CCALARM3A      EQU    0x25
CCALARM3       EQU    0x06    ; hours
CCALARM4A      EQU    0x27
CCALARM4       EQU    0xC0    ; alarm configuration

GETALARMA      EQU    0x1D    ; get alarm byte (just read it)

TIMEONV        EQU    0x2E    ; turn the timer display on (write to display memory)
TIMEOFFV       EQU    0x50    ; turn the timer display off (write to non visible)
WRTDISPREG     EQU    0x42    ; the register with the start location for the
                                             ; time string to be written. Assumed 01 bank
GETIMEA1       EQU    0x10    ; Port A output to get seconds
GETIMEA2       EQU    0x12    ; Port A output to get minutes
GETIMEA3       EQU    0x14    ; Port A output to get hours

; set time constants
SETTIMEC       EQU    0x5F    ; character for set time '_'
SETTDISPREG    EQU    0x11A   ; start of time display on screen
SETTINREG      EQU    0x070   ; start of time store in memory

CCCONTROLS     EQU    0x0E    ; sets UTI

SETIMEA1       EQU    0x20    ; Port A output to get seconds
SETIMEA2       EQU    0x22    ; Port A output to get minutes
SETIMEA3       EQU    0x24    ; Port A output to get hours

; code input/ change constants
MAININPUTB     EQU    0x10
MAININPUTR     EQU    0x00    ; main input display

ERRINPUTB      EQU    0x12
ERRINPUTR      EQU    0xD0    ; error: wrong code
ERRLENB        EQU    0x13
ERRLENR        EQU    0xC0    ; error: wrong length (change code)
ERRMATCHB      EQU    0x13
ERRMATCHR      EQU    0x70    ; error: matches other code (change code)
MISMATCHB      EQU    0x11
MISMATCHR      EQU    0xE0    ; error: when setting code do not match

ENTERAGAINB    EQU    0x13
ENTERAGAINR    EQU    0x20    ; enter code again (change code)
```

```
;CODECHAR      EQU    0x78    ; Ascii char for code character 'x'
CODECHAR       EQU    0x2A    ; Ascii char for code character '*'
CODESTART      EQU    0x11F   ; start of code enter field, must be 1 bank

TEMPREGB       EQU 0x45
TEMPREGR       EQU    0x46    ; temporary storage for setting the code display

; KimsConstants.inc
; written 12/02/03 by kim_shultz@hmc.edu
; include constants for KegLock project

; allocate variables
count equ 0x00
match equ 0x0A        ; use as flags to set which codes have been matched
codeSet equ 0x0D      ; use as flag to determine which code to set
lengthCode equ 0x0B
inputDigit equ 0x0C
maxlit EQU 0xFF                      ; number of times to iterate inner loop is 256
maxreg EQU 0x2C                      ; reserve file register for innermax
MSB1low equ 0xEF
MSB2low equ 0xDF
MSB3low equ 0xBF

; for inputDigit:
; stores digits 0-9 in hex
pound equ 0x0F        ; stores # as 0x0F
star equ 0x0E         ; stores * as 0x0E

; reserve space for codes
inputCode1 equ 0x10
inputCode2 equ 0x11
inputCode3 equ 0x12
inputCode4 equ 0x13
inputCode5 equ 0x14
inputCode6 equ 0x15

masterCode1 equ 0x16
masterCode2 equ 0x17
masterCode3 equ 0x18
masterCode4 equ 0x19
masterCode5 equ 0x1A
masterCode6 equ 0x1B

continCode1 equ 0x20
continCode2 equ 0x21
continCode3 equ 0x22
continCode4 equ 0x23
continCode5 equ 0x24
continCode6 equ 0x25

singleCode1 equ 0x26
singleCode2 equ 0x27
singleCode3 equ 0x28
singleCode4 equ 0x29
singleCode5 equ 0x2A          ; these should always be set to A
singleCode6 equ 0x2B          ; these should always be set to A

creatorCode1 equ 0x30
creatorCode2 equ 0x31
creatorCode3 equ 0x32
creatorCode4 equ 0x33
creatorCode5 equ 0x34
creatorCode6 equ 0x35
```

```
; displays.inc
; written 11/24/2003 by Damian_Small@hmc.edu
; display screens for the electronic code MicroP's project
       org     0x1000
;        1   5   10   15   20
;   DB  "12345678901234567890"
; in temp storage: memory locations
; 0x00  DB      "0123456789ABCDEF0123"
; 0x28  DB      "89ABCDEF0123456789AB"
; 0x14  DB      "456789ABCDEF01234567"
; 0x3C  DB      "CDEF0123456789ABCDEF"
; bank 10, 0x00
       DB      " Welcome to KegLock "
       DB      "---===00:00.00===---"
       DB      "Enter Code:         "
       DB      " *~CLEAR    #~ENTER "
; bank 10, 0x50
       DB      "    Master Menu:    "
       DB      "1~Master    3~Single"
       DB      "2~Continuous  4~Time"
       DB      "       0~Exit       "
; bank 10, 0xA0
       DB      "Setting Master Code:"
       DB      "    6-digit code    "
       DB      "Enter Code:         "
       DB      " *~CLEAR    #~ENTER "
; bank 10 0xF0
       DB      "Setting Continuous: "
       DB      "    6-digit code    "
       DB      "Enter Code:         "
       DB      " *~CLEAR    #~ENTER "
; bank 11 0x40
       DB      "Setting Single Code:"
       DB      "    4-digit code    "
       DB      "Enter Code:         "
       DB      " *~CLEAR    #~ENTER "
; bank 11 0x90
       DB      "   Setting Time:    "
       DB      "  (24-hour format)  "
       DB      "      00:00.00      "
       DB      " *~CLEAR    #~ENTER "
; bank 11 0xE0
       DB      "                    "
       DB      "    Sorry, Codes    "
       DB      "    do not match    "
       DB      "                    "
; bank 12 0x30
       DB      " ERROR ERROR ERROR  "
       DB      "Unknown fatal error:"
       DB      "    Reset System    "
       DB      " ERROR ERROR ERROR  "
; bank 12 0x80
       DB      " The Time is 6:00 AM"
       DB      " resetting valve... "
       DB      "  resetting code... "
       DB      "    Good Morning!   "
; bank 12 0xD0
       DB      "                    "
       DB      "   Invalid Code     "
       DB      "  Please Re-enter   "
       DB      "                    "
; bank 13 0x20
       DB      "                    "
       DB      "Please re-enter code"
       DB      "                    "
       DB      "                    "
; bank 13 0x70
       DB      "Error: The code you "
       DB      "  entered matches   "
       DB      "   another code.    "
       DB      "  Please try again  "
```

```
; bank 13 0xC0
        DB      "  Wrong number of   "
        DB      "  digits entered.   "
        DB      "                    "
        DB      "  Please try again  "
; bank 14 0x10
        DB      "Continuous Code has "
        DB      "   been entered:    "
        DB      " Valve will be open "
        DB      "until 6am or reentry"
; bank 14 0x60
        DB      "Continuous Code has "
        DB      "  been re-entered:  "
        DB      "  Valve turned off  "
        DB      "                    "
; bank 14 0xB0
        DB      "Single Code entered:"
        DB      "  Begin dispensing  "
        DB      " liquid refreshment!"
        DB      "    [          ]    "
; bank 15 0x00
        DB      "                    "
        DB      "  Continuous Code:  "
        DB      "                    "
        DB      "  *~RESET    #~SET  "
; bank 15 0x50
        DB      "                    "
        DB      "  Single Use Code:  "
        DB      "                    "
        DB      "  *~RESET    #~SET  "
        DB      0x00, 0x00     ; end of data
```