Lab 3

Creating a multitasking system that uses a keypad and user input to control the frequency of two LED blinking lights.

By: Maddy Reedy and Sabrina Luo
ME 305 - 01

To:	William R. Murray, Department of Mechanical Engineering, Cal Poly SLO wrmurray@calpoly.edu	
From:	Maddy Reedy MaReedy@calpoly.edu	Sabrina Luo Sluo10@calpoly.edu
Date:	04/10/25	
RE:	Lab 3 Memo – Cooperative Multi-Tasking LED Blinker with Robust I/O Interface	

Abstract

The overall goal of lab 3 was to create a multitasking system that would use a keypad and user input to control the speed of two LED blinking lights. The user would use the keypad to input a time in milliseconds, for one of the two blinking lights. The input digits would be displayed on an LCD screen, once the user presses enter the lights will begin to blink at the frequency displayed on the LCD screen. Each of our functions is managed as separate tasks that work together by taking turns, rather than using interrupts. This lab taught us how finite state machines and multitasking can be used to create responsive and organized systems.

Tasks

Task 1: Mastermind - Direct and control the other tasks such as determining which keys have been stuck

State 0: All MasterMind initialization of both hardware and software to be carried out or initiated in this state.

State 1: Wait until all initialization and initial display prompt is complete.

State 2: The Hub to check whether a struck key is waiting to be implemented. If no ASCII value is there, MasterMind will return and allow the next task to be executed. If ASCII value is pressed, the MasterMind will determine which key was pressed and set to the next state

State 3: If the ASCII value turns out to be a digit, and appropriate actions for a digit are taken for example, it will load that digit to the appropriate LCD address up to five addresses. It will then return to the hub state.

State 4: If the ASCII value turns out to be a function key, it would take appropriate actions for the function key, such as clearing buffer. It will then return to the hub state.

State 5: If the ASCII value turns out to be a BackSpace, it would take appropriate actions for the Backspace key. The cursor would have to sent back in addition to clearing the value in that address. It will then return to the hub state.

State 6: If the ASCII value turns out to be an Enter key, it would take appropriate actions for the Enter key, for example, converting BUFFER to a readable number and starting the process of the lights and the timings. It will then return to the hub state.

Task 2: Key Driver - To monitor and determine if keys have been struck and to report the ASCII values of the struck keys to Mastermind.

State 0: Initialize keypad drive (both hardware and software) and clear key flag.

State 1: Checks whether a key is pressed. If so, it will get Character, store the value, and raised the flag that a key has been pressed.

State 2: Test key flag and wait for MasterMind to acknowledge whether the current key is received.

Task 3 – Display- To carry out printed messages and data on the LCD module that seen from the MasterMind

State 0: Initializes all Display (both hardware and software) to be carried out

State 1: The hub state checks whether Boolean flags are set which determines if a specific message should be displayed. Display will set to a specific state that corresponds to the display of the selected message.

State 2: Displays the message stored in TIME1 at the upper left corner.

State 3: Displays the message stored in TIME 2 at the lower left corner.

State 4: Displays the message stored in F1PRMPT to column 8 of the first line on the LCD screen.

State 5: Displays the message stored in F2PRMPT to column 8 of the second line on the LCD screen.

Task 4: Pattern 1- Sets the for the first set of LED pattern

State 0: Set the first set of lights as PORTP outputs and ensure that the LEDs and the flag are off

State 1: LEDs are OFF until the ON1 key has been set

State 2: Lights up Green LED but have Red LED off.

State 3: Turns Green LED off.

State 4: Keep Green LED off but turn Red LED on

State 5: Turns Red LED off

State 6: Turns both LEDs on

State 7: Turns both LEDs off

Task 5: Timing 1 - The time it takes for the first set of lights to execute the pattern from Task 4

State 0: Initializes timing and clears the done flag

State 1: Waits to see if the first set of lights are to be on and the preparations for a countdown

State 2: Start and execute the countdown

Task 6: Pattern 2 - Sets the for the second set of LED pattern

State 0: Set the second set of lights as PORTP outputs and ensure that the LEDs and the flag are off

State 1: LEDs are OFF until the ON2 key has been set

State 2: Lights up Green LED but have Red LED off.

State 3: Turns Green LED off.

State 4: Keep Green LED off but turn Red LED on

State 5: Turns Red LED offt0pp0pji

State 6: Turns both LEDs on

State 7: Turns both LEDs off

Task 7: Timing 2 - The time it takes for the second set of lights to execute the pattern from Task 6

State 0: Initializes timing and clears the done flag

State 1: Waits to see if the first set of lights are to be on and the preparations for a countdown

State 2: Start and execute the countdown

Task 8: Count – sets a delay

State 0: Initializes Task 8

State 1: Goes to the delay subroutine which delays for 1ms.

Inter-Task Communication Variables

Variable	Meaning	Tasks Used	Tasks Cleared

TICKS_1	Period for first LED pair in milliseconds	Task 5: State 1	Task 1: State 0 Task 1: State 2
COUNT_1	Time remaining in period for first LED pair in milliseconds	Task 5: State 1 Task 5: State 2	
DONE_1	Boolean indicating change time for first LED pair	Task 5: State 2	Task 5: State 0 Task 5: State 1
ON1	Boolean indicating first LED pair operation state	Task 1: State 6	Task 1: State 2 Task 4: State 0
TICKS_2	Period for second LED pair in milliseconds	Task 1: State 6	Task 1: State 0 Task 1: State 2 Task 6: State 0
COUNT_2	Time remaining in period for second LED pair in milliseconds	Task 7: State 1 Task 7: State 2	
DONE_2	Boolean indicating change time for second LED pair	Task 7: State 2	Task 7: State 0 Task 7: State 1
ON2	Boolean indicating second LED pair operation state	Task 1: State 6	Task 1: State 0 Task 1: State 2
COUNT	Number of digits currently in BUFFER	Task 1: State 3 Task 1: State 5	Task 1: State 0
BUFFER	Buffer containing digits from keypad	Task 1: State 3 Task 1: State 6	Task 1: State 4
TEMP	Used in ASCII_2_Bin conversion routine	Task 1: State 6	Task 1: State 0 Task 1: State 2
Char_addr	Used to clear specific LCD addresses locations	Task 1: State 2	Task 1: State 2
error_ON	Flag to indicate there is an error	Task 1: State 3	Task 1: State 0
error1_ON	Flag to indicate error 1 has been triggered	Task 1: State 3 Task 1: State 6	Task 1: State 0 Task 1: State 2
error2_ON	Flag to indicate error 2 has been triggered	Task 1: State 6	Task 1: State 0 Task 1: State 2
errordelay	Period that message will be delayed	Task 1: State 3	Task 1: State 0 Task 1: State 2
delaycount	Amount of time error message delay will run	Task 1: State 3 Task 1: State 6	Task 1: State 2
F1FLAG	Flag to indicate F1 was pressed	Task 1: State 3 Task 1: State 6	Task 1: State 2
F2FLAG	Flag to indicate F2 was pressed	Task 1: State 2	Task 1: State 0 Task 1: State 2 Task 1: State 6
KEY_BUF	Buffer for most recent character	Task 1: State 2	Task 1: State 0 Task 1: State 2 Task 1: State 6
KEY_FLG	Boolean for key is available for MM	Task 2: State 1	Task 1: State 2 Task 2: State 0

DTIME1	Boolean to display "TIME1 = "	Task 1: State 0 Task 3: State 1	Task 3: State 2
DTIME2	Boolean to display "TIME2 = "	Task 1: State 0 Task 3: State 1	Task 3: State 3
DF1PRMPT	Boolean to display "<F1> to update LED1 period"	Task 1: State 0 Task 3: State 1	Task 3: State 4
DF2PRMPT	Boolean to display "<F2> to update LED2 period"	Task 1: State 0 Task 3: State 1	Task 3: State 5
FIRSTCH	Boolean for first char in DISPLAY subroutine	Task 3: State 0 Task 3: State 2 Task 3: State 3 Task 3: State 4 Task 3: State 5	Task 3: State 2 Task 3: State 3 Task 3: State 4 Task 3: State 5
DPTR	Address if next charter in current message	Task 3: State 2 Task 3: State 3 Task 3: State 4 Task 3: State 5	
t1state	State Variable for Task 1		
t2state	State Variable for Task 2	Task 1: State 0 Task 1: State 1 Task 1: State 2 Task 1: State 3 Task 1: State 4 Task 1: State 5 Task 1: State 6	Task 1: Mastermind
t3state	State Variable for Task 3	Task 2: State 0 Task 2: State 1 Task 2: State 2	Task 2: Mastermind
t4state	State Variable for Task 4	Task 3: State 0 Task 3: State 1 Task 3: State 2 Task 3: State 3 Task 3: State 4 Task 3: State 5	Task 3: Mastermind
t5state	State Variable for Task 5	Task 4: State 0 Task 4: State 1 Task 4: State 2 Task 4: State 3 Task 4: State 4 Task 4: State 5 Task 4: State 6 Task 4: State 7	Task 4: Mastermind
t6state	State Variable for Task 6	Task 5: State 0 Task 5: State 1 Task 5: State 2	Task 5: Mastermind
t7state	State Variable for Task 7	Task 6: State 0 Task 6: State 1	Task 7: Mastermind

		Task 6: State 2 Task 6: State 3 Task 6: State 4 Task 6: State 5 Task 6: State 6 Task 6: State 7	
t8state	State Variable for Task 8	Task 7: State 0 Task 7: State 1 Task 7: State 2	Task 7: Mastermind

Table 1. List of Inter-Task Communication Variables Meaning and There Set and Reset Locations

Finite State Machines

Task 1

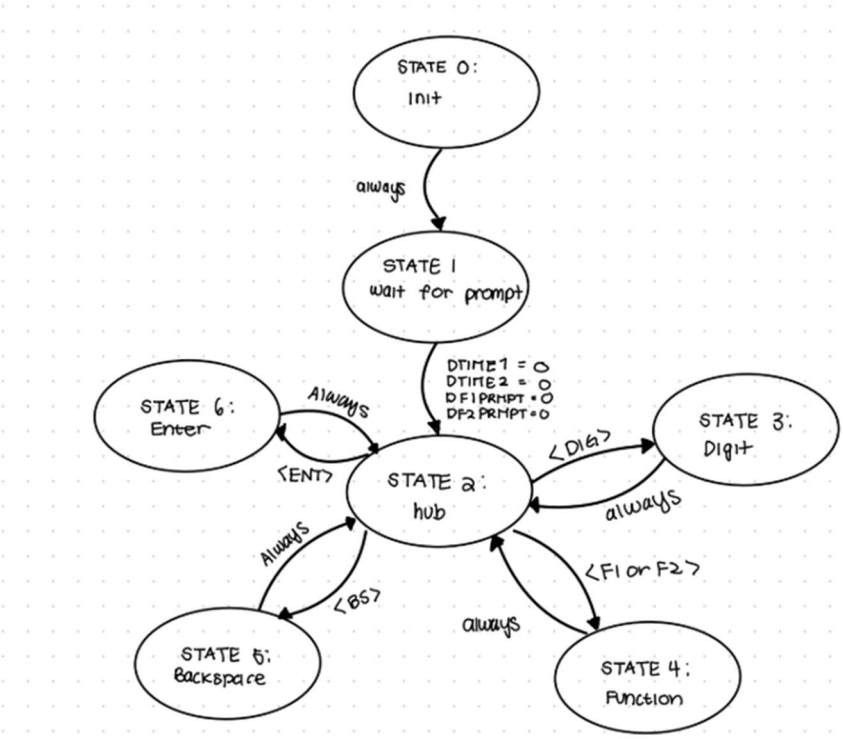

Figure 2. Finite State Machine for Task 1

Task 2

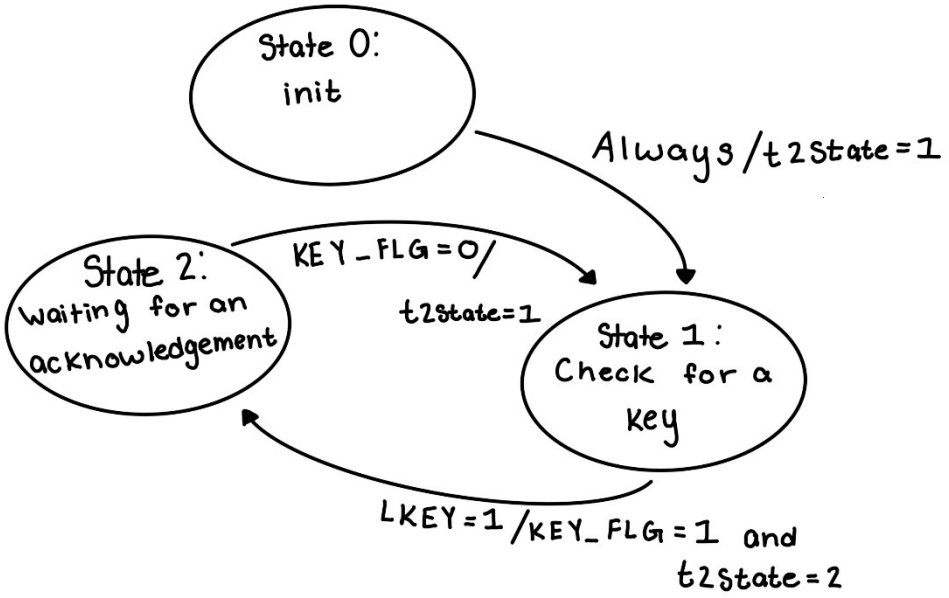

Figure 2. Finite State Machine for Task 2

Task 3

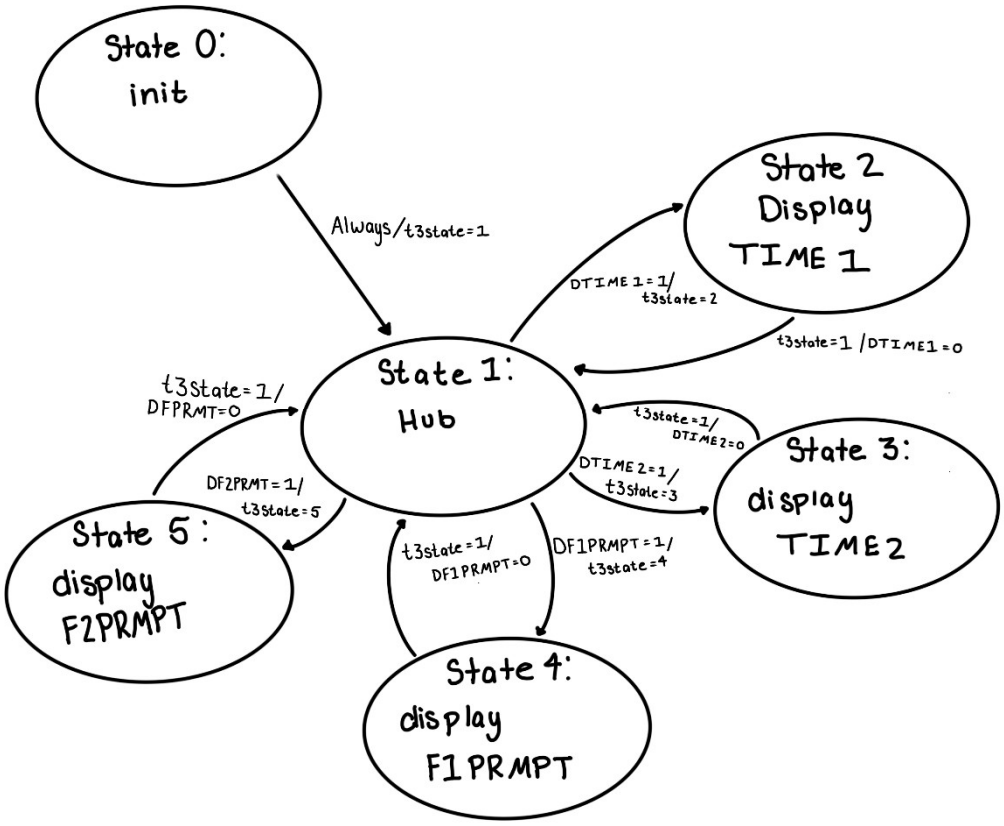

Figure 3. Finite State Machine for Task 3

Task 4

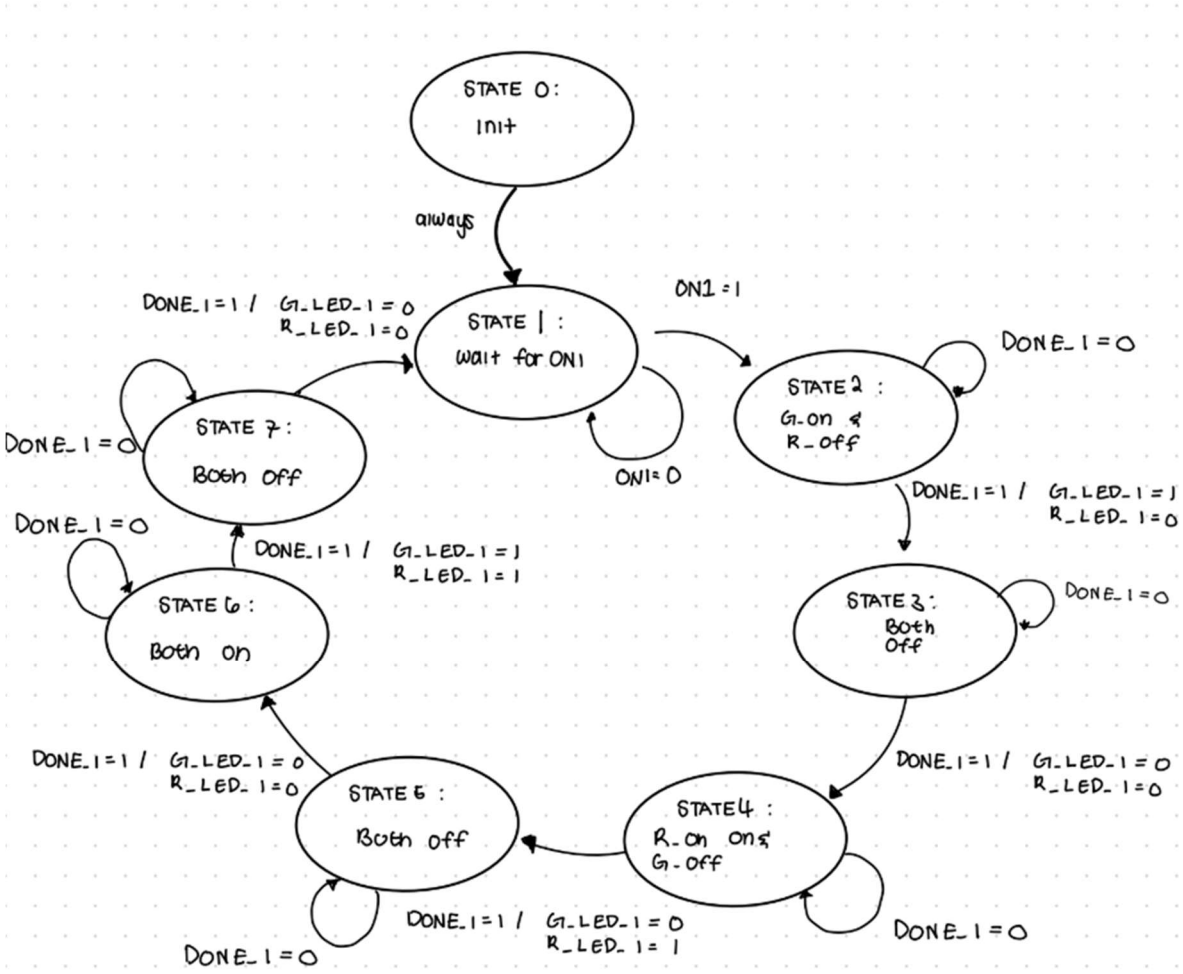

Figure 4. Finite State Machine for Task 4

Task 5

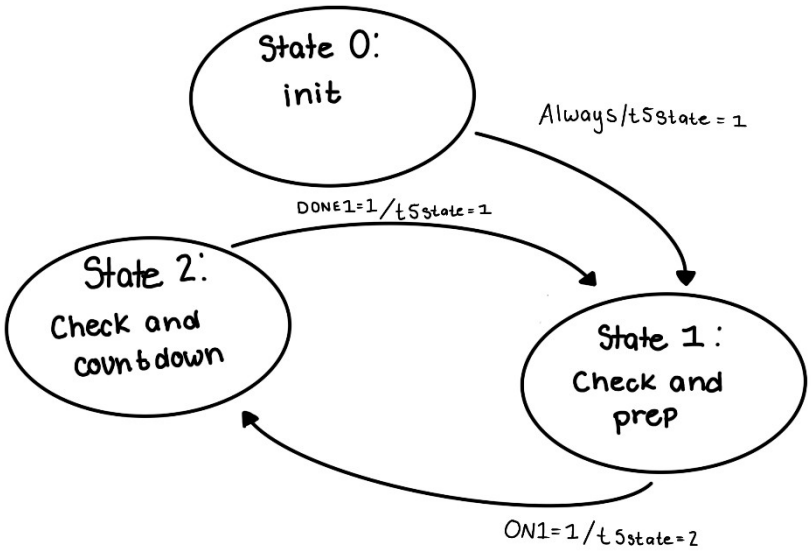

Figure 5. Finite State Machine for Task 5

Task 6

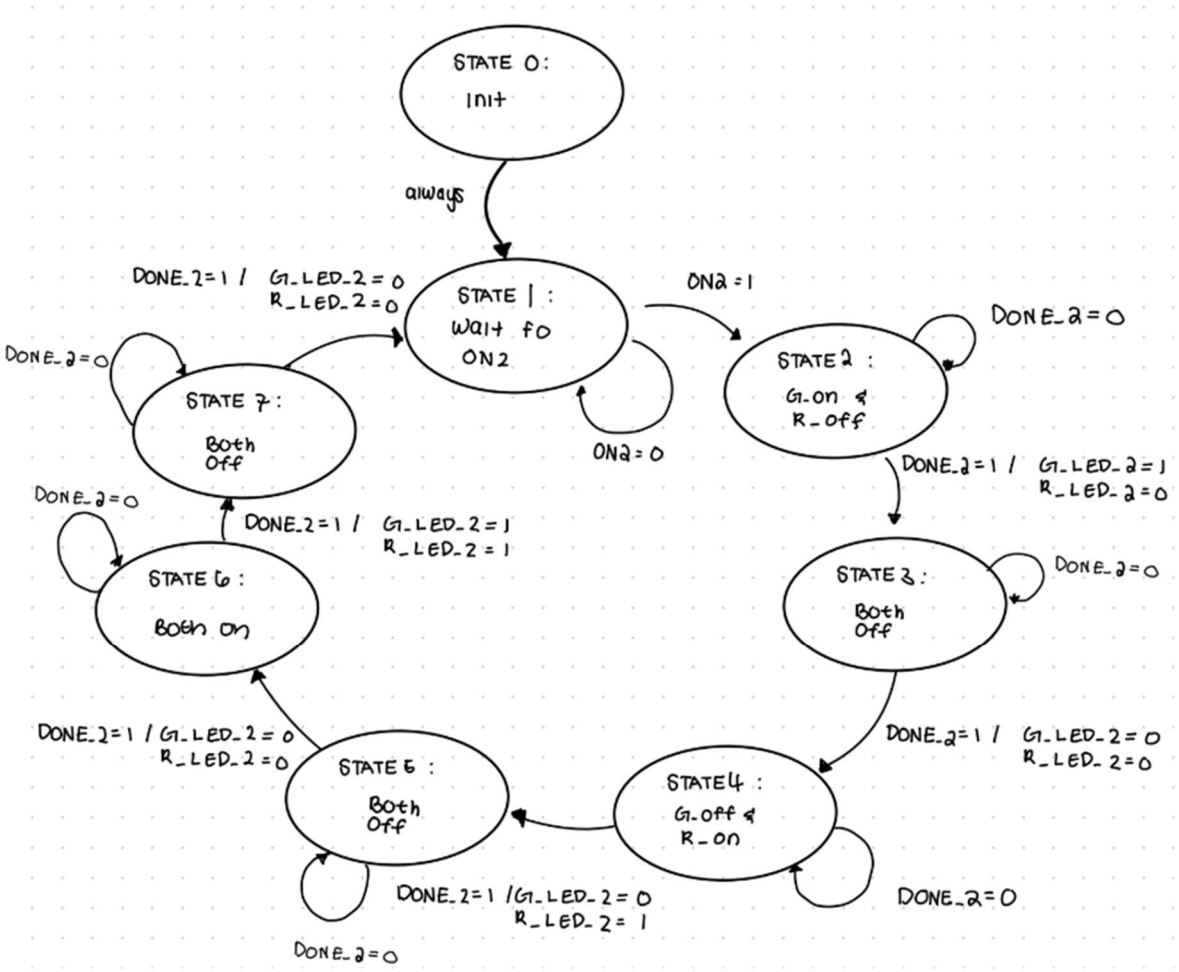

Figure 6. Finite State Machine for Task 6

Task 7

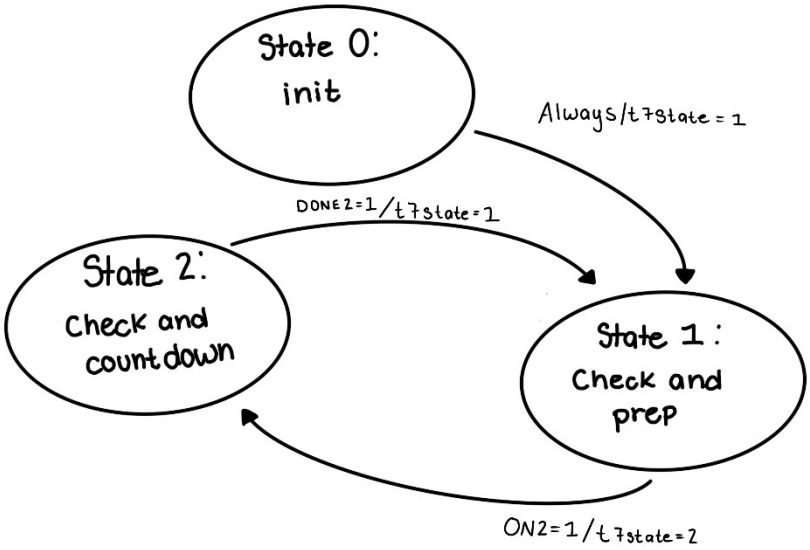

Figure 7. Finite State Machine for Task 7

Task 8

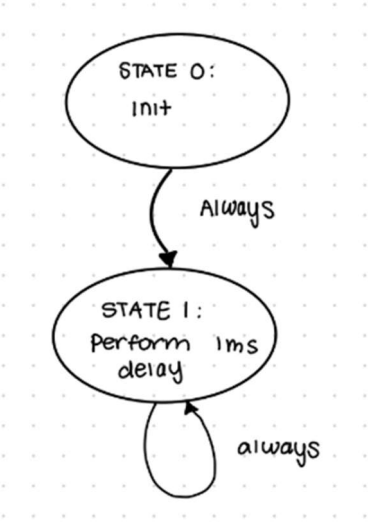

Figure 8. Finite State Machine for Task 8

Source Code

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

```
;* Lab 3 main [shell code from standard version]
*
;*****************************************************************************
*
;*
*
;* Author: Maddy Reedy & Sabrina Luo
*
;*  Cal Poly University
*
;*  May 2025                                                                 *
;*
*
;*****************************************************************************
*


;/---------------------------------------------------------------------------
\
;| Include all associated files
|
;\---------------------------------------------------------------------------
/
; The following are external files to be included during assembly


;/---------------------------------------------------------------------------
\
;| External Definitions
|
;\---------------------------------------------------------------------------
/
; All labels that are referenced by the linker need an external definition

              XDEF  main


;/---------------------------------------------------------------------------
\
;| External References
|
;\---------------------------------------------------------------------------
/
; All labels from other files must have an external reference

              XREF  ENABLE_MOTOR, STARTUP_MOTOR, UPDATE_MOTOR, DISABLE_MOTOR
              XREF  STARTUP_PWM, STARTUP_ENCODER, READ_ENCODER, DISABLE_ENCODER
              XREF  OUTDACA, OUTDACB
              XREF  INITLCD, CLRSCREEN, SETADDR, GETADDR, CURSOR_ON, DISP_OFF
              XREF  OUTCHAR, OUTCHAR_AT, OUTSTRING, OUTSTRING_AT
              XREF  INITKEY, LKEY_FLG, GETCHAR
              XREF  Entry, ISR_KEYPAD


;/---------------------------------------------------------------------------
\
```

```
;| Assembler Equates
|
;\-------------------------------------------------------------------------------
/
; Constant values can be equated here


PORTP          EQU   $0258             ; output port for LEDs
DDRP           EQU   $025A
G_LED_1        EQU   %00010000         ; green LED output pin for LED pair_1
R_LED_1        EQU   %00100000         ; red LED output pin for LED pair_1
LED_MSK_1      EQU   %00110000         ; LED pair_1
G_LED_2        EQU   %01000000         ; green LED output pin for LED pair_2
R_LED_2        EQU   %10000000         ; red LED output pin for LED pair_2
LED_MSK_2      EQU   %11000000         ; LED pair_2


F1             EQU   $F1
F2             EQU   $F2
BSPACE         EQU   $08
ENT            EQU   $0A
SPACE          EQU   $20
L1_lim         EQU   5
F1_start       EQU   $06
F2_start       EQU   $46
F1_delete      EQU   $08
delete         EQU   $20
F1_end         EQU   $16
mes_delay      EQU   $FF
L1_start       EQU   $07
L2_start       EQU   $45




;/-------------------------------------------------------------------------------
\
;| Variables in RAM
|
;\-------------------------------------------------------------------------------
/
; The following variables are located in unpaged ram


DEFAULT_RAM:  SECTION
; State Variables


TICKS_1        DS.W  1                 ; period for LED pair_1 [ms]
COUNT_1        DS.W  1                 ; time remaining in period for LED pair_1 [ms]
DONE_1         DS.B  1                 ; Boolean indicating change time for LED pair_1
ON1            DS.B  1                 ; Boolean indicating LED pair_1 operation state


TICKS_2        DS.W  1                 ; period for LED pair_2 in [ms]
```

```
COUNT_2       DS.W  1               ; time remaining in period for LED pair_2 [ms]
DONE_2        DS.B  1               ; Boolean indicating change time for LED pair_1
ON2           DS.B  1               ; Boolean indicating LED pair_2 operation state

COUNT         DS.B  1               ; number of digits currently in BUFFER
BUFFER        DS.B  5               ; buffer containing digits from keypad
RESULT        DS.W  1               ; result for ASCII->BCD->binary F
TEMP          DS.B  3               ; used in ASCII_2_Bin conversion routine
Char_addr     DS.W  1

error_ON      DS.B  1
error1_ON     DS.B  1
error2_ON     DS.B  1
errordelay    DS.B  1
delay_count   DS.B  1

F1FLAG        DS.B  1               ; flag to indicate F1 was pressed
F2FLAG        DS.B  1               ; flag to indicate F2 was pressed

KEY_BUF       DS.B  1               ; buffer for most recent character
KEY_FLG       DS.B  1               ; Boolean for key is available for MM

DTIME1        DS.B  1               ; Boolean to disp "TIME1 = "
DTIME2        DS.B  1               ; Boolean to disp "TIME2 = "
DF1PRMPT      DS.B  1               ; Boolean to disp "<F1> to update LED1 period"
DF2PRMPT      DS.B  1               ; Boolean to disp "<F1> to update LED1 period"

FIRSTCH       DS.B  1               ; Boolean for first char in DISPLAY subroutine
DPTR          DS.W  1               ; address of next charter in current message

t1state       DS.B  1               ; state variables for each of the tasks
t2state       DS.B  1
t3state       DS.B  1
t4state       DS.B  1
t5state       DS.B  1
t6state       DS.B  1
t7state       DS.B  1
t8state       DS.B  1



L1_chars      DS.B  1
L2_chars      DS.B  1
temp          DS.B  1
COUNT_char    DS.B  1
```

```
;/-------------------------------------------------------------------------------------
\
;|  Main Program Code
|
;\-------------------------------------------------------------------------------------
/
;       This file contains partial shell code for Lab 3 for ME 305
;
;
MyCode:       SECTION
main:
        clr   t1state                 ; initialize all tasks to state0
        clr   t2state
        clr   t3state
        clr   t4state
        clr   t5state
        clr   t6state
        clr   t7state
        clr   t8state
Loop:   ;bgnd
        jsr   TASK_1
        ;bgnd                  ; execute tasks endlessly
        jsr   TASK_2
        jsr   TASK_3
        ;bgnd
        jsr   TASK_4
        ;bgnd
        jsr   TASK_5
        ;bgnd
        jsr   TASK_6
       ; bgnd
        jsr   TASK_7
        ;bgnd
        jsr   TASK_8

        bra   Loop
;
;
;=============TASK_1:
MasterMind=============================================================
;
TASK_1: ldaa   t1state                ; get current t1state
        lbeq   t1state0
        deca
        lbeq   t1state1
        deca
        lbeq   t1state2
        deca
        lbeq   t1state3
        deca
        lbeq   t1state4
        deca
        lbeq   t1state5
        deca
```

```
        lbeq   t1state6
        rts                             ; undefined state - do nothing

t1state0:                               ; initialization state for TASK_1
        clr   COUNT                     ; MasterMind initialization
        clr   F1FLAG
        clr   F2FLAG
        clr   ON1
        clr   ON2
        clr   error_ON
        clr   error1_ON
        clr   error2_ON
        movw  #$00, TICKS_1
        movw  #$00, TICKS_2
        clr   TEMP
        ldaa  #$01                      ; set flags to display all four prompts
        staa  DTIME1
        staa  DTIME2
        staa  DF1PRMPT
        staa  DF2PRMPT
        movb  #$01, t1state             ; otherwise, set next state
        rts

t1state1:                               ; wait for splash screen to display
        tst   DTIME1                    ; finished with TIME1 prompt?
        bne   t1s1a                     ; exit if not done
        tst   DTIME2                    ; finished with TIME2 prompt?
        bne   t1s1a                     ; exit if not done
        tst   DF1PRMPT                  ; finished with F1 prompt?
        bne   t1s1a                     ; exit if not done
        tst   DF2PRMPT                  ; finished with F2 prompt?
        bne   t1s1a                     ; exit if not done
        movb  #$02, t1state             ; set next state
t1s1a:  rts

t1state2:                               ; decode input state
        tst   error_ON
        lbne   error_delay
        tst   KEY_FLG                   ; check for key stroke
        bne   t1s2a                     ; skip over exit code if char available
        rts
                                      ; return if no char available
t1s2a:
        ldab  KEY_BUF                   ; get character
        tst   F1FLAG                    ; if F1FLAG has been set,
        bne   t1s2d                     ;   skip test on F1 & F2
        tst   F2FLAG                    ; if F2FLAG has been set,
        bne   t1s2d                     ;   skip test on F1 & F2
        cmpb  #F1                       ; otherwise, check for F1 then F2
        bne   t1s2b
        movb  #$01, F1FLAG              ; set F1FLAG
        movb  #$04, t1state
```

```
        bra   t1state2F1setup          ; set next state accordingly
        lbra   exit_t1s2               ; branch to exit code
t1s2b:  cmpb  #F2
        bne   t1s2c
        movb  #$01, F2FLAG            ; set F2FLAG
        movb  #$04, t1state           ; set next state accordingly
        bra   t1state2F2setup
t1s2c:  bra   exit_t1s2               ; neither F1 nor F2 are active, so exit
t1s2d:  cmpb  #BSPACE
        bne   t1s2e
        movb  #$05, t1state           ; set next state accordingly
        bra   exit_t1s2               ; branch to exit code
t1s2e:  cmpb  #ENT
        bne   t1s2f
        movb  #$06, t1state           ; set next state accordingly
        bra   exit_t1s2               ; branch to exit code
t1s2f:
        cmpb  #$39                    ; compare to highest digit
        bhi   exit_t1s2               ; exit if not a valid digit
        cmpb  #$30                    ; compare to lowest digit
        blo   exit_t1s2               ; exit if not a valid digit
        movb  #$03, t1state
        bra   exit_t1s2

t1state2F1setup:
           clr  ON1
           movb  #$05, t4state
           movw  #$00, TICKS_1
           clr   TEMP
           ldaa #$07
           jsr  SETADDR
           movw #clear, Char_addr
           bra  t1state2setup


t1state2F2setup:
           clr  ON2
           movb  #$05, t6state
           movw  #$00, TICKS_2
           clr   TEMP
           ldaa #$47
           jsr  SETADDR
           movw #clear, Char_addr
           bra  t1state2setup

t1state2setup:
           ldx  Char_addr
           ldab 0, X
           beq  t1state2setupcomplete

           jsr  OUTCHAR
           incw Char_addr
           bra  t1state2setup
```

```
t1state2setupcomplete:
        movb  #$04, t1state
        suba  $05
        jsr   SETADDR
        bra   exit_t1s2

exit_t1s2:
        clr   KEY_FLG                  ; done with char, so clear KEY_FLG
        rts


t1state3:
           tst  F1FLAG
           bgt  t1state3F1
           tst  F2FLAG
           bgt  t1state3F2


t1state3F1:
        inc   COUNT                     ; digit state
        ldaa  COUNT
        cmpa #L1_lim
        lbhi t1state3overload

        deca
        ldx  #BUFFER
        stab  A,X

        inca
        ldab #F1_start
        aba
        ldab KEY_BUF
        jsr OUTCHAR_AT

        bra t1state3end

t1state3F2:
        inc   COUNT                     ; digit state
        ldaa COUNT
        cmpa #L1_lim
        lbhi t1state3overload

        deca
        ldx  #BUFFER
        stab  A,X

        inca
        ldab #F2_start
        aba
        ldab KEY_BUF
        jsr OUTCHAR_AT

        bra t1state3end
```

```
t1state3overload:
        dec COUNT
        movb  #$02, t1state            ; set next state
        rts
t1state3end:
        movb  #$02, t1state            ; set next state
        rts


t1state3_error2:
        inc  error2_ON
        tst  F1FLAG
        lbne  errorsetupF1

        lbra  errorsetupF2



t1state4:
        clr BUFFER
        clr BUFFER+1
        clr BUFFER+2
        clr BUFFER+3
        clr BUFFER+4



                                        ; function state
t1state4exit:
        movb  #$02, t1state

        rts


t1state5:
          tst COUNT
          ble t1s5c
          tst F1FLAG
          beq t1s5bF2                   ; <B_SPACE> state
t1s5b:
        ldaa COUNT
        ldab #F1_start
        aba
        ldab  #delete
        jsr   OUTCHAR_AT
        jsr   SETADDR
        dec   COUNT
        bra   t1s5c

t1s5bF2:ldaa COUNT
        ldab #F2_start
        aba
        ldab  #delete
        jsr   OUTCHAR_AT
        jsr   SETADDR
```

```
        dec   COUNT
        bra   t1s5c

 t1s5c:
        movb  #$02, t1state           ; set next state
        rts

t1state6:
        tst COUNT
        ble t1state6_error1
        ldaa #$35
        jsr  SETADDR
        tst F1FLAG
        lbne Conversion
        tst F2FLAG
        lbne Conversion2
                              ; If true, Turn on lights 2


t1state6c:
        clr COUNT
        clr F1FLAG
        clr F2FLAG
                            ; <ENT> state
        movb  #$02, t1state           ; set next state
        rts



t1state6_error1:
        inc  error1_ON
        tst  F1FLAG
        lbne  errorsetupF1

        lbra  errorsetupF2
;
;
;
;=============TASK_2:
Keypad_Driver=======================================================
;
TASK_2: ldaa  t2state                 ; get current t8state
        beq   t2state0
        deca
        beq   t2state1
        deca
        beq   t2state2
        rts                           ; undefined state - do nothing

t2state0:                             ; initialization state for TASK_2
        jsr   INITKEY                 ; initialize keypad
        clr   KEY_FLG                 ; clear key available flag
        movb  #$01, t2state           ; set next state
        rts
```

```
t2state1:                               ; checking state
        tst   LKEY_FLG                  ; check for a key stroke
        beq   t2s1a
        jsr   GETCHAR                   ; get char if available
        stab  KEY_BUF                   ; store key value
        movb  #$01, KEY_FLG             ; set key available flag
        movb  #$02, t2state             ; set next state
t2s1a:  rts


t2state2:                               ; waiting state
        tst   KEY_FLG                   ; check KEY_FLG handshake
        bne   t2s2a
        movb  #$01, t2state             ; set next state
t2s2a:  rts                             ; end TASK_2
;
;
;
;=============TASK_3:
Display=============================================================
;
TASK_3: ldaa   t3state                  ; get current t8state
        lbeq   t3state0
        deca
        lbeq   t3state1
        deca
        lbeq   t3state2
        deca
        lbeq   t3state3
        deca
        lbeq   t3state4
        deca
        lbeq   t3state5
        rts                             ; undefined state - do nothing


t3state0:                               ; initialization state for TASK_3
        jsr   INITLCD                   ; initialize the LCD module
        jsr   CURSOR_ON                 ; turn CURSOR ON
        movb  #$01, FIRSTCH             ; set first char flag
        movb  #$01, t3state             ; set next state
        rts


t3state1:                               ; display task hub state
        ldaa  DTIME1
        cmpa  #$01
        bne   t3s1a
        movb  #$02, t3state             ; set next state
        rts
t3s1a:  ldaa  DTIME2
        cmpa  #$01
        bne   t3s1b
        movb  #$03, t3state             ; set next state
```

```
        rts
t3s1b:  ldaa  DF1PRMPT
        cmpa  #$01
        bne   t3s1c
        movb  #$04, t3state           ; set next state
        rts
t3s1c:  ldaa  DF2PRMPT
        cmpa  #$01
        bne   t3s1k
        movb  #$05, t3state           ; set next state
t3s1k:  rts                           ; undefined state - do nothing

t3state2:                             ; display TIME1 state
        ldaa  FIRSTCH
        cmpa  #$01
        bne   t3s2a
        ldx   #TIME1                  ; address of beginning of message
        clra                          ; set LCDaddr to upper left
        jsr   PUTCHAR_1ST             ; display first character
        bra   t3s2done                ; go to t3s2done test
t3s2a:  jsr   PUTCHAR                 ; display next character

t3s2done:
        tst   FIRSTCH                 ; done if FIRSTCH has been reset
        beq   t3s2b                   ; branch if not done
        clr   DTIME1
        movb  #$01, t3state           ; done, so set next state
t3s2b:  rts

t3state3:                             ; display TIME2 state
        ldaa  FIRSTCH
        cmpa  #$01
        bne   t3s3a
        ldx   #TIME2                  ; address of beginning of message
        ldaa  #$40                    ; set LCDaddr to lower left
        jsr   PUTCHAR_1ST             ; display first character
        bra   t3s3done                ; go to t3s3done test

t3s3a:  jsr   PUTCHAR                 ; display next character

t3s3done:
        tst   FIRSTCH                 ; done if FIRSTCH has been reset
        beq   t3s3b                   ; branch if not done
        clr   DTIME2
        movb  #$01, t3state           ; done, so set next state
t3s3b:  rts

t3state4:                             ; display F1PRMPT state
        ldaa  FIRSTCH
        cmpa  #$01
```

```
        bne   t3s4a
        ldx   #F1PRMPT                ; address of beginning of message
        ldaa  #$08                    ; set LCDaddr to proper place in 1st line
        jsr   PUTCHAR_1ST             ; display first character
        bra   t3s4done                ; go to t3s4done test

t3s4a:  jsr   PUTCHAR                 ; display next character

t3s4done:
        tst   FIRSTCH                 ; done if FIRSTCH has been reset
        beq   t3s4b                   ; branch if not done
        clr   DF1PRMPT
        movb  #$01, t3state           ; done, so set next state
t3s4b:  rts

t3state5:                             ; display F2PRMPT state
        ldaa  FIRSTCH
        cmpa  #$01
        bne   t3s5a
        ldx   #F2PRMPT                ; address of beginning of message
        ldaa  #$48                    ; set LCDaddr to proper place in 2nd line
        jsr   PUTCHAR_1ST             ; display first character
        bra   t3s5done                ; go to t3s5done test

t3s5a:  jsr   PUTCHAR                 ; display next character

t3s5done:
        tst   FIRSTCH                 ; done if FIRSTCH has been reset
        beq   t3s5b                   ; branch if not done
        clr   DF2PRMPT
        movb  #$01, t3state           ; done, so set next state
        ldaa #$35
        jsr  SETADDR
t3s5b:  rts
;
;
;
;=============TASK_4:
Pattern_1=============================================================
;
TASK_4:
        ldaa   t4state                ; get current t8state
        lbeq   t4state0
        deca
        lbeq   t4state1
        deca
        lbeq   t4state2
        deca
        lbeq   t4state3
        deca
        lbeq   t4state4
```

```
        deca
        lbeq   t4state5
        deca
        lbeq   t4state6
        deca
        lbeq   t4state7
        rts                             ; undefined state - do nothing



t4state0:                               ; initialization for TASK_1
        clr   ON1                       ; clear LED1 ON flag
        bclr  PORTP, LED_MSK_1          ; ensure that LEDs are off when initialized
        bset  DDRP, LED_MSK_1           ; set LED_MSK_1 pins as PORTP outputs
        movb  #$01, t4state             ; set next state

t4state1:                               ; wait for ON1
        tst   ON1                       ; unless ON1 is set, make sure LEDs are OFF
        beq   t4s1a                     ;  and simply return
        movb  #$02, t4state             ; else set next state
t4s1a:  bclr  PORTP, LED_MSK_1          ; ensure that LEDs are off
        rts

t4state2:                               ; G_on & R_off
        bset  PORTP, G_LED_1            ; set state1 pattern on LEDs
        tst   DONE_1                    ; check LED1 done flag
        beq   exit_t4s2                 ; if not done, return
        movb  #$03, t4state             ; if done, set next state
exit_t4s2:
        rts

t4state3:                               ; both off
        bclr  PORTP, G_LED_1            ; set state1 pattern on LEDs
        tst   DONE_1                    ; check LED1 done flag
        beq   exit_t4s3                 ; if not done, return
        movb  #$04, t4state             ; if done, set next state
exit_t4s3:
        rts

t4state4:                               ; G_off & R_on
        bset  PORTP, R_LED_1            ; set state1 pattern on LEDs
        tst   DONE_1                    ; check LED1 done flag
        beq   exit_t4s4                 ; if not done, return
        movb  #$05, t4state             ; if done, set next state
exit_t4s4:
        rts

t4state5:                               ; both off
        bclr  PORTP, LED_MSK_1          ; set state1 pattern on LEDs
        tst   DONE_1                    ; check LED1 done flag
```

```
        beq   exit_t4s5               ; if not done, return
        movb  #$06, t4state           ; if done, set next state
exit_t4s5:
        rts

t4state6:                             ; G_on & R_on
        bset  PORTP, LED_MSK_1        ; set state1 pattern on LEDs
        tst   DONE_1                  ; check LED1 done flag
        beq   exit_t4s6               ; if not done, return
        movb  #$07, t4state           ; if done, set next state
exit_t4s6:
        rts

t4state7:                             ; both off
        bclr  PORTP, LED_MSK_1        ; set state1 pattern on LEDs
        tst   DONE_1                  ; check LED1 done flag
        beq   exit_t4s7               ; if not done, return
        movb  #$02, t4state           ; if done, set next state
exit_t4s7:
        rts


;
;
;
;=============TASK_5:
Timing_1=============================================================
;
TASK_5: ldaa  t5state                 ; get current t8state
        beq   t5state0
        deca
        beq   t5state1
        deca
        beq   t5state2
        rts                           ; undefined state - do nothing

t5state0:                             ; initialization for TASK_5
        clr   DONE_1                  ; init
        movb  #$01, t5state           ; set next state
        rts
t5state1:                             ; initialization for TASK_5
        clr   DONE_1                  ; to be safe, hold DONE_1 down until ON1=1
        tst   ON1
        beq   t5s1                    ; wait for ON1 to be set
        movw  TICKS_1, COUNT_1        ; when ON1 is set, get ready to count down
        movb  #$02, t5state           ; set next state
t5s1:   rts


t5state2:
        tst   ON1
        beq   t5s2a                   ; do not count down unless ON1 is set
        decw  COUNT_1                 ; decrement COUNT_1
        bne   t5s2b                   ; if not done, return
        movb  #$01, DONE_1            ; if done, set DONE_1 flag
```

```
t5s2a:  movb  #$01, t5state           ; set next state
t5s2b:  rts                           ; end TASK_5


;
;
;
;==============TASK_6:
Pattern_2=============================================================
;
TASK_6: ldaa   t6state                ; get current t8state
        lbeq   t6state0
        deca
        lbeq   t6state1
        deca
        lbeq   t6state2
        deca
        lbeq   t6state3
        deca
        lbeq   t6state4
        deca
        lbeq   t6state5
        deca
        lbeq   t6state6
        deca
        lbeq   t6state7
        rts                           ; undefined state - do nothing



t6state0:                             ; initialization for TASK_1
        clr   ON2                     ; clear LED1 ON flag
        bclr  PORTP, LED_MSK_2        ; ensure that LEDs are off when initialized
        bset  DDRP, LED_MSK_2         ; set LED_MSK_1 pins as PORTP outputs
        movb  #$01, t6state


        rts


t6state1:                             ; wait for ON1
        tst   ON2                     ; unless ON1 is set, make sure LEDs are OFF
        beq   t6s1a                   ;  and simply return
        movb  #$02, t6state           ; else set next state
t6s1a:  bclr  PORTP, LED_MSK_2        ; ensure that LEDs are off
        rts


t6state2:                             ; G_on & R_off
        bset  PORTP, G_LED_2          ; set state1 pattern on LEDs
        tst   DONE_2                  ; check LED1 done flag
        beq   exit_t6s2               ; if not done, return
        movb  #$03, t6state           ; if done, set next state
exit_t6s2:
```

```
        rts

t6state3:                               ; both off
        bclr  PORTP, G_LED_2            ; set state1 pattern on LEDs
        tst   DONE_2                    ; check LED1 done flag
        beq   exit_t6s3                 ; if not done, return
        movb  #$04, t6state             ; if done, set next state
exit_t6s3:
        rts

t6state4:                               ; G_off & R_on
        bset  PORTP, R_LED_2            ; set state1 pattern on LEDs
        tst   DONE_2                    ; check LED1 done flag
        beq   exit_t6s4                 ; if not done, return
        movb  #$05, t6state             ; if done, set next state
exit_t6s4:
        rts

t6state5:                               ; both off
        bclr  PORTP, LED_MSK_2          ; set state1 pattern on LEDs
        tst   DONE_2                    ; check LED1 done flag
        beq   exit_t6s5                 ; if not done, return
        movb  #$06, t6state             ; if done, set next state
exit_t6s5:
        rts

t6state6:                               ; G_on & R_on
        bset  PORTP, LED_MSK_2          ; set state1 pattern on LEDs
        tst   DONE_2                    ; check LED1 done flag
        beq   exit_t6s6                 ; if not done, return
        movb  #$07, t6state             ; if done, set next state
exit_t6s6:
        rts

t6state7:                               ; both off
        bclr  PORTP, LED_MSK_2          ; set state1 pattern on LEDs
        tst   DONE_2                    ; check LED1 done flag
        beq   exit_t6s7                 ; if not done, return
        movb  #$02, t6state             ; if done, set next state
exit_t6s7:
        rts


;
;=============TASK_7:
Timing_2=============================================================
;
  TASK_7:
        ldaa  t7state                   ; get current t8state
        beq   t7state0
        deca
        beq   t7state1
        deca
```

```
        beq   t7state2
        rts                           ; undefined state - do nothing

t7state0:                             ; initialization for TASK_5
        clr   DONE_2                  ; init
        movb  #$01, t7state           ; set next state
        rts
t7state1:                             ; initialization for TASK_5
        clr   DONE_2                  ; to be safe, hold DONE_1 down until ON1=1
        tst   ON2
        beq   t7s1                    ; wait for ON1 to be set
        movw  TICKS_2, COUNT_2        ; when ON1 is set, get ready to count down
        movb  #$02, t7state           ; set next state
t7s1:   rts


t7state2:
        tst   ON2
        beq   t7s2a                   ; do not count down unless ON1 is set
        decw  COUNT_2                 ; decrement COUNT_1
        bne   t7s2b                   ; if not done, return
        movb  #$01, DONE_2            ; if done, set DONE_1 flag
t7s2a:  movb  #$01, t7state           ; set next state
t7s2b:  rts



;


;==============TASK_8:
Countdown=============================================================
;
TASK_8: ldaa  t8state                 ; get current t8state
        beq   t8state0
        deca
        beq   t8state1
        rts                           ; undefined state - do nothing

t8state0:                             ; initialization for TASK_8
        movb  #$01, t8state           ; set next state
        rts


t8state1:
        jsr   DELAY_1ms
        rts                           ; end TASK_8
;
;
;
;/-------------------------------------------------------------------------------------
\
;| Subroutines
|
;\-------------------------------------------------------------------------------------
/
```

```
;=============Subroutine
ASCII_2_Bin=====================================================
;=
=
;=      Assumes standard COUNT & BUFFER scheme
=
;=      Uses the stack to avoid declaring TEMP space in RAM
=
;=      Returns:  result in X with A cleared if no error
=
;=                1 in A if MAGNITUDE TOO LARGE error, and
=
;=                2 in a if ZERO MAGNITUDE INAPPROPRIATE error
=
;=============================================================================================
=
;
ASCII_2_Bin:
        rts                             ; end subroutine ASCII_2_Bin
;
;
;
;=============Subroutine PUTCHAR_1ST and Subroutine
PUTCHAR=============================
;
PUTCHAR_1ST:
        stx   DPTR                      ; first entry point for subroutine
        jsr   SETADDR                   ;  to be used for first character only
        clr   FIRSTCH


PUTCHAR:                                ; second entry point for subroutine
        ldx   DPTR                      ;  to be used for all characters but the first
        ldab  0,X                       ; get character to be displayed
        beq   disp_done                 ; exit if last character in message
        inx                             ; point to next character
        stx   DPTR
        jsr   OUTCHAR                   ; display current character
        rts


disp_done:
        movb  #$01, FIRSTCH             ; reset first character flag
        rts                             ; end subroutine PUTCHAR_1ST and PUTCHAR
;
;
;
;=============Subroutine
Delay_1ms===========================================================
;=
=
;=      This subroutine delays for ~1.00ms
=
```

```
;=
=
;=============================================================================
=
;
DELAY_1ms:
        ldy   #1425                   ; (2)
INNER:                                ; inside loop
        cpy   #0                      ; (2)
        beq   EXIT                    ; (1)
        dey                           ; (1)
        bra   INNER                   ; (3)
EXIT:   rts                           ; (5) end subroutine DELAY_1ms


;=============Subroutine CONVERSION
==================================================
;=
=
;=      This subroutine                                          =
;=
=
;=============================================================================
=
;
Conversion: ;bgnd
            ldy TICKS_1
            ldd #$0A          ; load accumulator D with 'A'
            emul
            tstY
            bne conversionerror2        ; trigger error 1 if not equal to zero
            std TICKS_1


            ldx #BUFFER
            ldaa TEMP         ; load accumulator A with TEMP
            ldab A,X
            subb #$30         ; subtract 30 from accumulator B
            clra              ; clear accumulator A
            addd TICKS_1
            std TICKS_1


            inc TEMP          ; increases TEMP by 1
            dec COUNT         ; decreases COUNT by 1
            bne Conversion    ; if COUNT is less than or equal
                                ; to zero loop back to
                                ; conversion

Test_Result:;bgnd
            ldaa TICKS_1
            aba
            cmpa #$0
            beq conversionerror        ; if RESULT is equal to error2 got
                                ; to error 2
```

```
            ldaa #$00         ; load accumulator A with 0
            movb #$01, ON1
            clr  COUNT_char
            clr L1_start
            clr L1_chars
            clr COUNT
            clr F1FLAG
            clr F2FLAG
                              ; <ENT> state
            movb  #$02, t1state            ; set next state

            rts


 conversionerror:
            inc error1_ON
            bra errorsetupF1
 conversionerror2:
            inc error2_ON
            bra errorsetupF1

;=============Subroutine Error Messages
====================================================
;=
=
;=      This subroutine displays one of two error messages when triggered
=
;=
=
;=============================================================================================
=
;


errorsetupF1:                             ; Set up top line to clear
           ldaa #$07
           jsr  SETADDR
           movw #clear_line, Char_addr
           bra  clearline                 ; Then branch to clear the line

errorsetupF2:                             ; Set up bottom line to clear
           ldaa #$47
           jsr  SETADDR
           movw #clear_line, Char_addr
           bra  clearline                 ; Then branch to clear the line


clearline: ldx  Char_addr                 ; Clear correct line
           ldab 0, X
           beq  errordetermine            ; Then determine which error to display

           jsr  OUTCHAR
           incw Char_addr
           bra  clearline
                                          ; Find correct line and error
```

```
errordetermine:                         ; Turn on Error flag
          inc  error_ON                 ; Set error message delay
          movb #$7F, errordelay         ; Set amount of times error delay will go
          movb #$08, delay_count

          ldaa F1FLAG                   ; Add F1 flag to accumilator A
          ldab error1_ON                ; Add error 1 flag to accumilator B
          aba                           ; Add accumilator A and B
          cmpa #$02                     ; If result is equal to 2 then it must be
error 1
          beq  error1F1                    ; on the top line
                                        ; If result is not equal to 2 then
continue to search
          ldaa F1FLAG
          ldab error2_ON
          aba
          cmpa #$02
          beq  error2F1

          ldaa F2FLAG
          ldab error1_ON
          aba
          cmpa #$02
          beq  error1F2

          ldaa F2FLAG
          ldab error2_ON
          aba
          cmpa #$02
          beq  error2F2

error1F1:                               ; Set up top line for error message 1
          ldaa #$07
          jsr  SETADDR
          movw #error1, Char_addr
          bra  errordisplay             ; Then branch to display error message

error1F2:                               ; Set up bottom line for error message 1
          ldaa #$47
          jsr  SETADDR
          movw #error1, Char_addr
          bra  errordisplay             ; Then branch to display error message



error2F1:                               ; Set up top line for error message 2
          ldaa #$07
          jsr  SETADDR
          movw #error2, Char_addr
          bra  errordisplay             ; Then branch to display error message

error2F2:                               ; Set up bottom line for error message 2
          ldaa #$47
          jsr  SETADDR
          movw #error2, Char_addr
```

```
          bra  errordisplay              ; Then branch to display error message

errordisplay:                           ; Display correct error message on the
correct line
          ldx  Char_addr
          ldab 0, X
          beq  error_delay              ; Then branch to error message delay

          jsr  OUTCHAR
          incw Char_addr
          bra  errordisplay

error_delay:                            ; Delay error message so user can read
          ldaa #$35
          jsr  SETADDR                  ; Turn off cursor
          tst  errordelay               ; See if delay is zero
          ble  error_delaycheck           ; If zero branch to error_delaycheck
          dec  errordelay                 ; Else decrement_error delay

          movb  #$02, t1state           ; Declare Task 1 function state to be 2
          rts                           ; Return to Mastermind

error_delaycheck:                       ; Check how many times delay has been
triggered
          tst  delay_count              ; Check is delay_count is zero
          ble  line_determine             ; If zero branch to line_determine
          dec  delay_count                ; Else decrement delay_count
          movb #$7F, errordelay             ; and reset error_delay to 127 ms

          movb  #$02, t1state           ; Declare Task 1 function state to be 2
          rts                           ; Return to Mastermind


line_determine:                         ; Find which line to reset
          tst  F1FLAG
          bne  errorresetF1

          bra  errorresetF2



errorresetF1:                           ; Set up top line to display propmt 1
          ldaa #$07
          jsr  SETADDR
          movw #F1PRMPTRS, Char_addr
          bra  errorresetdisplay

errorresetF2:                           ; Set up bottom line to display propmt 2
          ldaa #$47
          jsr  SETADDR
          movw #F2PRMPTRS, Char_addr
          bra  errorresetdisplay

errorresetdisplay:                      ; Display propmt on proper line
          ldx  Char_addr
```

```
           ldab 0, X
           beq  errorreset                 ; Once displayed branch to errorreset

           jsr  OUTCHAR
           incw Char_addr
           bra  errorresetdisplay


errorreset:                                 ; Resets flags, buffer and display
           ldaa #$35
           jsr  SETADDR
           clr BUFFER
           clr BUFFER+1
           clr BUFFER+2
           clr BUFFER+3
           clr BUFFER+4
           clr COUNT

           clr F1FLAG
           clr F2FLAG

           clr error1_ON
           clr error2_ON
           clr error_ON

           movb  #$02, t1state             ; Declare Task 1 function state to be 2
           rts                             ; Return to Mastermind



;=============Subroutine CONVERSION 2
=====================================================
;=
=
;=      This subroutine                                                 =
;=
=
;=============================================================================================
=
;
Conversion2: ;bgnd
            ldy TICKS_2
            ldd #$0A          ; load accumulator D with 'A'
            emul
            tstY
            lbne conversion2error2        ; trigger error 1 if not equal to zero
            std TICKS_2


            ldx #BUFFER
            ldaa TEMP         ; load accumulator A with TEMP
            ldab A,X
            subb #$30         ; subtract 30 from accumulator B
            clra              ; clear accumulator A
```

```
            addd TICKS_2
            std TICKS_2

            inc TEMP          ; increases TEMP by 1
            dec COUNT         ; decreases COUNT by 1
            bne Conversion2    ; if COUNT is less than or equal
                               ; to zero loop back to
                               ; conversion


Test_Result2:ldaa TICKS_2
            aba
            cmpa #$0
            lbeq conversion2error        ; if RESULT is equal to error2 got
                                ; to error 2
            ldaa #$00         ; load accumulator A with 0
            movb #$01, ON2
            clr  COUNT_char
            clr L1_start
            clr L1_chars
            clr COUNT
            clr F1FLAG
            clr F2FLAG
            movb  #$02, t1state            ; set next state

            rts

conversion2error:
            inc error1_ON
            lbra errorsetupF2
 conversion2error2:
            inc error2_ON
            lbra errorsetupF2
;
;
;
;=============ASCII
Messages=============================================================
;
TIME1:        DC.B  'TIME1 = ',$00
F1PRMPT:      DC.B  '      <F1> to update LED1 period',$00
TIME2:        DC.B  'TIME2 = ',$00
F2PRMPT:      DC.B  '      <F2> to update LED2 period',$00
clear:        DC.B  '     ', $00
error1:       DC.B  'ZERO MAGNITUDE INAPPROPRIATE', $00
error2:       DC.B  'MAGNITUDE TOO LARGE', $00
clear_line:   DC.B  '                                    ', $00
F1PRMPTRS:    DC.B  '       <F1> to update LED1 period',$00
F2PRMPTRS:    DC.B  '       <F2> to update LED2 period',$00


;/-------------------------------------------------------------------------------
\
```

```
;| Vectors
|
;\-------------------------------------------------------------------------------
/


; Add interrupt and reset vectors here:
         ORG   $FFFE                  ; reset vector address
         DC.W  Entry
         ORG   $FFCE                  ; Key Wakeup interrupt vector address [Port J]
         DC.W  ISR_KEYPAD
```