```
; 4 KB Bios
; can be assembled correctly now
; corrected and compatibilty added by Malban
;
; assemble with comand line:
; .\ass\as09.exe -w200 -h0 -l -mcti bios.asm >error
;
; used the 6809 assembler:
; as09 [1.11].
; Copyright 1990-1994, Frank A. Vorstenbosch, Kingswood Software.
; Available at:
; http://www.falstaff.demon.co.uk/cross.html
;
                CODE
                ORG     $F000
                DATA
                ORG     $F000
                BSS
                ORG     $F000
                CODE
;-----------------------------------------------------------------------;
; This disassembly of the Vectrex ROM was done by Bruce Tomlin          ;
; (btomlin@aol.com), and is based in part on a disassembly done by      ;
; Fred Taft (fred@hp-pcd.cv.hp.com).                                    ;
;-----------------------------------------------------------------------;

                INCLUDE "VECTREX.INC"

;-----------------------------------------------------------------------;
;       F000    Start                                                   ;
;                                                                       ;
; Jump here to restart the Vectrex and re-initialize the OS.  If the    ;
; cold start flag is correct (it should be unless you just turned the   ;
; Vectrex on), the cold start code is skipped.                          ;
;                                                                       ;
; On cold start, the high score is cleared, and the power-on screen     ;
; is displayed with the power-on music.                                 ;
;-----------------------------------------------------------------------;

Start:          LDS     #Vec_Default_Stk ;Set up stack pointer
                JSR     Init_OS         ;Initialize system
                LDD     #$7321          ;Check cold start flag
                CMPD    Vec_Cold_Flag
                BEQ     Warm_Start      ;Branch if warm start
                STD     Vec_Cold_Flag   ;Initialize cold start flag
                INC     $C83B           ;Set high score invalid flag
                LDX     #Vec_High_Score ;Clear high score
                JSR     Clear_Score

;       First power-up loop.  This prints the "VECTREX"
;       power-on screen and plays the power-on music.

LF01C:          JSR     DP_to_C8        ;DP to RAM
                LDD     <Vec_Loop_Count ;When we have looped 257 times,
```

```
                CMPD    #$0101
                BNE     LF029
                STB     <Vec_Music_Flag ;start the intro music

;       Get the line pattern for the boundary box lines

LF029:          ASRB                    ;Get line pattern from table
                ANDB    #$03
                LDX     #DF0FD
                LDB     B,X
                STB     <Vec_Pattern    ;Store pattern

                LDB     #$02            ;Set up counter for two boxes
                STB     <$C824

;       Play the intro music

                LDU     #Intro_Music    ;Get address of music
                JSR     Init_Music_chk  ;Initialize the music
                JSR     Wait_Recal      ;Wait for next frame
                JSR     Do_Sound        ;Play music if active

;       Display power-up message

                JSR     Intensity_7F    ;Normal brightness
                LDA     $C826           ;Alternate size every 32 loops
                LDU     #Vec_Title     ;Load address of double high text
                BITA    #$20
                BEQ     LF052
                LEAU    <<(Vec_Title_2-Vec_Title),U  ;Skip double high text
LF052:          JSR     Print_List_hw   ;Print startup text

;       Draw the frame boxes

                LDX     #Intro_Boxes
LF058:          JSR     Moveto_ix_FF    ;Move to start of line
                LDA     #$03            ;Draw 4 lines
                JSR     Draw_Pat_VL_a
                DEC     $C824           ;Go back for next box
                BNE     LF058
                LDA     Vec_Loop_Count
                CMPA    #$01            ;Repeat for 512 counts
                BLS     LF01C

;-----------------------------------------------------------------------;
;       F06C    Warm_Start                                              ;
;                                                                       ;
; Jump here to restart the Vectrex without re-initializing the OS.      ;
;-----------------------------------------------------------------------;

;       Prepare for ROM check

Warm_Start:     JSR     DP_to_C8        ;DP to RAM
                LDA     #$CC            ;Set new line pattern
```

```
                STA     <Vec_Pattern
                LDD     #Copyright_Str  ;Save copyright string addr
                STD     <$C839
                CLR     <Vec_Loop_Count ;Clear loop counter
                CLR     <Vec_Loop_Count+1

;       Check for valid cartridge ROM

                LDU     #$0000          ;Look at address zero
                LDX     #Copyright_Str
                LDB     #$0B            ;11 bytes long
LF084:          LDA     ,U+             ;Compare next byte
                CMPA    ,X+
                BEQ     LF097           ;Okay if match
                CMPB    #$01            ;Not okay if last byte wrong
                BEQ     LF092
                CMPB    #$05            ;Okay if date wrong
                BLS     LF097
LF092:          LDU     #$E000          ;Bad cart; load Mine Storm addr
                BRA     LF09E

LF097:          DECB                    ;Go back for next byte
                BNE     LF084
                STB     <$C839          ;Store zero as address of
                STB     <$C83A          ;copyright string

;       Prepare to play game start-up music

LF09E:          INC     <Vec_Music_Flag ;Set music enable flag
                STU     <Vec_Run_Index  ;Save address of header
                LDU     ,U              ;Get address of music

;       Second power-up loop.  This prints the name of the
;       game, the copyright, and plays the start-up music

LF0A4:          JSR     DP_to_C8        ;DP to RAM
                LDD     #$F848          ;???
                STD     <Vec_Text_HW
                JSR     Init_Music_chk  ;Initialize the music
                JSR     Wait_Recal      ;Wait for next frame
                JSR     Do_Sound        ;Play the music if active

;       Display cartridge GCE copyright string

                JSR     Intensity_7F    ;Normal brightness
                LDD     #$C0C0          ;Print copyright string
                LDU     $C839
                JSR     Print_Str_d

;       Display current high score if any

                LDA     $C83B           ;Skip if no high score
                BNE     LF0D2
                DECA
```

```
                LDU     #Vec_High_Score
                STA     6,U
                LDD     #$68D0
                JSR     Print_Str_d

;       Display cartridge name

LF0D2:          LDU     Vec_Run_Index   ;Get cartridge header addr
                LEAU    2,U             ;Skip music addr
                JSR     Print_List_hw   ;Print it
                LDA     Vec_Music_Flag  ;Go back if music still playing
                BNE     LF0A4
                LDX     Vec_Loop_Count  ;Go back if count less than 126
                CMPX    #$007D
                BLS     LF0A4
                JMP     1,U             ;Jump into cartridge

;       Outer box (y,x)

Intro_Boxes:    FCB     $40,$D6
                FCB     $00,$56
                FCB     $81,$00
                FCB     $00,$A9
                FCB     $7E,$00

;       Inner box (y,x)

                FCB     $39,$DC
                FCB     $8E,$00
                FCB     $00,$4A
                FCB     $72,$00
                FCB     $00,$B6

;       Line patterns for boundary boxes

DF0FD:          FCB     %11100000
                FCB     %00111000
                FCB     %00001110
                FCB     %00000011

;       Copyright string

Copyright_Str:  FCC     "g GCE 1982"
                FCB     $80
here__:
Copyright_Len   EQU     here__-Copyright_Str

;       Title strings

Vec_Title:      FCB     $F1,$60         ;Height, width
                FCB     $27,$CF         ;Y,X
                FCC     "VECTREX"
                FCB     $80
```

```
Vec_Title_2:    FCB     $F3,$60         ;Height, width
                FCB     $26,$CF         ;Y,X
                FCC     "VECTREX"
                FCB     $80

                FCB     $FC,$60         ;Height, width
                FCB     $DF,$E9         ;Y,X
                FCC     "GCE"
                FCB     $80

                FCB     $FC,$38         ;Height, width
                FCB     $CC,$D1         ;Y,X
                FCC     "ENTERTAINING"
                FCB     $80

                FCB     $FC,$38         ;Height, width
                FCB     $BC,$DC         ;Y,X
                FCC     "NEW IDEAS"
                FCB     $80

                FCB     $00

;---------------------------------------------------------------------------;
;       F14C    Init_VIA                                                    ;
;                                                                           ;
; This routine is invoked during powerup, to initialize the VIA chip.       ;
; Among other things, it initializes the scale factor to 0x7F, and          ;
; sets up the direction for the port A and B data lines.                    ;
;                                                                           ;
; EXIT: DP = $D0                                                            ;
;                                                                           ;
;       D-reg, X-reg trashed                                                ;
;---------------------------------------------------------------------------;

Init_VIA:       BSR     DP_to_D0
                LDD     #$9FFF          ;Port A=all output
                STD     <VIA_DDR_b      ;Port B=OIIOOOOO
                LDD     #$0100          ;Port B sound BDIR=1
                STD     <VIA_port_b
                LDD     #$987F          ;ACR=$98 T1->PB7 enabled
                STA     <VIA_aux_cntl   ;auxiliary control register
                STB     <VIA_t1_cnt_lo  ;T1CL=$7F scale factor?
                JSR     Reset0Ref
                BRA     Set_Refresh

;---------------------------------------------------------------------------;
;       F164    Init_OS_RAM                                                 ;
;                                                                           ;
; This routine first clears the block of RAM in the range $C800 to          ;
; $C87A, and then it initializes the dot dwell time, the refresh time,      ;
; and the joystick enable flags.                                            ;
;                                                                           ;
; EXIT: DP = $C8                                                            ;
;                                                                           ;
```

```
;       D-reg, X-reg trashed                                            ;
;-----------------------------------------------------------------------;

Init_OS_RAM:    BSR     DP_to_C8        ;DP to RAM
                LDB     #$7A            ;Clear $C800-$C87A
                LDX     #$C800
                JSR     Clear_x_b
                LDD     #Vec_Random_Seed;Point $C87B to $C87D
                STD     <Vec_Seed_Ptr
LF173:          INC     <Vec_Random_Seed;Make sure random number
                BEQ     LF173           ;seed is non-zero!
                LDA     #$05            ;Init dot dwell (brightness)
                STA     <Vec_Dot_Dwell
                LDD     #$3075          ;Init refresh time to $7530
                STD     <$C83D
                LDD     #$0103          ;Init joystick enable flags
                STD     <Vec_Joy_Mux_1_X
                LDD     #$0507
                STD     <Vec_Joy_Mux_2_X
                RTS

;-----------------------------------------------------------------------;
;       F18B    Init_OS                                                 ;
;                                                                       ;
; This routine is responsible for setting up the initial system state,  ;
; each time the system is either reset or powered up.  It will          ;
; initialize the OS RAM area, initialize the VIA chip, and then clear   ;
; all the registers on the sound chip.                                  ;
;                                                                       ;
; EXIT: DP = $D0                                                        ;
;                                                                       ;
;       D-reg, X-reg trashed                                            ;
;-----------------------------------------------------------------------;

Init_OS:        BSR     Init_OS_RAM
                BSR     Init_VIA
                JMP     Clear_Sound

;-----------------------------------------------------------------------;
;       F192    Wait_Recal                                              ;
;                                                                       ;
; Wait for t2 (the refresh timer) to timeout, then restart it using     ;
; the value in $C83D.  then, recalibrate the vector generators to the   ;
; origin (0,0).  This routine MUST be called once every refresh         ;
; cycle, or your vectors will get out of whack.  This routine calls     ;
; Reset0Ref, so the integrators are left in zero mode.                  ;
;                                                                       ;
; EXIT: DP = $D0                                                        ;
;                                                                       ;
;       D-reg, X-reg trashed                                            ;
;-----------------------------------------------------------------------;

Wait_Recal:     LDX     Vec_Loop_Count  ;Increment loop counter
                LEAX    1,X
```

```
                STX     Vec_Loop_Count
                BSR     DP_to_D0        ;DP to I/O
                LDA     #$20
LF19E:          BITA    <VIA_int_flags  ;Wait for timer t2
                BEQ     LF19E

;-----------------------------------------------------------------------;
;       F1A2    Set_Refresh                                             ;
;                                                                       ;
; This routine loads the refresh timer (t2) with the value in $C83D-    ;
; $C83E, and recalibrates the vector generators, thus causing the pen   ;
; to be left at the origin (0,0).  The high order byte for the timer    ;
; is loaded from $C83E, and the low order byte is loaded from $C83D.    ;
; The refresh rate is calculated as follows:                            ;
;                                                                       ;
;   rate = (C83E)(C83D) / 1.5 mhz                                       ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;                                                                       ;
;       D-reg trashed                                                ;
;-----------------------------------------------------------------------;

Set_Refresh:    LDD     $C83D           ;Store refresh value
                STD     <VIA_t2_lo      ;into timer t2
                JMP     Recalibrate

;-----------------------------------------------------------------------;
;       F1AA    DP_to_D0                                                ;
;                                                                       ;
; Sets the DP register to $D0, so that all direct page addressing will  ;
; start at $D000 (the hardware I/O area).                               ;
;                                                                       ;
; EXIT: DP = $D0                                                        ;
;       A-reg = $D0                                                     ;
;-----------------------------------------------------------------------;

DP_to_D0:       LDA     #$D0
                TFR     A,DP
                RTS

;-----------------------------------------------------------------------;
;       F1AF    DP_to_C8                                                ;
;                                                                       ;
; Sets the DP register to $C8, so that all direct page addressing will  ;
; start at $C800 (OS RAM area).                                         ;
;                                                                       ;
;                                                                       ;
; EXIT: DP = $C8                                                        ;
;       A-reg = $C8                                                     ;
;-----------------------------------------------------------------------;

DP_to_C8:       LDA     #$C8
                TFR     A,DP
                RTS
```

```
;-------------------------------------------------------------------------;
;       F1B4    Read_Btns_Mask                                            ;
;       F1BA    Read_Btns                                                 ;
;                                                                         ;
; Both of these routines read the button states on the two joysticks,     ;
; and return their state in the following RAM locations:                  ;
;                                                                         ;
;       joystick 1, button 1:  $C812 = $01                                ;
;                   button 2:  $C813 = $02                                ;
;                   button 3:  $C814 = $04                                ;
;                   button 4:  $C815 = $08                                ;
;       joystick 2, button 1:  $C816 = $10                                ;
;                   button 2:  $C817 = $20                                ;
;                   button 3:  $C818 = $40                                ;
;                   button 4:  $C819 = $80                                ;
;                                                                         ;
;       C80F: Contains current state of all buttons;                      ;
;             1 = depressed, 0 = not depressed                            ;
;                                                                         ;
;       C810: Contains state of all buttons from LAST time these          ;
;             routines were called; if Read_Btns_Mask was called,         ;
;             then this is AND'ed with the passed in mask.                ;
;                                                                         ;
;       C811: Contains the same information as $C812-$C819                ;
;                                                                         ;
;        Bit 7                                        Bit 0               ;
;       +-----+-----+-----+-----+-----+-----+-----+-----+                 ;
;       | 2.4 | 2.3 | 2.2 | 2.1 | 1.4 | 2.3 | 1.2 | 1.1 |                 ;
;       +-----+-----+-----+-----+-----+-----+-----+-----+                 ;
;                                                                         ;
; If Read_Btns is called, the result will be the same as Read_Btns_Mask   ;
; with a mask of $FF, and a 1 will only be returned if the button         ;
; has transitioned to being pressed.                                      ;
;                                                                         ;
; If Read_Btns_Mask is called, then a mask, passed in in the A-reg        ;
; will be used to determine how the button state info is returned:        ;
;                                                                         ;
; If a bit is 0, then the current state of the button is to be returned   ;
; in the appropriate RAM location; 0 = not pressed, and 1 = pressed.      ;
;                                                                         ;
; If a bit is 1, then the appropriate RAM location is set to 1 only       ;
; on the depression transition of the button; additional calls will       ;
; return 0, until the button is released and then depressed again.        ;
;                                                                         ;
; ENTRY DP = $D0                                                          ;
;       A-reg = mask (for Read_Btns_Mask only)                            ;
;                                                                         ;
; Exit: A-reg = button transition state (same as $C811)                   ;
;                                                                         ;
;       B-reg, X-reg trashed                                              ;
;-------------------------------------------------------------------------;

Read_Btns_Mask: ANDA    Vec_Btn_State   ;Mask out "always" buttons
```

```
                STA     Vec_Btn_State
Read_Btns:      LDX     #Vec_Button_1_1 ;Point to button flags
                LDA     -3,X            ;Save previous state
                STA     -2,X
                LDA     #$0E            ;Sound chip register 0E to port A
                STA     <VIA_port_a
                LDD     #$1901          ;sound BDIR on, BC1 on, mux off
                STA     <VIA_port_b
                NOP                     ;pause
                STB     <VIA_port_b     ;sound BDIR off, BC1 off, mux off
                CLR     <VIA_DDR_a      ;DDR A to input
                LDD     #$0901          ;sound BDIR off, BC1 on, mux off
                STA     <VIA_port_b
                NOP                     ;pause
                LDA     <VIA_port_a     ;Read buttons
                COMA                    ;Convert to active high
                STA     -3,X            ;Save buttons
                STB     <VIA_port_b     ;sound BDIR off, BC1 off, mux off
                LDB     #$FF
                STB     <VIA_DDR_a      ;DDR A to output
                COMA                    ;Check for transitions
                ORA     -2,X
                COMA
                STA     -1,X            ;Store transition result
                PSHS    A               ;Save result for return value
                LDB     #$01            ;Initialize bit position
LF1EA:          TFR     B,A             ;Mask out bit
                ANDA    ,S
                STA     ,X+             ;Store masked bit
                ASLB                    ;Go back for next bit
                BNE     LF1EA
                PULS    A,PC            ;Get back transition bits and return

;-----------------------------------------------------------------------;
;       F1F5    Joy_Analog                                              ;
;       F1F8    Joy_Digital                                             ;
;                                                                       ;
; These routines read the current positions of the two joysticks.       ;
;                                                                       ;
; The joystick enable flags (C81F-C822) must be initialized to one of   ;
; the following values:                                                 ;
;                                                                       ;
;       0 - ignore; return no value.                                    ;
;       1 - return state of console 1 left/right position.              ;
;       3 - return state of console 1 up/down position.                 ;
;       5 - return state of console 2 left/right position.              ;
;       7 - return state of console 2 up/down position.                 ;
;                                                                       ;
; The joystick values are returned in $C81B-$C81E, where the value      ;
; returned in $C81B corresponds to the mask set in in $C81F, and so     ;
; on and so forth.                                                      ;
;                                                                       ;
; The joystick conversion is dependent on which routine is called.      ;
; Results for each routine are:                                         ;
```

```
;                                                                       ;
;       Joy_Digital:                                                    ;
;               The return value will be:                               ;
;                 < 0 if joystick is left of down of center.            ;
;                 = 0 if joystick is centered.                          ;
;                 > 0 if joystick is right or up of center.             ;
;                                                                       ;
;       Joy_Analog:                                                     ;
;               A successive approximation algorithm is used to read    ;
;               the actual value of the joystick pot, a signed value.   ;
;               In this case, $C81A must be set to a power of 2, to     ;
;               to control conversion resolution; 0x80 is least         ;
;               accurate, and 0x00is most accurate.                     ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;                                                                       ;
;       D-reg, X-reg trashed                                            ;
;-----------------------------------------------------------------------;

Joy_Analog:     DEC     $C823           ;Set analog mode flag
Joy_Digital:    LDX     #Vec_Joy_Mux_1_X;Point to first pot
LF1FB:          LDA     ,X+             ;Read it if enabled
                BNE     LF20B
LF1FF:          CMPX    #$C823          ;Go back untl all pots read
                BNE     LF1FB
                CLR     ,X              ;X points to $C823, clear it
                LDA     #$01
                STA     <VIA_port_b     ;disable mux
                RTS

LF20B:          STA     <VIA_port_b     ;enable mux and select pot
                CLR     <VIA_port_a     ;output $00 to D/A
                DEC     <VIA_port_b     ;disable mux
                LDB     #$60            ;delay and end up with B=$80
LF213:          INCB
                BPL     LF213
                LDA     $C823           ;check analog flag
                BMI     LF240           ;branch if analog pot
                LDA     #$20
                INC     <VIA_port_b     ;enable mux
                BITA    <VIA_port_b     ;test comparator
                BEQ     LF22D
                LDB     #$40            ;output $40 to D/A
                STB     <VIA_port_a
                BITA    <VIA_port_b     ;test comparator
                BNE     LF236
                BRA     LF235

LF22D:          LDB     #$C0            ;output $C0 to D/A
                STB     <VIA_port_a
                BITA    <VIA_port_b     ;test comparator
                BEQ     LF236
LF235:          CLRB
LF236:          STB     -5,X            ;store A/D result
```

```
                BRA     LF1FF           ;go back for next pot

LF23A:          TFR     B,A
                ORA     <VIA_port_a
                STA     <VIA_port_a
LF240:          LDA     #$20            ;test comparator
                BITA    <VIA_port_b
                BNE     LF24C           ;branch to go lower?
                TFR     B,A
                EORA    <VIA_port_a
                STA     <VIA_port_a
LF24C:          LSRB                    ;try next bit position
                CMPB    Vec_Joy_Resltn  ;check for accuracy threshold
                BNE     LF23A           ;go back if not finished
                LDB     <VIA_port_a     ;read D/A value
                BRA     LF236           ;go back to store it

;-----------------------------------------------------------------------;
;       F256    Sound_Byte                                              ;
;       F259    Sound_Byte_x                                            ;
;       F25B    Sound_Byte_raw                                          ;
;                                                                       ;
; All of these routines cause a byte of music data to be written to     ;
; the music chip.  Sound_Byte stores a shadow copy of the data into     ;
; $C800-$C80E, and Sound_Byte_x stores a shadow copy into a 15 byte     ;
; area pointed to by the X register.  Sound_Byte_raw does not store a   ;
; shadow copy of the data at all.                                       ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       A-reg = which of the 15 sound chip registers to modify          ;
;       B-reg = the byte of sound data                                  ;
;       X-reg = 15 byte shadow area (Sound_Byte_x only)                 ;
;                                                                       ;
; EXIT: X-reg = $C800 (Sound_Byte only)                                 ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Sound_Byte:     LDX     #Vec_Snd_Shadow ;point to shadow memory
Sound_Byte_x:   STB     A,X
Sound_Byte_raw: STA     <VIA_port_a     ;store register select byte
                LDA     #$19            ;sound BDIR on, BC1 on, mux off
                STA     <VIA_port_b
                LDA     #$01            ;sound BDIR off, BC1 off, mux off
                STA     <VIA_port_b
                LDA     <VIA_port_a     ;read sound chip status (?)
                STB     <VIA_port_a     ;store data byte
                LDB     #$11            ;sound BDIR on, BC1 off, mux off
                STB     <VIA_port_b
                LDB     #$01            ;sound BDIR off, BC1 off, mux off
                STB     <VIA_port_b
                RTS

;-----------------------------------------------------------------------;
```

```
;       F272    Clear_Sound                                             ;
;                                                                       ;
; This routine clears the 15 registers on the music chip and the soft   ;
; copy of their values (C800-C80E), by writing a byte of 0 to each      ;
; register.  This causes the sound chip to not make any sounds.         ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;                                                                       ;
;       D-reg, X-reg trashed                                            ;
;-----------------------------------------------------------------------;

Clear_Sound:    LDD     #$0E00
LF275:          BSR     Sound_Byte
                DECA
                BPL     LF275
                JMP     Init_Music_Buf

;-----------------------------------------------------------------------;
;       F27D    Sound_Bytes                                             ;
;       F284    Sound_Bytes_x? (apparently never used)                  ;
;                                                                       ;
; This routine copies a block of sound information into the sound       ;
; chip buffer (at $C800-$C80E) and into the registers on the music      ;
; chip.  The format for the block of sound data is as follows:          ;
;                                                                       ;
;   (register number), (music data),                                    ;
;   (register number), (music data),                                    ;
;         .                 .                                         ;
;         .                 .                                         ;
;       0xFF                                                            ;
;                                                                       ;
; As long as the register number is >= 0, then the music data will be   ;
; copied; however, as soon as a register number < 0 is encountered,     ;
; the copy will stop.                                                   ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       U-reg = pointer to the block of sound data                      ;
;                                                                       ;
;       D-reg, X-reg, U-reg trashed                                     ;
;-----------------------------------------------------------------------;

Sound_Bytes:    LDX     #Vec_Snd_Shadow ;Point to shadow memory
                BRA     Sound_Bytes_x

LF282:          BSR     Sound_Byte_x    ;Update the sound register
Sound_Bytes_x:  LDD     ,U++            ;Get next next pair of bytes
                BPL     LF282           ;Go back if not end of list
                RTS

;-----------------------------------------------------------------------;
;       F289    Do_Sound                                                ;
;       F28C    Do_Sound_x? (apparently never used)                     ;
;                                                                       ;
; This routine will start/continue making the sound which was first     ;
```

```
; set up by your call to Init_Music.  This routine should normally      ;
; be called right after your call to Wait_Recal.  It takes the next     ;
; music information, contained in the music buffer $C83F-$C84C, and     ;
; updates only those registers which differ from the last data written  ;
; to the sound chip.                                                    ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;                                                                       ;
;       D-reg, X-reg, U-reg trashed                                     ;
;-----------------------------------------------------------------------;

Do_Sound:       LDX     #Vec_Snd_Shadow ;point to shadow memory
Do_Sound_x:     LDU     #Vec_Music_Work ;point to sound buffer
                LDA     #$0D            ;init count for 14 registers
LF291:          LDB     ,U+             ;get next register
                CMPB    A,X             ;skip if unchanged
                BEQ     LF299
                BSR     Sound_Byte_x    ;else update register
LF299:          DECA                    ;go back for next register
                BPL     LF291
                RTS

;-----------------------------------------------------------------------;
;       F29D    Intensity_1F                                            ;
;       F2A1    Intensity_3F                                            ;
;       F2A5    Intensity_5F                                            ;
;       F2A9    Intensity_7F                                            ;
;       F2AB    Intensity_a                                             ;
;                                                                       ;
; Each of these routines are responsible for setting the vector/dot     ;
; intensity (commonly used to denote the z axis) to a specific value.   ;
; 0x00 is the lowest intensity, and 0xFF is the brightest intensity.    ;
; The intensity must be reset to the desired value after each call      ;
; to Wait_Recal; however, it can also be changed at any other time.     ;
; A copy of the new intensity value is saved in $C827.                  ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       A-reg = intensity (Intensity_a only)                            ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Intensity_1F:   LDA     #$1F
                BRA     Intensity_a

Intensity_3F:   LDA     #$3F
                BRA     Intensity_a

Intensity_5F:   LDA     #$5F
                BRA     Intensity_a

Intensity_7F:   LDA     #$7F
Intensity_a:    STA     <VIA_port_a     ;Store intensity in D/A
                STA     Vec_Brightness  ;Save intensity in $C827
```

```
                LDD     #$0504          ;mux disabled channel 2
                STA     <VIA_port_b
                STB     <VIA_port_b     ;mux enabled channel 2
                STB     <VIA_port_b     ;do it again just because
                LDB     #$01
                STB     <VIA_port_b     ;turn off mux
                RTS

;-----------------------------------------------------------------------;
;       F2BE    Dot_ix_b                                                ;
;       F2C1    Dot_ix                                                  ;
;                                                                       ;
; These routines draw a dot at the relative y and relative x            ;
; position pointed to by the X register.  Afterwards, the X register    ;
; is incremented by 2.                                                  ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       X-reg points to the (y,x) coordinate pair                       ;
;       B-reg contains the intensity (Dot_ix_b only)                    ;
;       $C828 contains the intensity (Dot_ix only)                      ;
;                                                                       ;
; EXIT  X-reg incremented by 2                                          ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Dot_ix_b:       STB     Vec_Dot_Dwell
Dot_ix:         LDD     ,X++

;-----------------------------------------------------------------------;
;       F2C3    Dot_d                                                   ;
;                                                                       ;
; This routine draws a dot at the relative y and relative x position    ;
; contained in the D register.  The intensity used is the value         ;
; already stored in $C828.                                              ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       A-reg = relative Y coordinate                                   ;
;       B-reg = relative X coordinate                                   ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Dot_d:          BSR     Moveto_d

;-----------------------------------------------------------------------;
;       F2C5    Dot_here                                                ;
;                                                                       ;
; This routine draws a dot at the current pen position.                 ;
; The intensity used is the value already stored in $C828.              ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;                                                                       ;
;       D-reg trashed                                                   ;
```

```
;-----------------------------------------------------------------------;

Dot_here:       LDA     #$FF            ;Set pattern to all 1's
                STA     <VIA_shift_reg  ;Store in VIA shift register
                LDB     Vec_Dot_Dwell   ;Get dot dwell (brightness)
LF2CC:          DECB                    ;Delay leaving beam in place
                BNE     LF2CC
                CLR     <VIA_shift_reg  ;Blank beam in VIA shift register
                RTS

;-----------------------------------------------------------------------;
;       F2D5    Dot_List                                                ;
;                                                                       ;
; This routine draws a series of dots, using the intensity already      ;
; set up in $C828.  The format for the dot list, which is pointed to    ;
; by the X register, is:                                                ;
;                                                                       ;
;     ( rel y, rel x), (rel y, rel x), .....                            ;
;                                                                       ;
; The number of dots to draw is specified in $C823.                     ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       X-reg points to the list of dot coordinates                     ;
;       $C823 specifies the number of dots to draw                      ;
;                                                                       ;
; EXIT: X-reg points to next byte after list                            ;
;       $C823 cleared                                                   ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

LF2D2:          DEC     $C823           ;Decrement counter
Dot_List:       BSR     Dot_ix          ;Draw next dot
                LDA     $C823           ;Check counter
                BNE     LF2D2           ;Go back until finished
                BRA     Reset0Ref       ;Go to Reset0Ref

;-----------------------------------------------------------------------;
;       F2DE    Dot_List_Reset                                          ;
;                                                                       ;
; This routine draws a series of dots, specified by the list pointed    ;
; to by the X register.  The list has the following format:             ;
;                                                                       ;
;     mode, relative y, relative x,                                     ;
;     mode, relative y, relative x,                                     ;
;       .      .           .                                           ;
;       .      .           .                                           ;
;     mode, relative y, relative x                                      ;
;     0x01                                                              ;
;                                                                       ;
; This routine will continue to traverse the list, until a mode > 0     ;
; is encountered; at that point, it will reset the zero reference       ;
; (the integrators).                                                    ;
;                                                                       ;
```

```
; ENTRY DP = $D0                                                    ;
;       X-reg points to the dot list                                ;
;                                                                   ;
; EXIT: X-reg points to next byte after the terminator              ;
;                                                                   ;
;       D-reg trashed                                               ;
;-------------------------------------------------------------------;

Dot_List_Reset: LDA     ,X+             ;get mode byte
                BGT     Reset0Ref       ;if >0 go to Reset0Ref
                BSR     Dot_ix          ;plot the dot
                BRA     Dot_List_Reset  ;dot_list@x_&_reset

;-------------------------------------------------------------------;
;       F2E6    Recalibrate                                         ;
;                                                                   ;
;       Recalibrate the vector generators.                          ;
;                                                                   ;
; ENTRY DP = $D0                                                    ;
;                                                                   ;
;       D-reg, X-reg trashed                                        ;
;-------------------------------------------------------------------;

Recalibrate:    LDX     #Recal_Points   ;$7F7F
                BSR     Moveto_ix_FF
                JSR     Reset0Int
                BSR     Moveto_ix       ;$8080
                BRA     Reset0Ref

;-------------------------------------------------------------------;
;       F2F2    Moveto_x_7F                                         ;
;                                                                   ;
; This routine forces the scale factor to 0x7F, and then moves the  ;
; pen to the location pointed to by the X register.  The relative y ;
; and relative x coordinates are both 2 byte quantities; however,   ;
; only the most signicant byte of each is of any interest.  The values ;
; pointed to by the X register have the following format:           ;
;                                                                   ;
;     X => (rel y hi),(rel y lo), (rel x hi), (rel x lo)            ;
;                                                                   ;
; The position moved to is obtained by y=(0,x) & x=(2,x).           ;
;                                                                   ;
; ENTRY DP = $D0                                                    ;
;       X-reg points to double-sized coordinate pair                ;
;                                                                   ;
;       D-reg trashed                                               ;
;-------------------------------------------------------------------;

Moveto_x_7F:    LDB     #$7F            ;Set scale factor to $7F
                STB     <VIA_t1_cnt_lo
                LDA     ,X              ;Get y high
                LDB     2,X             ;Get x high
                BRA     Moveto_d
```

```
;-----------------------------------------------------------------------;
;       F2FC    Moveto_d_7F                                             ;
;                                                                       ;
; This routine forces the scale factor to 0x7F, and then moves the      ;
; pen to the position specified in the D register.                      ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       A-reg = relative Y coordinate                                   ;
;       B-reg = relative X coordinate                                   ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Moveto_d_7F:    STA     <VIA_port_a     ;Store Y in D/A register
                PSHS    D               ;Save D-register on stack
                LDA     #$7F            ;Set scale factor to $7F
                STA     <VIA_t1_cnt_lo
                CLR     <VIA_port_b     ;Enable mux
                BRA     LF318

;-----------------------------------------------------------------------;
;       F308    Moveto_ix_FF                                            ;
;       F30C    Moveto_ix_7F                                            ;
;       F30E    Moveto_ix_b                                             ;
;                                                                       ;
; These routines force the scale factor to 0xFF, 0X7F, or the           ;
; A register, and then move the pen to the (y,x) position pointed to    ;
; by the X-register.  The X-register is then incremented by 2.          ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       X-reg points to the (y,x) coordinate pair                       ;
;       B-reg contains the scale factor (Moveto_ix_b only)              ;
;                                                                       ;
; EXIT: X-reg has been incremented by 2                                 ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Moveto_ix_FF:   LDB     #$FF
                BRA     Moveto_ix_b

Moveto_ix_7F:   LDB     #$7F
Moveto_ix_b:    STB     <VIA_t1_cnt_lo  ;Set scale factor

;-----------------------------------------------------------------------;
;       F310    Moveto_ix                                               ;
;                                                                       ;
; This routine uses the current scale factor, and moves the pen to the  ;
; (y,x) position pointed to by the X register.  The X register is then  ;
; incremented by 2.                                                     ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       X-reg points to the (y,x) coordinate pair                       ;
;                                                                       ;
```

```
; EXIT: X-reg has been incremented by 2                                ;
;                                                                      ;
;       D-reg trashed                                                  ;
;----------------------------------------------------------------------;

Moveto_ix:      LDD     ,X++

;----------------------------------------------------------------------;
;       F312    Moveto_d                                               ;
;                                                                      ;
; This routine uses the current scale factor, and moves the pen to the ;
; (y,x) position specified in D register.                              ;
;                                                                      ;
; ENTRY DP = $D0                                                       ;
;       A-reg = Y coordinate                                           ;
;       B-reg = X coordinate                                           ;
;                                                                      ;
;       D-reg trashed                                                  ;
;----------------------------------------------------------------------;

Moveto_d:       STA     <VIA_port_a     ;Store Y in D/A register
                CLR     <VIA_port_b     ;Enable mux
                PSHS    D               ;Save D-register on stack
LF318:          LDA     #$CE            ;Blank low, zero high?
                STA     <VIA_cntl
                CLR     <VIA_shift_reg  ;Clear shift regigster
                INC     <VIA_port_b     ;Disable mux
                STB     <VIA_port_a     ;Store X in D/A register
                CLR     <VIA_t1_cnt_hi  ;timer 1 count high
                PULS    D               ;Get back D-reg
                JSR     Abs_a_b
                STB     -1,S
                ORA     -1,S
                LDB     #$40
                CMPA    #$40
                BRA     LF345
                CMPA    #$64
                BLS     LF33B
                LDA     #$08
                BRA     LF33D

LF33B:          LDA     #$04            ;Wait for timer 1
LF33D:          BITB    <VIA_int_flags
                BEQ     LF33D
LF341:          DECA                    ;Delay a moment
                BNE     LF341
                RTS

LF345:          BITB    <VIA_int_flags  ;Wait for timer 1
                BEQ     LF345
                RTS

;----------------------------------------------------------------------;
;       F34A    Reset0Ref_D0                                           ;
```

```
;                                                                       ;
; This routine sets the DP register to D0, and then resets the          ;
; integrators.                                                          ;
;                                                                       ;
; EXIT: DP = $D0                                                        ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Reset0Ref_D0:   JSR     DP_to_D0
                BRA     Reset0Ref

;-----------------------------------------------------------------------;
;       F34F    Check0Ref                                               ;
;                                                                       ;
; This routine will check to see if the Reset0Ref enable flag ($C824)   ;
; is set, and if it is, then it will reset the integrators by calling   ;
; Reset0Ref.                                                            ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       $C824 = enable flag                                             ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Check0Ref:      LDA     Vec_0Ref_Enable
                BEQ     LF36A_RTS

;-----------------------------------------------------------------------;
;       F354    Reset0Ref                                               ;
;                                                                       ;
; This routine zeros the integrators, and resets the pen back to the    ;
; origin.  It leaves the integrators in zero mode, so nothing can be    ;
; drawn until a move is done, or $D00C is set to 0xCE to bring /ZERO    ;
; high.  This routine must be called every so often, to prevent your    ;
; vectors from getting out of whack.                                    ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Reset0Ref:      LDD     #$00CC
                STB     <VIA_cntl       ;/BLANK low and /ZERO low
                STA     <VIA_shift_reg  ;clear shift register
;-----------------------------------------------------------------------;
;       F35B    Reset_Pen                                               ;
;                                                                       ;
;       Reset the pen to the origin.                                    ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;
```

```
Reset_Pen:      LDD     #$0302
                CLR     <VIA_port_a     ;clear D/A register
                STA     <VIA_port_b     ;mux=1, disable mux
                STB     <VIA_port_b     ;mux=1, enable mux
                STB     <VIA_port_b     ;do it again
                LDB     #$01
                STB     <VIA_port_b     ;disable mux
LF36A_RTS:      RTS

;-----------------------------------------------------------------------;
;       F36B    Reset0Int                                               ;
;                                                                       ;
; This routine resets the integrators to zero.  It leaves the           ;
; integrators in zero mode, so nothing can be drawn until a move is     ;
; done, or D00C is set to 0xCE.                                         ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Reset0Int:      LDD     #$00CC
                STB     <VIA_cntl       ;blank low and zero low
                STA     <VIA_shift_reg  ;clear shift register
                RTS

;-----------------------------------------------------------------------;
;       F373    Print_Str_hwyx                                          ;
;                                                                       ;
; This routine prints a single string (up to an 0x80).  The parameter   ;
; block describing the string is pointed to by the U register.  The     ;
; format for the parameter block is as follows:                         ;
;                                                                       ;
;       height, width, rel y, rel x, string, 0x80                       ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       U-reg points to the string list                                 ;
;                                                                       ;
; EXIT: U-reg points to the byte after the terminating 0x80             ;
;                                                                       ;
;       D-reg, X-reg trashed                                            ;
;-----------------------------------------------------------------------;

Print_Str_hwyx: LDD     ,U++
                STD     Vec_Text_HW

;-----------------------------------------------------------------------;
;       F378    Print_Str_yx                                            ;
;                                                                       ;
; This routine prints a single string (up to an 0x80), using the       ;
; default height and width, as stored in $C82A.  The parameter block    ;
; describing the string is pointed to by the U register.  The format    ;
; for the parameter block is as follows:                                ;
```

```
;                                                                       ;
;       rel y, rel x, string, 0x80                                      ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       U-reg points to the string list                                 ;
;                                                                       ;
; EXIT: U-reg points to the byte after the terminating 0x80             ;
;                                                                       ;
;       D-reg, X-reg trashed                                            ;
;-----------------------------------------------------------------------;

Print_Str_yx:   LDD     ,U++

;-----------------------------------------------------------------------;
;       F37A    Print_Str_d                                             ;
;                                                                       ;
; This routine prints a single string (up to an 0x80), using the       ;
; default height and width, as stored in $C82A, and at the pen position ;
; specified in the D register.  The parameter block describing the      ;
; string is pointed to by the U register.  The format for the           ;
; parameter block is as follows:                                        ;
;                                                                       ;
;     string, 0x80                                                      ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       U-reg points to string list                                     ;
;       A-reg = relative Y position                                     ;
;       B-reg = relative X position                                     ;
;                                                                       ;
; EXIT: U-reg points to the byte after the terminating 0x80             ;
;                                                                       ;
;       D-reg, X-reg trashed                                            ;
;-----------------------------------------------------------------------;
   noopt        ; neccessary for assembling, this jsr is allways
                ; optimized to a short branch otherwise
Print_Str_d:    JSR     >Moveto_d_7F
   opt
                JSR     Delay_1
                JMP     Print_Str

;-----------------------------------------------------------------------;
;       F385    Print_List_hw                                           ;
;                                                                       ;
; This displays the group of strings described by the parameter block   ;
; which is pointed to by the U register.  The string parameter block    ;
; has the following format:                                             ;
;                                                                       ;
;     height, width, rel y, rel x, string, 0x80,                        ;
;     height, width, rel y, rel x, string, 0x80,                        ;
;     0x00                                                              ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       U-reg points to string list                                     ;
;                                                                       ;
```

```
; EXIT: U-reg points to null terminator byte                          ;
;                                                                     ;
;       D-reg, X-reg trashed                                          ;
;---------------------------------------------------------------------;

LF383:          BSR     Print_Str_hwyx
Print_List_hw:  LDA     ,U
                BNE     LF383
                RTS

;---------------------------------------------------------------------;
;       F38A    Print_List                                            ;
;       F38C    Print_List_chk                                        ;
;                                                                     ;
; This displays the group of strings described by the parameter block ;
; which is pointed to by the U register.  The string parameter block  ;
; has the following format:                                           ;
;                                                                     ;
;       rel y, rel x, string, 0x80,                                   ;
;       rel y, rel x, string, 0x80,                                   ;
;       0x00                                                          ;
;                                                                     ;
; The current string height and width to which the hardware is set will ;
; be used.                                                            ;
;                                                                     ;
; Print_List routine will first print the passed-in string, and THEN  ;
; check for the end of the string list.  Print_List_Chk will check for ;
; the end of the string list first.                                   ;
;                                                                     ;
; ENTRY DP = $D0                                                      ;
;       U-reg points to string list                                   ;
;                                                                     ;
; EXIT: U-reg points to null terminator byte                          ;
;                                                                     ;
;       D-reg, X-reg trashed                                          ;
;---------------------------------------------------------------------;

Print_List:     BSR     Print_Str_yx
Print_List_Chk: LDA     ,U
                BNE     Print_List
                RTS

;---------------------------------------------------------------------;
;       F391    Print_Ships_x                                         ;
;       F393    Print_Ships                                           ;
;                                                                     ;
; This routine displays the number of ships passed in the B register  ;
; followed by a minus sign and the ship icon character passed in the  ;
; A register at the (y,x) coordinates passed in the X register.  If   ;
; the B-register > 9, then the infinity symbol is displayed.          ;
;                                                                     ;
; Note: This routine uses bytes at a negative offset from the stack as ;
;       temporary storage, so hopefully an IRQ won't happen until the ;
;       string is finished bring printed!                             ;
```

```
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       A-reg = ship icon character                                     ;
;       B-reg = number of ships                                         ;
;       X-reg = (y,x) coordinates (Print_Ships only)                    ;
;       X-reg points to (y,x) coordinates (Print_Ships_x only)          ;
;                                                                       ;
;       D-reg, X-reg, U-reg trashed                                     ;
;-----------------------------------------------------------------------;

Print_Ships_x:  LDX     ,X
Print_Ships:    PSHS    B               ;Save B-reg
                LDB     #$80
                LEAU    -8,S            ;Point U into the stack
                PSHU    D               ;Save A-reg and a terminator
                PULS    A               ;Get back B-reg
                CMPA    #$09            ;If B-reg >9 then
                BLS     LF3A3
                LDA     #$6C-$30        ;load $6C = infinty symbol
LF3A3:          ADDA    #$30
                LDB     #'-'
                PSHU    D               ;Push digit and minus sign
                PSHU    X               ;Push (y,x) coordinates
                BRA     Print_Str_yx    ;Print it

;-----------------------------------------------------------------------;
;       F3AD    Mov_Draw_VLc_a                                          ;
;                                                                       ;
; This routine moves to the first location specified in vector list,    ;
; and then draws lines between the rest of coordinates in the list.     ;
; The number of vectors to draw is specified as the first byte in the   ;
; vector list.  The current scale factor is used.  The vector list has  ;
; the following format:                                                 ;
;                                                                       ;
;       count, rel y, rel x, rel y, rel x, ...                          ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       X-reg points to the vector list                                 ;
;                                                                       ;
; EXIT: X-reg points to next byte after list                            ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Mov_Draw_VLc_a: LDA     ,X+
                BRA     Mov_Draw_VL_a

;-----------------------------------------------------------------------;
;       F3B1    Mov_Draw_VL_b                                           ;
;                                                                       ;
; This routine moves to the first location specified in vector list,    ;
; and then draws lines between the rest of coordinates in the list.     ;
; The vector list has the following format:                             ;
;                                                                       ;
```

```
;       rel y, rel x, rel y, rel x, ...                                 ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       B-reg = scale factor                                            ;
;       $C823 = number of vectors to draw                               ;
;       X-reg points to the vector list                                 ;
;                                                                       ;
; EXIT: $C823 is cleared                                                ;
;                                                                       ;
; EXIT: X-reg points to next byte after list                            ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Mov_Draw_VL_b:  STB     <VIA_t1_cnt_lo  ;Set scale factor
                BRA     Mov_Draw_VL

;-----------------------------------------------------------------------;
;       F3B5    Mov_Draw_VLcs                                           ;
;                                                                       ;
; This routine moves to the first location specified in vector list,    ;
; and then draws lines between the rest of coordinates in the list.     ;
; The number of vectors to draw is specified as the first byte in the   ;
; vector list, and the scale factor is the second byte in the vector    ;
; list.  The vector list has the following format:                      ;
;                                                                       ;
;       count, scale, rel y, rel x, rel y, rel x, ...                   ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       X-reg points to the vector list                                 ;
;                                                                       ;
; EXIT: X-reg points to next byte after list                            ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Mov_Draw_VLcs:  LDD     ,X++

;-----------------------------------------------------------------------;
;       F3B7    Mov_Draw_VL_ab                                          ;
;       F3B9    Mov_Draw_VL_a                                           ;
;                                                                       ;
; This routine moves to the first location specified in vector list,    ;
; and then draws lines between the rest of coordinates in the list.     ;
; The vector list has the  following format:                            ;
;                                                                       ;
;       rel y, rel x, rel y, rel x, ...                                 ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       A-reg = number of vectors to draw                               ;
;       B-reg = scale factor to use (Draw_VL_ab only)                   ;
;       X-reg points to the vector list                                 ;
;                                                                       ;
; EXIT: X-reg points to next byte after list                            ;
```

```
;                                                                      ;
;       D-reg trashed                                                  ;
;----------------------------------------------------------------------;

Mov_Draw_VL_ab: STB     <VIA_t1_cnt_lo  ;Set scale factor
Mov_Draw_VL_a:  STA     $C823

;----------------------------------------------------------------------;
;       F3BC    Mov_Draw_VL                                            ;
;       F3BE    Mov_Draw_VL_d                                          ;
;                                                                      ;
; This routine moves to the first location specified in vector list,   ;
; and then draws lines between the rest of coordinates in the list.    ;
; The vector list has the following format:                            ;
;                                                                      ;
;       rel y, rel x, rel y, rel x, ...                                ;
;                                                                      ;
; Draw_VL_d starts at the (y,x) coordinates specified in the D register ;
; and ignores the first pair of coordinates in the vector list.        ;
;                                                                      ;
; ENTRY DP = $D0                                                       ;
;       $C823 = number of vectors to draw                              ;
;       D-reg = start coordinate (Draw_VL_d only)                      ;
;       X-reg points to the vector list (2,X for Mov_Draw_VL_d)        ;
;                                                                      ;
; EXIT: $C823 is cleared                                               ;
;                                                                      ;
; EXIT: X-reg points to next byte after list                           ;
;                                                                      ;
;       D-reg trashed                                                  ;
;----------------------------------------------------------------------;

Mov_Draw_VL:    LDD     ,X              ;Get next coordinate pair
Mov_Draw_VL_d:  STA     <VIA_port_a     ;Send Y to A/D
                CLR     <VIA_port_b     ;Enable mux
                LEAX    2,X             ;Point to next coordinate pair
                NOP                     ;Wait a moment
                INC     <VIA_port_b     ;Disable mux
                STB     <VIA_port_a     ;Send X to A/D
                LDD     #$0000          ;Shift reg=0 (no draw), T1H=0
                BRA     LF3ED           ;A->D00A, B->D005

;----------------------------------------------------------------------;
;       F3CE    Draw_VLc                                               ;
;                                                                      ;
; This routine draws vectors between the set of (y,x) points pointed   ;
; to by the X register.  The number of vectors to draw is specified    ;
; as the first byte in the vector list.  The current scale factor is   ;
; used.  The vector list has the following format:                     ;
;                                                                      ;
;       count, rel y, rel x, rel y, rel x, ...                         ;
;                                                                      ;
; ENTRY DP = $D0                                                       ;
;       X-reg points to the vector list                                ;
```

```
;                                                                       ;
; EXIT: X-reg points to next byte after list                            ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Draw_VLc:       LDA     ,X+
                BRA     Draw_VL_a

;-----------------------------------------------------------------------;
;       F3D2    Draw_VL_b                                               ;
;                                                                       ;
; This routine draws vectors between the set of (y,x) points pointed to ;
; by the X register.  The vector list has the following format:         ;
;                                                                       ;
;       rel y, rel x, rel y, rel x, ...                                 ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       B-reg = the scale factor                                        ;
;       X-reg points to the vector list                                 ;
;                                                                       ;
; EXIT: X-reg points to next byte after list                            ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Draw_VL_b:      STB     <VIA_t1_cnt_lo  ;Set scale factor
                BRA     Draw_VL

;-----------------------------------------------------------------------;
;       F3D6    Draw_VLcs                                               ;
;                                                                       ;
; This routine draws vectors between the set of (y,x) points pointed    ;
; to by the X register.  The number of vectors to draw is specified     ;
; as the first byte in the vector list.  The scale factor is specified  ;
; as the second byte in the vector list. The vector list has the        ;
; following format:                                                     ;
;                                                                       ;
;       count, scale, rel y, rel x, rel y, rel x, ...                   ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       X-reg points to the vector list                                 ;
;                                                                       ;
; EXIT: X-reg points to next byte after list                            ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Draw_VLcs:      LDD     ,X++

;-----------------------------------------------------------------------;
;       F3D8    Draw_VL_ab                                              ;
;                                                                       ;
; This routine draws vectors between the set of (y,x) points pointed    ;
```

```
; to by the X register.  The vector list has the following format:      ;
;                                                                       ;
;       rel y, rel x, rel y, rel x, ...                                 ;
;                                                                       ;
; ENTRY DP = $D0                                                       ;
;       A-reg = the number of vectors to draw                           ;
;       B-reg = the scale factor                                        ;
;       X-reg points to the vector list                                 ;
;                                                                       ;
; EXIT: X-reg points to next byte after list                           ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Draw_VL_ab:     STB     <VIA_t1_cnt_lo

;-----------------------------------------------------------------------;
;       F3DA    Draw_VL_a                                               ;
;                                                                       ;
; This routine draws vectors between the set of (y,x) points pointed    ;
; to by the register.  The current scale factor is used.  The vector    ;
; list has the following format:                                        ;
;                                                                       ;
;       rel y, rel x, rel y, rel x, ...                                 ;
;                                                                       ;
; ENTRY DP = $D0                                                       ;
;       A-reg = the number of vectors to draw                           ;
;       X-reg points to the vector list                                 ;
;                                                                       ;
; EXIT: X-reg points to next byte after list                           ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Draw_VL_a:      STA     $C823

;-----------------------------------------------------------------------;
;       F3DD    Draw_VL                                                 ;
;                                                                       ;
; This routine draws vectors between the set of (y,x) points pointed    ;
; to by the X register.  The number of vectors to draw must already be  ;
; specified in $C823.  The current scale factor is used.  The vector    ;
; list has the following format:                                        ;
;                                                                       ;
;       rel y, rel x, rel y, rel x, ...                                 ;
;                                                                       ;
; ENTRY DP = $D0                                                       ;
;       X-reg points to the vector list                                 ;
;                                                                       ;
; EXIT: X-reg points to next byte after list                           ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;
```

```
Draw_VL:        LDD     ,X

;-----------------------------------------------------------------------;
;       F3DF    Draw_Line_d                                             ;
;                                                                       ;
; This routine will draw a line from the current pen position, to the   ;
; point specified by the (y,x) pair specified in the D register.  The   ;
; current scale factor is used.  Before calling this routine, $C823     ;
; should be = 0, so that only the one vector will be drawn.             ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       A-reg = relative y position                                     ;
;       B-reg = relative x position                                     ;
;                                                                       ;
; EXIT: X-reg is incremented by 2                                       ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Draw_Line_d     STA     <VIA_port_a     ;Send Y to A/D
                CLR     <VIA_port_b     ;Enable mux
                LEAX    2,X             ;Point to next coordinate pair
                NOP                     ;Wait a moment
                INC     <VIA_port_b     ;Disable mux
                STB     <VIA_port_a     ;Send X to A/D
                LDD     #$FF00          ;Shift reg=$FF (solid line), T1H=0
LF3ED:          STA     <VIA_shift_reg  ;Put pattern in shift register
                STB     <VIA_t1_cnt_hi  ;Set T1H (scale factor?)
                LDD     #$0040          ;B-reg = T1 interrupt bit
LF3F4:          BITB    <VIA_int_flags  ;Wait for T1 to time out
                BEQ     LF3F4
                NOP                     ;Wait a moment more
                STA     <VIA_shift_reg  ;Clear shift register (blank output)
                LDA     $C823           ;Decrement line count
                DECA
                BPL     Draw_VL_a       ;Go back for more points
                JMP     Check0Ref       ;Reset zero reference if necessary

;-----------------------------------------------------------------------;
;       F404    Draw_VLp_FF                                             ;
;       F408    Draw_VLp_7F                                             ;
;                                                                       ;
; These routines force the scale factor to 0xFF or 0x7F, and then       ;
; process the vector list pointed to by the X register.  The vector     ;
; list has the following format:                                        ;
;                                                                       ;
;       pattern, rel y, rel x                                           ;
;       pattern, rel y, rel x                                           ;
;          .      .      .                                             ;
;          .      .      .                                             ;
;       pattern, rel y, rel x                                           ;
;       0x01                                                            ;
;                                                                       ;
; The list is terminated by a pattern byte with the high bit cleared.   ;
```

```
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       X-reg points to the vector list                                 ;
;                                                                       ;
; EXIT: X-reg points to the terminator byte                             ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Draw_VLp_FF:    LDB     #$FF
                BRA     Draw_VLp_b

Draw_VLp_7F:    LDB     #$7F
                BRA     Draw_VLp_b

;-----------------------------------------------------------------------;
;       F40C    Draw_VLp_scale                                          ;
;                                                                       ;
; This routine processes the vector list pointed to by the X register.  ;
; The first byte in the vector list is the scale factor.  The vector    ;
; list has the following format:                                        ;
;                                                                       ;
;       scale                                                           ;
;       pattern, rel y, rel x                                           ;
;       pattern, rel y, rel x                                           ;
;          .      .      .                                             ;
;          .      .      .                                             ;
;       pattern, rel y, rel x                                           ;
;       0x01                                                            ;
;                                                                       ;
; The list is terminated by a pattern byte with the high bit cleared.   ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       X-reg points to the vector list                                 ;
;                                                                       ;
; EXIT: X-reg points to the terminator byte                             ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Draw_VLp_scale: LDB     ,X+

;-----------------------------------------------------------------------;
;       F40E    Draw_VLp_b                                              ;
;                                                                       ;
; This routine draws patterned lines using the vector list pointed to   ;
; by the X register.  The vector list has the following format:         ;
;                                                                       ;
;       pattern, rel y, rel x                                           ;
;       pattern, rel y, rel x                                           ;
;          .      .      .                                             ;
;          .      .      .                                             ;
;       pattern, rel y, rel x                                           ;
;       0x01                                                            ;
```

```
;                                                                       ;
; The list is terminated by a pattern byte with the high bit cleared.   ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       B-reg = the scale factor                                        ;
;       X-reg points to the vector list                                 ;
;                                                                       ;
; EXIT: X-reg points to the terminator byte                             ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Draw_VLp_b:     STB     <VIA_t1_cnt_lo  ;Set scale factor

;-----------------------------------------------------------------------;
;       F410    Draw_VLp                                                ;
;                                                                       ;
; This routine draws patterned lines using the vector list pointed to   ;
; by the X-register.  The current scale factor is used.  The vector     ;
; list has the following format:                                        ;
;                                                                       ;
;       pattern, rel y, rel x                                           ;
;       pattern, rel y, rel x                                           ;
;          .      .      .                                             ;
;          .      .      .                                             ;
;       pattern, rel y, rel x                                           ;
;       0x01                                                            ;
;                                                                       ;
; The list is terminated by a pattern byte with the high bit cleared.   ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       X-reg points to the vector list                                 ;
;                                                                       ;
; EXIT: X-reg points to the terminator byte                             ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Draw_VLp:       LDD     1,X             ;Get next coordinate pair
                STA     <VIA_port_a     ;Send Y to A/D
                CLR     <VIA_port_b     ;Enable mux
                LDA     ,X              ;Get pattern byte?
                LEAX    3,X             ;Advance to next point in list
                INC     <VIA_port_b     ;Disable mux
                STB     <VIA_port_a     ;Send X to A/D
                STA     <VIA_shift_reg  ;Store pattern in shift register
                CLR     <VIA_t1_cnt_hi  ;Clear T1H
                LDD     #$0040          ;B-reg = T1 interrupt bit
LF425:          BITB    <VIA_int_flags  ;Wait for T1 to time out
                BEQ     LF425
                NOP                     ;Wait a moment more
                STA     <VIA_shift_reg  ;Clear shift register (blank output)
                LDA     ,X              ;Get next pattern byte
                BLE     Draw_VLp        ;Go back if high bit of pattern is set
```

```
                JMP     Check0Ref

;-----------------------------------------------------------------------;
;       F434    Draw_Pat_VL_a                                           ;
;       F437    Draw_Pat_VL                                             ;
;       F439    Draw_Pat_VL_d                                           ;
;                                                                       ;
; All of these routines draw a series of patterned vectors.  The        ;
; pattern to use must already be specified in $C829.  When using        ;
; Draw_Pat_VL or Draw_Pat_VL_d, the number of vectors to draw minus 1   ;
; must be specified in $C823; when using Draw_Pat_VL_a, the number of   ;
; vectors to draw minus 1 must be passed in in the A register.          ;
; The vector list, pointed to by the X register, has the following      ;
; format:                                                               ;
;                                                                       ;
;       rel y, rel x, rel y, rel x, ...                                 ;
;                                                                       ;
; Draw_Pat_VL_d starts at the (y,x) coordinates specified in the        ;
; D register and ignores the first pair of coordinates in the vector    ;
; list.                                                                 ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       X-reg points to the vector list                                 ;
;       A-reg = the number of vectors to draw (Draw_Pat_VL_a only)      ;
;       D-reg = start (Y,X) coordinate (Draw_Pat_VL_d only)             ;
;       $C829 contains the line pattern.                                ;
;                                                                       ;
; EXIT: X-reg points to next byte after list                            ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

LF433:          DECA
Draw_Pat_VL_a:  STA     $C823
Draw_Pat_VL:    LDD     ,X              ;Get next coordinate pair
Draw_Pat_VL_d:  STA     <VIA_port_a     ;Send Y to A/D
                CLR     <VIA_port_b     ;Enable mux
                LEAX    2,X             ;Point to next coordinate pair
                INC     <VIA_port_b     ;Disable mux
                STB     <VIA_port_a     ;Send X to A/D
                LDA     Vec_Pattern     ;Get default pattern
                LDB     #$40            ;B-reg = T1 interrupt bit
                STA     <VIA_shift_reg  ;Put pattern in shift register
                CLR     <VIA_t1_cnt_hi  ;Clear T1H (scale factor?)
                BITB    VIA_int_flags   ;Check if T1 timed out (note wasted byte)
                BEQ     LF45C           ;Update pattern if not

;               Don't reset the zero reference if last line is really short?

                CLR     <VIA_shift_reg  ;Clear shift register (blank output)
                LDA     $C823           ;Get line counter
                BNE     LF433           ;Go back for more points
                RTS
```

```
;               This code is for lines that are not really short lines

LF459:          LDA     Vec_Pattern     ;Get default pattern
LF45C:          STA     <VIA_shift_reg  ;Update pattern register
                NOP                     ;Wait a moment
                BITB    <VIA_int_flags  ;Check if T1 timed out
                BEQ     LF459           ;Update pattern again if not
                LDA     $C823           ;Get line counter
                CLR     <VIA_shift_reg  ;Clear shift register (blank output)
                TSTA                    ;Go back if more lines to draw
                BNE     LF433
                JMP     Check0Ref       ;Reset zero reference if necessary

;-------------------------------------------------------------------------;
;       F46E    Draw_VL_mode                                              ;
;                                                                         ;
; This routine processes the vector list pointed to by the X register.    ;
; The current scale factor is used.  The vector list has the following    ;
; format:                                                                 ;
;                                                                         ;
;       mode, rel y, rel x,                                               ;
;       mode, rel y, rel x,                                               ;
;        .      .      .                                                  ;
;        .      .      .                                                  ;
;       mode, rel y, rel x,                                               ;
;       0x01                                                              ;
;                                                                         ;
; where mode has the following meaning:                                   ;
;                                                                         ;
;       < 0  use the pattern in $C829                                     ;
;       = 0  move to specified endpoint                                   ;
;       = 1  end of list, so return                                       ;
;       > 1  draw to specified endpoint                                   ;
;                                                                         ;
; ENTRY DP = $D0                                                          ;
;       X-reg points to the vector list                                   ;
;       $C829 contains the line pattern.                                  ;
;                                                                         ;
; EXIT: X-reg points to next byte after terminator                        ;
;                                                                         ;
;       D-reg trashed                                                     ;
;-------------------------------------------------------------------------;

Draw_VL_mode:   LDA     Vec_0Ref_Enable ;Save old Check0Ref flag
                PSHS    A
                CLR     Vec_0Ref_Enable ;Don't reset the zero reference yet
LF476:          LDA     ,X+             ;Get the next mode byte
                BPL     LF47E
                BSR     Draw_Pat_VL     ;If <0, draw a patterned line
                BRA     LF476

LF47E:          BNE     LF485
                JSR     Mov_Draw_VL     ;If =0, move to the next point
                BRA     LF476
```

```
LF485:          DECA
                BEQ     LF48D
                JSR     Draw_VL         ;If <>1, draw a solid line
                BRA     LF476

LF48D:          PULS    A               ;If =1, exit
                STA     Vec_0Ref_Enable ;Restore old Check0Ref flag
                JMP     Check0Ref       ;Reset zero reference if necessary

;-----------------------------------------------------------------------;
;       F495    Print_Str                                               ;
;                                                                       ;
; This is the routine which does the actual printing of a string.  The  ;
; U register points to the start of the string, while $C82A contains    ;
; the height of the character, cell, and $C82B contains the width of    ;
; the character cell.  The string is terminated with an 0x80.           ;
;                                                                       ;
; The string is displayed by drawing 7 horizontal rows of dots.  The    ;
; first row is drawn for each character, then the second, etc.  The     ;
; character generation table is located at ($F9D4 + $20).  Only         ;
; characters 0x20-0x6F (upper case) are defined; the lower case         ;
; characters a-o produce special icons.                                 ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       U-reg points to the start of the string                         ;
;                                                                       ;
; EXIT: U-reg points to next byte after terminator                      ;
;                                                                       ;
;       D-reg, X-reg trashed                                            ;
;-----------------------------------------------------------------------;

Print_Str:      STU     Vec_Str_Ptr     ;Save string pointer
                LDX     #Char_Table-$20 ;Point to start of chargen bitmaps
                LDD     #$1883          ;$8x = enable RAMP?
                CLR     <VIA_port_a     ;Clear D/A output
                STA     <VIA_aux_cntl   ;Shift reg mode = 110, T1 PB7 enabled
                LDX     #Char_Table-$20 ;Point to start of chargen bitmaps
LF4A5:          STB     <VIA_port_b     ;Update RAMP, set mux to channel 1
                DEC     <VIA_port_b     ;Enable mux
                LDD     #$8081
                NOP                     ;Wait a moment
                INC     <VIA_port_b     ;Disable mux
                STB     <VIA_port_b     ;Enable RAMP, set mux to channel 0
                STA     <VIA_port_b     ;Enable mux
                TST     $C800           ;I think this is a delay only
                INC     <VIA_port_b     ;Enable RAMP, disable mux
                LDA     Vec_Text_Width  ;Get text width
                STA     <VIA_port_a     ;Send it to the D/A
                LDD     #$0100
                LDU     Vec_Str_Ptr     ;Point to start of text string
                STA     <VIA_port_b     ;Disable RAMP, disable mux
                BRA     LF4CB
```

```
LF4C7:          LDA     A,X             ;Get bitmap from chargen table
                STA     <VIA_shift_reg  ;Save in shift register
LF4CB:          LDA     ,U+             ;Get next character
                BPL     LF4C7           ;Go back if not terminator
                LDA     #$81
                STA     <VIA_port_b     ;Enable RAMP, disable mux
                NEG     <VIA_port_a     ;Negate text width to D/A
                LDA     #$01
                STA     <VIA_port_b     ;Disable RAMP, disable mux
                CMPX    #Char_Table_End-$20;     Check for last row
                BEQ     LF50A           ;Branch if last row
                LEAX    $50,X           ;Point to next chargen row
                TFR     U,D             ;Get string length
                SUBD    Vec_Str_Ptr
                SUBB    #$02            ; -  2
                ASLB                    ; *  2
                BRN     LF4EB           ;Delay a moment
LF4EB:          LDA     #$81
                NOP
                DECB
                BNE     LF4EB           ;Delay some more in a loop
                STA     <VIA_port_b     ;Enable RAMP, disable mux
                LDB     Vec_Text_Height ;Get text height
                STB     <VIA_port_a     ;Store text height in D/A
                DEC     <VIA_port_b     ;Enable mux
                LDD     #$8101
                NOP                     ;Wait a moment
                STA     <VIA_port_b     ;Enable RAMP, disable mux
                CLR     <VIA_port_a     ;Clear D/A
                STB     <VIA_port_b     ;Disable RAMP, disable mux
                STA     <VIA_port_b     ;Enable RAMP, disable mux
                LDB     #$03            ;$0x = disable RAMP?
                BRA     LF4A5           ;Go back for next scan line

LF50A:          LDA     #$98
                STA     <VIA_aux_cntl   ;T1->PB7 enabled
                JMP     Reset0Ref       ;Reset the zero reference

;---------------------------------------------------------------------;
;       F511    Random_3                                              ;
;       F517    Random                                                ;
;                                                                     ;
; This routine generates a random 1-byte number, and places it in the ;
; A register.  Random_3 runs through the random number generator      ;
; algorithm three times.  The random number seed is stored in the     ;
; three bytes pointed to by $C87B.                                    ;
;                                                                     ;
; EXIT: A-reg contains the generated random number                    ;
;                                                                     ;
;       All other registers are preserved.                            ;
;---------------------------------------------------------------------;

Random_3:       PSHS    B,X
                LDB     #$02
```

```
                BRA     LF51A

Random:         PSHS    B,X
                CLRB
LF51A:          LDX     Vec_Seed_Ptr
LF51D:          LDA     1,X
                ROLA
                ROLA
                ROLA
                ROLA
                EORA    2,X
                RORA
                ROL     ,X
                ROL     1,X
                ROL     2,X
                DECB
                BPL     LF51D
                LDA     ,X
                PULS    B,X,PC

;-----------------------------------------------------------------------;
;       F533    Init_Music_Buf                                          ;
;                                                                       ;
; This routine clears out the music work buffer, located at             ;
; $C83F-$C84C.                                                          ;
;                                                                       ;
;       X-reg, D-reg trashed                                            ;
;-----------------------------------------------------------------------;

Init_Music_Buf: LDB     #$0D
                LDX     #Vec_Music_Work
                BSR     Clear_x_b
                LDA     #$3F
                STA     6,X
                RTS

;-----------------------------------------------------------------------;
;       F53F    Clear_x_b                                               ;
;                                                                       ;
; This routine clears to 0 the block of memory starting at the          ;
; address contained in the X register, and continuing for the number    ;
; of bytes specified by B+1.                                            ;
;                                                                       ;
; ENTRY X-reg points to the start of the RAM to be cleared.             ;
;       B-reg = number of bytes minus 1 to clear.                       ;
;                                                                       ;
; EXIT: D-reg = $FFFF                                                   ;
;-----------------------------------------------------------------------;

Clear_x_b:      CLRA
                BRA     Clear_x_d

;-----------------------------------------------------------------------;
;       F542    Clear_C8_RAM    (never used by GCE carts?)               ;
```

```
;                                                                      ;
; This routine clears to 0 the block of memory in the range            ;
; $C800-$C8FF.                                                         ;
;                                                                      ;
; EXIT: X-reg = $C800                                                  ;
;       D-reg = $FFFF                                                  ;
;----------------------------------------------------------------------;

Clear_C8_RAM:   LDX     #$C800

;----------------------------------------------------------------------;
;       F545    Clear_x_256                                            ;
;       F548    Clear_x_d                                              ;
;                                                                      ;
; This routine clears the block of memory starting at the contained    ;
; in the X register to zero.                                           ;
;                                                                      ;
; ENTRY X-reg points to the start of RAM to be cleared                 ;
;       D-reg = number of bytes to clear minus 1 (Clear_x_d only)      ;
;                                                                      ;
; EXIT: D-reg = $FFFF                                                  ;
;----------------------------------------------------------------------;

Clear_x_256:    LDD     #$00FF
Clear_x_d:      CLR     D,X
                SUBD    #$0001
                BPL     Clear_x_d
                RTS

;----------------------------------------------------------------------;
;       F550    Clear_x_b_80                                           ;
;       F552    Clear_x_b_a                                            ;
;                                                                      ;
; This routine sets the block of memory pointed to by the X register   ;
; to $80 or the A register.  The B register specifies the number of    ;
; bytes to be cleared.                                                 ;
;                                                                      ;
; ENTRY A-reg = byte to be stored (Clear_x_b_a only)                   ;
;       B-reg = number of bytes to clear ($00 = 256)                   ;
;       X-reg points to start of memory block to clear                 ;
;                                                                      ;
; EXIT: A-reg = $80 (Clear_x_b_80 only)                                ;
;       B-reg = $00                                                    ;
;                                                                      ;
;       All other registers preserved.                                 ;
;----------------------------------------------------------------------;

Clear_x_b_80:   LDA     #$80
Clear_x_b_a:    STA     B,X
                DECB
                BNE     Clear_x_b_a
                STA     ,X
                RTS
```

```
;-------------------------------------------------------------------;
;       F55A    Dec_3_Counters                                      ;
;       F55E    Dec_6_Counters                                      ;
;                                                                   ;
; These routines check either the first three or all six of the     ;
; default counters at $C82E-$C833 and decrements those which are not;
; already zero.                                                     ;
;                                                                   ;
; EXIT: X-reg points to the default counters at $C82E               ;
;       B-reg = $FF                                                 ;
;                                                                   ;
;       All other registers preserved.                              ;
;-------------------------------------------------------------------;

Dec_3_Counters: LDB     #$02
                BRA     LF560

Dec_6_Counters: LDB     #$05
LF560:          LDX     #Vec_Counters

;-------------------------------------------------------------------;
;       F563    Dec_Counters                                        ;
;                                                                   ;
; This routine checks the counters pointed to by the X register and ;
; decrements those which are not already zero.                      ;
;                                                                   ;
; ENTRY B-reg = number of counters minus 1                          ;
;       X-reg points to counter bytes                               ;
;                                                                   ;
; EXIT: B-reg = $FF                                                 ;
;                                                                   ;
;       All other registers preserved.                              ;
;-------------------------------------------------------------------;

Dec_Counters:   TST     B,X
                BEQ     LF569
                DEC     B,X
LF569:          DECB
                BPL     Dec_Counters
                RTS

;-------------------------------------------------------------------;
;       F56D    Delay_3         30 cycles                           ;
;       F571    Delay_2         25 cycles                           ;
;       F575    Delay_1         20 cycles                           ;
;       F579    Delay_0         12 cycles                           ;
;       F57A    Delay_b         5;B + 10 cycles                     ;
;       F57D    Delay_RTS       5 cycles                            ;
;                                                                   ;
; Each of these routines loads the B-register with the indicated    ;
; value, and then loops until the B register value has decremented  ;
; below zero.  Delay_RTS is just an RTS instruction, but at least   ;
; one GCE cartridge calls it.                                       ;
;                                                                   ;
```

```
; Cycle counts do not include timing of the instructions used to       ;
; call the delay routines.                                             ;
;                                                                      ;
; ENTRY B-reg = delay count (Delay_b only)                             ;
;                                                                      ;
; EXIT: B-reg = $FF (except Delay_RTS)                                 ;
;----------------------------------------------------------------------;

Delay_3:        LDB     #$03            ;2 cycles
                BRA     Delay_b         ;3 cycles

Delay_2:        LDB     #$02            ;2 cycles
                BRA     Delay_b         ;3 cycles

Delay_1:        LDB     #$01            ;2 cycles
                BRA     Delay_b         ;3 cycles

Delay_0:        CLRB                    ;2 cycles
Delay_b:        DECB                    ;2 cycles
                BPL     Delay_b         ;3 cycles
Delay_RTS:      RTS                     ;5 cycles

;----------------------------------------------------------------------;
;       F57E    Bitmask_a                                              ;
;                                                                      ;
; This routine takes a bit number, specified in the A register, and    ;
; returns a bit mask with only the specified bit set.                  ;
;                                                                      ;
; ENTRY A-reg contains the bit number                                  ;
;                                                                      ;
; EXIT: A-reg contains the bit mask                                    ;
;                                                                      ;
;       X-reg trashed                                                  ;
;----------------------------------------------------------------------;

Bitmask_a:      LDX     #Bit_Masks
                LDA     A,X
                RTS

;----------------------------------------------------------------------;
;       F584    Abs_a_b                                                ;
;       F58B    Abs_b                                                  ;
;                                                                      ;
; This routine returns the absolute value of the two single byte       ;
; numbers passed in in the A and B registers.  Abs_b only uses the B   ;
; register.  There is a special case: 0x80 is returned as 0x7F.        ;
;                                                                      ;
; ENTRY A-reg contains first value                                     ;
;       B-reg contains second value (Abs_a_b only)                     ;
;                                                                      ;
; EXIT: A-reg contains absolute value of first value                   ;
;       B-reg contains absolute value of second value (Abs_a_b only)   ;
;                                                                      ;
;       All other registers preserved.                                 ;
```

```
;-----------------------------------------------------------------------;

Abs_a_b:        TSTA
                BPL     Abs_b
                NEGA
                BVC     Abs_b
                DECA
Abs_b:          TSTB
                BPL     LF592
                NEGB
                BVC     LF592
                DECB
LF592:          RTS

;-----------------------------------------------------------------------;
;       F593    Rise_Run_Angle                                          ;
;                                                                       ;
; Given a (rise,run) pair, this routine calculates the angle which      ;
; corresponds to that (rise,run) pair.  The returned angle is relative  ;
; to the x-axis (+ is CCW), so to convert it to a Vectrex angle         ;
; (relative to the y-axis, + is CCW), you must subtract the number 0x10 ;
; (90 degrees) from the returned value.                                 ;
;                                                                       ;
; ENTRY DP = $C8                                                        ;
;       A-reg = rise value                                              ;
;       B-reg = run value                                               ;
;                                                                       ;
; EXIT: A-reg = the angle from the x-axis                               ;
;       B-reg = the angle from the x-axis                               ;
;                                                                       ;
;       All other registers preserved.                                  ;
;-----------------------------------------------------------------------;

Rise_Run_Angle: PSHS    X
                STD     <Vec_RiseRun_Tmp
                ROLB
                LDB     #$00
                ROLB
                ROLA
                ROLB
                ASLB
                STB     <Vec_Angle
                LDD     <Vec_RiseRun_Tmp
                BSR     Abs_a_b
                STA     <Vec_RiseRun_Tmp
                CMPB    <Vec_RiseRun_Tmp
                BLS     LF5B2
                INC     <Vec_Angle
                EXG     A,B
                BRA     LF5B2

LF5B0:          LSRA
                LSRB
LF5B2:          CMPA    #$09
```

```
                BHI     LF5B0
                STD     <Vec_RiseRun_Tmp
                LDB     <Vec_Angle
                LDX     #DFC24
                LDB     B,X
                LDX     #DFC2C
                LDA     A,X
                ADDA    <$C835
                ADDA    #$0A
                BITB    #$01
                BNE     LF5D0
                ADDB    A,X
                BRA     LF5D3

LF5D0:          DECB
                SUBB    A,X
LF5D3:          STB     <Vec_Angle
                LDA     <Vec_Angle
                PULS    X,PC

;-----------------------------------------------------------------------;
;       F5D9    Get_Rise_Idx                                            ;
;       F5DB    Get_Run_Idx                                             ;
;                                                                       ;
; These routines are responsible for generating the two index pairs     ;
; which are required by the rest of the rotation routines.  Each index  ;
; pair is two bytes long, and has the following format:                 ;
;                                                                       ;
;       The high byte is obtained by masking the anglewith 0x1F (this   ;
;       forces the angle to be between 0 and 180 degrees), and then     ;
;       using this value to index into the multiplier table.            ;
;                                                                       ;
;       The lower byte contains information about whether the angle     ;
;       lies along either the x or y axis, and whether the rise/run     ;
;       will be positive or negative.                                   ;
;                                                                       ;
;               0 => positive rise, not on an axis, or                  ;
;                    negative run, not on an axis.                      ;
;            0x80 => negative rise, not on an axis, or                  ;
;                    positive run, not on an axis.                      ;
;               1 => positive rise, on an axis, or                      ;
;                    negative run, on an axis.                          ;
;            0x81 => negative rise, on an axis, or                      ;
;                    positive run, on an axis.                          ;
;                                                                       ;
; ENTRY A-reg = the angle value                                         ;
;                                                                       ;
; EXIT: A-reg = slope?                                                  ;
;       B-reg = slope direction?                                        ;
;                                                                       ;
;       X-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Get_Rise_Idx:   ADDA    #$10            ;Offset angle by 90 degrees
```

```
Get_Run_Idx:    LDX     #DFC6D          ;Get address of slope table
                CLRB
                BITA    #$20            ;If angle in 180-360,
                BEQ     LF5E5
                LDB     #$80            ;flag negative rise or positive run
LF5E5:          ANDA    #$1F            ;Mask to multiple of 180 degrees
                CMPA    #$10            ;If 90 degrees
                BNE     LF5EC
                INCB                    ;then rise or run is on an axis
LF5EC:          LDA     A,X             ;Get slope from slope table
                RTS

;-----------------------------------------------------------------------;
;       F5EF    Rise_Run_Idx                                            ;
;                                                                       ;
; This routine gets the index pair for both the rise and run, using     ;
; the passed-in angle value.                                            ;
;                                                                       ;
; ENTRY DP = $C8                                                        ;
;       $C836 contains the angle value                                  ;
;                                                                       ;
; EXIT: $C837-$C838 contains the index pair for the run                 ;
;       $C839-$C83A contains the index pair for the rise                ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Rise_Run_Idx:   PSHS    X               ;Save X-reg
                LDA     <Vec_Angle      ;Get angle
                BSR     Get_Run_Idx     ;Get run index pair for angle
                STD     <Vec_Run_Index
                LDA     <Vec_Angle      ;Get angle
                BSR     Get_Rise_Idx    ;Get rise index pair for angle
                STD     <Vec_Rise_Index
                PULS    X,PC            ;Restore X-reg and return

;-----------------------------------------------------------------------;
;       F5FF    Rise_Run_X                                              ;
;       F601    Rise_Run_Y                                              ;
;       F603    Rise_Run_Len                                            ;
;                                                                       ;
; This routine takes an angle value which is relative to the x- or      ;
; y-axis, and calculates the rise and run for that angle, relative to a ;
; passed-in scalar velocity value.  A large scalar value will cause an  ;
; object to move quickly, while a small scalar value will cause an      ;
; object to move more slowly.                                           ;
;                                                                       ;
; Keep in mind that most games store x & y coordinates as 2 bytes each, ;
; with the upper byte being the actual coordinate, and the lower byte   ;
; being that which is usually added to the rise/run value; when the     ;
; lower byte overflows into the hi byte, then the object will 'move'.   ;
; The rise/run values returned here are meant to be added to the low    ;
; byte -- NOT the hi byte!!                                             ;
;                                                                       ;
```

```
; ENTRY DP = $C8                                                    ;
;       A-reg = the scalar velocity value (except Rise_Run_Len)     ;
;       B-reg = the Vectrex angle value                             ;
;                                                                   ;
; EXIT: A-reg = the rise value                                      ;
;       B-reg = the run value                                       ;
;                                                                   ;
;       All other registers are saved.                              ;
;-------------------------------------------------------------------;

Rise_Run_X:     SUBB    #$10
Rise_Run_Y:     STB     <Vec_Angle
Rise_Run_Len:   STA     <Vec_RiseRun_Len
                BSR     Rise_Run_Idx    ;Get index pair of angle
                BSR     Xform_Run       ;Get run value
                NEGA
                PSHS    A               ;Save run value
                BSR     Xform_Rise      ;Get rise value
                PULS    B,PC            ;Restore run value and return

;-------------------------------------------------------------------;
;       F610    Rot_VL_ab                                           ;
;       F616    Rot_VL                                              ;
;                                                                   ;
; This routine rotates a vector list of length 'n+1', where 'n' is  ;
; specified by the value in the B register.  The A register contains;
; the rotation value, and the X contains a pointer to the vector list.;
; The U register contains a pointer to a buffer into which the      ;
; transformed points are to be saved.  The vector list has the      ;
; following format:                                                 ;
;                                                                   ;
;       rel y, rel x, rel y, rel x, ...                             ;
;                                                                   ;
; ENTRY A-reg = rotation angle value (Rot_VL_ab only)               ;
;       $C836 = rotation angle value (Rot_VL only)                  ;
;       B-reg = number of points - 1 (Rot_VL_ab only)               ;
;       $C823 = number of points - 1 (Rot_VL only)                  ;
;       X-reg points to original vector list                        ;
;       U-reg points to rotated vector list                         ;
;                                                                   ;
; EXIT: DP = $C8                                                    ;
;       X-reg points to next byte after list                        ;
;       U-reg points to next byte after rotated list                ;
;                                                                   ;
;       D-reg trashed                                               ;
;-------------------------------------------------------------------;

Rot_VL_ab:      STA     Vec_Angle
                STB     $C823
Rot_VL:         PSHS    DP
                JSR     DP_to_C8
                BSR     Rise_Run_Idx
                BRA     LF637
```

```
;-------------------------------------------------------------------------;
;       F61F    Rot_VL_Mode                                               ;
;       F62B    Rot_VL_M_dft                                              ;
;                                                                         ;
; This routine rotates a vector list having the following format:         ;
;                                                                         ;
;       mode, rel y, rel x,                                               ;
;       mode, rel y, rel x,                                               ;
;         .     .      .                                                  ;
;         .     .      .                                                  ;
;       mode, rel y, rel x,                                               ;
;       0x01                                                              ;
;                                                                         ;
; The A register contains the rotation value, and the X contains a        ;
; pointer to the vector list.  The U register contains a pointer to a     ;
; buffer into which the transformed points are to be saved.               ;
;                                                                         ;
; ENTRY DP = $C8                                                          ;
;       A-reg = rotation angle value (Rot_VL_Mode only)                   ;
;       X-reg points to original vector list                              ;
;       U-reg points to rotated vector list                               ;
;                                                                         ;
; EXIT: X-reg points to next byte after list                              ;
;       U-reg points to next byte after rotated list                      ;
;                                                                         ;
;       D-reg trashed                                                     ;
;-------------------------------------------------------------------------;

Rot_VL_Mode:    STA     Vec_Angle       ;Save angle
                PSHS    DP              ;Save DP register
                JSR     DP_to_C8        ;DP to RAM
                STA     <$C823          ;Store $C8 (negative value) into $C823
                BSR     Rise_Run_Idx    ;Get index pair of angle
Rot_VL_M_dft:   LDA     ,X+             ;Get mode byte
                STA     ,U+             ;Copy to destination
                BLE     LF637           ;Rotate if not end of list
                CLR     <$C823          ;Exit with $C823 cleared
                PULS    DP,PC           ;Restore DP register and return

LF635:          DEC     <$C823          ;Decrement count for (y,x) list
LF637:          LDA     ,X+             ;Get y coordinate
                BSR     Xform_Rise_a
                STA     ,U              ;Store partial y coordinate
                LDA     ,X              ;Get x coordinate
                BSR     Xform_Run_a
                ADDA    ,U              ;Add to partial y coordinate
                STA     ,U+             ;Store rotated y coordinate
                LDA     -1,X            ;Get y coordinate
                BSR     Xform_Run_a
                STA     ,U              ;Store partial x coordinate
                LDA     ,X+             ;Get x coordinate
                BSR     Xform_Rise_a
                SUBA    ,U              ;Add to partial x coordinate
                STA     ,U+             ;Store rotated x coordinate
```

```
                LDA     <$C823          ;Get counter
                BMI     Rot_VL_M_dft    ;If negative, go back to mode list loop
                BNE     LF635           ;If non-zero, go back to (y,x) list loop
                PULS    DP,PC

;-----------------------------------------------------------------------;
;       F65B    Xform_Run_a                                             ;
;       F65D    Xform_Run                                               ;
;                                                                       ;
; These two routines generate a run value, using the run index pair in  ;
; $C837-$C838.  For Xform_Run_a the scalar value is passed in the       ;
; A register, while for Xform_Run, the scalar value must already be in  ;
; $C83B.  The transformed value is return in the A register.            ;
;                                                                       ;
; ENTRY DP = $C8                                                        ;
;       A-reg = length for rise/run (Xform_Rise_a only)                 ;
;                                                                       ;
; EXIT: A-reg = run value                                               ;
;                                                                       ;
;       B-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Xform_Run_a:    STA     <Vec_RiseRun_Len
Xform_Run:      LDD     <Vec_Run_Index
                BRA     LF665

;-----------------------------------------------------------------------;
;       F661    Xform_Rise_a                                            ;
;       F663    Xform_Rise                                              ;
;                                                                       ;
; These two routines generate a rise value, using the rise index pair   ;
; in $C839-$C83A.  For Xform_Rise_a the scalar value is passed in the   ;
; A register, while for Xform_Rise, the scalar value must already be    ;
; in $C83B.  The transformed value is return in the A register.         ;
;                                                                       ;
; ENTRY DP = $C8                                                        ;
;       A-reg = length for rise/run (Xform_Run_a only)                  ;
;                                                                       ;
; EXIT: A-reg = rise value                                              ;
;                                                                       ;
;       B-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Xform_Rise_a:   STA     <Vec_RiseRun_Len
Xform_Rise:     LDD     <Vec_Rise_Index
LF665:          STB     <$C83C
                BITB    #$01
                BEQ     LF66F
                LDA     <Vec_RiseRun_Len
                BRA     LF679

LF66F:          LDB     <Vec_RiseRun_Len
                BPL     LF676
                COM     <$C83C
```

```
                NEGB
LF676:          MUL
                ADCA    #$00
LF679:          LDB     <$C83C
                BPL     LF67E
                NEGA
LF67E:          RTS

;-----------------------------------------------------------------------;
;       F67F    Move_Mem_a_1                                            ;
;       F683    Move_Mem_a                                              ;
;                                                                       ;
; This routine copies a block of memory, starting at the hi address,    ;
; and working down to the low address.  The base of the source address  ;
; is specified in the U register, and the base of the destination       ;
; address is  specified in the X register.  The A register contains     ;
; the number of bytes to copy; 0x80 is the maximum value which can      ;
; be specified.                                                         ;
;                                                                       ;
; ENTRY A-reg = byte count (Move_Mem_a only)                            ;
;       A-reg = byte count minus 1 (Move_Mem_a_1 only)                  ;
;       X-reg points to the destination                                 ;
;       U-reg points to the source                                      ;
;                                                                       ;
; EXIT  A-reg = $FF                                                     ;
;       B-reg = first byte of source                                    ;
;-----------------------------------------------------------------------;

Move_Mem_a_1:   LDB     A,U             ;Copy the byte
                STB     A,X
Move_Mem_a:     DECA                    ;Decrement the count
                BPL     Move_Mem_a_1    ;Go back until finished
LF686:          RTS

;-----------------------------------------------------------------------;
;       F687    Init_Music_chk                                          ;
;       F68D    Init_Music                                              ;
;       F692    Init_Music_dft                                          ;
;                                                                       ;
; These routines are responsible for filling the music work buffer      ;
; while a sound is being made.  It should be called once during each    ;
; refresh cycle.  If you want to start a new sound, then you must set   ;
; $C856 to 0x01, and point the U-register to the sound block.  If no    ;
; sound is in progress ($C856 = 0), then it returns immediately         ;
; (unless you called Init_Music or Init_Music_dft, which do not make    ;
; this check).  When a sound is in progress, $C856 will be set to 0x80. ;
;                                                                       ;
; These routines process a single note at a time, and calculate the     ;
; amplitude and course/fine tuning values for the 3 sound channels.     ;
; The values calculated are stored in the music work buffer, at         ;
; $C83F-$C84C.                                                          ;
;                                                                       ;
; Music data format:                                                    ;
;                                                                       ;
```

```
;       header word -> $C84F  32 nibble ADSR table                      ;
;       header word -> $C851  8-byte "twang" table                      ;
;       data bytes                                                      ;
;                                                                       ;
; The ADSR table is simply 32 nibbles (16 bytes) of amplitude values.   ;
;                                                                       ;
; The twang table is 8 signed bytes to modify the base frequency of     ;
; each note being played.  Each channel has a different limit to its    ;
; twang table index (6-8) to keep them out of phase to each other.      ;
;                                                                       ;
; Music data bytes:                                                     ;
;       Bits 0-5 = frequency                                            ;
;       Bit 6 clear = tone                                              ;
;       Bit 6 set = noise                                               ;
;       Bit 7 set = next music data byte is for next channel            ;
;       Bit 7 clear, play note with duration in next music data byte:   ;
;               bits 0-5 = duration                                     ;
;               bit 6 = unused                                          ;
;               bit 7 set = end of music                                ;
;                                                                       ;
; ENTRY DP = $C8                                                        ;
;       U-reg points to the start of the music data                     ;
;       $C84D points to frequency table (Init_Music_dft only)           ;
;       $C856 may need to be set.                                       ;
;                                                                       ;
;       D-reg, X-reg, Y-reg, U-reg trashed                              ;
;-----------------------------------------------------------------------;

Init_Music_chk: LDA     <Vec_Music_Flag ;Test sound active flag
                BMI     LF6B3           ;Continue sound if active
                BEQ     LF686           ;Return if sound not active
Init_Music:     LDX     #Freq_Table     ;Save pointer to frequency table
                STX     <Vec_Freq_Table
Init_Music_dft: LDA     #$80            ;Set sound active flag
                STA     <Vec_Music_Flag
                LDD     ,U++            ;Save address of ADSR table
                STD     <Vec_ADSR_Table
                LDD     ,U++            ;Save address of twang table
                STD     <Vec_Twang_Table
                STU     <Vec_Music_Ptr  ;Save pointer to music data
                JSR     Init_Music_Buf  ;Initialize music buffer
                LDD     #$1F1F
                STD     <Vec_ADSR_Timers+1;Init ADSR timers of chans 2 & 3
                LDD     #$0000
                STD     <Vec_Music_Freq+2;Clear frequency of channel 2
                STD     <Vec_Music_Freq+4;Clear frequency of channel 3
                STA     <Vec_Music_Chan ;A-reg = 0 (sound channel number?)
                BRA     LF6EC

; Continue currently playing sound here

LF6B3:          LDU     #Vec_ADSR_Timers;Get address of ADSR timers
                LDB     #$02            ;Count for three channels
LF6B8:          LDA     B,U             ;Get the channel's ADSR timer
```

```
                CMPA    #$1F
                BEQ     LF6C0           ;Skip if at maximum
                INC     B,U             ;Else increment the timer
LF6C0:          DECB                    ;Go back for the other channels
                BPL     LF6B8
                LDX     <Vec_Twang_Table
                LDU     #Vec_Music_Twang
                LDA     #$07            ;Twang limit is 6-8 depending on channel
LF6CA:          INC     ,U              ;Increment twang counter
                CMPA    ,U              ;Check against limit
                BGE     LF6D2
                CLR     ,U              ;Clear it if limit exceeded
LF6D2:          LDB     ,U+             ;Get twang count
                ANDB    #$07            ;Mask out low 3 bits
                LDB     B,X             ;Get twang value from table
                STB     ,U+             ;Update current twang value
                INCA                    ;Increment twang limit
                CMPA    #$09
                BLS     LF6CA           ;Go back until all three channels done
                DEC     <Vec_Duration   ;Decrement the duration timer
                BNE     LF74E           ;Update ADSR while note still playing
LF6E3:          LDA     <Vec_Music_Chan ;Go to next music channel
                DECA
                BPL     LF6EA           ;If < 0, set it to 2
                LDA     #$02
LF6EA:          STA     <Vec_Music_Chan
LF6EC:          LDB     [Vec_Music_Ptr] ;Get next byte of music data
                LDU     #Vec_ADSR_Timers;Clear ADSR timer for this channel
                CLR     A,U
                BITB    #$40            ;If $40 bit of music data set,
                BEQ     LF712           ;we're going to make some noise
                LDX     #Music_Table_1  ;Get bit mask for this channel
                LDA     A,X
                ANDA    <Vec_Music_Wk_7 ;Turn channel bit off for register 7
                STA     <Vec_Music_Wk_7
                LDA     <Vec_Music_Chan ;Set current channel bit in register 7
                ADDA    #$03
                LDA     A,X
                ORA     <Vec_Music_Wk_7
                STA     <Vec_Music_Wk_7
                ANDB    #$1F            ;Mask off low 5 bits of music data
                STB     <Vec_Music_Wk_6 ;and store in register 6
                BRA     LF735

LF712:          LDX     #Music_Table_2  ;If $40 bit of music data was cleared,
                LDA     A,X             ;Get bit mask for this channel
                ANDA    <Vec_Music_Wk_7 ;Turn channel bit off for register 7
                STA     <Vec_Music_Wk_7
                LDA     <Vec_Music_Chan ;Set current channel bit in register 7
                ADDA    #$03
                LDA     A,X
                ORA     <Vec_Music_Wk_7
                STA     <Vec_Music_Wk_7
                LDA     <Vec_Music_Chan ;Get $C855 * 2 + 3
```

```
                ASLA
                ADDA    #Vec_Music_Freq-Vec_ADSR_Timers
                LEAU    A,U             ;Point U-reg to #$C861 + $C855 * 2
                ANDB    #$3F            ;Mask off low 6 bits of music data,
                ASLB                    ;multiply by 2
                LDX     <Vec_Freq_Table ;Get pointer to note-to-frequency table
                LDD     B,X             ;Get note table data
                STD     ,U              ;Store in word at $C861-$C866
LF735:          LDX     <Vec_Music_Ptr  ;Re-get byte of music data
                LDB     ,X+
                STX     <Vec_Music_Ptr  ;Update music data pointer
                TSTB
                BMI     LF6E3           ;If byte>=$80, advance to next channel
                LDB     ,X+             ;Get second byte of music data
                BPL     LF748           ;If >=$80, (terminator)
                JSR     Init_Music_Buf  ;  clear music buffer,
                CLR     <Vec_Music_Flag ;  clear music flag,
                RTS                     ;  and exit

LF748:          STX     <Vec_Music_Ptr  ;Update music data pointer
                ANDB    #$3F            ;Duration in low 6 bits of second byte
                STB     <Vec_Duration   ;Store duration counter
LF74E:          LDY     <Vec_ADSR_Table ;Get pointer to ADSR table
                LDU     #Vec_ADSR_Timers;Point to ADSR timer table
                LDX     #Vec_Music_Wk_A
                LDA     #$02            ;Count for three channels
LF759:          LDB     ,U+             ;Get channel timer?
                BITB    #$01            ;Test low bit of ADSR index
                BEQ     LF766
                LSRB                    ;If odd, divide ADSR index by by 2
                LDB     B,Y             ;Get low nibble from ADSR table
                ANDB    #$0F
                BRA     LF76D

LF766:          LSRB                    ;If even, divide ADSR index by 2
                LDB     B,Y             ;Get high nibble from ADSR table
                LSRB
                LSRB
                LSRB
                LSRB
LF76D:          STB     A,X             ;Store ADSR value in regs 10-12
                DECA                    ;Decrement channel counter
                BPL     LF759           ;Go back for next channel
                LDU     #Vec_Music_Freq+6;Point to base frequency table
                LDX     #Vec_Music_Wk_5 ;Point to twang table
LF778:          LDD     ,--U            ;Get next base frequency
                TST     -8,U            ;Test twang value
                BPL     LF788
                NEG     -8,U            ;If <0, negate twang table entry
                SUBB    -8,U            ;Subtract negated value from frequency
                SBCA    #$00            ;Propagate borrow to high byte
                NEG     -8,U            ;Un-negate twang entry
                BRA     LF78C
```

```
LF788:          ADDB    -8,U            ;If >0 add twang to base frequency
                ADCA    #$00            ;Propagate carry to high byte
LF78C:          STD     ,X++            ;Store freq in regs 5/4, 3/2, 1/0
                CMPX    #Vec_Music_Work+14
                BNE     LF778
LF793_RTS:      RTS

;-----------------------------------------------------------------------;
;       F7A9    Select_Game                                             ;
;                                                                       ;
; This routine provides a game with the means for allowing the player   ;
; to choose the game number he would like to play, and the number of    ;
; players.  The game indicates the number of game versions available,   ;
; by placing the  value in the B register.  The number of players       ;
; allowed is specified in the A register.  If a parameter is passed in  ;
; with a value of 0, then the corresponding question will not be asked. ;
; The number of players selected is returned in $C879, while the game   ;
; number selected is returned in $C87A.                                 ;
;                                                                       ;
; This routine performs most of the work involved in allowing the       ;
; player to select a game version and the number of players.  It        ;
; displays the game # and player options, and allows the player a       ;
; certain amount of time to modify their values.  Anytime one of the    ;
; buttons is used to modify a value, the timer will be restarted.       ;
; When a button is pressed, the associated value is modified,           ;
; and then redisplayed on the screen.  This routine will return when    ;
; either the timer expires, or button 4 is pressed.                     ;
;                                                                       ;
; ENTRY A-reg = maximum number of players allowed                       ;
;       B-reg = number of game versions available                       ;
;                                                                       ;
; EXIT: DP = $C8                                                        ;
;       $C879 contains number of players selected                       ;
;       $C87A contains the game version selected                        ;
;                                                                       ;
;       D-reg, X-reg, Y-reg trashed                                     ;
;-----------------------------------------------------------------------;

Player_Str:     FDB     $20C0
                FDB     $40C0
                FCC     "PLAYER"
                FCB     $80

Game_Str:       FDB     $E0C0
                FDB     $01C0
                FCC     " GAME"
                FCB     $80

Select_Game:    STD     Vec_Max_Players ;Save max players and games
                TSTA                    ;If non-zero players specified,
                BEQ     LF7B1
                LDA     #$01            ;  set selection to 1
LF7B1:          TSTB                    ;If non-zero games specified,
                BEQ     LF7B6
```

```
                LDB     #$01            ;  set selection to 1
LF7B6:          STD     Vec_Num_Players ;Save default selection
                JSR     DP_to_C8        ;DP to RAM
                LDD     #$F850
                STD     <Vec_Text_HW
                STA     <$C83C          ;Set $C83C flag to non-zero
                BRA     LF82C

LF7C5:          JSR     Wait_Recal      ;Start with a fresh frame, DP to I/O
                CLRA                    ;Read buttons, all in direct mode
                JSR     Read_Btns_Mask
                JSR     Dec_3_Counters
                JSR     Intensity_7F    ;Brightness to normal
                LDA     Vec_Num_Players ;Display number of players
                LDY     #Player_Str
                BSR     Display_Option
                LDA     Vec_Num_Game    ;Display currently selected game
                LDY     #Game_Str
                BSR     Display_Option
                JSR     DP_to_C8        ;DP to RAM
                LDA     <$C83C          ;If $C83C=0, check buttons
                BEQ     LF7F1
                LDA     <Vec_Btn_State
                BNE     LF82C           ;If any button pressed, reset timers
                CLR     <$C83C          ;Clear $C83C flag
LF7F1:          LDA     <Vec_Counter_2  ;Return if counter 2 timed out
                BEQ     LF793_RTS
                LDA     <Vec_Counter_1  ;If repeat timer not timed out,
                BNE     LF7C5           ;  ignore the buttons
                LDA     <Vec_Button_1_4
                BNE     LF793_RTS       ;Return if button 4 pressed
                LDA     <Vec_Button_1_1
                BEQ     LF810
                LDA     <Vec_Num_Players; Ignore if no players option
                BEQ     LF810           ;If button 1 pressed,
                INCA                    ;   increment number of players
                CMPA    <Vec_Max_Players
                BLS     LF80C
                LDA     #$01            ;Reset to 1 if max players exceeded
LF80C:          STA     <Vec_Num_Players; Update number of players
                BRA     LF82C           ;Update timers and go back to the loop

LF810:          LDA     <Vec_Num_Game   ;Return to the loop if no game options
                BEQ     LF7C5
                LDB     <Vec_Button_1_2
                BEQ     LF821
                INCA                    ;If button 2 down, increment game
                CMPA    <Vec_Max_Games
                BLS     LF82A
                LDA     #$01            ;Reset to 1 if maximum exceeded
                BRA     LF82A

LF821:          LDB     <Vec_Button_1_3
                BEQ     LF7C5
```

```
                DECA                    ;If button 3 down, decrement game
                BNE     LF82A
                LDA     <Vec_Max_Games  ;Reset to max if zero reached
LF82A:          STA     <Vec_Num_Game
LF82C:          LDA     #$F3            ;Reset timers
                STA     <Vec_Counter_2
                COMA
                STA     <Vec_Counter_1
                BRA     LF7C5           ;Go back to the loop

;-----------------------------------------------------------------------;
;       F835    Display_Option  (not called by GCE cartridges)          ;
;                                                                       ;
; This routine displays the player or game option string, along with    ;
; the current value for that option.  The A-register contains the       ;
; value of the option, while the Y-register points to a block of the    ;
; following form:                                                       ;
;                                                                       ;
;       rel y, rel x,      ( for value )                                ;
;       rel y, rel x,      ( for option string)                         ;
;       option string,                                                  ;
;       0x80                                                            ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       A-reg=the option value.                                         ;
;       Y-reg points to the string block.                               ;
;                                                                       ;
;       D-reg, U-reg, X-reg trashed                                     ;
;-----------------------------------------------------------------------;

Display_Option: LDX     #$C85E          ;Point to temp storage
                PSHS    A               ;Save option
                BSR     Clear_Score     ;Clear scratch score accumulator
                LDA     ,S+             ;Get option back
                BEQ     LF84E           ;Exit printing nothing if option = zero
                BSR     Add_Score_a     ;Put option in scratch score accumulator
                TFR     X,U             ;Transfer X to be printed
                LDD     ,Y++            ;Get (y,x) of value
                JSR     Print_Str_d     ;Print value
                TFR     Y,U             ;Transfer Y to be printed
                JSR     Print_Str_yx    ;Print option string
LF84E:          RTS

;-----------------------------------------------------------------------;
;       F84F    Clear_Score                                             ;
;                                                                       ;
; This routine will initialize the passed-in score string (pointed to   ;
; by the X-register) to the following value:                            ;
;                                                                       ;
;       "     0",0x80                                                   ;
;                                                                       ;
; ENTRY X-reg points to seven byte score accumulator                    ;
;                                                                       ;
;       D-reg trashed                                                   ;
```

```
;--------------------------------------------------------------------;

Clear_Score:    LDD     #' '*256+' '    ;Store the leading blanks
                STD     ,X
                STD     2,X
                STA     4,X
                LDD     #'0'*256+$80    ;Store the zero and terminator byte
                STD     5,X
                RTS

;--------------------------------------------------------------------;
;       F85E    Add_Score_a                                          ;
;       F87C    Add_Score_d                                          ;
;                                                                    ;
; These routines take the BCD value in the D-register or the binary  ;
; value in the A-register, and add it to the 6-byte ASCII number     ;
; pointed by the X-register.                                         ;
;                                                                    ;
; ENTRY A-reg = binary value (Add_Score_a only)                      ;
;       D-reg = BCD value (Add_Score_d only)                         ;
;       U-reg = BCD conversion of A-reg (Add_Score_a only)           ;
;       X-reg points to six byte ASCII score accumulator             ;
;                                                                    ;
;       D-reg trashed                                                ;
;--------------------------------------------------------------------;

Add_Score_a:    LDU     #$0000          ;Initialize BCD result to zero
LF861:          CMPA    #99             ;Add in the hundreds
                BLS     LF86D
                SUBA    #100
                LEAU    $0100,U
                BRA     LF861

LF86D:          CMPA    #9              ;Add in the tens
                BLS     LF878
                SUBA    #10
                LEAU    $10,U
                BRA     LF86D

LF878:          LEAU    A,U             ;Add in the ones
                TFR     U,D             ;Move it to the D-register

Add_Score_d:    PSHS    A               ;Save BCD on stack in reverse order
                PSHS    B
                LDB     #$05
LF882:          CLRA                    ;Add zero to 10000 and 100000 digits
                CMPB    #$01
                BLS     LF897
                BITB    #$01            ;Add right nibble to hundreds and ones
                BEQ     LF88F
                LDA     ,S
                BRA     LF895

LF88F:          LDA     ,S+             ;Add left nibble to thousands and tens
```

```
                LSRA
                LSRA
                LSRA
                LSRA
LF895:          ANDA    #$0F            ;Isolate desired nibble
LF897:          ADDA    $C823           ;Add in carry ($C823 is normally zero)
                CLR     $C823           ;Clear carry
                ADDA    B,X             ;Add to digit
                CMPA    #'0'-1          ;If digit was a blank,
                BGT     LF8A5
                ADDA    #$10            ;  promote the result to a digit
LF8A5:          CMPA    #'9'            ;If a carry has occurred,
                BLS     LF8AE
                SUBA    #10             ;  subtract ten
                INC     $C823           ;  and set carry flag
LF8AE:          STA     B,X             ;Store resulting digit
                DECB                    ;Go back for more digits
                BPL     LF882
                CLR     $C823           ;Clear $C823 back to zero
                CLRB

;-----------------------------------------------------------------------;
;       F8B7    Strip_Zeros                                             ;
;                                                                       ;
; This routine strips the leading zeros from a score accumulator.       ;
;                                                                       ;
; ENTRY B-reg = first digit to start with (usually zero)                ;
;       X-reg points to six byte ASCII score accumulator                ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Strip_Zeros:    LDA     B,X             ;Test current digit
                CMPA    #'0'
                BNE     LF8C6           ;Exit if not zero
                LDA     #' '            ;Change it to a blank
                STA     B,X
                INCB
                CMPB    #$05
                BLT     Strip_Zeros
LF8C6:          RTS

;-----------------------------------------------------------------------;
;       F8C7    Compare_Score                                           ;
;                                                                       ;
; This routine will compare two BCD score strings, to determine which   ;
; one is higher.  The two strings are pointed to by the U and X         ;
; registers.  Depending upon how the scores compare, one of the         ;
; following values will be returned in the A-register:                  ;
;                                                                       ;
;       1) The scores are the same: a = 0                               ;
;       2) X score > U score:   a = 1                                   ;
;       3) U score > X score:   a = 2                                   ;
;                                                                       ;
```

```
; ENTRY X-reg points to first score string (terminated with $80)        ;
;       U-reg points to second score string                             ;
;                                                                       ;
; EXIT: A-reg returns result of the compare                             ;
;                                                                       ;
;       B-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Compare_Score:  PSHS    X,U             ;Save score pointers
                CLRA                    ;Default to scores are the same
LF8CA:          LDB     ,X+
                BMI     LF8D6           ;Return if end of string
                CMPB    ,U+
                BEQ     LF8CA           ;Continue if byte is the same
                BHI     LF8D5           ;Return 1 if X > U
                INCA                    ;Return 2 if U > X
LF8D5:          INCA
LF8D6:          PULS    X,U,PC          ;Restore pointers and return

;-----------------------------------------------------------------------;
;       F8D8    New_High_Score                                          ;
;                                                                       ;
; This routine compares a players score string, pointed to by the       ;
; X register, to the current hi score, pointed by the U register.  If   ;
; the player's score is higher than the currently saved hi score, then  ;
; the player's score will be copied into the hi score buffer pointed    ;
; to by the U register.                                                 ;
;                                                                       ;
; ENTRY X-reg points to a player's score string                         ;
;       U-reg points to the high score string (usually $CBEB?)          ;
;                                                                       ;
;       X-reg, U-reg, D-reg trashed                                     ;
;-----------------------------------------------------------------------;

New_High_Score: BSR     Compare_Score   ;Compare the scores
                CMPA    #$01
                BNE     LF8E4           ;Return if X is not > U
LF8DE:          LDA     ,X+             ;Copy the new high score
                STA     ,U+
                BPL     LF8DE           ;until end of string encountered
LF8E4:          RTS

;-----------------------------------------------------------------------;
;       F8E5    Obj_Will_Hit_u                                          ;
;       F8F3    Obj_Will_Hit                                            ;
;                                                                       ;
; This routine first modifies the position of the object, and then it   ;
; checks to see if the missile has hit the object.  The Y register      ;
; contains the (y,x) position of the object, the U register contains    ;
; a pointer to the (y,x) modification values, the X register contains   ;
; the missile (y,x) position, and the D register contains the           ;
; (height/2, width/2) of the object.                                    ;
;                                                                       ;
; (0,u) is temporarily added to the y position of the object, and       ;
```

```
; (1,u) is temporarily added to the x position.                          ;
;                                                                       ;
; ENTRY Y-reg = (y,x) position of the object                            ;
;       X-reg = (y,x) position of the missile                           ;
;       U-reg points to movement (y,x) (Mov_Obj_Hit_u only)             ;
;       U-reg = movement (y,x) (Mov_Obj_Hit only)                       ;
;       D-reg = (h/2,w/2) size of object                                ;
;                                                                       ;
; EXIT: Carry bit set if the object & missile have collided             ;
;                                                                       ;
;       ALL registers saved.  Even the original Y-register.             ;
;-----------------------------------------------------------------------;

Obj_Will_Hit_u: PSHS    Y               ;Save regs for the hit-test code
                PSHS    D,X,Y
                LDD     4,S             ;Get object position
                ADDA    ,U              ;Add it to the modification values
                ADDB    1,U
LF8EF:          STD     4,S             ;Put updated object position back
                BRA     LF903           ;Go do the hit-test

Obj_Will_Hit:   PSHS    Y               ;Save regs for the hit-test code
                PSHS    D,X,Y
                TFR     U,D             ;Get modification values
                ADDA    4,S             ;Add them to the object position
                ADDB    5,S
                BRA     LF8EF           ;Put update position back and hit-test

;-----------------------------------------------------------------------;
;       F8FF    Obj_Hit                                                 ;
;                                                                       ;
; Thit routine checks to see if a missile hashit an object.  If the     ;
; missile has hit the object, then the carry bit will be set;           ;
; otherwise, the carry bit will be cleared.  A hit is checked for in    ;
; the following fashion:                                                ;
;                                                                       ;
;   if (object y-height/2) <= missile y <= (object y+height/2)          ;
;                              and                                      ;
;      (object x-width/2)  <= missile x <= (object x+width/x)           ;
;                                                                       ;
;   then the missile hit, otherwise it missed.                          ;
;                                                                       ;
; ENTRY Y-reg = (y,x) position of the object                            ;
;       X-reg = (y,x) position of the missile                           ;
;       D-reg = (h/2,w/2) size of object                                ;
;                                                                       ;
; EXIT: Carry bit set if the object & missile have collided             ;
;                                                                       ;
;       All registers preserved.                                        ;
;-----------------------------------------------------------------------;

Obj_Hit:        PSHS    Y               ;Save some regs
                PSHS    D,X,Y
LF903:          TFR     S,X             ;Point X to the stack
```

```
                CLRB                    ;Offset to point to y
LF906:          ABX
                LDA     4,X             ;Get height/2
                ADDA    ,X              ;Add object y
                BVC     LF90F
                LDA     #$7F            ;Set to $7F if overflow
LF90F:          CMPA    2,X             ;Branch if missile out of range
                BLT     LF928
                LDA     4,X             ;Get height/2
                SUBA    ,X              ;Subtract object y
                BVC     LF91B
                LDA     #$80            ;Set to $80 if overflow
LF91B:          CMPA    2,X             ;Branch if missile out of range
                BGT     LF928
                INCB                    ;Offset to point to x
                CMPB    #$02
                BCS     LF906           ;Go back for x
                ORCC    #$01            ;Object in range, set carry
                BRA     LF92A

LF928:          ANDCC   #$FE            ;Object not in range, clear carry
LF92A:          PULS    D,X,Y
                PULS    Y,PC

;-----------------------------------------------------------------------;
;       F92E    Explosion_Snd                                           ;
;                                                                       ;
; This routine appears to generate some type of an explosion sound,     ;
; dependent upon the 4 bytes which are pointed to by the U register.    ;
; You will probably need to call Do_Sound for this to do anything.      ;
;                                                                       ;
; The format of the 4-byte block is:                                    ;
;       1)      Bits 0-2 = ?    Stored in $C85D                         ;
;               Bits 3-5 = ?    Stored in $C853                         ;
;               Bits 6-7 = 0                                            ;
;               Bits 0-2 and 3-5 are ORed and stored in bits 0-2 of     ;
;                                                               $C854   ;
;       2)      <0 = ?          Something to do with register 6         ;
;               =0 = ?                                                  ;
;               >0 = ?                                                  ;
;       3)      <0 = ?                                                  ;
;               =0 = ?                                                  ;
;               >0 = ?                                                  ;
;       4)      Speed?  Higher values = lower duration?                 ;
;                                                                       ;
; ENTRY DP = $C8                                                        ;
;       U-reg points to 4-byte block of data if $C867 high bit set      ;
;                                                                       ;
;       D-reg, X-reg trashed                                            ;
;-----------------------------------------------------------------------;

Explosion_Snd:  LDA     <Vec_Expl_Flag
                BPL     LF95B
                ANDA    #$7F
```

```
                STA     <Vec_Expl_Flag
                LDX     #Vec_Expl_1     ;Copy 4 bytes from U-reg to $C858
                LDA     #$04
                JSR     Move_Mem_a
                LSRB                    ;Divide first byte by 8
                LSRB
                LSRB
                ORB     <Vec_Expl_1     ;OR with first byte
                ANDB    #$07            ;AND with 7
                STB     <Vec_Expl_Chans ;store in $C854
                LDB     <Vec_Expl_1     ;Get first byte
                ANDB    #$38            ;Mask off bits 3-5
                STB     <Vec_Expl_ChanA ;store in $C853
                LDB     <Vec_Expl_1     ;Get first byte
                ANDB    #$07            ;AND with 7
                STB     <Vec_Expl_ChanB ;store in $C85D
                LDB     #$02            ;Start with channel number 2
                STB     <Vec_Expl_Chan
                LDA     #$7F            ;Initialize time count
                BRA     LF968

LF95B:          LDA     <Vec_Expl_Timer
                BEQ     LF9C9_RTS
                SUBA    <Vec_Expl_4
                BPL     LF968
                CLRB
                STB     <Vec_Expl_Timer
                BRA     LF9CA

LF968:          STA     <Vec_Expl_Timer
                LSRA
                LSRA
                LDB     <Vec_Expl_ChanA
                BEQ     LF97D
                STA     <Vec_Music_Wk_6
                LDB     <Vec_Expl_2
                BMI     LF97B
                BEQ     LF97D
                TFR     A,B
                COMB
LF97B:          STB     <Vec_Music_Wk_6
LF97D:          LSRA
                CMPA    #$07
                BLS     LF987
                CMPA    #$0F
                BEQ     LF987
                INCA
LF987:          LDB     <Vec_Expl_3
                BMI     LF991
                BEQ     LF98F
                EORA    #$0F
LF98F:          TFR     A,B
LF991:          BSR     LF9CA
                LDB     <Vec_Expl_ChanB
```

```
                BEQ     LF9C2
LF997:          LDA     <Vec_Expl_Chan  ;Get channel number
                DECA                    ;Decrement channel number
                BPL     LF99E
                LDA     #$02            ;Reset to 2 if less than zero
LF99E:          STA     <Vec_Expl_Chan  ;Save channel number
                JSR     Bitmask_a       ;Get bit mask of the channel
                BITA    <Vec_Expl_ChanB
                BEQ     LF997           ;Go back if not in for $C85D
                LDB     <Vec_Expl_Chan
                ASLB                    ;Negative of channel number ; 2
                NEGB
                LDX     #Vec_Music_Wk_1 ;  (registers 1, 3, and 5)
                LEAX    B,X
                JSR     Random
                ANDA    #$0F
                CMPA    #$05
                BHI     LF9BC
                ASLA
                ADDA    #$05
LF9BC:          STA     ,X
                LDA     <Vec_Random_Seed+1
                STA     1,X
LF9C2:          LDA     <Vec_Expl_1
                COMA
                ANDA    <Vec_Music_Wk_7
                STA     <Vec_Music_Wk_7
LF9C9_RTS:      RTS

LF9CA:          LDA     <Vec_Expl_Chans
                LDX     #Vec_Music_Wk_7
LF9CF:          TSTA                    ;Exit if all channels done
                BEQ     LF9DB_RTS
                LEAX    -1,X            ;Point to next register (8-10)
                LSRA
                BCC     LF9CF
                STB     ,X              ;Store noise value if chan in use
                BRA     LF9CF

LF9DB_RTS:      RTS

Bit_Masks:      FCB     $01,$02,$04,$08,$10,$20,$40,$80 ;For Bitmask_a

Music_Table_1:  FCB     $F7,$EF,$DF,$01,$02,$04         ;For noise

Music_Table_2:  FCB     $FE,$FD,$FB,$08,$10,$20         ;For music

Recal_Points:   FDB     $7F7F,$8080                     ;For Recalibrate

Char_Table:     FDB     $0020,$5050,$20C8,$2010,$1040,$2000,$0000,$0008
                FDB     $3020,$7070,$10F8,$30F8,$7070,$0060,$0000,$0070
                FDB     $7020,$F070,$F0F8,LF878,$8870,$0888,$8088,$88F8
                FDB     $F070,$F070,$F888,$8888,$8888,$F870,$8070,$2000
                FDB     $0020,$0820,$0000,$0038,$1020,$4444,$00FE,$FFFE
```

```
                FDB     $0070,$5050,$78C8,$5020,$2020,$A820,$0000,$0008
                FDB     $4860,$8888,$3080,$4008,$8888,$6060,$1000,$4088
                FDB     $8850,$4888,$4880,$8080,$8820,$0890,$80D8,$C888
                FDB     $8888,$8888,$A888,$8888,$8888,$0840,$8008,$5000
                FDB     $0070,$0C20,$7070,$0044,$1070,$0000,$6C82,$FFFE

                FDB     $0070,$50F8,$A010,$5040,$4010,$7020,$0000,$0010
                FDB     $4820,$0808,$50F0,$8010,$8888,$6000,$2078,$2008
                FDB     $A888,$4880,$4880,$8080,$8820,$08A0,$80A8,$A888
                FDB     $8888,$8840,$2088,$8888,$5050,$1040,$4008,$8800
                FDB     $70A8,$0A20,$88F8,$60BA,$3820,$0000,$9282,$FFFE

                FDB     $0020,$0050,$7020,$6000,$4010,$A8F8,$0070,$0020
                FDB     $4820,$7030,$9008,$F020,$7078,$0060,$4000,$1010
                FDB     $B888,$7080,$48E0,$E098,$F820,$08C0,$80A8,$9888
                FDB     $F088,$F020,$2088,$50A8,$2020,$2040,$2008,$0000
                FDB     $FE20,$0820,$88F8,$F0A2,$38F8,$8238,$9282,$FFFE

                FDB     $0000,$00F8,$7040,$A800,$4010,$A820,$4000,$0040
                FDB     $4820,$8008,$F808,$8840,$8808,$6060,$2078,$2020
                FDB     $B0F8,$4880,$4880,$8088,$8820,$08A0,$8088,$8888
                FDB     $80A8,$A010,$2088,$50A8,$5020,$4040,$1008,$0000
                FDB     $FE20,$78A8,$88F8,$F0BA,$7C20,$4444,$6C82,$FFFE

                FDB     $0000,$0050,$2898,$9000,$2020,$0020,$4000,$0080
                FDB     $4820,$8088,$1088,$8880,$8810,$6020,$1000,$4000
                FDB     $8088,$4888,$4880,$8088,$8820,$8890,$8888,$8888
                FDB     $8090,$9088,$2088,$20A8,$8820,$8040,$0808,$0000
                FDB     $4820,$F070,$7070,$6044,$6C50,$3882,$0082,$FFFE

Char_Table_End: FDB     $0020,$0050,$F898,$6800,$1040,$0000,$8000,$8080
                FDB     $3070,$F870,$1070,$7080,$7060,$0040,$0000,$0020
                FDB     $7888,$F070,$F0F8,$8078,$8870,$7088,$F888,$88F8
                FDB     $8068,$8870,$2070,$2050,$8820,$F870,$0870,$00F8
                FDB     $0020,$6020,$0000,$0038,$8288,$0000,$00FE,$FFFE

;       These tables are used by the rise/run calculations

DFC24:          FDB     $0011,$4130,$2110,$2031

DFC2C:          FDB     $0001,$0306,$0A0F,$151C,$242D,$0810,$0810,$0B08
                FDB     $100D,$0A08,$100E,$0B09,$0810,$0E0C,$0A09,$0810
                FDB     $0E0D,$0B0A,$0908,$100F,$0D0C,$0B0A,$0908,$100F
                FDB     $0E0C,$0B0A,$0909,$0810,$0F0E,$0D0C,$0B0A,$0909
                FCB     $08

DFC6D:          FDB     $0019,$324A,$6279,$8EA2,$B5C6,$D5E2,$EDF5,$FBFF
                FDB     $FFFF,$FBF5,$EDE2,$D5C6,$B5A2,$8E79,$624A,$3219

;       Music note to frequency table

Freq_Table:     FDB     $03BD,$0387,$0354,$0324,$02F7,$02CD,$02A4,$027E
                FDB     $025B,$0239,$0219,$01FB,$01DE,$01C3,$01AA,$0192
```

```
                FDB     $017C,$0166,$0152,$013F,$012D,$011C,$010C,$00FD
                FDB     $00EF,$00E2,$00D5,$00C9,$00BE,$00B3,$00A9,$00A0
                FDB     $0097,$008E,$0086,$007F,$0078,$0071,$006B,$0065
                FDB     $005F,$005A,$0055,$0050,$004B,$0047,$0043,$003F
                FDB     $003C,$0038,$0035,$0032,$002F,$002D,$002A,$0028
                FDB     $0026,$0024,$0022,$0020,$001E,$001C,$001B,$0000

;       FD0D = power-on music and music for Crazy Coaster and Narrow Escape

Intro_Music:    FDB     DFEE8,DFEB6,$931F,$0C93,$1F06,$989F,$243C,$1180

;       FD1D = music for Berzerk?

DFD1D:          FDB     DFD69,DFD79,$2107,$2107,$2107,$2107,$2107,$2107
                FDB     $210E,$999F,$240E,$959B,$200E,$2107,$2107,$2107
                FDB     $2107,$2107,$2107,$9DA3,$280E,$A0A6,$2B0E,$2202
                FDB     $2802,$2D02,$2802,$2202,$2802,$2D02,$2802,$2202
                FDB     $2802,$2D02,$2802,$2E02,$2D28,$2180

;       FD69 = ADSR table for Berzerk and FF7A

DFD69:          FDB     $EFFF,$FEDC,$BA00,$0000,$0000,$0000,$0000,$0000

;       FD79 = twang table for Berzerk and Scramble

DFD79:          FDB     $0001,$0201,$00FF,$FEFF

;       FD81 = music

DFD81:          FDB     DFDC3,DFEB6,$5124,$5006,$5006,$500C,$5006,$5006
                FDB     $5004,$5004,$5004,$5018,$5004,$5004,$5004,$500C
                FDB     $500C,$5024,$5006,$5006,$500C,$5006,$5006,$5004
                FDB     $5004,$5004,$5018,$5004,$5004,$5004,$500C,$5018
                FDB     $2680

;       FDC3 = ADSR table for FD81 and FF8F

DFDC3:          FDB     $FDBA,$9876,$5544,$3322,$1100,$0000,$0000,$0000

;       FDD3 = music for Scramble

DFDD3:          FDB     DFE28,DFD79,$981C,$103F,$0898,$1C04,$981C,$0498
                FDB     $1C10,$3F08,$981C,$0498,$1C04,$981C,$0893,$1808
                FDB     $981C,$089C,$1F08,$981C,$0893,$1808,$981C,$0893
                FDB     $1808,$981C,$089C,$1F08,$981C,$0893,$1808,$981C
                FDB     $0893,$1808,$981C,$089C,$1F08,$981C,$0893,$1808
                FCB     $9C,$1F,$30,$1A,$80

;       FE28 = ADSR table for Scramble, FF26, FF44, FF62

DFE28:          FDB     $FFFE,$DCBA,$9876,$5432,$1000,$0000,$0000,$0000

;       FE38 = music for Solar Quest
```

```
DFE38:          FDB     DFE66,DFEB6,$0C18,$1118,$0C18,$1118,$0C18,$1118
                FDB     $0C12,$0C06,$1118,$9D21,$189F,$2318,$A124,$18A3
                FDB     $2618,$9FA4,$2818,$0712,$0706,$003C,$1880

;       FE66 = ADSR table for Solar Quest

DFE66:          FDB     $DEEF,$FEDC,$BA00,$0000,$0000,$0000,$0000,$0000

;       FE76 = music

DFE76:          FDB     DFEB2,DFEB6,$1806,$1A06,$1C0C,$180C,$1A24,$2318
                FDB     $1706,$1806,$1A0C,$170C,$1824,$2418,$A428,$0CA3
                FDB     $260C,$A124,$0C9F,$230C,$9D21,$189A,$1F18,$1706
                FDB     $1806,$1A0C,$170C,$1824,$2424,$1880

;       FEB2 = ADSR table for FE76

DFEB2:          FDB     $FFEE,$DDCC

;       FEB6 = "flat" twang table

DFEB6:          FDB     $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

;       FEC6 = music

DFEC6:          FDB     DFEE8,DFEB6,$969A,$1D1E,$9195,$181E,$9498,$1B1E
                FDB     $8F94,$1814,$160A,$8C91,$1514,$160A,$9195,$1832
                FDB     $1880

;       FEE8 = ADSR table for FEC6

DFEE8:          FDB     $EEFF,$FFEE,$EEDD,$CCBB,$AA99,$8888,$8888,$8888

;       FEF8 = music for Melody Master

DFEF8:          FDB     DFF16,DFEB6,$1C06,$1F06,$1C06,$1806,$1A06,$1806
                FDB     $1506,$1306,$1806,$1306,$1706,$181E,$1880

;       FF16 = ADSR table for FEF8

DFF16:          FDB     $FFFF,$EEEE,$DDDD,$CCCC,$0000,$0000,$0000,$0000

;       FF26 = music

DFF26:          FDB     DFE28,DFEB6,$160F,$1605,$1605,$1605,$1A0F,$160F
                FDB     $1D0F,$1D05,$1D05,$1D05,$210F,$1D32,$1D80

;       FF44 = music

DFF44:          FDB     DFE28,DFEB6,$1606,$1602,$1602,$1602,$1A06,$1606
                FDB     $1D06,$1D02,$1D02,$1D02,$2106,$1D32,$1180

;       FF62 = music
```

```
DFF62:          FDB     DFE28,DFEB6,$1B0F,$1605,$1605,$1605,$1730,$1605
                FDB     $1605,$1605,$1730,$1680

;       FF7A = music

DFF7A:          FDB     DFD69,DFEB6,$A023,$12A0,$230C,$9C20,$069E,$2112
                FCB     $9C,$20,$32,$13,$80

;       FF8F = music

DFF8F:          FDB     DFDC3,DFEB6,$1604,$1604,$1604,$1604,$1A08,$1C80

;-----------------------------------------------------------------------;
;       FF9F    Draw_Grid_VL                                            ;
;                                                                       ;
; This routine apparently will draw a vector list using a 16x16 grid,   ;
; and occasionally using regular vector lists too.  This could possibly ;
; be useful for drawing gridded things like a chess board and all of    ;
; its pieces at the same time.                                          ;
;                                                                       ;
; The master vector list contains multiple sublists that start with     ;
; a flag byte:                                                          ;
;       Bit 7 = draw the next regular vector list (from X-reg) first    ;
;       Bit 6 = this is the last sublist in the master vector list      ;
;       Bits 5,4 = unused                                               ;
;       Bits 3-0 = number of points in this sublist (1-16)              ;
;                                                                       ;
; The points are stored as a pair of nibbles:                           ;
;       Bits 7-4 = Y coordinate (?)                                     ;
;       Bits 3-0 = X coordinate (?)                                     ;
;                                                                       ;
; ENTRY DP = $D0                                                        ;
;       X-reg points to regular vector lists                            ;
;       Y-reg points to master vector list                              ;
;                                                                       ;
; EXIT: X-reg points to next byte after last regular vector list used   ;
;       Y-reg points to next byte after end of master vector list       ;
;                                                                       ;
;       D-reg trashed                                                   ;
;-----------------------------------------------------------------------;

Draw_Grid_VL:   LDA     ,Y+             ;Get flag byte
                BRA     LFFAB           ;Jump into loop

LFFA3:          JSR     Mov_Draw_VL_d   ;Draw a regular vector list
                LDA     $C880           ;Clear vector list flag
                ANDA    #$7F
LFFAB:          STA     $C880           ;Save flag byte for vector count
LFFAE:          DEC     $C880           ;Decrement vector count
                LDA     ,Y              ;Get Y of next point
                ASRA
                ANDA    #$F8
                LDB     ,Y+             ;Get X of next point
                ASLB
```

```
ASLB
ASLB
ASLB
ASRB
ANDB    #$F8
TST     $C880           ;Draw a regular vector list?
BMI     LFFA3           ;Go back if so
JSR     Draw_Line_d     ;Draw a line to the new point
LDA     $C880           ;Check vector counter
BITA    #$0F
BNE     LFFAE           ;Go back if more vectors to draw
BITA    #$20            ;Check for end of list
BEQ     Draw_Grid_VL    ;Go back if more lists to draw
RTS

FCC     "KARRSOFT82LDMCBCJT82LDMCBCJ"

FDB     0,0             ;Unused

FDB     $CBF2           ;SWI3 vector
FDB     $CBF2           ;SWI2 vector
FDB     $CBF5           ;FIRQ vector
FDB     $CBF8           ;IRQ vector
FDB     $CBFB           ;SWI vector
FDB     $CBFB           ;NMI vector
FDB     Start           ;Reset vector

END     Start
```