(back to project page)

Asteroids Disassembly

```
*******************************************************************************
* Asteroids (rev 4)                                                           *
* Copyright 1979 Atari, Inc.                                                  *
*                                                                             *
* By Lyle Rains and Ed Logg.                                                  *
*******************************************************************************
* Disassembly by Nick Mikstas (https://www.nicholasmikstas.com/asteroids-     *
* fimrware) and Lonnie Howell / Mark McDougall                                *
* (http://computerarcheology.com/Arcade/Asteroids/).                          *
*                                                                             *
* The binary used is a concatenation of the four ROMs that are addressable by *
* the 6502.  The DVG ROM was placed at the end.                               *
*******************************************************************************
* Project created by Andy McFadden, using 6502bench SourceGen v1.8.  This is a *
* fairly straight conversion from the listings on nicholasmikstas.com, which  *
* in turn drew heavily on the material at computerarcheology.com.  I have done *
* some reformatting and correction of typographical errors.                   *
*                                                                             *
* Last updated 2021/11/04                                                     *
*******************************************************************************
* General memory map:                                                         *
*                                                                             *
*   $0000-03ff: RAM (1KB)                                                     *
*   $2000-3fff: various I/O                                                   *
*   $4000-47ff: vector RAM                                                    *
*   $5000-57ff: vector ROM                                                    *
*   $6800-7fff: game ROM                                                      *
*                                                                             *
* The memory from $0200-02ff and $0300-03ff hold state for player 1 and player *
* 2.  The hardware allows the two to be swapped, so the current player's state *
* can always be found at $0200.                                               *
*                                                                             *
* I/O port definitions:                                                       *
*                                                                             *
* IN0/IN1 bits are mapped to the high bit of a unique address.  Some things   *
* aren't as simple.                                                           *
*                                                                             *
*  DSW1 ($2800-2803):                                                         *
*   $2800: coinage: 0=free play, 1=1 coin 2 credits, 2=1/1, 3=2/1             *
*   $2801: right coin mult: 0=1x, 1=4x, 2=5x, 3=6x                            *
*   $2802: center coin mult & lives (%00=1x/4, %01=1x/3, %10=2x/4, %11=2x/3)  *
*   $2803: language: 0=en, 1=de, 2=fr, 3=es                                   *
*                                                                             *
*  OUTLATCH ($3200): UURCLP12                                                 *
*   UU=unused                                                                 *
*   R=right coin counter enable                                               *
*   C=center coin counter enable                                              *
*   L=left coin counter enable                                                *
*   P=active player; swaps RAM at $200-2ff with $300-3ff                      *
*   1=player 1 button LED                                                     *
*   2=player 2 button LED                                                     *
*                                                                             *
* For a "cocktail" cabinet, the "active player" bit also rotates the screen   *
* 180 degrees when player 2 is active.                                        *
*******************************************************************************
PlyrText        .eq     $01    {const}          ;PLAYER
YrScrText       .eq     $02    {const}          ;YOUR SCORE IS ONE OF THE TEN BEST
InitText        .eq     $03    {const}          ;PLEASE ENTER YOUR INITIALS
PlyrLamps       .eq     $03    {const}          ;Illuminate both player button lamps.
LargeAst        .eq     $04    {const}          ;Large asteroid.
MpuRamPages     .eq     $04    {const}          ;four pages = 1k MPU RAM.
PshRtText       .eq     $04    {const}          ;PUSH ROTATE TO SELECT LETTER
RamSwap         .eq     $04    {const}          ;Swap RAM banks 2 and 3.
PshHypText      .eq     $05    {const}          ;PUSH HYPERSPACE WHEN LETTER IS CORRECT
PshStrtText     .eq     $06    {const}          ;PUSH START
CoinCtrLeft     .eq     $08    {const}          ;Enable left coin counter.
OneTwoText      .eq     $08    {const}          ;1 COIN 2 PLAYS
CoinCtrCntr     .eq     $10    {const}          ;Enable center coin counter.
GoodRamFreq     .eq     $14    {const}          ;Thump frequency setting for good RAM.
MaxAsteroids    .eq     $1a    {const}          ;Max number of asteroids(26+1 = 27).
ShipIndex       .eq     $1b    {const}          ;Index to ship status.
ScrIndex        .eq     $1c    {const}          ;Index to saucer status.
BadRamFreq      .eq     $1d    {const}          ;Thump frequency setting for bad RAM.
CoinCtrRght     .eq     $20    {const}          ;Enable right coin counter.
LargeScrPnts    .eq     $20    {const}          ;200 points for a large saucer hit.
SelfTestWait    .eq     $24    {const}          ;36 3Khz clock wait (.144 seconds).
LValidCoin      .eq     $74    {const}          ;Indicate left coin mechanism valid coin.
CValidCoin      .eq     $75    {const}          ;Indicate center coin mechanism valid coin.
RValidCoin      .eq     $76    {const}          ;Indicate right coin mechanism valid coin.
EnableBit       .eq     $80    {const}          ;The MSB is used to check/set hardware enables.
```

```
SmallScrPnts    .eq     $99     {const}         ;990 points for a small saucer hit.
LabsOpcode      .eq     $a0     {const}         ;LABS vector state machine opcode.
HaltOpcode      .eq     $b0     {const}         ;HALT vector state machine opcode.
JsrlOpcode      .eq     $c0     {const}         ;JSRL vector state machine opcode.
RtslOpcode      .eq     $d0     {const}         ;RTSL vector state machine opcode.
JmplOpcode      .eq     $e0     {const}         ;JMPL vector state machine opcode.
SvecOpcode      .eq     $f0     {const}         ;SVEC vector state machine opcode.
ZeroPageRam     .eq     $00     {addr/256}      ;Through $00FF.
VecRamPtr       .eq     $02     {addr/2}        ;Pointer to current vector RAM location.
TestSFXInit_    .eq     $09                     ;Used to start SFX during self test routines.
GenByte0B       .eq     $0b                     ;General use byte.
GenByte0C       .eq     $0c                     ;General use byte.
GenByte0D       .eq     $0d                     ;General use byte.
InitialIndex    .eq     $10                     ;Current index to initial in high scores being processed.
BCDAddress      .eq     $15                     ;Stored address of BCD data byte(can only be zero page address).
BCDIndex        .eq     $16                     ;Index to BCD data byte.
ShipDrawUnused  .eq     $17                     ;Always #$00. Was used for something in ship drawing routines.
ZeroBlankBypass .eq     $17                     ;Fag to determine if zero blanking should be overridden.
CurrentPlyr     .eq     $18                     ;Current active player. #$00=Player 1, #$01=Player 2.
ScoreIndex      .eq     $19                     ;Offset to current player's score registers.
PrevGamePlyrs   .eq     $1a                     ;Number of players in the game that just ended.
NumPlayers      .eq     $1c                     ;Indicates if there is 1 or 2 players.
HighScores      .eq     $1d     {addr/20}       ;Through $30. Top 10 high scores. 2 bytes each.
ThisInitial     .eq     $31                     ;Current initial selected on high score entry screen(0-2).
Plyr1Rank       .eq     $32                     ;Player 1 rank in top score list*3 (0,3,6,9, etc).
Plyr2Rank       .eq     $33                     ;Player 2 rank in top score list*3 (0,3,6,9, etc).
HighScoreIntls  .eq     $34     {addr/30}       ;Through $51. High score initials. 3 bytes each.
PlayerScores    .eq     $52     {addr/4}        ;Base address of the player's scores.
Plr1ScoreBase   .eq     $52                     ;Base address of Player 1's score.
Plr2ScoreBase   .eq     $54                     ;Base address of Player 2's score.
ShipsPerGame    .eq     $56                     ;Number of ships a player starts with.
Plyr1Ships      .eq     $57                     ;Current number of player 1 ships.
Plyr2Ships      .eq     $58                     ;Current number of player 2 ships.
HyprSpcFlag     .eq     $59                     ;#$00=No hyperspace, #$01=Jump successful, #$80=Jump unsuccessful.
PlyrDispTimer   .eq     $5a                     ;Timer to display Player 1/Player 2 between waves.
FrameCounter    .eq     $5b                     ;Increments every 4 NMIs. If game loop not running, causes reset.
FrameTimer      .eq     $5c     {addr/2}        ;16-bit timer increments every frame, lower byte.
NmiCounter      .eq     $5e                     ;Increments every NMI period.
RandNum         .eq     $5f     {addr/2}        ;High byte of random number word.
ShipDir         .eq     $61                     ;Player's ship direction.
ScrBulletDir    .eq     $62                     ;Saucer bullet direction.
ShipBulletSR    .eq     $63                     ;Shift register for limiting ship fire rate.
ShipXAccel      .eq     $64                     ;Ship acceleration in the X direction.
ShipYAccel      .eq     $65                     ;Ship acceleration in the Y direction.
SFXTimers       .eq     $66     {addr/6}        ;Starting address for SFX timers.
FireSFXTimer    .eq     $66                     ;Time to play fire SFX.
ScrFrSFXTimer   .eq     $67                     ;Time to play saucer fire SFX.
ExLfSFXTimer    .eq     $68                     ;Time to play extra life SFX.
ExplsnSFXTimer  .eq     $69                     ;Time to play explosion SFX.
ShipFireSFX_    .eq     $6a                     ;Controls the ship fire SFX.
ThisVolFreq     .eq     $6c                     ;Current settings for the thump frequency and volume.
ThmpOnTime      .eq     $6d                     ;Time thump SFX stays on.
ThumpOffTime    .eq     $6e                     ;Time thump SFX stays off.
MultiPurpBits   .eq     $6f                     ;Storage for bits to set in the MultiPurp register.
NumCredits      .eq     $70                     ;Current number of credits.
DipSwitchBits   .eq     $71                     ;Storage for dip switch values.
SlamTimer       .eq     $72                     ;Decrements from #$0F if slam detected during coin insertion.
CoinMult        .eq     $73                     ;Number of coins after multipliers.
ValidCoins      .eq     $74     {addr/3}        ;Base address for valid coin registers below.
WaitCoinTimers  .eq     $77     {addr/3}        ;Base address for timers below.
CoinDropTimers  .eq     $7a     {addr/3}        ;Base address for timers below.
ShpDebrisXVel   .eq     $7d     {addr/12}       ;Through $88. X velocity of ship debris pieces, lower byte.
ShpDebrisYVel   .eq     $89     {addr/12}       ;Through $94. Y velocity of ship debris pieces, lower byte.
OnePageRam      .eq     $0100   {addr/256}      ;Through $01FF. The stack resides here.
StackTop        .eq     $01d0                   ;The stack should never grow past this point.
StackBottom     .eq     $01ff                   ;The stack should never shrink to this point.
Player1Ram      .eq     $0200   {addr/256}      ;Through $02FF. A total of 1K MCU RAM.
AstStatus       .eq     $0200   {addr/27}       ;Through $021A. 17 asteroids max-their current status:
ShipStatus      .eq     $021b                   ;0=No Ship Or In Hyperspace, 1=Alive, $A0-$FF=Ship Exploding.
ScrStatus       .eq     $021c                   ;0=No Saucer, 1=Small Saucer, 2=Large Saucer, MSB set=Exploding.
ShpShotTimer    .eq     $021f   {addr/4}        ;Through $0222. Timers for current ship bullets.
AstXSpeed       .eq     $0223   {addr/27}       ;Through $023D. Asteroid horiz speed. 255-192=Left, 1-63=Right.
ShipXSpeed      .eq     $023e                   ;Ship horizontal speed.
SaucerXSpeed    .eq     $023f                   ;Saucer horizontal speed.
AstYSpeed       .eq     $0246   {addr/27}       ;Through $0260. Asteroid vert speed. 255-192=Down, 1-63=Up.
ShipYSpeed      .eq     $0261                   ;Ship vertical speed.
SaucerYSpeed    .eq     $0262                   ;Saucer vertical speed.
AstXPosHi       .eq     $0269                   ;Through $0283. Asteroid horz position, high byte.
ShipXPosHi      .eq     $0284                   ;Ship X position, high byte.
ScrXPosHi       .eq     $0285                   ;Saucer X position, high byte.
AstYPosHi       .eq     $028c                   ;Through $02A6. Asteroid vert position, high byte.
ShipYPosHi      .eq     $02a7                   ;Ship Y position, high byte.
ScrYPosHi       .eq     $02a8                   ;Saucer Y position, high byte.
AstXPosLo       .eq     $02af                   ;Through $02C9. Asteroid horz position, low byte.
ShipXPosLo      .eq     $02ca                   ;Ship X position, low byte.
ScrXPosLo       .eq     $02cb                   ;Saucer X position, low byte.
```

```
                AstYPosLo       .eq     $02d2                   ;Through $02EC. Asteroid vert position, low byte.
                ShipYPosLo      .eq     $02ed                   ;Ship Y position, low byte.
                ScrYPosLo       .eq     $02ee                   ;Saucer Y position, low byte.
                AstPerWave      .eq     $02f5                   ;Asteroids per wave.
                CurAsteroids    .eq     $02f6                   ;Current number of asteroids.
                ScrTimer        .eq     $02f7                   ;Countdown timer for saucer spawn.
                ScrTmrReload    .eq     $02f8                   ;Reload value for saucer timer.
                AstBreakTimer   .eq     $02f9                   ;Set after asteroid hit. Prevents saucer spawn after last asteroid.
                ShipSpawnTmr    .eq     $02fa                   ;Ship spawn timer. #$81=waiting to re-spawn.
                ThmpSpeedTmr    .eq     $02fb                   ;Timer That controls thump SFX speed.
                ThmpOffReload   .eq     $02fc                   ;Reload value for ThumpOffTime register.
                ScrSpeedup      .eq     $02fd                   ;Saucer occurrences increase if asteroid count is below this value.
                Player2Ram      .eq     $0300  {addr/256}       ;Through $03FF.
                Clk3Khz         .eq     $2001                   ;3KHz clock.
                Halt            .eq     $2002                   ;Halt gives the vector state machine status. 1=busy, 0=idle.
                HyprSpcSw       .eq     $2003                   ;Hyperspace button status.
                FireSw          .eq     $2004                   ;Fire button status.
                DiagStep        .eq     $2005                   ;Diagnostic step. Draws diagonal lines on screen.
                SlamSw          .eq     $2006                   ;Slam switch status.
                SelfTestSw      .eq     $2007                   ;Self test DIP switch status.
                LeftCoinSw      .eq     $2400                   ;Left coin switch status.
                Player1Sw       .eq     $2403                   ;Player 1 button status.
                Player2Sw       .eq     $2404                   ;Player 2 button status.
                ThrustSw        .eq     $2405                   ;Thrust button status.
                RotRghtSw       .eq     $2406                   ;Rotation right button status.
                RotLeftSw       .eq     $2407                   ;Rotation left button status.
                DipSw           .eq     $2800  {addr/3}         ;Base address for the DIP switches.
                PlayTypeSw      .eq     $2800                   ;Play type DIP switches (switches 7 and 8);
                RghtCoinMechSw  .eq     $2801                   ;Coin multiplier DIP switches for right coin mechanism.
                CentCMShipsSw   .eq     $2802                   ;Coin multiplier center coin mechanism, ships per play DIP switches.
                LanguageSw      .eq     $2803                   ;Language selection DIP switches.
                DmaGo           .eq     $3000                   ;Writing this address starts the vector state machine.
                MultiPurp       .eq     $3200                   ;Multipurpose write register. Below are the bit functions:
                WdClear         .eq     $3400                   ;Clears the watchdog timer.
                ExpPitchVol     .eq     $3600                   ;Controls the explosion SFX pitch and volume.
                ThumpFreqVol    .eq     $3a00                   ;Controls the thump frequency and volume.
                SaucerSFX       .eq     $3c00                   ;Controls the saucer sound.
                SaucerFireSFX   .eq     $3c01                   ;Controls the saucer fire SFX.
                SaucerSFXSel    .eq     $3c02                   ;Controls the frequency of the saucer SFX.
                ShipThrustSFX   .eq     $3c03                   ;Controls the ship thrust SFX.
                ShipFireSFX     .eq     $3c04                   ;Controls the ship fire SFX.
                LifeSFX         .eq     $3c05                   ;Controls the life SFX.
                NoiseReset      .eq     $3e00                   ;Resets the noise SFX.
                VectorRam       .eq     $4000  {addr/2048}      ;Through $47FF. A total of 2K of vector RAM.
                VectorRom       .eq     $5000  {addr/2048}      ;Through $57FF. A total of 2k of vector ROM.
                VectorRomEnd    .eq     $5800
                ProgramRom      .eq     $6800  {addr/6144}
                ProgramRomEnd   .eq     $8000

                ;
                ; Game entry point and main loop.
                ;
                                .addrs  $6800
6800: 4c f3 7c                  jmp     RESET                   ;Initialize the game after power-up

6803: 20 fa 6e  InitGame        jsr     SilenceSFX              ;Turn off all SFX
6806: 20 d8 6e                  jsr     InitGameVars            ;Initialize various game variables
6809: 20 68 71  InitWaves       jsr     InitWaveVars            ;Initialize variables for the current asteroid wave
                ;
680c: ad 07 20  GameRunningLoop lda     SelfTestSw              ;Get self test switch status
680f: 30 fe     :Spin           bmi     :Spin                   ;Is self test active? If so, spin lock until watchdog reset
6811: 46 5b                     lsr     FrameCounter            ;Has a new frame started?
6813: 90 f7                     bcc     GameRunningLoop         ;If not, no processing to do until next frame; branch to wait
                ;
6815: ad 02 20  VectorWaitLoop1 lda     Halt                    ;Is the vector state machine busy?
6818: 30 fb                     bmi     VectorWaitLoop1         ;If so, loop until it is idle
681a: ad 01 40                  lda     VectorRam+1             ;Swap which half of vector RAM is read and which half is
681d: 49 02                     eor     #$02                    ; written. This is done by alternating the jump instruction
681f: 8d 01 40                  sta     VectorRam+1             ; at the beginning of the RAM between $4402 and $4002.
6822: 8d 00 30                  sta     DmaGo                   ;Start the vector state machine
6825: 8d 00 34                  sta     WdClear                 ;Clear the watchdog timer
6828: e6 5c                     inc     FrameTimer
682a: d0 02                     bne     SetVecRamPtr            ;Increment frame counter
682c: e6 5d                     inc     FrameTimer+1
682e: a2 40     SetVecRamPtr    ldx     #>VectorRam             ;Is vector RAM pointer currently pointing at $4400 range
6830: 29 02                     and     #$02
6832: d0 02                     bne     UpdateVecRamPtr         ;If so, branch to switch to $4000
6834: a2 44                     ldx     #>VectorRam+$400        ;Prepare to switch to $4400
6836: a9 02     UpdateVecRamPtr lda     #$02
6838: 85 02                     sta     VecRamPtr               ;Swap vector RAM pointer
683a: 86 03                     stx     VecRamPtr+1
                ;
683c: 20 85 68                  jsr     ChkPreGameStuff         ;Check if non game play functions need to be run
683f: b0 c2                     bcs     InitGame                ;Branch if attract mode is starting
6841: 20 5c 76                  jsr     CheckHighScore          ;Check if player just got the high score
6844: 20 90 6d                  jsr     ChkHighScrMsg           ;Do high score and initial entry message if appropriate
```

```
6847: 10 1b                     bpl     DoScreenText            ;Is game not in progress? If not, branch
6849: 20 c4 73                  jsr     ChkHighScrList          ;Check if high score list needs to be displayed
684c: b0 16                     bcs     DoScreenText            ;Is high scores list being displayed? If so, branch
684e: a5 5a                     lda     PlyrDispTimer           ;Is player not active?
6850: d0 0c                     bne     DoAsteroids             ;If not, branch
                ;
6852: 20 d7 6c                  jsr     UpdateShip              ;Update ship firing and position
6855: 20 74 6e                  jsr     EnterHyperspc           ;Check if player entered hyperspace
6858: 20 3f 70                  jsr     ChkExitHprspc           ;Check if coming out of hyperspace
685b: 20 93 6b                  jsr     UpdateScr               ;Update saucer status
685e: 20 57 6f  DoAsteroids     jsr     UpdateObjects           ;Update objects(asteroids, ship, saucer and bullets)
6861: 20 f0 69                  jsr     HitDetection            ;Do hit detection calculations for all objects
6864: 20 4f 72  DoScreenText    jsr     UpdateScreenText        ;Update in-game screen text and reserve lives
6867: 20 55 75                  jsr     ChkUpdateSFX            ;Check if SFX needs to be updated
                ;
686a: a9 7f                     lda     #$7f                    ;X beam coordinate 4 * $7F = $1D0 = 464
686c: aa                        tax                             ;Y beam coordinate 4 * $7F = $1D0 = 464
686d: 20 03 7c                  jsr     MoveBeam                ;Move the CRT beam to a new location
6870: 20 b5 77                  jsr     GetRandNum              ;Get a random number
6873: 20 c0 7b                  jsr     VecHalt                 ;Halt the vector state machine
6876: ad fb 02                  lda     ThmpSpeedTmr
6879: f0 03                     beq     ChkGameRunning          ;Is thump speed timer running? If so, decrement it
687b: ce fb 02                  dec     ThmpSpeedTmr
687e: 0d f6 02  ChkGameRunning  ora     CurAsteroids            ;Is the game running?
6881: d0 89                     bne     GameRunningLoop         ;If so, branch to keep it going
6883: f0 84                     beq     InitWaves               ;Game not running, branch to initialize variables

                ;
                ; Helper routines.
                ;
6885: a5 1c     ChkPreGameStuff lda     NumPlayers              ;Are there players currently playing?
6887: f0 14                     beq     ChkCoinsPerCredit       ;If not, branch
6889: a5 5a                     lda     PlyrDispTimer           ;Is the Player 1/Player 2 message being displayed?
688b: d0 03                     bne     DecPlyrTimer            ;If so, branch to decrement timer
688d: 4c 60 69                  jmp     ChkThmpFaster           ;Jump to check if the Thump SFX should be sped up

6890: c6 5a     DecPlyrTimer    dec     PlyrDispTimer           ;Decrement player text display timer
6892: 20 e2 69                  jsr     DrawPlyrNum             ;Draw player number on the display
6895: 18        NoResetReturn1  clc                             ;Return without reinitializing the game
6896: 60                        rts

6897: a9 02     DoFreePlay      lda     #$02                    ;Load 2 credits always for free play
6899: 85 70                     sta     NumCredits
689b: d0 13                     bne     CheckCredits            ;Branch always

                ChkCoinsPerCredit
689d: a5 71                     lda     DipSwitchBits           ;Get the number of coins per credit
689f: 29 03                     and     #$03                    ;Is free play active?
68a1: f0 f4                     beq     DoFreePlay              ;If so, branch to add 2 credits
68a3: 18                        clc
68a4: 69 07                     adc     #OneTwoText-1           ;Prepare to display the coins per play message on the display
68a6: a8                        tay
68a7: a5 32                     lda     Plyr1Rank               ;Is this the high score entry part?
68a9: 25 33                     and     Plyr2Rank               ;If so, branch
68ab: 10 03                     bpl     CheckCredits
68ad: 20 f6 77                  jsr     WriteText               ;Write coins per play text to the display
                ;
68b0: a4 70     CheckCredits    ldy     NumCredits              ;Are credits available?
68b2: f0 e1                     beq     NoResetReturn1          ;If not, return
68b4: a2 01                     ldx     #$01                    ;Has the Player 1 button been pressed?
68b6: ad 03 24                  lda     Player1Sw
68b9: 30 23                     bmi     Do1Player               ;If so, branch to start 1 player game.
68bb: c0 02                     cpy     #$02                    ;Are there at least 2 credits available?
68bd: 90 7c                     bcc     ChkStartText            ;If not, branch to skip 2 players check
68bf: ad 04 24                  lda     Player2Sw               ;Is the Player 2 button being pressed?
68c2: 10 77                     bpl     ChkStartText            ;If not, branch
                ;
68c4: a5 6f                     lda     MultiPurpBits
68c6: 09 04                     ora     #RamSwap
68c8: 85 6f                     sta     MultiPurpBits           ;Switch to Player 2 RAM
68ca: 8d 00 32                  sta     MultiPurp
68cd: 20 d8 6e                  jsr     InitGameVars            ;Initialize various game variables
68d0: 20 68 71                  jsr     InitWaveVars            ;Initialize variables for the current asteroid wave
68d3: 20 e8 71                  jsr     CenterShip              ;Center ship on display and zero velocity.
68d6: a5 56                     lda     ShipsPerGame            ;Initialize the Player's lives
68d8: 85 58                     sta     Plyr2Ships
68da: a2 02                     ldx     #$02                    ;Indicate this is a 2 player game
68dc: c6 70                     dec     NumCredits              ;Decrement credits for Player 2
                ;
68de: 86 1c     Do1Player       stx     NumPlayers              ;Store number of players this game (1 or 2)
68e0: c6 70                     dec     NumCredits              ;Decrement credits for player 1
68e2: a5 6f                     lda     MultiPurpBits           ;Clear Player 1 and 2 LEDs and RAM swap bit
68e4: 29 f8                     and     #%11111000
68e6: 45 1c                     eor     NumPlayers              ;Turn on LEDs indicating 1 or 2 player game
68e8: 85 6f                     sta     MultiPurpBits
68ea: 8d 00 32                  sta     MultiPurp
```

```
                ;
68ed: 20 e8 71                  jsr     CenterShip              ;Center ship on display and zero velocity
68f0: a9 01                     lda     #$01
68f2: 8d fa 02                  sta     ShipSpawnTmr            ;Initialize ship spawn timer for both players
68f5: 8d fa 03                  sta     ShipSpawnTmr+$100
68f8: a9 92                     lda     #$92
68fa: 8d f8 02                  sta     ScrTmrReload
68fd: 8d f8 03                  sta     ScrTmrReload+$100       ;Initialize saucer timer and reload value
6900: 8d f7 03                  sta     ScrTimer+$100
6903: 8d f7 02                  sta     ScrTimer
6906: a9 7f                     lda     #$7f
6908: 8d fb 02                  sta     ThmpSpeedTmr            ;Initialize thump speed timer for both players
690b: 8d fb 03                  sta     ThmpSpeedTmr+$100
690e: a9 05                     lda     #$05
6910: 8d fd 02                  sta     ScrSpeedup              ;Load initial asteroid count that causes more frequent saucers
6913: 8d fd 03                  sta     ScrSpeedup+$100
6916: a9 ff                     lda     #$ff
6918: 85 32                     sta     Plyr1Rank               ;Zero out both Player's rank
691a: 85 33                     sta     Plyr2Rank
691c: a9 80                     lda     #$80                    ;Load time for displaying Player 1/2
691e: 85 5a                     sta     PlyrDispTimer
6920: 0a                        asl     A
6921: 85 18                     sta     CurrentPlyr             ;Set current player to 1 and the score index for Player 1
6923: 85 19                     sta     ScoreIndex
6925: a5 56                     lda     ShipsPerGame            ;Set Player 1 reserve lives
6927: 85 57                     sta     Plyr1Ships
6929: a9 04                     lda     #$04
692b: 85 6c                     sta     ThisVolFreq
692d: 85 6e                     sta     ThumpOffTime            ;Set initial thump SFX values
692f: a9 30                     lda     #$30
6931: 8d fc 02                  sta     ThmpOffReload
6934: 8d fc 03                  sta     ThmpOffReload+$100
6937: 8d 00 3e                  sta     NoiseReset              ;Reset the noise SFX hardware
693a: 60                        rts

693b: a5 32     ChkStartText    lda     Plyr1Rank               ;Is this the high score entry part?
693d: 25 32                     and     Plyr1Rank
693f: 10 0b                     bpl     CheckUpdateLeds         ;If so, branch to move on
6941: a5 5c                     lda     FrameTimer              ;Is it time to display "PUSH START" on the display?
6943: 29 20                     and     #%00100000
6945: d0 05                     bne     CheckUpdateLeds         ;If not, branch
6947: a0 06                     ldy     #PshStrtText            ;Display "PUSH START"
6949: 20 f6 77                  jsr     WriteText               ;Write text to the display
                ;
694c: a5 5c     CheckUpdateLeds lda     FrameTimer              ;Update LEDs every 16 frames
694e: 29 0f                     and     #$0f                    ;Is this the 16th frame?
6950: d0 0c                     bne     NoResetReturn2          ;If not, branch to return from function
                ;
6952: a9 01                     lda     #$01
6954: c5 70                     cmp     NumCredits
6956: 69 01                     adc     #$01                    ;Turn on Player 1/Player 2 button LEDs
6958: 49 01                     eor     #$01
695a: 45 6f                     eor     MultiPurpBits
695c: 85 6f                     sta     MultiPurpBits
695e: 18        NoResetReturn2  clc                             ;Return without reinitializing the game
695f: 60                        rts

6960: a5 5c     ChkThmpFaster   lda     FrameTimer              ;Is it time to speed up the thump SFX?
6962: 29 3f                     and     #$3f
6964: d0 0a                     bne     ChkMoreShips            ;If not, branch
6966: ad fc 02                  lda     ThmpOffReload           ;Is the thump SFX at max speed?
6969: c9 08                     cmp     #$08
696b: f0 03                     beq     ChkMoreShips            ;If so, branch
696d: ce fc 02                  dec     ThmpOffReload           ;Speed up thump SFX by decreasing off time
                ;
6970: a6 18     ChkMoreShips    ldx     CurrentPlyr             ;Does the player have any ship remaining?
6972: b5 57                     lda     Plyr1Ships,x
6974: d0 1c                     bne     ChkShipStatus           ;If so, branch
6976: ad 1f 02                  lda     ShpShotTimer            ;Are there any ship bullets on the display?
6979: 0d 20 02                  ora     ShpShotTimer+1
697c: 0d 21 02                  ora     ShpShotTimer+2
697f: 0d 22 02                  ora     ShpShotTimer+3
6982: d0 0e                     bne     ChkShipStatus           ;If so, branch
                ;
6984: a0 07                     ldy     #$07                    ;Write "GAME OVER" to the display
6986: 20 f6 77                  jsr     WriteText               ;Write text to the display
6989: a5 1c                     lda     NumPlayers              ;Is this a 2 player game?
698b: c9 02                     cmp     #$02
698d: 90 03                     bcc     ChkShipStatus           ;If not, branch
698f: 20 e2 69                  jsr     DrawPlyrNum             ;Draw player number on the display
6992: ad 1b 02  ChkShipStatus   lda     ShipStatus              ;Does a ship still exist on the display?
6995: d0 36                     bne     NoResetReturn3          ;If so, branch to exit
6997: ad fa 02                  lda     ShipSpawnTmr            ;Is ship about to re-spawn?
699a: c9 80                     cmp     #$80
699c: d0 2f                     bne     NoResetReturn3          ;If so, branch to exit
699e: a9 10                     lda     #$10                    ;Start ship re-spawn timer
```

```
69a0: 8d fa 02                  sta     ShipSpawnTmr
69a3: a6 1c                     ldx     NumPlayers              ;Get number of players
69a5: a5 57                     lda     Plyr1Ships              ;Are there any ships in Player 1 or 2 reserves?
69a7: 05 58                     ora     Plyr2Ships
69a9: f0 24                     beq     NoCurntGame             ;If not, branch; a game is not currently being played
69ab: 20 2d 70                  jsr     SaucerReset             ;Reset saucer variables
69ae: ca                        dex                             ;Is this a 1 player game?
69af: f0 1c                     beq     NoResetReturn3          ;If so, branch to exit
69b1: a9 80                     lda     #$80                    ;Load player display timer for Player 2
69b3: 85 5a                     sta     PlyrDispTimer
                        ;
69b5: a5 18                     lda     CurrentPlyr             ;Change to next player
69b7: 49 01                     eor     #$01
69b9: aa                        tax                             ;Does the new player have any lives remaining?
69ba: b5 57                     lda     Plyr1Ships,x
69bc: f0 0f                     beq     NoResetReturn3          ;If not, branch to exit
69be: 86 18                     stx     CurrentPlyr
69c0: a9 04                     lda     #$04
69c2: 45 6f                     eor     MultiPurpBits           ;RAM swap to new player RAM
69c4: 85 6f                     sta     MultiPurpBits
69c6: 8d 00 32                  sta     MultiPurp
69c9: 8a                        txa
69ca: 0a                        asl     A                       ;Get index to new player score
69cb: 85 19                     sta     ScoreIndex
69cd: 18            NoResetReturn3  clc                         ;Return without reinitializing the game
69ce: 60                        rts

69cf: 86 1a         NoCurntGame     stx     PrevGamePlyrs       ;Keep track of any previous players
69d1: a9 ff                     lda     #$ff                    ;Set no current players
69d3: 85 1c                     sta     NumPlayers
69d5: 20 fa 6e                  jsr     SilenceSFX              ;Turn off all SFX
69d8: a5 6f                     lda     MultiPurpBits
69da: 29 f8                     and     #%11111000              ;Turn on both player button LEDs
69dc: 09 03                     ora     #PlyrLamps
69de: 85 6f                     sta     MultiPurpBits
69e0: 18                        clc                             ;Return without reinitializing the game
69e1: 60                        rts

69e2: a0 01         DrawPlyrNum     ldy     #PlyrText           ;Prepare to write "PLAYER" on the display
69e4: 20 f6 77                  jsr     WriteText               ;Write text to the display
69e7: a4 18                     ldy     CurrentPlyr             ;Get the current player number
69e9: c8                        iny                             ;Set it to the proper index for drawing
69ea: 98                        tya
69eb: 20 d1 7b                  jsr     DrawDigit               ;Draw a single digit on the display
69ee: 60                        rts

69ef: 62                        .dd1    $62                     ;checksum byte

                        ;
                        ; Hit detection.
                        ;
                        ; Need to check hit detection between all the on-screen objects.  Object 1 is
                        ; either a bullet (from either the ship or saucer), the player's ship, or the
                        ; saucer.  Object 2 is either an asteroid, the player's ship or the saucer.
                        ; Object 1 is the outer loop of the check and each object 1 is checked against
                        ; all of the object 2s.  The hit box value extends in both the positive and
                        ; negative directions in both the X and Y directions.
                        ;
                        • Clear variables
                        ]ObjXDiff       .var    $08     {addr/1}
                        ]ObjYDiff       .var    $09     {addr/1}
                        ]Obj2Status     .var    $0b     {addr/1}

69f0: a2 07         HitDetection    ldx     #$07                ;Prepare to check hit detection on bullets, ship and saucer
69f2: bd 1b 02      HitDetObj1Loop  lda     ShipStatus,x        ;Is the current object 1 slot active?
69f5: f0 02                     beq     HitDetNextObj1          ;If not, branch to increment to the next object 1
69f7: 10 04                     bpl     HitDetObj2              ;If MSB clear, this object needs hit detection, branch if so
69f9: ca            HitDetNextObj1  dex                         ;Move to next object to check
69fa: 10 f6                     bpl     HitDetObj1Loop          ;More object to check? if so, branch to do another
69fc: 60                        rts                             ;Done checking hit detection, exit

69fd: a0 1c         HitDetObj2      ldy     #28                 ;Prepare to check hits against asteroids, ship and saucer
69ff: e0 04                     cpx     #$04                    ;Are we checking hit detection against a ship bullet?
6a01: b0 07                     bcs     HitDetObj2Loop          ;If so, branch to check hit detection
6a03: 88                        dey                             ;Skip checking object 2 as a saucer, will be checked as object 1
6a04: 8a                        txa                             ;Is object 1 not the player's ship?
6a05: d0 03                     bne     HitDetObj2Loop          ;If not, branch to do hit detection
6a07: 88            HitDetNextObj2  dey                         ;Have we checked all the object 2s?
6a08: 30 ef                     bmi     HitDetNextObj1          ;If so, branch to increment to the next object 1
                        ;
6a0a: b9 00 02      HitDetObj2Loop  lda     AstStatus,y         ;Is the current object 2 slot active?
6a0d: f0 f8                     beq     HitDetNextObj2          ;If not, branch to increment to the next object 2
6a0f: 30 f6                     bmi     HitDetNextObj2          ;If MSB clear, this object needs hit detection, branch if not
6a11: 85 0b                     sta     ]Obj2Status             ;Store a copy of object 2's current status
                        ;
6a13: b9 af 02                  lda     AstXPosLo,y
```

```
6a16: 38                        sec                         ;Subtract objects 1 and 2 lower byte of
6a17: fd ca 02                  sbc     ShipXPosLo,x        ; their X positions and save the result
6a1a: 85 08                     sta     ]ObjXDiff
6a1c: b9 69 02                  lda     AstXPosHi,y         ;Subtract objects 1 and 2 upper byte of their X positions
6a1f: fd 84 02                  sbc     ShipXPosHi,x
6a22: 4a                        lsr     A                   ;Keep bit 8 in the difference. XDiff holds bits 8 to 1
6a23: 66 08                     ror     ]ObjXDiff
6a25: 0a                        asl     A                   ;Is the MSB of the positions the same?
6a26: f0 0c                     beq     CalcObjYDiff        ;If so, possible hit. Branch to calculate the Y difference
6a28: 10 6d                     bpl     HitDetNextObj2_     ;Distance too great, no chance of a hit; move to next object
6a2a: 49 fe                     eor     #%11111110          ;Negative value calculated; get ABS
6a2c: d0 69                     bne     HitDetNextObj2_     ;Is MSB the same? IF not, distance too great; move to next object
6a2e: a5 08                     lda     ]ObjXDiff           ;Need to convert XDiff to ABS since its negative
6a30: 49 ff                     eor     #$ff                ;Perform 1s compliment; it is now its ABS value-1
6a32: 85 08                     sta     ]ObjXDiff
                        ;
6a34: b9 d2 02          CalcObjYDiff    lda     AstYPosLo,y
6a37: 38                        sec                         ;Subtract objects 1 and 2 lower byte of
6a38: fd ed 02                  sbc     ShipYPosLo,x        ; their X positions and save the result
6a3b: 85 09                     sta     ]ObjYDiff
6a3d: b9 8c 02                  lda     AstYPosHi,y         ;Subtract objects 1 and 2 upper byte of their Y positions
6a40: fd a7 02                  sbc     ShipYPosHi,x
6a43: 4a                        lsr     A                   ;Keep bit 8 in the difference; YDiff holds bits 8 to 1
6a44: 66 09                     ror     ]ObjYDiff
6a46: 0a                        asl     A                   ;Is the MSB of the positions the same?
6a47: f0 0c                     beq     HitDetPart2         ;If so, possible hit; branch to calculate further
6a49: 10 4c                     bpl     HitDetNextObj2_     ;Distance too great; no chance of a hit; move to next object
6a4b: 49 fe                     eor     #%11111110          ;Negative value calculated; get ABS
6a4d: d0 48                     bne     HitDetNextObj2_     ;Is MSB the same? IF not, distance too great; move to next object
6a4f: a5 09                     lda     ]ObjYDiff           ;Need to convert YDiff to ABS since its negative
6a51: 49 ff                     eor     #$ff                ;Perform 1s compliment.  It is now its ABS value-1.
6a53: 85 09                     sta     ]ObjYDiff
                        ;
6a55: a9 2a             HitDetPart2     lda     #42         ;Small asteroid hit box 42 X 42 from center
6a57: 46 0b                     lsr     ]Obj2Status         ;Is this a small asteroid, ship or saucer?
6a59: b0 08                     bcs     HitDetShip          ;If so, branch
6a5b: a9 48                     lda     #72                 ;Medium asteroid hit box 72 X 72 from center
6a5d: 46 0b                     lsr     ]Obj2Status         ;Is this a medium asteroid or a saucer?
6a5f: b0 02                     bcs     HitDetShip          ;If so, branch
6a61: a9 84                     lda     #132                ;Large asteroid hit box 132 X 132 from center
                        ;
6a63: e0 01             HitDetShip      cpx     #$01        ;Is object 1 not the player's ship?
6a65: b0 02                     bcs     HitDetSaucer        ;If not, branch.
6a67: 69 1c                     adc     #28                 ;Ship hit box 42+28 = 70 X 70 from center
6a69: d0 0c             HitDetSaucer    bne     CheckObjHit ;Is object a saucer? If not, branch
6a6b: 69 12                     adc     #18                 ;Small saucer hit box 42+18 = 60 X 60 from center
6a6d: ae 1c 02                  ldx     ScrStatus
6a70: ca                        dex                         ;Is the object a small saucer?
6a71: f0 02                     beq     HitDetFinishScr     ;If so, branch
6a73: 69 12                     adc     #18                 ;Large saucer hit box 42+18+18 = 78 X 78 from center
6a75: a2 01             HitDetFinishScr ldx     #$01        ;Reload object 1 as a saucer
                        ;
                        ]ObjHitBox      .var    $0b    {addr/1}

6a77: c5 08             CheckObjHit     cmp     ]ObjXDiff   ;s object 1 X difference smaller than the hit box?
6a79: 90 1c                     bcc     HitDetNextObj2_     ;If not, no hit detected. Branch to check next object
6a7b: c5 09                     cmp     ]ObjYDiff           ;Is object 1 Y difference smaller than the hit box?
6a7d: 90 18                     bcc     HitDetNextObj2_     ;If not, no hit detected. Branch to check next object
6a7f: 85 0b                     sta     ]ObjHitBox          ;Store hit box value
6a81: 4a                        lsr     A                   ;/2
6a82: 18                        clc                         ;Add two hit box values together
6a83: 65 0b                     adc     ]ObjHitBox          ;Hit box value is now 1.5 X value set above, about sqrt(2)
6a85: 85 0b                     sta     ]ObjHitBox          ;This has the effect of making the hit box more circular
6a87: a5 09                     lda     ]ObjYDiff           ;Add the two difference values together
6a89: 65 08                     adc     ]ObjXDiff           ;If it causes a carry, The distance is too great
6a8b: b0 0a                     bcs     HitDetNextObj2_     ;Branch to move to next object
6a8d: c5 0b                     cmp     ]ObjHitBox          ;Is combined difference values grater than the hit box?
6a8f: b0 06                     bcs     HitDetNextObj2_     ;If so, branch to move to the next object
6a91: 20 0f 6b                  jsr     DoObjHit            ;Update object that got hit.
6a94: 4c f9 69          HitDetNextObj1_ jmp     HitDetNextObj1  ;Check next object 1 for a hit

6a97: 88                HitDetNextObj2_ dey                 ;Are there more object 2s to check?
6a98: 30 fa                     bmi     HitDetNextObj1_     ;If not, branch to move to the next object 1
6a9a: 4c 0a 6a                  jmp     HitDetObj2Loop      ;Check next object 2 for a hit

                        ;
                        ; Asteroid update routine.
                        ;
                        ]GenByte08      .var    $08    {addr/1}

6a9d: b9 00 02          UpdateAsteroid  lda     AstStatus,y
6aa0: 29 07                     and     #%00000111          ;Save current asteroid size.
6aa2: 85 08                     sta     ]GenByte08
6aa4: 20 b5 77                  jsr     GetRandNum          ;Get a random number
6aa7: 29 18                     and     #%00011000          ;Use it to set the asteroid type.
6aa9: 05 08                     ora     ]GenByte08
```

```
6aab: 9d 00 02                  sta     AstStatus,x             ;Save asteroid size and type.
                        ;
6aae: b9 af 02                  lda     AstXPosLo,y
6ab1: 9d af 02                  sta     AstXPosLo,x             ;Save asteroid X position
6ab4: b9 69 02                  lda     AstXPosHi,y
6ab7: 9d 69 02                  sta     AstXPosHi,x
6aba: b9 d2 02                  lda     AstYPosLo,y
6abd: 9d d2 02                  sta     AstYPosLo,x             ;Save asteroid Y position
6ac0: b9 8c 02                  lda     AstYPosHi,y
6ac3: 9d 8c 02                  sta     AstYPosHi,x
6ac6: b9 23 02                  lda     AstXSpeed,y
6ac9: 9d 23 02                  sta     AstXSpeed,x             ;Save asteroid velocity
6acc: b9 46 02                  lda     AstYSpeed,y
6acf: 9d 46 02                  sta     AstYSpeed,x
6ad2: 60                        rts

                        ;
                        ; Ship draw routine.
                        ;
                        • Clear variables
                        ]ShipDrawXInv   .var    $08     {addr/1}
                        ]ShipDrawYInv   .var    $09     {addr/1}
                        ]VecPtr         .var    $0b     {addr/2}

6ad3: 85 0b             DrawShip        sta     ]VecPtr                 ;Save the pointer to the ship vector data
6ad5: 86 0c                             stx     ]VecPtr+1
6ad7: a0 00             SetVecRAMData   ldy     #$00                    ;Start at beginning of vector data
6ad9: c8                GetShipOpCode   iny                             ;Get opcode byte from vector ROM
6ada: b1 0b                             lda     (]VecPtr),y
6adc: 45 09                             eor     ]ShipDrawYInv
6ade: 91 02                             sta     (VecRamPtr),y           ;Invert Y axis of VCTR data, if necessary
6ae0: 88                                dey
6ae1: c9 f0                             cmp     #SvecOpcode             ;Is this a SVEC vector opcode?
6ae3: b0 1e                             bcs     DrawShipSVEC            ;If so, branch to get the next SVEC byte
6ae5: c9 a0                             cmp     #LabsOpcode             ;Is this a LABS opcode?
6ae7: b0 16                             bcs     DrawShipRTSL            ;If not, branch because it must be an RTSL opcode
                        ;
6ae9: b1 0b                             lda     (]VecPtr),y             ;Load second byte of VCTR data into vector RAM
6aeb: 91 02                             sta     (VecRamPtr),y
6aed: c8                                iny
6aee: c8                                iny                             ;Move to 3rd byte of VCTR data and store in vector RAM
6aef: b1 0b                             lda     (]VecPtr),y
6af1: 91 02                             sta     (VecRamPtr),y
6af3: c8                                iny
6af4: b1 0b                             lda     (]VecPtr),y             ;Move to 4th byte of VCTR data
6af6: 45 08                             eor     ]ShipDrawXInv           ;Invert X axis of VCTR data, if necessary
6af8: 65 17                             adc     ShipDrawUnused
6afa: 91 02                             sta     (VecRamPtr),y           ;Store 4th byte in vector RAM
6afc: c8                NextShipOpCode  iny
6afd: d0 da                             bne     GetShipOpCode           ;Branch always

6aff: 88                DrawShipRTSL    dey                             ;Done with this segment of ship vector data
6b00: 4c 39 7c                          jmp     VecPtrUpdate            ;Update Vector RAM pointer

6b03: b1 0b             DrawShipSVEC    lda     (]VecPtr),y             ;Load second byte of SVEC data into vector RAM
6b05: 45 08                             eor     ]ShipDrawXInv           ;Invert X axis of SVEC data, if necessary
6b07: 18                                clc
6b08: 65 17                             adc     ShipDrawUnused
6b0a: 91 02                             sta     (VecRamPtr),y
6b0c: c8                                iny                             ;Branch always
6b0d: d0 ed                             bne     NextShipOpCode

                        ;
                        ; Update hit object.
                        ;
6b0f: e0 01             DoObjHit        cpx     #$01                    ;Is object 1 that hit object 2 a saucer?
6b11: d0 08                             bne     ChkObj1Ship             ;If not, branch
6b13: c0 1b                             cpy     #ShipIndex              ;Is object 2 that got hit the ship?
6b15: d0 12                             bne     ObjExplode              ;If not, branch
6b17: a2 00                             ldx     #$00                    ;Set object 1 as the ship
6b19: a0 1c                             ldy     #ScrIndex               ;Set object 2 as the saucer
6b1b: 8a                ChkObj1Ship     txa                             ;Is object 1 the ship?
6b1c: d0 1e                             bne     ClearObjRAM             ;If not, branch
                        ;
6b1e: a9 81                             lda     #$81                    ;Indicate the ship is waiting to re-spawn
6b20: 8d fa 02                          sta     ShipSpawnTmr
6b23: a6 18                             ldx     CurrentPlyr             ;Remove a life from the current player
6b25: d6 57                             dec     Plyr1Ships,x
6b27: a2 00                             ldx     #$00                    ;Indicate ship is object 1
                        ;
6b29: a9 a0             ObjExplode      lda     #$a0                    ;Indicate object is exploding
6b2b: 9d 1b 02                          sta     ShipStatus,x
6b2e: a9 00                             lda     #$00
6b30: 9d 3e 02                          sta     ShipXSpeed,x            ;Zero out the ship's velocity
6b33: 9d 61 02                          sta     ShipYSpeed,x
6b36: c0 1b                             cpy     #$1b                    ;Is object 2 an asteroid?
```

```
6b38: 90 0d                     bcc     ObjAsteroid             ;If so, branch
6b3a: b0 37                     bcs     SaucerHit               ;Must have been a saucer hit; branch always

6b3c: a9 00     ClearObjRAM     lda     #$00                    ;Remove the hit object from RAM
6b3e: 9d 1b 02                  sta     ShipStatus,x
6b41: c0 1b                     cpy     #ShipIndex              ;Is object 2 the ship?
6b43: f0 21                     beq     Obj2ShipHit             ;If so, branch
6b45: b0 2c                     bcs     SaucerHit               ;Was object 2 a saucer? If so, branch
                ;
6b47: 20 ec 75  ObjAsteroid     jsr     BreakAsteroid           ;Break down a hit asteroid
6b4a: b9 00 02  ObjHitSFX       lda     AstStatus,y             ;Change length of hit SFX based on object size
6b4d: 29 03                     and     #%00000011
6b4f: 49 02                     eor     #%00000010
6b51: 4a                        lsr     A
6b52: 6a                        ror     A
6b53: 6a                        ror     A
6b54: 09 3f                     ora     #$3f                    ;Set hit SFX minimum time
6b56: 85 69                     sta     ExplsnSFXTimer          ;Set hit SFX time
6b58: a9 a0                     lda     #$a0                    ;Indicate object is exploding
6b5a: 99 00 02                  sta     AstStatus,y
6b5d: a9 00                     lda     #$00
6b5f: 99 23 02                  sta     AstXSpeed,y             ;Zero out object velocity.
6b62: 99 46 02                  sta     AstYSpeed,y
6b65: 60                        rts

6b66: 8a        Obj2ShipHit     txa                             ;Get index to current player's reserve ships
6b67: a6 18                     ldx     CurrentPlyr
6b69: d6 57                     dec     Plyr1Ships,x            ;Remove a life from the current player
6b6b: aa                        tax
6b6c: a9 81                     lda     #$81                    ;Indicate the ship is waiting to re-spawn
6b6e: 8d fa 02                  sta     ShipSpawnTmr
6b71: d0 d7                     bne     ObjHitSFX               ;Branch always

6b73: ad f8 02  SaucerHit       lda     ScrTmrReload            ;Reset the saucer timer.
6b76: 8d f7 02                  sta     ScrTimer
6b79: a5 1c                     lda     NumPlayers              ;Is someone playing the game?
6b7b: f0 cd                     beq     ObjHitSFX               ;If not, branch to skip updating score
6b7d: 86 0d                     stx     GenByte0D               ;Save object 1 index
6b7f: a6 19                     ldx     ScoreIndex              ;Get index to current player's score
6b81: ad 1c 02                  lda     ScrStatus               ;Check to see if a small saucer was hit
6b84: 4a                        lsr     A
6b85: a9 99                     lda     #SmallScrPnts           ;Prepare to add small saucer points to score
6b87: b0 02                     bcs     AddSaucerPoints         ;Was a small saucer hit? If so, branch
6b89: a9 20                     lda     #LargeScrPnts           ;A large saucer was hit. Load large saucer points.
6b8b: 20 97 73  AddSaucerPoints jsr     UpdateScore             ;Add points to the current player's score
6b8e: a6 0d                     ldx     GenByte0D               ;Restore object 1 index
6b90: 4c 4a 6b                  jmp     ObjHitSFX               ;Set SFX for object being hit based on object size

                ;
                ; Update saucer routines.
                ;
                • Clear variables
                ]GenByte08      .var    $08    {addr/1}

6b93: a5 5c     UpdateScr       lda     FrameTimer              ;Update saucers only every 4th frame
6b95: 29 03                     and     #$03                    ;Is this the 4th frame?
6b97: f0 01                     beq     ChkScrExplode           ;If so, branch to continue processing
6b99: 60        EndUpdateScr    rts                             ;End update saucer routines

6b9a: ad 1c 02  ChkScrExplode   lda     ScrStatus               ;Is the saucer currently exploding?
6b9d: 30 fa                     bmi     EndUpdateScr            ;If so, branch to exit
6b9f: f0 03                     beq     DoScrTimers             ;Is no saucer active? if so, branch to update saucer timers
6ba1: 4c 34 6c                  jmp     ScrYVelocity            ;Saucer active. Update saucer Y velocity

6ba4: a5 1c     DoScrTimers     lda     NumPlayers              ;Is a game currently being played?
6ba6: f0 07                     beq     DoScrTmrUpdate          ;If not, branch to continue
6ba8: ad 1b 02                  lda     ShipStatus              ;Is the player's ship exploding or in hyperspace?
6bab: f0 ec                     beq     EndUpdateScr            ;If so, branch to exit saucer update routines
6bad: 30 ea                     bmi     EndUpdateScr
6baf: ad f9 02  DoScrTmrUpdate  lda     AstBreakTimer           ;Was an asteroid just hit?
6bb2: f0 03                     beq     UpdateScrTimer          ;If not, branch to update saucer timer
6bb4: ce f9 02                  dec     AstBreakTimer           ;Decrement asteroid hit timer
6bb7: ce f7 02  UpdateScrTimer  dec     ScrTimer                ;Is it time to re-spawn a saucer?
6bba: d0 dd                     bne     EndUpdateScr            ;If not, branch to exit
                ;
6bbc: a9 01                     lda     #$01                    ;Time to re-spawn a saucer. set timer just above 0
6bbe: 8d f7 02                  sta     ScrTimer                ; Just in case another factor keeps it from spawning
6bc1: ad f9 02                  lda     AstBreakTimer           ;Was an asteroid just hit?
6bc4: f0 0a                     beq     GenNewSaucer            ;If not, branch to spawn a saucer
6bc6: ad f6 02                  lda     CurAsteroids            ;If an asteroid was just hit and it was the last asteroid,
6bc9: f0 ce                     beq     EndUpdateScr            ; branch to end function; no saucer spawn on an empty screen
6bcb: cd fd 02                  cmp     ScrSpeedup              ;Has the asteroid number hit the saucer spawn speedup threshold?
6bce: b0 c9                     bcs     EndUpdateScr            ;If not, branch to end.
                ;
6bd0: ad f8 02  GenNewSaucer    lda     ScrTmrReload
6bd3: 38                        sec                             ;Saucer spawn speedup threshold hit. decrement saucer timer by 6
```

```
6bd4: e9 06                     sbc     #$06
6bd6: c9 20                     cmp     #32                     ;Is spawn timer below minimum value of 32?
6bd8: 90 03                     bcc     InitNewSaucer           ;If so, branch to initialize the new saucer
6bda: 8d f8 02                  sta     ScrTmrReload            ;Maintain a minimum saucer spawn timer
                        ;
6bdd: a9 00             InitNewSaucer   lda     #$00
6bdf: 8d cb 02                  sta     ScrXPosLo               ;Start saucer at left edge of the display
6be2: 8d 85 02                  sta     ScrXPosHi
6be5: 20 b5 77                  jsr     GetRandNum              ;Get a random number
6be8: 4a                        lsr     A
6be9: 6e ee 02                  ror     ScrYPosLo
6bec: 4a                        lsr     A                       ;Use three of the random bits to set the saucer Y position
6bed: 6e ee 02                  ror     ScrYPosLo
6bf0: 4a                        lsr     A
6bf1: 6e ee 02                  ror     ScrYPosLo
6bf4: c9 18                     cmp     #$18                    ;Is remaining random bits greater than limit?
6bf6: 90 02                     bcc     SetScrYPosHi            ;If not, branch
6bf8: 29 17                     and     #$17                    ;Limit max Y position high byte.
6bfa: 8d a8 02          SetScrYPosHi    sta     ScrYPosHi       ;Set high byte of saucer Y starting position
6bfd: a2 10                     ldx     #$10                    ;Randomly set saucer X movement direction
6bff: 24 60                     bit     RandNum+1               ;Is saucer moving from left to right?
6c01: 70 0c                     bvs     ScrXVelocity            ;If so, branch
6c03: a9 1f                     lda     #$1f
6c05: 8d 85 02                  sta     ScrXPosHi               ;Start saucer at right edge of the display
6c08: a9 ff                     lda     #$ff
6c0a: 8d cb 02                  sta     ScrXPosLo
6c0d: a2 f0                     ldx     #$f0                    ;Set saucer X velocity for a negative direction(right to left)
6c0f: 8e 3f 02          ScrXVelocity    stx     SaucerXSpeed    ;Save final saucer X velocity
                        ;
6c12: a2 02                     ldx     #$02                    ;Prepare to make a large saucer
6c14: ad f8 02                  lda     ScrTmrReload            ;Is it still early in the asteroid wave?
6c17: 30 17                     bmi     SetScrStatus            ;If so, branch to create a large saucer
6c19: a4 19                     ldy     ScoreIndex              ;Is the player's score above 3000?
6c1b: b9 53 00                  lda     PlayerScores+1,y
6c1e: c9 30                     cmp     #$30
6c20: b0 0d                     bcs     SetSmallScr             ;If so, branch to create a small saucer
6c22: 20 b5 77                  jsr     GetRandNum              ;Get a random number.
6c25: 85 08                     sta     ]GenByte08
6c27: ad f8 02                  lda     ScrTmrReload            ;Is the random number smaller than the saucer timer
6c2a: 4a                        lsr     A                       ; reload value / 2? If so, create a small saucer
6c2b: c5 08                     cmp     ]GenByte08
6c2d: b0 01                     bcs     SetScrStatus            ;Else branch to create a large saucer.
6c2f: ca                SetSmallScr     dex                     ;X=1; create a small saucer
6c30: 8e 1c 02          SetScrStatus    stx     ScrStatus       ;Store size of saucer and exit
6c33: 60                        rts

                        ;
                        ; For the routines below, a saucer is already active.  These routines update the
                        ; active saucer.
                        ;
6c34: a5 5c             ScrYVelocity    lda     FrameTimer      ;Randomly change saucer Y velocity every 128 frames
6c36: 0a                        asl     A                       ;Is it time to change the saucer's Y velocity?
6c37: d0 0c                     bne     ChkScrUpdate            ;If not, branch
6c39: 20 b5 77                  jsr     GetRandNum              ;Get a random number
6c3c: 29 03                     and     #%00000011              ;Keep the lower 2 bits for index into table below
6c3e: aa                        tax
6c3f: bd d1 6c                  lda     ScrYSpeedTbl,x          ;Load new Y velocity value for the saucer
6c42: 8d 62 02                  sta     SaucerYSpeed
6c45: a5 1c             ChkScrUpdate    lda     NumPlayers      ;Is a game being played?
6c47: f0 05                     beq     ChkScrFire              ;If not, branch to check saucer fire timer
6c49: ad fa 02                  lda     ShipSpawnTmr            ;Is the player actively playing?
6c4c: d0 05                     bne     ScrUpdateEnd            ;If not, branch to exit
6c4e: ce f7 02          ChkScrFire      dec     ScrTimer        ;Is it time for the saucer's next action?
6c51: f0 01                     beq     ScrUpdateAction         ;If so, branch to do saucer's next action
6c53: 60                ScrUpdateEnd    rts

6c54: a9 0a             ScrUpdateAction lda     #$0a            ;Reload saucer timer for next saucer action
6c56: 8d f7 02                  sta     ScrTimer
6c59: ad 1c 02                  lda     ScrStatus               ;Is this a big of small saucer?
6c5c: 4a                        lsr     A                       ;If its a large saucer, prepare to shoot a random shot
6c5d: f0 06                     beq     GetScrShpDistance       ;If its a small saucer, prepare to shoot an aimed shot
6c5f: 20 b5 77                  jsr     GetRandNum              ;Get a random number
6c62: 4c c2 6c                  jmp     ScrShoot                ;Prepare to generate a saucer bullet

                        GetScrShpDistance
6c65: ad 3f 02                  lda     SaucerXSpeed            ;Get saucer X direction velocity
6c68: c9 80                     cmp     #$80
6c6a: 6a                        ror     A                       ;/2 with sign extension
6c6b: 85 0c                     sta     GenByte0C               ;Save result
6c6d: ad ca 02                  lda     ShipXPosLo
6c70: 38                        sec                             ;Get difference between saucer and ship X position low byte
6c71: ed cb 02                  sbc     ScrXPosLo
6c74: 85 0b                     sta     GenByte0B               ;Save result
6c76: ad 84 02                  lda     ShipXPosHi              ;Get difference between saucer and ship X position high byte
6c79: ed 85 02                  sbc     ScrXPosHi
6c7c: 20 ec 77                  jsr     NextScrShpDist          ;Calculate next frame saucer/ship X distance
```

```
6c7f: c9 40                     cmp     #$40                    ;Is the saucer to the left of the ship?
6c81: 90 06                     bcc     SetSmallScrShotDir      ;If so, branch to shoot bullet to the right
6c83: c9 c0                     cmp     #$c0                    ;Is saucer to the far right of the ship?
6c85: b0 02                     bcs     SetSmallScrShotDir      ;If so, branch to shoot bullet to right so it can screen wrap
6c87: 49 ff                     eor     #$ff                    ;Change sign so bullet can shoot left
                    SetSmallScrShotDir
6c89: aa                        tax                             ;Save X distance data for bullet
                    ;
6c8a: ad 62 02                  lda     SaucerYSpeed            ;Get saucer Y velocity and set carry if traveling
6c8d: c9 80                     cmp     #$80                    ; in a negative direction
6c8f: 6a                        ror     A                       ;Divide speed by 2 and set MSB based on Y direction
6c90: 85 0c                     sta     GenByte0C
6c92: ad ed 02                  lda     ShipYPosLo
6c95: 38                        sec                             ;Get difference between saucer and ship X position low byte
6c96: ed ee 02                  sbc     ScrYPosLo
6c99: 85 0b                     sta     GenByte0B               ;Save result
6c9b: ad a7 02                  lda     ShipYPosHi              ;Get difference between saucer and ship X position high byte
6c9e: ed a8 02                  sbc     ScrYPosHi
6ca1: 20 ec 77                  jsr     NextScrShpDist          ;Calculate next frame saucer/ship Y distance
6ca4: a8                        tay                             ;Save Y distance data for bullet
                    ;
6ca5: 20 f0 76                  jsr     CalcScrShotDir          ;Calculate the small saucer's shot direction
6ca8: 85 62                     sta     ScrBulletDir            ;Saucer shot direction is the same type of data as ship direction
6caa: 20 b5 77                  jsr     GetRandNum              ;Get a random number
6cad: a6 19                     ldx     ScoreIndex
6caf: b4 53                     ldy     PlayerScores+1,x        ;Is the player's score less than 35,000?
6cb1: c0 35                     cpy     #$35                    ;If so, add inaccuracy to small saucer's bullet
6cb3: a2 00                     ldx     #$00
6cb5: 90 01                     bcc     ScrShotAddOffset
6cb7: e8                        inx                             ;Player's score is high, make saucer shot more accurate
                    ScrShotAddOffset
6cb8: 3d cd 6c                  and     ShotRndAddTbl,x         ;Mask random value to randomize saucer bullet
6cbb: 10 03                     bpl     RandomizeScrShot
6cbd: 1d cf 6c                  ora     ShotRndAddTbl+2,x       ;Is random value negative? If so, adjust bullet velocity.
                    RandomizeScrShot
6cc0: 65 62                     adc     ScrBulletDir            ;Add randomized value to small saucer shot
6cc2: 85 62         ScrShoot    sta     ScrBulletDir            ;Prepare to fire a bullet if a slot is available
                    ]NumBulletSlots .var    $0e    {addr/1}

6cc4: a0 03                     ldy     #$03                    ;Start index for saucer bullet slots
6cc6: a2 01                     ldx     #$01                    ;2 bullet slots for the saucer
6cc8: 86 0e                     stx     ]NumBulletSlots
6cca: 4c f2 6c                  jmp     FindBulletSlot          ;Find an empty saucer bullet slot

6ccd: 8f 87         ShotRndAddTbl   .bulk   $8f,$87             ;Mask for random value to add to small saucer shot
6ccf: 70 78                     .bulk   $70,$78                 ;If negative random, set bits to bring it close to the ship
                    ; This table sets the saucer Y velocity.  It is randomly set and moves the
                    ; saucer diagonally across the screen.
6cd1: f0            ScrYSpeedTbl    .dd1    $f0                 ;-16 moving down
6cd2: 00                        .dd1    $00                     ; 0  no Y velocity
6cd3: 00                        .dd1    $00                     ; 0  no Y velocity
6cd4: 10                        .dd1    $10                     ;+16 moving up
6cd5: 00 00                     .junk   2                       ;unused

                    ;
                    ; Update ship routine.
                    ;
6cd7: a5 1c         UpdateShip      lda     NumPlayers          ;Is a game currently being played?
6cd9: f0 21                     beq     EndUpdateShip           ;If not, branch to exit
6cdb: 0e 04 20                  asl     FireSw                  ;Shift current state of fire button into shift register
6cde: 66 63                     ror     ShipBulletSR
6ce0: 24 63                     bit     ShipBulletSR            ;Is MSB of bullet shift register set?
6ce2: 10 18                     bpl     EndUpdateShip           ;If not, branch to exit; limits fire rate
6ce4: 70 16                     bvs     EndUpdateShip           ;Is bit 6 set? If so, branch to exit; prevents auto fire
6ce6: ad fa 02                  lda     ShipSpawnTmr            ;Is ship waiting to spawn?
6ce9: d0 11                     bne     EndUpdateShip           ;If so, branch to exit
6ceb: aa                        tax                             ;Zero out X; indicates ship is updating in following functions
6cec: a9 03                     lda     #$03                    ;Prepare to check 4 bullet slots
6cee: 85 0e                     sta     ]NumBulletSlots
6cf0: a0 07                     ldy     #$07                    ;Set index to ship bullet slots.
                    ; Bullet generation routine.  The functions below are used for both ship bullets
                    ; and saucer bullets.
6cf2: b9 1b 02      FindBulletSlot  lda     ShipStatus,y        ;Get ship/saucer bullet status
6cf5: f0 06                     beq     BulletSlotFound         ;Is slot available? If so, branch to continue
6cf7: 88                        dey                             ;Move to next bullet slot
6cf8: c4 0e                     cpy     ]NumBulletSlots         ;Is there more bullet slots to check?
6cfa: d0 f6                     bne     FindBulletSlot          ;If so, branch check next bullet slot
6cfc: 60            EndUpdateShip   rts                         ;Done updating bullets

                    • Clear variables
                    ]ObjectXPosNeg  .var    $08    {addr/1}
                    ]ShotXDir       .var    $09    {addr/1}
                    ]ObjectYPosNeg  .var    $0b    {addr/1}
                    ]ShotYDir       .var    $0c    {addr/1}
                    ]ShipScrShot    .var    $0d    {addr/1}
```

```
6cfd: 86 0d        BulletSlotFound stx     ]ShipScrShot            ;Store index to bullet type being processed
6cff: a9 12                        lda     #18                     ;Set bullet to last for 18 frames
6d01: 99 1b 02                     sta     ShipStatus,y
6d04: b5 61                        lda     ShipDir,x               ;Get ship direction/saucer bullet direction
6d06: 20 d2 77                     jsr     CalcXThrust             ;Calculate X velocity for the bullet
6d09: a6 0d                        ldx     ]ShipScrShot            ;Reload index to ship direction/saucer bullet direction
6d0b: c9 80                        cmp     #$80                    ;Is ship/bullet facing left? If so, set carry bit
6d0d: 6a                           ror     A                       ;Divide direction value by 2 and save carry bit in MSB
6d0e: 85 09                        sta     ]ShotXDir
6d10: 18                           clc                             ;Add X velocity change to existing velocity
6d11: 7d 3e 02                     adc     ShipXSpeed,x            ;Is this a right to left traveling object?
6d14: 30 08                        bmi     ChkMaxNegXVel           ;If so, branch to set a velocity limit
6d16: c9 70                        cmp     #112                    ;Must be a left to right moving object
6d18: 90 0a                        bcc     SaveObjXVel             ;Has max X velocity been reached? if so, branch
6d1a: a9 6f                        lda     #111                    ;Set maximum X velocity (111 pixels per frame)
6d1c: d0 06                        bne     SaveObjXVel             ;Branch always

6d1e: c9 91        ChkMaxNegXVel   cmp     #145                    ;Has max X velocity been reached (right to left)?
6d20: b0 02                        bcs     SaveObjXVel             ;If not, branch
6d22: a9 91                        lda     #$91                    ;Maximum negative X velocity (-111 pixels per frame)
6d24: 99 3e 02     SaveObjXVel     sta     ShipXSpeed,y            ;Save updated X velocity. Done only once for bullets
6d27: b5 61                        lda     ShipDir,x               ;Get ship direction/saucer bullet direction
6d29: 20 d5 77                     jsr     CalcThrustDir           ;Calculate Y velocity for the bullet
6d2c: a6 0d                        ldx     ]ShipScrShot            ;Reload index to ship direction/saucer bullet direction
6d2e: c9 80                        cmp     #$80                    ;Is ship/bullet facing downward? If so, set carry bit
6d30: 6a                           ror     A                       ;Divide direction value by 2 and save carry bit in MSB
6d31: 85 0c                        sta     ]ShotYDir
6d33: 18                           clc                             ;Add Y velocity change to existing velocity
6d34: 7d 61 02                     adc     ShipYSpeed,x            ;Is this a top to bottom traveling object?
6d37: 30 08                        bmi     ChkMaxNegYVel           ;If so, branch to set a velocity limit
6d39: c9 70                        cmp     #112                    ;Must be a bottom to top moving object
6d3b: 90 0a                        bcc     SaveObjYVel             ;Has max Y velocity been reached? if so, branch
6d3d: a9 6f                        lda     #111                    ;Set maximum Y velocity (111 pixels per frame)
6d3f: d0 06                        bne     SaveObjYVel

6d41: c9 91        ChkMaxNegYVel   cmp     #145                    ;Has max Y velocity been reached (top to bottom)?
6d43: b0 02                        bcs     SaveObjYVel             ;If not, branch
6d45: a9 91                        lda     #145                    ;Maximum negative Y velocity (-111 pixels per frame)
6d47: 99 61 02     SaveObjYVel     sta     ShipYSpeed,y            ;Save updated Y velocity; done only once for bullets
6d4a: a2 00                        ldx     #$00                    ;Assume shot moving left to right
6d4c: a5 09                        lda     ]ShotXDir               ;Is shot moving left to right?
6d4e: 10 01                        bpl     SetShotXPos             ;If so, branch
6d50: ca                           dex                             ;Shot is moving right to left
6d51: 86 08        SetShotXPos     stx     ]ObjectXPosNeg          ;Store value used for properly updating shot X position
                   ;
6d53: a6 0d                        ldx     ]ShipScrShot            ;Reload index to ship direction/saucer bullet direction
6d55: c9 80                        cmp     #$80                    ;Is ship/bullet facing left? If so, set carry bit
6d57: 6a                           ror     A                       ;Divide direction value by 2 and save carry bit in MSB
6d58: 18                           clc                             ;Add value to the bullet X direction
6d59: 65 09                        adc     ]ShotXDir
6d5b: 18                           clc
6d5c: 7d ca 02                     adc     ShipXPosLo,x            ;Update lower byte of shot X position
6d5f: 99 ca 02                     sta     ShipXPosLo,y
6d62: a5 08                        lda     ]ObjectXPosNeg
6d64: 7d 84 02                     adc     ShipXPosHi,x            ;Update upper byte of shot X position with proper sign
6d67: 99 84 02                     sta     ShipXPosHi,y
6d6a: a2 00                        ldx     #$00                    ;Assume shot moving bottom to top
6d6c: a5 0c                        lda     ]ShotYDir               ;Is shot moving bottom to top?
6d6e: 10 01                        bpl     SetShotYPos             ;If so, branch
6d70: ca                           dex                             ;Shot is moving top to bottom
6d71: 86 0b        SetShotYPos     stx     ]ObjectYPosNeg          ;Store value used for properly updating shot Y position
                   ;
6d73: a6 0d                        ldx     ]ShipScrShot            ;Reload index to ship direction/saucer bullet direction
6d75: c9 80                        cmp     #$80                    ;Is ship/bullet facing down? If so, set carry bit
6d77: 6a                           ror     A                       ;Divide direction value by 2 and save carry bit in MSB
6d78: 18                           clc                             ;Add value to the bullet Y direction
6d79: 65 0c                        adc     ]ShotYDir
6d7b: 18                           clc
6d7c: 7d ed 02                     adc     ShipYPosLo,x            ;Update lower byte of shot Y position
6d7f: 99 ed 02                     sta     ShipYPosLo,y
6d82: a5 0b                        lda     ]ObjectYPosNeg
6d84: 7d a7 02                     adc     ShipYPosHi,x            ;Update upper byte of shot Y position with proper sign
6d87: 99 a7 02                     sta     ShipYPosHi,y
6d8a: a9 80                        lda     #$80
6d8c: 95 66                        sta     FireSFXTimer,x          ;Turn on SFX for the shot fired
6d8e: 60                           rts

6d8f: d6                           .dd1    $d6                     ;checksum byte

                   ;
                   ; High score message routines.
                   ;
                   • Clear variables
                   ]GlobalScale    .var    $00    {addr/1}
                   ]SelInitial     .var    $0c    {addr/1}
                   ]InitialDebounce .var   $63    {addr/1}
```

```
6d90: a5 32        ChkHighScrMsg   lda     Plyr1Rank               ;Did one of the players get a ranking in the top 10?
6d92: 25 33                        and     Plyr2Rank
6d94: 10 01                        bpl     GetPrevPlayers          ;If so, branch to keep going
6d96: 60                           rts                             ;Else exit

6d97: a5 1a        GetPrevPlayers  lda     PrevGamePlyrs           ;Get the number of players in the game that just ended
6d99: 4a                           lsr     A                       ;Was last game a single player game?
6d9a: f0 18                        beq     DoHighScrMsg            ;If so, branch
6d9c: a0 01                        ldy     #PlyrText               ;PLAYER
6d9e: 20 f6 77                     jsr     WriteText               ;Write text to the display
6da1: a0 02                        ldy     #$02                    ;Prepare to indicate player 2 high score
6da3: a6 33                        ldx     Plyr2Rank               ;Did player 2 get a high score?
6da5: 10 01                        bpl     DoPlayerDigits          ;If so, branch
6da7: 88                           dey                             ;Indicate player 1 got high score
                   ;
6da8: 84 18        DoPlayerDigits  sty     CurrentPlyr             ;Indicate which player got the high score
6daa: a5 5c                        lda     FrameTimer
6dac: 29 10                        and     #$10                    ;Should the player number be displayed?
6dae: d0 04                        bne     DoHighScrMsg            ;If not, branch
6db0: 98                           tya                             ;Set player's digit(1 or 2)
6db1: 20 d1 7b                     jsr     DrawDigit               ;Draw a single digit on the display
                   ;
6db4: 46 18        DoHighScrMsg    lsr     CurrentPlyr             ;Get current player
6db6: 20 b2 73                     jsr     SwapRAM                 ;Set RAM for current player
6db9: a0 02                        ldy     #YrScrText              ;YOUR SCORE IS ONE OF THE TEN BEST
6dbb: 20 f6 77                     jsr     WriteText               ;Write text to the display
6dbe: a0 03                        ldy     #InitText               ;PLEASE ENTER YOUR INITIALS
6dc0: 20 f6 77                     jsr     WriteText               ;Write text to the display
6dc3: a0 04                        ldy     #PshRtText              ;PUSH ROTATE TO SELECT LETTER
6dc5: 20 f6 77                     jsr     WriteText               ;Write text to the display
6dc8: a0 05                        ldy     #PshHypText             ;PUSH HYPERSPACE WHEN LETTER IS CORRECT
6dca: 20 f6 77                     jsr     WriteText               ;Write text to the display
                   ;
6dcd: a9 20                        lda     #$20                    ;Set global scale=2(*4)
6dcf: 85 00                        sta     ]GlobalScale
6dd1: a9 64                        lda     #$64                    ;X beam coordinate 4 * $64 = $190  = 400
6dd3: a2 39                        ldx     #$39                    ;Y beam coordinate 4 * $39 = $E4  = 228
6dd5: 20 03 7c                     jsr     MoveBeam                ;Move the CRT beam to a new location
6dd8: a9 70                        lda     #$70                    ;Set scale 7(/4)
6dda: 20 de 7c                     jsr     SpotKill                ;Draw zero vector to prevent spots on the screen
                   ;
6ddd: a6 18                        ldx     CurrentPlyr
6ddf: b4 32                        ldy     Plyr1Rank,x             ;Save the offset to the current player's initials
6de1: 84 0b                        sty     GenByte0B
6de3: 98                           tya                             ;Save index to player's current initial being changed
6de4: 18                           clc
6de5: 65 31                        adc     ThisInitial
6de7: 85 0c                        sta     ]SelInitial
6de9: 20 1a 6f                     jsr     DrawInitial             ;Draw a single initial on the display
6dec: a4 0b                        ldy     GenByte0B               ;Draw second initial
6dee: c8                           iny
6def: 20 1a 6f                     jsr     DrawInitial             ;Draw a single initial on the display
6df2: a4 0b                        ldy     GenByte0B               ;Draw third initial
6df4: c8                           iny
6df5: c8                           iny
6df6: 20 1a 6f                     jsr     DrawInitial             ;Draw a single initial on the display
                   ;
6df9: ad 03 20                     lda     HyprSpcSw               ;Get hyperspace button status
6dfc: 2a                           rol     A
6dfd: 26 63                        rol     ]InitialDebounce        ;Roll value into debounce register
6dff: a5 63                        lda     ]InitialDebounce
6e01: 29 1f                        and     #%00011111              ;Keep only lower 5 bits of debounce register
6e03: c9 07                        cmp     #%00000111              ;Hyperspace button must be pressed for 3 frames to register
6e05: d0 27                        bne     ChkScoreTimeUp          ;Did player select an initial? If not, branch
6e07: e6 31                        inc     ThisInitial             ;Move to the next initial
6e09: a5 31                        lda     ThisInitial
6e0b: c9 03                        cmp     #$03                    ;Has the last initial been selected?
6e0d: 90 13                        bcc     NextInitial             ;If not, branch
6e0f: a6 18                        ldx     CurrentPlyr
6e11: a9 ff                        lda     #$ff                    ;Zero out the current player's rank
6e13: 95 32                        sta     Plyr1Rank,x
6e15: a2 00        FinishHighScore ldx     #$00
6e17: 86 18                        stx     CurrentPlyr             ;Move to player 1 and zero out initial index
6e19: 86 31                        stx     ThisInitial
6e1b: a2 f0                        ldx     #$f0                    ;Reset frame timer
6e1d: 86 5d                        stx     FrameTimer+1
6e1f: 4c b2 73                     jmp     SwapRAM                 ;Set RAM for current player

6e22: e6 0c        NextInitial     inc     ]SelInitial             ;Increment initial index
6e24: a6 0c                        ldx     ]SelInitial
6e26: a9 f4                        lda     #$f4                    ;Reset frame timer
6e28: 85 5d                        sta     FrameTimer+1
6e2a: a9 0b                        lda     #$0b                    ;Set value of new initial to A
6e2c: 95 34                        sta     HighScoreIntls,x
6e2e: a5 5d        ChkScoreTimeUp  lda     FrameTimer+1            ;Has initial entry time expired?
```

```
6e30: d0 08                     bne     ScoreTimeRemain         ;If not, branch
6e32: a9 ff                     lda     #$ff
6e34: 85 32                     sta     Plyr1Rank               ;Zero out player's ranks and finish
6e36: 85 33                     sta     Plyr2Rank
6e38: 30 db                     bmi     FinishHighScore

6e3a: a5 5c     ScoreTimeRemain lda     FrameTimer              ;Only update displayed initial every 8th frame
6e3c: 29 07                     and     #$07                    ;Is this the 8th frame?
6e3e: d0 31                     bne     HighScoreEnd            ;If not, branch
6e40: ad 07 24                  lda     RotLeftSw               ;Has rotate left button been pressed?
6e43: 10 04                     bpl     ChkScoreRightBtn        ;If not, branch
6e45: a9 01                     lda     #$01                    ;Increment initial
6e47: d0 07                     bne     ChangeInitial           ;Branch always

                ChkScoreRightBtn
6e49: ad 06 24                  lda     RotRghtSw               ;Has rotate right button been pressed?
6e4c: 10 23                     bpl     HighScoreEnd            ;If not, branch to end
6e4e: a9 ff                     lda     #$ff                    ;Decrement initial
                ;
6e50: a6 0c     ChangeInitial   ldx     ]SelInitial             ;Update the selected initial
6e52: 18                        clc
6e53: 75 34                     adc     HighScoreIntls,x        ;Does value need to wrap around to Z?
6e55: 30 10                     bmi     SetInitialMax           ;If so, branch
6e57: c9 0b                     cmp     #$0b                    ;Is initial less than the index for A?
6e59: b0 0e                     bcs     ChkInitialMax           ;If so, branch to force index to SPACE
6e5b: c9 01                     cmp     #$01                    ;Is index for a number?
6e5d: f0 04                     beq     SetInitialMin           ;If so, branch to force index to A
6e5f: a9 00                     lda     #$00                    ;Set initial index to SPACE
6e61: f0 0c                     beq     SetInitial              ;Branch always

6e63: a9 0b     SetInitialMin   lda     #$0b                    ;Set initial index to A
6e65: d0 08                     bne     SetInitial              ;Branch always

6e67: a9 24     SetInitialMax   lda     #$24                    ;Set selected initial to Z
6e69: c9 25     ChkInitialMax   cmp     #$25                    ;Does initial index need to wrap to SPACE?
6e6b: 90 02                     bcc     SetInitial              ;If not, branch
6e6d: a9 00                     lda     #$00                    ;Set initial index to SPACE
6e6f: 95 34     SetInitial      sta     HighScoreIntls,x        ;Store new initial value
6e71: a9 00     HighScoreEnd    lda     #$00                    ;Done processing high score for this frame
6e73: 60                        rts

                ;
                ; Enter hyperspace routine.
                ;
6e74: a5 1c     EnterHyperspc   lda     NumPlayers              ;Is a game currently being played?
6e76: f0 5f                     beq     ChkHyprspcEnd           ;If not, branch to exit
6e78: ad 1b 02                  lda     ShipStatus              ;Is the player's ship currently exploding?
6e7b: 30 5a                     bmi     ChkHyprspcEnd           ;If so, branch to exit
6e7d: ad fa 02                  lda     ShipSpawnTmr            ;Is the ship currently waiting to spawn?
6e80: d0 55                     bne     ChkHyprspcEnd           ;If so, branch to exit
6e82: ad 03 20                  lda     HyprSpcSw               ;Has the hyperspace button been pressed?
6e85: 10 50                     bpl     ChkHyprspcEnd           ;If not, branch to exit
                ;
6e87: a9 00                     lda     #$00                    ;Indicate the ship has entered hyperspace
6e89: 8d 1b 02                  sta     ShipStatus
6e8c: 8d 3e 02                  sta     ShipXSpeed              ;Zero out ship velocity.
6e8f: 8d 61 02                  sta     ShipYSpeed
6e92: a9 30                     lda     #$30                    ;Set ship spawn timer
6e94: 8d fa 02                  sta     ShipSpawnTmr
6e97: 20 b5 77                  jsr     GetRandNum              ;Get a random number
6e9a: 29 1f                     and     #%00011111              ;Get lower 5 bits for new ship X position
6e9c: c9 1d                     cmp     #$1d                    ;Make sure value is capped
6e9e: 90 02                     bcc     MinHyprspcXPos          ;Is value greater than the maximum allowed? If not, branch
6ea0: a9 1c                     lda     #$1c                    ;Set X position to max value
6ea2: c9 03     MinHyprspcXPos  cmp     #$03                    ;Is value less than the minimum allowed? If not, branch
6ea4: b0 02                     bcs     SetHyprspcXPos
6ea6: a9 03                     lda     #$03                    ;Set X position to min value
6ea8: 8d 84 02  SetHyprspcXPos  sta     ShipXPosHi              ;Set the new X position for the ship
                ;
6eab: a2 05                     ldx     #$05                    ;Prepare to get a random number 5 times.
6ead: 20 b5 77  HyprspcRandLoop jsr     GetRandNum              ;Get a random number
6eb0: ca                        dex                             ;finished getting random numbers?
6eb1: d0 fa                     bne     HyprspcRandLoop         ;If not, branch to get another one
6eb3: 29 1f                     and     #%00011111              ;Get lower 5 bits of random number
6eb5: e8                        inx                             ;Assume a successful hyperspace jump
6eb6: c9 18                     cmp     #$18                    ;Check if random number causes a failed hyperspace jump
6eb8: 90 0c                     bcc     MaxHyprspcYPos          ;Jump failed? If not, branch
6eba: 29 07                     and     #%00000111              ;Take lower 3 bits of random number *2 + 4
6ebc: 0a                        asl     A                       ;Is the resulting value < current number of asteroids?
6ebd: 69 04                     adc     #$04                    ;If so, jump was unsuccessful
6ebf: cd f6 02                  cmp     CurAsteroids
6ec2: 90 02                     bcc     MaxHyprspcYPos          ;Was jump successful? If so, branch
                ;
6ec4: a2 80                     ldx     #$80                    ;Indicate an unsuccessful hyperspace jump.
6ec6: c9 15     MaxHyprspcYPos  cmp     #$15                    ;Make sure value is capped.
6ec8: 90 02                     bcc     MinHyprspcYPos          ;Is value greater than the maximum allowed? If not, branch
```

```
6eca: a9 14                     lda     #$14                    ;Set Y position to max value
6ecc: c9 03        MinHyprspcYPos  cmp     #$03                    ;Is value less than the minimum allowed? If not, branch
6ece: b0 02                     bcs     SetHyprspcYPos
6ed0: a9 03                     lda     #$03                    ;Set Y position to min value
6ed2: 8d a7 02     SetHyprspcYPos  sta     ShipYPosHi              ;Set the new Y position for the ship
6ed5: 86 59                     stx     HyprSpcFlag             ;Set the success or failure of the hyperspace jump
6ed7: 60           ChkHyprspcEnd   rts

                   ;
                   ; Initialize game variables.
                   ;
6ed8: a9 02        InitGameVars    lda     #$02                    ;Prepare to start wave 1 with 4 asteroids (+2 later)
6eda: 8d f5 02                  sta     AstPerWave
6edd: a2 03                     ldx     #$03                    ;Is the DIP switches set for 3 ships per game?
6edf: 4e 02 28                  lsr     CentCMShipsSw
6ee2: b0 01                     bcs     InitShipsPerGame        ;If so, branch
6ee4: e8                        inx                             ;4 ships per game
                   InitShipsPerGame
6ee5: 86 56                     stx     ShipsPerGame            ;Load initial ships to start this game with
6ee7: a9 00                     lda     #$00                    ;Prepare to zero variables
6ee9: a2 03                     ldx     #$03
6eeb: 9d 1b 02     VarZeroLoop     sta     ShipStatus,x
6eee: 9d 1f 02                  sta     ShpShotTimer,x
6ef1: 95 52                     sta     Plr1ScoreBase,x         ;Zero out ship status, saucer status and player scores
6ef3: ca                        dex
6ef4: 10 f5                     bpl     VarZeroLoop
6ef6: 8d f6 02                  sta     CurAsteroids            ;Zero out current number of asteroids
6ef9: 60                        rts

                   ;
                   ; Silence sound effects.
                   ;
6efa: a9 00        SilenceSFX      lda     #$00
6efc: 8d 00 36                  sta     ExpPitchVol
6eff: 8d 00 3a                  sta     ThumpFreqVol
6f02: 8d 00 3c                  sta     SaucerSFX
6f05: 8d 01 3c                  sta     SaucerFireSFX           ;Zero out SFX control registers
6f08: 8d 03 3c                  sta     ShipThrustSFX
6f0b: 8d 04 3c                  sta     ShipFireSFX
6f0e: 8d 05 3c                  sta     LifeSFX
                   ;
6f11: 85 69                     sta     ExplsnSFXTimer
6f13: 85 66                     sta     FireSFXTimer
6f15: 85 67                     sta     ScrFrSFXTimer           ;Zero out SFX timers
6f17: 85 68                     sta     ExLfSFXTimer
6f19: 60                        rts

                   ;
                   ; Draw initial.
                   ;
6f1a: b9 34 00     DrawInitial     lda     HighScoreIntls,y        ;Get value of currently selected initial
6f1d: 0a                        asl     A
6f1e: a8                        tay                             ;Does it have a value?
6f1f: d0 14                     bne     DrawChar                ;If so, branch to draw the initial
6f21: a5 32                     lda     Plyr1Rank               ;Is one of the players in the top 10?
6f23: 25 33                     and     Plyr2Rank
6f25: 30 0e                     bmi     DrawChar                ;If not, branch to write the existing initial
6f27: a9 72                     lda     #$72                    ;SVEC for drawing most of the underline
6f29: a2 f8                     ldx     #$f8
6f2b: 20 45 7d                  jsr     VecWriteWord            ;Write 2 bytes to vector RAM
6f2e: a9 01                     lda     #$01                    ;SVEC for drawing the rest of the underline
6f30: a2 f8                     ldx     #$f8
6f32: 4c 45 7d                  jmp     VecWriteWord            ;Write 2 bytes to vector RAM

6f35: be d5 56     DrawChar        ldx     CharPtrTbl+1,y          ;Draw the initial on the display
6f38: b9 d4 56                  lda     CharPtrTbl,y
6f3b: 4c 45 7d                  jmp     VecWriteWord            ;Write 2 bytes to vector RAM

                   ;
                   ; Draw reserve ship on display.
                   ;
                   • Clear variables
                   ]GlobalScale    .var    $00    {addr/1}
                   ]GenByte08      .var    $08    {addr/1}

6f3e: f0 16        DrawExtraLives  beq     EndDrawLives            ;Does payer have ships in reserve? If not, branch to exit
6f40: 84 08                     sty     ]GenByte08              ;Create counter value for number of ships to draw
6f42: a2 d5                     ldx     #$d5                    ;Y beam coordinate 4 * $D5 = $354 = 852
6f44: a0 e0                     ldy     #$e0                    ;Set global scale=14(/4)
6f46: 84 00                     sty     ]GlobalScale
6f48: 20 03 7c                  jsr     MoveBeam                ;Move the CRT beam to a new location
6f4b: a2 da        DrawLivesLoop   ldx     #$da                    ;Load JSRL to reserve ship vector data into vector RAM
6f4d: a9 54                     lda     #$54
6f4f: 20 fc 7b                  jsr     VecRomJSRL              ;Load JSRL command in vector RAM to vector ROM
6f52: c6 08                     dec     ]GenByte08              ;More ships to draw?
6f54: d0 f5                     bne     DrawLivesLoop           ;If so, branch
```

```
6f56: 60           EndDrawLives    rts

                   ;
                   ; Update objects routine.
                   ;
                   • Clear variables
                   ]ThisObjX       .var    $04    {addr/2}
                   ]ThisObjY       .var    $06    {addr/2}

6f57: a2 22        UpdateObjects   ldx     #34                 ;Prepare to check every object
6f59: bd 00 02     UpdateObjLoop   lda     AstStatus,x         ;Is the current object active?
6f5c: d0 04                        bne     UpdateCurObject     ;If so, branch to update it
6f5e: ca           NextObjUpdate   dex                         ;Move to next object
6f5f: 10 f8                        bpl     UpdateObjLoop       ;Done checking objects? If not, branch to do the next one
6f61: 60                           rts                         ;Done updating objects

6f62: 10 63        UpdateCurObject bpl     UpdateObjPos        ;Is current object exploding? If not, branch
6f64: 20 08 77                     jsr     TwosCompliment      ;Calculate the 2's compliment of the value in A
6f67: 4a                           lsr     A
6f68: 4a                           lsr     A                   ;Move upper nibble to lower nibble
6f69: 4a                           lsr     A
6f6a: 4a                           lsr     A
6f6b: e0 1b                        cpx     #ShipIndex          ;Is it the ship that is exploding? If not, branch
6f6d: d0 07                        bne     IncExplosion
6f6f: a5 5c                        lda     FrameTimer          ;Update ship explosion once every other frame
6f71: 29 01                        and     #$01                ;Ship explosion is twice as slow as other objects
6f73: 4a                           lsr     A                   ;Time to update ship explosion?
6f74: f0 01                        beq     SaveIncExplosion    ;If not, branch
6f76: 38           IncExplosion    sec                         ;Prepare to increment to next explosion state
                   SaveIncExplosion
6f77: 7d 00 02                     adc     AstStatus,x         ;Save updated explosion timer
6f7a: 30 25                        bmi     ObjectExploding     ;Is object still exploding? If so, branch
6f7c: e0 1b                        cpx     #ShipIndex          ;Did the ship just finish exploding?
6f7e: f0 13                        beq     ResetShip           ;If so, branch
6f80: b0 17                        bcs     ResetSaucer         ;Did the saucer just finish exploding? If so, branch
6f82: ce f6 02                     dec     CurAsteroids        ;Must have been an asteroid that finished exploding
6f85: d0 05                        bne     ClearObjSlot        ;Decrement number of asteroids. Any left? If so, branch
6f87: a0 7f                        ldy     #$7f                ;No more asteroids this wave
6f89: 8c fb 02                     sty     ThmpSpeedTmr        ;Reset thump speed to slowest speed
6f8c: a9 00        ClearObjSlot    lda     #$00                ;Free up asteroid slot
6f8e: 9d 00 02                     sta     AstStatus,x
6f91: f0 cb                        beq     NextObjUpdate       ;Branch always to move to the next object slot

6f93: 20 e8 71     ResetShip       jsr     CenterShip          ;Center ship on display and zero velocity
6f96: 4c 8c 6f                     jmp     ClearObjSlot        ;Set ship status to 0

6f99: ad f8 02     ResetSaucer     lda     ScrTmrReload        ;Reset saucer timer
6f9c: 8d f7 02                     sta     ScrTimer
6f9f: d0 eb                        bne     ClearObjSlot        ;Branch always

6fa1: 9d 00 02     ObjectExploding sta     AstStatus,x         ;Save updated exploding timer
6fa4: 29 f0                        and     #$f0
6fa6: 18                           clc                         ;Get scale to use for exploding object.
6fa7: 69 10                        adc     #$10
6fa9: e0 1b                        cpx     #ShipIndex          ;Special case. Is ship exploding?
6fab: d0 02                        bne     SetObjExplodeScale  ;If not, branch to save exploding scale
6fad: a9 00                        lda     #$00                ;Prepare to set ship explode scale to 7(/4)
                   SetObjExplodeScale
6faf: a8                           tay                         ;Save scale to use for object debris
6fb0: bd af 02                     lda     AstXPosLo,x
6fb3: 85 04                        sta     ]ThisObjX
6fb5: bd 69 02                     lda     AstXPosHi,x
6fb8: 85 05                        sta     ]ThisObjX+1         ;Make a copy of the object position in preparation for drawing
6fba: bd d2 02                     lda     AstYPosLo,x
6fbd: 85 06                        sta     ]ThisObjY
6fbf: bd 8c 02                     lda     AstYPosHi,x
6fc2: 85 07                        sta     ]ThisObjY+1
6fc4: 4c 27 70                     jmp     DoDrawObject        ;Prepare to draw current object on the display

6fc7: 18           UpdateObjPos    clc                         ;Assume object is moving from left to right
6fc8: a0 00                        ldy     #$00
6fca: bd 23 02                     lda     AstXSpeed,x         ;Is object moving from left to right?
6fcd: 10 01                        bpl     UpdateObjXPos       ;If so, branch
6fcf: 88                           dey                         ;Indicate object moving from right to left
6fd0: 7d af 02     UpdateObjXPos   adc     AstXPosLo,x         ;Add X velocity to current X position
6fd3: 9d af 02                     sta     AstXPosLo,x
6fd6: 85 04                        sta     ]ThisObjX           ;Make a copy of location for drawing the object
6fd8: 98                           tya                         ;Update the signed upper byte of the object X position
6fd9: 7d 69 02                     adc     AstXPosHi,x
6fdc: c9 20                        cmp     #$20                ;Is the object off the X edge of the display?
6fde: 90 0c                        bcc     SaveObjXPos         ;If not, branch
6fe0: 29 1f                        and     #$1f                ;Wrap object to the other X edge of the display
6fe2: e0 1c                        cpx     #ScrIndex           ;Is the object a saucer?
6fe4: d0 06                        bne     SaveObjXPos         ;If not, branch
6fe6: 20 2d 70                     jsr     SaucerReset         ;Reset saucer variables
6fe9: 4c 5e 6f                     jmp     NextObjUpdate       ;Check next object slot
```

```
6fec: 9d 69 02     SaveObjXPos     sta     AstXPosHi,x             ;Save the updated object X position
6fef: 85 05                        sta     ]ThisObjX+1
6ff1: 18                           clc                             ;Assume object is moving from top to bottom
6ff2: a0 00                        ldy     #$00
6ff4: bd 46 02                     lda     AstYSpeed,x             ;Is object moving from top to bottom?
6ff7: 10 02                        bpl     UpdateObjYPos           ;If so, branch
6ff9: a0 ff                        ldy     #$ff                    ;Indicate object moving from top to bottom
6ffb: 7d d2 02     UpdateObjYPos   adc     AstYPosLo,x             ;Add Y velocity to current Y position
6ffe: 9d d2 02                     sta     AstYPosLo,x
7001: 85 06                        sta     ]ThisObjY               ;Make a copy of location for drawing the object
7003: 98                           tya                             ;Update the signed upper byte of the object Y position
7004: 7d 8c 02                     adc     AstYPosHi,x
7007: c9 18                        cmp     #$18                    ;Is the object off the Y edge of the display?
7009: 90 08                        bcc     SaveObjYPos             ;If not, branch
700b: f0 04                        beq     WrapObjYPos             ;Is object on Y edge border? If so, branch to wrap object
700d: a9 17                        lda     #$17                    ;Place object at the upper edge of the display
700f: d0 02                        bne     SaveObjYPos             ;Branch always

7011: a9 00        WrapObjYPos     lda     #$00                    ;Put object at the bottom edge of the display
7013: 9d 8c 02     SaveObjYPos     sta     AstYPosHi,x             ;Save the updated object Y position
7016: 85 07                        sta     ]ThisObjY+1
7018: bd 00 02                     lda     AstStatus,x             ;Reload the object status for further processing
701b: a0 e0                        ldy     #$e0                    ;Prepare to set scale to 9(/1)
701d: 4a                           lsr     A                       ;Does object exist?
701e: b0 07                        bcs     DoDrawObject            ;If so, branch to prepare to draw current object on the display
7020: a0 f0                        ldy     #$f0                    ;Prepare to set scale to 8(/2)
7022: 4a                           lsr     A                       ;Does object exist?
7023: b0 02                        bcs     DoDrawObject            ;If so, branch to prepare to draw current object on the display
7025: a0 00                        ldy     #$00                    ;Prepare to set scale to 7(/4)
7027: 20 fe 72     DoDrawObject    jsr     DrawObject              ;Draw asteroid, ship, saucer
702a: 4c 5e 6f                     jmp     NextObjUpdate           ;Check next object slot

                   ;
                   ; Saucer reset.
                   ;
702d: ad f8 02     SaucerReset     lda     ScrTmrReload            ;Reset saucer timer
7030: 8d f7 02                     sta     ScrTimer
7033: a9 00                        lda     #$00
7035: 8d 1c 02                     sta     ScrStatus
7038: 8d 3f 02                     sta     SaucerXSpeed            ;Clear other saucer variables
703b: 8d 62 02                     sta     SaucerYSpeed
703e: 60                           rts

                   ;
                   ; Ship status updates.
                   ;
703f: a5 1c        ChkExitHprspc   lda     NumPlayers              ;Is a game being played?
7041: f0 42                        beq     ShipStsExit1            ;If not, branch to exit.
7043: ad 1b 02                     lda     ShipStatus              ;Is the Player's ship exploding?
7046: 30 3d                        bmi     ShipStsExit1            ;If so, branch to exit
7048: ad fa 02                     lda     ShipSpawnTmr            ;Is the ship currently waiting to respawn?
704b: f0 39                        beq     ChkPlyrInput            ;If not, branch
704d: ce fa 02                     dec     ShipSpawnTmr            ;Decrement the spawn timer. Still waiting to respawn?
7050: d0 33                        bne     ShipStsExit1            ;If so, branch to exit
7052: a4 59                        ldy     HyprSpcFlag             ;Did a hyperspace jump just fail?
7054: 30 19                        bmi     HyprspcFailed           ;If so, branch
7056: d0 10                        bne     HyprspcSuccess          ;Is ship in hyperspace? If so, branch
7058: 20 39 71                     jsr     IsReturnSafe            ;Check to see if safe for ship to exit hyperspace
705b: d0 24                        bne     ResetHyprspc            ;Did safety check succeed? If not, branch
705d: ac 1c 02                     ldy     ScrStatus               ;Is a saucer on the screen?
7060: f0 06                        beq     HyprspcSuccess          ;If not, branch to bring player out of hyperspace
7062: a0 02                        ldy     #$02                    ;Make sure spawn timer is not 0
7064: 8c fa 02                     sty     ShipSpawnTmr            ;Not safe to return from hyperspace
7067: 60                           rts

7068: a9 01        HyprspcSuccess  lda     #$01                    ;Indicate ship is no longer in hyperspace
706a: 8d 1b 02                     sta     ShipStatus
706d: d0 12                        bne     ResetHyprspc            ;Branch always

706f: a9 a0        HyprspcFailed   lda     #$a0                    ;Indicate the ship is exploding
7071: 8d 1b 02                     sta     ShipStatus
7074: a2 3e                        ldx     #$3e                    ;Set the explosion SFX timer
7076: 86 69                        stx     ExplsnSFXTimer
7078: a6 18                        ldx     CurrentPlyr             ;Decrement the player's extra lives
707a: d6 57                        dec     Plyr1Ships,x
707c: a9 81                        lda     #$81                    ;Set the ship spawn timer
707e: 8d fa 02                     sta     ShipSpawnTmr
7081: a9 00        ResetHyprspc    lda     #$00                    ;Clear the hyperspace status
7083: 85 59                        sta     HyprSpcFlag
7085: 60           ShipStsExit1    rts                             ;Exit ship status update routines

7086: ad 07 24     ChkPlyrInput    lda     RotLeftSw               ;Is rotate left being pressed?
7089: 10 04                        bpl     ChkRotRight             ;If not, branch
708b: a9 03                        lda     #$03                    ;Prepare to add 3 to ship direction
708d: d0 07                        bne     UpdateShipDir           ;Branch always
```

```
708f: ad 06 24     ChkRotRight     lda     RotRghtSw               ;Is rotate right being pressed?
7092: 10 07                        bpl     ChkThrust
7094: a9 fd                        lda     #$fd                    ;Prepare to subtract 3 to ship direction
7096: 18           UpdateShipDir   clc
7097: 65 61                        adc     ShipDir                 ;Update ship direction
7099: 85 61                        sta     ShipDir
                   ;
709b: a5 5c        ChkThrust       lda     FrameTimer              ;Update ship velocity only every other frame
709d: 4a                           lsr     A                       ;Time to update ship velocity?
709e: b0 e5                        bcs     ShipStsExit1            ;If not, branch to exit
70a0: ad 05 24                     lda     ThrustSw                ;Is thrust being pressed?
70a3: 10 3c                        bpl     ShipDecelerate          ;If not, branch
70a5: a9 80                        lda     #$80                    ;Enable the ship thrust SFX
70a7: 8d 03 3c                     sta     ShipThrustSFX
70aa: a0 00                        ldy     #$00                    ;Assume ship is facing right (positive X direction)
70ac: a5 61                        lda     ShipDir                 ;Get ship direction in preparation for thrust calculation
70ae: 20 d2 77                     jsr     CalcXThrust             ;Calculate thrust in X direction
70b1: 10 01                        bpl     UpdateShipXVel          ;Is ship facing right? If so, branch
70b3: 88                           dey                             ;Ship is facing left; set X for negative direction
70b4: 0a           UpdateShipXVel  asl     A                       ;Multiply thrust value by 2
70b5: 18                           clc
70b6: 65 64                        adc     ShipXAccel              ;Add thrust to ship's X acceleration
70b8: aa                           tax                             ;Save the acceleration in X
70b9: 98                           tya
70ba: 6d 3e 02                     adc     ShipXSpeed              ;Add the acceleration to the ship's X velocity
70bd: 20 25 71                     jsr     ChkShipMaxVel           ;Ensure ship does not exceed maximum velocity
70c0: 8d 3e 02                     sta     ShipXSpeed              ;Save current ship X velocity and acceleration
70c3: 86 64                        stx     ShipXAccel
                   ;
70c5: a0 00                        ldy     #$00                    ;Assume ship is facing up (positive Y direction)
70c7: a5 61                        lda     ShipDir                 ;Get ship direction in preparation for thrust calculation
70c9: 20 d5 77                     jsr     CalcThrustDir           ;Calculate thrust in Y direction
70cc: 10 01                        bpl     UpdateShipYVel          ;Is ship facing up? If so, branch
70ce: 88                           dey                             ;Ship is facing down; set Y for negative direction
70cf: 0a           UpdateShipYVel  asl     A                       ;Multiply thrust value by 2
70d0: 18                           clc
70d1: 65 65                        adc     ShipYAccel              ;Add thrust to ship's Y acceleration
70d3: aa                           tax                             ;Save the acceleration in X
70d4: 98                           tya
70d5: 6d 61 02                     adc     ShipYSpeed              ;Add the acceleration to the ship's Y velocity
70d8: 20 25 71                     jsr     ChkShipMaxVel           ;Ensure ship does not exceed maximum velocity
70db: 8d 61 02                     sta     ShipYSpeed              ;Save current ship Y velocity and acceleration
70de: 86 65                        stx     ShipYAccel
70e0: 60                           rts                             ;Done calculating ship acceleration

70e1: a9 00        ShipDecelerate  lda     #$00                    ;Turn off ship thrust SFX
70e3: 8d 03 3c                     sta     ShipThrustSFX
70e6: ad 3e 02                     lda     ShipXSpeed              ;Does ship need to be decelerated in the X direction?
70e9: 05 64                        ora     ShipXAccel
70eb: f0 18                        beq     DecelerateY             ;If not, branch to check Y deceleration
70ed: ad 3e 02                     lda     ShipXSpeed              ;Get ship X velocity and multiply by 2
70f0: 0a                           asl     A
70f1: a2 ff                        ldx     #$ff                    ;Assume positive X velocity; X acceleration = -1
70f3: 18                           clc
70f4: 49 ff                        eor     #$ff                    ;Is ship traveling in the positive X direction?
70f6: 30 02                        bmi     SetXDecelerate          ;If so, branch
70f8: e8                           inx                             ;Ship traveling in negative X direction
70f9: 38                           sec                             ;Set X deceleration to +1
70fa: 65 64        SetXDecelerate  adc     ShipXAccel              ;Update ship X acceleration
70fc: 85 64                        sta     ShipXAccel
70fe: 8a                           txa
70ff: 6d 3e 02                     adc     ShipXSpeed              ;Update ship X velocity
7102: 8d 3e 02                     sta     ShipXSpeed
                   ;
7105: a5 65        DecelerateY     lda     ShipYAccel              ;Does ship need to be decelerated in the Y direction?
7107: 0d 61 02                     ora     ShipYSpeed
710a: f0 18                        beq     DecelerateExit          ;If not, branch to exit
710c: ad 61 02                     lda     ShipYSpeed              ;Get ship Y velocity and multiply by 2
710f: 0a                           asl     A
7110: a2 ff                        ldx     #$ff                    ;Assume positive Y velocity; Y acceleration = -1
7112: 18                           clc
7113: 49 ff                        eor     #$ff                    ;Is ship traveling in the positive Y direction?
7115: 30 02                        bmi     SetYDecelerate          ;If so, branch
7117: 38                           sec                             ;Ship traveling in negative Y direction
7118: e8                           inx                             ;Set Y deceleration to +1
7119: 65 65        SetYDecelerate  adc     ShipYAccel              ;Update ship Y acceleration
711b: 85 65                        sta     ShipYAccel
711d: 8a                           txa
711e: 6d 61 02                     adc     ShipYSpeed              ;Update ship Y velocity
7121: 8d 61 02                     sta     ShipYSpeed
7124: 60           DecelerateExit  rts

7125: 30 09        ChkShipMaxVel   bmi     ChkMaxNegVel            ;Is ship traveling left/down (negative direction)? If so, branch
7127: c9 40                        cmp     #$40                    ;Is ship moving less than max velocity in positive direction?
7129: 90 0d                        bcc     ChkMaxExit              ;If so, branch to exit
```

```
712b: a2 ff                     ldx     #$ff                    ;Max positive velocity reached; set acceleration to -1
712d: a9 3f                     lda     #$3f                    ;Set velocity to max positive value
712f: 60                        rts

7130: c9 c0     ChkMaxNegVel    cmp     #$c0                    ;Is ship moving less than max velocity in negative direction?
7132: b0 04                     bcs     ChkMaxExit              ;If so, branch to exit
7134: a2 01                     ldx     #$01                    ;Max negative velocity reached; set acceleration to +1
7136: a9 c0                     lda     #$c0                    ;Set velocity to max negative value
7138: 60        ChkMaxExit      rts                             ;Done checking maximum ship velocity

                ;
                ; Safe hyperspace return routine.
                ;
7139: a2 1c     IsReturnSafe    ldx     #ScrIndex               ;Prepare to check all asteroids and saucer.
713b: bd 00 02  SafeCheckLoop   lda     AstStatus,x             ;Is current object slot active?
713e: f0 1e                     beq     NextSafeCheck           ;If not, branch to move to next object
7140: bd 69 02                  lda     AstXPosHi,x             ;Get object X position and compare to ship X position
7143: 38                        sec
7144: ed 84 02                  sbc     ShipXPosHi
7147: c9 04                     cmp     #$04                    ;Is object within +4 pixels of ship?
7149: 90 04                     bcc     SafeCheckY              ;If so, branch to check object's Y position
714b: c9 fc                     cmp     #$fc                    ;Is object within -4 pixels of ship?
714d: 90 0f                     bcc     NextSafeCheck           ;If not, branch to check next object's position
714f: bd 8c 02  SafeCheckY      lda     AstYPosHi,x             ;Get object Y position and compare to ship Y position
7152: 38                        sec
7153: ed a7 02                  sbc     ShipYPosHi
7156: c9 04                     cmp     #$04                    ;Is object within +4 pixels of ship?
7158: 90 09                     bcc     SafeCheckFail           ;If so, branch; not safe to exit hyperspace
715a: c9 fc                     cmp     #$fc                    ;Is object within -4 pixels of ship?
715c: b0 05                     bcs     SafeCheckFail           ;If so, branch; not safe to exit hyperspace
715e: ca        NextSafeCheck   dex                             ;Is there another object to check?
715f: 10 da                     bpl     SafeCheckLoop           ;If so, branch to check the object
7161: e8                        inx                             ;Safe to exit hyperspace; sets X to zero
7162: 60                        rts

7163: ee fa 02  SafeCheckFail   inc     ShipSpawnTmr            ;Not safe to exit hyperspace; ensures spawn timer is not zero
7166: 60                        rts

7167: 90                        .dd1    $90                     ;checksum byte

                ;
                ; Initialize asteroid wave variables.
                ;
                • Clear variables
                ]GenByte08      .var    $08    {addr/1}

7168: a2 1a     InitWaveVars    ldx     #MaxAsteroids           ;Start at highest asteroid status slot
716a: ad fb 02                  lda     ThmpSpeedTmr            ;Is wave about to start?
716d: d0 70                     bne     ZeroAstStatuses         ;If so, branch to skip most of this routine
716f: ad 1c 02                  lda     ScrStatus               ;Is a saucer active?
7172: d0 73                     bne     EndInitWave             ;If so, branch to skip this routine
7174: 8d 3f 02                  sta     SaucerXSpeed            ;Zero out saucer speed.
7177: 8d 62 02                  sta     SaucerYSpeed
717a: ee fd 02                  inc     ScrSpeedup              ;Increment the min number of asteroids that triggers saucers
717d: ad fd 02                  lda     ScrSpeedup              ; appearing more frequently
7180: c9 0b                     cmp     #11                     ;Max value is 11 asteroids
7182: 90 03                     bcc     InitAstPerWave
7184: ce fd 02                  dec     ScrSpeedup              ;Make sure value does not exceed 11 asteroids
                ;
7187: ad f5 02  InitAstPerWave  lda     AstPerWave              ;Increase number of asteroids by 2 every wave
718a: 18                        clc
718b: 69 02                     adc     #$02
718d: c9 0b                     cmp     #11                     ;Ensure 11 asteroids max per wave
718f: 90 02                     bcc     SetWaveAst
7191: a9 0b                     lda     #11                     ;Max initial asteroids per wave is 11
7193: 8d f6 02  SetWaveAst      sta     CurAsteroids            ;Set the number of asteroids for the current wave
7196: 8d f5 02                  sta     AstPerWave
7199: 85 08                     sta     ]GenByte08              ;Create a counter for decrementing through all asteroid slots
                ;
719b: a0 1c                     ldy     #ScrIndex               ;Offset to saucer speed X and Y values
                InitWaveAsteroids
719d: 20 b5 77                  jsr     GetRandNum              ;Get a random number
71a0: 29 18                     and     #$18                    ;Randomly select asteroid type
71a2: 09 04                     ora     #LargeAst               ;Make it a large asteroid
71a4: 9d 00 02                  sta     AstStatus,x             ;Store the results
71a7: 20 03 72                  jsr     SetAstVel               ;Set asteroid X and Y velocities
71aa: 20 b5 77                  jsr     GetRandNum              ;Get a random number
71ad: 4a                        lsr     A                       ;Shift right to save LSB in carry
71ae: 29 1f                     and     #%00011111              ;Keep lower 5 bits
71b0: 90 13                     bcc     AstPosScrBot            ;Is carry clear? If so, start asteroid at top/bottom of screen
71b2: c9 18                     cmp     #$18                    ;Is value beyond max Y position(6144/8=768)
71b4: 90 02                     bcc     AstPosScrRight          ;If not, branch to set Y position
71b6: 29 17                     and     #$17                    ;Limit Y position to < 768
71b8: 9d 8c 02  AstPosScrRight  sta     AstYPosHi,x             ;Set asteroid Y position
71bb: a9 00                     lda     #$00
71bd: 9d 69 02                  sta     AstXPosHi,x             ;Set X to 0; asteroid originates at left/right of screen
```

```
71c0: 9d af 02                  sta     AstXPosLo,x
71c3: f0 0b                     beq     NextAstPos              ;Branch always

71c5: 9d 69 02      AstPosScrBot    sta     AstXPosHi,x         ;Set asteroid X position
71c8: a9 00                     lda     #$00
71ca: 9d 8c 02                  sta     AstYPosHi,x             ;Set Y to 0; asteroid originates at top/bottom of screen
71cd: 9d d2 02                  sta     AstYPosLo,x
71d0: ca            NextAstPos  dex                             ;Move to next asteroid index
71d1: c6 08                     dec     ]GenByte08              ;Are there more asteroid positions to process?
71d3: d0 c8                     bne     InitWaveAsteroids       ;If so, branch to do another one
71d5: a9 7f                     lda     #$7f
71d7: 8d f7 02                  sta     ScrTimer                ;Set initial saucer timer and thump SFX values
71da: a9 30                     lda     #$30
71dc: 8d fc 02                  sta     ThmpOffReload
71df: a9 00         ZeroAstStatuses lda     #$00                ;Zero out the asteroid statuses
71e1: 9d 00 02      :Loop       sta     AstStatus,x
71e4: ca                        dex                             ;More asteroid statuses to zero?
71e5: 10 fa                     bpl     :Loop                   ;If so, branch to do another
71e7: 60            EndInitWave rts

                    ;
                    ; Center ship on screen.
                    ;
71e8: a9 60         CenterShip  lda     #$60
71ea: 8d ca 02                  sta     ShipXPosLo              ;Set lower XY ship position bytes for screen center
71ed: 8d ed 02                  sta     ShipYPosLo
71f0: a9 00                     lda     #$00
71f2: 8d 3e 02                  sta     ShipXSpeed              ;Set ship XY speed to 0.
71f5: 8d 61 02                  sta     ShipYSpeed
71f8: a9 10                     lda     #$10
71fa: 8d 84 02                  sta     ShipXPosHi
71fd: a9 0c                     lda     #$0c                    ;Set upper XY ship position bytes for screen center
71ff: 8d a7 02                  sta     ShipYPosHi
7202: 60                        rts

                    ;
                    ; Set asteroid velocities.
                    ;
7203: 20 b5 77      SetAstVel   jsr     GetRandNum              ;Get a random number
7206: 29 8f                     and     #$8f                    ;Keep the sign bit and lower nibble
7208: 10 02                     bpl     SetAstXVel              ;Is this a negative number?
720a: 09 f0                     ora     #$f0                    ;If so, sign extend the byte
720c: 18            SetAstXVel  clc                             ;Add the new X velocity to the old velocity
720d: 79 23 02                  adc     AstXSpeed,y
7210: 20 33 72                  jsr     GetAstVelocity          ;Get an X velocity to assign to the asteroid
7213: 9d 23 02                  sta     AstXSpeed,x
7216: 20 b5 77                  jsr     GetRandNum              ;Get a random number
7219: 20 b5 77                  jsr     GetRandNum              ;Get a random number
721c: 20 b5 77                  jsr     GetRandNum              ;Get a random number
721f: 20 b5 77                  jsr     GetRandNum              ;Get a random number
7222: 29 8f                     and     #%10001111              ;Keep the sign bit and lower nibble
7224: 10 02                     bpl     SetAstYVel              ;Is this a negative number?
7226: 09 f0                     ora     #$f0                    ;If so, sign extend the byte
7228: 18            SetAstYVel  clc                             ;Add the new Y velocity to the old velocity
7229: 79 46 02                  adc     AstYSpeed,y
722c: 20 33 72                  jsr     GetAstVelocity          ;Get a Y velocity to assign to the asteroid
722f: 9d 46 02                  sta     AstYSpeed,x
7232: 60                        rts

7233: 10 0d         GetAstVelocity  bpl     SetPosVel           ;Is speed faster than max speed of -31?
7235: c9 e1                     cmp     #$e1                    ;If so, branch to check min negative speed
7237: b0 02                     bcs     ChkNegTooSlow
7239: a9 e1                     lda     #$e1                    ;Set max negative speed to -31
723b: c9 fb         ChkNegTooSlow   cmp     #$fb                ;Is value faster than -6?
723d: 90 0f                     bcc     AstVelExit              ;If so, branch to exit
723f: a9 fa                     lda     #$fa                    ;Set minimum negative speed to -6
7241: 60                        rts

7242: c9 06         SetPosVel   cmp     #6                      ;Is speed above min speed of +6?
7244: b0 02                     bcs     ChkPosTooFast           ;If so, branch to check max speed
7246: a9 06                     lda     #6                      ;Set min positive speed to +6
7248: c9 20         ChkPosTooFast   cmp     #32                 ;Is value greater than +31?
724a: 90 02                     bcc     AstVelExit              ;If not, branch to exit
724c: a9 1f                     lda     #31                     ;Set max positive speed to +31
724e: 60            AstVelExit  rts                             ;Return the velocity in A

                    ;
                    ; Update screen text.
                    ;
                    • Clear variables
                    ]GlobalScale    .var    $00    {addr/1}

                    UpdateScreenText
724f: a9 10                     lda     #$10                    ;Set global scale=1(*2)
7251: 85 00                     sta     ]GlobalScale
7253: a9 50                     lda     #>VecCredits            ;Draw copyright text at bottom of the display
```

```
7255: a2 a4                     ldx     #<VecCredits
7257: 20 fc 7b                  jsr     VecRomJSRL              ;Load JSRL command in vector RAM to vector ROM
725a: a9 19                     lda     #$19                    ;X beam coordinate 4 * $19 = $64  = 100
725c: a2 db                     ldx     #$db                    ;Y beam coordinate 4 * $DB = $36C = 876.
725e: 20 03 7c                  jsr     MoveBeam                ;Move the CRT beam to a new location.
7261: a9 70                     lda     #$70                    ;Set scale 7(/4)
7263: 20 de 7c                  jsr     SpotKill                ;Draw zero vector to prevent spots on the screen
                        ;
7266: a2 00                     ldx     #$00                    ;Indicate number string should be drawn on the display
7268: a5 1c                     lda     NumPlayers              ;Is this a 2 player game?
726a: c9 02                     cmp     #$02
726c: d0 18                     bne     DrawPlr1Score           ;If not, branch to draw just player 1's score without blinking
726e: a5 18                     lda     CurrentPlyr             ;Is player 2 playing?
7270: d0 14                     bne     DrawPlr1Score           ;If so, branch to draw player 1's score without blinking
7272: a2 20                     ldx     #$20                    ;Override the zero blanking function
7274: ad 1b 02                  lda     ShipStatus              ;Is player 1's ship in play or in hyperspace?
7277: 05 59                     ora     HyprSpcFlag             ;If so, branch to draw player 1's score without blinking
7279: d0 0b                     bne     DrawPlr1Score
727b: ad fa 02                  lda     ShipSpawnTmr            ;Is player 1 waiting to respawn?
727e: 30 06                     bmi     DrawPlr1Score           ;If so, branch to draw player 1's score without blinking
7280: a5 5c                     lda     FrameTimer              ;Blink player 1's score every 16 frames.  This occurs
7282: 29 10                     and     #$10                    ; when switching from one player to the next
7284: f0 0d                     beq     DrawShipLives           ;Time to draw the score? If not, branch to turn it off
                        ;
7286: a9 52             DrawPlr1Score   lda     #Plr1ScoreBase          ;Prepare to draw Player 1's score on the display
7288: a0 02                     ldy     #$02                    ;2 bytes for player 1's score
728a: 38                        sec                             ;Blank leading zeros
728b: 20 3f 77                  jsr     DrawNumberString        ;Draw a string of numbers on the display
728e: a9 00                     lda     #$00                    ;Draw a trailing zero
7290: 20 8b 77                  jsr     ChkSetDigitPntr         ;Prepare to draw a trailing zero after the score
7293: a9 28             DrawShipLives   lda     #$28            ;X beam coordinate 4 * $28 = $A0 = 160
7295: a4 57                     ldy     Plyr1Ships              ;Get current number of reserve ships for Player 1
7297: 20 3e 6f                  jsr     DrawExtraLives          ;Draw player's reserve ships on the display
729a: a9 00                     lda     #$00                    ;Set global scale to 0(*1)
729c: 85 00                     sta     ]GlobalScale
729e: a9 78                     lda     #$78                    ;X beam coordinate 4 * $78 = $1E0 = 480
72a0: a2 db                     ldx     #$db                    ;Y beam coordinate 4 * $DB = $36C = 876
72a2: 20 03 7c                  jsr     MoveBeam                ;Move the CRT beam to a new location
72a5: a9 50                     lda     #$50                    ;Set scale 5(/16)
72a7: 20 de 7c                  jsr     SpotKill                ;Draw zero vector to prevent spots on the screen
                        ;
72aa: a9 1d                     lda     #HighScores             ;Prepare to draw the high score on the display
72ac: a0 02                     ldy     #$02                    ;2 bytes for the high score
72ae: 38                        sec                             ;Blank leading zeros
72af: 20 3f 77                  jsr     DrawNumberString        ;Draw a string of numbers on the display.
72b2: a9 00                     lda     #$00                    ;Draw a trailing zero
72b4: 20 d1 7b                  jsr     DrawDigit               ;Draw a single digit on the display
72b7: a9 10                     lda     #$10                    ;Set global scale=1(*2)
72b9: 85 00                     sta     ]GlobalScale
72bb: a9 c0                     lda     #$c0                    ;X beam coordinate 4 * $C0 = $300 = 768
72bd: a2 db                     ldx     #$db                    ;Y beam coordinate 4 * $DB = $36C = 876
72bf: 20 03 7c                  jsr     MoveBeam                ;Move the CRT beam to a new location
72c2: a9 50                     lda     #$50                    ;Set scale 5(/16)
72c4: 20 de 7c                  jsr     SpotKill                ;Draw zero vector to prevent spots on the screen
                        ;
72c7: a2 00                     ldx     #$00                    ;Indicate number string should be drawn on the display
72c9: a5 1c                     lda     NumPlayers              ;Is this a 2 player game?
72cb: c9 01                     cmp     #$01
72cd: f0 2e                     beq     EndScreenText           ;If not, branch to exit
72cf: 90 18                     bcc     DrawPlr2Score           ;Is a game active? If not, branch to draw player 2's score
72d1: a5 18                     lda     CurrentPlyr             ;Is player 1 playing?
72d3: f0 14                     beq     DrawPlr2Score           ;If so, branch to draw player 2's score without blinking
72d5: a2 20                     ldx     #$20                    ;Override the zero blanking function
72d7: ad 1b 02                  lda     ShipStatus              ;Is player 2's ship in play or in hyperspace?
72da: 05 59                     ora     HyprSpcFlag             ;If so, branch to draw player 2's score without blinking
72dc: d0 0b                     bne     DrawPlr2Score
72de: ad fa 02                  lda     ShipSpawnTmr            ;Is player 2 waiting to respawn?
72e1: 30 06                     bmi     DrawPlr2Score           ;If so, branch to draw player 2's score without blinking
72e3: a5 5c                     lda     FrameTimer              ;Blink player 2's score every 16 frames.  This occurs
72e5: 29 10                     and     #$10                    ; when switching from one player to the next
72e7: f0 0d                     beq     DrawPlr2Ships           ;Time to draw the score? If not, branch to turn it off
72e9: a9 54             DrawPlr2Score   lda     #Plr2ScoreBase          ;Prepare to draw Player 2's score on the display
72eb: a0 02                     ldy     #$02                    ;2 bytes for the high score
72ed: 38                        sec                             ;Blank leading zeros
72ee: 20 3f 77                  jsr     DrawNumberString        ;Draw a string of numbers on the display
72f1: a9 00                     lda     #$00                    ;Draw a trailing zero
72f3: 20 8b 77                  jsr     ChkSetDigitPntr         ;Prepare to draw a trailing zero after the score
72f6: a9 cf             DrawPlr2Ships   lda     #$cf            ;X beam coordinate 4 * $CF = $33C = 828
72f8: a4 58                     ldy     Plyr2Ships              ;Get current number of reserve ships for Player 2
72fa: 4c 3e 6f                  jmp     DrawExtraLives          ;Draw player's reserve ships on the display

72fd: 60                EndScreenText   rts                     ;Done drawing screen text

                        ;
                        ; Draw object routines.
                        ;
```

```
                    • Clear variables
                    ]GlobalScale    .var    $00    {addr/1}
                    ]ThisObjX       .var    $04    {addr/2}
                    ]ThisObjY       .var    $06    {addr/2}

72fe: 84 00         DrawObject      sty     ]GlobalScale            ;Save scale data
7300: 86 0d                         stx     GenByte0D               ;Save a copy of the index to the object to draw.
7302: a5 05                         lda     ]ThisObjX+1
7304: 4a                            lsr     A
7305: 66 04                         ror     ]ThisObjX
7307: 4a                            lsr     A                       ;Divide the object's X position by 8
7308: 66 04                         ror     ]ThisObjX
730a: 4a                            lsr     A
730b: 66 04                         ror     ]ThisObjX
730d: 85 05                         sta     ]ThisObjX+1
                    ;
730f: a5 07                         lda     ]ThisObjY+1
7311: 18                            clc
7312: 69 04                         adc     #$04
7314: 4a                            lsr     A
7315: 66 06                         ror     ]ThisObjY
7317: 4a                            lsr     A
7318: 66 06                         ror     ]ThisObjY               ;Add 1024 object's Y position and divide by 8
731a: 4a                            lsr     A
731b: 66 06                         ror     ]ThisObjY
731d: 85 07                         sta     ]ThisObjY+1
                    ;
731f: a2 04                         ldx     #$04                    ;Prepare to write 4 bytes to vector RAM
7321: 20 1c 7c                      jsr     SetLABSData             ;Write LABS instruction in vector RAM
7324: a9 70                         lda     #$70                    ;Set the scale of the object
7326: 38                            sec
7327: e5 00                         sbc     ]GlobalScale
7329: c9 a0                         cmp     #$a0                    ;Is the scale 9 or smaller?
732b: 90 0e                         bcc     DrawSpotKill            ;If so, branch
                    DrawMultiSpotKill
732d: 48                            pha                             ;Save A on the stack
732e: a9 90                         lda     #$90                    ;Set scale 9(/1)
7330: 20 de 7c                      jsr     SpotKill                ;Draw zero vector to prevent spots on the screen
7333: 68                            pla                             ;Restore A from the stack
7334: 38                            sec                             ;Subtract #$10 from scale value
7335: e9 10                         sbc     #$10                    ;Is value below #$A0?
7337: c9 a0                         cmp     #$a0                    ;If not, branch to run the spot kill routine again.
7339: b0 f2                         bcs     DrawMultiSpotKill
                    ;
733b: 20 de 7c      DrawSpotKill    jsr     SpotKill                ;Draw zero vector to prevent spots on the screen
733e: a6 0d                         ldx     GenByte0D               ;Restore index to object to draw
7340: bd 00 02                      lda     AstStatus,x             ;Is the object exploding?
7343: 10 16                         bpl     DrawObjNoExplode        ;If not, branch to draw the normal object
7345: e0 1b                         cpx     #ShipIndex              ;Is it the ship exploding?
7347: f0 0c                         beq     DrawShipExplode         ;If so, branch
7349: 29 0c                         and     #$0c                    ;Get index into shrapnel table
734b: 4a                            lsr     A
734c: a8                            tay
734d: b9 f8 50                      lda     SharpPatPtrTbl,y        ;Store JSRL data in vector RAM for the Shrapnel graphics
7350: be f9 50                      ldx     SharpPatPtrTbl+1,y
7353: d0 1b                         bne     SaveObjVecData          ;Branch always

7355: 20 65 74      DrawShipExplode jsr     DoShipExplsn            ;Draw the ship exploding
7358: a6 0d                         ldx     GenByte0D               ;Restore index to object being drawn
735a: 60                            rts                             ;Exit after drawing ship fragments

                    DrawObjNoExplode
735b: e0 1b                         cpx     #ShipIndex              ;Is it the ship that needs to be drawn?
735d: f0 17                         beq     DoDrawShip              ;If so, branch
735f: e0 1c                         cpx     #ScrIndex               ;Is it the saucer that needs to be drawn?
7361: f0 19                         beq     DoDrawSaucer            ;If so, branch
7363: b0 1f                         bcs     DoDrawBullet            ;Is it a bullet that needs to be drawn? If so, branch
7365: 29 18                         and     #$18                    ;Must be an asteroid
7367: 4a                            lsr     A
7368: 4a                            lsr     A                       ;Get the asteroid type bits
7369: a8                            tay
736a: b9 de 51                      lda     AstPtrnPtrTbl,y         ;Get asteroid vector data and write it to vector RAM
736d: be df 51                      ldx     AstPtrnPtrTbl+1,y
7370: 20 45 7d      SaveObjVecData  jsr     VecWriteWord            ;Write 2 bytes to vector RAM.
7373: a6 0d                         ldx     GenByte0D               ;Restore index to object
7375: 60                            rts                             ;Finished loading object data into vector RAM

7376: 20 0b 75      DoDrawShip      jsr     UpdateShipDraw          ;Update the drawing of the player's ship
7379: a6 0d                         ldx     GenByte0D               ;Restore index to object
737b: 60                            rts                             ;Finished loading ship data into vector RAM

737c: ad 50 52      DoDrawSaucer    lda     ScrPtrnPtrTbl           ;Get saucer vector data and write it to vector RAM
737f: ae 51 52                      ldx     ScrPtrnPtrTbl+1
7382: d0 ec                         bne     SaveObjVecData          ;Branch always

7384: a9 70         DoDrawBullet    lda     #$70                    ;Set scale 7(/4)
```

```
7386: a2 f0                     ldx     #$f0                    ;Prepare to draw a dot at full brightness(bullet)
7388: 20 e0 7c                  jsr     DrawDot                 ;Draw a dot on the screen
738b: a6 0d                     ldx     GenByte0D               ;Restore index to object
738d: a5 5c                     lda     FrameTimer              ;Decrement shot timer every 4th frame
738f: 29 03                     and     #$03                    ;Is it time to decrement the shot timer?
7391: d0 03                     bne     DrawObjectDone          ;If not, branch
7393: de 00 02                  dec     AstStatus,x             ;Decrement shot timer
7396: 60            DrawObjectDone  rts                         ;Done with object vector data

                    ;
                    ; Update score.
                    ;
7397: f8            UpdateScore     sed                         ;Put ALU into decimal mode
7398: 75 52                     adc     Plr1ScoreBase,x         ;Add value in Accumulator to score
739a: 95 52                     sta     Plr1ScoreBase,x         ;Does upper byte need to be updated?
739c: 90 12                     bcc     UpdateScoreExit         ;If not, branch to exit
739e: b5 53                     lda     PlayerScores+1,x
73a0: 69 00                     adc     #$00                    ;Increment upper score byte.
73a2: 95 53                     sta     PlayerScores+1,x
73a4: 29 0f                     and     #$0f                    ;Check if extra life should be granted
73a6: d0 08                     bne     UpdateScoreExit         ;Extra life granted at 10,000 points
                    ;
73a8: a9 b0                     lda     #$b0                    ;Play extra life SFX
73aa: 85 68                     sta     ExLfSFXTimer
73ac: a6 18                     ldx     CurrentPlyr             ;Increment reserve ships
73ae: f6 57                     inc     Plyr1Ships,x
73b0: d8            UpdateScoreExit cld                         ;Put ALU back into binary mode
73b1: 60                        rts

                    ;
                    ; Swap RAM.
                    ;
                    ]GenByte08      .var    $08    {addr/1}

73b2: a5 18         SwapRAM         lda     CurrentPlyr             ;Get current player (0 or 1 value)
73b4: 0a                        asl     A
73b5: 0a                        asl     A
73b6: 85 08                     sta     ]GenByte08              ;Move the LSB to the third bit position
                    ;
73b8: a5 6f                     lda     MultiPurpBits
73ba: 29 fb                     and     #%11111011
73bc: 05 08                     ora     ]GenByte08              ;Set the player RAM based on the current player
73be: 85 6f                     sta     MultiPurpBits
73c0: 8d 00 32                  sta     MultiPurp
73c3: 60                        rts

                    ;
                    ; Draw high scores list.
                    ;
                    • Clear variables
                    ]GlobalScale    .var    $00    {addr/1}
                    ]HiScrRank      .var    $0d    {addr/1}
                    ]HiScrBeamYLoc  .var    $0e    {addr/1}
                    ]HiScrIndex     .var    $0f    {addr/1}

73c4: a5 1c         ChkHighScrList  lda     NumPlayers              ;Is a game currently being played?
73c6: f0 02                     beq     ChkDrawScrList          ;If not, branch to see if its time to show the high score list
73c8: 18            SkipScrList     clc                         ;Indicate the high scores list is not being displayed
73c9: 60                        rts                             ;Exit high score list drawing routines

73ca: a5 5d         ChkDrawScrList  lda     FrameTimer+1            ;Is it time to draw the high scores list?
73cc: 29 04                     and     #$04
73ce: d0 f8                     bne     SkipScrList             ;If not, branch to exit
73d0: a5 1d                     lda     HighScores              ;Is the high scores list empty?
73d2: 05 1e                     ora     HighScores+1
73d4: f0 f2                     beq     SkipScrList             ;If so, branch to exit
                    ;
73d6: a0 00                     ldy     #$00                    ;Prepare to display HIGH SCORES text
73d8: 20 f6 77                  jsr     WriteText               ;Write text to the display
73db: a2 00                     ldx     #$00                    ;Start at the first high score index
73dd: 86 10                     stx     InitialIndex            ;Start at the first initial index
73df: a9 01                     lda     #$01                    ;Appears not to be used
73e1: 85 00                     sta     ]GlobalScale
73e3: a9 a7                     lda     #$a7                    ;Y beam coordinate = 4 * $A7 = $29C = 668
73e5: 85 0e                     sta     ]HiScrBeamYLoc          ;Set top row of high score list
73e7: a9 10                     lda     #$10                    ;Set global scale=1(*2)
73e9: 85 00                     sta     ]GlobalScale
                    ;
73eb: b5 1d         HighScoresLoop  lda     HighScores,x            ;Is there a high score at the current location?
73ed: 15 1e                     ora     HighScores+1,x
73ef: f0 67                     beq     HighScoreExit           ;If not, done with high score list; branch to exit
73f1: 86 0f                     stx     ]HiScrIndex             ;Store index to the current high score
73f3: a9 5f                     lda     #$5f                    ;X beam coordinate 4 * $5F = $17C = 380
73f5: a6 0e                     ldx     ]HiScrBeamYLoc          ;Set the Y beam coordinate based on current line being written
73f7: 20 03 7c                  jsr     MoveBeam                ;Move the CRT beam to a new location
73fa: a9 40                     lda     #$40                    ;Set scale 4(/32)
```

```
73fc: 20 de 7c                  jsr     SpotKill                ;Draw zero vector to prevent spots on the screen
                        ;
73ff: a5 0f                     lda     ]HiScrIndex             ;Get index to current high score to draw
7401: 4a                        lsr     A
7402: f8                        sed
7403: 69 01                     adc     #$01                    ;Increment by 1 (base 10)
7405: d8                        cld
7406: 85 0d                     sta     ]HiScrRank              ;Get the rank number of the current high score
7408: a9 0d                     lda     #GenByte0D              ;[HiScrRank]
740a: 38                        sec                             ;Blank leading zeros
740b: a0 01                     ldy     #$01                    ;Single byte for player's rank
740d: a2 00                     ldx     #$00                    ;No override of zero blanking
740f: 20 3f 77                  jsr     DrawNumberString        ;Draw a string of numbers on the display
7412: a9 40                     lda     #$40                    ;Set the brightness of the dot
7414: aa                        tax
7415: 20 e0 7c                  jsr     DrawDot                 ;Draw a dot on the screen
7418: a0 00                     ldy     #$00                    ;Draw a SPACE on the display
741a: 20 35 6f                  jsr     DrawChar                ;Draw a single character on the display
741d: a5 0f                     lda     ]HiScrIndex             ;Move to next high score to draw
741f: 18                        clc
7420: 69 1d                     adc     #HighScores             ;Prepare to draw next high score on the display
7422: a0 02                     ldy     #$02                    ;2 bytes per high score
7424: 38                        sec                             ;Blank leading zeros
7425: a2 00                     ldx     #$00                    ;No override of zero blanking
7427: 20 3f 77                  jsr     DrawNumberString        ;Draw a string of numbers on the display
742a: a9 00                     lda     #$00                    ;Draw a trailing zero
742c: 20 d1 7b                  jsr     DrawDigit               ;Draw a single digit on the display
742f: a0 00                     ldy     #$00                    ;Draw a SPACE on the display
7431: 20 35 6f                  jsr     DrawChar                ;Draw a single character on the display
7434: a4 10                     ldy     InitialIndex            ;Draw the first initial of this high score
7436: 20 1a 6f                  jsr     DrawInitial             ;Draw a single initial on the display.
7439: e6 10                     inc     InitialIndex            ;Draw the second initial of this high score
743b: a4 10                     ldy     InitialIndex
743d: 20 1a 6f                  jsr     DrawInitial             ;Draw a single initial on the display.
7440: e6 10                     inc     InitialIndex            ;Draw the third initial of this high score
7442: a4 10                     ldy     InitialIndex
7444: 20 1a 6f                  jsr     DrawInitial             ;Draw a single initial on the display.
                        ;
7447: e6 10                     inc     InitialIndex            ;Move to the next initial index
7449: a5 0e                     lda     ]HiScrBeamYLoc
744b: 38                        sec                             ;Move down to the next high score row on the display
744c: e9 08                     sbc     #$08
744e: 85 0e                     sta     ]HiScrBeamYLoc
7450: a6 0f                     ldx     ]HiScrIndex             ;Move to the next high score slot
7452: e8                        inx
7453: e8                        inx                             ;Have all 10 high scores been drawn on the display?
7454: e0 14                     cpx     #20                     ;If not, branch to draw the next one
7456: 90 93                     bcc     HighScoresLoop
7458: 38                HighScoreExit   sec                     ;Indicate the high scores list is being displayed
7459: 60                        rts                             ;Exit high score list drawing routines

                        ;
                        ; Find a free asteroid slot.
                        ;
745a: a2 1a             GetFreeAstSlot  ldx     #26             ;Prepare to check 27 asteroid slots
745c: bd 00 02          NextAstSlotLoop lda     AstStatus,x     ;Is this slot free?
745f: f0 03                     beq     EndFreeAstSlot          ;If so, exit. A free slot is available
7461: ca                        dex                             ;More slots to test?
7462: 10 f8                     bpl     NextAstSlotLoop         ;If so, branch to check the next slot
7464: 60                EndFreeAstSlot  rts                     ;Asteroid slot found or no slot available

                        ;
                        ; Ship explosion routines.
                        ;
                        • Clear variables
                        ]ThisDebrisX    .var    $04     {addr/2}
                        ]ThisDebrisY    .var    $06     {addr/2}
                        ]ShipDebrisPtr  .var    $09     {addr/1}
                        ]VecPtrCopy     .var    $0b     {addr/2}

7465: ad 1b 02          DoShipExplsn    lda     ShipStatus      ;Is this the first frame of the ship explosion?
7468: c9 a2                     cmp     #$a2                    ;If so, load the initial debris data
746a: b0 22                     bcs     GetNumDebris            ;If not, branch to skip loading data
                        ;
746c: a2 0a                     ldx     #10                     ;Prepare to load 12 values from ShipExpVelTbl
                        LoadShipExplLoop
746e: bd ec 50                  lda     ShipExpVelTbl,x         ;Get byte of ship debris X velocity
7471: 4a                        lsr     A
7472: 4a                        lsr     A
7473: 4a                        lsr     A                       ;Save only the upper nibble and shift to lower nibble
7474: 4a                        lsr     A
7475: 18                        clc
7476: 69 f8                     adc     #$f8                    ;Sign extend the nibble to fill the whole byte
7478: 49 f8                     eor     #$f8
747a: 95 7e                     sta     ShpDebrisXVel+1,x       ;Save signed value into RAM
                        ;
```

```
747c: bd ed 50                  lda     ShipExpVelTbl+1,x       ;Get byte of ship debris Y velocity
747f: 4a                        lsr     A
7480: 4a                        lsr     A
7481: 4a                        lsr     A                       ;Save only the upper nibble and shift to lower nibble
7482: 4a                        lsr     A
7483: 18                        clc
7484: 69 f8                     adc     #$f8                    ;Sign extend the nibble to fill the whole byte
7486: 49 f8                     eor     #$f8
7488: 95 8a                     sta     ShpDebrisYVel+1,x       ;Save signed value into RAM
748a: ca                        dex                             ;Move to next 2 bytes in the table
748b: ca                        dex                             ;Are there more bytes to load from the table?
748c: 10 e0                     bpl     LoadShipExplLoop        ;if so, loop to load 2 more bytes
                ;
748e: ad 1b 02  GetNumDebris    lda     ShipStatus
7491: 49 ff                     eor     #$ff
7493: 29 70                     and     #%01110000              ;Calculate the pointer into the ship debris data based
7495: 4a                        lsr     A                       ; on the ship status counter.  This has the effect of making
7496: 4a                        lsr     A                       ; the debris disappear one by one over time
7497: 4a                        lsr     A
7498: aa                        tax
7499: 86 09     ShipDebrisLoop  stx     ]ShipDebrisPtr          ;Update ship debris index
749b: a0 00                     ldy     #$00                    ;Assume the X velocity for this debris piece is positive
749d: bd ec 50                  lda     ShipExpVelTbl,x         ;Is the debris piece moving in a positive X direction?
74a0: 10 01                     bpl     GetDebrisXVel           ;If so, branch
74a2: 88                        dey                             ;The X velocity for this debris piece is negative
74a3: 18        GetDebrisXVel   clc                             ;Update fractional part of debris X position
74a4: 75 7d                     adc     ShpDebrisXVel,x
74a6: 95 7d                     sta     ShpDebrisXVel,x
74a8: 98                        tya
74a9: 75 7e                     adc     ShpDebrisXVel+1,x       ;Update integer part of debris X position
74ab: 95 7e                     sta     ShpDebrisXVel+1,x
74ad: 85 04                     sta     ]ThisDebrisX            ;Save current debris X position
74af: 84 05                     sty     ]ThisDebrisX+1          ;Save current debris X direction
                ;
74b1: a0 00                     ldy     #$00                    ;Assume the Y velocity for this debris piece is positive
74b3: bd ed 50                  lda     ShipExpVelTbl+1,x       ;Is the debris piece moving in a positive Y direction?
74b6: 10 01                     bpl     GetDebrisYVel           ;If so, branch
74b8: 88                        dey                             ;The Y velocity for this debris piece is negative
74b9: 18        GetDebrisYVel   clc                             ;Update fractional part of debris Y position
74ba: 75 89                     adc     ShpDebrisYVel,x
74bc: 95 89                     sta     ShpDebrisYVel,x
74be: 98                        tya
74bf: 75 8a                     adc     ShpDebrisYVel+1,x       ;Update integer part of debris Y position
74c1: 95 8a                     sta     ShpDebrisYVel+1,x
74c3: 85 06                     sta     ]ThisDebrisY            ;Save current debris Y position
74c5: 84 07                     sty     ]ThisDebrisY+1          ;Save current debris Y direction
                ;
74c7: a5 02                     lda     VecRamPtr
74c9: 85 0b                     sta     ]VecPtrCopy             ;Save a copy of the vector RAM pointer
74cb: a5 03                     lda     VecRamPtr+1
74cd: 85 0c                     sta     ]VecPtrCopy+1
74cf: 20 49 7c                  jsr     CalcDebrisPos           ;Calculate the position of the exploded ship pieces
74d2: a4 09                     ldy     ]ShipDebrisPtr          ;Write the ship debris vector data to the vector RAM
74d4: b9 e0 50                  lda     ShipExpPtrTbl,y
74d7: be e1 50                  ldx     ShipExpPtrTbl+1,y
74da: 20 45 7d                  jsr     VecWriteWord            ;Write 2 bytes to vector RAM
74dd: a4 09                     ldy     ]ShipDebrisPtr          ;Draw the exact same line from above except backwards
74df: b9 e1 50                  lda     ShipExpPtrTbl+1,y
74e2: 49 04                     eor     #$04                    ;Backtrack in the Y direction
74e4: aa                        tax
74e5: b9 e0 50                  lda     ShipExpPtrTbl,y
74e8: 29 0f                     and     #$0f                    ;Set the brightness of the backtracked vector to 0
74ea: 49 04                     eor     #$04                    ;Backtrack in the X direction
74ec: 20 45 7d                  jsr     VecWriteWord            ;Write 2 bytes to vector RAM
                ;
74ef: a0 ff                     ldy     #$ff                    ;Prepare to write 4 bytes to vector RAM
74f1: c8        VecBackTrack    iny                             ;Get position of the data where this function first started
74f2: b1 0b                     lda     (]VecPtrCopy),y         ; writing to vector RAM
74f4: 91 02                     sta     (VecRamPtr),y           ;Copy the data again into the current position in vector RAM
74f6: c8                        iny                             ; except draw it backwards to backtrack the XY position to
74f7: b1 0b                     lda     (]VecPtrCopy),y         ; the starting point
74f9: 49 04                     eor     #$04                    ;Draw the exact same line from CalcDebrisPos except backwards
74fb: 91 02                     sta     (VecRamPtr),y           ;This places the pointer back to the middle of the ship's position
74fd: c0 03                     cpy     #$03                    ;Does the second word of the VCTR opcode need to be written
74ff: 90 f0                     bcc     VecBackTrack            ;If so, branch to write second word
                ;
7501: 20 39 7c                  jsr     VecPtrUpdate            ;Update Vector RAM pointer
7504: a6 09                     ldx     ]ShipDebrisPtr          ;Move to next pair of ship debris data
7506: ca                        dex
7507: ca                        dex                             ;Is there more ship debris data to process?
7508: 10 8f                     bpl     ShipDebrisLoop          ;If so, branch
750a: 60                        rts

                ;
                ; Update the player's ship drawing.
                ;
```

```
                  • Clear variables
                  ]ShipDrawXInv   .var    $08    {addr/1}
                  ]ShipDrawYInv   .var    $09    {addr/1}
                  ]VecPtr         .var    $0b    {addr/2}

750b: a2 00       UpdateShipDraw  ldx     #$00                ;Used for inverting index into ship direction table
750d: 86 17                       stx     ShipDrawUnused      ;Always 0; not used for anything
750f: a0 00                       ldy     #$00                ;Assume ship is pointing up
7511: a5 61                       lda     ShipDir             ;Is ship pointing down?
7513: 10 06                       bpl     SaveShipDir         ;If not, branch
7515: a0 04                       ldy     #$04                ;Set value indicating ship Y direction is inverted
7517: 8a                          txa
7518: 38                          sec                         ;Subtract ship direction from #$00 to invert index into
7519: e5 61                       sbc     ShipDir             ; ShipDirPtrTbl
751b: 85 08       SaveShipDir     sta     ]ShipDrawXInv       ;Save current index calculations
751d: 24 08                       bit     ]ShipDrawXInv       ;Is ship pointing down and left?
751f: 30 02                       bmi     InvertShipX         ;If so, branch to invert X axis of ship
7521: 50 07                       bvc     SetShipInvAxes      ;Is ship pointing up and left? If not, branch
7523: a2 04       InvertShipX     ldx     #$04                ;Set value indicating ship X direction is inverted.
7525: a9 80                       lda     #$80
7527: 38                          sec                         ;Subtract modified ship direction from #$80 to get
7528: e5 08                       sbc     ]ShipDrawXInv       ; proper index into ShipDirPtrTbl
752a: 86 08       SetShipInvAxes  stx     ]ShipDrawXInv       ;Save the X and Y axis inversion indicators
752c: 84 09                       sty     ]ShipDrawYInv
                  ;
752e: 4a                          lsr     A
752f: 29 fe                       and     #$fe                ;Do final calculations on index for ShipDirPtrTbl
7531: a8                          tay
7532: b9 6e 52                    lda     ShipDirPtrTbl,y     ;Get pointer to ship vector data for current direction
7535: be 6f 52                    ldx     ShipDirPtrTbl+1,y
7538: 20 d3 6a                    jsr     DrawShip            ;Draw the Player's ship on the display
753b: ad 05 24                    lda     ThrustSw            ;Is the thrust button being pressed?
753e: 10 14                       bpl     EndUpdShpDraw       ;If not, branch to exit
7540: a5 5c                       lda     FrameTimer          ;Show thrust animation every 4th frame
7542: 29 04                       and     #$04                ;Is this the fourth frame?
7544: f0 0e                       beq     EndUpdShpDraw       ;If not, branch to exit
7546: c8                          iny                         ;Prepare to move vector ROM pointer to thrust data
7547: c8                          iny
7548: 38                          sec
7549: a6 0c                       ldx     ]VecPtr+1           ;Increment vector ROM pointer by 2 bytes
754b: 98                          tya
754c: 65 0b                       adc     ]VecPtr             ;Pointer is now at thrust vector data
754e: 90 01                       bcc     DrawShipThrust      ;Draw thrust vectors on the display
7550: e8                          inx                         ;Increment the upper byte of the vector data pointer
7551: 20 d3 6a    DrawShipThrust  jsr     DrawShip            ;Draw the Player's ship on the display
7554: 60          EndUpdShpDraw   rts                         ;Finished updating the ship and thrust graphics

                  ;
                  ; SFX control routines.
                  ;
7555: a5 1c       ChkUpdateSFX    lda     NumPlayers          ;Is an active game in progress?
7557: d0 01                       bne     UpdateSFX           ;If so, branch to update SFX
7559: 60                          rts

755a: a2 00       UpdateSFX       ldx     #$00                ;Prepare to turn off saucer SFX if it is exploding or not present
755c: ad 1c 02                    lda     ScrStatus           ;Is a saucer currently exploding?
755f: 30 0a                       bmi     UpdateScrSFX        ;If so, branch
7561: f0 08                       beq     UpdateScrSFX        ;Is a saucer present? If not, branch to ensure the SFX is off
7563: 6a                          ror     A
7564: 6a                          ror     A                   ;Use saucer size to set proper saucer SFX
7565: 6a                          ror     A
7566: 8d 02 3c                    sta     SaucerSFXSel
7569: a2 80                       ldx     #$80                ;Turn on saucer SFX
756b: 8e 00 3c    UpdateScrSFX   stx     SaucerSFX           ;Enable/disable saucer SFX
                  ;
756e: a2 01                       ldx     #$01                ;Select the saucer fire SFX
7570: 20 cd 75                    jsr     StartSFXTimer       ;Start SFX timer, if applicable
7573: 8d 01 3c                    sta     SaucerFireSFX       ;Store updated status of the SFX
                  ;
7576: ca                          dex                         ;Select the ship fire SFX
7577: 20 cd 75                    jsr     StartSFXTimer       ;Start SFX timer, if applicable
757a: 8d 04 3c                    sta     ShipFireSFX         ;Store updated status of the SFX
                  ;
757d: ad 1b 02                    lda     ShipStatus          ;Is the ship currently on the screen?
7580: c9 01                       cmp     #$01
7582: f0 04                       beq     ChkNumAsteroids     ;If so, branch
7584: 8a                          txa                         ;Load A with #$00. No ship on the screen
7585: 8d 03 3c                    sta     ShipThrustSFX       ;Turn off the thrust SFX
                  ;
7588: ad f6 02    ChkNumAsteroids lda     CurAsteroids        ;Are there asteroids left in this wave?
758b: f0 11                       beq     ThumpSFXOff         ;If not, branch to reset thump SFX
758d: ad 1b 02                    lda     ShipStatus          ;Is the ship exploding?
7590: 30 0c                       bmi     ThumpSFXOff         ;If so, branch to reset the thump SFX
7592: 05 59                       ora     HyprSpcFlag         ;Is the ship not active and not in hyperspace?
7594: f0 08                       beq     ThumpSFXOff         ;If so, branch to reset the thump SFX
7596: a5 6d                       lda     ThmpOnTime          ;Is the thump SFX currently playing?
```

```
7598: f0 14                     beq     ChkThumpOffTime         ;If not, branch
759a: c6 6d                     dec     ThmpOnTime              ;Decrement thump on timer
759c: d0 21                     bne     ChkExplTimer            ;Is thump on timer still active? if so, branch
                ;
759e: a5 6c     ThumpSFXOff     lda     ThisVolFreq
75a0: 29 0f                     and     #$0f                    ;Turn off the thump SFX
75a2: 85 6c                     sta     ThisVolFreq
75a4: 8d 00 3a                  sta     ThumpFreqVol
75a7: ad fc 02                  lda     ThmpOffReload
75aa: 85 6e                     sta     ThumpOffTime            ;Set thump off timer to max value
75ac: 10 11                     bpl     ChkExplTimer
75ae: c6 6e     ChkThumpOffTime dec     ThumpOffTime            ;Decrement the thump off timer
75b0: d0 0d                     bne     ChkExplTimer            ;Is it time to turn thump SFX back on? If not, branch
75b2: a9 04                     lda     #$04                    ;Set the thump on timer
75b4: 85 6d                     sta     ThmpOnTime
75b6: a5 6c                     lda     ThisVolFreq
75b8: 49 14                     eor     #%00010100              ;Toggle the thump volume bit on and set the frequency
75ba: 85 6c                     sta     ThisVolFreq
75bc: 8d 00 3a                  sta     ThumpFreqVol
75bf: a5 69     ChkExplTimer    lda     ExplsnSFXTimer
75c1: aa                        tax                             ;Is the explosion SFX timer active?
75c2: 29 3f                     and     #$3f                    ;If not, branch to skip decrementing it
75c4: f0 01                     beq     UpdateExplTimer
75c6: ca                        dex                             ;Decrement explosion SFX timer.
75c7: 86 69     UpdateExplTimer stx     ExplsnSFXTimer
75c9: 8e 00 36                  stx     ExpPitchVol             ;Update explosion timer, pitch and volume
75cc: 60                        rts

75cd: b5 6a     StartSFXTimer   lda     ShipFireSFX_,x          ;Is the selected SFX active?
75cf: 30 0c                     bmi     ChkSFXTimer             ;If so, branch to check SFX timer status
75d1: b5 66                     lda     SFXTimers,x             ;Is the selected SFX timer currently active?
75d3: 10 12                     bpl     TurnOffSFX              ;If so, branch to turn it off
75d5: a9 10                     lda     #$10                    ;Initialize the timer for the selected SFX
75d7: 95 66                     sta     SFXTimers,x
75d9: a9 80     TurnOnSFX       lda     #$80                    ;Turn on the selected SFX
75db: 30 0c                     bmi     UpdateSFXStatus         ;Branch always

75dd: b5 66     ChkSFXTimer     lda     SFXTimers,x             ;Get the tier value for the selected SFX
75df: f0 06                     beq     TurnOffSFX              ;Is the timer expired? If so, branch to turn off
75e1: 30 04                     bmi     TurnOffSFX              ;Has the timer gone negative, if so, branch to turn off
75e3: d6 66                     dec     SFXTimers,x             ;Decrement the selected SFX timer
75e5: d0 f2                     bne     TurnOnSFX               ;Is the timer still active? If so, branch to turn SFX on
75e7: a9 00     TurnOffSFX      lda     #$00                    ;Turn off the selected SFX
75e9: 95 6a     UpdateSFXStatus sta     ShipFireSFX_,x          ;Update the SFX status
75eb: 60                        rts

                ;
                ; Split asteroid.
                ;
                • Clear variables
                ]GenByte0E      .var    $0e    {addr/1}

75ec: 86 0d     BreakAsteroid   stx     GenByte0D               ;Save a copy of the object 1 index
75ee: a9 50                     lda     #80                     ;Set asteroid break timer to 80 frames
75f0: 8d f9 02                  sta     AstBreakTimer
75f3: b9 00 02                  lda     AstStatus,y
75f6: 29 78                     and     #%01111000              ;Save the asteroid status except the size.
75f8: 85 0e                     sta     ]GenByte0E
75fa: b9 00 02                  lda     AstStatus,y             ;Reduce the asteroid size by 1
75fd: 29 07                     and     #%00000111
75ff: 4a                        lsr     A
7600: aa                        tax                             ;Does the asteroid still exist?
7601: f0 02                     beq     SaveAstStatus           ;If not, branch to skip combining size with status
7603: 05 0e                     ora     ]GenByte0E              ;Combine the other asteroid properties with the new size
7605: 99 00 02  SaveAstStatus   sta     AstStatus,y             ;Save the status of the new asteroid back into RAM
                ;
7608: a5 1c                     lda     NumPlayers              ;Is a game currently being played?
760a: f0 11                     beq     SplitAsteroid           ;If not, branch to skip updating score
760c: a5 0d                     lda     GenByte0D               ;Did the ship crash into the asteroid?
760e: f0 04                     beq     DoAstScore              ;If so, branch to add points to score
7610: c9 04                     cmp     #$04                    ;Was it a saucer or saucer bullet that hit the asteroid?
7612: 90 09                     bcc     SplitAsteroid           ;If so, branch to skip updating the score
7614: bd 59 76  DoAstScore      lda     AstPointsTbl,x          ;Get asteroid points from table based on asteroid size
7617: a6 19                     ldx     ScoreIndex
7619: 18                        clc
761a: 20 97 73                  jsr     UpdateScore             ;Add points to the current player's score
                ;
761d: be 00 02  SplitAsteroid   ldx     AstStatus,y             ;Was the asteroid completely destroyed?
7620: f0 34                     beq     BreakAstEnd             ;If so, branch to end; asteroid not split
7622: 20 5a 74                  jsr     GetFreeAstSlot          ;Find a free asteroid slot
7625: 30 2f                     bmi     BreakAstEnd             ;Was a free slot available? If not, branch to end
7627: ee f6 02                  inc     CurAsteroids            ;Increment total number of asteroids
762a: 20 9d 6a                  jsr     UpdateAsteroid          ;Update new asteroid
762d: 20 03 72                  jsr     SetAstVel               ;Set asteroid X and Y velocities
7630: bd 23 02                  lda     AstXSpeed,x             ;Get lower 5 bits asteroid X velocity and * 2
7633: 29 1f                     and     #%00011111
```

```
7635: 0a                        asl     A
7636: 5d af 02                  eor     AstXPosLo,x             ;Use this value to offset the X position of the new asteroid
7639: 9d af 02                  sta     AstXPosLo,x
763c: 20 5c 74                  jsr     NextAstSlotLoop         ;Find a free asteroid slot
763f: 30 15                     bmi     BreakAstEnd             ;Was a free slot found? If not, branch to exit
7641: ee f6 02                  inc     CurAsteroids            ;Increment total number of asteroids
7644: 20 9d 6a                  jsr     UpdateAsteroid          ;Update new asteroid
7647: 20 03 72                  jsr     SetAstVel               ;Set asteroid X and Y velocities
764a: bd 46 02                  lda     AstYSpeed,x             ;Get lower 5 bits asteroid Y velocity and * 2
764d: 29 1f                     and     #%00011111
764f: 0a                        asl     A
7650: 5d d2 02                  eor     AstYPosLo,x             ;Use this value to offset the Y position of the new asteroid
7653: 9d d2 02                  sta     AstYPosLo,x
7656: a6 0d         BreakAstEnd ldx     GenByte0D               ;Restore the object 1 index before exiting function
7658: 60                        rts

                    ;
                    ; Points awarded for destroying asteroids of different sizes.  These are BCD
                    ; values divided by 10; the score increases will be 100, 50, 20.
                    ;
7659: 10 05 02      AstPointsTbl    .bulk   $10,$05,$02

                    ;
                    ; Check for high score.
                    ;
765c: a5 1c         CheckHighScore  lda     NumPlayers              ;Is a game currently being played?
765e: 10 38                     bpl     ChkHghScrEnd            ;If not, branch to end
7660: a2 02                     ldx     #$02                    ;Start with player 2's score
7662: 85 5d                     sta     FrameTimer+1
7664: 85 32                     sta     Plyr1Rank               ;Reset the frame timer and player's ranks
7666: 85 33                     sta     Plyr2Rank
7668: a0 00         PlyrScoreLoop   ldy     #$00                    ;Start at the beginning of the high scores list
                    ChkHighScoreLoop
766a: b9 1d 00                  lda     HighScores,y            ;Compare the player's score with each entry in the high
766d: d5 52                     cmp     PlayerScores,x          ; score list
766f: b9 1e 00                  lda     HighScores+1,y
7672: f5 53                     sbc     PlayerScores+1,x        ;Is the player's score higher than the current score entry?
7674: 90 23                     bcc     PlayerHighScore         ;If so, branch to add player to the list
7676: c8                        iny                             ;Move to next entry in the high score table
7677: c8                        iny
7678: c0 14                     cpy     #$14                    ;Have all 10 entries been checked(2 bytes per entry)?
767a: 90 ee                     bcc     ChkHighScoreLoop        ;If no, branch to check the next entry
767c: ca            NextPlayerScore dex                         ;Move to next player to check their score
767d: ca                        dex                             ;Is there another player to check?
767e: 10 e8                     bpl     PlyrScoreLoop           ;If so, branch
                    ;
7680: a5 33                     lda     Plyr2Rank               ;Did player 2 get a high score?
7682: 30 0e                     bmi     FinishHghScore          ;If not, branch to wrap up this routine
7684: c5 32                     cmp     Plyr1Rank               ;Did player 1 get a better score than player 2?
7686: 90 0a                     bcc     FinishHghScore          ;If not, branch to wrap up this routine
7688: 69 02                     adc     #$02                    ;Did player 1 make the last ranking?
768a: c9 1e                     cmp     #30
768c: 90 02                     bcc     SetPlyrRank             ;If not, branch so both players can enter scores
768e: a9 ff                     lda     #$ff                    ;Player 2's score is scrubbed as it is 11th place.
7690: 85 33         SetPlyrRank     sta     Plyr2Rank               ;Set player 2's rank
7692: a9 00         FinishHghScore  lda     #$00
7694: 85 1c                     sta     NumPlayers              ;Indicate game is over and prepare to enter high score initials
7696: 85 31                     sta     ThisInitial
7698: 60            ChkHghScrEnd    rts                             ;Done checking for high a score

7699: 86 0b         PlayerHighScore stx     GenByte0B               ;Store index to current player being processed
769b: 84 0c                     sty     GenByte0C               ;Store index into high scores table.
769d: 8a                        txa
769e: 4a                        lsr     A                       ;Calculate player's rank(each rank increments by 3)
769f: aa                        tax
76a0: 98                        tya
76a1: 4a                        lsr     A                       ;Calculate index into high scores initials table
76a2: 65 0c                     adc     GenByte0C
76a4: 85 0d                     sta     GenByte0D               ;Store index into high scores initials
76a6: 95 32                     sta     Plyr1Rank,x             ;Store player's rank
76a8: a2 1b                     ldx     #27                     ;Start at lowest initials to preserve(rank 9)
76aa: a0 12                     ldy     #18                     ;Start at lowest score to preserve(rank 9)
76ac: e4 0d         ShiftScoresLoop cpx     GenByte0D               ;Has the the player's slot been reached in the high scores list?
76ae: f0 1f                     beq     ClearInitials           ;If so, branch to end shifting ranks
76b0: b5 31                     lda     ThisInitial,x
76b2: 95 34                     sta     HighScoreIntls,x
76b4: b5 32                     lda     Plyr1Rank,x             ;Get initials in high score table and move them down a rank
76b6: 95 35                     sta     HighScoreIntls+1,x
76b8: b5 33                     lda     Plyr2Rank,x
76ba: 95 36                     sta     HighScoreIntls+2,x
76bc: b9 1b 00                  lda     HighScores-2,y
76bf: 99 1d 00                  sta     HighScores,y            ;Get score in high score table and move it down a rank
76c2: b9 1c 00                  lda     HighScores-1,y
76c5: 99 1e 00                  sta     HighScores+1,y
76c8: 88                        dey                             ;Move to next score in table
76c9: 88                        dey
```

```
76ca: ca                        dex
76cb: ca                        dex                             ;Move to next initials in table
76cc: ca                        dex
76cd: d0 dd                     bne     ShiftScoresLoop         ;More scores to shift down the ranks? If so, branch
                ;
76cf: a9 0b     ClearInitials   lda     #$0b                    ;Set first initial to A
76d1: 95 34                     sta     HighScoreIntls,x
76d3: a9 00                     lda     #$00                    ;Set second and third initial to SPACE
76d5: 95 35                     sta     HighScoreIntls+1,x
76d7: 95 36                     sta     HighScoreIntls+2,x
76d9: a9 f0                     lda     #$f0                    ;Set frame timer for displaying initials
76db: 85 5d                     sta     FrameTimer+1
76dd: a6 0b                     ldx     GenByte0B               ;Load index to current player being processed
76df: a4 0c                     ldy     GenByte0C               ;Load player's index into high score table
76e1: b5 53                     lda     PlayerScores+1,x
76e3: 99 1e 00                  sta     HighScores+1,y          ;Transfer player's score into the high score table
76e6: b5 52                     lda     PlayerScores,x
76e8: 99 1d 00                  sta     HighScores,y
76eb: a0 00                     ldy     #$00                    ;Branch always to check next player's score
76ed: f0 8d                     beq     NextPlayerScore

76ef: 6e                        .dd1    $6e                     ;checksum byte

                ;
                ; Calculate small saucer shot velocity.
                ;
76f0: 98        CalcScrShotDir  tya                             ;Load the Y distance between the saucer and the ship
76f1: 10 09                     bpl     ScrShotXDir             ;Is Y direction positive? If so, branch to do X direction
76f3: 20 08 77                  jsr     TwosCompliment          ;Calculate the 2's compliment of the Y distance
76f6: 20 fc 76                  jsr     ScrShotXDir             ;Calculate the X direction of the saucer shot
76f9: 4c 08 77                  jmp     TwosCompliment          ;Calculate the 2's compliment of the value in A

76fc: a8        ScrShotXDir     tay                             ;Save the modified Y shot distance
76fd: 8a                        txa                             ;Get the the raw X shot distance
76fe: 10 0e                     bpl     CalcScrShotAngle        ;Is X direction positive? If so, branch to calculate shot angle
7700: 20 08 77                  jsr     TwosCompliment          ;Calculate the 2's compliment of the value in A
7703: 20 0e 77                  jsr     CalcScrShotAngle        ;Calculate the small saucer's shot angle
7706: 49 80                     eor     #$80                    ;Set the appropriate quadrant for the bullet
                ;
                ; 2's compliment.
                ;
7708: 49 ff     TwosCompliment  eor     #$ff
770a: 18                        clc                             ;Calculate the 2's compliment of the value in A
770b: 69 01                     adc     #$01
770d: 60                        rts

                ;
                ; Calculate small saucer shot angle.
                ;
                • Clear variables
                ]ShotXYDistance .var    $0c    {addr/1}

                CalcScrShotAngle
770e: 85 0c                     sta     ]ShotXYDistance         ;Store shot modified X distance
7710: 98                        tya
7711: c5 0c                     cmp     ]ShotXYDistance         ;Is X and Y distance the same?
7713: f0 10                     beq     ShotAngle45             ;If so, angle is 45 degrees.  Branch to set and exit
7715: 90 11                     bcc     LookUpAngle             ;Is angle in lower 45 degrees of quadrant? if so, branch
7717: a4 0c                     ldy     ]ShotXYDistance         ;Swap X and Y components as the shot is
7719: 85 0c                     sta     ]ShotXYDistance         ; in the upper 45 degrees of the quadrant
771b: 98                        tya
771c: 20 28 77                  jsr     LookUpAngle             ;Look up angle but return to find proper quadrant
771f: 38                        sec                             ;Set the appropriate quadrant for the bullet
7720: e9 40                     sbc     #$40
7722: 4c 08 77                  jmp     TwosCompliment          ;Calculate the 2's compliment of the value in A

7725: a9 20     ShotAngle45     lda     #$20                    ;Player's ship is at a 45 degree angle to the saucer
7727: 60                        rts

7728: 20 6c 77  LookUpAngle     jsr     FindScrAngleIndex       ;Find the index in the table below for the shot angle
772b: bd 2f 77                  lda     ShotAngleTbl,x
772e: 60                        rts                             ;Look up the proper angle and exit

                ;
                ; The following table divides 45 degrees of a circle into 16 pieces.  Its used
                ; to calculate the direction of a bullet from a small saucer to the player's
                ; ship.  The other angles in the circle are derived from this table.
                ;
772f: 00 02 05 07+ ShotAngleTbl .bulk   $00,$02,$05,$07,$0a,$0c,$0f,$11,$13,$15,$17,$19,$1a,$1c,$1d,$1f

                ;
                ; Draw a string of numbers.
                ;
                • Clear variables
                ]ptr            .var    $00    {addr/2}
```

```
                DrawNumberString
773f: 08                        php                             ;Save carry bit status
7740: 86 17                     stx     ZeroBlankBypass         ;Save flag indicating if Zero blank should be overridden
7742: 88                        dey                             ;Adjust index so it is a zero based index
7743: 84 16                     sty     BCDIndex
7745: 18                        clc
7746: 65 16                     adc     BCDIndex                ;Use index to calculate actual address of BCD data byte
7748: 85 15                     sta     BCDAddress
774a: 28                        plp                             ;Restore carry bit status
774b: aa                        tax                             ;Get address to BCD byte to draw
                ;
                DrawNumStringLoop
774c: 08                        php                             ;Save carry bit status.
774d: b5 00                     lda     ]ptr,x
774f: 4a                        lsr     A
7750: 4a                        lsr     A                       ;Get upper BCD digit to draw
7751: 4a                        lsr     A
7752: 4a                        lsr     A
7753: 28                        plp                             ;Restore carry bit status
7754: 20 85 77                  jsr     SetDigitVecPtr          ;Set vector RAM pointer to digit JSR
7757: a5 16                     lda     BCDIndex                ;Is this the lower byte of the digit string?
7759: d0 01                     bne     DoLowerDigit            ;If so, disable zero blank function
775b: 18                        clc                             ;Draw zeros, if present
                ;
775c: a6 15     DoLowerDigit    ldx     BCDAddress              ;Get lower BCD digit to draw
775e: b5 00                     lda     ]ptr,x
7760: 20 85 77                  jsr     SetDigitVecPtr          ;Set vector RAM pointer to digit JSR
7763: c6 15                     dec     BCDAddress              ;Decrement to next BCD data byte
7765: a6 15                     ldx     BCDAddress
7767: c6 16                     dec     BCDIndex                ;Is there more digits to draw in the number string?
7769: 10 e1                     bpl     DrawNumStringLoop       ;If so, branch to get next digit byte
776b: 60                        rts

                ;
                ; Small saucer shot angle calculation.
                ;
                • Clear variables
                ]ShotAngleTemp  .var    $0b     {addr/1}
                ]ShotXYDistance .var    $0c     {addr/1}

                FindScrAngleIndex
776c: a0 00                     ldy     #$00                    ;Zero out working variable
776e: 84 0b                     sty     ]ShotAngleTemp
7770: a0 04                     ldy     #$04                    ;Prepare to loop 4 times
                ScrAngleIndexLoop
7772: 26 0b                     rol     ]ShotAngleTemp          ;Roll upper bit of working variable into A
7774: 2a                        rol     A
7775: c5 0c                     cmp     ]ShotXYDistance         ;Is A now larger than the given distance?
7777: 90 02                     bcc     UpdateAngleCount        ;If not, branch to do next loop.
7779: e5 0c                     sbc     ]ShotXYDistance         ;Subtract Distance from A to get update proper angle index
                UpdateAngleCount
777b: 88                        dey                             ;Does another loop need to be run?
777c: d0 f4                     bne     ScrAngleIndexLoop       ;If so, branch to do another loop
                ;
777e: a5 0b                     lda     ]ShotAngleTemp          ;Move the final index bit into position
7780: 2a                        rol     A
7781: 29 0f                     and     #$0f                    ;Limit the index to 16 values
7783: aa                        tax
7784: 60                        rts                             ;Done finding angle index

                ;
                ; Score pointer calculation.
                ;
                ; This function can do one of two things:
                ; 1) it can write a command in vector RAM to draw a digit, or
                ; 2) or set a pointer to the next data to process, overriding the zero blanking
                ; function.
                ;
                ; This function is used to blink a zero score at the beginning of a 2 player
                ; game.  If $17 is #$00, draw digit. If it is any other value, get a pointer to
                ; the draw JSR.
                ;
                • Clear variables
                ]ShipDrawXInv   .var    $08     {addr/1}
                ]ShipDrawYInv   .var    $09     {addr/1}
                ]VecPtr         .var    $0b     {addr/2}

7785: 90 04     SetDigitVecPtr  bcc     ChkSetDigitPntr         ;Is zero blanking active? If not, branch
7787: 29 0f                     and     #$0f                    ;Is the digit to draw 0?
7789: f0 27                     beq     DisplayDigit            ;If so, branch
778b: a6 17     ChkSetDigitPntr ldx     ZeroBlankBypass         ;Is the zero blank override flag set?
778d: f0 23                     beq     DisplayDigit            ;If not, branch to draw digit
778f: 29 0f                     and     #$0f
7791: 18                        clc                             ;Add 1 to digit index to skip the SPACE character.
7792: 69 01                     adc     #$01
7794: 08                        php                             ;Save processor status
```

```
7795: 0a                        asl     A                       ;Get lower byte of pointer
7796: a8                        tay                             ;Manually set bits into the proper position.
7797: b9 d4 56                  lda     CharPtrTbl,y
779a: 0a                        asl     A                       ;Store value in lower byte of vector pointer
779b: 85 0b                     sta     ]VecPtr
779d: b9 d5 56                  lda     CharPtrTbl+1,y          ;Get upper byte of pointer
77a0: 2a                        rol     A                       ;Manually set bits into the proper position
77a1: 29 1f                     and     #$1f                    ;Get rid of the opcode bits
77a3: 09 40                     ora     #$40                    ;Set MSB of the address manually
77a5: 85 0c                     sta     ]VecPtr+1               ;Store value in upper byte of vector pointer
                ;
77a7: a9 00                     lda     #$00                    ;Disable XY axis inversion
77a9: 85 08                     sta     ]ShipDrawXInv
77ab: 85 09                     sta     ]ShipDrawYInv
77ad: 20 d7 6a                  jsr     SetVecRAMData           ;Update vector RAM with character data.
77b0: 28                        plp                             ;Restore processor status and exit
77b1: 60                        rts

77b2: 4c cb 7b      DisplayDigit    jmp     PrepDrawDigit       ;Draw a digit on the display

                ;
                ; Random number generator.
                ;
77b5: 06 5f         GetRandNum      asl     RandNum
77b7: 26 60                         rol     RandNum+1           ;Use a shift register to store the random number
77b9: 10 02                         bpl     RandNumBit
77bb: e6 5f                         inc     RandNum             ;Increment lower byte
77bd: a5 5f         RandNumBit      lda     RandNum             ;If the second bit set in the random number?
77bf: 2c d1 77                      bit     RandNumBitTbl
77c2: f0 04                         beq     RandNumORUB         ;If not, branch to move on
77c4: 49 01                         eor     #$01                ;Invert LSB of random number
77c6: 85 5f                         sta     RandNum
77c8: 05 60         RandNumORUB     ora     RandNum+1           ;Is new random number = 0?
77ca: d0 02                         bne     RandNumDone         ;If not, branch to exit
77cc: e6 5f                         inc     RandNum             ;Ensure random number is never 0
77ce: a5 5f         RandNumDone     lda     RandNum             ;Return lower byte or random number
77d0: 60                            rts

77d1: 02            RandNumBitTbl   .dd1    $02                 ;Used by random number generator above

                ;
                ; Thrust calculation routines.
                ;
77d2: 18            CalcXThrust     clc                         ;Adding #$40 to ship/bullet direction will set MSB if facing left
77d3: 69 40                         adc     #$40
77d5: 10 08         CalcThrustDir   bpl     GetVelocityVal      ;Is ship/saucer bullet facing right/up? If so, branch
77d7: 29 7f                         and     #$7f                ;Ship/saucer bullet is facing left/down; cear direction MSB
77d9: 20 df 77                      jsr     GetVelocityVal      ;Get ship/saucer bullet velocity for this XY component
77dc: 4c 08 77                      jmp     TwosCompliment      ;Calculate the 2's compliment of the value in A

77df: c9 41         GetVelocityVal  cmp     #$41                ;Is ship/saucer bullet facing right/up?
77e1: 90 04                         bcc     LookupThrustVal     ;If so, branch
77e3: 49 7f                         eor     #$7f                ;Ship/saucer bullet is facing left/down. Need to lookup
77e5: 69 00                         adc     #$00                ; table in reverse order
77e7: aa            LookupThrustVal tax
77e8: bd b9 57                      lda     ThrustTbl,x         ;Get velocity value from lookup table
77eb: 60                            rts

                ;
                ; Next frame saucer/ship distance.
                ;
                • Clear variables

77ec: 06 0b         NextScrShpDist  asl     GenByte0B
77ee: 2a                            rol     A                   ;Get the signed difference between
77ef: 06 0b                         asl     GenByte0B           ; the ship and saucer upper 4 bits
77f1: 2a                            rol     A
77f2: 38                            sec                         ;Predict next location of saucer with respect to the ship
77f3: e5 0c                         sbc     GenByte0C           ; by subtracting the current saucer XY velocity from the
77f5: 60                            rts                         ; from the saucer/ship distance

                ;
                ; Text writing routines.
                ;
                ]VecRomPtr      .var    $08    {addr/2}

77f6: ad 03 28      WriteText       lda     LanguageSw          ;Get the language dip switch settings
77f9: 29 03                         and     #%00000011
77fb: 0a                            asl     A                   ;*2; 2 bytes per entry in the pointer table below
77fc: aa                            tax                         ;Save index into table in X
77fd: a9 10                         lda     #$10                ;Appears to have no effect
77ff: 85 00                         sta     ZeroPageRam
7801: bd 88 78                      lda     LanguagePtrTbl+1,x
7804: 85 09                         sta     ]VecRomPtr+1        ;Get pointer to language data from the table below
7806: bd 87 78                      lda     LanguagePtrTbl,x
7809: 85 08                         sta     ]VecRomPtr
```

```
780b: 71 08                     adc     ()VecRomPtr),y          ;Add offset to desired text message
780d: 85 08                     sta     ]VecRomPtr
780f: 90 02                     bcc     GetTextPos              ;Does upper byte need to be incremented?
7811: e6 09                     inc     ]VecRomPtr+1            ;If not, branch to move on
                        ;
7813: 98                GetTextPos      tya
7814: 0a                        asl     A                       ;*2; each entry in the table below is 2 bytes
7815: a8                        tay
7816: b9 71 78                  lda     TextPosTbl,y            ;Get the screen position for the desired text
7819: be 72 78                  ldx     TextPosTbl+1,y
781c: 20 03 7c                  jsr     MoveBeam                ;Move the CRT beam to a new location
781f: a9 70                     lda     #$70                    ;Set scale 7(/4)
7821: 20 de 7c                  jsr     SpotKill                ;Draw zero vector to prevent spots on the screen
                        ;
7824: a0 00                     ldy     #$00                    ;Zero out index values
7826: a2 00                     ldx     #$00
7828: a1 08             TextWriteLoop   lda     ()VecRomPtr,x)          ;Get the character byte from ROM
782a: 85 0b                     sta     GenByte0B
782c: 4a                        lsr     A                       ;Move the upper 5 bits into the proper position
782d: 4a                        lsr     A
782e: 20 4d 78                  jsr     TextWriteIncPtr         ;Increment the vector ROM pointer and write to RAM
7831: a1 08                     lda     ()VecRomPtr,x)          ;Get the next character byte from ROM
7833: 2a                        rol     A
7834: 26 0b                     rol     GenByte0B               ;Roll the 2 upper bits into the working variable
7836: 2a                        rol     A
7837: a5 0b                     lda     GenByte0B               ;;Move the next 5 character bits into the proper position
7839: 2a                        rol     A
783a: 0a                        asl     A
783b: 20 53 78                  jsr     CheckNextChar           ;Check if the next character is valid and write to RAM
783e: a1 08                     lda     ()VecRomPtr,x)          ;Get the next text character byte
7840: 85 0b                     sta     GenByte0B
7842: 20 4d 78                  jsr     TextWriteIncPtr         ;Increment the vector ROM pointer
7845: 46 0b                     lsr     GenByte0B               ;Is the last bit 0?
7847: 90 df                     bcc     TextWriteLoop           ;If not, branch to write another character
7849: 88                TextWriteDone   dey                     ;Last byte was end string character, compensate
784a: 4c 39 7c                  jmp     VecPtrUpdate            ;Update Vector RAM pointer

784d: e6 08             TextWriteIncPtr inc     ]VecRomPtr
784f: d0 02                     bne     CheckNextChar           ;Increment vector ROM pointer
7851: e6 09                     inc     ]VecRomPtr+1
7853: 29 3e             CheckNextChar   and     #%00111110              ;Is the data empty? If so, end of string found
7855: d0 04                     bne     PrepWriteChar           ;If not, branch to write character to display
7857: 68                        pla                             ;Pull last return address from stack and update
7858: 68                        pla                             ; the vector RAM pointer
7859: d0 ee                     bne     TextWriteDone
785b: c9 0a             PrepWriteChar   cmp     #$0a                    ;Is the character non-indexed?
785d: 90 02                     bcc     VecRamWriteChar         ;;If so, add offset to get to the indexed characters
785f: 69 0d                     adc     #$0d
7861: aa                VecRamWriteChar tax
7862: bd d2 56                  lda     CharPtrTbl-2,x
7865: 91 02                     sta     (VecRamPtr),y
7867: c8                        iny
7868: bd d3 56                  lda     CharPtrTbl-1,x          ;Store routine for writing desired character in vector RAM
786b: 91 02                     sta     (VecRamPtr),y
786d: c8                        iny
786e: a2 00                     ldx     #$00
7870: 60                        rts

7871: 64 b6             TextPosTbl      .bulk   $64,$b6                 ;X=4*$64=$190=400. Y=4*$B6=$2D8=728
7873: 64 b6                             .bulk   $64,$b6                 ;X=4*$64=$190=400. Y=4*$B6=$2D8=728
7875: 0c aa                             .bulk   $0c,$aa                 ;X=4*$0C=$30 =48.  Y=4*$AA=$2A8=680
7877: 0c a2                             .bulk   $0c,$a2                 ;X=4*$0C=$30 =48.  Y=4*$A2=$288=648
7879: 0c 9a                             .bulk   $0c,$9a                 ;X=4*$0C=$30 =48.  Y=4*$9A=$268=616
787b: 0c 92                             .bulk   $0c,$92                 ;X=4*$0C=$30 =48.  Y=4*$92=$248=584
787d: 64 c6                             .bulk   $64,$c6                 ;X=4*$64=$190=400. Y=4*$C6=$318=792
787f: 64 9d                             .bulk   $64,$9d                 ;X=4*$64=$190=400. Y=4*$9D=$274=628
7881: 50 39                             .bulk   $50,$39                 ;X=4*$50=$140=320. Y=4*$39=$E4 =228
7883: 50 39                             .bulk   $50,$39                 ;X=4*$50=$140=320. Y=4*$39=$E4 =228
7885: 50 39                             .bulk   $50,$39                 ;X=4*$50=$140=320. Y=4*$39=$E4 =228
                        ; Text table pointers.
7887: 1e 57             LanguagePtrTbl  .dd2    EnglishTextTbl
7889: 8f 78                             .dd2    GermanTextTbl           ;Text table pointers
788b: 46 79                             .dd2    FrenchTextTbl
788d: f3 79                             .dd2    SpanishTextTbl
                        ;
                        ; German message offsets.
                        ;
788f: 0b                GermanTextTbl   .dd1    $0b
7890: 15                                .dd1    $15
7891: 1b                                .dd1    $1b
7892: 35                                .dd1    $35
7893: 4d                                .dd1    $4d
7894: 65                                .dd1    $65
7895: 7f                                .dd1    $7f
7896: 8d                                .dd1    $8d
7897: 93                                .dd1    $93
```

```
7898: 9f                        .dd1    $9f
7899: ab                        .dd1    $ab
                ;
                ; Message text, 5 bits per char.  (See comments at $5729.)
                ;
789a: 64 d2 3b 2e+              .bulk   $64,$d2,$3b,$2e,$c2,$6c,$5a,$4c,$93,$6f ;HOECHSTERGEBNIS
78a4: bd 1a 4c 12+              .bulk   $bd,$1a,$4c,$12,$b0,$40 ;SPIELER
78aa: 6b 2c 0a 6c+              .bulk   $6b,$2c,$0a,$6c,$5a,$4c,$93,$6e,$0b,$6e,$c0,$52,$6c,$92,$b8,$50 ;IHR ERGEBNIS IST EINES DER ZEHN BES
                                 +      $4d,$82,$f2,$58,$90,$4c,$4d,$f0,$4c,$80
78c4: 33 70 c2 42+              .bulk   $33,$70,$c2,$42,$5a,$4c,$4c,$82,$bb,$52,$0b,$58,$b2,$42,$6c,$9a ;BITTE GEBEN SIE IHRE INITIALEN EIN
                                 +      $c3,$4a,$82,$64,$0a,$5a,$90,$00
78dc: f6 6c 09 b2+              .bulk   $f6,$6c,$09,$b2,$3b,$2e,$c1,$4c,$4c,$b6,$2b,$20,$0d,$a6,$c1,$70 ;ZUR BUCHSTABENWAHL ROTATE DRUECKEN
                                 +      $48,$50,$b6,$52,$3b,$d2,$90,$00
78f4: da 64 90 4c+              .bulk   $da,$64,$90,$4c,$c9,$d8,$be,$0a,$32,$42,$9b,$c2,$67,$68,$4d,$ae ;WENN BUCHSTABE OK HYPERSPACE DRUECK
                                 +      $a1,$4e,$48,$50,$b6,$52,$3b,$d2,$90,$00
790e: be 0a b6 1e+              .bulk   $be,$0a,$b6,$1e,$94,$d2,$a2,$92,$0a,$2c,$ca,$4e,$7a,$65 ;STARTKNOEPFE DRUECKEN
791c: bd 1a 4c 12+              .bulk   $bd,$1a,$4c,$12,$92,$13 ;SPIELENDE
7922: 18 62 ca 64+              .bulk   $18,$62,$ca,$64,$f2,$42,$20,$6e,$a3,$52,$82,$40 ;1 MUENZE 2 SPIELE
792e: 18 62 ca 64+              .bulk   $18,$62,$ca,$64,$f2,$42,$18,$6e,$a3,$52,$80,$00 ;1 MUENZE 1 SPIEL
793a: 20 62 ca 64+              .bulk   $20,$62,$ca,$64,$f2,$64,$08,$c2,$bd,$1a,$4c,$00 ;2 MUENZEN 1 SPIEL
                ;
                ; French message offsets.
                ;
7946: 0b        FrenchTextTbl   .dd1    $0b
7947: 15                        .dd1    $15
7948: 19                        .dd1    $19
7949: 31                        .dd1    $31
794a: 41                        .dd1    $41
794b: 57                        .dd1    $57
794c: 73                        .dd1    $73
794d: 7f                        .dd1    $7f
794e: 89                        .dd1    $89
794f: 95                        .dd1    $95
7950: a1                        .dd1    $a1
                ;
                ; Message text, 5 bits per char.
                ;
7951: 8a 5a 84 12+              .bulk   $8a,$5a,$84,$12,$cd,$82,$b9,$e6,$b2,$40 ;MEILLEUR SCORE
795b: 74 f2 4d 83               .bulk   $74,$f2,$4d,$83         ;JOUER
795f: d4 f0 b2 42+              .bulk   $d4,$f0,$b2,$42,$b9,$e6,$b2,$42,$4d,$f0,$0e,$64,$0a,$12,$b8,$46 ;VOTRE SCORE EST UN DES 10 MEILLEURS
                                 +      $10,$62,$4b,$60,$82,$72,$b5,$c0
7977: be a8 0a 64+              .bulk   $be,$a8,$0a,$64,$c5,$92,$f0,$74,$9d,$c2,$6c,$9a,$c3,$4a,$82,$6f ;SVP ENTREZ VOS INITIALES
7987: a4 f2 bd d2+              .bulk   $a4,$f2,$bd,$d2,$f0,$6c,$9e,$0a,$c2,$42,$a4,$f2,$b0,$74,$9d,$c2 ;POUSSEZ ROTATE POUR VOS INITIALES
                                 +      $6c,$9a,$c3,$4a,$82,$6f
799d: a4 f2 bd d2+              .bulk   $a4,$f2,$bd,$d2,$f0,$58,$ed,$12,$b5,$e8,$29,$d2,$0d,$72,$2c,$90 ;POUSSEZ HYPERSPACE QUAND LETTRE COR
                                 +      $0c,$12,$c6,$2c,$48,$4e,$9d,$ac,$49,$f0,$48,$00
79b9: 2d 28 cf 52+              .bulk   $2d,$28,$cf,$52,$b0,$6e,$cd,$82,$be,$0a,$b6,$00 ;APPUYER SUR START
79c5: 53 64 0a 12+              .bulk   $53,$64,$0a,$12,$0d,$0a,$b6,$1a,$48,$00 ;FIN DE PARTIE
79cf: 18 68 6a 4e+              .bulk   $18,$68,$6a,$4e,$48,$48,$0b,$a6,$ca,$72,$b5,$c0 ;1 PIECE 2 JOUEURS
79db: 18 68 6a 4e+              .bulk   $18,$68,$6a,$4e,$48,$46,$0b,$a6,$ca,$72,$b0,$00 ;1 PIECE 1 JOUEUR
79e7: 20 68 6a 4e+              .bulk   $20,$68,$6a,$4e,$4d,$c2,$18,$5c,$9e,$52,$cd,$80 ;2 PIECES 1 JOUEUR
                ;
                ; Spanish message offsets.
                ;
79f3: 0b        SpanishTextTbl  .dd1    $0b
79f4: 11                        .dd1    $11
79f5: 17                        .dd1    $17
79f6: 31                        .dd1    $31
79f7: 45                        .dd1    $45
79f8: 5f                        .dd1    $5f
79f9: 6b                        .dd1    $6b
79fa: 73                        .dd1    $73
79fb: 7d                        .dd1    $7d
79fc: 89                        .dd1    $89
79fd: 93                        .dd1    $93
                ;
                ; Message text, 5 bits per char.
                ;
79fe: b2 4e 9d 90+              .bulk   $b2,$4e,$9d,$90,$b8,$00 ;RECORDS
7a04: 76 56 2a 26+              .bulk   $76,$56,$2a,$26,$b0,$40 ;JUGADOR
7a0a: be 42 a6 64+              .bulk   $be,$42,$a6,$64,$c1,$5c,$48,$52,$be,$0a,$0a,$64,$c5,$92,$0c,$26 ;SU PUNTAJE ESTA ENTRE LOS DIEZ MEJO
                                 +      $b8,$50,$6a,$7c,$0c,$52,$74,$ec,$4d,$c0
7a24: a4 ec 0a 8a+              .bulk   $a4,$ec,$0a,$8a,$d4,$ec,$0a,$64,$c5,$92,$0d,$f2,$b8,$5a,$93,$4e ;POR FAVOR ENTRE SUS INICIALES
                                 +      $69,$60,$4d,$c0
7a38: 9d 2c 6c 4a+              .bulk   $9d,$2c,$6c,$4a,$0d,$a6,$c1,$70,$48,$68,$2d,$8a,$0d,$d2,$82,$4e ;OPRIMA ROTATE PARA SELECCIONAR LA L
                                 +      $3b,$66,$91,$6c,$0c,$0a,$0c,$12,$c5,$8b
7a52: 9d 2c 6c 4a+              .bulk   $9d,$2c,$6c,$4a,$0b,$3a,$a2,$6c,$bd,$0a,$3a,$40 ;OPRIMA HYPERSPACE
7a5e: a6 60 b9 6c+              .bulk   $a6,$60,$b9,$6c,$0d,$f0,$2d,$b1 ;PULSAR START
7a66: 76 52 5c c2+              .bulk   $76,$52,$5c,$c2,$c2,$6c,$8b,$64,$2a,$27 ;JUEGO TERMINADO
7a70: 18 54 69 d8+              .bulk   $18,$54,$69,$d8,$28,$48,$0b,$b2,$4a,$e6,$b8,$00 ;1 FICHA 2 JUEGOS
7a7c: 18 54 69 d8+              .bulk   $18,$54,$69,$d8,$28,$46,$0b,$b2,$4a,$e7 ;1 FICHA 1 JUEGO
7a86: 20 54 69 d8+              .bulk   $20,$54,$69,$d8,$2d,$c2,$18,$5c,$ca,$56,$98,$00 ;2 FICHAS 1 JUEGO
                ;
7a92: 52                        .dd1    $52                     ;checksum byte

                ;
```

```
                ; Check coin insertion routines.
                ;
                CheckCoinsInserted
7a93: a2 02                     ldx     #$02                    ;Prepare to check all 3 coin mechanisms
7a95: bd 00 24  CheckCoinsLoop  lda     LeftCoinSw,x            ;Get status of coin switch and store it in the carry bit
7a98: 0a                        asl     A
7a99: b5 7a                     lda     CoinDropTimers,x        ;Get coin drop timer value
7a9b: 29 1f                     and     #$1f                    ;Was a coin insertion detected?
7a9d: 90 37                     bcc     CheckDropTimerVal       ;If not, branch to check coin drop timer
7a9f: f0 10                     beq     CheckSlamSw             ;Has coin drop timer run until it hit 0? If so, branch
7aa1: c9 1b                     cmp     #$1b                    ;Has timer just started with detected coin insertion?
7aa3: b0 0a                     bcs     DecDropTimer            ;If so, branch
7aa5: a8                        tay                             ;Wait 7 NMI periods(28ms). During this time, the coin
7aa6: a5 5e                     lda     NmiCounter              ; switch should be active, the drop timer should be active
7aa8: 29 07                     and     #$07                    ; and the slam switch should not be active. If these
7aaa: c9 07                     cmp     #$07                    ; conditions are true, decrement the coin drop timer
7aac: 98                        tya
7aad: 90 02                     bcc     CheckSlamSw             ;Check slam switch during first 7 NMIs
7aaf: e9 01     DecDropTimer    sbc     #$01                    ;Things check out so far, decrement coin drop timer
7ab1: 95 7a     CheckSlamSw     sta     CoinDropTimers,x        ;Update the coin drop timer.
                ;
7ab3: ad 06 20                  lda     SlamSw                  ;Get the slam switch status
7ab6: 29 80                     and     #$80                    ;Was a slam detected?
7ab8: f0 04                     beq     CheckSlamTimer          ;If not, branch to move on
7aba: a9 f0                     lda     #$f0                    ;Slam detected. Set slam timer
7abc: 85 72                     sta     SlamTimer
7abe: a5 72     CheckSlamTimer  lda     SlamTimer               ;Is the slam timer active?
7ac0: f0 08                     beq     CheckWaitTimer          ;If not, branch to move on
7ac2: c6 72                     dec     SlamTimer
7ac4: a9 00                     lda     #$00                    ;Decrement the slam timer and hold the other timers
7ac6: 95 7a                     sta     CoinDropTimers,x        ; in their zero state until the slam timer clears
7ac8: 95 77                     sta     WaitCoinTimers,x
7aca: 18        CheckWaitTimer  clc                             ;Is this wait timer finished?
7acb: b5 77                     lda     WaitCoinTimers,x
7acd: f0 23                     beq     CheckNextMech           ;If so, branch to see if another mechanism needs to be checked
7acf: d6 77                     dec     WaitCoinTimers,x        ;Is this timer still active? decrement and branch if done
7ad1: d0 1f                     bne     CheckNextMech
7ad3: 38                        sec                             ;Branch always
7ad4: b0 1c                     bcs     CheckNextMech

                CheckDropTimerVal
7ad6: c9 1b                     cmp     #$1b                    ;If timer is a high value, the coin switch cleared too
7ad8: b0 09                     bcs     ResetDropTimer          ; soon. False flag. Branch to reset the drop timer
7ada: b5 7a                     lda     CoinDropTimers,x        ;Max value after add is #$3F
7adc: 69 20                     adc     #$20
7ade: 90 d1                     bcc     CheckSlamSw             ;Branch always
7ae0: f0 01                     beq     ResetDropTimer          ;This code does not appear to be accessed
7ae2: 18                        clc
7ae3: a9 1f     ResetDropTimer  lda     #$1f                    ;Prepare to reset the coin drop timer
7ae5: b0 ca                     bcs     CheckSlamSw             ;If carry is set, something funny happened, check slam switch
7ae7: 95 7a                     sta     CoinDropTimers,x        ;Reset the coin drop timer
7ae9: b5 77                     lda     WaitCoinTimers,x        ;is this the first transition of the wait timer?
7aeb: f0 01                     beq     SetWaitTimer            ;If so, branch to set timer and move to next coin mech
7aed: 38                        sec                             ;Timer transition already happened, prepare to do more processing
7aee: a9 78     SetWaitTimer    lda     #$78                    ;Load the wait timer
7af0: 95 77                     sta     WaitCoinTimers,x
7af2: 90 23     CheckNextMech   bcc     DoNextCoinMech          ;Branch to check next coin mech if timer transition just happened
7af4: a9 00                     lda     #$00                    ;Is this the left coin mech?
7af6: e0 01                     cpx     #$01
7af8: 90 16                     bcc     CalcMult                ;If so, branch to increment coins; no multipliers this coin mech
7afa: f0 0c                     beq     CCoinMechMult           ;Is this the center coin mech? If so, branch to calc multiplier
                ;
7afc: a5 71                     lda     DipSwitchBits           ;Only option left is the right coin mechanism
7afe: 29 0c                     and     #$0c
7b00: 4a                        lsr     A                       ;Get the Dip switch values and /4
7b01: 4a                        lsr     A
7b02: f0 0c                     beq     CalcMult                ;If no multiplier active, branch to increment coins
7b04: 69 02                     adc     #$02                    ;Multiplier active on the right coin mech. Get the shifted
7b06: d0 08                     bne     CalcMult                ; DIP switch value and add 2 for a range between 4-6; Branch always

7b08: a5 71     CCoinMechMult   lda     DipSwitchBits           ;Check if there is a multiplier active on the center coin mech.
7b0a: 29 10                     and     #$10
7b0c: f0 02                     beq     CalcMult                ;If not, branch to increment the coins
7b0e: a9 01                     lda     #$01                    ;Multiplier active; add an additional coin
7b10: 38        CalcMult        sec                             ;Add at least one coin
7b11: 65 73                     adc     CoinMult                ;Add the any others from multipliers
7b13: 85 73                     sta     CoinMult                ;Update the total coins
7b15: f6 74                     inc     ValidCoins,x            ;Indicate a valid coin; used for incrementing coin counter
7b17: ca        DoNextCoinMech  dex                             ;Are there coin mechanisms left to check?
7b18: 30 03                     bmi     CalcCoinsPerPlay        ;If not, next step is to update coins
7b1a: 4c 95 7a                  jmp     CheckCoinsLoop          ;Check next coin mechanism

                CalcCoinsPerPlay
7b1d: a5 71                     lda     DipSwitchBits           ;Get the coins per play value; On = 0, Off = 1
7b1f: 29 03                     and     #$03
7b21: a8                        tay                             ;Is free play active?
```

```
7b22: f0 12                     beq     UpdateCoinMult          ;If so, branch to add 0 coins
7b24: 4a                        lsr     A
7b25: 69 00                     adc     #$00                    ;Get the number of coins required to get a credit
7b27: 49 ff                     eor     #$ff                    ; and subtract the number of current coins; if more
7b29: 38                        sec                             ; coins are needed, branch to finish for this frame
7b2a: 65 73                     adc     CoinMult                ;Else add up to 2 credits this frame
7b2c: 90 0a                     bcc     CreditUpdateDone
7b2e: c0 02                     cpy     #$02                    ;Do 2 credits need to be added?
7b30: b0 02                     bcs     Add1Credit              ;If not, branch to add only 1
7b32: e6 70                     inc     NumCredits              ;Add the first of 2 credits
7b34: e6 70             Add1Credit      inc     NumCredits      ;Increment the credits
7b36: 85 73             UpdateCoinMult  sta     CoinMult                ;Store updated coin value
                        ;
                        CreditUpdateDone
7b38: a5 5e                     lda     NmiCounter              ;Is this an odd NMI period?
7b3a: 4a                        lsr     A
7b3b: b0 27                     bcs     EndCoinCheck            ;If so, branch to end, if not, keep processing
7b3d: a0 00                     ldy     #$00                    ;Prepare to check all 3 valid coin indicators
7b3f: a2 02                     ldx     #$02
7b41: b5 74             ValidCoinLoop1  lda     ValidCoins,x
7b43: f0 09                     beq     NextValidCoin1
7b45: c9 10                     cmp     #$10                    ;This function continues a valid coin timer
7b47: 90 05                     bcc     NextValidCoin1          ;During this time, the coin counters are enabled
7b49: 69 ef                     adc     #$ef                    ;The counter will last for 16 NMIs when a single
7b4b: c8                        iny                             ; coin is inserted; the counter will last longer
7b4c: 95 74                     sta     ValidCoins,x            ; if more coins are added
7b4e: ca                NextValidCoin1  dex
7b4f: 10 f0                     bpl     ValidCoinLoop1
7b51: 98                        tya                             ;Is a valid coin counter active from above?
7b52: d0 10                     bne     EndCoinCheck
                        ;
7b54: a2 02                     ldx     #$02                    ;Prepare to check all 3 valid coin indicators
7b56: b5 74             ValidCoinLoop2  lda     ValidCoins,x
7b58: f0 07                     beq     NextValidCoin2
7b5a: 18                        clc
7b5b: 69 ef                     adc     #$ef                    ;This function will initiate a valid coin
7b5d: 95 74                     sta     ValidCoins,x            ; timer; the coin counters will be enabled
7b5f: 30 03                     bmi     EndCoinCheck            ; at this time
7b61: ca                NextValidCoin2  dex
7b62: 10 f2                     bpl     ValidCoinLoop2
7b64: 60                EndCoinCheck    rts                             ;Done checking coin insertion

                        ;
                        ; NMI handler.  These arrive at 3000/12 = 250Hz.
                        ;
7b65: 48                NMI             pha                             ;Push A, Y and X onto the stack
7b66: 98                                tya
7b67: 48                                pha
7b68: 8a                                txa
7b69: 48                                pha
7b6a: d8                                cld                             ;Set processor to binary mode
7b6b: ad ff 01                  lda     StackBottom             ;Has the stack overflowed or underflowed?
7b6e: 0d d0 01                  ora     StackTop                ;If so, spin lock until watchdog reset
7b71: d0 fe             :Spin           bne     :Spin
                        ;
7b73: e6 5e                     inc     NmiCounter              ;Is it time to start a new frame(every 4th NMI)?
7b75: a5 5e                     lda     NmiCounter
7b77: 29 03                     and     #$03
7b79: d0 08                     bne     CheckCoins              ;If not, branch to skip frame counter increment
7b7b: e6 5b                     inc     FrameCounter            ;Start a new frame. 62.5 frames per second
7b7d: a5 5b                     lda     FrameCounter            ;Have more than 3 frames passed without being acknowledged?
7b7f: c9 04                     cmp     #$04
7b81: b0 fe             :Spin           bcs     :Spin                   ;If so, something is wrong. Spin lock until watchdog reset
                        ;
7b83: 20 93 7a          CheckCoins      jsr     CheckCoinsInserted      ;Check if player inserted any coins
7b86: a5 6f                     lda     MultiPurpBits           ;Get the multipurpose bits and discard the coin
7b88: 29 c7                     and     #%11000111              ; counter enable bits. They will be set next
7b8a: 24 74                     bit     LValidCoin              ;Was a valid coin detected in the left coin mech?
7b8c: 10 02                     bpl     CheckCValidCoin         ;If not, branch to check the next coin mech
7b8e: 09 08                     ora     #CoinCtrLeft            ;Activate the left coin counter
7b90: 24 75             CheckCValidCoin bit     CValidCoin              ;Was a valid coin detected in the center coin mech?
7b92: 10 02                     bpl     CheckRValidCoin         ;If not, branch to check the next coin mech
7b94: 09 10                     ora     #CoinCtrCntr            ;Activate the center coin counter.
7b96: 24 76             CheckRValidCoin bit     RValidCoin              ;Was a valid coin detected in the right coin mech?
7b98: 10 02                     bpl     UpdateCoinCounters      ;If not, branch to update the active coin counters
7b9a: 09 20                     ora     #CoinCtrRght            ;Activate the right coin counter
                        UpdateCoinCounters
7b9c: 85 6f                     sta     MultiPurpBits           ;Update the current states of the coin counters
7b9e: 8d 00 32                  sta     MultiPurp
                        ;
7ba1: a5 72                     lda     SlamTimer               ;Is slam timer active?
7ba3: f0 04                     beq     UpdateSlamSFX           ;If not, branch
7ba5: a9 80                     lda     #$80                    ;Slam detected; start the slam SFX
7ba7: d0 0e                     bne     EnDisSlamSFX

7ba9: a5 68             UpdateSlamSFX   lda     ExLfSFXTimer            ;Slam has not recently been active; disable slam SFX
```

```
7bab: f0 0a                     beq     EnDisSlamSFX
7bad: a5 5c                     lda     FrameTimer              ;Is this an odd frame?
7baf: 6a                        ror     A
7bb0: 90 02                     bcc     FrameTimerRoll3         ;If not, branch to skip decrementing SFX timer
7bb2: c6 68                     dec     ExLfSFXTimer            ;Decrement SFX timer every other frame
7bb4: 6a            FrameTimerRoll3 ror     A
7bb5: 6a                        ror     A                       ;Rolling the value creates a unique SFX
7bb6: 6a                        ror     A
7bb7: 8d 05 3c      EnDisSlamSFX    sta     LifeSFX                 ;Enables or disables slam SFX
7bba: 68                        pla                             ;Pull A, Y and X from the stack
7bbb: aa                        tax
7bbc: 68                        pla
7bbd: a8                        tay
7bbe: 68                        pla
7bbf: 40                        rti                             ;Return from interrupt

                    ;
                    ; Vector drawing routines.
                    ;
7bc0: a9 b0         VecHalt         lda     #HaltOpcode             ;Write HALT command to vector RAM
7bc2: a0 00         VecRam2Write    ldy     #$00                    ;Write 2 bytes to vector RAM
7bc4: 91 02                         sta     (VecRamPtr),y
7bc6: c8                            iny
7bc7: 91 02                         sta     (VecRamPtr),y
7bc9: d0 6e                         bne     VecPtrUpdate            ;Branch always; update Vector RAM pointer
7bcb: 90 04         PrepDrawDigit   bcc     DrawDigit               ;Draw a single digit on the display
7bcd: 29 0f                         and     #$0f                    ;Is a blank space to be drawn?
7bcf: f0 05                         beq     PrepDigitPointer        ;If so, branch
7bd1: 29 0f         DrawDigit       and     #$0f                    ;Save lower nibble and add 1 to it
7bd3: 18                            clc
7bd4: 69 01                         adc     #$01                    ;Adding 1 skips the "space" character.
                    ;
                    PrepDigitPointer
7bd6: 08                            php                             ;Save the processor status on the stack
7bd7: 0a                            asl     A                       ;*2; the digit pointers are 2 bytes
7bd8: a0 00                         ldy     #$00                    ;Start at current vector RAM pointer position
7bda: aa                            tax
7bdb: bd d4 56                      lda     CharPtrTbl,x            ;Load the JSRL command that draws the appropriate digit
7bde: 91 02                         sta     (VecRamPtr),y
7be0: bd d5 56                      lda     CharPtrTbl+1,x
7be3: c8                            iny
7be4: 91 02                         sta     (VecRamPtr),y
7be6: 20 39 7c                      jsr     VecPtrUpdate            ;Update Vector RAM pointer
7be9: 28                            plp                             ;Restore the processor status from the stack
7bea: 60                            rts

7beb: 4a            UnusedFunc00    lsr     A
7bec: 29 0f                         and     #$0f                    ;Appears to be an unused function
7bee: 09 e0                         ora     #$e0
                    ;
7bf0: a0 01         VecRamPtrUpdate ldy     #$01                    ;Load upper byte of JSRL word into vector RAM
7bf2: 91 02                         sta     (VecRamPtr),y
7bf4: 88                            dey                             ;Decrement index to load lower byte
7bf5: 8a                            txa
7bf6: 6a                            ror     A                       ;Convert byte into proper address format
7bf7: 91 02                         sta     (VecRamPtr),y           ;Store lower byte
7bf9: c8                            iny                             ;Increment index for proper JSRL return address
7bfa: d0 3d                         bne     VecPtrUpdate            ;Update Vector RAM pointer

7bfc: 4a            VecRomJSRL      lsr     A                       ;Shift right to preserve JSRL upper address bit
7bfd: 29 0f                         and     #$0f                    ;Keep upper address nibble
7bff: 09 c0                         ora     #JsrlOpcode             ;Add JSRL opcode
7c01: d0 ed                         bne     VecRamPtrUpdate         ;Branch always; update vector RAM with JSR

                    • Clear variables
                    ]GlobalScale    .var    $00    {addr/1}
                    ]MovBeamX       .var    $04    {addr/2}
                    ]MovBeamY       .var    $06    {addr/2}

7c03: a0 00         MoveBeam        ldy     #$00
7c05: 84 05                         sty     ]MovBeamX+1             ;Zero out X and Y upper address bytes
7c07: 84 07                         sty     ]MovBeamY+1
7c09: 0a                            asl     A
7c0a: 26 05                         rol     ]MovBeamX+1
7c0c: 0a                            asl     A                       ;Break X address byte into the proper opcode format
7c0d: 26 05                         rol     ]MovBeamX+1
7c0f: 85 04                         sta     ]MovBeamX
7c11: 8a                            txa                             ;Move Y address byte into A
7c12: 0a                            asl     A
7c13: 26 07                         rol     ]MovBeamY+1
7c15: 0a                            asl     A                       ;Break Y address byte into the proper opcode format
7c16: 26 07                         rol     ]MovBeamY+1
7c18: 85 06                         sta     ]MovBeamY
                    ;
7c1a: a2 04                         ldx     #$04                    ;Prepare to load 4 bytes into vector RAM
7c1c: b5 02         SetLABSData     lda     VecRamPtr,x             ;Get lower byte of upper LABS word
```

```
7c1e: a0 00                     ldy     #$00
7c20: 91 02                     sta     (VecRamPtr),y           ;Store it in vector RAM
7c22: b5 03                     lda     VecRamPtr+1,x           ;Get upper byte of upper LABS word
7c24: 29 0f                     and     #$0f
7c26: 09 a0                     ora     #LabsOpcode             ;Add LABS opcode to the LABS instruction
7c28: c8                        iny
7c29: 91 02                     sta     (VecRamPtr),y           ;Store it in vector RAM
7c2b: b5 00                     lda     VecRamPtr-2,x           ;Get lower byte of lower LABS word
7c2d: c8                        iny
7c2e: 91 02                     sta     (VecRamPtr),y           ;Store it in vector RAM
7c30: b5 01                     lda     VecRamPtr-1,x           ;Get upper byte of lower LABS word
7c32: 29 0f                     and     #$0f
7c34: 05 00                     ora     ]GlobalScale            ;Add global scale data to the CUR instruction
7c36: c8                        iny
7c37: 91 02                     sta     (VecRamPtr),y           ;Store it in vector RAM
7c39: 98            VecPtrUpdate    tya                             ;Y has the number of bytes to increment vector ROM pointer by
7c3a: 38                        sec
7c3b: 65 02                     adc     VecRamPtr               ;Update vector ROM pointer
7c3d: 85 02                     sta     VecRamPtr
7c3f: 90 02                     bcc     :Return                 ;Does upper byte of pointer need to increment? if not, branch
7c41: e6 03                     inc     VecRamPtr+1             ;Increment upper pointer byte
7c43: 60            :Return         rts

                    ; Unused?
7c44: a9 d0         VecRamRTS       lda     #RtslOpcode             ;Prepare to write RTSL opcode to vector RAM
7c46: 4c c2 7b                  jmp     VecRam2Write            ;Write the same byte twice to vector RAM

                    ;
                    ; Calculate ship debris position.
                    ;
                    ; The purpose of this function is to calculate the proper starting point for the
                    ; selected piece of ship debris.  It needs to take into account any out of
                    ; bounds conditions if the ship is close to any of the 4 edges of the display.
                    ; It draws a VCTR with zero brightness to the proper starting position.
                    ;
                    • Clear variables
                    ]GenByte01      .var    $01    {addr/1}
                    ]ThisDebrisX    .var    $04    {addr/2}
                    ]ThisDebrisY    .var    $06    {addr/2}
                    ]GenByte08      .var    $08    {addr/1}

7c49: a5 05         CalcDebrisPos   lda     ]ThisDebrisX+1          ;Is the debris traveling in a negative X direction?
7c4b: c9 80                     cmp     #$80
7c4d: 90 11                     bcc     ChkYDebris              ;If not, branch
7c4f: 49 ff                     eor     #$ff                    ;Convert negative direction into a positive
7c51: 85 05                     sta     ]ThisDebrisX+1          ; number by using two's compliment
7c53: a5 04                     lda     ]ThisDebrisX
7c55: 49 ff                     eor     #$ff                    ;Lower byte contains debris absolute value position
7c57: 69 00                     adc     #$00                    ;Upper byte contains debris direction
7c59: 85 04                     sta     ]ThisDebrisX
7c5b: 90 02                     bcc     :NoInc
7c5d: e6 05                     inc     ]ThisDebrisX+1
7c5f: 38            :NoInc          sec                             ;Set bit to indicate debris is moving in negative X direction
7c60: 26 08         ChkYDebris      rol     ]GenByte08              ;Save X direction bit
7c62: a5 07                     lda     ]ThisDebrisY+1          ;Is the debris traveling in a negative Y direction?
7c64: c9 80                     cmp     #$80
7c66: 90 11                     bcc     ChkPosXYUB              ;If not, branch
7c68: 49 ff                     eor     #$ff                    ;Convert negative direction into a positive
7c6a: 85 07                     sta     ]ThisDebrisY+1          ; number by using two's compliment
7c6c: a5 06                     lda     ]ThisDebrisY
7c6e: 49 ff                     eor     #$ff                    ;Lower byte contains debris absolute value position
7c70: 69 00                     adc     #$00                    ;Upper byte contains debris direction
7c72: 85 06                     sta     ]ThisDebrisY
7c74: 90 02                     bcc     :NoInc
7c76: e6 07                     inc     ]ThisDebrisY+1
7c78: 38            :NoInc          sec                             ;Set bit to indicate debris is moving in negative Y direction
7c79: 26 08         ChkPosXYUB      rol     ]GenByte08              ;Save Y direction bit
7c7b: a5 05                     lda     ]ThisDebrisX+1          ;Is debris piece close to the lowest X or Y border?
7c7d: 05 07                     ora     ]ThisDebrisY+1
7c7f: f0 0a                     beq     ChkPosXYLB              ;If so, branch to check lower byte for edge proximity
7c81: a2 00                     ldx     #$00                    ;Prepare to clip debris if at max XY position
7c83: c9 02                     cmp     #$02                    ;Is debris at maximum XY edge of screen?
7c85: b0 24                     bcs     SetScaleAndDirBits      ;If so, branch
7c87: a0 01                     ldy     #$01                    ;Not at edge of screen; prepare to calculate proper scaling
7c89: d0 10                     bne     PrepXYMult2

7c8b: a0 02         ChkPosXYLB      ldy     #$02                    ;Prepare to set scaling if not at screen edge
7c8d: a2 09                     ldx     #$09                    ;Prepare to clip debris if at min XY position
7c8f: a5 04                     lda     ]ThisDebrisX
7c91: 05 06                     ora     ]ThisDebrisY            ;Is debris at minimum XY edge of screen?
7c93: f0 16                     beq     SetScaleAndDirBits      ;If so, branch
7c95: 30 04                     bmi     PrepXYMult2             ;Is proper scaling already set? If so, branch
7c97: c8            CalcShiftVal    iny
7c98: 0a                        asl     A                       ;Calculate proper scaling value for displacing this debris
7c99: 10 fc                     bpl     CalcShiftVal
                    ;
```

```
7c9b: 98          PrepXYMult2     tya                         ;Transfer scaling value to X
7c9c: aa                          tax
7c9d: a5 05                       lda     ]ThisDebrisX+1      ;Move debris X direction into A
                  RestoreDebrisPos
7c9f: 06 04                       asl     ]ThisDebrisX
7ca1: 2a                          rol     A
7ca2: 06 06                       asl     ]ThisDebrisY        ;Restore the debris position from a single byte back to 2 bytes
7ca4: 26 07                       rol     ]ThisDebrisY+1
7ca6: 88                          dey
7ca7: d0 f6                       bne     RestoreDebrisPos
7ca9: 85 05                       sta     ]ThisDebrisX+1      ;Save restored upper byte of debris X position
                  ;
                  SetScaleAndDirBits
7cab: 8a                          txa
7cac: 38                          sec
7cad: e9 0a                       sbc     #$0a                ;Compute scaling bits
7caf: 49 ff                       eor     #$ff
7cb1: 0a                          asl     A
7cb2: 66 08                       ror     ]GenByte08          ;Get Y direction bit
7cb4: 2a                          rol     A
7cb5: 66 08                       ror     ]GenByte08
7cb7: 2a                          rol     A                   ;Get X direction bit
7cb8: 0a                          asl     A
7cb9: 85 08                       sta     ]GenByte08          ;Save the completed configuration bits back to RAM
                  ;
7cbb: a0 00                       ldy     #$00                ;Write the Y position lower byte to vector RAM
7cbd: a5 06                       lda     ]ThisDebrisY
7cbf: 91 02                       sta     (VecRamPtr),y
7cc1: a5 08                       lda     ]GenByte08
7cc3: 29 f4                       and     #%11110100          ;Get the scale and Y direction bits for the VCTR opcode
7cc5: 05 07                       ora     ]ThisDebrisY+1      ;Combine the Y position upper byte
7cc7: c8                          iny
7cc8: 91 02                       sta     (VecRamPtr),y       ;Write the byte to vector RAM
7cca: a5 04                       lda     ]ThisDebrisX
7ccc: c8                          iny
7ccd: 91 02                       sta     (VecRamPtr),y       ;Write the X position lower byte to vector RAM
7ccf: a5 08                       lda     ]GenByte08
7cd1: 29 02                       and     #$02                ;Get the X direction bit
7cd3: 0a                          asl     A
7cd4: 05 01                       ora     ]GenByte01          ;Set brightness for this vector(should be 0)
7cd6: 05 05                       ora     ]ThisDebrisX+1      ;Combine the X position upper byte
7cd8: c8                          iny
7cd9: 91 02                       sta     (VecRamPtr),y       ;Write the byte to vector RAM
7cdb: 4c 39 7c                    jmp     VecPtrUpdate        ;Update Vector RAM pointer

                  ;
                  ; Spot kill.
                  ;
7cde: a2 00       SpotKill        ldx     #$00                ;Prepare to draw a dot with brightness 0
7ce0: a0 01       DrawDot         ldy     #$01                ;Store scale in vector RAM
7ce2: 91 02                       sta     (VecRamPtr),y
7ce4: 88                          dey
7ce5: 98                          tya
7ce6: 91 02                       sta     (VecRamPtr),y       ;Set X and Y delta values to 0
7ce8: c8                          iny
7ce9: c8                          iny
7cea: 91 02                       sta     (VecRamPtr),y
7cec: c8                          iny
7ced: 8a                          txa                         ;Store dot brightness in vector RAM
7cee: 91 02                       sta     (VecRamPtr),y
7cf0: 4c 39 7c                    jmp     VecPtrUpdate        ;Update Vector RAM pointer

                  ;
                  ; Reset.
                  ;
7cf3: a2 fe       RESET           ldx     #$fe                ;Set stack pointer to #$FE
7cf5: 9a                          txs
7cf6: d8                          cld                         ;Set processor to binary mode
7cf7: a9 00                       lda     #$00                ;Prepare to clear the RAM
7cf9: aa                          tax
7cfa: ca          RamClearLoop    dex                         ;Loop until all RAM is zeroed
7cfb: 9d 00 03                    sta     Player2Ram,x
7cfe: 9d 00 02                    sta     Player1Ram,x
7d01: 9d 00 01                    sta     OnePageRam,x
7d04: 95 00                       sta     ZeroPageRam,x
7d06: d0 f2                       bne     RamClearLoop        ;More bytes to clear? If so, loop to write more
                  ;
7d08: ac 07 20                    ldy     SelfTestSw          ;Is the self test switch set for test mode?
7d0b: 30 43                       bmi     DoSelfTest          ;If so, branch to do self test routine
7d0d: e8                          inx                         ;Write JUMP to RAM address $4402 opcode to vector RAM
7d0e: 8e 00 40                    stx     VectorRam           ;The vector RAM is divided in half and one half is written to
7d11: a9 e2                       lda     #JmplOpcode+2       ; while the other half is read. The read/write halves are
7d13: 8d 01 40                    sta     VectorRam+1         ; swapped every frame
7d16: a9 b0                       lda     #HaltOpcode
7d18: 8d 03 40                    sta     VectorRam+3         ;Write HALT opcode to vector RAM address $4002
                  ;
```

```
7d1b: 85 32                     sta     Plyr1Rank               ;Write some initial data to player's rank
7d1d: 85 33                     sta     Plyr2Rank
7d1f: a9 03                     lda     #PlyrLamps
7d21: 85 6f                     sta     MultiPurpBits           ;Turn on the Player 1 and 2 LEDs
7d23: 8d 00 32                  sta     MultiPurp
7d26: 2d 00 28                  and     PlayTypeSw              ;Get how many coins to play a game
7d29: 85 71                     sta     DipSwitchBits
7d2b: ad 01 28                  lda     RghtCoinMechSw
7d2e: 29 03                     and     #%00000011
7d30: 0a                        asl     A                       ;Get the coin multiplier for the Right coin mech
7d31: 0a                        asl     A
7d32: 05 71                     ora     DipSwitchBits
7d34: 85 71                     sta     DipSwitchBits
7d36: ad 02 28                  lda     CentCMShipsSw
7d39: 29 02                     and     #$02
7d3b: 0a                        asl     A
7d3c: 0a                        asl     A                       ;Get the coin multiplier for the center coin mech
7d3d: 0a                        asl     A
7d3e: 05 71                     ora     DipSwitchBits
7d40: 85 71                     sta     DipSwitchBits
7d42: 4c 03 68                  jmp     InitGame                ;Initialize the game after reset

7d45: a0 00             VecWriteWord    ldy     #$00            ;Write 2 bytes into vector RAM
7d47: 91 02                     sta     (VecRamPtr),y
7d49: c8                        iny
7d4a: 8a                        txa
7d4b: 91 02                     sta     (VecRamPtr),y
7d4d: 4c 39 7c                  jmp     VecPtrUpdate            ;Update Vector RAM pointer

                        ;
                        ; Self test routines.
                        ;
7d50: 9d 00 40          DoSelfTest      sta     VectorRam,x             ;Loop and clear all 2K of vector RAM
7d53: 9d 00 41                  sta     VectorRam+$100,x
7d56: 9d 00 42                  sta     VectorRam+$200,x
7d59: 9d 00 43                  sta     VectorRam+$300,x
7d5c: 9d 00 44                  sta     VectorRam+$400,x
7d5f: 9d 00 45                  sta     VectorRam+$500,x
7d62: 9d 00 46                  sta     VectorRam+$600,x
7d65: 9d 00 47                  sta     VectorRam+$700,x
7d68: e8                        inx
7d69: d0 e5                     bne     DoSelfTest              ;More RAM to clear? If so, branch
7d6b: 8d 00 34                  sta     WdClear                 ;Clear the watchdog timer
                        ;
7d6e: a2 00                     ldx     #$00                    ;Prepare for RAM check test
                        RamPage0TestLoop
7d70: b5 00                     lda     ZeroPageRam,x           ;RAM address to check should always start out as 0
7d72: d0 47                     bne     RamPage0Fail
7d74: a9 11                     lda     #$11                    ;Four bit RAM; load a single bit per RAM
                        RamPage0ByteTest
7d76: 95 00                     sta     ZeroPageRam,x           ;Store the bit pattern in RAM
7d78: a8                        tay                             ;Read the value back out of RAM
7d79: 55 00                     eor     ZeroPageRam,x           ;Compare it with itself
7d7b: d0 3e                     bne     RamPage0Fail            ;Is the value the same? If not, branch to failure
7d7d: 98                        tya                             ;Rotate the bit pattern in the RAM
7d7e: 0a                        asl     A
7d7f: 90 f5                     bcc     RamPage0ByteTest        ;More bits to test at this address? If so, branch
7d81: e8                        inx                             ;Done testing that RAM address
7d82: d0 ec                     bne     RamPage0TestLoop        ;More addresses in Page 0 to test? If so, branch
7d84: 8d 00 34                  sta     WdClear                 ;Clear the watchdog timer.
                        ;
                        • Clear variables
                        ]GenPtr00       .var    $00    {addr/2}

7d87: 8a                        txa                             ;Clear A by transferring the #$00 in X
7d88: 85 00                     sta     ]GenPtr00               ;Clear address $00
7d8a: 2a                        rol     A                       ;Get the set carry bit and put in A; A = #$01
7d8b: 85 01             RamTestNextPage sta     ]GenPtr00+1             ;Load the next bank upper address
                        ;
7d8d: a0 00                     ldy     #$00                    ;Start at beginning of the bank
                        RamPageNTestLoop
7d8f: a2 11                     ldx     #$11                    ;Four bit RAM; load a single bit per RAM
7d91: b1 00                     lda     (]GenPtr00),y           ;Byte read should be equal to 0 at first
7d93: d0 2a                     bne     RamPageNFail            ;If not 0, branch; bad RAM found
                        RamPageNByteTest
7d95: 8a                        txa                             ;Store the bit pattern in RAM
7d96: 91 00                     sta     (]GenPtr00),y
7d98: 51 00                     eor     (]GenPtr00),y           ;Read the value back out and compare to the original
7d9a: d0 23                     bne     RamPageNFail            ;Do the values match? If not, branch; bad RAM
7d9c: 8a                        txa                             ;Shift the bit pattern left by one
7d9d: 0a                        asl     A
7d9e: aa                        tax
7d9f: 90 f4                     bcc     RamPageNByteTest        ;Done writing to this address? If not branch
7da1: c8                        iny                             ;Increment to next address
7da2: d0 eb                     bne     RamPageNTestLoop        ;Done with this page? If not, branch to write another byte
7da4: 8d 00 34                  sta     WdClear                 ;Clear the watchdog timer.
```

```
                        ;
7da7: e6 01                     inc     ]GenPtr00+1             ;Increment to the next page
7da9: a6 01                     ldx     ]GenPtr00+1
7dab: e0 04                     cpx     #MpuRamPages            ;Have the 4 MPU RAM pages been checked?
7dad: 90 e0                     bcc     RamPageNTestLoop        ;If not, branch to check next page
                        ;
7daf: a9 40                     lda     #$40                    ;MPU RAM check complete; move on to vector RAM.
7db1: e0 40                     cpx     #$40                    ;More vector RAM to check?
7db3: 90 d6                     bcc     RamTestNextPage         ;If so, branch to check more
7db5: e0 48                     cpx     #$48                    ;Have all the vector RAM pages been checked?
7db7: 90 d6                     bcc     RamPageNTestLoop        ;If not, branch to check the next one
7db9: b0 69                     bcs     RomTest                 ;RAM test passed; move to the ROM/PROM test

7dbb: a0 00             RamPage0Fail    ldy     #$00            ;Zero page RAM failed, Y = #$00
7dbd: f0 0e                     beq     MakeRamList             ;Branch always

7dbf: a0 00             RamPageNFail    ldy     #$00            ;MPU RAM failed, Y = #$00
7dc1: a6 01                     ldx     ]GenPtr00+1             ;Get RAM page that failed
7dc3: e0 04                     cpx     #MpuRamPages            ;Was it MPU RAM that failed?
7dc5: 90 06                     bcc     MakeRamList             ;If so, branch
7dc7: c8                        iny                             ;Lower vector RAM failed, Y = #$01
7dc8: e0 44                     cpx     #>VectorRam+$400        ;Was it lower vector RAM half that failed?
7dca: 90 01                     bcc     MakeRamList             ;If so, branch
7dcc: c8                        iny                             ;Upper vector RAM failed, Y = #$02
7dcd: c9 10             MakeRamList     cmp     #$10            ;Detected difference stored in A. If difference was in upper
7dcf: 2a                        rol     A                       ; nibble, upper RAM is bad and bit is rolled into LSB A
7dd0: 29 1f                     and     #%00011111              ;Check for lower nibble difference
7dd2: c9 02                     cmp     #$02                    ;If one exists, a bit will be rolled into LSB A
7dd4: 2a                        rol     A
7dd5: 29 03                     and     #$03                    ;Keep only two lower bits as they are the failure bits
7dd7: 88                ShiftRamPairs   dey                     ;Decrement Y to move to next RAM pairs
7dd8: 30 04                     bmi     BadRamToneLoop          ;Finished shifting? If so, play RAM tones
7dda: 0a                        asl     A                       ;Need to move the bad RAM pairs up in memory
7ddb: 0a                        asl     A                       ; to make room for the next RAM pairs
7ddc: 90 f9                     bcc     ShiftRamPairs           ;More Ram pairs to shift? If so, branch
                        ;
7dde: 4a                BadRamToneLoop  lsr     A               ;Move RAM good/bad bit to carry
7ddf: a2 14                     ldx     #GoodRamFreq            ;Assume RAM is good and prepare to play good RAM tone
7de1: 90 02                     bcc     LoadRamThump            ;Is good/bad bit cleared? If so, branch; RAM is good
7de3: a2 1d                     ldx     #BadRamFreq             ;This is bad RAM; load bad RAM thump frequency
                        ;
7de5: 8e 00 3a          LoadRamThump    stx     ThumpFreqVol    ;Play RAM tone
7de8: a2 00                     ldx     #$00
7dea: a0 08                     ldy     #$08                    ;Play tone for 256*8 3KHz periods (.68 seconds)
7dec: 2c 01 20          BadRamPlayTone  bit     Clk3Khz
7def: 10 fb                     bpl     BadRamPlayTone          ;Wait for 1 3KHz period (333us)
7df1: 2c 01 20          :Loop           bit     Clk3Khz
7df4: 30 fb                     bmi     :Loop
7df6: ca                        dex                             ;One more 3KHz period has passed
7df7: 8d 00 34                  sta     WdClear                 ;Clear the watchdog timer
7dfa: d0 f0                     bne     BadRamPlayTone          ;Has 256 3KHz periods elapsed? If not, branch to wait more
7dfc: 88                        dey                             ;Another 256 3KHz periods have passed
7dfd: d0 ed                     bne     BadRamPlayTone          ;More time left to play RAM tone? If so, branch
                        ;
7dff: 8e 00 3a                  stx     ThumpFreqVol            ;Turn off thump SFX
7e02: a0 08                     ldy     #$08                    ;Prepare to wait another .68 seconds
7e04: 2c 01 20          BadRamWaitTone  bit     Clk3Khz
7e07: 10 fb                     bpl     BadRamWaitTone          ;Wait for 1 3KHz period (333us)
7e09: 2c 01 20          :Loop2          bit     Clk3Khz
7e0c: 30 fb                     bmi     :Loop2
7e0e: ca                        dex                             ;One more 3KHz period has passed
7e0f: 8d 00 34                  sta     WdClear                 ;Clear the watchdog timer
7e12: d0 f0                     bne     BadRamWaitTone          ;Has 256 3KHz periods elapsed? If not, branch to wait more
7e14: 88                        dey                             ;Another 256 3KHz periods have passed
7e15: d0 ed                     bne     BadRamWaitTone          ;More time left to play RAM tone? If so, branch
7e17: aa                        tax                             ;Are there still more RAMs to play tones for?
7e18: d0 c4                     bne     BadRamToneLoop          ;If so, branch to do the next RAM chip
7e1a: 8d 00 34          BadRamCheckTest sta     WdClear         ;Clear the watchdog timer
                        ;
7e1d: ad 07 20                  lda     SelfTestSw              ;Is self test still enabled?
7e20: 30 f8                     bmi     BadRamCheckTest         ;If so, loop until it is disabled
7e22: 10 fe             BadRamSpinLock  bpl     BadRamSpinLock  ;Self test released; spin lock until watchdog reset

                        • Clear variables
                        ]GenByte00      .var    $00     {addr/1}
                        ]VecRomPtr      .var    $08     {addr/2}
                        ]RomChecksum    .var    $0d     {addr/8}
                        ]DiagStepState  .var    $15     {addr/1}
                        ]RamSwapResults .var    $16     {addr/1}

7e24: a9 00             RomTest         lda     #<VectorRom
7e26: a8                        tay
7e27: aa                        tax                             ;Point to the start of the vector ROM
7e28: 85 08                     sta     ]VecRomPtr
7e2a: a9 50                     lda     #>VectorRom
7e2c: 85 09             RomTestKBLoop   sta     ]VecRomPtr+1    ;Prepare to test 1Kb of ROM (#$0400 bytes)
```

```
7e2e: a9 04                     lda     #$04
7e30: 85 0b                     sta     GenByte0B
                ;
7e32: a9 ff                     lda     #$ff                    ;Prepare to invert all the bits
7e34: 51 08     RomTestBankLoop eor     ()VecRomPtr),y          ;Keep a running checksum on ROM contents
7e36: c8                        iny                             ;Move to the next address
7e37: d0 fb                     bne     RomTestBankLoop         ;Is this page done? If not, branch to get another byte
7e39: e6 09                     inc     ]VecRomPtr+1            ;Move to next ROM page
7e3b: c6 0b                     dec     GenByte0B               ;Is 1 KB of ROM done?
7e3d: d0 f5                     bne     RomTestBankLoop         ;If not, branch to start next page
7e3f: 95 0d                     sta     ]RomChecksum,x          ;Store checksum for this 1Kb of ROM
7e41: e8                        inx                             ;Move to next checksum storage byte
7e42: 8d 00 34                  sta     WdClear                 ;Clear the watchdog timer
                ;
7e45: a5 09                     lda     ]VecRomPtr+1            ;Are we at the end of the vector ROM?
7e47: c9 58                     cmp     #>VectorRomEnd          ;If not, branch to get checksum of another Kb
7e49: 90 e1                     bcc     RomTestKBLoop
7e4b: d0 02                     bne     RomChecksumDone         ;Are checking the program ROM? If so, branch
7e4d: a9 68                     lda     #>ProgramRom            ;Start checking the program ROM
7e4f: c9 80     RomChecksumDone cmp     #>ProgramRomEnd         ;Are we done checking the program ROM?
7e51: 90 d9                     bcc     RomTestKBLoop           ;If not, branch to do another Kb
7e53: 8d 00 03                  sta     Player2Ram              ;Store ProgramRomEndUB(#$80) into RAM location $0300
7e56: a2 04                     ldx     #RamSwap                ;Swap RAM locations $0200-$02FF with $0300-$03FF
7e58: 8e 00 32                  stx     MultiPurp
7e5b: 86 15                     stx     ]DiagStepState          ;Initialize DiagStepState with #$04; draws initial
7e5d: a2 00                     ldx     #$00                    ; line on screen if DiagStep is active
7e5f: cd 00 02                  cmp     AstStatus               ;The value of A stored in Player2Ram should now be here.
7e62: f0 01                     beq     CheckPlr2Ram            ;Did the RAM swap successfully? If so, branch
7e64: e8                        inx                             ;There was a RAM swap problem; increment X
7e65: ad 00 03  CheckPlr2Ram    lda     Player2Ram              ;The final bit pattern written in the RAM test routine
7e68: c9 88                     cmp     #$88                    ; should be here
7e6a: f0 01                     beq     CheckSwapDone
7e6c: e8                        inx                             ;There was a RAM swap problem; increment X
7e6d: 86 16     CheckSwapDone   stx     ]RamSwapResults         ;Store the results of the RAM swap test
7e6f: a9 10                     lda     #$10                    ;Written but not read
7e71: 85 00                     sta     ]GenByte00
                ;
                SelfTestMainLoop
7e73: a2 24                     ldx     #SelfTestWait           ;Wait for 36 3KHz periods(12 ms)
                SelfTestWaitLoop
7e75: ad 01 20                  lda     Clk3Khz
7e78: 10 fb                     bpl     SelfTestWaitLoop        ;Wait for 1 3KHz period (333us)
7e7a: ad 01 20  :Loop           lda     Clk3Khz
7e7d: 30 fb                     bmi     :Loop
7e7f: ca                        dex                             ;Has 12 ms elapsed?
7e80: 10 f3                     bpl     SelfTestWaitLoop        ;If not, branch to wait some more
                ;
7e82: 2c 02 20  VectorWaitLoop2 bit     Halt                    ;Is the vector state machine busy?
7e85: 30 fb                     bmi     VectorWaitLoop2         ;If so, loop until it is idle
7e87: 8d 00 34                  sta     WdClear
                ;
7e8a: a9 00                     lda     #<VectorRam
7e8c: 85 02                     sta     VecRamPtr               ;Set vector RAM pointer to start of RAM
7e8e: a9 40                     lda     #>VectorRam
7e90: 85 03                     sta     VecRamPtr+1
7e92: ad 05 20                  lda     DiagStep                ;Is diagnostic step active?
7e95: 10 5b                     bpl     ShowDipStatus           ;If not, branch to next test
7e97: a6 15                     ldx     ]DiagStepState          ;Get current diagnostic step state
7e99: ad 03 20                  lda     HyprSpcSw               ;Has the hyperspace button been pressed?
7e9c: 10 0a                     bpl     LoadDiagVects           ;If so, update diagnostic step
7e9e: 4d 09 00                  eor:    TestSFXInit_            ;Was hyperspace button just pressed?
7ea1: 10 05                     bpl     LoadDiagVects           ;If not, branch to display current DiagStep lines
7ea3: ca                        dex                             ;Hyperspace was pressed; can anymore DiagStep lines be added?
7ea4: f0 02                     beq     LoadDiagVects           ;If not, branch to draw existing DiagStep lines
7ea6: 86 15                     stx     ]DiagStepState          ;Add another DiagStep line to the display
                ;
7ea8: bc bb 7e  LoadDiagVects   ldy     DiagStepIdxTbl-1,x      ;Get the current address into the vector RAM
7eab: a9 b0                     lda     #HaltOpcode             ;Add a HALT to the last addresses
7ead: 91 02                     sta     (VecRamPtr),y
7eaf: 88                        dey                             ;Move down to the next word in the vector RAM
7eb0: 88                        dey
                DiagStepWriteLoop
7eb1: b9 c0 7e                  lda     DiagStepDatTbl,y        ;Get next byte from the table below and write to vector RAM
7eb4: 91 02                     sta     (VecRamPtr),y
7eb6: 88                        dey                             ;Any more bytes to write to vector RAM?
7eb7: 10 f8                     bpl     DiagStepWriteLoop       ;If so, branch to get another byte
7eb9: 4c 9d 7f                  jmp     GetTestButtons          ;Jump to get user inputs

                ;
                ; The values in the table are indexes to the data tables below.  The actual
                ; index is the value in the table - 2. Must account for $B000 (HALT) written to
                ; the end of each segment. The $B000 is overwritten by the next segment allowing
                ; drawing continuation.
                ;
7ebc: 33 1d 17 0d DiagStepIdxTbl  .bulk   $33,$1d,$17,$0d
                ; Draws the first line. Written to vector RAM at $4000.
```

```
7ec0: 80 a0        DiagStepDatTbl  .dd2    $a080                   ;LABS x=0 y=128 sc=0
7ec2: 00 00                        .dd2    $0000
7ec4: 00 70                        .dd2    $7000                   ;VCTR x=+0 y=+0 sc=7 b=0
7ec6: 00 00                        .dd2    $0000
7ec8: ff 92                        .dd2    $92ff                   ;VCTR x=+1023 y=+767 sc=9 b=7
7eca: ff 73                        .dd2    $73ff
                   ; Draws the second line. Written to vector RAM at $400C.
7ecc: d0 a1                        .dd2    $a1d0                   ;LABS x=560 y=464 sc=0
7ece: 30 02                        .dd2    $0230
7ed0: 00 70                        .dd2    $7000                   ;VCTR x=+0 y=+0 sc=7 b=0
7ed2: 00 00                        .dd2    $0000
7ed4: 7f fb                        .dd2    $fb7f                   ;SVEC x=+-3 y=+3 sc=3 b=7
                   ; Draws the third line. Written to vector RAM at $4016.
7ed6: 0d e0                        .dd2    $e00d                   ;JMPL a=$000d ($401a)
7ed8: 00 b0                        .dd2    $b000                   ;HALT
7eda: 7e fa                        .dd2    $fa7e                   ;SVEC x=+-2 y=+2 sc=3 b=7
                   ; Draws the last triangle. Written to vector RAM at $401C.
7edc: 11 c0                        .dd2    $c011                   ;JSRL a=$0011 ($4022)
7ede: 78 fe                        .dd2    $fe78                   ;SVEC x=+0 y=+-2 sc=3 b=7
7ee0: 00 b0                        .dd2    $b000                   ;HALT
7ee2: 13 c0                        .dd2    $c013                   ;JSRL a=$0013 ($4026)
7ee4: 00 d0                        .dd2    $d000                   ;RTSL
7ee6: 15 c0                        .dd2    $c015                   ;JSRL a=$0015 ($402a)
7ee8: 00 d0                        .dd2    $d000                   ;RTSL
7eea: 17 c0                        .dd2    $c017                   ;JSRL a=$0017 ($402e)
7eec: 00 d0                        .dd2    $d000                   ;RTSL
7eee: 7a f8                        .dd2    $f87a                   ;SVEC x=+2 y=+0 sc=3 b=7
7ef0: 00 d0                        .dd2    $d000                   ;RTSL

                   ]tmp_08         .var    $08    {addr/1}
                   ]TestSFXInit    .var    $09    {addr/1}
                   ]tmp_0a         .var    $0a    {addr/1}

7ef2: a9 50        ShowDipStatus   lda     #>VectorRom             ;Prepare to load cross-hatch pattern on the screen
7ef4: a2 00                        ldx     #<VectorRom
7ef6: 20 fc 7b                     jsr     VecRomJSRL              ;Load JSRL command in vector RAM to vector ROM
7ef9: a9 69                        lda     #$69                    ;X beam coordinate 4 * $69 = $1A4 = 420
7efb: a2 93                        ldx     #$93                    ;Y beam coordinate 4 * $93 = $24C = 588
7efd: 20 03 7c                     jsr     MoveBeam                ;Move the CRT beam to a new location
7f00: a9 30                        lda     #$30                    ;Set scale 3(/64)
7f02: 20 de 7c                     jsr     SpotKill                ;Draw zero vector to prevent spots on the screen
                   ;
7f05: a2 03                        ldx     #$03                    ;Prepare to read the 4 pairs of DIP switches
                   DrawDipStatusLoop
7f07: bd 00 28                     lda     DipSw,x                 ;Get selected DIP switch pair status
7f0a: 29 01                        and     #%00000001              ;Keep only lower DIP switch status
7f0c: 86 0b                        stx     GenByte0B               ;Save a copy of the DIP pair currently being checked
7f0e: 20 d1 7b                     jsr     DrawDigit               ;Draw a single digit on the display
7f11: a6 0b                        ldx     GenByte0B               ;Restore a copy of the DIP pair currently being checked
7f13: bd 00 28                     lda     DipSw,x                 ;Reload the selected DIP switch pair status
7f16: 29 02                        and     #%00000010              ;Keep only upper DIP switch status
7f18: 4a                           lsr     A                       ;Move it to the LSB
7f19: 20 d1 7b                     jsr     DrawDigit               ;Draw a single digit on the display
7f1c: a6 0b                        ldx     GenByte0B               ;Reload the selected DIP switch pair status
7f1e: ca                           dex                             ;Does another DIP switch pair need to be checked?
7f1f: 10 e6                        bpl     DrawDipStatusLoop       ;If so, branch to get the next pair
                   ;
7f21: a9 7a                        lda     #$7a                    ;X beam coordinate 4 * $7A = $1E8 = 488
7f23: a2 9d                        ldx     #$9d                    ;Y beam coordinate 4 * $9D = $274 = 628
7f25: 20 03 7c                     jsr     MoveBeam                ;Move the CRT beam to a new location
7f28: a9 10                        lda     #$10                    ;Set scale 1(/256)
7f2a: 20 de 7c                     jsr     SpotKill                ;Draw zero vector to prevent spots on the screen
                   ;
7f2d: ad 02 28                     lda     CentCMShipsSw           ;Get the center coin mechanism switch status and display it
7f30: 29 02                        and     #%00000010
7f32: 4a                           lsr     A
7f33: 69 01                        adc     #$01
7f35: 20 d1 7b                     jsr     DrawDigit               ;Draw a single digit on the display
7f38: ad 01 28                     lda     RghtCoinMechSw          ;Get the right coin mechanism switches status
7f3b: 29 03                        and     #%00000011
7f3d: aa                           tax                             ;Use the switches status to display the coin multiplier value
7f3e: bd f5 7f                     lda     CoinMultTbl,x
7f41: 20 d1 7b                     jsr     DrawDigit               ;Draw a single digit on the display
7f44: a5 16                        lda     ]RamSwapResults         ;Was there a RAM swap error?
7f46: f0 07                        beq     VerifyChecksum          ;If not, branch to move to the next test
7f48: a2 88                        ldx     #<VecBankErr            ;Prepare to write bank error message to the display
7f4a: a9 50                        lda     #>VecBankErr
7f4c: 20 fc 7b                     jsr     VecRomJSRL              ;Load JSR command in vector RAM to vector ROM.
                   ;
7f4f: a2 96        VerifyChecksum  ldx     #$96                    ;Y beam coordinate 4 * $96 = $258 = 600
7f51: 8e 0c 00                     stx:    GenByte0C               ;Store base value for Y beam coordinate
7f54: a2 07                        ldx     #$07                    ;Prepare to check all 8 checksum values
7f56: b5 0d        ChecksumLoop    lda     ]RomChecksum,x          ;Is this checksum correct?
7f58: f0 37                        beq     NextChecksum            ;If so, branch to get next checksum
7f5a: 48                           pha                             ;Incorrect checksum; save checksum value on stack
7f5b: 8e 0b 00                     stx:    GenByte0B               ;Save current checksum index
```

```
7f5e: ae 0c 00                  ldx:    GenByte0C               ;Prepare to move the Y beam position to display failure info
7f61: 8a                        txa                             ;Move the Y bean position down by 32
7f62: 38                        sec
7f63: e9 08                     sbc     #$08
7f65: 8d 0c 00                  sta:    GenByte0C               ;Save new beam position
7f68: a9 20                     lda     #$20                    ;X beam coordinate 4 * $20 = $80 = 128
7f6a: 20 03 7c                  jsr     MoveBeam                ;Move the CRT beam to a new location
7f6d: a9 70                     lda     #$70                    ;Set scale 7(/4)
7f6f: 20 de 7c                  jsr     SpotKill                ;Draw zero vector to prevent spots on the screen
                        ;
7f72: ad 0b 00                  lda:    GenByte0B               ;Write failing checksum index to the display
7f75: 20 d1 7b                  jsr     DrawDigit               ;Draw a single digit on the display
7f78: ad d4 56                  lda     CharPtrTbl              ;Point to first entry in character pointer table (space)
7f7b: ae d5 56                  ldx     CharPtrTbl+1            ;Prepare to write a space to the display
7f7e: 20 45 7d                  jsr     VecWriteWord            ;Write 2 bytes to vector RAM
7f81: 68                        pla                             ;Get the incorrect checksum value again
7f82: 48                        pha                             ;Store it right back on the stack
7f83: 4a                        lsr     A
7f84: 4a                        lsr     A                       ;Prepare to write the upper nibble to the display
7f85: 4a                        lsr     A
7f86: 4a                        lsr     A
7f87: 20 d1 7b                  jsr     DrawDigit               ;Draw a single digit on the display
7f8a: 68                        pla                             ;Prepare to write the lower nibble to the display
7f8b: 20 d1 7b                  jsr     DrawDigit               ;Draw a single digit on the display
7f8e: ae 0b 00                  ldx:    GenByte0B               ;Get the next checksum index to check
7f91: ca            NextChecksum    dex                     ;Is there another checksum to check?
7f92: 10 c2                     bpl     ChecksumLoop            ;If so, branch
                        ;
7f94: a9 7f                     lda     #$7f                    ;X beam coordinate 4 * $7F = $1FC = 508
7f96: aa                        tax                             ;Y beam coordinate 4 * $7F = $1FC = 508
7f97: 20 03 7c                  jsr     MoveBeam                ;Move the CRT beam to a new location
7f9a: 20 c0 7b                  jsr     VecHalt                 ;Halt the vector state machine
                        ;
7f9d: a9 00         GetTestButtons  lda     #$00                    ;Prepare to get the statuses of 5 switches
7f9f: a2 04                     ldx     #$04
7fa1: 3e 03 20      GetBtnsLoop1    rol     HyprSpcSw,x             ;Get the status of: self test, slam, diagnostic step,
7fa4: 6a                        ror     A                       ; fire and hyperspace switches
7fa5: ca                        dex                             ;More switches to get the status of?
7fa6: 10 f9                     bpl     GetBtnsLoop1            ;If so, branch
7fa8: a8                        tay                             ;Prepare to get the statuses of 8 switches
7fa9: a2 07                     ldx     #$07
7fab: 3e 00 24      GetBtnsLoop2    rol     LeftCoinSw,x            ;Get the status of: rotate left, rotate right, thrust,
7fae: 2a                        rol     A                       ; 2 player start, 1 player start, right coin, center coin
7faf: ca                        dex                             ; left coin switches
7fb0: 10 f9                     bpl     GetBtnsLoop2            ;More switches to get the status of? If so, branch
                        ;
7fb2: aa                        tax
7fb3: 45 08                     eor     ]tmp_08                 ;Store bits indicating button changes
7fb5: 86 08                     stx     ]tmp_08
7fb7: 08                        php                             ;Save processor status.
7fb8: a9 04                     lda     #RamSwap                ;Swap RAM pages
7fba: 8d 00 32                  sta     MultiPurp
7fbd: 2e 03 20                  rol     HyprSpcSw
7fc0: 2a                        rol     A
7fc1: 2e 04 20                  rol     FireSw
7fc4: 2a                        rol     A
7fc5: 2e 07 24                  rol     RotLeftSw               ;Save the statuses of the player inputs into X
7fc8: 2a                        rol     A
7fc9: 2e 06 24                  rol     RotRghtSw
7fcc: 2a                        rol     A
7fcd: 2e 05 24                  rol     ThrustSw
7fd0: 2a                        rol     A
7fd1: aa                        tax
7fd2: 28                        plp                             ;Restore processor status
7fd3: d0 09                     bne     ButtonChanged           ;Were buttons changed? if so, branch
7fd5: 45 0a                     eor     ]tmp_0a                 ;Was a button change detected?
7fd7: d0 05                     bne     ButtonChanged           ;If so, branch to make a sound
7fd9: 98                        tya                             ;Was a button changed detected?
7fda: 45 09                     eor     ]TestSFXInit
7fdc: f0 02                     beq     DoHardwareWrite         ;If not, branch to turn off the SFX
                        ;
7fde: a9 80         ButtonChanged   lda     #EnableBit              ;Button change detected, set the MSB
7fe0: 8d 05 3c      DoHardwareWrite sta     LifeSFX                 ;Play/halt SFX
7fe3: 8d 00 32                  sta     MultiPurp               ;No effect
7fe6: 8d 00 30                  sta     DmaGo                   ;Start/stop the vector state machine
7fe9: 86 0a                     stx     ]tmp_0a                 ;Store current button statuses
7feb: 84 09                     sty     ]TestSFXInit
7fed: ad 07 20                  lda     SelfTestSw              ;Is self test switch still on? If so, loop
7ff0: 10 fe         :Spin           bpl     :Spin                   ;self test released; spin lock until watchdog reset
7ff2: 4c 73 7e                  jmp     SelfTestMainLoop        ;Stay in self test loop

                        ;
                        ; The table below sets the right coin mechanism multiplier based on the settings
                        ; of DIP switches 5 and 6.
                        ;
7ff5: 01 04 05 06   CoinMultTbl     .bulk   $01,$04,$05,$06
```

```
                        ;
7ff9: 4e                .dd1    $4e                 ;checksum byte
                        ;
7ffa: 65 7b             .dd2    NMI                 ;NMI vector
7ffc: f3 7c             .dd2    RESET               ;RESET vector
7ffe: f3 7c             .dd2    RESET               ;IRQ vector
                        .adrend ↑ $6800

                        ********************************************************************************
                        *                                                                              *
                        * Start of vector ROM.                                                         *
                        *                                                                              *
                        * The first item is a test pattern: diamond grid with a series of horizontal   *
                        * lines of varying brightness in the center.                                   *
                        *                                                                              *
                        ********************************************************************************
```

```
                        .addrs  $5000
5000: 80 a0             .dd2    $a080               ;LABS x=0 y=128 sc=0
5002: 00 00             .dd2    $0000
5004: 00 70             .dd2    $7000               ;VCTR x=+0 y=+0 sc=7 b=0
5006: 00 00             .dd2    $0000
5008: 00 90             .dd2    $9000               ;VCTR x=+1023 y=+0 sc=9 b=7
500a: ff 73             .dd2    $73ff
500c: ff 92             .dd2    $92ff               ;VCTR x=+0 y=+767 sc=9 b=7
500e: 00 70             .dd2    $7000
5010: 00 90             .dd2    $9000               ;VCTR x=-1023 y=+0 sc=9 b=7
5012: ff 77             .dd2    $77ff
5014: ff 96             .dd2    $96ff               ;VCTR x=+0 y=-767 sc=9 b=7
5016: 00 70             .dd2    $7000
5018: ff 92             .dd2    $92ff               ;VCTR x=+767 y=+767 sc=9 b=7
501a: ff 72             .dd2    $72ff
501c: 00 86             .dd2    $8600               ;VCTR x=+512 y=-512 sc=8 b=7
501e: 00 72             .dd2    $7200
5020: fe 87             .dd2    $87fe               ;VCTR x=-1022 y=-1022 sc=8 b=7
5022: fe 77             .dd2    $77fe
5024: 00 92             .dd2    $9200               ;VCTR x=-512 y=+512 sc=9 b=7
5026: 00 76             .dd2    $7600
5028: fe 81             .dd2    $81fe               ;VCTR x=+512 y=+510 sc=8 b=7
502a: 00 72             .dd2    $7200
502c: ff 96             .dd2    $96ff               ;VCTR x=+767 y=-767 sc=9 b=7
502e: ff 72             .dd2    $72ff
5030: 7f a3             .dd2    $a37f               ;LABS x=1023 y=895 sc=0
5032: ff 03             .dd2    $03ff
5034: 00 70             .dd2    $7000               ;VCTR x=+0 y=+0 sc=7 b=0
5036: 00 00             .dd2    $0000
5038: ff 96             .dd2    $96ff               ;VCTR x=-767 y=-767 sc=9 b=7
503a: ff 76             .dd2    $76ff
503c: fe 81             .dd2    $81fe               ;VCTR x=-512 y=+510 sc=8 b=7
503e: 00 76             .dd2    $7600
5040: 00 92             .dd2    $9200               ;VCTR x=+512 y=+512 sc=9 b=7
5042: 00 72             .dd2    $7200
5044: fe 87             .dd2    $87fe               ;VCTR x=+1022 y=-1022 sc=8 b=7
5046: fe 73             .dd2    $73fe
5048: 00 86             .dd2    $8600               ;VCTR x=-512 y=-512 sc=8 b=7
504a: 00 76             .dd2    $7600
504c: ff 92             .dd2    $92ff               ;VCTR x=-767 y=+767 sc=9 b=7
504e: ff 76             .dd2    $76ff
5050: fc a1             .dd2    $a1fc               ;LABS x=500 y=508 sc=0
5052: f4 01             .dd2    $01f4
5054: 00 70             .dd2    $7000               ;VCTR x=+0 y=+0 sc=7 b=0
5056: 00 00             .dd2    $0000
5058: db f0             .dd2    $f0db               ;SVEC x=+3 y=+0 sc=2 b=13
505a: 00 f9             .dd2    $f900               ;SVEC x=+0 y=+1 sc=1 b=0
505c: cf f0             .dd2    $f0cf               ;SVEC x=+-3 y=+0 sc=2 b=12
505e: 00 f9             .dd2    $f900               ;SVEC x=+0 y=+1 sc=1 b=0
5060: bb f0             .dd2    $f0bb               ;SVEC x=+3 y=+0 sc=2 b=11
5062: 00 f9             .dd2    $f900               ;SVEC x=+0 y=+1 sc=1 b=0
5064: af f0             .dd2    $f0af               ;SVEC x=+-3 y=+0 sc=2 b=10
5066: 00 f9             .dd2    $f900               ;SVEC x=+0 y=+1 sc=1 b=0
5068: 9b f0             .dd2    $f09b               ;SVEC x=+3 y=+0 sc=2 b=9
506a: 00 f9             .dd2    $f900               ;SVEC x=+0 y=+1 sc=1 b=0
506c: 8f f0             .dd2    $f08f               ;SVEC x=+-3 y=+0 sc=2 b=8
506e: 00 f9             .dd2    $f900               ;SVEC x=+0 y=+1 sc=1 b=0
5070: 7b f0             .dd2    $f07b               ;SVEC x=+3 y=+0 sc=2 b=7
5072: 00 f9             .dd2    $f900               ;SVEC x=+0 y=+1 sc=1 b=0
5074: 6f f0             .dd2    $f06f               ;SVEC x=+-3 y=+0 sc=2 b=6
5076: 00 f9             .dd2    $f900               ;SVEC x=+0 y=+1 sc=1 b=0
5078: 5b f0             .dd2    $f05b               ;SVEC x=+3 y=+0 sc=2 b=5
507a: 00 f9             .dd2    $f900               ;SVEC x=+0 y=+1 sc=1 b=0
507c: 4f f0             .dd2    $f04f               ;SVEC x=+-3 y=+0 sc=2 b=4
507e: 00 f9             .dd2    $f900               ;SVEC x=+0 y=+1 sc=1 b=0
5080: 3b f0             .dd2    $f03b               ;SVEC x=+3 y=+0 sc=2 b=3
```

```
5082: 00 f9                     .dd2    $f900                   ;SVEC x=+0 y=+1 sc=1 b=0
5084: 2f f0                     .dd2    $f02f                   ;SVEC x=+-3 y=+0 sc=2 b=2
5086: 7c d0                     .dd2    $d07c                   ;RTSL

                ;
                ; Bank error message, for self test.
                ;
```


```
5088: e4 a0     VecBankErr      .dd2    $a0e4                   ;LABS x=350 y=228 sc=1
508a: 5e 11                     .dd2    $115e
508c: 00 70                     .dd2    $7000                   ;VCTR x=+0 y=+0 sc=7 b=0
508e: 00 00                     .dd2    $0000
5090: 80 ca                     .dd2    $ca80                   ;JSRL a=$0a80 ($5500)
5092: 78 ca                     .dd2    $ca78                   ;JSRL a=$0a78 ($54f0)
5094: d8 ca                     .dd2    $cad8                   ;JSRL a=$0ad8 ($55b0)
5096: c7 ca                     .dd2    $cac7                   ;JSRL a=$0ac7 ($558e)
5098: 2c cb                     .dd2    $cb2c                   ;JSRL a=$0b2c ($5658)
509a: 9b ca                     .dd2    $ca9b                   ;JSRL a=$0a9b ($5536)
509c: f3 ca                     .dd2    $caf3                   ;JSRL a=$0af3 ($55e6)
509e: f3 ca                     .dd2    $caf3                   ;JSRL a=$0af3 ($55e6)
50a0: dd ca                     .dd2    $cadd                   ;JSRL a=$0add ($55ba)
50a2: f3 ea                     .dd2    $eaf3                   ;JMPL a=$0af3 ($55e6)

                ;
                ; Copyright notice.
                ;
```

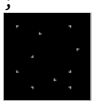

```
50a4: 80 a0     VecCredits      .dd2    $a080                   ;LABS x=400 y=128 sc=0
50a6: 90 01                     .dd2    $0190
50a8: 00 70                     .dd2    $7000                   ;VCTR x=+0 y=+0 sc=7 b=0
50aa: 00 00                     .dd2    $0000
50ac: 73 f5                     .dd2    $f573                   ;SVEC x=+3 y=+-1 sc=0 b=7
50ae: 73 f1                     .dd2    $f173                   ;SVEC x=+3 y=+1 sc=0 b=7
50b0: 78 f1                     .dd2    $f178                   ;SVEC x=+0 y=+1 sc=2 b=7
50b2: 77 f1                     .dd2    $f177                   ;SVEC x=+-3 y=+1 sc=0 b=7
50b4: 77 f5                     .dd2    $f577                   ;SVEC x=+-3 y=+-1 sc=0 b=7
50b6: 78 f5                     .dd2    $f578                   ;SVEC x=+0 y=+-1 sc=2 b=7
50b8: 80 31                     .dd2    $3180                   ;VCTR x=+512 y=+384 sc=3 b=0
50ba: 00 02                     .dd2    $0200
50bc: 75 f8                     .dd2    $f875                   ;SVEC x=+-1 y=+0 sc=1 b=7
50be: 70 fd                     .dd2    $fd70                   ;SVEC x=+0 y=+-1 sc=1 b=7
50c0: 71 f8                     .dd2    $f871                   ;SVEC x=+1 y=+0 sc=1 b=7
50c2: 02 fd                     .dd2    $fd02                   ;SVEC x=+2 y=+-1 sc=1 b=0
50c4: 2e cb                     .dd2    $cb2e                   ;JSRL a=$0b2e ($565c)
50c6: 63 cb                     .dd2    $cb63                   ;JSRL a=$0b63 ($56c6)
50c8: 56 cb                     .dd2    $cb56                   ;JSRL a=$0b56 ($56ac)
50ca: 63 cb                     .dd2    $cb63                   ;JSRL a=$0b63 ($56c6)
50cc: 2c cb                     .dd2    $cb2c                   ;JSRL a=$0b2c ($5658)
50ce: 78 ca                     .dd2    $ca78                   ;JSRL a=$0a78 ($54f0)
50d0: 02 cb                     .dd2    $cb02                   ;JSRL a=$0b02 ($5604)
50d2: 78 ca                     .dd2    $ca78                   ;JSRL a=$0a78 ($54f0)
50d4: f3 ca                     .dd2    $caf3                   ;JSRL a=$0af3 ($55e6)
50d6: ba ca                     .dd2    $caba                   ;JSRL a=$0aba ($5574)
50d8: 2c cb                     .dd2    $cb2c                   ;JSRL a=$0b2c ($5658)
50da: ba ca                     .dd2    $caba                   ;JSRL a=$0aba ($5574)
50dc: d8 ca                     .dd2    $cad8                   ;JSRL a=$0ad8 ($55b0)
50de: 8d ea                     .dd2    $ea8d                   ;JMPL a=$0a8d ($551a)
                ;
                ; Exploding ship pieces.  Parts are copied into RAM.
                ;
50e0: c6 ff     ShipExpPtrTbl   .dd2    $ffc6                   ;SVEC x=+-2 y=+-3 sc=1 b=12
50e2: c1 fe                     .dd2    $fec1                   ;SVEC x=+1 y=+-2 sc=1 b=12
50e4: c3 f1                     .dd2    $f1c3                   ;SVEC x=+3 y=+1 sc=0 b=12
50e6: cd f1                     .dd2    $f1cd                   ;SVEC x=+-1 y=+1 sc=2 b=12
50e8: c7 f1                     .dd2    $f1c7                   ;SVEC x=+-3 y=+1 sc=0 b=12
50ea: c1 fd                     .dd2    $fdc1                   ;SVEC x=+1 y=+-1 sc=1 b=12
                ;
                ; Ship explosion velocity (X,Y).
                ;
50ec: d8 1e     ShipExpVelTbl   .bulk   $d8,$1e
50ee: 32 ec                     .bulk   $32,$ec
50f0: 00 c4                     .bulk   $00,$c4
50f2: 3c 14                     .bulk   $3c,$14
50f4: 0a 46                     .bulk   $0a,$46
50f6: d8 d8                     .bulk   $d8,$d8
                ;
                ; Shrapnel Patterns. This is used when the player's shot hits something. Notice
                ; that all four patterns are the same just slightly spread out. This is
                ; extremely clever. You could use one pattern and vary the scale to make it look
                ; like it is spreading out. But the scale jumps are powers-of-two. These
```

```
                ; slightly-scaled patterns can be used to take up the gaps in the large scaling
                ; doubles!
                ;
                ; JSRLs to four shrapnel patterns.
                ;
50f8: d0 c8     SharpPatPtrTbl  .dd2    $c8d0                   ;JSRL a=$08d0 ($51a0)
50fa: b5 c8                     .dd2    $c8b5                   ;JSRL a=$08b5 ($516a)
50fc: 96 c8                     .dd2    $c896                   ;JSRL a=$0896 ($512c)
50fe: 80 c8                     .dd2    $c880                   ;JSRL a=$0880 ($5100)

                ;
                ; Shrapnel pattern 1.
                ;
```

```
5100: 0d f8                     .dd2    $f80d                   ;SVEC x=+-1 y=+0 sc=3 b=0 <<<
5102: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
5104: 0d fd                     .dd2    $fd0d                   ;SVEC x=+-1 y=+-1 sc=3 b=0
5106: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
5108: 09 fd                     .dd2    $fd09                   ;SVEC x=+1 y=+-1 sc=3 b=0
510a: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
510c: 0b f1                     .dd2    $f10b                   ;SVEC x=+3 y=+1 sc=2 b=0
510e: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
5110: 0a f5                     .dd2    $f50a                   ;SVEC x=+2 y=+-1 sc=2 b=0
5112: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
5114: 08 f9                     .dd2    $f908                   ;SVEC x=+0 y=+1 sc=3 b=0
5116: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
5118: 09 f3                     .dd2    $f309                   ;SVEC x=+1 y=+3 sc=2 b=0
511a: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
511c: 0d f3                     .dd2    $f30d                   ;SVEC x=+-1 y=+3 sc=2 b=0
511e: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
5120: 80 54                     .dd2    $5480                   ;VCTR x=-512 y=-128 sc=5 b=0
5122: 00 06                     .dd2    $0600
5124: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
5126: 0f f1                     .dd2    $f10f                   ;SVEC x=+-3 y=+1 sc=2 b=0
5128: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
512a: 00 d0                     .dd2    $d000                   ;RTSL

                ;
                ; Shrapnel pattern 2.
                ;
```

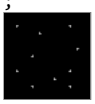


```
512c: 00 30                     .dd2    $3000                   ;VCTR x=-896 y=+0 sc=3 b=0 <<<
512e: 80 07                     .dd2    $0780
5130: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
5132: 80 37                     .dd2    $3780                   ;VCTR x=-896 y=-896 sc=3 b=0
5134: 80 07                     .dd2    $0780
5136: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
5138: 80 37                     .dd2    $3780                   ;VCTR x=+896 y=-896 sc=3 b=0
513a: 80 03                     .dd2    $0380
513c: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
513e: e0 40                     .dd2    $40e0                   ;VCTR x=+672 y=+224 sc=4 b=0
5140: a0 02                     .dd2    $02a0
5142: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
5144: c0 35                     .dd2    $35c0                   ;VCTR x=+896 y=-448 sc=3 b=0
5146: 80 03                     .dd2    $0380
5148: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
514a: 80 33                     .dd2    $3380                   ;VCTR x=+0 y=+896 sc=3 b=0
514c: 00 00                     .dd2    $0000
514e: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
5150: a0 42                     .dd2    $42a0                   ;VCTR x=+224 y=+672 sc=4 b=0
5152: e0 00                     .dd2    $00e0
5154: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
5156: a0 42                     .dd2    $42a0                   ;VCTR x=-224 y=+672 sc=4 b=0
5158: e0 04                     .dd2    $04e0
515a: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
515c: e0 44                     .dd2    $44e0                   ;VCTR x=-896 y=-224 sc=4 b=0
515e: 80 07                     .dd2    $0780
5160: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
5162: e0 40                     .dd2    $40e0                   ;VCTR x=-672 y=+224 sc=4 b=0
5164: a0 06                     .dd2    $06a0
5166: 78 f8                     .dd2    $f878                   ;SVEC x=+0 y=+0 sc=3 b=7
5168: 00 d0                     .dd2    $d000                   ;RTSL

                ;
                ; Shrapnel patern 3.
                ;
```

```
516a: 07 f8                     .dd2    $f807               ;SVEC x=+-3 y=+0 sc=1 b=0 <<<
516c: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
516e: 07 ff                     .dd2    $ff07               ;SVEC x=+-3 y=+-3 sc=1 b=0
5170: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
5172: 03 ff                     .dd2    $ff03               ;SVEC x=+3 y=+-3 sc=1 b=0
5174: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
5176: c0 40                     .dd2    $40c0               ;VCTR x=+576 y=+192 sc=4 b=0
5178: 40 02                     .dd2    $0240
517a: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
517c: 80 35                     .dd2    $3580               ;VCTR x=+768 y=-384 sc=3 b=0
517e: 00 03                     .dd2    $0300
5180: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
5182: 00 fb                     .dd2    $fb00               ;SVEC x=+0 y=+3 sc=1 b=0
5184: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
5186: 40 42                     .dd2    $4240               ;VCTR x=+192 y=+576 sc=4 b=0
5188: c0 00                     .dd2    $00c0
518a: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
518c: 40 42                     .dd2    $4240               ;VCTR x=-192 y=+576 sc=4 b=0
518e: c0 04                     .dd2    $04c0
5190: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
5192: c0 44                     .dd2    $44c0               ;VCTR x=-768 y=-192 sc=4 b=0
5194: 00 07                     .dd2    $0700
5196: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
5198: c0 40                     .dd2    $40c0               ;VCTR x=-576 y=+192 sc=4 b=0
519a: 40 06                     .dd2    $0640
519c: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
519e: 00 d0                     .dd2    $d000               ;RTSL

                ;
                ; Shrapnel pattern 4.
                ;
```



```
51a0: 00 30                     .dd2    $3000               ;VCTR x=-640 y=+0 sc=3 b=0 <<<
51a2: 80 06                     .dd2    $0680
51a4: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
51a6: 80 36                     .dd2    $3680               ;VCTR x=-640 y=-640 sc=3 b=0
51a8: 80 06                     .dd2    $0680
51aa: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
51ac: 80 36                     .dd2    $3680               ;VCTR x=+640 y=-640 sc=3 b=0
51ae: 80 02                     .dd2    $0280
51b0: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
51b2: 40 31                     .dd2    $3140               ;VCTR x=+960 y=+320 sc=3 b=0
51b4: c0 03                     .dd2    $03c0
51b6: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
51b8: 40 35                     .dd2    $3540               ;VCTR x=+640 y=-320 sc=3 b=0
51ba: 80 02                     .dd2    $0280
51bc: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
51be: 80 32                     .dd2    $3280               ;VCTR x=+0 y=+640 sc=3 b=0
51c0: 00 00                     .dd2    $0000
51c2: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
51c4: c0 33                     .dd2    $33c0               ;VCTR x=+320 y=+960 sc=3 b=0
51c6: 40 01                     .dd2    $0140
51c8: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
51ca: c0 33                     .dd2    $33c0               ;VCTR x=-320 y=+960 sc=3 b=0
51cc: 40 05                     .dd2    $0540
51ce: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
51d0: a0 44                     .dd2    $44a0               ;VCTR x=-640 y=-160 sc=4 b=0
51d2: 80 06                     .dd2    $0680
51d4: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
51d6: 40 31                     .dd2    $3140               ;VCTR x=-960 y=+320 sc=3 b=0
51d8: c0 07                     .dd2    $07c0
51da: 78 f8                     .dd2    $f878               ;SVEC x=+0 y=+0 sc=3 b=7
51dc: 00 d0                     .dd2    $d000               ;RTSL
                ;
                ; Asteroid patterns.
                ;
51de: f3 c8     AstPtrnPtrTbl   .dd2    $c8f3               ;JSRL a=$08f3 ($51e6)
51e0: ff c8                     .dd2    $c8ff               ;JSRL a=$08ff ($51fe)
51e2: 0d c9                     .dd2    $c90d               ;JSRL a=$090d ($521a)
51e4: 1a c9                     .dd2    $c91a               ;JSRL a=$091a ($5234)
```

```
51e6: 08 f9                     .dd2    $f908               ;SVEC x=+0 y=+1 sc=3 b=0 <<<
51e8: 79 f9                     .dd2    $f979               ;SVEC x=+1 y=+1 sc=3 b=7
51ea: 79 fd                     .dd2    $fd79               ;SVEC x=+1 y=+-1 sc=3 b=7
51ec: 7d f6                     .dd2    $f67d               ;SVEC x=+-1 y=+-2 sc=2 b=7
51ee: 79 f6                     .dd2    $f679               ;SVEC x=+1 y=+-2 sc=2 b=7
51f0: 8f f6                     .dd2    $f68f               ;SVEC x=+-3 y=+-2 sc=2 b=8
51f2: 8f f0                     .dd2    $f08f               ;SVEC x=+-3 y=+0 sc=2 b=8
51f4: 7d f9                     .dd2    $f97d               ;SVEC x=+-1 y=+1 sc=3 b=7
```

```
51f6: 78 fa                     .dd2    $fa78                   ;SVEC x=+0 y=+2 sc=3 b=7
51f8: 79 f9                     .dd2    $f979                   ;SVEC x=+1 y=+1 sc=3 b=7
51fa: 79 fd                     .dd2    $fd79                   ;SVEC x=+1 y=+-1 sc=3 b=7
51fc: 00 d0                     .dd2    $d000                   ;RTSL
```

```
51fe: 0a f1                     .dd2    $f10a                   ;SVEC x=+2 y=+1 sc=2 b=0 <<<
5200: 7a f1                     .dd2    $f17a                   ;SVEC x=+2 y=+1 sc=2 b=7
5202: 7d f9                     .dd2    $f97d                   ;SVEC x=+-1 y=+1 sc=3 b=7
5204: 7e f5                     .dd2    $f57e                   ;SVEC x=+-2 y=+-1 sc=2 b=7
5206: 7e f1                     .dd2    $f17e                   ;SVEC x=+-2 y=+1 sc=2 b=7
5208: 7d fd                     .dd2    $fd7d                   ;SVEC x=+-1 y=+-1 sc=3 b=7
520a: 79 f6                     .dd2    $f679                   ;SVEC x=+1 y=+-2 sc=2 b=7
520c: 7d f6                     .dd2    $f67d                   ;SVEC x=+-1 y=+-2 sc=2 b=7
520e: 79 fd                     .dd2    $fd79                   ;SVEC x=+1 y=+-1 sc=3 b=7
5210: 79 f1                     .dd2    $f179                   ;SVEC x=+1 y=+1 sc=2 b=7
5212: 8b f5                     .dd2    $f58b                   ;SVEC x=+3 y=+-1 sc=2 b=8
5214: 8a f3                     .dd2    $f38a                   ;SVEC x=+2 y=+3 sc=2 b=8
5216: 7d f9                     .dd2    $f97d                   ;SVEC x=+-1 y=+1 sc=3 b=7
5218: 00 d0                     .dd2    $d000                   ;RTSL
```

```
521a: 0d f8                     .dd2    $f80d                   ;SVEC x=+-1 y=+0 sc=3 b=0 <<<
521c: 7e f5                     .dd2    $f57e                   ;SVEC x=+-2 y=+-1 sc=2 b=7
521e: 7a f7                     .dd2    $f77a                   ;SVEC x=+2 y=+-3 sc=2 b=7
5220: 7a f3                     .dd2    $f37a                   ;SVEC x=+2 y=+3 sc=2 b=7
5222: 78 f7                     .dd2    $f778                   ;SVEC x=+0 y=+-3 sc=2 b=7
5224: 79 f8                     .dd2    $f879                   ;SVEC x=+1 y=+0 sc=3 b=7
5226: 7a f3                     .dd2    $f37a                   ;SVEC x=+2 y=+3 sc=2 b=7
5228: 78 f9                     .dd2    $f978                   ;SVEC x=+0 y=+1 sc=3 b=7
522a: 7e f3                     .dd2    $f37e                   ;SVEC x=+-2 y=+3 sc=2 b=7
522c: 7f f0                     .dd2    $f07f                   ;SVEC x=+-3 y=+0 sc=2 b=7
522e: 7f f7                     .dd2    $f77f                   ;SVEC x=+-3 y=+-3 sc=2 b=7
5230: 7a f5                     .dd2    $f57a                   ;SVEC x=+2 y=+-1 sc=2 b=7
5232: 00 d0                     .dd2    $d000                   ;RTSL
```

```
5234: 09 f0                     .dd2    $f009                   ;SVEC x=+1 y=+0 sc=2 b=0 <<<
5236: 7b f1                     .dd2    $f17b                   ;SVEC x=+3 y=+1 sc=2 b=7
5238: 68 f1                     .dd2    $f168                   ;SVEC x=+0 y=+1 sc=2 b=6
523a: 7f f2                     .dd2    $f27f                   ;SVEC x=+-3 y=+2 sc=2 b=7
523c: 7f f0                     .dd2    $f07f                   ;SVEC x=+-3 y=+0 sc=2 b=7
523e: 69 f6                     .dd2    $f669                   ;SVEC x=+1 y=+-2 sc=2 b=6
5240: 7f f0                     .dd2    $f07f                   ;SVEC x=+-3 y=+0 sc=2 b=7
5242: 78 f7                     .dd2    $f778                   ;SVEC x=+0 y=+-3 sc=2 b=7
5244: 7a f7                     .dd2    $f77a                   ;SVEC x=+2 y=+-3 sc=2 b=7
5246: 7b f1                     .dd2    $f17b                   ;SVEC x=+3 y=+1 sc=2 b=7
5248: 69 f5                     .dd2    $f569                   ;SVEC x=+1 y=+-1 sc=2 b=6
524a: 69 f9                     .dd2    $f969                   ;SVEC x=+1 y=+1 sc=3 b=6
524c: 7f f2                     .dd2    $f27f                   ;SVEC x=+-3 y=+2 sc=2 b=7
524e: 00 d0                     .dd2    $d000                   ;RTSL
                ;
                ; Flying saucer.  The same shape is used for big and small versions.
                ;
5250: 29 c9     ScrPtrnPtrTbl   .dd2    $c929                   ;JSRL a=$0929 ($5252)
```

```
5252: 0e f1                     .dd2    $f10e                   ;SVEC x=+-2 y=+1 sc=2 b=0 <<<
5254: ca f8                     .dd2    $f8ca                   ;SVEC x=+2 y=+0 sc=3 b=12
5256: 0b f6                     .dd2    $f60b                   ;SVEC x=+3 y=+-2 sc=2 b=0
5258: 00 60                     .dd2    $6000                   ;VCTR x=-640 y=+0 sc=6 b=13
525a: 80 d6                     .dd2    $d680
525c: db f6                     .dd2    $f6db                   ;SVEC x=+3 y=+-2 sc=2 b=13
525e: ca f8                     .dd2    $f8ca                   ;SVEC x=+2 y=+0 sc=3 b=12
5260: db f2                     .dd2    $f2db                   ;SVEC x=+3 y=+2 sc=2 b=13
5262: df f2                     .dd2    $f2df                   ;SVEC x=+-3 y=+2 sc=2 b=13
5264: cd f2                     .dd2    $f2cd                   ;SVEC x=+-1 y=+2 sc=2 b=12
5266: cd f8                     .dd2    $f8cd                   ;SVEC x=+-1 y=+0 sc=3 b=12
5268: cd f6                     .dd2    $f6cd                   ;SVEC x=+-1 y=+-2 sc=2 b=12
526a: df f6                     .dd2    $f6df                   ;SVEC x=+-3 y=+-2 sc=2 b=13
526c: 00 d0                     .dd2    $d000                   ;RTSL
                ;
                ; Pointers to ship and thrust vector shape data.
                ;
```

```
                ; The code doesn't perform rotations, so there is a different shape for each
                ; direction in the first quadrant.  Other quadrants are handled by flipping
                ; about the X and/or Y axis.
                ;
                ; Each ship shape is paired with a matching thrust cone.  The thrust is only
                ; drawn if the player is accelerating.
                ;
526e: 90 52     ShipDirPtrTbl   .dd2    ShipDir0
5270: a8 52                     .dd2    ShipDir4
5272: cc 52                     .dd2    ShipDir8
5274: f0 52                     .dd2    ShipDir12
5276: 14 53                     .dd2    ShipDir16
5278: 36 53                     .dd2    ShipDir20
527a: 5a 53                     .dd2    ShipDir24
527c: 7e 53                     .dd2    ShipDir28
527e: a2 53                     .dd2    ShipDir32
5280: c6 53                     .dd2    ShipDir36
5282: ea 53                     .dd2    ShipDir40
5284: 0e 54                     .dd2    ShipDir44
5286: 32 54                     .dd2    ShipDir48
5288: 56 54                     .dd2    ShipDir52
528a: 7a 54                     .dd2    ShipDir56
528c: 9e 54                     .dd2    ShipDir60
528e: c2 54                     .dd2    ShipDir64

                ;
                
5290: 0f f6     ShipDir0        .dd2    $f60f
5292: c8 fa                     .dd2    $fac8                   ;SVEC x=+0 y=+2 sc=3 b=12
5294: bd f9                     .dd2    $f9bd                   ;SVEC x=+-1 y=+1 sc=3 b=11
5296: 00 65                     .dd2    $6500                   ;VCTR x=+768 y=-256 sc=6 b=12
5298: 00 c3                     .dd2    $c300
529a: 00 65                     .dd2    $6500                   ;VCTR x=-768 y=-256 sc=6 b=12
529c: 00 c7                     .dd2    $c700
529e: b9 f9                     .dd2    $f9b9                   ;SVEC x=+1 y=+1 sc=3 b=11
52a0: 00 d0                     .dd2    $d000                   ;RTSL

                
52a2: ce f9     ThrustDir0      .dd2    $f9ce                   ;SVEC x=+-2 y=+1 sc=3 b=12
52a4: ca f9                     .dd2    $f9ca                   ;SVEC x=+2 y=+1 sc=3 b=12
52a6: 00 d0                     .dd2    $d000                   ;RTSL

                
52a8: 40 46     ShipDir4        .dd2    $4640                   ;VCTR x=-704 y=-576 sc=4 b=0
52aa: c0 06                     .dd2    $06c0
52ac: 00 52                     .dd2    $5200                   ;VCTR x=-48 y=+512 sc=5 b=12
52ae: 30 c4                     .dd2    $c430
52b0: c0 41                     .dd2    $41c0                   ;VCTR x=-544 y=+448 sc=4 b=12
52b2: 20 c6                     .dd2    $c620
52b4: b0 64                     .dd2    $64b0                   ;VCTR x=+792 y=-176 sc=6 b=12
52b6: 18 c3                     .dd2    $c318
52b8: 48 65                     .dd2    $6548                   ;VCTR x=-736 y=-328 sc=6 b=12
52ba: e0 c6                     .dd2    $c6e0
52bc: 20 42                     .dd2    $4220                   ;VCTR x=+448 y=+544 sc=4 b=12
52be: c0 c1                     .dd2    $c1c0
52c0: 00 d0                     .dd2    $d000                   ;RTSL

                
52c2: d0 50     ThrustDir4      .dd2    $50d0                   ;VCTR x=-528 y=+208 sc=5 b=12
52c4: 10 c6                     .dd2    $c610
52c6: 60 42                     .dd2    $4260                   ;VCTR x=+960 y=+608 sc=4 b=12
52c8: c0 c3                     .dd2    $c3c0
52ca: 00 d0                     .dd2    $d000                   ;RTSL

                
52cc: 80 46     ShipDir8        .dd2    $4680                   ;VCTR x=-640 y=-640 sc=4 b=0
52ce: 80 06                     .dd2    $0680
52d0: e0 43                     .dd2    $43e0                   ;VCTR x=-192 y=+992 sc=4 b=12
52d2: c0 c4                     .dd2    $c4c0
52d4: a0 41                     .dd2    $41a0                   ;VCTR x=-608 y=+416 sc=4 b=12
```

```
52d6: 60 c6                     .dd2    $c660
52d8: 68 64                     .dd2    $6468               ;VCTR x=+800 y=-104 sc=6 b=12
52da: 20 c3                     .dd2    $c320
52dc: 90 65                     .dd2    $6590               ;VCTR x=-704 y=-400 sc=6 b=12
52de: c0 c6                     .dd2    $c6c0
52e0: 60 42                     .dd2    $4260               ;VCTR x=+416 y=+608 sc=4 b=12
52e2: a0 c1                     .dd2    $c1a0
52e4: 00 d0                     .dd2    $d000               ;RTSL
```

```
52e6: 90 50                     .dd2    $5090               ;VCTR x=-560 y=+144 sc=5 b=12
52e8: 30 c6                     .dd2    $c630
52ea: c0 42                     .dd2    $42c0               ;VCTR x=+896 y=+704 sc=4 b=12
52ec: 80 c3                     .dd2    $c380
52ee: 00 d0                     .dd2    $d000               ;RTSL
```

```
52f0: c0 46         ShipDir12   .dd2    $46c0               ;VCTR x=-576 y=-704 sc=4 b=0
52f2: 40 06                     .dd2    $0640
52f4: e0 43                     .dd2    $43e0               ;VCTR x=-288 y=+992 sc=4 b=12
52f6: 20 c5                     .dd2    $c520
52f8: 60 41                     .dd2    $4160               ;VCTR x=-640 y=+352 sc=4 b=12
52fa: 80 c6                     .dd2    $c680
52fc: 18 64                     .dd2    $6418               ;VCTR x=+808 y=-24 sc=6 b=12
52fe: 28 c3                     .dd2    $c328
5300: d0 65                     .dd2    $65d0               ;VCTR x=-664 y=-464 sc=6 b=12
5302: 98 c6                     .dd2    $c698
5304: 80 42                     .dd2    $4280               ;VCTR x=+352 y=+640 sc=4 b=12
5306: 60 c1                     .dd2    $c160
5308: 00 d0                     .dd2    $d000               ;RTSL
530a: 60 50                     .dd2    $5060               ;VCTR x=-560 y=+96 sc=5 b=12
530c: 30 c6                     .dd2    $c630
530e: 20 43                     .dd2    $4320               ;VCTR x=+832 y=+800 sc=4 b=12
5310: 40 c3                     .dd2    $c340
5312: 00 d0                     .dd2    $d000               ;RTSL
```

```
5314: 0e f7         ShipDir16   .dd2    $f70e               ;SVEC x=+-2 y=+-3 sc=2 b=0
5316: c0 43                     .dd2    $43c0               ;VCTR x=-384 y=+960 sc=4 b=12
5318: 80 c5                     .dd2    $c580
531a: 20 41                     .dd2    $4120               ;VCTR x=-672 y=+288 sc=4 b=12
531c: a0 c6                     .dd2    $c6a0
531e: 38 60                     .dd2    $6038               ;VCTR x=+808 y=+56 sc=6 b=12
5320: 28 c3                     .dd2    $c328
5322: 10 66                     .dd2    $6610               ;VCTR x=-608 y=-528 sc=6 b=12
5324: 60 c6                     .dd2    $c660
5326: a0 42                     .dd2    $42a0               ;VCTR x=+288 y=+672 sc=4 b=12
5328: 20 c1                     .dd2    $c120
532a: 00 d0                     .dd2    $d000               ;RTSL
532c: 30 50                     .dd2    $5030               ;VCTR x=-576 y=+48 sc=5 b=12
532e: 40 c6                     .dd2    $c640
5330: 60 43                     .dd2    $4360               ;VCTR x=+736 y=+864 sc=4 b=12
5332: e0 c2                     .dd2    $c2e0
5334: 00 d0                     .dd2    $d000               ;RTSL
```

```
5336: 20 47         ShipDir20   .dd2    $4720               ;VCTR x=-448 y=-800 sc=4 b=0
5338: c0 05                     .dd2    $05c0
533a: 80 43                     .dd2    $4380               ;VCTR x=-480 y=+896 sc=4 b=12
533c: e0 c5                     .dd2    $c5e0
533e: e0 40                     .dd2    $40e0               ;VCTR x=-704 y=+224 sc=4 b=12
5340: c0 c6                     .dd2    $c6c0
5342: 88 60                     .dd2    $6088               ;VCTR x=+800 y=+136 sc=6 b=12
5344: 20 c3                     .dd2    $c320
5346: 48 66                     .dd2    $6648               ;VCTR x=-560 y=-584 sc=6 b=12
5348: 30 c6                     .dd2    $c630
534a: c0 42                     .dd2    $42c0               ;VCTR x=+224 y=+704 sc=4 b=12
534c: e0 c0                     .dd2    $c0e0
534e: 00 d0                     .dd2    $d000               ;RTSL
5350: 10 54                     .dd2    $5410               ;VCTR x=-576 y=-16 sc=5 b=12
5352: 40 c6                     .dd2    $c640
5354: a0 43                     .dd2    $43a0               ;VCTR x=+672 y=+928 sc=4 b=12
5356: a0 c2                     .dd2    $c2a0
5358: 00 d0                     .dd2    $d000               ;RTSL
```

```
535a: 60 47        ShipDir24       .dd2    $4760               ;VCTR x=-352 y=-864 sc=4 b=0
535c: 60 05                        .dd2    $0560
535e: 60 43                        .dd2    $4360               ;VCTR x=-576 y=+864 sc=4 b=12
5360: 40 c6                        .dd2    $c640
5362: 80 40                        .dd2    $4080               ;VCTR x=-704 y=+128 sc=4 b=12
5364: c0 c6                        .dd2    $c6c0
5366: d8 60                        .dd2    $60d8               ;VCTR x=+784 y=+216 sc=6 b=12
5368: 10 c3                        .dd2    $c310
536a: 80 66                        .dd2    $6680               ;VCTR x=-496 y=-640 sc=6 b=12
536c: f0 c5                        .dd2    $c5f0
536e: c0 42                        .dd2    $42c0               ;VCTR x=+128 y=+704 sc=4 b=12
5370: 80 c0                        .dd2    $c080
5372: 00 d0                        .dd2    $d000               ;RTSL
5374: 40 54                        .dd2    $5440               ;VCTR x=-560 y=-64 sc=5 b=12
5376: 30 c6                        .dd2    $c630
5378: e0 43                        .dd2    $43e0               ;VCTR x=+576 y=+992 sc=4 b=12
537a: 40 c2                        .dd2    $c240
537c: 00 d0                        .dd2    $d000               ;RTSL
```

```
537e: 80 47        ShipDir28       .dd2    $4780               ;VCTR x=-256 y=-896 sc=4 b=0
5380: 00 05                        .dd2    $0500
5382: 20 43                        .dd2    $4320               ;VCTR x=-640 y=+800 sc=4 b=12
5384: 80 c6                        .dd2    $c680
5386: 40 40                        .dd2    $4040               ;VCTR x=-736 y=+64 sc=4 b=12
5388: e0 c6                        .dd2    $c6e0
538a: 20 61                        .dd2    $6120               ;VCTR x=+760 y=+288 sc=6 b=12
538c: f8 c2                        .dd2    $c2f8
538e: b0 66                        .dd2    $66b0               ;VCTR x=-432 y=-688 sc=6 b=12
5390: b0 c5                        .dd2    $c5b0
5392: e0 42                        .dd2    $42e0               ;VCTR x=+64 y=+736 sc=4 b=12
5394: 40 c0                        .dd2    $c040
5396: 00 d0                        .dd2    $d000               ;RTSL
5398: 80 54                        .dd2    $5480               ;VCTR x=-560 y=-128 sc=5 b=12
539a: 30 c6                        .dd2    $c630
539c: 10 52                        .dd2    $5210               ;VCTR x=+240 y=+528 sc=5 b=12
539e: f0 c0                        .dd2    $c0f0
53a0: 00 d0                        .dd2    $d000               ;RTSL
```

```
53a2: 80 47        ShipDir32       .dd2    $4780               ;VCTR x=-192 y=-896 sc=4 b=0
53a4: c0 04                        .dd2    $04c0
53a6: e0 42                        .dd2    $42e0               ;VCTR x=-736 y=+736 sc=4 b=12
53a8: e0 c6                        .dd2    $c6e0
53aa: 00 40                        .dd2    $4000               ;VCTR x=-736 y=+0 sc=4 b=12
53ac: e0 c6                        .dd2    $c6e0
53ae: 68 61                        .dd2    $6168               ;VCTR x=+728 y=+360 sc=6 b=12
53b0: d8 c2                        .dd2    $c2d8
53b2: d8 66                        .dd2    $66d8               ;VCTR x=-360 y=-728 sc=6 b=12
53b4: 68 c5                        .dd2    $c568
53b6: e0 42                        .dd2    $42e0               ;VCTR x=+0 y=+736 sc=4 b=12
53b8: 00 c0                        .dd2    $c000
53ba: 00 d0                        .dd2    $d000               ;RTSL
53bc: b0 54                        .dd2    $54b0               ;VCTR x=-544 y=-176 sc=5 b=12
53be: 20 c6                        .dd2    $c620
53c0: 20 52                        .dd2    $5220               ;VCTR x=+176 y=+544 sc=5 b=12
53c2: b0 c0                        .dd2    $c0b0
53c4: 00 d0                        .dd2    $d000               ;RTSL
```

```
53c6: a0 47        ShipDir36       .dd2    $47a0               ;VCTR x=-96 y=-928 sc=4 b=0
53c8: 60 04                        .dd2    $0460
53ca: 80 42                        .dd2    $4280               ;VCTR x=-800 y=+640 sc=4 b=12
53cc: 20 c7                        .dd2    $c720
53ce: 40 44                        .dd2    $4440               ;VCTR x=-736 y=-64 sc=4 b=12
53d0: e0 c6                        .dd2    $c6e0
53d2: b0 61                        .dd2    $61b0               ;VCTR x=+688 y=+432 sc=6 b=12
53d4: b0 c2                        .dd2    $c2b0
53d6: f8 66                        .dd2    $66f8               ;VCTR x=-288 y=-760 sc=6 b=12
53d8: 20 c5                        .dd2    $c520
53da: e0 42                        .dd2    $42e0               ;VCTR x=-64 y=+736 sc=4 b=12
53dc: 40 c4                        .dd2    $c440
```

```
53de: 00 d0                     .dd2    $d000               ;RTSL
53e0: f0 54                     .dd2    $54f0               ;VCTR x=-528 y=-240 sc=5 b=12
53e2: 10 c6                     .dd2    $c610
53e4: 30 52                     .dd2    $5230               ;VCTR x=+128 y=+560 sc=5 b=12
53e6: 80 c0                     .dd2    $c080
53e8: 00 d0                     .dd2    $d000               ;RTSL
```

```
53ea: a0 47         ShipDir40   .dd2    $47a0               ;VCTR x=+0 y=-928 sc=4 b=0
53ec: 00 00                     .dd2    $0000
53ee: 40 42                     .dd2    $4240               ;VCTR x=-864 y=+576 sc=4 b=12
53f0: 60 c7                     .dd2    $c760
53f2: 80 44                     .dd2    $4480               ;VCTR x=-704 y=-128 sc=4 b=12
53f4: c0 c6                     .dd2    $c6c0
53f6: f0 61                     .dd2    $61f0               ;VCTR x=+640 y=+496 sc=6 b=12
53f8: 80 c2                     .dd2    $c280
53fa: 10 67                     .dd2    $6710               ;VCTR x=-216 y=-784 sc=6 b=12
53fc: d8 c4                     .dd2    $c4d8
53fe: c0 42                     .dd2    $42c0               ;VCTR x=-128 y=+704 sc=4 b=12
5400: 80 c4                     .dd2    $c480
5402: 00 d0                     .dd2    $d000               ;RTSL
5404: 40 46                     .dd2    $4640               ;VCTR x=-992 y=-576 sc=4 b=12
5406: e0 c7                     .dd2    $c7e0
5408: 30 52                     .dd2    $5230               ;VCTR x=+64 y=+560 sc=5 b=12
540a: 40 c0                     .dd2    $c040
540c: 00 d0                     .dd2    $d000               ;RTSL
```

```
540e: a0 47         ShipDir44   .dd2    $47a0               ;VCTR x=+96 y=-928 sc=4 b=0
5410: 60 00                     .dd2    $0060
5412: e0 41                     .dd2    $41e0               ;VCTR x=-896 y=+480 sc=4 b=12
5414: 80 c7                     .dd2    $c780
5416: e0 44                     .dd2    $44e0               ;VCTR x=-704 y=-224 sc=4 b=12
5418: c0 c6                     .dd2    $c6c0
541a: 30 62                     .dd2    $6230               ;VCTR x=+584 y=+560 sc=6 b=12
541c: 48 c2                     .dd2    $c248
541e: 20 67                     .dd2    $6720               ;VCTR x=-136 y=-800 sc=6 b=12
5420: 88 c4                     .dd2    $c488
5422: c0 42                     .dd2    $42c0               ;VCTR x=-224 y=+704 sc=4 b=12
5424: e0 c4                     .dd2    $c4e0
5426: 00 d0                     .dd2    $d000               ;RTSL
5428: a0 46                     .dd2    $46a0               ;VCTR x=-928 y=-672 sc=4 b=12
542a: a0 c7                     .dd2    $c7a0
542c: 40 52                     .dd2    $5240               ;VCTR x=+16 y=+576 sc=5 b=12
542e: 10 c0                     .dd2    $c010
5430: 00 d0                     .dd2    $d000               ;RTSL
```

```
5432: 80 47         ShipDir48   .dd2    $4780               ;VCTR x=+192 y=-896 sc=4 b=0
5434: c0 00                     .dd2    $00c0
5436: 80 41                     .dd2    $4180               ;VCTR x=-960 y=+384 sc=4 b=12
5438: c0 c7                     .dd2    $c7c0
543a: 20 45                     .dd2    $4520               ;VCTR x=-672 y=-288 sc=4 b=12
543c: a0 c6                     .dd2    $c6a0
543e: 60 62                     .dd2    $6260               ;VCTR x=+528 y=+608 sc=6 b=12
5440: 10 c2                     .dd2    $c210
5442: 28 67                     .dd2    $6728               ;VCTR x=-56 y=-808 sc=6 b=12
5444: 38 c4                     .dd2    $c438
5446: a0 42                     .dd2    $42a0               ;VCTR x=-288 y=+672 sc=4 b=12
5448: 20 c5                     .dd2    $c520
544a: 00 d0                     .dd2    $d000               ;RTSL
544c: e0 46                     .dd2    $46e0               ;VCTR x=-864 y=-736 sc=4 b=12
544e: 60 c7                     .dd2    $c760
5450: 40 52                     .dd2    $5240               ;VCTR x=-48 y=+576 sc=5 b=12
5452: 30 c4                     .dd2    $c430
5454: 00 d0                     .dd2    $d000               ;RTSL
```

```
5456: 80 47         ShipDir52   .dd2    $4780               ;VCTR x=+256 y=-896 sc=4 b=0
5458: 00 01                     .dd2    $0100
545a: 20 41                     .dd2    $4120               ;VCTR x=-992 y=+288 sc=4 b=12
545c: e0 c7                     .dd2    $c7e0
545e: 60 45                     .dd2    $4560               ;VCTR x=-640 y=-352 sc=4 b=12
5460: 80 c6                     .dd2    $c680
```

```
5462: 98 62                     .dd2    $6298                   ;VCTR x=+464 y=+664 sc=6 b=12
5464: d0 c1                     .dd2    $c1d0
5466: 28 67                     .dd2    $6728                   ;VCTR x=+24 y=-808 sc=6 b=12
5468: 18 c0                     .dd2    $c018
546a: 80 42                     .dd2    $4280                   ;VCTR x=-352 y=+640 sc=4 b=12
546c: 60 c5                     .dd2    $c560
546e: 00 d0                     .dd2    $d000                   ;RTSL
5470: 40 47                     .dd2    $4740                   ;VCTR x=-800 y=-832 sc=4 b=12
5472: 20 c7                     .dd2    $c720
5474: 30 52                     .dd2    $5230                   ;VCTR x=-96 y=+560 sc=5 b=12
5476: 60 c4                     .dd2    $c460
5478: 00 d0                     .dd2    $d000                   ;RTSL
```

```
547a: 60 47         ShipDir56   .dd2    $4760                   ;VCTR x=+352 y=-864 sc=4 b=0
547c: 60 01                     .dd2    $0160
547e: c0 40                     .dd2    $40c0                   ;VCTR x=-992 y=+192 sc=4 b=12
5480: e0 c7                     .dd2    $c7e0
5482: a0 45                     .dd2    $45a0                   ;VCTR x=-608 y=-416 sc=4 b=12
5484: 60 c6                     .dd2    $c660
5486: c0 62                     .dd2    $62c0                   ;VCTR x=+400 y=+704 sc=6 b=12
5488: 90 c1                     .dd2    $c190
548a: 20 67                     .dd2    $6720                   ;VCTR x=+104 y=-800 sc=6 b=12
548c: 68 c0                     .dd2    $c068
548e: 60 42                     .dd2    $4260                   ;VCTR x=-416 y=+608 sc=4 b=12
5490: a0 c5                     .dd2    $c5a0
5492: 00 d0                     .dd2    $d000                   ;RTSL
5494: 80 47                     .dd2    $4780                   ;VCTR x=-704 y=-896 sc=4 b=12
5496: c0 c6                     .dd2    $c6c0
5498: 30 52                     .dd2    $5230                   ;VCTR x=-144 y=+560 sc=5 b=12
549a: 90 c4                     .dd2    $c490
549c: 00 d0                     .dd2    $d000                   ;RTSL
```

```
549e: 20 47         ShipDir60   .dd2    $4720                   ;VCTR x=+448 y=-800 sc=4 b=0
54a0: c0 01                     .dd2    $01c0
54a2: 30 50                     .dd2    $5030                   ;VCTR x=-512 y=+48 sc=5 b=12
54a4: 00 c6                     .dd2    $c600
54a6: c0 45                     .dd2    $45c0                   ;VCTR x=-544 y=-448 sc=4 b=12
54a8: 20 c6                     .dd2    $c620
54aa: e0 62                     .dd2    $62e0                   ;VCTR x=+328 y=+736 sc=6 b=12
54ac: 48 c1                     .dd2    $c148
54ae: 18 67                     .dd2    $6718                   ;VCTR x=+176 y=-792 sc=6 b=12
54b0: b0 c0                     .dd2    $c0b0
54b2: 20 42                     .dd2    $4220                   ;VCTR x=-448 y=+544 sc=4 b=12
54b4: c0 c5                     .dd2    $c5c0
54b6: 00 d0                     .dd2    $d000                   ;RTSL
54b8: c0 47                     .dd2    $47c0                   ;VCTR x=-608 y=-960 sc=4 b=12
54ba: 60 c6                     .dd2    $c660
54bc: 10 52                     .dd2    $5210                   ;VCTR x=-208 y=+528 sc=5 b=12
54be: d0 c4                     .dd2    $c4d0
54c0: 00 d0                     .dd2    $d000                   ;RTSL
```

```
54c2: 0a f7         ShipDir64   .dd2    $f70a                   ;SVEC x=+2 y=+-3 sc=2 b=0
54c4: ce f8                     .dd2    $f8ce                   ;SVEC x=+-2 y=+0 sc=3 b=12
54c6: cd fd                     .dd2    $fdcd                   ;SVEC x=+-1 y=+-1 sc=3 b=12
54c8: 00 63                     .dd2    $6300                   ;VCTR x=+256 y=+768 sc=6 b=12
54ca: 00 c1                     .dd2    $c100
54cc: 00 67                     .dd2    $6700                   ;VCTR x=+256 y=-768 sc=6 b=12
54ce: 00 c1                     .dd2    $c100
54d0: cd f9                     .dd2    $f9cd                   ;SVEC x=+-1 y=+1 sc=3 b=12
54d2: 00 d0                     .dd2    $d000                   ;RTSL
54d4: cd fe                     .dd2    $fecd                   ;SVEC x=+-1 y=+-2 sc=3 b=12
54d6: cd fa                     .dd2    $facd                   ;SVEC x=+-1 y=+2 sc=3 b=12
54d8: 00 d0                     .dd2    $d000                   ;RTSL

                    ;
                    ; Extra lives vector data.  Draw one of these for each remaining life.
                    ;
```

```
54da: 0e f7         ExtLivesDat .dd2    $f70e                   ;SVEC x=+-2 y=+-3 sc=2 b=0
54dc: 7a f8                     .dd2    $f87a                   ;SVEC x=+2 y=+0 sc=3 b=7
54de: 79 fd                     .dd2    $fd79                   ;SVEC x=+1 y=+-1 sc=3 b=7
```

```
54e0: 00 63                     .dd2    $6300                   ;VCTR x=-256 y=+768 sc=6 b=7
54e2: 00 75                     .dd2    $7500
54e4: 00 67                     .dd2    $6700                   ;VCTR x=-256 y=-768 sc=6 b=7
54e6: 00 75                     .dd2    $7500
54e8: 79 f9                     .dd2    $f979                   ;SVEC x=+1 y=+1 sc=3 b=7
54ea: c0 60                     .dd2    $60c0                   ;VCTR x=+640 y=+192 sc=6 b=0
54ec: 80 02                     .dd2    $0280
54ee: 9f d0                     .dd2    $d09f                   ;RTSL

                ;
                ; Alphanumeric vector data.
                ;
                ; Each character leaves the vector pointed a short distance to the right of the
                ; initial position, so that you can output multiple characters without having to
                ; explicitly reposition each time.
                ;
```

```
54f0: 70 fa                     .dd2    $fa70                   ;SVEC x=+0 y=+2 sc=1 b=7 <<<
54f2: 72 f2                     .dd2    $f272                   ;SVEC x=+2 y=+2 sc=0 b=7
54f4: 72 f6                     .dd2    $f672                   ;SVEC x=+2 y=+-2 sc=0 b=7
54f6: 70 fe                     .dd2    $fe70                   ;SVEC x=+0 y=+-2 sc=1 b=7
54f8: 06 f9                     .dd2    $f906                   ;SVEC x=+-2 y=+1 sc=1 b=0
54fa: 72 f8                     .dd2    $f872                   ;SVEC x=+2 y=+0 sc=1 b=7
54fc: 02 f6                     .dd2    $f602                   ;SVEC x=+2 y=+-2 sc=0 b=0
54fe: 00 d0                     .dd2    $d000                   ;RTSL
```

```
5500: 70 fb                     .dd2    $fb70                   ;SVEC x=+0 y=+3 sc=1 b=7 <<<
5502: 73 f0                     .dd2    $f073                   ;SVEC x=+3 y=+0 sc=0 b=7
5504: 71 f5                     .dd2    $f571                   ;SVEC x=+1 y=+-1 sc=0 b=7
5506: 70 f5                     .dd2    $f570                   ;SVEC x=+0 y=+-1 sc=0 b=7
5508: 75 f5                     .dd2    $f575                   ;SVEC x=+-1 y=+-1 sc=0 b=7
550a: 77 f0                     .dd2    $f077                   ;SVEC x=+-3 y=+0 sc=0 b=7
550c: 03 f0                     .dd2    $f003                   ;SVEC x=+3 y=+0 sc=0 b=0
550e: 71 f5                     .dd2    $f571                   ;SVEC x=+1 y=+-1 sc=0 b=7
5510: 70 f5                     .dd2    $f570                   ;SVEC x=+0 y=+-1 sc=0 b=7
5512: 75 f5                     .dd2    $f575                   ;SVEC x=+-1 y=+-1 sc=0 b=7
5514: 77 f0                     .dd2    $f077                   ;SVEC x=+-3 y=+0 sc=0 b=7
5516: 03 f8                     .dd2    $f803                   ;SVEC x=+3 y=+0 sc=1 b=0
5518: 00 d0                     .dd2    $d000                   ;RTSL
```

```
551a: 70 fb                     .dd2    $fb70                   ;SVEC x=+0 y=+3 sc=1 b=7 <<<
551c: 72 f8                     .dd2    $f872                   ;SVEC x=+2 y=+0 sc=1 b=7
551e: 06 ff                     .dd2    $ff06                   ;SVEC x=+-2 y=+-3 sc=1 b=0
5520: 72 f8                     .dd2    $f872                   ;SVEC x=+2 y=+0 sc=1 b=7
5522: 02 f0                     .dd2    $f002                   ;SVEC x=+2 y=+0 sc=0 b=0
5524: 00 d0                     .dd2    $d000                   ;RTSL
```

```
5526: 70 fb                     .dd2    $fb70                   ;SVEC x=+0 y=+3 sc=1 b=7 <<<
5528: 72 f0                     .dd2    $f072                   ;SVEC x=+2 y=+0 sc=0 b=7
552a: 72 f6                     .dd2    $f672                   ;SVEC x=+2 y=+-2 sc=0 b=7
552c: 70 f6                     .dd2    $f670                   ;SVEC x=+0 y=+-2 sc=0 b=7
552e: 76 f6                     .dd2    $f676                   ;SVEC x=+-2 y=+-2 sc=0 b=7
5530: 76 f0                     .dd2    $f076                   ;SVEC x=+-2 y=+0 sc=0 b=7
5532: 03 f8                     .dd2    $f803                   ;SVEC x=+3 y=+0 sc=1 b=0
5534: 00 d0                     .dd2    $d000                   ;RTSL
```

```
5536: 70 fb                     .dd2    $fb70                   ;SVEC x=+0 y=+3 sc=1 b=7 <<<
5538: 72 f8                     .dd2    $f872                   ;SVEC x=+2 y=+0 sc=1 b=7
553a: 05 f7                     .dd2    $f705                   ;SVEC x=+-1 y=+-3 sc=0 b=0
553c: 77 f0                     .dd2    $f077                   ;SVEC x=+-3 y=+0 sc=0 b=7
553e: 00 f7                     .dd2    $f700                   ;SVEC x=+0 y=+-3 sc=0 b=0
5540: 72 f8                     .dd2    $f872                   ;SVEC x=+2 y=+0 sc=1 b=7
5542: 02 f0                     .dd2    $f002                   ;SVEC x=+2 y=+0 sc=0 b=0
5544: 00 d0                     .dd2    $d000                   ;RTSL
```

```
5546: 70 fb                 .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7 <<<
5548: 72 f8                 .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
554a: 05 f7                 .dd2    $f705               ;SVEC x=+-1 y=+-3 sc=0 b=0
554c: 77 f0                 .dd2    $f077               ;SVEC x=+-3 y=+0 sc=0 b=7
554e: 00 f7                 .dd2    $f700               ;SVEC x=+0 y=+-3 sc=0 b=0
5550: 03 f8                 .dd2    $f803               ;SVEC x=+3 y=+0 sc=1 b=0
5552: 00 d0                 .dd2    $d000               ;RTSL
```

```
5554: 70 fb                 .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7 <<<
5556: 72 f8                 .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
5558: 70 f6                 .dd2    $f670               ;SVEC x=+0 y=+-2 sc=0 b=7
555a: 06 f6                 .dd2    $f606               ;SVEC x=+-2 y=+-2 sc=0 b=0
555c: 72 f0                 .dd2    $f072               ;SVEC x=+2 y=+0 sc=0 b=7
555e: 70 f6                 .dd2    $f670               ;SVEC x=+0 y=+-2 sc=0 b=7
5560: 76 f8                 .dd2    $f876               ;SVEC x=+-2 y=+0 sc=1 b=7
5562: 03 f8                 .dd2    $f803               ;SVEC x=+3 y=+0 sc=1 b=0
5564: 00 d0                 .dd2    $d000               ;RTSL
```

```
5566: 70 fb                 .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7 <<<
5568: 00 f7                 .dd2    $f700               ;SVEC x=+0 y=+-3 sc=0 b=0
556a: 72 f8                 .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
556c: 00 f3                 .dd2    $f300               ;SVEC x=+0 y=+3 sc=0 b=0
556e: 70 ff                 .dd2    $ff70               ;SVEC x=+0 y=+-3 sc=1 b=7
5570: 02 f0                 .dd2    $f002               ;SVEC x=+2 y=+0 sc=0 b=0
5572: 00 d0                 .dd2    $d000               ;RTSL
```

```
5574: 72 f8                 .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7 <<<
5576: 06 f0                 .dd2    $f006               ;SVEC x=+-2 y=+0 sc=0 b=0
5578: 70 fb                 .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7
557a: 02 f0                 .dd2    $f002               ;SVEC x=+2 y=+0 sc=0 b=0
557c: 76 f8                 .dd2    $f876               ;SVEC x=+-2 y=+0 sc=1 b=7
557e: 03 ff                 .dd2    $ff03               ;SVEC x=+3 y=+-3 sc=1 b=0
5580: 00 d0                 .dd2    $d000               ;RTSL
```

```
5582: 00 f2                 .dd2    $f200               ;SVEC x=+0 y=+2 sc=0 b=0 <<<
5584: 72 f6                 .dd2    $f672               ;SVEC x=+2 y=+-2 sc=0 b=7
5586: 72 f0                 .dd2    $f072               ;SVEC x=+2 y=+0 sc=0 b=7
5588: 70 fb                 .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7
558a: 01 ff                 .dd2    $ff01               ;SVEC x=+1 y=+-3 sc=1 b=0
558c: 00 d0                 .dd2    $d000               ;RTSL
```

```
558e: 70 fb                 .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7 <<<
5590: 03 f0                 .dd2    $f003               ;SVEC x=+3 y=+0 sc=0 b=0
5592: 77 f7                 .dd2    $f777               ;SVEC x=+-3 y=+-3 sc=0 b=7
5594: 73 f7                 .dd2    $f773               ;SVEC x=+3 y=+-3 sc=0 b=7
5596: 03 f0                 .dd2    $f003               ;SVEC x=+3 y=+0 sc=0 b=0
5598: 00 d0                 .dd2    $d000               ;RTSL
```

```
559a: 00 fb                 .dd2    $fb00               ;SVEC x=+0 y=+3 sc=1 b=0 <<<
559c: 70 ff                 .dd2    $ff70               ;SVEC x=+0 y=+-3 sc=1 b=7
559e: 72 f8                 .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
55a0: 02 f0                 .dd2    $f002               ;SVEC x=+2 y=+0 sc=0 b=0
55a2: 00 d0                 .dd2    $d000               ;RTSL
```

```
55a4: 70 fb                     .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7 <<<
55a6: 72 f6                     .dd2    $f672               ;SVEC x=+2 y=+-2 sc=0 b=7
55a8: 72 f2                     .dd2    $f272               ;SVEC x=+2 y=+2 sc=0 b=7
55aa: 70 ff                     .dd2    $ff70               ;SVEC x=+0 y=+-3 sc=1 b=7
55ac: 02 f0                     .dd2    $f002               ;SVEC x=+2 y=+0 sc=0 b=0
55ae: 00 d0                     .dd2    $d000               ;RTSL
```

```
55b0: 70 fb                     .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7 <<<
55b2: 72 ff                     .dd2    $ff72               ;SVEC x=+2 y=+-3 sc=1 b=7
55b4: 70 fb                     .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7
55b6: 01 ff                     .dd2    $ff01               ;SVEC x=+1 y=+-3 sc=1 b=0
55b8: 00 d0                     .dd2    $d000               ;RTSL
```

```
55ba: 70 fb                     .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7 <<<
55bc: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
55be: 70 ff                     .dd2    $ff70               ;SVEC x=+0 y=+-3 sc=1 b=7
55c0: 76 f8                     .dd2    $f876               ;SVEC x=+-2 y=+0 sc=1 b=7
55c2: 03 f8                     .dd2    $f803               ;SVEC x=+3 y=+0 sc=1 b=0
55c4: 00 d0                     .dd2    $d000               ;RTSL
```

```
55c6: 70 fb                     .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7 <<<
55c8: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
55ca: 70 f7                     .dd2    $f770               ;SVEC x=+0 y=+-3 sc=0 b=7
55cc: 76 f8                     .dd2    $f876               ;SVEC x=+-2 y=+0 sc=1 b=7
55ce: 03 f7                     .dd2    $f703               ;SVEC x=+3 y=+-3 sc=0 b=0
55d0: 03 f0                     .dd2    $f003               ;SVEC x=+3 y=+0 sc=0 b=0
55d2: 00 d0                     .dd2    $d000               ;RTSL
```

```
55d4: 70 fb                     .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7 <<<
55d6: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
55d8: 70 fe                     .dd2    $fe70               ;SVEC x=+0 y=+-2 sc=1 b=7
55da: 76 f6                     .dd2    $f676               ;SVEC x=+-2 y=+-2 sc=0 b=7
55dc: 76 f0                     .dd2    $f076               ;SVEC x=+-2 y=+0 sc=0 b=7
55de: 02 f2                     .dd2    $f202               ;SVEC x=+2 y=+2 sc=0 b=0
55e0: 72 f6                     .dd2    $f672               ;SVEC x=+2 y=+-2 sc=0 b=7
55e2: 02 f0                     .dd2    $f002               ;SVEC x=+2 y=+0 sc=0 b=0
55e4: 00 d0                     .dd2    $d000               ;RTSL
```

```
55e6: 70 fb                     .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7 <<<
55e8: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
55ea: 70 f7                     .dd2    $f770               ;SVEC x=+0 y=+-3 sc=0 b=7
55ec: 76 f8                     .dd2    $f876               ;SVEC x=+-2 y=+0 sc=1 b=7
55ee: 01 f0                     .dd2    $f001               ;SVEC x=+1 y=+0 sc=0 b=0
55f0: 73 f7                     .dd2    $f773               ;SVEC x=+3 y=+-3 sc=0 b=7
55f2: 02 f0                     .dd2    $f002               ;SVEC x=+2 y=+0 sc=0 b=0
55f4: 00 d0                     .dd2    $d000               ;RTSL
```

```
55f6: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7 <<<
55f8: 70 f3                     .dd2    $f370               ;SVEC x=+0 y=+3 sc=0 b=7
55fa: 76 f8                     .dd2    $f876               ;SVEC x=+-2 y=+0 sc=1 b=7
55fc: 70 f3                     .dd2    $f370               ;SVEC x=+0 y=+3 sc=0 b=7
55fe: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
5600: 01 ff                     .dd2    $ff01               ;SVEC x=+1 y=+-3 sc=1 b=0
5602: 00 d0                     .dd2    $d000               ;RTSL
```

```
5604: 02 f0                 .dd2    $f002               ;SVEC x=+2 y=+0 sc=0 b=0 <<<
5606: 70 fb                 .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7
5608: 06 f0                 .dd2    $f006               ;SVEC x=+-2 y=+0 sc=0 b=0
560a: 72 f8                 .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
560c: 01 ff                 .dd2    $ff01               ;SVEC x=+1 y=+-3 sc=1 b=0
560e: 00 d0                 .dd2    $d000               ;RTSL
```

```
5610: 00 fb                 .dd2    $fb00               ;SVEC x=+0 y=+3 sc=1 b=0 <<<
5612: 70 ff                 .dd2    $ff70               ;SVEC x=+0 y=+-3 sc=1 b=7
5614: 72 f8                 .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
5616: 70 fb                 .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7
5618: 01 ff                 .dd2    $ff01               ;SVEC x=+1 y=+-3 sc=1 b=0
561a: 00 d0                 .dd2    $d000               ;RTSL
```

```
561c: 00 fb                 .dd2    $fb00               ;SVEC x=+0 y=+3 sc=1 b=0 <<<
561e: 71 ff                 .dd2    $ff71               ;SVEC x=+1 y=+-3 sc=1 b=7
5620: 71 fb                 .dd2    $fb71               ;SVEC x=+1 y=+3 sc=1 b=7
5622: 01 ff                 .dd2    $ff01               ;SVEC x=+1 y=+-3 sc=1 b=0
5624: 00 d0                 .dd2    $d000               ;RTSL
```

```
5626: 00 fb                 .dd2    $fb00               ;SVEC x=+0 y=+3 sc=1 b=0 <<<
5628: 70 ff                 .dd2    $ff70               ;SVEC x=+0 y=+-3 sc=1 b=7
562a: 72 f2                 .dd2    $f272               ;SVEC x=+2 y=+2 sc=0 b=7
562c: 72 f6                 .dd2    $f672               ;SVEC x=+2 y=+-2 sc=0 b=7
562e: 70 fb                 .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7
5630: 01 ff                 .dd2    $ff01               ;SVEC x=+1 y=+-3 sc=1 b=0
5632: 00 d0                 .dd2    $d000               ;RTSL
```

```
5634: 72 fb                 .dd2    $fb72               ;SVEC x=+2 y=+3 sc=1 b=7 <<<
5636: 06 f8                 .dd2    $f806               ;SVEC x=+-2 y=+0 sc=1 b=0
5638: 72 ff                 .dd2    $ff72               ;SVEC x=+2 y=+-3 sc=1 b=7
563a: 02 f0                 .dd2    $f002               ;SVEC x=+2 y=+0 sc=0 b=0
563c: 00 d0                 .dd2    $d000               ;RTSL
```

```
563e: 02 f0                 .dd2    $f002               ;SVEC x=+2 y=+0 sc=0 b=0 <<<
5640: 70 fa                 .dd2    $fa70               ;SVEC x=+0 y=+2 sc=1 b=7
5642: 76 f2                 .dd2    $f276               ;SVEC x=+-2 y=+2 sc=0 b=7
5644: 02 f8                 .dd2    $f802               ;SVEC x=+2 y=+0 sc=1 b=0
5646: 76 f6                 .dd2    $f676               ;SVEC x=+-2 y=+-2 sc=0 b=7
5648: 02 fe                 .dd2    $fe02               ;SVEC x=+2 y=+-2 sc=1 b=0
564a: 00 d0                 .dd2    $d000               ;RTSL
```

```
564c: 00 fb                 .dd2    $fb00               ;SVEC x=+0 y=+3 sc=1 b=0 <<<
564e: 72 f8                 .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
5650: 76 ff                 .dd2    $ff76               ;SVEC x=+-2 y=+-3 sc=1 b=7
5652: 72 f8                 .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
5654: 02 f0                 .dd2    $f002               ;SVEC x=+2 y=+0 sc=0 b=0
5656: 00 d0                 .dd2    $d000               ;RTSL
```

```
5658: 03 f8                 .dd2    $f803               ;SVEC x=+3 y=+0 sc=1 b=0 <<<
565a: 00 d0                 .dd2    $d000               ;RTSL
```

```
565c: 02 f0                     .dd2    $f002               ;SVEC x=+2 y=+0 sc=0 b=0 <<<
565e: 70 fb                     .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7
5660: 02 ff                     .dd2    $ff02               ;SVEC x=+2 y=+-3 sc=1 b=0
5662: 00 d0                     .dd2    $d000               ;RTSL
```

```
5664: 00 fb                     .dd2    $fb00               ;SVEC x=+0 y=+3 sc=1 b=0 <<<
5666: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
5668: 70 f7                     .dd2    $f770               ;SVEC x=+0 y=+-3 sc=0 b=7
566a: 76 f8                     .dd2    $f876               ;SVEC x=+-2 y=+0 sc=1 b=7
566c: 70 f7                     .dd2    $f770               ;SVEC x=+0 y=+-3 sc=0 b=7
566e: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
5670: 02 f0                     .dd2    $f002               ;SVEC x=+2 y=+0 sc=0 b=0
5672: 00 d0                     .dd2    $d000               ;RTSL
```

```
5674: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7 <<<
5676: 70 fb                     .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7
5678: 76 f8                     .dd2    $f876               ;SVEC x=+-2 y=+0 sc=1 b=7
567a: 00 f7                     .dd2    $f700               ;SVEC x=+0 y=+-3 sc=0 b=0
567c: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
567e: 02 f7                     .dd2    $f702               ;SVEC x=+2 y=+-3 sc=0 b=0
5680: 00 d0                     .dd2    $d000               ;RTSL
```

```
5682: 00 fb                     .dd2    $fb00               ;SVEC x=+0 y=+3 sc=1 b=0 <<<
5684: 70 f7                     .dd2    $f770               ;SVEC x=+0 y=+-3 sc=0 b=7
5686: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
5688: 00 f3                     .dd2    $f300               ;SVEC x=+0 y=+3 sc=0 b=0
568a: 70 ff                     .dd2    $ff70               ;SVEC x=+0 y=+-3 sc=1 b=7
568c: 02 f0                     .dd2    $f002               ;SVEC x=+2 y=+0 sc=0 b=0
568e: 00 d0                     .dd2    $d000               ;RTSL
```

```
5690: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7 <<<
5692: 70 f3                     .dd2    $f370               ;SVEC x=+0 y=+3 sc=0 b=7
5694: 76 f8                     .dd2    $f876               ;SVEC x=+-2 y=+0 sc=1 b=7
5696: 70 f3                     .dd2    $f370               ;SVEC x=+0 y=+3 sc=0 b=7
5698: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
569a: 01 ff                     .dd2    $ff01               ;SVEC x=+1 y=+-3 sc=1 b=0
569c: 00 d0                     .dd2    $d000               ;RTSL
```

```
569e: 00 f3                     .dd2    $f300               ;SVEC x=+0 y=+3 sc=0 b=0 <<<
56a0: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
56a2: 70 f7                     .dd2    $f770               ;SVEC x=+0 y=+-3 sc=0 b=7
56a4: 76 f8                     .dd2    $f876               ;SVEC x=+-2 y=+0 sc=1 b=7
56a6: 70 fb                     .dd2    $fb70               ;SVEC x=+0 y=+3 sc=1 b=7
56a8: 03 ff                     .dd2    $ff03               ;SVEC x=+3 y=+-3 sc=1 b=0
56aa: 00 d0                     .dd2    $d000               ;RTSL
```

```
56ac: 00 fb                     .dd2    $fb00               ;SVEC x=+0 y=+3 sc=1 b=0 <<<
56ae: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7
56b0: 70 ff                     .dd2    $ff70               ;SVEC x=+0 y=+-3 sc=1 b=7
56b2: 02 f0                     .dd2    $f002               ;SVEC x=+2 y=+0 sc=0 b=0
56b4: 00 d0                     .dd2    $d000               ;RTSL
```

```
56b6: 72 f8                     .dd2    $f872               ;SVEC x=+2 y=+0 sc=1 b=7 <<<
```

```
56b8: 70 fb                     .dd2    $fb70                   ;SVEC x=+0 y=+3 sc=1 b=7
56ba: 76 f8                     .dd2    $f876                   ;SVEC x=+-2 y=+0 sc=1 b=7
56bc: 70 ff                     .dd2    $ff70                   ;SVEC x=+0 y=+-3 sc=1 b=7
56be: 00 f3                     .dd2    $f300                   ;SVEC x=+0 y=+3 sc=0 b=0
56c0: 72 f8                     .dd2    $f872                   ;SVEC x=+2 y=+0 sc=1 b=7
56c2: 02 f7                     .dd2    $f702                   ;SVEC x=+2 y=+-3 sc=0 b=0
56c4: 00 d0                     .dd2    $d000                   ;RTSL
```

```
56c6: 02 f8                     .dd2    $f802                   ;SVEC x=+2 y=+0 sc=1 b=0 <<<
56c8: 70 fb                     .dd2    $fb70                   ;SVEC x=+0 y=+3 sc=1 b=7
56ca: 76 f8                     .dd2    $f876                   ;SVEC x=+-2 y=+0 sc=1 b=7
56cc: 70 f7                     .dd2    $f770                   ;SVEC x=+0 y=+-3 sc=0 b=7
56ce: 72 f8                     .dd2    $f872                   ;SVEC x=+2 y=+0 sc=1 b=7
56d0: 02 f7                     .dd2    $f702                   ;SVEC x=+2 y=+-3 sc=0 b=0
56d2: 00 d0                     .dd2    $d000                   ;RTSL
                        ;
                        ; JSRL commands to access the characters above.
                        ;
56d4: 2c cb             CharPtrTbl      .dd2    $cb2c           ;JSRL a=$0b2c ($5658) ' '
56d6: dd ca                     .dd2    $cadd                   ;JSRL a=$0add ($55ba) '0'
56d8: 2e cb                     .dd2    $cb2e                   ;JSRL a=$0b2e ($565c)
56da: 32 cb                     .dd2    $cb32                   ;JSRL a=$0b32 ($5664)
56dc: 3a cb                     .dd2    $cb3a                   ;JSRL a=$0b3a ($5674)
56de: 41 cb                     .dd2    $cb41                   ;JSRL a=$0b41 ($5682)
56e0: 48 cb                     .dd2    $cb48                   ;JSRL a=$0b48 ($5690)
56e2: 4f cb                     .dd2    $cb4f                   ;JSRL a=$0b4f ($569e)
56e4: 56 cb                     .dd2    $cb56                   ;JSRL a=$0b56 ($56ac)
56e6: 5b cb                     .dd2    $cb5b                   ;JSRL a=$0b5b ($56b6)
56e8: 63 cb                     .dd2    $cb63                   ;JSRL a=$0b63 ($56c6) '9'
56ea: 78 ca                     .dd2    $ca78                   ;JSRL a=$0a78 ($54f0) 'A'
56ec: 80 ca                     .dd2    $ca80                   ;JSRL a=$0a80 ($5500)
56ee: 8d ca                     .dd2    $ca8d                   ;JSRL a=$0a8d ($551a)
56f0: 93 ca                     .dd2    $ca93                   ;JSRL a=$0a93 ($5526)
56f2: 9b ca                     .dd2    $ca9b                   ;JSRL a=$0a9b ($5536)
56f4: a3 ca                     .dd2    $caa3                   ;JSRL a=$0aa3 ($5546)
56f6: aa ca                     .dd2    $caaa                   ;JSRL a=$0aaa ($5554)
56f8: b3 ca                     .dd2    $cab3                   ;JSRL a=$0ab3 ($5566)
56fa: ba ca                     .dd2    $caba                   ;JSRL a=$0aba ($5574)
56fc: c1 ca                     .dd2    $cac1                   ;JSRL a=$0ac1 ($5582)
56fe: c7 ca                     .dd2    $cac7                   ;JSRL a=$0ac7 ($558e)
5700: cd ca                     .dd2    $cacd                   ;JSRL a=$0acd ($559a)
5702: d2 ca                     .dd2    $cad2                   ;JSRL a=$0ad2 ($55a4)
5704: d8 ca                     .dd2    $cad8                   ;JSRL a=$0ad8 ($55b0)
5706: dd ca                     .dd2    $cadd                   ;JSRL a=$0add ($55ba)
5708: e3 ca                     .dd2    $cae3                   ;JSRL a=$0ae3 ($55c6)
570a: ea ca                     .dd2    $caea                   ;JSRL a=$0aea ($55d4)
570c: f3 ca                     .dd2    $caf3                   ;JSRL a=$0af3 ($55e6)
570e: fb ca                     .dd2    $cafb                   ;JSRL a=$0afb ($55f6)
5710: 02 cb                     .dd2    $cb02                   ;JSRL a=$0b02 ($5604)
5712: 08 cb                     .dd2    $cb08                   ;JSRL a=$0b08 ($5610)
5714: 0e cb                     .dd2    $cb0e                   ;JSRL a=$0b0e ($561c)
5716: 13 cb                     .dd2    $cb13                   ;JSRL a=$0b13 ($5626)
5718: 1a cb                     .dd2    $cb1a                   ;JSRL a=$0b1a ($5634)
571a: 1f cb                     .dd2    $cb1f                   ;JSRL a=$0b1f ($563e)
571c: 26 cb                     .dd2    $cb26                   ;JSRL a=$0b26 ($564c) 'Z'
                        ;
                        ; English message offsets.
                        ;
571e: 0b                EnglishTextTbl  .dd1    $0b             ;HIGH SCORES
571f: 13                        .dd1    $13                     ;PLAYER
5720: 19                        .dd1    $19                     ;YOUR SCORE IS ...
5721: 2f                        .dd1    $2f                     ;PLEASE ENTER YOUR ...
5722: 41                        .dd1    $41                     ;PUSH ROTATE TO ...
5723: 55                        .dd1    $55                     ;PUSH HYPERSPACE WHEN ...
5724: 6f                        .dd1    $6f                     ;PUSH START
5725: 77                        .dd1    $77                     ;GAME OVER
5726: 7d                        .dd1    $7d                     ;1 COIN 2 PLAYS
5727: 87                        .dd1    $87                     ;1 COIN 1 PLAY
5728: 91                        .dd1    $91                     ;2 COINS 1 PLAY
                        ;
                        ; Message text.  This uses 5 bits per character, storing 3 characters in 2
                        ; bytes.  The character index mapping is:
                        ;
                        ;  1 - space
                        ;  2 - '0'
                        ;  3 - '1'
                        ;  4 - '2'
                        ;  [5,30] - [A,Z]
                        ;
                        ; For example, the first message begins with "HIG", which is stored:
                        ;
                        ;     H     I     G
```

```
                ;   01100 01101 01011 0 = 01100011 01010110 = $63 $56
                ;
                ; The end of the string is identified by index zero, or (if all three characters
                ; are present) by setting the low bit of the word to 1.
                ;
5729: 63 56 60 6e+              .bulk   $63,$56,$60,$6e,$3c,$ec,$4d,$c0 ;HIGH SCORES
5731: a4 0a ea 6c+              .bulk   $a4,$0a,$ea,$6c,$08,$00 ;PLAYER
5737: ec f2 b0 6e+              .bulk   $ec,$f2,$b0,$6e,$3c,$ec,$48,$5a,$b8,$66,$92,$42,$9a,$82,$c3,$12 ;YOUR SCORE IS ONE OF THE TEN BEST
                                 +      $0e,$12,$90,$4c,$4d,$f1
574d: a4 12 2d d2+              .bulk   $a4,$12,$2d,$d2,$0a,$64,$c2,$6c,$0f,$66,$cd,$82,$6c,$9a,$c3,$4a ;PLEASE ENTER YOUR INITIALS
                                 +      $85,$c0
575f: a6 6e 60 6c+              .bulk   $a6,$6e,$60,$6c,$9e,$0a,$c2,$42,$c4,$c2,$ba,$60,$49,$f0,$0c,$12 ;PUSH ROTATE TO SELECT LETTER
                                 +      $c6,$12,$b0,$00
5773: a6 6e 60 58+              .bulk   $a6,$6e,$60,$58,$ed,$12,$b5,$e8,$29,$d2,$0e,$d8,$4c,$82,$82,$70 ;PUSH HYPERSPACE WHEN LETTER IS CORR
                                 +      $c2,$6c,$0b,$6e,$09,$e6,$b5,$92,$3e,$00
578d: a6 6e 60 6e+              .bulk   $a6,$6e,$60,$6e,$c1,$6c,$c0,$00 ;PUSH START
5795: 59 62 48 66+              .bulk   $59,$62,$48,$66,$d2,$6d ;GAME OVER
579b: 18 4e 9b 64+              .bulk   $18,$4e,$9b,$64,$09,$02,$a4,$0a,$ed,$c0 ;1 COIN 2 PLAYS
57a5: 18 4e 9b 64+              .bulk   $18,$4e,$9b,$64,$08,$c2,$a4,$0a,$e8,$00 ;1 COIN 1 PLAY
57af: 20 4e 9b 64+              .bulk   $20,$4e,$9b,$64,$b8,$46,$0d,$20,$2f,$40 ;2 COINS 1 PLAY
                ;
                ; Sine lookup table, for vertical thrust.  Offset by 64 to get cosine for
                ; horizontal thrust.
                ;
57b9: 00 03 06 09+ ThrustTbl    .bulk   $00,$03,$06,$09,$0c,$10,$13,$16,$19,$1c,$1f,$22,$25,$28,$2b,$2e
                                 +      $31,$33,$36,$39,$3c,$3f,$41,$44,$47,$49,$4c,$4e,$51,$53,$55,$58
                                 +      $5a,$5c,$5e,$60,$62,$64,$66,$68,$6a,$6b,$6d,$6f,$70,$71,$73,$74
                                 +      $75,$76,$78,$79,$7a,$7a,$7b,$7c,$7d,$7d,$7e,$7e,$7e,$7f,$7f,$7f
                                 +      $7f
57fa: 00 00 00 00+              .junk   6                       ;unused
                                .adrend ↑ $5000
```

Symbol Table

No exported symbols found.
