;Lab 3 code

;L. Carter Price

;Sara Visitacion

;Assembler Equates

PORTS = $00D6 ; output port for LEDs

DDRS = $00D7

LED\_MSK\_1 = 0b00000011 ; LED\_1 output pins

R\_LED\_1 = 0b00000001 ; red LED\_1 output pin

G\_LED\_1 = 0b00000010 ; green LED\_1 output pin

LED\_MSK\_2 = 0b00001100 ; LED\_2 output pins

R\_LED\_2 = 0b00000100 ; red LED\_1 output pin

G\_LED\_2 = 0b00001000 ; green LED\_1 output pin

;===================================================================

;RAM

.area bss

BACKflag:: .blkb 1 ;backspace flag NOT USED YET

DPTR:: .blkb 2 ;address of next character to be read and displayed

FIRSTCH:: .blkb 1 ;first character flag

DLINE1:: .blkb 1 ;display line 1 flag

DLINE2:: .blkb 1 ;display line 2 flag

L1:: .blkb 1 ;initializes original prompt

L2:: .blkb 1 ;initializes original prompt

F1flag:: .blkb 1 ;indicates the line 1 being written to

F2flag:: .blkb 1 ;indicates line 2 being written to

DIGITflag:: .blkb 1 ;indicates that a digit is to be displayed

BUFFER:: .blkb 6 ;stores the entered digit

POINTER:: .blkb 2 ;points to address of the next digit in buffer

COUNT:: .blkb 1 ;number of digits entered

FIRSTdig:: .blkb 1 ;indicates the first digit on the line

TEMP:: .blkb 1 ;the ascii of whatever was entered in Keypad

RESULT:: .blkb 2 ;the BCD of the entered value

TICKS\_1:: .blkb 2 ;the amount of time for blink pair 1

COUNT\_1:: .blkb 2 ;decrements each time through, controls blink time LED1

DONE\_1:: .blkb 1 ;communicates when to switch to next step in task LED1

TICKS\_2:: .blkb 2 ;the amount of time for blink pair 1 COUNT\_2:: .blkb 2 ;decrements each time through, controls blink time LED2

DONE\_2:: .blkb 1 ;communicates when to switch to next step in task LED2

t1state:: .blkb 1 ;the state of task 4

t2state:: .blkb 1 ;the state of task 5

t3state:: .blkb 1 ;the state of task 8

t4state:: .blkb 1 ;the state of task 6

t5state:: .blkb 1 ;the state of task 7

ON\_1:: .blkb 1 ;indicates if LED pair 1 should be on

ON\_2:: .blkb 1 ;indicates if LED pair 2 should be on

TOOHIGHflag:: .blkb 1 ;indicates if enter value is too high

SHOWflag:: .blkb 1 ;indicates if screen should pause on an error

SHOWCOUNT:: .blkb 2 ;counts the time to show error

NODIGflag:: .blkb 1 ;indicates no digits entered

ZEROflag:: .blkb 1 ;indicates a zero was entered

CLINE1flag:: .blkb 1 ;flags to clear line 1

CLINE2flag:: .blkb 1 ;flags to clear line 2

COUNTBUFF:: .blkb 1 ;counter in clear buffer routine

FIRSTRUN1:: .blkb 1 ;flag to initialize the keypad

FIRSTRUN2:: .blkb 1 ;flag to initialize the display

;==================================================================

.area text

\_main:: movb #$01, L1

movb #$01, L2

movb #$01, FIRSTCH

movw #BUFFER,POINTER ;moves the contents of BUFFER into POINTER

movw #$03E8,SHOWCOUNT

movb #$06, COUNTBUFF

movb #$01, FIRSTRUN1

movb #$01, FIRSTRUN2

clr t1state ;initialize all tasks to state0

clr t2state

clr t3state

clr t4state

clr t5state

clr ON\_1

clr ON\_2

TOP:

jsr TASK\_1 ;mastermind

jsr TASK\_2 ;Keypad Driver

jsr TASK\_3 ;Display Driver

jsr TASK\_4 ;pattern\_1

jsr TASK\_5 ;count LED pair 1

jsr TASK\_6 ;pattern\_2

jsr TASK\_7 ;count LED pair 2

jsr TASK\_8 ;delay

bra TOP

;==========================================================================

;MASTERMIND

TASK\_1: tst L1 ;test L1

bne Prompt1

tst L2 ;test L2

bne Prompt2

tst SHOWflag ;tests the show flag

bne SHOW

ldab TEMP ;loads TEMP val from keypad

cmpb #$F1 ;compares accumulator B to see if F1 is pressed

beq F1Press

ldab TEMP

cmpb #$F2 ;compares acc b to see if F2 pressed

lbeq F2Press

ldab TEMP

CMPB #$0A ;compares acc b to ent ascii value

LBEQ ENTERpress

CMPB #$08 ;compare acc b to BACKSPACE ascii value

LBEQ BACKSPACE

CMPB #$30 ;compares acc b to 30

LBLT ERROR ;branch to Loop if less than 30

CMPB #$39 ;compares acc b to 39

LBLE DIGITpress ;branches to digit

rts

SHOW:

ldd SHOWCOUNT ;this puts up the error screen for 1.5 seconds

SUBD #0001 ;it is based on passes through main showcount

std SHOWCOUNT

tst SHOWCOUNT

beq RESETSHOW

rts

RESETSHOW: clr TEMP ;resets the show count to display another error

clr SHOWflag

movw #1500, SHOWCOUNT

tst F1flag

bne RESETL1

tst F2flag

bne RESETL2

rts

RESETL1: movb #$01, L1 ;this restores the line 1 prompt, setting L1 flag

clr F1flag

clr F2flag

rts

RESETL2: movb #$01, L2 ;this restores the line 1 prompt, setting L2 flag

clr F1flag

clr F2flag

rts

Prompt1

movb #$01, DLINE1 ;set the Display Line 1 flag

rts

Prompt2:

movb #$01, DLINE2 ;set the Display Line 2 flag

rts

F1Press:

bclr PORTS, LED\_MSK\_1 ;turns off LED pair 1

movb #$01, CLINE1flag ;sets the clear entry line flag

movb #$01, F1flag ;set the f1 flag

movb #$01, FIRSTdig ;sets the first digit flag

clr F2flag ;clears the f2 flag

clr TEMP ;clears temp key value

clr COUNT ;clears count

clr ON\_1 ;clears the on flag for LED 1

jsr clrBUFF

movw #BUFFER, POINTER ;moves address of buffer into pointer

rts

F2Press:

bclr PORTS, LED\_MSK\_2 ;turns off LED pair 2

movb #$01, CLINE2flag ;sets the clear entry line 2 flag

movb #$01, F2flag ;set the f2 flag

movb #$01, FIRSTdig ;sets first digit flag

ldaa #$48 ;places the cursor in correct spot

jsr SETADDR

jsr CURSOR

clr F1flag ;clears the f1 flag

clr TEMP ;clears temp key value

clr COUNT ;clears count

clr ON\_2 ;clears the on flag for LED 2

jsr clrBUFF ;clears buffer

movw #BUFFER,POINTER ;moves address of buffer into pointer

rts

clrBUFF: movw #BUFFER, POINTER ;this subroutine clears the buffer

BUFF: ldx POINTER ;loads x with pointer

ldab 0,x ;loads b with contents of address in x

clrb ; clears b

stab 0,x ;stores back in buffer

inx

stx POINTER ; stores x back in pointer

dec COUNTBUFF ;dec count

tst COUNTBUFF

beq BUFFDONE

bra BUFF

BUFFDONE: movb #$06, COUNTBUFF ;resets the clear buffer counter

rts

ENTERpress:

tst FIRSTdig ;test for first digit pressed

bne NODIG ;branch to no digit flag set

tst COUNT ;test if count is greater than 0

beq ERROR

jsr BCD ;jumps to BCD conversion subroutine

clr TEMP ;clears the value stored in TEMP

rts

NODIG: movb #$01, NODIGflag ;sets the no digit flag

rts

DIGITpress: tst F1flag ;tests if the f1flag is set

bne PROCEED

tst F2flag ;tests if the f2flag is set

bne PROCEED

rts

PROCEED: movb #$01, DIGITflag ;sets the digit flag

ldab COUNT ;places current LCD address in accumulator A

cmpb #$05 ;makes sure don't type more than 5 digits

beq MAXdig ;clears dig flag so typing stops

rts

ERROR: rts

MAXdig: clr DIGITflag ;clears the digit flag, so no typing

rts

BACKSPACE: movb #$01, BACKflag ;sets the backspace flag

ldab COUNT ;makes sure that you can't backspace past 0

cmpb #$00

beq NOBS

rts

NOBS: clr BACKflag ;clears backspace flag

rts

BCD: ;ASCII to BCD converter

movw #BUFFER, POINTER ;moves address of buffer to pointer

movw #$0000, RESULT ;clears result

ldy #$0000 ;clears regis y

clra

clrb

LOOP1:

ldy #$000A ;loads 10 in register y

ldd RESULT ;loads result in acc d

emul ;multiplies d \* y stores low in d high in y

cpy #$0000 ;compares y to 0

bne TOOBIG ;if y is greater than 0 than entry is too large

std RESULT ;stores acc into result

ldx POINTER ;loads pointer into x

dab 0,X ;loads b with contents of address stored in x

subb #$30 ;subtracts 30 from b

clra

addd RESULT ;add acc d and RESULT

BCS TOOBIG ;brances to toobig if carry flag set

std RESULT ;stores acc d in result

dec COUNT ;decrement count

tst COUNT ;tests count to see if 0

beq DONE1

inx ;increments x

stx POINTER ;stores contents of x in pointer

bra LOOP1

TOOBIG: ;sets the magnitude too high flag

movb #$01, TOOHIGHflag

rts

ZERO: movb #$01, ZEROflag ;sets the zero flag

rts

DONE1: ;BGND ;finishes the BCD conversion

ldd #$0000 ;loads 0 into acc d

cpd RESULT ;compares d to result

beq ZERO ;if result is 0 branch to Zero

tst F1flag ;test if pair 1

bne SET1

tst F2flag ;tests if pair 2

bne SET2

rts

SET1:

movw #$0000, TICKS\_1 ;clears ticks

movw RESULT, TICKS\_1 ;stores new entered value in ticks

;BGND

movb #$01, ON\_1 ;sets the pair 1 on flag

movw #$0000, RESULT ;clears result

clr COUNT ;clears count

movw #$0000, COUNT\_1 ;clears COUNT for pair 1 in later task

clr t1state

clr t2state

clr t3state

rts

SET2:

movw #$0000, TICKS\_2 ;clears ticks

movw RESULT, TICKS\_2 ;stores new result in ticks2

;BGND

movb #$01, ON\_2 ;sets the pair 2 flag

movw #$0000, RESULT ;clears result

clr COUNT

movw #$0000, COUNT\_2 ;clears the count for pair 2 in later task

clr t3state

clr t4state

clr t5state

rts

;==========================================================================

;Keypad Driver

TASK\_2: tst FIRSTRUN1 ;test need to initialize

bne initKEY

bra startKEY

initKEY: jsr KP\_ACTIVE ;initializes the keypad

jsr INITKEY

jsr FLUSH\_BFR

clr FIRSTRUN1

startKEY: tst L$KEY\_FLG ;test key available flag

bne SKIP

jsr GETCHAR ;gets the character entered, stores in b

stab TEMP ;stores the contents of b in temp

SKIP: rts ;returns to subroutine

;==========================================================================

;DISPLAY Task

TASK\_3: tst FIRSTRUN2

bne initDIS

bra startDIS

initDIS: jsr INITLCD ;initializes the screen

jsr CLRSCREEN

clr FIRSTRUN2

startDIS: tst DLINE1 ;test display prompt line 1 flag

lbne initDisplay1

tst DLINE2 ;test display prompt line 2 flag

lbne initDisplay2

tst TOOHIGHflag ;test too high error flag

Lbne HIGHdis

tst NODIGflag ;test no digits entered flag

lbne NODIGdis

tst ZEROflag ;tests the Zero digits entered flag

lbne ZEROdis

tst F1flag ;test the F1flag

bne TOPtime

tst F2flag ;test the F2flag

bne BOTTOMtime

rts

TOPtime: tst CLINE1flag ;clear line 1 flag

lbne CLINE1dis

tst BACKflag ;backspace flag

bne BS

tst DIGITflag ;test Digitflag

beq RETURN

tst FIRSTdig

beq Ddigit

ldaa #$08 ;loads the LCD address for top line

jsr Ddigit\_1st

rts

BOTTOMtime: tst CLINE2flag

lbne CLINE2dis

tst BACKflag

bne BS

tst DIGITflag ;test Digitflag

beq RETURN

tst FIRSTdig

beq Ddigit

ldaa #$48 ;loads the LCD address for top line number

jsr Ddigit\_1st

rts

Ddigit\_1st: ;prepares the cursor for the first digit

jsr SETADDR

jsr CURSOR

clr FIRSTdig

Ddigit: ldab TEMP ;echoes the digit entered in keypad

jsr OUTCHAR

clr TEMP

ldx POINTER ;loads acc X with pointer

stab 0,X ;Stores contents of acc B into the location of add in x

inc COUNT

INX

stx POINTER

clr DIGITflag ;clears the DIGITflag so the program cycles

rts

RETURN: rts

BS:

jsr LOAD\_ADDR ;load current address in acc A

SUBA #$01 ;subtract 1 from address

jsr SETADDR

jsr CURSOR

ldab #$20 ;load space into B

jsr OUTCHAR ;prints character in B

jsr LOAD\_ADDR ;load current address in acc A

SUBA #$01

jsr SETADDR

jsr CURSOR

clr BACKflag ;clear the Backspace flag

clr TEMP

ldx POINTER ;load POINTER into X

dex

clr 0,X ;clears the value stored at the address in X

dec COUNT

stx POINTER

rts

initDisplay1:

tst FIRSTCH ;test if the first character is true

lbeq DCHAR

ldaa #$00 ;loads the LCD address

ldx #LINE1 ;starting address of string to be displayed

jsr DCHAR\_1st

lbra BOTTOM

initDisplay2:

tst FIRSTCH ;test if the first character is true

lbeq DCHAR

ldaa #$40 ;loads the LCD address

ldx #LINE2 ;starting address of string to be displayed

jsr DCHAR\_1st

lbra BOTTOM

HIGHdis: tst F1flag ;test which line to display the error

bne HIGH1

tst F2flag

bne HIGH2

rts

HIGH1: tst FIRSTCH ;test if the first character is true

lbeq DCHAR

ldaa #$08 ;loads the LCD address

ldx #HIGH ;starting address of string to be displayed

jsr DCHAR\_1st

lbra BOTTOM

HIGH2: tst FIRSTCH ;test if the first character is true

lbeq DCHAR

ldaa #$48 ;loads the LCD address

ldx #HIGH ;starting address of string to be displayed

jsr DCHAR\_1st

lbra BOTTOM

NODIGdis: tst F1flag

lbne NODIG1

tst F2flag

lbne NODIG2

rts

NODIG1: tst FIRSTCH ;test if the first character is true

lbeq DCHAR

ldaa #$08 ;loads the LCD address

ldx #NODIGIT ;starting address of string to be displayed

jsr DCHAR\_1st

lbra BOTTOM

NODIG2: tst FIRSTCH ;test if the first character is true

beq DCHAR

ldaa #$48 ;loads the LCD address

ldx #NODIGIT ;starting address of string to be displayed

jsr DCHAR\_1st

bra BOTTOM

ZEROdis: tst F1flag ;displays the Zero error message

bne ZERO1

tst F2flag

bne ZERO2

rts

ZERO1: tst FIRSTCH ;test if the first character is true

beq DCHAR

ldaa #$08 ;loads the LCD address

ldx #ZEROMAG ;starting address of string to be displayed

jsr DCHAR\_1st

bra BOTTOM

ZERO2: tst FIRSTCH ;test if the first character is true

beq DCHAR

ldaa #$48 ;loads the LCD address

ldx #ZEROMAG ;starting address of string to be displayed

jsr DCHAR\_1st

bra BOTTOM

CLINE1dis: tst FIRSTCH ;test if the first character is true

beq DCHAR

ldaa #$08 ;loads the LCD address

ldx #CLINE ;starting address of string to be displayed

jsr DCHAR\_1st

bra BOTTOM

CLINE2dis: tst FIRSTCH ;test if the first character is true

beq DCHAR

ldaa #$48 ;loads the LCD address

ldx #CLINE ;starting address of string to be displayed

jsr DCHAR\_1st

bra BOTTOM

DCHAR\_1st: STX DPTR ;store contents of X in DPTR

jsr SETADDR ;set the address of of cursor to current location

clr FIRSTCH ;clear variable FIRSTCH

DCHAR:

ldx DPTR ;load x with DPTR

ldab 0,x ;load acc b with contents of the address located in X

beq DONE

jsr OUTCHAR

inx

stx DPTR

BOTTOM:

tst FIRSTCH ;test if firstchar entered for another string

bne DONELINE ;branch to done

rts

DONE: ; sets firstchar to true to leave the DCHAR state

movb #$01, FIRSTCH

bra BOTTOM

DONELINE:

tst L1 ;test L1

bne CL1 ;branch to CL1 if not 0

tst L2 ; tests for the error/clear/print display flags

bne CL2 ; to go and clear them

tst TOOHIGHflag

bne CHIGH

tst NODIGflag

bne CNODIG

tst ZEROflag

bne CZERO

tst CLINE1flag

bne CLRCLINE1

tst CLINE2flag

bne CLRCLINE2

rts

CL1: clr L1 ;clears the print line 1 flag

clr DLINE1

rts

CL2: clr L2 ;clears the print line 2 flag

clr DLINE2

rts

CHIGH: clr TOOHIGHflag ;clears the high flag

movb #$01, SHOWflag ;sets showflag

rts

CNODIG: clr NODIGflag ;clears no digit flag

movb #$01, SHOWflag ;sets the show flag

rts

CZERO: clr ZEROflag ;clears the zero flag

movb #$01, SHOWflag ;sets the show flag

rts

CLRCLINE1: clr CLINE1flag ;clears the clear line flag

ldaa #$08 ;resets cursor

jsr SETADDR

jsr CURSOR

rts

CLRCLINE2: clr CLINE2flag ;clears the clear line flag

ldaa #$48 ;sets the cursor

jsr SETADDR

jsr CURSOR

rts

;=========================================================================

; Subroutine TASK\_4 ; pattern\_1 for LED pair 1

TASK\_4:

tst ON\_1

bne START1

rts

START1: clr F1flag

ldaa t1state ; get current t1state and branch accordingly

beq t1state0

deca

beq t1state1

deca

beq t1state2

deca

beq t1state3

deca

beq t1state4

deca

beq t1state5

deca

beq t1state6

rts ; undefined state - do nothing but return

t1state0: ; init TASK\_4

bclr PORTS, LED\_MSK\_1 ; ensure that LEDs are off when initialized

bset DDRS, LED\_MSK\_1 ; set LED\_MSK\_1 pins as PORTS outputs

movb #$01, t1state ; set next state

rts

t1state1: ; G, not R

bset PORTS, G\_LED\_1 ; set state1 pattern on LEDs

tst DONE\_1 ; check TASK\_4 done flag

beq exit\_t1s1 ; if not done, return

movb #$02, t1state ; if done, set next state

exit\_t1s1:

rts

t1state2: ; not G, not R

bclr PORTS, G\_LED\_1 ; set state2 pattern on LEDs

tst DONE\_1 ; check TASK\_4 done flag

beq exit\_t1s2 ; if not done, return

movb #$03, t1state ; if done, set next state

exit\_t1s2:

rts

t1state3: ; not G, R

bset PORTS, R\_LED\_1 ; set state3 pattern on LEDs

tst DONE\_1 ; check TASK\_4 done flag

beq exit\_t1s3 ; if not done, return

movb #$04, t1state ; if done, set next state

exit\_t1s3:

rts

t1state4: ; not G, not R

bclr PORTS, LED\_MSK\_1 ; set state4 pattern on LEDs

tst DONE\_1 ; check TASK\_4 done flag

beq exit\_t1s4 ; if not done, return

movb #$05, t1state ; if done, set next state

exit\_t1s4:

rts

t1state5: ; G, R

bset PORTS, LED\_MSK\_1 ; set state5 pattern on LEDs

tst DONE\_1 ; check TASK\_4 done flag

beq exit\_t1s5 ; if not done, return

movb #$06, t1state ; if done, set next state

exit\_t1s5

rts

t1state6: ; not G, not R

bclr PORTS, LED\_MSK\_1 ; set state6 pattern on LEDs

tst DONE\_1 ; check TASK\_4 done flag

beq exit\_t1s6 ; if not done, return

movb #$01, t1state ; if done, set next state

exit\_t1s6:

rts

; end TASK\_4

;

;==========================================================================

;

; Subroutine TASK\_5 ; count down LED\_1 pair

TASK\_5: tst ON\_1

bne STARTt5

rts

STARTt5: ldaa t2state ; get current t2state and branch accordingly

beq t2state0

deca

beq t2state1

deca

beq t2state2

rts ; undefined state - do nothing but return

t2state0: ; initialization for TASK\_2

clr DONE\_1

movb #$01, t2state ; set next state

rts

t2state1: ; (re)initialize COUNT\_1

movw TICKS\_1, COUNT\_1

ldx COUNT\_1

dex ; decrement COUNT\_1

stx COUNT\_1 ; store decremented COUNT\_1

clr DONE\_1

movb #$02, t2state ; set next state

rts

t2state2: ; count down COUNT\_1

ldx COUNT\_1

beq setdone\_1 ; test to see if COUNT\_1 is already zero

dex ; decrement COUNT\_1

stx COUNT\_1 ; store decremented COUNT\_1

bne exit\_t2s2 ; if not done, return

setdone\_1:

movb #$01, DONE\_1 ; if done, set DONE\_1 flag

movb #$01, t2state ; set next state

exit\_t2s2:

rts

; end TASK\_5

;

;=============================================================================

; Subroutine TASK\_6 ; pattern\_2

TASK\_6:

tst ON\_2

bne START2

rts

START2:

clr F2flag

ldaa t4state ; get current t4state and branch accordingly

beq t4state0

deca

beq t4state1

deca

beq t4state2

deca

beq t4state3

deca

beq t4state4

deca

beq t4state5

deca

beq t4state6

rts ; undefined state - do nothing but return

t4state0: ; init TASK\_6

bclr PORTS, LED\_MSK\_2 ; ensure that LEDs are off when initialized

bset DDRS, LED\_MSK\_2 ; set LED\_MSK\_1 pins as PORTS outputs

movb #$01, t4state ; set next state

rts

t4state1: ; G, not R

bset PORTS, G\_LED\_2 ; set state1 pattern on LEDs

tst DONE\_2 ; check TASK\_6 done flag

beq exit\_t4s1 ; if not done, return

movb #$02, t4state ; if done, set next state

exit\_t4s1:

rts

t4state2: ; not G, not R

bclr PORTS, G\_LED\_2 ; set state2 pattern on LEDs

tst DONE\_2 ; check TASK\_6 done flag

beq exit\_t4s2 ; if not done, return

movb #$03, t4state ; if done, set next state

exit\_t4s2:

rts

t4state3: ; not G, R

bset PORTS, R\_LED\_2 ; set state3 pattern on LEDs

tst DONE\_2 ; check TASK\_6 done flag

beq exit\_t4s3 ; if not done, return

movb #$04, t4state ; if done, set next state

exit\_t4s3:

rts

t4state4: ; not G, not R

bclr PORTS, LED\_MSK\_2 ; set state4 pattern on LEDs

tst DONE\_2 ; check TASK\_6 done flag

beq exit\_t4s4 ; if not done, return

movb #$05, t4state ; if done, set next state

exit\_t4s4:

rts

t4state5: ; G, R

bset PORTS, LED\_MSK\_2 ; set state5 pattern on LEDs

tst DONE\_2 ; check TASK\_6 done flag

beq exit\_t4s5 ; if not done, return

movb #$06, t4state ; if done, set next state

exit\_t4s5:

rts

t4state6: ; not G, not R

bclr PORTS, LED\_MSK\_2 ; set state6 pattern on LEDs

tst DONE\_2 ; check TASK\_6 done flag

beq exit\_t4s6 ; if not done, return

movb #$01, t4state ; if done, set next state

exit\_t4s6:

rts

; end TASK\_6

;

;=========================================================================

;

; Subroutine TASK\_7 ; count down LED\_2 pair

TASK\_7:

tst ON\_2

bne STARTt7

rts

STARTt7:

ldaa t5state ; get current t5state and branch accordingly

beq t5state0

deca

beq t5state1

deca

beq t5state2

rts ; undefined state - do nothing but return

t5state0: ; initialization for TASK\_2

clr DONE\_2

movb #$01, t5state ; set next state

rts

t5state1: ; (re)initialize COUNT\_2

movw TICKS\_2, COUNT\_2

ldx COUNT\_2

dex ; decrement COUNT\_2

stx COUNT\_2 ; store decremented COUNT\_2

clr DONE\_2

movb #$02, t5state ; set next state

rts

t5state2: ; count down COUNT\_2

ldx COUNT\_2

beq setdone\_2 ; test to see if COUNT\_2 is already zero

dex ; decrement COUNT\_2

stx COUNT\_2 ; store decremented COUNT\_2

bne exit\_t5s2 ; if not done, return

setdone\_2:

movb #$01, DONE\_2 ; if done, set DONE\_2 flag

movb #$01, t5state ; set next state

exit\_t5s2:

rts

; end TASK\_7

;==========================================================================

; Subroutine TASK\_8 ; delay 1.00ms

TASK\_8:

ldaa t3state ; get current t3state and branch accordingly

beq t3state0

deca

beq t3state1

rts ; undefined state - do nothing but return

t3state0: ; initialization for TASK\_3

; no initialization required

movb #$01, t3state ; set next state

rts

t3state1:

jsr DELAY\_1ms

rts

; end TASK\_8

;

;==========================================================================

LINE1: .ascii 'TIME 1= LED PAIR 1 <Press F1>'

.byte $00

LINE2: .ascii 'TIME 2= LED PAIR 2 <Press F2>'

.byte $00

HIGH: .ascii ' MAGNITUDE TOO LARGE '

.byte $00

NODIGIT: .ascii ' NO DIGITS '

.byte $00

ZEROMAG: .ascii 'ZERO MAGNITUDE INAPPROPRIATE '

.byte $00

CLINE: .ascii ' '

.byte $00

;==========================================================================

; Subroutine Delay\_1ms delays for ~1.00ms

DELAY\_1ms:

ldy #$0262

INNER: ; inside loop

cpy #0

beq EXIT

dey

bra INNER

EXIT:

rts ; exit DELAY\_1ms

; end subroutine DELAY\_1ms

;==========================================================================

.area interrupt\_vectors (abs)

.org $FFFE

.word \_\_start