```
;**************************************************************************************
;* Lab 4 [includes LibV2.2]                                                           *
;**************************************************************************************
;* Summary: Function Generator                                                        *
;*   -                                                                                *
;*                                                                                    *
;* Author: Benjamin Fields, Miles Alderman                                            *
;*   Cal Poly University                                                              *
;*   Spring 2022                                                                      *
;*                                                                                    *
;* Revision History:                                                                  *
;*   -                                                                                *
;*                                                                                    *
;* ToDo:                                                                              *
;*   -                                                                                *
;**************************************************************************************

;/------------------------------------------------------------------------------------\
;| 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, DISABLE_MOTOR
              XREF  STARTUP_MOTOR, UPDATE_MOTOR, CURRENT_MOTOR
              XREF  STARTUP_PWM, STARTUP_ATD0, STARTUP_ATD1
              XREF  OUTDACA, OUTDACB
              XREF  STARTUP_ENCODER, READ_ENCODER
              XREF  INITLCD, SETADDR, GETADDR, CURSOR_ON, CURSOR_OFF, DISP_OFF
              XREF  OUTCHAR, OUTCHAR_AT, OUTSTRING, OUTSTRING_AT
              XREF  INITKEY, LKEY_FLG, GETCHAR
              XREF  LCDTEMPLATE, UPDATELCD_L1, UPDATELCD_L2
              XREF  LVREF_BUF, LVACT_BUF, LERR_BUF,LEFF_BUF, LKP_BUF, LKI_BUF
              XREF  Entry, ISR_KEYPAD

;/------------------------------------------------------------------------------------\
;| Assembler Equates                                                                  |
;\------------------------------------------------------------------------------------/
```

```
; Constant values can be equated here

INTERVAL      EQU   $03E0        ;number of clock pulses that equal 0.1ms from 10.2MHz
clock

TIOS          EQU   $0040        ; addr of tios register
Chan0         EQU   %00000001    ;offset for channel 0

TCTL2         EQU   $0049        ; TCTL2 register that contains OL0 and OM0
OL0           EQU   %00000001    ; mask for 0L0 in TCTL2

TFLG1         EQU   $004E        ; Main timer interupt flag 1
C0F           EQU   %00000001    ; mask for C0F in TFLG1

TMSK1         EQU   $004C        ; Timer interupt mask
C0I           EQU   %00000001    ; mask for C0I

TSCR          EQU   $0046        ; Timer system control
TSCR_msk      EQU   %10100000    ; mask for timer system control

TCNT          EQU   $0044        ; first byte of timer count register

TC0           EQU   $0050        ; Timer channel 0 first (high) byte
;/------------------------------------------------------------------------------------\
;| Variables in RAM                                                                   |
;\------------------------------------------------------------------------------------/
; The following variables are located in unpaged ram

DEFAULT_RAM:  SECTION

NINT:               DS.B     1
timerstate:         DS.B     1
masterstate:        DS.B     1
displaystate:       DS.B     1
keystate:           DS.B     1
fxngenstate:        DS.B     1
RUN_FLG:            DS.B     1
FIRSTCH:            DS.B     1
DPTR:               DS.W     1
LASTCH:             DS.B     1
ERRORCOUNT:         DS.W     1
KEY_COUNT:          DS.B     1        ; Number of digits in buffer
KEY_BUFFER:         DS.B     1        ; Intermediate ascii key holder
BUFFER:             DS.B     3        ; Storage for pressed keys pre-translation
ERR_FLG             DS.B     1        ; Flag for there was an error, go into error state
TEMP:               DS.B     1        ; temporary used in ascii->bcd
RESULT:             DS.W     1        ; used for result storage of ascii -> bcd
conversion
COUNT:              DS.B     1        ; used for counting in ascii -> bcd
ECHO:               DS.B     1        ; echo character for indexed addressing storage
```

```
DMESS_EB:           DS.B     1
DMESS_EZ:           DS.B     1
DMESS_EN:           DS.B     1
DMESS_ENT:          DS.B     1
DMESS_BS:           DS.B     1
DMESS_NINT:         DS.B     1
DMESS_RESET:        DS.B     1
KEY_FLG:            DS.B     1

DMESS_tmp_err:      DS.B     1        ; for testing delete later TODO

NEW_BTI:            DS.B     1        ; from Murray: flag that ready for new BTI
CSEG:               DS.B     1
CINT:               DS.B     1
LSEG:               DS.B     1
VALUE:              DS.W     1
SEGINC:             DS.W     1
SEGPTR:             DS.W     1
WAVE:               DS.B     1

DWAVE:              DS.B     1
DPROMPT:            DS.B     1
CURSOR_ADD:         DS.B     1
WAVEPTR:            DS.W     1
NINT_OK:            DS.B     1
MS_WAVE_FLG:        DS.B     1


;/------------------------------------------------------------------------------------\
;|  Main Program Code                                                                  |
;\------------------------------------------------------------------------------------/
; Your code goes here

MyCode:       SECTION

main:
        clr   NINT
        clr   masterstate
        clr   timerstate
        clr   displaystate
        clr   keystate
        clr   fxngenstate
        clr   RUN_FLG
        clr   KEY_COUNT
        clr   KEY_BUFFER
        clr   ERR_FLG
        clr   TEMP
        clrw  RESULT
        clr   COUNT
        clr   DMESS_EB
        clr   DMESS_EZ
        clr   DMESS_EN
        clr   DMESS_ENT
        clr   DMESS_BS
        clr   DMESS_NINT
        clr   DMESS_RESET
        clr   KEY_FLG
        clr   NEW_BTI
        clr   DMESS_tmp_err
        clr   WAVE
        clr   DWAVE
        clr   DPROMPT
        clr   CURSOR_ADD
        clr   WAVEPTR
        clr   NINT_OK
        clr   ECHO
        clr   MS_WAVE_FLG
        clr   CSEG
        clr   LSEG
        clr   VALUE
        clr   SEGINC
        clr   SEGPTR
        clr   CINT
top:
        jsr   timer_channel_0
        jsr   MASTERMIND
        jsr   display
        jsr   keypad
        jsr   function_generator
        bra   top

spin:   bra  spin

;//////////////////////////TIMER CHANNEL 0/////////////////////////////////////
timer_channel_0:

        ldaa  timerstate
        beq   timerstate0
        deca
        beq   timerstate1
        deca
        beq   timerstate2
        rts

timerstate0:
        bset  TIOS, Chan0              ; set timer chan 0 for output compare
        bset  TCTL2, OL0               ; toggle tc0 for successful compare, OM0:OL0 
should be 01
        bset  TFLG1, C0F               ; clear timer output compare flag by writing a 1 
to it
```

```
        cli                            ; clear I bit to enable maskable interupts
        bset  TMSK1, C0I               ; enable timer overflow flag to trigger input
        bset  TSCR, TSCR_msk           ; freeze mode when debugger stops executing
        ldd   TCNT                     ; load $0044:$0045 into d
        addd  #INTERVAL                ; add interval to timer count
        std   TC0                      ; store in timer channel 0
        movb  #$01, timerstate
        bclr  TMSK1, C0I
        bclr  TCTL2, OL0               ; initiate with interrupts off
        rts

timerstate1:                           ; waiting to turn on interrupts
        tst   RUN_FLG                  ; if RUN=1, enable interrupts
        beq   timerstate1exit
        bset  TMSK1, C0I               ; enable timer overflow flag to trigger input
        bset  TCTL2, OL0               ; set output to toggle
        movb  #$02, timerstate         ; go to wait for interrupt disable

timerstate1exit:
        rts

timerstate2:                           ; waiting to turn off interrupts
        tst   RUN_FLG
        bne   timerstate2exit          ; if RUN=0, fall through else exit
        bclr  TMSK1, C0I               ; disable timer overflow flag
        bclr  TCTL2, OL0               ; clear toggle output
        movb  #$01, timerstate         ; go to wait for interrupt enable

timerstate2exit:
        rts

;//////////////////////////MASTERMIND/////////////////////////////////////////////////

MASTERMIND:

masterloop:
        ldaa   masterstate
        lbeq   masterstate0             ; init state
        deca
        lbeq   masterstate1             ; waiting for key press state
        deca
        lbeq   masterstate2             ; decode state
        deca
        lbeq   masterstate3             ; wave key state
        deca
        lbeq   masterstate4             ; nint key state
        deca
        lbeq   masterstate5             ; backspace key state
        deca
        lbeq   masterstate6             ; enter key state
        deca
        lbeq   masterstate7             ; error state


        bra   masterloop

masterstate0: ; // INIT STATE
///////////////////////////////////////////////////////////
        movb  #$01, masterstate
        movb  #$01, MS_WAVE_FLG         ;flag to set initial post decode waiting for wave
        rts

masterstate1: ; // WAITING FOR KEY STATE
//////////////////////////////////////////////////
        tst   ERR_FLG                    ; no error if flag is 0
        bne   errorstateset              ; go to error routine if there is one
        tst   KEY_FLG                    ; test if key has been pressed
        beq   exitmasterstate1
        movb  #$02, masterstate          ; if so go to decode state
        bra   exitmasterstate1

errorstateset:                          ; TODO: will the specific error determination all
be in error state?
        movb  #$07, masterstate
        rts

exitmasterstate1:
        rts

masterstate2: ; // DECODE STATE
/////////////////////////////////////////////////////////
        ; decode state will only figure out which key it is and then redirected to the
appropriate state
        ; error checking and more advanced case handling will be done in respective key
states


        ldaa  KEY_BUFFER                ; load ascii code for pressed key

        cmpa  #$08                      ; check if key is a backspace key
        beq   ms_goto_bs

        cmpa  #$0A                      ; check if key is an enter key
        beq   ms_goto_ent

        cmpa  #$30                      ; check if key is less than ascii for 0
        blt   ignore

        cmpa  #$39                      ; check if key is more than ascii 9
        bhi   ignore
        tst   MS_WAVE_FLG               ; check if we are currently waiting for wave
select
        beq   ms_goto_nint              ; if not branch to nint designation

ms_goto_wave:                                        ; if it didn't branch, we have a
digit!
        movb  #$03, masterstate         ; go into WAVE state next pass through MM

        rts

ms_goto_nint:
        movb  #$04, masterstate         ; go into NINT state next pass through MM
        rts

ms_goto_bs:
        movb  #$05, masterstate         ; go into backspace state next pass through MM
        rts

ms_goto_ent:
        movb  #$06, masterstate         ; go into enter state next pass through MM
        rts

ignore:
        jsr   clear_key
        rts


masterstate3: ;//  WAVE KEY STATE
///////////////////////////////////////////////////////////


waveinteruption:

        movb   #$00, RUN_FLG            ; turn off interrupt running

        ldaa   KEY_BUFFER
        cmpa   #$30                     ; if digit is 0, send display reset message
        beq    wavereset                ; keep waiting for a wave

        cmpa   #$31                     ; check if digit is less than 1
        blt    digexit

        cmpa   #$34                     ; check if digit is more than 4
        bhi    digexit

        suba   #$30                     ; made it through! convert from ascii to bcd
        staa   WAVE                     ; digit has now selected wave

        movb   #$01, DWAVE              ; set DWAVE flag - WHY?
        movb   #$01, NEW_BTI
        movb   #$00, MS_WAVE_FLG        ; on next pass through MM-digit assume for NINT

        bra    digexit

wavereset:
        movb   #$01, DMESS_RESET        ;send display reset message

digexit:
        jsr    clear_key
        rts


masterstate4: ;// NINT KEY STATE
///////////////////////////////////////////////////////////

nintput:
        ldab   KEY_COUNT
        cmpb   #$02                     ;test key count, if more than 2 in buffer
already,
        bhi    dig_exit                 ;don't store
        jsr    buffer_store             ;if ok store in buffer
        movb   #$01, DPROMPT            ;tell display to print the digit
        bra    dig_exit

dig_exit:
        jsr    clear_key
        rts


masterstate5: ;//  BACKSPACE KEY STATE
///////////////////////////////////////////////////////
        tst    KEY_COUNT                ; check that key count isn't at 0
        beq    bs_exit                  ; if there are no digits to backspace, ignore key
press
              ; if there's somethign to bs:
        dec    KEY_COUNT                ; decrement key_count
        movb   #$01, DMESS_BS           ; set the display backspace flag

        bra    bs_exit

bs_exit:
        jsr    clear_key
        rts

masterstate6: ;//  ENTER KEY STATE
///////////////////////////////////////////////////////////


        tst    MS_WAVE_FLG              ; test if currently accepting waves
        bne    ent_exit                 ; exit if not
        tst    DPROMPT                  ; first test if it is an appropriate time to
press enter
        bne    ent_exit                 ; exit if not

        tst    KEY_COUNT                ; check for zero key error
        lbeq   null_error

        jsr    asc_decode               ; translate ascii to BCD
        staa   ERR_FLG                  ; a is error code from ascii -> bcd, 0 if no
error

        cmpa   #$02
        beq    zero_error               ; test for zero result

        cmpa   #$01                     ; check for magnitude thats too large for nint
        beq    magnitude_error

enter:                                  ; only occurs if completely valid
        movb  #$01, DMESS_ENT           ; set the display enter flag
        clr   KEY_COUNT                 ; reset key_count to 0
        clr   DPROMPT                   ; clear prompt message
        movb  #$01, NINT_OK             ; signal OK to start generating
        movb  #$01, MS_WAVE_FLG         ; signal OK to accept new wave numbers
        ldd   RESULT
        stab  NINT
        bra   ent_exit

ent_exit:                               ; now that we are exiting...
        jsr   clear_key                 ; we are done with key
        rts

magnitude_error:
        movb  #$01, DMESS_EB            ; set magnitude error flag
        jsr   clear_key
        movb  #$07, masterstate         ; go to error decode state
        rts

zero_error:
        movb  #$01, DMESS_EZ            ; set error flag for zero nint error
        jsr   clear_key
        movb  #$07, masterstate         ; go to error decode state
        rts

null_error:
        movb  #01,  DMESS_EN
        jsr   clear_key
        movb  #$07, masterstate         ; go to error decode state
        rts
```

```
masterstate7: ;//  ERROR KEY
/////////////////////////////////////////////////////////////////
        ; test if it should stay in error state and not allow additional key presses
        ;load errocount1 into x
        ;load errorcount2 into y
        ;subtract them
        ; if 0, they're equal (error decrementing finished) and we want to exit error
state

        ldx   ERRORCOUNT
        cpx   #$0BB8                   ; compare error count to max value, if not exit
        bne   errorstate_exit          ; if max, value has been reset, duration done
        movb  #$01, masterstate        ; exit error state on next pass
        clr   ERR_FLG
        jsr   clear_key
        clr   KEY_COUNT


errorstate_exit:
        rts

;//////////////////////////DISPLAY////////////////////////////////////////////////////

display:
        ldaa   displaystate                 ; Display Task state cycling
        lbeq   displaystateinit0
        deca
        lbeq   displaystateinit1
        deca
        lbeq   displaystatehub
        deca
        lbeq   displaystateWAVE
        deca
        lbeq   displaystateNINT
        deca
        lbeq   displaystateBS
        deca
        lbeq   displaystateENT
        deca
        lbeq   displaystateEB
        deca
        lbeq   displaystateEZ
        deca
        lbeq   displaystateEN
        deca
        lbeq   errordelay
        deca
        lbeq   displaystatekeyreset
        rts

displaystateinit0:
        jsr   INITLCD
        movb  #$01, FIRSTCH
        movb  #$01, displaystate
        movw  #$0BB8, ERRORCOUNT
        rts


displaystateinit1:
        jsr   startscreen                     ;after initialization
        tst   FIRSTCH
        beq   displaystateinitexit
        movb  #$02, displaystate
        movb  #$00, LASTCH
        jsr   CURSOR_ON
        ldaa  #$00
        staa  CURSOR_ADD
        jsr   SETADDR

        rts

displaystateinitexit:
        rts

displaystatehub:
        tst   DWAVE
        lbne   displaysetWAVE
        tst   DPROMPT
        lbne   displaysetNINT
        tst   DMESS_BS
        lbne   displaysetBS
        tst   DMESS_ENT
        lbne   displaysetENT
        tst   DMESS_EB
        lbne   displaysetEB
        tst   DMESS_EZ
        lbne   displaysetEZ
        tst   DMESS_EN
        lbne   displaysetEN
        tst   DMESS_RESET
        lbne   displaystatereset

        rts

displaysetWAVE:
        movb  #$03, displaystate
        rts
```

```
displaysetNINT:
        movb  #$04, displaystate
        rts

displaysetBS:
        movb  #$05, displaystate
        rts

displaysetENT:
        movb  #$06, displaystate
        rts

displaysetEB:
        movb  #$07, displaystate
        decw  ERRORCOUNT
        rts

displaysetEZ:
        movb  #$08, displaystate
        decw  ERRORCOUNT
        rts

displaysetEN:
        movb  #$09, displaystate
        decw  ERRORCOUNT
        rts

errordelay:                                   ;error delay loop
        tstw  ERRORCOUNT                 ;
        beq   errorexit                       ;if error counter is 0, go to reset routine
        jsr   DELAY_1ms                       ;if error counts remain, delay 1ms
        decw  ERRORCOUNT                      ;if not, decrement error count
        rts

errorexit:                                    ;error reset routine
        movw  #$0BB8, ERRORCOUNT              ;reload error count timer
        movb  #$0B, displaystate              ;change display state to screen reprint
        rts

displaystatereset:
        tst   FIRSTCH                    ;test if cursor is in correct position
        lbeq  PUTCHAR                    ;if so start/continue printing
        tst   LASTCH
        bne   displaystateresetexit
        ldaa  #$40                       ;if not, new cursor address to first line, first
pos
        ldx   #CLR_MESS                  ;load x with blank lower line
        jsr   PUTCHAR1ST                 ;set cursor to stated cursor address
        rts
```

```
displaystateresetexit:
        movb  #$00,  LASTCH
        movb  #$01,  FIRSTCH
        movb  #$02,  displaystate
        movb  #$00,  DMESS_RESET
        ldaa  #$00
        jsr   SETADDR
        jsr   CURSOR_ON
        rts

displaystatekeyreset:
        tst   FIRSTCH                    ;test if cursor is in correct position
        lbeq  PUTCHAR                    ;if so start/continue printing
        tst   LASTCH
        bne   displaystatekeyexit
        ldaa  #$55                       ;if not, new cursor address to NINT cursor
position
        ldx   #KEYCLR_MESS               ;load x with black number input
        jsr   PUTCHAR1ST                 ;set cursor to stated cursor address
        rts

displaystatekeyexit:
        movb  #$00,  LASTCH
        movb  #$01,  FIRSTCH
        movb  #$02,  displaystate
        ldaa  #$5B
        staa  CURSOR_ADD
        jsr   CURSOR_ON
        jsr   SETADDR

        rts


startscreen:
        tst   FIRSTCH                    ;test if cursor is in correct position
        lbeq  PUTCHAR                    ;if so start/continue printing
        ldaa  #$00                       ;if not, new cursor address to first line, first
pos
        ldx   #SELECTION_SCREEN          ;load x with default screen message 1
        jsr   PUTCHAR1ST                 ;set cursor to stated cursor address
        rts

displaystateWAVE:
        tst   FIRSTCH                    ;test if this is first character in message
        lbeq  PUTCHAR                    ;if so keep printing
        tst   LASTCH
        bne   waveexit
        ldaa  WAVE
        cmpa  #$01
        beq   sawdisp                    ;go to saw display if WAVE = 1
        cmpa  #$02
        beq   sine7disp                  ;go to sine7 display if WAVE = 2
        cmpa  #$03
        beq   squaredisp                 ;go to square display if WAVE = 3
        cmpa  #$04
        beq   sine15disp                 ;go to sine15 display if WAVE = 4
        rts

sawdisp:
        ldaa  #$40                       ;load starting message address
        ldx   #SAW_MESS                  ;load 1st character of message memory location
        jsr   PUTCHAR1ST                 ;initialize printing
        rts

sine7disp:
        ldaa  #$40
        ldx   #SINE7_MESS
        jsr   PUTCHAR1ST
        rts

squaredisp:
        ldaa  #$40
        ldx   #SQUARE_MESS
        jsr   PUTCHAR1ST
        rts

sine15disp:
        ldaa  #$40
        ldx   #SINE15_MESS
        jsr   PUTCHAR1ST
        rts

waveexit:
        movb  #$00, LASTCH
        movb  #$00, DWAVE
        movb  #$02, displaystate
        movb  #$01, FIRSTCH
        ldaa  #$5B
        staa  CURSOR_ADD
        jsr   SETADDR
        jsr   CURSOR_ON
        rts


displaystateNINT:
        ldy   #BUFFER                    ;get address of first buffer character
        ldaa  KEY_COUNT                  ;get keycount and decrement for proper offset
        deca
        ldx   A, Y                       ;get offset address of key to print
        xgdx                             ;exchange values of d and x
        clrx                             ;clear x
        staa  ECHO
        ldab  ECHO
        ldaa  CURSOR_ADD
        jsr   OUTCHAR_AT                 ;print key at cursor address
        ldaa  CURSOR_ADD
        inca
        staa  CURSOR_ADD                 ;change cursor address to next digit location
        jsr   SETADDR                    ;move cursor to stated location
        movb  #$00, LASTCH
        movb  #$01, FIRSTCH
        movb  #$00, DPROMPT          ;reset flags, printing conditions
        movb  #$02, displaystate         ;go back to display hub
        rts

displaystateBS:
        ldaa  CURSOR_ADD                 ;load a with current cursor address
        deca                             ;go back a space
        staa  CURSOR_ADD                 ;save that address
        ldab  #$20
        jsr   OUTCHAR_AT                 ;print a space to previous digit location
        ldaa  CURSOR_ADD
        jsr   SETADDR                    ;move cursor to previous digit location
        movb  #$02, displaystate         ;go back to display hub
        movb  #$00, LASTCH
        movb  #$00, DMESS_BS             ;reset flags, printing conditions
        rts

displaystateENT:
        jsr   CURSOR_OFF
        movb  #$00, CURSOR_ADD           ;hide cursor
        ldaa  CURSOR_ADD
        jsr   SETADDR                    ;move cursor to hide address
        movb  #$02, displaystate         ;go back to display hub
        movb  #$00, DMESS_ENT            ;clear enter message flag
        rts


displaystateEB:
        tst   FIRSTCH
        lbeq  PUTCHAR
        tst   LASTCH
        bne   EBexit
        ldaa  #$55
        ldx   #EB_MESS
        jsr   PUTCHAR1ST
        rts

EBexit:
        movb  #$00, LASTCH
        movb  #$0A, displaystate
        movb  #$00, DMESS_EB
        movb  #$01, FIRSTCH
        rts

displaystateEZ:
        tst   FIRSTCH
        lbeq  PUTCHAR
        tst   LASTCH
        bne   EZexit
        ldaa  #$55
        ldx   #EZ_MESS
        jsr   PUTCHAR1ST
        rts

EZexit:
        movb  #$00, LASTCH
        movb  #$0A, displaystate
        movb  #$00, DMESS_EZ
        movb  #$01, FIRSTCH
        rts

displaystateEN:
        tst   FIRSTCH
        lbeq  PUTCHAR
        tst   LASTCH
        bne   ENexit
        ldaa  #$55
        ldx   #EN_MESS
        jsr   PUTCHAR1ST
        rts

ENexit:
        movb  #$00, LASTCH
        movb  #$0A, displaystate
        movb  #$00, DMESS_EN
        movb  #$01, FIRSTCH
        rts

PUTCHAR1ST:
        stx   DPTR
        jsr   SETADDR
        clr   FIRSTCH

PUTCHAR:
        ldx   DPTR
        ldab  0,X
        beq   DONE
        inx
        stx   DPTR
        jsr   OUTCHAR
        rts
DONE:
        movb  #$01, FIRSTCH
        movb  #$01, LASTCH
        rts
;//////////////////////////KEYPAD/////////////////////////////////////////////////////

keypad:

keyloop:
        ldaa  keystate                 ; get current t1state and branch accordingly
        beq   keystate0
        deca
        beq   keystate1
        deca
        beq   keystate2

        bra   keyloop


keystate0:                             ;init keypad state
        jsr   INITKEY
        movb  #$01, keystate           ;go to keystate 1 on next passthrough
        rts

keystate1:
        tst   LKEY_FLG                 ;see if key was pressed
        beq   exitkeystate1            ;if no key pressed, rts
        movb  #$01,   KEY_FLG          ;set keyflag if key pressed
        jsr   GETCHAR                  ;get character
        stab  KEY_BUFFER               ;store character in key buffer
        movb  #$02, keystate           ;go to state 2 on next passthrough

exitkeystate1:
        rts

keystate2:
        tst   KEY_FLG
        bne   exitkeystate2            ;if key flag cleared by mastermind
        movb  #$01, keystate           ;go back to state 1

exitkeystate2:
        rts

;//////////////////////////FUNCTION_GENERATOR/////////////////////////////////////////

function_generator:

        ldaa  fxngenstate
        lbeq   fxngenstate0
        deca
        lbeq   fxngenstate1
        deca
        lbeq   fxngenstate2
        deca
        lbeq   fxngenstate3
        deca
        lbeq   fxngenstate4

fxngenstate0:
        movb  #$01, fxngenstate     ; initialize

fxngenstate1:
        tst   WAVE                  ; test if new wave has been selected
        beq   fxns1exit             ; if not, exit
        ldaa  WAVE                  ; test respective waves for loading proper addresses
        cmpa  #$01
        beq   sawfxnset
        cmpa  #$02
        beq   sine7fxnset
        cmpa  #$03
        beq   squarefxnset
        cmpa  #$04
        beq   sine15fxnset
        bra   fxns1exit

sawfxnset:
        ldx   #SAW_WAVE              ; load respective wave beginning address
        stx   WAVEPTR                ; into wave pointer
        bra   fxnsetexit

sine7fxnset:
        ldx   #SINE7_WAVE
        stx   WAVEPTR
        bra   fxnsetexit

squarefxnset:
        ldx   #SQUARE_WAVE
        stx   WAVEPTR
        bra   fxnsetexit

sine15fxnset:
        ldx   #SINE15_WAVE
        stx   WAVEPTR
        bra   fxnsetexit

fxnsetexit:
        movb  #$02, fxngenstate     ; if so, move to wave loading next pass
        rts
```

```
fxns1exit:
        rts


fxngenstate2:                       ; NEW WAVE

        tst    DWAVE                ; wait for display of wave message
        bne    fxngens2exit
        ldx    WAVEPTR              ; point to start of data for wave
        movb   0,X, CSEG            ; get number of wave segments
        movw   1,X, VALUE           ; get initial value for DAC
        movb   3,X, LSEG            ; load segment length
        movw   4,X, SEGINC          ; load segment increment
        inx                         ; inc SEGPTR to next segment
        inx
        inx
        inx
        inx
        inx
        stx    SEGPTR               ; store incremented SEGPTR for next segment
        movb   #$03, fxngenstate    ; set next state
fxngens2exit:
        rts

fxngenstate3:
        tst   NINT_OK               ;test if NINT value successfully entered
        beq   fxngens3exit
        movb  #$01, RUN_FLG
        movb  #$00, NINT_OK
        movb  #$04, fxngenstate
        ldaa  LSEG
        adda  #$01
        staa  LSEG
        rts

fxngens3exit:
        rts

fxngenstate4:                   ; DISPLAY WAVE
        tst    RUN_FLG
        beq    fxngens4c        ; do not update function generator if RUN=0
        tst    NEW_BTI
        beq    fxngens4e        ; do not update function generator if NEWBTI=0
        dec    LSEG             ; decrement segment length counter
        bne    fxngens4b        ; if not at end, simply update DAC output
        dec    CSEG             ; if at end, decrement segment counter
        bne    fxngens4a        ; if not last segment, skip reinit of wave
        ldx    WAVEPTR          ; point to start of data for wave
        movb   0,X, CSEG        ; get number of wave segments
        inx                     ; inc SEGPTR to start of first segment
        inx
        inx
        stx    SEGPTR           ; store incremented SEGPTR
fxngens4a:
        ldx    SEGPTR           ; point to start of new segment
        movb   0,X, LSEG        ; initialize segment length counter
        movw   1,X, SEGINC      ; load segment increment
        inx                     ; inc SEGPTR to next segment
        inx
        inx
        stx    SEGPTR           ; store incremented SEGPTR
fxngens4b:
        ldd    VALUE            ; get current DAC input value
        addd   SEGINC           ; add SEGINC to current DAC input value
        std    VALUE            ; store incremented DAC input value
        bra    fxngens4d
fxngens4c:
        movb   #$01, fxngenstate    ; set next state
fxngens4d:
        clr    NEW_BTI
fxngens4e:
        rts

ISR:

        dec   CINT
        bne   NOT_YET
        ldd   VALUE
        jsr   OUTDACA
        movb  NINT, CINT
        movb  #$01, NEW_BTI

NOT_YET:
        ldd   TC0                     ; load $0044:$0045 into d
        addd  #INTERVAL                ; add interval to timer count
        std   TC0                      ; store in timer channel 0
        bset  TFLG1, C0F               ; clear timer output compare flag by writing a 1
to it
        rti




;/-------------------------------------------------------------------------------------\
;| Subroutines                                                                         |
;\-------------------------------------------------------------------------------------/
; General purpose subroutines go here
;// BUFFER STORE //
buffer_store:
        ldaa  KEY_COUNT
        cmpa  #$03                     ; make sure there aren't more than 3 keys in
buffer (would overflow)
        bhs   clear_key
        movb  #$01, DMESS_NINT
        ldx   #BUFFER

        ldab  KEY_BUFFER               ; store digit
        stab  a, x               ; a should still be key_count
        inc   KEY_COUNT                ; +1 keys in buffer now
        rts                            ; exit, key clearing done in state3 before exit

 ;// CLEAR KEY ////
clear_key:
        clr   KEY_BUFFER
        clr   KEY_FLG
        movb  #$01, masterstate         ; go back to waiting for key press
        rts

;//  ASCII DECODE //////
asc_decode:
        ; NOTE: most of these variables could be circumvented by using storage that's
already defined
        ; this could be implemented later, but the fast solution was used first.
        ; OUTPUTS: x result, a is error code
        ; ERROR CODES: 0 if no error, 1 if overflow error, 2 if zero error
        clr   COUNT                          ; prep intermediate variables
        clr   TEMP
        clrw  RESULT

        movb  KEY_COUNT, COUNT              ; move byte to decrementer

        ; store the registers and accumulators
        pshc                                 ; push ccr to stack
        pshb                                 ; push b to stack
        pshy                                 ; push y to stack

        ldx   #BUFFER                        ; load x for indexed addressing


while:                                       ; loop through each digit

        ldaa  TEMP                           ; counter for number of digits to index

        ldab  a,x                            ; retrieve desired value from buffer
        subb  #$30                           ; get BCD by subtracting 30
        inc   TEMP                           ; increment TEMP

        ldy   RESULT                         ; load y with current result
        aby                                  ; add latest digit
        sty   RESULT                         ; then store back in result
        ldd   RESULT
        tsta
        bne   overflowerror                  ; check that adding didn't create overflow

        dec   COUNT                          ; decrement count
        ldab  COUNT                          ; load into b to check if done
        cmpb  #$00                           ; if count is zero, the subroutine is done
        beq   return                         ; if that was last digit, don't mult by 10

        ldd   #$000A                         ; for mult by 10, y already loaded
        emul                                 ; y x d, store in y:d
        cmpa  #$00                           ; y should be empty or we have overflow
        bne   overflowerror                  ; error routine if error occurred
        std   RESULT                         ; store the result in result
        bra   while                          ; keep looping while count > 0

return:
        ldx   RESULT                         ; load x
        cpx   #$0000                           ; check for zero result
        beq   zeroerror                      ; if result was empty, zero error
        ldaa  #$00                           ; if it got here, it didn't hit an error
                    ; return result in x register
        bra   restore                        ; exit routine

overflowerror:
        ldaa  #$01                           ; error code for overflow error
        bra   restore                        ; exit routine

zeroerror:
        ldaa  #$02                           ; error code for zero error
        bra   restore                        ; exit routine

restore:                                     ; LIFO, restores accumulators, registers,
CCR,
        puly                                 ; restore index register y from stack
        pulb                                 ; restore accumulator b from stack
        pulc                                 ; restore ccr from stack
        rts                                  ; ouput is

DELAY_1ms:
        ldy   #$0584
INNER:                                 ; inside loop
        cpy   #0
        beq   EXIT
        dey
        bra   INNER
EXIT:
        rts                            ; exit DELAY_1ms
```

```
;/-------------------------------------------------------------------------------------\
;| ASCII Messages and Constant Data                                                    |
;\-------------------------------------------------------------------------------------/
; Any constants can be defined here

SELECTION_SCREEN:       DC.B    '1: SAW, 2: SINE-7, 3: SQUARE, 4: SINE-15', $00
SAW_MESS:               DC.B    'SAWTOOTH WAVE        NINT:     [1-->255]', $00
SINE7_MESS:             DC.B    '7-SEGMENT SINE WAVE  NINT:     [1-->255]', $00
SQUARE_MESS:            DC.B    'SQUARE WAVE          NINT:     [1-->255]', $00
SINE15_MESS:            DC.B    '15-SEGMENT SINE WAVE NINT:     [1-->255]', $00
KEYCLR_MESS:            DC.B    'NINT:     [1-->255]', $00
CLR_MESS:               DC.B    '                                        ', $00
EB_MESS:                DC.B    'MAGNITUDE TOO LARGE', $00
EZ_MESS:                DC.B    'ZERO MAGNITUDE     ', $00
EN_MESS:                DC.B    'NO DIGITS ENTERED  ', $00

SAW_WAVE:
                        DC.B  2                 ; number of segments for SAWTOOTH
                        DC.W  0                 ; initial DAC input value
                        DC.B  19                ; length for segment_1
                        DC.W  172               ; increment for segment_1
                        DC.B  1                 ; length for segment_2
                        DC.W  -3268             ; increment for segment_2

SINE7_WAVE:
                        DC.B  7                 ; number of segments for SINE-7
                        DC.W  2048              ; initial DAC input value
                        DC.B  25                ; length for segment_1
                        DC.W  32                ; increment for segment_1
                        DC.B  50                ; length for segment_2
                        DC.W  16                ; increment for segment_2
                        DC.B  50                ; length for segment_3
                        DC.W  -16               ; increment for segment_3
                        DC.B  50                ; length for segment_4
                        DC.W  -32               ; increment for segment_4
                        DC.B  50                ; length for segment_5
                        DC.W  -16               ; increment for segment_5
                        DC.B  50                ; length for segment_6
                        DC.W  16                ; increment for segment_6
                        DC.B  25                ; length for segment_7
                        DC.W  32                ; increment for segment_7
SQUARE_WAVE:
                        DC.B  4                 ; number of segments for SQUARE
                        DC.W  3276              ; initial DAC input value
                        DC.B  9                 ; length for segment_1
                        DC.W  0                 ; increment for segment_1
                        DC.B  1                 ; length for segment_2
                        DC.W  -3276             ; increment for segment_2
                        DC.B  9                 ; length for segment_3
                        DC.W  0                 ; increment for segment_3
                        DC.B  1                 ; length for segment_4
                        DC.W  3276              ; increment for segment_4

SINE15_WAVE:
                        DC.B  15                ; number of segments for SINE-15
                        DC.W  2048              ; initial DAC input value
                        DC.B  10                ; length for segment_1
                        DC.W  41                ; increment for segment_1
                        DC.B  21                ; length for segment_2
                        DC.W  37                ; increment for segment_2
                        DC.B  21                ; length for segment_3
                        DC.W  25                ; increment for segment_3
                        DC.B  21                ; length for segment_4
                        DC.W  9                 ; increment for segment_4
                        DC.B  21                ; length for segment_5
                        DC.W  -9                ; increment for segment_5
                        DC.B  21                ; length for segment_6
                        DC.W  -25               ; increment for segment_6
                        DC.B  21                ; length for segment_7
                        DC.W  -37               ; increment for segment_7
                        DC.B  20                ; length for segment_8
                        DC.W  -41               ; increment for segment_8
                        DC.B  21                ; length for segment_9
                        DC.W  -37               ; increment for segment_9
                        DC.B  21                ; length for segment_10
                        DC.W  -25               ; increment for segment_10
                        DC.B  21                ; length for segment_11
                        DC.W  -9                ; increment for segment_11
                        DC.B  21                ; length for segment_12
                        DC.W  9                 ; increment for segment_12
                        DC.B  21                ; length for segment_13
                        DC.W  25                ; increment for segment_13
                        DC.B  21                ; length for segment_14
                        DC.W  37                ; increment for segment_14
                        DC.B  10                ; length for segment_15
                        DC.W  41                ; increment for segment_15



;/------------------------------------------------------------------------------------\
;| Vectors                                                                            |
;\------------------------------------------------------------------------------------/
; Add interrupt and reset vectors here
        ORG   $FFEE                    ; timer ch0 vector address
        DC.W  ISR
        ORG   $FFFE                    ; reset vector address
        DC.W  Entry
        ORG   $FFCE                    ; Key Wakeup interrupt vector address [Port J]
```

```
DC.W  ISR_KEYPAD
```