```
;
; Reverse engineering of Robotron 2084 Solid Blue Label by Scott
Tunstall (Paisley, Scotland.)
;
; All questions, comments, corrections - please send to
scott.tunstall@ntlworld.com
;
;

num_players_d EQU $0040
credits_d EQU $0051
JMP_PRINT_STRING_LARGE_FONT EQU $5F99
PRINT_STRING_LARGE_FONT EQU $6147
TEXT_FUNCTIONS EQU $61A2                    ; changed from subs to
functions
TEXT_PTRS EQU $6291
COPY_NIB_XYB1 EQU $6F0C
def_wel_msg_ptr EQU $6F0F
COPY_NIB_XYB EQU $6F11
CLEAR_CMOS EQU $6F21
LOAD_CMOS_DEFS1 EQU $6F2C
LOAD_CMOS_DEFS2 EQU $6F3B
def_wel_msg EQU $6F65

object_metadata_list_pointer EQU $9811      ; linked list of object
metadata. see $D196
object_list_pointer EQU $9813               ; linked list of
objects, begins at $A9E0
function_call_list_pointer EQU $9815        ; maintains a forward
only linked list of functions to call - see $D1E3

spheroids_enforcers_quarks_sparks_shells EQU $9817     ; pointer to
linked list of spheroids, enforcers, quarks, sparks and tankshells.
free_object_list_pointer EQU $981B                     ; pointer to
a linked list of object entries free to use.
second_object_metadata_list_pointer EQU $981D          ; pointer to
a linked list of object metadata, used by progs and cruise missiles.
Begins at $B0E8. See code at $D705
family_list_pointer EQU $981F                          ; pointer to
linked list of all family members
grunts_hulks_brains_progs_cruise_tanks EQU $9821       ; pointer to
linked list of grunts, hulks, brains, progs, cruise missiles and
tanks
electrode_list_pointer EQU $9823                       ; pointer to
linked list of all electrodes


current_wave EQU $982B
current_lives_left EQU $982C

current_player EQU $983F                    ; 1 = currently player
one playing, 2 = currently player two playing
num_players EQU $9840                       ; 1 = one player game,
2 = two player game
```

```
rom_control_flag EQU $9845                    ; used to set
rom_enable_scr_ctrl.
family_member_list_entry_pointer EQU $9849    ; not same as $981F.
This field is a pointer to an entry in a quick lookup list used by
hulks to find family members to target.
                                              ; the list starts at #
$B354 and ends at #$B3A4 (50 bytes, 2 bytes per family member
pointer, meaning can hold 25 family members max)

credits EQU $9851
draw_flags EQU $9859                           ; Used in interrupt
handler drawing routines.
player_object_start EQU $985A                  ; start of player
object in memory.
player_animation_frame_metadata_pointer EQU $985C
player_blitter_destination $985E

player_x EQU $09864
player_y EQU $09866
animation_index EQU $9870                                  ; index
into current player animation (0-based)

9872

lasers_fired_by_player EQU $9887            ; number of lasers fired
by the player.
laser_horizontal_direction EQU $9888        ; used in collision
detection routines. horizontal axis of the player's laser ($FF =
left, 0 = none, $1 = right)
laser_vertical_direction EQU $9889          ; used in collision
detection routines. vertical axis of the player's laser ($FF = up, 0
= none, 1=down)
number_of_sparks_on_screen EQU $988A         ; current number of
enforcer missiles (sparks) on screen
grunt_list EQU $988B                          ; linked list of grunts

number_of_family_members_saved EQU $988D     ; number of family
members saved this wave
number_of_cruise_missiles_on screen EQU $988E     ; number of cruise
missiles (missiles fired by brain) on screen
current_wall_colour EQU $988F                ; colour of the wall at
edge of playfield
current_electrode_colour EQU $9890           ; colour to draw
electrodes in
flattened_laser_colour EQU $9891             ;
current_electrode_animation_frame_metadata_list EQU $9893
brain_progging_flag EQU $9895                ; when non-zero it
means brain is progging a human
explosion_list_entry_pointer

temp_enforcer_count EQU $98ED
tank_shell_count EQU $98F1                   ; number of tank shells
currently on screen
```

```
; $B3A4 - $B3E3 reserved for use by electrodes


highscore EQU $B3E8
hs_inits EQU $B3EA

; $BDE4 is the start of the game state for player 1
p1_score EQU $BDE4
p1_next_free_man EQU $BDE8
p1_men EQU $BDEC
p1_wave EQU $BDED

???? EQU $BDEE
???? EQU $BDEF

; These variables are initialised by the code @ $2B0B
p1_grunts EQU $BDFA
p1_electrodes EQU $BDFB
p1_mommies EQU $BDFC
p1_daddies EQU $BDFD
p1_mikeys EQU $BDFE
p1_hulks EQU $BDFF
p1_brains EQU $BE00
p1_sphereoids EQU $BE01
p1_quarks EQU $BE02
p1_tanks EQU $BE03


; $BE20 is the start of the game state for player 2

p2_score EQU $BE20
p2_next_free_man EQU $BE24
p2_men EQU $BE28
p2_wave EQU $BE29
p2_grunts EQU $BE36
p2_electrodes EQU $BE37
p2_mommies EQU $BE38
p2_daddies EQU $BE39
p2_mikeys EQU $BE3A
p2_hulks EQU $BE3B
p2_brains EQU $BE3C
p2_sphereoids EQU $BE3D
p2_quarks EQU $BE3E
p2_tanks EQU $BE3F

; I've not applied labels to these guys because I'm not sure what to
call them!
$BE5C - used by grunt AI to determine how often grunts move. See
$39E6 . This field is updated when grunts are killed, to make game
faster - see $3A96.
$BE5D - used by grunt AI to throttle the speed of the grunts. This
represents the lowest value that can go into $BE5C - see $3A9A.
$BE5E - used by spheroid and quark initialisation routines when
determining how many enforcers and tanks respectively to drop. See
```

$1193 and $4B66
$BE5F - used by enforcer. This value is used to determine when an enforcer fires a spark. See $136D
$BE60 - used by sphereoid to determine delay before spawning enforcer. See $118B and $11F2
$BE61 - used by hulk. This value is used to determine how fast a hulk moves (ie: how often its update routine is called) - see $0098
$BE62 - used by brain. This value is used to determine how often to fire a cruise missile at the player. See $1B46
$BE63 - used by brain. This value is used to determine how fast a brain moves (ie: how often its update routine is called) - see $1C9C
$BE64 - used by tank. This value is used to determine how often a tank fires a shell. See $4D55
$BE65 - used by
$BE66 - used by quark to determine delay before spawning tank. See $4B5E

```
cur_grunts EQU $BE68
cur_electrodes EQU $BE69
cur_mommies EQU $BE6A
cur_daddies EQU $BE6B
cur_mikeys EQU $BE6C
cur_hulks EQU $BE6D
cur_brains EQU $BE6E
cur_sphereoids EQU $BE6F
cur_quarks EQU $BE70
cur_tanks EQU $BE71
stacktop EQU $BF70
color_registers EQU $C000
widget_pia_dataa EQU $C804
widget_pia_ctrla EQU $C805
widget_pia_datab EQU $C806
widget_pia_ctrlb EQU $C807
rom_pia_dataa EQU $C80C
rom_pia_ctrla EQU $C80D
rom_pia_datab EQU $C80E
rom_pia_ctrlb EQU $C80F
rom_enable_scr_ctrl EQU $C900
start_blitter EQU $CA00
blitter_mask EQU $CA01
blitter_source EQU $CA02
blitter_dest EQU $CA04
    blitter_dest_h EQU $CA04
    blitter_dest_l EQU $CA05


; mamedev.org - http://mamedev.org/source/src/mame/drivers/
williams.c.html contains incorrect information about the Williams'
blitter.
; their docs say the width and height is written to $CA07 and $CA06
respectively, which is wrong.
; The correction is that that width is written to $CA06 and height
to $CA07.  Please see http://www.seanriddle.com/blittest.html
; I proved this with my own experiments, specifically within the
```

```
code block $8DC9

blitter_w_h EQU $CA06
    blitter_width EQU $CA06
         blitter_height EQU $CA07

vidctrs  EQU $CB00                          ; raster position
watchdog EQU $CBFF


; CMOS settings
extra_man_every EQU $CC01                   ; 20 = every 20000, 25 =
every 25000, 30 = every 30000, 50 = every 50000
fancy_attract_mode_on_off EQU $CC13         ; 1 = on, 0 = off
turns_per_player EQU $CC03
difficulty_of_play EQU $CC15
letters_for_highest_score_name EQU $CC17


credits_cmos EQU $CD00
top_score EQU $CD32
high_scores EQU $CD68
score_chksum EQU $CD60
CLR_SCREEN1 EQU $D012
LOAD_DA51_PALETTE1 EQU $D033
FLIP_SCR_UP1 EQU $D099
FLIP_SCR_DOWN1 EQU $D09C
LDA_NIB_X1 EQU $D0A2
STA_NIB_X1 EQU $D0AB
FLIP_SCR_UP EQU $D4FC
FLIP_SCR_DOWN EQU $D503
LDA_NIB_X EQU $D512
STA_NIB_X EQU $D52B
ADDA_CREDS EQU $D541
LOAD_DA51_PALETTE EQU $D795
colorpalette2 EQU $DA51
CLR_SCREEN EQU $DB7C
CHECK_CMOS1 EQU $E3D0
GET_INITIALS1 EQU $E3D3
CHKSUM_SCORES EQU $E5BC
CHECK_CMOS EQU $E5F5
MOVE_SCORES EQU $E6CC
COPY_XYA EQU $E6EC
GET_INITIALS EQU $E6F7
CHECK_SCORE EQU $E72A
CHECK_NUM_SCORES1 EQU $E82C
CHECK_NUM_SCORES2 EQU $E830
CMP_HSINIT_X EQU $E85B
SAVE_HS EQU $E875
RESET EQU $F431
LOAD_F60B_PALETTE EQU $F5F5
colorpalette1 EQU $F60B
LOAD_F6EE_PALETTE EQU $F6AD
colorpalette3 EQU $F6EE
```

```
RAM_TEST EQU $FD65
CHK_ROM_CHKSUMS EQU $FF3F
rom_checksums EQU $FFB5



ORG $0000

*** Robotron Blue Label
0000: 7E 01 6D    JMP   $016D           ; initialise all hulks

0003: 7E 02 B2    JMP   $02B2           ; initialise all family
members

0006: 7E 00 9E    JMP   $009E           ; ensure object stays in
bounds

0009: 7E 03 51    JMP   $0351


; pointer to "1000" animation frame metadata - this is read by $1329
when drawing a spheroid's points value after its shot
000C: 04 85

; pointer to animation frame metadata of mommy standing still,
facing left
000E: 05 2F

; pointer to animation frame metadata of daddy standing still,
facing left
0010: 07 FF

; pointer to animation frame metadata of mikey standing still,
facing left
0012: 0B 3B

; pointer to animation frame metadata of HULK standing still, facing
left
0014: 0C F9

0016: 01 CC 03 CF 04 37
001C: D0 01 10 06 00 D0 03 04 17 00 E0 01 20 0D 00 E0
002C: 01 18

002E: 1A 00       ORCC  #$00



;
; Looks like this routine waits until some flags are cleared, and
when they are the hulk can go stomping!
;
;
```

```
0030: 96 59       LDA   $59
0032: 85 7F       BITA  #$7F
0034: 27 08       BEQ   $003E             ; if bits 0..6 are clear
then go to the animate hulk routines
0036: 86 08       LDA   #$08              ; game cycles before
calling routine below
0038: 8E 00 30    LDX   #$0030            ; address of routine to
call
003B: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call



ANIMATE_HULK:

003E: AE 47       LDX   $0007,U           ; get pointer to hulk
object

COMPUTE_HULK_ANIMATION_FRAME:
0040: 10 AE 4D    LDY   $000D,U           ; get hulk animation
table pointer (see $01CC for description of how it's laid out)
0043: A6 4B       LDA   $000B,U           ; get animation index
into A
0045: 31 A6       LEAY  A,Y               ; Y+= A. Now Y points to
correct animation table entry.
0047: E6 A4       LDB   ,Y                ; get byte at Y into B.
Now B is an offset to be added to $0CF9 (see $0054)
0049: 2A 04       BPL   $004F             ; if bit 7 of the byte
is not set, then we're not at the end of the animation sequence,
goto $004F
004B: 6F 4B       CLR   $000B,U           ; set index to 0, to
start the animation off again
004D: 20 F1       BRA   $0040

004F: 8B 03       ADDA  #$03              ; add 3 to bump to next
entry in animation table.
0051: A7 4B       STA   $000B,U           ; set animation index to
A
0053: 4F          CLRA                    ; clear A as we don't
want it affecting calculation below
0054: C3 0C F9    ADDD  #$0CF9            ; #$0CF9 + B to give
pointer to the animation frame metadata for correct hulk animation
frame
0057: ED 02       STD   $0002,X           ; store D in animation
frame metadata pointer.
0059: A6 21       LDA   $0001,Y           ; read X delta of
animation table entry
005B: 5F          CLRB
005C: 47          ASRA                    ; move bit 0 of A into
carry, while preserving bit 7 (thus retaining sign bit)
005D: 56          RORB                    ; and move carry into
most significant bit of B. Now A = whole part of delta, B =
```

```
fractional part
005E: E3 0A       ADDD  $000A,X             ; add to hulk's X
coordinate
0060: 34 06       PSHS  B,A                 ; save D (computed new X
coordinate) on stack
0062: E6 22       LDB   $0002,Y             ; read Y delta of
animation table entry
0064: EB 0C       ADDB  $000C,X             ; add to hulk's Y
coordinate
0066: 8D 36       BSR   $009E               ; ensure computed X and
Y coordinates are in bounds
0068: 27 04       BEQ   $006E               ; if components are in
bounds, update actual X and Y coordinates
006A: 32 62       LEAS  $0002,S             ; discard B and A pushed
on the stack @ $0060
006C: 20 22       BRA   $0090

006E: E7 0C       STB   $000C,X             ; update Y coordinate of
hulk
0070: 35 06       PULS  A,B                 ; restore computed new X
coordinate from stack (see $0060)
0072: ED 0A       STD   $000A,X             ; and store in X
coordinate of hulk
0074: E6 0C       LDB   $000C,X
0076: EE 02       LDU   $0002,X             ; get animation frame
metadata pointer into U
0078: 8E 98 23    LDX   #$9823              ; pointer to linked list
of electrodes (hulks stomp electrodes)
007B: 34 46       PSHS  U,B,A
007D: BD D0 27    JSR   $D027               ; JMP $D7C9 - collision
detection function
0080: 35 46       PULS  A,B,U
0082: 8E 98 1F    LDX   #$981F              ; pointer to linked list
of family members (hulks kill family members)
0085: BD D0 27    JSR   $D027               ; JMP $D7C9 - collision
detection function
0088: DE 15       LDU   $15
008A: AE 47       LDX   $0007,U             ; get pointer to hulk
object into X
008C: 6A 4C       DEC   $000C,U             ; decrement hulk move
counter
008E: 26 02       BNE   $0092               ; if hulk move counter !
=0 goto $0092
0090: 8D 74       BSR   $0106               ; change direction
0092: BD D0 8D    JSR   $D08D               ; JMP $DB2F - draw
object
0095: 8E 00 3E    LDX   #$003E              ; address of function to
call
0098: B6 BE 61    LDA   $BE61               ; get counter to say how
long it will be before this routine is called again (to speed up
hulk movement as wave progresses.)
009B: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call
```

```
; Ensure an object is within bounds of the playfield
;
; A = X coordinate of where object would *like* to move to
; B = Y coordinate of where object would *like* to move to
; X = object pointer
;
; Returns:
; Zero flag set if object coordinates in A & B are valid
;

ENSURE_OBJECT_IN_BOUNDS:
009E: 34 06       PSHS  B,A
00A0: 81 07       CMPA  #$07                ; X < 7?
00A2: 25 10       BCS   $00B4               ; yes, goto $00B4
00A4: C1 18       CMPB  #$18                ; Y < #$18?
00A6: 25 0C       BCS   $00B4               ; yes, goto $00B4
00A8: E3 98 02    ADDD  [$02,X]             ; add in width and
height of the object
00AB: 81 8F       CMPA  #$8F                ; X+width > #$8F ?
00AD: 22 05       BHI   $00B4               ; yes, so leave routine
00AF: C1 EA       CMPB  #$EA                ; Y+height > $#EA ?
00B1: 22 01       BHI   $00B4               ; yes, so leave routine
00B3: 4F          CLRA                      ; otherwise set zero
flag (A is restored by next instruction)
00B4: 35 86       PULS  A,B,PC ;(PUL? PC=RTS)


;
; Make the hulk react to being hit by player bullets
;

HULK_BULLET_COLLISION_HANDLER:
00B6: 96 48       LDA   $48                 ; player collision
detection?
00B8: 26 49       BNE   $0103               ; yes
00BA: 96 88       LDA   $88                 ; A = bullet X delta
00BC: 5F          CLRB
00BD: 0D 84       TST   $84                 ; read random number
variable
00BF: 2B 01       BMI   $00C2               ; if bit 7 set, go to
$00C2
00C1: 48          ASLA                      ; A *= 2 (double the
impact of the bullet on the X axis)
00C2: E3 0A       ADDD  $000A,X             ; D+= hulk X coordinate
00C4: 34 06       PSHS  B,A                 ; save D (computed new X
coordinate) on the stack for later (see 00DD)
00C6: D6 89       LDB   $89                 ; B = bullet Y delta
00C8: 96 86       LDA   $86                 ; read random number
variable
00CA: 81 C0       CMPA  #$C0                ; if > #$C0
00CC: 24 01       BCC   $00CF               ; jump to $CF
00CE: 58          ASLB                      ; else double the impact
of the bullet on the Y axis
```

```
00CF: EB 0C       ADDB  $000C,X             ; B+= hulk Y coordinate
00D1: A6 E4       LDA   ,S                  ; A = hulk computed new
X coordinate from stack
00D3: 8D C9       BSR   $009E               ; ensure hulk X and Y
are within boundaries
00D5: 27 04       BEQ   $00DB               ; if in boundaries, go
to $00DB
00D7: 32 62       LEAS  $0002,S             ; adjust stack pointer
to discard X and Y coordinates pushed earlier
00D9: 20 21       BRA   $00FC               ; and finish processing
hulk

; hulk's new position has been validated, so update object's X and Y
coordinates
00DB: E7 0C       STB   $000C,X             ; hulk Y coordinate = B
00DD: 35 06       PULS  A,B                 ; restore D
00DF: ED 0A       STD   $000A,X             ; hulk X coordinate = D
00E1: E6 0C       LDB   $000C,X             ; get hulk Y coordinate
into B
00E3: EE 02       LDU   $0002,X             ; get current animation
frame metadata pointer
00E5: 34 10       PSHS  X
00E7: 8E 98 23    LDX   #$9823              ; start of linked list
for electrodes
00EA: 34 46       PSHS  U,B,A
00EC: BD D0 27    JSR   $D027               ; JMP $D7C9 - collision
detection function
00EF: 35 46       PULS  A,B,U
00F1: 8E 98 1F    LDX   #$981F              ; start of linked list
for family members
00F4: BD D0 27    JSR   $D027               ; JMP $D7C9 - collision
detection function
00F7: 35 10       PULS  X
00F9: BD D0 8D    JSR   $D08D               ; JMP $DB2F - draw
object
00FC: CC 00 1C    LDD   #$001C
00FF: BD D0 4B    JSR   $D04B               ; JMP $D3C7
0102: 39          RTS

0103: 7E D0 18    JMP   $D018               ; JMP $D7C9 - collision
detection function




CHANGE_HULK_DIRECTION:
0106: 96 86       LDA   $86                 ; get a random number
0108: 84 1F       ANDA  #$1F                ; mask with #$1F (so
that number lays in 0..31 decimal)
010A: 4C          INCA                      ; add 1 to it (to ensure
its nonzero)
010B: A7 4C       STA   $000C,U             ; set in "move count"
variable
```

```
; LDY [$09,U] gets a pointer to the *next* object in the object
metadata linked list.
; An interesting bug: this instruction is supposed to return an
active object but there are times when a hulk is created and there's
nothing
; else on screen yet. As a result *(U + 9) computes to WORD 0 (which
I call NULL) and the contents of memory addresses $0000 and $0001
are read into Y.
; You get Y=7E 01. Go look at ORG $0000 and see for yourself...
;
; Now, $7E01 certainly does not point to any real object, so the
hulk ends up going on a wild goose chase, chasing an object that
does not exist.
; This probably explains why hulks wander off doing their own thing
and getting stuck in the corner at times.
;

010D: 10 AE D8 09 LDY   [$09,U]             ; get pointer to target
object to stalk
0111: 26 04       BNE   $0117               ; if not NULL goto $117
0113: 10 8E 98 5A LDY   #$985A              ; player_object_start.
Looks like we're wanting to make the hulk stalk the player
0117: EC 4D       LDD   $000D,U             ; get hulk animation
table pointer (see $01CC for description of how it's laid out)
0119: 10 83 01 CC CMPD  #$01CC              ; is the hulk moving
left?
011D: 27 26       BEQ   $0145               ; yes
011F: 10 83 01 D9 CMPD  #$01D9              ; is the hulk moving
right?
0123: 27 20       BEQ   $0145               ; yes


; this code makes the hulk move around the object pointed to in
register Y - its "target".
; think of it like this - the hulk identifies a point (X,Y) to move
to, where
; point.X is in the range ((current target object's X coordinate -
15) ... current target's X coordinate + 16)
; point.Y is in the range ((current target object's Y coordinate -
15) ... current target's Y coordinate +16)
;
0125: 96 84       LDA   $84                 ; get a random number
0127: 84 1F       ANDA  #$1F                ; mask in lower 5 bits
(to give a number between 0..31 decimal)
0129: 8B F0       ADDA  #$F0                ; add #$F0 (-15
decimal) - this should give a number between -15 and 16
012B: AB 24       ADDA  $0004,Y             ; add to most
significant byte of target objects blitter destination (which in
this case is the X component of address)
012D: 81 8F       CMPA  #$8F                ; far right position of
playfield area
012F: 23 06       BLS   $0137
0131: 81 CF       CMPA  #$CF                ; No idea why this is
here, this is an invalid coordinate.
```

```
0133: 23 02       BLS   $0137
0135: 86 07       LDA   #$07                ; left-most of position
playfield area
0137: A1 04       CMPA  $0004,X             ; if 7 is <= the most
significant byte of the hulk's blitter destination, the hulk will
move left
0139: 23 05       BLS   $0140
013B: CC 01 D9    LDD   #$01D9              ; pointer to animation
table to make hulk move right
013E: 20 1F       BRA   $015F               ; set hulk animation
table pointer

0140: CC 01 CC    LDD   #$01CC              ; pointer to animation
table to make hulk move left
0143: 20 1A       BRA   $015F               ; set hulk animation
table pointer

0145: 96 85       LDA   $85                 ; get another random
number
0147: 84 1F       ANDA  #$1F                ; mask in lower 5 bits
(to give a number between 0..31 decimal)
0149: 8B F0       ADDA  #$F0                ; add #$F0 (-15
decimal) - this should give a number between -15 and 16
014B: AB 25       ADDA  $0005,Y             ; add to least
significant byte of target objects blitter destination (the Y
component)
014D: 81 06       CMPA  #$06                ; far left edge of
playfield
014F: 24 02       BCC   $0153               ; if A>6 goto $0153
0151: 86 EA       LDA   #$EA                ; far right edge of
playfield
0153: A1 05       CMPA  $0005,X             ; if A is <= the least
significant byte of the hulk's blitter destination (the Y
component), the hulk will move up
0155: 23 05       BLS   $015C
0157: CC 01 E6    LDD   #$01E6              ; pointer to animation
table to make hulk move down
015A: 20 03       BRA   $015F

015C: CC 01 F3    LDD   #$01F3              ; pointer to animation
table to make hulk move up
015F: ED 4D       STD   $000D,U             ; set animation table
pointer to U
0161: 4F          CLRA
0162: A7 4B       STA   $000B,U             ; set index into
animation table to 0 - first frame
0164: E6 D8 0D    LDB   [$0D,U]             ; read offset from
animation table (remember, each entry in the table has 3 bytes,
first byte is offset to add to animation frame metadata list start )
0167: C3 0C F9    ADDD  #$0CF9              ; add offset to
animation frame metadata list start for hulk, giving you a pointer
in D to the animation frame metadata ...
016A: ED 02       STD   $0002,X             ; store to animation
frame metadata pointer
```

```
016C: 39          RTS


INITIALISE_ALL_HULKS:
016D: B6 BE 6D    LDA   cur_hulks           ; get count of hulks
into A
0170: 34 02       PSHS  A                   ; save count on stack
for use in loop
0172: 27 06       BEQ   $017A               ; if 0, exit
0174: 8D 06       BSR   $017C               ; create and initialise
a hulk
0176: 6A E4       DEC   ,S                  ; decrement hulk count
on stack
0178: 26 FA       BNE   $0174               ; if we've not set up
all hulks, go back to $0174
017A: 35 82       PULS  A,PC ;(PUL? PC=RTS)




;
; This function initialises a hulk object.
;
;

INITIALISE_SINGLE_HULK:
017C: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
017F: 00 30       ; address of function to call
; at this point, X=  newly created object metadata for hulk
0181: 33 84       LEAU  ,X                  ; U = X = pointer to
object metadata
0183: BD D0 7B    JSR   $D07B               ; JMP $D2DA - reserve
entry in list used by grunts, hulks, brains, progs, cruise missiles
and tanks (starts at $9821)
0186: CC 0C F9    LDD   #$0CF9              ; pointer to animation
frame metadata (first 2 bytes at 0CF9 are 07 10 width & height ,
next 2 bytes 0D 1D pointer to image)
0189: ED 02       STD   $0002,X             ; store current
animation frame metadata pointer
018B: ED 88 14    STD   $14,X               ; store previous
animation frame metadata pointer (previous = current)
018E: EF 06       STU   $0006,X             ; set pointer to object
metadata in this object
0190: AF 47       STX   $0007,U             ; store address of this
object to U + 7
0192: CC 00 B6    LDD   #$00B6              ; address of function
to call when hulk is hit
0195: ED 08       STD   $0008,X
0197: BD 38 8E    JSR   $388E               ; JMP $38FE -compute
safe rectangle for player

; find a start position for the hulk
019A: BD 26 C3    JSR   $26C3               ; JMP $3199 - get
```

```
random position on playfield for object (returns: A = X coordinate,
B = Y coordinate)

; this block of code is here to ensure the hulk X and Y coordinates
are valid
; and that the hulk's not too near to the player
019D: D1 2B       CMPB  $2B                 ;
019F: 23 0C       BLS   $01AD
01A1: D1 2C       CMPB  $2C
01A3: 24 08       BCC   $01AD               ; >
01A5: 91 2D       CMPA  $2D
01A7: 23 04       BLS   $01AD               ; <=
01A9: 91 2E       CMPA  $2E
01AB: 23 ED       BLS   $019A               ; if the coordinate is
invalid, then go get another one

; if we get here, hulk X and Y coordinates are valid
01AD: ED 04       STD   $0004,X             ; set "last" blitter
destination
01AF: A7 0A       STA   $000A,X             ; set current hulk X
coordinate (whole part)
01B1: E7 0C       STB   $000C,X             ; set current hulk Y
coordinate
01B3: 96 84       LDA   $84                 ; get a random number
01B5: 81 C0       CMPA  #$C0
01B7: 23 05       BLS   $01BE               ; if number <= #$C0
(192 decimal) make the hulk stalk a family member
01B9: CC B3 A2    LDD   #$B3A2              ; set target pointer to
very last entry in family member list - which might not have
anything in it
01BC: 20 02       BRA   $01C0

; get a family member for the hulk to stalk
01BE: 8D 77       BSR   $0237               ; get a family member
from the family member list into D
01C0: ED 49       STD   $0009,U             ; store into target
01C2: BD 01 06    JSR   $0106               ; pick a direction for
hulk
01C5: BD 38 8B    JSR   $388B               ; JMP $393C - blit hulk
in solid colour invisible to player
01C8: 6F 88 13    CLR   $13,X
01CB: 39          RTS



; HULK MOVEMENT TABLES

; Animation tables. 13 bytes per animation sequence, terminated by
$FF.
;
; format (offsets are zero-based):
;
; byte 0: offset into animation frame metadata list (note it's a
multiple of 4, each entry in animation frame metadata list is 4
```

```
bytes long).
; Add this offset to #$0CF9 and that gives you a pointer to the
animation frame metadata to use to draw hulk.
;
; byte 1 is a packed byte which contains a delta to add to hulk's
current X coordinate. The byte isn't added "as-is", but instead goes
under this process:
;
; LDA, byte
; CLRB
; ASRA          ; move bit 0 of A into carry
; RORB          ; move carry into bit 7 of B.  A is now the whole
part of the step, and B is fractional part of step (in 256ths).
;               ; for example, if the hulk was to move 2.5 pixels
each step, A = 1 (remember 1 byte = 2 pixels), B = 128 (given that
128 is half of 256.... )
;
; finally:
; ADDD  $000A,X  - to compute where the hulk would LIKE to move to,
in the X axis, and store result in D.
;
;
; byte 2 is a signed byte to add to hulk's current Y coordinate.
;
; *Remember, the Hulk only goes in 4 directions..... hence only 4
animations :)

; animation 1 (move hulk left)
01CC: 00 FD 00   - frame 1
01CF: 04 FC 00   - frame 2
01D2: 00 FD 00   - frame 3
01D5: 08 FC 00   - frame 4
01D8: FF         - indicates end of animation sequence, roll back to
frame 1 (see $004B for code which does just that)

; animation 2 (move hulk right)
01D9: 0C 03 00
01DC: 10 04 00
01DF: 0C 03 00
01E2: 14 04 00
01E5: FF

; animation 3 (move hulk down)
01E6: 18 00 02
01E9: 1C 00 02
01EC: 18 00 02
01EF: 20 00 02
01F2: FF

; animation 4 (move hulk up)
01F3: 18 00 FE
01F6: 1C 00 FE
01F9: 18 00 FE
01FC: 20 00 FE
```

```
01FF: FF



; Clear the family member list. zero from $B354 to B3A3.
;
; Hulks and brains use this list to find family members to target.
;
;

CLEAR_FAMILY_MEMBER_LIST:
0200: 8E B3 54    LDX   #$B354                 ; start of family
member list
0203: 9F 49       STX   $49
0205: 6F 80       CLR   ,X+
0207: 8C B3 A4    CMPX  #$B3A4
020A: 26 F9       BNE   $0205
020C: 39          RTS


; Add a family member object to the family member list.
;
; X =  pointer to family member object to add to list
;
ADD_ENTRY_TO_FAMILY_MEMBER_LIST:
020D: 34 16       PSHS  X,B,A
020F: 8E B3 54    LDX   #$B354
0212: EC 81       LDD   ,X++                   ; read entry at X and
add 2 to X
0214: 26 FC       BNE   $0212                  ; if entry is not
null, its occupied, so go read next entry
0216: EC 62       LDD   $0002,S                ; D = X from stack,
which is pointer to family member object to add
0218: ED 1E       STD   -2,X                   ; write family member
object into list
021A: 35 96       PULS  A,B,X,PC ;(PUL? PC=RTS)



; This routine removes a family member from the family member list.
; The family member is either dead or has been saved by the player.
; Brains and Hulks will no longer consider this object in their AI.
;
; X = pointer to family member object to remove from list.
;
;

REMOVE_FAMILY_LIST_ENTRY:
021C: 34 16       PSHS  X,B,A
021E: 8E B3 54    LDX   #$B354              ; start of object list
0221: EC 62       LDD   $0002,S             ; D = X from stack
0223: 10 A3 81    CMPD  ,X++                ; compare D to *X
0226: 27 09       BEQ   $0231               ; if a match, goto
```

```
$0231
0228: 8C B3 A4    CMPX  #$B3A4              ; have we hit end of
list?
022B: 26 F6       BNE   $0223               ; no, goto $0223
022D: 1A 10       ORCC  #$10                ; Disable interrupts
022F: 20 FE       BRA   $022F               ; infinite loop, force
watchdog to reset
0231: 4F          CLRA                      ; clear A & B (which
both form D) - this will zero out (nullify) the object pointer in
the list
0232: 5F          CLRB                      ; effectively saying
"this object does not exist any more"
0233: ED 1E       STD   -2,X                ; overwrite the object
pointer in the list (remember X++ increments X by 2 after the CMPD
instruction, moving us past the entry list we want)
0235: 35 96       PULS  A,B,X,PC ;(PUL? PC=RTS)



;
; Get next family member from the list of family members.
;
; Returns: D = family member
;

GET_FAMILY_MEMBER_FROM_LIST:
0237: 34 10       PSHS  X
0239: 9E 49       LDX   $49                 ; get pointer to
current entry in family member list
023B: 8C B3 A4    CMPX  #$B3A4              ; are we past the end
of the list?
023E: 25 09       BCS   $0249               ; no, goto $0249
0240: 8E B3 54    LDX   #$B354              ; yes, so reset pointer
to point to start of family member list
0243: 20 04       BRA   $0249

0245: 9C 49       CPX   $49                 ; if X == pointer to
current entry then that means the entire list has been scanned. No
family members left.
0247: 27 14       BEQ   $025D               ; if no family members
left (all dead or rescued) then goto $025D
0249: EC 81       LDD   ,X++                ; read family member
pointer entry and add 2 to X to bump X to next entry
024B: 26 0A       BNE   $0257               ; if pointer not NULL
goto $0257
024D: 8C B3 A4    CMPX  #$B3A4              ; is X past the end of
the list?
0250: 25 F3       BCS   $0245               ; no, goto $0245
0252: 8E B3 54    LDX   #$B354              ; yes, so start at
beginning of list
0255: 20 EE       BRA   $0245

0257: 9F 49       STX   $49                 ; save pointer to NEXT
family member object pointer slot in $49 - it might be null
```

```
0259: 30 1E       LEAX  $-2,X               ; X= X -2. This is
because the X++ in $0249 has added 2 to X which we need to undo
025B: 1F 10       TFR   X,D                 ; D = X
025D: 35 90       PULS  X,PC ;(PUL? PC=RTS)



ANIMATE_FAMILY_MEMBER:
025F: AE 47       LDX   $0007,U             ; get pointer to family
member object
0261: 10 AE 4D    LDY   $000D,U             ; get pointer to
animation tables (see $03CF)
0264: A6 4B       LDA   $000B,U             ; get index into tables
(a multiple of 3)
0266: 31 A6       LEAY  A,Y                 ; Y+= A
0268: E6 A4       LDB   ,Y                  ; B is now offset to
add to animation frame metadata start pointer (see $0275)
026A: 2A 04       BPL   $0270
026C: 6F 4B       CLR   $000B,U             ; reset index to 0
026E: 20 F1       BRA   $0261

0270: 8B 03       ADDA  #$03                ; bump index to next
entry in tables
0272: A7 4B       STA   $000B,U             ; update index
0274: 4F          CLRA                      ; set a to 0, so only B
is used in the ADDD below.
0275: E3 49       ADDD  $0009,U             ; add in animation
frame metadata start pointer in B.  D now points to animation frame
metadata.
0277: ED 02       STD   $0002,X             ; set current animation
frame metadata pointer
0279: A6 21       LDA   $0001,Y             ; get X delta
027B: 5F          CLRB
027C: 47          ASRA
027D: 56          RORB                      ; move bit 0 of A into
bit 7 of B. A holds whole part of X delta, and B holds fractional
part
027E: E3 0A       ADDD  $000A,X             ; D += object X
coordinate
0280: 34 06       PSHS  B,A
0282: E6 22       LDB   $0002,Y             ; get Y delta
0284: EB 0C       ADDB  $000C,X             ; B += object Y
coordinate
0286: 34 40       PSHS  U
0288: CE 98 23    LDU   #$9823              ; start of electrode
linked list
028B: BD 26 C6    JSR   $26C6               ; JMP $3085 - rectangle
intersection function
028E: 35 40       PULS  U
0290: 26 05       BNE   $0297               ; if family member has
walked into an electrode then goto $0297, to make member change
direction
0292: BD 00 06    JSR   $0006               ; JMP $009E: ensure
object stays in bounds
```

```
0295: 27 04       BEQ   $029B               ; yes, object has new
position, update coords

; if we get here, then the family member has tried to move to an
invalid position
; such as off playfield OR into an electrode.
0297: 32 62       LEAS  $0002,S             ; discard A & B pushed
on stack (see $0286 above)
0299: 20 0A       BRA   $02A5               ; make the family
member change direction, then draw the family member, and we're
done'

029B: E7 0C       STB   $000C,X             ; set object Y
coordinate
029D: 35 06       PULS  A,B
029F: ED 0A       STD   $000A,X             ; set object X
coordinate
02A1: 6A 4C       DEC   $000C,U             ; decrement walk
countdown
02A3: 26 02       BNE   $02A7               ; if 0, make family
member change direction

02A5: 8D 6B       BSR   $0312               ; call change direction
routine
02A7: BD D0 8D    JSR   $D08D               ; JMP $DB2F - draw
object
02AA: 86 08       LDA   #$08                ; delay before calling
function
02AC: 8E 02 5F    LDX   #$025F              ; address of routine to
call (animate family member)
02AF: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call


INITIALISE_FAMILY_MEMBERS:
02B2: BD 02 00    JSR   $0200               ; clear (zero) family
list from $B354 to $B3A3
02B5: 8E 0B 3B    LDX   #$0B3B              ; load X with pointer
to animation frame metadata for first mikey (memory at this $0B3B:
03,0B = width, height; 0B 6B = pointer to first mikey image)
02B8: CE 03 30    LDU   #$0330              ; Load U with address
of routine that reduces mikey count
02BB: B6 BE 6C    LDA   cur_mikeys
02BE: 8D 14       BSR   $02D4
02C0: 8E 05 2F    LDX   #$052F              ; load X with pointer
to animation frame metadata for first mommy (memory at $052F: O4,OE
= width, height; 05 5F = pointer to first mummy image)
02C3: CE 03 35    LDU   #$0335              ; Load U with address
of routine that reduces mommy count
02C6: B6 BE 6A    LDA   cur_mommies
02C9: 8D 09       BSR   $02D4
02CB: 8E 07 FF    LDX   #$07FF              ; load X with pointer
to animation frame metadata for first daddy (memory at $07FF: O5,OD
= width, height; 08 2F = pointer to first mummy image)
```

```
02CE: CE 03 3A    LDU   #$033A              ; load U with address
of routine that reduces daddy count
02D1: B6 BE 6B    LDA   cur_daddies
;
; U = routine to call to remove family member from game (ie: when
family member killed or rescued)
; X = pointer to animation frame metadata for family member
; A = number of particular family member
;
02D4: 34 52       PSHS  U,X,A
02D6: 4D          TSTA                      ; number of family type
== 0?
02D7: 27 37       BEQ   $0310               ; Yes, so just exit
02D9: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
02DC: 02 5F        ; pointer to function (which looks like address
of the animation routine)

02DE: 33 84       LEAU  ,X                  ; U += X
02E0: BD D0 87    JSR   $D087               ; JMP $D2F2    -
reserve object [for family member]
; X = freshly reserved object
02E3: EC 61       LDD   $0001,S             ; load D with pointer
to animation frame metadata, from stack
02E5: ED 02       STD   $0002,X             ; set current animation
frame metadata pointer
02E7: ED 88 14    STD   $14,X               ; set previous
animation frame metadata pointer (previous = current)
02EA: ED 49       STD   $0009,U             ; set animation frame
metadata list start pointer
02EC: EF 06       STU   $0006,X             ; save pointer to
object metadata in family member object
02EE: AF 47       STX   $0007,U             ; set pointer to this
object in U + 7
02F0: EC 63       LDD   $0003,S             ; D = routine to kill
family member
02F2: ED 08       STD   $0008,X             ; store routine
02F4: BD 26 C3    JSR   $26C3               ; JMP $3199 - get
random position on playfield for object (returns: A = X coordinate,
B = Y coordinate)
02F7: ED 04       STD   $0004,X             ; "last" blitter
destination = D
02F9: A7 0A       STA   $000A,X             ; set current family
member X coordinate (whole part)
02FB: E7 0C       STB   $000C,X             ; set current family
member Y coordinate
02FD: 96 84       LDA   $84                 ; get a random number
02FF: 84 07       ANDA  #$07
0301: 4C          INCA
0302: A7 44       STA   $0004,U
0304: 8D 0C       BSR   $0312               ; set up the initial
animation for the family member
0306: BD D0 18    JSR   $D018               ; JMP $DAF2 - blit
object
```

```
0309: BD 02 0D    JSR   $020D                ; reserve an entry for
this family member object in list in $B354 onwards... hulks use the
list to get targets!
030C: 6A E4       DEC   ,S                   ; decrement temporary
count of family member type to process
030E: 26 C9       BNE   $02D9                ; if !=0, more family
members to process
0310: 35 D2       PULS  A,X,U,PC ;(PUL? PC=RTS)


CHANGE_FAMILY_MEMBER_DIRECTION:
0312: 96 86       LDA   $86                  ; get a random number
0314: 84 7F       ANDA  #$7F                 ; mask with #$7F (127
decimal) to give number between 0 and 127
0316: 4C          INCA                       ; add 1, to ensure A is
nonzero
0317: A7 4C       STA   $000C,U              ; make A count of how
many cycles to walk in particular direction
0319: 96 84       LDA   $84                  ; get a random number
031B: 84 07       ANDA  #$07                 ; we have 7 animations
for each family member and...
031D: C6 0D       LDB   #$0D                 ; ... each animation
sequence occupies 0D (13 decimal) bytes exactly
031F: 3D          MUL                        ; multiply A by B to
give offset into animation tables
0320: C3 03 CF    ADDD  #$03CF               ; compute where in
animation tables (see 03CF for description) to start from
0323: ED 4D       STD   $000D,U              ; store result
0325: 4F          CLRA
0326: A7 4B       STA   $000B,U              ; reset index into
animation tables (see $0261)
0328: E6 D8 0D    LDB   [$0D,U]              ; read first byte from
animation tables- this byte is the offset part (see $03CF for
description)
032B: E3 49       ADDD  $0009,U              ; add offset to
animation frame metadata list start (see $02EA)
032D: ED 02       STD   $0002,X              ; set animation frame
metadata pointer
032F: 39          RTS

0330: 7A BE 6C    DEC   cur_mikeys
0333: 20 08       BRA   $033D

0335: 7A BE 6A    DEC   cur_mommies
0338: 20 03       BRA   $033D

033A: 7A BE 6B    DEC   cur_daddies



;
; This routine is called when a family member is saved by the
player, killed by a hulk, or about to be prog'd.
;
```

```
;

FAMILY_MEMBER_SAVED_KILLED_OR_PROGGED:
033D: BD 02 1C    JSR   $021C               ; remove family member
from list in $B354
0340: BD D0 8A    JSR   $D08A               ; JMP $D2D2 -
deallocate family member object
0343: BD D0 15    JSR   $D015               ; JMP $DB03 - erase
object from screen
0346: EC 04       LDD   $0004,X             ; get blitter
destination
0348: AE 06       LDX   $0006,X             ; get pointer to object
metadata into X
034A: BD D0 5D    JSR   $D05D               ; JMP $D218 -
deallocate object metadata entry
034D: 0D 95       TST   $95                 ; is the family member
being prog'd? (see $1CFF)
034F: 26 51       BNE   $03A2               ; if yes, goto $03A2
(just an RTS)
0351: BD D0 6C    JSR   $D06C               ; JMP $D32B - create
entity with params and add to linked list at $9817
0354: 03 A3       ; parameters to pass to $D06C - function to call
on next game cycle
0356: 04 7D       ; parameters to pass to $D06C - animation frame
metadata pointer (which is a 2x2 black square!)
0358: 03 90       ; parameters to pass to $D06C - function to call
when this object has a collision
; X = pointer to new object
035A: 81 89       CMPA  #$89                ;
035C: 25 02       BCS   $0360
035E: 86 89       LDA   #$89
0360: ED 04       STD   $0004,X
0362: 0D 95       TST   $95                 ; is this family member
being progged?
0364: 26 2D       BNE   $0393               ; yes, so draw the
"death" image
0366: 0D 48       TST   $48                 ; was it the player who
saved this family member?
0368: 27 29       BEQ   $0393               ; no, so draw the
"death" image
036A: 86 3C       LDA   #$3C
036C: A7 49       STA   $0009,U
036E: 0C 8D       INC   $8D                 ; increment number of
family members saved
0370: 96 8D       LDA   $8D
0372: 81 05       CMPA  #$05                ; have we saved 5
members?
0374: 23 02       BLS   $0378               ; if lower or same as
5, goto $0378
0376: 86 05       LDA   #$05                ; otherwise, cap at 5
0378: 48          ASLA                      ; A *= 2;
0379: 48          ASLA                      ; A *= 2;
037A: CE 04 81    LDU   #$0481              ; animation metadata
start for bonus. compute whether to show 1000,2000,3000,4000,5000
```

```
037D: 33 C6       LEAU  A,U                  ; U = U + A
037F: EF 02       STU   $0002,X              ; set animation
metadata pointer for object
0381: CE 03 C3    LDU   #$03C3               ; Start of points value
table. Real entries begin at $03C5
0384: 44          LSRA                       ; divide A by 2. A is
now an offset into points value table.
0385: EC C6       LDD   A,U                  ; D = *(U+A). D now
holds score parameter to pass to $DB9C.
                                             ; If A==0, score value
is 1000 points, A==2, score value 2000 points etc.
0387: BD D0 0C    JSR   $D00C                ; JMP $DB9C - update
player score
038A: CC 00 26    LDD   #$0026
038D: BD D0 4B    JSR   $D04B                ; JMP $D3C7
0390: 4F          CLRA
0391: 35 86       PULS  A,B,PC ;(PUL? PC=RTS)

0393: 86 5A       LDA   #$5A                ; countdown before image
disappears
0395: A7 49       STA   $0009,U
0397: CC 04 37    LDD   #$0437              ; pointer to "family
death" image
039A: ED 02       STD   $0002,X             ; set animation frame
metadata pointer
039C: CC 00 2B    LDD   #$002B
039F: BD D0 4B    JSR   $D04B               ; JMP $D3C7
03A2: 39          RTS


;
; Draw the points value for rescuing the family member
;
DRAW_POINTS_FOR_RESCUING_FAMILY_MEMBER:
03A3: AE 47       LDX   $0007,U             ; load X with pointer to
object
03A5: EC 04       LDD   $0004,X             ; blitter destination
03A7: 10 AE 02    LDY   $0002,X             ; pointer to image
struct
03AA: BD D0 21    JSR   $D021               ; JMP $DA82 - do blit
without transparency
03AD: 6A 49       DEC   $0009,U
03AF: 27 08       BEQ   $03B9
03B1: 86 01       LDA   #$01                ; delay before calling
function
03B3: 8E 03 A3    LDX   #$03A3              ; address of function to
call
03B6: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

03B9: BD D0 1E    JSR   $D01E               ; JMP $DABF: clear image
rectangle
03BC: DC 1B       LDD   $1B
03BE: ED 84       STD   ,X
```

```
03C0: 9F 1B       STX   $1B
03C2: 7E D0 63    JMP   $D063              ; JMP $D1F3


; points added to score when you rescue a family member
RESCUE_FAMILY_POINTS_TABLE:
03C5: 02 10                                ; 1000 points
03C7: 02 20                                ; 2000 points
03C9: 02 30                                ; 3000 points
03CB: 02 40                                ; 4000 points
03CF: 02 50                                ; 5000 points



; Family member animation tables. Used by Mommy, Daddy and Mikey.
;
; 13 (decimal) bytes per animation sequence, terminated by $FF.
;
; format of each entry:
; byte 0: offset into animation frame metadata list (note it's a
multiple of 4, each entry in animation frame metadata list is 4
bytes long)
;
; byte 1 is a packed byte containing a delta to add to family
member's current X coordinate. It needs to be unpacked before it can
be used.
; The byte represents a whole and (optionally) a fractional part.
; The byte goes under this process:
;
; LDA, byte
; CLRB              ; clear B
; ASRA              ; shift bit 0 of A into carry, preserving bit 7
(the sign bit). A now is the whole part of the delta.
; RORB              ; move carry into fractional part of number.  B
now is the fractional part of the delta.
; ADDD  $000A,X  - adds delta to the X coordinate, storing result in
D.
;
; Result:
; A = whole part of new coordinate, B = fractional part.
;
; For example, in $03D3 we have $FF as our value. Let's put this
byte through the process above:
; $FF ASR 1 = $FF (remember ASR instruction preserves bit 7.), with
1 in carry
; RORB moves carry into bit 7 of B, which was cleared to zero by the
CLRB.
; So now A = $FF, B = $80.
;
; What does this mean?
; $FF is -1 as a signed byte.
; $80 = .5 (256 / 128 represented as a fraction is .5, a half!)
;
; This means the family member will have -1.5 added to its current X
```

```
coordinate, moving the family member LEFT 2.5 pixels (remember, 1
byte = 2 pixels)
;
;
; byte 2 is a signed byte to add to the family member's current Y
coordinate. No unpacking necessary.
;


03CF: 00 FE 00
03D2: 04 FF 00
03D5: 00 FE 00
03D8: 08 FF 00
03DB: FF            - marks end of this animation sequence

03DC: 0C 02 00
03DF: 10 01 00
03E2: 0C 02 00
03E5: 14 01 00
03E8: FF            - marks end of animation sequence

03E9: 18 00 01
03EC: 1C 00 01
03EF: 18 00 01
03F2: 20 00 01
03F5: FF            - marks end of animation sequence

03F6: 24 00 FF
03F9: 28 00 FF
03FC: 24 00 FF
03FF: 2C 00 FF
0402: FF            - marks end of animation sequence

0403: 00 FE FF
0406: 08 FF FF
0409: 00 FE FF
040C: 08 FF FF
040F: FF            - marks end of animation sequence

0410: 0C 02 FF
0413: 10 01 FF
0416: 0C 02 FF
0419: 14 01 FF
041C: FF            - marks end of animation sequence

041D: 0C 02 01
0420: 10 01 01
0423: 0C 02 01
0426: 14 01 01
0429: FF            - marks end of animation sequence


042A: 00 FE 01 04 FF 01 00 FE 01 08 FF
0435: 01 FF 06 0B 04 3B 00 00 AA A0 00 00 00 0A AA AA
```

```
0445: 00 00 0A 0F FA FF 0A 00 AA 0A A0 AA 0A A0 00 A0
0455: AA A0 A0 00 00 0A 0A 0A 00 00 00 00 A0 A0 00 00
0465: 00 00 0A 00 00 00 00 00 A0 A0 00 00 00 AA 00 0A
0475: A0 00 00 0A 00 0A 00 00 02 02 04 81 00 00 00 00
0485: 06 05 04 99 06 05 04 B7 06 05 04 D5 06 05 04 F3
0495: 06 05 05 11 0F F0 FF FF FF F0 00 F0 F0 F0 F0 F0
04A5: 00 F0 F0 F0 F0 F0 00 F0 F0 F0 F0 F0 0F FF FF FF
04B5: FF F0 0F FF BB BB BB B0 00 0F B0 B0 B0 B0 0F FF
04C5: B0 B0 B0 B0 0F 00 B0 B0 B0 B0 0F FF BB BB BB B0
04D5: 0F FF EE EE EE E0 00 0F E0 E0 E0 E0 0F FF E0 E0
04E5: E0 E0 00 0F E0 E0 E0 E0 0F FF EE EE EE E0 0F 0F
04F5: BB BB BB B0 0F 0F B0 B0 B0 B0 0F FF B0 B0 B0 B0
0505: 00 0F B0 B0 B0 B0 00 0F BB BB BB B0 0A AA EE EE
0515: EE E0 0A 00 E0 E0 E0 E0 0A AA E0 E0 E0 E0 00 0A
0525: E0 E0 E0 E0 0A AA EE EE EE E0 04 0E 05 5F 04 0E
0535: 05 97 04 0E 05 CF 04 0E 06 07 04 0E 06 3F 04 0E
0545: 06 77 04 0E 06 AF 04 0E 06 E7 04 0E 07 1F 04 0E
0555: 07 57 04 0E 07 8F 04 0E 07 C7 00 55 00 00 00 25
0565: 50 00 00 62 50 00 00 22 55 00 00 03 30 00 00 34
0575: 30 00 00 04 30 00 00 39 30 00 03 33 33 00 00 69
0585: 60 00 00 09 00 00 00 09 00 00 00 33 00 00 00 00
0595: 00 00 00 55 00 00 00 25 50 00 00 62 50 00 00 22
05A5: 55 00 00 03 30 00 00 33 34 00 00 03 34 00 09 33
05B5: 39 00 63 33 33 00 66 90 90 00 09 00 90 00 09 00
05C5: 90 00 33 03 30 00 00 00 00 00 00 55 00 00 00 25
05D5: 50 00 00 62 50 00 00 22 55 00 00 03 30 00 00 43
05E5: 34 00 04 43 34 00 09 33 39 00 03 33 33 60 00 90
05F5: 96 60 09 00 90 00 09 00 90 00 33 03 30 00 00 00
0605: 00 00 00 05 50 00 00 55 20 00 00 52 60 00 05 52
0615: 20 00 00 33 00 00 00 34 30 00 00 34 00 00 00 39
0625: 30 00 03 66 63 00 00 66 60 00 00 09 00 00 00 09
0635: 00 00 00 03 30 00 00 00 00 00 00 05 50 00 00 55
0645: 20 00 00 52 60 00 05 52 20 00 00 33 00 00 04 33
0655: 30 00 04 33 00 00 09 33 39 00 66 63 33 00 66 60
0665: 90 00 00 90 09 00 00 90 09 00 00 33 03 30 00 00
0675: 00 00 00 05 50 00 00 55 20 00 00 52 60 00 05 52
0685: 20 00 00 33 00 00 00 33 40 00 00 33 44 00 09 33
0695: 39 00 03 33 66 60 00 90 66 60 00 90 09 00 00 90
06A5: 09 00 00 33 03 30 00 00 00 00 00 55 50 00 05 22
06B5: 25 00 05 62 65 00 55 22 25 50 33 33 33 30 40 33
06C5: 30 40 40 03 00 40 90 33 30 90 63 33 33 00 60 90
06D5: 90 00 00 90 90 00 00 90 90 00 03 30 33 00 00 00
06E5: 00 00 00 55 50 00 05 22 25 00 05 62 65 00 55 22
06F5: 25 50 33 33 33 30 40 33 30 40 90 03 00 40 60 33
0705: 30 90 63 33 33 00 00 90 90 00 00 90 90 00 00 90
0715: 33 00 00 30 00 00 00 30 00 00 00 55 50 00 05 22
0725: 25 00 05 62 65 00 55 22 25 50 33 33 33 30 40 33
0735: 30 40 40 03 00 90 90 33 30 00 63 33 33 00 60 90
0745: 90 00 00 90 90 00 03 30 90 00 00 00 30 00 00 00
0755: 30 00 00 55 50 00 05 55 55 00 05 55 55 00 55 55
0765: 55 50 33 33 33 30 40 33 30 40 40 03 00 40 90 33
0775: 30 90 03 33 33 60 00 90 90 60 00 90 90 00 00 90
0785: 90 00 03 30 33 00 00 00 00 00 00 55 50 00 05 55
0795: 55 00 05 55 55 00 55 55 55 50 33 33 33 30 40 33
```

```
07A5: 30 40 90 03 00 40 00 33 30 90 03 33 33 60 00 90
07B5: 90 60 00 90 90 00 00 90 33 00 00 90 00 00 00 30
07C5: 00 00 00 55 50 00 05 55 55 00 05 55 55 00 55 55
07D5: 55 50 33 33 33 30 40 33 30 40 40 03 00 90 90 33
07E5: 30 60 03 33 33 60 00 90 90 00 00 90 90 00 03 30
07F5: 90 00 00 00 90 00 00 00 30 00 05 0D 08 2F 05 0D
0805: 08 70 05 0D 08 B1 05 0D 08 F2 05 0D 09 33 05 0D
0815: 09 74 05 0D 09 B5 05 0D 09 F6 05 0D 0A 37 05 0D
0825: 0A 78 05 0D 0A B9 05 0D 0A FA 00 02 55 00 00 00
0835: 09 25 00 00 00 02 25 00 00 00 07 77 00 00 00 07
0845: 77 00 00 00 07 77 00 00 00 07 77 00 00 00 07 27
0855: 00 00 00 88 88 00 00 00 88 88 00 00 00 88 88 00
0865: 00 00 02 20 00 00 00 00 00 00 00 00 02 55 00 00
0875: 00 09 25 00 00 00 02 25 00 00 00 07 77 00 00 00
0885: 07 77 70 00 00 07 77 70 00 00 07 77 70 00 00 27
0895: 77 20 00 00 07 88 88 00 00 70 88 88 00 00 70 88
08A5: 88 00 02 20 22 00 00 00 00 00 00 00 00 02 55 00
08B5: 00 00 09 25 00 00 00 02 25 00 00 00 07 77 00 00
08C5: 00 07 77 70 00 00 07 77 70 00 00 77 77 70 00 00
08D5: 27 77 20 00 88 88 07 00 00 88 88 07 00 00 88 88
08E5: 07 00 00 02 20 22 00 00 00 00 00 00 00 00 55 20
08F5: 00 00 00 52 90 00 00 00 52 20 00 00 00 77 70 00
0905: 00 00 77 70 00 00 00 77 70 00 00 00 77 70 00 00
0915: 00 72 70 00 00 00 87 88 00 00 00 87 88 00 00 00
0925: 87 88 00 00 00 02 20 00 00 00 00 00 00 00 00 55
0935: 20 00 00 00 52 90 00 00 00 52 20 00 00 00 77 70
0945: 00 00 07 77 70 00 00 07 77 70 00 00 07 77 70 00
0955: 00 02 77 72 00 00 00 77 78 88 00 00 70 87 88 00
0965: 00 70 87 88 00 00 22 02 20 00 00 00 00 00 00 00
0975: 55 20 00 00 00 52 90 00 00 00 52 20 00 00 00 77
0985: 70 00 00 07 77 70 00 00 07 77 70 00 00 07 77 77
0995: 00 00 02 77 72 00 00 88 77 70 00 00 88 78 07 00
09A5: 00 88 78 07 00 00 00 22 02 20 00 00 00 00 00 00
09B5: 00 22 20 00 00 00 92 90 00 00 00 22 20 00 00 77
09C5: 81 87 70 00 77 71 77 70 00 70 71 70 70 00 70 71
09D5: 70 70 00 20 77 70 20 00 00 70 70 88 00 00 70 70
09E5: 88 00 00 70 70 88 00 02 20 22 00 00 00 00 00 00
09F5: 00 00 22 20 00 00 00 92 90 00 00 00 22 20 00 00
0A05: 77 81 87 70 00 77 71 77 70 00 70 71 70 70 00 20
0A15: 71 70 70 00 00 77 70 20 00 00 70 70 88 00 00 70
0A25: 70 88 00 00 70 22 88 00 00 20 00 00 00 00 20 00
0A35: 00 00 00 22 20 00 00 00 92 90 00 00 00 22 20 00
0A45: 00 77 81 87 70 00 77 71 77 70 00 70 71 70 70 00
0A55: 70 71 70 20 00 20 77 70 88 00 00 70 70 88 00 00
0A65: 70 70 88 00 02 20 70 00 00 00 00 20 00 00 00 00
0A75: 20 00 00 00 00 55 50 00 00 00 55 50 00 00 00 45
0A85: 40 00 00 77 77 77 70 00 77 77 77 70 00 70 77 70
0A95: 70 00 70 77 70 70 00 20 77 70 20 08 80 70 70 00
0AA5: 08 80 70 70 00 08 80 70 70 00 00 02 20 22 00 00
0AB5: 00 00 00 00 00 00 55 50 00 00 00 55 50 00 00 00
0AC5: 45 40 00 00 77 77 77 70 00 77 77 77 70 00 70 77
0AD5: 70 70 00 20 77 70 70 08 80 77 70 20 08 80 70 70
0AE5: 00 08 80 70 70 00 00 00 70 22 00 00 00 70 00 00
0AF5: 00 00 20 00 00 00 00 55 50 00 00 00 55 50 00 00
```

```
0B05: 00 45 40 00 00 77 77 77 70 00 77 77 77 70 00 70
0B15: 77 70 70 00 70 77 70 20 00 20 77 70 00 08 80 70
0B25: 70 00 08 80 70 70 00 08 82 20 70 00 00 00 00 70
0B35: 00 00 00 00 20 00 03 0B 0B 6B 03 0B 0B 8C 03 0B
0B45: 0B AD 03 0B 0B CE 03 0B 0B EF 03 0B 0C 10 03 0B
0B55: 0C 31 03 0B 0C 52 03 0B 0C 73 03 0B 0C 94 03 0B
0B65: 0C B5 03 0B 0C D6 02 22 00 09 22 00 02 22 00 00
0B75: 20 00 01 11 00 01 91 00 01 91 00 00 10 00 00 10
0B85: 00 09 90 00 00 00 00 02 22 00 09 22 00 02 22 00
0B95: 00 20 00 01 11 00 09 11 00 90 10 90 01 01 00 01
0BA5: 00 10 99 09 90 00 00 00 02 22 00 09 22 00 02 22
0BB5: 00 00 20 00 01 11 00 01 11 90 90 10 90 01 01 00
0BC5: 01 00 10 99 09 90 00 00 00 02 22 00 02 29 00 02
0BD5: 22 00 00 20 00 01 11 00 01 91 00 01 91 00 00 10
0BE5: 00 00 10 00 00 99 00 00 00 00 02 22 00 02 29 00
0BF5: 02 22 00 00 20 00 01 11 00 91 11 00 90 10 90 01
0C05: 01 00 10 01 00 99 09 90 00 00 00 02 22 00 02 29
0C15: 00 02 22 00 00 20 00 01 11 00 01 19 00 90 10 90
0C25: 01 01 00 10 01 00 99 09 90 00 00 00 02 22 00 09
0C35: 29 00 02 92 00 00 20 00 11 11 10 91 11 90 91 11
0C45: 90 01 01 00 01 01 00 99 09 90 00 00 00 02 22 00
0C55: 09 29 00 02 92 00 00 20 00 91 11 10 91 11 90 01
0C65: 11 90 01 01 00 01 09 90 01 00 00 99 00 00 02 22
0C75: 00 09 29 00 02 92 00 00 20 00 11 11 90 91 11 90
0C85: 91 11 00 01 01 00 99 01 00 00 01 00 00 09 90 02
0C95: 22 00 02 22 00 02 22 00 00 20 00 11 11 10 91 11
0CA5: 90 91 11 90 01 01 00 01 01 00 99 09 90 00 00 00
0CB5: 02 22 00 02 22 00 02 22 00 00 20 00 91 11 10 91
0CC5: 11 90 01 11 90 01 01 00 01 09 90 01 00 00 99 00
0CD5: 00 02 22 00 02 22 00 02 22 00 00 20 00 11 11 90
0CE5: 91 11 90 91 11 00 01 01 00 99 01 00 00 01 00 00
0CF5: 09 90 07 10 07 10 0D 1D 07 10 0D 8D 07 10 0D FD
0D05: 07 10 0F BD 07 10 10 2D 07 10 10 9D 07 10 0E 6D
0D15: 07 10 0E DD 07 10 0F 4D 00 00 0B BB 00 00 00 00
0D25: 00 00 10 00 00 00 00 00 00 10 00 00 00 00 06 66
0D35: AA 66 00 00 00 06 66 AA 66 00 00 00 06 66 AA 66
0D45: 00 00 00 06 66 AA 66 00 00 00 06 66 AA 66 00 00
0D55: 00 06 66 AA 66 00 00 00 06 6A AA 66 00 00 00 00
0D65: 00 11 00 00 00 00 00 00 11 00 00 00 00 00 00 11
0D75: 00 00 00 00 00 11 11 00 00 00 00 00 00 00 00 00
0D85: 00 00 00 00 00 00 00 00 00 00 0B BB 00 00 00 00
0D95: 00 00 10 00 00 00 00 00 00 10 00 00 00 00 06 66
0DA5: AA 66 00 00 00 06 66 AA A6 00 00 00 06 66 6A AA
0DB5: 00 00 00 06 66 66 AA A0 00 00 A6 66 66 6A A0 00
0DC5: 0A A6 66 66 A6 00 00 00 06 66 66 66 00 00 00 00
0DD5: 11 01 10 00 00 10 01 10 00 11 10 00 01 11 00 00
0DE5: 00 11 00 00 10 00 00 11 10 00 00 00 00 00 00 00
0DF5: 00 00 00 00 00 00 00 00 00 00 0B BB 00 00 00 00
0E05: 00 00 10 00 00 00 00 00 00 10 00 00 00 00 06 66
0E15: AA 66 00 00 00 06 6A AA 66 00 00 00 06 AA A6 66
0E25: 00 00 00 0A AA 66 66 00 00 A0 AA A6 66 66 A0 00
0E35: 0A A6 66 66 66 AA 00 00 06 66 66 66 00 00 00 00
0E45: 11 01 10 00 00 10 01 10 00 11 10 00 01 11 00 00
0E55: 00 11 00 00 10 00 00 11 10 00 00 00 00 00 00 00
```

```
0E65: 00 00 00 00 00 00 00 00 00 00 0B BB 00 00 00 00
0E75: 00 00 10 00 00 00 00 00 00 10 00 00 00 AA A6 66
0E85: 66 66 AA A0 AA A6 66 66 66 AA A0 AA 06 66 66 66
0E95: 0A A0 AA 06 66 66 66 0A A0 AA 06 66 66 66 0A A0
0EA5: AA A6 66 66 66 AA A0 A0 A6 66 66 66 A0 A0 A0 A0
0EB5: 11 01 10 A0 A0 00 00 11 01 10 00 00 00 00 11 01
0EC5: 10 00 00 00 11 11 01 11 10 00 00 00 00 00 00 00
0ED5: 00 00 00 00 00 00 00 00 00 00 0B BB 00 00 00 00
0EE5: 00 00 10 00 00 00 00 00 00 10 00 00 00 AA A6 66
0EF5: 66 66 AA A0 AA A6 66 66 66 AA A0 AA 06 66 66 66
0F05: 0A A0 AA 06 66 66 66 0A A0 AA 06 66 66 66 0A A0
0F15: AA A6 66 66 66 AA A0 A0 A6 66 66 66 A0 A0 A0 A0
0F25: 11 01 10 A0 A0 00 11 11 01 10 00 00 00 00 00 01
0F35: 10 00 00 00 00 00 01 10 00 00 00 00 00 01 10 00
0F45: 00 00 00 00 01 11 10 00 00 00 0B BB 00 00 00 00
0F55: 00 00 10 00 00 00 00 00 00 10 00 00 00 AA A6 66
0F65: 66 66 AA A0 AA A6 66 66 66 AA A0 AA 06 66 66 66
0F75: 0A A0 AA 06 66 66 66 0A A0 AA 06 66 66 66 0A A0
0F85: AA A6 66 66 66 AA A0 A0 A6 66 66 66 A0 A0 A0 A0
0F95: 11 01 10 A0 A0 00 00 11 01 11 10 00 00 00 11 00
0FA5: 00 00 00 00 00 11 00 00 00 00 00 00 11 00 00 00
0FB5: 00 00 11 11 00 00 00 00 00 00 BB B0 00 00 00 00
0FC5: 00 01 00 00 00 00 00 00 01 00 00 00 00 00 66 AA
0FD5: 66 60 00 00 00 66 AA 66 60 00 00 00 66 AA 66 60
0FE5: 00 00 00 66 AA 66 60 00 00 00 66 AA 66 60 00 00
0FF5: 00 66 AA 66 60 00 00 00 66 AA A6 60 00 00 00 00
1005: 11 00 00 00 00 00 00 11 00 00 00 00 00 00 11 00
1015: 00 00 00 00 00 11 11 00 00 00 00 00 00 00 00 00
1025: 00 00 00 00 00 00 00 00 00 00 BB B0 00 00 00 00
1035: 00 01 00 00 00 00 00 00 01 00 00 00 00 00 66 AA
1045: 66 60 00 00 00 6A AA 66 60 00 00 00 AA A6 66 60
1055: 00 00 0A AA 66 66 60 00 00 0A A6 66 66 6A 00 00
1065: 00 6A 66 66 6A A0 00 00 66 66 66 60 00 00 00 01
1075: 10 11 00 00 00 01 11 00 01 10 01 00 11 00 00 00
1085: 11 10 00 01 11 00 00 01 00 00 00 00 00 00 00 00
1095: 00 00 00 00 00 00 00 00 00 00 BB B0 00 00 00 00
10A5: 00 01 00 00 00 00 00 00 01 00 00 00 00 00 66 AA
10B5: 66 60 00 00 00 66 AA A6 60 00 00 00 66 6A AA 60
10C5: 00 00 00 66 66 AA A0 A0 00 0A 66 66 6A AA 00 00
10D5: AA 66 66 66 60 00 00 00 66 66 66 60 00 00 00 01
10E5: 10 11 00 00 00 01 11 00 01 10 01 00 11 00 00 00
10F5: 11 10 00 01 11 00 00 01 00 00 00 00 00 00 00 00
1105: 00 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF
1115: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1125: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1135: FF FF FF FF FF FF FF FF

113D: FF FF FF    STU   $FFFF
1140: 7E 11 68    JMP   $1168

1143: 7E 12 FA    JMP   $12FA

1146: 14 F2 18 D2 18 E6 D8 01 08 18 00 D1 01 08 08 00
1156: D0 01 08 1D 00 D0 03 04 17 00 D0 01 04 15 01 08
```

```
1166: 14 00


INITIALISE_ALL_SPHEROIDS:
1168: B6 BE 6F    LDA   cur_sphereoids       ; get number of
sphereoids into A
116B: 34 02       PSHS  A                    ; save spheroid count
on stack
116D: 27 3E       BEQ   $11AD                ; if spheroid count
==0, goto $11AD
117D: 86 09       LDA   #$09                 ; set X coordinate of
spheroid to be #$09 - tentatively; depends on next 2 lines of code
117F: 0D 84       TST   $84                  ; read a random number
1181: 2B 02       BMI   $1185                ; if bit 7 set, goto
$1185
1183: 86 87       LDA   #$87                 ; set X coordinate of
spheroid to be #$87
1185: ED 04       STD   $0004,X              ; set sphereoid current
blit destination
1187: A7 0A       STA   $000A,X              ; set sphereoid X
coordinate
1189: E7 0C       STB   $000C,X              ; set sphereoid Y
coordinate
118B: B6 BE 60    LDA   $BE60                ; read enforcer drop
countdown setting
118E: BD D0 3F    JSR   $D03F                ; JMP $D6B6 - get a
random number lower than or equal to A
1191: A7 49       STA   $0009,U              ; set countdown before
creating first enforcer
1193: B6 BE 5E    LDA   $BE5E                ; read enforcer drop
count variable
1196: BD D0 3F    JSR   $D03F                ; JMP $D6B6 - get a
random number lower than or equal to A
1199: 44          LSRA                       ; divide number by 2,
move bit 0 into carry
119A: 89 00       ADCA  #$00                 ; ensure that A is non
zero
119C: A7 4A       STA   $000A,U              ; set count of
enforcers to drop
119E: BD 12 5F    JSR   $125F                ; pick initial
direction
11A1: CC 15 02    LDD   #$1502               ; pointer to collision
detection animation frame metadata for spheroid (see $D7F4)
11A4: ED 88 16    STD   $16,X
11A7: 9F 17       STX   $17                  ; store pointer to
spheroid object in linked list
11A9: 6A E4       DEC   ,S                   ; decrement spheroid
count on stack
11AB: 26 C2       BNE   $116F
11AD: 35 82       PULS  A,PC ;(PUL? PC=RTS)


ANIMATE_SPHEROID:
```

```
11AF: AE 47       LDX   $0007,U             ; get pointer to object
11B1: EC 02       LDD   $0002,X             ; get current animation
frame metadata pointer
11B3: C3 00 04    ADDD  #$0004              ; bump to next image's
metadata pointer
11B6: 10 83 15 02 CMPD  #$1502              ; end of animation
frame metadata list?
11BA: 23 0B       BLS   $11C7
11BC: CC 14 F2    LDD   #$14F2              ; back to first
spheroid image
11BF: 0D 59       TST   $59
11C1: 26 04       BNE   $11C7
11C3: 6A 49       DEC   $0009,U             ; decrement enforcer
drop countdown
11C5: 27 14       BEQ   $11DB               ; if countdown == 0,
then time to drop an enforcer, goto $11DB
11C7: ED 02       STD   $0002,X             ; set current animation
frame metadata pointer
11C9: 6A 4E       DEC   $000E,U             ; decrement move
countdown before enforcer decides where to move to next
11CB: 26 03       BNE   $11D0               ; if countdown !=0,
enforcer continues on its current path, goto $11D0
; sphereoid wants to go somewhere else
11CD: BD 12 5F    JSR   $125F               ; set spheroid movement
curvature factors and move countdown
11D0: BD 12 79    JSR   $1279               ; update spheroid X and
Y movement deltas
11D3: 86 02       LDA   #$02                ; delay before calling
function
11D5: 8E 11 AF    LDX   #$11AF              ; address of routine to
jump to next time for this object
11D8: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call


DROP_ENFORCER:
11DB: B6 BE 60    LDA   $BE60               ; read enforcer drop
delay setting
11DE: 44          LSRA                      ; this time, divide it
by 4 - making this spheroid drop its next enforcer much faster than
it did first time
11DF: 44          LSRA
11E0: BD D0 3F    JSR   $D03F               ; JMP $D6B6 - get a
random number lower than or equal to A
11E3: A7 49       STA   $0009,U             ; set countdown before
dropping enforcer
11E5: AE 47       LDX   $0007,U             ; load X with pointer
to object
11E7: EC 02       LDD   $0002,X             ; load D with pointer
to animation frame metadata
11E9: C3 00 04    ADDD  #$0004              ; bump to next
animation frame metadata
11EC: 10 83 15 0E CMPD  #$150E              ; at end of spheroid
animation (last animation frame in list)?
```

```
11F0: 23 1C       BLS   $120E               ; no
11F2: 6A 49       DEC   $0009,U             ; decrement countdown
before dropping enforcer
11F4: 26 15       BNE   $120B               ; countdown !=0, not
time to drop enforcer, so goto $120B

; at this point, the spheroid wants to drop an enforcer. But there
can only be 8 enforcers maximum on
; screen at once.
11F6: 96 ED       LDA   temp_enforcer_count
11F8: 81 08       CMPA  #$08                ; 8 enforcers at the
moment?
11FA: 24 DF       BCC   $11DB               ; if >= 8, goto $11DB
11FC: 96 42       LDA   $42
11FE: 81 11       CMPA  #$11
1200: 24 D9       BCC   $11DB
1202: BD 13 54    JSR   $1354               ; create an enforcer
1205: 6A 4A       DEC   $000A,U             ; decrement enforcers
dropped counter
1207: 27 19       BEQ   $1222               ; if counter == 0 goto
$1222. Time for the spheroid to disappear!!!
1209: 20 D0       BRA   $11DB

120B: CC 14 F2    LDD   #$14F2              ; start of spheroid
animation frame metadata

120E: ED 02       STD   $0002,X             ; save pointer to
current animation frame metadata
1210: 6A 4E       DEC   $000E,U             ; decrement spheroid
"countdown before I change direction" counter.
1212: 26 03       BNE   $1217               ; if counter!=0, it's
not time to change direction, goto $1217.
1214: BD 12 5F    JSR   $125F               ; change spheroid
direction and set countdown before changing direction.
1217: BD 12 79    JSR   $1279               ; update spheroid
movement deltas
121A: 86 02       LDA   #$02                ; delay before calling
function
121C: 8E 11 E5    LDX   #$11E5              ; address of function
to call next
121F: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call


;
; The spheroid has dropped all of its enforcers and needs to go.
;
TIME_FOR_SPHEROID_TO_EXIT_PRONTO:
1222: CC 00 00    LDD   #$0000              ; set Y delta (do not
move vertically)
1225: ED 88 10    STD   $10,X               ; store Y delta
1228: CC 01 00    LDD   #$0100              ; X delta - move right
122B: 0D 84       TST   $84                 ; is bit 7 of the
random number variable set?
```

```
122D: 2A 01       BPL   $1230               ; if bit 7 not set,
goto $1230
122F: 40          NEGA                      ; flip X delta to move
left
1230: ED 0E       STD   $000E,X             ; set X delta
1232: AE 47       LDX   $0007,U             ; load X with pointer
to object
1234: EC 02       LDD   $0002,X             ; load D with pointer
to animation frame metadata
1236: C3 00 04    ADDD  #$0004              ; bump to next
animation frame metadata
1239: 10 83 15 02 CMPD  #$1502              ; have we hit the end
of the animation frame metadata list?
123D: 23 0D       BLS   $124C               ; no, set pointer to
current animation frame metadata
123F: A6 0A       LDA   $000A,X             ; get X coordinate
(whole part)
1241: 81 0A       CMPA  #$0A                ; compare to #$0A (10
decimal)
1243: 23 11       BLS   $1256               ; <= , spheroid is at
leftmost edge of screen and needs to go
1245: 81 85       CMPA  #$85                ; compare to #$85 (133
decimal)
1247: 24 0D       BCC   $1256               ; >=, spheroid is at
rightmost edge of screen and needs to go
1249: CC 14 F2    LDD   #$14F2              ; start of spheroid
animation frame metadata list (to begin animation from start)
124C: ED 02       STD   $0002,X             ; set current animation
frame metadata pointer
124E: 86 02       LDA   #$02                ; delay before calling
function
1250: 8E 12 32    LDX   #$1232              ; address of function
to call for this object next
1253: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

1256: BD D0 75    JSR   $D075               ; JMP $D31B -
deallocate object and erase object from screen
1259: 7A BE 6F    DEC   cur_sphereoids      ; reduce spheroid count
125C: 7E D0 63    JMP   $D063               ; JMP $D1F3


; OK, I realise its a long label, but you gotta be descriptive ;)
; See $DCFF for details on how the spheroid glides so smoothly
SET_SPHEROID_CURVATURE_FACTORS_AND_COUNTDOWN_BEFORE_CHANGING_DIRECTI
ON:
125F: 96 85       LDA   $85                 ; get a random number
into A
1261: 84 1F       ANDA  #$1F                ; mask in bits 0..5
1263: 8B F0       ADDA  #$F0  (-15
decimal)
1265: A7 4C       STA   $000C,U             ; set X curvature
factor
1267: 96 86       LDA   $86                 ; get another random
```

```
number into A
1269: 98 84       EORA  $84                 ; xor with another
random number...
126B: 84 3F       ANDA  #$3F
126D: 8B E0       ADDA  #$E0
126F: A7 4D       STA   $000D,U             ; set Y curvature
factor
1271: 86 0F       LDA   #$0F
1273: BD D0 42    JSR   $D042               ; JMP $D6AC - multiply
A by a random number and put result in A
1276: A7 4E       STA   $000E,U             ; set countdown before
spheroid changes direction
1278: 39          RTS


UPDATE_SPHEROID_MOVEMENT_DELTAS:
1279: E6 4C       LDB   $000C,U             ; B = Y curvature
factor
127B: 1D          SEX                       ; sign extend B into A
(so D = sign extended version of B)
127C: E3 0E       ADDD  $000E,X             ; add X delta to D
127E: 10 83 01 00 CMPD  #$0100              ; is Y delta facing
down?
1282: 2D 03       BLT   $1287               ; <
1284: CC 01 00    LDD   #$0100
1287: 10 83 FF 00 CMPD  #$FF00
128B: 2E 03       BGT   $1290
128D: CC FF 00    LDD   #$FF00
1290: ED 0E       STD   $000E,X             ; set X delta
1292: 43          COMA
1293: 53          COMB
1294: 58          ASLB
1295: 49          ROLA
1296: 58          ASLB
1297: 49          ROLA
1298: 1F 89       TFR   A,B
129A: 1D          SEX
129B: E3 0E       ADDD  $000E,X             ; add X delta to D
129D: ED 0E       STD   $000E,X             ; set X delta
129F: E6 4D       LDB   $000D,U
12A1: 1D          SEX
12A2: E3 88 10    ADDD  $10,X
12A5: 10 83 02 00 CMPD  #$0200
12A9: 2D 03       BLT   $12AE
12AB: CC 02 00    LDD   #$0200
12AE: 10 83 FE 00 CMPD  #$FE00
12B2: 2E 03       BGT   $12B7
12B4: CC FE 00    LDD   #$FE00
12B7: ED 88 10    STD   $10,X              ; set Y delta
12BA: 43          COMA
12BB: 53          COMB
12BC: 58          ASLB
12BD: 49          ROLA
12BE: 1F 89       TFR   A,B
```

```
12C0: 1D          SEX
12C1: E3 88 10    ADDD  $10,X               ; D+= Y delta
12C4: ED 88 10    STD   $10,X               ; set Y delta to D
12C7: 39          RTS



SPHEROID_COLLISION_HANDLER:
12C8: 96 48       LDA   $48                 ; player collision
detection called this?
12CA: 26 24       BNE   $12F0               ; yes
12CC: BD D0 78    JSR   $D078               ; JMP $D320 - deallocate
object and erase object from screen (2)
12CF: DE 1B       LDU   $1B
12D1: EC C4       LDD   ,U
12D3: DD 1B       STD   $1B
; spheroid is dead
12D5: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
12D8: 12 F1          ; pointer to function

12DA: EF 07       STU   $0007,X              ; set object metadata
pointer in this object
12DC: CC 14 F6    LDD   #$14F6               ; animation frame
metadata pointer
12DF: ED 42       STD   $0002,U
12E1: 7A BE 6F    DEC   cur_sphereoids
12E4: CC 02 10    LDD   #$0210
12E7: BD D0 0C    JSR   $D00C                ; JMP $DB9C - update
player score
12EA: CC 11 51    LDD   #$1151
12ED: 7E D0 4B    JMP   $D04B                ; JMP $D3C7

12F0: 39          RTS



;
; When you shoot a spheroid, it's drawn a different colour (the same
colour as the player score, to be precise)
; for a short while, to signify it's dying. Then it disappears and
the points value of the spheroid appears in its place.
;

DRAW_SPHEROID_IN_DEATH_THROES:
12F1: CC FF AA    LDD   #$FFAA               ; A = colour to draw
points value of spheroid in, B = colour to draw dying spheroid in
12F4: ED 4B       STD   $000B,U              ; set colour remap
values in object metadata
12F6: 86 07       LDA   #$07                 ; countdown before
spheroid disappears and spheroid points value appears in its place
12F8: A7 4D       STA   $000D,U
12FA: AE 47       LDX   $0007,U              ; X = pointer to object
from object metadata
```

```
12FC: 10 AE 02    LDY   $0002,X             ; Y = pointer to
animation frame metadata
12FF: EC 04       LDD   $0004,X             ; D= blitter
destination
1301: BD D0 1E    JSR   $D01E               ; JMP $DABF - clear
image rectangle
1304: 31 24       LEAY  $0004,Y             ; Set Y to pointer to
animation frame metadata
1306: 10 AF 02    STY   $0002,X
1309: 6A 4D       DEC   $000D,U             ; decrement countdown
before score value is shown
130B: 27 11       BEQ   $131E               ; if countdown is 0,
time to show the points value of the sphereoid, goto $131E
130D: A6 4C       LDA   $000C,U             ; otherwise, draw the
spheroid in the colour specified at $12F1
130F: 97 2D       STA   $2D                 ; set colour remap
value
1311: EC 04       LDD   $0004,X             ; D = blitter
destination
1313: BD D0 90    JSR   $D090               ; JMP $DA9E - do solid
and transparent blit
1316: 86 02       LDA   #$02                ; delay before calling
function
1318: 8E 12 FA    LDX   #$12FA              ; address of function
to call
131B: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call


;
; The spheroid is dead, now show its points value
;
;

DRAW_SPHEROID_POINTS_VALUE:
131E: EC 04       LDD   $0004,X             ; D= blitter
destination of spheroid
1320: C3 01 05    ADDD  #$0105              ; adjust position for
score image
1323: ED 04       STD   $0004,X             ; set blitter
destination
1325: 86 1E       LDA   #$1E
1327: A7 49       STA   $0009,U             ; set countdown of how
long score is displayed
1329: FC 00 0C    LDD   $000C               ; read points value
animation frame metadata pointer
132C: ED 02       STD   $0002,X             ; set pointer to
animation frame metadata
132E: AE 47       LDX   $0007,U             ; get pointer to object
from object metadata
1330: A6 4B       LDA   $000B,U
1332: 97 2D       STA   $2D                 ; set solid colour
1334: 10 AE 02    LDY   $0002,X             ; set pointer to
animation frame metadata
```

```
1337: EC 04       LDD   $0004,X             ; set D = blitter
destination
1339: BD D0 90    JSR   $D090               ; JMP $DA9E - do solid
and transparent blit, to draw points value of spheroid
133C: 6A 49       DEC   $0009,U             ; decrement countdown
of score display
133E: 27 08       BEQ   $1348               ; if 0, the score has
been displayed long enough, goto $1348
1340: 86 02       LDA   #$02                ; delay before calling
function
1342: 8E 13 2E    LDX   #$132E              ; address of function
to call
1345: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

; clear the score off screen
1348: BD D0 1E    JSR   $D01E               ; JMP $DABF: clear
image rectangle
134B: DC 1B       LDD   $1B                 ; read "free object
entry" pointer
134D: ED 84       STD   ,X                  ; store in this object
(creating a forward only linked list that includes this object)
134F: 9F 1B       STX   $1B                 ; set "free object
entry" linked list to begin with this object
1351: 7E D0 63    JMP   $D063               ; JMP $D1F3



;
; X = pointer to spheroid that is spawning the enforcer
;
;
;
;

CREATE_ENFORCER:
1354: 34 76       PSHS  U,Y,X,B,A
1356: 1F 12       TFR   X,Y                 ; Spheroid pointer -> Y
1358: BD D0 6C    JSR   $D06C               ; JMP $D32B - create
entity with params and add to linked list at $9817

; parameters to pass to $D06C
135B: 13 90       ; address of function to call after 1 game cycle
135D: 18 D2       ; animation frame metadata pointer
135E: 14 83       ; address of routine to handle collision

1361: 27 1C       BEQ   $137F               ; if Z flag set then
enforcer could not be created, goto $137F
; X = pointer to freshly created object
1363: A6 2A       LDA   $000A,Y             ; get X coordinate from
Spheroid
1365: E6 2C       LDB   $000C,Y             ; get Ycoordinate from
Spheroid
1367: A7 0A       STA   $000A,X             ; set X coordinate
```

```
(whole part) of Enforcer
1369: E7 0C       STB   $000C,X             ; set Y coordinate of
Enforcer
136B: ED 04       STD   $0004,X             ; set blitter
destination
136D: B6 BE 5F    LDA   $BE5F               ; read enforcer control
variable
1370: BD D0 3F    JSR   $D03F               ; JMP $D6B6 - get a
random number lower than or equal to A
1373: A7 4D       STA   $000D,U             ; set delay on how long
it is before enforcer fires a spark
1375: 9F 17       STX   $17
1377: CC 11 4C    LDD   #$114C
137A: BD D0 4B    JSR   $D04B               ; JMP $D3C7
137D: 0C ED       INC   temp_enforcer_count
137F: 35 F6       PULS  A,B,X,Y,U,PC ;(PUL? PC=RTS)


;
; The enforcer is spawning here. This function is called to advance
the spawn one frame at a time.
;

ENFORCER_SPAWN_ANIMATION:
1381: AE 47       LDX   $0007,U             ; get pointer to
enforcer object from object metadata
1383: EC 02       LDD   $0002,X             ; get animation frame
metadata pointer
1385: C3 00 04    ADDD  #$0004              ; bump to next
animation frame metadata in the list
1388: 10 83 18 E6 CMPD  #$18E6              ; last animation frame
of spawn?
138C: 24 0A       BCC   $1398               ; if >= last frame then
goto $1398 - the enforcer's ready to attack!
138E: ED 02       STD   $0002,X             ; set animation frame
metadata pointer
1390: 86 08       LDA   #$08                ; set delay before next
call to this routine, to advance spawn
1392: 8E 13 81    LDX   #$1381              ; address of function
to call
1395: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

; when we get here, the enforcer has spawned and is ready to attack!
ENFORCER_SPAWN_COMPLETE:
1398: ED 02       STD   $0002,X             ; set animation frame
metadata pointer
139A: BD 13 B5    JSR   $13B5               ; pick the location
that the enforcer wants to move to


;
; This routine is the enforcer AI. I haven't called this
ANIMATE_ENFORCER because the enforcer has no animations!
```

```
;

ENFORCER_AI:
139D: AE 47       LDX   $0007,U             ; get pointer to object
into X
139F: 6A 4E       DEC   $000E,U             ; decrement countdown
before enforcer selects a new location to move to
13A1: 26 03       BNE   $13A6               ; if countdown !=0,
keep the enforcer going in its current direction, goto $13A6
13A3: BD 13 B5    JSR   $13B5               ; otherwise, the
enforcer must evaluate where it wants to move to next
13A6: 6A 4D       DEC   $000D,U             ; decrement enforcer
spark countdown. When countdown == 0, the enforcer will fire a
spark.
13A8: 26 03       BNE   $13AD               ; if countdown !=0,
goto $13AD
13AA: BD 14 04    JSR   $1404               ; counter is ==0, call
function to fire a spark.
13AD: 86 03       LDA   #$03                ; delay before calling
this enforcer AI function again
13AF: 8E 13 9D    LDX   #$139D              ; address of enforcer
AI function
13B2: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call


;
; Pick a destination for the enforcer to move to.
;
; See $DCFF for information on how the enforcer glides so smoothly.
;

PICK_ENFORCER_DESTINATION:
13B5: BD D0 3C    JSR   $D03C               ; JMP $D6C8 - get
random numbers into A and B
13B8: 84 1F       ANDA  #$1F                ; make A a number
between 0..31 decimal
13BA: A7 4E       STA   $000E,U             ; countdown before
enforcer selects new location to move to
13BC: 4F          CLRA                      ; A = 0
13BD: 57          ASRB                      ; move bit 0 of B into
carry while preserving bit 7 (bit 7 never moves)
13BE: 57          ASRB
13BF: 57          ASRB
13C0: DB 5E       ADDB  $5E                 ; B+= most significant
byte of player blitter destination (the horizontal component)
; at this point B is a Y coordinate that the enforcer is considering
moving to.
13C2: C1 07       CMPB  #$07                ; at top-most of
playfield?
13C4: 24 02       BCC   $13C8               ; no, goto $13C8
13C6: C6 07       LDB   #$07
13C8: 81 8F       CMPA  #$8F                ; why is A being read
when it's zero (see $13BC) - don't they mean B? Could this be the
```

```
enforcer bug?
13CA: 23 0A       BLS   $13D6               ; as a result of A
always being 0, this code always executes

; I've put a breakpoint on this block of code in MAME debugger and
it's never been called.
13CC: C1 CF       CMPB  #$CF
13CE: 24 04       BCC   $13D4
13D0: C6 8F       LDB   #$8F
13D2: 20 02       BRA   $13D6
13D4: C6 07       LDB   #$07
; end of code that is never called

13D6: E0 0A       SUBB  $000A,X             ; B-= X coordinate
(whole part) of enforcer
13D8: 82 00       SBCA  #$00
13DA: 58          ASLB                      ; move bit 7 of B into
carry
13DB: 49          ROLA
;
; if you want to see how the deltas affect the horizontal movement
of the enforcer, or indeed any omnidirectional object, put a
breakpoint on the line below,
; and just before executing the instruction, set register D to 0,
$FF00 or $0100 in the MAME debugger.
; $FF00 = move to left, $0100 = move to right.If you set D to 0,
there will no movement on the horizontal axis at all.
; Tinker with the least significant byte (the fractional part) to
have the enforcer move in oblique angles.
;
13DC: ED 0E       STD   $000E,X             ; set X delta of
enforcer
13DE: 4F          CLRA
13DF: D6 85       LDB   $85                 ; get a random number
into B
13E1: 57          ASRB
13E2: 57          ASRB
13E3: 57          ASRB
13E4: DB 5F       ADDB  $5F                 ; B+= least significant
byte of player blitter destination (the vertical component)
13E6: C1 EA       CMPB  #$EA                ; at very bottom of
playfield?
13E8: 23 02       BLS   $13EC               ; if <= bottom of
playfield goto $13EC
13EA: C6 EA       LDB   #$EA
13EC: C1 18       CMPB  #$18                ; > top of playfield?
13EE: 24 0A       BCC   $13FA               ; yes
13F0: C1 0C       CMPB  #$0C
13F2: 24 04       BCC   $13F8
13F4: C6 EA       LDB   #$EA
13F6: 20 02       BRA   $13FA

13F8: C6 18       LDB   #$18
13FA: E0 0C       SUBB  $000C,X              ; B-= Y coordinate of
```

```
enforcer
13FC: 82 00       SBCA  #$00
13FE: 58          ASLB
13FF: 49          ROLA
;
; if you want to see how the deltas affect the vertical movement of
the enforcer, or indeed any omnidirectional object, put a breakpoint
on the line below,
; and just before executing the instruction, set register D to 0,
$FF00 or $0100 in the MAME debugger.
; $FF00 = move up, $0100 = move down.If you set D to 0, there will
no movement on the vertical axis at all.
; Tinker with the least significant byte (the fractional part) to
have the enforcer move in oblique angles.
;
1400: ED 88 10    STD   $10,X                ; set Y delta of
enforcer
1403: 39          RTS

;
; Enforcer missiles are called "sparks".  As you can see from this
function, there's a maximum of 20 on screen at once.
;

CREATE_SPARK:
1404: 34 50       PSHS  U,X
1406: 1F 12       TFR   X,Y
1408: B6 BE 5F    LDA   $BE5F                ; read enforcer spark
control variable
140B: BD D0 3F    JSR   $D03F                ; JMP $D6B6 - get a
random number lower than or equal to A
140E: A7 4D       STA   $000D,U              ; store in "countdown
before fire next spark" field
1410: 96 8A       LDA   $8A                  ; get count of sparks
on screen
1412: 81 14       CMPA  #$14                 ; compare to #$14 (20
decimal)
1414: 24 6B       BCC   $1481                ; if count >= 20
decimal, goto 1481. Can't create any more sparks
1416: 96 42       LDA   $42
1418: 81 11       CMPA  #$11
141A: 24 65       BCC   $1481
141C: BD D0 6C    JSR   $D06C                ; JMP $D32B - create
entity with params and add to linked list at $9817
141F: 14 A8       ; address of function to call after 1 game cycle
1421: 1A 34       ; animation frame metadata pointer to spark first
frame
1423: 14 DC       ; address of routine to handle collision
1425: 27 5A       BEQ   $1481
1427: 0C 8A       INC   $8A                  ; increment number of
sparks on screen
1429: EC 24       LDD   $0004,Y
142B: A7 0A       STA   $000A,X              ; store A in X
coordinate (whole part) of spark
```

```
142D: E7 0C       STB   $000C,X             ; store B in Y
coordinate of spark
142F: ED 04       STD   $0004,X             ; store D in blitter
destination of spark
;
; After doing a bit of toying about with the A and B registers, I've
deduced this part of the CREATE_SPARK function is to calculate the
; spark's initial angle, speed and longevity.
;
1431: D6 84       LDB   $84                 ; get a random number
into B
1433: C4 1F       ANDB  #$1F                ; and with #$1F
(preserve bits 0..5) giving a number from 0..31 decimal
1435: CB F0       ADDB  #$F0                ; B+= #$F0   (240
decimal) - or you could look at it this way, B -= #$0F
1437: 96 5E       LDA   $5E                 ; get player's blitter
destination hi byte
1439: 81 17       CMPA  #$17
143B: 24 01       BCC   $143E               ; if >= #$17, goto 143E
143D: 5F          CLRB
143E: DB 5E       ADDB  $5E                 ; add player's blitter
destination hi byte
1440: 4F          CLRA
1441: E0 04       SUBB  $0004,X             ; B-= spark's blitter
destination hi byte
1443: 82 00       SBCA  #$00                ; if the subtraction
caused a carry, make A = $FF, else A=0
1445: 58          ASLB
1446: 49          ROLA
1447: 58          ASLB
1448: 49          ROLA
1449: ED 0E       STD   $000E,X             ; set X delta of object
144B: D6 86       LDB   $86
144D: C4 1F       ANDB  #$1F
144F: CB F0       ADDB  #$F0
1451: DB 5F       ADDB  $5F                 ; add player's blitter
destination lo byte
1453: 4F          CLRA
1454: E0 05       SUBB  $0005,X             ; B-= spark's blitter
destination lo byte
1456: 82 00       SBCA  #$00                ; if the subtraction
caused a carry, make A = $FF, else A=0
1458: 58          ASLB
1459: 49          ROLA
145A: 58          ASLB
145B: 49          ROLA
145C: ED 88 10    STD   $10,X               ; set Y delta of object
145F: D6 86       LDB   $86
1461: C4 1F       ANDB  #$1F
1463: CB F0       ADDB  #$F0
1465: 1D          SEX                       ; make D = sign
extended version of B
1466: ED 49       STD   $0009,U             ; set X curvature
factor
```

```
1468: D6 85       LDB   $85                 ; get a random number
146A: C4 1F       ANDB  #$1F
146C: CB F0       ADDB  #$F0
146E: 1D          SEX                       ; make D = sign
extended version of B
146F: ED 4B       STD   $000B,U             ; set Y curvature
factor
1471: 9F 17       STX   $17
1473: 96 85       LDA   $85                 ; get a "random" number
(probably same value as $1468)
1475: 84 0F       ANDA  #$0F                ; mask in 4 least
significant bits leaving a number 0- #$0F (15 decimal)
1477: 8B 14       ADDA  #$14                ; add #$14 (20 decimal)
to the result
1479: A7 4E       STA   $000E,U             ; set counter for how
long this spark will live
147B: CC 11 56    LDD   #$1156
147E: BD D0 4B    JSR   $D04B               ; JMP $D3C7
1481: 35 D0       PULS  X,U,PC ;(PUL? PC=RTS)


ENFORCER_COLLISION_HANDLER:
1483: 96 48       LDA   $48                 ; called by player
collision detection?
1485: 26 20       BNE   $14A7               ; yes
1487: BD D0 78    JSR   $D078               ; JMP $D320 -
deallocate object and erase object from screen (2)
148A: 9E 1B       LDX   $1B                 ; X = pointer to free
object linked list
148C: EC 84       LDD   ,X
148E: DD 1B       STD   $1B                 ; mark this object as
the first entry in the free object linked list
1490: BD 5B 43    JSR   $5B43               ; JMP $5C1F - create an
explosion
1493: DC 1B       LDD   $1B
1495: ED 84       STD   ,X
1497: 9F 1B       STX   $1B
1499: 0A ED       DEC   temp_enforcer_count
149B: CC 01 15    LDD   #$0115
149E: BD D0 0C    JSR   $D00C               ; JMP $DB9C - update
player score
14A1: CC 11 5B    LDD   #$115B
14A4: 7E D0 4B    JMP   $D04B               ; JMP $D3C7
14A7: 39          RTS



SPARK_ANIMATION:
14A8: AE 47       LDX   $0007,U             ; get pointer to
enforcer object from object metadata
14AA: EC 02       LDD   $0002,X             ; get animation frame
metadata pointer
14AC: C3 00 04    ADDD  #$0004              ; bump 4 to move to
next animation frame metadata
```

```
14AF: 10 83 1A 40 CMPD  #$1A40              ; gone past end of
animation frame metadata list?
14B3: 23 03       BLS   $14B8               ; if not, go to $14B8
14B5: CC 1A 34    LDD   #$1A34              ; pointer to start of
spark animation
14B8: ED 02       STD   $0002,X             ; set animation frame
metadata pointer

;
14BA: EC 0E       LDD   $000E,X             ; read X delta of
object
14BC: E3 49       ADDD  $0009,U             ; take into account
curvature factor of spark
14BE: ED 0E       STD   $000E,X             ; and store back to X
delta
14C0: EC 88 10    LDD   $10,X               ; read Y delta of
object
14C3: E3 4B       ADDD  $000B,U             ; take into account
curvature factor of spark
14C5: ED 88 10    STD   $10,X               ; and store back to Y
delta
14C8: 6A 4E       DEC   $000E,U             ; life counter, when
zero the spark disappears.
14CA: 27 08       BEQ   $14D4
14CC: 86 04       LDA   #$04                ; delay before calling
function
14CE: 8E 14 A8    LDX   #$14A8              ; address of next
function to call for this object
14D1: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

; spark has burnt out
14D4: BD D0 75    JSR   $D075               ; JMP $D31B -
deallocate object and erase object from screen
14D7: 0A 8A       DEC   $8A                 ; decrement count of
sparks on screen
14D9: 7E D0 63    JMP   $D063               ; JMP $D1F3

14DC: 96 48       LDA   $48                 ; player collision
detection?
14DE: 26 11       BNE   $14F1               ; yes
14E0: BD D0 78    JSR   $D078               ; JMP $D320 -
deallocate object and erase object from screen (2)
14E3: 0A 8A       DEC   $8A                 ; decrement count of
sparks on screen
14E5: CC 00 25    LDD   #$0025
14E8: BD D0 0C    JSR   $D00C               ; JMP $DB9C - update
player score
14EB: CC 11 60    LDD   #$1160
14EE: 7E D0 4B    JMP   $D04B               ; JMP $D3C7

14F1: 39          RTS

14F2: 08 0F 15 12 08 0F 15 8A 08 0F 16 02 08 0F 16 7A
```

```
1502: 08 0F 16 F2 08 0F 17 6A 08 0F 17 E2 08 0F 18 5A

1512: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1522: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1532: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1542: 00 00 00 00 00 00 00 00 00 00 00 0F 00 00 00 00
1552: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1562: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1572: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1582: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1592: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
15A2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
15B2: 00 00 00 00 00 00 00 00 00 00 00 0F 00 00 00 00
15C2: 00 00 00 FF F0 00 00 00 00 00 00 0F 00 00 00 00
15D2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
15E2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
15F2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1602: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1612: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1622: 00 00 00 00 00 00 00 00 00 00 00 FF F0 00 00 00
1632: 00 00 0F FF FF 00 00 00 00 00 0F F0 FF 00 00 00
1642: 00 00 0F FF FF 00 00 00 00 00 00 FF F0 00 00 00
1652: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1662: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1672: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1682: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1692: 00 00 00 00 00 00 00 00 00 00 00 FF F0 00 00 00
16A2: 00 00 0F FF FF 00 00 00 00 00 FF F0 FF F0 00 00
16B2: 00 00 FF 00 0F F0 00 00 00 00 FF F0 FF F0 00 00
16C2: 00 00 0F FF FF 00 00 00 00 00 00 FF F0 00 00 00
16D2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
16E2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
16F2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1702: 00 00 00 00 00 00 00 00 00 00 0F FF FF 00 00 00
1712: 00 00 FF FF FF F0 00 00 00 0F FF 00 0F FF 00 00
1722: 00 0F F0 00 00 FF 00 00 00 0F F0 00 00 FF 00 00
1732: 00 0F F0 00 00 FF 00 00 00 0F FF 00 0F FF 00 00
1742: 00 00 FF FF FF F0 00 00 00 00 0F FF FF 00 00 00
1752: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1762: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1772: 00 00 00 00 00 00 00 00 00 00 0F FF FF 00 00 00
1782: 00 0F FF FF FF FF 00 00 00 0F F0 00 00 FF 00 00
1792: 00 FF 00 00 00 0F F0 00 00 FF 00 00 00 0F F0 00
17A2: 00 FF 00 00 00 0F F0 00 00 FF 00 00 00 0F F0 00
17B2: 00 FF 00 00 00 0F F0 00 00 0F F0 00 00 FF 00 00
17C2: 00 0F FF FF FF FF 00 00 00 00 0F FF FF 00 00 00
17D2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
17E2: 00 00 00 00 00 00 00 00 00 00 00 FF F0 00 00 00
17F2: 00 00 FF FF FF F0 00 00 00 00 00 00 00 00 00 00
1802: 00 F0 00 00 00 00 F0 00 00 F0 00 00 00 00 F0 00
1812: 0F F0 00 00 00 00 FF 00 0F F0 00 00 00 00 FF 00
1822: 0F F0 00 00 00 00 FF 00 00 F0 00 00 00 00 F0 00
1832: 00 F0 00 00 00 00 F0 00 00 00 00 00 00 00 00 00
1842: 00 00 FF FF FF F0 00 00 00 00 00 FF F0 00 00 00
```

```
1852: 00 00 00 00 00 00 00 00 00 00 0F 00 00 00 00
1862: 00 00 00 FF F0 00 00 00 00 00 00 00 00 00 00 00
1872: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1882: 00 00 00 00 00 00 00 00 0F 00 00 00 00 00 0F 00
1892: FF 00 00 00 00 00 0F F0 0F 00 00 00 00 00 0F 00
18A2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
18B2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
18C2: 00 00 00 FF F0 00 00 00 00 00 00 0F 00 00 00 00
18D2: 05 0B 19 21 05 0B 19 58 05 0B 19 8F 05 0B 19 C6
18E2: 05 0B 19 FD 05 0B 18 EA 00 00 80 00 00 00 08 88
18F2: 00 00 00 8A AA 80 00 08 FF FF F8 00 00 08 88 00
1902: 00 90 88 78 80 90 09 97 77 99 00 90 77 77 70 90
1912: 00 00 80 00 00 00 DD DD D0 00 0D DD DD DD 00 00
1922: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1932: 00 00 00 00 00 80 00 00 00 08 F8 00 00 00 00 70
1942: 00 00 00 00 D0 00 00 00 00 00 00 00 00 00 00 00
1952: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1962: 00 00 00 00 00 00 00 80 00 00 00 08 F8 00 00 00
1972: 00 80 00 00 00 09 79 00 00 00 00 D0 00 00 00 00
1982: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1992: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 88 00
19A2: 00 00 08 FF 80 00 00 00 88 00 00 00 09 77 90 00
19B2: 00 00 77 00 00 00 00 DD 00 00 00 00 00 00 00 00
19C2: 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 08
19D2: A8 00 00 00 8F FF 80 00 00 00 80 00 00 00 08 78
19E2: 00 00 09 97 77 99 00 00 07 77 00 00 00 00 80 00
19F2: 00 00 0D DD 00 00 00 00 00 00 00 00 00 00 00 00
1A02: 00 00 88 00 00 00 08 AA 80 00 00 8F FF F8 00 00
1A12: 00 88 00 00 00 98 88 89 00 00 09 77 90 00 00 97
1A22: 77 79 00 00 00 88 00 00 00 0D DD D0 00 00 00 00
1A32: 00 00 04 07 1A 44 04 07 1A 60 04 07 1A 7C 04 07
1A42: 1A 98 00 0B 00 00 00 0B 00 00 00 0B 00 00 FF FB
1A52: FF F0 00 0B 00 00 00 0B 00 00 00 0B 00 00 00 00
1A62: 00 00 0F 00 0B 00 00 F0 B0 00 00 0B 00 00 00 B0
1A72: F0 00 0B 00 0F 00 00 00 00 00 00 0F 00 00 00 0F
1A82: 00 00 00 0F 00 00 BB BB BB B0 00 0F 00 00 00 0F
1A92: 00 00 00 0F 00 00 00 00 00 00 0B 00 0F 00 00 B0
1AA2: F0 00 00 0B 00 00 00 F0 B0 00 0F 00 0B 00 00 00
1AB2: 00 00 FF FF FF FF FF FF FF FF FF FF FF FF FF

1AC0: 7E 1A F4    JMP   $1AF4

1AC3: 21 41       BRN   $1B06
1AC5: 7E 1D AF    JMP   $1DAF

1AC8: 7E 1D C2    JMP   $1DC2

1ACB: 1F 68 D0 01 04 14 01 08 11 00 D0 02 04 17 00 D0
1ADB: 01 04 14 02 04 17 00 C8 01 08 15 01 08 14 00 D0
1AEB: 02 03 12 00 D8 01 08 11 00



INITIALISE_ALL_BRAINS:
```

```
1AF4: 0F 95       CLR   $95                 ; clear "progging" flag
1AF6: B6 BE 6E    LDA   cur_brains          ; get count of brains
1AF9: 34 02       PSHS  A                   ; save on the stack
1AFB: 27 58       BEQ   $1B55               ; exit if count == 0
1AFD: D6 95       LDB   $95

;
; Security related code.
;
1AFF: 10 8E D0 15 LDY   #$D015
1B03: EB A4       ADDB  ,Y                  ; B+= *Y
1B05: 31 28       LEAY  $0008,Y             ; Y = Y + 8
1B07: 10 8C EA B1 CMPY  #$EAB1
1B0B: 25 F6       BCS   $1B03
1B0D: C1 4A       CMPB  #$4A                ; if after all that
adding in $1b03 the total is #$4A... (For the blue label this is
always true)
1B0F: 27 0E       BEQ   $1B1F               ; The ROM hasn't been
tampered with, security check passes, so goto $1B1F.

; corrupt game state deliberately.
1B11: 96 85       LDA   $85
1B13: 81 20       CMPA  #$20
1B15: 24 08       BCC   $1B1F
1B17: 86 98       LDA   #$98
1B19: D6 86       LDB   $86
1B1B: 1F 02       TFR   D,Y                 ; so Y will be #$98xx,
which is where game state resides.
1B1D: 63 A4       COM   ,Y                  ; Change game state and
ruin the game...


; normal game resumes
1B1F: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
1B22: 1B D8          ; pointer to function

1B24: 33 84       LEAU  ,X                  ; U = X = object
metadata entry
1B26: BD D0 7B    JSR   $D07B               ; JMP $D2DA - reserve
entry in list used by grunts, hulks, brains, progs, cruise missiles
and tanks (starts at $9821)
; X= object entry
1B29: CC 21 59    LDD   #$2159              ; pointer to animation
frame metadata for brain standing still (start of wave)
1B2C: ED 02       STD   $0002,X             ; set current animation
frame metadata pointer
1B2E: ED 88 14    STD   $14,X               ; set previous
animation frame metadata pointer (previous = current)
1B31: EF 06       STU   $0006,X             ; set pointer to object
metadata
1B33: AF 47       STX   $0007,U             ; set pointer to object
in object metadata
1B35: CC 1C B2    LDD   #$1CB2              ; *pointer to pointer*
```

```
to animation frame metadata of brain moving down (as you see at the
start of the wave)
1B38: ED 4D       STD   $000D,U             ; store in object
metadata
1B3A: 6F 4B       CLR   $000B,U             ; clear animation table
index (0-based)
1B3C: CC 1D D6    LDD   #$1DD6              ; address of routine to
call when brain is shot
1B3F: ED 08       STD   $0008,X
1B41: 8D 14       BSR   $1B57               ; set the brain's
initial position
1B43: BD 1B 95    JSR   $1B95               ; find nearest family
member to prog
; Y = family member
1B46: B6 BE 62    LDA   $BE62               ; read brain cruise
missile fire delay field
1B49: BD D0 3F    JSR   $D03F               ; JMP $D6B6 - get a
random number lower than or equal to A
1B4C: A7 4C       STA   $000C,U             ; set countdown to fire
cruise missile
1B4E: BD D0 18    JSR   $D018               ; JMP $DAF2 - do blit
1B51: 6A E4       DEC   ,S                  ; reduce count of
brains left to initialise on stack
1B53: 26 CA       BNE   $1B1F               ; if !=0 , do next
brain
1B55: 35 82       PULS  A,PC ;(PUL? PC=RTS)


SET_BRAIN_INITIAL_POSITION:
1B57: BD 26 C3    JSR   $26C3               ; JMP $3199 - get
random position on playfield for object (returns: A = X coordinate,
B = Y coordinate)
1B5A: ED 04       STD   $0004,X             ; set blitter
destination
1B5C: A7 0A       STA   $000A,X             ; set brain X
coordinate (whole part)
1B5E: E7 0C       STB   $000C,X             ; set brain Y
coordinate
1B60: 10 AE 02    LDY   $0002,X             ; Y = pointer to
animation frame metadata
1B63: 0D 84       TST   $84                 ; read random number
variable
1B65: 2B 17       BMI   $1B7E               ; if bit 7 is set goto
$1b7e, to do some tinkering with the Y component of the brain
position.
1B67: 86 10       LDA   #$10                ; number to multiply by
(16 decimal)
1B69: BD D0 42    JSR   $D042               ; JMP $D6AC - multiply
A by a random number and put result in A
; A = result of multiplication
1B6C: 0D 85       TST   $85                 ; read another random
number variable
1B6E: 2B 04       BMI   $1B74               ; if bit 7 is set
(number is negative) goto $1b74
```

```
1B70: 8B 07       ADDA  #$07                ; add in left border
coordinate
1B72: 20 05       BRA   $1B79
1B74: AB A4       ADDA  ,Y                  ; add width of brain
image to A
1B76: 40          NEGA                      ; A = -A
1B77: 8B 8F       ADDA  #$8F                ; #$8F is furthest
permissible right position on playfield. So calc is: A= (#$8F -
Abs(A))
1B79: A7 04       STA   $0004,X             ; set most significant
byte of blitter destination
1B7B: A7 0A       STA   $000A,X             ; set brain X
coordinate (whole part)
1B7D: 39          RTS

;
; this piece of code adjusts the Y component of the brain's
position.
;
1B7E: 86 20       LDA   #$20
1B80: BD D0 42    JSR   $D042               ; JMP $D6AC - multiply
A by a random number and put result in A
; A = result of multiplication
1B83: 0D 86       TST   $86                 ; read yet another
random number variable
1B85: 2B 04       BMI   $1B8B               ; if bit 7 is set
(number is negative) goto $1B8B
1B87: 8B 18       ADDA  #$18                ; add in top border
coordinate
1B89: 20 05       BRA   $1B90

1B8B: AB 21       ADDA  $0001,Y             ; add height of brain
image to A
1B8D: 40          NEGA                      ; A= -A
1B8E: 8B EA       ADDA  #$EA                ; #$EA is furthermost
permissible down position on playfield. So calc is: A = (#$EA -
Abs(A))
1B90: A7 05       STA   $0005,X             ; store in least
significant byte of blitter destination (the Y part)
1B92: A7 0C       STA   $000C,X             ; set Y coordinate of
brain position (whole part)
1B94: 39          RTS


; Find the nearest family member to program. Now here's where the
infamous "brains all go for the same family member"
; bug (or "feature" depending what way you look at it) resides.
;
; On starting wave 5 and inspecting the list entries, I found that
the family member entries aren't in the list by the time this
function has been called.
; All entries in the list are NULL (WORD 0). As a result, the
pointer to the first entry of the list, $B354, is returned in Y, for
every brain.
```

```
; Thus, *all* Brains get a target of $B354.
;
; When $B354 *is* populated with a pointer to a family member, guess
what family member gets the slot? That's right - Mikey!
; that's why all Brains go for Mikey straight away.
;
; if this function was called *after* all family members were added
to the list on wave start, it would work properly.
;
; Expects:
; X = pointer to brain object
;
; returns: Y = pointer to entry in family member list (e.g. $B354)
;
;

FIND_NEAREST_FAMILY_MEMBER_TO_PROG:
1B95: CC FF FF    LDD   #$FFFF              ; D = closest distance
1B98: 10 8E B3 54 LDY   #$B354              ; start of family
member pointer list.
1B9C: 34 66       PSHS  U,Y,B,A
1B9E: EE A4       LDU   ,Y                  ; read family member
pointer from list
1BA0: 27 28       BEQ   $1BCA               ; if null, goto $1BCA
1BA2: 4F          CLRA
1BA3: E6 44       LDB   $0004,U             ; get blitter
destination hi byte (X component of address) of family member into B
1BA5: E0 04       SUBB  $0004,X             ; B-= blitter
destination hi byte (X coordinate) of brain
1BA7: 82 00       SBCA  #$00                ; if there's a carry, A
will be made #$FF and the bpl below will not execute
1BA9: 2A 04       BPL   $1BAF
1BAB: 43          COMA                      ; these 2 lines negate
D, making D a positive number again
1BAC: 50          NEGB                      ;
1BAD: 82 FF       SBCA  #$FF                ; A+= -1
1BAF: DD 2B       STD   $2B
1BB1: 4F          CLRA
1BB2: E6 45       LDB   $0005,U             ; get blitter
destination lo byte (Y component of screen address) of family member
1BB4: E0 05       SUBB  $0005,X             ;
1BB6: 82 00       SBCA  #$00                ; if there's a carry, A
will be made #$FF and the bpl below will not execute
1BB8: 2A 04       BPL   $1BBE
1BBA: 43          COMA                      ; these 2 lines negate
D, making D a positive number again
1BBB: 50          NEGB
1BBC: 82 FF       SBCA  #$FF                ; A+= -1
1BBE: D3 2B       ADDD  $2B
1BC0: 10 A3 E4    CMPD  ,S                  ; is D, our distance,
higher than the "distance to nearest family member" value on the
stack?
1BC3: 22 05       BHI   $1BCA               ; yes, we've not found
a family member closer this time, goto $1BCA
```

```
1BC5: ED E4       STD   ,S                  ; no, this distance is
lower than previous "closest", so record the "closer" distance.
1BC7: 10 AF 62    STY   $0002,S             ; and update Y (pointer
to the current closest family member ) on stack.
1BCA: 31 22       LEAY  $0002,Y             ; Y += 2. Point to next
family member pointer in list
1BCC: 10 8C B3 A4 CMPY  #$B3A4              ; at end of list?
1BD0: 26 CC       BNE   $1B9E               ; if not at end of
list, goto $1B9E
1BD2: 35 66       PULS  A,B,Y,U             ; Y will now hold the
closest family member
1BD4: 10 AF 49    STY   $0009,U             ; store family member
as brain target
1BD7: 39          RTS

1BD8: 96 59       LDA   $59
1BDA: 85 7F       BITA  #$7F
1BDC: 27 08       BEQ   $1BE6
1BDE: 86 04       LDA   #$04                ; delay before calling
function
1BE0: 8E 1B D8    LDX   #$1BD8              ; address of function
to call (this one!)
1BE3: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

1BE6: 86 0C       LDA   #$0C                ; delay before calling
function
1BE8: 8E 1B EE    LDX   #$1BEE              ; function to call
(animate brain)
1BEB: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call



BRAIN_AI:
1BEE: AE 47       LDX   $0007,U             ; get pointer to brain
object into X
1BF0: CC 00 00    LDD   #$0000              ; clear horizontal and
vertical movement temp variables
1BF3: DD 2B       STD   $2B
1BF5: 10 AE D8 09 LDY   [$09,U]             ; is our target still
alive?
1BF9: 26 14       BNE   $1C0F               ; if target is not null
then it is alive, goto $1C0F
1BFB: 10 8E 98 5A LDY   #$985A              ; Brain needs a target.
$985A is the player object.
1BFF: B6 BE 6A    LDA   cur_mommies         ; count how many family
members there are on screen
1C02: BB BE 6B    ADDA  cur_daddies
1C05: BB BE 6C    ADDA  cur_mikeys
1C08: 27 05       BEQ   $1C0F               ; if 0 family members
goto $1C0F - the brains will target the player
1C0A: BD 1B 95    JSR   $1B95               ; otherwise, family
members exist, so find nearest family member to program
```

```
; Y = target family member
1C0D: 20 E6       BRA   $1BF5

; here: X = brain, Y = target
ANIMATE_BRAIN:
1C0F: A6 24       LDA   $0004,Y             ; get hi byte (X
coordinate) of target's blitter destination into A
1C11: A0 04       SUBA  $0004,X             ; A-= hi byte of brain
blitter destination
1C13: 8B 02       ADDA  #$02                ; A+=2
1C15: 81 04       CMPA  #$04                ;
1C17: 23 0B       BLS   $1C24               ; if the brain's close
enough on the horizontal axis to its target, it doesn't need to face
left or right
1C19: C6 01       LDB   #$01                ; set horizontal
movement direction to be right for this brain
1C1B: A6 24       LDA   $0004,Y             ; get hi byte of target
blitter destination into A
1C1D: A1 04       CMPA  $0004,X             ; compare to hi byte of
brain blitter destination
1C1F: 24 01       BCC   $1C22               ; if target is to the
right goto $1C22
1C21: 50          NEGB                      ; make B (our
horizontal direction register) = $FF
1C22: D7 2B       STB   $2B                 ; store horizontal
movement value for brain (1= move brain right, $FF = move brain
left)
1C24: A6 25       LDA   $0005,Y             ; get lo byte (Y
component) of target blitter destination into A
1C26: C6 01       LDB   #$01                ; set vertical movement
direction to be down for this brain
1C28: A1 05       CMPA  $0005,X             ;
1C2A: 24 01       BCC   $1C2D               ; if target is below
brain goto $1C2D
1C2C: 50          NEGB                      ; make B (our vertical
direction register) = $FF (up). The brain's target is above it
1C2D: D7 2C       STB   $2C                 ; store vertical
movement direction for brain (1= move brain down, $FF = move brain
up)
1C2F: EC 04       LDD   $0004,X             ; get blitter
destination of brain
1C31: 9B 2B       ADDA  $2B                 ; add horizontal
movement value (see $1C22)
1C33: DB 2C       ADDB  $2C                 ; add vertical movement
value (see $1C2D)
1C35: BD 00 06    JSR   $0006               ; ensure brain stays in
bounds
1C38: 27 04       BEQ   $1C3E               ; if brain is in bounds
goto $1C3E
1C3A: 90 2B       SUBA  $2B                 ; otherwise undo the
movement brain attempted to make
1C3C: D0 2C       SUBB  $2C
1C3E: A7 0A       STA   $000A,X             ; update brain's X
coordinate (whole part)
```

```
1C40: E7 0C       STB   $000C,X             ; update brain's Y
coordinate
1C42: 10 8C 98 5A CMPY  #$985A              ; is player this brains
target?
1C46: 27 10       BEQ   $1C58               ; yes, goto $1C58

; if we get here, the brain is going after a family member.
; how close is this brain to the family member?
; X = brain object pointer, Y = family member pointer, D = brain's
blitter position
1C48: E0 25       SUBB  $0005,Y             ; B-= target's Y
component of blitter destination
1C4A: CB 03       ADDB  #$03                ; B+= 3, to adjust B. I
find the reason for this hard to explain even though I know "why" in
my head.
1C4C: C1 06       CMPB  #$06                ; is target 6 pixels
(or less) beneath the adjusted B?
1C4E: 22 08       BHI   $1C58               ; if >6 pixels beneath,
target is out of reach, goto $1C58
1C50: A0 24       SUBA  $0004,Y             ; A-= target's X
coordinate
1C52: 8B 03       ADDA  #$03                ; A+= 3, to adjust A.
1C54: 81 06       CMPA  #$06                ; is target <=12 pixels
to the right of the adjusted A? Remember 2 pixels per byte.
1C56: 23 6A       BLS   $1CC2               ; if <= than 12 pixels,
BEGIN PROGRAMMING THE HUMAN!!!!!!!!!!!! (sorry, got carried away
there.)
1C58: 96 2B       LDA   $2B                 ; read horizontal
direction temp field
1C5A: 27 0C       BEQ   $1C68               ; if 0, meaning the
brain is not moving horizontally, goto $1C68
1C5C: 2B 05       BMI   $1C63               ; if $FF (up) goto
$1C63
1C5E: CC 1C AA    LDD   #$1CAA              ; pointer to walk right
animation table
1C61: 20 11       BRA   $1C74

1C63: CC 1C A2    LDD   #$1CA2              ; pointer to walk left
animation table
1C66: 20 0C       BRA   $1C74

1C68: 96 2C       LDA   $2C                 ; read vertical
direction temp field
1C6A: 2B 05       BMI   $1C71               ; if bit 7 is set (ie:
the value is negative) goto $1C71 to make brain walk up
1C6C: CC 1C B2    LDD   #$1CB2              ; pointer to walk down
animation table
1C6F: 20 03       BRA   $1C74

1C71: CC 1C BA    LDD   #$1CBA              ; pointer to walk up
animation table

; D = pointer to animation sequence table for brain (see $1CA2)
1C74: 10 A3 4D    CMPD  $000D,U             ; are we moving in a
```

```
direction that uses the same animation sequence as the one last
used?
1C77: 27 04       BEQ   $1C7D               ; yes, no need to
change the animation table we're using, goto $1C7D
1C79: ED 4D       STD   $000D,U             ; no, different
animation needed, set animation table pointer in object metadata
1C7B: 20 08       BRA   $1C85

1C7D: E6 4B       LDB   $000B,U             ; get index into
animation table into B
1C7F: CB 02       ADDB  #$02                ; bump index to next
entry
1C81: C1 08       CMPB  #$08                ; have we hit the end
of the animation sequence?
1C83: 25 01       BCS   $1C86               ; if not, goto $1C86
1C85: 5F          CLRB                      ; yes, so we need to
reset the index into the animation table back to 0
1C86: E7 4B       STB   $000B,U             ; and update the index
to point to the correct animation frame metadata entry
1C88: 10 AE 4D    LDY   $000D,U             ; Y = animation
sequence table pointer
1C8B: EC A5       LDD   B,Y                 ; D = *(animation
sequence table pointer+index into table)
1C8D: ED 02       STD   $0002,X             ; set animation frame
metadata pointer to D
1C8F: BD D0 8D    JSR   $D08D               ; JMP $DB2F - draw
object
1C92: 6A 4C       DEC   $000C,U             ; decrement cruise
missile countdown. When zero, brain will fire a cruise missile.
1C94: 26 03       BNE   $1C99
1C96: BD 20 06    JSR   $2006               ; fire a cruise
missile!
1C99: 8E 1B EE    LDX   #$1BEE              ; pointer to "animate
brain" routine
1C9C: B6 BE 63    LDA   $BE63               ; delay before calling
function
1C9F: 7E D0 66    JMP   $D066

BRAIN_ANIMATION_TABLES:
1CA2: 21 41                                 ; address of animation
frame metadata for brain walking left #1
1CA4: 21 45                                 ; address of animation
frame metadata for brain walking left #2
1CA6: 21 41                                 ; address of animation
frame metadata for brain walking left #1
1CA8: 21 49                                 ; address of animation
frame metadata for brain walking left #3
1CAA: 21 4D                                 ; address of animation
frame metadata for brain walking right #1
1CAC: 21 51                                 ; do I really need to
repeat myself :)
1CAE: 21 4D
1CB0: 21 55
1CB2: 21 59                                 ; address of animation
```

```
frame metadata for brain moving down....
1CB4: 21 5D
1CB6: 21 59
1CB8: 21 61
1CBA: 21 65                                 ; address of animation
frame metadata for brain moving up....
1CBC: 21 69
1CBE: 21 65
1CC0: 21 6D



;
; The brain's got a family member to program!
; X = pointer to brain object
; Y = pointer to family member object
;
BEGIN_PROGRAMMING_FAMILY_MEMBER:
1CC2: A6 0A       LDA   $000A,X             ; get X coordinate
(whole part) of brain
1CC4: A1 24       CMPA  $0004,Y             ; compare to hi byte (X
component) of family member blitter destination
1CC6: 25 12       BCS   $1CDA               ; if brain.X <= family
member.X goto $1CDA
1CC8: A6 0A       LDA   $000A,X             ; get X coordinate
(whole part) of brain
1CCA: A0 B8 02    SUBA  [$02,Y]             ; subtract width of
family member being prog'd
1CCD: 80 01       SUBA  #$01
1CCF: 81 07       CMPA  #$07                ; is the result past
the left edge of the playfield?
1CD1: 25 07       BCS   $1CDA               ; yes, so result is
invalid, goto $1CDA
1CD3: A7 2A       STA   $000A,Y             ; set X coordinate of
family member being prog'd. Now family member will be standing left
of brain.
1CD5: CC 21 41    LDD   #$2141              ; animation frame
pointer of brain facing left
1CD8: 20 0D       BRA   $1CE7
1CDA: A6 0A       LDA   $000A,X             ; get X coordinate
(whole part) of brain
1CDC: 8B 08       ADDA  #$08                ; X += 8 (16 pixels)
1CDE: 81 8B       CMPA  #$8B                ; X > #$8B (past the
right edge of the playfield)?
1CE0: 24 E6       BCC   $1CC8               ; yes, X coordinate is
invalid, goto $1CC8
1CE2: A7 2A       STA   $000A,Y             ; store to X coordinate
of family member. Now family member will be standing right of brain.
1CE4: CC 21 4D    LDD   #$214D              ; animation frame
pointer of brain facing right, programming human
1CE7: ED 02       STD   $0002,X             ; save pointer to
current animation frame metadata
1CE9: A6 0C       LDA   $000C,X             ; read brain Y
coordinate
```

```
1CEB: 8B 02       ADDA  #$02                ; Y+ = 2
1CED: A7 2C       STA   $000C,Y             ; set family member Y
coordinate.
1CEF: BD D0 15    JSR   $D015               ; JMP $DB03 - erase
object from screen
1CF2: 6F 0B       CLR   $000B,X
1CF4: 6F 88 12    CLR   $12,X               ; clear flag for image
to be shifted right one pixel
1CF7: BD 1D AF    JSR   $1DAF               ; draw brain in
programming mode!!
1CFA: AE D8 09    LDX   [$09,U]             ; get pointer to family
member object from family member list ($B354 - $B3A4.)
; X = family member object
1CFD: 34 50       PSHS  U,X
1CFF: 86 01       LDA   #$01
1D01: 97 95       STA   $95                 ; set "you are being
prog'd!" flag which collision handler will read
1D03: AD 98 08    JSR   [$08,X]             ; call family member's
collision handler
1D06: 35 50       PULS  X,U
1D08: 0F 95       CLR   $95
1D0A: EC 84       LDD   ,X                  ; mark family member
object as free
1D0C: DD 1B       STD   $1B
1D0E: AF 49       STX   $0009,U             ; set brain's target in
object metadata
1D10: 6F 0B       CLR   $000B,X
1D12: 6F 88 12    CLR   $12,X
1D15: 10 AE 06    LDY   $0006,X             ; Y = pointer to family
member object's metadata
1D18: EC 29       LDD   $0009,Y             ; D = pointer to object
metadata's very own animation frame metadata pointer. Confused?
1D1A: ED 02       STD   $0002,X             ; set family member
objects animation frame metadata pointer (to "standing still" image)
1D1C: 86 14       LDA   #$14
1D1E: A7 4B       STA   $000B,U
1D20: AE 47       LDX   $0007,U             ; get pointer to brain
object from brain object's metadata
1D22: CC 1A EA    LDD   #$1AEA
1D25: BD D0 4B    JSR   $D04B               ; JMP $D3C7
1D28: BD 1D AF    JSR   $1DAF               ; draw the brain
prog'ing away.
1D2B: AE 49       LDX   $0009,U             ; X = pointer to family
member object from brain object's metadata
1D2D: EC 04       LDD   $0004,X             ; D = blit destination
of family member
1D2F: 10 AE 02    LDY   $0002,X             ; Y = pointer to
animation frame metadata
1D32: BD D0 1E    JSR   $D01E               ; JMP $DABF: clear
image rectangle
1D35: A6 0A       LDA   $000A,X             ; A = X coordinate
(whole part)

DRAW_PROG_STARTING_TO_SHAKE:
```

```
1D37: D6 84       LDB   $84                 ; get a random number
1D39: C4 07       ANDB  #$07                ; mask in bits 0..2,
giving us a number between 0..7 in B
1D3B: EB 0C       ADDB  $000C,X             ; B = B + Y coordinate
1D3D: C1 DC       CMPB  #$DC                ; B <= #$DC? (will the
shaking prog be drawn within the playfield?)
1D3F: 23 02       BLS   $1D43               ; yes, no need to
adjust Y, goto $1D43
1D41: C6 DC       LDB   #$DC                ; No, so make B #$DC.
The prog's Y coordinate has been adjusted to keep it in the
playfield.
1D43: ED 04       STD   $0004,X             ; blit destination = D
1D45: CC AA BB    LDD   #$AABB              ; set remap colour 1
and 2
1D48: 8D 78       BSR   $1DC2               ; draw rectangle in
colour A, then family member image as solid with colour B
1D4A: 86 02       LDA   #$02                ; delay before calling
function
1D4C: 8E 1D 52    LDX   #$1D52              ; address of routine to
call next
1D4F: 7E D0 66    JMP   $D066


1D52: AE 49       LDX   $0009,U             ; X = pointer to family
member object being programmed
1D54: EC 04       LDD   $0004,X             ; D = blitter
destination
1D56: 10 AE 02    LDY   $0002,X             ; Y = pointer to
animation frame metadata
1D59: BD D0 1E    JSR   $D01E               ; JMP $DABF: clear
image rectangle

; this is another part that makes the prog "shake" vertically as the
Brain is programming it...
1D5C: A6 0A       LDA   $000A,X             ; A = X coordinate
(whole part)
1D5E: D6 84       LDB   $84                 ; get a random number
into B
1D60: C4 07       ANDB  #$07                ; make number from 0..7
1D62: 50          NEGB
1D63: EB 0C       ADDB  $000C,X             ; B += Y coordinate.
This gives us a coordinate to draw the shaking prog at - but we
never update the prog's actual Y coordinate in memory
1D65: C1 18       CMPB  #$18                ; is B > top border
wall of game ?
1D67: 24 02       BCC   $1D6B               ; Yes, calculated Y
coordinate is within game bounds, no need to adjust it, goto $1D6B
1D69: C6 18       LDB   #$18                ; otherwise draw prog
at very top of game confines
1D6B: ED 04       STD   $0004,X             ; blit destination = D
1D6D: CC AA BB    LDD   #$AABB              ; set remap colour 1
and 2
1D70: 8D 50       BSR   $1DC2               ; draw rectangle in
colour A, then family member as solid with colour B
```

```
1D72: 86 02       LDA   #$02                ; delay before calling
function
1D74: 8E 1D 7A    LDX   #$1D7A              ; address of function
to call next
1D77: 7E D0 66    JMP   $D066

1D7A: 6A 4B       DEC   $000B,U             ; decrement the
prog'ing countdown. When 0 the prog'ing is done
1D7C: 26 A2       BNE   $1D20               ; if countdown !=0 then
goto 1D20, keep drawing brain proging and family member being
prog'd.
1D7E: CC 1A EF    LDD   #$1AEF
1D81: BD D0 4B    JSR   $D04B               ; JMP $D3C7
1D84: AE 49       LDX   $0009,U             ; get pointer to family
member target from brain object metadata
1D86: DC 1B       LDD   $1B
1D88: ED 84       STD   ,X
1D8A: 9F 1B       STX   $1B                 ; mark family member as
a free object (the programming's done!)
1D8C: EC 04       LDD   $0004,X             ; D = blitter
destination
1D8E: 10 AE 02    LDY   $0002,X             ; Y = pointer to
animation frame metadata
1D91: BD D0 1E    JSR   $D01E               ; JMP $DABF: clear
image rectangle
; no idea why these are reloaded here, the previous subroutine saved
D on stack and didn't corrupt Y
1D94: EC 04       LDD   $0004,X             ; D = blitter
destination
1D96: 10 AE 02    LDY   $0002,X             ; Y = pointer to
animation frame metadata
1D99: BD 1E 19    JSR   $1E19               ; create the prog
1D9C: AE 47       LDX   $0007,U             ; get pointer to object
from object metadata
1D9E: EC 04       LDD   $0004,X             ; D = blitter
destination
1DA0: 10 AE 02    LDY   $0002,X             ; Y = pointer to
animation frame metadata
1DA3: BD D0 1E    JSR   $D01E               ; JMP $DABF: clear
image rectangle
1DA6: BD D0 18    JSR   $D018               ; JMP $DAF2 - do blit
1DA9: BD 1B 95    JSR   $1B95               ; find another family
member to prog!!!!
1DAC: 7E 1B EE    JMP   $1BEE


DRAW_BRAIN_IN_PROGGING_STATE:
1DAF: C6 BB       LDB   #$BB                ; remap colour
1DB1: D7 2D       STB   $2D
1DB3: A6 0A       LDA   $000A,X             ; A = object X
coordinate (whole part)
1DB5: E6 0C       LDB   $000C,X             ; read object Y
coordinate
1DB7: ED 04       STD   $0004,X             ; set blitter
```

```
destination pointer
1DB9: 10 AE 02    LDY   $0002,X             ; Y = pointer to
animation frame metadata
1DBC: BD D0 93    JSR   $D093               ; JMP $DA61 - blit
rectangle with colour remap
1DBF: 7E D0 18    JMP   $D018               ; JMP $DAF2 - do blit


; Blit an image as a single solid colour, with another solid colour
as its background.
; Think of when the progs are being programmed, mommy/daddy/mikey is
drawn as a solid shape
; and has a rapidly cycling background colour.
;
; X = object pointer
; A = solid colour to draw as rectangular background
; B = solid colour to draw image belonging to X in
;

BLIT_IMAGE_AS_SOLID_COLOUR_WITH_SOLID_BACKGROUND:
1DC2: 34 06       PSHS  B,A
1DC4: 97 2D       STA   $2D                 ; store remap colour 1
1DC6: EC 04       LDD   $0004,X             ; D = blitter
destination
1DC8: BD D0 93    JSR   $D093               ; JMP $DA61 - blit
rectangle with colour remap
1DCB: A6 61       LDA   $0001,S             ; A = B (remap colour
2)
1DCD: 97 2D       STA   $2D                 ; $2D = remap colour 2
1DCF: A6 04       LDA   $0004,X             ; get blitter
destination
1DD1: BD D0 90    JSR   $D090               ; JMP $DA9E - do solid
and transparent blit
1DD4: 35 86       PULS  A,B,PC ;(PUL? PC=RTS)


BRAIN_COLLISION_HANDLER:
1DD6: 96 48       LDA   $48                 ; player collision
detection?
1DD8: 26 3C       BNE   $1E16               ; yes, goto $1E16. We
don't do anything here.
1DDA: 7A BE 6E    DEC   cur_brains          ; oh dear, the brain's
been hit by a laser. Reduce brain count
1DDD: BD 5B 4F    JSR   $5B4F               ; JMP $5C0A
1DE0: BD D0 7E    JSR   $D07E               ; JMP $D2C2 - remove
baddy from baddies list
1DE3: AE 06       LDX   $0006,X             ; X = pointer to object
metadata for this object
1DE5: 33 84       LEAU  ,X
1DE7: BD D0 5D    JSR   $D05D               ; JMP $D218 -
deallocate object metadata entry
1DEA: EC 42       LDD   $0002,U
1DEC: 10 83 1D 52 CMPD  #$1D52
1DF0: 25 17       BCS   $1E09
```

```
1DF2: AE 49       LDX   $0009,U             ; get pointer to family
member target from brain object metadata
1DF4: DC 1B       LDD   $1B                 ; get free object entry
list pointer
1DF6: ED 84       STD   ,X                  ; store the family
member object as next object in free object linked list
1DF8: 9F 1B       STX   $1B                 ; make this dead family
member object start of the free object entry linked list
1DFA: 10 AE 02    LDY   $0002,X             ; Y = pointer to
animation frame metadata
1DFD: EC 04       LDD   $0004,X             ; D = blitter
destination
1DFF: BD D0 1E    JSR   $D01E               ; JMP $DABF: clear
image rectangle
1E02: 0C 95       INC   $95
1E04: BD 00 09    JSR   $0009               ; JSR $0351 - draw
points value for family member saved from progging
1E07: 0F 95       CLR   $95
1E09: CC 1A CD    LDD   #$1ACD
1E0C: BD D0 4B    JSR   $D04B               ; JMP $D3C7
1E0F: CC 01 50    LDD   #$0150
1E12: BD D0 0C    JSR   $D00C               ; JMP $DB9C - update
player score
1E15: 39          RTS

1E16: 7E D0 18    JMP   $D018               ; JMP $DAF2 - do blit


;
; D = blitter destination of family member that's been prog'd
; Y = pointer to animation frame metadata (so that prog can use
frames of family member when rendering itself)

CREATE_PROG:
1E19: 34 56       PSHS  U,X,B,A
1E1B: DC 1D       LDD   $1D                 ; do we have space for
prog in the prog list?
1E1D: 27 34       BEQ   $1E53               ; no, so goto $1E53,
which is an RTS
1E1F: 4F          CLRA
1E20: 8E 1E AB    LDX   #$1EAB              ; address of prog
animation function to call
1E23: BD D0 5A    JSR   $D05A               ; JMP $D243 - reserve
object metadata entry in list @ $981D and call function in X
1E26: 33 84       LEAU  ,X                  ; U = object metadata
entry
1E28: BD 1F FC    JSR   $1FFC               ; call function to
clear (zero) bytes 7..31 decimal in object metadata
1E2B: 86 11       LDA   #$11
1E2D: A7 4F       STA   $000F,U             ; set prog trail index
to start of trail list, which begins in memory at U + #$11 (17
decimal).
                                            ; for more info on the
"prog trail" see $1EFD
```

```
1E2F: BD D0 7B    JSR   $D07B               ; JMP $D2DA - reserve
entry in list used by grunts, hulks, brains, progs, cruise missiles
and tanks (starts at $9821)
; X = pointer to reserved object
1E32: AF 47       STX   $0007,U             ; save pointer to this
prog object in prog object metadata
1E34: EF 06       STU   $0006,X             ; set pointer to
metadata in this prog object
1E36: 10 AF 49    STY   $0009,U             ; set pointer to
animation frame metadata in object metadata (phew!)
1E39: 10 AF 02    STY   $0002,X             ; set current animation
frame metadata pointer
1E3C: 10 AF 88 14 STY   $14,X               ; set previous
animation frame metadata pointer (previous = current)
1E40: EC E4       LDD   ,S                  ; read D from stack
(remember, A and B comprise 16 bit register D)
1E42: A7 0A       STA   $000A,X             ; set X coordinate
(whole part)
1E44: E7 0C       STB   $000C,X             ; set Y coordinate
1E46: CC 1F 1F    LDD   #$1F1F              ; address of function
to call when this prog is in a collision
1E49: ED 08       STD   $0008,X
1E4B: 8D 08       BSR   $1E55               ; compute distances
from player for prog to move towards
1E4D: 8D 22       BSR   $1E71               ; set initial prog
direction
1E4F: 4F          CLRA
1E50: 5F          CLRB                      ; D = 0
1E51: ED 04       STD   $0004,X             ; set blitter
destination to be NULL
1E53: 35 D6       PULS  A,B,X,U,PC ;(PUL? PC=RTS)


;
; This function here computes two variables: X Distance and Y
Distance.
; The X Distance and Y Distance variables are added on to the Player
X and Player Y coordinates respectively.
; The Prog will run to the resulting point(PlayerX + X Distance,
PlayerY + Y Distance) coordinates - AS LONG AS THE
; COORDINATES ARE WITHIN PLAYFIELD BOUNDS.
;
; Obviously, the smaller the distances, the closer the prog chases
towards the player.
;

COMPUTE_X_AND_Y_DISTANCES_TO_CHASE:
1E55: 86 0F       LDA   #$0F                ; load A with 15
(decimal)
1E57: BD D0 3F    JSR   $D03F               ; JMP $D6B6 - get a
random number lower than or equal to A
1E5A: 8B F0       ADDA  #$F0                ; A+= -16 (decimal).
This will make the number in A negative
1E5C: 40          NEGA                      ; make number positive
```

```
1E5D: 48          ASLA                      ; multiply by 2
1E5E: 48          ASLA                      ; multiply by 2 again
1E5F: 8B E0       ADDA  #$E0                ; A+= -32
1E61: A7 4B       STA   $000B,U             ; save as X Distance
1E63: 86 12       LDA   #$12                ; load a with 18
decimal
1E65: BD D0 3F    JSR   $D03F               ; JMP $D6B6 - get a
random number lower than or equal to A
1E68: 8B ED       ADDA  #$ED                ; A+= -19 (decimal).
This will make the number in A negative
1E6A: 40          NEGA                      ; make number positive
1E6B: 48          ASLA                      ; multiply by 2
1E6C: 8B EE       ADDA  #$EE                ; A+= -18 (decimal)
1E6E: A7 4C       STA   $000C,U             ; save as Y Distance
1E70: 39          RTS


; Instead of me wittering on about X-axis and Y axis, I'll make this
simple.
;
; A prog can move in the following directions:
;
; LEFT, RIGHT, UP, DOWN. That is all.
;
; No combinations thereof are permitted.
;

CHANGE_PROG_DIRECTION:
1E71: 96 85       LDA   $85                 ; read a random number
1E73: 2B 18       BMI   $1E8D               ; if negative (bit 7
set) goto $1E8D - make the prog move vertically

; if we get here, the prog is going to move horizontally...
1E75: 96 64       LDA   $64                 ; Get player X
coordinate
1E77: AB 4B       ADDA  $000B,U             ; add in X distance, to
give target X
1E79: 81 BF       CMPA  #$BF                ; is target X *way* off
the playfield bounds??
1E7B: 23 02       BLS   $1E7F               ; no, goto $1E7F
1E7D: 86 07       LDA   #$07                ; otherwise, set target
X to be left border edge
1E7F: A1 0A       CMPA  $000A,X             ; compare to prog's X
coordinate (whole part)
1E81: 23 05       BLS   $1E88               ; if target X <= prog
X, goto $1E88 to make the prog move LEFT

1E83: CC 1F D8    LDD   #$1FD8              ; pointer to animation
table to move prog RIGHT
1E86: 20 1B       BRA   $1EA3               ; skip over the code
that moves the prog vertically

1E88: CC 1F CC    LDD   #$1FCC              ; pointer to animation
table to move prog LEFT
```

```
1E8B: 20 16       BRA   $1EA3               ; skip over the code
that moves the prog vertically

; if we get here, the prog is going to move vertically...
1E8D: 96 66       LDA   $66                 ; Get player Y
coordinate
1E8F: AB 4C       ADDA  $000C,U             ; add in Y distance, to
give target Y
1E91: 81 FC       CMPA  #$FC                ; is target Y *way* off
the playfield bounds?
1E93: 23 02       BLS   $1E97               ;
1E95: 86 18       LDA   #$18
1E97: A1 0C       CMPA  $000C,X             ; compare to prog's Y
coordinate
1E99: 23 05       BLS   $1EA0               ; if target Y<= prog Y,
goto $1EA0 to make the prog move UP

1E9B: CC 1F E4    LDD   #$1FE4              ; pointer to animation
table to move prog DOWN
1E9E: 20 03       BRA   $1EA3

1EA0: CC 1F F0    LDD   #$1FF0              ; pointer to animation
table to move prog  UP
1EA3: ED 4D       STD   $000D,U             ; update animation
table pointer for prog
1EA5: 86 FD       LDA   #$FD                ; set A to -3 decimal
(this might look strange, but look at code at $1EB3, which adds 3 to
this value before it is used)
1EA7: A7 88 13    STA   $13,X               ; set index into
animation tables
1EAA: 39          RTS


ANIMATE_PROG:
1EAB: AE 47       LDX   $0007,U             ; get pointer to object
into X
1EAD: 10 AE 4D    LDY   $000D,U             ; get pointer to prog
animation table into Y (see $1FCC for description of animation
table)
1EB0: A6 88 13    LDA   $13,X               ; get index into
animation tables (as each entry in the table takes 3 bytes, this
number is a multiple of 3)
1EB3: 8B 03       ADDA  #$03                ; bump index to next
entry in animation table, by adding 3
1EB5: 81 09       CMPA  #$09                ; there's 12 bytes per
entry
1EB7: 23 01       BLS   $1EBA               ; if we're not at the
end of the animation table then goto $1EBA
1EB9: 4F          CLRA                      ; reset index - taking
the animation back to the start
1EBA: A7 88 13    STA   $13,X               ; set animation index
1EBD: 31 A6       LEAY  A,Y                 ; Y = Y + A
1EBF: E6 A4       LDB   ,Y                  ; read offset into
animation metadata list (see docs at $1FCC)
```

```
1EC1: 4F          CLRA                      ; clear A so that
doesn't affect the addition below
1EC2: E3 49       ADDD  $0009,U             ; add D to animation
frame metadata list pointer. Now D points to animation frame
metadata, used for the blit funcs.
1EC4: ED 02       STD   $0002,X             ; set pointer to
animation frame metadata
1EC6: EC 21       LDD   $0001,Y             ; get direction deltas
to add to X and Y coordinates (see $1FCC)
1EC8: AB 0A       ADDA  $000A,X             ; A += X coordinate of
prog (whole part)
1ECA: EB 0C       ADDB  $000C,X             ; B += Y coordinate of
prog
1ECC: BD 00 06    JSR   $0006               ; test that object is
in bounds
1ECF: 26 13       BNE   $1EE4               ; if zero flag not set,
object coordinate is invalid, goto $1EE4 to change prog direction
1ED1: A7 0A       STA   $000A,X             ; X coordinate (whole
part) = A
1ED3: E7 0C       STB   $000C,X             ; Y coordinate = B
1ED5: 96 84       LDA   $84                 ; get a random number
1ED7: 81 F8       CMPA  #$F8                ; compare to #$F8 (248
decimal)
1ED9: 23 03       BLS   $1EDE
1EDB: BD 1E 55    JSR   $1E55               ; call
COMPUTE_X_AND_Y_DISTANCES_TO_CHASE
1EDE: 96 86       LDA   $86                 ; read a random number
1EE0: 81 E4       CMPA  #$E4                ; compare to #$E4 (228
decimal)
1EE2: 23 03       BLS   $1EE7               ; if <= goto $1EE7, do
not change prog direction
1EE4: BD 1E 71    JSR   $1E71               ; change prog direction
1EE7: 10 AE 02    LDY   $0002,X             ; Y = pointer to
animation frame metadata
1EEA: A6 4F       LDA   $000F,U             ; get prog trail list
index into A
1EEC: EC C6       LDD   A,U                 ; D = blitter
destination
1EEE: BD D0 1E    JSR   $D01E               ; JMP $DABF: clear
image rectangle (erase the "tail" of the prog)
1EF1: CC EE 00    LDD   #$EE00              ; colours
1EF4: BD 1D C2    JSR   $1DC2               ; draw rectangle in
colour A, then prog as solid with colour B
1EF7: A6 0A       LDA   $000A,X             ; A = X coordinate of
prog (whole part)
1EF9: E6 0C       LDB   $000C,X             ; B = Y coordinate of
prog (whole part)
1EFB: ED 04       STD   $0004,X             ; set blitter
destination of object
1EFD: 1F 02       TFR   D,Y                 ; Y = blitter
destination
;
; The prog trail is a list - not a queue - of 7 pointers. Each
pointer contains a screen address where blits of the prog have been
```

```
made.
; The prog trail is used to remember where the prog was blitted, so
that the "trail" of the prog can be erased from the screen.
;
1EFF: A6 4F       LDA   $000F,U             ; get prog trail list
index
1F01: 10 AF C6    STY   A,U                 ; * (A+U) = blitter
destination
1F04: 8B 02       ADDA  #$02                ; bump index to next
item in prog trail, effectively "growing" the trail
1F06: 81 1F       CMPA  #$1F                ; are we at the end of
the list?
1F08: 25 02       BCS   $1F0C               ; no, goto $1F0C
1F0A: 86 11       LDA   #$11                ; reset prog trail
index to start of list
1F0C: A7 4F       STA   $000F,U             ; update prog trail
index to be A
1F0E: 10 AE 02    LDY   $0002,X             ; Y = animation frame
metadata pointer
1F11: CC 00 AA    LDD   #$00AA              ; remap colours 1 + 2.
1F14: BD 1D C2    JSR   $1DC2               ; blit solid background
in A + then do solid and transparent blit with B
1F17: 86 03       LDA   #$03                ; delay before calling
function
1F19: 8E 1E AB    LDX   #$1EAB              ; address of function
to call
1F1C: 7E D0 66    JMP   $D066



PROG_COLLISION_DETECTION:
1F1F: 96 48       LDA   $48                 ; is the player calling
this function?
1F21: 26 44       BNE   $1F67               ; yes, goto $1F67
(don't do anything)
;
; The prog has been killed. We need to erase the trail of the prog,
as well as the prog itself.
;
1F23: 34 10       PSHS  X                   ; save pointer to prog
object on stack
1F25: 10 AE 02    LDY   $0002,X             ; Y = pointer to
animation frame metadata
1F28: AE 06       LDX   $0006,X             ; get pointer to object
metadata for this object into X
1F2A: 86 11       LDA   #$11                ; start of prog trail
which begins at (X + #$11) (#$11 is 17 decimal)
1F2C: 34 02       PSHS  A
1F2E: EC 86       LDD   A,X                 ; Get pointer to where
prog was blitted
1F30: BD D0 1E    JSR   $D01E               ; JMP $DABF: clear
image rectangle
1F33: 35 02       PULS  A
1F35: 8B 02       ADDA  #$02                ; bump to next entry in
```

```
the prog trail
1F37: 81 1F       CMPA  #$1F                ; have we reached the
end of the trail? (ie: all segments of prog trail have been erased)
1F39: 25 F1       BCS   $1F2C               ; no, goto $1F2C and
clear rest of prog trail
1F3B: BD D0 5D    JSR   $D05D               ; JMP $D218 -
deallocate object metadata entry
1F3E: 35 10       PULS  X                   ; restore pointer to
prog object from stack
1F40: CC 1F 68    LDD   #$1F68              ; pointer to animation
frame metadata
1F43: ED 02       STD   $0002,X             ; set current animation
frame metadata pointer
1F45: 86 8A       LDA   #$8A
1F47: A1 04       CMPA  $0004,X             ; compare to Hi byte (X
component) of blitter destination
1F49: 24 02       BCC   $1F4D               ; if A > #$8A (138
decimal) goto $1F4D
1F4B: A7 04       STA   $0004,X
1F4D: 86 DB       LDA   #$DB                ;
1F4F: A1 05       CMPA  $0005,X             ; compare to Lo byte (y
component) of blitter destination
1F51: 24 02       BCC   $1F55               ; if A > #$DB (219
decimal) goto $1F55
1F53: A7 05       STA   $0005,X
1F55: BD 5B 43    JSR   $5B43               ; JMP $5C1F - create an
explosion
1F58: BD D0 7E    JSR   $D07E               ; JMP $D2C2 - remove
baddy from baddies list
1F5B: CC 1A DA    LDD   #$1ADA
1F5E: BD D0 4B    JSR   $D04B               ; JMP $D3C7
1F61: CC 01 10    LDD   #$0110
1F64: BD D0 0C    JSR   $D00C               ; JMP $DB9C - update
player score
1F67: 39          RTS

; Animation frame metadata
1F68: 06 10 1F 6C


1F6C: AA AA AA AA AA A0 AA 00 00 00 0A A0
1F78: AA 0B B0 BB 0A A0 AA 0B B0 BB 0A A0 AA 0B B0 BB
1F88: 0A A0 AA 00 00 00 0A A0 AA AA A0 AA AA A0 AA A0
1F98: 00 00 AA A0 AA 00 00 00 0A A0 AA 0A 00 0A 0A A0
1FA8: AA 0A 00 0A 0A A0 AA AA 0A 0A AA A0 AA AA 0A 0A
1FB8: AA A0 AA AA 0A 0A AA A0 AA 00 0A 00 0A A0 AA AA
1FC8: AA AA AA A0


;
; Animation tables for prog.
;
; Byte 0: offset into family member animation metadata list.
; Byte 1: signed offset to add to X coordinate of prog
```

```
; Byte 2: signed offset to add to Y coordinate of prog
;
;

PROG_ANIMATION_TABLES:

1FCC :
00 FE 00
04 FE 00
00 FE 00
08 FE 00

1FD8:
0C 02 00
10 02 00
0C 02 00
14 02 00

1FE4:
18 00 04
1C 00 04
18 00 04
20 00 04

1FF0:
24 00 FC
28 00 FC
24 00 FC
2C 00 FC


;
;
;

CLEAR_BYTES_7_TO_31_FROM_U:
1FFC: 86 07       LDA   #$07
1FFE: 6F C6       CLR   A, U
2000: 4C          INCA
2001: 81 1F       CMPA  #$1F                ; compare A to #$1F (31
decimal)
2003: 25 F9       BCS   $1FFE               ; if lower, goto $1FFE
2005: 39          RTS



;
; The missiles that Brains fire at the player are called "Cruise
missiles"
;
;

CREATE_CRUISE_MISSILE:
2006: 34 50       PSHS  U,X
```

```
2008: B6 BE 62    LDA   $BE62
200B: BD D0 3F    JSR   $D03F                ; JMP $D6B6 - get a
random number lower than or equal to A
200E: A7 4C       STA   $000C,U
2010: 96 8E       LDA   $8E                  ; get count of cruise
missiles currently on screen
2012: 81 08       CMPA  #$08                 ; 8?
2014: 24 43       BCC   $2059                ; if >=8, we have
enough missiles on screen as it is, goto $2059
2016: DC 1D       LDD   $1D                  ; do we have a slot for
the cruise missile to go into?
2018: 27 3F       BEQ   $2059                ; if not, goto $2059
201A: 31 84       LEAY  ,X                   ; Y = X
201C: 4F          CLRA
201D: 8E 20 B0    LDX   #$20B0
2020: BD D0 5A    JSR   $D05A                ; JMP $D243 - reserve
object metadata entry in list @ $981D and call function in X
2023: 33 84       LEAU  ,X
2025: 8D D5       BSR   $1FFC                ; clear bytes (U+7 to U
+31)
2027: BD D0 7B    JSR   $D07B                ; JMP $D2DA - reserve
entry in list used by grunts, hulks, brains, progs, cruise missiles
and tanks (starts at $9821)
202A: CC 20 5B    LDD   #$205B               ; pointer to collision
detection animation frame metadata for cruise missile (see $D7F4)
202D: ED 88 16    STD   $16,X                ;
2030: CC 20 6B    LDD   #$206B               ; pointer to cruise
missile animation frame metadata
2033: ED 02       STD   $0002,X              ; set current animation
frame metadata pointer
2035: ED 88 14    STD   $14,X                ; set previous
animation frame metadata pointer (previous = current)
2038: CC 21 19    LDD   #$2119               ; set address of
routine to call when shot
203B: ED 08       STD   $0008,X
203D: EF 06       STU   $0006,X              ; set pointer to object
metadata in object
203F: AF 47       STX   $0007,U              ; set pointer to object
in object metadata
2041: EC 24       LDD   $0004,Y
2043: C3 03 04    ADDD  #$0304
2046: ED 04       STD   $0004,X              ; set "last" blitter
destination
2048: ED 0A       STD   $000A,X              ; set coordinates
204A: BD 20 7B    JSR   $207B                ; pick direction for
cruise missile to go in
204D: 0C 8E       INC   $8E                  ; increment cruise
missile count
204F: 86 0D       LDA   #$0D
2051: A7 4C       STA   $000C,U              ; reset "cruise missile
trail" list index to beginning
2053: CC 1A E2    LDD   #$1AE2
2056: BD D0 4B    JSR   $D04B                ; JMP $D3C7
2059: 35 D0       PULS  X,U,PC ;(PUL? PC=RTS)
```

```
205B: 03 04 20 5F FF FF FF FF FF FF FF FF FF FF FF

206B: 03 04 20 6F 00 00 00 00 FF 00 00 FF 00 00 00 00


;
; Pick a direction for cruise missile
;

PICK_CRUISE_MISSILE_DIRECTION:
207B: CC 00 00    LDD   #$0000
207E: ED 49       STD   $0009,U             ; clear deltas of
cruise missile
2080: 96 84       LDA   $84                 ; get a random number
2082: 2A 13       BPL   $2097               ; if bit 7 is not set,
no adjustment of horizontal movement to be done, goto $2097
2084: 84 0F       ANDA  #$0F                ; mask off bottom 4
bits
2086: 8B FA       ADDA  #$FA                ; add -6 to A
2088: 9B 64       ADDA  $64                 ; add in player's X
coordinate ($9864) to give target X
208A: C6 01       LDB   #$01                ; X delta =1 (moving
right)
208C: A1 0A       CMPA  $000A,X             ; compare A to cruise
missile X coordinate
208E: 24 01       BCC   $2091               ; if target X >= X
coordinate, goto $2091
2090: 50          NEGB                      ; X delta = -1 (moving
left)
2091: E7 49       STB   $0009,U             ; set X delta of cruise
missile
2093: 96 86       LDA   $86                 ; read a random number
2095: 2B 11       BMI   $20A8               ; if bit 7 set, goto
$20A8
2097: 96 85       LDA   $85                 ; get another random
number
2099: 84 0F       ANDA  #$0F                ; mask off bottom 4
bits, to give a number from 0..15 in A
209B: 8B FA       ADDA  #$FA                ; add -6 to A
209D: C6 01       LDB   #$01                ; Y delta = 1 (moving
down)
209F: 9B 66       ADDA  $66                 ; add in player's Y
coordinate ($9866)
20A1: A1 0B       CMPA  $000B,X             ; compare A to cruise
missile Y coordinate
20A3: 24 01       BCC   $20A6               ; if A >= Y coordinate,
goto $20A6
20A5: 50          NEGB                      ; Y delta = -1 (moving
up)
20A6: E7 4A       STB   $000A,U             ; set Y delta of cruise
missile
20A8: 86 07       LDA   #$07
20AA: BD D0 3F    JSR   $D03F               ; JMP $D6B6 - get a
```

```
random number lower than or equal to A
20AD: A7 4B       STA   $000B,U             ; set countdown before
cruise missile changes direction
20AF: 39          RTS



20B0: AE 47       LDX   $0007,U             ; get object pointer
from object metadata
20B2: 6A 4B       DEC   $000B,U             ; decrement countdown
before cruise missile changes direction
20B4: 26 03       BNE   $20B9               ; if countdown != 0,
goto $20B9
20B6: BD 20 7B    JSR   $207B               ; otherwise, countdown
is zero, its time for cruise missile to change direction
20B9: BD 20 C7    JSR   $20C7               ; move cruise missile
(creating a trail)
20BC: BD 20 C7    JSR   $20C7               ; move cruise missile
again (creating a trail)
20BF: 86 02       LDA   #$02                ; delay before calling
function
20C1: 8E 20 B0    LDX   #$20B0              ; address of function
to call
20C4: 7E D0 66    JMP   $D066


MOVE_CRUISE_MISSILE:
20C7: EC 0A       LDD   $000A,X             ; D = objects
coordinates (MSB = X coordinate, LSB = Y coordinate).
20C9: AB 49       ADDA  $0009,U             ; add cruise missile's
X delta to most significant byte of coordinate
20CB: 81 07       CMPA  #$07                ; cruise missile at
left-most border?
20CD: 24 06       BCC   $20D5               ; no, goto $20D5
20CF: A0 49       SUBA  $0009,U             ; yes, cruise missile
has reached edge of screen, can't move any further, so undo change
made at $20C9
20D1: 60 49       NEG   $0009,U             ; negate X delta of
cruise missile (making it move in opposite direction horizontally)
20D3: 20 F4       BRA   $20C9

20D5: 81 8E       CMPA  #$8E                ; cruise missile at
right-most border?
20D7: 22 F6       BHI   $20CF               ; yes, goto $20CF to
undo change made at $20C9

20D9: EB 4A       ADDB  $000A,U             ; add cruise missile's
Y delta to most significant byte (whole part) of Y coordinate
20DB: C1 18       CMPB  #$18                ; cruise missile at
top-most border?
20DD: 24 06       BCC   $20E5               ; no, goto $20E5
20DF: E0 4A       SUBB  $000A,U             ; undo change made at
$20D9
20E1: 60 4A       NEG   $000A,U             ; negate Y delta of
```

```
cruise missile (making it move in opposite direction horizontally)
20E3: 20 F4       BRA   $20D9

20E5: C1 EA       CMPB  #$EA                ; cruise missile at
bottom-most border?
20E7: 22 F6       BHI   $20DF               ; yes, undo change made
at $20D9

; A = X coordinate
; B = Y coordinate
; X = pointer to cruise missile object

DRAW_CRUISE_MISSILE:
20E9: 10 8E DD DD LDY   #$DDDD              ; pixel colours to
write (remember 4 bits per pixel)
20ED: 10 AF 98 0A STY   [$0A,X]             ; write 4 pixels to
screen RAM
20F1: ED 0A       STD   $000A,X             ; save screen
20F3: 83 01 01    SUBD  #$0101
20F6: ED 04       STD   $0004,X             ; update blitter
destination
20F8: 10 8E 00 00 LDY   #$0000              ; 4 black pixels
20FC: A6 4C       LDA   $000C,U             ; read index into
"cruise missile trail" list
20FE: 10 AF D6    STY   [A,U]               ; write black pixels to
trail (thus erasing trail)
2101: CC AA AA    LDD   #$AAAA              ; colour of cruise
missile "head"
2104: 10 AE 0A    LDY   $000A,X             ; get blitter address
of head
2107: ED A4       STD   ,Y                  ; draw cruise missle
head
2109: A6 4C       LDA   $000C,U             ; get "cruise missile
trail" list index
210B: 10 AF C6    STY   A,U                 ; save blitter address
of head of cruise missile in list
210E: 8B 02       ADDA  #$02                ; bump to next entry in
list
2110: 81 1F       CMPA  #$1F                ; at end of list buffer
(meaning cruise missile trail can't expand any more)?
2112: 25 02       BCS   $2116               ; no, goto $2116
2114: 86 0D       LDA   #$0D                ; yes, reset "cruise
missile trail" list index to beginning
2116: A7 4C       STA   $000C,U             ; update cruise missile
trail list index
2118: 39          RTS



CRUISE_MISSILE_COLLISION_HANDLER:
2119: 0A 8E       DEC   $8E                 ; reduce count of
cruise missiles on screen
211B: 96 48       LDA   $48                 ; player collision
detection?
```

```
211D: 26 21       BNE   $2140               ; yes
211F: BD D0 7E    JSR   $D07E               ; JMP $D2C2 - remove
baddy from baddies list
2122: AE 06       LDX   $0006,X             ; get pointer to object
metadata into X
2124: CE 00 00    LDU   #$0000              ; 4 black pixels
2127: 86 0D       LDA   #$0D                ; start of "cruise
missile trail" list
2129: EF 96       STU   [A,X]               ; write pixels,
deleting the "tail" of the cruise missile
                                            ; one segment at a time
212B: 8B 02       ADDA  #$02                ; bump index to next
item in "cruise missile trail" list
212D: 81 1F       CMPA  #$1F                ; have all segments of
the cruise missile's trail been erased?
212F: 26 F8       BNE   $2129               ; no, goto $2129
2131: BD D0 5D    JSR   $D05D               ; JMP $D218 -
deallocate object metadata entry
2134: CC 00 25    LDD   #$0025
2137: BD D0 0C    JSR   $D00C               ; JMP $DB9C - update
player score
213A: CC 1A D5    LDD   #$1AD5
213D: BD D0 4B    JSR   $D04B               ; JMP $D3C7
2140: 39          RTS


;
; Brain animation frame metadata. Each animation frame takes up 4
bytes as you can see.
; First two bytes are width, height of the animation frame.
; Next 2 bytes are a pointer to the actual image data.

; brain walking left #1 (this frame also used for "programming")
2141: 07 10               ; width: 7 bytes (14 pixels, remember 2
pixels per byte), height: 10 (16 decimal)  pixels
2143: 21 71               ; pointer to animation frame metadata for
walking left frame 1

; brain walking left #2
2145: 07 10               ; width: 7 bytes (14 pixels), height: 10
(16 decimal)  pixels
2147: 21 E1               ; pointer to animation frame metadata for
walking left frame 2

; brain walking left #3
2149: 07 10               ; I am sure...
214B: 22 51               ; ... you get the drift!!

; brain walking right #1 (this frame also used for "programming")
214D: 07 10
214F: 22 C1

; brain walking right #2
2151: 07 10
```

```
2153: 23 31

; brain walking right #3
2155: 07 10
2157: 23 A1

; brain moving down #1.
2159: 07 10
215B: 24 11

; brain moving down #2
215D: 07 10
215F: 24 81

; brain moving down #3
2161: 07 10
2163: 24 F1

; brain moving up #1
2165: 07 10
2167: 25 61

; brain moving up #2
2169: 07 10
216B: 25 D1

; brain moving up #3
216D: 07 10
216F: 26 41

2171: 00 00 7C 7C 70 00 00 00 0C 0C 0C 7C 00 00 00 7C
2181: 7C C0 CC 70 00 07 0C C0 C7 C0 C7 00 0C CC 7C CC
2191: C7 C7 00 07 77 7C C7 C0 CC 00 00 07 77 CC 0C C7
21A1: 00 00 7A A7 77 0C 70 00 00 77 77 7C C0 00 00 00
21B1: 07 77 70 00 00 00 00 00 06 70 00 00 00 00 66 66
21C1: 00 00 00 00 00 00 06 00 00 00 00 00 00 06 00 00
21D1: 00 00 00 04 66 00 00 00 00 00 00 00 00 00 00 00
21E1: 00 00 7C 7C 70 00 00 00 0C 0C 0C 7C 00 00 00 7C
21F1: 7C C0 CC 70 00 07 0C C0 C7 C0 C7 00 0C CC 7C CC
2201: C7 C7 00 07 77 7C C7 C0 CC 00 00 07 77 CC 0C C7
2211: 00 00 7A A7 77 0C 70 00 00 77 77 7C C0 00 00 00
2221: 07 77 70 00 00 00 00 00 06 70 00 00 00 00 66 66
2231: 00 00 00 00 00 00 06 60 00 00 00 00 40 40 06 00
2241: 00 00 00 04 00 60 00 00 00 00 00 00 00 00 00 00
2251: 00 00 7C 7C 70 00 00 00 0C 0C 0C 7C 00 00 00 7C
2261: 7C C0 CC 70 00 07 0C C0 C7 C0 C7 00 0C CC 7C CC
2271: C7 C7 00 07 77 7C C7 C0 CC 00 00 07 77 CC 0C C7
2281: 00 00 7A A7 77 0C 70 00 00 77 77 7C C0 00 00 00
2291: 07 77 70 00 00 00 00 00 06 70 00 00 00 00 66 66
22A1: 00 00 00 00 00 00 06 40 00 00 00 00 60 60 04 00
22B1: 00 00 00 06 00 40 00 00 00 00 00 00 00 00 00 00
22C1: 00 07 CC 7C 00 00 00 00 CC 7C CC 70 00 00 07 C7
22D1: 0C 70 7C 00 00 77 CC 7C 7C CC 70 00 CC 0C CC 0C
22E1: 70 70 00 7C C0 7C CC CC 70 00 77 CC CC 77 70 00
```

```
22F1: 00 0C C7 C7 7A A7 00 00 00 07 77 77 77 00 00 00
2301: 00 07 77 70 00 00 00 00 07 60 00 00 00 00 00 00
2311: 66 66 00 00 00 00 00 60 00 00 00 00 00 00 60 00
2321: 00 00 00 00 00 66 40 00 00 00 00 00 00 00 00 00
2331: 00 07 CC 7C 00 00 00 00 CC 7C CC 70 00 00 07 C7
2341: 0C 70 7C 00 00 77 CC 7C 7C CC 70 00 CC 0C CC 0C
2351: 70 70 00 7C C0 7C CC CC 70 00 77 CC CC 77 70 00
2361: 00 0C C7 C7 7A A7 00 00 00 07 77 77 77 00 00 00
2371: 00 07 77 70 00 00 00 00 07 60 00 00 00 00 00 00
2381: 66 66 00 00 00 00 06 60 00 00 00 00 00 60 04 04
2391: 00 00 00 00 06 00 40 00 00 00 00 00 00 00 00 00
23A1: 00 07 CC 7C 00 00 00 00 CC 7C CC 70 00 00 07 C7
23B1: 0C 70 7C 00 00 77 CC 7C 7C CC 70 00 CC 0C CC 0C
23C1: 70 70 00 7C C0 7C CC CC 70 00 77 CC CC 77 70 00
23D1: 00 0C C7 C7 7A A7 00 00 00 07 77 77 77 00 00 00
23E1: 00 07 77 70 00 00 00 00 07 60 00 00 00 00 00 00
23F1: 66 66 00 00 00 00 04 60 00 00 00 00 00 40 06 06
2401: 00 00 00 00 04 00 60 00 00 00 00 00 00 00 00 00
2411: 00 00 0C 70 00 00 00 00 7C C7 7C 7C 70 00 07 CC
2421: 0C CC 0C 77 00 77 C7 C7 C0 C0 CC 70 CC C0 CC C7
2431: CC 0C 70 C7 C7 7C 0C 77 C7 C0 70 C7 77 C7 70 7C
2441: 70 0C CA AA 7A AA 77 00 07 77 77 77 77 70 00 00
2451: 00 07 77 00 00 00 00 00 07 67 00 00 00 00 00 66
2461: 66 60 00 00 00 06 00 60 06 00 00 00 00 06 06 00
2471: 00 00 00 00 66 06 60 00 00 00 00 00 00 00 00 00
2481: 00 00 0C 70 00 00 00 00 7C C7 7C 7C 70 00 07 CC
2491: 0C CC 0C 77 00 77 C7 C7 C0 C0 CC 70 CC C0 CC C7
24A1: CC 0C 70 C7 C7 7C 0C 77 C7 C0 70 C7 77 C7 70 7C
24B1: 70 0C CA AA 7A AA 77 00 07 77 77 77 77 70 00 00
24C1: 00 07 77 00 00 00 00 00 07 67 00 00 00 00 00 66
24D1: 66 60 00 00 00 06 00 66 06 00 00 00 00 06 06 60
24E1: 00 00 00 00 06 00 00 00 00 00 00 66 00 00 00 00
24F1: 00 00 0C 70 00 00 00 00 7C C7 7C 7C 70 00 07 CC
2501: 0C CC 0C 77 00 77 C7 C7 C0 C0 CC 70 CC C0 CC C7
2511: CC 0C 70 C7 C7 7C 0C 77 C7 C0 70 C7 77 C7 70 7C
2521: 70 0C CA AA 7A AA 77 00 07 77 77 77 77 70 00 00
2531: 00 07 77 00 00 00 00 00 07 67 00 00 00 00 00 66
2541: 66 60 00 00 00 06 06 60 06 00 00 00 00 66 06 00
2551: 00 00 00 00 00 06 00 00 00 00 00 00 06 60 00 00
2561: 00 00 07 C0 00 00 00 00 7C 70 C7 C7 70 00 0C 70
2571: C7 CC C7 C7 00 7C 00 C7 0C 0C 07 70 7C C7 7C CC
2581: 7C 7C C0 77 7C C0 C0 C7 C7 70 77 00 CC 0C 07 C7
2591: 70 07 7C 77 CC CC 77 00 00 C7 70 C7 77 70 00 00
25A1: 00 07 77 00 00 00 00 00 07 67 00 00 00 00 00 66
25B1: 66 60 00 00 00 06 00 60 06 00 00 00 00 06 06 00
25C1: 00 00 00 00 66 06 60 00 00 00 00 00 00 00 00 00
25D1: 00 00 07 C0 00 00 00 00 7C 70 C7 C7 70 00 0C 70
25E1: C7 CC C7 C7 00 7C 00 C7 0C 0C 07 70 7C C7 7C CC
25F1: 7C 7C C0 77 7C C0 C0 C7 C7 70 77 00 CC 0C 07 C7
2601: 70 07 7C 77 CC CC 77 00 00 C7 70 C7 77 70 00 00
2611: 00 07 77 00 00 00 00 00 07 67 00 00 00 00 00 66
2621: 66 60 00 00 00 06 00 66 06 00 00 00 00 06 06 60
2631: 00 00 00 00 06 00 00 00 00 00 00 66 00 00 00 00
2641: 00 00 07 C0 00 00 00 00 7C 70 C7 C7 70 00 0C 70
```

```
2651: C7 CC C7 C7 00 7C 00 C7 0C 0C 07 70 7C C7 7C CC
2661: 7C 7C C0 77 7C C0 C0 C7 C7 70 77 00 CC 0C 07 C7
2671: 70 07 7C 77 CC CC 77 00 00 C7 70 C7 77 70 00 00
2681: 00 07 77 00 00 00 00 00 07 67 00 00 00 00 00 66
2691: 66 60 00 00 00 06 06 60 06 00 00 00 00 66 06 00
26A1: 00 00 00 00 00 06 00 00 00 00 00 00 06 60 00 00
26B1: 55 55 55 55 55 55 55 55 FF FF FF FF FF FF FF

26C0: 7E 2F C2    JMP   $2FC2

26C3: 7E 31 99    JMP   $3199

26C6: 7E 30 85    JMP   $3085

26C9: 7E 34 E0    JMP   $34E0

26CC: 7E 26 FC    JMP   $26FC

26CF: 7E 26 F5    JMP   $26F5

26D2: 7E 34 AF    JMP   $34AF

26D5: 35 EB       PULS  CC,A,DP,Y,U,PC ;(PUL? PC=RTS)

26D7: 35 AE       PULS  A,B,DP,Y,PC ;(PUL? PC=RTS)

26D9: EE 02       LDU   $0002,X
26DB: 08 11       ASL   $11
26DD: 01          Illegal Opcode
26DE: 20 17       BRA   $26F7

26E0: 00 F0       NEG   $F0
26E2: 01          Illegal Opcode
26E3: 10 28 00 F0 LBVC  $27D7
26E7: 01          Illegal Opcode
26E8: 10 25 00 E0 LBCS  $27CC
26EC: 1D          SEX
26ED: 04 0E       LSR   $0E
26EF: 00 D0       NEG   $D0
26F1: 01          Illegal Opcode
26F2: 08 01       ASL   $01
26F4: 00 86       NEG   $86
26F6: 02          Illegal Opcode
26F7: 20 05       BRA   $26FE

26F9: 7E D0 63    JMP   $D063               ; JMP $D1F3

26FC: 86 01       LDA   #$01
26FE: 10 8E C3 B3 LDY   #$C3B3
2702: 0D 59       TST   $59
2704: 2A F3       BPL   $26F9
2706: F6 CC 05    LDB   $CC05
2709: C4 0F       ANDB  #$0F
270B: C1 09       CMPB  #$09
```

```
270D: 26 02       BNE   $2711
270F: 97 51       STA   $51
2711: 91 51       CMPA  $51
2713: 22 E4       BHI   $26F9
2715: 97 40       STA   $40
2717: 0F 4F       CLR   $4F
2719: 0F 50       CLR   $50
271B: C6 08       LDB   #$08
271D: BD D0 BA    JSR   $D0BA
2720: CE D0 15    LDU   #$D015
2723: 40          NEGA
2724: 8B 9A       ADDA  #$9A
2726: 9B 51       ADDA  $51
2728: 19          DAA
2729: 97 51       STA   $51
272B: 8E CD 00    LDX   #credits_cmos
272E: BD D0 AB    JSR   STA_NIB_X1
2731: 96 40       LDA   $40
2733: 80 02       SUBA  #$02
2735: CC 26 E1    LDD   #$26E1
2738: 25 03       BCS   $273D
273A: CC 26 E6    LDD   #$26E6
273D: BD D0 4B    JSR   $D04B               ; JMP $D3C7
2740: 4F          CLRA
2741: AB C4       ADDA  ,U
2743: 33 48       LEAU  $0008,U
2745: 11 83 EA B1 CMPU  #$EAB1
2749: 25 F6       BCS   $2741
274B: A7 A9 FA BF STA   $FABF,Y
274F: 86 7F       LDA   #$7F
2751: 97 59       STA   $59
2753: BD D0 12    JSR   CLR_SCREEN1
2756: 8E BD E4    LDX   #p1_score           ; set score to 0
2759: 6F 80       CLR   ,X+
275B: 8C BE 5C    CMPX  #$BE5C
275E: 26 F9       BNE   $2759
2760: 8E CC 02    LDX   #$CC02              ; address of value in
CMOS
2763: BD D0 A2    JSR   LDA_NIB_X1          ; pack 2 bytes into 1
2766: BD D0 C6    JSR   $D0C6               ; JMP $D5D8 - convert
from BCD to normal number
2769: B7 BD EC    STA   p1_men
276C: 86 01       LDA   #$01
276E: 97 3F       STA   $3F                 ; player 1
2770: 86 01       LDA   #$01
2772: B7 BD ED    STA   p1_wave             ; wave 1
2775: 97 48       STA   $48                 ; used in the collision
detection function. I thought it was a player hit flag, but it's
not.
2777: 8E CC 00    LDX   #$CC00              ; address of value in
CMOS
277A: BD D0 A2    JSR   LDA_NIB_X1          ; pack 2 bytes into 1
277D: 5F          CLRB
277E: 44          LSRA
```

```
277F: 56          RORB
2780: 44          LSRA
2781: 56          RORB
2782: 44          LSRA
2783: 56          RORB
2784: 44          LSRA
2785: 56          RORB
2786: DD 46       STD   $46
2788: FD BD E9    STD   $BDE9
278B: BD 2B 7C    JSR   $2B7C
278E: 8E BD E4    LDX   #p1_score
2791: A6 80       LDA   ,X+
2793: A7 88 3B    STA   $3B,X
2796: 8C BE 20    CMPX  #p2_score
2799: 26 F6       BNE   $2791
279B: 96 40       LDA   $40
279D: 4A          DECA
279E: 26 03       BNE   $27A3
27A0: 7F BE 28    CLR   p2_men
27A3: 86 7F       LDA   #$7F
27A5: 97 59       STA   $59
27A7: BD D0 45    JSR   $D045               ; JMP $D699 - get addr
of current player game state into X
27AA: 6A 08       DEC   $0008,X             ; reduce player lives
left
27AC: BD D0 12    JSR   CLR_SCREEN1
27AF: BD D0 60    JSR   $D060               ; JMP $D1FF
27B2: AE 9F 98 11 LDX   [$9811,X]
27B6: 26 0B       BNE   $27C3
27B8: BD D0 36    JSR   $D036               ; JMP $D6EC
27BB: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
27BE: 27 C3       ; pointer to function
27C0: 7E D0 96    JMP   $D096               ; JMP $D196

27C3: BD D0 33    JSR   LOAD_DA51_PALETTE1
27C6: BD 5B 40    JSR   $5B40               ; clear explosion list
27C9: BD D0 30    JSR   $D030
27CC: BD 2A 21    JSR   $2A21
27CF: BD D0 99    JSR   FLIP_SCR_UP1
27D2: B6 C8 06    LDA   widget_pia_datab
27D5: 2A 08       BPL   $27DF
27D7: 96 3F       LDA   $3F                 ; read player number
27D9: 4A          DECA                      ; reduce by 1
27DA: 27 03       BEQ   $27DF
27DC: BD D0 9C    JSR   FLIP_SCR_DOWN1      ; if player number is
not 0, it must be player 2's turn. Flip screen (for cocktail
cabinets)
27DF: BD 34 AF    JSR   $34AF
27E2: BD D0 24    JSR   $D024
27E5: 0F F2       CLR   $F2
27E7: 96 48       LDA   $48
27E9: 26 05       BNE   $27F0
27EB: BD 34 C0    JSR   $34C0
```

```
27EE: 20 36       BRA   $2826

27F0: FC BD E5    LDD   $BDE5
27F3: 26 0B       BNE   $2800
27F5: 86 11       LDA   #$11
27F7: 97 CF       STA   $CF
27F9: 86 71       LDA   #$71
27FB: BD 5F 96    JSR   $5F96               ; JMP $613F: print
string in small font
27FE: 20 03       BRA   $2803

2800: BD 34 C0    JSR   $34C0               ; PRINT_WAVE_NUMBER
2803: 0F 48       CLR   $48
2805: 96 40       LDA   $40
2807: 4A          DECA
2808: 27 1C       BEQ   $2826
280A: 96 90       LDA   $90
280C: 97 CF       STA   $CF
280E: D6 3F       LDB   $3F                 ; read player number
2810: 86 67       LDA   #$67
2812: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
2815: 86 73       LDA   #$73
2817: 8E 28 1D    LDX   #$281D
281A: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

281D: 0F CF       CLR   $CF
281F: D6 3F       LDB   $3F
2821: 86 67       LDA   #$67
2823: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT


BEGIN_WAVE:
2826: BD 2B 0B    JSR   $2B0B               ; Read in object counts
for this wave (how many grunts, hulks, family members etc etc)

; U = #$BE72
2829: BD 2F 8C    JSR   $2F8C               ; WAVE_START_PLAYER
282C: 0F 06       CLR   $06
282E: 7F C0 06    CLR   $C006                ; clear a colour
register.
2831: BD 00 00    JSR   $0000               ; JMP $016D -
initialise all hulks
2834: BD 1A C0    JSR   $1AC0               ; JMP $1AF4 -
initialise all brains
2837: BD 4B 00    JSR   $4B00               ; JMP $4D10 -
initialise all tanks
283A: BD 00 03    JSR   $0003               ; JMP $02B2 -
initialise all family members
283D: BD 38 83    JSR   $3883               ; JMP $3950 -
initialise all electrodes
2840: BD 38 80    JSR   $3880               ; JMP $38AA -
initialise all grunts
2843: BD 29 A4    JSR   $29A4               ; clear all baddies
```

```
from screen
2846: 86 08       LDA   #$08
2848: 97 92       STA   $92
284A: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
284D: 35 4B       ; pointer to function
284F: BD 29 8F    JSR   $298F               ; draw all electrodes
2852: BD D0 33    JSR   LOAD_DA51_PALETTE1
2855: BD 11 40    JSR   $1140               ; JMP $1168 -
initialise all spheroids
2858: BD 4B 03    JSR   $4B03               ; JMP $4B36 -
initialise all quarks
285B: 86 19       LDA   #$19
285D: 97 59       STA   $59
285F: B6 BE 6E    LDA   cur_brains          ; any brains on this
wave?
2862: 27 0D       BEQ   $2871               ; no, goto $2871

; ok, this is where the cool brain warping in fx are done
2864: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
2867: 41 40       ; pointer to function
2869: 86 96       LDA   #$96                ; delay before calling
function
286B: 8E 28 74    LDX   #$2874              ; address of function
to call (brain wave related)
286E: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

2871: 7E 28 FE    JMP   $28FE

;
; This code is related to the brain wave.
;
2874: BD 29 F5    JSR   $29F5
2877: BD 29 8F    JSR   $298F
287A: 86 06       LDA   #$06
287C: 8E 28 82    LDX   #$2882
287F: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

2882: BD 29 D2    JSR   $29D2
2885: 86 04       LDA   #$04
2887: 8E 28 8D    LDX   #$288D
288A: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

288D: BD 29 83    JSR   $2983
2890: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
2893: 31 B5       ; pointer to function
2895: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
2898: 30 B3       ; pointer to function
```

```
289A: 0F 59       CLR   $59
289C: 86 0C       LDA   #$0C
289E: 8E 28 A4    LDX   #$28A4
28A1: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

28A4: BD 29 83    JSR   $2983
28A7: BD 29 8F    JSR   $298F
28AA: 86 0A       LDA   #$0A
28AC: 8E 28 B2    LDX   #$28B2
28AF: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

28B2: BD 29 83    JSR   $2983
28B5: BD 29 8F    JSR   $298F
28B8: 86 04       LDA   #$04
28BA: 97 92       STA   $92
28BC: 0F CF       CLR   $CF
28BE: 86 71       LDA   #$71                ; clear the "(C) 1982
Williams Electronics" message at the bottom
28C0: BD 5F 96    JSR   $5F96               ; JMP $613F: print
string in small font
28C3: BD 34 C0    JSR   $34C0
28C6: 8E 20 FB    LDX   #$20FB
28C9: CE 2C 03    LDU   #$2C03
28CC: A6 C0       LDA   ,U+
28CE: 88 5A       EORA  #$5A
28D0: 27 05       BEQ   $28D7
28D2: BD 5F 90    JSR   $5F90
28D5: 20 F5       BRA   $28CC

28D7: 7E 2A 85    JMP   $2A85

28DA:  (C) 1982 WILLIAMS ELECTRONICS I
28FA: NC.

28FE: DE 15       LDU   $15
2900: 6F 47       CLR   $0007,U
2902: 8E 98 21    LDX   #$9821              ; pointer to baddies
list start
2905: AF 49       STX   $0009,U
2907: 9E 21       LDX   $21                 ; read first entry in
baddies list
2909: AF 4B       STX   $000B,U
290B: AF 4D       STX   $000D,U
290D: 86 01       LDA   #$01
290F: 34 02       PSHS  A
2911: AE 49       LDX   $0009,U
2913: AE 84       LDX   ,X
2915: 27 15       BEQ   $292C
2917: BD 29 B5    JSR   $29B5
291A: A6 47       LDA   $0007,U
291C: 84 03       ANDA  #$03
291E: 81 03       CMPA  #$03
```

```
2920: 26 05       BNE   $2927
2922: BD 5B 58    JSR   $5B58
2925: 20 03       BRA   $292A

2927: BD 5B 46    JSR   $5B46               ; JMP $5BC6
292A: AF 49       STX   $0009,U
292C: 6C 47       INC   $0007,U
292E: A6 47       LDA   $0007,U
2930: 81 20       CMPA  #$20
2932: 23 0D       BLS   $2941
2934: 10 AE 4D    LDY   $000D,U
2937: 27 18       BEQ   $2951
2939: 10 AE A4    LDY   ,Y
293C: 27 13       BEQ   $2951
293E: 10 AF 4D    STY   $000D,U
2941: 6A E4       DEC   ,S
2943: 26 CC       BNE   $2911
2945: 35 02       PULS  A
2947: 8D 1C       BSR   $2965
2949: 86 01       LDA   #$01
294B: 8E 29 0D    LDX   #$290D
294E: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

2951: 35 02       PULS  A
2953: 86 02       LDA   #$02
2955: 8E 29 5B    LDX   #$295B
2958: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

295B: 8D 26       BSR   $2983
295D: 86 0A       LDA   #$0A
295F: 8E 28 74    LDX   #$2874
2962: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

2965: 34 10       PSHS  X
2967: 86 04       LDA   #$04
2969: 97 2B       STA   $2B
296B: AE 4B       LDX   $000B,U
296D: AC 4D       CPX   $000D,U
296F: 26 05       BNE   $2976
2971: 8E 98 21    LDX   #$9821              ; pointer to baddies
list start
2974: 20 03       BRA   $2979

2976: BD D0 18    JSR   $D018               ; JMP $DAF2 - do blit
2979: AE 84       LDX   ,X
297B: 0A 2B       DEC   $2B
297D: 26 EE       BNE   $296D
297F: AF 4B       STX   $000B,U
2981: 35 90       PULS  X,PC ;(PUL? PC=RTS)
```

```
DRAW_ALL_GRUNTS_HULKS_BRAINS_PROGS_CRUISE_TANKS:
2983: 9E 21       LDX   $21                  ; pointer to list start
2985: 27 07       BEQ   $298E
2987: BD D0 18    JSR   $D018                ; JMP   $DAF2 - do blit
298A: AE 84       LDX   ,X
298C: 26 F9       BNE   $2987
298E: 39          RTS


DRAW_ALL_ELECTRODES:
298F: 9E 23       LDX   $23                  ; pointer to electrodes
list start
2991: 27 0C       BEQ   $299F
2993: 96 90       LDA   $90                  ; load electrode colour
2995: BD 38 88    JSR   $3888                ; JMP   $3942 - do
solid & transparent blit
2998: AF 98 06    STX   [$06,X]
299B: AE 84       LDX   ,X
299D: 26 F4       BNE   $2993
299F: 39          RTS


CLEAR_ALL_ELECTRODES:
29A0: 9E 23       LDX   $23                  ; pointer to electrodes
list start
29A2: 20 02       BRA   $29A6


CLEAR_ALL_GRUNTS_HULKS_BRAINS_PROGS_CRUISE_TANKS:
29A4: 9E 21       LDX   $21                  ; pointer to baddies
list start
29A6: 27 0C       BEQ   $29B4
29A8: EC 04       LDD   $0004,X              ; D = "last" blitter
destination
29AA: 10 AE 02    LDY   $0002,X              ; Y = pointer to
animation frame metadata
29AD: BD D0 1E    JSR   $D01E                ; JMP $DABF: clear
image rectangle
29B0: AE 84       LDX   ,X                   ; X = next object in
chain
29B2: 26 F4       BNE   $29A8                ; if X!=NULL then $29A8
29B4: 39          RTS


29B5: 34 46       PSHS  U,B,A
29B7: EE 02       LDU   $0002,X
29B9: E6 C4       LDB   ,U
29BB: A6 04       LDA   $0004,X
29BD: 48          ASLA
29BE: 24 02       BCC   $29C2
29C0: 86 FF       LDA   #$FF
29C2: 3D          MUL
29C3: AB 04       ADDA  $0004,X
29C5: 97 A6       STA   $A6
```

```
29C7: E6 41       LDB   $0001,U
29C9: A6 05       LDA   $0005,X
29CB: 3D          MUL
29CC: AB 05       ADDA  $0005,X
29CE: 97 A7       STA   $A7
29D0: 35 C6       PULS  A,B,U,PC ;(PUL? PC=RTS)

29D2: 8E 98 5A    LDX   #$985A              ; player_object_start
29D5: 10 AE 02    LDY   $0002,X             ; animation frame
metadata pointer
29D8: A6 21       LDA   $0001,Y
29DA: 34 02       PSHS  A
29DC: A6 E4       LDA   ,S
29DE: 9B 5F       ADDA  $5F
29E0: 4A          DECA
29E1: 97 A7       STA   $A7
29E3: 4F          CLRA
29E4: BD 46 86    JSR   $4686
29E7: 43          COMA
29E8: BD 46 86    JSR   $4686
29EB: A6 E4       LDA   ,S
29ED: 80 03       SUBA  #$03
29EF: A7 E4       STA   ,S
29F1: 2A E9       BPL   $29DC
29F3: 35 82       PULS  A,PC ;(PUL? PC=RTS)

29F5: 8E 98 5A    LDX   #$985A              ; player_object_start
29F8: 10 AE 02    LDY   $0002,X             ; get pointer to
animation frame metadata for player
29FB: A6 21       LDA   $0001,Y             ; A = height of
animation frame
29FD: 34 02       PSHS  A
29FF: A6 E4       LDA   ,S
2A01: 9B 5F       ADDA  $5F
2A03: 4A          DECA
2A04: 97 A7       STA   $A7
2A06: BD 5B 46    JSR   $5B46               ; JMP $5BC6
2A09: 6A E4       DEC   ,S
2A0B: 26 F2       BNE   $29FF
2A0D: A6 A4       LDA   ,Y
2A0F: A7 E4       STA   ,S
2A11: A6 E4       LDA   ,S
2A13: 9B 5E       ADDA  $5E
2A15: 4A          DECA
2A16: 97 A6       STA   $A6
2A18: BD 5B 58    JSR   $5B58
2A1B: 6A E4       DEC   ,S
2A1D: 26 F2       BNE   $2A11
2A1F: 35 82       PULS  A,PC ;(PUL? PC=RTS)

2A21: BD D0 45    JSR   $D045               ; JMP $D699 - get addr
of current player game state into X
2A24: CE 2A 4B    LDU   #$2A4B
2A27: A6 09       LDA   $0009,X
```

```
2A29: 4A          DECA
2A2A: 81 09       CMPA  #$09
2A2C: 23 04       BLS   $2A32
2A2E: 80 0A       SUBA  #$0A
2A30: 20 F8       BRA   $2A2A

2A32: 33 C6       LEAU  A,U
2A34: A6 C4       LDA   ,U
2A36: 97 8F       STA   $8F
2A38: A6 4A       LDA   $000A,U
2A3A: 97 90       STA   $90
2A3C: E6 C8 14    LDB   $14,U
2A3F: BE 38 86    LDX   $3886
2A42: 3A          ABX
2A43: 9F 93       STX   $93
2A45: A6 C8 1E    LDA   $1E,U
2A48: 97 91       STA   $91
2A4A: 39          RTS

2A4B: 22 55 11 EE 77 33 44 88 00 CC FF EE BB DD EE FF
2A5B: 11 BB DD AA 00 10 20 30 40 50 70 80 00 60 99 00
2A6B: 99 66 99 99 99 11 AA 99

2A73: B6 BE 68    LDA   cur_grunts
2A76: BB BE 6F    ADDA  cur_sphereoids
2A79: 9B ED       ADDA  temp_enforcer_count
2A7B: BB BE 6E    ADDA  cur_brains
2A7E: BB BE 71    ADDA  cur_tanks
2A81: BB BE 70    ADDA  cur_quarks
2A84: 39          RTS

2A85: DE 15       LDU   $15
2A87: 86 12       LDA   #$12
2A89: A7 47       STA   $0007,U
2A8B: 0F F0       CLR   $F0
2A8D: 8D E4       BSR   $2A73
2A8F: 26 24       BNE   $2AB5
2A91: BD D0 45    JSR   $D045               ; JMP $D699 - get addr
of current player game state into X
2A94: 6C 09       INC   $0009,X
2A96: 26 02       BNE   $2A9A
2A98: 6C 09       INC   $0009,X
2A9A: 6C 08       INC   $0008,X
2A9C: CC 26 EB    LDD   #$26EB
2A9F: BD D0 4B    JSR   $D04B               ; JMP $D3C7
2AA2: BD 2B 7C    JSR   $2B7C
2AA5: BD D0 60    JSR   $D060               ; JMP $D1FF
2AA8: 86 7F       LDA   #$7F
2AAA: 97 59       STA   $59
2AAC: BD D0 12    JSR   CLR_SCREEN1
2AAF: BD 57 00    JSR   $5700
2AB2: 7E 27 A3    JMP   $27A3
```

```
;
; Easter egg code
;

EASTER_EGG_PART_1:
2AB5: B6 C8 04    LDA   widget_pia_dataa
2AB8: 81 58       CMPA  #$58                ; move right + 1p +
fire up
2ABA: 26 03       BNE   $2ABF
2ABC: BD D0 69    JSR   $D069               ; JMP D30E
2ABF: 6A 47       DEC   $0007,U
2AC1: 26 31       BNE   $2AF4
2AC3: 86 0F       LDA   #$0F
2AC5: A7 47       STA   $0007,U
; Update grunt speed as level progresses
2AC7: B6 BE 68    LDA   cur_grunts
2ACA: 81 1E       CMPA  #$1E
2ACC: 24 26       BCC   $2AF4
2ACE: CC FF FE    LDD   #$FFFE
2AD1: 0D F0       TST   $F0
2AD3: 26 03       BNE   $2AD8
2AD5: CC FE FC    LDD   #$FEFC
2AD8: BB BE 5D    ADDA  $BE5D
2ADB: 0F F0       CLR   $F0
2ADD: 81 01       CMPA  #$01
2ADF: 2C 02       BGE   $2AE3
2AE1: 86 01       LDA   #$01
2AE3: B7 BE 5D    STA   $BE5D
2AE6: FB BE 5C    ADDB  $BE5C
2AE9: F1 BE 5D    CMPB  $BE5D               ; compare to grunt
speed throttle setting
2AEC: 2C 03       BGE   $2AF1
2AEE: F6 BE 5D    LDB   $BE5D               ; set grunt movement
speed to the throttle setting, which is the fastest the grunts can
move for this level
2AF1: F7 BE 5C    STB   $BE5C               ; update grunt movement
speed
2AF4: 96 EE       LDA   $EE
2AF6: 4C          INCA
2AF7: 81 96       CMPA  #$96
2AF9: 25 06       BCS   $2B01
2AFB: C6 06       LDB   #$06
2AFD: BD D0 BD    JSR   $D0BD               ; JMP $D655
2B00: 4F          CLRA
2B01: 97 EE       STA   $EE
2B03: 86 0F       LDA   #$0F
2B05: 8E 2A 8D    LDX   #$2A8D
2B08: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call


; Set enemies

SET_WAVE_OBJECT_COUNTS:
```

```
2B0B: BD D0 45    JSR   $D045               ; JMP $D699 - get addr
of current player game state into X
2B0E: 30 0A       LEAX  $000A,X             ; X+=10. Points to
$BDEE
2B10: CE BE 5C    LDU   #$BE5C
2B13: A6 80       LDA   ,X+                 ; read byte from x+
2B15: A7 C0       STA   ,U+                 ; store byte at u+
2B17: 11 83 BE 72 CMPU  #$BE72
2B1B: 26 F6       BNE   $2B13

; wonder why they didn't just push X on the stack at $2B0B and pop
it off here instead of calling function to set X?
2B1D: BD D0 45    JSR   $D045               ; JMP $D699 - get addr
of current player game state into X
2B20: A6 09       LDA   $0009,X             ; read wave number
2B22: 81 04       CMPA  #$04                ; compare wave number
to 4
2B24: 22 32       BHI   $2B58               ; wave number > 4? If
so goto $2B58, which exits function
2B26: E6 08       LDB   $0008,X             ; B = number of lives
left
2B28: DD 2B       STD   $2B                 ; save wave number &
lives left
2B2A: 5D          TSTB                      ; lives left == 0?
2B2B: 27 12       BEQ   $2B3F               ; yes, goto $2B3F - the
player must be having a shocker of a game
2B2D: 81 02       CMPA  #$02                ; wave number == 2 ?
2B2F: 22 27       BHI   $2B58               ; if >2 then exit
;
; OK, we're in wave 1 or 2. The following code reads the number of
lives the player has left and determines
; if the number is the same as or higher as the "turns per player"
setting in the CMOS.
; If the number is the same or higher, then the player's doing OK
and the routine exits.
; Otherwise the player's not doing too great, so some remedial
action is needed - the enforcer and grunts
; speed settings are changed.
;
; Here's Larry Demar's comments after I raised this discovery:
; <BEGIN QUOTE>
; "The Bozo mode was not in the original Robotron code. There were
some complaints from the field about how brutal
; the game was to new players and Scott has discovered the code put
in to address these complaints.
; In that 2nd release the default difficulty was also moved down
from 5 to 3.
; If you are down a man very early in the game then the settings are
dialed as low as they can go through the first wave.
; <END QUOTE>
;
2B31: 8E CC 02    LDX   #$CC02              ; address of turns per
player setting in CMOS
2B34: BD D0 A2    JSR   LDA_NIB_X1          ; convert 2 bytes to
```

```
BCD
2B37: BD D0 C6    JSR   $D0C6               ; JMP $D5D8 - convert
from BCD to normal number
2B3A: 4A          DECA
2B3B: 91 2C       CMPA  $2C                 ; compare to lives left
2B3D: 23 19       BLS   $2B58               ; if turns per player
<= player lives left, player's doing OK, goto $2b58 and exit
2B3F: 96 2B       LDA   $2B                 ; read wave number
2B41: 8E 2B 55    LDX   #$2B55              ; $2B55 is start of
remedial action table - 4 bytes.
2B44: 48          ASLA                      ;
2B45: 48          ASLA                      ; multiplies A by 4
2B46: 30 86       LEAX  A,X                 ; X = X + A
2B48: EC 81       LDD   ,X++
2B4A: B7 BE 60    STA   $BE60               ; set enforcer spawn
control variable
2B4D: F7 BE 5F    STB   $BE5F               ; set enforcer spark
control variable
2B50: EC 84       LDD   ,X
2B52: B7 BE 5C    STA   $BE5C               ; set grunt delay
control variable
2B55: F7 BE 5D    STB   $BE5D               ; set grunt delay
throttle control variable
2B58: 39          RTS



;
; Settings to make game a bit easier
; Nicknamed "BOZO MODE" apparently.
;
BOZO_MODE_TABLE:
2B59: 26 60 1E 0F
2B5D: 26 60 19 0C
2B61: 24 30 14 0A
2B65: 1E 1E 0F 07


2B69: BD D0 45    JSR   $D045
2B6C: 30 0A       LEAX  $000A,X
2B6E: CE BE 5C    LDU   #$BE5C
2B71: A6 C0       LDA   ,U+
2B73: A7 80       STA   ,X+
2B75: 11 83 BE 72 CMPU  #$BE72
2B79: 26 F6       BNE   $2B71
2B7B: 39          RTS



2B7C: 8E CC 14    LDX   #$CC14
2B7F: BD D0 A5    JSR   $D0A5
2B82: BD D0 B4    JSR   $D0B4
2B85: BD D0 45    JSR   $D045               ; JMP $D699  - get addr
of current player game state into X
```

```
2B88: C1 05       CMPB  #$05
2B8A: 24 14       BCC   $2BA0
2B8C: A6 09       LDA   $0009,X             ; read wave number
2B8E: 81 0E       CMPA  #$0E
2B90: 25 02       BCS   $2B94
2B92: C6 05       LDB   #$05
2B94: 81 05       CMPA  #$05
2B96: 25 08       BCS   $2BA0
2B98: A6 08       LDA   $0008,X             ; read lives left
2B9A: 81 03       CMPA  #$03
2B9C: 25 02       BCS   $2BA0
2B9E: C6 05       LDB   #$05
2BA0: C0 05       SUBB  #$05
2BA2: D7 2C       STB   $2C
2BA4: 2A 01       BPL   $2BA7
2BA6: 50          NEGB
2BA7: D7 2B       STB   $2B
2BA9: E6 09       LDB   $0009,X             ; read wave number
2BAB: CE 2C 22    LDU   #$2C22
2BAE: 30 0A       LEAX  $000A,X
2BB0: C1 28       CMPB  #$28
2BB2: 23 04       BLS   $2BB8
2BB4: C0 14       SUBB  #$14
2BB6: 20 F8       BRA   $2BB0

2BB8: 11 83 2E 24 CMPU  #$2E24
2BBC: 25 06       BCS   $2BC4
2BBE: 33 5D       LEAU  $FFFD,U
2BC0: A6 C5       LDA   B,U
2BC2: 20 31       BRA   $2BF5

2BC4: A6 C5       LDA   B,U
2BC6: 34 06       PSHS  B,A
2BC8: E6 5E       LDB   $FFFE,U
2BCA: C4 1F       ANDB  #$1F
2BCC: 96 2B       LDA   $2B
2BCE: 3D          MUL
2BCF: 35 02       PULS  A
2BD1: 3D          MUL
2BD2: 89 00       ADCA  #$00
2BD4: D6 2C       LDB   $2C
2BD6: E8 5E       EORB  $FFFE,U
2BD8: 35 04       PULS  B
2BDA: 2A 09       BPL   $2BE5
2BDC: 40          NEGA
2BDD: 27 06       BEQ   $2BE5
2BDF: AB C5       ADDA  B,U
2BE1: 25 06       BCS   $2BE9
2BE3: 20 08       BRA   $2BED

2BE5: AB C5       ADDA  B,U
2BE7: 25 0A       BCS   $2BF3
2BE9: A1 5F       CMPA  $FFFF,U
2BEB: 24 02       BCC   $2BEF
```

```
2BED: A6 5F       LDA   $FFFF,U
2BEF: A1 C4       CMPA  ,U
2BF1: 23 02       BLS   $2BF5
2BF3: A6 C4       LDA   ,U
2BF5: A7 80       STA   ,X+
2BF7: 33 C8 2B    LEAU  $2B,U
2BFA: 11 83 2F 8B CMPU  #$2F8B
2BFE: 25 B8       BCS   $2BB8
2C00: 6F 80       CLR   ,X+
2C02: 39          RTS

2C03: 01 19 06 7A 6B 63 62 68 7A 0D 13 16 16 13 1B

2C12: 17
2C13: 09 7A 1F 16 1F 19 67 7A 13 14 19 67 5A 8E 0A 14
2C23: 14 0F 0F 0F 0F 0F 0F 0F 0F 0F 0E 0E 0E 0E 0E 0D
2C33: 0D 0D 0D 0D 0E 0E 0E 0E 0E 0E 0D 0D 0D 0D 0D 0D
2C43: 0C 0C 0C 0C 0C 0C 0F 0C 8E 03 0A 09 07 06 05 05
2C53: 05 05 04 04 04 04 04 04 04 04 04 04 04 04 04 04
2C63: 04 04 04 04 04 03 03 04 03 03 03 03 03 03 03 03
2C73: 03 04 03 0E 08 0C 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A
2C83: 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A 0B 0B 0B 0B 0B 0B
2C93: 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 8E 0D
2CA3: 28 1E 1C 1A 18 16 14 12 12 10 0E 0E 0E 0E 0E 0E
2CB3: 0E 0E 0E 0E 0E 0F 0F 0F 0F 0F 0F 0F 0F 0F 0E 0E
2CC3: 0E 0E 0E 0E 0E 0E 0E 0E 0E 8E 0C 28 1E 1C 1A 18
2CD3: 1E 14 12 10 12 19 0C 0C 0C 19 19 0C 0C 0C 12 14
2CE3: 0E 0E 0E 0E 0E 19 0E 0E 12 19 0C 0C 0C 0C 19 0C
2CF3: 0C 0C 12 14 8E 05 09 08 08 07 07 07 07 07 06 06
2D03: 06 06 05 05 05 05 05 05 05 05 05 05 05 05 05 05
2D13: 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 8E
2D23: 19 50 40 40 40 40 40 28 28 26 26 26 26 26 26 26
2D33: 26 26 24 24 24 24 20 20 20 20 20 20 20 1E 1E 1E
2D43: 1E 1E 19 19 19 19 19 19 19 19 8E 06 0A 08 08 08
2D53: 08 08 07 07 07 07 07 07 07 07 07 07 06 06 06 06
2D63: 06 06 06 06 06 06 06 06 06 06 06 06 06 06 06 06
2D73: 06 06 06 06 06 8E 14 28 20 20 20 20 20 20 20 1E
2D83: 1E 1E 1E 1E 1E 1C 1C 1C 1C 1C 1C 1C 1E 1E 1E 1E
2D93: 1E 1E 1C 1C 1C 1C 1C 1A 1A 1A 1A 1A 18 18 18 18
2DA3: 0E A0 FF B0 B0 B0 B0 B0 B0 B0 B0 B0 B0 B0 B0 B0
2DB3: B0 B0 B0 B0 B0 B0 B0 B8 B8 B8 B8 B8 B8 B8 B8 B8
2DC3: B8 C0 C0 C0 C0 C0 C0 C0 C0 C0 C0 8E 0C 30 10 10
2DD3: 10 10 10 10 10 10 10 10 10 10 10 10 0F 0F 0F 0F
2DE3: 0F 0F 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E
2DF3: 0E 0E 0E 0E 0E 0E 0E 28 44 32 32 32 32 32 32 32
2E03: 32 32 32 32 32 38 38 38 38 38 38 38 38 38 38 38
2E13: 38 38 38 38 38 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C 3C
2E23: 3C 0F 11 16 22 14 20 00 23 3C 19 23 00 23 1B 19
2E33: 23 00 23 46 19 23 00 23 00 19 23 00 23 4B 19 23
2E43: 00 23 1E 1B 23 00 23 50 1E 05 0F 19 19 14 19 00
2E53: 19 00 14 19 00 19 05 14 19 00 19 00 14 19 00 19
2E63: 00 14 19 00 19 00 14 19 00 19 00 0F 19 00 19 00
2E73: 0F 01 01 02 02 0F 03 04 03 03 00 03 03 03 05 00
2E83: 03 03 03 03 08 03 03 03 03 19 03 03 03 03 00 03
```

```
2E93: 03 03 03 00 03 03 03 03 0A 01 01 02 02 00 03 04
2EA3: 03 03 16 03 03 03 05 00 03 03 03 03 08 03 03 03
2EB3: 03 00 03 03 03 03 19 03 03 03 03 00 03 03 03 03
2EC3: 0A 00 01 02 02 01 03 04 03 03 00 03 03 03 05 16
2ED3: 03 03 03 03 08 03 03 03 03 01 03 03 03 03 00 03
2EE3: 03 03 03 19 03 03 03 03 0A 00 05 06 07 00 07 0C
2EF3: 08 04 00 08 0D 08 14 02 03 0E 08 03 02 08 0F 08
2F03: 0D 01 08 10 08 04 01 08 10 08 19 02 08 10 08 06
2F13: 02 00 00 00 00 0F 00 00 00 00 14 00 00 00 00 14
2F23: 00 00 00 00 14 00 00 00 00 15 00 00 00 00 16 00
2F33: 00 00 00 17 00 00 00 00 19 00 01 03 04 01 04 00
2F43: 05 05 01 05 00 05 02 01 05 00 05 05 02 05 00 05
2F53: 06 01 05 00 05 05 01 05 00 05 02 01 05 00 05 05
2F63: 01 00 00 00 00 00 00 0A 00 00 00 00 0C 00 00 00
2F73: 00 0C 00 00 00 00 0C 00 07 00 00 0C 01 01 01 01
2F83: 0D 01 02 02 02 0E 02 01 01
```

```
; WAVE START - SETS UP PLAYER POSITION

WAVE_START_PLAYER:
2F8C: 8E 98 5A    LDX   #$985A              ; player_object_start
2F8F: CC 36 03    LDD   #$3603              ; set animation frame
metadata pointer to $3603. First 2 bytes at 3603: 04 0C (width and
height), next 2 bytes 37 3B (pointer to actual image)
2F92: ED 02       STD   $0002,X             ; set current animation
frame metadata pointer
2F94: ED 88 14    STD   $14,X               ; set previous
animation frame metadata pointer (previous = current)
2F97: 0F 70       CLR   $70
2F99: 0F 71       CLR   $71
2F9B: CC 30 7B    LDD   #$307B
2F9E: DD 72       STD   $72
2FA0: CC 4A 7C    LDD   #$4A7C              ; A = 4A, B = 7C
(middle of the screen)
2FA3: ED 04       STD   $0004,X             ; set "last" blitter
destination
2FA5: A7 0A       STA   $000A,X             ; Set player X
coordinate (whole part) to #$4A (74 decimal)
2FA7: 6F 0B       CLR   $000B,X             ; set fractional part
of X coordinate to 0
2FA9: E7 0C       STB   $000C,X             ; Set player Y
coordinate to #$7C (124 decimal)
2FAB: 6F 0D       CLR   $000D,X             ; set fractional part
of Y coordinate to 0
2FAD: 0F 87       CLR   $87
2FAF: 0F 48       CLR   $48                 ; flag used in
collision detection. When set to 1, it means player collision
detection routine is checking for collisions.
2FB1: 0F 8D       CLR   $8D                 ; number of family
members saved = 0
2FB3: 0F 8E       CLR   $8E                 ; number of cruise
missiles on screen = 0
```

```
2FB5: 0F 8A       CLR   $8A                 ; number of sparks on
screen = 0
2FB7: 0F 95       CLR   $95                 ; clear the "family
member being prog'd" flag
2FB9: 0F ED       CLR   temp_enforcer_count
2FBB: 86 02       LDA   #$02
2FBD: 97 EF       STA   $EF
2FBF: 0F F1       CLR   $F1                 ; number of tank shells
on screen = 0
2FC1: 39          RTS

2FC2: 96 59       LDA   $59
2FC4: 85 01       BITA  #$01
2FC6: 27 01       BEQ   $2FC9
2FC8: 39          RTS

2FC9: 4D          TSTA
2FCA: 2A 04       BPL   $2FD0
2FCC: 96 52       LDA   $52
2FCE: 20 03       BRA   $2FD3



; c804 widget_pia_dataa (widget = I/O board)
; bit 0  Move up
; bit 1  Move down
; bit 2  Move left
; bit 3  Move right
; bit 4  1 Player
; bit 5  2 Players
; bit 6  Fire up
; bit 7  Fire down

;c806 widget_pia_datab
; bit 0  Fire left
; bit 1  Fire right
; bit 2
; bit 3
; bit 4
; bit 5
; bit 6
; bit 7


MOVE_PLAYER:
2FD0: B6 C8 04    LDA   widget_pia_dataa     ; read movement stick
bits (bits 0-3)
2FD3: 8E 98 5A    LDX   #$985A               ; player_object_start
(x doesn't appear to be used in this routine?)
2FD6: CE 30 31    LDU   #$3031               ; pointer to player
movement and animation sequence descriptor table (see $3031)
2FD9: 84 0F       ANDA  #$0F                 ; Only want the
movement bits from the stick and nothing else
```

```
2FDB: 48          ASLA
2FDC: 48          ASLA                      ; A = A * 4
2FDD: 33 C6       LEAU  A,U                 ; U = U + A
2FDF: EC C4       LDD   ,U                  ; Load A with
horizontal direction delta and B with vertical direction delta
2FE1: DB 66       ADDB  $66                 ; Add B to $9866
(player_y)
2FE3: C1 18       CMPB  #$18                ; is Y < #$18 ?
2FE5: 25 06       BCS   $2FED               ; yes, so don't update
Y coordinate
2FE7: C1 DF       CMPB  #$DF                ; is Y > #$DF ?
2FE9: 22 02       BHI   $2FED               ; yes, so don't update
Y coordinate
2FEB: D7 66       STB   $66                 ; store B in $9866
(player_y)
2FED: 5F          CLRB                      ; clear B because we're
moving the fractional part of the delta into it
2FEE: 47          ASRA                      ; shift bit 0 into
carry, while preserving bit 7 (the sign bit)
2FEF: 56          RORB                      ; move carry into B, to
give us the fractional part
2FF0: D3 64       ADDD  $64                 ; Add D to $9864
(player_x)
2FF2: 81 07       CMPA  #$07                ; is player X past far
left boundary of screen? (invalid)
2FF4: 25 06       BCS   $2FFC               ; yes, coordinate is
invalid, goto $2FFC
2FF6: 81 8C       CMPA  #$8C                ; is player X past far
right boundary of screen? (invalid)
2FF8: 22 02       BHI   $2FFC               ; yes, coordinate is
invalid, goto $2FFC
2FFA: DD 64       STD   $64                 ; store D in $9864
(player_x)
2FFC: EC 42       LDD   $0002,U             ; get pointer to
animation table into D (see $3071 for description of table)
2FFE: 27 30       BEQ   $3030
3000: 10 93 72    CMPD  $72                 ; are we still using
the same the same animation table as before?
3003: 27 06       BEQ   $300B               ; yes, don't need to
update pointer to it, goto $300B
3005: DD 72       STD   $72                 ; animation needs to
change, so update pointer to current animation sequence
3007: 0F 71       CLR   $71                 ; set index into
animation to 0 (the start)
3009: 0F 70       CLR   $70
;
; The player may have moved but that doesn't necessarily mean the
animation frame changes also.
; It appears that $70, when 0, is the flag that says "OK, time to
change animation frame"
;
300B: D6 70       LDB   $70                 ; is it time to change
the animation frame in the sequence?
300D: 26 17       BNE   $3026               ; no, goto $3026
```

```
300F: DE 72       LDU   $72                 ; get pointer to
animation sequence
3011: 96 71       LDA   $71                 ; read index into
animation sequence
3013: E6 C6       LDB   A,U                 ; get byte from
animation sequence into B
3015: 26 04       BNE   $301B               ; if byte is not 0,
where zero indicates the end of the animation sequence, goto $301B
3017: 0F 71       CLR   $71                 ; otherwise, byte is 0,
animation sequence needs to start at first frame (index 0)
3019: E6 C4       LDB   ,U                  ; read first byte from
301B: 0C 71       INC   $71                 ; bump index into
animation sequence to next entry
301D: 5A          DECB                      ; B--;
301E: 58          ASLB                      ;
301F: 58          ASLB                      ; Multiply B by 4
3020: 4F          CLRA
3021: C3 35 EB    ADDD  #$35EB              ; compute pointer to
current animation frame metadata for current player animation
3024: DD 5C       STD   $5C                 ; set animation frame
metadata pointer for player
3026: 96 70       LDA   $70                 ; read animation frame
change countup - yes, count *up*
3028: 4C          INCA                      ; count up, erm, counts
up by 1...
3029: 81 02       CMPA  #$02                ; is countup < 2?
302B: 25 01       BCS   $302E               ; yes, <2, so goto
$302E
302D: 4F          CLRA                      ; OK count up is 2, so
reset count up to 0. $300B will pick this up, and its time to change
animation frame
302E: 97 70       STA   $70                 ; update count up.
3030: 39          RTS

PLAYER_MOVEMENT_AND_ANIMATION_DESCRIPTORS:
; 4 bytes per entry.
; Byte 0: delta to be added to player X. This includes a fractional
part: if bit 0 is set, player moves 1/2 a step more
; Byte 1: delta to be added to player Y.
; Byte 2 and 3: a pointer to the animation tables for player sprite.

3031: 00 00 00 00

; up
3035: 00 FF               ; movement delta X and Y
3037: 30 80     ; animation table pointer (see $3080)

; down
3039: 00 01     ; movement delta X and Y
303B: 30 7B     ; animation table pointer

;
303D: 00 00
303F: 00 00
```

```
' left
3041: FF 00
3043: 30 71

; up left
3045: FF FF
3047: 30 71

; down left
3049: FF 01
304B: 30 71

; right
3051: 01 00
3053: 30 76

; up right
3055: 01 FF
3057: 30 76

; down right
3059: 01 01
305B: 30 76


;
; Each player animation sequence is 5 bytes long.
; The system knows an animation sequence has ended and needs to
start from the beginning, when a zero byte is encountered.
; Non zero bytes are read, multiplied by 4, and added to #35EB to
give pointer to animation frame metadata for required animation
frame
;
; e.g. take animation sequence for moving right, beginning at $3076
; First byte is 4.
; Multiply by 4 = $10 (16 decimal).
; Add #$10 to #$35EB, gives you #$35FB, which is the animation frame
metadata for player moving right, animation frame #1
;
; Next byte is 5.
; Multiply by 4 = $14 (20 decimal)
; Add #$14 to #$35EB, gives you #$35FF, which is the animation frame
metadata for player moving right, animation frame #2
;
;

PLAYER_ANIMATION_SEQUENCE_TABLES:
; animation sequence for moving left
3071: 01 02 01 03 00

; animation sequence for moving right
3076: 04 05 04 06 00
```

```
; animation sequence for moving down
307B: 07 08 07 09 00

; animation sequence for moving up
3080: 0A 0B 0A 0C 00


;
; X = pointer to object
; A =
; U = pointer to linked list of objects to check for collision with
;

CHECK_IF_ANOTHER_OBJECT_PRESENT:
3085: 34 46            PSHS  U,B,A
3087: 34 06            PSHS  B,A
3089: E3 98 02         ADDD  [$02,X]         ; add in width & height
of animation frame metadata
308C: 34 06            PSHS  B,A                   ; save height
(in B) and width (in A) on stack
308E: 20 1B            BRA   $30AB
3090: EC 44            LDD   $4,U
3092: A1 E4            CMPA  ,S
3094: 24 15            BCC   $30AB
3096: E1 61            CMPB  $1,S
3098: 24 11            BCC   $30AB
309A: E3 D8 02         ADDD  [$02,U]
309D: A1 62            CMPA  $2,S
309F: 23 0A            BLS   $30AB
30A1: E1 63            CMPB  $3,S
30A3: 23 06            BLS   $30AB
30A5: 34 40            PSHS  U
30A7: AC E1            CMPX  ,S++
30A9: 26 04            BNE   $30AF
30AB: EE C4            LDU   ,U              ; get next object in
the list
30AD: 26 E1            BNE   $3090
30AF: 32 64            LEAS  $4,S
30B1: 35 C6            PULS  A,B,U,PC ; (PUL? PC=RTS)


PLAYER_COLLISION_DETECTION:
30B3: 86 01       LDA   #$01
30B5: 97 48       STA   $48                   ; Set flag to say it's
the player calling the collision detection function
30B7: DC 5E       LDD   $5E                   ; D = blitter
destination of player
30B9: DE 6E       LDU   $6E                   ; U = animation frame
metadata pointer
30BB: 8E 98 21    LDX   #$9821                ; X = pointer to
grunts_hulks_brains_progs_cruise_tanks list
30BE: BD D0 27    JSR   $D027                 ; JMP $D7C9 - collision
detection function
30C1: 26 2C       BNE   $30EF                 ; if collision, goto
```

```
$30EF, KILL_PLAYER
30C3: DC 5E       LDD   $5E
30C5: DE 6E       LDU   $6E
30C7: 8E 98 23    LDX   #$9823              ; pointer to electrode
linked list
30CA: BD D0 27    JSR   $D027               ; JMP $D7C9 - collision
detection function
30CD: 26 20       BNE   $30EF               ; if collision, goto
$30EF, KILL_PLAYER
30CF: DC 5E       LDD   $5E
30D1: DE 6E       LDU   $6E
30D3: 8E 98 17    LDX   #$9817              ; pointer to
spheroids_enforcers_quarks_sparks_shells list
30D6: BD D0 27    JSR   $D027               ; JMP $D7C9 - collision
detection function
30D9: 26 14       BNE   $30EF               ; if collision, goto
$30EF, KILL_PLAYER
30DB: DC 5E       LDD   $5E
30DD: DE 6E       LDU   $6E
30DF: 8E 98 1F    LDX   #$981F              ; family member linked
list start
30E2: BD D0 27    JSR   $D027               ; JMP $D7C9 - collision
detection function
30E5: 0F 48       CLR   $48
30E7: 86 01       LDA   #$01                ; delay before calling
function
30E9: 8E 30 B3    LDX   #$30B3              ; address of function
to call for this object next
30EC: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

; Player has hit something

KILL_PLAYER:
30EF: CC 26 D9    LDD   #$26D9
30F2: BD D0 4B    JSR   $D04B               ; JMP $D3C7
30F5: 86 1B       LDA   #$1B
30F7: 97 59       STA   $59
30F9: BD D0 60    JSR   $D060               ; JMP D1FF
30FC: C6 07       LDB   #$07
30FE: BD D0 BD    JSR   $D0BD               ; JMP $D655
3101: BD D0 24    JSR   $D024
3104: BD 5B 4C    JSR   $5B4C               ; Make player flash
when dying
3107: BD D0 45    JSR   $D045               ; JMP $D699 - get addr
of current player game state into X (but why? results never used!)
310A: B6 BD EC    LDA   p1_men              ; any more men left for
either player?
310D: BA BE 28    ORA   p2_men
3110: 26 1E       BNE   $3130
3112: 86 FF       LDA   #$FF                ; no more men left
3114: 97 59       STA   $59
3116: CC 1C 0A    LDD   #$1C0A              ; width = 1C, height =
0A
```

```
3119: 8E 3C 7E    LDX   #$3C7E               ; blitter dest
311C: BD D0 1B    JSR   $D01B                ; JMP $DADF - clear
rectangle to black
311F: 86 28       LDA   #$28
3121: C6 AA       LDB   #$AA
3123: D7 CF       STB   $CF
3125: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT        ; print
GAME OVER
3128: 86 78       LDA   #$78
312A: 8E E3 D3    LDX   #GET_INITIALS1
312D: 7E D0 66    JMP   $D066                ; JMP $D1E3 - allocate
function call

3130: BD D0 45    JSR   $D045                ; JMP $D699 - get addr
of current player game state into X
3133: A6 0B       LDA   $000B,X
3135: B7 BE 5D    STA   $BE5D
3138: B1 BE 5C    CMPA  $BE5C
313B: 23 03       BLS   $3140
313D: B7 BE 5C    STA   $BE5C
3140: 4F          CLRA
3141: D6 ED       LDB   temp_enforcer_count
3143: 27 1E       BEQ   $3163
3145: C0 04       SUBB  #$04
3147: 2B 03       BMI   $314C
3149: 4C          INCA
314A: 20 F9       BRA   $3145

314C: 4D          TSTA
314D: 26 06       BNE   $3155
314F: 7D BE 6F    TST   cur_sphereoids
3152: 26 01       BNE   $3155
3154: 4C          INCA
3155: BB BE 6F    ADDA  cur_sphereoids
3158: A1 88 1D    CMPA  $1D,X
315B: 23 03       BLS   $3160
315D: A6 88 1D    LDA   $1D,X
3160: B7 BE 6F    STA   cur_sphereoids
3163: BD 2B 69    JSR   $2B69
3166: BD D0 45    JSR   $D045                ; JMP $D699 - get addr
of current player game state into X
3169: E6 08       LDB   $0008,X
316B: 26 1C       BNE   $3189
316D: CC 1C 20    LDD   #$1C20               ; width = 1C (28
decimal), height = 20 (32 decimal)
3170: 8E 3C 77    LDX   #$3C77               ; blitter dest
3173: BD D0 1B    JSR   $D01B                ; JMP $DADF - clear
rectangle to black
3176: 86 4B       LDA   #$4B
3178: C6 AA       LDB   #$AA
317A: D7 CF       STB   $CF
317C: D6 3F       LDB   $3F
317E: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
3181: 86 60       LDA   #$60
```

```
3183: 8E 31 89    LDX   #$3189
3186: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

3189: 96 3F       LDA   $3F
318B: 88 03       EORA  #$03
318D: BD D0 48    JSR   $D048
3190: E6 08       LDB   $0008,X
3192: 27 F7       BEQ   $318B
3194: 97 3F       STA   $3F
3196: 7E 27 A3    JMP   $27A3


; Generate random position for an object at the start of the wave
;
; X = pointer to object
;
; Returns: A = X coordinate, B = Y coordinate
;

COMPUTE_INITIAL_POSITION:
3199: EC 98 02    LDD   [$02,X]             ; get width and height
into D
319C: 34 06       PSHS  B,A                 ; save B (height) and A
(width)
319E: 86 88       LDA   #$88
31A0: A0 E0       SUBA  ,S+                 ; compute #$88 minus
width held on stack.
31A2: BD D0 42    JSR   $D042               ; JMP $D6AC - multiply
A by a random number and put result in A
31A5: 8B 06       ADDA  #$06                ; add left border
31A7: 1F 89       TFR   A,B
31A9: 86 D2       LDA   #$D2                ; compute #$D2 minus
height on stack
31AB: A0 E0       SUBA  ,S+
31AD: BD D0 42    JSR   $D042               ; JMP $D6AC - multiply
A by a random number and put result in A
31B0: 8B 17       ADDA  #$17                ; add border top
31B2: 1E 89       EXG   A,B                 ; swap A and B so that
A = X coordinate, B = Y coordinate
31B4: 39          RTS



31B5: 6F 47       CLR   $0007,U
31B7: 6F 48       CLR   $0008,U
31B9: 96 59       LDA   $59
31BB: 2A 04       BPL   $31C1

31BD: DC 52       LDD   $52                   ;
31BF: 20 06       BRA   $31C7
```

```
; c804 widget_pia_dataa (widget = I/O board)
; bit 0  Move up
; bit 1  Move down
; bit 2  Move left
; bit 3  Move right
; bit 4  1 Player
; bit 5  2 Players
; bit 6  Fire up
; bit 7  Fire down

;c806 widget_pia_datab
; bit 0  Fire left
; bit 1  Fire right
; bit 2
; bit 3
; bit 4
; bit 5
; bit 6
; bit 7

READ_PLAYER_FIRE_BUTTONS:
31C1: B6 C8 04    LDA   widget_pia_dataa
31C4: F6 C8 06    LDB   widget_pia_datab
31C7: 54          LSRB                      ; move "fire left" into
carry
31C8: 46          RORA                      ;
31C9: 54          LSRB                      ; move "fire right" bit
into carry
31CA: 46          RORA
; at this point in A:
; bit 7: Fire right status
; bit 6: Fire left status
; bit 5: Fire up status
; bit 4: Fire down status
;
; all other bits - don't care
31CB: 84 F0       ANDA  #$F0                ; mask off top 4 bits
(the fire button bits in A)
31CD: E6 47       LDB   $0007,U             ; U is set to AA85 at
this point.
31CF: A7 47       STA   $0007,U
31D1: E1 47       CMPB  $0007,U
31D3: 26 54       BNE   $3229
31D5: 6C 48       INC   $0008,U
31D7: E6 48       LDB   $0008,U
31D9: C1 02       CMPB  #$02
31DB: 27 04       BEQ   $31E1
31DD: C4 07       ANDB  #$07
31DF: 26 4E       BNE   $322F
31E1: D6 87       LDB   $87                 ; how many player lasers
have been fired?
31E3: C1 04       CMPB  #$04
31E5: 24 46       BCC   $322D               ; if 4 or more.... can't
create any more, goto $322D
```

```
31E7: 0D 13       TST   $13
31E9: 27 44       BEQ   $322F
31EB: 44          LSRA                    ; shift bits right twice
31EC: 44          LSRA                    ;
31ED: 34 02       PSHS  A                 ; and then save on stack
31EF: 44          LSRA                    ; shift bits right once
31F0: AB E0       ADDA  ,S+               ; and add bits to the
value on stack

; A = computed offset into laser description table beginning at
$3237 (see $3237 for docs)
31F2: 10 8E 32 37 LDY   #$3237
31F6: 31 A6       LEAY  A,Y               ; Y+= A
31F8: 81 42       CMPA  #$42
31FA: 24 33       BCC   $322F
31FC: AE A4       LDX   ,Y                ; X= function to call to
initialise laser
31FE: 27 2F       BEQ   $322F
3200: 4F          CLRA
3201: BD D0 57    JSR   $D057             ; JMP $D25A - reserve
object metadata entry in list @ $9813 and call function in X
3204: 1F 13       TFR   X,U               ; U = object metadata
entry
3206: BD D0 51    JSR   $D051             ; reserve an object
entry - JMP $D28F.

; X = reserved object for laser.
; Y = pointer to entry in laser descriptor table (see $3237 for
docs)

3209: AF 47       STX   $0007,U           ; set pointer to this
object in object metadata
320B: EC 24       LDD   $0004,Y           ;
320D: C3 35 AE    ADDD  #$35AE            ; calculate address of
animation frame metadata for laser
3210: ED 02       STD   $0002,X           ; set current animation
frame metadata pointer
3212: ED 88 14    STD   $14,X             ; set previous animation
frame metadata pointer (previous = current)
3215: DC 5E       LDD   $5E               ; read player blitter
destination (A = X component, B = Y component)
3217: AB 22       ADDA  $0002,Y           ; add horizontal part of
laser offset to high byte of player's blitter destination
3219: A7 0A       STA   $000A,X           ; set X coordinate of
laser
321B: EB 23       ADDB  $0003,Y           ; add part of laser
offset to low byte of player's blitter destination
321D: E7 0C       STB   $000C,X           ; and store in vertical
component of laser object
321F: 0C 87       INC   $87               ; increment counter of
lasers fired
3221: CC 26 F0    LDD   #$26F0
3224: BD D0 4B    JSR   $D04B             ; JMP $D3C7
3227: 20 06       BRA   $322F
```

```
3229: 6F 48       CLR   $0008,U
322B: 20 02       BRA   $322F

322D: 6A 48       DEC   $0008,U
322F: 86 01       LDA   #$01                ; delay before calling
function
3231: 8E 31 B9    LDX   #$31B9              ; address of function
3234: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call


;
; Laser descriptor table. Describes the "strategy" to move the
laser, positioning and animation frame metadata for a laser.
;
; There is an entry for each direction the player can fire a laser
in (8 entries) and two null entries which are never used.
;
; Each entry in the table takes 6 bytes.
;
; The first two bytes is the address of the function that
initialises *and* moves the laser.
; The third byte is the signed X offset to add to the players X
coordinate, to place the laser in its start position.
; The fourth byte is the signed Y offset to add to the players Y
coordinate, to place the laser in its start position.
; The fifth and sixth bytes form the offset into the block of data
beginning at $35AE, which holds all the animation frame
; metadata for the laser.
;
; For example, take the 2nd entry beginning with 32 AD.
; 32AD - the first two bytes - is the address of the function that
moves the laser in the given direction.
;
; Let's inspect the next 2 bytes.
; 02 FF means the laser is going to be 4 pixels to the right of the
player (remember, 2 pixels to a byte, so a value of 2 = 4 pixels)
and
; one pixel above the player. (#$FF is -1 decimal as a signed byte)
;
; And let's inspect the last 2 bytes for this entry.
; 00 04 means an offset of 4 is added to $35AE. $35AE + 4 = $35B2.
Whats at memory address 35B2? 4 bytes like so:
; 01 06 35 C1
; As we know the animation frame metadata structure, we can see that
we have an animation frame that is
; 01 bytes (2 pixels) wide, 6 bytes high, and the actual pixel data
begins at $35C1.
;

LASER_DESCRIPTOR TABLE:
3237:
```

```
; null entry, never used
00 00
00 00
00 00

; fire up
32 AD                                   ; address of function to
move laser
02 FF                                   ; offset from players
current position. 2 bytes (=4 pixels) to right and -1 pixels above.
00 04                                   ; offset from $35AE

; fire down
32 C7                                   ; address of function to
move laser
02 04                                   ; offset from players
current position. 2 bytes (=4 pixels) to right and 4 pixels below
top of player.
00 04                                   ; offset from $35AE

; null entry, never used
00 00
00 00
00 00

; fire left
32 93                                   ; address of function to
move laser
00 04                                   ; offset from players
current position. 0 bytes to right and 4 pixels below top of player.
00 00                                   ; offset from $35AE

; fire up & left
33 FF                                   ; address of function to
move laser
00 00                                   ; offset from players
current position. 0 bytes to right and 0 pixels below top of player.
00 0C                                   ; offset from $35AE

; fire down & left
33 DC                                   ; address of function to
move laser
00 04                                   ; offset from players
current position. 0 bytes to right and 4 pixels below top of player.
00 08                                   ; offset from $35AE

; null entry, never used
00 00
00 00
00 00

; fire right
32 79
                ; address of function to move laser
```

```
02 04                                       ; offset from players
current position. 2 bytes (4 pixels) to right and 4 pixels below top
of player.
00 00                                       ; offset from $35AE

; fire up & right
34 22                                       ; address of function to
move laser
02 00                                       ; offset from players
current position. 2 bytes (4 pixels) to right and 0 pixels below top
of player.
00 08                                       ; offset from $35AE

; fire down & right
34 45                                       ; address of function to
move laser
02 04                                       ; offset from players
current position. 2 bytes (4 pixels) to right and 4 pixels below top
of player.
00 0C                                       ; offset from $35AE



MOVE_PLAYER_LASER_RIGHT:
3279: AE 47       LDX   $0007,U             ; get pointer to laser
object from object metadata
327B: A6 0A       LDA   $000A,X             ; get X coordinate into
A
327D: 8B 03       ADDA  #$03                ; add 3 bytes (6 pixels)
to A
327F: 81 8D       CMPA  #$8D                ; at far right edge of
playfield?
3281: 22 5E       BHI   $32E1               ; yes, laser is out of
bounds, goto $32E1
3283: A7 0A       STA   $000A,X             ; no, so update laser's
X coordinate
3285: CC 01 00    LDD   #$0100              ; set direction
parameters for collision detection function (see docs @ $346B)
3288: BD 34 6B    JSR   $346B               ; call function to
handle any collisions with this laser
328B: 86 01       LDA   #$01                ; delay before calling
this function
328D: 8E 32 79    LDX   #$3279             ; address of this
function
3290: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call


;
; Routine to move player laser left
;
;

MOVE_PLAYER_LASER_LEFT:
```

```
3293: AE 47       LDX   $0007,U            ; get pointer to laser
object from object metadata
3295: A6 0A       LDA   $000A,X            ; get X coordinate into
A
3297: 80 03       SUBA  #$03               ; subtract 3 bytes (6
pixels, remember 2 pixels per byte) to A
3299: 81 07       CMPA  #$07               ; at far left edge of
playfield?
329B: 25 48       BCS   $32E5              ; if <, then laser is
out of bounds, goto $32E5
329D: A7 0A       STA   $000A,X            ; no, so update laser's
X coordinate
329F: CC FF 00    LDD   #$FF00             ; set direction
parameters for collision detection function (see docs @ $346B)
32A2: BD 34 6B    JSR   $346B              ; call function to
handle any collisions with this laser
32A5: 86 01       LDA   #$01               ; delay before calling
this function
32A7: 8E 32 93    LDX   #$3293             ; address of this
function
32AA: 7E D0 66    JMP   $D066              ; JMP $D1E3 - allocate
function call


;
; Routine to move player laser up
;

MOVE_PLAYER_LASER_UP:
32AD: AE 47       LDX   $0007,U            ; get pointer to laser
object from object metadata
32AF: A6 0C       LDA   $000C,X            ; get Y coordinate into
A
32B1: 8B FA       ADDA  #$FA               ; subtract 6 from A
(results in move of 6 pixels)
32B3: 81 18       CMPA  #$18               ; at top of playfield?
32B5: 25 34       BCS   $32EB              ; yes, laser has hit top
border wall, goto $32EB
32B7: A7 0C       STA   $000C,X            ; no, update laser's Y
coordinate
32B9: CC 00 FF    LDD   #$00FF             ; set direction
parameters for collision detection function (see docs @ $346B)
32BC: BD 34 6B    JSR   $346B              ; call function to
handle any collisions with this laser
32BF: 86 01       LDA   #$01               ; delay before calling
this function
32C1: 8E 32 AD    LDX   #$32AD             ; address of this
function
32C4: 7E D0 66    JMP   $D066              ; JMP $D1E3 - allocate
function call

;
; Routine to move player laser down
;
```

```
MOVE_PLAYER_LASER_DOWN:
32C7: AE 47       LDX   $0007,U             ; get pointer to laser
object from object metadata
32C9: A6 0C       LDA   $000C,X             ; get vertical position
into A
32CB: 8B 06       ADDA  #$06                ; add 6 to A (results in
move of 6 pixels)
32CD: 81 E5       CMPA  #$E5                ; at bottom of
playfield?
32CF: 22 1E       BHI   $32EF               ; yes, laser has hit
bottom border wall, goto $32EF
32D1: A7 0C       STA   $000C,X             ; no, update laser's Y
coordinate
32D3: CC 00 01    LDD   #$0001              ; set direction
parameters for collision detection function (see docs @ $346B)
32D6: BD 34 6B    JSR   $346B               ; call function to
handle any collisions with this laser
32D9: 86 01       LDA   #$01                ; delay before calling
this function
32DB: 8E 32 C7    LDX   #$32C7              ; address of this
function
32DE: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call


; called by MOVE_PLAYER_LASER_RIGHT
32E1: 86 90       LDA   #$90                ; X coordinate of far
right border
32E3: 20 02       BRA   $32E7

; called by MOVE_PLAYER_LASER_LEFT
32E5: 86 06       LDA   #$06                ; X coordinate of far
left border

32E7: E6 0C       LDB   $000C,X             ; B = vertical position
32E9: 20 0A       BRA   $32F5

; called by MOVE_PLAYER_LASER_UP
32EB: C6 17       LDB   #$17                ; Y coordinate of
topmost border
32ED: 20 02       BRA   $32F1

; called by MOVE_PLAYER_LASER_DOWN
32EF: C6 EB       LDB   #$EB                ; Y coordinate of
bottommost border
32F1: A6 0A       LDA   $000A,X             ; get X coordinate of
laser into A
32F3: 20 2F       BRA   $3324

;
; This routine draws the laser "impacting" on the border walls. The
laser's impact spreads vertically. (think of a bullet hitting a
solid wall and you'll get what I mean)
```

```
;
; D = address on screen to draw a "flattened out" laser
;

DRAW_LASER_SPLASH_DAMAGE_VERTICAL:
32F5: ED 49       STD   $0009,U             ; store address on
screen of part of wall where laser hit
32F7: BD 34 A3    JSR   $34A3               ; dispose of laser
object and erase laser image
32FA: AE 49       LDX   $0009,U
32FC: 96 91       LDA   $91                 ; get colour to draw
splash damage in A...
32FE: D6 91       LDB   $91                 ; ... and B
3300: ED 1F       STD   $-1,X               ; write to screen 1
pixel above of where laser hit. 4 pixels are written - 2
horizontally and 2 vertically
3302: A7 01       STA   $0001,X             ; write to screen 1
pixel below where laser hit
3304: 86 02       LDA   #$02                ; delay
3306: 8E 33 0C    LDX   #$330C              ; address of function
to call
3309: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

;
; remove splash damage from wall
;
UNDO_LASER_SPLASH_DAMAGE_VERTICAL:
330C: 96 8F       LDA   $8F                 ; get wall colour
330E: AE 49       LDX   $0009,U             ; get address on screen
where laser "splash damage" was drawn
3310: A7 1F       STA   $-1,X               ; and undo what was
done at $3300
3312: A7 01       STA   $0001,X             ; and undo what was
done at $3302.
3314: 86 01       LDA   #$01                ; delay
3316: 8E 33 1C    LDX   #$331C              ; address of function
to call
3319: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call


; remove final trace of laser
RESTORE_WALL_VERTICAL:
331C: 96 8F       LDA   $8F                 ; get wall colour
331E: A7 D8 09    STA   [$09,U]             ; and finally erase
last vestige of laser
3321: 7E D0 63    JMP   $D063               ; JMP $D1F3



;
; This routine draws the laser "impacting" on the walls. The laser's
impact spreads horizontally. (think of a bullet hitting a solid wall
```

```
and you'll get what I mean)
;
; D = address on screen to draw a "flattened out" laser
;

DRAW_LASER_SPLASH_DAMAGE_HORIZONTAL:
3324: C1 EA       CMPB  #$EA                ; did laser hit bottom
part of wall?
3326: 24 02       BCC   $332A               ; yes, goto $332A
3328: C6 16       LDB   #$16                ; ok, set Y coordinate
to #$16 (22 decimal) where laser hit wall
332A: 81 06       CMPA  #$06                ; did laser hit left
part of wall?
332C: 22 01       BHI   $332F
332E: 4C          INCA
332F: ED 49       STD   $0009,U             ; store address on
screen of part of wall where laser hit
3331: BD 34 A3    JSR   $34A3               ; dispose of laser
object and erase laser image
3334: 96 91       LDA   $91                 ; get colour to draw
flattened laser in A...
3336: D6 91       LDB   $91                 ; ... and B
3338: AE 49       LDX   $0009,U             ; get address on screen
where laser hit wall
333A: ED 84       STD   ,X                  ; draw 4 pixels (2
bytes) at part of wall where laser hit
;
; Some notes. As you recall, the Williams hardware has 4 bits per
pixel. Therefore 2 pixels are packed into a single byte.
; The left nibble (bits 7..4) is the first pixel and the right
nibble (bits 3..0) is the second pixel.
333C: 96 8F       LDA   $8F                 ; get wall colour
333E: 84 F0       ANDA  #$F0                ; preserve the leftmost
pixel (the left nibble, as described above)
3340: 34 02       PSHS  A                   ; save result on stack
3342: 96 91       LDA   $91                 ; get colour to draw
flattened laser into A
3344: 84 0F       ANDA  #$0F                ; preserve the
rightmost pixel (the right nibble)
3346: AB E0       ADDA  ,S+                 ; combine the leftmost
pixel of the wall colour and the rightmost pixel of the flattened
laser. Now A holds 2 pixels
3348: 1F 89       TFR   A,B                 ;
334A: ED 89 FF 00 STD   $FF00,X             ; write D 2 pixels to
left of where the laser hit the wall - creating splash damage
334E: 86 02       LDA   #$02                ; delay before calling
function
3350: 8E 33 56    LDX   #$3356              ; address of function
to call
3353: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

;
; remove splash damage from wall
```

```
;
UNDO_HORIZONTAL_SPLASH_DAMAGE:
3356: AE 49       LDX   $0009,U             ; get address on screen
where laser "splash damage" was drawn
3358: 96 8F       LDA   $8F                 ; get wall colour into
A..
335A: D6 8F       LDB   $8F                 ; and B.. Now D has 4
pixel's worth of wall colour.
335C: ED 89 FF 00 STD   $FF00,X             ; write D 2 pixels (1
byte) to left of where laser hit wall. Undoes what was done at $334A
3360: 84 0F       ANDA  #$0F                ; remove left hand
pixel from A.
3362: C4 0F       ANDB  #$0F                ; remove left hand
pixel from B.
3364: 34 06       PSHS  B,A                 ; push remaining pixels
on stack
3366: 96 91       LDA   $91                 ; get colour to draw
splash damage into A...
3368: D6 91       LDB   $91                 ; and B. Now D has 4
pixel's worth of splash damage colour.
336A: 84 F0       ANDA  #$F0                ; remove right hand
pixel from A
336C: C4 F0       ANDB  #$F0                ; remove right hand
pixel from B
336E: E3 E1       ADDD  ,S++                ; combine those pixels
with those on the stack
3370: ED 84       STD   ,X                  ; and write to the
screen. The laser's splash damage is ALMOST removed now.
RESTORE_WALL_HORIZONTAL completes the job.
3372: 86 01       LDA   #$01                ; delay before calling
function
3374: 8E 33 7A    LDX   #$337A              ; address of function
to call
3377: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

; remove final trace of laser
RESTORE_WALL_HORIZONTAL:
337A: 96 8F       LDA   $8F                 ; get wall colour into
A..
337C: D6 8F       LDB   $8F                 ; and B..
337E: ED D8 09    STD   [$09,U]             ; set part of wall
where laser hit back to wall colour - removing last trace of laser
3381: 7E D0 63    JMP   $D063               ; JMP $D1F3

3384: 7E 32 F5    JMP   $32F5               ; jump to
DRAW_LASER_SPLASH_DAMAGE_VERTICAL

3387: 20 9B       BRA   $3324               ; go to
DRAW_LASER_SPLASH_DAMAGE_HORIZONTAL


;
; When we get here, the laser has hit a border wall.
```

```
;
; A = horizontal direction of laser ($FF = left, 0 =no horizontal
movement, 1=right)
; B = vertical direction of laser ($FF = up, 0 = no vertical
movement, 1 = down)
; Y = packed word containing offsets to add to laser position.
; Most significant byte:
;

LASER_HIT_WALL:
3389: D7 2C       STB   $2C                 ; store vertical
direction
338B: 5F          CLRB
338C: 47          ASRA
338D: 56          RORB
338E: DD 2D       STD   $2D
3390: 1F 20       TFR   Y,D                 ; A and B now hold
3392: AB 0A       ADDA  $000A,X             ; A+= X coordinate of
laser
3394: EB 0C       ADDB  $000C,X             ; B+= Y coordinate of
laser
3396: A7 0A       STA   $000A,X             ; update X coordinate
of laser
3398: E7 0C       STB   $000C,X             ; update Y coordinate
of laser
339A: DC 2D       LDD   $2D
339C: E3 0A       ADDD  $000A,X
339E: ED 0A       STD   $000A,X
33A0: E6 0C       LDB   $000C,X
33A2: DB 2C       ADDB  $2C
33A4: E7 0C       STB   $000C,X
33A6: C1 EA       CMPB  #$EA                ; hit bottom wall?
33A8: 22 DD       BHI   $3387               ; yes, goto $3387,
which then branches to DRAW_LASER_SPLASH_DAMAGE_HORIZONTAL
33AA: C1 18       CMPB  #$18                ; hit top wall?
33AC: 25 D9       BCS   $3387               ; yes, goto $3387
33AE: 81 8F       CMPA  #$8F                ; hit right wall?
33B0: 22 D2       BHI   $3384               ; yes, goto $3384,
which then jumps to DRAW_LASER_SPLASH_DAMAGE_VERTICAL
33B2: 81 07       CMPA  #$07                ; hit left wall?
33B4: 25 CE       BCS   $3384               ; yes, goto $3384
33B6: 20 E2       BRA   $339A

; called by MOVE_PLAYER_LASER_DOWN_LEFT
33B8: 10 8E 00 05 LDY   #$0005
33BC: CC FF 01    LDD   #$FF01
33BF: 20 C8       BRA   $3389

; called by MOVE_LASER_UP_LEFT
33C1: 10 8E 00 00 LDY   #$0000
33C5: CC FF FF    LDD   #$FFFF
33C8: 20 BF       BRA   $3389

; called by MOVE_LASER_UP_RIGHT
```

```
33CA: 10 8E 02 00 LDY   #$0200
33CE: CC 01 FF    LDD   #$01FF
33D1: 20 B6       BRA   $3389

; called by MOVE_LASER_DOWN_RIGHT
33D3: 10 8E 02 05 LDY   #$0205
33D7: CC 01 01    LDD   #$0101
33DA: 20 AD       BRA   $3389


;
; Routine to move player laser down left
;
; I wonder why these routines are down here, instead of being up
there with the rest of the laser movement code?
; Could it be, that the diagonal firing was added later when Vid
Kidz decided 4-way firing wasn't good enough?
;

MOVE_PLAYER_LASER_DOWN_LEFT:
33DC: AE 47       LDX   $0007,U             ; get pointer to laser
object from object metadata
33DE: A6 0A       LDA   $000A,X             ; get X coordinate of
laser into A
33E0: 80 03       SUBA  #$03                ; subtract 3
(effectively 6 pixels) from X coordinate
33E2: E6 0C       LDB   $000C,X             ; get vertical position
into B
33E4: CB 06       ADDB  #$06                ; add 6 to vertical
position
33E6: 81 07       CMPA  #$07                ; has laser hit left
border wall?
33E8: 25 CE       BCS   $33B8               ; yes, goto $33B8
33EA: C1 E5       CMPB  #$E5                ; has laser hit bottom
border wall??
33EC: 22 CA       BHI   $33B8               ; yes, goto $33B8
33EE: A7 0A       STA   $000A,X             ; update X coordinate
of laser
33F0: E7 0C       STB   $000C,X             ; update Y coordinate
of laser
33F2: CC FF 01    LDD   #$FF01              ; set direction
parameters for collision detection function (see docs @ $346B)
33F5: 8D 74       BSR   $346B               ; call function to
handle any collisions with this laser
33F7: 86 01       LDA   #$01
33F9: 8E 33 DC    LDX   #$33DC              ; address of function
to call (this one)
33FC: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call


;
; Routine to move player laser up left.
;
```

```
MOVE_LASER_UP_LEFT:
33FF: AE 47       LDX   $0007,U             ; get pointer to laser
object from object metadata
3401: A6 0A       LDA   $000A,X             ; get X coordinate into
A
3403: 80 03       SUBA  #$03                ; subtract 3
(effectively 6 pixels) from X coordinate
3405: 81 07       CMPA  #$07                ; has laser hit left
border wall?
3407: 25 B8       BCS   $33C1               ; yes, goto $33C1
3409: E6 0C       LDB   $000C,X             ; get vertical position
into B
340B: C0 06       SUBB  #$06                ; subtract 6 from Y
coordinate
340D: C1 18       CMPB  #$18                ; has laser hit top
border wall?
340F: 25 B0       BCS   $33C1               ; yes, goto $33C1
3411: A7 0A       STA   $000A,X             ; update X coordinate
of laser
3413: E7 0C       STB   $000C,X             ; update Y coordinate
of laser
3415: CC FF FF    LDD   #$FFFF              ; set direction
parameters for collision detection function (see docs @ $346B)
3418: 8D 51       BSR   $346B               ; call function to
handle any collisions with this laser
341A: 86 01       LDA   #$01
341C: 8E 33 FF    LDX   #$33FF              ; address of function
to call (this one)
341F: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call


MOVE_LASER_UP_RIGHT:
3422: AE 47       LDX   $0007,U             ; get pointer to laser
object from object metadata
3424: A6 0A       LDA   $000A,X             ; get X coordinate into
A
3426: 8B 03       ADDA  #$03                ; add 3 (effectively 6
pixels) to X coordinate
3428: 81 8D       CMPA  #$8D                ; has laser hit far
right border wall?
342A: 22 9E       BHI   $33CA               ; yes, goto $33CA
342C: E6 0C       LDB   $000C,X             ; get vertical position
into B
342E: C0 06       SUBB  #$06                ; subtract 6 from Y
coordinate
3430: C1 18       CMPB  #$18                ; has laser hit top
border wall?
3432: 25 96       BCS   $33CA
3434: A7 0A       STA   $000A,X             ; update X coordinate
of laser
3436: E7 0C       STB   $000C,X             ; update Y coordinate
of laser
```

```
3438: CC 01 FF    LDD   #$01FF               ; set direction
parameters for collision detection function (see docs @ $346B)
343B: 8D 2E       BSR   $346B                ; call function to
handle any collisions with this laser
343D: 86 01       LDA   #$01
343F: 8E 34 22    LDX   #$3422               ; address of function
to call (this one)
3442: 7E D0 66    JMP   $D066                ; JMP $D1E3 - allocate
function call


MOVE_LASER_DOWN_RIGHT:
3445: AE 47       LDX   $0007,U              ; get pointer to laser
object from object metadata
3447: A6 0A       LDA   $000A,X              ; get X coordinate into
A
3449: 8B 03       ADDA  #$03                 ; add 3 (effectively 6
pixels) to X coordinate
344B: 81 8D       CMPA  #$8D                 ; has laser hit far
right border wall?
344D: 22 84       BHI   $33D3                ; yes, goto $33D3
344F: E6 0C       LDB   $000C,X              ; get vertical position
into B
3451: CB 06       ADDB  #$06                 ; add 6 to Y coordinate
3453: C1 E5       CMPB  #$E5                 ; has laser hit far
bottom border wall?
3455: 23 03       BLS   $345A                ; no, goto $345A
3457: 7E 33 D3    JMP   $33D3

345A: A7 0A       STA   $000A,X              ; update X coordinate
of laser
345C: E7 0C       STB   $000C,X              ; update Y coordinate
of laser
345E: CC 01 01    LDD   #$0101               ; set direction
parameters for collision detection function (see docs @ $346B)
3461: 8D 08       BSR   $346B                ; call function to
handle any collisions with this laser
3463: 86 01       LDA   #$01
3465: 8E 34 45    LDX   #$3445               ; address of function
to call (this one)
3468: 7E D0 66    JMP   $D066                ; JMP $D1E3 - allocate
function call



;
; Expects:
; A = horizontal direction of laser ($FF = left, 0 = not moving
horizontally, 1=right)
; B = vertical direction of laser ($FF = up, 0 = not moving
vertically, 1 = down)
;
;
```

```
LASER_COLLISION_DETECTION:
346B: DD 88       STD   $88                 ; set player laser
directions
346D: BD D0 8D    JSR   $D08D               ; JMP $DB2F
3470: 34 50       PSHS  U,X
3472: EE 02       LDU   $0002,X
3474: EC 04       LDD   $0004,X
3476: 8E 98 23    LDX   #$9823              ; electrode list
pointer
3479: BD D0 27    JSR   $D027               ; JMP $D7C9 - collision
detection function
347C: 26 1E       BNE   $349C
347E: AE E4       LDX   ,S
3480: EE 02       LDU   $0002,X
3482: EC 04       LDD   $0004,X
3484: 8E 98 21    LDX   #$9821              ; pointer to grunts/
hulks/brains/progs/cruise missile/tank list
3487: BD D0 27    JSR   $D027               ; JMP $D7C9 - collision
detection function
348A: 26 10       BNE   $349C
348C: AE E4       LDX   ,S
348E: EE 02       LDU   $0002,X
3490: EC 04       LDD   $0004,X
3492: 8E 98 17    LDX   #$9817              ; pointer to quarks,
sparks, sphereoids, enforcers, tank shells
3495: BD D0 27    JSR   $D027               ; JMP $D7C9 - collision
detection function
3498: 26 02       BNE   $349C
349A: 35 D0       PULS  X,U,PC ;(PUL? PC=RTS)

349C: 35 50       PULS  X,U
349E: 8D 03       BSR   $34A3               ; dispose of laser
object and erase laser image
34A0: 7E D0 63    JMP   $D063               ; JMP $D1F3


;
; Erase the laser image from the screen and free the laser object
;
;

ERASE_LASER_AND_FREE_OBJECT:
34A3: BD D0 15    JSR   $D015               ; JMP $DB03 - erase
object from screen
34A6: DC 1B       LDD   $1B
34A8: ED 84       STD   ,X
34AA: 9F 1B       STX   $1B                 ; mark this laser as
current first free object
34AC: 0A 87       DEC   $87                 ; reduce count of
lasers on screen
34AE: 39          RTS
```

```
34AF: 34 76       PSHS  U,Y,X,B,A
34B1: 96 40       LDA   $40
34B3: BD D0 0F    JSR   $D00F               ; JMP $DC13 - draw
player scores
34B6: 4A          DECA
34B7: 26 FA       BNE   $34B3
34B9: 8D 6B       BSR   $3526               ; draw border walls
34BB: BD 26 C9    JSR   $26C9               ; JMP $34E0 - draw
player lives remaining
34BE: 35 F6       PULS  A,B,X,Y,U,PC ;(PUL? PC=RTS)


; print wave at bottom of screen
PRINT_WAVE_NUMBER:
34C0: B6 BD ED    LDA   p1_wave
34C3: BD D0 2A    JSR   $D02A               ; Convert wave number
to BCD and store in A
34C6: 1F 89       TFR   A,B                 ; B = wave number
34C8: 86 68       LDA   #$68
34CA: BD 5F 96    JSR   $5F96               ; JMP $613F: print
string in small font
34CD: 96 40       LDA   $40                 ; read how many players
are playing (1 or 2)
34CF: 4A          DECA                      ; reduce count by 1
34D0: 27 0D       BEQ   $34DF               ; if count == 0, then
it's a one player game, no need to draw player 2's wave, goto $34DF
34D2: B6 BE 29    LDA   p2_wave             ; Convert wave number
to BCD and store in A
34D5: BD D0 2A    JSR   $D02A               ; JMP $613F: print
string in small font
34D8: 1F 89       TFR   A,B
34DA: 86 72       LDA   #$72
34DC: BD 5F 96    JSR   $5F96               ; JMP $613F: print
string in small font
34DF: 39          RTS


DRAW_LIVES_REMAINING:
34E0: 8E 2E 0E    LDX   #$2E0E              ; X= blitter
destination
34E3: CC 15 08    LDD   #$1508              ; A = width (#$15), B =
height (#$08)
34E6: BD D0 1B    JSR   $D01B               ; JMP $DADF: clear
rectangle to black
34E9: 8E 6E 0E    LDX   #$6E0E              ; X= blitter
destination
34EC: BD D0 1B    JSR   $D01B               ; JMP $DADF - clear
rectangle to black
34EF: 10 8E 35 92 LDY   #$3592              ; set blitter source
34F3: B6 BD EC    LDA   p1_men              ; read number of player
1 lives left
34F6: 27 14       BEQ   $350C               ; if no lives left, go
do player 2 lives @ $350C
34F8: 81 07       CMPA  #$07                ; is number of lives
```

```
left <=7?
34FA: 23 02       BLS   $34FE               ; yes, goto $34FE
34FC: 86 07       LDA   #$07                ; lives left is >7, so
set 7 images to be drawn max, to fit into space allocated for player
1 lives left
34FE: 97 2B       STA   $2B
3500: CC 2E 0E    LDD   #$2E0E              ; blitter destination
3503: BD D0 21    JSR   $D021               ; JMP $DA82 - do blit
without transparency
3506: 8B 04       ADDA  #$04                ; increment X component
of blitter destination for player life image (4 bytes = 8 pixels)
3508: 0A 2B       DEC   $2B                 ; decrement number of
player life images left to draw
350A: 26 F7       BNE   $3503               ; if we've not drawn
all lives goto $3503
350C: B6 BE 28    LDA   p2_men              ; read number of player
2 lives left
350F: 27 14       BEQ   $3525               ; if no lives left,
goto $3525, which is an RTS
3511: 81 07       CMPA  #$07                ; is number of lives
left <=7?
3513: 23 02       BLS   $3517               ; yes, goto $3517
3515: 86 07       LDA   #$07                ; lives left is >7, so
set 7 images to be drawn max, to fit into space allocated for player
2 lives left
3517: 97 2B       STA   $2B
3519: CC 6E 0E    LDD   #$6E0E              ; blitter destination
351C: BD D0 21    JSR   $D021               ; JMP $DA82 - do blit
without transparency
351F: 8B 04       ADDA  #$04
3521: 0A 2B       DEC   $2B                 ; decrement number of
player life icons left to draw
3523: 26 F7       BNE   $351C               ; increment X component
of blitter destination for player life image (4 bytes = 8 pixels)
3525: 39          RTS


DRAW_BORDER_WALLS:
3526: 8E 06 16    LDX   #$0616
3529: 96 8F       LDA   $8F                 ; read current wall
colour
352B: A7 89 8A 00 STA   $8A00,X             ; plot pixels for right
hand border wall
352F: A7 80       STA   ,X+                 ; plot pixels for left
hand border wall
3531: 8C 06 EC    CMPX  #$06EC              ; all side walls drawn?
3534: 23 F5       BLS   $352B               ; if not, goto $352B
3536: 8E 07 16    LDX   #$0716              ; screen address for
top border
3539: D6 8F       LDB   $8F                 ; read current wall
colour (now D = wall colour)
353B: ED 84       STD   ,X                  ; write top border wall
353D: ED 89 00 D5 STD   $00D5,X             ; write bottom border
wall
```

```
3541: 30 89 01 00 LEAX  $0100,X             ; move to next pixel
pair across (remember Williams graphics hardware screen layout)
3545: 8C 8F 16    CMPX  #$8F16              ; all top & bottom
walls drawn?
3548: 23 F1       BLS   $353B               ; if not, goto $353B
354A: 39          RTS                       ; done


354B: DE 27       LDU   $27
354D: 11 83 B3 E4 CMPU  #$B3E4
3551: 25 03       BCS   $3556
3553: CE B3 A4    LDU   #$B3A4
3556: 96 92       LDA   $92                 ; A = count of objects
to process
3558: 34 02       PSHS  A
355A: 10 9E 93    LDY   $93                 ; Y = pointer to list
of animation frame metadata's
355D: AE C1       LDX   ,U++                ; get object pointer
from U
355F: 27 21       BEQ   $3582               ; if X == 0 then get
next pointer
3561: EC A4       LDD   ,Y                  ; read width and height
3563: 88 04       EORA  #$04
3565: C8 04       EORB  #$04
3567: 1A 10       ORCC  #$10
3569: FD CA 06    STD   blitter_w_h
356C: D6 90       LDB   $90                 ; read blitter mask
356E: F7 CA 01    STB   blitter_mask
3571: EC 22       LDD   $0002,Y             ; read pointer to
actual image
3573: FD CA 02    STD   blitter_source
3576: EC 04       LDD   $0004,X             ; read blitter
destination from object
3578: FD CA 04    STD   blitter_dest
357B: C6 1A       LDB   #$1A                ; blitter flags: 11010
- transparent, solid
357D: F7 CA 00    STB   start_blitter
3580: 1C EF       ANDCC #$EF                ; clear interrupt flag
3582: 6A E4       DEC   ,S                  ; reduce count of
objects to process by one
3584: 26 D7       BNE   $355D               ; if !=0, get next one
3586: 32 61       LEAS  $0001,S
3588: DF 27       STU   $27
358A: 86 02       LDA   #$02
358C: 8E 35 4B    LDX   #$354B
358F: 7E D0 66    JMP   $D066              ; JMP $D1E3 - allocate
function call

3592: 03 08       COM   $08
3594: 35 96       PULS  A,B,X,PC ;(PUL? PC=RTS)

3596: 02 22 00 BB 0B B0 BB 0B B0 00 20 00 88 08 80 30
35A6: 80 30 08 08 00 88 08 80 03 01 35 BE 01 06 35 C1
35B6: 03 06 35 C7 03 06 35 D9 AA AA AA A0 A0 A0 A0 A0
```

```
35C6: A0 00 00 0A 00 00 A0 00 0A 00 00 A0 00 0A 00 00
35D6: A0 00 00 A0 00 00 0A 00 00 00 A0 00 00 0A 00 00
35E6: 00 A0 00 00 0A 04 0C 36 1B 04 0C 36 4B 04 0C 36
35F6: 7B 04 0C 36 AB 04 0C 36 DB 04 0C 37 0B 04 0C 37
3606: 3B 04 0C 37 6B 04 0C 37 9B 04 0C 37 CB 04 0C 37
3616: FB 04 0C 38 2B 0B 22 20 00 0B BB BB 00 0B 22 2B
3626: 00 00 22 20 00 00 09 00 00 00 99 90 00 00 93 90
3636: 00 00 93 90 00 00 03 00 00 00 01 00 00 00 99 00
3646: 00 00 00 00 00 0B 22 20 00 0B BB BB 00 0B 22 2B
3656: 00 00 22 20 00 00 09 00 00 00 09 90 00 03 33 93
3666: 00 00 09 93 00 00 90 93 00 00 90 90 00 09 90 90
3676: 00 00 00 00 00 0B 22 20 00 0B BB BB 00 0B 22 2B
3686: 00 00 22 20 00 00 09 00 00 00 09 90 00 03 39 93
3696: 00 00 09 93 00 00 90 93 00 00 90 90 00 09 90 90
36A6: 00 00 00 00 00 02 22 B0 00 BB BB B0 00 B2 22 B0
36B6: 00 02 22 00 00 00 90 00 00 09 99 00 00 09 39 00
36C6: 00 09 39 00 00 00 30 00 00 00 90 00 00 00 99 00
36D6: 00 00 00 00 00 02 22 B0 00 BB BB B0 00 B2 22 B0
36E6: 00 02 22 00 00 00 90 00 00 09 90 00 00 39 93 30
36F6: 00 39 90 00 00 39 09 00 00 09 09 00 00 09 09 90
3706: 00 00 00 00 00 02 22 B0 00 BB BB B0 00 B2 22 B0
3716: 00 02 22 00 00 00 90 00 00 09 90 00 00 39 33 30
3726: 00 39 90 00 00 39 09 00 00 09 09 00 00 09 09 90
3736: 00 00 00 00 00 00 22 20 00 BB B2 BB B0 B0 B0 B0
3746: B0 0B 22 2B 00 00 09 00 00 09 93 99 00 39 93 99
3756: 30 30 93 90 30 30 90 90 30 00 90 90 00 09 90 99
3766: 00 00 00 00 00 00 22 20 00 BB B2 BB B0 B0 B0 B0
3776: B0 0B 22 2B 00 00 09 00 00 09 93 99 00 39 93 99
3786: 30 30 93 90 30 30 90 90 00 09 90 90 00 00 00 90
3796: 00 00 00 99 00 00 22 20 00 BB B2 BB B0 B0 B0 B0
37A6: B0 0B 22 2B 00 00 09 00 00 09 93 99 00 39 93 99
37B6: 30 30 93 90 30 00 90 90 30 00 90 99 00 00 90 00
37C6: 00 09 90 00 00 BB 22 2B B0 B2 22 22 B0 B2 22 22
37D6: B0 00 22 20 00 00 09 00 00 09 99 99 00 39 99 99
37E6: 30 30 99 90 30 30 90 90 30 00 90 90 00 09 90 99
37F6: 00 00 00 00 00 BB 22 2B B0 B2 22 22 B0 B2 22 22
3806: B0 00 22 20 00 00 09 00 00 09 99 99 00 39 99 99
3816: 30 30 99 90 30 30 90 90 00 09 90 90 00 00 00 90
3826: 00 00 00 99 00 BB 22 2B B0 B2 22 22 B0 B2 22 22
3836: B0 00 22 20 00 00 09 00 00 09 99 99 00 39 99 99
3846: 30 30 99 90 30 00 90 90 30 00 90 99 00 00 90 00
3856: 00 09 90 00 00 FF FF FF FF FF FF FF FF FF FF FF
3866: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
3876: FF FF FF FF FF FF FF FF FF FF

3880: 7E 38 AA    JMP   $38AA

3883: 7E 39 50    JMP   $3950
3886: 3B          RTI

3887: 05          Illegal Opcode
3888: 7E 39 42    JMP   $3942

388B: 7E 39 3C    JMP   $393C
```

```
388E: 7E 38 FE    JMP   $38FE

3891: 40          NEGA
3892: 63 3B       COM   $FFFB,Y
3894: 05          Illegal Opcode
3895: 7E 3A D9    JMP   $3AD9

;
; when I see illegal opcode I just know this is data used somewhere!
;

3898: D0 01       SUBB  $01
389A: 0C 14       INC   $14
389C: 01          Illegal Opcode
389D: 08 17       ASL   $17
389F: 00 C0       NEG   $C0
38A1: 01          Illegal Opcode
38A2: 0A 06       DEC   $06
38A4: 00 D0       NEG   $D0
38A6: 01          Illegal Opcode
38A7: 08 17       ASL   $17



INITIALISE_ALL_GRUNTS:
38AA: B6 BE 68    LDA   cur_grunts          ; read number of grunts
38AD: 34 02       PSHS  A
38AF: 27 4B       BEQ   $38FC               ; if we have no grunts,
goto $38FC
38B1: BD D0 7B    JSR   $D07B               ; JMP $D2DA - reserve
entry in list used by grunts, hulks, brains, progs, cruise missiles
and tanks (starts at $9821)
; X = newly reserved object entry
38B4: CC 40 63    LDD   #$4063              ; set blitter source to
start image's metadata
38B7: ED 02       STD   $0002,X             ; set current animation
frame metadata pointer
38B9: ED 88 14    STD   $14,X               ; set previous
animation frame metadata pointer (previous = current)
38BC: 8D D0       BSR   $388E
38BE: BD 26 C3    JSR   $26C3               ; JMP $3199 - get
random position on playfield for object (returns: A = X coordinate,
B = Y coordinate)
38C1: D1 2B       CMPB  $2B
38C3: 23 0C       BLS   $38D1
38C5: D1 2C       CMPB  $2C
38C7: 24 08       BCC   $38D1
38C9: 91 2D       CMPA  $2D
38CB: 23 04       BLS   $38D1
38CD: 91 2E       CMPA  $2E
38CF: 25 ED       BCS   $38BE               ; position is invalid,
go compute a new position
38D1: ED 04       STD   $0004,X             ; blitter destination =
```

```
D
38D3: A7 0A       STA   $000A,X             ; X coordinate = A
38D5: E7 0C       STB   $000C,X             ; Y coordinate = B
38D7: 1F 03       TFR   D,U                 ; U = blitter
destination
38D9: EC 98 02    LDD   [$02,X]             ; get width and height
of initial grunt image into D
38DC: BD D0 03    JSR   $D003               ; JMP $DE0F -
TEST_FOR_PIXELS_WITHIN_RECTANGLE
38DF: 26 DD       BNE   $38BE               ; if Z==0 then there is
something under the grunt, so new coordinates are required
38E1: B6 BE 5C    LDA   $BE5C               ;
38E4: BD D0 42    JSR   $D042               ; JMP $D6AC - multiply
A by a random number and put result in A
38E7: A7 88 13    STA   $13,X               ; set movement delay
field
38EA: CC 3A 76    LDD   #$3A76              ; address of routine to
jump to when grunt hits something
38ED: ED 08       STD   $0008,X             ;
38EF: 8D 9A       BSR   $388B               ; JMP $393C - blit
grunt in solid colour invisible to player
38F1: 6A E4       DEC   ,S                  ; decrement count of
grunts on stack
38F3: 26 BC       BNE   $38B1               ; if !=0 then we've got
more grunts to process, goto $38B1
38F5: 9F 8B       STX   $8B                 ; store index of last
grunt in $8B.
38F7: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
38FA: 39 B7          ; pointer to function
38FC: 35 82          PULS A, PC ;(PUL? PC=RTS)




;
; When a wave starts, there is an invisible rectangular "safe" area
containing the player that the enemies cannot be placed into.
; this is to give the player a fighting chance. As you advance
through waves the safe area gets smaller.
;
;

COMPUTE_PLAYER_SAFE_RECTANGLE:
38FE: 34 56       PSHS  U,X,B,A
3900: BD D0 45    JSR   $D045               ; JMP $D699 - get addr
of current player game state into X
3903: A6 09       LDA   $0009,X             ; read wave number
3905: 81 0A       CMPA  #$0A                ; compare to #$0A (10
decimal)
3907: 25 04       BCS   $390D               ; if < goto $390D
3909: 86 06       LDA   #$06                ; ok, wave number is 10
or more, so use wave 6's information
390B: 20 06       BRA   $3913
```

```
390D: 81 05       CMPA  #$05                ; compare wave number
to 5
390F: 23 02       BLS   $3913               ; if <= 5 goto $3913
3911: 86 05       LDA   #$05
3913: 8E 39 20    LDX   #$3920              ; address of safe area
rectangle list. Each rectangle occupies 4 bytes.
3916: 48          ASLA
3917: 48          ASLA                      ; multiply wave number
in A by 4
3918: 30 86       LEAX  A,X                 ; X+= A
391A: EC 84       LDD   ,X
391C: DD 2B       STD   $2B
391E: EC 02       LDD   $0002,X
3920: DD 2D       STD   $2D
3922: 35 D6       PULS  A,B,X,U,PC ;(PUL? PC=RTS)


; wave 1 safe area
3924: 40 B0
3926: 1A 7A

; wave 2 safe area
3928: 48 A8
392A: 1A 7A

; wave 3 safe area
392C: 50 A0
392E: 2A 6A

; wave 4 safe area
3930: 54 9D
3932: 30 60

; wave 5 safe area
3934: 5D 96
3936: 35 59

; wave 6 safe area
3938: 62 94
393A: 38 5C



;
; Draw an object in a solid colour that the player can't see.
;
; This function is called during the wave setup when objects like
baddies and family members are given their initial places.
; It draws the objects in a colour that we can't see, but the system
can detect, to ensure no objects overlap at wave start.
;
; We do not want objects to be stacked on top of other objects - for
example a grunt being stacked on top of an electrode, that would
```

```
kill them both!
;

BLIT_IN_SOLID_COLOUR_INVISIBLE_TO_PLAYER:
393C: 34 26       PSHS  Y,B,A
393E: 86 66       LDA   #$66                ; solid colour
3940: 20 02       BRA   $3944               ; do solid and
transparent blit

3942: 34 26       PSHS  Y,B,A
3944: 97 2D       STA   $2D                 ; set solid colour
3946: EC 04       LDD   $0004,X             ; D = blitter
destination
3948: 10 AE 02    LDY   $0002,X             ; Y = pointer to
animation frame metadata
394B: BD D0 90    JSR   $D090               ; JMP $DA9E - do solid
and transparent blit
394E: 35 A6       PULS  A,B,Y,PC ;(PUL? PC=RTS)



INITIALISE_ALL_ELECTRODES:
3950: 34 70       PSHS  U,Y,X
3952: 8E B3 A4    LDX   #$B3A4
3955: 9F 27       STX   $27
3957: 31 84       LEAY  ,X                  ; Y = X (#$B3A4)
3959: 6F 80       CLR   ,X+
395B: 8C B3 E4    CMPX  #$B3E4
395E: 25 F9       BCS   $3959
3960: B6 BE 69    LDA   cur_electrodes      ; read number of
electrodes
3963: 34 02       PSHS  A                   ; save number on stack
3965: 27 4E       BEQ   $39B5               ; 0?
3967: BD D0 81    JSR   $D081               ; JMP $D2E7 - reserve
an electrode object entry
396A: DC 93       LDD   $93                 ; get current electrode
animation frame metadata pointer
396C: ED 02       STD   $0002,X             ; set current animation
frame metadata pointer
396E: ED 88 14    STD   $14,X               ; set previous
animation frame metadata pointer (previous = current)
3971: 8D 8B       BSR   $38FE               ; compute safe
rectangle for player
3973: DC 2B       LDD   $2B
3975: C3 03 FC    ADDD  #$03FC
3978: DD 2B       STD   $2B
397A: DC 2D       LDD   $2D
397C: C3 02 FD    ADDD  #$02FD
397F: DD 2D       STD   $2D
3981: BD 26 C3    JSR   $26C3               ; JMP $3199 - get
random position on playfield for object (returns: A = X coordinate,
B = Y coordinate)
3984: D1 2B       CMPB  $2B
3986: 23 0C       BLS   $3994
```

```
3988: D1 2C       CMPB  $2C
398A: 24 08       BCC   $3994
398C: 91 2D       CMPA  $2D
398E: 23 04       BLS   $3994
3990: 91 2E       CMPA  $2E
3992: 25 ED       BCS   $3981               ; position is invalid,
go recompute another position

3994: ED 04       STD   $0004,X             ; current blitter
destination
3996: A7 0A       STA   $000A,X             ; set object X
coordinate
3998: E7 0C       STB   $000C,X             ; set object Y
coordinate
399A: EE 04       LDU   $0004,X             ; U = blitter
destination of object
399C: EC 98 02    LDD   [$02,X]             ; D= width and height
of object
399F: BD D0 03    JSR   $D003               ; JMP $DE0F -
TEST_FOR_PIXELS_WITHIN_RECTANGLE
39A2: 26 DD       BNE   $3981               ; Z flag is non-zero,
pixels have been found, can't place grunt here, goto $3981
39A4: CC 3A A9    LDD   #$3AA9              ; Address of function
to handle electrode collision detection
39A7: ED 08       STD   $0008,X
39A9: 10 AF 06    STY   $0006,X             ; set pointer to object
metadata to Y
39AC: 31 22       LEAY  $0002,Y
39AE: BD 38 8B    JSR   $388B               ; JMP $393C - blit
electrode in solid colour invisible to player
39B1: 6A E4       DEC   ,S                  ; decrement electrode
count on stack
39B3: 26 B2       BNE   $3967
39B5: 35 F2       PULS  A,X,Y,U,PC ;(PUL? PC=RTS)

39B7: 96 59       LDA   $59
39B9: 85 7F       BITA  #$7F
39BB: 27 08       BEQ   $39C5
39BD: 86 02       LDA   #$02                ; delay before calling
function
39BF: 8E 39 B7    LDX   #$39B7              ; start of this
function
39C2: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

39C5: 86 0A       LDA   #$0A                ; delay before calling
function
39C7: 8E 39 CD    LDX   #$39CD              ; address of function
that moves the grunt
39CA: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call
```

```
;
; Called by $D1E0
;

GRUNT_AI:
39CD: 5F          CLRB
39CE: B6 BE 68    LDA   cur_grunts          ; read number of grunts
39D1: 34 06       PSHS  B,A
39D3: 27 0F       BEQ   $39E4               ; 0? if so, go to $39E4
- we're done
39D5: 9E 8B       LDX   $8B                 ; get pointer to FIRST
grunt
39D7: 20 02       BRA   $39DB

; here, X = pointer to a grunt object. The first two bytes of the
object point
; to the NEXT grunt object in the list, making it a forward-only
linked list type
; setup.

39D9: AE 84       LDX   ,X                  ; get pointer to NEXT
grunt
39DB: 6A 88 13    DEC   $13,X               ; decrement move
countdown counter
39DE: 27 06       BEQ   $39E6               ; if zero its time for
grunt to move
39E0: 6A E4       DEC   ,S                  ; decrement grunt count
(on the stack)
39E2: 26 F5       BNE   $39D9               ; go get next grunt
39E4: 20 7E       BRA   $3A64               ; we're done with
grunts

MOVE_GRUNT:
39E6: B6 BE 5C    LDA   $BE5C               ; read grunt speed
control field
39E9: BD D0 42    JSR   $D042               ; JMP $D6AC - multiply
A by a random number and put result in A
39EC: A7 88 13    STA   $13,X               ; set move countdown
counter to random number
39EF: E6 0C       LDB   $000C,X             ; get grunt Y
coordinate
39F1: D0 66       SUBB  $66                 ; subtract player_y
39F3: 22 08       BHI   $39FD               ; no carry? if so, go
to $39FD
39F5: C1 FE       CMPB  #$FE                ; is difference from
grunt Y to player Y >-2?
39F7: 22 16       BHI   $3A0F               ; yes, so don't make
any adjustments to grunt Y
39F9: C6 04       LDB   #$04                ; we're wanting to move
grunt +4 pixels down
39FB: 20 06       BRA   $3A03
39FD: C1 02       CMPB  #$02                ; is difference from
grunt Y to player Y <2 ?
39FF: 25 0E       BCS   $3A0F               ; yes, so don't make
```

```
any adjustments to grunt Y
3A01: C6 FC       LDB   #$FC                ; move grunt -4 pixels
up
3A03: EB 0C       ADDB  $000C,X             ; add in grunt Y
coordinate
3A05: C1 DE       CMPB  #$DE                ; is grunt Y > $#DE?
(past bottom border)
3A07: 22 06       BHI   $3A0F               ; Yes, go to $3A0F, do
not update grunt Y coordinate
3A09: C1 18       CMPB  #$18                ; is grunt Y < $#18
(past top border)?
3A0B: 25 02       BCS   $3A0F               ; Yes, go to $3A0F, do
not update grunt Y coordinate

; if we get here, grunt Y position about to be updated
3A0D: E7 0C       STB   $000C,X             ; update grunt Y
coordinate

; now read the grunt's X position
3A0F: E6 0A       LDB   $000A,X             ; get grunt X
coordinate
3A11: D0 64       SUBB  $64                 ; subtract player_X
coordinate
3A13: 22 04       BHI   $3A19               ; no carry? if so, go
to $3A19
3A15: C6 02       LDB   #$02                ; we're wanting to move
grunt 2 pixels to the right
3A17: 20 06       BRA   $3A1F

3A19: C1 01       CMPB  #$01
3A1B: 25 0E       BCS   $3A2B
3A1D: C6 FE       LDB   #$FE                ; -2 bytes to left
(which is 4 pixels)
3A1F: EB 0A       ADDB  $000A,X             ; add in grunt's X
coordinate
3A21: C1 8A       CMPB  #$8A                ; is grunt X > $#8A
(past right border) ?
3A23: 22 06       BHI   $3A2B               ; Yes, go to $3A2B, do
not update grunt X coordinate
3A25: C1 07       CMPB  #$07                ; is grunt X < #$07
(past left border) ?
3A27: 25 02       BCS   $3A2B               ; Yes, go to $3A2B, do
not update grunt X coordinate
3A29: E7 0A       STB   $000A,X             ; update grunt X
coordinate

DRAW_GRUNT:
3A2B: EC 02       LDD   $0002,X             ; get animation frame
metadata pointer
3A2D: C3 00 04    ADDD  #$0004              ; add 4 to bump to next
animation frame's metadata (each metadata entry is 4 bytes long)
3A30: 10 83 40 6F CMPD  #$406F              ; got to invalid frame?
(meaning, past end of animation sequence)
3A34: 23 03       BLS   $3A39               ; no
```

```
3A36: CC 40 63    LDD   #$4063              ; reset blitter source
to animation start image's metadata
3A39: ED 02       STD   $0002,X             ; store animation frame
metadata pointer
3A3B: BD D0 8D    JSR   $D08D               ; JMP $DB2F - draw
grunt at new position
3A3E: 6C 61       INC   $0001,S
3A40: EE 02       LDU   $0002,X             ; get animation frame
metadata pointer
3A42: EC 04       LDD   $0004,X             ; set D to current
blitter destination
3A44: 34 10       PSHS  X                   ; push grunt object
pointer
3A46: 8E 98 23    LDX   #$9823              ; pointer to linked
list of electrodes (these kill grunts)
3A49: BD D0 27    JSR   $D027               ; JMP $D7C9 - collision
detection function
3A4C: 35 10       PULS  X                   ; restore object
pointer
3A4E: 27 0E       BEQ   $3A5E               ; if zero flag set, no
collision occurred, goto $3A5E
3A50: 10 AE 84    LDY   ,X                  ; Y = next grunt in
list
3A53: 8D 21       BSR   $3A76               ; call grunt collision
handler
3A55: 6A E4       DEC   ,S                  ; decrement grunt count
on stack
3A57: 27 0B       BEQ   $3A64               ; if 0, we're done with
grunts
3A59: 30 A4       LEAX  ,Y                  ; X = Y. So now X is
next grunt in list
3A5B: 7E 39 DB    JMP   $39DB               ; process the grunt at
X

3A5E: 6A E4       DEC   ,S                  ; decrement grunt count
on stack
3A60: 10 26 FF 75 LBNE  $39D9               ; if !=0 then process
grunt
3A64: EC E1       LDD   ,S++                ; restore A and B
3A66: 27 06       BEQ   $3A6E

3A68: CC 38 A0    LDD   #$38A0
3A6B: BD D0 4B    JSR   $D04B               ; JMP $D3C7
3A6E: 86 04       LDA   #$04                ; delay before calling
function
3A70: 8E 39 CD    LDX   #$39CD              ; address of function
to call
3A73: 7E D0 66    JMP   $D066               ;



;
; X = grunt that was in a collision
;
```

```
GRUNT_COLLISION_HANDLER:
3A76: 96 48       LDA   $48                 ; is it the player
collision detection routine invoking this handler?
3A78: 26 2C       BNE   $3AA6               ; yes, goto $3AA6
3A7A: BD 5B 43    JSR   $5B43               ; JMP $5C1F - create an
explosion
3A7D: 9C 8B       CPX   $8B                 ; compare X to last
grunt created pointer
3A7F: 26 04       BNE   $3A85               ; if != then goto $3A85
3A81: EC 84       LDD   ,X                  ; get pointer to next
object in object list (which must be a grunt)
3A83: DD 8B       STD   $8B                 ; and store in $8B -
the pointer to the grunt list
3A85: BD D0 7E    JSR   $D07E               ; JMP $D2C2 - remove
baddy from baddies list
3A88: CC 01 10    LDD   #$0110
3A8B: BD D0 0C    JSR   $D00C               ; JMP $DB9C - update
player score
3A8E: CC 38 98    LDD   #$3898              ; pointer to data to
use
3A91: BD D0 4B    JSR   $D04B               ; JMP $D3C7
3A94: C6 E0       LDB   #$E0
3A96: B6 BE 5C    LDA   $BE5C               ; read grunt speed
control field.
3A99: 3D          MUL
3A9A: B1 BE 5D    CMPA  $BE5D               ; compare to grunt
movement delay minimum (the lower this value is, the faster the
grunts can move)
3A9D: 25 03       BCS   $3AA2               ; if A <
3A9F: B7 BE 5C    STA   $BE5C               ; set grunt speed
control field. This will increase the grunts speed as more die
3AA2: 7A BE 68    DEC   cur_grunts          ; reduce grunt count
3AA5: 39          RTS

3AA6: 7E D0 18    JMP   $D018               ; JMP   $DAF2 - do blit


;
;

ELECTRODE_COLLISION_HANDLER:
3AA9: 96 48       LDA   $48                 ; was it the player
that called this routine?
3AAB: 26 27       BNE   $3AD4               ; yes, so do nothing,
just exit
3AAD: BD D0 84    JSR   $D084               ; deallocate electrode
object
3AB0: CC 00 00    LDD   #$0000
3AB3: ED 98 06    STD   [$06,X]
3AB6: BD D0 15    JSR   $D015               ; JMP $DB03 - erase
object from screen
3AB9: 7A BE 69    DEC   cur_electrodes
3ABC: DC 13       LDD   $13
```

```
3ABE: 27 13       BEQ   $3AD3
3AC0: 33 84       LEAU  ,X
3AC2: AE 84       LDX   ,X
3AC4: 9F 1B       STX   $1B
3AC6: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
3AC9: 3A D9       ; pointer to function
3ACB: EF 07       STU   $7,X
3ACD: CC 38 A5    LDD   #$38A5
3AD0: 7E D0 4B    JMP   $D04B
3AD3: 39          RTS

3AD4: 96 90       LDA   $90
3AD6: 7E 38 88    JMP   $3888               ; JMP   $3942

3AD9: AE 47       LDX   $0007,U
3ADB: 10 AE 02    LDY   $0002,X
3ADE: 20 17       BRA   $3AF7               ; would have been
better going to 3AFA, it just stores same value as it read


;
; this code is called when an electrode dies
;

ELECTRODE_DEATH:
3AE0: AE 47       LDX   $0007,U             ; get object pointer
3AE2: 10 AE 02    LDY   $0002,X             ; load y with animation
frame metadata pointer
3AE5: 31 25       LEAY  $0005,Y
3AE7: A6 A4       LDA   ,Y
3AE9: 26 0C       BNE   $3AF7
3AEB: BD D0 15    JSR   $D015               ; JMP $DB03 - erase
object from screen
3AEE: DC 1B       LDD   $1B
3AF0: ED 84       STD   ,X
3AF2: 9F 1B       STX   $1B
3AF4: 7E D0 63    JMP   $D063               ; JMP $D1F3
3AF7: 10 AF 02    STY   $0002,X
3AFA: BD D0 8D    JSR   $D08D               ; JMP $DB2F - draw
object
3AFD: A6 24       LDA   $0004,Y
3AFF: 8E 3A E0    LDX   #$3AE0              ; address of routine to
call next
3B02: 7E D0 66    JMP   $D066

3B05: 05 09 3B 95 06 05 09 3B C2 03 05 09 3B EF 02 00
3B15: 05 09 3C 1C 06 05 09 3C 49 03 05 09 3C 76 02 00
3B25: 05 09 3C A3 06 05 09 3C D0 03 05 09 3C FD 02 00
3B35: 05 09 3D 2A 06 05 09 3D 57 03 05 09 3D 84 02 00
3B45: 03 09 3D B1 06 03 09 3D CC 03 03 09 3D E7 02 00
3B55: 05 09 3E 02 06 05 09 3E 2F 03 05 09 3E 5C 02 00
3B65: 09 07 3E 89 06 09 07 3E C8 03 09 07 3F 07 02 00
3B75: 05 09 3F 46 06 05 09 3F 73 03 05 09 3F A0 02 00
```

```
3B85: 05 0A 3F CD 06 05 0A 3F FF 03 05 0A 40 31 02 00
3B95: 00 00 90 00 00 09 00 90 09 00 00 90 90 90 00 00
3BA5: 09 99 00 00 99 99 99 99 90 00 09 99 00 00 00 90
3BB5: 90 90 00 09 00 90 09 00 00 00 90 00 00 00 00 00
3BC5: 00 00 00 00 A0 00 00 00 A0 A0 A0 00 00 0A AA 00
3BD5: 00 0A AA AA AA 00 00 0A AA 00 00 00 A0 A0 A0 00
3BE5: 00 00 A0 00 00 00 00 00 00 00 00 00 00 00 00 00
3BF5: 00 00 00 00 00 00 00 00 00 00 00 A0 00 00 00 0A
3C05: AA 00 00 00 00 A0 00 00 00 00 00 00 00 00 00 00
3C15: 00 00 00 00 00 00 00 00 00 90 00 00 09 09 09 09
3C25: 00 00 90 90 90 00 09 09 09 09 00 90 90 00 90 90
3C35: 09 09 09 09 00 00 90 90 90 00 09 09 09 09 00 00
3C45: 00 90 00 00 00 00 00 00 00 00 00 A0 00 00 00 0A
3C55: 0A 00 00 00 A0 A0 A0 00 0A 0A 0A 0A 00 00 A0 A0
3C65: A0 00 00 0A 0A 00 00 00 00 A0 00 00 00 00 00 00
3C75: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3C85: 00 00 A0 00 00 00 0A AA 00 00 00 00 A0 00 00 00
3C95: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 99 99
3CA5: 99 99 90 99 99 99 99 90 99 99 99 99 90 99 99 99
3CB5: 99 90 99 99 99 99 90 99 99 99 99 90 99 99 99 99
3CC5: 90 99 99 99 99 90 99 99 99 99 90 00 00 00 00 00
3CD5: 0A AA AA AA 00 0A AA AA AA 00 0A AA AA AA 00 0A
3CE5: AA AA AA 00 0A AA AA AA 00 0A AA AA AA 00 0A AA
3CF5: AA AA 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3D05: 00 00 00 00 00 00 00 00 0A AA 00 00 00 0A AA 00
3D15: 00 00 0A AA 00 00 00 00 00 00 00 00 00 00 00 00
3D25: 00 00 00 00 00 00 00 00 00 90 00 00 00 09 90 00
3D35: 00 00 99 90 00 00 09 99 90 00 00 99 99 90 00 09
3D45: 99 99 90 00 99 99 99 90 09 99 99 99 90 99 99 99
3D55: 99 90 00 00 00 00 00 00 00 00 00 00 00 00 00 0A
3D65: 00 00 00 00 AA 00 00 00 0A AA 00 00 00 AA AA 00
3D75: 00 0A AA AA 00 00 AA AA AA 00 00 00 00 00 00 00
3D85: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3D95: 00 00 00 00 00 00 A0 00 00 00 0A A0 00 00 00 AA
3DA5: A0 00 00 00 00 00 00 00 00 00 00 00 99 99 90 99
3DB5: 99 90 99 99 90 99 99 90 99 99 90 99 99 90 99 99
3DC5: 90 99 99 90 99 99 90 00 00 00 0A AA 00 0A AA 00
3DD5: 0A AA 00 0A AA 00 0A AA 00 0A AA 00 0A AA 00 00
3DE5: 00 00 00 00 00 00 00 00 00 A0 00 00 A0 00 00 A0
3DF5: 00 00 A0 00 00 A0 00 00 00 00 00 00 00 00 00 90
3E05: 00 00 00 09 99 00 00 00 99 99 90 00 09 99 99 99
3E15: 00 99 99 99 99 90 09 99 99 99 00 00 99 99 90 00
3E25: 00 09 99 00 00 00 00 90 00 00 00 00 00 00 00 00
3E35: 00 A0 00 00 00 0A AA 00 00 00 AA AA A0 00 0A AA
3E45: AA AA 00 00 AA AA A0 00 00 0A AA 00 00 00 00 A0
3E55: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3E65: 00 00 00 00 00 00 00 00 A0 00 00 00 0A AA 00 00
3E75: 00 00 A0 00 00 00 00 00 00 00 00 00 00 00 00 00
3E85: 00 00 00 00 99 99 99 99 99 99 99 99 90 90 00 90
3E95: 00 90 00 90 90 90 99 90 90 90 90 90 90 90 90 90
3EA5: 00 90 90 90 00 90 00 90 90 99 90 90 90 90 99 90
3EB5: 90 90 00 90 00 90 00 99 90 90 99 99 99 99 99 99
3EC5: 99 99 90 00 00 00 00 00 00 00 00 00 00 0A AA AA AA
3ED5: AA AA AA AA 00 0A A0 A0 A0 A0 A0 A0 AA 00 0A 00
```

```
3EE5: A0 A0 A0 00 00 0A 00 0A AA A0 A0 A0 A0 AA AA 00
3EF5: 0A AA AA AA AA AA AA AA 00 00 00 00 00 00 00 00
3F05: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3F15: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0A AA
3F25: AA AA AA AA 00 00 00 00 00 00 00 00 00 00 00 00
3F35: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3F45: 00 00 00 90 00 00 09 99 99 99 00 09 99 09 99 00
3F55: 09 90 00 99 00 99 00 00 09 90 09 90 00 99 00 09
3F65: 99 09 99 00 09 99 99 99 00 00 00 90 00 00 00 00
3F75: 00 00 00 00 00 A0 00 00 00 AA AA A0 00 00 AA 0A
3F85: A0 00 0A A0 00 AA 00 00 AA 0A A0 00 00 AA AA A0
3F95: 00 00 00 A0 00 00 00 00 00 00 00 00 00 00 00 00
3FA5: 00 00 00 00 00 00 00 A0 00 00 00 0A AA 00 00 00
3FB5: AA 0A A0 00 00 0A AA 00 00 00 00 A0 00 00 00 00
3FC5: 00 00 00 00 00 00 00 00 99 99 99 99 90 90 00 00
3FD5: 00 90 90 99 99 90 90 90 90 00 90 90 90 90 90 90
3FE5: 90 90 99 90 90 90 90 00 00 90 90 99 99 99 90 90
3FF5: 00 00 00 00 90 09 99 99 99 90 00 00 00 00 00 00
4005: 00 00 00 00 00 AA AA A0 00 00 A0 00 A0 00 A0 A0
4015: A0 A0 00 A0 AA A0 A0 00 A0 00 00 A0 00 AA AA AA
4025: A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
4035: 00 00 00 00 00 00 00 AA AA A0 00 00 A0 00 A0 00
4045: 00 A0 A0 A0 00 00 AA A0 00 00 00 00 00 00 00 00
4055: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 05 0D
4065: 40 73 05 0D 40 B4 05 0D 40 73 05 0D 40 F5 00 01
4075: 11 00 00 00 66 66 60 00 00 CC CC C0 00 00 01 11
4085: 00 00 11 91 11 91 10 51 19 99 11 50 50 11 91 10
4095: 50 50 01 11 00 50 00 01 11 00 00 00 11 01 10 00
40A5: 00 11 01 10 00 05 55 05 55 00 00 00 00 00 00 00
40B5: 01 11 00 00 00 66 66 60 00 00 CC CC C0 00 00 01
40C5: 11 00 00 11 91 11 91 10 51 19 99 11 50 50 11 91
40D5: 10 50 50 01 11 00 50 00 11 11 00 00 00 11 01 10
40E5: 00 05 55 01 10 00 00 00 01 10 00 00 00 05 55 00
40F5: 00 01 11 00 00 00 66 66 60 00 00 CC CC C0 00 00
4105: 01 11 00 00 11 91 11 91 10 51 19 99 11 50 50 11
4115: 91 10 50 50 01 11 00 50 00 01 11 10 00 00 11 01
4125: 10 00 00 11 05 55 00 00 11 00 00 00 05 55 00 00
4135: 00 FF FF FF FF FF FF FF FF FF FF

4140: 7E 45 9B    JMP   $459B

4143: FF 01
4145: 01 13 00 99 22 55 11 99 22 55 11 99 22 55 11 99
4155: 22 55 11 99 22 55 11 AA CC AA CC AA CC AA CC AA
4165: CC AA CC AA CC AA CC AA CC AA CC 99 77 99 77 99
4175: 77 99 77 99 77 99 77 99 77 99 77 99 77 99 77 11
4185: 55 11 55 11 55 11 55 11 55 11 55 11 55 11 55 11
4195: 55 11 55 FF EE DD CC BB AA FF EE DD CC BB AA FF
41A5: EE DD CC BB AA FF EE 11 66 77 BB AA 11 66 77 BB
41B5: AA 11 66 77 BB AA 11 66 77 BB AA 33 55 33 55 AA
41C5: 33 55 33 55 AA 33 55 33 55 AA 33 55 33 55 AA 41
41D5: 48 41 5C 41 70 41 84 41 98 41 AC 41 C0 41 98

41E4: 20 43       BRA   $4229
```

```
41E6: OPYRIGHT 1982 WILLIAMS ELECTRONI
4206: CS INC.

420E: 34 30       PSHS  Y,X
4210: 8E B3 ED    LDX   #$B3ED
4213: BF B3 E4    STX   $B3E4
4216: 31 89 00 81 LEAY  $0081,X
421A: 10 AF 84    STY   ,X
421D: 10 8C B8 F7 CMPY  #$B8F7
4221: 24 04       BCC   $4227
4223: 30 A4       LEAX  ,Y
4225: 20 EF       BRA   $4216

4227: 10 8E 00 00 LDY   #$0000
422B: 10 AF 84    STY   ,X
422E: 10 BF B3 E6 STY   $B3E6
4232: 35 B0       PULS  X,Y,PC ;(PUL? PC=RTS)

4234: 34 66       PSHS  U,Y,B,A
4236: 8D 37       BSR   $426F
4238: 25 27       BCS   $4261
423A: 10 BF B3 E6 STY   $B3E6
423E: AF 26       STX   $0006,Y
4240: EC 84       LDD   ,X
4242: ED 22       STD   $0002,Y
4244: 3D          MUL
4245: E7 28       STB   $0008,Y
4247: 33 2D       LEAU  $000D,Y
4249: EF 24       STU   $0004,Y
424B: 8D 16       BSR   $4263
424D: CE 43 38    LDU   #$4338
4250: EF 29       STU   $0009,Y
4252: BD D0 39    JSR   $D039               ; JMP $D6CD - get a
random number into A
4255: 84 07       ANDA  #$07
4257: 48          ASLA
4258: CE 41 D4    LDU   #$41D4
425B: EC C6       LDD   A,U
425D: ED 2B       STD   $000B,Y
425F: 30 22       LEAX  $0002,Y
4261: 35 E6       PULS  A,B,Y,U,PC ;(PUL? PC=RTS)

4263: 34 26       PSHS  Y,B,A
4265: 10 AE 24    LDY   $0004,Y
4268: 6F A0       CLR   ,Y+
426A: 5A          DECB
426B: 26 FB       BNE   $4268
426D: 35 A6       PULS  A,B,Y,PC ;(PUL? PC=RTS)

426F: 34 10       PSHS  X
4271: 10 BE B3 E4 LDY   $B3E4
4275: 27 0E       BEQ   $4285
4277: AE A4       LDX   ,Y
```

```
4279: BF B3 E4    STX   $B3E4
427C: BE B3 E6    LDX   $B3E6
427F: AF A4       STX   ,Y
4281: 1C FE       ANDCC #$FE                ; clear carry flag
4283: 35 90       PULS  X,PC ;(PUL? PC=RTS)

4285: 1A 01       ORCC  #$01
4287: 35 90       PULS  X,PC ;(PUL? PC=RTS)

4289: AE 29       LDX   $0009,Y
428B: 34 10       PSHS  X
428D: 30 01       LEAX  $0001,X
428F: AF 29       STX   $0009,Y
4291: 96 84       LDA   $84
4293: 84 07       ANDA  #$07
4295: AE 2B       LDX   $000B,Y
4297: 30 86       LEAX  A,X
4299: BF B3 E8    STX   highscore
429C: AE 26       LDX   $0006,Y
429E: E6 28       LDB   $0008,Y
42A0: 35 C0       PULS  U,PC ;(PUL? PC=RTS)

42A2: 34 76       PSHS  U,Y,X,B,A
42A4: AE 02       LDX   $0002,X
42A6: 10 AE 24    LDY   $0004,Y
42A9: A6 C4       LDA   ,U
42AB: 2A 13       BPL   $42C0
42AD: 84 7F       ANDA  #$7F
42AF: BD 42 F5    JSR   $42F5
42B2: EC C1       LDD   ,U++
42B4: B1 B3 EA    CMPA  hs_inits
42B7: 24 1C       BCC   $42D5
42B9: 53          COMB
42BA: E4 A6       ANDB  A,Y
42BC: E7 A6       STB   A,Y
42BE: 20 F2       BRA   $42B2

42C0: 85 40       BITA  #$40
42C2: 26 13       BNE   $42D7
42C4: 8D 2F       BSR   $42F5
42C6: EC C1       LDD   ,U++
42C8: B1 B3 EA    CMPA  hs_inits
42CB: 24 08       BCC   $42D5
42CD: E4 86       ANDB  A,X
42CF: EA A6       ORB   A,Y
42D1: E7 A6       STB   A,Y
42D3: 20 F1       BRA   $42C6

42D5: 35 F6       PULS  A,B,X,Y,U,PC ;(PUL? PC=RTS)

42D7: 84 3F       ANDA  #$3F
42D9: 8D 1A       BSR   $42F5
42DB: EC C1       LDD   ,U++
42DD: B1 B3 EA    CMPA  hs_inits
```

```
42E0: 24 F3       BCC   $42D5
42E2: E4 86       ANDB  A,X
42E4: 27 F5       BEQ   $42DB
42E6: E6 9F B3 E8 LDB   [$B3E8,X]
42EA: 7C B3 E9    INC   $B3E9
42ED: E4 5F       ANDB  $FFFF,U
42EF: EA A6       ORB   A,Y
42F1: E7 A6       STB   A,Y
42F3: 20 E6       BRA   $42DB

42F5: CE 43 BB    LDU   #$43BB
42F8: 4A          DECA
42F9: 48          ASLA
42FA: EE C6       LDU   A,U
42FC: F7 B3 EA    STB   hs_inits
42FF: 39          RTS

4300: 34 10       PSHS  X
4302: AE A4       LDX   ,Y
4304: AF 9F B3 EB STX   [$B3EB,X]
4308: BE B3 E4    LDX   $B3E4
430B: AF A4       STX   ,Y
430D: 10 BF B3 E4 STY   $B3E4
4311: 10 BE B3 EB LDY   $B3EB
4315: 35 90       PULS  X,PC ;(PUL? PC=RTS)

4317: 10 8E B3 E6 LDY   #$B3E6
431B: 20 0C       BRA   $4329

431D: BD 42 89    JSR   $4289
4320: 11 83 43 BA CMPU  #$43BA
4324: 27 0D       BEQ   $4333
4326: BD 42 A2    JSR   $42A2
4329: 10 BF B3 EB STY   $B3EB
432D: 10 AE A4    LDY   ,Y
4330: 26 EB       BNE   $431D
4332: 39          RTS

4333: BD 43 00    JSR   $4300
4336: 20 F1       BRA   $4329

4338: 43          COMA
4339: 83 46 86    SUBD  #$4686
433C: 44          LSRA
433D: 84 42       ANDA  #$42
433F: 82 41       SBCA  #$41
4341: 81 47       CMPA  #$47
4343: 87          Illegal Opcode
4344: 44          LSRA
4345: 46          RORA
4346: 84 42       ANDA  #$42
4348: 86 43       LDA   #$43
434A: 82 41       SBCA  #$41
434C: 83 45 81    SUBD  #$4581
```

```
434F: 48          ASLA
4350: 85 47       BITA  #$47
4352: 88 44       EORA  #$44
4354: 87          Illegal Opcode
4355: 45          Illegal Opcode
4356: 42          Illegal Opcode
4357: 84 46       ANDA  #$46
4359: 85 47       BITA  #$47
435B: 82 48       SBCA  #$48
435D: 86 43       LDA   #$43
435F: 87          Illegal Opcode
4360: 41          Illegal Opcode
4361: 88 42       EORA  #$42
4363: 83 44 81    SUBD  #$4481
4366: 45          Illegal Opcode
4367: 47          ASRA
4368: 82 46       SBCA  #$46
436A: 84 85       ANDA  #$85
436C: 48          ASLA
436D: 87          Illegal Opcode
436E: 43          COMA
436F: 86 44       LDA   #$44
4371: 42          Illegal Opcode
4372: 88 45       EORA  #$45
4374: 83 41 84    SUBD  #$4184
4377: 43          COMA
4378: 82 47       SBCA  #$47
437A: 46          RORA
437B: 85 48       BITA  #$48
437D: 81 42       CMPA  #$42
437F: 83 44 87    SUBD  #$4487
4382: 41          Illegal Opcode
4383: 86 45       LDA   #$45
4385: 88 43 46 84 48 81 47 85 44 82 41 45 83 42 86 88
4395: 43 87 46 84 48 85 07 81 04 05 82 01 83 02 86 03
43A5: 88 06 87 08 84 07 85 04 81 05 01 82 83 02 03 88
43B5: 08 84 87 07 04 00 43 CB 44 05 44 3F 44 79 44 B3
43C5: 44 ED 45 27 45 61 00 F0 06 0F 09 0F 0C F0 13 F0
43D5: 17 0F 18 0F 1D F0 22 F0 25 0F 2B F0 2E 0F 30 F0
43E5: 32 0F 3C F0 3E 0F 43 0F 45 F0 48 F0 4C 0F 50 0F
43F5: 51 F0 5A F0 5D 0F 60 0F 65 F0 6A 0F 6E 0F 73 F0
4405: 01 0F 03 F0 08 F0 0F 0F 12 F0 14 0F 1B 0F 1E F0
4415: 20 F0 22 0F 2D F0 2F 0F 33 0F 35 F0 39 0F 3D F0
4425: 43 F0 47 0F 4A 0F 4E F0 53 F0 55 0F 5B 0F 5B F0
4435: 63 0F 68 F0 6D F0 71 0F 73 0F 04 0F 07 F0 09 F0
4445: 0A 0F 14 F0 15 0F 1C 0F 1F F0 21 F0 23 0F 29 0F
4455: 2E F0 34 0F 36 F0 3A F0 3D 0F 42 F0 44 0F 49 0F
4465: 4D F0 55 F0 57 0F 59 0F 5C F0 61 F0 68 0F 6A F0
4475: 6D 0F 72 0F 00 0F 02 F0 0A F0 0C 0F 10 F0 16 0F
4485: 1A F0 1E 0F 21 0F 23 F0 2C 0F 2F F0 33 F0 36 0F
4495: 38 F0 3B 0F 42 0F 46 F0 49 F0 4D 0F 52 F0 54 0F
44A5: 5A 0F 5D F0 61 0F 64 F0 69 0F 6C F0 70 F0 02 0F
44B5: 06 F0 08 0F 0E F0 13 0F 16 F0 19 F0 1D 0F 24 0F
44C5: 27 F0 29 F0 2B 0F 30 0F 31 F0 3C 0F 3E F0 40 0F
```

```
44D5: 41 F0 4B F0 4E 0F 52 0F 57 F0 59 F0 5E 0F 62 F0
44E5: 65 0F 67 F0 6B 0F 6F F0 01 F0 07 0F 0B 0F 0D F0
44F5: 11 0F 15 F0 1A 0F 1C F0 20 0F 25 F0 2A F0 2D 0F
4505: 31 0F 32 F0 38 0F 3F F0 44 F0 46 0F 4B 0F 4F F0
4515: 54 F0 56 0F 58 0F 5F F0 60 F0 66 F0 6C 0F 6F 0F
4525: 71 F0 03 0F 04 F0 0D 0F 0F F0 11 F0 12 0F 18 F0
4535: 1F 0F 26 F0 27 0F 28 0F 2C F0 34 F0 37 0F 3B F0
4545: 3F 0F 40 F0 45 0F 48 0F 4A F0 53 0F 56 F0 58 F0
4555: 5C 0F 62 0F 64 0F 6B F0 70 0F 72 F0 05 F0 05 0F
4565: 0B F0 0E 0F 10 0F 17 F0 19 0F 1B F0 24 F0 26 0F
4575: 28 F0 2A 0F 35 0F 37 F0 39 F0 3A 0F 41 0F 47 F0
4585: 4C F0 4F 0F 50 F0 51 0F 5E F0 5F 0F 63 F0 66 0F
4595: 67 0F 69 F0 6E F0


;
; This code is called during a brain wave
;
;

459B: BD 42 0E    JSR   $420E
459E: 9E 21       LDX   $21                 ; get pointer to linked
list
45A0: 27 52       BEQ   $45F4
45A2: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
45A5: 46 07       ; pointer to function
45A7: 9E 21       LDX   $21
45A9: CC 00 00    LDD   #$0000
45AC: 10 8E 00 00 LDY   #$0000
45B0: 34 06       PSHS  B,A
45B2: 34 06       PSHS  B,A
45B4: EC 02       LDD   $0002,X             ; set current animation
frame metadata pointer
45B6: ED 88 14    STD   $14,X               ; set previous
animation frame metadata pointer (previous = current)
45B9: 31 21       LEAY  $0001,Y
45BB: 10 8C 00 0F CMPY  #$000F
45BF: 22 05       BHI   $45C6
45C1: 10 A3 E4    CMPD  ,S
45C4: 27 11       BEQ   $45D7
45C6: 10 8E 00 00 LDY   #$0000
45CA: ED E4       STD   ,S
45CC: 34 10       PSHS  X
45CE: AE 02       LDX   $0002,X
45D0: BD 42 34    JSR   $4234
45D3: AF 64       STX   $0004,S
45D5: 35 10       PULS  X
45D7: EC 62       LDD   $0002,S
45D9: ED 02       STD   $0002,X
45DB: AE 84       LDX   ,X
45DD: 26 D5       BNE   $45B4
45DF: 32 64       LEAS  $0004,S
45E1: BD 46 39    JSR   $4639
```

```
45E4: BD 43 17    JSR   $4317
45E7: BE B3 E6    LDX   $B3E6
45EA: 27 08       BEQ   $45F4
45EC: 86 01       LDA   #$01
45EE: 8E 45 E1    LDX   #$45E1
45F1: 7E D0 66    JMP   $D066

45F4: 9E 21       LDX   $21
45F6: 27 09       BEQ   $4601
45F8: EC 88 14    LDD   $14,X               ; get previous
animation frame metadata pointer
45FB: ED 02       STD   $0002,X             ; set current animation
frame metadata pointer (current = previous)
45FD: AE 84       LDX   ,X
45FF: 26 F7       BNE   $45F8
4601: BD F0 09    JSR   $F009               ; clear explosion list
4604: 7E D0 63    JMP   $D063               ; JMP $D1F3

4607: CC 41 43    LDD   #$4143
460A: BD D0 4B    JSR   $D04B               ; JMP $D3C7
460D: 86 48       LDA   #$48
460F: A7 47       STA   $0007,U
4611: C6 12       LDB   #$12
4613: BD D0 06    JSR   $D006
4616: 6A 47       DEC   $0007,U
4618: 27 08       BEQ   $4622
461A: 86 01       LDA   #$01
461C: 8E 46 11    LDX   #$4611
461F: 7E D0 66    JMP   $D066

4622: 86 24       LDA   #$24
4624: A7 47       STA   $0007,U
4626: C6 12       LDB   #$12
4628: BD D0 06    JSR   $D006
462B: 6A 47       DEC   $0007,U
462D: 10 27 8A 32 LBEQ  $D063               ; JMP $D1F3
4631: 86 02       LDA   #$02
4633: 8E 46 26    LDX   #$4626
4636: 7E D0 66    JMP   $D066



4639: 9E 21       LDX   $21                 ; get pointer to list
463B: 27 23       BEQ   $4660
463D: 10 AE 02    LDY   $0002,X             ; get animation frame
metadata pointer
4640: EC A4       LDD   ,Y                  ; A = width, B = height
4642: 88 04       EORA  #$04                ; necessary to xor
width and height for the blitter op
4644: C8 04       EORB  #$04
4646: 1A 10       ORCC  #$10                ; disable interrupts
4648: FD CA 06    STD   blitter_w_h
464B: EE 22       LDU   $0002,Y             ; get actual image data
pointer into U
```

```
464D: FF CA 02    STU   blitter_source
4650: EC 04       LDD   $0004,X             ; get current object's
blit destination
4652: FD CA 04    STD   blitter_dest
4655: 86 06       LDA   #$06                ; sync with E clock
(for blit from RAM to RAM)
4657: B7 CA 00    STA   start_blitter
465A: 1C EF       ANDCC #$EF                ; clear interrupt flag
465C: AE 84       LDX   ,X                  ; get next object in
the list
465E: 26 DD       BNE   $463D               ; if not null then
process it
4660: 39          RTS

4661: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
4671: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

4680: 7E 46 8C    JMP   $468C

4683: 7E 47 3F    JMP   $473F

4686: 7E 46 E6    JMP   $46E6

4689: 7E 4A 79    JMP   $4A79

468C: 34 30       PSHS  Y,X
468E: 8E BB E4    LDX   #$BBE4
4691: 9F C0       STX   $C0
4693: 31 88 33    LEAY  $33,X
4696: 10 AF 84    STY   ,X
4699: 10 8C BD E2 CMPY  #$BDE2
469D: 24 04       BCC   $46A3
469F: 30 A4       LEAX  ,Y
46A1: 20 F0       BRA   $4693

46A3: 10 8E 00 00 LDY   #$0000
46A7: 10 AF 84    STY   ,X
46AA: 10 9F BC    STY   $BC
46AD: 10 9F BE    STY   $BE
46B0: 35 B0       PULS  X,Y,PC ;(PUL? PC=RTS)


;
;
;
;

46B2: 34 20       PSHS  Y
46B4: DE C0       LDU   $C0
46B6: 27 12       BEQ   $46CA
46B8: 10 AE C4    LDY   ,U
46BB: 10 9F C0    STY   $C0
46BE: 10 9E BC    LDY   $BC
46C1: 10 AF C4    STY   ,U
```

```
46C4: DF BC       STU   $BC
46C6: 1C FE       ANDCC #$FE                ; clear carry flag to
indicate that this function succeeded
46C8: 35 A0       PULS  Y,PC ;(PUL? PC=RTS)
46CA: 1A 01       ORCC  #$01                ; set carry flag to
indicate that this function failed
46CC: 35 A0       PULS  Y,PC ;(PUL? PC=RTS)


46CE: 34 20       PSHS  Y
46D0: DE C0       LDU   $C0
46D2: 27 F6       BEQ   $46CA
46D4: 10 AE C4    LDY   ,U
46D7: 10 9F C0    STY   $C0
46DA: 10 9E BE    LDY   $BE
46DD: 10 AF C4    STY   ,U
46E0: DF BE       STU   $BE
46E2: 1C FE       ANDCC #$FE                ; clear carry flag
46E4: 35 A0       PULS  Y,PC ;(PUL? PC=RTS)

46E6: 34 76       PSHS  U,Y,X,B,A
46E8: BD 46 CE    JSR   $46CE
46EB: 25 40       BCS   $472D
46ED: A7 C8 12    STA   $12,U
46F0: EC 04       LDD   $0004,X
46F2: AE 02       LDX   $0002,X
46F4: ED 4B       STD   $000B,U
46F6: A7 44       STA   $0004,U
46F8: D6 A7       LDB   $A7
46FA: E7 45       STB   $0005,U
46FC: E0 4C       SUBB  $000C,U
46FE: 25 04       BCS   $4704
4700: E1 01       CMPB  $0001,X
4702: 25 0B       BCS   $470F
4704: E6 84       LDB   ,X
4706: 54          LSRB
4707: E7 46       STB   $0006,U
4709: EB 4C       ADDB  $000C,U
470B: E7 45       STB   $0005,U
470D: 20 02       BRA   $4711

470F: E7 46       STB   $0006,U
4711: EC 84       LDD   ,X
4713: ED 4D       STD   $000D,U
4715: C6 01       LDB   #$01
4717: E7 C8 11    STB   $11,U
471A: 88 04       EORA  #$04
471C: C8 04       EORB  #$04
471E: ED 4F       STD   $000F,U
4720: AE 02       LDX   $0002,X
4722: AF 42       STX   $0002,U
4724: CC 10 00    LDD   #$1000
4727: ED 48       STD   $0008,U
4729: 6F 47       CLR   $0007,U
```

```
472B: 8D 02       BSR   $472F
472D: 35 F6       PULS  A,B,X,Y,U,PC ;(PUL? PC=RTS)


;
; U = pointer to ???
;
;
;

472F: AE 42       LDX   $0002,U             ; X = pointer to pixel
data
4731: E6 4D       LDB   $000D,U             ; B = width of image
4733: A6 4E       LDA   $000E,U             ; A = height of image
(number of rows to proces)
4735: 33 C8 13    LEAU  $13,U
4738: AF C1       STX   ,U++                ; store address of
pixel row at U, then increment U by 2
473A: 3A          ABX                       ; X += width of image.
X now points to start of next pixel row in image
473B: 4A          DECA                      ; decrement row counter
473C: 26 FA       BNE   $4738               ; if not zero, goto
$4738
473E: 39          RTS


;
;
;
;
;

473F: 34 76       PSHS  U,Y,X,B,A
4741: BD 46 B2    JSR   $46B2               ; get next
available ??? for use
4744: 25 44       BCS   $478A               ; if carry is set,
function failed
4746: A7 C8 12    STA   $12,U
4749: EC 04       LDD   $0004,X             ; D = blitter
destination
474B: AE 02       LDX   $0002,X             ; X = pointer to
animation frame metadata
474D: ED 4B       STD   $000B,U             ; save blitter
destination
474F: A7 44       STA   $0004,U
4751: D6 A7       LDB   $A7
4753: E7 45       STB   $0005,U
4755: E0 4C       SUBB  $000C,U
4757: 25 04       BCS   $475D
4759: E1 01       CMPB  $0001,X             ;
475B: 25 0B       BCS   $4768               ; if B < height, goto
$4768
475D: E6 84       LDB   ,X                  ; B = height
475F: 54          LSRB                      ; B /= 2
```

```
4760: E7 46       STB   $0006,U
4762: EB 4C       ADDB  $000C,U
4764: E7 45       STB   $0005,U
4766: 20 02       BRA   $476A

4768: E7 46       STB   $0006,U
476A: EC 84       LDD   ,X                    ; A = width, B =
height
476C: ED 4D       STD   $000D,U
476E: E7 C8 11    STB   $11,U
4771: C6 01       LDB   #$01
4773: 88 04       EORA  #$04                   ; xor width with 4
4775: C8 04       EORB  #$04                   ; xor height with 4
(this must be precalculations for blitting)
4777: ED 4F       STD   $000F,U                ;
4779: AE 02       LDX   $0002,X                ; X = pointer to
pixel data for animation frame
477B: AF 42       STX   $0002,U
477D: CC 01 00    LDD   #$0100
4780: ED 48       STD   $0008,U
4782: 6F 47       CLR   $0007,U
4784: 86 10       LDA   #$10
4786: A7 4A       STA   $000A,U
4788: 8D A5       BSR   $472F                  ;
478A: 35 F6       PULS  A,B,X,Y,U,PC ;(PUL? PC=RTS)



; Y = pointer to list of
; D = Blitter destination
; U = blitter flags
; $BA = offset to add to D after each blit, to space out the
explosion segments

DRAW_EXPLOSION_SEGMENTS:
478C: AE A1       LDX   ,Y++
478E: BF CA 02    STX   blitter_source
4791: FD CA 04    STD   blitter_dest
4794: FF CA 00    STU   start_blitter
4797: D3 BA       ADDD  $BA
4799: AE A1       LDX   ,Y++
479B: BF CA 02    STX   blitter_source
479E: FD CA 04    STD   blitter_dest
47A1: FF CA 00    STU   start_blitter
47A4: D3 BA       ADDD  $BA
47A6: AE A1       LDX   ,Y++
47A8: BF CA 02    STX   blitter_source
47AB: FD CA 04    STD   blitter_dest
47AE: FF CA 00    STU   start_blitter
47B1: D3 BA       ADDD  $BA
47B3: AE A1       LDX   ,Y++
47B5: BF CA 02    STX   blitter_source
47B8: FD CA 04    STD   blitter_dest
47BB: FF CA 00    STU   start_blitter
```

```
47BE: D3 BA       ADDD  $BA
47C0: AE A1       LDX   ,Y++
47C2: BF CA 02    STX   blitter_source
47C5: FD CA 04    STD   blitter_dest
47C8: FF CA 00    STU   start_blitter
47CB: D3 BA       ADDD  $BA
47CD: AE A1       LDX   ,Y++
47CF: BF CA 02    STX   blitter_source
47D2: FD CA 04    STD   blitter_dest
47D5: FF CA 00    STU   start_blitter
47D8: D3 BA       ADDD  $BA
47DA: AE A1       LDX   ,Y++
47DC: BF CA 02    STX   blitter_source
47DF: FD CA 04    STD   blitter_dest
47E2: FF CA 00    STU   start_blitter
47E5: D3 BA       ADDD  $BA
47E7: AE A1       LDX   ,Y++
47E9: BF CA 02    STX   blitter_source
47EC: FD CA 04    STD   blitter_dest
47EF: FF CA 00    STU   start_blitter
47F2: D3 BA       ADDD  $BA
47F4: AE A1       LDX   ,Y++
47F6: BF CA 02    STX   blitter_source
47F9: FD CA 04    STD   blitter_dest
47FC: FF CA 00    STU   start_blitter
47FF: D3 BA       ADDD  $BA
4801: AE A1       LDX   ,Y++
4803: BF CA 02    STX   blitter_source
4806: FD CA 04    STD   blitter_dest
4809: FF CA 00    STU   start_blitter
480C: D3 BA       ADDD  $BA
480E: AE A1       LDX   ,Y++
4810: BF CA 02    STX   blitter_source
4813: FD CA 04    STD   blitter_dest
4816: FF CA 00    STU   start_blitter
4819: D3 BA       ADDD  $BA
481B: AE A1       LDX   ,Y++
481D: BF CA 02    STX   blitter_source
4820: FD CA 04    STD   blitter_dest
4823: FF CA 00    STU   start_blitter
4826: D3 BA       ADDD  $BA
4828: AE A1       LDX   ,Y++
482A: BF CA 02    STX   blitter_source
482D: FD CA 04    STD   blitter_dest
4830: FF CA 00    STU   start_blitter
4833: D3 BA       ADDD  $BA
4835: AE A1       LDX   ,Y++
4837: BF CA 02    STX   blitter_source
483A: FD CA 04    STD   blitter_dest
483D: FF CA 00    STU   start_blitter
4840: D3 BA       ADDD  $BA
4842: AE A1       LDX   ,Y++
4844: BF CA 02    STX   blitter_source
4847: FD CA 04    STD   blitter_dest
```

```
484A: FF CA 00    STU   start_blitter
484D: D3 BA       ADDD  $BA
484F: AE A1       LDX   ,Y++
4851: BF CA 02    STX   blitter_source
4854: FD CA 04    STD   blitter_dest
4857: FF CA 00    STU   start_blitter
485A: 1C EF       ANDCC #$EF                ; clear carry flag
485C: 35 A0       PULS  Y,PC ;(PUL? PC=RTS)



485E: FD CA 04    STD   blitter_dest
4861: FF C9 FF    STU   $C9FF
4864: D3 BA       ADDD  $BA
4866: FD CA 04    STD   blitter_dest
4869: FF C9 FF    STU   $C9FF
486C: D3 BA       ADDD  $BA
486E: FD CA 04    STD   blitter_dest
4871: FF C9 FF    STU   $C9FF
4874: D3 BA       ADDD  $BA
4876: FD CA 04    STD   blitter_dest
4879: FF C9 FF    STU   $C9FF
487C: D3 BA       ADDD  $BA
487E: FD CA 04    STD   blitter_dest
4881: FF C9 FF    STU   $C9FF
4884: D3 BA       ADDD  $BA
4886: FD CA 04    STD   blitter_dest
4889: FF C9 FF    STU   $C9FF
488C: D3 BA       ADDD  $BA
488E: FD CA 04    STD   blitter_dest
4891: FF C9 FF    STU   $C9FF
4894: D3 BA       ADDD  $BA
4896: FD CA 04    STD   blitter_dest
4899: FF C9 FF    STU   $C9FF
489C: D3 BA       ADDD  $BA
489E: FD CA 04    STD   blitter_dest
48A1: FF C9 FF    STU   $C9FF
48A4: D3 BA       ADDD  $BA
48A6: FD CA 04    STD   blitter_dest
48A9: FF C9 FF    STU   $C9FF
48AC: D3 BA       ADDD  $BA
48AE: FD CA 04    STD   blitter_dest
48B1: FF C9 FF    STU   $C9FF
48B4: D3 BA       ADDD  $BA
48B6: FD CA 04    STD   blitter_dest
48B9: FF C9 FF    STU   $C9FF
48BC: D3 BA       ADDD  $BA
48BE: FD CA 04    STD   blitter_dest
48C1: FF C9 FF    STU   $C9FF
48C4: D3 BA       ADDD  $BA
48C6: FD CA 04    STD   blitter_dest
48C9: FF C9 FF    STU   $C9FF
48CC: D3 BA       ADDD  $BA
48CE: FD CA 04    STD   blitter_dest
```

```
48D1: FF C9 FF    STU   $C9FF
48D4: D3 BA       ADDD  $BA
48D6: FD CA 04    STD   blitter_dest
48D9: FF C9 FF    STU   $C9FF
48DC: D3 BA       ADDD  $BA
48DE: 1C EF       ANDCC #$EF                ; clear carry flag
48E0: 39          RTS

48E1: E6 A8 11    LDB   $11,Y
48E4: C0 10       SUBB  #$10
48E6: 50          NEGB
48E7: 58          ASLB
48E8: 58          ASLB
48E9: 58          ASLB
48EA: 8E 48 5E    LDX   #$485E
48ED: 3A          ABX
48EE: EC 27       LDD   $0007,Y
48F0: DD BA       STD   $BA
48F2: 96 45       LDA   $45
48F4: C6 12       LDB   #$12
48F6: 1F 03       TFR   D,U
48F8: CC 00 00    LDD   #$0000
48FB: 1A 10       ORCC  #$10
48FD: FD CA 01    STD   blitter_mask
4900: EC 2F       LDD   $000F,Y
4902: FD CA 06    STD   blitter_w_h
4905: EC 2B       LDD   $000B,Y
4907: 6E 84       JMP   ,X

4909: CE 98 BC    LDU   #$98BC
490C: 10 AC C4    CMPY  ,U
490F: 27 08       BEQ   $4919
4911: EE C4       LDU   ,U
4913: 26 F7       BNE   $490C
4915: 1A 10       ORCC  #$10
4917: 20 FE       BRA   $4917

4919: EC A4       LDD   ,Y
491B: ED C4       STD   ,U
491D: DC C0       LDD   $C0
491F: ED A4       STD   ,Y
4921: 10 9F C0    STY   $C0
4924: 31 C4       LEAY  ,U
4926: 39          RTS

4927: EC 28       LDD   $0008,Y
4929: 83 01 00    SUBD  #$0100
492C: A1 28       CMPA  $0008,Y
492E: 26 03       BNE   $4933
4930: E7 29       STB   $0009,Y
4932: 39          RTS

4933: BD 48 E1    JSR   $48E1
4936: 96 59       LDA   $59
```

```
4938: 26 08       BNE   $4942
493A: DC 5E       LDD   $5E
493C: A7 24       STA   $0004,Y
493E: EB 26       ADDB  $0006,Y
4940: E7 25       STB   $0005,Y
4942: A6 2E       LDA   $000E,Y
4944: 97 C2       STA   $C2
4946: EC 28       LDD   $0008,Y
4948: 83 01 00    SUBD  #$0100
494B: 81 01       CMPA  #$01
494D: 22 12       BHI   $4961
494F: CE 98 BE    LDU   #$98BE
4952: 20 B8       BRA   $490C

4954: 6A 2A       DEC   $000A,Y
4956: 27 B1       BEQ   $4909
4958: A6 2E       LDA   $000E,Y
495A: 97 C2       STA   $C2
495C: EC 28       LDD   $0008,Y
495E: C3 01 00    ADDD  #$0100
4961: 97 BB       STA   $BB
4963: ED 28       STD   $0008,Y
4965: 44          LSRA
4966: 97 C5       STA   $C5
4968: E6 A8 12    LDB   $12,Y
496B: 2A 01       BPL   $496E
496D: 40          NEGA
496E: A7 27       STA   $0007,Y
4970: 97 BA       STA   $BA
4972: E6 26       LDB   $0006,Y
4974: 26 02       BNE   $4978
4976: D7 C5       STB   $C5
4978: 96 BB       LDA   $BB
497A: E6 26       LDB   $0006,Y
497C: D7 B9       STB   $B9
497E: 3D          MUL
497F: DD C3       STD   $C3
4981: E6 25       LDB   $0005,Y
4983: 4F          CLRA
4984: 93 C3       SUBD  $C3
4986: DB C5       ADDB  $C5
4988: 89 00       ADCA  #$00
498A: 26 04       BNE   $4990
498C: C1 18       CMPB  #$18
498E: 22 0E       BHI   $499E



4990: 0A C2       DEC   $C2
4992: 0A B9       DEC   $B9
4994: DB BB       ADDB  $BB
4996: 89 00       ADCA  #$00
4998: 26 F6       BNE   $4990
499A: C1 18       CMPB  #$18
```

```
499C: 23 F2       BLS   $4990
499E: E7 2C       STB   $000C,Y
49A0: 96 B9       LDA   $B9
49A2: D6 BA       LDB   $BA
49A4: 2B 2C       BMI   $49D2
49A6: 3D          MUL
49A7: DD C3       STD   $C3
49A9: E6 24       LDB   $0004,Y
49AB: 4F          CLRA
49AC: 93 C3       SUBD  $C3
49AE: 4D          TSTA
49AF: 26 04       BNE   $49B5
49B1: C1 07       CMPB  #$07
49B3: 22 41       BHI   $49F6
49B5: 0F B8       CLR   $B8
49B7: 0A C2       DEC   $C2
49B9: 0C B8       INC   $B8
49BB: DB BA       ADDB  $BA
49BD: 89 00       ADCA  #$00
49BF: 26 F6       BNE   $49B7
49C1: C1 07       CMPB  #$07
49C3: 23 F2       BLS   $49B7
49C5: E7 2B       STB   $000B,Y
49C7: D6 B8       LDB   $B8
49C9: 96 BB       LDA   $BB
49CB: 3D          MUL
49CC: EB 2C       ADDB  $000C,Y
49CE: E7 2C       STB   $000C,Y
49D0: 20 26       BRA   $49F8

49D2: 50          NEGB
49D3: 3D          MUL
49D4: EB 2D       ADDB  $000D,Y
49D6: EB 24       ADDB  $0004,Y
49D8: 89 00       ADCA  #$00
49DA: 26 04       BNE   $49E0
49DC: C1 8F       CMPB  #$8F
49DE: 23 14       BLS   $49F4
49E0: 0F B8       CLR   $B8
49E2: 0A C2       DEC   $C2
49E4: 0C B8       INC   $B8
49E6: DB BA       ADDB  $BA
49E8: 89 FF       ADCA  #$FF
49EA: 26 F6       BNE   $49E2
49EC: C1 8F       CMPB  #$8F
49EE: 22 F2       BHI   $49E2
49F0: E0 2D       SUBB  $000D,Y
49F2: 20 D1       BRA   $49C5

49F4: E0 2D       SUBB  $000D,Y
49F6: E7 2B       STB   $000B,Y
49F8: 30 A8 13    LEAX  $13,Y
49FB: E6 2E       LDB   $000E,Y
49FD: D0 C2       SUBB  $C2
```

```
49FF: 58          ASLB
4A00: 3A          ABX
4A01: 96 C2       LDA   $C2
4A03: 4A          DECA
4A04: D6 BB       LDB   $BB
4A06: 3D          MUL
4A07: EB 2C       ADDB  $000C,Y
4A09: 89 00       ADCA  #$00
4A0B: 27 0A       BEQ   $4A17
4A0D: 0A C2       DEC   $C2
4A0F: 27 48       BEQ   $4A59
4A11: D0 BB       SUBB  $BB
4A13: 82 00       SBCA  #$00
4A15: 26 F6       BNE   $4A0D
4A17: C1 EA       CMPB  #$EA
4A19: 24 F2       BCC   $4A0D
4A1B: 96 C2       LDA   $C2
4A1D: 4A          DECA
4A1E: D6 BA       LDB   $BA
4A20: 2B 19       BMI   $4A3B
4A22: 3D          MUL
4A23: EB 2D       ADDB  $000D,Y
4A25: EB 2B       ADDB  $000B,Y
4A27: 89 00       ADCA  #$00
4A29: 27 0A       BEQ   $4A35
4A2B: 0A C2       DEC   $C2
4A2D: 27 2A       BEQ   $4A59
4A2F: D0 BA       SUBB  $BA
4A31: 82 00       SBCA  #$00
4A33: 26 F6       BNE   $4A2B
4A35: C1 8F       CMPB  #$8F
4A37: 22 F2       BHI   $4A2B
4A39: 20 1A       BRA   $4A55

4A3B: 50          NEGB
4A3C: 3D          MUL
4A3D: DD C3       STD   $C3
4A3F: E6 2B       LDB   $000B,Y
4A41: 4F          CLRA
4A42: 93 C3       SUBD  $C3
4A44: 4D          TSTA
4A45: 27 0A       BEQ   $4A51
4A47: 0A C2       DEC   $C2
4A49: 27 0E       BEQ   $4A59
4A4B: D0 BA       SUBB  $BA
4A4D: 82 FF       SBCA  #$FF
4A4F: 26 F6       BNE   $4A47
4A51: C1 07       CMPB  #$07
4A53: 23 F2       BLS   $4A47
4A55: 96 C2       LDA   $C2
4A57: 26 03       BNE   $4A5C
4A59: 7E 49 09    JMP   $4909
```

```
;
; This is the routine that famously crashes Robotron with the "shot
in the corner" bug.
; You'll encounter what you think are magic  numbers below ($10 (16
decimal) and $0D (13 decimal)).
; Here's why those numbers are here:
;
; Look at the code starting at 478C. This set of instructions is
repeated *16* (decimal) times:
; 478C: AE A1       LDX   ,Y++
; 478E: BF CA 02    STX   blitter_source
; 4791: FD CA 04    STD   blitter_dest
; 4794: FF CA 00    STU   start_blitter
; 4797: D3 BA       ADDD  $BA
;
; How many bytes does this set of instructions take up?
; 13 (decimal).
;
; So, you can now see what the reason is for the presence of the 16
and 13 in the code below.
;
; The function below computes where in the set of 16 blits it should
jump to. If it needs to draw
; 16 segments of an explosion it will jump to $478C (the very first
blit), if it computes a need to draw 15 segments
; it will jump to $4799, 14  and so on.
;
;

4A5C: A7 A8 11    STA   $11,Y
4A5F: 80 10       SUBA  #$10        ; A -= 16
4A61: 40          NEGA              ; A = (0 - A), giving negative A
4A62: C6 0D       LDB   #$0D
4A64: 3D          MUL               ; D = A * #$0D (13 dec)
4A65: C3 47 8C    ADDD  #$478C      ; D += 478C to give address to
jump to in the explosion segment draw
4A68: 34 26       PSHS  Y,B,A       ; push Y and return address (D)
to stack
4A6A: CE 0A 0A    LDU   #$0A0A
4A6D: EC 2F       LDD   $000F,Y
4A6F: 1A 10       ORCC  #$10
4A71: FD CA 06    STD   blitter_w_h
4A74: EC 2B       LDD   $000B,Y     ; set blitter destination
4A76: 31 84       LEAY  ,X
4A78: 39          RTS               ; pull return address off stack
(this is what causes the crash at times)

4A79: 10 9E BC    LDY   $BC
4A7C: 27 0B       BEQ   $4A89
4A7E: BD 48 E1    JSR   $48E1
4A81: BD 49 54    JSR   $4954
4A84: 10 AE A4    LDY   ,Y
4A87: 26 F5       BNE   $4A7E
```

```
4A89: 10 9E BE    LDY   $BE
4A8C: 27 08       BEQ   $4A96
4A8E: BD 49 27    JSR   $4927
4A91: 10 AE A4    LDY   ,Y
4A94: 26 F8       BNE   $4A8E
4A96: 39          RTS

4A97: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
4AA7: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
4AB7: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
4AC7: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
4AD7: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
4AE7: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
4AF7: FF FF FF FF FF FF FF FF FF

4B00: 7E 4D 10    JMP   $4D10

4B03: 7E 4B 36    JMP   $4B36

4B06: 50 C2 50 0E 50 26 D0 01 08 11 00 C8 01 08 04 00
4B16: C8 01 04 14 01 01 13 00 D0 01 03 01 01 04 15 01
4B26: 04 13 00 D0 01 04 15 01 08 11 00 D0 01 08 19 00


INITIALISE_ALL_QUARKS:
4B36: B6 BE 70    LDA   cur_quarks          ; get count of quarks
into A
4B39: 34 02       PSHS  A                   ; save count on stack
4B3B: 27 43       BEQ   $4B80               ; if no quarks, exit
4B3D: BD D0 6C    JSR   $D06C               ; JMP $D32B - create
entity with params and add to linked list at $9817
; parameters to pass to $D06C
4B40: 4B FB       ; address of function to call after 1 game cycle
4B42: 50 C6       ; animation frame metadata pointer
4B44: 4B C9       ; address of routine to handle collision
4B46: 27 38       BEQ   $4B80
4B48: C6 1A       LDB   #$1A
4B4A: BD D0 39    JSR   $D039               ; JMP $D6CD - get a
random number into A
4B4D: 2A 02       BPL   $4B51
4B4F: C6 DC       LDB   #$DC
4B51: 86 7E       LDA   #$7E
4B53: BD D0 42    JSR   $D042               ; JMP $D6AC - multiply A
by a random number and put result in A
4B56: 8B 06       ADDA  #$06
4B58: ED 04       STD   $0004,X             ; set blitter
destination
4B5A: A7 0A       STA   $000A,X             ; set quark X coordinate
(whole part)
4B5C: E7 0C       STB   $000C,X             ; set quark Y coordinate
4B5E: B6 BE 66    LDA   $BE66
4B61: BD D0 3F    JSR   $D03F               ; JMP $D6B6 - get a
random number lower than or equal to A
4B64: A7 49       STA   $0009,U             ; set countdown before
```

```
spawning first tank
4B66: B6 BE 5E    LDA   $BE5E               ; read tank spawn
control variable
4B69: BD D0 3F    JSR   $D03F               ; JMP $D6B6 - get a
random number lower than or equal to A
4B6C: 44          LSRA
4B6D: 89 00       ADCA  #$00
4B6F: A7 4A       STA   $000A,U             ; set number of tanks
this quark can spawn
4B71: CC 50 D2    LDD   #$50D2              ; pointer to collision
detection animation frame metadata for quark (see $D7F4)
4B74: ED 88 16    STD   $16,X
4B77: 9F 17       STX   $17
4B79: BD 4B 82    JSR   $4B82
4B7C: 6A E4       DEC   ,S                  ; decrement count of
quarks on stack
4B7E: 26 BD       BNE   $4B3D
4B80: 35 82       PULS  A,PC ;(PUL? PC=RTS)



CHANGE_QUARK_DIRECTION:
4B82: B6 BE 67    LDA   $BE67
4B85: BD D0 3F    JSR   $D03F               ; JMP $D6B6 - get a
random number lower than or equal to A
4B88: E6 0A       LDB   $000A,X
4B8A: C1 0C       CMPB  #$0C
4B8C: 23 09       BLS   $4B97
4B8E: C1 83       CMPB  #$83
4B90: 24 04       BCC   $4B96
4B92: D6 86       LDB   $86
4B94: 2A 01       BPL   $4B97
4B96: 40          NEGA
4B97: 1F 89       TFR   A,B
4B99: 1D          SEX
4B9A: 58          ASLB
4B9B: 49          ROLA
4B9C: 58          ASLB
4B9D: 49          ROLA
4B9E: ED 0E       STD   $000E,X             ; set X delta
4BA0: B6 BE 67    LDA   $BE67
4BA3: BD D0 3F    JSR   $D03F               ; JMP $D6B6 - get a
random number lower than or equal to A
4BA6: E6 0C       LDB   $000C,X
4BA8: C1 1D       CMPB  #$1D
4BAA: 23 09       BLS   $4BB5
4BAC: C1 D6       CMPB  #$D6
4BAE: 24 04       BCC   $4BB4
4BB0: D6 85       LDB   $85
4BB2: 2B 01       BMI   $4BB5
4BB4: 40          NEGA
4BB5: 1F 89       TFR   A,B
4BB7: 1D          SEX
4BB8: 58          ASLB
```

```
4BB9: 49          ROLA
4BBA: 58          ASLB
4BBB: 49          ROLA
4BBC: 58          ASLB
4BBD: 49          ROLA
4BBE: ED 88 10    STD   $10,X
4BC1: 96 84       LDA   $84
4BC3: 84 1F       ANDA  #$1F
4BC5: 4C          INCA
4BC6: A7 4E       STA   $000E,U
4BC8: 39          RTS


QUARK_COLLISION_HANDLER:
4BC9: 96 48       LDA   $48                ; has player been hit?
4BCB: 26 2D       BNE   $4BFA              ; if yes, goto $4BFA
4BCD: BD D0 78    JSR   $D078              ; JMP $D320 - deallocate
object and erase object from screen (2)
4BD0: DE 1B       LDU   $1B
4BD2: EC C4       LDD   ,U
4BD4: DD 1B       STD   $1B
4BD6: BD D0 54    JSR   $D054              ; JMP $D281 - reserve
object metadata entry and call function
4BD9: 11 43       ; pointer to function
4BDB: EF 07       STU   $0007,X
4BDD: CC 50 C6    LDD   #$50C6             ; quark animation
metadata
4BE0: ED 42       STD   $0002,U
4BE2: CC DD DD    LDD   #$DDDD
4BE5: ED 0B       STD   $000B,X
4BE7: 86 08       LDA   #$08
4BE9: A7 0D       STA   $000D,X
4BEB: CC 4B 29    LDD   #$4B29
4BEE: BD D0 4B    JSR   $D04B               ; JMP $D3C7
4BF1: CC 02 10    LDD   #$0210
4BF4: BD D0 0C    JSR   $D00C               ; JMP $DB9C - update
player score
4BF7: 7A BE 70    DEC   cur_quarks
4BFA: 39          RTS


ANIMATE_QUARK:
4BFB: AE 47       LDX   $0007,U             ; get pointer to quark
object
4BFD: EC 02       LDD   $0002,X             ; get pointer to
animation frame metadata into D
4BFF: C3 00 04    ADDD  #$0004              ; bump D to pointer to
next animation frame metadata
4C02: 10 83 50 D2 CMPD  #$50D2              ; hit end of animation
frame metadata for quark?
4C06: 23 0B       BLS   $4C13               ; no
4C08: CC 50 C2    LDD   #$50C2              ; back to first image
of quark
4C0B: 0D 59       TST   $59
```

```
4C0D: 26 04       BNE   $4C13
4C0F: 6A 49       DEC   $0009,U             ; decrement tank spawn
countdown
4C11: 27 11       BEQ   $4C24               ; if 0, goto $4C24, it
4C13: ED 02       STD   $0002,X             ; set pointer to
animation frame metadata
4C15: 6A 4E       DEC   $000E,U             ; decrement countdown
before quark picks new direction to move in
4C17: 26 03       BNE   $4C1C               ; if countdown not hit
0, no direction change, goto $4C1C
4C19: BD 4B 82    JSR   $4B82               ; otherwise, pick a new
direction for quark to move in
4C1C: 86 03       LDA   #$03                ; delay before calling
function
4C1E: 8E 4B FB    LDX   #$4BFB              ; pointer to animate
quark routine to call
4C21: 7E D0 66    JMP   $D066

SET_TANK_SPAWN_COUNTDOWN:
4C24: B6 BE 66    LDA   $BE66               ;
4C27: 44          LSRA
4C28: 4C          INCA
4C29: BD D0 3F    JSR   $D03F               ; JMP $D6B6 - get a
random number lower than or equal to A
4C2C: A7 49       STA   $0009,U

4C2E: AE 47       LDX   $0007,U             ; get object pointer
into X
4C30: 6A 49       DEC   $0009,U
4C32: 26 1C       BNE   $4C50
4C34: 96 42       LDA   $42
4C36: 81 11       CMPA  #$11
4C38: 24 EA       BCC   $4C24
4C3A: 96 13       LDA   $13                 ; this piece of code
determines if we have any free objects we can use ..
4C3C: 9A 1B       ORA   $1B                 ; ... to hold a new
tank
4C3E: 27 E4       BEQ   $4C24               ; if not, goto $4C24,
we'll wait for a free object to become available (e.g.: something
dies)
4C40: B6 BE 71    LDA   cur_tanks
4C43: 81 14       CMPA  #$14                ; compare number of
tanks to #$14  (20 dec.)
4C45: 24 DD       BCC   $4C24               ; if we've got >= 20
tanks, go to $4C24
4C47: BD 4C AC    JSR   $4CAC               ; spawn a tank!
4C4A: 6A 4A       DEC   $000A,U             ; decrement counter of
tanks left to spawn
4C4C: 27 21       BEQ   $4C6F
4C4E: 20 D4       BRA   $4C24

4C50: EC 02       LDD   $0002,X            ; get pointer to
animation frame metadata
4C52: C3 00 04    ADDD  #$0004             ; bump to next image
```

```
4C55: 10 83 50 E2 CMPD  #$50E2              ; hit end of animation
frame metadata list for tank?
4C59: 23 03       BLS   $4C5E              ; if not, go to $4C5E
4C5B: CC 50 C2    LDD   #$50C2              ; reset animation frame
metadata pointer to start of animation frame metadata list
4C5E: ED 02       STD   $0002,X             ; set animation frame
metadata pointer
4C60: 6A 4E       DEC   $000E,U
4C62: 26 03       BNE   $4C67
4C64: BD 4B 82    JSR   $4B82
4C67: 86 03       LDA   #$03                ; delay before calling
function
4C69: 8E 4C 2E    LDX   #$4C2E              ; address of function to
call next
4C6C: 7E D0 66    JMP   $D066

4C6F: CC 00 00    LDD   #$0000
4C72: ED 0E       STD   $000E,X             ; set X delta
4C74: CC 02 00    LDD   #$0200
4C77: 0D 84       TST   $84
4C79: 2A 01       BPL   $4C7C
4C7B: 40          NEGA
4C7C: ED 88 10    STD   $10,X                           ; set Y
delta

;
; the Quark's in a mood to disappear here....
;

4C7F: AE 47       LDX   $0007,U             ; get object pointer
4C81: EC 02       LDD   $0002,X             ; get animation frame
metadata pointer into D
4C83: 83 00 04    SUBD  #$0004              ; D -= 4 (point D to
previous animation frame metadata)
4C86: 10 83 50 C2 CMPD  #$50C2              ; beyond start of
animation frame metadata list?
4C8A: 24 0D       BCC   $4C99               ; no
4C8C: A6 0C       LDA   $000C,X             ; read object Y
coordinate
4C8E: 81 1A       CMPA  #$1A                ; <= #$1A (26 decimal) ?
4C90: 23 11       BLS   $4CA3               ; yes, so remove this
object from playfield
4C92: 81 DA       CMPA  #$DA                ; >#$DA (218 decimal) ?
4C94: 24 0D       BCC   $4CA3               ; yes, so remove this
object from playfield
4C96: CC 50 E2    LDD   #$50E2              ; load D with address of
last quark animation frame metadata
4C99: ED 02       STD   $0002,X             ; set animation frame
metadata pointer
4C9B: 86 03       LDA   #$03                ; delay before calling
function
4C9D: 8E 4C 7F    LDX   #$4C7F              ; function to call next
4CA0: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call
```

```
4CA3: BD D0 75    JSR   $D075               ; JMP $D31B - deallocate
object and erase object from screen
4CA6: 7A BE 70    DEC   cur_quarks
4CA9: 7E D0 63    JMP   $D063               ; JMP $D1F3



;
; Spawn a tank.
;
; X = pointer to quark giving birth
;

SPAWN_TANK:
4CAC: 34 76       PSHS  U,Y,X,B,A
4CAE: 31 84       LEAY  ,X                  ; Y = X
4CB0: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
4CB3: 4D 83       ; values to pass to the function
4CB5: 33 84       LEAU  ,X                  ; U = X
4CB7: BD D0 7B    JSR   $D07B               ; JMP $D2DA - reserve
entry in list used by grunts, hulks, brains, progs, cruise missiles
and tanks (starts at $9821)
; X = pointer to object
4CBA: 7C BE 71    INC   cur_tanks
4CBD: CC 50 0E    LDD   #$500E              ; $500E - pointer to
animation frame metadata
4CC0: ED 02       STD   $0002,X             ; set current animation
frame metadata pointer
4CC2: ED 88 14    STD   $14,X               ; set previous animation
frame metadata pointer (previous = current)
4CC5: EF 06       STU   $0006,X             ; set pointer to object
metadata
4CC7: AF 47       STX   $0007,U             ; set back-pointer to
object, from object metadata
4CC9: CC 4D F2    LDD   #$4DF2              ; store pointer to
routine that handles tank collision
4CCC: ED 08       STD   $0008,X
4CCE: CC 4B 31    LDD   #$4B31
4CD1: BD D0 4B    JSR   $D04B               ; JMP $D3C7
4CD4: EC 24       LDD   $0004,Y             ; D = blitter
destination of spheroid
4CD6: C1 18       CMPB  #$18                ; compare vertical
position of spheroid to #$18 (24 decimal)
4CD8: 27 01       BEQ   $4CDB
4CDA: 5A          DECB
4CDB: C3 02 06    ADDD  #$0206              ; birth place of tank =
4 pixels from left of spheroid, 6 pixels from top
4CDE: ED 04       STD   $0004,X             ; set blitter
destination
4CE0: A7 0A       STA   $000A,X             ; set X coordinate
(whole part)
4CE2: E7 0C       STB   $000C,X             ; set Y coordinate
```

```
4CE4: BD 4E 11    JSR   $4E11               ; pick a destination for
tank to move to
4CE7: B6 BE 64    LDA   $BE64               ; read "tank fire delay"
4CEA: A7 4D       STA   $000D,U
4CEC: BD D0 18    JSR   $D018               ; JMP   $DAF2 - do blit
4CEF: 35 F6       PULS  A,B,X,Y,U,PC ;(PUL? PC=RTS)

4CF1:  (C) WILLIAMS ELECTRONICS INC.


;
;
;
;

INITIALISE_ALL_TANKS:
4D10: B6 BE 71    LDA   cur_tanks           ; get count of tanks for
this wave into A
4D13: 34 02       PSHS  A
4D15: 27 4A       BEQ   $4D61               ; if count ==0, no tanks
on this wave, goto $4D61
4D17: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
4D1A: 4D 8B       ; data to pass to function (looks like an address
to call)
4D1C: 8B 33       LEAU, X
4D1E: 84 BD       JSR   $D07B               ; JMP $D2DA - reserve
entry in list used by grunts, hulks, brains, progs, cruise missiles
and tanks (starts at $9821)
4D21: CC 50 26    LDD   #$5026
4D24: ED 02       STD   $0002,X             ; set current animation
frame metadata pointer
4D26: ED 88 14    STD   $14,X               ; set previous animation
frame metadata pointer (previous = current)
4D29: EF 06       STU   $0006,X             ; set pointer to object
metadata to U
4D2B: AF 47       STX   $0007,U             ; set pointer to this
object in object metadata
4D2D: CC 4D F2    LDD   #$4DF2              ; address of tank
collision handler routine
4D30: ED 08       STD   $0008,X
4D32: BD 38 8E    JSR   $388E               ; JMP $38FE -compute
safe rectangle for player
4D35: BD 26 C3    JSR   $26C3               ; JMP $3199 - get random
position on playfield for object (returns: A = X coordinate, B = Y
coordinate)
; determine if the computed position is valid (meaning, not in the
player's safe area)
4D38: D1 2B       CMPB  $2B
4D3A: 23 0C       BLS   $4D48
4D3C: D1 2C       CMPB  $2C
4D3E: 24 08       BCC   $4D48
4D40: 91 2D       CMPA  $2D
4D42: 23 04       BLS   $4D48
```

```
4D44: 91 2E       CMPA  $2E
4D46: 23 ED       BLS   $4D35             ; position is invalid,
get another one
4D48: ED 04       STD   $0004,X           ; set blitter
destination of object
4D4A: A7 0A       STA   $000A,X           ; set X coordinate
(whole part) of tank
4D4C: E7 0C       STB   $000C,X           ; set Y coordinate of
tank
4D4E: BD 4E 11    JSR   $4E11             ; pick initial direction
for tank to move
4D51: 96 84       LDA   $84               ; get a random number
into A
4D53: 84 1F       ANDA  #$1F              ; mask off lower 5 bits
giving a number from 0..31
4D55: BB BE 64    ADDA  $BE64             ; add on "tank fire
delay variable" to set countdown before firing.
4D58: A7 4D       STA   $000D,U           ; set countdown before
firing first shell
4D5A: BD 38 8B    JSR   $388B             ; JMP $393C - blit tank
in solid colour invisible to player
4D5D: 6A E4       DEC   ,S                ; decrement count of
tanks left to do on the stack
4D5F: 26 B6       BNE   $4D17             ; if !=0, goto $4D17
4D61: 35 82       PULS  A,PC ;(PUL? PC=RTS)

4D63: AE 47       LDX   $0007,U
4D65: BD D0 15    JSR   $D015             ; JMP $DB03 - erase
object from screen
4D68: EC 04       LDD   $0004,X           ; get blitter
destination into D
4D6A: 10 AE 02    LDY   $0002,X
4D6D: AB 24       ADDA  $0004,Y
4D6F: EB 25       ADDB  $0005,Y
4D71: A7 0A       STA   $000A,X
4D73: E7 0C       STB   $000C,X
4D75: 31 26       LEAY  $0006,Y
4D77: 10 AF 02    STY   $0002,X
4D7A: BD D0 18    JSR   $D018             ; JMP   $DAF2 - do blit
4D7D: 10 8C 50 26 CMPY  #$5026
4D81: 24 08       BCC   $4D8B
4D83: 86 0C       LDA   #$0C              ; delay before calling
function
4D85: 8E 4D 63    LDX   #$4D63            ; address of function to
call
4D88: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call

4D8B: 96 59       LDA   $59
4D8D: 85 7F       BITA  #$7F
4D8F: 27 08       BEQ   $4D99
4D91: 86 0F       LDA   #$0F              ; delay before calling
function
4D93: 8E 4D 8B    LDX   #$4D8B            ; address of function to
```

```
call
4D96: 7E D0 66    JMP   $D066              ; JMP $D1E3 - allocate
function call

4D99: AE 47       LDX   $0007,U            ; get object pointer in
X
4D9B: 6A 4D       DEC   $000D,U            ; decrement tank fire
countdown
4D9D: 26 03       BNE   $4DA2              ; if countdown !=0 goto
$4DA2
4D9F: BD 4E 46    JSR   $4E46              ; if countdown is 0 then
fire a tank shell
4DA2: A6 4B       LDA   $000B,U            ; get horizontal delta
of tank
4DA4: 5F          CLRB
4DA5: 47          ASRA
4DA6: 56          RORB
4DA7: E3 0A       ADDD  $000A,X            ; add delta to X
coordinate of tank
4DA9: 34 06       PSHS  B,A                ; save result of
addition in D on stack
4DAB: E6 4C       LDB   $000C,U            ; get vertical delta of
tank
4DAD: EB 0C       ADDB  $000C,X            ; add delta to vertical
position of tank
4DAF: BD 00 06    JSR   $0006              ; test that object is in
bounds
4DB2: 27 04       BEQ   $4DB8              ; yes tank is in bounds,
goto $4DB8
4DB4: 32 62       LEAS  $0002,S            ; discard items pushed
on stack @ 4DA9
4DB6: 20 2F       BRA   $4DE7
4DB8: E7 0C       STB   $000C,X            ; update vertical
position of tank
4DBA: 35 06       PULS  A,B                ; restore new X
coordinate of tank from stack
4DBC: ED 0A       STD   $000A,X            ; update X coordinate of
tank
;
; This piece of code animates the tank. If the tank is moving to the
LEFT, the tank animation runs backwards,
; but if the tank is moving in any other direction the animation
runs forwards.
; I never noticed this in the game before....
;
4DBE: EC 02       LDD   $0002,X            ; read current animation
frame metadata pointer
4DC0: 6D 4B       TST   $000B,U            ; is the tank moving to
the right (or just not moving left) ?
4DC2: 2A 0E       BPL   $4DD2              ; yes
4DC4: 83 00 04    SUBD  #$0004             ; bump animation frame
metadata pointer to previous animation (making tank animation go
*backwards*)
4DC7: 10 83 50 26 CMPD  #$5026             ; have we gone past the
```

```
start animation frame?
4DCB: 24 11       BCC   $4DDE               ; no, goto $4DDE
4DCD: CC 50 32    LDD   #$5032              ; yes, so set pointer to
last tank animation
4DD0: 20 0C       BRA   $4DDE

4DD2: C3 00 04    ADDD  #$0004              ; bump to next animation
frame metadata in list for tank
4DD5: 10 83 50 32 CMPD  #$5032              ; gone past last
animation frame metadata in list?
4DD9: 23 03       BLS   $4DDE               ; no
4DDB: CC 50 26    LDD   #$5026              ; yes, reset to first
animation frame metadata for tank
4DDE: ED 02       STD   $0002,X             ; set animation frame
metadata pointer
4DE0: BD D0 8D    JSR   $D08D               ; JMP $DB2F - draw
object
4DE3: 6A 4E       DEC   $000E,U             ; decrement "tank move"
countdown
4DE5: 26 03       BNE   $4DEA               ; if !=0, not time to
change direction, goto $4DEA
4DE7: BD 4E 11    JSR   $4E11               ; otherwise, countdown
hit 0, find a new direction for the tank to move in
4DEA: 96 EF       LDA   $EF                 ; delay before calling
function
4DEC: 8E 4D 99    LDX   #$4D99              ; address of function to
call
4DEF: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call


;
TANK_COLLISION_HANDLER:
4DF2: 96 48       LDA   $48                 ; player collision
detection?
4DF4: 26 1A       BNE   $4E10               ; yes
4DF6: 7A BE 71    DEC   cur_tanks
4DF9: BD 5B 4F    JSR   $5B4F               ; JMP $5C0A
4DFC: BD D0 7E    JSR   $D07E               ; JMP $D2C2 - remove
baddy from baddies list
4DFF: AE 06       LDX   $0006,X             ; get pointer to object
metadata into X
4E01: BD D0 5D    JSR   $D05D               ; JMP $D218 - deallocate
object metadata entry
4E04: CC 01 20    LDD   #$0120
4E07: BD D0 0C    JSR   $D00C               ; JMP $DB9C - update
player score
4E0A: CC 4B 0C    LDD   #$4B0C
4E0D: BD D0 4B    JSR   $D04B               ; JMP $D3C7
4E10: 39          RTS



;
```

```
; Tank needs to move somewhere.
; Depending on a random number, the tank either moves towards a
random position on the screen
; OR gravitates towards the player.
;

PICK_DESTINATION_TO_MOVE_TANK_TO:
4E11: 96 84       LDA   $84                ; get a random number
into A
4E13: 81 60       CMPA  #$60               ; compare to #$60 (96
decimal)
4E15: 23 05       BLS   $4E1C              ; if lower or the same
goto $4E1C - gravitate towards player
4E17: BD 26 C3    JSR   $26C3              ; JMP $3199 - get random
position on playfield for object (returns: A = X coordinate, B = Y
coordinate)
4E1A: 20 02       BRA   $4E1E              ; and skip over the next
line...

4E1C: DC 5E       LDD   $5E                ; read player's blitter
destination

4E1E: DD 2B       STD   $2B
; what this piece of code does is check to see if the screen
destination in $2B
; is 16 pixels above or below the tank. If it's less than 16 pixels,
the B register is cleared to indicate "no vertical movement"
4E20: E0 05       SUBB  $0005,X            ; B = B - low byte of
tank blitter destination (vertical component)
4E22: 24 01       BCC   $4E25              ; if the subtraction
didn't cause a carry, goto $4E25
4E24: 50          NEGB                     ; OK, the subtraction
caused a carry and the result is negative. NEGB will make the result
positive again.
4E25: C1 10       CMPB  #$10               ; is the vertical
distance between the tank and the destination #$10 (16 decimal) ?
4E27: 24 03       BCC   $4E2C              ; it's more than 16
pixels, goto $4E2C
4E29: 5F          CLRB                     ; ok, distance is 16
pixels or less, clear the B register (see $4E3C comments) to make
tank move horizontally only
4E2A: 20 09       BRA   $4E35              ; and skip over the
lines which determine whether to move tank up or down (as we're not
moving vertically)

4E2C: DC 2B       LDD   $2B                ; get screen destination
where tank wants move into A & B (A=horizontal part, B = vertical
part)
4E2E: E1 05       CMPB  $0005,X            ; compare vertical part
of screen destination with tank's current vertical position
4E30: C6 01       LDB   #$01               ; when B = 1, tank will
move down
4E32: 24 01       BCC   $4E35              ; if vertical pos of
screen destination > tank's current vertical position goto $4E35
```

```
4E34: 50          NEGB                      ; B = #$FF, tank will
move up

4E35: A1 04       CMPA  $0004,X             ; compare horizontal pos
of screen destination with tank's current X coordinate
4E37: 86 01       LDA   #$01                ; when A = 1, tank will
move right
4E39: 24 01       BCC   $4E3C               ; if horizontal pos of
screen destination > tank's current X coordinate goto $4E35
4E3B: 40          NEGA                      ; move tank left
;
; At this point: A = horizontal direction of tank ($FF = left, 0 =
no horizontal movement, 1=right)
; B = vertical direction of tank ($FF = up, 0 = no vertical
movement, 1= down)
;
4E3C: ED 4B       STD   $000B,U              ; $000B = horizontal
direction, $000C = vertical direction
4E3E: 96 84       LDA   $84                  ; read a random number
4E40: 84 1F       ANDA  #$1F                 ; mask in bits 0..4,
giving us a number from 0..31
4E42: 4C          INCA                       ; ensure number is
nonzero
4E43: A7 4E       STA   $000E,U              ; set "move tank"
countdown (see $4DE3)
4E45: 39          RTS



;
; X = pointer to tank
;
CREATE_TANK_SHELL:
4E46: 34 50       PSHS  U,X
4E48: 31 84       LEAY  ,X                   ; Y = pointer to tank
4E4A: B6 BE 64    LDA   $BE64
4E4D: A7 4D       STA   $000D,U
4E4F: 96 42       LDA   $42
4E51: 81 11       CMPA  #$11
4E53: 10 24 01 3B LBCC  $4F92
4E57: 96 F1       LDA   $F1                  ; read count of tank
shells on screen
4E59: 81 14       CMPA  #$14                 ; compare to #$14 (20
decimal)
4E5B: 10 22 01 33 LBHI  $4F92                ; if higher than 20
decimal, goto $4F92 (exits function)
4E5F: 0C F1       INC   $F1                  ; increment count of
tank shells on screen
4E61: BD D0 6C    JSR   $D06C                ; JMP $D32B - create
entity with params and add to linked list at $9817
4E64: 4F 94       ; parameters to pass to $D06C - function to call
on next game cycle
4E66: 4F EE       ; parameters to pass to $D06C - animation frame
metadata pointer for tank shell
```

```
4E68: 4F D5       ; parameters to pass to $D06C - function to call
when this object has a collision
4E6A: 10 27 01 24 LBEQ  $4F92                ; if the zero flag is
set, then can't create shell, goto $4F92
4E6E: EC 24       LDD   $4,Y                 ; get blitter
destination of tank
4E70: C3 01 00    ADDD  #$0100               ; add 256 to it, making
result 2 pixels to right of previous
4E73: A7 0A       STA   $A,X                 ; set X coordinate of
shell
4E75: E7 0C       STB   $C,X                 ; set Y coordinate of
shell
4E77: ED 04       STD   $4,X                 ; set blitter
destination of shell
4E79: C6 80       LDB   #$80
4E7B: D1 84       CMPB  $84                  ; compare to a random
number
4E7D: 23 55       BLS   $4ED4
4E7F: D6 86       LDB   $86
4E81: C4 1F       ANDB  #$1F
4E83: CB F0       ADDB  #$F0
4E85: 96 5E       LDA   $5E                  ; read player blitter
destination hi byte (X component)
4E87: 81 11       CMPA  #$11
4E89: 24 01       BCC   $4E8C
4E8B: 5F          CLRB
4E8C: DB 5E       ADDB  $5E
4E8E: 4F          CLRA
4E8F: E0 04       SUBB  $4,X
4E91: 82 00       SBCA  #$00
4E93: 34 02       PSHS  A
4E95: 2A 01       BPL   $4E98
4E97: 50          NEGB
4E98: B6 BE 65    LDA   $BE65
4E9B: 3D          MUL
4E9C: 1F 89       TFR   A,B
4E9E: A6 E0       LDA   ,S
+                                       ; restore A from stack
(see $4E93)
4EA0: 2A 01       BPL   $4EA3
4EA2: 53          COMB
4EA3: 58          ASLB
4EA4: 49          ROLA
4EA5: 58          ASLB
4EA6: 49          ROLA
4EA7: 58          ASLB
4EA8: 49          ROLA
4EA9: ED 0E       STD   $000E,X              ; set X delta
4EAB: D6 86       LDB   $86
4EAD: C4 1F       ANDB  #$1F
4EAF: CB F0       ADDB  #$F0
4EB1: DB 5F       ADDB  $5F                  ; add lo byte (Y
component) of player blitter destination
4EB3: 4F          CLRA
```

```
4EB4: E0 05       SUBB  $0005,X
4EB6: 82 00       SBCA  #$00
4EB8: 34 02       PSHS  A
4EBA: 2A 01       BPL   $4EBD
4EBC: 50          NEGB
4EBD: B6 BE 65    LDA   $BE65
4EC0: 3D          MUL
4EC1: 1F 89       TFR   A,B
4EC3: A6 E0       LDA   ,S+
4EC5: 2A 01       BPL   $4EC8
4EC7: 53          COMB
4EC8: 58          ASLB
4EC9: 49          ROLA
4ECA: 58          ASLB
4ECB: 49          ROLA
4ECC: 58          ASLB
4ECD: 49          ROLA
4ECE: ED 88 10    STD   $10,X                ; set Y delta
4ED1: 7E 4F 82    JMP   $4F82

4ED4: BD D0 39    JSR   $D039               ; JMP $D6CD - get a
random number into A
4ED7: 44          LSRA
4ED8: 25 29       BCS   $4F03
4EDA: 4F          CLRA
4EDB: D6 84       LDB   $84
4EDD: C4 1F       ANDB  #$1F
4EDF: CB F0       ADDB  #$F0
4EE1: EB 05       ADDB  $0005,X
4EE3: DB 5F       ADDB  $5F
4EE5: 89 00       ADCA  #$00
4EE7: 44          LSRA
4EE8: 56          RORB
4EE9: 96 86       LDA   $86
4EEB: 84 07       ANDA  #$07
4EED: 27 0A       BEQ   $4EF9
4EEF: 96 5E       LDA   $5E                  ; get hi byte (X
component) of player blitter destination
4EF1: 81 4B       CMPA  #$4B
4EF3: 25 0A       BCS   $4EFF
4EF5: 86 8F       LDA   #$8F
4EF7: 20 33       BRA   $4F2C

4EF9: 96 5E       LDA   $5E                  ; get hi byte (X
component) of player blitter destination
4EFB: 81 4B       CMPA  #$4B
4EFD: 23 F6       BLS   $4EF5
4EFF: 86 07       LDA   #$07
4F01: 20 29       BRA   $4F2C

4F03: 4F          CLRA
4F04: D6 84       LDB   $84                  ; get a random number
into A
4F06: C4 0F       ANDB  #$0F
```

```
4F08: CB F8       ADDB  #$F8
4F0A: EB 04       ADDB  $0004,X
4F0C: DB 5E       ADDB  $5E
4F0E: 89 00       ADCA  #$00
4F10: 44          LSRA
4F11: 56          RORB
4F12: 96 86       LDA   $86
4F14: 84 07       ANDA  #$07
4F16: 27 0A       BEQ   $4F22
4F18: 96 5F       LDA   $5F                   ; get lo byte (Y
component) of player blitter destination
4F1A: 81 81       CMPA  #$81
4F1C: 25 0A       BCS   $4F28
4F1E: 86 EA       LDA   #$EA
4F20: 20 08       BRA   $4F2A

4F22: 96 5F       LDA   $5F                   ; get lo byte (Y
component) of player blitter destination
4F24: 81 81       CMPA  #$81
4F26: 23 F6       BLS   $4F1E
4F28: 86 18       LDA   #$18
4F2A: 1E 89       EXG   A,B
4F2C: 97 2B       STA   $2B
4F2E: 4F          CLRA
4F2F: E0 05       SUBB  $0005,X
4F31: 82 00       SBCA  #$00
4F33: ED 88 10    STD   $10,X               ; set Y delta
4F36: D6 2B       LDB   $2B
4F38: 4F          CLRA
4F39: E0 04       SUBB  $0004,X
4F3B: 82 00       SBCA  #$00
4F3D: ED 0E       STD   $000E,X             ; set X delta
4F3F: F6 BE 65    LDB   $BE65
4F42: 86 40       LDA   #$40
4F44: 3D          MUL
4F45: 1F 89       TFR   A,B
4F47: 4F          CLRA
4F48: 58          ASLB
4F49: 49          ROLA
4F4A: 58          ASLB
4F4B: 49          ROLA
4F4C: 34 06       PSHS  B,A
4F4E: 43          COMA
4F4F: 53          COMB
4F50: 34 06       PSHS  B,A
4F52: 58          ASLB
4F53: 49          ROLA
4F54: 34 06       PSHS  B,A
4F56: 43          COMA
4F57: 53          COMB
4F58: 34 06       PSHS  B,A
4F5A: EC 0E       LDD   $000E,X             ; read X delta
4F5C: 10 A3 64    CMPD  $0004,S
4F5F: 2F 1F       BLE   $4F80
```

```
4F61: 10 A3 66    CMPD  $0006,S
4F64: 2C 1A       BGE   $4F80
4F66: EC 88 10    LDD   $10,X
4F69: 10 A3 62    CMPD  $0002,S
4F6C: 2F 12       BLE   $4F80
4F6E: 10 A3 E4    CMPD  ,S
4F71: 2C 0D       BGE   $4F80
4F73: 58          ASLB
4F74: 49          ROLA
4F75: ED 88 10    STD   $10,X               ; set Y delta
4F78: EC 0E       LDD   $000E,X             ; read X delta
4F7A: 58          ASLB
4F7B: 49          ROLA
4F7C: ED 0E       STD   $000E,X             ; set X delta
4F7E: 20 DC       BRA   $4F5C

4F80: 32 68       LEAS  $0008,S
4F82: 9F 17       STX   $17
4F84: 96 85       LDA   $85
4F86: 84 1F       ANDA  #$1F
4F88: 8B 30       ADDA  #$30
4F8A: A7 4E       STA   $000E,U
4F8C: CC 4B 11    LDD   #$4B11
4F8F: BD D0 4B    JSR   $D04B               ; JMP $D3C7
4F92: 35 D0       PULS  X,U,PC ;(PUL? PC=RTS)


MAKE_TANK_SHELL_BOUNCE_IF_HITS_BORDER_WALL:
4F94: AE 47       LDX   $0007,U
4F96: EC 0A       LDD   $000A,X             ; get tank shell X
coordinate into D
4F98: E3 0E       ADDD  $000E,X             ; add X delta
4F9A: 81 07       CMPA  #$07                ; hit left border wall?
4F9C: 25 23       BCS   $4FC1               ; yes, so goto $4FC1 to
make the tank shell bounce off
4F9E: 81 8B       CMPA  #$8B                ; hit right border wall?
4FA0: 22 1F       BHI   $4FC1               ; yes, so goto $4FC1 to
make the tank shell bounce off
4FA2: EC 0C       LDD   $000C,X             ; get tank shell Y
coordinate into D
4FA4: E3 88 10    ADDD  $10,X               ; add Y delta to D
4FA7: 81 18       CMPA  #$18                ; hit top border wall?
4FA9: 25 1C       BCS   $4FC7               ; yes, so goto $4FC7 to
make the tank shell bounce off
4FAB: 81 E3       CMPA  #$E3                ; hit bottom border
wall?
4FAD: 22 18       BHI   $4FC7               ; yes, so goto $4FC7 to
make the tank shell bounce off
4FAF: 6A 4E       DEC   $000E,U             ; decrement tank shell
lifespan counter
4FB1: 27 08       BEQ   $4FBB               ; if =0 goto $4FBB to
remove the tank shell from the playfield
4FB3: 86 02       LDA   #$02
4FB5: 8E 4F 94    LDX   #$4F94
```

```
4FB8: 7E D0 66    JMP   $D066              ; JMP $D1E3 - allocate
function call

;
; Remove a tank shell from the screen.
;
; This is where the infamous "only 20 tank shells fired in a wave"
bug is to be found.
; The "current number of tank shells on screen" variable at $98F1
should be decremented when a tank shell "fizzles out".
; However, that never happens, so the game logic thinks there is
always 20 shells on screen, and prevents no more from being created.
;

REMOVE_TANK_SHELL:
; if you were able somehow to add DEC $F1 here, the bug would be
fixed....
4FBB: BD D0 75    JSR   $D075              ; JMP $D31B - deallocate
object and erase object from screen             ; JMP $D31B
4FBE: 7E D0 63    JMP   $D063              ; JMP $D1F3


;
; Called when the tank shell hits the left-most or right-most border
; The shell will fly off in the opposite direction
;
TANK_SHELL_BOUNCE_HORIZONTAL:
4FC1: 63 0E       COM   $000E,X            ; flip bits X delta
4FC3: 63 0F       COM   $000F,X
4FC5: 20 06       BRA   $4FCD


;
; Called when the tank shell hits the top-most or bottom-most border
; The shell will fly off in the opposite direction
;
TANK_SHELL_BOUNCE_VERTICAL:
4FC7: 63 88 10    COM   $10,X              ; flip bits Y delta
4FCA: 63 88 11    COM   $11,X
4FCD: CC 4B 16    LDD   #$4B16
4FD0: BD D0 4B    JSR   $D04B              ; JMP $D3C7
4FD3: 20 DE       BRA   $4FB3


SHELL_COLLISION_HANDLER:
4FD5: 96 48       LDA   $48                ; did the player
collision routine call this function?
4FD7: 26 14       BNE   $4FED              ; yes, goto $4FED
4FD9: 0A F1       DEC   $F1                ; decrement number of
tank shells on screen count
4FDB: BD 5B 43    JSR   $5B43              ; JMP $5C1F - create an
explosion
4FDE: BD D0 78    JSR   $D078              ; JMP $D320 - deallocate
object and erase object from screen (2)
```

```
4FE1: CC 00 25    LDD   #$0025
4FE4: BD D0 0C    JSR   $D00C               ; JMP $DB9C – update
player score
4FE7: CC 4B 1E    LDD   #$4B1E
4FEA: 7E D0 4B    JMP   $D04B               ; JMP $D3C7

4FED: 39          RTS

4FEE: 04 07 4F F2 00 0A 00 00 0A CC CA 00 0C 0B 0C A0
4FFE: AC BB BC A0 0C 0B 0C A0 0A CC CA 00 00 0A 00 00
500E: 02 04 50 36 FF FF 04 07 50 3E 00 FF 04 08 50 5A
501E: FF FE 06 0C 50 7A 00 FE 07 10 55 1E 07 10 55 8E
502E: 07 10 55 FE 07 10 56 6E 01 00 11 10 11 10 97 90
503E: 00 11 00 00 00 11 00 00 11 11 11 00 16 66 61 00
504E: 11 11 11 00 79 79 79 00 07 97 90 00 00 11 10 00
505E: 00 19 10 00 11 11 11 10 16 60 66 10 16 06 06 10
506E: 11 11 11 10 97 79 77 90 07 97 79 00 00 00 11 11
507E: 00 00 00 00 19 91 00 00 00 00 11 11 00 00 00 00
508E: 01 10 00 00 00 11 11 11 11 00 00 10 60 06 01 00
509E: 00 16 06 60 61 00 00 10 60 06 01 00 00 11 11 11
50AE: 11 00 00 97 79 77 97 00 07 00 00 00 00 70 00 79
50BE: 77 97 79 00 08 0F 50 E6 08 0F 51 5E 08 0F 51 D6
50CE: 08 0F 52 4E 08 0F 52 C6 08 0F 53 3E 08 0F 53 B6
50DE: 08 0F 54 2E 08 0F 54 A6 00 00 00 00 00 00 00 00
50EE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50FE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
510E: 00 00 00 00 00 00 00 00 00 00 00 EE E0 00 00 00
511E: 00 00 00 EA E0 00 00 00 00 00 00 EE E0 00 00 00
512E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
513E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
514E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
515E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
516E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
517E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
518E: 00 00 00 EE E0 00 00 00 00 00 00 EA E0 00 00 00
519E: 00 00 00 EE E0 00 00 00 00 00 00 00 00 00 00 00
51AE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
51BE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
51CE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
51DE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
51EE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
51FE: 00 00 0E EE EE 00 00 00 00 00 0E A0 AE 00 00 00
520E: 00 00 0E 0A 0E 00 00 00 00 00 0E A0 AE 00 00 00
521E: 00 00 0E EE EE 00 00 00 00 00 00 00 00 00 00 00
522E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
523E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
524E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
525E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
526E: 00 00 EE EE EE E0 00 00 00 00 EA 00 0A E0 00 00
527E: 00 00 E0 A0 A0 E0 00 00 00 00 E0 0A 00 E0 00 00
528E: 00 00 E0 A0 A0 E0 00 00 00 00 EA 00 0A E0 00 00
529E: 00 00 EE EE EE E0 00 00 00 00 00 00 00 00 00 00
52AE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
52BE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
52CE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
52DE: 00 0E EE EE EE EE 00 00 00 0E A0 00 00 AE 00 00
52EE: 00 0E 0A 00 0A 0E 00 00 00 0E 00 A0 A0 0E 00 00
52FE: 00 0E 00 00 00 0E 00 00 00 0E 00 A0 A0 0E 00 00
530E: 00 0E 0A 00 0A 0E 00 00 00 0E A0 00 00 AE 00 00
531E: 00 0E EE EE EE EE 00 00 00 00 00 00 00 00 00 00
532E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
533E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
534E: 00 EE 00 EE E0 0E E0 00 00 EA 00 00 00 0A E0 00
535E: 00 00 A0 00 00 A0 00 00 00 00 0A 00 0A 00 00 00
536E: 00 E0 00 E0 E0 00 E0 00 00 E0 00 00 00 00 E0 00
537E: 00 E0 00 E0 E0 00 E0 00 00 00 0A 00 0A 00 00 00
538E: 00 00 A0 00 00 A0 00 00 00 EA 00 00 00 0A E0 00
539E: 00 EE 00 EE E0 0E E0 00 00 00 00 00 00 00 00 00
53AE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
53BE: 0E E0 00 EE E0 00 EE 00 0E A0 00 00 00 00 AE 00
53CE: 00 0A 00 00 00 0A 00 00 00 00 A0 00 00 A0 00 00
53DE: 00 00 0E 00 0E 00 00 00 0E 00 00 00 00 00 0E 00
53EE: 0E 00 00 00 00 00 0E 00 0E 00 00 00 00 00 0E 00
53FE: 00 00 0E 00 0E 00 00 00 00 00 A0 00 00 A0 00 00
540E: 00 0A 00 00 00 0A 00 00 0E A0 00 00 00 00 AE 00
541E: 0E E0 00 EE E0 00 EE 00 00 00 00 00 00 00 00 00
542E: E0 00 00 0E 00 00 00 E0 0A 00 00 0E 00 00 0A 00
543E: 00 A0 00 00 00 00 A0 00 00 0A 00 00 00 0A 00 00
544E: 00 00 E0 00 00 E0 00 00 00 00 00 00 00 00 00 00
545E: 00 00 00 00 00 00 00 00 EE 00 00 00 00 00 0E E0
546E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
547E: 00 00 E0 00 00 E0 00 00 00 0A 00 00 00 0A 00 00
548E: 00 A0 00 00 00 00 A0 00 0A 00 00 0E 00 00 0A 00
549E: E0 00 00 0E 00 00 00 E0 00 00 00 0E 00 00 00 00
54AE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
54BE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
54CE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
54DE: E0 00 00 00 00 00 00 E0 00 00 00 00 00 00 00 00
54EE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
54FE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
550E: 00 00 00 00 00 00 00 00 00 00 00 0E 00 00 00 00
551E: 00 00 11 11 10 00 00 00 00 19 09 10 00 00 00 00
552E: 10 00 10 00 00 00 00 11 11 10 00 00 00 00 00 10
553E: 00 00 00 01 11 11 11 11 11 00 01 00 00 00 00 01
554E: 00 01 06 60 00 66 01 00 01 06 06 06 06 01 00 01
555E: 00 66 66 60 01 00 01 00 00 00 00 01 00 01 11 11
556E: 11 11 11 00 09 77 79 77 79 77 00 70 00 00 00 00
557E: 00 70 07 00 00 00 00 09 00 00 79 77 79 77 70 00
558E: 00 00 11 11 10 00 00 00 00 19 09 10 00 00 00 00
559E: 10 00 10 00 00 00 00 11 11 10 00 00 00 00 00 10
55AE: 00 00 00 01 11 11 11 11 11 00 01 00 00 00 00 01
55BE: 00 01 06 06 06 66 01 00 01 06 06 00 00 01 00 01
55CE: 06 06 66 66 01 00 01 00 00 00 00 01 00 01 11 11
55DE: 11 11 11 00 07 97 77 97 77 97 00 70 00 00 00 00
55EE: 00 70 07 00 00 00 00 07 00 00 97 77 97 77 90 00
55FE: 00 00 11 11 10 00 00 00 00 10 00 10 00 00 00 00
560E: 10 00 10 00 00 00 00 11 11 10 00 00 00 00 00 10
561E: 00 00 00 01 11 11 11 11 11 00 01 00 00 00 00 01
```

```
562E: 00 01 00 66 06 60 01 00 01 06 06 06 06 01 00 01
563E: 06 60 60 66 01 00 01 00 00 00 00 01 00 01 11 11
564E: 11 11 11 00 07 79 77 79 77 79 00 70 00 00 00 00
565E: 00 70 09 00 00 00 00 07 00 00 77 79 77 79 70 00
566E: 00 00 11 11 10 00 00 00 00 10 00 10 00 00 00 00
567E: 10 00 10 00 00 00 00 11 11 10 00 00 00 00 00 10
568E: 00 00 00 01 11 11 11 11 11 00 01 00 00 00 00 01
569E: 00 01 06 66 06 06 01 00 01 00 00 06 06 01 00 01
56AE: 06 66 66 06 01 00 01 00 00 00 00 01 00 01 11 11
56BE: 11 11 11 00 07 77 97 77 97 77 00 90 00 00 00 00
56CE: 00 90 07 00 00 00 00 07 00 00 77 97 77 97 70 00
56DE: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
56EE: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
56FE: FF FF

5700: 7E 57 03    JMP   $5703

5703: 35 06       PULS  A,B
5705: DE 15       LDU   $15
5707: ED 47       STD   $0007,U
5709: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
570C: 59 B0          ; values to pass to the function
570E: 86 EF       LDA   #$EF
5710: 8E 3B 80    LDX   #$3B80
5713: 10 8E 5A 82 LDY   #$5A82
5717: AF 49       STX   $0009,U
5719: 10 AF 4B    STY   $000B,U
571C: A7 4D       STA   $000D,U
571E: 86 01       LDA   #$01
5720: 8E 57 26    LDX   #$5726
5723: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

5726: AE 49       LDX   $0009,U
5728: 10 AE 4B    LDY   $000B,U
572B: A6 4D       LDA   $000D,U
572D: C6 02       LDB   #$02
572F: E7 4E       STB   $000E,U
5731: BD 5A 11    JSR   $5A11
5734: 4D          TSTA
5735: 27 1A       BEQ   $5751
5737: 81 12       CMPA  #$12
5739: 26 04       BNE   $573F
573B: 86 EF       LDA   #$EF
573D: 20 12       BRA   $5751

573F: 81 F1       CMPA  #$F1
5741: 26 04       BNE   $5747
5743: 86 DE       LDA   #$DE
5745: 20 0A       BRA   $5751

5747: 81 23       CMPA  #$23
5749: 26 04       BNE   $574F
```

```
574B: 86 F1       LDA   #$F1
574D: 20 02       BRA   $5751

574F: 80 22       SUBA  #$22
5751: 8C 06 16    CMPX  #$0616
5754: 27 0E       BEQ   $5764
5756: 30 89 FE FE LEAX  $FEFE,X
575A: 31 A9 01 02 LEAY  $0102,Y
575E: 6A 4E       DEC   $000E,U
5760: 26 CF       BNE   $5731
5762: 20 B3       BRA   $5717

5764: 4D          TSTA
5765: 27 03       BEQ   $576A
5767: 4F          CLRA
5768: 20 A6       BRA   $5710

576A: 6E D8 07    JMP   [$07,U]

576D: 59 84 1F 00 59 00 3F 00 58 A8 3F 00 57 9D 0F 00
577D: 57 B5 0F 00 57 CD 0F 00 57 E9 0F 00 58 39 1F 00
578D: 58 6B 1F 00 58 91 0F 00 58 07 1F 00 59 58 1F 00
579D: 01 02 03 04 05 06 07 0F 17 1F 2D 34 3A 7A BA FA
57AD: F8 F0 E0 D0 C0 C0 00 00 C0 C0 D0 E0 F0 F8 FA BA
57BD: 7A 3A 34 2D 1F 17 0F 07 06 05 04 03 02 01 00 00
57CD: C0 C1 C2 C3 C4 C5 C6 C7 87 87 47 47 07 07 47 47
57DD: 87 87 C7 C7 C6 C5 C4 C3 C2 C1 00 00 38 38 31 3A
57ED: 3B 3C 2D 2E 2F 27 1F 17 17 0F 07 07 0F 17 17 1F
57FD: 27 2F 2E 2D 2C 3B 3A 39 00 00 38 39 3A 3B 3C 3D
580D: 3E 3F 37 2F 27 17 0F 07 06 05 04 03 02 01 01 01
581D: 49 CA DA E8 F8 F9 FA FB FD FF BF 3F 3E C0 C0 C0
582D: 07 07 38 38 38 07 C0 38 FF FF 00 00 38 39 3A 3B
583D: 3C 3D 3E 3F 37 2F 27 1F 17 47 47 87 87 C7 C7 C6
584D: C5 CC CB CA DA E8 F8 F9 FA FB FD FF BF 3F 3E C0
585D: C0 C0 07 07 38 38 38 07 C0 38 FF FF 00 00 38 39
586D: 3A 3B 3C 3D 3E 3F 37 2F 27 1F 17 47 47 87 87 C7
587D: C7 C6 C5 CC CB CA DA E8 F8 F9 FA FB FD FF BF 3F
588D: 3E 3C 00 00 37 2F 27 1F 17 47 47 87 87 C7 C7 C6
589D: C5 CC CB CA C0 D0 98 38 33 00 00 07 0F 17 1F 27
58AD: 2F 37 3F 3F 7F 7F BF BF FF FF FF BF BF 7F 7F 3F
58BD: 3F 3E 3D 3C 3B 3A 39 38 38 30 28 20 08 08 49 52
58CD: A5 FB FC FD FE FF FF FE FD FC FB FA F9 F8 F0 E8
58DD: E0 D8 D0 C8 C0 80 40 01 01 01 01 02 03 04 05 06
58ED: 4F EF F7 FF FF F7 EF E7 DF D7 CF C7 87 87 47 47
58FD: 07 00 00 07 0F 17 1F 27 2F 37 3F 3F 7F 7F BF BF
590D: FF FF FF BF BF 7F 7F 3F 3F 3E 3D 3C 3B 3A 39 38
591D: 38 78 78 B8 B8 F8 F8 F9 FA FB FC FD FE FF FF FE
592D: FD FC FB FA F9 F8 F0 E8 E0 D8 D0 C8 C0 C1 C2 C3
593D: C4 C5 C6 C7 C7 CF D7 DF E7 EF F7 FF FF F7 EF E7
594D: DF D7 CF C7 87 87 47 47 07 00 00 07 0F 17 1F 27
595D: 2F 37 3F 3E 3D 3C 3B 3A 39 38 38 78 78 B8 B8 F8
596D: F8 F0 E8 E0 D8 D0 C8 C0 80 41 01 01 02 03 04 05
597D: 06 07 07 07 07 00 00 07 0F 17 1F 27 2F 37 3F 3E
598D: 3D 3C 3B 3A 39 38 38 78 78 B8 B8 F8 F8 F0 E8 E0
```

```
599D: D8 D0 C8 C0 C1 C2 C3 C4 C5 C6 C7 C7 87 87 47 47
59AD: 07 00 00 10

59B1: 8E 57 6D    LDX   #$576D
59B4: BD D0 39    JSR   $D039
59B7: 84 0F       ANDA  #$0F
59B9: 81 0C       CMPA  #$0C
59BB: 24 F7       BCC   $59B4
59BD: 48          ASLA
59BE: 48          ASLA
59BF: 1F 89       TFR   A,B
59C1: AE A5       LDX   B,Y
59C3: CB 02       ADDB  #$02
59C5: BD D0 39    JSR   $D039               ; JMP $D6CD - get a
random number into A
59C8: A4 A5       ANDA  B,Y
59CA: AF 47       STX   $0007,U
59CC: 30 86       LEAX  A,X
59CE: AF 49       STX   $0009,U
59D0: AE 49       LDX   $0009,U
59D2: 30 01       LEAX  $0001,X
59D4: 6D 84       TST   ,X
59D6: 26 02       BNE   $59DA
59D8: AE 47       LDX   $0007,U
59DA: AF 49       STX   $0009,U
59DC: 10 8E 98 01 LDY   #$9801
59E0: A6 80       LDA   ,X+
59E2: 26 04       BNE   $59E8
59E4: AE 47       LDX   $0007,U
59E6: 20 F8       BRA   $59E0

59E8: A7 A0       STA   ,Y+
59EA: 10 8C 98 10 CMPY  #$9810
59EE: 25 F0       BCS   $59E0
59F0: C6 00       LDB   #$00
59F2: A6 4B       LDA   $000B,U
59F4: 4A          DECA
59F5: 81 05       CMPA  #$05
59F7: 25 02       BCS   $59FB
59F9: 86 04       LDA   #$04
59FB: A7 4B       STA   $000B,U
59FD: 8E 98 01    LDX   #$9801
5A00: E7 86       STB   A,X
5A02: 30 05       LEAX  $0005,X
5A04: 8C 98 10    CMPX  #$9810
5A07: 25 F7       BCS   $5A00
5A09: 86 01       LDA   #$01
5A0B: 8E 59 D0    LDX   #$59D0
5A0E: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

5A11: 34 76       PSHS  U,Y,X,B,A
5A13: 84 F0       ANDA  #$F0
5A15: 97 CC       STA   $CC
```

```
5A17: A6 E4       LDA   ,S
5A19: 84 0F       ANDA  #$0F
5A1B: 97 CD       STA   $CD
5A1D: 1F 10       TFR   X,D
5A1F: 97 CA       STA   $CA
5A21: D7 C8       STB   $C8
5A23: 1F 20       TFR   Y,D
5A25: 97 CB       STA   $CB
5A27: D7 C9       STB   $C9
5A29: D0 C8       SUBB  $C8
5A2B: 56          RORB
5A2C: 24 02       BCC   $5A30
5A2E: 0A C9       DEC   $C9
5A30: BD 5A C3    JSR   $5AC3
5A33: 96 C8       LDA   $C8
5A35: BD 5A 89    JSR   $5A89
5A38: 4C          INCA
5A39: BD 5A CC    JSR   $5ACC
5A3C: 8D 6C       BSR   $5AAA
5A3E: 96 C9       LDA   $C9
5A40: BD 5A C3    JSR   $5AC3
5A43: 8D 44       BSR   $5A89
5A45: 4A          DECA
5A46: BD 5A CC    JSR   $5ACC
5A49: 8D 5F       BSR   $5AAA
5A4B: 96 CA       LDA   $CA
5A4D: BD 5A D5    JSR   $5AD5
5A50: 8D 09       BSR   $5A5B
5A52: 96 CB       LDA   $CB
5A54: BD 5A DA    JSR   $5ADA
5A57: 8D 02       BSR   $5A5B
5A59: 35 F6       PULS  A,B,X,Y,U,PC ;(PUL? PC=RTS)

5A5B: 34 17       PSHS  X,B,A,CC
5A5D: 8D 1C       BSR   $5A7B
5A5F: 1A 10       ORCC  #$10
5A61: B7 CA 01    STA   blitter_mask
5A64: 86 05       LDA   #$05
5A66: C8 04       EORB  #$04
5A68: FD CA 06    STD   blitter_w_h
5A6B: CC 00 00    LDD   #$0000
5A6E: FD CA 02    STD   blitter_source
5A71: BF CA 04    STX   blitter_dest
5A74: 86 12       LDA   #$12
5A76: B7 CA 00    STA   start_blitter
5A79: 35 97       PULS  CC,A,B,X,PC ;(PUL? PC=RTS)

5A7B: 34 04       PSHS  B
5A7D: D6 C8       LDB   $C8
5A7F: 5C          INCB
5A80: 1F 01       TFR   D,X
5A82: D6 C9       LDB   $C9
5A84: D0 C8       SUBB  $C8
5A86: 5A          DECB
```

```
5A87: 35 82       PULS  A,PC ;(PUL? PC=RTS)

5A89: 34 17       PSHS  X,B,A,CC
5A8B: 8D 28       BSR   $5AB5
5A8D: 5C          INCB
5A8E: 1A 10       ORCC  #$10                ; disable interrupts
5A90: B7 CA 01    STA   blitter_mask
5A93: C8 04       EORB  #$04                ; XOR height with 4 as
the blitter chip needs this
5A95: F7 CA 06    STB   blitter_height      ; set height
5A98: 86 05       LDA   #$05                ; 1 XORed with 4 for the
blitter chip = 5
5A9A: B7 CA 07    STA   blitter_width
5A9D: FD CA 02    STD   blitter_source
5AA0: BF CA 04    STX   blitter_dest
5AA3: 86 12       LDA   #$12
5AA5: B7 CA 00    STA   start_blitter
5AA8: 35 97       PULS  CC,A,B,X,PC ;(PUL? PC=RTS)

5AAA: 34 17       PSHS  X,B,A,CC
5AAC: 8D 07       BSR   $5AB5
5AAE: 5A          DECB
5AAF: 30 89 01 00 LEAX  $0100,X
5AB3: 20 D9       BRA   $5A8E

5AB5: 34 04       PSHS  B
5AB7: 1F 89       TFR   A,B
5AB9: 96 CA       LDA   $CA
5ABB: 1F 01       TFR   D,X
5ABD: D6 CB       LDB   $CB
5ABF: D0 CA       SUBB  $CA
5AC1: 35 82       PULS  A,PC ;(PUL? PC=RTS)

5AC3: D6 CC       LDB   $CC
5AC5: 54          LSRB
5AC6: 54          LSRB
5AC7: 54          LSRB
5AC8: 54          LSRB
5AC9: DA CC       ORB   $CC
5ACB: 39          RTS

5ACC: D6 CD       LDB   $CD
5ACE: 58          ASLB
5ACF: 58          ASLB
5AD0: 58          ASLB
5AD1: 58          ASLB
5AD2: DA CD       ORB   $CD
5AD4: 39          RTS

5AD5: D6 CC       LDB   $CC
5AD7: DA CD       ORB   $CD
5AD9: 39          RTS

5ADA: D6 CC       LDB   $CC
```

```
5ADC: 54          LSRB
5ADD: 54          LSRB
5ADE: 54          LSRB
5ADF: 54          LSRB
5AE0: 34 04       PSHS  B
5AE2: D6 CD       LDB   $CD
5AE4: 58          ASLB
5AE5: 58          ASLB
5AE6: 58          ASLB
5AE7: 58          ASLB
5AE8: EA E0       ORB   ,S+
5AEA: 39          RTS

5AEB: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5AFB: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5B0B: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5B1B: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5B2B: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5B3B: FF FF FF FF FF

5B40: 7E 5B 5E    JMP   $5B5E

5B43: 7E 5C 1F    JMP   $5C1F

5B46: 7E 5B C6    JMP   $5BC6

5B49: 7E 5E 15    JMP   $5E15

5B4C: 7E 5E 38    JMP   $5E38

5B4F: 7E 5C 0A    JMP   $5C0A

5B52: 7E 5C 14    JMP   $5C14

5B55: 7E 5B 98    JMP   $5B98

5B58: 7E 5B B1    JMP   $5BB1

5B5B: 7E 5B BB    JMP   $5BBB

5B5E: 0F 96       CLR   $96
5B60: 0F 97       CLR   $97
5B62: 0F 98       CLR   $98
5B64: 0F 99       CLR   $99
5B66: BD F0 09    JSR   $F009               ; clear explosion list
5B69: 7E 46 80    JMP   $4680

5B6C: 34 10       PSHS  X
5B6E: DE 1B       LDU   $1B
5B70: 27 0E       BEQ   $5B80
5B72: AE C4       LDX   ,U
5B74: 9F 1B       STX   $1B
5B76: 9E 96       LDX   $96
5B78: AF C4       STX   ,U
```

```
5B7A: DF 96       STU   $96
5B7C: 1C FE       ANDCC #$FE                ; clear carry flag
5B7E: 35 90       PULS  X,PC ;(PUL? PC=RTS)

5B80: 1A 01       ORCC  #$01
5B82: 35 90       PULS  X,PC ;(PUL? PC=RTS)

5B84: 34 10       PSHS  X
5B86: DE 1B       LDU   $1B
5B88: 27 F6       BEQ   $5B80
5B8A: AE C4       LDX   ,U
5B8C: 9F 1B       STX   $1B
5B8E: 9E 98       LDX   $98
5B90: AF C4       STX   ,U
5B92: DF 98       STU   $98
5B94: 1C FE       ANDCC #$FE                ; clear carry flag
5B96: 35 90       PULS  X,PC ;(PUL? PC=RTS)

5B98: 34 76       PSHS  U,Y,X,B,A
5B9A: 8D E8       BSR   $5B84
5B9C: 25 6A       BCS   $5C08
5B9E: ED 49       STD   $0009,U
5BA0: CC 0E 0E    LDD   #$0E0E
5BA3: ED C8 10    STD   $10,U
5BA6: 20 30       BRA   $5BD8



; Read the CMOS "Fancy Attract Mode" setting.
;
; Returns: A with fancy attract mode flag setting. 1 = On, 0 = off
;

GET_FANCY_ATTRACT_MODE_FLAG:
5BA8: 34 02       PSHS  A
5BAA: B6 CC 13    LDA   $CC13               ; "fancy attract mode"
on? (=YES)
5BAD: 84 0F       ANDA  #$0F
5BAF: 35 82       PULS  A,PC ;(PUL? PC=RTS)

5BB1: 8D F5       BSR   $5BA8               ; read fancy attract
mode flag
5BB3: 10 26 94 58 LBNE  $F00F               ; JMP $F066
5BB7: 34 76       PSHS  U,Y,X,B,A
5BB9: 20 0D       BRA   $5BC8

5BBB: 8D EB       BSR   $5BA8               ; read fancy attract
mode flag
5BBD: 10 26 94 4B LBNE  $F00C               ; if fancy attract mode
is on, goto $F00C (which is a JMP $F0D7)
5BC1: 34 76       PSHS  U,Y,X,B,A
5BC3: 7E 5C 3A    JMP   $5C3A

5BC6: 34 76       PSHS  U,Y,X,B,A
```

```
5BC8: 8D BA       BSR   $5B84
5BCA: 25 3C       BCS   $5C08
5BCC: CC 0A 0A    LDD   #$0A0A
5BCF: ED C8 10    STD   $10,U
5BD2: EC 04       LDD   $0004,X
5BD4: ED 49       STD   $0009,U
5BD6: AE 02       LDX   $0002,X
5BD8: D6 A7       LDB   $A7
5BDA: E7 44       STB   $0004,U
5BDC: E0 4A       SUBB  $000A,U
5BDE: 25 04       BCS   $5BE4
5BE0: E1 01       CMPB  $0001,X
5BE2: 25 0B       BCS   $5BEF
5BE4: E6 84       LDB   ,X
5BE6: 54          LSRB
5BE7: E7 45       STB   $0005,U
5BE9: EB 4A       ADDB  $000A,U
5BEB: E7 44       STB   $0004,U
5BED: 20 02       BRA   $5BF1

5BEF: E7 45       STB   $0005,U
5BF1: EC 84       LDD   ,X
5BF3: ED 4B       STD   $000B,U
5BF5: C6 01       LDB   #$01
5BF7: E7 4F       STB   $000F,U
5BF9: 88 04       EORA  #$04
5BFB: C8 04       EORB  #$04
5BFD: ED 4D       STD   $000D,U
5BFF: AE 02       LDX   $0002,X
5C01: AF 42       STX   $0002,U
5C03: CC 10 00    LDD   #$1000
5C06: ED 46       STD   $0006,U
5C08: 35 F6       PULS  A,B,X,Y,U,PC ;(PUL? PC=RTS)



5C0A: CC 01 00    LDD   #$0100
5C0D: DD 88       STD   $88               ; set player laser
direction fields
5C0F: 8D 0E       BSR   $5C1F             ; create explosion
5C11: 7E 5B 5B    JMP   $5B5B


5C14: CC 01 01    LDD   #$0101            ; down, right
5C17: DD 88       STD   $88               ; set laser direction
fields
5C19: 8D 04       BSR   $5C1F             ; create explosion
5C1B: 86 FF       LDA   #$FF
5C1D: 97 88       STA   $88



;
```

```
; When an enemy has been shot, this routine is called to make it
explode.
;
; X = enemy that's been shot
; $88 = horizontal direction of player's laser ($FF = left, 0=no
horizontal movement, 1 = right)
; $89 =vertical direction of players laser ($FF = up, 0=no vertical
movement, 1 = down)
;

CREATE_EXPLOSION:
5C1F: 34 76       PSHS  U,Y,X,B,A
5C21: 96 88       LDA   $88                 ; A = horizontal
direction of player's laser ($FF = left, 0=laser moving vertical
only, 1 = right)
5C23: 26 07       BNE   $5C2C               ; if the players laser
is moving on the horizontal axis, goto $5C2C
5C25: BD 5B 5B    JSR   $5B5B               ; JMP $5BBB (which when
fancy mode is on, goes to $F0D7)
5C28: 24 7D       BCC   $5CA7               ; if carry clear, exit
5C2A: 20 13       BRA   $5C3F

5C2C: D6 89       LDB   $89                 ; B = vertical direction
of players laser ($FF = up, 0=laser moving horizontal only, 1 =
down)
5C2E: 27 0A       BEQ   $5C3A               ; if players laser is
moving on the horizontal axis but not vertical axis, goto $5C3A
5C30: 98 89       EORA  $89                 ; A = A xor laser
vertical direction
5C32: 43          COMA                      ; flip bits
5C33: BD 46 83    JSR   $4683               ; JMP $473F
5C36: 24 6F       BCC   $5CA7               ; if carry clear, exit
5C38: 20 05       BRA   $5C3F

5C3A: BD 5B 6C    JSR   $5B6C
5C3D: 24 0A       BCC   $5C49
5C3F: 10 AE 02    LDY   $0002,X              ; Y = pointer to
animation frame metadata
5C42: EC 04       LDD   $0004,X              ; D = blitter
destination
5C44: BD D0 1E    JSR   $D01E                ; JMP $DABF: clear
image rectangle
5C47: 20 5E       BRA   $5CA7                ; exit

5C49: EC 04       LDD   $0004,X
5C4B: AE 02       LDX   $0002,X
5C4D: ED 49       STD   $0009,U
5C4F: CC 0A 0A    LDD   #$0A0A
5C52: ED C8 10    STD   $10,U
5C55: D6 A7       LDB   $A7
5C57: E7 44       STB   $0004,U
5C59: E0 4A       SUBB  $000A,U
5C5B: 25 04       BCS   $5C61
5C5D: E1 01       CMPB  $0001,X
```

```
5C5F: 25 0B       BCS   $5C6C
5C61: E6 01       LDB   $0001,X
5C63: 54          LSRB
5C64: E7 45       STB   $0005,U
5C66: EB 4A       ADDB  $000A,U
5C68: E7 44       STB   $0004,U
5C6A: 20 02       BRA   $5C6E

5C6C: E7 45       STB   $0005,U
5C6E: 10 8E C3 A5 LDY   #$C3A5
5C72: EC 84       LDD   ,X
5C74: ED 4B       STD   $000B,U
5C76: E7 4F       STB   $000F,U
5C78: C6 01       LDB   #$01
5C7A: 88 04       EORA  #$04
5C7C: C8 04       EORB  #$04
5C7E: ED 4D       STD   $000D,U
5C80: 96 59       LDA   $59
5C82: 2B 16       BMI   $5C9A
5C84: A6 A9 FA CD LDA   $FACD,Y
5C88: 81 4A       CMPA  #$4A
5C8A: 27 0E       BEQ   $5C9A
5C8C: 96 86       LDA   $86
5C8E: 26 0A       BNE   $5C9A
5C90: D6 85       LDB   $85
5C92: 86 98       LDA   #$98
5C94: 34 06       PSHS  B,A
5C96: EF F4       STU   [,S]
5C98: 35 06       PULS  A,B
5C9A: AE 02       LDX   $0002,X
5C9C: AF 42       STX   $0002,U
5C9E: CC 01 00    LDD   #$0100
5CA1: ED 46       STD   $0006,U
5CA3: 86 10       LDA   #$10
5CA5: A7 48       STA   $0008,U
5CA7: 35 F6       PULS  A,B,X,Y,U,PC ;(PUL? PC=RTS)



5CA9: E6 2F       LDB   $000F,Y
5CAB: C0 10       SUBB  #$10
5CAD: 50          NEGB
5CAE: 86 06       LDA   #$06
5CB0: 3D          MUL
5CB1: BE F0 17    LDX   $F017
5CB4: 3A          ABX
5CB5: 34 10       PSHS  X
5CB7: A6 26       LDA   $0006,Y
5CB9: 97 A8       STA   $A8
5CBB: A6 29       LDA   $0009,Y
5CBD: C6 12       LDB   #$12
5CBF: 1A 10       ORCC  #$10
5CC1: B7 CA 04    STA   blitter_dest
```

```
5CC4: A6 2A       LDA   $000A,Y
5CC6: CE 00 00    LDU   #$0000
5CC9: FF CA 01    STU   blitter_mask
5CCC: B7 CA 03    STA   $CA03
5CCF: EE 2D       LDU   $000D,Y
5CD1: FF CA 06    STU   blitter_w_h
5CD4: CE CA 05    LDU   #$CA05
5CD7: 8E CA 00    LDX   #start_blitter
5CDA: 39          RTS

5CDB: CE 98 96    LDU   #$9896
5CDE: 10 AC C4    CMPY  ,U
5CE1: 27 08       BEQ   $5CEB
5CE3: EE C4       LDU   ,U
5CE5: 26 F7       BNE   $5CDE
5CE7: 1A 10       ORCC  #$10
5CE9: 20 FE       BRA   $5CE9

5CEB: EC A4       LDD   ,Y
5CED: ED C4       STD   ,U
5CEF: DC 1B       LDD   $1B
5CF1: ED A4       STD   ,Y
5CF3: 10 9F 1B    STY   $1B
5CF6: 31 C4       LEAY  ,U
5CF8: 39          RTS

5CF9:  ROBOTRON: 2084  COPYRIGHT 1982
5D19: WILLIAMS ELECTRONICS INC.  ALL R
5D39: IGHTS RESERVED

5D48: EC 26       LDD   $0006,Y
5D4A: 83 00 80    SUBD  #$0080
5D4D: A1 26       CMPA  $0006,Y
5D4F: 26 03       BNE   $5D54
5D51: E7 27       STB   $0007,Y
5D53: 39          RTS

5D54: BD 5C A9    JSR   $5CA9
5D57: 96 59       LDA   $59
5D59: 26 08       BNE   $5D63
5D5B: DC 5E       LDD   $5E
5D5D: A7 29       STA   $0009,Y
5D5F: EB 25       ADDB  $0005,Y
5D61: E7 24       STB   $0004,Y
5D63: A6 2C       LDA   $000C,Y
5D65: 97 9C       STA   $9C
5D67: EC 26       LDD   $0006,Y
5D69: 81 01       CMPA  #$01
5D6B: 22 08       BHI   $5D75
5D6D: 8D 27       BSR   $5D96
5D6F: CE 98 98    LDU   #$9898
5D72: 7E 5C DE    JMP   $5CDE

5D75: 83 00 80    SUBD  #$0080
```

```
5D78: 81 01       CMPA  #$01
5D7A: 22 1A       BHI   $5D96
5D7C: A6 A8 10    LDA   $10,Y
5D7F: 81 0E       CMPA  #$0E
5D81: 26 EC       BNE   $5D6F
5D83: 86 01       LDA   #$01
5D85: 20 0F       BRA   $5D96

5D87: 6A 28       DEC   $0008,Y
5D89: 10 27 FF 4E LBEQ  $5CDB
5D8D: A6 2C       LDA   $000C,Y
5D8F: 97 9C       STA   $9C
5D91: EC 26       LDD   $0006,Y
5D93: C3 01 00    ADDD  #$0100
5D96: 97 A8       STA   $A8
5D98: ED 26       STD   $0006,Y
5D9A: 44          LSRA
5D9B: E6 25       LDB   $0005,Y
5D9D: 26 01       BNE   $5DA0
5D9F: 4F          CLRA
5DA0: 97 9F       STA   $9F
5DA2: AE 22       LDX   $0002,Y
5DA4: 96 A8       LDA   $A8
5DA6: E6 25       LDB   $0005,Y
5DA8: 3D          MUL
5DA9: DD 9D       STD   $9D
5DAB: E6 24       LDB   $0004,Y
5DAD: 4F          CLRA
5DAE: 93 9D       SUBD  $9D
5DB0: DB 9F       ADDB  $9F
5DB2: 89 00       ADCA  #$00
5DB4: 26 04       BNE   $5DBA
5DB6: C1 18       CMPB  #$18
5DB8: 22 16       BHI   $5DD0
5DBA: 0A 9C       DEC   $9C
5DBC: DB A8       ADDB  $A8
5DBE: 89 00       ADCA  #$00
5DC0: 26 F8       BNE   $5DBA
5DC2: C1 18       CMPB  #$18
5DC4: 23 F4       BLS   $5DBA
5DC6: E7 2A       STB   $000A,Y
5DC8: EC 2B       LDD   $000B,Y
5DCA: D0 9C       SUBB  $9C
5DCC: 3D          MUL
5DCD: 3A          ABX
5DCE: 20 02       BRA   $5DD2

5DD0: E7 2A       STB   $000A,Y
5DD2: 96 9C       LDA   $9C
5DD4: 4A          DECA
5DD5: D6 A8       LDB   $A8
5DD7: 3D          MUL
5DD8: EB 2A       ADDB  $000A,Y
5DDA: 89 00       ADCA  #$00
```

```
5DDC: 27 08       BEQ   $5DE6
5DDE: 0A 9C       DEC   $9C
5DE0: D0 A8       SUBB  $A8
5DE2: 82 00       SBCA  #$00
5DE4: 26 F8       BNE   $5DDE
5DE6: C1 EA       CMPB  #$EA
5DE8: 24 F4       BCC   $5DDE
5DEA: 96 9C       LDA   $9C
5DEC: 10 27 FE EB LBEQ  $5CDB
5DF0: A7 2F       STA   $000F,Y
5DF2: 80 10       SUBA  #$10
5DF4: 40          NEGA
5DF5: C6 0B       LDB   #$0B
5DF7: 3D          MUL
5DF8: F3 F0 15    ADDD  $F015
5DFB: 34 26       PSHS  Y,B,A               ; save return address on
stack
5DFD: EE A8 10    LDU   $10,Y
5E00: EC 2D       LDD   $000D,Y             ; get blitter width &
height
5E02: 1A 10       ORCC  #$10
5E04: FD CA 06    STD   blitter_w_h
5E07: A6 29       LDA   $0009,Y             ; get blitter
destination
5E09: B7 CA 04    STA   blitter_dest
5E0C: E6 2B       LDB   $000B,Y
5E0E: A6 2A       LDA   $000A,Y
5E10: 10 8E CA 05 LDY   #$CA05
5E14: 39          RTS                       ; pop return address off
stack (see $5DFB)

5E15: 10 9E 96    LDY   $96
5E18: 27 0B       BEQ   $5E25
5E1A: BD 5C A9    JSR   $5CA9
5E1D: BD 5D 87    JSR   $5D87               ; compute type of blit
to perform and do blit
5E20: 10 AE A4    LDY   ,Y
5E23: 26 F5       BNE   $5E1A
5E25: 10 9E 98    LDY   $98
5E28: 27 08       BEQ   $5E32
5E2A: BD 5D 48    JSR   $5D48
5E2D: 10 AE A4    LDY   ,Y
5E30: 26 F8       BNE   $5E2A
5E32: BD F0 12    JSR   $F012
5E35: 7E 46 89    JMP   $4689

5E38: DE 15       LDU   $15
5E3A: 35 06       PULS  A,B
5E3C: ED 4D       STD   $000D,U
5E3E: 86 0A       LDA   #$0A
5E40: A7 47       STA   $0007,U
5E42: 8E 98 5A    LDX   #$985A              ; player_object_start
5E45: BD D0 15    JSR   $D015               ; JMP $DB03 - erase
object from screen
```

```
5E48: 8E 98 5A    LDX   #$985A              ; player_object_start
5E4B: 86 99       LDA   #$99
5E4D: BD 38 88    JSR   $3888               ; JMP $3942
5E50: 86 02       LDA   #$02
5E52: 8E 5E 58    LDX   #$5E58
5E55: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

5E58: 96 84       LDA   $84
5E5A: 84 03       ANDA  #$03
5E5C: 8E 5E AE    LDX   #$5EAE
5E5F: A6 86       LDA   A,X
5E61: 8E 98 5A    LDX   #$985A              ; player_object_start
5E64: BD 38 88    JSR   $3888               ; JMP $3942
5E67: 6A 47       DEC   $0007,U
5E69: 27 08       BEQ   $5E73
5E6B: 86 06       LDA   #$06
5E6D: 8E 5E 48    LDX   #$5E48
5E70: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

5E73: BD D0 60    JSR   $D060               ; JMP $D1FF
5E76: BD D0 24    JSR   $D024
5E79: BD D0 5D    JSR   $D05D               ; JMP $D218 - deallocate
object metadata entry
5E7C: DE 15       LDU   $15
5E7E: 8E 5E B2    LDX   #$5EB2
5E81: AF 47       STX   $0007,U
5E83: 8E 98 5A    LDX   #$985A              ; player_object_start
5E86: 86 CC       LDA   #$CC
5E88: BD 38 88    JSR   $3888               ; JMP   $3942
5E8B: AE 47       LDX   $0007,U
5E8D: A6 80       LDA   ,X+
5E8F: 97 0C       STA   $0C
5E91: 27 0A       BEQ   $5E9D
5E93: AF 47       STX   $0007,U
5E95: 86 04       LDA   #$04
5E97: 8E 5E 8B    LDX   #$5E8B
5E9A: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

5E9D: 8E 98 5A    LDX   #$985A              ; player_object_start
5EA0: 6F 88 12    CLR   $12,X
5EA3: BD D0 15    JSR   $D015               ; JMP $DB03 - erase
object from screen
5EA6: BD D0 24    JSR   $D024
5EA9: DE 15       LDU   $15
5EAB: 6E D8 0D    JMP   [$0D,U]

5EAE: 00 11 33 77 FF F6 AD A4 5B 52 09 00

5EBA: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5ECA: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5EDA: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

```
5EEA: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5EFA: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5F0A: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5F1A: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5F2A: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5F3A: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5F4A: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5F5A: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5F6A: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5F7A: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
5F8A: FF FF FF FF FF FF

5F90: 7E 5F BE    JMP   $5FBE

5F93: 7E 60 23    JMP   $6023

5F96: 7E 61 3F    JMP   $613F

JMP_PRINT_STRING_LARGE_FONT:
5F99: 7E 61 47    JMP   PRINT_STRING_LARGE_FONT

5F9C: 7E 5F A2    JMP   $5FA2

5F9F: 7E 60 96    JMP   $6096

5FA2: 34 02       PSHS  A
5FA4: 86 11       LDA   #$11
5FA6: 97 CF       STA   $CF
5FA8: 35 02       PULS  A
5FAA: 34 02       PSHS  A
5FAC: 86 07       LDA   #$07
5FAE: 97 D2       STA   $D2
5FB0: 0F D0       CLR   $D0
5FB2: 0F D5       CLR   $D5
5FB4: 0F D7       CLR   $D7
5FB6: 86 01       LDA   #$01
5FB8: 97 D1       STA   $D1
5FBA: 9F D3       STX   $D3
5FBC: 35 82       PULS  A,PC ;(PUL? PC=RTS)



; Blit an alphanumeric or symbolic character to the screen, using
the small font
;
; A = ordinal of character to print (uses ASCII for A-Z)
; X = address of screen to blit character to
; $CF = Colour to draw character in
; $D7 = ???
; $D0 : if non-zero, set blitter to shift 1 pixel to right


BLIT_SMALL_CHARACTER:
5FBE: 34 66       PSHS  U,Y,B,A
```

```
5FC0: C6 39       LDB   #$39               ; prevent watchdog from
resetting system
5FC2: F7 CB FF    STB   watchdog
5FC5: C6 05       LDB   #$05
5FC7: D7 D2       STB   $D2
5FC9: 81 20       CMPA  #$20               ; SPACE?
5FCB: 26 02       BNE   $5FCF              ; no, goto $5FCF
5FCD: 86 3A       LDA   #$3A
5FCF: 10 BE E9 C8 LDY   $E9C8              ; contains pointer to
SMALL_CHARACTER_TABLE, which resolves to $E9CC
5FD3: 81 5E       CMPA  #$5E
5FD5: 22 4A       BHI   $6021
5FD7: 80 30       SUBA  #$30               ; subtract #$30 (48
decimal) from A
5FD9: 25 46       BCS   $6021              ; if there was a carry
(ie: result is negative) exit. A must be invalid
5FDB: 48          ASLA                     ; A = A * 2, to give an
offset to add to Y
5FDC: 10 AE A6    LDY   A,Y                ; read from
SMALL_CHARACTER_TABLE+ A to get a pointer to character to print
5FDF: A6 A4       LDA   ,Y                 ; get width of character
in pixels
5FE1: 44          LSRA                     ; divide by 2, to give
number of bytes to blit
5FE2: 4C          INCA                     ; ensure value is
nonzero
5FE3: 88 04       EORA  #$04               ; for blitter purposes.
Blitter needs width XOR 4
5FE5: C6 01       LDB   #$01
5FE7: 34 21       PSHS  Y,CC
5FE9: 1A 10       ORCC  #$10               ; disable interrupts
5FEB: FD CA 06    STD   blitter_w_h
5FEE: 31 21       LEAY  $0001,Y            ; Y ++ . Now Y points to
the pixel data to blit
5FF0: 10 BF CA 02 STY   blitter_source
5FF4: 96 CF       LDA   $CF
5FF6: B7 CA 01    STA   blitter_mask
5FF9: BF CA 04    STX   blitter_dest


5FFC: 86 1A       LDA   #$1A               ; Blitter flags:
Transparent + take source + remap
5FFE: D6 D0       LDB   $D0                ; read "blitter must do
pixel shift" flag
6000: 27 02       BEQ   $6004              ; if 0, no pixel shift
required

; if we get here then the character needs to be drawn one pixel to
the right. Remember, on the Robotron hardware there are 2 pixels per
byte,
; and in order for the blitter to begin drawing from an "odd" pixel
the special "shift one pixel right" bit needs to be set, as below:
6002: 86 3A       LDA   #$3A               ; Blitter flags:
Transparent + take source + remap + shift one pixel right
```

```
6004: B7 CA 00    STA   start_blitter
6007: 35 21       PULS  CC,Y
; Y = pointer to character to print
6009: A6 A4       LDA   ,Y                  ; get width of character
in pixels
600B: 4C          INCA                      ; add 1
600C: 5F          CLRB
600D: 44          LSRA                      ; then divide by 2, to
give number of bytes to adjust X by
600E: 30 8B       LEAX  D,X                 ; X += D. Now X points
to blit destination for *next* character
6010: 24 0F       BCC   $6021
; If we get here, the LSRA at $600D has caused a carry meaning the
width of the character is an odd number.
; the next character might require blitting on an odd pixel, so we
need to set/reset (toggle) the blitter pixel shift flag accordingly.
6012: 96 D0       LDA   $D0                 ; read "blitter must do
pixel shift" flag
6014: 27 07       BEQ   $601D               ; if no pixel shift
required, goto $601D
6016: 30 89 01 00 LEAX  $0100,X             ; add $0100 (256
decimal) to bump to next 2 pixels on same row
601A: 5F          CLRB
601B: 20 02       BRA   $601F

601D: C6 FF       LDB   #$FF
601F: D7 D0       STB   $D0                 ; set "blitter must do
pixel shift" flag
6021: 35 E6       PULS  A,B,Y,U,PC ;(PUL? PC=RTS)




; Blit an alphanumeric or symbolic character to the screen, using
the large font
;
; A = ordinal of character to blit (uses ASCII for A-Z)
; X = address of screen to blit character to
; $CF = Colour to draw character in
; $D7 = ???
; $D0 : if non-zero, set blitter to shift 1 pixel to right

BLIT_LARGE_CHARACTER:
6023: 34 66       PSHS  U,Y,B,A
6025: C6 39       LDB   #$39                ; prevent watchdog from
resetting system
6027: F7 CB FF    STB   watchdog
602A: C6 07       LDB   #$07
602C: D7 D2       STB   $D2
602E: 81 20       CMPA  #$20                ; SPACE?
6030: 26 02       BNE   $6034
6032: 86 3A       LDA   #$3A                ; if a space, change to
#$3A (58 decimal)
```

```
6034: 10 BE E9 CA LDY   $E9CA               ; Y always resolves to #
$EC34, which is the address of LARGE_CHARACTER_TABLE
6038: 81 5E       CMPA  #$5E                ; if > #$5E (94 decimal)
then return
603A: 22 E5       BHI   $6021
603C: 80 30       SUBA  #$30                ; subtract #$30 (48
decimal)
603E: 25 E1       BCS   $6021               ; if there was a carry
(ie: result is negative) exit. A must be invalid.
6040: 48          ASLA                      ; A = A * 2, to give an
offset to add to Y
6041: 10 AE A6    LDY   A,Y                 ; read from $EC34+ A to
get pointer to large character to print
6044: A6 A4       LDA   ,Y                  ; get width of character
in pixels
6046: 44          LSRA                      ; divide by 2 to give
number of bytes to blit (remember, 2 pixels per byte on Robotron
hardware)
6047: 4C          INCA                      ; add 1
6048: 88 04       EORA  #$04                ; for blitter purposes.
Blitter needs width XOR 4
604A: D6 D7       LDB   $D7                 ; get height adjustment
into B
604C: 2F 01       BLE   $604F               ; if Z | (N xor V) == 1
goto $604C
604E: 50          NEGB
604F: CB 06       ADDB  #$06
6051: C8 04       EORB  #$04                ; for blitter purposes.
Blitter needs height xor 4
6053: 34 21       PSHS  Y,CC
6055: 1A 10       ORCC  #$10                ; disable interrupts
6057: FD CA 06    STD   blitter_w_h
605A: 31 21       LEAY  $0001,Y             ; Y = Y + 1
605C: D6 D7       LDB   $D7
605E: 2F 05       BLE   $6065
6060: 88 04       EORA  #$04
6062: 3D          MUL
6063: 31 A5       LEAY  B,Y
6065: 10 BF CA 02 STY   blitter_source
6069: 96 CF       LDA   $CF                 ; get colour to blit
character in
606B: B7 CA 01    STA   blitter_mask
606E: BF CA 04    STX   blitter_dest
6071: 86 1A       LDA   #$1A                ; blitter flags (11010b)
- transparency mode + solid mode
6073: D6 D0       LDB   $D0                 ; read "blitter must do
pixel shift" flag
6075: 27 8D       BEQ   $6004               ; if no pixel shift
required, goto $6004
6077: 20 89       BRA   $6002               ; start the blit at X,
one pixel to right

6079: 0F D0       CLR   $D0
607B: 34 66       PSHS  U,Y,B,A
```

```
607D: 86 05       LDA   #$05
607F: 5F          CLRB
6080: 20 2F       BRA   $60B1

6082: 0F D0       CLR   $D0
6084: 34 66       PSHS  U,Y,B,A
6086: 86 05       LDA   #$05
6088: C6 01       LDB   #$01
608A: 20 25       BRA   $60B1

608C: 0F D0       CLR   $D0
608E: 34 66       PSHS  U,Y,B,A
6090: 86 07       LDA   #$07
6092: C6 01       LDB   #$01
6094: 20 1B       BRA   $60B1



6096: 0F D0       CLR   $D0
6098: 34 66       PSHS  U,Y,B,A
609A: C6 07       LDB   #$07                ; large font please
609C: D7 D2       STB   $D2
609E: C6 02       LDB   #$02
60A0: D7 D1       STB   $D1
60A2: 8D 65       BSR   $6109
60A4: A6 E4       LDA   ,S
60A6: 8D 65       BSR   $610D
60A8: 35 E6       PULS  A,B,Y,U,PC ;(PUL? PC=RTS)


60AA: 0F D0       CLR   $D0
60AC: 34 66       PSHS  U,Y,B,A
60AE: 86 07       LDA   #$07
60B0: 5F          CLRB
60B1: 97 D2       STA   $D2
60B3: D7 D1       STB   $D1
60B5: A6 E4       LDA   ,S
60B7: 20 02       BRA   $60BB

60B9: 34 66       PSHS  U,Y,B,A
60BB: 0F D6       CLR   $D6
60BD: 8D 4A       BSR   $6109
60BF: 0C D6       INC   $D6
60C1: A6 E4       LDA   ,S
60C3: 8D 48       BSR   $610D
60C5: 35 E6       PULS  A,B,Y,U,PC ;(PUL? PC=RTS)

60C7: 0F D0       CLR   $D0
60C9: 34 66       PSHS  U,Y,B,A
60CB: 86 05       LDA   #$05
60CD: 5F          CLRB
60CE: 20 1B       BRA   $60EB

60D0: 0F D0       CLR   $D0
```

```
60D2: 34 66       PSHS  U,Y,B,A
60D4: 86 05       LDA   #$05
60D6: C6 01       LDB   #$01
60D8: 20 11       BRA   $60EB

60DA: 0F D0       CLR   $D0
60DC: 34 66       PSHS  U,Y,B,A
60DE: 86 07       LDA   #$07
60E0: C6 01       LDB   #$01
60E2: 20 07       BRA   $60EB

60E4: 0F D0       CLR   $D0
60E6: 34 66       PSHS  U,Y,B,A
60E8: 86 07       LDA   #$07
60EA: 5F          CLRB
60EB: 97 D2       STA   $D2
60ED: D7 D1       STB   $D1
60EF: A6 E4       LDA   ,S
60F1: 20 02       BRA   $60F5

60F3: 34 66       PSHS  U,Y,B,A
60F5: 0F D6       CLR   $D6
60F7: 8D 10       BSR   $6109
60F9: A6 E4       LDA   ,S
60FB: 8D 10       BSR   $610D
60FD: A6 61       LDA   $0001,S
60FF: 8D 08       BSR   $6109
6101: 0C D6       INC   $D6
6103: A6 61       LDA   $0001,S
6105: 8D 06       BSR   $610D
6107: 35 E6       PULS  A,B,Y,U,PC ;(PUL? PC=RTS)

6109: 44          LSRA
610A: 44          LSRA
610B: 44          LSRA
610C: 44          LSRA
610D: 84 0F       ANDA  #$0F
610F: 26 08       BNE   $6119
6111: D6 D6       LDB   $D6
6113: 26 04       BNE   $6119
6115: D6 D1       LDB   $D1
6117: 26 0F       BNE   $6128
6119: 0C D6       INC   $D6
611B: 8B 30       ADDA  #$30
611D: D6 D2       LDB   $D2                 ; read font-size flag
611F: C1 07       CMPB  #$07                ; large font?
6121: 10 27 FE FE LBEQ  $6023               ; yes, branch to
BLIT_LARGE_CHARACTER routine
6125: 7E 5F BE    JMP   $5FBE               ; no, jump to
BLIT_SMALL_CHARACTER routine

6128: C1 02       CMPB  #$02
612A: 26 0E       BNE   $613A
612C: 30 89 02 00 LEAX  $0200,X
```

```
6130: D6 D2       LDB   $D2
6132: C1 05       CMPB  #$05
6134: 27 04       BEQ   $613A
6136: 30 89 01 00 LEAX  $0100,X
613A: 39          RTS

613B: 34 66       PSHS  U,Y,B,A
613D: 20 12       BRA   $6151



; Prints a text string in small characters.
;
; A = index of text string to draw (see TEXT_PTRS)
; B = parameter to insert into string (along the lines of in C:
printf("texthere: %d", param) or C#: string.Format("texthere: {0}",
param))
; for example, $34C0 (PRINT_WAVE_NUMBER) calls this function,
passing the wave number in B. This function then draws the string
"[n] WAVE"
;
; X = screen address to start drawing text string
;

PRINT_STRING_SMALL_FONT:
613F: 0F D0       CLR   $D0                 ; clear "blitter must do
pixel shift" flag -see $6002 for more info
6141: 34 66       PSHS  U,Y,B,A
6143: C6 05       LDB   #$05                ; small font size
indicator
6145: 20 06       BRA   $614D



; Prints a text string in large characters.
;
; A = index of text string to draw (see TEXT_PTRS)
; X = screen address to start drawing text string
;

PRINT_STRING_LARGE_FONT:
6147: 0F D0       CLR   $D0                 ; clear "blitter must do
pixel shift" flag -see $6002 for more info
6149: 34 66       PSHS  U,Y,B,A
614B: C6 07       LDB   #$07                ; large font size
indicator

614D: D7 D2       STB   $D2                 ; set font-size flag
(see $6190 for how this affects what size of font is rendered)
614F: 0F D7       CLR   $D7
6151: 34 42       PSHS  U,A

;
; Security related code
```

```
;

6153: CE 99 4B    LDU   #$994B
6156: C6 37       LDB   #$37
6158: 0D 59       TST   $59
615A: 2B 13       BMI   $616F
615C: E1 C8 A8    CMPB  -$58,U
615F: 27 0E       BEQ   $616F
6161: 96 85       LDA   $85
6163: 81 30       CMPA  #$30
6165: 22 08       BHI   $616F
6167: D6 84       LDB   $84                   ; read a random number
6169: 86 98       LDA   #$98                  ; make D = $98xx
616B: 1F 03       TFR   D,U
616D: 63 C4       COM   ,U                    ; flip the bits at U,
corrupting game state!!! The game will eventually go kaboom...
616F: 35 42       PULS  A,U
6171: 1F 89       TFR   A,B
6173: 4F          CLRA
6174: 58          ASLB
6175: 49          ROLA
6176: 10 8E 62 91 LDY   #TEXT_PTRS
617A: 10 AE AB    LDY   D,Y                   ; Y = *(D+Y)
617D: A6 A0       LDA   ,Y+                   ; Read character
617F: 27 1F       BEQ   $61A0                 ; if == 0, then end of
string
6181: 81 17       CMPA  #$17                  ; compare to #$17 (23
decimal)
6183: 24 0B       BCC   $6190                 ; >= 23 decimal? This
is a character to print, goto $6190
; if A is < 23 decimal then this means its a special instruction,
for example to change the text colour or change the position of
where to write text.
; The appropriate function is looked up in the TEXT_FUNCTIONS table
and called.
6185: 4A          DECA
6186: 48          ASLA                        ; multiply A by 2 to
give an offset to add to TEXT_FUNCTIONS below
6187: CE 61 A2    LDU   #TEXT_FUNCTIONS
618A: EE C6       LDU   A,U                   ; U = *(A+U)
618C: AD C4       JSR   ,U                    ; call function at U
618E: 20 ED       BRA   $617D                 ; get next character

6190: D6 D2       LDB   $D2                   ; read "font size"
field
6192: C1 07       CMPB  #$07                  ; do we want LARGE
font??
6194: 26 05       BNE   $619B                 ; no, goto $619B, print
string in small text
6196: BD 60 23    JSR   $6023                 ; call
BLIT_LARGE_CHARACTER routine
6199: 20 E2       BRA   $617D                 ; go read next
character/instruction
```

```
619B: BD 5F BE    JSR   $5FBE                ; call
BLIT_SMALL_CHARACTER routine
619E: 20 DD       BRA   $617D                ; go read next
character/instruction

61A0: 35 E6       PULS  A,B,Y,U,PC ;(PUL? PC=RTS)


;
; List of subroutines to call
;

TEXT_FUNCTIONS:
61A2: 61 D8                                  ; do nothing (just an
RTS)
61A4: 61 D8                                  ; do nothing
61A6: 61 D8                                  ; do nothing
61A8: 61 D9                                  ; SET_TEXT_COLOUR_FLAG
61AA: 61 DE
61AC: 62 16
61AE: 62 1D
61B0: 62 2A
61B2: 62 37
61B4: 62 3C
61B6: 62 44
61B8: 62 49
61BA: 62 4E
61BC: 62 53
61BE: 62 75
61C0: 62 58
61C2: 62 6B
61C4: 62 0F                                  ;
PRINT_AT_SPECIFIC_POSITION
61C6: 62 70
61C8: 61 CE
61CA: 62 3F
61CC: 62 5D

61CE: B6 C8 06    LDA   widget_pia_datab
61D1: 10 2B 00 A0 LBMI  $6275
61D5: EC A1       LDD   ,Y++
61D7: 39          RTS

61D8: 39          RTS

;
; Used to change text colours during PRINT_STRING_LARGE_FONT
function
;
; Y = pointer to colour byte
;
SET_TEXT_COLOUR_FLAG:
61D9: A6 A0       LDA   ,Y+
61DB: 97 CF       STA   $CF
```

```
61DD: 39          RTS



;
; Changes text position.
; X = screen blit address
; Y = pointer to 3 bytes:
; Byte 0: offset to add to X component of screen address
; Byte 1: offset to add to Y component of screen address

CHANGE_TEXT_POSITION:
61DE: 1F 10       TFR   X,D
61E0: AB A4       ADDA  ,Y
61E2: EB 21       ADDB  $0001,Y
61E4: 1F 01       TFR   D,X
61E6: 6D 22       TST   $0002,Y
61E8: 27 22       BEQ   $620C
61EA: 6D A4       TST   ,Y
61EC: 2B 0F       BMI   $61FD
61EE: 96 D0       LDA   $D0
61F0: 27 06       BEQ   $61F8
61F2: 30 89 01 00 LEAX  $0100,X
61F6: 86 FF       LDA   #$FF
61F8: 4C          INCA
61F9: 97 D0       STA   $D0
61FB: 20 0F       BRA   $620C

61FD: 96 D0       LDA   $D0
61FF: 81 01       CMPA  #$01
6201: 27 06       BEQ   $6209
6203: 30 89 FF 00 LEAX  $FF00,X
6207: 86 02       LDA   #$02
6209: 4A          DECA
620A: 97 D0       STA   $D0
620C: 31 23       LEAY  $0003,Y
620E: 39          RTS


PRINT_AT_SPECIFIC_POSITION:
620F: AE A1       LDX   ,Y++                ; set blitter
destination for character
6211: A6 A0       LDA   ,Y+                 ; value to go into $D0
6213: 97 D0       STA   $D0
6215: 39          RTS

6216: D6 D0       LDB   $D0
6218: D7 D5       STB   $D5
621A: 9F D3       STX   $D3
621C: 39          RTS

621D: 96 D2       LDA   $D2
621F: 81 07       CMPA  #$07
6221: 27 06       BEQ   $6229
```

```
6223: 86 07       LDA   #$07
6225: 97 D2       STA   $D2
6227: 30 1F       LEAX  $-1,X
6229: 39          RTS

622A: 96 D2       LDA   $D2
622C: 81 05       CMPA  #$05
622E: 27 F9       BEQ   $6229
6230: 86 05       LDA   #$05
6232: 97 D2       STA   $D2
6234: 30 01       LEAX  $0001,X
6236: 39          RTS

6237: 86 01       LDA   #$01
6239: 97 D1       STA   $D1
623B: 39          RTS

623C: 0F D1       CLR   $D1
623E: 39          RTS

623F: 86 02       LDA   #$02
6241: 97 D1       STA   $D1
6243: 39          RTS

6244: EC A1       LDD   ,Y++
6246: 7E 60 F3    JMP   $60F3

6249: EC B1       LDD   [,Y++]
624B: 7E 60 F3    JMP   $60F3

624E: A6 A0       LDA   ,Y+
6250: 7E 60 B9    JMP   $60B9

6253: A6 B1       LDA   [,Y++]
6255: 7E 60 B9    JMP   $60B9

6258: A6 63       LDA   $0003,S
625A: 7E 60 B9    JMP   $60B9

625D: A6 63       LDA   $0003,S
625F: 81 05       CMPA  #$05
6261: 23 05       BLS   $6268
6263: 81 FB       CMPA  #$FB
6265: 24 01       BCC   $6268
6267: 4F          CLRA
6268: 97 D7       STA   $D7
626A: 39          RTS

626B: EC 64       LDD   $0004,S
626D: 7E 60 F3    JMP   $60F3

6270: A6 B1       LDA   [,Y++]
6272: 97 CF       STA   $CF
6274: 39          RTS
```

```
6275: 32 78       LEAS  $FFF8,S
6277: E6 6B       LDB   $000B,S
6279: E7 61       STB   $0001,S
627B: EC 6C       LDD   $000C,S
627D: ED 62       STD   $0002,S
627F: CC 62 8E    LDD   #$628E
6282: ED 66       STD   $0006,S
6284: EC A1       LDD   ,Y++
6286: 10 AF 64    STY   $0004,S
6289: 1F 02       TFR   D,Y
628B: 7E 61 7D    JMP   $617D

628E: 1F 32       TFR   U,Y
6290: 39          RTS

TEXT_PTRS:
6291: 6D 1B                                   ;
6293: 6C D1                                   ; OPERATIONAL
6295: 6D 58                                   ; RAM
6297: 6D 4D                                   ; ROM ERROR
6299: 6B B4                                   ; ALL ROMS OK
629B: 6B FC                                   ; NO
629D: 6C 08                                   ; NO CMOS
629F: 6C 25                                   ; CMOS RAM
62A1: 6C 2F                                   ; OR WRITE PROTECT
FAILURE
62A3: 6C 8A
62A5: 6D 37
62A7: 6D 42
62A9: 6B A2
62AB: 6A D0
62AD: 6A D8
62AF: 6A E0
62B1: 6A EB                                   ; #$10
62B3: 6A FC
62B5: 6B 06
62B7: 6B 12
62B9: 6B 1E
62BB: 6B 29
62BD: 6B 36
62BF: 6B 43
62C1: 6B 51
62C3: 6B 56
62C5: 6B 65
62C7: 6B 70
62C9: 6B 7D
62CB: 6B 8A
62CD: 6A 9B
62CF: 69 62
62D1: 6A 5C                                   ;#$20
62D3: 6A 6C
62D5: 6A 7E
62D7: 69 9E
```

```
62D9: 69 B3
62DB: 69 CC
62DD: 69 E9
62DF: 69 FC
62E1: 6D E4
62E3: 69 55
62E5: 69 58
62E7: 69 5B
62E9: 67 90
62EB: 67 96
62ED: 68 14
62EF: 68 24
62F1: 68 32                                     ;#$30
62F3: 68 44
62F5: 68 4C
62F7: 68 56
62F9: 68 6B
62FB: 68 75
62FD: 68 85
62FF: 68 B6
6301: 68 C9
6303: 68 DC
6305: 68 FF
6307: 69 0A
6309: 69 13
630B: 69 2A
630D: 69 35
630F: 69 4E
6311: 65 4C                                     ;#$40
6313: 65 5B
6315: 65 9F
6317: 65 A2
6319: 65 B9
631B: 65 BF
631D: 65 D5
631F: 66 14
6321: 66 18
6323: 66 1B
6325: 66 2C
6327: 6D F2
6329: 66 33
632B: 66 3A
632D: 65 6E
632F: 66 EB
6331: 67 49                                     ;#$50
6333: 65 F0
6335: 66 51
6337: 66 73
6339: 66 85
633B: 66 9C
633D: 66 AC
633F: 66 C1
6341: 66 D0
6343: 66 E1
```

```
6345: 66 41
6347: 69 6E
6349: 64 28
634B: 69 7F
634D: 64 A6
634F: 64 FD
6351: 65 29                                     ;#$60
6353: 65 3C
6355: 65 42
6357: 65 45
6359: 64 35
635B: 63 CF
635D: 65 82
635F: 6E 01
6361: 6E 0B                                     ; print wave number in
register B
6363: 6C C4
6365: 6A 13
6367: 6A 33                                     ;
6369: 6A 37                                     ;  CREDIT
636B: 6A 58
636D: 63 95
636F: 63 CA
6371: 6D 6C                                     ;#$70
6373: 6D B7
6375: 6E 1F
6377: 6E 28                                     ; MOMMY
6379: 6E 3B                                     ; DADDY
637B: 6E 41                                     ; MIKEY
637D: 6E 2E                                     ; GRUNT- 100
637F: 6E 47                                     ; INDESTRUCTIBLE HULK
6381: 6E 5B                                     ; SPHEREOID - 1000
6383: 6E 7D                                     ; ENFORCER - 150
6385: 6E 99                                     ; BRAIN - 500
6387: 66 66                                     ; FIFTY
6389: 6E BD                                     ; PROG - 100
638B: 68 05                                     ;
638D: 63 BB                                     ; ROBOTRON HERO
638F: 6D AD
6391: 6E C8                                     ;#$80
6393: 6E DD

6935 - 6EFF : reserved for text.


6F00: 7E 70 FD    JMP   $70FD

6F03: 7E 74 B8    JMP   $74B8

6F06: 7E 75 2C    JMP   $752C

6F09: 7E 75 3A    JMP   $753A

COPY_NIB_XYB1:
```

```
6F0C: 7E 6F 11    JMP   COPY_NIB_XYB

def_wel_msg_ptr:
6F0F: 6F 65       CLR   $0005,S
COPY_NIB_XYB:
6F11: 34 02       PSHS  A
6F13: A6 80       LDA   ,X+
6F15: 1E 12       EXG   X,Y
6F17: BD D0 AB    JSR   STA_NIB_X1
6F1A: 1E 12       EXG   X,Y
6F1C: 5A          DECB
6F1D: 26 F4       BNE   $6F13
6F1F: 35 82       PULS  A,PC ;(PUL? PC=RTS)

CLEAR_CMOS:
6F21: 8E CC 00    LDX   #$CC00
6F24: 6F 80       CLR   ,X+
6F26: 8C D0 00    CMPX  #$D000
6F29: 26 F9       BNE   $6F24
6F2B: 39          RTS

LOAD_CMOS_DEFS1:
6F2C: 34 36       PSHS  Y,X,B,A
6F2E: 8E 6F 53    LDX   #$6F53
6F31: 10 8E CC 00 LDY   #$CC00
6F35: C6 12       LDB   #$12
6F37: 8D D8       BSR   COPY_NIB_XYB
6F39: 35 B6       PULS  A,B,X,Y,PC ;(PUL? PC=RTS)

LOAD_CMOS_DEFS2:
6F3B: 34 36       PSHS  Y,X,B,A
6F3D: 8E 6F 65    LDX   #def_wel_msg
6F40: 10 8E CC 24 LDY   #$CC24
6F44: C6 34       LDB   #$34
6F46: 8D C9       BSR   COPY_NIB_XYB
6F48: BD 74 91    JSR   $7491
6F4B: 8E CC 8E    LDX   #$CC8E
6F4E: BD D0 AB    JSR   STA_NIB_X1
6F51: 35 B6       PULS  A,B,X,Y,PC ;(PUL? PC=RTS)


*** default CMOS settings
6F53: 25 03 03 01 04 01 01 00 00 01 03 03 00 00 00 00
6F63: 00 00 20 20

6F67:    PRESENTED BY        WILLIAMS
6F87: ELECTRONICS INC=/(

6F99: 01 04 01 01 00 00 01 04 01 02 04 00 06 00 01 01
6FA9: 00 00 01 04 01 01 00 00 01 16 06 02 00 00 01 04
6FB9: 01 02 00 00 01 00 04 01 00 00 01 00 02 01 00 00
6FC9: 01 00 02 02 00 00 01 04 01 01 00 00 00 50 70 41
6FD9: 00 1E 01 20 70 4C 00 27 00 09 70 5A 00 32 00 99
6FE9: 70 57 01 3B 00 99 70 57 01 44 00 99 70 57 01 4D
```

```
6FF9: 01 99 70 57 01 56 00 99 70 57 01 5F 00 99 70 57
7009: 01 68 00 01 70 6F 00 73 00 10 70 79 00 7C 03 20
7019: 70 84 00 85 00 01 70 74 00 90 00 01 70 74 00 99
7029: 00 01 70 74 00 A2 00 01 70 74 00 AB 00 01 70 74
7039: 00 B4 00 01 70 74 00 BD 00 5A 20 40 25 41 30 4E
7049: 50 66 FF 00 42 02 43 03 44 04 45 05 42 FF 00 42
7059: FF 00 46 01 52 02 53 03 54 04 55 05 56 06 57 07
7069: 58 08 59 09 7B FF 00 48 01 47 FF 00 48 01 51 FF
7079: 00 49 03 4A 05 44 06 4C 08 4D FF 00 44 04 42 FF

7089: 34 22       PSHS  Y,A
708B: 86 11       LDA   #$11
708D: 6D 44       TST   $0004,U
708F: 27 02       BEQ   $7093
7091: 86 16       LDA   #$16
7093: E6 45       LDB   $0005,U
7095: 1F 01       TFR   D,X
7097: A6 E4       LDA   ,S
7099: BD 5F 96    JSR   $5F96               ; JMP $613F: print
string in small font
709C: 8D 02       BSR   $70A0
709E: 35 A2       PULS  A,Y,PC ;(PUL? PC=RTS)

70A0: 34 36       PSHS  Y,X,B,A
70A2: 1E 12       EXG   X,Y
70A4: BD D0 A5    JSR   $D0A5
70A7: C1 FF       CMPB  #$FF
70A9: 26 02       BNE   $70AD
70AB: C6 EE       LDB   #$EE
70AD: 1E 12       EXG   X,Y
70AF: 1E 10       EXG   X,D
70B1: 86 59       LDA   #$59
70B3: E6 45       LDB   $0005,U
70B5: 1E 10       EXG   X,D
70B7: 34 06       PSHS  B,A
70B9: CC 3C 05    LDD   #$3C05              ; width = 3C, height =
05
70BC: BD D0 1B    JSR   $D01B               ; JMP $DADF - clear
rectangle to black
70BF: 35 06       PULS  A,B
70C1: 8D 05       BSR   $70C8
70C3: BD 5F 96    JSR   $5F96               ; JMP $613F: print
string in small font
70C6: 35 B6       PULS  A,B,X,Y,PC ;(PUL? PC=RTS)

70C8: 34 20       PSHS  Y
70CA: 10 AE 42    LDY   $0002,U
70CD: E1 A1       CMPB  ,Y++
70CF: 24 FC       BCC   $70CD
70D1: A6 3D       LDA   $FFFD,Y
70D3: 35 A0       PULS  Y,PC ;(PUL? PC=RTS)

70D5: BD D0 12    JSR   CLR_SCREEN1
70D8: 86 66       LDA   #$66
```

```
70DA: 97 CF       STA   $CF
70DC: CE 6F D5    LDU   #$6FD5
70DF: 10 8E CC 00 LDY   #$CC00
70E3: 86 2E       LDA   #$2E
70E5: 8E CC 24    LDX   #$CC24
70E8: 34 10       PSHS  X
70EA: 8D 9D       BSR   $7089
70EC: 33 46       LEAU  $0006,U
70EE: 31 22       LEAY  $0002,Y
70F0: 4C          INCA
70F1: 10 AC E4    CMPY  ,S
70F4: 26 F4       BNE   $70EA
70F6: 86 2D       LDA   #$2D
70F8: BD 5F 96    JSR   $5F96               ; JMP $613F: print
string in small font
70FB: 35 90       PULS  X,PC ;(PUL? PC=RTS)

70FD: BD 74 AB    JSR   $74AB
7100: 27 03       BEQ   $7105
7102: BD 6F 2C    JSR   LOAD_CMOS_DEFS1
7105: 8E CC 18    LDX   #$CC18
7108: 6F 80       CLR   ,X+
710A: 8C CC 24    CMPX  #$CC24
710D: 25 F9       BCS   $7108
710F: BD 74 78    JSR   $7478
7112: 8D C1       BSR   $70D5
7114: CE 6F D5    LDU   #$6FD5
7117: 10 8E CC 00 LDY   #$CC00
711B: 86 2E       LDA   #$2E
711D: BD 74 33    JSR   $7433
7120: BD 71 FE    JSR   $71FE
7123: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
7126: 72 9F          ; pointer to function
7128: BD 72 1D    JSR   $721D
712B: 8D 08       BSR   $7135
712D: 86 01       LDA   #$01
712F: 8E 71 28    LDX   #$7128
7132: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7135: 35 06       PULS  A,B
7137: DE 15       LDU   $15
7139: ED 4D       STD   $000D,U
713B: 86 20       LDA   #$20
713D: A7 47       STA   $0007,U
713F: B6 C8 04    LDA   widget_pia_dataa
7142: 2B 22       BMI   $7166                ; Fire down (bit 7) is
set
7144: 85 40       BITA  #$40                ; Fire up?
7146: 26 03       BNE   $714B
7148: 6E D8 0D    JMP   [$0D,U]

714B: 8D 32       BSR   $717F
```

```
714D: 86 01       LDA   #$01
714F: 8E 71 55    LDX   #$7155
7152: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7155: B6 C8 04    LDA   widget_pia_dataa
7158: 85 40       BITA  #$40
715A: 27 EC       BEQ   $7148
715C: 6A 47       DEC   $0007,U
715E: 26 ED       BNE   $714D
7160: 86 05       LDA   #$05
7162: A7 47       STA   $0007,U
7164: 20 E5       BRA   $714B

7166: 8D 3D       BSR   $71A5
7168: 86 01       LDA   #$01
716A: 8E 71 70    LDX   #$7170
716D: 7E D0 66    JMP   $D066

7170: B6 C8 04    LDA   widget_pia_dataa
7173: 2A D3       BPL   $7148
7175: 6A 47       DEC   $0007,U
7177: 26 EF       BNE   $7168
7179: 86 05       LDA   #$05
717B: A7 47       STA   $0007,U
717D: 20 E7       BRA   $7166

717F: BD 74 3E    JSR   $743E
7182: 1F 21       TFR   Y,X
7184: BD D0 A2    JSR   LDA_NIB_X1
7187: 10 8C CC 00 CMPY  #$CC00
718B: 27 32       BEQ   $71BF
718D: 8B 01       ADDA  #$01
718F: 19          DAA
7190: 25 12       BCS   $71A4
7192: A1 41       CMPA  $0001,U
7194: 22 0E       BHI   $71A4
7196: 1F 21       TFR   Y,X
7198: BD D0 AB    JSR   STA_NIB_X1
719B: BD 74 78    JSR   $7478
719E: BD 70 A0    JSR   $70A0
71A1: BD 74 49    JSR   $7449
71A4: 39          RTS

71A5: BD 74 3E    JSR   $743E
71A8: 1F 21       TFR   Y,X
71AA: BD D0 A2    JSR   LDA_NIB_X1
71AD: 4D          TSTA
71AE: 27 F4       BEQ   $71A4
71B0: 10 8C CC 00 CMPY  #$CC00
71B4: 27 0F       BEQ   $71C5
71B6: 8B 99       ADDA  #$99
71B8: 19          DAA
71B9: A1 C4       CMPA  ,U
```

```
71BB: 25 E7       BCS   $71A4
71BD: 20 D7       BRA   $7196

71BF: 8D 0A       BSR   $71CB
71C1: A6 01       LDA   $0001,X
71C3: 20 D1       BRA   $7196

71C5: 8D 04       BSR   $71CB
71C7: A6 1F       LDA   $FFFF,X
71C9: 20 CB       BRA   $7196

71CB: 8E 71 DB    LDX   #$71DB
71CE: A1 84       CMPA  ,X
71D0: 23 07       BLS   $71D9
71D2: 30 01       LEAX  $0001,X
71D4: 8C 71 DF    CMPX  #$71DF
71D7: 25 F5       BCS   $71CE
71D9: 39          RTS

71DA: 00 00       NEG   $00
71DC: 20 25       BRA   $7203

71DE: 30 50       LEAX  $FFF0,U
71E0: 50          NEGB
71E1: 34 12       PSHS  X,A
71E3: A6 44       LDA   $0004,U
71E5: 27 13       BEQ   $71FA
71E7: 8E CC 04    LDX   #$CC04
71EA: 4A          DECA
71EB: 27 03       BEQ   $71F0
71ED: 8E CC 14    LDX   #$CC14
71F0: BD D0 A2    JSR   LDA_NIB_X1
71F3: 4D          TSTA
71F4: 27 04       BEQ   $71FA
71F6: 1A 01       ORCC  #$01
71F8: 35 92       PULS  A,X,PC ;(PUL? PC=RTS)

71FA: 1C FE       ANDCC #$FE                ; clear carry flag
71FC: 35 92       PULS  A,X,PC ;(PUL? PC=RTS)

71FE: 34 16       PSHS  X,B,A
7200: E6 45       LDB   $0005,U
7202: 86 0C       LDA   #$0C
7204: 1F 01       TFR   D,X
7206: 86 2C       LDA   #$2C
7208: BD 5F 96    JSR   $5F96               ; JMP $613F: print
string in small font
720B: 35 96       PULS  A,B,X,PC ;(PUL? PC=RTS)

720D: 34 16       PSHS  X,B,A
720F: E6 45       LDB   $0005,U
7211: 86 0C       LDA   #$0C
7213: 1F 01       TFR   D,X
7215: CC 03 05    LDD   #$0305              ; width 3, height 5
```

```
7218: BD D0 1B    JSR   $D01B               ; JMP $DADF - clear
rectangle to black
721B: 35 96       PULS  A,B,X,PC ;(PUL? PC=RTS)

721D: 35 06       PULS  A,B
721F: DE 15       LDU   $15
7221: ED 4D       STD   $000D,U
7223: 86 30       LDA   #$30
7225: A7 47       STA   $0007,U
7227: B6 C8 04    LDA   widget_pia_dataa
722A: 46          RORA
722B: 25 21       BCS   $724E
722D: 46          RORA
722E: 25 03       BCS   $7233
7230: 6E D8 0D    JMP   [$0D,U]

7233: 8D 33       BSR   $7268
7235: 86 01       LDA   #$01
7237: 8E 72 3D    LDX   #$723D
723A: 7E D0 66    JMP   $D066                   ; JMP $D1E3 -
allocate function call

723D: B6 C8 04    LDA   widget_pia_dataa
7240: 85 02       BITA  #$02
7242: 27 EC       BEQ   $7230
7244: 6A 47       DEC   $0007,U
7246: 26 ED       BNE   $7235
7248: 86 08       LDA   #$08
724A: A7 47       STA   $0007,U
724C: 20 E5       BRA   $7233

724E: 8D 34       BSR   $7284
7250: 86 01       LDA   #$01
7252: 8E 72 58    LDX   #$7258
7255: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7258: B6 C8 04    LDA   widget_pia_dataa
725B: 46          RORA
725C: 24 D2       BCC   $7230
725E: 6A 47       DEC   $0007,U
7260: 26 EE       BNE   $7250
7262: 86 08       LDA   #$08
7264: A7 47       STA   $0007,U
7266: 20 E6       BRA   $724E

7268: BD 74 3E    JSR   $743E
726B: 8D A0       BSR   $720D
726D: 10 8C CC 22 CMPY  #$CC22
7271: 27 0D       BEQ   $7280
7273: 31 22       LEAY  $0002,Y
7275: 4C          INCA
7276: 33 46       LEAU  $0006,U
7278: BD 71 E1    JSR   $71E1
```

```
727B: 25 F0       BCS   $726D
727D: BD 74 33    JSR   $7433
7280: BD 71 FE    JSR   $71FE
7283: 39          RTS

7284: BD 74 3E    JSR   $743E
7287: 8D 84       BSR   $720D
7289: 10 8C CC 00 CMPY  #$CC00
728D: 27 F1       BEQ   $7280
728F: 31 3E       LEAY  $FFFE,Y
7291: 4A          DECA
7292: 33 5A       LEAU  $FFFA,U
7294: BD 71 E1    JSR   $71E1
7297: 25 F0       BCS   $7289
7299: BD 74 33    JSR   $7433
729C: 7E 71 FE    JMP   $71FE

729F: 86 01       LDA   #$01
72A1: 8E 72 A7    LDX   #$72A7
72A4: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

72A7: B6 C8 0C    LDA   rom_pia_dataa
72AA: 85 02       BITA  #$02
72AC: 27 F1       BEQ   $729F
72AE: BD D0 60    JSR   $D060               ; JMP $D1FF
72B1: BD D0 12    JSR   CLR_SCREEN1
72B4: B6 C8 0C    LDA   rom_pia_dataa
72B7: 85 02       BITA  #$02
72B9: 26 F9       BNE   $72B4
72BB: B6 CC 1B    LDA   $CC1B
72BE: 84 0F       ANDA  #$0F
72C0: 27 14       BEQ   $72D6
72C2: 7F CC 1B    CLR   $CC1B
72C5: BD 74 78    JSR   $7478
72C8: BD D0 12    JSR   CLR_SCREEN1
72CB: BD 75 15    JSR   $7515
72CE: 86 40       LDA   #$40
72D0: 8E 72 D6    LDX   #$72D6
72D3: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

72D6: B6 CC 1D    LDA   $CC1D
72D9: 84 0F       ANDA  #$0F
72DB: 27 11       BEQ   $72EE
72DD: 7F CC 1D    CLR   $CC1D
72E0: BD 74 78    JSR   $7478
72E3: BD E3 D9    JSR   $E3D9
72E6: 86 40       LDA   #$40
72E8: 8E 72 EE    LDX   #$72EE
72EB: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

72EE: B6 CC 21    LDA   $CC21
```

```
72F1: 84 0F       ANDA  #$0F
72F3: 27 5D       BEQ   $7352
72F5: 7F CC 21    CLR   $CC21
72F8: BD 74 78    JSR   $7478
72FB: 86 3A       LDA   #$3A
72FD: 8E CC 24    LDX   #$CC24
7300: C6 32       LDB   #$32
7302: BD D0 AB    JSR   STA_NIB_X1
7305: 5A          DECB
7306: 26 FA       BNE   $7302
7308: BD D0 12    JSR   CLR_SCREEN1
730B: 86 5C       LDA   #$5C
730D: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
7310: 10 8E CC 24 LDY   #$CC24
7314: 8E 25 30    LDX   #$2530
7317: CC 19 80    LDD   #$1980
731A: BD 75 3A    JSR   $753A
731D: C6 30       LDB   #$30
731F: 8E CC 88    LDX   #$CC88
7322: 10 8E CC 24 LDY   #$CC24
7326: BD 73 8A    JSR   $738A
7329: 86 5C       LDA   #$5C
732B: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
732E: BD 74 0D    JSR   $740D
7331: 10 8E CC 56 LDY   #$CC56
7335: 8E 25 40    LDX   #$2540
7338: CC 19 80    LDD   #$1980
733B: BD 75 3A    JSR   $753A
733E: C6 40       LDB   #$40
7340: 8E CC 8A    LDX   #$CC8A
7343: 10 8E CC 56 LDY   #$CC56
7347: 8D 41       BSR   $738A
7349: BD 74 91    JSR   $7491
734C: 8E CC 8E    LDX   #$CC8E
734F: BD D0 AB    JSR   STA_NIB_X1
7352: B6 CC 23    LDA   $CC23
7355: 84 0F       ANDA  #$0F
7357: 27 09       BEQ   $7362
7359: 7F CC 23    CLR   $CC23
735C: BD 74 78    JSR   $7478
735F: BD E3 D6    JSR   $E3D6
7362: B6 CC 1F    LDA   $CC1F
7365: 84 0F       ANDA  #$0F
7367: 27 0B       BEQ   $7374
7369: 7F CC 1F    CLR   $CC1F
736C: BD 74 78    JSR   $7478
736F: 8D 08       BSR   $7379
7371: 7E F0 06    JMP   $F006

7374: 8D 03       BSR   $7379
7376: 7E D0 00    JMP   $D000

7379: B6 CC 19    LDA   $CC19
737C: 84 0F       ANDA  #$0F
```

```
737E: 27 09       BEQ   $7389
7380: 7C CC 8C    INC   $CC8C
7383: 7C CC 8C    INC   $CC8C
7386: 7F CC 19    CLR   $CC19
7389: 39          RTS

738A: DE 15       LDU   $15
738C: 86 25       LDA   #$25
738E: ED 47       STD   $0007,U
7390: 35 06       PULS  A,B
7392: ED 4D       STD   $000D,U
7394: AF 4B       STX   $000B,U
7396: 10 AF 49    STY   $0009,U
7399: 86 25       LDA   #$25
739B: BD D0 AB    JSR   STA_NIB_X1
739E: 86 65       LDA   #$65
73A0: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT  ;print CENTER
THE LINE
73A3: 86 04       LDA   #$04
73A5: 8E 73 AB    LDX   #$73AB
73A8: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

73AB: B6 C8 0C    LDA   rom_pia_dataa
73AE: 85 02       BITA  #$02
73B0: 26 F1       BNE   $73A3
73B2: B6 C8 06    LDA   widget_pia_datab
73B5: 85 02       BITA  #$02
73B7: 27 10       BEQ   $73C9
73B9: AE 4B       LDX   $000B,U
73BB: BD D0 A2    JSR   LDA_NIB_X1
73BE: AE 4B       LDX   $000B,U
73C0: 4C          INCA
73C1: 81 3A       CMPA  #$3A
73C3: 23 15       BLS   $73DA
73C5: 86 3A       LDA   #$3A
73C7: 20 11       BRA   $73DA

73C9: 46          RORA
73CA: 24 1D       BCC   $73E9
73CC: AE 4B       LDX   $000B,U
73CE: BD D0 A2    JSR   LDA_NIB_X1
73D1: AE 4B       LDX   $000B,U
73D3: 4A          DECA
73D4: 81 13       CMPA  #$13
73D6: 24 02       BCC   $73DA
73D8: 86 13       LDA   #$13
73DA: BD D0 AB    JSR   STA_NIB_X1
73DD: A7 47       STA   $0007,U
73DF: 8D 2C       BSR   $740D
73E1: 86 10       LDA   #$10
73E3: 8E 73 B2    LDX   #$73B2
73E6: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call
```

```
73E9: B6 C8 0C    LDA   rom_pia_dataa
73EC: 85 02       BITA  #$02
73EE: 26 08       BNE   $73F8
73F0: 86 04       LDA   #$04
73F2: 8E 73 B2    LDX   #$73B2
73F5: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call

73F8: BD D0 12    JSR   CLR_SCREEN1
73FB: 86 04       LDA   #$04
73FD: 8E 74 03    LDX   #$7403
7400: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call

7403: B6 C8 0C    LDA   rom_pia_dataa
7406: 85 02       BITA  #$02
7408: 26 F1       BNE   $73FB
740A: 6E D8 0D    JMP   [$0D,U]

740D: 0F D0       CLR   $D0
740F: AE 47       LDX   $0007,U           ; get object pointer
7411: 30 89 FF 00 LEAX  $FF00,X           ; blitter destination
7415: 86 5A       LDA   #$5A              ; width
7417: C6 09       LDB   #$09              ; height
7419: BD D0 1B    JSR   $D01B             ; JMP $DADF - clear
rectangle to black
741C: AE 47       LDX   $0007,U           ; get object pointer
741E: 0F D0       CLR   $D0
7420: 10 AE 49    LDY   $0009,U
7423: C6 19       LDB   #$19
7425: 1E 12       EXG   X,Y
7427: BD D0 A2    JSR   LDA_NIB_X1
742A: 1E 12       EXG   X,Y
742C: BD 5F 93    JSR   $5F93
742F: 5A          DECB
7430: 26 F3       BNE   $7425
7432: 39          RTS

7433: 10 BF B3 EC STY   $B3EC
7437: FF B3 EF    STU   $B3EF
743A: B7 B3 EE    STA   $B3EE
743D: 39          RTS

743E: 10 BE B3 EC LDY   $B3EC
7442: FE B3 EF    LDU   $B3EF
7445: B6 B3 EE    LDA   $B3EE
7448: 39          RTS

7449: 10 8C CC 04 CMPY  #$CC04
744D: 27 01       BEQ   $7450
744F: 39          RTS

7450: 1F 21       TFR   Y,X
```

```
7452: BD D0 A2    JSR   LDA_NIB_X1
7455: 48          ASLA
7456: 34 02       PSHS  A
7458: 48          ASLA
7459: AB E0       ADDA  ,S+
745B: 8E 6F 99    LDX   #$6F99
745E: 30 86       LEAX  A,X
7460: 33 46       LEAU  $0006,U
7462: 31 22       LEAY  $0002,Y
7464: A6 80       LDA   ,X+
7466: 34 10       PSHS  X
7468: 1F 21       TFR   Y,X
746A: BD D0 AB    JSR   STA_NIB_X1
746D: 35 10       PULS  X
746F: BD 70 A0    JSR   $70A0
7472: 10 8C CC 10 CMPY  #$CC10
7476: 25 E8       BCS   $7460
7478: 34 12       PSHS  X,A
747A: 8D 08       BSR   $7484
747C: 8E CC 8C    LDX   #$CC8C
747F: BD D0 AB    JSR   STA_NIB_X1
7482: 35 92       PULS  A,X,PC ;(PUL? PC=RTS)

7484: 34 34       PSHS  Y,X,B
7486: 8E CC 00    LDX   #$CC00
7489: 10 8E CC 24 LDY   #$CC24
748D: 8D 09       BSR   $7498
748F: 35 B4       PULS  B,X,Y,PC ;(PUL? PC=RTS)

7491: 8E CC 24    LDX   #$CC24
7494: 10 8E CC 8C LDY   #$CC8C
7498: 10 9F 2B    STY   $2B
749B: 4F          CLRA
749C: E6 80       LDB   ,X+
749E: C4 0F       ANDB  #$0F
74A0: 34 04       PSHS  B
74A2: AB E0       ADDA  ,S+
74A4: 9C 2B       CPX   $2B
74A6: 26 F4       BNE   $749C
74A8: 8B 37       ADDA  #$37
74AA: 39          RTS

74AB: 8D D7       BSR   $7484
74AD: 34 02       PSHS  A
74AF: 8E CC 8C    LDX   #$CC8C
74B2: BD D0 A2    JSR   LDA_NIB_X1
74B5: A1 E0       CMPA  ,S+
74B7: 39          RTS

74B8: 8D 6B       BSR   $7525
74BA: 8D EF       BSR   $74AB
74BC: 27 3A       BEQ   $74F8
74BE: 86 39       LDA   #$39
74C0: B7 CB FF    STA   watchdog
```

```
74C3: BD 6F 2C    JSR   LOAD_CMOS_DEFS1
74C6: 86 39       LDA   #$39
74C8: B7 CB FF    STA   watchdog
74CB: 8D AB       BSR   $7478
74CD: 86 39       LDA   #$39
74CF: B7 CB FF    STA   watchdog
74D2: BD D0 12    JSR   CLR_SCREEN1
74D5: 86 39       LDA   #$39
74D7: B7 CB FF    STA   watchdog
74DA: 8D 23       BSR   $74FF
74DC: BD E3 DC    JSR   $E3DC
74DF: BD E3 D0    JSR   CHECK_CMOS1
74E2: 8D C7       BSR   $74AB
74E4: 27 15       BEQ   $74FB
74E6: 86 4F       LDA   #$4F
74E8: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
74EB: 86 39       LDA   #$39
74ED: B7 CB FF    STA   watchdog
74F0: B6 C8 0C    LDA   rom_pia_dataa
74F3: 85 02       BITA  #$02
74F5: 27 F4       BEQ   $74EB
74F7: 39          RTS

74F8: 7E E3 D0    JMP   CHECK_CMOS1

74FB: 86 50       LDA   #$50
74FD: 20 E9       BRA   $74E8

74FF: 8E CD 02    LDX   #$CD02
7502: C6 04       LDB   #$04
7504: A6 80       LDA   ,X+
7506: 84 0F       ANDA  #$0F
7508: 81 09       CMPA  #$09
750A: 23 03       BLS   $750F
750C: 5A          DECB
750D: 27 06       BEQ   $7515
750F: 8C CD 32    CMPX  #top_score
7512: 26 F0       BNE   $7504
7514: 39          RTS

7515: 86 5B       LDA   #$5B
7517: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
751A: 8E CD 02    LDX   #$CD02
751D: 6F 80       CLR   ,X+
751F: 8C CD 32    CMPX  #top_score
7522: 26 F9       BNE   $751D
7524: 39          RTS

7525: 8D 05       BSR   $752C
7527: 27 FB       BEQ   $7524
7529: 7E 6F 3B    JMP   LOAD_CMOS_DEFS2

752C: BD 74 91    JSR   $7491
752F: 34 02       PSHS  A
```

```
7531: 8E CC 8E    LDX   #$CC8E
7534: BD D0 A2    JSR   LDA_NIB_X1
7537: A1 E0       CMPA  ,S+
7539: 39          RTS

753A: DE 15       LDU   $15
753C: ED 47       STD   $0007,U
753E: AF 49       STX   $0009,U
7540: 10 AF 4B    STY   $000B,U
7543: 35 06       PULS  A,B
7545: ED 4D       STD   $000D,U
7547: 86 04       LDA   #$04
7549: 8E 75 4F    LDX   #$754F
754C: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

754F: 8D 34       BSR   $7585
7551: 26 F4       BNE   $7547
7553: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
7556: 77 49       ; pointer to function
7558: EC 4D       LDD   $000D,U
755A: ED 0D       STD   $000D,X
755C: EC 47       LDD   $0007,U
755E: ED 07       STD   $0007,X
7560: EF 09       STU   $0009,X
7562: 9F D8       STX   $D8
7564: 86 99       LDA   #$99
7566: 97 CF       STA   $CF
7568: 0F D0       CLR   $D0
756A: 0F DA       CLR   $DA
756C: 0F DB       CLR   $DB
756E: 0F DC       CLR   $DC
7570: 8D 08       BSR   $757A
7572: 86 02       LDA   #$02
7574: 8E 75 8B    LDX   #$758B
7577: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

757A: BD 76 2A    JSR   $762A
757D: BD 76 36    JSR   $7636
7580: 86 3A       LDA   #$3A
7582: 7E 76 5D    JMP   $765D

7585: B6 C8 04    LDA   widget_pia_dataa
7588: 85 40       BITA  #$40
758A: 39          RTS

758B: B6 C8 04    LDA   widget_pia_dataa
758E: 46          RORA
758F: 10 25 01 16 LBCS  $76A9
7593: 46          RORA
7594: 10 25 00 F3 LBCS  $768B
7598: 8D EB       BSR   $7585
```

```
759A: 27 D6       BEQ   $7572
759C: A6 48       LDA   $0008,U
759E: 84 80       ANDA  #$80
75A0: 8B 20       ADDA  #$20
75A2: A7 48       STA   $0008,U
75A4: BD 76 48    JSR   $7648
75A7: 81 5E       CMPA  #$5E
75A9: 27 4D       BEQ   $75F8
75AB: 96 D0       LDA   $D0
75AD: 97 DC       STA   $DC
75AF: BD 76 45    JSR   $7645
75B2: BD 76 48    JSR   $7648
75B5: AE 49       LDX   $0009,U
75B7: 9F DA       STX   $DA
75B9: BD 5F 93    JSR   $5F93
75BC: AF 49       STX   $0009,U
75BE: BD 76 72    JSR   $7672
75C1: 6A 47       DEC   $0007,U
75C3: 27 2B       BEQ   $75F0
75C5: BD 75 7A    JSR   $757A
75C8: 86 02       LDA   #$02
75CA: 8E 75 D0    LDX   #$75D0
75CD: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

75D0: 8D B3       BSR   $7585
75D2: 27 0F       BEQ   $75E3
75D4: A6 47       LDA   $0007,U
75D6: 4A          DECA
75D7: 27 EF       BEQ   $75C8
75D9: 6A 48       DEC   $0008,U
75DB: 27 06       BEQ   $75E3
75DD: A6 48       LDA   $0008,U
75DF: 81 80       CMPA  #$80
75E1: 26 E5       BNE   $75C8
75E3: BD 75 85    JSR   $7585
75E6: 27 8A       BEQ   $7572
75E8: A6 48       LDA   $0008,U
75EA: 84 80       ANDA  #$80
75EC: 8B 04       ADDA  #$04
75EE: 20 B2       BRA   $75A2

75F0: 9E D8       LDX   $D8
75F2: BD D0 5D    JSR   $D05D               ; JMP $D218 - deallocate
object metadata entry
75F5: 6E D8 0D    JMP   [$0D,U]

75F8: 8D 4B       BSR   $7645
75FA: 8D 2E       BSR   $762A
75FC: 86 3A       LDA   #$3A
75FE: 8D 5D       BSR   $765D
7600: 9E DA       LDX   $DA
7602: AF 49       STX   $0009,U
7604: D6 DC       LDB   $DC
```

```
7606: D7 D0       STB   $D0
7608: 6C 47       INC   $0007,U
760A: 10 AE 4B    LDY   $000B,U
760D: 31 3F       LEAY  $FFFF,Y
760F: 10 8C C0 00 CMPY  #color_registers
7613: 25 02       BCS   $7617
7615: 31 3F       LEAY  $FFFF,Y
7617: 10 AF 4B    STY   $000B,U
761A: 86 01       LDA   #$01
761C: 8E 76 22    LDX   #$7622
761F: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7622: BD 75 85    JSR   $7585
7625: 26 F3       BNE   $761A
7627: 7E 75 6A    JMP   $756A

762A: 34 06       PSHS  B,A
762C: AE 49       LDX   $0009,U             ; blitter destination
762E: CC 04 07    LDD   #$0407              ; width = 4, height = 7
7631: BD D0 1B    JSR   $D01B               ; JMP $DADF - clear
rectangle to black
7634: 35 86       PULS  A,B,PC ;(PUL? PC=RTS)

7636: 86 99       LDA   #$99
7638: AE 49       LDX   $0009,U
763A: A7 08       STA   $0008,X
763C: A7 89 01 08 STA   $0108,X
7640: A7 89 02 08 STA   $0208,X
7644: 39          RTS

7645: 4F          CLRA
7646: 20 F0       BRA   $7638

7648: 10 AE 4B    LDY   $000B,U
764B: 10 8C C0 00 CMPY  #color_registers
764F: 24 03       BCC   $7654
7651: A6 A4       LDA   ,Y
7653: 39          RTS

7654: 34 10       PSHS  X
7656: AE 4B       LDX   $000B,U
7658: BD D0 A2    JSR   LDA_NIB_X1
765B: 35 90       PULS  X,PC ;(PUL? PC=RTS)

765D: 10 AE 4B    LDY   $000B,U
7660: 10 8C C0 00 CMPY  #color_registers
7664: 24 03       BCC   $7669
7666: A7 A4       STA   ,Y
7668: 39          RTS

7669: 34 10       PSHS  X
766B: AE 4B       LDX   $000B,U
766D: BD D0 AB    JSR   STA_NIB_X1
```

```
7670: 35 90       PULS  X,PC ;(PUL? PC=RTS)

7672: 10 AE 4B    LDY   $000B,U
7675: 31 21       LEAY  $0001,Y
7677: 10 8C C0 00 CMPY  #color_registers
767B: 25 02       BCS   $767F
767D: 31 21       LEAY  $0001,Y
767F: 10 AF 4B    STY   $000B,U
7682: 39          RTS

7683: 8E 20 00    LDX   #$2000
7686: 30 1F       LEAX  $FFFF,X
7688: 26 FC       BNE   $7686
768A: 39          RTS

768B: 86 0A       LDA   #$0A
768D: 8D 7F       BSR   $770E
768F: 8D F2       BSR   $7683
7691: F6 C8 04    LDB   widget_pia_dataa
7694: C5 02       BITB  #$02
7696: 10 27 FE FE LBEQ  $7598
769A: 4A          DECA
769B: 26 F2       BNE   $768F
769D: 86 01       LDA   #$01
769F: 8E 76 A5    LDX   #$76A5
76A2: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

76A5: 86 01       LDA   #$01
76A7: 20 E4       BRA   $768D

76A9: 86 0A       LDA   #$0A
76AB: 8D 19       BSR   $76C6
76AD: 8D D4       BSR   $7683
76AF: F6 C8 04    LDB   widget_pia_dataa
76B2: 56          RORB
76B3: 10 24 FE E1 LBCC  $7598
76B7: 4A          DECA
76B8: 26 F3       BNE   $76AD
76BA: 86 01       LDA   #$01
76BC: 8E 76 C2    LDX   #$76C2
76BF: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

76C2: 86 01       LDA   #$01
76C4: 20 E5       BRA   $76AB

76C6: 34 02       PSHS  A
76C8: BD 76 48    JSR   $7648
76CB: 4C          INCA
76CC: 6D 48       TST   $0008,U
76CE: 2A 15       BPL   $76E5
76D0: 0D DA       TST   $DA
76D2: 27 04       BEQ   $76D8
```

```
76D4: 81 5E       CMPA  #$5E
76D6: 23 06       BLS   $76DE
76D8: 81 5D       CMPA  #$5D
76DA: 23 02       BLS   $76DE
76DC: 86 30       LDA   #$30
76DE: 81 3E       CMPA  #$3E
76E0: 26 1B       BNE   $76FD
76E2: 4C          INCA
76E3: 20 18       BRA   $76FD

76E5: 81 5A       CMPA  #$5A
76E7: 23 0E       BLS   $76F7
76E9: 0D DA       TST   $DA
76EB: 27 08       BEQ   $76F5
76ED: 81 5E       CMPA  #$5E
76EF: 22 04       BHI   $76F5
76F1: 86 5E       LDA   #$5E
76F3: 20 02       BRA   $76F7

76F5: 86 3A       LDA   #$3A
76F7: 81 3B       CMPA  #$3B
76F9: 26 02       BNE   $76FD
76FB: 86 41       LDA   #$41
76FD: BD 76 5D    JSR   $765D
7700: BD 76 2A    JSR   $762A
7703: D6 D0       LDB   $D0
7705: AE 49       LDX   $0009,U
7707: BD 5F 93    JSR   $5F93
770A: D7 D0       STB   $D0
770C: 35 82       PULS  A,PC ;(PUL? PC=RTS)

770E: 34 02       PSHS  A
7710: BD 76 48    JSR   $7648
7713: 4A          DECA
7714: 6D 48       TST   $0008,U
7716: 2A 15       BPL   $772D
7718: 81 30       CMPA  #$30
771A: 24 0A       BCC   $7726
771C: 0D DA       TST   $DA
771E: 27 04       BEQ   $7724
7720: 86 5E       LDA   #$5E
7722: 20 02       BRA   $7726

7724: 86 5D       LDA   #$5D
7726: 81 3E       CMPA  #$3E
7728: 26 1D       BNE   $7747
772A: 4A          DECA
772B: 20 1A       BRA   $7747

772D: 81 39       CMPA  #$39
772F: 26 0A       BNE   $773B
7731: 0D DA       TST   $DA
7733: 27 04       BEQ   $7739
7735: 86 5E       LDA   #$5E
```

```
7737: 20 02       BRA   $773B

7739: 86 5A       LDA   #$5A
773B: 81 40       CMPA  #$40
773D: 26 02       BNE   $7741
773F: 86 3A       LDA   #$3A
7741: 81 5D       CMPA  #$5D
7743: 26 02       BNE   $7747
7745: 86 5A       LDA   #$5A
7747: 20 B4       BRA   $76FD

7749: 6D 48       TST   $0008,U
774B: 2B 34       BMI   $7781
774D: 86 FF       LDA   #$FF
774F: 8E 77 55    LDX   #$7755
7752: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call

7755: 86 FF       LDA   #$FF
7757: 8E 77 5D    LDX   #$775D
775A: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call

775D: 86 82       LDA   #$82
775F: 8E 77 65    LDX   #$7765
7762: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call

7765: 6A 47       DEC   $0007,U
7767: 26 E4       BNE   $774D
7769: AE 49       LDX   $0009,U
776B: 33 84       LEAU  ,X
776D: BD 76 48    JSR   $7648
7770: 81 5E       CMPA  #$5E
7772: 26 05       BNE   $7779
7774: 86 3A       LDA   #$3A
7776: BD 76 5D    JSR   $765D
7779: DE 15       LDU   $15
777B: BD D0 5D    JSR   $D05D             ; JMP $D218 - deallocate
object metadata entry
777E: 6E D8 0D    JMP   [$0D,U]

7781: B6 C8 0C    LDA   rom_pia_dataa
7784: 85 02       BITA  #$02
7786: 26 E1       BNE   $7769
7788: 86 01       LDA   #$01
778A: 8E 77 81    LDX   #$7781
778D: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call

7790: FF FF FF    STU   $FFFF
7793: FF FF FF    STU   $FFFF
7796: FF FF FF    STU   $FFFF
7799: FF FF FF    STU   $FFFF
```

```
779C: FF FF FF    STU   $FFFF
779F: FF 7E 77    STU   $7E77
77A2: A5 79       BITA  $FFF9,S
77A4: 2D BD       BLT   $7763
77A6: D0 60       SUBB  $60
77A8: 0F F4       CLR   $F4
77AA: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
77AD: 77 B9       ; pointer to function
77AF: 8E 77 E4    LDX   #$77E4
77B2: 4F          CLRA
77B3: BD D0 5A    JSR   $D05A               ; JMP $D243 - reserve
object metadata entry in list @ $981D and call function in X
77B6: 7E D0 63    JMP   $D063               ; JMP $D1F3

77B9: 96 51       LDA   $51
77BB: A7 47       STA   $0007,U
77BD: 96 51       LDA   $51
77BF: A1 47       CMPA  $0007,U
77C1: 27 05       BEQ   $77C8
77C3: 0C F4       INC   $F4
77C5: 7E 79 9D    JMP   $799D

77C8: 86 08       LDA   #$08
77CA: 8E 77 BD    LDX   #$77BD
77CD: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

77D0: 8A 77       ORA   #$77
77D2: 8A CA       ORA   #$CA
77D4: 8B 16       ADDA  #$16
77D6: 8A CA       ORA   #$CA
77D8: 8B 95       ADDA  #$95
77DA: 8A 77       ORA   #$77
77DC: 8A CA       ORA   #$CA
77DE: 8B CB       ADDA  #$CB
77E0: 8C 07 00    CMPX  #$0700
77E3: 00 BD       NEG   $BD
77E5: D0 12       SUBB  $12
77E7: BD DF 40    JSR   $DF40
77EA: 0F F4       CLR   $F4
77EC: 7E 79 9D    JMP   $799D

77EF: 0F F4       CLR   $F4
77F1: BD D0 60    JSR   $D060               ; JMP $D1FF
77F4: BD D0 30    JSR   $D030
77F7: BD D0 12    JSR   CLR_SCREEN1
77FA: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
77FD: 77 B9       ; pointer to function
77FF: 0F 0C          CLR    $0C
7801: 0F 0E          CLR   $0E
7803: 86 03       LDA   #$03
7805: 8E 78 0B    LDX   #$780B
```

```
7808: 7E D0 66    JMP   $D066

780B: 8E 18 3A    LDX   #$183A
780E: AF 49       STX   $0009,U
7810: 8E 9A B0    LDX   #$9AB0
7813: 10 8E 77 D0 LDY   #$77D0
7817: 10 AF 47    STY   $0007,U
781A: AF 4B       STX   $000B,U
781C: 86 01       LDA   #$01
781E: 8E 78 24    LDX   #$7824
7821: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7824: 10 AE 4B    LDY   $000B,U
7827: CC 0B 0E    LDD   #$0B0E
782A: ED A4       STD   ,Y
782C: C6 0D       LDB   #$0D
782E: ED 24       STD   $0004,Y
7830: 30 2A       LEAX  $000A,Y
7832: AF 22       STX   $0002,Y
7834: 30 A9 00 A4 LEAX  $00A4,Y
7838: AF 26       STX   $0006,Y
783A: AE 49       LDX   $0009,U
783C: AF 28       STX   $0008,Y
783E: 10 AE 47    LDY   $0007,U
7841: EE A4       LDU   ,Y
7843: 27 33       BEQ   $7878
7845: 5F          CLRB
7846: 86 CE       LDA   #$CE
7848: 34 10       PSHS  X
784A: BD 8D 69    JSR   $8D69                ; call RENDER_GRAPHIC
784D: DE 15       LDU   $15
784F: 30 89 02 00 LEAX  $0200,X
7853: AF 49       STX   $0009,U
7855: 35 10       PULS  X
7857: 10 AE 4B    LDY   $000B,U
785A: 31 2A       LEAY  $000A,Y
785C: CC 0B 1B    LDD   #$0B1B
785F: BD D0 B7    JSR   $D0B7                ; JMP $DE59 :
COPY_FROM_SCREEN_RAM_TO_RAM
7862: AE 4B       LDX   $000B,U
7864: 30 89 01 33 LEAX  $0133,X
7868: 10 AE 47    LDY   $0007,U
786B: 31 22       LEAY  $0002,Y
786D: 20 A8       BRA   $7817

786F: 34 02       PSHS  A
7871: B6 CC 13    LDA   $CC13
7874: 84 0F       ANDA  #$0F
7876: 35 82       PULS  A,PC ;(PUL? PC=RTS)

7878: 8D F5       BSR   $786F
787A: 27 03       BEQ   $787F
787C: BD D0 12    JSR   CLR_SCREEN1
```

```
787F: 86 07       LDA   #$07
7881: 97 0C       STA   $0C
7883: 86 3F       LDA   #$3F
7885: 97 0E       STA   $0E
7887: 8D E6       BSR   $786F
7889: 10 27 0F 19 LBEQ  $87A6
788D: 96 59       LDA   $59
788F: 84 FB       ANDA  #$FB
7891: 97 59       STA   $59
7893: DE 15       LDU   $15
7895: 10 8E 9A B0 LDY   #$9AB0
7899: 10 AF 47    STY   $0007,U
789C: 86 09       LDA   #$09
789E: A7 49       STA   $0009,U
78A0: 20 08       BRA   $78AA

78A2: 86 08       LDA   #$08
78A4: 8E 78 AA    LDX   #$78AA
78A7: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

78AA: 10 AE 47    LDY   $0007,U
78AD: EC 28       LDD   $0008,Y
78AF: CB 0D       ADDB  #$0D
78B1: D7 A7       STB   $A7
78B3: C0 0D       SUBB  #$0D
78B5: 30 A4       LEAX  ,Y
78B7: BD 5B 55    JSR   $5B55
78BA: CB 0E       ADDB  #$0E
78BC: D7 A7       STB   $A7
78BE: 30 24       LEAX  $0004,Y
78C0: BD 5B 55    JSR   $5B55
78C3: 31 A9 01 33 LEAY  $0133,Y
78C7: 10 AF 47    STY   $0007,U
78CA: 6A 49       DEC   $0009,U
78CC: 26 D4       BNE   $78A2
78CE: 86 20       LDA   #$20
78D0: 8E 87 A6    LDX   #$87A6
78D3: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

78D6: 86 3F       LDA   #$3F
78D8: 97 0F       STA   $0F
78DA: 86 07       LDA   #$07
78DC: 97 0D       STA   $0D
78DE: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
78E1: D0 C3       ; pointer to function
78E3: 10 8E 8C E8 LDY   #$8CE8
78E7: 5F          CLRB
78E8: 8E 39 5C    LDX   #$395C
78EB: 86 FD       LDA   #$FD
78ED: EE B1       LDU   [,Y++]
78EF: 27 05       BEQ   $78F6
```

```
78F1: BD 8D 69    JSR   $8D69
78F4: 20 F7       BRA   $78ED

78F6: EE A1       LDU   ,Y++
78F8: 27 05       BEQ   $78FF
78FA: BD 8D 69    JSR   $8D69               ; call RENDER_GRAPHIC
78FD: 20 F7       BRA   $78F6

78FF: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
7902: 79 2D          ; pointer to function
7904: 10 8E 79 75 LDY   #$7975
7908: 8E 98 0E    LDX   #$980E
790B: 86 01       LDA   #$01
790D: DE 15       LDU   $15
790F: AF 49       STX   $0009,U
7911: 10 AF 4B    STY   $000B,U
7914: A7 4D       STA   $000D,U
7916: AE 49       LDX   $0009,U
7918: AF 47       STX   $0007,U
791A: AE 47       LDX   $0007,U
791C: A6 80       LDA   ,X+
791E: 27 F6       BEQ   $7916
7920: A7 D8 0B    STA   [$0B,U]
7923: AF 47       STX   $0007,U
7925: A6 4D       LDA   $000D,U
7927: 8E 79 1A    LDX   #$791A
792A: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

792D: 96 86       LDA   $86
792F: 2A 10       BPL   $7941
7931: 86 07       LDA   #$07
7933: 97 0C       STA   $0C
7935: BD D0 39    JSR   $D039               ; JMP $D6CD - get a
random number into A
7938: 84 07       ANDA  #$07
793A: 4C          INCA
793B: 8E 79 41    LDX   #$7941
793E: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7941: 96 84       LDA   $84
7943: 84 03       ANDA  #$03
7945: 27 0A       BEQ   $7951
7947: 0F 0C       CLR   $0C
7949: 86 03       LDA   #$03
794B: 8E 79 51    LDX   #$7951
794E: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7951: 8E 79 8B    LDX   #$798B
7954: BD D0 39    JSR   $D039               ; JMP $D6CD - get a
random number into A
```

```
7957: 84 0F       ANDA  #$0F
7959: A6 86       LDA   A,X
795B: 97 0C       STA   $0C
795D: 86 07       LDA   #$07
795F: 8E 79 65    LDX   #$7965
7962: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7965: 96 86       LDA   $86
7967: 84 03       ANDA  #$03
7969: 27 C2       BEQ   $792D
796B: 0F 0C       CLR   $0C
796D: 86 04       LDA   #$04
796F: 8E 79 2D    LDX   #$792D
7972: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7975: 3F 3F 3F 37 2F 27 1F 17 0F 07 07 07 0F 17 1F 27
7985: 2F 37 3F 3F 3F 00

798B: FF C0 C7 1F 07 07 C0 C7 FF C0 C7 16 07 FF C0 C7

799B: 20 37       BRA   $79D4

799D: 96 F4       LDA   $F4
799F: 8A 80       ORA   #$80
79A1: 97 F4       STA   $F4
79A3: 8D 0A       BSR   $79AF
79A5: 86 81       LDA   #$81
79A7: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT       ;print
SAVE THE LAST HUMAN FAMILY
79AA: 8E 83 B2    LDX   #$83B2
79AD: 20 2A       BRA   $79D9

79AF: BD D0 60    JSR   $D060               ; JMP $D1FF
79B2: BD D0 C0    JSR   $D0C0
79B5: BD D0 12    JSR   CLR_SCREEN1
79B8: BD D0 24    JSR   $D024
79BB: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
79BE: 7E C8       ; pointer to function
79C0: 96 59       LDA   $59
79C2: 84 F3       ANDA  #$F3
79C4: 97 59       STA   $59
79C6: 0F 3F       CLR   $3F
79C8: 86 CC       LDA   #$CC
79CA: 97 8F       STA   $8F
79CC: BD 26 D2    JSR   $26D2               ; JMP $34AF
79CF: 86 80       LDA   #$80
79D1: 7E 5F 99    JMP   JMP_PRINT_STRING_LARGE_FONT       ;print
ROBOTRON: 2084

79D4: 8D D9       BSR   $79AF
79D6: 8E 7F A3    LDX   #$7FA3
```

```
79D9: DE 15       LDU   $15
79DB: AF 47       STX   $0007,U
79DD: AE 47       LDX   $0007,U
79DF: A6 80       LDA   ,X+
79E1: AF 47       STX   $0007,U
79E3: 81 09       CMPA  #$09
79E5: 22 08       BHI   $79EF
79E7: 8E 7A 08    LDX   #$7A08
79EA: 48          ASLA
79EB: AD 96       JSR   [A,X]
79ED: 20 EE       BRA   $79DD

79EF: 81 5F       CMPA  #$5F
79F1: 25 06       BCS   $79F9
79F3: 8E 79 DD    LDX   #$79DD
79F6: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call

79F9: AE 49       LDX   $0009,U
79FB: BD 5F 93    JSR   $5F93
79FE: AF 49       STX   $0009,U
7A00: 86 03       LDA   #$03
7A02: 8E 79 DD    LDX   #$79DD
7A05: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call

7A08: 7A 93 7A    DEC   $937A
7A0B: 9C 7A       CPX   $7A
7A0D: B6 7A C1    LDA   $7AC1
7A10: 7A C7 7A    DEC   $C77A
7A13: 60 7A       NEG   $FFFA,S
7A15: 84 7A       ANDA  #$7A
7A17: 5A          DECB
7A18: 7A 29 7A    DEC   $297A
7A1B: 1C 96       ANDCC #$96              ; clear carry, negative,
half carry, fast interrupt flags
7A1D: F4 84 7F    ANDB  $847F
7A20: 97 F4       STA   $F4
7A22: 10 27 FD C9 LBEQ  $77EF
7A26: 7E 79 9B    JMP   $799B

7A29: 8E 7A 31    LDX   #$7A31
7A2C: 86 00       LDA   #$00
7A2E: 7E D0 57    JMP   $D057             ; JMP $D25A - reserve
object metadata entry in list @ $9813 and call function in X

7A31: 86 0E       LDA   #$0E
7A33: A7 47       STA   $0007,U
7A35: 86 10       LDA   #$10
7A37: 8E 7A 3D    LDX   #$7A3D
7A3A: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call

7A3D: BD D0 39    JSR   $D039             ; JMP $D6CD - get a
```

```
random number into A
7A40: 84 06       ANDA  #$06
7A42: 8E 7A 52    LDX   #$7A52
7A45: 10 AE 86    LDY   A,X
7A48: BD 7B 0F    JSR   $7B0F
7A4B: 6A 47       DEC   $0007,U
7A4D: 26 E6       BNE   $7A35
7A4F: 7E D0 63    JMP   $D063               ; JMP $D1F3

7A52: 84 F5       ANDA  #$F5
7A54: 85 17       BITA  #$17
7A56: 85 40       BITA  #$40
7A58: 85 67       BITA  #$67
7A5A: BD 7D 6E    JSR   $7D6E
7A5D: 97 CF       STA   $CF
7A5F: 39          RTS

7A60: 86 14       LDA   #$14
7A62: C6 D8       LDB   #$D8
7A64: 1F 01       TFR   D,X                 ; X = $14D8, blitter
dest
7A66: CC 74 06    LDD   #$7406              ; width = 74, height =
06
7A69: BD D0 1B    JSR   $D01B               ; JMP $DADF - clear
rectangle to black
7A6C: 96 D0       LDA   $D0
7A6E: 34 02       PSHS  A
7A70: BD 7D 6E    JSR   $7D6E
7A73: C6 D8       LDB   #$D8
7A75: 0F D0       CLR   $D0
7A77: 1F 01       TFR   D,X
7A79: BD 7D 6E    JSR   $7D6E
7A7C: BD 5F 96    JSR   $5F96               ; JMP $613F: print
string in small font
7A7F: 35 02       PULS  A
7A81: 97 D0       STA   $D0
7A83: 39          RTS

7A84: 96 59       LDA   $59
7A86: 8A 0C       ORA   #$0C
7A88: 97 59       STA   $59
7A8A: BD D0 30    JSR   $D030
7A8D: BD D0 60    JSR   $D060               ; JMP $D1FF
7A90: 7E 77 A0    JMP   $77A0

7A93: BD 7D 5E    JSR   $7D5E
7A96: 10 AF 49    STY   $0009,U
7A99: 0F D0       CLR   $D0
7A9B: 39          RTS

7A9C: BD 7D 6E    JSR   $7D6E
7A9F: 1F 89       TFR   A,B
7AA1: 86 14       LDA   #$14
7AA3: ED 49       STD   $0009,U
```

```
7AA5: 8E 14 30    LDX   #$1430
7AA8: 0F D0       CLR   $D0
7AAA: BD 7D 6E    JSR   $7D6E
7AAD: 1F 89       TFR   A,B
7AAF: C0 10       SUBB  #$10
7AB1: 86 74       LDA   #$74              ; width = 74, height = B
7AB3: 7E D0 1B    JMP   $D01B             ; JMP $DADF - clear
rectangle to black

7AB6: E6 4A       LDB   $000A,U
7AB8: 86 14       LDA   #$14
7ABA: CB 0B       ADDB  #$0B
7ABC: ED 49       STD   $0009,U
7ABE: 0F D0       CLR   $D0
7AC0: 39          RTS

7AC1: BD 7D 5E    JSR   $7D5E
7AC4: 7E 7B 0F    JMP   $7B0F

7AC7: BD 7D 6E    JSR   $7D6E
7ACA: 8E 79 DD    LDX   #$79DD
7ACD: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call

7AD0: 10 8E FD 80 LDY   #$FD80
7AD4: CC FF 06    LDD   #$FF06
7AD7: 20 07       BRA   $7AE0

7AD9: 10 8E 02 80 LDY   #$0280
7ADD: CC 02 06    LDD   #$0206
7AE0: BD D0 6F    JSR   $D06F             ; JMP $D2FD - reserve an
object entry
7AE3: 10 AF 0E    STY   $000E,X
7AE6: 10 AE 47    LDY   $0007,U
7AE9: EB 2C       ADDB  $000C,Y
7AEB: E7 0C       STB   $000C,X
7AED: AB 2A       ADDA  $000A,Y
7AEF: A7 0A       STA   $000A,X
7AF1: ED 04       STD   $0004,X
7AF3: FC 26 D7    LDD   $26D7
7AF6: ED 02       STD   $0002,X           ; set current animation
frame metadata pointer
7AF8: ED 88 14    STD   $14,X             ; set previous animation
frame metadata pointer (previous = current)
7AFB: 9F 17       STX   $17
7AFD: AF 47       STX   $0007,U
7AFF: A6 49       LDA   $0009,U
7B01: 8E 7B 07    LDX   #$7B07
7B04: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call


7B07: AE 47       LDX   $0007,U
7B09: BD D0 75    JSR   $D075             ; JMP $D31B - deallocate
```

```
object and erase object from screen              ; JMP $D31B
7B0C: 7E D0 63    JMP   $D063               ; JMP $D1F3

7B0F: 34 12       PSHS  X,A
7B11: 86 00       LDA   #$00
7B13: 8E 7B 1E    LDX   #$7B1E
7B16: BD D0 5A    JSR   $D05A               ; JMP $D243 - reserve
object metadata entry in list @ $981D and call function in X
7B19: 10 AF 07    STY   $0007,X
7B1C: 35 92       PULS  A,X,PC ;(PUL? PC=RTS)

7B1E: BD D0 6F    JSR   $D06F               ; JMP $D2FD
7B21: AF 4B       STX   $000B,U
7B23: 6F C8 12    CLR   $12,U
7B26: 8D 11       BSR   $7B39
7B28: 8D 0F       BSR   $7B39
7B2A: DE 15       LDU   $15
7B2C: AE 4B       LDX   $000B,U
7B2E: EC 02       LDD   $0002,X             ; get pointer to current
animation frame metadata
7B30: ED 88 14    STD   $14,X               ; set previous animation
frame metadata pointer (previous = current)
7B33: 9F 17       STX   $17
7B35: 8D 02       BSR   $7B39
7B37: 20 FC       BRA   $7B35

7B39: 35 06       PULS  A,B
7B3B: DE 15       LDU   $15
7B3D: ED 4E       STD   $000E,U
7B3F: BD 7D 6E    JSR   $7D6E
7B42: 48          ASLA
7B43: 8E 7B 58    LDX   #$7B58
7B46: AE 86       LDX   A,X
7B48: 9F 2B       STX   $2B
7B4A: AE 4B       LDX   $000B,U
7B4C: 10 AE 49    LDY   $0009,U
7B4F: AD 9F 98 2B JSR   [$982B,X]
7B53: DE 15       LDU   $15
7B55: 6E D8 0E    JMP   [$0E,U]

7B58: 8D A0       BSR   $7AFA
7B5A: 7D 31 7D    TST   $317D
7B5D: 4B          Illegal Opcode
7B5E: 7D 19 7D    TST   $197D
7B61: 1F 7D       TFR   inv,inv
7B63: 25 7D       BCS   $7BE2
7B65: 2B 7C       BMI   $7BE3
7B67: C1 7C       CMPB  #$7C
7B69: B2 7C B9    SBCA  $7CB9
7B6C: 7C 8F 7C    INC   $8F7C
7B6F: 87          Illegal Opcode
7B70: 7C 81 7C    INC   $817C
7B73: 6F 7C       CLR   $FFFC,S
7B75: 68 7C       ASL   $FFFC,S
```

```
7B77: 54          LSRB
7B78: 7C 5C 7C    INC   $5C7C
7B7B: 4E          Illegal Opcode
7B7C: 7C 1F 7C    INC   $1F7C
7B7F: 26 7C       BNE   $7BFD
7B81: 37 7C       PULU  B,DP,X,Y,S
7B83: 05          Illegal Opcode
7B84: 7C CB D0    INC   $CBD0
7B87: 63 7D       COM   $FFFD,S
7B89: 41          Illegal Opcode
7B8A: 7B          Illegal Opcode
7B8B: 90 7C       SUBA  $7C
7B8D: D9 7B       ADCB  $7B
7B8F: FD 35 06    STD   $3506
7B92: ED C8 10    STD   $10,U
7B95: BD 7C 54    JSR   $7C54
7B98: 6F 88 12    CLR   $12,X
7B9B: BD 7D 5E    JSR   $7D5E
7B9E: 10 AF C8 15 STY   $15,U
7BA2: BD 7D 5E    JSR   $7D5E
7BA5: 10 AF C8 17 STY   $17,U
7BA9: A6 0A       LDA   $000A,X
7BAB: E6 0C       LDB   $000C,X
7BAD: ED 04       STD   $0004,X
7BAF: ED 06       STD   $0006,X
7BB1: 86 03       LDA   #$03
7BB3: 8E 7B B9    LDX   #$7BB9
7BB6: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7BB9: AE 4B       LDX   $000B,U
7BBB: 8D 2F       BSR   $7BEC
7BBD: EC 04       LDD   $0004,X
7BBF: AB 0E       ADDA  $000E,X
7BC1: EB 88 10    ADDB  $10,X
7BC4: ED 04       STD   $0004,X
7BC6: EC C8 15    LDD   $15,U
7BC9: 10 AE 02    LDY   $0002,X
7BCC: BD 1A C8    JSR   $1AC8
7BCF: EC 04       LDD   $0004,X
7BD1: ED 06       STD   $0006,X
7BD3: A6 C8 18    LDA   $18,U
7BD6: 27 05       BEQ   $7BDD
7BD8: 8D 1A       BSR   $7BF4
7BDA: BD D0 18    JSR   $D018                ; JMP   $DAF2 - do blit
7BDD: 6A C8 17    DEC   $17,U
7BE0: 26 CF       BNE   $7BB1
7BE2: 8D 10       BSR   $7BF4
7BE4: 8D 06       BSR   $7BEC
7BE6: BD 7C 5C    JSR   $7C5C
7BE9: 6E D8 10    JMP   [$10,U]

7BEC: EC 06       LDD   $0006,X             ; D= blitter
destination
```

```
7BEE: 10 AE 02    LDY   $0002,X             ; Y = pointer to
animation frame metadata
7BF1: 7E D0 1E    JMP   $D01E               ; JMP $DABF: clear
image rectangle

7BF4: EC 04       LDD   $0004,X
7BF6: A7 0A       STA   $000A,X
7BF8: E7 0C       STB   $000C,X
7BFA: 6F 0B       CLR   $000B,X
7BFC: 39          RTS

7BFD: AF 47       STX   $0007,U
7BFF: BD 7C 54    JSR   $7C54
7C02: 7E 38 95    JMP   $3895

7C05: BD 7D 5E    JSR   $7D5E
7C08: 8E 7B 35    LDX   #$7B35
7C0B: 86 00       LDA   #$00
7C0D: BD D0 5A    JSR   $D05A               ; JMP $D243 - reserve
object metadata entry in list @ $981D and call function in X
7C10: 10 AF 07    STY   $0007,X
7C13: EC 4B       LDD   $000B,U
7C15: ED 0B       STD   $000B,X
7C17: EC 49       LDD   $0009,U
7C19: ED 09       STD   $0009,X
7C1B: 6F 88 12    CLR   $12,X
7C1E: 39          RTS

7C1F: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
7C22: 7A D0          ; pointer to function
7C24: 20 05       BRA   $7C2B

7C26: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
7C29: 7A D9          ; pointer to function
7C2B: EC 4B          LDD   $B,U
7C2D: ED 07       STD   $0007,X
7C2F: BD 7D 6E    JSR   $7D6E
7C32: A7 09       STA   $0009,X
7C34: 7E 7C 87    JMP   $7C87

7C37: BD 7D 6E    JSR   $7D6E
7C3A: BD 7D 5E    JSR   $7D5E
7C3D: E6 C8 12    LDB   $12,U
7C40: 26 03       BNE   $7C45
7C42: A7 C8 12    STA   $12,U
7C45: 6A C8 12    DEC   $12,U
7C48: 27 03       BEQ   $7C4D
7C4A: 10 AF 47    STY   $0007,U
7C4D: 39          RTS

7C4E: BD 7D 5E    JSR   $7D5E
7C51: 7E 7B 0F    JMP   $7B0F
```

```
7C54: BD D0 75    JSR   $D075               ; JMP $D31B - deallocate
object and erase object from screen             ; JMP $D31B
7C57: EC 84       LDD   ,X
7C59: DD 1B       STD   $1B
7C5B: 39          RTS

7C5C: EC 02       LDD   $0002,X             ; get pointer to current
animation frame metadata
7C5E: ED 88 14    STD   $14,X               ; set previous animation
frame metadata pointer (previous = current)
7C61: DC 17       LDD   $17
7C63: ED 84       STD   ,X                  ; set next object
pointer at (*x)
7C65: 9F 17       STX   $17
7C67: 39          RTS

7C68: BD 7D 5E    JSR   $7D5E
7C6B: 10 AF 47    STY   $0007,U
7C6E: 39          RTS

7C6F: BD D0 75    JSR   $D075               ; JMP $D31B - deallocate
object and erase object from screen             ; JMP $D31B
7C72: 86 A6       LDA   #$A6
7C74: 97 A7       STA   $A7
7C76: CC FF 00    LDD   #$FF00
7C79: DD 88       STD   $88
7C7B: BD 5B 43    JSR   $5B43               ; JMP $5C1F - create an
explosion
7C7E: 7E D0 63    JMP   $D063               ; JMP $D1F3

7C81: BD D0 75    JSR   $D075               ; JMP $D31B - deallocate
object and erase object from screen             ; JMP $D31B
7C84: 7E D0 63    JMP   $D063               ; JMP $D1F3

7C87: 35 10       PULS  X
7C89: BD 7D 6E    JSR   $7D6E
7C8C: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7C8F: 35 06       PULS  A,B
7C91: ED C8 10    STD   $10,U
7C94: BD 7D 5E    JSR   $7D5E
7C97: 10 AF C8 13 STY   $13,U
7C9B: 10 AE 49    LDY   $0009,U
7C9E: BD 7D 41    JSR   $7D41
7CA1: A6 C8 13    LDA   $13,U
7CA4: 8E 7C AA    LDX   #$7CAA
7CA7: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7CAA: 6A C8 14    DEC   $14,U
7CAD: 26 EC       BNE   $7C9B
7CAF: 6E D8 10    JMP   [$10,U]
```

```
7CB2: BD 7D 5E    JSR   $7D5E
7CB5: 10 AF 0E    STY   $000E,X
7CB8: 39          RTS

7CB9: BD 7D 5E    JSR   $7D5E
7CBC: 10 AF 88 10 STY   $10,X
7CC0: 39          RTS

7CC1: BD 7D 5E    JSR   $7D5E
7CC4: 1F 20       TFR   Y,D
7CC6: E7 0C       STB   $000C,X
7CC8: A7 0A       STA   $000A,X
7CCA: 39          RTS

7CCB: BD 7D 5E    JSR   $7D5E
7CCE: 1F 20       TFR   Y,D
7CD0: AB 0A       ADDA  $000A,X
7CD2: A7 0A       STA   $000A,X
7CD4: EB 0C       ADDB  $000C,X
7CD6: E7 0C       STB   $000C,X
7CD8: 39          RTS

7CD9: A6 05       LDA   $0005,X
7CDB: A7 C8 13    STA   $13,U
7CDE: 86 40       LDA   #$40
7CE0: A7 C8 15    STA   $15,U
7CE3: BD D0 39    JSR   $D039             ; JMP $D6CD - get a
random number into A
7CE6: 84 07       ANDA  #$07
7CE8: AB C8 13    ADDA  $13,U
7CEB: A7 05       STA   $0005,X
7CED: 86 01       LDA   #$01
7CEF: 8E 7C F5    LDX   #$7CF5
7CF2: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call

7CF5: AE 4B       LDX   $000B,U
7CF7: BD D0 39    JSR   $D039             ; JMP $D6CD - get a
random number into A
7CFA: 84 07       ANDA  #$07
7CFC: 40          NEGA
7CFD: AB C8 13    ADDA  $13,U
7D00: A7 05       STA   $0005,X
7D02: 86 01       LDA   #$01
7D04: 8E 7D 0A    LDX   #$7D0A
7D07: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call

7D0A: AE 4B       LDX   $000B,U
7D0C: 6A C8 15    DEC   $15,U
7D0F: 26 D2       BNE   $7CE3
7D11: A6 C8 13    LDA   $13,U
7D14: A7 05       STA   $0005,X
```

```
7D16: 7E D0 63    JMP   $D063               ; JMP $D1F3

7D19: BD 7D 6E    JSR   $7D6E
7D1C: 6E B8 04    JMP   [$04,Y]

7D1F: BD 7D 6E    JSR   $7D6E
7D22: 6E B8 06    JMP   [$06,Y]

7D25: BD 7D 6E    JSR   $7D6E
7D28: 6E B8 08    JMP   [$08,Y]

7D2B: BD 7D 6E    JSR   $7D6E
7D2E: 6E B8 0A    JMP   [$0A,Y]

7D31: BD 7D 5E    JSR   $7D5E
7D34: 10 AF 49    STY   $0009,U
7D37: 4F          CLRA
7D38: A7 4D       STA   $000D,U
7D3A: 8D 14       BSR   $7D50
7D3C: AE 4B       LDX   $000B,U
7D3E: ED 02       STD   $0002,X
7D40: 39          RTS

7D41: A6 4D       LDA   $000D,U
7D43: 4C          INCA
7D44: A1 22       CMPA  $0002,Y
7D46: 25 F0       BCS   $7D38
7D48: 4F          CLRA
7D49: 20 ED       BRA   $7D38

7D4B: BD 7D 6E    JSR   $7D6E
7D4E: 20 EA       BRA   $7D3A

7D50: 34 60       PSHS  U,Y
7D52: DE 15       LDU   $15
7D54: 10 AE 49    LDY   $0009,U
7D57: E6 23       LDB   $0003,Y
7D59: 3D          MUL
7D5A: E3 B4       ADDD  [,Y]
7D5C: 35 E0       PULS  Y,U,PC ;(PUL? PC=RTS)

7D5E: 34 50       PSHS  U,X
7D60: 8D 07       BSR   $7D69
7D62: 10 AE 81    LDY   ,X++
7D65: AF 47       STX   $0007,U
7D67: 35 D0       PULS  X,U,PC ;(PUL? PC=RTS)

7D69: DE 15       LDU   $15
7D6B: AE 47       LDX   $0007,U
7D6D: 39          RTS

7D6E: 34 50       PSHS  U,X
7D70: 8D F7       BSR   $7D69
7D72: A6 80       LDA   ,X+
```

```
7D74: 20 EF       BRA   $7D65

7D76: 5F          CLRB
7D77: ED C8 13    STD   $13,U
7D7A: A6 2C       LDA   $000C,Y
7D7C: 40          NEGA
7D7D: 20 1C       BRA   $7D9B

7D7F: C6 03       LDB   #$03
7D81: ED C8 13    STD   $13,U
7D84: A6 2C       LDA   $000C,Y
7D86: 5F          CLRB
7D87: 20 12       BRA   $7D9B

7D89: C6 06       LDB   #$06
7D8B: ED C8 13    STD   $13,U
7D8E: E6 2C       LDB   $000C,Y
7D90: 20 08       BRA   $7D9A

7D92: C6 09       LDB   #$09
7D94: ED C8 13    STD   $13,U
7D97: E6 2C       LDB   $000C,Y
7D99: 50          NEGB
7D9A: 4F          CLRA
7D9B: ED C8 15    STD   $15,U
7D9E: 35 06       PULS  A,B
7DA0: ED C8 10    STD   $10,U
7DA3: CC 7D F2    LDD   #$7DF2
7DA6: ED C8 17    STD   $17,U
7DA9: 10 AE 49    LDY   $0009,U
7DAC: 8E 7D B4    LDX   #$7DB4
7DAF: A6 2D       LDA   $000D,Y
7DB1: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7DB4: AE 4B       LDX   $000B,U
7DB6: EC C8 15    LDD   $15,U
7DB9: 8D 24       BSR   $7DDF
7DBB: 10 AE C8 17 LDY   $17,U
7DBF: 31 21       LEAY  $0001,Y
7DC1: 10 8C 7D F3 CMPY  #$7DF3
7DC5: 25 04       BCS   $7DCB
7DC7: 10 8E 7D EF LDY   #$7DEF
7DCB: A6 A4       LDA   ,Y
7DCD: 10 AF C8 17 STY   $17,U
7DD1: AB C8 14    ADDA  $14,U
7DD4: BD 7D 38    JSR   $7D38
7DD7: 6A C8 13    DEC   $13,U
7DDA: 26 CD       BNE   $7DA9
7DDC: 6E D8 10    JMP   [$10,U]

7DDF: 34 04       PSHS  B
7DE1: 5F          CLRB
7DE2: 47          ASRA
```

```
7DE3: 56          RORB
7DE4: E3 0A       ADDD  $000A,X
7DE6: ED 0A       STD   $000A,X
7DE8: 35 04       PULS  B
7DEA: EB 0C       ADDB  $000C,X
7DEC: E7 0C       STB   $000C,X
7DEE: 39          RTS

7DEF: 00 01       NEG   $01
7DF1: 00 02       NEG   $02
7DF3: 5F          CLRB
7DF4: 20 0A       BRA   $7E00

7DF6: C6 0D       LDB   #$0D
7DF8: 20 06       BRA   $7E00

7DFA: C6 1A       LDB   #$1A
7DFC: 20 02       BRA   $7E00

7DFE: C6 27       LDB   #$27
7E00: AE B8 0C    LDX   [$0C,Y]
7E03: 3A          ABX
7E04: AF C8 13    STX   $13,U
7E07: A7 C8 15    STA   $15,U
7E0A: 35 06       PULS  A,B
7E0C: ED C8 10    STD   $10,U
7E0F: 86 08       LDA   #$08
7E11: 8E 7E 17    LDX   #$7E17
7E14: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7E17: 8D 08       BSR   $7E21
7E19: 6A C8 15    DEC   $15,U
7E1C: 26 F1       BNE   $7E0F
7E1E: 6E D8 10    JMP   [$10,U]

7E21: AE 4B       LDX   $000B,U
7E23: 10 AE C8 13 LDY   $13,U
7E27: EC 21       LDD   $0001,Y
7E29: 8D B4       BSR   $7DDF
7E2B: A6 A4       LDA   ,Y
7E2D: 44          LSRA
7E2E: 44          LSRA
7E2F: BD 7D 38    JSR   $7D38
7E32: 31 23       LEAY  $0003,Y
7E34: E6 A4       LDB   ,Y
7E36: C1 FF       CMPB  #$FF
7E38: 26 02       BNE   $7E3C
7E3A: 31 34       LEAY  $FFF4,Y
7E3C: 10 AF C8 13 STY   $13,U
7E40: 39          RTS

7E41: 00 0E       NEG   $0E
7E43: 0C 04       INC   $04
```

```
7E45: 7D F3 7D    TST   $F37D
7E48: F6 7D FA    LDB   $7DFA
7E4B: 7D FE 00    TST   $FE00
7E4E: 18          Illegal Opcode
7E4F: 1A CB       ORCC  #$CB
7E51: 01          Illegal Opcode
7E52: 04 00       LSR   $00
7E54: 10 0C       Illegal Opcode
7E56: 04 7D       LSR   $7D
7E58: F3 7D F6    ADDD  $7DF6
7E5B: 7D FA 7D    TST   $FA7D
7E5E: FE 00 18    LDU   $0018
7E61: 00 12       NEG   $12
7E63: 0C 04       INC   $04
7E65: 7D F3 7D    TST   $F37D
7E68: F6 7D FA    LDB   $7DFA
7E6B: 7D FE 00    TST   $FE00
7E6E: 18          Illegal Opcode
7E6F: 00 14       NEG   $14
7E71: 0C 04       INC   $04
7E73: 7D F3 7D    TST   $F37D
7E76: F6 7D FA    LDB   $7DFA
7E79: 7D FE 00    TST   $FE00
7E7C: 16 1A C3    LBRA  $9942

7E7F: 0C 04       INC   $04
7E81: 7D 76 7D    TST   $767D
7E84: 7F 7D 89    CLR   $7D89
7E87: 7D 92 02    TST   $9202
7E8A: 08 38       ASL   $38
7E8C: 91 03       CMPA  $03
7E8E: 04 38       LSR   $38
7E90: 93 24       SUBD  $24
7E92: 04 11       LSR   $11
7E94: 48          ASLA
7E95: 06 04       ROR   $04
7E97: 26 D5       BNE   $7E6E
7E99: 0C 04       INC   $04
7E9B: 7D 76 7D    TST   $767D
7E9E: 7F 7D 89    CLR   $7D89
7EA1: 7D 92 01    TST   $9201
7EA4: 02          Illegal Opcode
7EA5: 4B          Illegal Opcode
7EA6: 06 09       ROR   $09
7EA8: 04 11       LSR   $11
7EAA: 46          RORA
7EAB: 08 04       ASL   $04
7EAD: 4B          Illegal Opcode
7EAE: 08 05       ASL   $05
7EB0: 06 4B       ROR   $4B
7EB2: 0A 04       DEC   $04
7EB4: 04 00       LSR   $00
7EB6: 0C 05       INC   $05
7EB8: 04 00       LSR   $00
```

```
7EBA: 1A 01       ORCC  #$01
7EBC: 04 0D       LSR   $0D
7EBE: F4 26 05    ANDB  $2605
7EC1: 0C F4       INC   $F4
7EC3: 7E 79 9D    JMP   $799D

7EC6: 0C F4       INC   $F4
7EC8: CC 10 05    LDD   #$1005              ; width = 10, height = 5
7ECB: 8E 50 EE    LDX   #$50EE              ; blitter destination
7ECE: BD D0 1B    JSR   $D01B               ; JMP $DADF - clear
rectangle to black
7ED1: D6 51       LDB   $51
7ED3: E7 47       STB   $0007,U
7ED5: 86 7F       LDA   #$7F
7ED7: 8D 55       BSR   $7F2E
7ED9: 86 20       LDA   #$20
7EDB: A7 4A       STA   $000A,U
7EDD: 86 08       LDA   #$08
7EDF: 8E 7E E5    LDX   #$7EE5
7EE2: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7EE5: 8D 39       BSR   $7F20
7EE7: 96 51       LDA   $51
7EE9: A1 47       CMPA  $0007,U
7EEB: 26 D0       BNE   $7EBD
7EED: 6A 4A       DEC   $000A,U
7EEF: 26 EC       BNE   $7EDD
7EF1: 8E CC 00    LDX   #$CC00
7EF4: BD D0 A5    JSR   $D0A5
7EF7: 5D          TSTB
7EF8: 27 DF       BEQ   $7ED9
7EFA: 8D 47       BSR   $7F43
7EFC: 86 7D       LDA   #$7D
7EFE: 8D 2E       BSR   $7F2E
7F00: 86 0A       LDA   #$0A
7F02: A7 4A       STA   $000A,U
7F04: 96 51       LDA   $51
7F06: A1 47       CMPA  $0007,U
7F08: 27 04       BEQ   $7F0E
7F0A: 8D 37       BSR   $7F43
7F0C: 20 AF       BRA   $7EBD

7F0E: 86 08       LDA   #$08
7F10: 8E 7F 16    LDX   #$7F16
7F13: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

7F16: 8D 08       BSR   $7F20
7F18: 6A 4A       DEC   $000A,U
7F1A: 26 E8       BNE   $7F04
7F1C: 8D 25       BSR   $7F43
7F1E: 20 A8       BRA   $7EC8
```

```
7F20: 96 F4       LDA   $F4
7F22: 2B 09       BMI   $7F2D
7F24: B6 C8 04    LDA   widget_pia_dataa
7F27: 81 41       CMPA  #$41
7F29: 10 27 FB 57 LBEQ  $7A84
7F2D: 39          RTS

7F2E: 34 06       PSHS  B,A
7F30: 96 D0       LDA   $D0
7F32: D6 CF       LDB   $CF
7F34: 34 06       PSHS  B,A
7F36: EC 62       LDD   $0002,S
7F38: BD 5F 96    JSR   $5F96               ; JMP $613F: print
string in small font
7F3B: 35 06       PULS  A,B
7F3D: 97 D0       STA   $D0
7F3F: D7 CF       STB   $CF
7F41: 35 86       PULS  A,B,PC ;(PUL? PC=RTS)

7F43: 34 16       PSHS  X,B,A
7F45: 8E 20 EE    LDX   #$20EE              ; blitter destination
7F48: CC 50 05    LDD   #$5005              ; width = 50, height = 5
7F4B: BD D0 1B    JSR   $D01B               ; JMP $DADF - clear
rectangle to black
7F4E: 35 96       PULS  A,B,X,PC ;(PUL? PC=RTS)

7F50:  COPYRIGHT 1982 WILLIAMS ELECTRO
7F70: NICS INC.  ALL RIGHTS RESERVED.
7F90:  "ROBOTRON: 2084"

7FA3: 8 ]INSPIRED BY HIS NEVER ENDIN
7FC3: GQUEST FOR PROGRESS<IN *2084
7FE3: ] MAN PERFECTS THE *ROBOTRONS?
8003:  A ROBOT SPECIES SO ADVANCED
8023:  THATMAN IS INFERIOR TO HIS OWN
8043:  CREATION= ]GUIDED BY THEIR
8063: INFALLIBLE LOGIC<THE *ROBOTRON
8083: S] CONCLUDE?0*THE HUMAN RAC
80A3: E IS INEFFICIENT<AND THEREFORE
80C3: MUST BE DESTROYED=p6@]7YOU A
80E3: RE THE LAST HOPE OF MANKIND=`
8103: DUE TO A GENETIC ENGINEERING ER
8123: ROR<YOU POSSESS SUPERHUMAN POWE
8143: RS=]YOUR MISSION IS TO* STOP
8163:  THE ROBOTRONS<]AND *SAVE THE
8183:  LAST HUMAN FAMILY?w sXt
81A3: Xup~pX 3THE FORCE OF GRO
81C3: UND ROVINGUNIT NETWORK TERMINAT
81E3: OR *tv[GRUNT\:ROBOTRONS3 SEE
8203: K TODESTROY YOU=X U!THE*
8223: w HULK ROBOTRONS USEEK OUTAND
8243:  ELIMINATE THE LAST HUMAN FAMILY
8263: =y~ p ]=THE*,x SPHEREOID
8283: S AND QUARKS  ]ARE PROGRAMMED
```

```
82A3: TO MANUFACTURE*3yENFORCER AND
82C3:  TANK ROBOTRONS= ~X BEWARE
82E3: OF THE INGENIOUS* zBRAIN ROBO
8303: TRONS THAT POSSESSTHE POWER T
8323: O REPROGRAMHUMANS INTO SINISTER
8343:  *PROGS=s,| Ld  ~@]AS
8363: YOU STRUGGLE TO SAVEHUMANITY< B
8383: E SURE TO AVOID*ELECTRODES] I
83A3: N YOUR PATH

83AE: 3D          MUL
83AF: D0 AD       SUBB  $AD
83B1: 06 03       ROR   $03
83B3: 87          Illegal Opcode
83B4: 15          Illegal Opcode
83B5: FF 09 01    STU   $0901
83B8: 7E 97 07    JMP   $9707

83BB: 18          Illegal Opcode
83BC: A0 15       SUBA  $FFF5,X
83BE: 84 0F       ANDA  #$0F
83C0: 04 60       LSR   $60
83C2: 02          Illegal Opcode
83C3: 06 0B       ROR   $0B
83C5: 90 0B       SUBA  $0B
83C7: FF 0B FF    STU   $0BFF
83CA: 03 50       COM   $50
83CC: 04 68       LSR   $68
83CE: 02          Illegal Opcode
83CF: 03 0B       COM   $0B
83D1: 80 0B       SUBA  #$0B
83D3: 80 02       SUBA  #$02
83D5: 06 0B       ROR   $0B
83D7: 40          NEGA
83D8: 03 18       COM   $18
83DA: 05          Illegal Opcode
83DB: 23 02       BLS   $83DF
83DD: 00 0B       NEG   $0B
83DF: FF 02 06    STU   $0206
83E2: 0B          Illegal Opcode
83E3: E0 06       SUBB  $0006,X
83E5: 3C 02       CWAI  #$02
83E7: 03 0B       COM   $0B
83E9: 38          Illegal Opcode
83EA: 05          Illegal Opcode
83EB: 38          Illegal Opcode
83EC: 02          Illegal Opcode
83ED: 03 0B       COM   $0B
83EF: 20 02       BRA   $83F3

83F1: 06 0B       ROR   $0B
83F3: E3 06       ADDD  $0006,X
83F5: 20 02       BRA   $83F9
```

```
83F7: 00 0B       NEG   $0B
83F9: 3C 02       CWAI  #$02
83FB: 06 0B       ROR   $0B
83FD: F1 0B 30    CMPB  $0B30
8400: 05          Illegal Opcode
8401: 20 02       BRA   $8405

8403: 00 0B       NEG   $0B
8405: 20 02       BRA   $8409

8407: 06 0B       ROR   $0B
8409: 90 0B       SUBA  $0B
840B: 90 04       SUBA  $04
840D: 80 0D       SUBA  #$0D
840F: 0B          Illegal Opcode
8410: FF 0B 60    STU   $0B60
8413: 13          SYNC
8414: 1D          SEX
8415: 05          Illegal Opcode
8416: 14          Illegal Opcode
8417: 0A 84       DEC   $84
8419: 13          SYNC
841A: 12          NOP
841B: 1B          Illegal Opcode
841C: 05          Illegal Opcode
841D: 14          Illegal Opcode
841E: 08 84       ASL   $84
8420: 1A 13       ORCC  #$13
8422: 1D          SEX
8423: 05          Illegal Opcode
8424: 14          Illegal Opcode
8425: 14          Illegal Opcode
8426: 84 21       ANDA  #$21
8428: 12          NOP
8429: 1B          Illegal Opcode
842A: 05          Illegal Opcode
842B: 14          Illegal Opcode
842C: 05          Illegal Opcode
842D: 84 28       ANDA  #$28
842F: 13          SYNC
8430: 1D          SEX
8431: 05          Illegal Opcode
8432: 14          Illegal Opcode
8433: 0A 84       DEC   $84
8435: 2F 0B       BLE   $8442
8437: FF 0B FF    STU   $0BFF
843A: 0B          Illegal Opcode
843B: 50          NEGB
843C: 13          SYNC
843D: 0B          Illegal Opcode
843E: 10 14       Illegal Opcode
8440: 0E 84       JMP   $84

8442: 3C 0B       CWAI  #$0B
```

```
8444: E0 12       SUBB  $FFF2,X
8446: 0B          Illegal Opcode
8447: 0B          Illegal Opcode
8448: 14          Illegal Opcode
8449: 0A 84       DEC   $84
844B: 45          Illegal Opcode
844C: 0B          Illegal Opcode
844D: FF 0B E3    STU   $0BE3
8450: 0B          Illegal Opcode
8451: 10 13       Illegal Opcode
8453: 0E 80       JMP   $80

8455: 13          SYNC
8456: 18          Illegal Opcode
8457: 08 13       ASL   $13
8459: 20 08       BRA   $8463

845B: 14          Illegal Opcode
845C: 03 84       COM   $84
845E: 58          ASLB
845F: 0B          Illegal Opcode
8460: FF 0B 44    STU   $0B44
8463: 12          NOP
8464: 09 C0       ROL   $C0
8466: 0B          Illegal Opcode
8467: E0 12       SUBB  $FFF2,X
8469: 08 FF       ASL   $FF
846B: 0B          Illegal Opcode
846C: 90 13       SUBA  $13
846E: 04 10       LSR   $10
8470: 13          SYNC
8471: 06 10       ROR   $10
8473: 13          SYNC
8474: 08 01       ASL   $01
8476: 17 01 7E    LBSR  $85F7
8479: 41          Illegal Opcode
847A: 07 18       ASR   $18
847C: A0 11       SUBA  $FFF1,X
847E: 84 AD       ANDA  #$AD
8480: 11 84       Illegal Opcode
8482: C6 04       LDB   #$04
8484: 28 0F       BVC   $8495
8486: 01          Illegal Opcode
8487: 7E B5 02    JMP   $B502

848A: 00 16       NEG   $16
848C: 00 04       NEG   $04
848E: 10 0B       Illegal Opcode
8490: 30 0F       LEAX  $000F,X
8492: 01          Illegal Opcode
8493: 7E 53 07    JMP   $5307

8496: 18          Illegal Opcode
8497: A0 0B       SUBA  $000B,X
```

```
8499: 40          NEGA
849A: 10 04       Illegal Opcode
849C: 30 06       LEAX  $0006,X
849E: 10 04       Illegal Opcode
84A0: 10 05       Illegal Opcode
84A2: 13          SYNC
84A3: 03 12       COM   $12
84A5: 0F 01       CLR   $01
84A7: 7E B9 10    JMP   $B910

84AA: 0B          Illegal Opcode
84AB: 80 0C       SUBA  #$0C
84AD: 01          Illegal Opcode
84AE: 7E 53 07    JMP   $5307

84B1: 18          Illegal Opcode
84B2: A0 0F       SUBA  $000F,X
84B4: 0B          Illegal Opcode
84B5: 70 10 04    NEG   $1004
84B8: 1E 0F       EXG   D,inv
84BA: 01          Illegal Opcode
84BB: 7E B5 02    JMP   $B502

84BE: 01          Illegal Opcode
84BF: 16 00 04    LBRA  $84C6

84C2: 10 0B       Illegal Opcode
84C4: 30 0C       LEAX  $000C,X
84C6: 01          Illegal Opcode
84C7: 7E 61 07    JMP   $6107

84CA: 18          Illegal Opcode
84CB: A0 0F       SUBA  $000F,X
84CD: 0B          Illegal Opcode
84CE: D8 10       EORB  $10
84D0: 04 15       LSR   $15
84D2: 0F 01       CLR   $01
84D4: 7E B5 02    JMP   $B502

84D7: 02          Illegal Opcode
84D8: 16 00 04    LBRA  $84DF

84DB: 10 0B       Illegal Opcode
84DD: 30 0F       LEAX  $000F,X
84DF: 01          Illegal Opcode
84E0: 7E 61 07    JMP   $6107

84E3: 18          Illegal Opcode
84E4: A0 0B       SUBA  $000B,X
84E6: C0 10       SUBB  #$10
84E8: 04 04       LSR   $04
84EA: 05          Illegal Opcode
84EB: 08 04       ASL   $04
84ED: 10 05       Illegal Opcode
```

```
84EF: 18          Illegal Opcode
84F0: 03 01       COM   $01
84F2: 0E 84       JMP   $84

84F4: A5 01       BITA  $0001,X
84F6: 7E 8B 07    JMP   $8B07

84F9: 86 85       LDA   #$85
84FB: 0B          Illegal Opcode
84FC: 10 16       Illegal Opcode
84FE: FE 04 02    LDU   $0402
8501: 01          Illegal Opcode
8502: 0B          Illegal Opcode
8503: 08 16       ASL   $16
8505: FD 04 02    STD   $0402
8508: 00 0B       NEG   $0B
850A: 20 16       BRA   $8522

850C: FE 04 02    LDU   $0402
850F: 02          Illegal Opcode
8510: 14          Illegal Opcode
8511: 02          Illegal Opcode
8512: 84 FB       ANDA  #$FB
8514: 0E 85       JMP   $85

8516: 8B 01       ADDA  #$01
8518: 7E 8B 07    JMP   $8B07

851B: 88 90       EORA  #$90
851D: 0B          Illegal Opcode
851E: 09 16       ROL   $16
8520: FE 02 02    LDU   $0202
8523: 02          Illegal Opcode
8524: 0B          Illegal Opcode
8525: 12          NOP
8526: 16 FE 02    LBRA  $832B

8529: 02          Illegal Opcode
852A: 00 0B       NEG   $0B
852C: 12          NOP
852D: 16 FE 02    LBRA  $8332

8530: 02          Illegal Opcode
8531: 01          Illegal Opcode
8532: 0B          Illegal Opcode
8533: 0B          Illegal Opcode
8534: 16 FE 02    LBRA  $8339

8537: 02          Illegal Opcode
8538: 00 14       NEG   $14
853A: 02          Illegal Opcode
853B: 85 1D       BITA  #$1D
853D: 0E 85       JMP   $85
```

```
853F: 8B 01       ADDA  #$01
8541: 7E 8B 07    JMP   $8B07

8544: 88 BC       EORA  #$BC
8546: 0B          Illegal Opcode
8547: 10 16       Illegal Opcode
8549: FE FD 02    LDU   $FD02
854C: 01          Illegal Opcode
854D: 0B          Illegal Opcode
854E: 06 16       ROR   $16
8550: FE FD 02    LDU   $FD02
8553: 00 0B       NEG   $0B
8555: 1A 16       ORCC  #$16
8557: FE FD 02    LDU   $FD02
855A: 02          Illegal Opcode
855B: 0B          Illegal Opcode
855C: 08 16       ASL   $16
855E: FE FD 14    LDU   $FD14
8561: 02          Illegal Opcode
8562: 85 46       BITA  #$46
8564: 0E 85       JMP   $85

8566: 8B 01       ADDA  #$01
8568: 7E 8B 07    JMP   $8B07

856B: 88 B1       EORA  #$B1
856D: 0B          Illegal Opcode
856E: 04 16       LSR   $16
8570: FE FE 02    LDU   $FE02
8573: 01          Illegal Opcode
8574: 0B          Illegal Opcode
8575: 09 16       ROL   $16
8577: FD FE 02    STD   $FE02
857A: 00 0B       NEG   $0B
857C: 18          Illegal Opcode
857D: 16 FE FE    LBRA  $847E

8580: 02          Illegal Opcode
8581: 02          Illegal Opcode
8582: 0B          Illegal Opcode
8583: 15          Illegal Opcode
8584: 16 FE FE    LBRA  $8485

8587: 14          Illegal Opcode
8588: 02          Illegal Opcode
8589: 85 74       BITA  #$74
858B: 02          Illegal Opcode
858C: 00 16       NEG   $16
858E: FC 02 0B    LDD   $020B
8591: 04 02       LSR   $02
8593: 02          Illegal Opcode
8594: 16 FE FE    LBRA  $8495

8597: 0B          Illegal Opcode
```

```
8598: 20 02       BRA   $859C

859A: 00 16       NEG   $16
859C: FC 02 0B    LDD   $020B
859F: 12          NOP
85A0: 0D 01       TST   $01
85A2: 7E 6F 07    JMP   $6F07

85A5: 10 C0       Illegal Opcode
85A7: 15          Illegal Opcode
85A8: 85 B1       BITA  #$B1
85AA: 04 1C       LSR   $1C
85AC: 06 10       ROR   $10
85AE: 04 36       LSR   $36
85B0: 0C 0B       INC   $0B
85B2: 7C 16 FE    INC   $16FE
85B5: 00 0B       NEG   $0B
85B7: 0B          Illegal Opcode
85B8: 14          Illegal Opcode
85B9: 0A 85       DEC   $85
85BB: B3 17 01    SUBD  $1701
85BE: 7E A9 07    JMP   $A907

85C1: 88 87       EORA  #$87
85C3: 11 85       Illegal Opcode
85C5: D6 11       LDB   $11
85C7: 85 E3       BITA  #$E3
85C9: 11 86       Illegal Opcode
85CB: 03 11       COM   $11
85CD: 86 1A       LDA   #$1A
85CF: 08 FF       ASL   $FF
85D1: 40          NEGA
85D2: 0A 02       DEC   $02
85D4: 60 0C       NEG   $000C,X
85D6: 01          Illegal Opcode
85D7: 7E A5 07    JMP   $A507

85DA: 0A B4       DEC   $B4
85DC: 08 00       ASL   $00
85DE: C0 0A       SUBB  #$0A
85E0: 02          Illegal Opcode
85E1: 60 0C       NEG   $000C,X
85E3: 01          Illegal Opcode
85E4: 7E AD 07    JMP   $AD07

85E7: 10 B4       Illegal Opcode
85E9: 0F 0B       CLR   $0B
85EB: 10 10       Illegal Opcode
85ED: 0B          Illegal Opcode
85EE: 14          Illegal Opcode
85EF: 16 00 FF    LBRA  $86F1

85F2: 18          Illegal Opcode
85F3: 14          Illegal Opcode
```

```
85F4: 03 85       COM   $85
85F6: ED 0F       STD   $000F,X
85F8: 01          Illegal Opcode
85F9: 7E B1 10    JMP   $B110

85FC: 08 00       ASL   $00
85FE: 80 0A       SUBA  #$0A
8600: 02          Illegal Opcode
8601: 70 0D 01    NEG   $0D01
8604: 7E 93 07    JMP   $9307

8607: 80 87       SUBA  #$87
8609: 0F 0B       CLR   $0B
860B: 10 10       Illegal Opcode
860D: 0B          Illegal Opcode
860E: 14          Illegal Opcode
860F: 18          Illegal Opcode
8610: 14          Illegal Opcode
8611: 05          Illegal Opcode
8612: 86 0D       LDA   #$0D
8614: 08 FF       ASL   $FF
8616: A0 0B       SUBA  $000B,X
8618: 30 0D       LEAX  $000D,X
861A: 01          Illegal Opcode
861B: 7E 41 07    JMP   $4107

861E: 0A B4       DEC   $B4
8620: 0F 0B       CLR   $0B
8622: C0 10       SUBB  #$10
8624: 04 20       LSR   $20
8626: 11 86       Illegal Opcode
8628: 8D 05       BSR   $862F
862A: 0A 03       DEC   $03
862C: 08 04       ASL   $04
862E: 1A 16       ORCC  #$16
8630: 03 FE       COM   $FE
8632: 0B          Illegal Opcode
8633: 02          Illegal Opcode
8634: 15          Illegal Opcode
8635: 86 8C       LDA   #$8C
8637: 19          DAA
8638: AA BB       ORA   [D,Y]
863A: 28 00       BVC   $863C
863C: 11 86       Illegal Opcode
863E: 48          ASLA
863F: 08 02       ASL   $02
8641: 00 19       NEG   $19
8643: 00 AA       NEG   $AA
8645: 2C 00       BGE   $8647
8647: 0C 01       INC   $01
8649: 7E 41 07    JMP   $4107

864C: 31 BC 02    LEAY  [$02,Y]
864F: 03 0F       COM   $0F
```

```
8651: 0B          Illegal Opcode
8652: 03 11       COM   $11
8654: 86 66       LDA   #$66
8656: 11 86       Illegal Opcode
8658: 7B          Illegal Opcode
8659: 0B          Illegal Opcode
865A: 04 10       LSR   $10
865C: 08 02       ASL   $02
865E: 00 19       NEG   $19
8660: EE 00       LDU   $0000,X
8662: 2C 00       BGE   $8664
8664: 00 0C       NEG   $0C
8666: 01          Illegal Opcode
8667: 7E 41 07    JMP   $4107

866A: 31 BC 0F    LEAY  [$0F,Y]
866D: 0B          Illegal Opcode
866E: 04 02       LSR   $02
8670: 03 10       COM   $10
8672: 08 02       ASL   $02
8674: 00 19       NEG   $19
8676: EE 00       LDU   $0000,X
8678: 2C 00       BGE   $867A
867A: 0C 01       INC   $01
867C: 7E 41 07    JMP   $4107

867F: 31 BC 02    LEAY  [$02,Y]
8682: 03 08       COM   $08
8684: 02          Illegal Opcode
8685: 00 19       NEG   $19
8687: EE 00       LDU   $0000,X
8689: 2C 00       BGE   $868B
868B: 0C 1A       INC   $1A
868D: 01          Illegal Opcode
868E: 7E 7D 07    JMP   $7D07

8691: 0A A0       DEC   $A0
8693: 04 10       LSR   $10
8695: 11 86       Illegal Opcode
8697: A4 05       ANDA  $0005,X
8699: 0C 04       INC   $04
869B: 10 19       Illegal Opcode
869D: BB BB 38    ADDA  $BB38
86A0: 01          Illegal Opcode
86A1: 04 0B       LSR   $0B
86A3: 0D 01       TST   $01
86A5: 86 B0       LDA   #$B0
86A7: 07 1A       ASR   $1A
86A9: A4 08       ANDA  $0008,X
86AB: 00 60       NEG   $60
86AD: 0B          Illegal Opcode
86AE: 40          NEGA
86AF: 0C 86       INC   $86
86B1: B4 01 04    ANDA  $0104
```

```
86B4: 86 B6       LDA   #$B6
86B6: 09 02       ROL   $02
86B8: 86 BA       LDA   #$BA
86BA: DD DD       STD   $DD
86BC: DD DD       STD   $DD
86BE: DD DD       STD   $DD
86C0: DD DA       STD   $DA
86C2: A0 DD DD DD SUBA  [$DDDD,U]
86C6: DD DD       STD   $DD
86C8: DD DD       STD   $DD
86CA: DA A0       ORB   $A0
86CC: 01          Illegal Opcode
86CD: 7E 8F 07    JMP   $8F07

86D0: 86 C1       LDA   #$C1
86D2: 02          Illegal Opcode
86D3: 0C 0F       INC   $0F
86D5: 0B          Illegal Opcode
86D6: 48          ASLA
86D7: 11 86       Illegal Opcode
86D9: ED 0B       STD   $000B,X
86DB: 0A 11       DEC   $11
86DD: 86 F9       LDA   #$F9
86DF: 0B          Illegal Opcode
86E0: 0A 11       DEC   $11
86E2: 87          Illegal Opcode
86E3: 07 0B       ASR   $0B
86E5: 10 10       Illegal Opcode
86E7: 19          DAA
86E8: 00 DD       NEG   $DD
86EA: B8 00 1B    EORA  $001B
86ED: 01          Illegal Opcode
86EE: 7E 8F 07    JMP   $8F07

86F1: 68 C1       ASL   ,U++
86F3: 19          DAA
86F4: 00 FF       NEG   $FF
86F6: 8D 00       BSR   $86F8
86F8: 1B          Illegal Opcode
86F9: 01          Illegal Opcode
86FA: 7E 8F 07    JMP   $8F07

86FD: 72          Illegal Opcode
86FE: C1 02       CMPB  #$02
8700: 04 19       LSR   $19
8702: 00 AA       NEG   $AA
8704: 91 00       CMPA  $00
8706: 1B          Illegal Opcode
8707: 01          Illegal Opcode
8708: 7E 8F 07    JMP   $8F07

870B: 7C C1 02    INC   $C102
870E: 08 19       ASL   $19
8710: 00 CC       NEG   $CC
```

```
8712: 92 00       SBCA  $00
8714: 1B          Illegal Opcode
8715: 01          Illegal Opcode
8716: 7E 41 07    JMP   $4107

8719: 50          NEGB
871A: 68 11       ASL   $FFF1,X
871C: 87          Illegal Opcode
871D: 39          RTS

871E: 11 87       Illegal Opcode
8720: 42          Illegal Opcode
8721: 11 87       Illegal Opcode
8723: 5B          Illegal Opcode
8724: 11 87       Illegal Opcode
8726: 74 11 87    LSR   $1187
8729: 8D 04       BSR   $872F
872B: 09 0F       ROL   $0F
872D: 01          Illegal Opcode
872E: 7E B5 02    JMP   $B502

8731: 00 16       NEG   $16
8733: 00 05       NEG   $05
8735: 10 0B       Illegal Opcode
8737: 50          NEGB
8738: 0C 01       INC   $01
873A: 7E 97 07    JMP   $9707

873D: 68 6A       ASL   $000A,S
873F: 03 80       COM   $80
8741: 0C 01       INC   $01
8743: 7E 53 07    JMP   $5307

8746: 48          ASLA
8747: 69 0F       ROL   $000F,X
8749: 0B          Illegal Opcode
874A: 10 10       Illegal Opcode
874C: 04 0A       LSR   $0A
874E: 0F 01       CLR   $01
8750: 7E B5 02    JMP   $B502

8753: 01          Illegal Opcode
8754: 16 00 04    LBRA  $875B

8757: 10 0B       Illegal Opcode
8759: 50          NEGB
875A: 0C 01       INC   $01
875C: 7E 61 07    JMP   $6107

875F: 40          NEGA
8760: 6B          Illegal Opcode
8761: 0F 0B       CLR   $0B
8763: 20 10       BRA   $8775
```

```
8765: 04 0C       LSR   $0C
8767: 0F 01       CLR   $01
8769: 7E B5 02    JMP   $B502

876C: 02          Illegal Opcode
876D: 16 00 02    LBRA  $8772

8770: 10 0B       Illegal Opcode
8772: 50          NEGB
8773: 0C 01       INC   $01
8775: 7E 41 07    JMP   $4107

8778: 38          Illegal Opcode
8779: 68 0F       ASL   $000F,X
877B: 0B          Illegal Opcode
877C: 30 10       LEAX  $FFF0,X
877E: 04 0D       LSR   $0D
8780: 0F 01       CLR   $01
8782: 7E B5 02    JMP   $B502

8785: 03 16       COM   $16
8787: 00 05       NEG   $05
8789: 10 0B       Illegal Opcode
878B: 50          NEGB
878C: 0C 01       INC   $01
878E: 7E 61 07    JMP   $6107

8791: 30 6B       LEAX  $000B,S
8793: 0F 0B       CLR   $0B
8795: 40          NEGA
8796: 10 04       Illegal Opcode
8798: 0F 0F       CLR   $0F
879A: 01          Illegal Opcode
879B: 7E B5 02    JMP   $B502

879E: 04 16       LSR   $16
87A0: 00 02       NEG   $02
87A2: 10 0B       Illegal Opcode
87A4: 50          NEGB
87A5: 0C BD       INC   $BD
87A7: D0 30       SUBB  $30
87A9: 96 59       LDA   $59
87AB: 8A 04       ORA   #$04
87AD: 97 59       STA   $59
87AF: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
87B2: 78 D6       ; pointer to function
87B4: BD 8A 3A    JSR   $8A3A
87B7: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
87BA: 87 F8       ; pointer to function
87BC: BD 78 6F    JSR   $786F
87BF: 26 15       BNE   $87D6
87C1: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
```

```
object metadata entry and call function
87C4: 8A 4F       ; pointer to function
87C6: 86 FF       LDA   #$FF
87C8: 8E 87 CE    LDX   #$87CE
87CB: 7E D0 66    JMP   $D066              ; JMP $D1E3 - allocate
function call

87CE: 86 FF       LDA   #$FF
87D0: 8E 79 9B    LDX   #$799B
87D3: 7E D0 66    JMP   $D066              ; JMP $D1E3 - allocate
function call


87D6: BD 88 BF    JSR   $88BF
87D9: 86 1C       LDA   #$1C               ; number of "W" logos on
screen to move round (28 decimal)
87DB: DE 15       LDU   $15
87DD: A7 47       STA   $0007,U
87DF: 9E E6       LDX   $E6                ; load X with blitter
destination
87E1: 96 E9       LDA   $E9                ; A = colour to draw W
in
87E3: BD 8A 19    JSR   $8A19              ; draw the williams logo
in the given colour
87E6: BD 89 6C    JSR   $896C
87E9: 86 04       LDA   #$04
87EB: 8E 87 F1    LDX   #$87F1
87EE: 7E D0 66    JMP   $D066              ; JMP $D1E3 - allocate
function call

87F1: 6A 47       DEC   $0007,U            ; decrement countdown of
"W" left to move
87F3: 26 EA       BNE   $87DF
87F5: 7E 88 CD    JMP   $88CD

87F8: BD 6F 06    JSR   $6F06
87FB: 27 13       BEQ   $8810
87FD: BE 6F 0F    LDX   def_wel_msg_ptr
8800: 10 8E B3 EA LDY   #hs_inits
8804: EC 81       LDD   ,X++
8806: ED A1       STD   ,Y++
8808: 10 8C B4 1E CMPY  #$B41E
880C: 25 F6       BCS   $8804
880E: 20 12       BRA   $8822

8810: 8E CC 24    LDX   #$CC24
8813: 10 8E B3 EA LDY   #hs_inits
8817: BD D0 A8    JSR   $D0A8
881A: ED A1       STD   ,Y++
881C: 10 8C B4 1E CMPY  #$B41E
8820: 25 F5       BCS   $8817
8822: B6 B4 1C    LDA   $B41C
8825: C6 86       LDB   #$86
8827: 10 8E B3 EA LDY   #hs_inits
```

```
882B: 8D 15       BSR   $8842
882D: B6 B4 1D    LDA   $B41D
8830: C6 96       LDB   #$96
8832: 10 8E B4 03 LDY   #$B403
8836: 8D 0A       BSR   $8842
8838: D6 51       LDB   $51
883A: 86 70       LDA   #$70
883C: BD 5F 96    JSR   $5F96                 ; JMP $613F: print
string in small font
883F: 7E D0 63    JMP   $D063                 ; JMP $D1F3

8842: 35 10       PULS  X
8844: DE 15       LDU   $15
8846: AF 47       STX   $0007,U
8848: 1F 01       TFR   D,X
884A: 86 19       LDA   #$19
884C: A7 49       STA   $0009,U
884E: 86 66       LDA   #$66
8850: 97 CF       STA   $CF
8852: A6 A0       LDA   ,Y+
8854: BD 5F 93    JSR   $5F93
8857: AF 4A       STX   $000A,U
8859: 10 AF 4C    STY   $000C,U
885C: 86 02       LDA   #$02
885E: 8E 88 64    LDX   #$8864
8861: 7E D0 66    JMP   $D066                 ; JMP $D1E3 -
allocate function call

8864: AE 4A       LDX   $000A,U
8866: 10 AE 4C    LDY   $000C,U
8869: 6A 49       DEC   $0009,U
886B: 26 E5       BNE   $8852
886D: 6E D8 07    JMP   [$07,U]

8870:  ROBOTRON: 2084  COPYRIGHT 1982
8890: WILLIAMS ELECTRONICS INC.  ALL R
88B0: IGHTS RESERVED

88BF: BD 89 4B    JSR   $894B
88C2: 10 8E B4 26 LDY   #$B426              ; memory destination to
copy Williams Logo Template, once rendered, to
88C6: 86 10       LDA   #$10                ; Colour 1 = 1, Colour 2
= 0
88C8: 8D 6A       BSR   $8934               ; Draw the Williams W
Logo Template
88CA: 7E 89 4B    JMP   $894B

88CD: 8D F0       BSR   $88BF
88CF: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
88D2: 8A 4F       ; pointer to function
88D4: DE 15       LDU   $15
88D6: 8E 02 C0    LDX   #$02C0
88D9: AF 4D       STX   $000D,U
```

```
88DB: 86 06       LDA   #$06
88DD: A7 4B       STA   $000B,U
88DF: 9E E6       LDX   $E6
88E1: 96 E9       LDA   $E9
88E3: 20 0A       BRA   $88EF

88E5: 86 06       LDA   #$06
88E7: A7 4B       STA   $000B,U
88E9: BD 89 6C    JSR   $896C
88EC: BD 89 E6    JSR   $89E6
88EF: 10 9E EB    LDY   $EB
88F2: F6 CB 00    LDB   vidctrs
88F5: D7 2B       STB   $2B
88F7: E1 21       CMPB  $0001,Y
88F9: 25 0E       BCS   $8909
88FB: E0 21       SUBB  $0001,Y
88FD: C1 14       CMPB  #$14
88FF: 23 EE       BLS   $88EF
8901: D6 2B       LDB   $2B
8903: C1 EC       CMPB  #$EC
8905: 24 E8       BCC   $88EF
8907: 20 0D       BRA   $8916

8909: E6 21       LDB   $0001,Y
890B: C1 1E       CMPB  #$1E
890D: 23 E0       BLS   $88EF
890F: F0 CB 00    SUBB  vidctrs
8912: C1 1E       CMPB  #$1E
8914: 23 D9       BLS   $88EF
8916: BD 89 F8    JSR   $89F8
8919: BD 8A 19    JSR   $8A19
891C: DE 15       LDU   $15
891E: 6A 4B       DEC   $000B,U
8920: 26 C7       BNE   $88E9
8922: AE 4D       LDX   $000D,U
8924: 30 1F       LEAX  $-1,X
8926: 10 27 F0 71 LBEQ  $799B
892A: AF 4D       STX   $000D,U
892C: 86 01       LDA   #$01
892E: 8E 88 E5    LDX   #$88E5
8931: 7E D0 66    JMP   $D066              ; JMP $D1E3 - allocate
function call



; Draws the first "W" on the screen, which is then used as a
template for the others.
;
; $E6 = position on screen to draw item
; A = colours to render item in. Left nibble = colour 0, right
nibble = colour 1
; Y = memory destination to copy the rendered template to - RAW.
;
```

```
DRAW_WILLIAMS_LOGO_TEMPLATE:
8934: 34 22       PSHS  Y,A
8936: 9E E6       LDX   $E6                 ; get position on screen
to draw item
8938: 35 02       PULS  A                   ; hmm, why did they do
this? A hasn't been changed since $8934...
893A: 34 10       PSHS  X
893C: 5F          CLRB
893D: CE 8C F4    LDU   #$8CF4              ; pointer to
instructions on how to render graphic
8940: BD 8D 69    JSR   $8D69               ; Call RENDER_GRAPHIC to
Draw the large "W" for WILLIAMS logo
8943: 35 30       PULS  X,Y
8945: CC 0E 1B    LDD   #$0E1B              ; width and height
8948: 7E D0 B7    JMP   $D0B7               ; JMP $DE59 :
COPY_FROM_SCREEN_RAM_TO_RAM



894B: 34 16       PSHS  X,B,A
894D: 86 05       LDA   #$05
894F: C6 0F       LDB   #$0F
8951: DD E6       STD   $E6                 ; blitter destination of
"W"
8953: 0F E8       CLR   $E8
8955: 0F EA       CLR   $EA
8957: 86 77       LDA   #$77                ; colour to draw "W" in
8959: 97 E9       STA   $E9
895B: 8E B5 AE    LDX   #$B5AE              ; pointer to list of "W"
logos to move around the screen
895E: 9F EB       STX   $EB
8960: CC 13 AF    LDD   #$13AF
8963: ED 81       STD   ,X++
8965: 8C B5 E6    CMPX  #$B5E6
8968: 25 F9       BCS   $8963
896A: 35 96       PULS  A,B,X,PC ;(PUL? PC=RTS)

896C: 34 04       PSHS  B
896E: 96 E8       LDA   $E8
8970: 84 03       ANDA  #$03
8972: 4A          DECA
8973: 2B 4E       BMI   $89C3
8975: 4A          DECA
8976: 2B 3B       BMI   $89B3
8978: 4A          DECA
8979: 2B 24       BMI   $899F

; if we get here, the "W" in question is moving UP.
897B: DC E6       LDD   $E6                 ; load D with blitter
destination of "W"
897D: C0 20       SUBB  #$20                ; Y component of screen
destination -= #$20 (32 decimal) , moving 32 pixels UP
897F: 25 04       BCS   $8985
```

```
8981: C1 0F       CMPB  #$0F
8983: 22 4F       BHI   $89D4
8985: 96 EA       LDA   $EA
8987: 8B 02       ADDA  #$02
8989: 81 10       CMPA  #$10
898B: 25 08       BCS   $8995
898D: 0F EA       CLR   $EA
898F: 86 15       LDA   #$15
8991: C6 0F       LDB   #$0F
8993: 20 3D       BRA   $89D2

8995: 97 EA       STA   $EA
8997: C6 0F       LDB   #$0F
8999: 86 05       LDA   #$05
899B: 9B EA       ADDA  $EA
899D: 20 33       BRA   $89D2

; the "W" is moving LEFT.
899F: DC E6       LDD   $E6                ; load D with blitter
destination of "W"
89A1: 80 10       SUBA  #$10               ; X component of screen
destination -= #$10 (16 decimal) - remember 2 pixels per byte, so
moving 32 pixels LEFT
89A3: 25 04       BCS   $89A9
89A5: 81 05       CMPA  #$05
89A7: 22 2B       BHI   $89D4
89A9: 86 05       LDA   #$05
89AB: D6 EA       LDB   $EA
89AD: 50          NEGB
89AE: 58          ASLB
89AF: CB CF       ADDB  #$CF
89B1: 20 1F       BRA   $89D2

; the "W" is moving DOWN
89B3: DC E6       LDD   $E6                ; load D with blitter
destination of "W"
89B5: CB 20       ADDB  #$20               ; Y component of screen
destination += #$20 (32 decimal) , moving 32 pixels DOWN
89B7: C1 CF       CMPB  #$CF
89B9: 25 19       BCS   $89D4
89BB: C6 CF       LDB   #$CF
89BD: 86 85       LDA   #$85
89BF: 90 EA       SUBA  $EA
89C1: 20 0F       BRA   $89D2

; the "W" is moving RIGHT
89C3: DC E6       LDD   $E6                ; load D with blitter
destination of "W"
89C5: 8B 10       ADDA  #$10               ; ; X component of
screen destination += #$10 (16 decimal) - remember 2 pixels per
byte, so moving 32 pixels RIGHT
89C7: 81 85       CMPA  #$85
89C9: 25 09       BCS   $89D4
89CB: 86 85       LDA   #$85
```

```
89CD: D6 EA       LDB   $EA
89CF: 58          ASLB
89D0: CB 0F       ADDB  #$0F

;
; D= blitter destination
;
89D2: 0C E8       INC   $E8
89D4: DD E6       STD   $E6                 ; update blitter
destination of "W"
89D6: 1F 01       TFR   D,X
89D8: 96 E9       LDA   $E9                 ; get colour to draw "W"
in. 2 colours are packed into this byte, 1 colour per nibble, but
both colours are the same.
89DA: 80 11       SUBA  #$11                ; subtract 1 from each
nibble (ie: left nibble = left nibble - 1, right nibble = right
nibble -1)
89DC: 25 02       BCS   $89E0               ; if a carry occurred,
goto $89E0, reset colour of W back to $77
89DE: 26 02       BNE   $89E2               ;
89E0: 86 77       LDA   #$77                ; colour
89E2: 97 E9       STA   $E9                 ; set colour to draw "W"
in
89E4: 35 84       PULS  B,PC ;(PUL? PC=RTS)

89E6: 34 10       PSHS  X
89E8: 9E EB       LDX   $EB                 ; get pointer to current
"W" logo screen coordinates
89EA: 30 02       LEAX  $0002,X             ; X+= 2, X now points to
next "W"'s screen coordinates
89EC: 8C B5 E6    CMPX  #$B5E6              ; have we done all the
"W" logos?
89EF: 25 03       BCS   $89F4               ; no, goto $89F4 to
update logo pointer
89F1: 8E B5 AE    LDX   #$B5AE              ; yes, all logos have
been done, reset pointer to first "W"
89F4: 9F EB       STX   $EB
89F6: 35 90       PULS  X,PC ;(PUL? PC=RTS)



CLEAR_WILLIAMS_LOGO:
89F8: 34 17       PSHS  X,B,A,CC
89FA: AE 9F 98 EB LDX   [$98EB]
89FE: 1A 10       ORCC  #$10                ; disable interrupts
8A00: BF CA 04    STX   blitter_dest
8A03: CC 0A 1F    LDD   #$0A1F
8A06: FD CA 06    STD   blitter_w_h
8A09: 7F CA 01    CLR   blitter_mask
8A0C: CC B4 26    LDD   #$B426              ; see $88C2
8A0F: FD CA 02    STD   blitter_source
8A12: 86 1E       LDA   #$1E                ; sync with E clock
(we're blitting from RAM to RAM), transparency mode, solid mode
8A14: B7 CA 00    STA   start_blitter
```

```
8A17: 35 97       PULS  CC,A,B,X,PC ;(PUL? PC=RTS)



;
; This routine is responsible for drawing the "W" williams logo in
attract mode.
;
; A = colour to draw W in
; X = screen destination
;

BLIT_WILLIAMS_LOGO:
8A19: 34 03       PSHS  A,CC
8A1B: AF 9F 98 EB STX   [$98EB]
8A1F: 1A 10       ORCC  #$10                ; disable interrupts
8A21: B7 CA 01    STA   blitter_mask
8A24: BF CA 04    STX   blitter_dest
8A27: CC 0A 1F    LDD   #$0A1F
8A2A: FD CA 06    STD   blitter_w_h
8A2D: CC B4 26    LDD   #$B426
8A30: FD CA 02    STD   blitter_source
8A33: 86 1E       LDA   #$1E                ; sync with E clock
(we're blitting from RAM to RAM), transparency mode, solid mode
8A35: B7 CA 00    STA   start_blitter
8A38: 35 83       PULS  CC,A,PC ;(PUL? PC=RTS)


8A3A: 34 32       PSHS  Y,X,A
8A3C: 8E 8A 70    LDX   #$8A70
8A3F: 10 8E 98 01 LDY   #$9801
8A43: A6 80       LDA   ,X+
8A45: A7 A0       STA   ,Y+
8A47: 10 8C 98 08 CMPY  #$9808
8A4B: 25 F6       BCS   $8A43
8A4D: 35 B2       PULS  A,X,Y,PC ;(PUL? PC=RTS)

8A4F: 8E 98 01    LDX   #$9801
8A52: AF 49       STX   $0009,U
8A54: 8D E4       BSR   $8A3A
8A56: AE 49       LDX   $0009,U
8A58: 30 01       LEAX  $0001,X
8A5A: 8C 98 08    CMPX  #$9808
8A5D: 25 03       BCS   $8A62
8A5F: 8E 98 01    LDX   #$9801
8A62: AF 49       STX   $0009,U
8A64: 86 FF       LDA   #$FF
8A66: A7 84       STA   ,X
8A68: 86 03       LDA   #$03
8A6A: 8E 8A 54    LDX   #$8A54
8A6D: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

8A70: 07 C0       ASR   $C0
```

```
8A72: 17 30 C7    LBSR  $BB3C
8A75: 1F 3F       TFR   U,inv



;
; Data here
;
;

8A77: 01          Illegal Opcode
8A78: 39          RTS

8A79: 90 22       SUBA  $22
8A7B: 57          ASRB
8A7C: 22 90       BHI   $8A0E
8A7E: 21 59       BRN   $8AD9
8A80: 21 90       BRN   $8A12
8A82: 21 43       BRN   $8AC7
8A84: 27 4F       BEQ   $8AD5
8A86: 21 90       BRN   $8A18
8A88: 21 43       BRN   $8ACD
8A8A: 21 05       BRN   $8A91
8A8C: 22 4D       BHI   $8ADB
8A8E: 22 90       BHI   $8A20
8A90: 21 43       BRN   $8AD5
8A92: 21 06       BRN   $8A9A
8A94: 21 42       BRN   $8AD8
8A96: 2C 90       BGE   $8A28
8A98: 21 43       BRN   $8ADD
8A9A: 21 06       BRN   $8AA2
8A9C: 21 42       BRN   $8AE0
8A9E: 21 90       BRN   $8A30
8AA0: C6 21       LDB   #$21
8AA2: 43          COMA
8AA3: 22 05       BHI   $8AAA
8AA5: 21 42       BRN   $8AE9
8AA7: 21 90       BRN   $8A39
8AA9: 21 44       BRN   $8AEF
8AAB: 27 42       BEQ   $8AEF
8AAD: 21 90       BRN   $8A3F
8AAF: 21 4D       BRN   $8AFE
8AB1: 21 90       BRN   $8A43
8AB3: 23 4B       BLS   $8B00
8AB5: 22 90       BHI   $8A47
8AB7: 02          Illegal Opcode
8AB8: 2A 43       BPL   $8AFD
8ABA: 2B 90       BMI   $8A4C
8ABC: 0A 22       DEC   $22
8ABE: 4D          TSTA
8ABF: 22 90       BHI   $8A51
8AC1: 0B          Illegal Opcode
8AC2: 22 4C       BHI   $8B10
8AC4: 22 90       BHI   $8A56
```

```
8AC6: 0C 2E       INC   $2E
8AC8: 90 A0       SUBA  $A0
8ACA: 01          Illegal Opcode
8ACB: 39          RTS

8ACC: 90 22       SUBA  $22
8ACE: 57          ASRB
8ACF: 22 90       BHI   $8A61
8AD1: 21 59       BRN   $8B2C
8AD3: 21 90       BRN   $8A65
8AD5: 21 43       BRN   $8B1A
8AD7: 33 43       LEAU  $0003,U
8AD9: 21 90       BRN   $8A6B
8ADB: 21 42       BRN   $8B1F
8ADD: 22 11       BHI   $8AF0
8ADF: 22 42       BHI   $8B23
8AE1: 21 90       BRN   $8A73
8AE3: 21 42       BRN   $8B27
8AE5: 21 13       BRN   $8AFA
8AE7: 21 42       BRN   $8B2B
8AE9: 21 90       BRN   $8A7B
8AEB: C8 21       EORB  #$21
8AED: 42          Illegal Opcode
8AEE: 24 10       BCC   $8B00
8AF0: 21 42       BRN   $8B34
8AF2: 21 90       BRN   $8A84
8AF4: 21 45       BRN   $8B3B
8AF6: 22 0F       BHI   $8B07
8AF8: 21 42       BRN   $8B3C
8AFA: 21 90       BRN   $8A8C
8AFC: 21 46       BRN   $8B44
8AFE: 22 0D       BHI   $8B0D
8B00: 22 42       BHI   $8B44
8B02: 21 90       BRN   $8A94
8B04: 21 47       BRN   $8B4D
8B06: 2F 43       BLE   $8B4B
8B08: 21 90       BRN   $8A9A
8B0A: 21 59       BRN   $8B65
8B0C: 21 90       BRN   $8A9E
8B0E: 22 57       BHI   $8B67
8B10: 22 90       BHI   $8AA2
8B12: 01          Illegal Opcode
8B13: 39          RTS

8B14: 90 A0       SUBA  $A0
8B16: 01          Illegal Opcode
8B17: 39          RTS

8B18: 90 22       SUBA  $22
8B1A: 57          ASRB
8B1B: 21 90       BRN   $8AAD
8B1D: 21 58       BRN   $8B77
8B1F: 22 90       BHI   $8AB1
8B21: 21 42       BRN   $8B65
```

```
8B23: 27 50       BEQ   $8B75
8B25: 21 90       BRN   $8AB7
8B27: 21 42       BRN   $8B6B
8B29: 21 05       BRN   $8B30
8B2B: 22 4F       BHI   $8B7C
8B2D: 21 90       BRN   $8ABF
8B2F: 21 42       BRN   $8B73
8B31: 21 06       BRN   $8B39
8B33: 21 4F       BRN   $8B84
8B35: 21 90       BRN   $8AC7
8B37: 21 42       BRN   $8B7B
8B39: 21 06       BRN   $8B41
8B3B: 21 43       BRN   $8B80
8B3D: 29 43       BVS   $8B82
8B3F: 21 90       BRN   $8AD1
8B41: 21 42       BRN   $8B85
8B43: 21 06       BRN   $8B4B
8B45: 21 42       BRN   $8B89
8B47: 22 07       BHI   $8B50
8B49: 22 42       BHI   $8B8D
8B4B: 21 90       BRN   $8ADD
8B4D: 21 42       BRN   $8B91
8B4F: 21 06       BRN   $8B57
8B51: 21 42       BRN   $8B95
8B53: 21 09       BRN   $8B5E
8B55: 21 42       BRN   $8B99
8B57: 21 90       BRN   $8AE9
8B59: C5 21       BITB  #$21
8B5B: 42          Illegal Opcode
8B5C: 22 05       BHI   $8B63
8B5E: 21 42       BRN   $8BA2
8B60: 21 09       BRN   $8B6B
8B62: 21 42       BRN   $8BA6
8B64: 21 90       BRN   $8AF6
8B66: 21 43       BRN   $8BAB
8B68: 27 42       BEQ   $8BAC
8B6A: 21 09       BRN   $8B75
8B6C: 21 42       BRN   $8BB0
8B6E: 21 90       BRN   $8B00
8B70: 21 4C       BRN   $8BBE
8B72: 21 09       BRN   $8B7D
8B74: 21 42       BRN   $8BB8
8B76: 21 90       BRN   $8B08
8B78: 22 4B       BHI   $8BC5
8B7A: 22 07       BHI   $8B83
8B7C: 22 42       BHI   $8BC0
8B7E: 21 90       BRN   $8B10
8B80: 01          Illegal Opcode
8B81: 2A 43       BPL   $8BC6
8B83: 29 43       BVS   $8BC8
8B85: 21 90       BRN   $8B17
8B87: 0A 21       DEC   $21
8B89: 4E          Illegal Opcode
8B8A: 22 90       BHI   $8B1C
```

```
8B8C: 0A 22       DEC   $22
8B8E: 4B          Illegal Opcode
8B8F: 23 90       BLS   $8B21
8B91: 0B          Illegal Opcode
8B92: 2D 90       BLT   $8B24
8B94: A0 01       SUBA  $0001,X
8B96: 23 90       BLS   $8B28
8B98: 22 41       BHI   $8BDB
8B9A: 22 90       BHI   $8B2C
8B9C: 21 43       BRN   $8BE1
8B9E: 21 90       BRN   $8B30
8BA0: C4 21       ANDB  #$21
8BA2: 43          COMA
8BA3: 22 90       BHI   $8B35
8BA5: 21 44       BRN   $8BEB
8BA7: 35 90       PULS  X,PC ;(PUL? PC=RTS)

8BA9: 21 58       BRN   $8C03
8BAB: 22 90       BHI   $8B3D
8BAD: 21 59       BRN   $8C08
8BAF: 21 90       BRN   $8B41
8BB1: C1 21       CMPB  #$21
8BB3: 44          LSRA
8BB4: 27 4D       BEQ   $8C03
8BB6: 22 90       BHI   $8B48
8BB8: 21 43       BRN   $8BFD
8BBA: 22 05       BHI   $8BC1
8BBC: 2F 90       BLE   $8B4E
8BBE: 21 43       BRN   $8C03
8BC0: 21 90       BRN   $8B52
8BC2: C4 22       ANDB  #$22
8BC4: 41          Illegal Opcode
8BC5: 22 90       BHI   $8B57
8BC7: 01          Illegal Opcode
8BC8: 23 90       BLS   $8B5A
8BCA: A0 01       SUBA  $0001,X
8BCC: 39          RTS

8BCD: 90 22       SUBA  $22
8BCF: 57          ASRB
8BD0: 22 90       BHI   $8B62
8BD2: 21 59       BRN   $8C2D
8BD4: 21 90       BRN   $8B66
8BD6: 21 42       BRN   $8C1A
8BD8: 28 4F       BVC   $8C29
8BDA: 21 90       BRN   $8B6C
8BDC: 21 42       BRN   $8C20
8BDE: 21 06       BRN   $8BE6
8BE0: 21 4F       BRN   $8C31
8BE2: 21 90       BRN   $8B74
8BE4: 21 42       BRN   $8C28
8BE6: 21 06       BRN   $8BEE
8BE8: 22 4C       BHI   $8C36
8BEA: 22 90       BHI   $8B7C
```

```
8BEC: 21 42       BRN   $8C30
8BEE: 21 07       BRN   $8BF7
8BF0: 2E 90       BGT   $8B82
8BF2: 21 42       BRN   $8C36
8BF4: 21 90       BRN   $8B86
8BF6: C8 21       EORB  #$21
8BF8: 42          Illegal Opcode
8BF9: 37 90       PULU  X,PC ;(PUL? PC=RTS)

8BFB: 21 58       BRN   $8C55
8BFD: 22 90       BHI   $8B8F
8BFF: 22 57       BHI   $8C58
8C01: 22 90       BHI   $8B93
8C03: 01          Illegal Opcode
8C04: 39          RTS

8C05: 90 A0       SUBA  $A0
8C07: 03 24       COM   $24
8C09: 0E 24       JMP   $24

8C0B: 90 02       SUBA  $02
8C0D: 22 42       BHI   $8C51
8C0F: 22 0C       BHI   $8C1D
8C11: 22 42       BHI   $8C55
8C13: 22 90       BHI   $8BA5
8C15: 02          Illegal Opcode
8C16: 21 44       BRN   $8C5C
8C18: 21 0C       BRN   $8C26
8C1A: 21 44       BRN   $8C60
8C1C: 21 90       BRN   $8BAE
8C1E: C5 02       BITB  #$02
8C20: 22 42       BHI   $8C64
8C22: 22 0C       BHI   $8C30
8C24: 22 42       BHI   $8C68
8C26: 22 90       BHI   $8BB8
8C28: 03 24       COM   $24
8C2A: 0E 24       JMP   $24

8C2C: 90 A0       SUBA  $A0
8C2E: 01          Illegal Opcode
8C2F: 21 4C       BRN   $8C7D
8C31: 23 4C       BLS   $8C7F
8C33: 21 90       BRN   $8BC5
8C35: 02          Illegal Opcode
8C36: 21 4C       BRN   $8C84
8C38: 21 4C       BRN   $8C86
8C3A: 21 90       BRN   $8BCC
8C3C: 03 21       COM   $21
8C3E: 57          ASRB
8C3F: 21 90       BRN   $8BD1
8C41: 04 21       LSR   $21
8C43: 55          Illegal Opcode
8C44: 21 90       BRN   $8BD6
8C46: 05          Illegal Opcode
```

```
8C47: 21 43       BRN   $8C8C
8C49: 26 41       BNE   $8C8C
8C4B: 26 43       BNE   $8C90
8C4D: 21 90       BRN   $8BDF
8C4F: 06 21       ROR   $21
8C51: 43          COMA
8C52: 21 04       BRN   $8C58
8C54: 21 04       BRN   $8C5A
8C56: 21 43       BRN   $8C9B
8C58: 21 90       BRN   $8BEA
8C5A: 07 21       ASR   $21
8C5C: 43          COMA
8C5D: 21 03       BRN   $8C62
8C5F: 21 03       BRN   $8C64
8C61: 21 43       BRN   $8CA6
8C63: 21 90       BRN   $8BF5
8C65: 08 21       ASL   $21
8C67: 43          COMA
8C68: 23 41       BLS   $8CAB
8C6A: 23 43       BLS   $8CAF
8C6C: 21 90       BRN   $8BFE
8C6E: 09 21       ROL   $21
8C70: 4B          Illegal Opcode
8C71: 21 90       BRN   $8C03
8C73: 0A 21       DEC   $21
8C75: 49          ROLA
8C76: 21 90       BRN   $8C08
8C78: 0B          Illegal Opcode
8C79: 21 43       BRN   $8CBE
8C7B: 21 43       BRN   $8CC0
8C7D: 21 90       BRN   $8C0F
8C7F: 0C 28       INC   $28
8C81: 90 A0       SUBA  $A0
8C83: 0D 21       TST   $21
8C85: 45          Illegal Opcode
8C86: 21 90       BRN   $8C18
8C88: 0C 21       INC   $21
8C8A: 46          RORA
8C8B: 21 90       BRN   $8C1D
8C8D: 0B          Illegal Opcode
8C8E: 21 47       BRN   $8CD7
8C90: 21 90       BRN   $8C22
8C92: 0A 21       DEC   $21
8C94: 48          ASLA
8C95: 21 90       BRN   $8C27
8C97: 09 21       ROL   $21
8C99: 45          Illegal Opcode
8C9A: 21 43       BRN   $8CDF
8C9C: 21 90       BRN   $8C2E
8C9E: 08 21       ASL   $21
8CA0: 45          Illegal Opcode
8CA1: 22 43       BHI   $8CE6
8CA3: 21 90       BRN   $8C35
8CA5: 08 21       ASL   $21
```

```
8CA7: 44          LSRA
8CA8: 21 01       BRN   $8CAB
8CAA: 21 43       BRN   $8CEF
8CAC: 21 90       BRN   $8C3E
8CAE: 08 21       ASL   $21
8CB0: 43          COMA
8CB1: 21 02       BRN   $8CB5
8CB3: 21 43       BRN   $8CF8
8CB5: 21 90       BRN   $8C47
8CB7: 08 21       ASL   $21
8CB9: 43          COMA
8CBA: 24 43       BCC   $8CFF
8CBC: 25 90       BCS   $8C4E
8CBE: 08 21       ASL   $21
8CC0: 4E          Illegal Opcode
8CC1: 21 90       BRN   $8C53
8CC3: C3 08 28    ADDD  #$0828
8CC6: 43          COMA
8CC7: 25 90       BCS   $8C59
8CC9: 0F 21       CLR   $21
8CCB: 43          COMA
8CCC: 21 90       BRN   $8C5E
8CCE: C3 0E 21    ADDD  #$0E21
8CD1: 44          LSRA
8CD2: 21 90       BRN   $8C64
8CD4: C4 0F       ANDB  #$0F
8CD6: 21 43       BRN   $8D1B
8CD8: 21 90       BRN   $8C6A
8CDA: 10 21 42 21 LBRN  $CEFF
8CDE: 90 11       SUBA  $11
8CE0: 21 41       BRN   $8D23
8CE2: 21 90       BRN   $8C74
8CE4: 12          NOP
8CE5: 22 90       BHI   $8C77
8CE7: A0 F0       SUBA  [,S+]
8CE9: 19          DAA
8CEA: F0 1B 8C    SUBB  $1B8C
8CED: F2 8C 2E    SBCB  $8C2E
8CF0: 8C 83 00    CMPX  #$8300
8CF3: 00



;
; Plotting instructions for the Williams "W" Logo
;

8CF4:  0A 47 90 08 4B 90 06 4F 90 05 51 90 04 44 22 4D
8D04:  90 03 47 22 4C 90 02 49 24 4A 90 02 4A 25 48 90
8D14:  01 4C 27 46 90 01 4D 28 44 90 4D 27 47 90 48 22
8D24:  42 26 49 90 49 27 4B 90 4A 24 4D 90 49 27 4B 90
8D34:  48 22 42 26 49 90 4D 27 47 90 01 4D 28 44 90 01
8D44:  4C 27 46 90 02 4A 25 48 90 02 49 24 4A 90 03 47
8D54:  22 4C 90 04 44 22 4D 90 05 51 90 06 4F 90 08 4B
```

```
8D64:  90 0A 47 90 A0 34



8D66: 47          ASRA
8D67: 90 A0       SUBA  $A0



;
; Renders an image, such as the "W" Williams logo, following the
drawing instructions contained at U.
;
; A = packed byte containing 2 colours to render with. Bits 7-4 and
3-0 represent 2 different colours (palette indexes) to use. I will
call them "Colour 1" and "Colour 2" in the disassembly.
;
; U = pointer to data containing instructions on how to draw, and
what colours to use
; X = screen address to start rendering
;


RENDER_GRAPHIC:
; I suggest you skip to $8D85 for the real meat. This is just
setting up 2 colours, in $DF and $E0
8D69: 34 62       PSHS  U,Y,A
8D6B: 84 F0       ANDA  #$F0                ; preserve bits 7-4
8D6D: 34 02       PSHS  A
8D6F: 44          LSRA                      ; shift bits 7-4...
8D70: 44          LSRA
8D71: 44          LSRA
8D72: 44          LSRA                      ; to bits 3..0 . Right
nibble now holds a value from 0-#$0F (15 decimal), left nibble is
cleared.
8D73: AA E0       ORA   ,S+                 ; OR in original value
(pushed at $8D6D) so now left nibble is binary equivalent to the
right nibble. Adjust stack to discard item pushed at $8D6D
8D75: 97 DF       STA   $DF                 ; store in Colour 1
field
8D77: A6 E4       LDA   ,S                  ; read A that was pushed
on stack
8D79: 84 0F       ANDA  #$0F                ; keep bits 3-0, discard
the rest
8D7B: 34 02       PSHS  A
8D7D: 48          ASLA                      ; move bits 3-0...
8D7E: 48          ASLA
8D7F: 48          ASLA
8D80: 48          ASLA                      ; to bits 7-4.   Left
nibble is set. Right nibble is clear.
8D81: AA E0       ORA   ,S+                 ; OR in original value
so now right nibble is binary equivalent to the left nibble. Adjust
stack to discard item pushed at $8D7B
8D83: 97 E0       STA   $E0                 ; store in Colour 2
```

```
field

;
; This code works like the following:
; REPEAT
;     Gosub PROCESS_SET_OF_RENDER_INSTRUCTIONS
;     U now points to next list of render instructions, to do
another part of the graphic being built up
; UNTIL carry flag set

8D85: 8D 04       BSR   $8D8B             ; process a set of
instructions
8D87: 24 FC       BCC   $8D85             ; if carry is clear then
there's another set of instructions to process, continue until carry
is set meaning no more instructions.
8D89: 35 E2       PULS  A,Y,U,PC ;(PUL? PC=RTS)



;
; Process a set of render instructions.
;
; X = pointer to screen address to render to
; U = pointer to data containing set of render instructions
; $DF = colour 1  (left nibble and right nibble must be binary
equal, otherwise rendering artifacts will occur)
; $E0 = colour 2  (left nibble and right nibble must be binary
equal, otherwise rendering artifacts will occur)
;


PROCESS_SET_OF_RENDER_INSTRUCTIONS:
8D8B: A6 C4       LDA   ,U                ; read instruction at U
into A
8D8D: 43          COMA                    ; flip the bits
8D8E: 85 C0       BITA  #$C0              ; is either bit 6 or bit
7 of the flipped bits set?
8D90: 27 02       BEQ   $8D94             ; no, skip next line
8D92: DF E4       STU   $E4
8D94: 9F DD       STX   $DD               ; save pixel plot start
screen address in $DD (see $8E02 for code that uses it)
8D96: A6 C0       LDA   ,U+               ; read same instruction
at U again, then increment U by a byte, to move to next instruction.
8D98: 2A 27       BPL   $8DC1             ; if bit 7 is not set,
goto $8DC1

;
; if we get here, bit 7 of the instruction is set, meaning this is a
special instruction.
;
; Bit 6 set: repeat instruction set for given count
; Bit 5 set: rendering complete - set carry flag and exit
; Bit 4 set: move to next pixel pair. (remember, in screen memory
layout, 1 byte = 2 pixels.)
```

```
;
8D9A: 85 20       BITA  #$20                ; is bit 5 set?
8D9C: 27 03       BEQ   $8DA1               ; no, goto $8DA1
8D9E: 1A 01       ORCC  #$01                ; yes, set carry flag -
which means the rendering operation is entirely complete. (see
$8D87)
8DA0: 39          RTS                       ; we're done.

8DA1: 85 10       BITA  #$10                ; is bit 4 set?
8DA3: 27 05       BEQ   $8DAA               ; no, goto $8DAA
8DA5: 8D 5B       BSR   $8E02               ; yes, the plotting for
this pixel column has been done, move to next pixel column
8DA7: 1C FE       ANDCC #$FE                ; clear carry flag
(indicating to caller that plotting is not finished)
8DA9: 39          RTS

8DAA: 85 40       BITA  #$40                ; is bit 6 set?
8DAC: 27 E8       BEQ   $8D96               ; no, goto $8D96, get
next instruction
8DAE: 84 0F       ANDA  #$0F                ; mask in right nibble.
Now A = count of times to repeat instruction set
8DB0: 97 E1       STA   $E1                 ; save countdown
variable
8DB2: DF E2       STU   $E2                 ; save pointer to
instruction set
8DB4: DE E4       LDU   $E4                 ; change instruction set
pointer to * $E4 (see $8D92 for info)
8DB6: 8D DC       BSR   $8D94               ; process instructions
8DB8: 0A E1       DEC   $E1                 ; decrement countdown
variable
8DBA: 26 F8       BNE   $8DB4               ; if countdown is not
complete, repeat the instruction set again
8DBC: DE E2       LDU   $E2                 ; otherwise, the
instructions have been repeated as necessary, restore pointer to
instruction set
8DBE: 1C FE       ANDCC #$FE                ; clear carry flag
(indicating to caller that plotting is not finished)
8DC0: 39          RTS


;
; if we get here, we either have a DRAW LINE or CHANGE PLOTTING
START SCREEN ADDRESS command.
;
8DC1: 85 60       BITA  #$60                ; is this a render
vertical line command? (either bit 6 or bit 5 is set)
8DC3: 26 04       BNE   $8DC9               ; yes, goto $8DC9, to
draw line
8DC5: 30 86       LEAX  A,X                 ; otherwise, change
plotting screen address to be X + A, effectively moving A pixels
DOWN from last plot. Remember the Robotron screen memory map.
8DC7: 20 CD       BRA   $8D96               ; and go process next
byte.
```

```
;
; A contains both flags and a line height value.
; Bit 6 of A set = draw line using Colour 1 in $DF
; Bit 5 of A set = draw line using Colour 2 in $E0
;
; Bits 4..0 - these bits contain a value specifying how high, in
pixels, the line is going to be.
;
; X = screen address to draw line *vertically* from.
;
; For example, if A was set to $45 (69 decimal) this would mean use
colour 1, and write 5 pixels (note: single pixels, not pairs of
pixels) starting from screen address X.
; The end result is that a vertical line 1 pixel in width, 5 pixels
in height is drawn.
;

RENDER_VERTICAL_LINE:
8DC9: 34 03       PSHS  A,CC
8DCB: 84 1F       ANDA  #$1F                ; extract height part
from A.
8DCD: 1A 10       ORCC  #$10                ; disable interrupts
8DCF: BF CA 04    STX   blitter_dest
8DD2: 30 86       LEAX  A,X                 ; X (screen address to
blit to) += height
8DD4: 88 04       EORA  #$04
8DD6: B7 CA 07    STA   blitter_height
8DD9: 86 05       LDA   #$05                ; set width (1 pixel, as
it is XORed with 4 by blitter chip)
8DDB: B7 CA 06    STA   blitter_width
8DDE: FD CA 02    STD   blitter_source
8DE1: A6 61       LDA   $0001,S             ; read the value of A
pushed on the stack
8DE3: 85 40       BITA  #$40                ; if bit 6 is set, we
want to draw line using colour 1
8DE5: 27 04       BEQ   $8DEB               ; if bit 6 is not set
goto $8DEB, draw line using colour 2
8DE7: 96 DF       LDA   $DF                 ; bit 6 is set, so use
colour 1
8DE9: 20 02       BRA   $8DED               ; and then draw the
line...

8DEB: 96 E0       LDA   $E0                 ; use colour 2
8DED: B7 CA 01    STA   blitter_mask        ; set colour to draw in
8DF0: 86 12       LDA   #$12                ; solid mode
8DF2: 5D          TSTB                      ; when B = 0, blit to
even pixels *per row* only. When B nonzero, blit to odd pixels *per
row* only.
8DF3: 27 04       BEQ   $8DF9               ; if B=0 goto $8DF9
8DF5: 8A 80       ORA   #$80                ; OR in Blit odd pixels
only flag
8DF7: 20 02       BRA   $8DFB               ; do the line draw
```

```
8DF9: 8A 40       ORA   #$40                ; Blit even pixels only
8DFB: B7 CA 00    STA   start_blitter       ; execute the line draw
8DFE: 35 03       PULS  CC,A
8E00: 20 94       BRA   $8D96               ; go get next render
instruction


MOVE_TO_NEXT_PIXEL:
8E02: 9E DD       LDX   $DD                 ; get X (pixel plot
screen address) from field
8E04: 5D          TSTB                      ; if B is 0, make it #
$FF.... so that odd pixels are blitted to next time
8E05: 27 06       BEQ   $8E0D
8E07: C6 FF       LDB   #$FF                ; else B is not 0, so
needs made to 0... so that even pixels are blitted to next time
8E09: 30 89 01 00 LEAX  $0100,X             ; bump X to point to
next pixel column, but same row, on screen
8E0D: 53          COMB                      ; flip bits in B.
8E0E: 39          RTS

8E0F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8E1F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8E2F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8E3F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8E4F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8E5F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8E6F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8E7F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8E8F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8E9F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8EAF: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8EBF: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8ECF: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8EDF: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8EEF: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8EFF: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8F0F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8F1F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8F2F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8F3F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8F4F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8F5F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8F6F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8F7F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8F8F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8F9F: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8FAF: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8FBF: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8FCF: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8FDF: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8FEF: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
8FFF: FF
```

*** RAM
*** I/O space
9000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
91A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
91B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
91C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
91D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
91E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
91F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9260: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
92A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
92B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
92C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
92D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
92E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
92F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9300: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9310: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```
9320: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9330: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9340: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9350: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9360: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9370: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9380: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9390: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
93A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
93B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
93C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
93D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
93E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
93F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9400: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9410: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9420: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9430: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9440: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9450: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9460: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9470: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9480: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9490: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
94A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
94B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
94C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
94D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
94E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
94F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9500: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9510: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9520: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9530: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9540: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9550: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9560: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9570: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9580: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9590: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
95A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
95B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
95C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
95D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
95E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
95F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9600: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9610: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9620: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9630: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9640: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9650: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9660: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9670: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
9680: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9690: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
96A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
96B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
96C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
96D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
96E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
96F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9700: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9710: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9720: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9730: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9740: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9750: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9760: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9770: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9780: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9790: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
97A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
97B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
97C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
97D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
97E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
97F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9800: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9810: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9820: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9830: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9840: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9850: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9860: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9870: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9880: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9890: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
98A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
98B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
98C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
98D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
98E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
98F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9900: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9910: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9920: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9930: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9940: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9950: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9960: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9970: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9980: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9990: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
99A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
99B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
99C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
99D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
99E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
99F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9A00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9A10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9A20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9A30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9A40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9A50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9A60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9A70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9A80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9A90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9AA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9AB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9AC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9AD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9AE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9AF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9B00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9B10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9B20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9B30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9B40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9B50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9B60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9B70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9B80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9B90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9BA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9BB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9BC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9BD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9BE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9BF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9C00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9C10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9C20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9C30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9C40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9C50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9C60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9C70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9C80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9C90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9CA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9CB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9CC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9CD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9CE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9CF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9D00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9D10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9D20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9D30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
9D40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9D50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9D60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9D70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9D80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9D90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9DA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9DB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9DC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9DD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9DE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9DF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9E00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9E10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9E20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9E30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9E40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9E50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9E60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9E70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9E80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9E90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9EA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9EB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9EC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9ED0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9EE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9EF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9F00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9F10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9F20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9F30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9F40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9F50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9F60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9F70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9F80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9F90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9FA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9FB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9FC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9FD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9FE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9FF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
A0A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A0B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A0C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A0D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A0E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A0F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A1A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A1B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A1C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A1D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A1E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A1F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A260: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A2A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A2B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A2C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A2D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A2E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A2F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A300: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A310: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A320: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A330: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A340: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A350: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A360: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A370: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A380: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A390: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A3A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A3B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A3C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A3D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A3E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A3F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
A400: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A410: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A420: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A430: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A440: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A450: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A460: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A470: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A480: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A490: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A4A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A4B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A4C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A4D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A4E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A4F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A500: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A510: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A520: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A530: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A540: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A550: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A560: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A570: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A580: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A590: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A5A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A5B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A5C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A5D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A5E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A5F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A600: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A610: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A620: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A630: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A640: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A650: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A660: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A670: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A680: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A690: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A6A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A6B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A6C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A6D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A6E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A6F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A700: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A710: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A720: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A730: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A740: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A750: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
A760: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A770: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A780: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A790: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A7A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A7B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A7C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A7D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A7E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A7F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A800: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A810: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A820: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A830: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A840: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A850: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A860: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A870: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A880: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A890: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A8A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A8B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A8C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A8D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A8E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A8F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A900: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A910: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A920: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A930: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A940: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A950: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A960: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A970: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A980: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A990: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A9A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A9B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A9C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A9D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A9E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A9F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AA00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AA10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AA20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AA30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AA40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AA50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AA60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AA70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AA80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AA90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AAA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AAB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
AAC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AAD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AAE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AAF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AB00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AB10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AB20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AB30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AB40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AB50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AB60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AB70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AB80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AB90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ABA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ABB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ABC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ABD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ABE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ABF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AC00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AC10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AC20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AC30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AC40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AC50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AC60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AC70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AC80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AC90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ACA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ACB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ACC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ACD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ACE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ACF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AD00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AD10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AD20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AD30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AD40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AD50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AD60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AD70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AD80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AD90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ADA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ADB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ADC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ADD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ADE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ADF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AE00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AE10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
AE20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AE30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AE40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AE50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AE60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AE70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AE80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AE90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AEA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AEB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AEC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AED0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AEE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AEF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AF00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AF10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AF20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AF30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AF40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AF50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AF60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AF70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AF80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AF90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AFA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AFB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AFC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AFD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AFE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
AFF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B0F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
B180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B1A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B1B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B1C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B1D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B1E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B1F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B260: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B2A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B2B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B2C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B2D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B2E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B2F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B300: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B310: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B320: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B330: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B340: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B350: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B360: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B370: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B380: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B390: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B3A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B3B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B3C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B3D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B3E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B3F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B400: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B410: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B420: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B430: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B440: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B450: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B460: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B470: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B480: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B490: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B4A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B4B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B4C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B4D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
B4E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B4F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B500: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B510: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B520: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B530: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B540: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B550: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B560: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B570: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B580: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B590: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B5A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B5B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B5C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B5D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B5E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B5F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B600: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B610: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B620: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B630: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B640: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B650: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B660: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B670: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B680: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B690: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B6A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B6B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B6C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B6D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B6E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B6F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B700: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B710: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B720: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B730: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B740: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B750: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B760: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B770: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B780: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B790: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B7A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B7B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B7C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B7D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B7E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B7F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B800: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B810: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B820: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B830: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
B840: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B850: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B860: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B870: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B880: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B890: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B8A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B8B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B8C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B8D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B8E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B8F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B900: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B910: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B920: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B930: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B940: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B950: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B960: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B970: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B980: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B990: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B9A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B9B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B9C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B9D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B9E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B9F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BA00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BA10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BA20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BA30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BA40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BA50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BA60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BA70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BA80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BA90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BAA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BAB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BAC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BAD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BAE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BAF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BB00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BB10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BB20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BB30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BB40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BB50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BB60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BB70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BB80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BB90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
BBA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BBB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BBC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BBD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BBE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BBF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BC00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BC10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BC20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BC30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BC40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BC50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BC60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BC70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BC80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BC90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BCA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BCB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BCC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BCD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BCE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BCF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BD00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BD10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BD20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BD30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BD40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BD50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BD60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BD70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BD80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BD90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BDA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BDB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BDC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BDD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BDE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BDF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BE00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BE10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BE20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BE30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BE40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BE50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BE60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BE70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BE80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BE90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BEA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BEB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BEC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BED0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BEE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BEF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
BF00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BF10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BF20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BF30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BF40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BF50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BF60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BF70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BF80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BF90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BFA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BFB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BFC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BFD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BFE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BFF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

color_registers:
C000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C0F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C1A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C1B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C1C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C1D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C1E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C1F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
C240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C260: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C2A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C2B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C2C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C2D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C2E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C2F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C300: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C310: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C320: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C330: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C340: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C350: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C360: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C370: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C380: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C390: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C3A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C3B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C3C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C3D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C3E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C3F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C400: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C410: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C420: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C430: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C440: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C450: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C460: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C470: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C480: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C490: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C4A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C4B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C4C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C4D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C4E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C4F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C500: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C510: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C520: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C530: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C540: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C550: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C560: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C570: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C580: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C590: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
C5A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C5B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C5C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C5D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C5E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C5F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C600: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C610: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C620: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C630: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C640: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C650: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C660: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C670: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C680: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C690: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C6A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C6B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C6C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C6D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C6E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C6F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C700: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C710: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C720: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C730: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C740: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C750: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C760: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C770: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C780: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C790: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C7A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C7B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C7C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C7D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C7E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C7F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C800: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C810: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C820: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C830: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C840: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C850: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C860: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C870: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C880: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C890: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C8A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C8B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C8C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C8D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C8E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C8F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
C900: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C910: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C920: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C930: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C940: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C950: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C960: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C970: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C980: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C990: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C9A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C9B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C9C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C9D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C9E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C9F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CA00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CA10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CA20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CA30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CA40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CA50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CA60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CA70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CA80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CA90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CAA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CAB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CAC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CAD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CAE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CAF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CB00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CB10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CB20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CB30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CB40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CB50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CB60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CB70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CB80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CB90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CC00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CC10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CC20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CC30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CC40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CC50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
CC60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CC70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CC80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CC90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CCA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CCB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CCC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CCD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CCE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CCF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CD00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CD10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CD20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CD30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CD40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CD50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CD60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CD70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CD80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CD90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CDA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CDB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CDC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CDD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CDE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CDF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CE00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CE10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CE20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CE30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CE40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CE50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CE60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CE70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CE80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CE90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CEA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CEB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CEC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CED0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CEE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CEF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CF00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CF10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CF20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CF30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CF40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CF50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CF60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CF70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CF80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CF90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CFA0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CFB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
CFC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CFD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CFE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CFF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

D000: 7E D1 06    JMP   $D106

D003: 7E DE 0F    JMP   $DE0F

D006: 7E D3 B6    JMP   $D3B6

D009: 7E DC 11    JMP   $DC11

D00C: 7E DB 9C    JMP   $DB9C

D00F: 7E DC 13    JMP   $DC13

CLR_SCREEN1:
D012: 7E DB 7C    JMP   CLR_SCREEN

D015: 7E DB 03    JMP   $DB03

D018: 7E DA F2    JMP   $DAF2

D01B: 7E DA DF    JMP   $DADF

D01E: 7E DA BF    JMP   $DABF

D021: 7E DA 82    JMP   $DA82

D024: 7E D8 9E    JMP   $D89E

D027: 7E D7 C9    JMP   $D7C9

D02A: 7E D5 F5    JMP   $D5F5

D02D: 7E D5 E2    JMP   $D5E2

D030: 7E D7 A5    JMP   $D7A5

LOAD_DA51_PALETTE1:
D033: 7E D7 95    JMP   LOAD_DA51_PALETTE

D036: 7E D6 EC    JMP   $D6EC

D039: 7E D6 CD    JMP   $D6CD

D03C: 7E D6 C8    JMP   $D6C8

D03F: 7E D6 B6    JMP   $D6B6

D042: 7E D6 AC    JMP   $D6AC

D045: 7E D6 99    JMP   $D699
```

```
D048: 7E D6 A8    JMP   $D6A8
D04B: 7E D3 C7    JMP   $D3C7
D04E: 7E D2 A7    JMP   $D2A7
D051: 7E D2 8F    JMP   $D28F
D054: 7E D2 81    JMP   $D281
D057: 7E D2 5A    JMP   $D25A
D05A: 7E D2 43    JMP   $D243
D05D: 7E D2 18    JMP   $D218
D060: 7E D1 FF    JMP   $D1FF
D063: 7E D1 F3    JMP   $D1F3
D066: 7E D1 E3    JMP   $D1E3
D069: 7E D3 0E    JMP   $D30E
D06C: 7E D3 2B    JMP   $D32B
D06F: 7E D2 FD    JMP   $D2FD
D072: 7E D3 06    JMP   $D306
D075: 7E D3 1B    JMP   $D31B
D078: 7E D3 20    JMP   $D320
D07B: 7E D2 DA    JMP   $D2DA
D07E: 7E D2 C2    JMP   $D2C2
D081: 7E D2 E7    JMP   $D2E7
D084: 7E D2 CA    JMP   $D2CA
D087: 7E D2 F2    JMP   $D2F2
D08A: 7E D2 D2    JMP   $D2D2
D08D: 7E DB 2F    JMP   $DB2F
D090: 7E DA 9E    JMP   $DA9E
D093: 7E DA 61    JMP   $DA61
D096: 7E D1 96    JMP   $D196
```

```
FLIP_SCR_UP1:
D099: 7E D4 FC    JMP   FLIP_SCR_UP

FLIP_SCR_DOWN1:
D09C: 7E D5 03    JMP   FLIP_SCR_DOWN

D09F: 7E D5 C0    JMP   $D5C0

LDA_NIB_X1:
D0A2: 7E D5 12    JMP   LDA_NIB_X

D0A5: 7E D5 23    JMP   $D523

D0A8: 7E D5 21    JMP   $D521

STA_NIB_X1:
D0AB: 7E D5 2B    JMP   STA_NIB_X

D0AE: 7E D5 39    JMP   $D539

D0B1: 7E D5 37    JMP   $D537

D0B4: 7E D5 E2    JMP   $D5E2

D0B7: 7E DE 59    JMP   $DE59

D0BA: 7E D6 5B    JMP   $D65B

D0BD: 7E D6 55    JMP   $D655

D0C0: 7E D1 8A    JMP   $D18A

D0C3: 7E DA 0D    JMP   $DA0D

D0C6: 7E D5 D8    JMP   $D5D8

D0C9: EF 01       STU   $0001,X
D0CB: 20 1E       BRA   $D0EB

D0CD: 00 FF       NEG   $FF
D0CF: 01          Illegal Opcode
D0D0: 20 0C       BRA   $D0DE

D0D2: 00 FF       NEG   $FF
D0D4: 01          Illegal Opcode
D0D5: 20 20       BRA   $D0F7

D0D7: 00 FF       NEG   $FF
D0D9: 03 10       COM   $10
D0DB: 24 00       BCC   $D0DD
D0DD: FF 01 20    STU   $0120
D0E0: 27 00       BEQ   $D0E2
D0E2: FF 01 20    STU   $0120
```

```
D0E5: 2D 00       BLT   $D0E7
D0E7: FF 02 10    STU   $0210
D0EA: 35 00       PULS
D0EC: FF 01 20    STU   $0120
D0EF: 3A          ABX
D0F0: 00 FF       NEG   $FF
D0F2: 01          Illegal Opcode
D0F3: 20 3E       BRA   $D133

D0F5: 00 00       NEG   $00
D0F7: 34 FF       PSHS  PC,U,Y,X,DP,B,A,CC
D0F9: 35 00       PULS
D0FB: 34 00       PSHS
D0FD: 3C C8       CWAI  #$C8
D0FF: 0C C8       INC   $C8
D101: 0E C8       JMP   $C8

D103: 04 C8       LSR   $C8
D105: 06 1A       ROR   $1A
D107: FF 10 CE    STU   $10CE
D10A: BF 70 86    STX   $7086
D10D: 98 1F       EORA  $1F
D10F: 8B 86       ADDA  #$86
D111: 01          Illegal Opcode
D112: B7 C9 00    STA   rom_enable_scr_ctrl
D115: 8E D0 F6    LDX   #$D0F6
D118: 4F          CLRA
D119: 5F          CLRB
D11A: ED 98 08    STD   [$08,X]
D11D: EC 81       LDD   ,X++
D11F: ED 98 06    STD   [$06,X]
D122: 8C D0 FE    CMPX  #$D0FE
D125: 26 F1       BNE   $D118
D127: 86 FF       LDA   #$FF
D129: B7 C8 0E    STA   rom_pia_datab
D12C: BD D0 12    JSR   CLR_SCREEN1
D12F: 86 3F       LDA   #$3F
D131: B7 C8 0E    STA   rom_pia_datab

D134: 8E 98 00    LDX   #$9800              ; clear memory used to
hold linked lists of objects
D137: 6F 80       CLR   ,X+
D139: C6 39       LDB   #$39
D13B: F7 CB FF    STB   watchdog            ; make sure watchdog is
kept happy
D13E: 8C BF 70    CMPX  #stacktop
D141: 26 F4       BNE   $D137               ; if not at stacktop,
keep clearing
D143: CC A5 5A    LDD   #$A55A
D146: DD 85       STD   $85
D148: 86 60       LDA   #$60
D14A: 97 41       STA   $41
D14C: BD D0 99    JSR   FLIP_SCR_UP1
D14F: BD D0 36    JSR   $D036               ; JMP $D6EC
```

```
D152: 8D 36       BSR   $D18A
D154: BD 6F 03    JSR   $6F03
D157: CC FF FF    LDD   #$FFFF
D15A: DD 2F       STD   $2F
D15C: DD 31       STD   $31
D15E: 86 02       LDA   #$02
D160: 97 40       STA   $40
D162: 8E CD 00    LDX   #credits_cmos
D165: BD D0 A2    JSR   LDA_NIB_X1          ; convert 2 bytes to
single byte BCD value
D168: 1F 89       TFR   A,B
D16A: 81 20       CMPA  #$20
D16C: 22 06       BHI   $D174
D16E: C4 0F       ANDB  #$0F
D170: C1 09       CMPB  #$09
D172: 25 04       BCS   $D178
D174: 4F          CLRA
D175: BD D0 AB    JSR   STA_NIB_X1
D178: 97 51       STA   $51
D17A: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
D17D: 77 A0          ; pointer to function
D17F: 86 2C       LDA   #$2C
D181: B7 C8 0E    STA   $C80E
D184: 03 59       COM   $59
D186: 1C 00       ANDCC #$00                ; clear all flags
D188: 20 0C       BRA   $D196

D18A: BD D0 33    JSR   LOAD_DA51_PALETTE1
D18D: BD 5F 9C    JSR   $5F9C
D190: BD 5B 40    JSR   $5B40               ; clear explosion list
D193: 7E D0 30    JMP   $D030



D196: 8E 98 11    LDX   #$9811
D199: 9F 15       STX   $15
D19B: 96 10       LDA   $10
D19D: 81 02       CMPA  #$02
D19F: 25 FA       BCS   $D19B
D1A1: 48          ASLA
D1A2: 48          ASLA
D1A3: 48          ASLA
D1A4: 9B 42       ADDA  $42
D1A6: 44          LSRA
D1A7: 97 42       STA   $42
D1A9: 0F 10       CLR   $10
D1AB: BD D6 CD    JSR   $D6CD
D1AE: 96 59       LDA   $59
D1B0: 85 04       BITA  #$04
D1B2: 26 03       BNE   $D1B7
D1B4: BD 5B 49    JSR   $5B49
D1B7: 9E 33       LDX   $33
D1B9: 26 0C       BNE   $D1C7
```

```
D1BB: 9E 37       LDX   $37
D1BD: 27 17       BEQ   $D1D6
D1BF: DC 39       LDD   $39
D1C1: 0F 37       CLR   $37
D1C3: 0F 38       CLR   $38
D1C5: 20 06       BRA   $D1CD

D1C7: DC 35       LDD   $35
D1C9: 0F 33       CLR   $33
D1CB: 0F 34       CLR   $34
D1CD: D4 59       ANDB  $59
D1CF: 26 E6       BNE   $D1B7
D1D1: BD D0 57    JSR   $D057                ; JMP $D25A - reserve
object metadata entry in list @ $9813 and call function in X
D1D4: 20 E1       BRA   $D1B7

D1D6: DE 11       LDU   $11
D1D8: 27 13       BEQ   $D1ED


D1DA: 6A 44       DEC   $0004,U              ; decrement delay
counter
D1DC: 26 0B       BNE   $D1E9                ; if !=0 then go do
next object
D1DE: DF 15       STU   $15                  ; save address of
current object being processed
D1E0: 6E D8 02    JMP   [$02,U]              ; call routine to
handle object


; Allocate function call.
;
; A = initial delay (in game cycles?) before calling function
; X = address of function to jump to
;

ALLOCATE_FUNCTION_CALL:
D1E3: DE 15       LDU   $15                  ; get pointer to free
metadata slot
D1E5: A7 44       STA   $0004,U              ; set initial delay of
object
D1E7: AF 42       STX   $0002,U              ; store address of
routine to jump to
D1E9: EE C4       LDU   ,U                   ; read next object
entry
D1EB: 26 ED       BNE   $D1DA                ; if !=NULL then go
process it
D1ED: 10 CE BF 70 LDS   #stacktop
D1F1: 20 A3       BRA   $D196



D1F3: 9E 15       LDX   $15
D1F5: 10 CE BF 70 LDS   #stacktop
```

```
D1F9: 8D 1D       BSR   $D218               ; free object metadata
entry in X
D1FB: 33 84       LEAU  ,X
D1FD: 20 EA       BRA   $D1E9               ; process object entry
in U



D1FF: 34 12       PSHS  X,A
D201: 8E 98 11    LDX   #$9811              ; object metadata list
start
D204: AE 84       LDX   ,X
D206: 27 0E       BEQ   $D216
D208: 9C 15       CPX   $15
D20A: 27 F8       BEQ   $D204
D20C: A6 05       LDA   $0005,X
D20E: 81 01       CMPA  #$01
D210: 27 F2       BEQ   $D204
D212: 8D 04       BSR   $D218
D214: 20 EE       BRA   $D204

D216: 35 92       PULS  A,X,PC ;(PUL? PC=RTS)



; Called when an object is gone (e.g.: when it's been killed,
rescued) and its associated metadata
; needs to be freed for use by other objects.
;
; X = pointer to an object's metadata
;

FREE_OBJECT_METADATA_ENTRY:
D218: 34 46       PSHS  U,B,A
D21A: CE 98 11    LDU   #$9811              ; object metadata list
start
D21D: AC C4       CPX   ,U                  ; x == *u ? (have we
matched x in the list?)
D21F: 26 18       BNE   $D239               ; if x != *u, goto
$d239
D221: EC 84       LDD   ,X                  ; get pointer to NEXT
object metadata entry into D
D223: ED C4       STD   ,U                  ; store in list
D225: A6 06       LDA   $0006,X
D227: 27 06       BEQ   $D22F
D229: DC 1D       LDD   $1D
D22B: 9F 1D       STX   $1D
D22D: 20 04       BRA   $D233
D22F: DC 13       LDD   $13
D231: 9F 13       STX   $13
D233: ED 84       STD   ,X
D235: 30 C4       LEAX  ,U
D237: 35 C6       PULS  A,B,U,PC ;(PUL? PC=RTS)
```

```
D239: EE C4       LDU   ,U                  ; get next entry in
object metadata list
D23B: 26 E0       BNE   $D21D               ; if not null goto $D21D
D23D: 8D 00       BSR   $D23F
D23F: 1A 10       ORCC  #$10                ; disable interrupts
D241: 20 FE       BRA   $D241               ; put in an infinite
loop - this must be to invoke the watchdog!



; Well well well, looks like there's 2 object metadata lists, not
just the one.
;
; Returns: X = pointer to object metadata
;

RESERVE_OBJECT_METADATA_ENTRY_1D:
D243: 34 62       PSHS  U,Y,A
D245: DE 1D       LDU   $1D
D247: 26 01       BNE   $D24A               ; ???????? !!!!! $D24A
is invalid code. But this routine still works, I stepped through it
in MAME.
D249: BD D2 3F    JSR   $D23F               ; invoke watchdog
D24C: 10 AE C4    LDY   ,U
D24F: 10 9F 1D    STY   $1D
D252: 86 01       LDA   #$01                ; flag to say that this
object is in list $1D (see $D225)
D254: A7 46       STA   $0006,U
D256: A6 E4       LDA   ,S                  ; read A from stack
D258: 20 11       BRA   $D26B



;
; Objects have their own state, but also have metadata which records
important things RELATED TO the object (such as countdown timers
; that when reach zero, permit the object to move). I call these
things "object metadata".
;
; On entry: X = function to call
; A = ???
;
; On exit: X = pointer to object metadata entry

RESERVE_OBJECT_METADATA_ENTRY_13:
D25A: 34 62       PSHS  U,Y,A
D25C: DE 13       LDU   $13                  ; get valid object
metadata entry
D25E: 26 03       BNE   $D263                ; if not NULL goto
$D263
D260: BD D2 3F    JSR   $D23F                ; OK, this value is
null, jump into an infinite loop to invoke watchdog
D263: 10 AE C4    LDY   ,U                   ; Y = *U. Get next
valid object metadata entry..
D266: 10 9F 13    STY   $13                  ; ...and store in $13
```

```
D269: 6F 46       CLR   $0006,U
D26B: AF 42       STX   $0002,U             ; function to call (see
$D1E0)
D26D: A7 45       STA   $0005,U
D26F: 86 01       LDA   #$01
D271: A7 44       STA   $0004,U             ; delay
D273: AE 9F 98 15 LDX   [$9815]
D277: EF 9F 98 15 STU   [$9815]
D27B: AF C4       STX   ,U
D27D: 30 C4       LEAX  ,U                  ; X= U
D27F: 35 E2       PULS  A,Y,U,PC ;(PUL? PC=RTS)



;
; Strange bit of code here. Called a lot.
; You'll see blocks of code like the following in the disassembly:
;
; BD D0 54    JSR   $D054                  ; JMP $D281
; 77 A0          ; pointer to function to call
; EF 07       STU   $0007,X
;
; What happens is that the function loads the return address (the
address the system would jump to
; when it encounters an RTS or a PULS PC) from the stack into U.
Cunningly, the return address
; points to 2 bytes which are parameters. A PULU X reads the 2 bytes
from the return address into X,
; in this case 77 A0.
;
; The return address is then updated on the stack to what U is
*after* the PULU.
; When an RTS or PULS PC is hit, the system will pop the return
address off the stack,
; which of course now points to the instruction immediately
*following* the 2 parameter bytes:
; STU $0007,X . The parameter bytes never get processed by the CPU
as instructions (which is a good thing).
;
; The system continues as normal.
;
; This is a crude way of passing parameters to a function, wonder
why the Vid Kidz did it this way?
;
; Returns: X set to object metadata entry.
;

RESERVE_OBJECT_METADATA_AND_CALL_FUNCTION:
D281: 34 42       PSHS  U,A
D283: EE 63       LDU   $0003,S             ; get return address
off the stack and put into U
D285: 37 10       PULU  X                   ; pull parameters from
U into X
D287: EF 63       STU   $0003,S             ; update return address
```

```
of this function to be == U.
D289: 86 00       LDA   #$00
D28B: 8D CD       BSR   $D25A               ; reserve object
metadata entry
D28D: 35 C2       PULS  A,U,PC ;(PUL? PC=RTS)


;
; Reserve an object in the linked list for use by a game entity
(Player, grunt etc etc.)
; Returns: X = the newly reserved object
;

RESERVE_OBJECT_IN_LINKED_LIST:
D28F: 34 06       PSHS  B,A
D291: 9E 1B       LDX   $1B                 ; read pointer to free
object in linked list
D293: 26 03       BNE   $D298               ; if not 0 (end of
available space) go to $D298
D295: BD D2 3F    JSR   $D23F               ; pointer is zero - so
go to a subroutine that ends up infinite loop. I think this is to
force the watchdog to reset the game
D298: EC 84       LDD   ,X                  ; D = *pointer - now D
points to the *next* free object, as this one has been taken up
D29A: DD 1B       STD   $1B                 ; save free object
pointer
D29C: C6 02       LDB   #$02                ; zero from position X
+2 to X+#$18 - clear allocated object's internal state to 0
D29E: 6F 85       CLR   B,X
D2A0: 5C          INCB
D2A1: C1 18       CMPB  #$18
D2A3: 26 F9       BNE   $D29E
D2A5: 35 86       PULS  A,B,PC ;(PUL? PC=RTS)


; Called when an object dies or needs to disappear. The object entry
is then free for re-use.
;
; X = Object to free
; U = pointer to linked list that contains the object
;

FREE_OBJECT:
D2A7: AC C4       CPX   ,U                  ; find X in the list
D2A9: 26 10       BNE   $D2BB
; OK we've found X in the list
D2AB: 10 AE D4    LDY   [,U]                ; need to check to see
what this does
D2AE: 10 AF C4    STY   ,U
D2B1: 10 9E 1B    LDY   $1B                 ; get pointer to next
object
D2B4: 9F 1B       STX   $1B                 ; mark this object in X
as current free object
D2B6: 10 AF 84    STY   ,X                  ; set pointer to next
```

```
free object at (*X) - thus creating a chain of "free objects"
D2B9: 35 F0       PULS  X,Y,U,PC ;(PUL? PC=RTS)

D2BB: EE C4       LDU   ,U
D2BD: 26 E8       BNE   $D2A7
D2BF: BD D2 3F    JSR   $D23F                ; go to a subroutine
that forces an infinite loop - to force watchdog



;
; grunts, hulks, brains, progs, cruise missiles and tanks
;

FREE_GRUNT_HULK_BRAIN_PROG_CRUISE_OR_TANK:
D2C2: 34 70       PSHS  U,Y,X
D2C4: CE 98 21    LDU   #$9821               ; pointer to
grunts_hulks_brains_progs_cruise_tanks list start
D2C7: 7E D0 4E    JMP   $D04E                ; JMP $D2A7: free
object for use


FREE_ELECTRODE_OBJECT:
D2CA: 34 70       PSHS  U,Y,X
D2CC: CE 98 23    LDU   #$9823               ; pointer to electrode
list
D2CF: 7E D0 4E    JMP   $D04E                ; JMP $D2A7: free
object for use


FREE_FAMILY_MEMBER_OBJECT:
D2D2: 34 70       PSHS  U,Y,X
D2D4: CE 98 1F    LDU   #$981F               ; pointer to family
list
D2D7: 7E D0 4E    JMP   $D04E                ; JMP $D2A7: free
object for use


RESERVE_GRUNT_HULK_BRAIN_PROG_CRUISE_TANK:
D2DA: 34 06       PSHS  B,A
D2DC: BD D0 51    JSR   $D051                ; JMP $D28F - reserve
an object entry
; X = the newly reserved object
D2DF: DC 21       LDD   $21                  ; get pointer to last
object created
D2E1: 9F 21       STX   $21                  ; store pointer to
freshly created object in $21
D2E3: ED 84       STD   ,X                   ; create linked list
from X to D
D2E5: 35 86       PULS  A,B,PC ;(PUL? PC=RTS)


RESERVE_ELECTRODE_OBJECT:
D2E7: 34 06       PSHS  B,A
```

```
D2E9: BD D0 51    JSR   $D051               ; reserve an object
entry        JMP $D28F
; X = the newly reserved object
D2EC: DC 23       LDD   $23                 ; get pointer to last
electrode created
D2EE: 9F 23       STX   $23                 ; store pointer to
freshly created object in $21
D2F0: 20 F1       BRA   $D2E3



RESERVE_FAMILY_MEMBER_OBJECT:
D2F2: 34 06       PSHS  B,A
D2F4: BD D0 51    JSR   $D051               ; reserve an object
entry      JMP $D28F
; at this point X = our new object
D2F7: DC 1F       LDD   $1F                 ; get pointer to last
family object created
D2F9: 9F 1F       STX   $1F                 ; store object entry
into family object pointer
D2FB: 20 E6       BRA   $D2E3


;
;
;

RESERVE_OBJECT:
D2FD: 34 06       PSHS  B,A
D2FF: BD D0 51    JSR   $D051               ; JMP $D28F: reserve an
object entry
; X = the newly reserved object
D302: DC 17       LDD   $17
D304: 20 DD       BRA   $D2E3               ; *X = D


;
; X = object to remove
;

FREE_ENFORCER_QUARK_SPARK_SHELL:
D306: 34 70       PSHS  U,Y,X
D308: CE 98 17    LDU   #$9817              ; address of list for
those types of entities
D30B: 7E D0 4E    JMP   $D04E               ; JMP $D2A7: free
object for use



D30E: 10 8E AE D9 LDY   #$AED9
D312: BD D0 54    JSR   $D054               ; JMP D281
D315: D3 68       ; pointer to function
D317: 10 AF 09    STY   $0009,X
```

```
D31A: 39          RTS

FREE_OBJECT_AND_ERASE_SPRITE:
D31B: 8D E9       BSR   $D306              ; deallocate object's
metadata
D31D: 7E D0 15    JMP   $D015              ; JMP $DB03: erase
object's sprite from screen



FREE_OBJECT_AND_ERASE_SPRITE2:
D320: 34 10       PSHS  X
D322: 8D F7       BSR   $D31B
D324: AE 06       LDX   $0006,X
D326: BD D0 5D    JSR   $D05D              ; JMP $D218 - deallocate
object metadata entry
D329: 35 90       PULS  X,PC ;(PUL? PC=RTS)



; Create an entity with parameters. An entity in this case can be a
spark, an enforcer, a quark, or a tank shell.
;
; Returns: pointer to new entity in X
;
; Notes:
; When this function is called, the function obtains the function
return address from the stack,
; pulls *6* bytes from the return address, which are its parameters
(see below for description)
; then modifies the return address on the stack to point to the
instruction *immediately following the last parameter*.
; When the function returns (ie: hits RTS) the game continues from
the line of code following the parameters! Quite smart eh?
;
; There are 3 parameters, all are pointers:
; First parameter: pointer to constructor to initialise object
; Second parameter: animation frame metadata pointer
; Third parameter:  pointer to collision detection routine
;

CREATE_ENFORCER_QUARK_SPARK_SHELL:
D32B: 34 26       PSHS  Y,B,A
D32D: 9E 1B       LDX   $1B                ; read free object slot
D32F: 27 2E       BEQ   $D35F              ; if its null we don't
have any free objects - must be a lot happening! - so just exit
D331: 9E 13       LDX   $13                ; read object linked
list pointer
D333: 27 2A       BEQ   $D35F              ; if its null we've no
object slots available either
D335: 4F          CLRA
D336: EE 64       LDU   $0004,S            ; U = return address
from stack
D338: 37 10       PULU  X                  ; pull pointer to
```

```
function that initialises object (akin to a constructor in C++/Java/
C# etc.) from U into X
D33A: BD D0 57    JSR   $D057                ; JMP $D25A - reserve
object metadata entry in list @ $9813 and call function in X
; X = pointer to object metadata entry
D33D: 31 84       LEAY  ,X                  ; Y = X
D33F: BD D0 6F    JSR   $D06F                ; JMP $D2FD - reserve
an object entry & store at $17.
; X = pointer to freshly created object
D342: EC C1       LDD   ,U++                ; read next 2 bytes
(animation frame metadata pointer) from U into D
D344: ED 88 14    STD   $14,X               ; set previous
animation frame metadata pointer
D347: ED 02       STD   $0002,X             ; set current animation
frame metadata pointer (previous = current)
D349: 37 06       PULU  A,B                 ; pull pointer to
collision detection routine from U into D
D34B: ED 08       STD   $0008,X             ; store pointer to
routine that handles collision detection
D34D: EF 64       STU   $0004,S
D34F: 33 A4       LEAU  ,Y                  ; U = Y
D351: EF 06       STU   $0006,X             ; set pointer to object
metadata in this object
D353: AF 47       STX   $0007,U             ; set pointer to this
object in the object metadata entry
D355: 4F          CLRA
D356: 5F          CLRB
D357: ED 88 10    STD   $10,X               ; set Y delta to 0
D35A: ED 0E       STD   $000E,X             ; set X delta to 0
D35C: 43          COMA                      ; flip bits (xor with #
$FF) to set z flag
D35D: 35 A6       PULS  A,B,Y,PC ;(PUL? PC=RTS)
;
; If we get here, the object can't be created, so we need to clear
the z flag
;
D35F: EE 64       LDU   $0004,S             ; U = Y from stack
D361: 33 48       LEAU  $0008,U
D363: EF 64       STU   $0004,S
D365: 4F          CLRA                      ; clear zero flag to
indicate failure
D366: 35 A6       PULS  A,B,Y,PC ;(PUL? PC=RTS)

D368: 96 F2       LDA   $F2
D36A: 26 47       BNE   $D3B3
D36C: 0C F2       INC   $F2
D36E: 86 03       LDA   #$03
D370: AE 49       LDX   $0009,U
D372: 30 89 28 57 LEAX  $2857,X
D376: AF 49       STX   $0009,U
D378: A7 47       STA   $0007,U
D37A: 86 08       LDA   #$08
D37C: 8E D3 82    LDX   #$D382
D37F: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
```

```
function call



;
; This is the code that handles the second actions to show the
Easter Egg:
; Move the "move" joystick up and the "fire" joystick down and press
the two player start button.
;

EASTER_EGG_STEP_2:
D382: B6 C8 04    LDA   widget_pia_dataa
D385: 81 A1       CMPA  #$A1                ; Up + 2 player + Fire
down
D387: 27 0A       BEQ   $D393
D389: 81 58       CMPA  #$58
D38B: 27 ED       BEQ   $D37A
D38D: 6A 47       DEC   $0007,U
D38F: 26 E9       BNE   $D37A
D391: 20 1E       BRA   $D3B1

D393: 86 03       LDA   #$03
D395: A7 47       STA   $0007,U
D397: 86 08       LDA   #$08
D399: 8E D3 9F    LDX   #$D39F
D39C: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call


;
; This is the code that handles the third actions to show the Easter
Egg:
; Move the "move" joystick down and the "fire" joystick up.
;

EASTER_EGG_STEP_3:
D39F: B6 C8 04    LDA   widget_pia_dataa
D3A2: 81 42       CMPA  #$42                ; Fire up + Move Down
D3A4: 26 03       BNE   $D3A9
D3A6: 6E D8 09    JMP   [$09,U]                           ; jump
to $D730 - EASTER EGG ACTIVATED!!!!

D3A9: 81 A1       CMPA  #$A1
D3AB: 27 EA       BEQ   $D397
D3AD: 6A 47       DEC   $0007,U
D3AF: 26 E6       BNE   $D397
D3B1: 0F F2       CLR   $F2
D3B3: 7E D0 63    JMP   $D063               ; JMP $D1F3

D3B6: 34 07       PSHS  B,A,CC
D3B8: 1A FF       ORCC  #$FF
D3BA: 86 3F       LDA   #$3F
D3BC: B7 C8 0E    STA   rom_pia_datab
```

```
D3BF: 53          COMB
D3C0: C4 3F       ANDB  #$3F
D3C2: F7 C8 0E    STB   rom_pia_datab
D3C5: 35 87       PULS  CC,A,B,PC ;(PUL? PC=RTS)


; D = pointer to data. But what's this data used for?
;
;

D3C7: 34 17       PSHS  X,B,A,CC
D3C9: 1F 01       TFR   D,X                   ; X = D
D3CB: A6 84       LDA   ,X                    ; read byte from X
D3CD: 91 56       CMPA  $56
D3CF: 25 0D       BCS   $D3DE                 ; <
D3D1: 97 56       STA   $56
D3D3: 30 1E       LEAX  -$2,X                 ; X = X - 2
D3D5: 1A 10       ORCC  #$10                  ; disable interrupts
D3D7: 9F 54       STX   $54
D3D9: CC 01 01    LDD   #$0101
D3DC: DD 57       STD   $57
D3DE: 35 97       PULS  CC,A,B,X,PC ;(PUL? PC=RTS)

D3E0: 96 57       LDA   $57
D3E2: 27 1E       BEQ   $D402
D3E4: 0A 57       DEC   $57
D3E6: 26 1A       BNE   $D402
D3E8: 9E 54       LDX   $54
D3EA: 0A 58       DEC   $58
D3EC: 26 0E       BNE   $D3FC
D3EE: 30 03       LEAX  $0003,X
D3F0: 9F 54       STX   $54
D3F2: A6 84       LDA   ,X
D3F4: 26 04       BNE   $D3FA
D3F6: 97 56       STA   $56
D3F8: 20 08       BRA   $D402


; c80c rom_pia_dataa
;  bit 0  Auto Up
;  bit 1  Advance
;  bit 2  Right Coin
;  bit 3  High Score Reset
;  bit 4  Left Coin
;  bit 5  Center Coin
;  bit 6  Slam Door Tilt
;  bit 7  Hand Shake from sound board

D3FA: 97 58       STA   $58
D3FC: EC 01       LDD   $0001,X
D3FE: 97 57       STA   $57
D400: 8D B4       BSR   $D3B6
D402: B6 C8 0C    LDA   rom_pia_dataa
D405: 85 40       BITA  #$40                  ; read "slam door
```

```
tilt" bit
D407: 27 04       BEQ   $D40D
D409: 86 3C       LDA   #$3C
D40B: 97 4B       STA   $4B
D40D: 96 4B       LDA   $4B
D40F: 27 02       BEQ   $D413
D411: 0A 4B       DEC   $4B
D413: 96 4C       LDA   $4C
D415: 27 02       BEQ   $D419
D417: 0A 4C       DEC   $4C
D419: 96 4E       LDA   $4E
D41B: 27 02       BEQ   $D41F
D41D: 0A 4E       DEC   $4E
D41F: 96 4D       LDA   $4D
D421: 27 02       BEQ   $D425
D423: 0A 4D       DEC   $4D
D425: 96 59       LDA   $59
D427: 2A 24       BPL   $D44D
D429: 96 CE       LDA   $CE
D42B: 26 6A       BNE   $D497
D42D: 96 31       LDA   $31
D42F: 9A 32       ORA   $32
D431: 43          COMA
D432: D6 31       LDB   $31
D434: D7 32       STB   $32
D436: F6 C8 04    LDB   widget_pia_dataa
D439: D7 31       STB   $31
D43B: 94 31       ANDA  $31
D43D: 84 30       ANDA  #$30
D43F: 27 0C       BEQ   $D44D
D441: 8E 26 CC    LDX   #$26CC
D444: 85 10       BITA  #$10
D446: 26 03       BNE   $D44B
D448: 8E 26 CF    LDX   #$26CF
D44B: 8D 3B       BSR   $D488
D44D: 96 CE       LDA   $CE
D44F: 26 46       BNE   $D497
D451: 96 2F       LDA   $2F
D453: 9A 30       ORA   $30
D455: 43          COMA
D456: D6 2F       LDB   $2F
D458: D7 30       STB   $30
D45A: F6 C8 0C    LDB   rom_pia_dataa
D45D: C4 3F       ANDB  #$3F
D45F: D7 2F       STB   $2F
D461: 95 2F       BITA  $2F
D463: 27 32       BEQ   $D497
D465: 8E 00 78    LDX   #$0078
D468: 30 1F       LEAX  $FFFF,X
D46A: 26 FC       BNE   $D468
D46C: F6 C8 0C    LDB   rom_pia_dataa
D46F: D4 2F       ANDB  $2F
D471: D7 2F       STB   $2F
D473: 94 2F       ANDA  $2F
```

```
D475: 27 20       BEQ   $D497
D477: 8E D4 96    LDX   #$D496
D47A: 30 02       LEAX  $0002,X
D47C: 44          LSRA
D47D: 24 FB       BCC   $D47A
D47F: AE 84       LDX   ,X
D481: 8D 05       BSR   $D488
D483: 86 01       LDA   #$01
D485: A7 42       STA   $0002,U
D487: 39          RTS

D488: CE 98 33    LDU   #$9833
D48B: EC C4       LDD   ,U
D48D: 27 02       BEQ   $D491
D48F: 33 44       LEAU  $0004,U
D491: AF C4       STX   ,U
D493: 6F 42       CLR   $0002,U
D495: 6F 43       CLR   $0003,U
D497: 39          RTS

D498: 00 00       NEG   $00
D49A: F0 03 D6    SUBB  $03D6
D49D: 15          Illegal Opcode
D49E: E3 DF D6 0C ADDD  [$D60C,U]
D4A2: D6 1E       LDB   $1E
D4A4: 00 00       NEG   $00
D4A6: 00 00       NEG   $00
D4A8:  ROBOTRON: 2084 (TM)  COPYRIGHT
D4C8: 1982 WILLIAMS ELECTRONICS INC.
D4E8: ALL RIGHTS RESERVED

FLIP_SCR_UP:
D4FC: 34 06       PSHS  B,A
D4FE: CC 01 3C    LDD   #$013C
D501: 20 05       BRA   $D508

FLIP_SCR_DOWN:
D503: 34 06       PSHS  B,A
D505: CC 03 34    LDD   #$0334
D508: 97 45       STA   $45
D50A: B7 C9 00    STA   rom_enable_scr_ctrl
D50D: F7 C8 07    STB   widget_pia_ctrlb
D510: 35 86       PULS  A,B,PC ;(PUL? PC=RTS)



LDA_NIB_X:

; Pack 2 nibbles into A
;
; X = pointer to two bytes.
; Byte 0: number from 0 - #$0F (15 decimal)
; Byte 1: number from 0 - #$0F (15 decimal)
;
```

```
; Byte 0's value will be moved to bits 4..7 of A
; Byte 1's value will be moved to bits 0..3 of A
;
; IMPORTANT: on exit, X will be incremented by 2.

D512: A6 01       LDA   $0001,X             ; read number from X + 1
D514: 84 0F       ANDA  #$0F                ; mask in bit 0..3
D516: 34 02       PSHS  A                   ; save on stack for
addition @ $D51E
D518: A6 81       LDA   ,X++                ; read number from X
D51A: 48          ASLA
D51B: 48          ASLA
D51C: 48          ASLA
D51D: 48          ASLA                      ; shift bits from 0..3
to 4..7
D51E: AB E0       ADDA  ,S+                 ; add in value of A on
stack
D520: 39          RTS

D521: 8D EF       BSR   LDA_NIB_X           ; convert 2 bytes to
single byte BCD value
D523: 34 02       PSHS  A
D525: 8D EB       BSR   LDA_NIB_X           ; convert 2 bytes to
single byte BCD value
D527: 1F 89       TFR   A,B
D529: 35 82       PULS  A,PC ;(PUL? PC=RTS)

STA_NIB_X:
D52B: 34 02       PSHS  A
D52D: A7 01       STA   $0001,X
D52F: 44          LSRA
D530: 44          LSRA
D531: 44          LSRA
D532: 44          LSRA
D533: A7 81       STA   ,X++
D535: 35 82       PULS  A,PC ;(PUL? PC=RTS)

D537: 8D F2       BSR   STA_NIB_X
D539: 34 02       PSHS  A
D53B: 1F 98       TFR   B,A
D53D: 8D EC       BSR   STA_NIB_X
D53F: 35 82       PULS  A,PC ;(PUL? PC=RTS)

ADDA_CREDS:
D541: 34 12       PSHS  X,A
D543: 9B 51       ADDA  $51
D545: 19          DAA
D546: 24 02       BCC   $D54A
D548: 86 99       LDA   #$99
D54A: 97 51       STA   $51
D54C: 8E CD 00    LDX   #credits_cmos
D54F: BD D0 AB    JSR   STA_NIB_X1
D552: 35 92       PULS  A,X,PC ;(PUL? PC=RTS)
```

```
D554: 34 16       PSHS  X,B,A
D556: C6 03       LDB   #$03
D558: 20 0A       BRA   $D564

D55A: 34 16       PSHS  X,B,A
D55C: C6 02       LDB   #$02
D55E: 20 04       BRA   $D564

D560: 34 16       PSHS  X,B,A
D562: C6 01       LDB   #$01
D564: BD D0 BD    JSR   $D0BD               ; JMP $D655
D567: 58          ASLB
D568: 8E CC 04    LDX   #$CC04
D56B: 3A          ABX
D56C: BD D0 A5    JSR   $D0A5
D56F: BD D0 B4    JSR   $D0B4
D572: 96 4F       LDA   $4F
D574: 34 04       PSHS  B
D576: AB E4       ADDA  ,S
D578: 97 4F       STA   $4F
D57A: 96 50       LDA   $50
D57C: AB E0       ADDA  ,S+
D57E: 97 50       STA   $50
D580: 8E CC 10    LDX   #$CC10
D583: BD D0 A5    JSR   $D0A5
D586: BD D0 B4    JSR   $D0B4
D589: 34 04       PSHS  B
D58B: A1 E0       CMPA  ,S+
D58D: 24 02       BCC   $D591
D58F: 35 96       PULS  A,B,X,PC ;(PUL? PC=RTS)

D591: 8E CC 0C    LDX   #$CC0C
D594: BD D0 A5    JSR   $D0A5
D597: BD D0 B4    JSR   $D0B4
D59A: 8D 24       BSR   $D5C0
D59C: 34 02       PSHS  A
D59E: D7 50       STB   $50
D5A0: 8E CC 0E    LDX   #$CC0E
D5A3: BD D0 A5    JSR   $D0A5
D5A6: 96 4F       LDA   $4F
D5A8: 8D 38       BSR   $D5E2
D5AA: 8D 14       BSR   $D5C0
D5AC: 4D          TSTA
D5AD: 27 04       BEQ   $D5B3
D5AF: 0F 50       CLR   $50
D5B1: 0F 4F       CLR   $4F
D5B3: AB E0       ADDA  ,S+
D5B5: 19          DAA
D5B6: C6 04       LDB   #$04
D5B8: BD D0 BA    JSR   $D0BA
D5BB: BD D5 41    JSR   ADDA_CREDS
D5BE: 35 96       PULS  A,B,X,PC ;(PUL? PC=RTS)

D5C0: 34 04       PSHS  B
```

```
D5C2: 5D          TSTB
D5C3: 26 03       BNE   $D5C8
D5C5: 4F          CLRA
D5C6: 35 84       PULS  B,PC ;(PUL? PC=RTS)

D5C8: 1E 89       EXG   A,B
D5CA: 86 99       LDA   #$99
D5CC: 8B 01       ADDA  #$01
D5CE: 19          DAA
D5CF: E0 E4       SUBB  ,S
D5D1: 24 F9       BCC   $D5CC
D5D3: EB E0       ADDB  ,S+
D5D5: 39          RTS

D5D6: D8 3E       EORB  $3E


;
; A = packed byte where bits 4..7 represent the first number and
0..3 represent second.
; Returns: unpacked value in A
;

CONVERT_BCD_TO_NUMBER:
D5D8: 34 04       PSHS  B                   ;
D5DA: 1F 89       TFR   A,B                    ;
D5DC: 8D 04       BSR   $D5E2
D5DE: 1F 98       TFR   B,A                 ;
D5E0: 35 84       PULS  B,PC ;(PUL? PC=RTS)

D5E2: 34 02       PSHS  A
D5E4: 4F          CLRA
D5E5: C1 10       CMPB  #$10                ; compare B to #$10 (16
dec)
D5E7: 25 06       BCS   $D5EF               ; if < #$10 goto $D5EF
D5E9: 8B 0A       ADDA  #$0A                ; A += #$0A (10 dec);
D5EB: C0 10       SUBB  #$10                ; B -= #$10 (16 dec)
D5ED: 20 F6       BRA   $D5E5

D5EF: 34 02       PSHS  A
D5F1: EB E0       ADDB  ,S+
D5F3: 35 82       PULS  A,PC ;(PUL? PC=RTS)


; A = number to convert to bcd
; returns: A = bcd equivalent

CONVERT_NUMBER_TO_BCD:
D5F5: 34 04       PSHS  B
D5F7: 1F 89       TFR   A,B                 ; B = number
D5F9: 4F          CLRA                      ; A = 0
D5FA: C1 0A       CMPB  #$0A                ; 10?
D5FC: 25 07       BCS   $D605               ; <
D5FE: 8B 10       ADDA  #$10                ; A+=16
```

```
D600: 19          DAA                       ; Decimal adjust A
D601: C0 0A       SUBB  #$0A                ; B-=10
D603: 20 F5       BRA   $D5FA               ;

D605: 34 04       PSHS  B
D607: AB E0       ADDA  ,S+                    ; A = A + B
D609: 19          DAA
D60A: 35 84       PULS  B,PC ;(PUL? PC=RTS)

D60C: 8E 98 4C    LDX   #$984C
D60F: 10 8E D5 60 LDY   #$D560
D613: 20 10       BRA   $D625

D615: 8E 98 4D    LDX   #$984D
D618: 10 8E D5 54 LDY   #$D554
D61C: 20 07       BRA   $D625

D61E: 8E 98 4E    LDX   #$984E
D621: 10 8E D5 5A LDY   #$D55A
D625: 96 4B       LDA   $4B
D627: 26 27       BNE   $D650
D629: A6 84       LDA   ,X
D62B: 26 23       BNE   $D650
D62D: 86 16       LDA   #$16
D62F: A7 84       STA   ,X
D631: 10 AF 49    STY   $0009,U
D634: 86 0A       LDA   #$0A
D636: 8E D6 3C    LDX   #$D63C
D639: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

D63C: 96 4B       LDA   $4B
D63E: 26 10       BNE   $D650
D640: 86 05       LDA   #$05
D642: D6 84       LDB   $84
D644: C4 07       ANDB  #$07
D646: 3D          MUL
D647: C3 D0 CE    ADDD  #$D0CE
D64A: BD D0 4B    JSR   $D04B               ; JMP $D3C7
D64D: AD D8 09    JSR   [$09,U]
D650: 7E D0 63    JMP   $D063               ; JMP $D1F3

D653: 5E          Illegal Opcode
D654: 31

D655: 34 16       PSHS  X,B,A
D657: 86 01       LDA   #$01
D659: 20 02       BRA   $D65D
D65B: 34 16       PSHS  X,B,A
D65D: C4 0F       ANDB  #$0F
D65F: 58          ASLB
D660: 34 04       PSHS  B
D662: 58          ASLB
D663: EB E0       ADDB  ,S+
```

```
D665: 8E CC FC    LDX   #$CCFC
D668: 3A          ABX
D669: BD D0 A5    JSR   $D0A5
D66C: 34 04       PSHS  B
D66E: BD D0 A5    JSR   $D0A5
D671: 34 04       PSHS  B
D673: BD D0 A5    JSR   $D0A5
D676: 34 04       PSHS  B
D678: AB E4       ADDA  ,S
D67A: 19          DAA
D67B: A7 E4       STA   ,S
D67D: A6 61       LDA   $1,S
D67F: 89 00       ADCA  #$00
D681: 19          DAA
D682: A7 61       STA   $1,S
D684: A6 62       LDA   $2,S
D686: 89 00       ADCA  #$00
D688: 19          DAA
D689: 30 1A       LEAX  -$6,X
D68B: BD D0 AB    JSR   $D0AB
D68E: 35 04       PULS  B
D690: 35 02       PULS  A
D692: BD D0 B1    JSR   $D0B1
D695: 35 02       PULS  A
D697: 35 96       PULS  A,B,X,PC ; (PUL? PC=RTS)



;
; Get a pointer to the state of the current player.
; Returns: X = pointer to state
;

LOAD_X_WITH_ADDR_OF_CURRENT_PLAYER_STATE:
D699: 34 02       PSHS  A
D69B: 96 3F       LDA   $3F                   ; read player number
D69D: 8E BD E4    LDX   #p1_score
D6A0: 4A          DECA                      ; player number --
D6A1: 27 03       BEQ   $D6A6               ; if we're player 1
this value will be 0, return
D6A3: 8E BE 20    LDX   #p2_score           ; otherwise we're
player 2...
D6A6: 35 82       PULS  A,PC ;(PUL? PC=RTS)

D6A8: 34 02       PSHS  A
D6AA: 20 F1       BRA   $D69D



;
; A = number to multiply random number with
;

MULTIPLY_A_BY_RANDOM_NUMBER:
D6AC: 34 04       PSHS  B
```

```
D6AE: 1F 89       TFR   A,B
D6B0: 8D 1B       BSR   $D6CD                ; call random number
generator
D6B2: 3D          MUL
D6B3: 4C          INCA
D6B4: 35 84       PULS  B,PC ;(PUL? PC=RTS)


;
; Get a random number lower than or equal to the value in register
A.
;

GET_RANDOM_NUMBER_LOWER_THAN_OR_EQUAL_TO_A:
D6B6: 34 02       PSHS  A                  ; save a on stack
D6B8: 8D 13       BSR   $D6CD              ; get a random number
D6BA: A1 E4       CMPA  ,S                 ; compare random number
to A to on stack
D6BC: 23 03       BLS   $D6C1              ; if random number <= A
on stack, goto $D6C1
D6BE: 44          LSRA                     ; divide random number
by 2
D6BF: 20 F9       BRA   $D6BA              ; repeat compare

D6C1: 4D          TSTA                     ; check if A is 0
D6C2: 26 01       BNE   $D6C5              ; no
D6C4: 4C          INCA                     ; otherwise, make A = 1
D6C5: 32 61       LEAS  $0001,S            ; discard A on stack
D6C7: 39          RTS                      ; return


GET_RANDOM_NUMBER_INTO_A_AND_B:
D6C8: 8D 03       BSR   $D6CD              ; get a random number
D6CA: D6 86       LDB   $86
D6CC: 39          RTS



;
; Pseudo random number generator.
;
; $9884, $9885 and $9886 are affected by this call.
; Returns: A = random number
;

GENERATE_RANDOM_NUMBER:
D6CD: 34 04       PSHS  B
D6CF: D6 84       LDB   $84               ; DP set to 98 here, so
real address is $9884
D6D1: 86 03       LDA   #$03
D6D3: 3D          MUL                     ; D = 3 * B
D6D4: CB 11       ADDB  #$11
D6D6: 96 86       LDA   $86
D6D8: 44          LSRA
```

```
D6D9: 44          LSRA
D6DA: 44          LSRA
D6DB: 98 86       EORA  $86
D6DD: 44          LSRA
D6DE: 06 85       ROR   $85
D6E0: 06 86       ROR   $86
D6E2: DB 86       ADDB  $86
D6E4: D9 85       ADCB  $85
D6E6: D7 84       STB   $84
D6E8: 96 84       LDA   $84                     ; a= "random" number
D6EA: 35 84       PULS  B,PC ;(PUL? PC=RTS)



;
; This routine initialises the object lists
;

INITIALISE_LISTS:
D6EC: 34 56       PSHS  U,X,B,A
D6EE: 4F          CLRA
D6EF: 5F          CLRB                        ; make D = NULL

;
D6F0: 8E A9 E0    LDX   #$A9E0
D6F3: CE 97 6F    LDU   #$976F
D6F6: 9F 13       STX   $13                   ; set main object list
pointer
D6F8: 30 0F       LEAX  $000F,X               ; X+=#$0F (15 decimal)
D6FA: AF 11       STX   -$F,X                 ; store pointer to
object at X in the previous object (X-15 decimal), establishing a
forward only linked list
D6FC: 8C B0 D9    CMPX  #$B0D9
D6FF: 26 F7       BNE   $D6F8
D701: ED 84       STD   ,X                    ; terminate list with
NULL
D703: DD 11       STD   $11

; initialise list used by progs & cruise missiles
D705: 8E B0 E8    LDX   #$B0E8
D708: 9F 1D       STX   $1D
D70A: 30 88 1F    LEAX  $1F,X                 ; X+= #$1F (31 decimal)
D70D: AF 88 E1    STX   -$1F,X
D710: 8C B3 35    CMPX  #$B335
D713: 26 F5       BNE   $D70A
D715: ED 84       STD   ,X                    ; terminate list with
NULL

; Initialise function call list
D717: 8E 98 11    LDX   #$9811                ;
D71A: 9F 15       STX   $15
D71C: C6 07       LDB   #$07
D71E: 1F 01       TFR   D,X
D720: AB 84       ADDA  ,X
```

```
D722: 30 88 10    LEAX  $10,X              ; X = X + #$10 (16
decimal)
D725: 8C 89 35    CMPX  #$8935
D728: 25 F6       BCS   $D720
D72A: A7 C9 01 84 STA   $0184,U
D72E: 35 D6       PULS  A,B,X,U,PC ;(PUL? PC=RTS)

D730: BD D0 60    JSR   $D060             ; JMP $D1FF
D733: 86 FF       LDA   #$FF
D735: 97 59       STA   $59
D737: 86 01       LDA   #$01              ; delay before calling
function
D739: 8E D7 3F    LDX   #$D73F            ; address of function to
call
D73C: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call


SHOW_EASTER_EGG_CREDITS:
D73F: BD D0 12    JSR   CLR_SCREEN1
D742: BD 5F 9C    JSR   $5F9C             ; JMP $5FA2
D745: C6 7F       LDB   #$7F
D747: D7 01       STB   $01
D749: 10 BE D5 D6 LDY   $D5D6
D74D: 31 A5       LEAY  B,Y
D74F: A6 A0       LDA   ,Y+
D751: 27 2C       BEQ   $D77F
D753: 81 02       CMPA  #$02
D755: 26 09       BNE   $D760
D757: B6 C8 04    LDA   widget_pia_dataa
D75A: 85 40       BITA  #$40              ; fire UP pressed?
D75C: 27 30       BEQ   $D78E             ; no, goto $D78E
D75E: 20 EF       BRA   $D74F

D760: 81 01       CMPA  #$01
D762: 26 06       BNE   $D76A
D764: AE A1       LDX   ,Y++
D766: 0F D0       CLR   $D0
D768: 20 E5       BRA   $D74F

D76A: 8B 2E       ADDA  #$2E
D76C: CE D7 4F    LDU   #$D74F
D76F: 34 40       PSHS  U
D771: FE D6 53    LDU   $D653
D774: F6 D7 ED    LDB   $D7ED
D777: 33 C5       LEAU  B,U
D779: 33 C5       LEAU  B,U
D77B: 33 C5       LEAU  B,U
D77D: 6E C4       JMP   ,U

D77F: 86 01       LDA   #$01
D781: 8E D7 87    LDX   #$D787
D784: 7E D0 66    JMP   $D066             ; JMP $D1E3 - allocate
function call
```

```
D787: B6 C8 04    LDA   widget_pia_dataa
D78A: 85 40       BITA  #$40                ; fire UP pressed?
D78C: 26 F1       BNE   $D77F               ; yes, goto $D77F
D78E: BD D0 12    JSR   CLR_SCREEN1         ; otherwise clear the
screen
D791: 6E 9F EF FE JMP   [$EFFE,X]           ; and reboot!!!

LOAD_DA51_PALETTE:
D795: 8E DA 51    LDX   #colorpalette2
D798: CE 98 00    LDU   #$9800
D79B: C6 10       LDB   #$10
D79D: A6 80       LDA   ,X+
D79F: A7 C0       STA   ,U+
D7A1: 5A          DECB
D7A2: 26 F9       BNE   $D79D
D7A4: 39          RTS

D7A5: 34 17       PSHS  X,B,A,CC
D7A7: 1A FF       ORCC  #$FF
D7A9: 8E 99 00    LDX   #$9900              ; set start of object
linked list and store in $981B
D7AC: 9F 1B       STX   $1B
D7AE: 30 88 18    LEAX  $18,X               ; add #$18 to X
D7B1: AF 88 E8    STX   -$18,X              ; set *(X-#$18) to X -
this is to establish a forward-only linked list
D7B4: 8C A9 C8    CMPX  #$A9C8
D7B7: 26 F5       BNE   $D7AE
D7B9: 4F          CLRA                      ; set D to 0
D7BA: 5F          CLRB
D7BB: ED 84       STD   ,X                  ; mark end of list with
two zeros
D7BD: DD 21       STD   $21
D7BF: DD 17       STD   $17                 ; zero all objects
except player linked list pointer
D7C1: DD 19       STD   $19
D7C3: DD 23       STD   $23                 ; zero electrodes
linked list pointer
D7C5: DD 1F       STD   $1F                 ; zero family linked
list pointer
D7C7: 35 97       PULS  CC,A,B,X,PC ;(PUL? PC=RTS)


;
; Checks for a collision between the main object (object ONE) and a
list of other objects.
;
; D = blitter destination of object ONE.
; X = pointer to linked list of objects to compare object ONE
against.
; U = animation frame metadata pointer of object ONE.
; $48 = 1 if its the player collision detection routine calling this
function, 0 otherwise
;
```

```
; Returns:
;
;


COLLISION_DETECTION_FUNCTION:
D7C9: DD 7C       STD   $7C                 ; save blitter
destination of object ONE to $7C
D7CB: E3 C4       ADDD  ,U                  ; add in width and
height of object ONE
D7CD: DD 7E       STD   $7E                 ; store in $7E. Think
of $7C to $7F defining the rectangular area that object ONE occupies
now.
D7CF: 20 17       BRA   $D7E8
; X = pointer to object OTHER - the object that *may* have collided
with object ONE
D7D1: EC 04       LDD   $0004,X             ; get blitter
destination of object OTHER to compare against object ONEs
boundaries
D7D3: 27 13       BEQ   $D7E8               ; if NULL, process next
object

; perform rectangle intersection check.
; $7C,$7D = X and Y coordinates of top left of object ONE'S
rectangle
; $7E,$7F = X and Y coordinates of bottom right of object ONE'S
rectangle
; D = blitter destination of object OTHER
D7D5: 91 7E       CMPA  $7E                 ; compare A (the X
component) to bottom right X coordinate of rectangle
D7D7: 24 0F       BCC   $D7E8               ; if A is >= this value
then there is no intersection, goto $D7E8
D7D9: D1 7F       CMPB  $7F                 ; compare B (the Y
component) to bottom right Y coordinate of rectangle
D7DB: 24 0B       BCC   $D7E8               ; if B is >= this value
then there is no intersection, goto $D7E8
D7DD: E3 98 02    ADDD  [$02,X]             ; D+= width & height of
object being compared to object ONE
D7E0: 91 7C       CMPA  $7C                 ; compare A to top left
X coordinate of rectangle
D7E2: 23 04       BLS   $D7E8               ; if A <= this value
then there is no intersection, goto $D7E8
D7E4: D1 7D       CMPB  $7D                 ; compare B to top left
Y coordinate of rectangle
D7E6: 22 06       BHI   $D7EE               ; if > then a possible
collision

; if we get here, no collision has taken place, so get next object
in the list, and try that
D7E8: AE 84       LDX   ,X                  ; get next object in
list
D7EA: 26 E5       BNE   $D7D1               ; if object is not
NULL, go to $D7D1
D7EC: 39          RTS                       ; otherwise we're done
```

```
D7ED: Unused byte
; At this point:
; U = animation frame metadata pointer of object ONE
; D = blitter destination of object OTHER
; X = pointer to object OTHER
; $7C,$7D = X and Y coordinates of top left of rectangle of object
ONE
; $7E,$7F = X and Y coordinates of bottom right of rectangle of
object ONE
D7EE: DF 82       STU   $82                ; store animation frame
metadata pointer
D7F0: 0D 48       TST   $48                ; is it the player
that's calling this function? (see $30B3)
D7F2: 26 06       BNE   $D7FA              ; yes
D7F4: 10 AE 88 16 LDY   $16,X              ; has collision
detection animation frame metadata been supplied for this object?
D7F8: 26 03       BNE   $D7FD              ; yes, so use the width
and height in the metadata, goto $D7FD
D7FA: 10 AE 02    LDY   $0002,X            ; get pointer to
animation frame metadata of object that we collided with
D7FD: A3 A4       SUBD  ,Y                 ; subtract width &
height of object that might have collided, from its blitter
destination
D7FF: 10 9F 2D    STY   $2D                ; store animation frame
metadata pointer in $2D
D802: DD 2B       STD   $2B                ; store adjusted blitter
destination to $2B
D804: 4F          CLRA                     ; D= 0
D805: 5F          CLRB
D806: DD 76       STD   $76                ; $76,$77 = 0
D808: DD 78       STD   $78                ; $78,$79 = 0
; at this point:
; U = animation frame metadata pointer of object ONE
; X = pointer to object OTHER
; Y = pointer to animation frame metadata of object OTHER
; $2B = adjusted (see $D7FD) blitter destination of object OTHER
; $2D = animation frame metadata pointer of object OTHER
; $7C,$7D = X and Y coordinate of top left of rectangle of object
ONE
; $7E,$7F = X and Y coordinate of bottom right of rectangle of
object ONE
D80A: DC 2B       LDD   $2B                ; D = adjusted blitter
destination of object OTHER (see $D7FD)
D80C: D0 7D       SUBB  $7D                ; B (the Y component of
blitter destination) -= top Y coordinate of object ONE, to give
vertical distance in pixels
D80E: 22 05       BHI   $D815              ; if no carry after
subtraction and non-zero result, ie distance is a non zero positive
number, goto $D815
D810: 50          NEGB                     ; Make B a positive
number
D811: D7 77       STB   $77                ; $77 = B
D813: 20 02       BRA   $D817
```

```
D815: D7 79       STB   $79                 ; $79 = B
; now do horizontal axis
D817: 90 7C       SUBA  $7C                 ; A (the X component of
blitter destination) -= left X coordinate of object ONE, to give
horizontal distance in pixels div 2
D819: 22 05       BHI   $D820               ; if no carry after
subtraction and non-zero result, ie distance is a non zero positive
number, goto $D815
D81B: 40          NEGA                      ; Make A a positive
number
D81C: 97 76       STA   $76                 ; $76 = A
D81E: 20 02       BRA   $D822

D820: 97 78       STA   $78

D822: DC 2B       LDD   $2B                 ; get adjusted blitter
destination of object OTHER (see $D7FD) into D
D824: E3 A4       ADDD  ,Y                  ; and add width and
height of image, which restores D back to the *real* blitter
destination of object OTHER
D826: D0 7F       SUBB  $7F                 ; B-= Y coordinate of
bottom right of rectangle of object ONE
D828: 22 01       BHI   $D82B               ; if result is a non-
zero positive number, goto $D82B
D82A: 5F          CLRB                      ; otherwise result is 0
or negative, so set B to 0
D82B: 90 7E       SUBA  $7E                 ;
D82D: 22 01       BHI   $D830               ; if result is a non-
zero positive number, goto $D82B
D82F: 4F          CLRA                      ; otherwise result is 0
or negative, so set A to 0
D830: DD 80       STD   $80
; at this point:
; U = animation frame metadata pointer of object ONE
; X = pointer to object OTHER
; Y = pointer to animation frame metadata of object OTHER
; $2B = adjusted (see $D7FD) blitter destination of object OTHER
; $2D = animation frame metadata pointer of object OTHER
D832: EC A4       LDD   ,Y                  ; get width and height
of object OTHER into D
D834: 93 76       SUBD  $76
D836: 93 80       SUBD  $80
D838: DD 74       STD   $74
D83A: A6 C4       LDA   ,U                  ; get width of object
ONE
D83C: 97 7B       STA   $7B
D83E: D6 79       LDB   $79
D840: 3D          MUL
D841: EE 42       LDU   $0002,U             ; U now = pointer to
very first 2 pixels of animation frame for object ONE
D843: 33 CB       LEAU  D,U                 ; U = U + D
D845: A6 A4       LDA   ,Y
D847: 97 7A       STA   $7A
D849: D6 77       LDB   $77
```

```
D84B: 3D          MUL
D84C: 10 AE 22    LDY   $0002,Y             ; Y = pointer to very
first 2 pixels of animation frame for object OTHER
D84F: 31 AB       LEAY  D,Y                 ; Y = Y + D
D851: 96 76       LDA   $76
D853: 31 A6       LEAY  A,Y                 ; Y = Y + A
D855: 96 78       LDA   $78
D857: 33 C6       LEAU  A,U                 ; U = U + A
D859: D6 74       LDB   $74
D85B: 5A          DECB
D85C: A6 C5       LDA   B,U                 ; read pixels at U + B
D85E: 27 2C       BEQ   $D88C               ; if 0 then consider
these pixels as transparent, do not use in collision detection, goto
$D88C
D860: A6 A5       LDA   B,Y                 ; read pixels at Y + B
D862: 27 28       BEQ   $D88C               ; if 0 then consider
these pixels as transparent, do not use in collision detection, goto
$D88C
D864: 31 A5       LEAY  B,Y                 ; Y = Y + B
D866: 1F 20       TFR   Y,D
D868: DE 2D       LDU   $2D                 ; U - pointer to
animation frame metadata of object OTHER
D86A: A3 42       SUBD  $0002,U
D86C: 10 AE 04    LDY   $0004,X             ; Y = blitter
destination of object OTHER
D86F: E0 C4       SUBB  ,U
D871: 82 00       SBCA  #$00
D873: 25 08       BCS   $D87D
D875: 31 21       LEAY  $0001,Y             ; Y++
D877: E0 C4       SUBB  ,U
D879: 82 00       SBCA  #$00
D87B: 24 F8       BCC   $D875
D87D: EB C4       ADDB  ,U
D87F: 1F 98       TFR   B,A
D881: 5F          CLRB
D882: 33 AB       LEAU  D,Y
D884: DF A6       STU   $A6
D886: AD 98 08    JSR   [$08,X]              ; call routine to
handle collision the object pointed to by X
D889: 86 01       LDA   #$01
D88B: 39          RTS

D88C: 5A          DECB
D88D: 2A CD       BPL   $D85C
D88F: DC 7A       LDD   $7A
D891: 31 A6       LEAY  A,Y
D893: 33 C5       LEAU  B,U
D895: 0A 75       DEC   $75
D897: 26 C0       BNE   $D859
D899: DE 82       LDU   $82
D89B: 7E D7 E8    JMP   $D7E8

D89E: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
```

```
D8A1: D9 DF       ; pointer to function
D8A3: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
D8A6: D9 D2       ; pointer to function
D8A8: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
D8AB: DA 0D       ; pointer to function
D8AD: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
D8B0: D9 81       ; pointer to function
D8B2: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
D8B5: D9 AE       ; pointer to function
D8B7: BD D0 54    JSR   $D054               ; JMP $D281 - reserve
object metadata entry and call function
D8BA: D9 8E       ; pointer to function
D8BC: 39          RTS

D8BD: 01          Illegal Opcode
D8BE: 0C 28       INC   $28
D8C0: 26 1A 1B 25 0C 1B 25 0C 24 21 14 21 26 24 21 20 ;Easter Egg
Text (-#$2E)
D8D0: 11 0C 04 21 0A 06 02 01 0C 58 16 17 25 1B 19 20
D8E0: 17 16 0C 17 2A 15 1E 27 25 1B 28 17 1E 2B 0C 18
D8F0: 21 24 01 0C 68 29 1B 1E 1E 1B 13 1F 25 0C 17 1E
D900: 17 15 26 24 21 20 1B 15 25 0C 1B 20 15 0F 02 01
D910: 0C 78 14 2B 0C 17 27 19 17 20 17 0C 22 0F 0C 1C
D920: 13 24 28 1B 25 0C 13 20 16 0C 1E 13 29 24 17 20
D930: 15 17 0C 17 0F 0C 16 17 1F 13 24 02 01 0C A8 15
D940: 21 22 2B 24 1B 19 1A 26 0C 03 0B 0A 04 0C 29 1B
D950: 1E 1E 1B 13 1F 25 0C 17 1E 17 15 26 24 21 20 1B
D960: 15 25 0C 1B 20 15 0F 01 0C B8 13 1E 1E 0C 24 1B
D970: 19 1A 26 25 0C 24 17 25 17 24 28 17 16

D97D: 00 17       NEG   $17
D97F: 22 30       BHI   $D9B1
D981: BD D9 E8    JSR   $D9E8
D984: D9 8A       ADCB  $8A
D986: 98 0B       EORA  $0B
D988: 00 08       NEG   $08
D98A: 38          Illegal Opcode
D98B: 07 C0       ASR   $C0
D98D: 00 BD       NEG   $BD
D98F: D9 E8       ADCB  $E8
D991: D9 97       ADCB  $97
D993: 98 0C       EORA  $0C
D995: 00 02       NEG   $02
D997: C0 C0       SUBB  #$C0
D999: D0 E0       SUBB  $E0
D99B: F0 F8 FA    SUBB  $F8FA
D99E: BA 7A 3A    ORA   $7A3A
D9A1: 34 2D       PSHS  Y,DP,B,CC
D9A3: 1F 17       TFR   X,inv
D9A5: 0F 07       CLR   $07
```

```
D9A7: 06 05       ROR   $05
D9A9: 04 03       LSR   $03
D9AB: 02          Illegal Opcode
D9AC: 01          Illegal Opcode
D9AD: 00 BD       NEG   $BD
D9AF: D9 E8       ADCB  $E8
D9B1: D9 B7       ADCB  $B7
D9B3: 98 0E       EORA  $0E
D9B5: 00 01       NEG   $01
D9B7: C0 C1       SUBB  #$C1
D9B9: C2 C3       SBCB  #$C3
D9BB: C4 C5       ANDB  #$C5
D9BD: C6 C7       LDB   #$C7
D9BF: 87          Illegal Opcode
D9C0: 87          Illegal Opcode
D9C1: 47          ASRA
D9C2: 47          ASRA
D9C3: 07 07       ASR   $07
D9C5: 47          ASRA
D9C6: 47          ASRA
D9C7: 87          Illegal Opcode
D9C8: 87          Illegal Opcode
D9C9: C7          Illegal Opcode
D9CA: C7          Illegal Opcode
D9CB: C6 C5       LDB   #$C5
D9CD: C4 C3       ANDB  #$C3
D9CF: C2 C1       SBCB  #$C1
D9D1: 00 BD       NEG   $BD
D9D3: D9 E8       ADCB  $E8
D9D5: D9 DB       ADCB  $DB
D9D7: 98 0F       EORA  $0F
D9D9: 00 06       NEG   $06
D9DB: 07 07       ASR   $07
D9DD: 2F 00       BLE   $D9DF
D9DF: BD D9 E8    JSR   $D9E8
D9E2: DA 2C       ORB   $2C
D9E4: 98 0D       EORA  $0D
D9E6: 00 02       NEG   $02
D9E8: AE E1       LDX   ,S++
D9EA: EC 81       LDD   ,X++
D9EC: ED 47       STD   $0007,U
D9EE: EC 81       LDD   ,X++
D9F0: ED 4B       STD   $000B,U
D9F2: EC 81       LDD   ,X++
D9F4: ED 49       STD   $0009,U
D9F6: 6F 49       CLR   $0009,U
D9F8: AE 47       LDX   $0007,U
D9FA: E6 49       LDB   $0009,U
D9FC: 6C 49       INC   $0009,U
D9FE: A6 85       LDA   B,X
DA00: 27 F4       BEQ   $D9F6
DA02: A7 D8 0B    STA   [$0B,U]
DA05: A6 4A       LDA   $000A,U
DA07: 8E D9 F8    LDX   #$D9F8
```

```
DA0A: 7E D0 66    JMP   $D066              ; JMP $D1E3 - allocate
function call

DA0D: 86 FF       LDA   #$FF
DA0F: 97 0A       STA   $0A
DA11: 86 02       LDA   #$02
DA13: 8E DA 19    LDX   #$DA19
DA16: 7E D0 66    JMP   $D066              ; JMP $D1E3 - allocate
function call

DA19: 96 84       LDA   $84
DA1B: 84 1F       ANDA  #$1F
DA1D: 8E DA 2C    LDX   #$DA2C
DA20: A6 86       LDA   A,X
DA22: 97 0A       STA   $0A
DA24: 86 06       LDA   #$06
DA26: 8E DA 0D    LDX   #$DA0D
DA29: 7E D0 66    JMP   $D066              ; JMP $D1E3 - allocate
function call

DA2C: 38          Illegal Opcode
DA2D: 39          RTS

DA2E: 3A          ABX
DA2F: 3B          RTI

DA30: 3C 3D       CWAI  #$3D
DA32: 3E          Illegal Opcode
DA33: 3F          SWI
DA34: 37 2F       PULU  CC,A,B,DP,Y
DA36: 27 1F       BEQ   $DA57
DA38: 17 47 47    LBSR  $2182
DA3B: 87          Illegal Opcode
DA3C: 87          Illegal Opcode
DA3D: C7          Illegal Opcode
DA3E: C7          Illegal Opcode
DA3F: C6 C5       LDB   #$C5
DA41: CC CB CA    LDD   #$CBCA
DA44: DA E8       ORB   $E8
DA46: F8 F9 FA    EORB  $F9FA
DA49: FB FD FF    ADDB  $FDFF
DA4C: BF 3F 3E    STX   $3F3E
DA4F: 3C 00       CWAI  #$00

colorpalette2:
DA51: 00 07 17 C7 1F 3F 38 C0 A4 FF 38 17 CC 81 81 07


;
; D = blitter destination
; Y = animation frame metadata pointer (first 2 bytes are w & h,
next 2 bytes are blit source address)
; $2D = remap colour
```

```
BLIT_RECTANGLE_WITH_COLOUR_REMAP:
DA61: 34 07       PSHS  B,A,CC
DA63: 1A 10       ORCC  #$10
DA65: FD CA 04    STD   blitter_dest
DA68: EC A4       LDD   ,Y                          ; get width &
height into D
DA6A: 88 04       EORA  #$04
DA6C: C8 04       EORB  #$04
DA6E: FD CA 06    STD   blitter_w_h
DA71: EC 22       LDD   $0002,Y                      ; get blitter
source into D
DA73: FD CA 02    STD   blitter_source
DA76: D6 2D       LDB   $2D
DA78: F7 CA 01    STB   blitter_mask
DA7B: 86 12       LDA   #$12                       ; solid mode
DA7D: B7 CA 00    STA   start_blitter
DA80: 35 87       PULS  CC,A,B,PC ;(PUL? PC=RTS)



;
; D = blitter destination
; Y = pointer to image struct (first 2 bytes are w & h, next 2 are
pointer to blitter source)
;

BLIT_IMAGE_NO_TRANSPARENCY:
DA82: 34 07       PSHS  B,A,CC
DA84: 1A 10       ORCC  #$10
DA86: FD CA 04    STD   blitter_dest
DA89: EC A4       LDD   ,Y                   ; get width & height
into D
DA8B: 88 04       EORA  #$04
DA8D: C8 04       EORB  #$04
DA8F: FD CA 06    STD   blitter_w_h
DA92: EC 22       LDD   $0002,Y               ; get blitter source
into D
DA94: FD CA 02    STD   blitter_source
DA97: 86 02       LDA   #$02                ; draw image without
transparency
DA99: B7 CA 00    STA   start_blitter
DA9C: 35 87       PULS  CC,A,B,PC ;(PUL? PC=RTS)



; Blit an image in a solid colour. Basically take an image and remap
all the nonzero bytes
; to the colour specified in $2D.
;
; D = blitter destination
; Y = pointer to animation frame metadata (first 2 bytes are w & h,
next 2 are pointer to blitter source)
; $2D = value to use in solid colour blit
```

```
BLIT_IMAGE_IN_SOLID_COLOUR_AND_TRANSPARENCY:
DA9E: 34 07       PSHS  B,A,CC
DAA0: 1A 10       ORCC  #$10
DAA2: FD CA 04    STD   blitter_dest         ; set blitter
destination
DAA5: EC A4       LDD   ,Y
DAA7: 88 04       EORA  #$04
DAA9: C8 04       EORB  #$04
DAAB: FD CA 06    STD   blitter_w_h          ; set blitter width &
height
DAAE: EC 22       LDD   $0002,Y              ; get blitter source
DAB0: FD CA 02    STD   blitter_source
DAB3: D6 2D       LDB   $2D
DAB5: F7 CA 01    STB   blitter_mask         ; set solid colour to
draw
DAB8: 86 1A       LDA   #$1A                 ; blitter flags: solid
mode, transparent
DABA: B7 CA 00    STA   start_blitter
DABD: 35 87       PULS  CC,A,B,PC ;(PUL? PC=RTS)


; Clear a rectangular area matching the supplied animation frame
metadata's width & height fields.
;
; D = blitter destination
; Y = pointer to animation frame metadata (first 2 bytes are width &
height, next 2 are pointer to blitter source)
;

CLEAR_IMAGE_RECTANGLE_TO_ZERO:
DABF: 34 07       PSHS  B,A,CC
DAC1: 1A 10       ORCC  #$10
DAC3: FD CA 04    STD   blitter_dest
DAC6: EC A4       LDD   ,Y                   ; get width and height
of image into D
DAC8: 88 04       EORA  #$04
DACA: C8 04       EORB  #$04
DACC: FD CA 06    STD   blitter_w_h
DACF: EC 22       LDD   $0002,Y              ; get pointer to actual
image into D

DAD1: FD CA 02    STD   blitter_source
DAD4: CC 12 00    LDD   #$1200               ; solid mode, no
transparency. Note that bit 3 is missing, so blitter is clearing
rectangle
DAD7: F7 CA 01    STB   blitter_mask         ; set 0 as mask colour
(background)
DADA: B7 CA 00    STA   start_blitter
DADD: 35 87       PULS  CC,A,B,PC ;(PUL? PC=RTS)


; Clears a specified area (to black).
;
; X = Blitter dest
```

```
; D = Width & Height
;

CLEAR_RECTANGULAR_AREA:
DADF: 34 07       PSHS  B,A,CC
DAE1: 1A 10       ORCC  #$10
DAE3: BF CA 04    STX   blitter_dest
DAE6: 88 04       EORA  #$04
DAE8: C8 04       EORB  #$04
DAEA: FD CA 06    STD   blitter_w_h
DAED: CC 00 00    LDD   #$0000              ; set 0 as source, it
doesn't matter for this routine
DAF0: 20 DF       BRA   $DAD1


;
; This routine is called when you want to blit an object with
transparency, no extra effects.
; Family members, grunts etc use this routine to draw themselves, so
it's quite an important routine to observe.
;
; X = object pointer
;

BLIT_OBJECT:
DAF2: 34 47       PSHS  U,B,A,CC
DAF4: EE 02       LDU   $0002,X             ; U = pointer to
animation frame metadata
DAF6: EC C4       LDD   ,U                  ; D = width & height
DAF8: 88 04       EORA  #$04
DAFA: C8 04       EORB  #$04
DAFC: 1A 10       ORCC  #$10
DAFE: FD CA 06    STD   blitter_w_h
DB01: 20 58       BRA   $DB5B               ; Finalise blit


;
; X = pointer to object
;

ERASE_PREVIOUS_IMAGE:
DB03: 34 47       PSHS  U,B,A,CC
DB05: EE 88 14    LDU   $14,X               ; get previous
animation frame metadata pointer into U
DB08: EC C4       LDD   ,U                  ; get width and height
DB0A: EE 42       LDU   $0002,U             ; get actual image data
pointer into U
DB0C: 88 04       EORA  #$04
DB0E: C8 04       EORB  #$04
DB10: 1A 10       ORCC  #$10
DB12: FD CA 06    STD   blitter_w_h
DB15: EC 04       LDD   $0004,X             ; get current object's
"last" blit destination
DB17: FD CA 04    STD   blitter_dest
```

```
DB1A: FF CA 02    STU   blitter_source       ; set blitter source to
actual image data pointer in U
DB1D: CC 1A 00    LDD   #$1A00               ; blitter flags:solid
mode, transparent. B= colour 0 (background) to overwrite
DB20: F7 CA 01    STB   blitter_mask
DB23: E6 88 12    LDB   $12,X
DB26: 2A 02       BPL   $DB2A
DB28: 8A 20       ORA   #$20                 ; one pixel to the
right
DB2A: B7 CA 00    STA   start_blitter
DB2D: 35 C7       PULS  CC,A,B,U,PC ;(PUL? PC=RTS)


; Erase previous animation frame at old (previous) position before
object moved,
; draw new animation frame at new object position (where object
moved to)
; X = pointer to object (grunt, brain etc) being processed
;

ERASE_THEN_DRAW_NEW:
DB2F: 34 47       PSHS  U,B,A,CC
DB31: EE 88 14    LDU   $14,X                ; get previous
animation frame metadata pointer into U
DB34: EC C4       LDD   ,U                   ; D = width and height
DB36: EE 42       LDU   $0002,U              ; get image data
pointer into U
DB38: 88 04       EORA  #$04
DB3A: C8 04       EORB  #$04
DB3C: 1A 10       ORCC  #$10
DB3E: FD CA 06    STD   blitter_w_h          ; set width & height on
blitter
DB41: EC 04       LDD   $0004,X              ; read "last" blitter
destination from object
DB43: FD CA 04    STD   blitter_dest
DB46: FF CA 02    STU   blitter_source
DB49: CC 1A 00    LDD   #$1A00               ; A: 1A = solid mode +
transparency mode)
DB4C: F7 CA 01    STB   blitter_mask         ; B: 0 (colour 0,
black)
DB4F: E6 88 12    LDB   $12,X
DB52: 2A 02       BPL   $DB56
DB54: 8A 20       ORA   #$20                 ; shift image one pixel
to right
DB56: B7 CA 00    STA   start_blitter        ; erase object
DB59: EE 02       LDU   $0002,X              ; get pointer to
current animation frame metadata


; U = pointer to animation frame metadata
DB5B: EF 88 14    STU   $14,X                ; set pointer to
previous animation frame metadata (making previous = current)
DB5E: EE 42       LDU   $0002,U              ; get pointer to actual
previous image to draw
```

```
DB60: FF CA 02    STU   blitter_source
DB63: A6 0A       LDA   $000A,X              ; read object X
coordinate (whole part)
DB65: E6 0C       LDB   $000C,X              ; read object Y
coordinate
DB67: ED 04       STD   $0004,X              ; set "last" blitter
destination pointer
DB69: FD CA 04    STD   blitter_dest
DB6C: 86 0A       LDA   #$0A                 ; transparency mode
DB6E: E6 0B       LDB   $000B,X
DB70: E7 88 12    STB   $12,X                ; set flag to say image
needs shifted pixel right (bit 7 set = Yes)
DB73: 2A 02       BPL   $DB77
DB75: 8A 20       ORA   #$20                 ; shift image a pixel
right
DB77: B7 CA 00    STA   start_blitter        ; draw object in new
position
DB7A: 35 C7       PULS  CC,A,B,U,PC ;(PUL? PC=RTS)

CLR_SCREEN:
DB7C: 34 76       PSHS  U,Y,X,B,A
DB7E: CE 98 00    LDU   #$9800
DB81: 8E 00 00    LDX   #$0000               ; X = 0
DB84: 1F 12       TFR   X,Y                  ; Y = X
DB86: 1F 10       TFR   X,D                  ; D = X
DB88: 36 36       PSHU  Y,X,B,A              ; write Y,X,D (all 0,
where 0 = black pixel) to screen RAM
DB8A: 36 36       PSHU  Y,X,B,A
DB8C: 36 36       PSHU  Y,X,B,A
DB8E: 36 36       PSHU  Y,X,B,A
DB90: 36 36       PSHU  Y,X,B,A
DB92: 36 10       PSHU  X
DB94: 11 83 00 00 CMPU  #$0000               ; are we at the start
of screen RAM
DB98: 26 EE       BNE   $DB88                ; no
DB9A: 35 F6       PULS  A,B,X,Y,U,PC ;(PUL? PC=RTS)



; Updates the player score.
;
; B= a number (in BCD) and A is the number of trailing zeros to put
after it.
; Example, B=#$15 and A=3  means 3 trailing zeros = 15000 decimal to
add to score.
;
; to add 100 to score, set a=1, b=#$10
; to add 150 to score, set a=1, b=#$15
; to add 1000 to score, set a= 2, b = #$10
; to add 1500 to score, set a= 2, b = #$15
; to add 10000 to score, set a = 3, b = #$10
; to add 100000 to score, set a = 4, b = #$10
; to add 150000 to score, set a = 4, b = #$15
;
```

```
; I'm sure you get the drift!!!

UPDATE_PLAYER_SCORE:
DB9C: 34 76       PSHS  U,Y,X,B,A
DB9E: 0C F0       INC   $F0
DBA0: 44          LSRA                    ; divide number of
trailing zeros by 2 - will also set carry if number of trailing
zeros is odd number
DBA1: 34 02       PSHS  A                 ; save on stack
DBA3: 86 00       LDA   #$00
DBA5: 24 08       BCC   $DBAF
DBA7: 58          ASLB                    ; move 4 most
significant bits of B into 4 least significant bits of A
DBA8: 49          ROLA
DBA9: 58          ASLB
DBAA: 49          ROLA
DBAB: 58          ASLB
DBAC: 49          ROLA
DBAD: 58          ASLB
DBAE: 49          ROLA
DBAF: BD D0 45    JSR   $D045            ; JMP $D699 - get addr
of current player game state into X. In this case X = pointer to
score
DBB2: DD 2B       STD   $2B
DBB4: C6 03       LDB   #$03
DBB6: E0 E0       SUBB  ,S+               ; subtract index on
stack from B. B is now the index into the first digit to update.
DBB8: A6 85       LDA   B,X               ; A = *(X + B)
DBBA: 9B 2C       ADDA  $2C
DBBC: 19          DAA                     ; remember, we're
working with BCD here, so have to use Decimal Adjust A
DBBD: A7 85       STA   B,X               ; update score digit at
(X+B)
DBBF: 5A          DECB                    ; point to previous
digit (ie: if we're at the tens digit, goto the hundreds digit, if
we're at the hundreds digit go to the thousands and so on)
DBC0: 2B 0E       BMI   $DBD0             ; if b==-1 then goto
$DBD0
DBC2: A6 85       LDA   B,X               ; A = *(X+B)
DBC4: 99 2B       ADCA  $2B               ; if we had a carry from
the last add, then be sure to take that into account
DBC6: 19          DAA                     ; and again, ensure that
we have valid BCD digits
DBC7: A7 85       STA   B,X               ; update score digit at
(X+B)
DBC9: 86 00       LDA   #$00              ; OK, we set the value
to carry over to zero. So only carry flag will be added to remaining
digits
DBCB: 97 2B       STA   $2B
DBCD: 5A          DECB                    ; point to previous
digit (ie: if we're at the tens digit, goto the hundreds digit, if
we're at the hundreds digit go to the thousands and so on)
DBCE: 2A F2       BPL   $DBC2             ; if b>=0 then goto
$DBC2, to add the carry flag to next digit(s)
```

```
; this part of the code checks to see if we get a new life or not.
DBD0: DC 46       LDD   $46                 ; read XYZ value of
"bonus life every XYZ". 20000 decimal will be represented as D=
$2000, 25000 will be represented as D=$2500
DBD2: 27 38       BEQ   $DC0C
DBD4: 31 04       LEAY  $0004,X             ; Y = X + 4
DBD6: EC 84       LDD   ,X                  ; read first 4 digits of
score (from left to right)
DBD8: 10 A3 A4    CMPD  ,Y
DBDB: 26 05       BNE   $DBE2
DBDD: EC 02       LDD   $0002,X
DBDF: 10 A3 22    CMPD  $0002,Y
DBE2: 25 28       BCS   $DC0C
DBE4: A6 22       LDA   $0002,Y
DBE6: 9B 47       ADDA  $47
DBE8: 19          DAA
DBE9: A7 22       STA   $0002,Y
DBEB: A6 21       LDA   $0001,Y
DBED: 99 46       ADCA  $46
DBEF: 19          DAA
DBF0: A7 21       STA   $0001,Y
DBF2: A6 A4       LDA   ,Y
DBF4: 89 00       ADCA  #$00
DBF6: 19          DAA
DBF7: A7 A4       STA   ,Y
DBF9: CC D0 C9    LDD   #$D0C9
DBFC: BD D0 4B    JSR   $D04B               ; JMP $D3C7
DBFF: BD D0 45    JSR   $D045               ; JMP $D699 - get addr
of current player game state into X
DC02: 6C 08       INC   $0008,X             ; increment player lives
DC04: BD 26 C9    JSR   $26C9               ; JMP $34E0 - draw
player lives remaining
DC07: C6 05       LDB   #$05
DC09: BD D0 BD    JSR   $D0BD               ; JMP $D655
DC0C: 8D 03       BSR   $DC11
DC0E: 35 76       PULS  A,B,X,Y,U
DC10: 39          RTS



DC11: 96 3F       LDA   $3F                 ; read current player


;
; A = number of players
;

DRAW_PLAYER_SCORES:
DC13: C6 11       LDB   #$11                ; colour to draw player
score in
DC15: 91 3F       CMPA  $3F                 ; compare to current
player
DC17: 26 02       BNE   $DC1B
```

```
DC19: C6 AA       LDB   #$AA                ; colour to draw player
score in
DC1B: 34 02       PSHS  A
DC1D: D7 CF       STB   $CF                 ; save colour
DC1F: 4A          DECA                      ; decrement number of
players by 1.
DC20: 26 08       BNE   $DC2A               ; if the zero flag is
not set, then this must be a 2 player game, goto $DC2A
DC22: 8E 18 0E    LDX   #$180E              ; blitter destination
DC25: CE BD E4    LDU   #p1_score
DC28: 20 06       BRA   $DC30

DC2A: 8E 58 0E    LDX   #$580E              ; blitter destination
DC2D: CE BE 20    LDU   #p2_score
DC30: CC 15 06    LDD   #$1506              ; Width = 15, height = 6
DC33: BD D0 1B    JSR   $D01B               ; JMP $DADF - clear
rectangle to black
DC36: 30 89 FD 00 LEAX  $-300,X             ; make X point to screen
address 6 pixels to the left
DC3A: 0F D6       CLR   $D6
; U = pointer to score to render
DC3C: A6 C4       LDA   ,U                  ; get millions part of
score
DC3E: 84 0F       ANDA  #$0F                ; mask off lower nibble
DC40: BD 5F 9F    JSR   $5F9F               ; JMP
$6096             ; draw millions part of score
DC43: A6 41       LDA   $0001,U
DC45: BD 5F 9F    JSR   $5F9F               ; JMP
$6096             ; draw hundred thousands part of score
DC48: A6 42       LDA   $0002,U
DC4A: BD 5F 9F    JSR   $5F9F               ; JMP
$6096             ; draw thousands and hundreds part of score
DC4D: 0C D6       INC   $D6
DC4F: A6 43       LDA   $0003,U
DC51: BD 5F 9F    JSR   $5F9F               ; JMP
$6096             ; draw tens part of score
DC54: 35 82       PULS  A,PC ;(PUL? PC=RTS)


;
; Interrupt handler
;
; Draws and updates spheroids, enforcers, sparks and player
;
;

DC56: B6 C8 0E    LDA   rom_pia_datab
DC59: 86 01       LDA   #$01
DC5B: 9A 45       ORA   $45
DC5D: B7 C9 00    STA   rom_enable_scr_ctrl ; change rom access
state
DC60: B6 CB 00    LDA   vidctrs               ; read beam counter
(raster vertical position)
DC63: 81 80       CMPA  #$80
```

```
DC65: 25 32       BCS   $DC99               ; if beam counter < #$80
(128 decimal) goto $DC99
DC67: 96 43       LDA   $43
DC69: 26 20       BNE   $DC8B
DC6B: 0C 43       INC   $43
DC6D: 0C 10       INC   $10
DC6F: BD D3 E0    JSR   $D3E0
DC72: BD 26 C0    JSR   $26C0
DC75: B6 CB 00    LDA   vidctrs              ; read beam counter
(raster vertical position)
DC78: 7F CA 01    CLR   blitter_mask
DC7B: D6 45       LDB   $45
DC7D: C5 02       BITB  #$02
DC7F: 27 0C       BEQ   $DC8D
DC81: 8B 10       ADDA  #$10               ; add #$10 to the beam
counter value in A
DC83: 97 41       STA   $41                ; and store in $9841
DC85: BD DC FF    JSR   $DCFF              ; draw *and* move all
spheroids, sparks, enforcers
DC88: BD DD 4E    JSR   $DD4E              ; draw player (version
1)
DC8B: 20 54       BRA   $DCE1              ; restore rom access
state and return from interrupt

DC8D: 80 10       SUBA  #$10               ; substract #$10 from
the beam counter value in A
DC8F: 97 41       STA   $41                ; and store in $9841
DC91: BD DD 90    JSR   $DD90              ; draw *and* move all
objects (version 2)
DC94: BD DD 3E    JSR   $DD3E              ; draw player (version
2)
DC97: 20 48       BRA   $DCE1              ; restore rom access
state and return from interrupt

DC99: D6 43       LDB   $43
DC9B: 27 44       BEQ   $DCE1
DC9D: 0F 43       CLR   $43
DC9F: 0C 10       INC   $10
DCA1: C6 39       LDB   #$39               ; ensure watchdog
doesn't reset
DCA3: F7 CB FF    STB   watchdog
DCA6: 81 04       CMPA  #$04
DCA8: 22 1B       BHI   $DCC5
DCAA: CE C0 10    LDU   #$C010
DCAD: DC 0A       LDD   $0A
DCAF: 9E 0C       LDX   $0C
DCB1: 10 9E 0E    LDY   $0E
DCB4: 36 36       PSHU  Y,X,B,A
DCB6: DC 04       LDD   $04
DCB8: 9E 06       LDX   $06
DCBA: 10 9E 08    LDY   $08
DCBD: 36 36       PSHU  Y,X,B,A
DCBF: DC 00       LDD   $00
DCC1: 9E 02       LDX   $02
```

```
DCC3: 36 16       PSHU  X,B,A
DCC5: 0C 44       INC   $44
DCC7: BD D4 4D    JSR   $D44D
DCCA: 7F CA 01    CLR   blitter_mask
DCCD: D6 45       LDB   $45
DCCF: C5 02       BITB  #$02
DCD1: 27 08       BEQ   $DCDB
DCD3: BD DC E7    JSR   $DCE7               ; draw all spheroids,
sparks, enforcers (but does not move objects) version 1
DCD6: BD DD 3E    JSR   $DD3E               ; draw player
DCD9: 20 06       BRA   $DCE1               ; restore rom write
state and return from interrupt

DCDB: BD DD 60    JSR   $DD60               ; draw all spheroids,
sparks, enforcers  (but does not move objects) version 2
DCDE: BD DD 4E    JSR   $DD4E               ; draw player
DCE1: 96 45       LDA   $45
DCE3: B7 C9 00    STA   rom_enable_scr_ctrl
DCE6: 3B          RTI                       ; return from interrupt


;
; Not all objects
;
;

DRAW_ALL_OBJECTS_EXCEPT_PLAYER_V1:
DCE7: 96 59       LDA   $59
DCE9: 85 08       BITA  #$08
DCEB: 26 11       BNE   $DCFE
DCED: 9E 17       LDX   $17
DCEF: 27 0D       BEQ   $DCFE
DCF1: EC 04       LDD   $0004,X             ; get blitter
destination
DCF3: D1 41       CMPB  $41
DCF5: 22 03       BHI   $DCFA
DCF7: BD DD CE    JSR   $DDCE               ; draw object X at
destination D
DCFA: AE 84       LDX   ,X
DCFC: 26 F3       BNE   $DCF1
DCFE: 39          RTS




; Here's the routine that draws and moves all the objects that are
"omnidirectional" - that is to say they are not limited to 4 or 8
; degrees of movement like the other objects. Ever wondered how they
glide so smoothly?
;
; Thanks to Jim Bowley for seeing what I couldn't see in the
enforcer movement routine.
; Also thanks to Eugene Jarvis, Larry DeMar for confirming what Jim
thought!! Yes, the men themselves (Eugene via Eugene's wife.) Thanks
```

```
very much.
;
; I (Scott) will try to explain this in plain English for the casual
readers who are interested.
;
; You will notice in the code below references to $000E,X and $10,X.
; In an omnidirectional object, these fields are the horizontal and
vertical movement deltas (= values to be repeatedly added to X and Y
coordinates of an object)
;
; The deltas are comprised of 16 bits.
; The most significant byte, bits 15..8, is the *signed* integer
part of the delta. This can be zero. Bit 15 set means delta is
negative.
; The least significant byte, bits 7..0, is the FRACTIONAL part of
the delta. Think of these as a fraction N/256.
; ** Example: if you wanted to represent a delta of 0.5, set the
most significant byte to 0, and least significant byte to #$80 (128
decimal, which is half of 256)
;
; the horizontal delta is added to the current X coordinate of the
object ($000A,X) and if X coordinate is within playfield written
back to $000A,X.
;
; The vertical delta is added to the current Y coordinate of the
object ($000C,X) and if Y coordinate is within playfield written
back to $000C,X.
;
; After the additions are done the system then takes only the *most
significant bytes* of the X and Y coordinates - NOT the full 16 bits
- and uses
; them to form a memory address for the blitter to write to, which
is held in $0004,X.
;
;

DRAW_AND_MOVE_ALL_SPHEROIDS_ENFORCERS_SPARKS_V1:
DCFF: 96 59       LDA   $59
DD01: 85 08       BITA  #$08
DD03: 26 72       BNE   $DD77
DD05: 9E 17       LDX   $17                 ; get pointer to linked
list of all spheroid, enforcer, spark objects into X
DD07: 27 34       BEQ   $DD3D               ; if null, goto $DD3D
DD09: EC 0A       LDD   $000A,X             ; get X coordinate into
D
DD0B: EE 02       LDU   $0002,X             ; get animation frame
metadata pointer into U
DD0D: E3 0E       ADDD  $000E,X             ; add X delta
DD0F: 81 07       CMPA  #$07                ; at leftmost of
playfield area?
DD11: 25 0A       BCS   $DD1D               ; <7, so yes, invalid
coordinate, do not update X coordinate and goto $DD1D
DD13: AB C4       ADDA  ,U                  ; add width of animation
frame (remember first byte of animation frame metadata is width)
```

```
DD15: 81 90       CMPA  #$90               ; > #$90 (144 decimal) ?
DD17: 22 04       BHI   $DD1D              ; yes, invalid
coordinate, do not update X coordinate and goto $DD1D
DD19: A0 C4       SUBA  ,U
DD1B: ED 0A       STD   $000A,X            ; update X coordinate
with D
; now do Y coordinate part
DD1D: EC 0C       LDD   $000C,X            ; get Y coordinate into
D. A = whole part, B = fractional part
DD1F: E3 88 10    ADDD  $10,X              ; add Y delta
DD22: 81 18       CMPA  #$18               ; at topmost of
playfield area? (#$18 = 24 decimal)
DD24: 25 0A       BCS   $DD30              ; <#$18, so yes, invalid
coordinate, do not update Y coordinate and goto $DD30
DD26: AB 41       ADDA  $0001,U            ; add height of
animation frame (second byte of animation frame metadata is height)
DD28: 81 EB       CMPA  #$EB               ; at bottom-most of
playfield area? (235 decimal)
DD2A: 22 04       BHI   $DD30              ; if higher than #$EB
then invalid coordinate, do not update Y coordinate and goto $DD30
DD2C: A0 41       SUBA  $0001,U
DD2E: ED 0C       STD   $000C,X            ; update Y coordinate
with D
DD30: EC 04       LDD   $0004,X            ; get blitter
destination into D
DD32: D1 41       CMPB  $41                ; "if vertical part of
blitter destination is <= beam counter variable, do not draw
object""
DD34: 23 03       BLS   $DD39
DD36: BD DD CE    JSR   $DDCE              ; draw object X at
destination D
DD39: AE 84       LDX   ,X                 ; get pointer to next
object
DD3B: 26 CC       BNE   $DD09              ; if not null goto $DD09
DD3D: 39          RTS



DRAW_PLAYER_V1:
DD3E: 96 59       LDA   $59
DD40: 85 10       BITA  #$10
DD42: 26 09       BNE   $DD4D
DD44: 8E 98 5A    LDX   #$985A             ; player_object_start
DD47: DC 5E       LDD   $5E                ; player blitter
destination
DD49: D1 41       CMPB  $41
DD4B: 23 11       BLS   $DD5E              ; draw player
DD4D: 39          RTS


DRAW_PLAYER_V2:
DD4E: 96 59       LDA   $59
DD50: 85 10       BITA  #$10
DD52: 26 F9       BNE   $DD4D
```

```
DD54: 8E 98 5A    LDX   #$985A              ; player_object_start
DD57: DC 5E       LDD   $5E                 ; player blitter
destination
DD59: D1 41       CMPB  $41
DD5B: 22 71       BHI   $DDCE               ; draw object X at
destination D
DD5D: 39          RTS

DD5E: 20 6E       BRA   $DDCE               ; draw object X at
destination D



DRAW_ALL_OBJECTS_EXCEPT_PLAYER_V2:
DD60: 96 59       LDA   $59
DD62: 85 08       BITA  #$08
DD64: 26 10       BNE   $DD76
DD66: 9E 17       LDX   $17
DD68: 27 0C       BEQ   $DD76
DD6A: EC 04       LDD   $0004,X
DD6C: D1 41       CMPB  $41
DD6E: 23 02       BLS   $DD72
DD70: 8D 5C       BSR   $DDCE               ; draw object X at
destination D
DD72: AE 84       LDX   ,X
DD74: 26 F4       BNE   $DD6A
DD76: 39          RTS



DRAW_ALL_OBJECTS_EXCEPT_PLAYER_V3:
DD77: 96 59       LDA   $59
DD79: 85 02       BITA  #$02
DD7B: 26 12       BNE   $DD8F
DD7D: 96 44       LDA   $44
DD7F: 84 07       ANDA  #$07
DD81: 26 0C       BNE   $DD8F
DD83: 9E 17       LDX   $17
DD85: 27 08       BEQ   $DD8F
DD87: EC 04       LDD   $0004,X
DD89: 8D 43       BSR   $DDCE               ; draw object X at
destination D
DD8B: AE 84       LDX   ,X
DD8D: 26 F8       BNE   $DD87
DD8F: 39          RTS



DRAW_AND_MOVE_ALL_SPHEROIDS_ENFORCERS_SPARKS_V2:
DD90: 96 59       LDA   $59
DD92: 85 08       BITA  #$08
DD94: 26 E1       BNE   $DD77
DD96: 9E 17       LDX   $17               ; get pointer to all
objects except player, into X
```

```
DD98: 27 33       BEQ   $DDCD               ; if null, goto $DDCD
DD9A: EC 0A       LDD   $000A,X             ; get X coordinate into
D
DD9C: EE 02       LDU   $0002,X             ; get animation frame
metadata pointer into U
DD9E: E3 0E       ADDD  $000E,X             ; add X delta
DDA0: 81 07       CMPA  #$07                ; at leftmost of
playfield area?
DDA2: 25 0A       BCS   $DDAE               ; <7, so yes, invalid
coordinate, do not update X coordinate and goto $DDAE
DDA4: AB C4       ADDA  ,U                  ; add width of animation
frame (remember first byte of animation frame metadata is width)
DDA6: 81 90       CMPA  #$90                ; > #$90 (144 decimal) ?
DDA8: 22 04       BHI   $DDAE               ; yes, invalid
coordinate, do not update X coordinate and goto $DDAE
DDAA: A0 C4       SUBA  ,U
DDAC: ED 0A       STD   $000A,X             ; update X coordinate
with D
; now do Y coordinate part
DDAE: EC 0C       LDD   $000C,X             ; get Y coordinate into
D
DDB0: E3 88 10    ADDD  $10,X               ; add Y delta
DDB3: 81 18       CMPA  #$18                ; at topmost of
playfield area? (#$18 = 24 decimal)
DDB5: 25 0A       BCS   $DDC1               ; <#$18, so yes, invalid
coordinate, do not update Y coordinate and goto $DDC1
DDB7: AB 41       ADDA  $0001,U             ; add height of
animation frame (second byte of animation frame metadata is height)
DDB9: 81 EB       CMPA  #$EB                ; at bottom-most of
playfield area? (235 decimal)
DDBB: 22 04       BHI   $DDC1               ; if higher than #$EB
then invalid coordinate, do not update Y coordinate and goto $DD30
DDBD: A0 41       SUBA  $0001,U
DDBF: ED 0C       STD   $000C,X             ; update Y coordinate
with D
DDC1: EC 04       LDD   $0004,X             ; get blitter
destination into D
DDC3: D1 41       CMPB  $41                 ; "if vertical part of
blitter destination is > beam counter variable, do not draw object""
DDC5: 22 02       BHI   $DDC9
DDC7: 8D 05       BSR   $DDCE               ; draw object X at
destination D
DDC9: AE 84       LDX   ,X                  ; get pointer to next
object
DDCB: 26 CD       BNE   $DD9A               ; if not null goto $DD9A
DDCD: 39          RTS


; Draw object pointed at by X, to blitter destination D.
;
; D = blitter destination
; X = pointer to object (grunt, etc.)
;
```

```
DRAW_OBJECT_TO_D:
DDCE: FD CA 04    STD   blitter_dest
DDD1: EE 88 14    LDU   $14,X                ; get pointer to
previous animation frame metadata into U
DDD4: 37 26       PULU  A,B,Y                ; A= width, B = height,
Y = pointer to actual image
DDD6: 88 04       EORA  #$04
DDD8: C8 04       EORB  #$04
DDDA: FD CA 06    STD   blitter_w_h
DDDD: 10 BF CA 02 STY   blitter_source
DDE1: 86 1A       LDA   #$1A                 ; solid mode,
transparency
DDE3: E6 88 12    LDB   $12,X                ; read flag to see if
image needs shifted one pixel right (bit 7 = yes)
DDE6: 2A 02       BPL   $DDEA
DDE8: 8A 20       ORA   #$20                 ; shift image one pixel
to right
DDEA: B7 CA 00    STA   start_blitter
DDED: A6 0A       LDA   $000A,X              ; A = X coordinate
(whole part)
DDEF: E6 0C       LDB   $000C,X              ; B = Y coordinate
DDF1: ED 04       STD   $0004,X              ; blitter destination =
D
DDF3: FD CA 04    STD   blitter_dest
DDF6: EE 02       LDU   $0002,X              ; U = current animation
frame metadata pointer
DDF8: EF 88 14    STU   $14,X                ; set previous
animation frame metadata pointer (previous = current)
DDFB: EC 42       LDD   $0002,U              ; D = pointer to actual
image to blit
DDFD: FD CA 02    STD   blitter_source
DE00: 86 0A       LDA   #$0A                 ; transparency mode
DE02: E6 0B       LDB   $000B,X
DE04: E7 88 12    STB   $12,X                ; set flag to see if
image needs to be shifted right one pixel
DE07: 2A 02       BPL   $DE0B
DE09: 8A 20       ORA   #$20                 ; shift image one pixel
to right
DE0B: B7 CA 00    STA   start_blitter
DE0E: 39          RTS




;
; Tests for pixels within a rectangle.
; This function is called to ensure it's "safe" to place an object
at a given position.
;
; For example, at $38DC the grunt initialisation routine calls this
function to ensure that no grunts are placed
; directly on top of electrodes that would kill them instantly. See
```

```
also $393C
;
; A = width of rectangle
; B = height of rectangle
; U = screen position representing top left of rectangle
;
; Returns: Z flag is zero if no pixels detected in rectangle


TEST_FOR_PIXELS_WITHIN_RECTANGLE:
DE0F: 34 56       PSHS  U,X,B,A
DE11: 8E DA 05    LDX   #$DA05
DE14: 34 06       PSHS  B,A                 ; save height and width
on the stack
DE16: C6 FE       LDB   #$FE
DE18: D4 45       ANDB  $45
DE1A: D7 45       STB   $45
DE1C: F7 C9 00    STB   rom_enable_scr_ctrl  ; turn off ROM to allow
reads from screen RAM
DE1F: 5F          CLRB
DE20: 6A 61       DEC   $0001,S             ; decrement height (B)
by 1 on the stack
DE22: A6 61       LDA   $0001,S             ; get adjusted height
into A
DE24: EA C6       ORB   A,U                 ; B = B | *(U+A)   -
basically get values of pixels and OR them into B
DE26: 4A          DECA                      ; decrement height
counter
DE27: 2A FB       BPL   $DE24               ; if a>=0 goto $DE24

; At this point B holds a value which if non-zero means there are
some pixels
DE29: 86 37       LDA   #$37
DE2B: 33 C9 01 00 LEAU  $0100,U             ; bump U to point to
next pixel to the right of previous
DE2F: 6A E4       DEC   ,S                  ; decrement width
counter
DE31: 26 EF       BNE   $DE22               ; if width counter !=0
then goto DE22

; at this point B is either non-zero (meaning some pixels were
found) or zero (meaning no pixels were found)
; this looks to me like security code, to make the game behave
unpredictably.
DE33: A1 89 BE EE CMPA  -$4112,X            ; compare A to *$98EE
DE37: 27 12       BEQ   $DE4B
DE39: 96 86       LDA   $86                 ; read a random number
DE3B: 81 01       CMPA  #$01                ; is number > 1 ?
DE3D: 22 0C       BHI   $DE4B               ; if higher than 1 goto
$DE4B (normal service is resumed)

; haha, looks like this piece of code is trying to corrupt some
fields in the object state!!!
DE3F: 34 04       PSHS  B
```

```
DE41: D6 85       LDB   $85
DE43: 86 98       LDA   #$98                ; to form address #
$98xx where B register = xx
DE45: 1F 01       TFR   D,X
DE47: 6A 84       DEC   ,X                  ; ouch! change the
value in *X - who knows what this will break?
DE49: 35 04       PULS  B
DE4B: 86 01       LDA   #$01                ; enable ROM again
DE4D: 9A 45       ORA   $45
DE4F: 97 45       STA   $45
DE51: B7 C9 00    STA   rom_enable_scr_ctrl
DE54: 5D          TSTB                      ; set zero flag
according to B. if B is non zero (meaning, there are pixels found)
then Z flag = 0
DE55: 32 62       LEAS  $0002,S             ; discard B and A
pushed on stack at $DE14
DE57: 35 D6       PULS  A,B,X,U,PC ;(PUL? PC=RTS)



;
; X = source screen address
; Y = destination screen address
; A = width (in bytes. Remember in Robotron, 2 pixels per byte.)
; B = height in pixels
;

COPY_FROM_SCREEN_RAM_TO_RAM:
DE59: 34 36       PSHS  Y,X,B,A
DE5B: 34 16       PSHS  X,B,A
DE5D: D6 45       LDB   $45                 ; get field containing
current rom_enable_scr_ctrl state
DE5F: C4 FE       ANDB  #$FE                ; clear bit 0
DE61: D7 45       STB   $45
DE63: F7 C9 00    STB   rom_enable_scr_ctrl  ; switch ROM out, so we
can access screen RAM. We can now do screen RAM to screen RAM copy
DE66: E6 84       LDB   ,X
DE68: E7 A0       STB   ,Y+
DE6A: 30 89 01 00 LEAX  $0100,X             ; move to next pixel
pair horizontally in source
DE6E: 4A          DECA                      ; decrement counter for
how many bytes to do horizontally
DE6F: 26 F5       BNE   $DE66
DE71: AE 62       LDX   $0002,S             ; restore X from stack
(pushed at $DE5B)
DE73: 30 01       LEAX  $0001,X             ; add 1 to X to move to
next pixel row down (remember the Robotron screen layout!)
DE75: AF 62       STX   $0002,S             ; update X on stack
DE77: A6 E4       LDA   ,S                  ; restore counter for
how many bytes to do horizontally
DE79: 6A 61       DEC   $0001,S             ; decrement counter for
how many bytes to do vertically
DE7B: 26 E9       BNE   $DE66
DE7D: 32 64       LEAS  $0004,S             ; discard items pushed
```

```
on stack at $DE5B
DE7F: 96 45       LDA   $45
DE81: 8A 01       ORA   #$01
DE83: 97 45       STA   $45
DE85: B7 C9 00    STA   rom_enable_scr_ctrl  ; restore
rom_enable_scr_ctrl state, now rom is switched in again
DE88: 35 B6       PULS  A,B,X,Y,PC ;(PUL? PC=RTS)

DE8A:  ROBOTRON: 2084 (TM)  COPYRIGHT
DEAA: 1982 WILLIAMS ELECTRONICS INC.
DECA:  ALL RIGHTS RESERVED

DEDE: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
DEEE: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
DEFE: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
DF0E: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
DF1E: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
DF2E: FF FF FF FF FF FF 0A 0A 0A 0A 0A 0A 0A FF FF FF
DF3E: FF FF

DF40: 35 06       PULS  A,B
DF42: DE 15       LDU   $15
DF44: ED C8 19    STD   $19,U
DF47: BD D0 54    JSR   $D054           ; JMP $D281 - reserve object
metadata entry and call function
DF4A: E2 6F       ; pointer to function
DF4C: BD E1 E3    JSR   $E1E3
DF4F: 86 99       LDA   #$99
DF51: 97 D8       STA   $D8
DF53: 86 CC       LDA   #$CC
DF55: 97 D9       STA   $D9
DF57: 10 8E CF 6E LDY   #$CF6E
DF5B: DE 15       LDU   $15
DF5D: 86 07       LDA   #$07
DF5F: A7 47       STA   $0007,U
DF61: CC 05 02    LDD   #$0502
DF64: ED 48       STD   $0008,U
DF66: CC 09 34    LDD   #$0934
DF69: ED 4A       STD   $000A,U
DF6B: 86 01       LDA   #$01
DF6D: A7 4D       STA   $000D,U
DF6F: 8E 1A 35    LDX   #$1A35
DF72: BD E0 5C    JSR   $E05C
DF75: 86 AA       LDA   #$AA
DF77: 97 D8       STA   $D8
DF79: 86 DD       LDA   #$DD
DF7B: 97 D9       STA   $D9
DF7D: 10 8E CD 5A LDY   #$CD5A
DF81: BD E0 E9    JSR   $E0E9
DF84: 8E CD 5E    LDX   #$CD5E
DF87: BD D0 A2    JSR   LDA_NIB_X1
DF8A: 30 1C       LEAX  $FFFC,X
DF8C: 81 3A       CMPA  #$3A
DF8E: 26 05       BNE   $DF95
```

```
DF90: 8C CD 38    CMPX  #$CD38
DF93: 24 F2       BCC   $DF87
DF95: 30 02       LEAX  $0002,X
DF97: 34 10       PSHS  X
DF99: 8E 15 7A    LDX   #$157A
DF9C: 86 31       LDA   #$31
DF9E: BD 5F 93    JSR   $5F93
DFA1: 86 5C       LDA   #$5C
DFA3: BD 5F 93    JSR   $5F93
DFA6: 30 89 03 00 LEAX  $0300,X
DFAA: 10 8E CD 38 LDY   #$CD38
DFAE: 1E 12       EXG   X,Y
DFB0: BD D0 A2    JSR   LDA_NIB_X1
DFB3: 1E 12       EXG   X,Y
DFB5: BD 5F 93    JSR   $5F93
DFB8: 10 AC E4    CMPY  ,S
DFBB: 23 F1       BLS   $DFAE
DFBD: 30 89 02 00 LEAX  $0200,X
DFC1: 10 8E CD 60 LDY   #score_chksum
DFC5: BD E1 37    JSR   $E137
DFC8: 9F 2B       STX   $2B
DFCA: 8E CC 16    LDX   #$CC16
DFCD: BD D0 A2    JSR   LDA_NIB_X1
DFD0: 81 03       CMPA  #$03
DFD2: 27 23       BEQ   $DFF7
DFD4: 9E 2B       LDX   $2B
DFD6: 30 89 05 00 LEAX  $0500,X
DFDA: 86 5B       LDA   #$5B
DFDC: BD 5F 93    JSR   $5F93
DFDF: 10 8E CD 32 LDY   #top_score
DFE3: C6 03       LDB   #$03
DFE5: 1E 12       EXG   X,Y
DFE7: BD D0 A2    JSR   LDA_NIB_X1
DFEA: 1E 12       EXG   X,Y
DFEC: BD 5F 93    JSR   $5F93
DFEF: 5A          DECB
DFF0: 26 F3       BNE   $DFE5
DFF2: 86 5C       LDA   #$5C
DFF4: BD 5F 93    JSR   $5F93
DFF7: DE 15       LDU   $15
DFF9: 86 05       LDA   #$05
DFFB: A7 47       STA   $0007,U
DFFD: CC 0C 03    LDD   #$0C03
E000: ED 48       STD   $0008,U
E002: CC 07 28    LDD   #$0728
E005: ED 4A       STD   $000A,U
E007: 86 02       LDA   #$02
E009: A7 4D       STA   $000D,U
E00B: 8E 14 88    LDX   #$1488
E00E: 10 8E CD 68 LDY   #high_scores
E012: 8D 48       BSR   $E05C
E014: 86 6E       LDA   #$6E
E016: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
E019: BD D0 54    JSR   $D054                   ; JMP $D281 -
```

```
reserve object metadata entry and call function
E01C: E2 E0       ; pointer to function
E01E: BD D0 54    JSR   $D054                   ; JMP $D281 -
reserve object metadata entry and call function
E021: E2 9A       ; pointer to function
E023: BD D0 54    JSR   $D054                   ; JMP $D281 -
reserve object metadata entry and call function
E026: E2 E9       ; pointer to function
E028: BD D0 54    JSR   $D054                    ; JMP $D281 -
reserve object metadata entry and call function
E02B: E2 F5       ; pointer to function
E02D: DE 15       LDU   $15
E02F: 86 C8       LDA   #$C8
E031: A7 47       STA   $0007,U
E033: 86 03       LDA   #$03
E035: 8E E0 3B    LDX   #$E03B
E038: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

E03B: 6A 47       DEC   $0007,U
E03D: 26 F4       BNE   $E033
E03F: 86 FF       LDA   #$FF
E041: A7 47       STA   $0007,U
E043: 86 04       LDA   #$04
E045: 8E E0 4B    LDX   #$E04B
E048: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

E04B: B6 C8 06    LDA   widget_pia_datab
E04E: 84 03       ANDA  #$03
E050: BA C8 04    ORA   widget_pia_dataa
E053: 27 04       BEQ   $E059
E055: 6A 47       DEC   $0007,U
E057: 26 EA       BNE   $E043
E059: 6E D8 19    JMP   [$19,U]

E05C: 35 06       PULS  A,B
E05E: DE 15       LDU   $15
E060: ED C8 12    STD   $12,U
E063: AF C8 16    STX   $16,U
E066: AF 4E       STX   $000E,U
E068: 10 AF C8 10 STY   $10,U
E06C: A6 48       LDA   $0008,U
E06E: A7 4C       STA   $000C,U
E070: 86 04       LDA   #$04
E072: A7 C8 18    STA   $18,U
E075: AE 4E       LDX   $000E,U
E077: 10 AE C8 10 LDY   $10,U
E07B: 8D 6C       BSR   $E0E9
E07D: E6 4D       LDB   $000D,U
E07F: 86 6F       LDA   #$6F
E081: BD E1 6E    JSR   $E16E
E084: 34 10       PSHS  X
E086: C6 03       LDB   #$03
```

```
E088: 1E 12       EXG   X,Y
E08A: BD D0 A2    JSR   LDA_NIB_X1
E08D: 1E 12       EXG   X,Y
E08F: BD E1 84    JSR   $E184
E092: 5A          DECB
E093: 26 F3       BNE   $E088
E095: 35 10       PULS  X
E097: A6 47       LDA   $0007,U
E099: C6 03       LDB   #$03
E09B: 3D          MUL
E09C: 54          LSRB
E09D: 5C          INCB
E09E: 1F 98       TFR   B,A
E0A0: E6 47       LDB   $0007,U
E0A2: C1 05       CMPB  #$05
E0A4: 26 01       BNE   $E0A7
E0A6: 4C          INCA
E0A7: 5F          CLRB
E0A8: 30 8B       LEAX  D,X
E0AA: BD E1 37    JSR   $E137
E0AD: A6 4D       LDA   $000D,U
E0AF: 8B 01       ADDA  #$01
E0B1: 19          DAA
E0B2: A7 4D       STA   $000D,U
E0B4: 1F 10       TFR   X,D
E0B6: A6 C8 16    LDA   $16,U
E0B9: EB 4A       ADDB  $000A,U
E0BB: 1F 01       TFR   D,X
E0BD: 6A 4C       DEC   $000C,U
E0BF: 27 13       BEQ   $E0D4
E0C1: 6A C8 18    DEC   $18,U
E0C4: 26 B5       BNE   $E07B
E0C6: AF 4E       STX   $000E,U
E0C8: 10 AF C8 10 STY   $10,U
E0CC: 86 01       LDA   #$01
E0CE: 8E E0 70    LDX   #$E070
E0D1: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

E0D4: EC C8 16    LDD   $16,U
E0D7: AB 4B       ADDA  $000B,U
E0D9: A7 C8 16    STA   $16,U
E0DC: 1F 01       TFR   D,X
E0DE: A6 48       LDA   $0008,U
E0E0: A7 4C       STA   $000C,U
E0E2: 6A 49       DEC   $0009,U
E0E4: 26 DB       BNE   $E0C1
E0E6: 6E D8 12    JMP   [$12,U]

E0E9: 34 36       PSHS  Y,X,B,A
E0EB: 31 26       LEAY  $0006,Y
E0ED: 8E BD E4    LDX   #p1_score
E0F0: 8D 1C       BSR   $E10E
E0F2: 27 0C       BEQ   $E100
```

```
E0F4: D6 40       LDB   $40
E0F6: 5A          DECB
E0F7: 27 0F       BEQ   $E108
E0F9: 8E BE 20    LDX   #p2_score
E0FC: 8D 10       BSR   $E10E
E0FE: 26 08       BNE   $E108
E100: 8D 23       BSR   $E125
E102: 27 04       BEQ   $E108
E104: 96 D9       LDA   $D9
E106: 20 02       BRA   $E10A

E108: 96 D8       LDA   $D8
E10A: 97 CF       STA   $CF
E10C: 35 B6       PULS  A,B,X,Y,PC ;(PUL? PC=RTS)

E10E: 34 20       PSHS  Y
E110: 1E 12       EXG   X,Y
E112: BD D0 A2    JSR   LDA_NIB_X1
E115: 84 0F       ANDA  #$0F
E117: C6 04       LDB   #$04
E119: A1 A0       CMPA  ,Y+
E11B: 26 06       BNE   $E123
E11D: BD D0 A2    JSR   LDA_NIB_X1
E120: 5A          DECB
E121: 26 F6       BNE   $E119
E123: 35 A0       PULS  Y,PC ;(PUL? PC=RTS)

E125: 31 21       LEAY  $0001,Y
E127: 30 2E       LEAX  $000E,Y
E129: C6 07       LDB   #$07
E12B: A6 80       LDA   ,X+
E12D: A8 A0       EORA  ,Y+
E12F: 84 0F       ANDA  #$0F
E131: 26 03       BNE   $E136
E133: 5A          DECB
E134: 26 F5       BNE   $E12B
E136: 39          RTS

E137: 1E 12       EXG   X,Y
E139: BD D0 A2    JSR   LDA_NIB_X1
E13C: BD D0 A5    JSR   $D0A5
E13F: 1E 12       EXG   X,Y
E141: 84 0F       ANDA  #$0F
E143: 26 07       BNE   $E14C
E145: 5D          TSTB
E146: 26 04       BNE   $E14C
E148: 86 63       LDA   #$63
E14A: 20 0D       BRA   $E159

E14C: 34 20       PSHS  Y
E14E: 1F 02       TFR   D,Y
E150: 86 62       LDA   #$62
E152: BD E1 6E    JSR   $E16E
E155: 35 20       PULS  Y
```

```
E157: 86 2A       LDA   #$2A
E159: 34 02       PSHS  A
E15B: 1E 12       EXG   X,Y
E15D: BD D0 A8    JSR   $D0A8
E160: 1E 12       EXG   X,Y
E162: 34 20       PSHS  Y
E164: 1F 02       TFR   D,Y
E166: A6 62       LDA   $0002,S
E168: 8D 04       BSR   $E16E
E16A: 35 20       PULS  Y
E16C: 35 82       PULS  A,PC ;(PUL? PC=RTS)

E16E: 34 52       PSHS  U,X,A
E170: 8E 5F 99    LDX   #JMP_PRINT_STRING_LARGE_FONT
E173: A6 47       LDA   $0007,U
E175: 81 07       CMPA  #$07
E177: 27 03       BEQ   $E17C
E179: 8E 5F 96    LDX   #$5F96
E17C: 1F 13       TFR   X,U
E17E: 35 12       PULS  A,X
E180: AD C4       JSR   ,U                  ; JMP $613F: print
string in small font
E182: 35 C0       PULS  U,PC ;(PUL? PC=RTS)

E184: 34 52       PSHS  U,X,A
E186: 8E 5F 93    LDX   #$5F93
E189: A6 47       LDA   $0007,U
E18B: 81 07       CMPA  #$07
E18D: 27 ED       BEQ   $E17C
E18F: 8E 5F 90    LDX   #$5F90
E192: 20 E8       BRA   $E17C

E194:  ROBOTRON: 2084  COPYRIGHT 1982
E1B4: WILLIAMS ELECTRONICS INC.  ALL R
E1D4: IGHTS RESERVED

E1E3: 35 06       PULS  A,B
E1E5: DE 15       LDU   $15
E1E7: ED 4F       STD   $000F,U
E1E9: 8E 98 00    LDX   #$9800
E1EC: CC 00 00    LDD   #$0000
E1EF: ED 81       STD   ,X++
E1F1: 8C 98 10    CMPX  #$9810
E1F4: 25 F9       BCS   $E1EF
E1F6: 8E 06 0D    LDX   #$060D
E1F9: AF C8 17    STX   $17,U
E1FC: 86 11       LDA   #$11
E1FE: A7 C8 15    STA   $15,U
E201: 8E 3E 7D    LDX   #$3E7D
E204: AF C8 11    STX   $11,U
E207: 10 8E 59 7F LDY   #$597F
E20B: 10 AF C8 13 STY   $13,U
E20F: 86 02       LDA   #$02
E211: A7 C8 16    STA   $16,U
```

```
E214: AE C8 11    LDX   $11,U
E217: 10 AE C8 13 LDY   $13,U
E21B: A6 C8 15    LDA   $15,U
E21E: BD E3 13    JSR   $E313
E221: AC C8 17    CPX   $17,U
E224: 27 32       BEQ   $E258
E226: 30 89 FE FE LEAX  $FEFE,X
E22A: 31 A9 01 02 LEAY  $0102,Y
E22E: 8D 17       BSR   $E247
E230: 6A C8 16    DEC   $16,U
E233: 26 E9       BNE   $E21E
E235: A7 C8 15    STA   $15,U
E238: AF C8 11    STX   $11,U
E23B: 10 AF C8 13 STY   $13,U
E23F: 86 01       LDA   #$01
E241: 8E E2 0F    LDX   #$E20F
E244: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

E247: 34 04       PSHS  B
E249: E6 C8 17    LDB   $17,U
E24C: C1 06       CMPB  #$06
E24E: 26 06       BNE   $E256
E250: 80 11       SUBA  #$11
E252: 26 02       BNE   $E256
E254: 86 88       LDA   #$88
E256: 35 84       PULS  B,PC ;(PUL? PC=RTS)

E258: AE C8 17    LDX   $17,U
E25B: 8C 06 0D    CMPX  #$060D
E25E: 27 03       BEQ   $E263
E260: 6E D8 0F    JMP   [$0F,U]

E263: 8E 0E 1D    LDX   #$0E1D
E266: AF C8 17    STX   $17,U
E269: 6F C8 15    CLR   $15,U
E26C: 7E E2 01    JMP   $E201

E26F: 8E E2 FE    LDX   #$E2FE
E272: A6 80       LDA   ,X+
E274: 8D 13       BSR   $E289
E276: AF 47       STX   $0007,U
E278: 86 03       LDA   #$03
E27A: 8E E2 80    LDX   #$E280
E27D: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

E280: AE 47       LDX   $0007,U
E282: 8C E3 13    CMPX  #$E313
E285: 25 EB       BCS   $E272
E287: 20 E6       BRA   $E26F

E289: 10 8E 98 01 LDY   #$9801
E28D: E6 21       LDB   $0001,Y
```

```
E28F: E7 A0       STB   ,Y+
E291: 10 8C 98 08 CMPY  #$9808
E295: 25 F6       BCS   $E28D
E297: A7 A4       STA   ,Y
E299: 39          RTS

E29A: 10 8E 98 0A LDY   #$980A
E29E: 8E E2 C2    LDX   #$E2C2
E2A1: CC E2 C2    LDD   #$E2C2
E2A4: ED 4B       STD   $000B,U
E2A6: 10 AF 49    STY   $0009,U
E2A9: 20 02       BRA   $E2AD

E2AB: AE 4B       LDX   $000B,U
E2AD: AF 47       STX   $0007,U
E2AF: AE 47       LDX   $0007,U
E2B1: A6 80       LDA   ,X+
E2B3: 27 F6       BEQ   $E2AB
E2B5: A7 D8 09    STA   [$09,U]
E2B8: AF 47       STX   $0007,U
E2BA: 86 04       LDA   #$04
E2BC: 8E E2 AF    LDX   #$E2AF
E2BF: 7E D0 66    JMP   $D066               ; JMP $D1E3 – allocate
function call

E2C2: 07 07 07 07 07 07 07 07 57 A7 FF FF A7 57 00 FF
E2D2: FF FF FF FF FF FF E4 D2 C0 C0 C0 D2 E4

E2DF: 00 8E       NEG   $8E
E2E1: E2 C9 10 8E SBCB  $108E,U
E2E5: 98 09       EORA  $09
E2E7: 20 B8       BRA   $E2A1

E2E9: 8E E2 D8    LDX   #$E2D8
E2EC: 10 8E 98 0C LDY   #$980C
E2F0: CC E2 D1    LDD   #$E2D1
E2F3: 20 AF       BRA   $E2A4

E2F5: 8E E2 D1    LDX   #$E2D1
E2F8: 10 8E 98 0D LDY   #$980D
E2FC: 20 F2       BRA   $E2F0

E2FE: 37 2F 27 1F 17 47 47 87 87 C7 C7 C6 C5 CC CB CA
E30E: C0 D0 98

E311: 38          Illegal Opcode
E312: 33 34       LEAU  $FFF4,Y
E314: 36 DE       PSHU  PC,S,X,DP,B,A
E316: 15          Illegal Opcode
E317: 84 F0       ANDA  #$F0
E319: A7 4B       STA   $000B,U
E31B: A6 E4       LDA   ,S
E31D: 84 0F       ANDA  #$0F
E31F: A7 4C       STA   $000C,U
```

```
E321: 1F 10       TFR   X,D
E323: A7 49       STA   $0009,U
E325: E7 47       STB   $0007,U
E327: 1F 20       TFR   Y,D
E329: A7 4A       STA   $000A,U
E32B: E7 48       STB   $0008,U
E32D: E0 47       SUBB  $0007,U
E32F: 56          RORB
E330: 24 02       BCC   $E334
E332: 6A 48       DEC   $0008,U
E334: A6 47       LDA   $0007,U
E336: 8D 3B       BSR   $E373
E338: 4C          INCA
E339: 8D 4A       BSR   $E385
E33B: A6 48       LDA   $0008,U
E33D: 8D 34       BSR   $E373
E33F: 4A          DECA
E340: 8D 43       BSR   $E385
E342: A6 49       LDA   $0009,U
E344: 8D 0B       BSR   $E351
E346: 8D 17       BSR   $E35F
E348: A6 4A       LDA   $000A,U
E34A: 8D 05       BSR   $E351
E34C: 4A          DECA
E34D: 8D 10       BSR   $E35F
E34F: 35 B6       PULS  A,B,X,Y,PC ;(PUL? PC=RTS)

E351: 34 16       PSHS  X,B,A
E353: 8D 14       BSR   $E369
E355: 5C          INCB
E356: A6 4B       LDA   $000B,U
E358: A7 81       STA   ,X++
E35A: 5A          DECB
E35B: 26 FB       BNE   $E358
E35D: 35 96       PULS  A,B,X,PC ;(PUL? PC=RTS)

E35F: 34 16       PSHS  X,B,A
E361: 8D 06       BSR   $E369
E363: 30 01       LEAX  $0001,X
E365: A6 4C       LDA   $000C,U
E367: 20 EF       BRA   $E358

E369: E6 47       LDB   $0007,U
E36B: 1F 01       TFR   D,X
E36D: E6 48       LDB   $0008,U
E36F: E0 47       SUBB  $0007,U
E371: 54          LSRB
E372: 39          RTS

E373: 34 16       PSHS  X,B,A
E375: 8D 16       BSR   $E38D
E377: 5C          INCB
E378: A6 4B       LDA   $000B,U
E37A: A7 84       STA   ,X
```

```
E37C: 30 89 01 00 LEAX  $0100,X
E380: 5A          DECB
E381: 26 F7       BNE   $E37A
E383: 35 96       PULS  A,B,X,PC ;(PUL? PC=RTS)

E385: 34 16       PSHS  X,B,A
E387: 8D 04       BSR   $E38D
E389: A6 4C       LDA   $000C,U
E38B: 20 ED       BRA   $E37A

E38D: 1F 89       TFR   A,B
E38F: A6 49       LDA   $0009,U
E391: 1F 01       TFR   D,X
E393: E6 4A       LDB   $000A,U
E395: E0 49       SUBB  $0009,U
E397: 39          RTS

E398: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
E3A8: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
E3B8: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
E3C8: FF FF FF FF FF FF FF FF

CHECK_CMOS1:
E3D0: 7E E5 F5    JMP   CHECK_CMOS

GET_INITIALS1:
E3D3: 7E E6 F7    JMP   GET_INITIALS

E3D6: 7E E8 A5    JMP   $E8A5

E3D9: 7E E4 33    JMP   $E433

E3DC: 7E E4 13    JMP   $E413

E3DF: 7E E3 E2    JMP   $E3E2

E3E2: 86 18       LDA   #$18
E3E4: A7 47       STA   $0007,U
E3E6: 86 3F       LDA   #$3F
E3E8: B7 C8 0E    STA   rom_pia_datab
E3EB: 86 08       LDA   #$08
E3ED: 8E E3 F3    LDX   #$E3F3
E3F0: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

E3F3: B6 C8 0C    LDA   rom_pia_dataa
E3F6: 85 08       BITA  #$08
E3F8: 27 16       BEQ   $E410
E3FA: 6A 47       DEC   $0007,U
E3FC: 26 ED       BNE   $E3EB
E3FE: 10 8E CD 32 LDY   #top_score
E402: 8E E4 6C    LDX   #$E46C
E405: C6 17       LDB   #$17
E407: BD 6F 0C    JSR   COPY_NIB_XYB1
```

```
E40A: BD E5 BC    JSR   CHKSUM_SCORES
E40D: 7F C8 0E    CLR   rom_pia_datab
E410: 7E D0 63    JMP   $D063               ; JMP $D1F3

E413: 10 8E CD 68 LDY   #high_scores
E417: C6 08       LDB   #$08
E419: BD E5 E3    JSR   $E5E3
E41C: A8 26       EORA  $0006,Y
E41E: 84 0F       ANDA  #$0F
E420: 27 03       BEQ   $E425
E422: 5A          DECB
E423: 27 0E       BEQ   $E433
E425: 86 39       LDA   #$39
E427: B7 CB FF    STA   watchdog
E42A: 31 2E       LEAY  $000E,Y
E42C: 10 8C CF 6E CMPY  #$CF6E
E430: 25 E7       BCS   $E419
E432: 39          RTS

E433: 86 39       LDA   #$39
E435: B7 CB FF    STA   watchdog
E438: 8E E4 6C    LDX   #$E46C
E43B: 10 8E CD 32 LDY   #top_score
E43F: C6 92       LDB   #$92
E441: BD 6F 0C    JSR   COPY_NIB_XYB1
E444: 8E E4 FE    LDX   #$E4FE
E447: 10 8E CE 56 LDY   #$CE56
E44B: C6 8C       LDB   #$8C
E44D: BD 6F 0C    JSR   COPY_NIB_XYB1
E450: BD E5 BC    JSR   CHKSUM_SCORES
E453: 10 8E CD 68 LDY   #high_scores
E457: BD E5 DB    JSR   $E5DB
E45A: 86 39       LDA   #$39
E45C: B7 CB FF    STA   watchdog
E45F: 31 2E       LEAY  $000E,Y
E461: 10 8C CF 6E CMPY  #$CF6E
E465: 25 F0       BCS   $E457
E467: 86 5D       LDA   #$5D
E469: 7E 5F 99    JMP   JMP_PRINT_STRING_LARGE_FONT

E46C: BILWILLY:ELKTRIX:::::::VID
E48C: !EKID!5DRJ!'LEDEPJUJE
E4AC: RPKID MLGSSRfEUNAf
E4CC: 5JRSRPCJMAKJF0 MRS 5PGD
E4EC: NJM    eNHD    `

E4FE: DON VIV GWWCRB UMDR
E51E: ueBACrVW:RppMPT``SUEU M
E53E: OMD DADDySFDDxAKDDwCWK
E55E: 30TMH2pEJS1 RAY0eGAY)eRK
E57E: M(UCNS'U

E58A: 3A 3A 3A 00 01 00 00 C0 01 01 1B 01 01 02 00 C0
E59A: 01 FF 3C 01 FF 00 01 20 00 01 C0 36 01 60 3D 02
```

```
E5AA: 0A 11 02 40 3E 00

E5B0: 34 34       PSHS  Y,X,B
E5B2: 8E E5 8A    LDX   #$E58A
E5B5: C6 07       LDB   #$07
E5B7: BD 6F 0C    JSR   COPY_NIB_XYB1
E5BA: 35 B4       PULS  B,X,Y,PC ;(PUL? PC=RTS)

CHKSUM_SCORES:
E5BC: 34 02       PSHS  A
E5BE: 8D 05       BSR   $E5C5
E5C0: B7 CD 60    STA   score_chksum
E5C3: 35 82       PULS  A,PC ;(PUL? PC=RTS)

E5C5: 34 10       PSHS  X
E5C7: 8E CD 32    LDX   #top_score
E5CA: 4F          CLRA
E5CB: AB 84       ADDA  ,X
E5CD: 30 01       LEAX  $0001,X
E5CF: 8C CD 60    CMPX  #score_chksum
E5D2: 27 F9       BEQ   $E5CD
E5D4: 8C CD 68    CMPX  #high_scores
E5D7: 26 F2       BNE   $E5CB
E5D9: 35 90       PULS  X,PC ;(PUL? PC=RTS)

E5DB: 34 02       PSHS  A
E5DD: 8D 04       BSR   $E5E3
E5DF: A7 26       STA   $0006,Y
E5E1: 35 82       PULS  A,PC ;(PUL? PC=RTS)

E5E3: 34 24       PSHS  Y,B
E5E5: C6 0E       LDB   #$0E
E5E7: 4F          CLRA
E5E8: C1 08       CMPB  #$08
E5EA: 27 02       BEQ   $E5EE
E5EC: AB A4       ADDA  ,Y
E5EE: 31 21       LEAY  $0001,Y
E5F0: 5A          DECB
E5F1: 26 F5       BNE   $E5E8
E5F3: 35 A4       PULS  B,Y,PC ;(PUL? PC=RTS)

CHECK_CMOS:
E5F5: 86 32       LDA   #$32
E5F7: 34 02       PSHS  A
E5F9: 10 8E CD 68 LDY   #high_scores
E5FD: 8D E4       BSR   $E5E3
E5FF: A8 26       EORA  $0006,Y
E601: 84 0F       ANDA  #$0F
E603: 27 0F       BEQ   $E614
E605: BD E6 CC    JSR   MOVE_SCORES
E608: 7F CD 00    CLR   credits_cmos
E60B: 7F CD 01    CLR   $CD01
E60E: 6A E4       DEC   ,S
E610: 27 12       BEQ   $E624
```

```
E612: 20 E9       BRA   $E5FD

E614: 86 03       LDA   #$03
E616: C6 04       LDB   #$04
E618: 8D 68       BSR   $E682
E61A: 25 E9       BCS   $E605
E61C: 31 2E       LEAY  $000E,Y
E61E: 10 8C CF 6E CMPY  #$CF6E
E622: 25 D9       BCS   $E5FD
E624: 35 02       PULS  A
E626: 8E E4 95    LDX   #$E495
E629: 10 8E CF 6E LDY   #$CF6E
E62D: C6 46       LDB   #$46
E62F: BD 6F 0C    JSR   COPY_NIB_XYB1
E632: 8D 91       BSR   $E5C5
E634: B8 CD 60    EORA  score_chksum
E637: 84 0F       ANDA  #$0F
E639: 27 02       BEQ   $E63D
E63B: 8D 0F       BSR   $E64C
E63D: 10 8E CD 32 LDY   #top_score
E641: 86 17       LDA   #$17
E643: C6 04       LDB   #$04
E645: 8D 3B       BSR   $E682
E647: 24 02       BCC   $E64B
E649: 8D 01       BSR   $E64C
E64B: 39          RTS

E64C: 8E CD 38    LDX   #$CD38
E64F: 86 3A       LDA   #$3A
E651: BD D0 AB    JSR   STA_NIB_X1
E654: 8C CD 60    CMPX  #score_chksum
E657: 25 F8       BCS   $E651
E659: 8E CD 68    LDX   #high_scores
E65C: 10 8E CD 38 LDY   #$CD38
E660: 86 06       LDA   #$06
E662: BD E6 EC    JSR   COPY_XYA
E665: 10 8E CD 32 LDY   #top_score
E669: BD E6 EC    JSR   COPY_XYA
E66C: 8E CD 6E    LDX   #$CD6E
E66F: 10 8E CD 60 LDY   #score_chksum
E673: 86 08       LDA   #$08
E675: BD E6 EC    JSR   COPY_XYA
E678: BD E5 BC    JSR   CHKSUM_SCORES
E67B: 10 8E CD 68 LDY   #high_scores
E67F: 7E E6 CC    JMP   MOVE_SCORES

E682: 34 16       PSHS  X,B,A
E684: C6 39       LDB   #$39
E686: F7 CB FF    STB   watchdog                ; stop watchdog from
resetting system
E689: 1F 21       TFR   Y,X
E68B: BD D0 A5    JSR   $D0A5
E68E: C1 41       CMPB  #$41
E690: 24 04       BCC   $E696
```

```
E692: C1 3A       CMPB  #$3A
E694: 26 32       BNE   $E6C8
E696: C1 5A       CMPB  #$5A
E698: 22 2E       BHI   $E6C8
E69A: 4A          DECA
E69B: 26 EE       BNE   $E68B
E69D: A6 61       LDA   $0001,S
E69F: BD D0 A5    JSR   $D0A5
E6A2: C4 0F       ANDB  #$0F
E6A4: C1 09       CMPB  #$09
E6A6: 22 20       BHI   $E6C8
E6A8: 4A          DECA
E6A9: BD D0 A5    JSR   $D0A5
E6AC: 34 04       PSHS  B
E6AE: C4 0F       ANDB  #$0F
E6B0: C1 09       CMPB  #$09
E6B2: 35 04       PULS  B
E6B4: 22 12       BHI   $E6C8
E6B6: C4 F0       ANDB  #$F0
E6B8: C1 99       CMPB  #$99
E6BA: 22 0C       BHI   $E6C8
E6BC: 4A          DECA
E6BD: 26 EA       BNE   $E6A9
E6BF: 1C FE       ANDCC #$FE                ; clear carry flag
E6C1: 86 39       LDA   #$39
E6C3: B7 CB FF    STA   watchdog
E6C6: 35 96       PULS  A,B,X,PC ;(PUL? PC=RTS)

E6C8: 1A 01       ORCC  #$01
E6CA: 20 F5       BRA   $E6C1

MOVE_SCORES:
E6CC: 34 36       PSHS  Y,X,B,A
E6CE: 30 2E       LEAX  $000E,Y
E6D0: 8C CF 6E    CMPX  #$CF6E
E6D3: 24 0F       BCC   $E6E4
E6D5: 86 0E       LDA   #$0E
E6D7: 8D 13       BSR   COPY_XYA
E6D9: 31 2E       LEAY  $000E,Y
E6DB: 30 0E       LEAX  $000E,X
E6DD: 86 39       LDA   #$39
E6DF: B7 CB FF    STA   watchdog
E6E2: 20 EC       BRA   $E6D0

E6E4: BD E5 B0    JSR   $E5B0
E6E7: BD E5 DB    JSR   $E5DB
E6EA: 35 B6       PULS  A,B,X,Y,PC ;(PUL? PC=RTS)

COPY_XYA:
E6EC: 34 36       PSHS  Y,X,B,A
E6EE: E6 80       LDB   ,X+
E6F0: E7 A0       STB   ,Y+
E6F2: 4A          DECA
E6F3: 26 F9       BNE   $E6EE
```

```
E6F5: 35 B6       PULS  A,B,X,Y,PC ;(PUL? PC=RTS)

GET_INITIALS:
E6F7: BD D0 60    JSR   $D060                   ; JMP $D1FF
E6FA: BD D0 12    JSR   CLR_SCREEN1
E6FD: BD D0 33    JSR   LOAD_DA51_PALETTE1
E700: BD D0 99    JSR   FLIP_SCR_UP1
E703: 8E BD E4    LDX   #p1_score
E706: C6 01       LDB   #$01
E708: 8D 20       BSR   CHECK_SCORE
E70A: D6 40       LDB   $40
E70C: 5A          DECB
E70D: 27 18       BEQ   $E727
E70F: BD D0 12    JSR   CLR_SCREEN1
E712: B6 C8 06    LDA   widget_pia_datab
E715: 2A 03       BPL   $E71A
E717: BD D0 9C    JSR   FLIP_SCR_DOWN1
E71A: 8E BE 20    LDX   #p2_score
E71D: C6 02       LDB   #$02
E71F: 8D 09       BSR   CHECK_SCORE
E721: BD D0 12    JSR   CLR_SCREEN1
E724: BD D0 99    JSR   FLIP_SCR_UP1
E727: 7E 77 A0    JMP   $77A0

CHECK_SCORE:
E72A: 35 20       PULS  Y
E72C: 10 BF B3 E4 STY   $B3E4
E730: BF B3 E8    STX   highscore
E733: D7 3F       STB   $3F
E735: BD E9 0E    JSR   $E90E
E738: 24 5B       BCC   $E795
E73A: BD E8 B5    JSR   $E8B5
E73D: 10 8E CD 68 LDY   #high_scores
E741: 8E CF 60    LDX   #$CF60
E744: BD E8 8F    JSR   $E88F
E747: 8E CD 60    LDX   #score_chksum
E74A: 10 8E CD 6E LDY   #$CD6E
E74E: 86 08       LDA   #$08
E750: BD E6 EC    JSR   COPY_XYA
E753: 8E CD 32    LDX   #top_score
E756: 10 8E CD 68 LDY   #high_scores
E75A: 86 06       LDA   #$06
E75C: BD E6 EC    JSR   COPY_XYA
E75F: 10 8E CD 68 LDY   #high_scores
E763: BD E5 DB    JSR   $E5DB
E766: BE B3 E8    LDX   highscore
E769: 10 8E CD 60 LDY   #score_chksum
E76D: C6 04       LDB   #$04
E76F: BD 6F 0C    JSR   COPY_NIB_XYB1
E772: BD E5 BC    JSR   CHKSUM_SCORES
E775: 8E E5 8A    LDX   #$E58A
E778: 10 8E CD 32 LDY   #top_score
E77C: C6 03       LDB   #$03
E77E: BD 6F 0C    JSR   COPY_NIB_XYB1
```

```
E781: BD D0 12    JSR   CLR_SCREEN1
E784: 86 60       LDA   #$60
E786: B7 B3 E6    STA   $B3E6
E789: 8E CC 16    LDX   #$CC16
E78C: BD D0 A5    JSR   $D0A5
E78F: C1 03       CMPB  #$03
E791: 27 42       BEQ   $E7D5
E793: 20 22       BRA   $E7B7

E795: BD E9 1B    JSR   $E91B
E798: 25 09       BCS   $E7A3
E79A: BD E9 38    JSR   $E938
E79D: 25 04       BCS   $E7A3
E79F: 6E 9F B3 E4 JMP   [$B3E4,X]

E7A3: 7F B3 E6    CLR   $B3E6
E7A6: CC E5 91    LDD   #$E591
E7A9: 10 8C CF 6E CMPY  #$CF6E
E7AD: 26 03       BNE   $E7B2
E7AF: CC E5 99    LDD   #$E599
E7B2: BD E9 71    JSR   $E971
E7B5: 86 5F       LDA   #$5F
E7B7: D6 3F       LDB   $3F
E7B9: BD D0 12    JSR   CLR_SCREEN1
E7BC: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
E7BF: CC 3A 3A    LDD   #$3A3A
E7C2: FD B3 EA    STD   hs_inits
E7C5: B7 B3 EC    STA   $B3EC
E7C8: CC 03 00    LDD   #$0300
E7CB: 8E 46 80    LDX   #$4680
E7CE: 10 8E B3 EA LDY   #hs_inits
E7D2: BD 6F 09    JSR   $6F09
E7D5: BD E9 1B    JSR   $E91B
E7D8: 24 06       BCC   $E7E0
E7DA: 8E CF EC    LDX   #$CFEC
E7DD: BD E8 75    JSR   SAVE_HS
E7E0: BD E9 38    JSR   $E938
E7E3: 24 43       BCC   $E828
E7E5: 7D B3 E6    TST   $B3E6
E7E8: 27 1C       BEQ   $E806
E7EA: 8E B3 EA    LDX   #hs_inits
E7ED: 10 8E CD 32 LDY   #top_score
E7F1: C6 03       LDB   #$03
E7F3: BD 6F 0C    JSR   COPY_NIB_XYB1
E7F6: BD E5 BC    JSR   CHKSUM_SCORES
E7F9: 86 05       LDA   #$05
E7FB: 8D 2F       BSR   CHECK_NUM_SCORES1
E7FD: 24 29       BCC   $E828
E7FF: 1F 12       TFR   X,Y
E801: BD E6 CC    JSR   MOVE_SCORES
E804: 20 12       BRA   $E818

E806: BD E8 30    JSR   CHECK_NUM_SCORES2
E809: 34 01       PSHS  CC
```

```
E80B: 34 10       PSHS  X
E80D: 10 AC E1    CMPY  ,S++
E810: 22 02       BHI   $E814
E812: 8D 61       BSR   SAVE_HS
E814: 35 01       PULS  CC
E816: 24 10       BCC   $E828
E818: BD D0 12    JSR   CLR_SCREEN1    ;print MAX ENTRIES
E81B: 86 64       LDA   #$64
E81D: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
E820: 86 60       LDA   #$60
E822: 8E E8 28    LDX   #$E828
E825: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

E828: 6E 9F B3 E4 JMP   [$B3E4,X]

CHECK_NUM_SCORES1:
E82C: 34 26       PSHS  Y,B,A
E82E: 20 0C       BRA   $E83C

CHECK_NUM_SCORES2:
E830: 34 26       PSHS  Y,B,A
E832: 8E CD 32    LDX   #top_score
E835: 8D 24       BSR   CMP_HSINIT_X
E837: 86 04       LDA   #$04
E839: 25 01       BCS   $E83C
E83B: 4C          INCA
E83C: 97 2B       STA   $2B
E83E: 8E CD 68    LDX   #high_scores
E841: 8D 18       BSR   CMP_HSINIT_X
E843: 24 04       BCC   $E849
E845: 0A 2B       DEC   $2B
E847: 27 0E       BEQ   $E857
E849: 30 0E       LEAX  $000E,X
E84B: 8C CF 6E    CMPX  #$CF6E
E84E: 25 F1       BCS   $E841
E850: 8E CF 60    LDX   #$CF60
E853: 1C FE       ANDCC #$FE                ; clear carry flag
E855: 35 A6       PULS  A,B,Y,PC ;(PUL? PC=RTS)

E857: 1A 01       ORCC  #$01
E859: 35 A6       PULS  A,B,Y,PC ;(PUL? PC=RTS)

CMP_HSINIT_X:
E85B: 34 10       PSHS  X
E85D: 10 8E B3 EA LDY   #hs_inits
E861: C6 03       LDB   #$03
E863: BD D0 A2    JSR   LDA_NIB_X1
E866: A1 A0       CMPA  ,Y+
E868: 26 07       BNE   $E871
E86A: 5A          DECB
E86B: 26 F6       BNE   $E863
E86D: 1A 01       ORCC  #$01
E86F: 35 90       PULS  X,PC ;(PUL? PC=RTS)
```

```
E871: 1C FE       ANDCC #$FE                ; clear carry flag
E873: 35 90       PULS  X,PC ;(PUL? PC=RTS)

SAVE_HS:
E875: 34 20       PSHS  Y
E877: BD E8 8F    JSR   $E88F
E87A: 8E B3 EA    LDX   #hs_inits
E87D: C6 03       LDB   #$03
E87F: BD 6F 0C    JSR   COPY_NIB_XYB1
E882: BE B3 E8    LDX   highscore
E885: C6 04       LDB   #$04
E887: BD 6F 0C    JSR   COPY_NIB_XYB1
E88A: 35 20       PULS  Y
E88C: 7E E5 DB    JMP   $E5DB

E88F: 34 30       PSHS  Y,X
E891: 1F 12       TFR   X,Y
E893: 10 AC 62    CMPY  $0002,S
E896: 27 0B       BEQ   $E8A3
E898: 30 32       LEAX  $FFF2,Y
E89A: 86 0E       LDA   #$0E
E89C: BD E6 EC    JSR   COPY_XYA
E89F: 31 32       LEAY  $FFF2,Y
E8A1: 20 F0       BRA   $E893

E8A3: 35 B0       PULS  X,Y,PC ;(PUL? PC=RTS)

E8A5: 35 06       PULS  A,B
E8A7: FD B3 E4    STD   $B3E4
E8AA: C6 01       LDB   #$01
E8AC: 8D 07       BSR   $E8B5
E8AE: BD E5 BC    JSR   CHKSUM_SCORES
E8B1: 6E 9F B3 E4 JMP   [$B3E4,X]

E8B5: 35 20       PULS  Y
E8B7: 10 BF B3 E6 STY   $B3E6
E8BB: 4F          CLRA
E8BC: 1F 02       TFR   D,Y
E8BE: CC E5 99    LDD   #$E599
E8C1: BD E9 71    JSR   $E971
E8C4: 8E CC 16    LDX   #$CC16
E8C7: BD D0 A5    JSR   $D0A5
E8CA: BD D0 12    JSR   CLR_SCREEN1
E8CD: C1 03       CMPB  #$03
E8CF: 26 0E       BNE   $E8DF
E8D1: D6 3F       LDB   $3F
E8D3: 86 5F       LDA   #$5F
E8D5: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
E8D8: 86 03       LDA   #$03
E8DA: 8E 46 80    LDX   #$4680
E8DD: 20 0D       BRA   $E8EC

E8DF: 86 5E       LDA   #$5E
```

```
E8E1: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
E8E4: 1F 98       TFR   B,A
E8E6: BD D0 C6    JSR   $D0C6               ; JMP   $D5D8 - convert
from BCD to normal number
E8E9: 8E 2D 80    LDX   #$2D80
E8EC: 10 8E B3 FE LDY   #$B3FE
E8F0: C6 3A       LDB   #$3A
E8F2: E7 A2       STB   ,-Y
E8F4: 10 8C B3 EA CMPY  #hs_inits
E8F8: 22 F8       BHI   $E8F2
E8FA: 5F          CLRB
E8FB: BD 6F 09    JSR   $6F09
E8FE: 8E B3 EA    LDX   #hs_inits
E901: 10 8E CD 38 LDY   #$CD38
E905: C6 14       LDB   #$14
E907: BD 6F 0C    JSR   COPY_NIB_XYB1
E90A: 6E 9F B3 E6 JMP   [$B3E6,X]

E90E: 34 30       PSHS  Y,X
E910: 10 8E CD 60 LDY   #score_chksum
E914: BE B3 E8    LDX   highscore
E917: 8D 38       BSR   $E951
E919: 35 B0       PULS  X,Y,PC ;(PUL? PC=RTS)

E91B: 34 10       PSHS  X
E91D: 10 8E CF 74 LDY   #$CF74
E921: BE B3 E8    LDX   highscore
E924: 8D 2B       BSR   $E951
E926: 25 0C       BCS   $E934
E928: 31 2E       LEAY  $000E,Y
E92A: 10 8C CF FA CMPY  #$CFFA
E92E: 25 F4       BCS   $E924
E930: 1C FE       ANDCC #$FE                ; clear carry flag
E932: 35 90       PULS  X,PC ;(PUL? PC=RTS)

E934: 31 3A       LEAY  $FFFA,Y
E936: 35 90       PULS  X,PC ;(PUL? PC=RTS)

E938: 34 10       PSHS  X
E93A: 10 8E CD 60 LDY   #score_chksum
E93E: BE B3 E8    LDX   highscore
E941: 8D 0E       BSR   $E951
E943: 25 EF       BCS   $E934
E945: 31 2E       LEAY  $000E,Y
E947: 10 8C CF 60 CMPY  #$CF60
E94B: 25 F4       BCS   $E941
E94D: 1C FE       ANDCC #$FE                ; clear carry flag
E94F: 35 90       PULS  X,PC ;(PUL? PC=RTS)

E951: 34 36       PSHS  Y,X,B,A
E953: 1E 12       EXG   X,Y
E955: C6 04       LDB   #$04
E957: BD D0 A2    JSR   LDA_NIB_X1
E95A: C1 04       CMPB  #$04
```

```
E95C: 26 02       BNE   $E960
E95E: 84 0F       ANDA  #$0F
E960: A1 A0       CMPA  ,Y+
E962: 22 05       BHI   $E969
E964: 25 07       BCS   $E96D
E966: 5A          DECB
E967: 26 EE       BNE   $E957
E969: 1C FE       ANDCC #$FE                ; clear carry flag
E96B: 35 B6       PULS  A,B,X,Y,PC ;(PUL? PC=RTS)

E96D: 1A 01       ORCC  #$01
E96F: 35 B6       PULS  A,B,X,Y,PC ;(PUL? PC=RTS)

E971: 0F 56       CLR   $56
E973: 7E D0 4B    JMP   $D04B               ; JMP $D3C7

E976: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
E986: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
E996: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
E9A6: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
E9B6: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
E9C6: FF FF


E9C8: E9 CC                               ; pointer to
SMALL_CHARACTER_TABLE
E9CA: EC 34                               ; pointer to
LARGE_CHARACTER_TABLE

SMALL_CHARACTER_TABLE:
E9CC: EA 2A                               ; pointer to small
character 0
E9CE: EA 35                               ; pointer to small
character 1
E9D0: EA 40                               ; pointer to small
character 2
E9D2: EA 4B                               ; pointer to small
character 3
E9D4: EA 56                               ; pointer to small
character 4
E9D6: EA 61                               ; pointer to small
character 5
E9D8: EA 6C                               ; pointer to small
character 6
E9DA: EA 77                               ; pointer to small
character 7
E9DC: EA 82                               ; pointer to small
character 8
E9DE: EA 8D                               ; pointer to small
character 9
E9E0: EA 98                               ; pointer to black
square (used as SPACE character)
E9E2: EA 9E                               ; pointer to small
"!" (exclamation mark)
```

```
E9E4: EA A4                                     ; pointer to small
"," (comma)
E9E6: EA AF                                     ; pointer to small
"." (full stop/period)
E9E8: EA B5                                     ; pointer to small up
arrow
E9EA: EA C5                                     ; pointer to small
":" (colon)
E9EC: EA CB
EA D6 EA E1 EA EC EA F7
E9F6: EB 02 EB 0D EB 18 EB 23 EB 2E EB 39 EB 44 EB 4F
EA06: EB 5A EB 6A EB 75 EB 80 EB 8B EB 96 EB A1 EB AC
EA16: EB B7 EB C2 EB CD EB DD EB E8 EB F3 EB FE EC 09
EA26: EC 14 EC 29 03 FF F0 F0 F0 F0 F0 F0 F0 FF F0 03
EA36: 0F 00 0F 00 0F 00 0F 00 0F 00 03 FF F0 00 F0 FF
EA46: F0 F0 00 FF F0 03 FF F0 00 F0 0F F0 00 F0 FF F0
EA56: 03 F0 F0 F0 F0 FF F0 00 F0 00 F0 03 FF F0 F0 00
EA66: FF F0 00 F0 FF F0 03 FF F0 F0 00 FF F0 F0 F0 FF
EA76: F0 03 FF F0 00 F0 00 F0 00 F0 00 F0 03 FF F0 F0
EA86: F0 FF F0 F0 F0 FF F0 03 FF F0 F0 F0 FF F0 00 F0
EA96: 00 F0 01 00 00 00 00 00 01 F0 F0 F0 00 F0 03 00
EAA6: 00 00 00 00 00 FF 00 0F 00 01 00 00 00 00 F0 05
EAB6: 00 F0 00 0F FF 00 FF FF F0 00 F0 00 00 F0 00 01
EAC6: 00 F0 00 F0 00 03 00 00 00 00 FF F0 00 00 00 00
EAD6: 03 FF F0 F0 F0 FF F0 F0 F0 F0 F0 03 FF F0 F0 F0
EAE6: FF 00 F0 F0 FF F0 03 FF F0 F0 00 F0 00 F0 00 FF
EAF6: F0 03 FF 00 F0 F0 F0 F0 F0 F0 FF 00 03 FF F0 F0
EB06: 00 FF 00 F0 00 FF F0 03 FF F0 F0 00 FF 00 F0 00
EB16: F0 00 03 FF F0 F0 00 F0 F0 F0 F0 FF F0 03 F0 F0
EB26: F0 F0 FF F0 F0 F0 F0 F0 03 FF F0 0F 00 0F 00 0F
EB36: 00 FF F0 03 00 F0 00 F0 00 F0 F0 F0 FF F0 03 F0
EB46: F0 F0 F0 FF 00 F0 F0 F0 F0 03 F0 00 F0 00 F0 00
EB56: F0 00 FF F0 05 FF FF F0 F0 F0 F0 F0 F0 F0 F0 00
EB66: F0 F0 00 F0 03 FF F0 F0 F0 F0 F0 F0 F0 F0 F0 03
EB76: FF F0 F0 F0 F0 F0 F0 F0 FF F0 03 FF F0 F0 F0 FF
EB86: F0 F0 00 F0 00 03 FF F0 F0 F0 FF F0 00 F0 00 F0
EB96: 03 FF F0 F0 F0 FF 00 F0 F0 F0 F0 03 FF F0 F0 00
EBA6: FF F0 00 F0 FF F0 03 FF F0 0F 00 0F 00 0F 00 0F
EBB6: 00 03 F0 F0 F0 F0 F0 F0 F0 F0 FF F0 03 F0 F0 F0
EBC6: F0 F0 F0 0F 00 0F 00 05 F0 00 F0 F0 00 F0 F0 F0
EBD6: F0 F0 F0 F0 FF FF F0 03 F0 F0 F0 F0 0F 00 F0 F0
EBE6: F0 F0 03 F0 F0 F0 F0 FF F0 0F 00 0F 00 03 FF F0
EBF6: 00 F0 0F 00 F0 00 FF F0 03 0F 00 F0 00 F0 00 F0
EC06: 00 0F 00 03 0F 00 00 F0 00 F0 00 F0 0F 00 07 00
EC16: 0F 00 00 00 00 F0 00 FF 00 FF 00 00 00 F0 00 00
EC26: 0F 00 00 03 00 F0 00 F0 0F 00 F0 00 F0 00



CHARACTER_POINTER_TABLE:
EC34: EC 92                                     ; pointer to large
character 0
EC36: EC A5                                     ; pointer to large
character 1
```

```
EC38: EC B8                                     ; pointer to large
character 2
EC3A: EC CB                                     ; pointer to large
character 3
EC3C: EC DE                                     ; pointer to large
character 4
EC3E: EC F1                                     ; pointer to large
character 5
EC40: ED 04                                     ; pointer to large
character 6
EC42: ED 17                                     ; pointer to large
character 7
EC44: ED 2A                                     ; pointer to large
character 8
EC46: ED 3D                                     ; pointer to large
character 9
EC48: ED 50                                     ; pointer to black block
(used as a space)
EC4A: ED 5D                                     ; pointer to large
"!" (exclamation mark)
EC4C: ED 6C                                     ; pointer to large
"," (comma)
EC4E: ED 79                                     ; pointer to large
"." (full stop/ period)
EC50: ED 86                                     ; pointer to large red
square
EC52: ED A5                                     ; pointer to large
character ":" (colon)
EC54: ED B2                                     ; pointer to large
"–" (minus sign)
EC56: ED BF                                     ; pointer to large
character A
EC58: ED D2                                     ; pointer to large
character B
EC5A: ED E5                                     ; pointer to large
character C
EC5C: ED F8                                     ; pointer to large
character D
EC5E: EE 0B                                     ; pointer to large
character E
EC60: EE 1E                                     ; pointer to large
character F
EC62: EE 31                                     ; pointer to large
character G
EC64: EE 44                                     ; pointer to large
character H
EC66: EE
57                                                          ;
pointer to large character I
EC68: EE 6A                                     ; pointer to large
character J
EC6A: EE 7D                                     ; pointer to large
character K
EC6C: EE 90                                     ; pointer to large
```

```
character L
EC6E: EE A3                                     ; pointer to large
character M
EC70: EE B6                                     ; pointer to large
character N
EC72: EE C9                                     ; pointer to large
character O
EC74: EE DC                                     ; pointer to large
character P
EC76: EE EF                                     ; pointer to large
character Q
EC78: EF 02                                     ; pointer to large
character R
EC7A: EF 15                                     ; pointer to large
character S
EC7C: EF 28                                     ; pointer to large
character T
EC7E: EF 3B                                     ; pointer to large
character U
EC80: EF 4E                                     ; pointer to large
character V
EC82: EF 61                                     ; pointer to large
character W
EC84: EF 74                                     ; pointer to large
character X
EC86: EF 87                                     ; pointer to large
character Y
EC88: EF 9A                                     ; pointer to large
character Z
EC8A: EF AD                                     ; pointer to large
character "(" (left parentheses)
EC8C: EF BA                                     ; pointer to large
character ")" (right parentheses)
EC8E: EF C7                                     ; pointer to large
character "/" (forward slash)
EC90: EF DA                                     ; pointer to large
character <- (left arrow)


EC92: 05 99 99 90
EC96: 90 00 90 90 00 90 90 00 90 90 00 90 99 99 90 05
ECA6: 00 99 00 00 09 00 00 09 00 00 09 00 00 09 00 09
ECB6: 99 90 05 09 99 90 00 00 90 99 99 90 90 00 00 90
ECC6: 00 00 99 99 00 05 09 99 90 00 00 90 09 99 90 00
ECD6: 00 90 00 00 90 99 99 90 05 90 00 00 90 09 00 90
ECE6: 09 00 99 99 90 00 09 00 00 09 00 05 99 99 00 90
ECF6: 00 00 99 99 90 00 00 90 00 00 90 99 99 90 05 99
ED06: 99 00 90 00 00 99 99 90 90 00 90 90 00 90 99 99
ED16: 90 05 99 99 90 00 00 90 00 09 00 00 90 00 09 00
ED26: 00 90 00 00 05 99 99 90 90 00 90 99 99 90 90 00
ED36: 90 90 00 90 99 99 90 05 99 99 90 90 00 90 99 99
ED46: 90 00 00 90 00 00 90 09 99 90 03 00 00 00 00 00
ED56: 00 00 00 00 00 00 00 02 FF 00 FF 00 FF 00 FF 00
ED66: 00 00 FF 00 FF 00 02 00 00 00 00 00 00 FF 00 FF
```

```
ED76: 00 F0 00 02 00 00 00 00 00 00 00 FF 00 FF 00
ED86: 09 FF FF FF FF F0 FF FF FF FF F0 FF FF FF FF F0
ED96: FF FF FF FF F0 FF FF FF FF F0 FF FF FF FF F0

EDA5: 02
EDA6: 00 00 FF 00 FF 00 00 00 FF 00 FF 00 03 00 00 00
EDB6: 00 00 00 22 20 00 00 00 00 05 66 66 60 60 06 60
EDC6: 66 66 60 60 06 60 60 06 60 60 06 60 05 66 66 00
EDD6: 60 06 00 66 66 60 66 00 60 66 00 60 66 66 60 05
EDE6: 66 66 60 66 00 00 66 00 00 66 00 00 66 00 00 66
EDF6: 66 60 05 66 66 00 66 00 60 66 00 60 66 00 60 66
EE06: 00 60 66 66 00 05 66 66 60 60 00 00 66 66 60 66
EE16: 00 00 66 00 00 66 66 60 05 66 66 60 60 00 00 66
EE26: 66 60 66 00 00 66 00 00 66 00 00 05 66 66 60 66
EE36: 06 60 66 00 00 66 06 60 66 00 60 66 66 60 05 66
EE46: 00 60 66 00 60 66 66 60 66 00 60 66 00 60 66 00
EE56: 60


EE57: 04 06 60 00 06 60 00 06 60 00 06 60 00 06 60
EE66: 00 06 60 00 05 00 00 60 00 00 60 00 00 60 00 00
EE76: 60 66 00 60 66 66 60 05 66 00 60 66 06 00 66 60
EE86: 00 66 60 00 66 06 00 66 00 60 05 60 00 00 60 00
EE96: 00 60 00 00 60 00 00 66 66 60 66 66 60 05 66 66
EEA6: 60 60 60 60 60 60 60 60 00 60 60 00 60 60 00 60
EEB6: 05 66 66 60 60 00 60 66 00 60 66 00 60 66 00 60
EEC6: 66 00 60 05 66 66 60 60 06 60 60 00 60 60 00 60
EED6: 60 00 60 66 66 60 05 66 66 60 60 00 60 66 66 60
EEE6: 66 00 00 66 00 00 66 00 00 05 66 66 60 66 00 60
EEF6: 66 00 60 66 00 60 66 06 00 66 60 60 05 66 66 00
EF06: 60 06 00 66 66 60 66 00 60 66 00 60 66 00 60 05
EF16: 66 66 60 60 00 00 66 66 60 00 00 60 66 00 60 66
EF26: 66 60 05 66 66 60 00 60 00 00 66 00 00 66 00 00
EF36: 66 00 00 66 00 05 60 06 60 60 06 60 60 06 60 60
EF46: 06 60 60 06 60 66 66 60 05 66 00 60 66 00 60 66
EF56: 00 60 66 00 60 06 06 00 00 60 00 05 60 00 60 60
EF66: 00 60 60 00 60 60 60 60 60 60 60 66 66 60 05 60
EF76: 00 60 06 06 00 00 60 00 00 60 00 06 06 00 60 00
EF86: 60 05 66 00 60 66 00 60 66 66 60 00 60 00 00 60
EF96: 00 00 60 00 05 66 66 60 00 06 00 00 60 00 06 00
EFA6: 00 66 66 60 66 66 60 03 00 F0 0F 00 F0 00 F0 00
EFB6: 0F 00 00 F0 03 F0 00 0F 00 00 F0 00 F0 0F 00 F0
EFC6: 00 04 00 0F 00 00 F0 00 00 F0 00 0F 00 00 0F 00
EFD6: 00 F0 00 00 05 00 F0 00 0F F0 00 FF FF F0 0F F0
EFE6: 00 00 F0 00 00 00 00 FF FF FF D1 06 D1 06 D1 06
EFF6: D1 06 DC 56 D1 06 D1 06 D1 06

F000: 7E F4 31    JMP   RESET

F003: 7E F4 A0    JMP   $F4A0

F006: 7E F6 1B    JMP   $F61B

F009: 7E F0 1D    JMP   $F01D
```

```
F00C: 7E F0 D7    JMP   $F0D7

F00F: 7E F0 66    JMP   $F066

F012: 7E F3 4A    JMP   $F34A

F015: F1 24 F1    CMPB  $24F1
F018: D5 F3       BITB  $F3
F01A: 68 F3       ASL   [,--S]



;
; Reserve 1694 decimal bytes starting at #$B3E4 for a forward only
linked list used for explosions.
; Each entry in the list uses 242 (!) bytes, and so there are 7
entries - meaning 7 explosions max.
; I could probably pick a better label than RESET_EXPLOSION_LIST -
am open to suggestions :)

RESET_EXPLOSION_LIST:
F01D: 34 16       PSHS  X,B,A
F01F: 8E B3 E4    LDX   #$B3E4              ; start of explosion
forward only linked list
F022: 9F AD       STX   $AD
F024: 30 89 00 F2 LEAX  $00F2,X             ; X+= 242 decimal (bumps
X to NEXT entry)
F028: AF 89 FF 0E STX   -$00F2,X            ; store X in X-242
decimal (the previous entry), creating a forward only linked list
F02C: 8C BA 82    CMPX  #$BA82
F02F: CC 00 00    LDD   #$0000              ; mark end of explosion
list with NULL
F032: ED 84       STD   ,X                  ; store null at end of
list
F034: DD A9       STD   $A9
F036: DD AB       STD   $AB
; on exit, $AD = B3E4, $A9 = 0, $AB = 0
F038: 35 96       PULS  A,B,X,PC ; (PUL? PC=RTS)



;
; Get a pointer to a "free" (available for use) entry in the
explosion list into U
;

GET_FREE_EXPLOSION_LIST_ENTRY:
F03A: 34 10       PSHS  X
F03C: DE AD       LDU   $AD                 ; get pointer to current
explosion list entry
F03E: 27 0E       BEQ   $F04E               ; if at end of list goto
$F04E
F040: AE C4       LDX   ,U                  ; get NEXT explosion
list pointer into X
```

```
F042: 9F AD       STX   $AD                ; and save in $AD
F044: 9E A9       LDX   $A9
F046: AF C4       STX   ,U
F048: DF A9       STU   $A9
F04A: 1C FE       ANDCC #$FE               ; clear carry flag
F04C: 35 90       PULS  X,PC ;(PUL? PC=RTS)

F04E: 1A 01       ORCC  #$01               ; set carry flag
F050: 35 90       PULS  X,PC ;(PUL? PC=RTS)

F052: 34 10       PSHS  X
F054: DE AD       LDU   $AD
F056: 27 F6       BEQ   $F04E
F058: AE C4       LDX   ,U
F05A: 9F AD       STX   $AD
F05C: 9E AB       LDX   $AB
F05E: AF C4       STX   ,U
F060: DF AB       STU   $AB
F062: 1C FE       ANDCC #$FE               ; clear carry flag
F064: 35 90       PULS  X,PC ;(PUL? PC=RTS)


F066: 34 76       PSHS  U,Y,X,B,A
F068: BD F0 52    JSR   $F052
F06B: 25 66       BCS   $F0D3
F06D: EC 04       LDD   $0004,X
F06F: AE 02       LDX   $0002,X
F071: ED 49       STD   $0009,U
F073: D6 A6       LDB   $A6
F075: E7 44       STB   $0004,U
F077: E0 49       SUBB  $0009,U
F079: 25 04       BCS   $F07F
F07B: E1 84       CMPB  ,X
F07D: 25 0A       BCS   $F089
F07F: E6 84       LDB   ,X
F081: E7 45       STB   $0005,U
F083: EB 49       ADDB  $0009,U
F085: E7 44       STB   $0004,U
F087: 20 03       BRA   $F08C

F089: 58          ASLB
F08A: E7 45       STB   $0005,U
F08C: CC 46 46    LDD   #$4646
F08F: ED 4F       STD   $000F,U
F091: EC 84       LDD   ,X
F093: ED 4B       STD   $000B,U
F095: 86 01       LDA   #$01
F097: A7 C8 11    STA   $11,U
F09A: 88 04       EORA  #$04
F09C: C8 04       EORB  #$04
F09E: ED 4D       STD   $000D,U
F0A0: AE 02       LDX   $0002,X
F0A2: AF 42       STX   $0002,U
F0A4: CC 10 00    LDD   #$1000
```

```
F0A7: ED 46       STD   $0006,U
F0A9: 30 C8 11    LEAX  $11,U
F0AC: 9F B0       STX   $B0
F0AE: 10 AE 42    LDY   $0002,U
F0B1: E6 4C       LDB   $000C,U
F0B3: D7 B3       STB   $B3
F0B5: 9E B0       LDX   $B0
F0B7: 30 01       LEAX  $0001,X
F0B9: 9F B0       STX   $B0
F0BB: A6 4B       LDA   $000B,U
F0BD: 97 B2       STA   $B2
F0BF: A6 A0       LDA   ,Y+
F0C1: A7 84       STA   ,X
F0C3: 3A          ABX
F0C4: 48          ASLA
F0C5: 48          ASLA
F0C6: 48          ASLA
F0C7: 48          ASLA
F0C8: A7 84       STA   ,X
F0CA: 3A          ABX
F0CB: 0A B2       DEC   $B2
F0CD: 26 F0       BNE   $F0BF
F0CF: 0A B3       DEC   $B3
F0D1: 26 E2       BNE   $F0B5
F0D3: 1C FE       ANDCC #$FE                ; clear carry flag
F0D5: 35 F6       PULS  A,B,X,Y,U,PC ;(PUL? PC=RTS)



;
; X = pointer to object
;

F0D7: 34 76       PSHS  U,Y,X,B,A
F0D9: BD D0 15    JSR   $D015               ; JMP $DB03 - erase
object from screen
F0DC: BD F0 3A    JSR   $F03A               ; get a slot in the
explosion list
; U = pointer to explosion slot
F0DF: 25 F2       BCS   $F0D3               ; if carry is set, then
that means there's no slots left, so clear carry flag and exit
F0E1: EC 04       LDD   $0004,X             ; D = blitter
destination
F0E3: AE 02       LDX   $0002,X             ; X = animation frame
metadata pointer
F0E5: ED 49       STD   $0009,U             ; store blitter
destination into explosion slot
F0E7: D6 A6       LDB   $A6
F0E9: E7 44       STB   $0004,U
F0EB: E0 49       SUBB  $0009,U
F0ED: 25 04       BCS   $F0F3
F0EF: E1 84       CMPB  ,X                  ; compare to width of
animation frame
```

```
F0F1: 25 0A       BCS   $F0FD
F0F3: E6 84       LDB   ,X
F0F5: E7 45       STB   $0005,U
F0F7: EB 49       ADDB  $0009,U
F0F9: E7 44       STB   $0004,U
F0FB: 20 03       BRA   $F100

F0FD: 58          ASLB
F0FE: E7 45       STB   $0005,U
F100: CC 46 46    LDD   #$4646
F103: ED 4F       STD   $000F,U
F105: EC 84       LDD   ,X
F107: ED 4B       STD   $000B,U
F109: 48          ASLA
F10A: A7 C8 11    STA   $11,U
F10D: 86 01       LDA   #$01
F10F: 88 04       EORA  #$04
F111: C8 04       EORB  #$04
F113: ED 4D       STD   $000D,U
F115: AE 02       LDX   $0002,X
F117: AF 42       STX   $0002,U
F119: CC 00 00    LDD   #$0000
F11C: ED 46       STD   $0006,U
F11E: 86 10       LDA   #$10
F120: A7 48       STA   $0008,U
F122: 20 85       BRA   $F0A9



;
; This is to draw an object in a staggered (broken up) state, such
as during an explosion when enemy killed
; This routine is also used to draw the word ROBOTRON forming up in
large letters on title screen.
;
; X = pointer to image data (blitter source)
; Y = $CA05 (blitter destination LSB)
; B = width in bytes of image
; $A8 = variable that is used to determine how far apart the
explosion segments are.
;

; If you are looking for references in the code to this routine, you
won't find any obvious direct calls.
; Check out $5DFB which computes the address of routine to jump to.
This is one of those routines called.


DRAW_STAGGERED_IMAGE:
F124: A7 A4       STA   ,Y
F126: BF CA 02    STX   blitter_source
F129: FF CA 00    STU   start_blitter
F12C: 3A          ABX                           ; X+= B
F12D: 9B A8       ADDA  $A8
```

```
F12F: A7 A4       STA   ,Y
F131: BF CA 02    STX   blitter_source
F134: FF CA 00    STU   start_blitter
F137: 3A          ABX                           ; X+= B
F138: 9B A8       ADDA  $A8
F13A: A7 A4       STA   ,Y
F13C: BF CA 02    STX   blitter_source
F13F: FF CA 00    STU   start_blitter
F142: 3A          ABX                           ; X+= B
F143: 9B A8       ADDA  $A8
F145: A7 A4       STA   ,Y
F147: BF CA 02    STX   blitter_source
F14A: FF CA 00    STU   start_blitter
F14D: 3A          ABX
F14E: 9B A8       ADDA  $A8
F150: A7 A4       STA   ,Y
F152: BF CA 02    STX   blitter_source
F155: FF CA 00    STU   start_blitter
F158: 3A          ABX
F159: 9B A8       ADDA  $A8
F15B: A7 A4       STA   ,Y
F15D: BF CA 02    STX   blitter_source
F160: FF CA 00    STU   start_blitter
F163: 3A          ABX
F164: 9B A8       ADDA  $A8
F166: A7 A4       STA   ,Y
F168: BF CA 02    STX   blitter_source
F16B: FF CA 00    STU   start_blitter
F16E: 3A          ABX
F16F: 9B A8       ADDA  $A8
F171: A7 A4       STA   ,Y
F173: BF CA 02    STX   blitter_source
F176: FF CA 00    STU   start_blitter
F179: 3A          ABX
F17A: 9B A8       ADDA  $A8
F17C: A7 A4       STA   ,Y
F17E: BF CA 02    STX   blitter_source
F181: FF CA 00    STU   start_blitter
F184: 3A          ABX
F185: 9B A8       ADDA  $A8
F187: A7 A4       STA   ,Y
F189: BF CA 02    STX   blitter_source
F18C: FF CA 00    STU   start_blitter
F18F: 3A          ABX
F190: 9B A8       ADDA  $A8
F192: A7 A4       STA   ,Y
F194: BF CA 02    STX   blitter_source
F197: FF CA 00    STU   start_blitter
F19A: 3A          ABX
F19B: 9B A8       ADDA  $A8
F19D: A7 A4       STA   ,Y
F19F: BF CA 02    STX   blitter_source
F1A2: FF CA 00    STU   start_blitter
F1A5: 3A          ABX
```

```
F1A6: 9B A8       ADDA  $A8
F1A8: A7 A4       STA   ,Y
F1AA: BF CA 02    STX   blitter_source
F1AD: FF CA 00    STU   start_blitter
F1B0: 3A          ABX
F1B1: 9B A8       ADDA  $A8
F1B3: A7 A4       STA   ,Y
F1B5: BF CA 02    STX   blitter_source
F1B8: FF CA 00    STU   start_blitter
F1BB: 3A          ABX
F1BC: 9B A8       ADDA  $A8
F1BE: A7 A4       STA   ,Y
F1C0: BF CA 02    STX   blitter_source
F1C3: FF CA 00    STU   start_blitter
F1C6: 3A          ABX
F1C7: 9B A8       ADDA  $A8
F1C9: A7 A4       STA   ,Y
F1CB: BF CA 02    STX   blitter_source
F1CE: FF CA 00    STU   start_blitter
F1D1: 1C EF       ANDCC #$EF                ; clear interrupt flag
F1D3: 35 A0       PULS  Y,PC ;(PUL? PC=RTS)



;
;
; X = CA00 (start_blitter)
; U = CA05 (LSB of blitter_dest)
;
;

DRAW_STAGGERED_IMAGE2:
F1D5: A7 C4       STA   ,U
F1D7: E7 84       STB   ,X
F1D9: 9B A8       ADDA  $A8
F1DB: A7 C4       STA   ,U
F1DD: E7 84       STB   ,X
F1DF: 9B A8       ADDA  $A8
F1E1: A7 C4       STA   ,U
F1E3: E7 84       STB   ,X
F1E5: 9B A8       ADDA  $A8
F1E7: A7 C4       STA   ,U
F1E9: E7 84       STB   ,X
F1EB: 9B A8       ADDA  $A8
F1ED: A7 C4       STA   ,U
F1EF: E7 84       STB   ,X
F1F1: 9B A8       ADDA  $A8
F1F3: A7 C4       STA   ,U
F1F5: E7 84       STB   ,X
F1F7: 9B A8       ADDA  $A8
F1F9: A7 C4       STA   ,U
F1FB: E7 84       STB   ,X
F1FD: 9B A8       ADDA  $A8
```

```
F1FF: A7 C4       STA   ,U
F201: E7 84       STB   ,X
F203: 9B A8       ADDA  $A8
F205: A7 C4       STA   ,U
F207: E7 84       STB   ,X
F209: 9B A8       ADDA  $A8
F20B: A7 C4       STA   ,U
F20D: E7 84       STB   ,X
F20F: 9B A8       ADDA  $A8
F211: A7 C4       STA   ,U
F213: E7 84       STB   ,X
F215: 9B A8       ADDA  $A8
F217: A7 C4       STA   ,U
F219: E7 84       STB   ,X
F21B: 9B A8       ADDA  $A8
F21D: A7 C4       STA   ,U
F21F: E7 84       STB   ,X
F221: 9B A8       ADDA  $A8
F223: A7 C4       STA   ,U
F225: E7 84       STB   ,X
F227: 9B A8       ADDA  $A8
F229: A7 C4       STA   ,U
F22B: E7 84       STB   ,X
F22D: 9B A8       ADDA  $A8
F22F: A7 C4       STA   ,U
F231: E7 84       STB   ,X
F233: 1C EF       ANDCC #$EF                ; clear interrupt flag
F235: 39          RTS



;
;
; These instructions beginning at F1D5 each take up 6 bytes.
;
; A7 C4       STA   ,U
; E7 84       STB   ,X
; 9B A8       ADDA  $A8
;
;

F236: E6 A8 11    LDB   $11,Y
F239: C0 10       SUBB  #$10
F23B: 50          NEGB
F23C: 86 06       LDA   #$06                ; size of instruction
group (6 bytes)
F23E: 3D          MUL
F23F: 8E F1 D5    LDX   #$F1D5               ; start of
instructions
F242: 3A          ABX                        ; X+= B. This computes
the address of the first blit to execute.
F243: 34 10       PSHS  X                    ; save address of
function in X to return to on stack
                                             ; so when the RTS
```

```
executes, the blit will be done.
F245: A6 26       LDA   $0006,Y
F247: 97 A8       STA   $A8
F249: EC 29       LDD   $0009,Y
F24B: 1A 10       ORCC  #$10
F24D: F7 CA 05    STB   blitter_dest_l
F250: E6 2F       LDB   $000F,Y
F252: CA 10       ORB   #$10
F254: C4 F7       ANDB  #$F7
F256: CE 00 00    LDU   #$0000
F259: FF CA 01    STU   blitter_mask
F25C: F7 CA 03    STB   blitter_dest_h
F25F: EE 2D       LDU   $000D,Y
F261: FF CA 06    STU   blitter_w_h
F264: CE CA 04    LDU   #blitter_dest
F267: 8E CA 00    LDX   #start_blitter
F26A: 39          RTS

F26B: CE 98 A9    LDU   #$98A9
F26E: 10 AC C4    CMPY  ,U
F271: 27 08       BEQ   $F27B
F273: EE C4       LDU   ,U
F275: 26 F7       BNE   $F26E
F277: 1A 10       ORCC  #$10
F279: 20 FE       BRA   $F279

F27B: EC A4       LDD   ,Y
F27D: ED C4       STD   ,U
F27F: DC AD       LDD   $AD
F281: ED A4       STD   ,Y
F283: 10 9F AD    STY   $AD
F286: 31 C4       LEAY  ,U
F288: 39          RTS

F289: EC 26       LDD   $0006,Y
F28B: 83 00 80    SUBD  #$0080
F28E: A1 26       CMPA  $0006,Y
F290: 26 03       BNE   $F295
F292: E7 27       STB   $0007,Y
F294: 39          RTS

F295: BD F2 36    JSR   $F236
F298: 96 59       LDA   $59
F29A: 26 0B       BNE   $F2A7
F29C: D6 5F       LDB   $5F
F29E: E7 2A       STB   $000A,Y
F2A0: E6 25       LDB   $0005,Y
F2A2: 54          LSRB
F2A3: DB 5E       ADDB  $5E
F2A5: E7 24       STB   $0004,Y
F2A7: A6 2B       LDA   $000B,Y
F2A9: 48          ASLA
F2AA: 4A          DECA
F2AB: 97 AF       STA   $AF
```

```
F2AD: EC 26       LDD   $0006,Y
F2AF: 83 00 80    SUBD  #$0080
F2B2: 4D          TSTA
F2B3: 22 15       BHI   $F2CA
F2B5: CE 98 AB    LDU   #$98AB
F2B8: 7E F2 6E    JMP   $F26E

F2BB: 6A 28       DEC   $0008,Y
F2BD: 27 AC       BEQ   $F26B
F2BF: A6 2B       LDA   $000B,Y
F2C1: 48          ASLA
F2C2: 4A          DECA
F2C3: 97 AF       STA   $AF
F2C5: EC 26       LDD   $0006,Y
F2C7: C3 01 00    ADDD  #$0100
F2CA: ED 26       STD   $0006,Y
F2CC: 97 A8       STA   $A8
F2CE: 44          LSRA
F2CF: E6 25       LDB   $0005,Y
F2D1: 26 01       BNE   $F2D4
F2D3: 4F          CLRA
F2D4: 97 B2       STA   $B2
F2D6: 30 A8 12    LEAX  $12,Y
F2D9: 96 A8       LDA   $A8
F2DB: E6 25       LDB   $0005,Y
F2DD: 3D          MUL
F2DE: DD B0       STD   $B0
F2E0: E6 24       LDB   $0004,Y
F2E2: 4F          CLRA
F2E3: 93 B0       SUBD  $B0
F2E5: DB B2       ADDB  $B2
F2E7: 89 00       ADCA  #$00
F2E9: 26 04       BNE   $F2EF
F2EB: C1 07       CMPB  #$07
F2ED: 22 16       BHI   $F305
F2EF: 0A AF       DEC   $AF
F2F1: DB A8       ADDB  $A8
F2F3: 89 00       ADCA  #$00
F2F5: 26 F8       BNE   $F2EF
F2F7: C1 07       CMPB  #$07
F2F9: 23 F4       BLS   $F2EF
F2FB: E7 29       STB   $0009,Y
F2FD: EC 2B       LDD   $000B,Y
F2FF: 90 AF       SUBA  $AF
F301: 3D          MUL
F302: 3A          ABX
F303: 20 02       BRA   $F307

F305: E7 29       STB   $0009,Y
F307: 96 AF       LDA   $AF
F309: 4A          DECA
F30A: D6 A8       LDB   $A8
F30C: 3D          MUL
F30D: EB 29       ADDB  $0009,Y
```

```
F30F: 89 00       ADCA  #$00
F311: 27 08       BEQ   $F31B
F313: 0A AF       DEC   $AF
F315: D0 A8       SUBB  $A8
F317: 82 00       SBCA  #$00
F319: 26 F8       BNE   $F313
F31B: C1 8F       CMPB  #$8F
F31D: 24 F4       BCC   $F313
F31F: 96 AF       LDA   $AF
F321: 10 27 FF 46 LBEQ  $F26B
F325: A7 A8 11    STA   $11,Y
F328: 80 10       SUBA  #$10
F32A: 40          NEGA
F32B: C6 0B       LDB   #$0B
F32D: 3D          MUL
F32E: C3 F1 24    ADDD  #$F124
F331: 34 26       PSHS  Y,B,A
F333: EE 2F       LDU   $000F,Y
F335: EC 2D       LDD   $000D,Y
F337: 1A 10       ORCC  #$10
F339: FD CA 06    STD   blitter_w_h
F33C: A6 2A       LDA   $000A,Y
F33E: B7 CA 05    STA   blitter_dest_l
F341: E6 2C       LDB   $000C,Y
F343: A6 29       LDA   $0009,Y
F345: 10 8E CA 04 LDY   #blitter_dest
F349: 39          RTS

F34A: 10 9E A9    LDY   $A9
F34D: 27 0B       BEQ   $F35A
F34F: BD F2 36    JSR   $F236
F352: BD F2 BB    JSR   $F2BB
F355: 10 AE A4    LDY   ,Y
F358: 26 F5       BNE   $F34F
F35A: 10 9E AB    LDY   $AB
F35D: 27 08       BEQ   $F367
F35F: BD F2 89    JSR   $F289
F362: 10 AE A4    LDY   ,Y
F365: 26 F8       BNE   $F35F
F367: 39          RTS

F368: 17 21 90 0D 21 08 22 90 0C 22 07 21 41 21 90 0B
F378: 21 41 21 06 21 42 21 90 0A 21 42 21 05 21 43 21
F388: 90 09 21 43 21 04 21 44 21 90 08 21 44 21 03 21
F398: 45 21 90 08 21 44 21 02 21 46 21 90 08 21 44 21
F3A8: 01 21 43 21 43 21 90 08 21 43 21 01 21 43 22 43
F3B8: 21 90 08 21 43 21 01 21 42 21 01 21 43 21 90 C4
F3C8: 08 21 43 23 42 21 01 21 43 21 90 08 21 48 21 01
F3D8: 21 43 21 90 C2 08 2A 01 21 43 21 90 12 21 44 21
F3E8: 90 C3 A0 0B 28 44 21 90 0A 21 4A 23 90 09 21 4B
F3F8: 21 90 08 21 4D 21 90 07 21 44 27 44 21 90 06 21
F408: 44 21 07 21 44 21 90 05 21 44 21 09 21 44 21 90
F418: 04 21 44 2D 44 21 90 03 21 57 21 90 02 21 59 21
F428: 90 01 21 5B 21 90 3F 90 A0
```

```
RESET:
F431: 1A FF       ORCC  #$FF
F433: 10 CE BF 70 LDS   #stacktop
F437: 7F C8 0D    CLR   rom_pia_ctrla
F43A: 7F C8 0C    CLR   rom_pia_dataa
F43D: 86 3C       LDA   #$3C
F43F: B7 C8 0D    STA   rom_pia_ctrla
F442: 7F C8 0F    CLR   rom_pia_ctrlb
F445: 86 C0       LDA   #$C0
F447: B7 C8 0E    STA   rom_pia_datab
F44A: 86 3C       LDA   #$3C
F44C: B7 C8 0F    STA   rom_pia_ctrlb
F44F: 86 C0       LDA   #$C0
F451: B7 C8 0E    STA   rom_pia_datab
F454: 86 01       LDA   #$01
F456: B7 C9 00    STA   rom_enable_scr_ctrl
F459: 8E F6 0B    LDX   #colorpalette1
F45C: 10 8E C0 00 LDY   #color_registers
F460: EC 81       LDD   ,X++
F462: ED A1       STD   ,Y++
F464: 8C F6 1B    CMPX  #$F61B
F467: 25 F7       BCS   $F460
F469: 86 02       LDA   #$02
F46B: 10 8E F4 75 LDY   #$F475
F46F: 8E 00 00    LDX   #$0000
F472: 7E FD 65    JMP   RAM_TEST

F475: 10 8E F4 7C LDY   #$F47C
F479: 7E FF 3F    JMP   CHK_ROM_CHKSUMS

F47C: 86 34       LDA   #$34
F47E: B7 C8 0D    STA   rom_pia_ctrla
F481: B7 C8 0F    STA   rom_pia_ctrlb
F484: 7F C8 0E    CLR   rom_pia_datab
F487: 86 98       LDA   #$98          ;set direct page $9800
F489: 1F 8B       TFR   A,DP
F48B: 10 CE BF 70 LDS   #stacktop
F48F: BD D0 12    JSR   CLR_SCREEN1
F492: 86 01       LDA   #$01                ; INITIAL TESTS
INDICATE:
F494: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
F497: 10 8E D0 00 LDY   #$D000
F49B: 86 07       LDA   #$07
F49D: 7E FE 7F    JMP   $FE7F

F4A0: 86 00       LDA   #$00
F4A2: A7 45       STA   $0005,U
F4A4: 96 CE       LDA   $CE
F4A6: 26 0F       BNE   $F4B7
F4A8: 86 02       LDA   #$02
F4AA: 8E F4 B0    LDX   #$F4B0
F4AD: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call
```

```
F4B0: B6 C8 0C    LDA   rom_pia_dataa
F4B3: 85 02       BITA  #$02
F4B5: 26 03       BNE   $F4BA
F4B7: 7E D0 63    JMP   $D063               ; JMP $D1F3

F4BA: BD D0 60    JSR   $D060               ; JMP $D1FF
F4BD: BD D0 99    JSR   FLIP_SCR_UP1
F4C0: 86 FF       LDA   #$FF
F4C2: 97 CE       STA   $CE
F4C4: 97 59       STA   $59
F4C6: BD F5 F5    JSR   LOAD_F60B_PALETTE
F4C9: BD D0 12    JSR   CLR_SCREEN1
F4CC: B6 C8 0C    LDA   rom_pia_dataa
F4CF: 46          RORA
F4D0: 10 25 06 1A LBCS  $FAEE
F4D4: 1A BF       ORCC  #$BF
F4D6: 10 8E F4 DD LDY   #$F4DD
F4DA: 7E FF 0D    JMP   $FF0D

F4DD: 86 39       LDA   #$39
F4DF: B7 CB FF    STA   watchdog
F4E2: B6 C8 0C    LDA   rom_pia_dataa
F4E5: 85 02       BITA  #$02
F4E7: 26 F4       BNE   $F4DD
F4E9: 10 8E F4 F0 LDY   #$F4F0
F4ED: 7E FF 3F    JMP   CHK_ROM_CHKSUMS

F4F0: 86 98       LDA   #$98           ;set direct page $9800
F4F2: 1F 8B       TFR   A,DP
F4F4: BD D0 12    JSR   CLR_SCREEN1
F4F7: 86 04       LDA   #$04
F4F9: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT  ;print ALL ROMS
OK
F4FC: C6 03       LDB   #$03
F4FE: 8E 70 00    LDX   #$7000
F501: 86 39       LDA   #$39
F503: B7 CB FF    STA   watchdog
F506: B6 C8 0C    LDA   rom_pia_dataa
F509: 85 02       BITA  #$02
F50B: 26 16       BNE   $F523
F50D: 30 1F       LEAX  $FFFF,X
F50F: 8C 00 00    CMPX  #$0000
F512: 26 ED       BNE   $F501
F514: 5A          DECB
F515: 26 E7       BNE   $F4FE
F517: 10 8E F5 23 LDY   #$F523
F51B: 8E 00 00    LDX   #$0000
F51E: 86 FF       LDA   #$FF
F520: 7E FD 65    JMP   RAM_TEST

F523: 86 01       LDA   #$01
F525: B7 C9 00    STA   rom_enable_scr_ctrl
F528: 86 98       LDA   #$98           ;set direct page $9800
```

```
F52A: 1F 8B       TFR   A,DP
F52C: BD D0 12    JSR   CLR_SCREEN1
F52F: 86 05       LDA   #$05           ;print NO
F531: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
F534: 86 39       LDA   #$39
F536: B7 CB FF    STA   watchdog
F539: B6 C8 0C    LDA   rom_pia_dataa
F53C: 85 02       BITA  #$02
F53E: 26 F4       BNE   $F534
F540: 8E 98 00    LDX   #$9800
F543: 4F          CLRA
F544: A7 80       STA   ,X+
F546: C6 39       LDB   #$39
F548: F7 CB FF    STB   watchdog
F54B: 8C BF 71    CMPX  #$BF71
F54E: 25 F4       BCS   $F544
F550: BD F5 F5    JSR   LOAD_F60B_PALETTE
F553: CC A5 5A    LDD   #$A55A
F556: DD 85       STD   $85
F558: 97 CE       STA   $CE
F55A: BD D0 36    JSR   $D036              ; JMP $D6EC
F55D: BD D0 C0    JSR   $D0C0
F560: BD D0 99    JSR   FLIP_SCR_UP1
F563: 86 FF       LDA   #$FF
F565: 97 59       STA   $59
F567: BD D0 54    JSR   $D054              ; JMP $D281 - reserve
object metadata entry and call function
F56A: F5 73       ; pointer to function
F56C: 8D 7A       BSR   $F5E8
F56E: 1C 00       ANDCC #$00               ; clear all flags
F570: 7E D0 96    JMP   $D096              ; JMP $D196

F573: BD FA AF    JSR   $FAAF
F576: BD FD 00    JSR   $FD00
F579: 1C 01       ANDCC #$01               ; preserve carry flag
and clear all others
F57B: 86 06       LDA   #$06
F57D: 24 28       BCC   $F5A7
F57F: C6 2F       LDB   #$2F
F581: 8C CD 00    CMPX  #credits_cmos
F584: 22 02       BHI   $F588
F586: C6 1F       LDB   #$1F
F588: 1A 10       ORCC  #$10
F58A: 10 CE F5 93 LDS   #$F593
F58E: 86 03       LDA   #$03
F590: 7E FE 93    JMP   $FE93

F593: 10 CE BF 70 LDS   #stacktop
F597: 8D 4F       BSR   $F5E8
F599: 86 98       LDA   #$98               ; set direct page $9800
F59B: 1F 8B       TFR   A,DP
F59D: 1C EF       ANDCC #$EF               ; clear interrupt flag
F59F: 86 07       LDA   #$07
F5A1: C1 1F       CMPB  #$1F
```

```
F5A3: 22 02       BHI   $F5A7
F5A5: 86 08       LDA   #$08
F5A7: BD D0 12    JSR   CLR_SCREEN1
F5AA: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
F5AD: BD FA AF    JSR   $FAAF
F5B0: DE 15       LDU   $15
F5B2: 6F 49       CLR   $0009,U
F5B4: BD FC 90    JSR   $FC90
F5B7: BD FC 99    JSR   $FC99
F5BA: BD FA DD    JSR   $FADD
F5BD: 24 F8       BCC   $F5B7
F5BF: 86 3F       LDA   #$3F
F5C1: B7 C8 0E    STA   rom_pia_datab
F5C4: 86 01       LDA   #$01
F5C6: 8E F5 CC    LDX   #$F5CC
F5C9: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

F5CC: 86 2C       LDA   #$2C
F5CE: B7 C8 0E    STA   rom_pia_datab
F5D1: BD FA AF    JSR   $FAAF
F5D4: BD F9 28    JSR   $F928
F5D7: BD FA AF    JSR   $FAAF
F5DA: BD F8 88    JSR   $F888
F5DD: BD FA DD    JSR   $FADD
F5E0: 24 03       BCC   $F5E5
F5E2: BD FA AF    JSR   $FAAF
F5E5: 7E F6 77    JMP   $F677

F5E8: 7F C8 0E    CLR   rom_pia_datab
F5EB: 86 34       LDA   #$34
F5ED: B7 C8 0D    STA   rom_pia_ctrla
F5F0: 4C          INCA
F5F1: B7 C8 0F    STA   rom_pia_ctrlb
F5F4: 39          RTS

LOAD_F60B_PALETTE:
F5F5: 8E F6 0B    LDX   #colorpalette1
F5F8: 10 8E 98 00 LDY   #$9800
F5FC: CE C0 00    LDU   #color_registers
F5FF: EC 81       LDD   ,X++
F601: ED A1       STD   ,Y++
F603: ED C1       STD   ,U++
F605: 8C F6 1B    CMPX  #$F61B
F608: 25 F5       BCS   $F5FF
F60A: 39          RTS

colorpalette1:
F60B: 00 07 17 C7 1F 3F 38 C0 A4 FF 38 17 CC 81 81 07

F61B: 86 3F       LDA   #$3F
F61D: 1F 8A       TFR   A,CC
F61F: 8D D4       BSR   LOAD_F60B_PALETTE
F621: 86 85       LDA   #$85
```

```
F623: BE B9 EA    LDX   $B9EA
F626: 30 89 12 34 LEAX  $1234,X
F62A: 10 8E F6 31 LDY   #$F631
F62E: 7E FD 65    JMP   RAM_TEST

F631: 10 8E F6 38 LDY   #$F638
F635: 7E FF 3F 86 98 1F 8B 10 CE BF 70 BD FD 00 24 1F
F645: 86 03 8D 26 86 08 8C CD 00 23 02 86 07 C6 39 F7
F655: CB FF BD D0 12 BD 5F 99 86 39 B7 CB FF 20 F9 8D
F665: 47 10 8E F6 1B 86 04 7E FE 7F 10 8E F6 76 BD FF
F675: 1D 39 BD F6 FE BD FA AF BD D0 12 86 07 97 00 BD
F685: FA AF 86 38 97 00 BD FA AF 86 C0 97 00 BD FA AF
F695: 8D 16 BD FA AF 7E FA EE 9F 2B 30 89 10 00 30 89
F6A5: FF 00 8C 98 00 22 F7 39 8E 98 00 10 8E F6 EE CE
F6B5: C0 00 EC A1 ED 81 ED C1 86 39 B7 CB FF 8C 98 10
F6C5: 25 F0 CC 00 00 8E 00 00 8D CE ED 83 34 02 86 39
F6D5: B7 CB FF 35 02 9C 2B 26 F1 30 89 09 00 4D 26 03
F6E5: 8E 0D 00 C3 11 11 24 E0 39 05 05 28 28 80 80 00
F6F5: 00 AD AD 2D 2D A8 A8 85 85

F6FE: BD D0 12    JSR   CLR_SCREEN1
F701: 4F          CLRA
F702: BD F9 1D    JSR   $F91D
F705: 86 FF       LDA   #$FF
F707: 97 01       STA   $01
F709: 86 C0       LDA   #$C0
F70B: 97 02       STA   $02
F70D: 86 38       LDA   #$38
F70F: 97 03       STA   $03
F711: 86 07       LDA   #$07
F713: 97 04       STA   $04
F715: 10 8E F8 10 LDY   #$F810
F719: CC 01 01    LDD   #$0101
F71C: AE A4       LDX   ,Y
F71E: ED 81       STD   ,X++
F720: AC 22       CPX   $0002,Y
F722: 26 FA       BNE   $F71E
F724: 31 24       LEAY  $0004,Y
F726: 10 8C F8 38 CMPY  #$F838
F72A: 26 F0       BNE   $F71C
F72C: 86 11       LDA   #$11
F72E: 10 8E F7 F0 LDY   #$F7F0
F732: AE A4       LDX   ,Y
F734: 9F 2B       STX   $2B
F736: A7 84       STA   ,X
F738: 0C 2B       INC   $2B
F73A: 9E 2B       LDX   $2B
F73C: AC 22       CPX   $0002,Y
F73E: 26 F6       BNE   $F736
F740: 31 24       LEAY  $0004,Y
F742: 10 8C F8 10 CMPY  #$F810
F746: 26 EA       BNE   $F732
F748: 10 8E F8 38 LDY   #$F838
F74C: AE A4       LDX   ,Y
```

```
F74E: 9F 2B       STX   $2B
F750: A6 24       LDA   $0004,Y
F752: A7 84       STA   ,X
F754: 0C 2B       INC   $2B
F756: 9E 2B       LDX   $2B
F758: AC 22       CPX   $0002,Y
F75A: 26 F6       BNE   $F752
F75C: 31 25       LEAY  $0005,Y
F75E: 10 8C F8 74 CMPY  #$F874
F762: 26 E8       BNE   $F74C
F764: 10 8E F8 74 LDY   #$F874
F768: AE A4       LDX   ,Y
F76A: A6 24       LDA   $0004,Y
F76C: A7 80       STA   ,X+
F76E: AC 22       CPX   $0002,Y
F770: 26 FA       BNE   $F76C
F772: 31 25       LEAY  $0005,Y
F774: 10 8C F8 88 CMPY  #$F888
F778: 26 EE       BNE   $F768
F77A: 86 21       LDA   #$21
F77C: B7 43 7E    STA   $437E
F77F: 86 20       LDA   #$20
F781: B7 93 7E    STA   $937E
F784: 8E 4B 0A    LDX   #$4B0A
F787: 1A 10       ORCC  #$10
F789: 7F C9 00    CLR   rom_enable_scr_ctrl
F78C: A6 84       LDA   ,X
F78E: C6 01       LDB   #$01
F790: F7 C9 00    STB   rom_enable_scr_ctrl
F793: 1C EF       ANDCC #$EF                ; clear interrupt flag
F795: 84 F0       ANDA  #$F0
F797: 8A 02       ORA   #$02
F799: A7 80       STA   ,X+
F79B: 8C 4B 6D    CMPX  #$4B6D
F79E: 26 E7       BNE   $F787
F7A0: 8E 4B 90    LDX   #$4B90
F7A3: 1A 10       ORCC  #$10
F7A5: 7F C9 00    CLR   rom_enable_scr_ctrl
F7A8: A6 84       LDA   ,X
F7AA: C6 01       LDB   #$01
F7AC: F7 C9 00    STB   rom_enable_scr_ctrl
F7AF: 1C EF       ANDCC #$EF                ; clear interrupt flag
F7B1: 84 F0       ANDA  #$F0
F7B3: 8A 02       ORA   #$02
F7B5: A7 80       STA   ,X+
F7B7: 8C 4B F3    CMPX  #$4BF3
F7BA: 26 E7       BNE   $F7A3
F7BC: 8E 0B 18    LDX   #$0B18
F7BF: 9F 2B       STX   $2B
F7C1: 9E 2B       LDX   $2B
F7C3: 1A 10       ORCC  #$10
F7C5: 7F C9 00    CLR   rom_enable_scr_ctrl
F7C8: A6 84       LDA   ,X
F7CA: C6 01       LDB   #$01
```

```
F7CC: F7 C9 00    STB   rom_enable_scr_ctrl
F7CF: 1C EF       ANDCC #$EF                ; clear interrupt flag
F7D1: 84 F0       ANDA  #$F0
F7D3: 8A 01       ORA   #$01
F7D5: A7 84       STA   ,X
F7D7: D6 2C       LDB   $2C
F7D9: CB 22       ADDB  #$22
F7DB: 25 04       BCS   $F7E1
F7DD: D7 2C       STB   $2C
F7DF: 20 E0       BRA   $F7C1

F7E1: C6 18       LDB   #$18
F7E3: D7 2C       STB   $2C
F7E5: D6 2B       LDB   $2B
F7E7: CB 10       ADDB  #$10
F7E9: D7 2B       STB   $2B
F7EB: C1 9B       CMPB  #$9B
F7ED: 26 D2       BNE   $F7C1
F7EF: 39          RTS

F7F0: 04 07       LSR   $07
F7F2: 94 07       ANDA  $07
F7F4: 04 29       LSR   $29
F7F6: 94 29       ANDA  $29
F7F8: 04 4B       LSR   $4B
F7FA: 94 4B       ANDA  $4B
F7FC: 04 6D       LSR   $6D
F7FE: 94 6D       ANDA  $6D
F800: 04 8F       LSR   $8F
F802: 94 8F       ANDA  $8F
F804: 04 B1       LSR   $B1
F806: 94 B1       ANDA  $B1
F808: 04 D3       LSR   $D3
F80A: 94 D3       ANDA  $D3
F80C: 04 F5       LSR   $F5
F80E: 94 F5       ANDA  $F5
F810: 03 07       COM   $07
F812: 03 F5       COM   $F5
F814: 13          SYNC
F815: 07 13       ASR   $13
F817: F5 23 07    BITB  $2307
F81A: 23 F5       BLS   $F811
F81C: 33 07       LEAU  $0007,X
F81E: 33 F5       LEAU  [B,S]
F820: 43          COMA
F821: 07 43       ASR   $43
F823: F5 53 07    BITB  $5307
F826: 53          COMB
F827: F5 63 07    BITB  $6307
F82A: 63 F5       COM   [B,S]
F82C: 73 07 73    COM   $0773
F82F: F5 83 07    BITB  $8307
F832: 83 F5 93    SUBD  #$F593
F835: 07 93       ASR   $93
```

```
F837: F5 45 05    BITB  $4505
F83A: 52          Illegal Opcode
F83B: 05          Illegal Opcode
F83C: 44          LSRA
F83D: 45          Illegal Opcode
F83E: 06 52       ROR   $52
F840: 06 44       ROR   $44
F842: 45          Illegal Opcode
F843: 07 52       ASR   $52
F845: 07 00       ASR   $00
F847: 45          Illegal Opcode
F848: 08 52       ASL   $52
F84A: 08 33       ASL   $33
F84C: 45          Illegal Opcode
F84D: 09 52       ROL   $52
F84F: 09 33       ROL   $33
F851: 45          Illegal Opcode
F852: F3 52 F3    ADDD  $52F3
F855: 33 45       LEAU  $0005,U
F857: F4 52 F4    ANDB  $52F4
F85A: 33 45       LEAU  $0005,U
F85C: F5 52 F5    BITB  $52F5
F85F: 00 45       NEG   $45
F861: F6 52 F6    LDB   $52F6
F864: 44          LSRA
F865: 45          Illegal Opcode
F866: F7 52 F7    STB   $52F7
F869: 44          LSRA
F86A: 04 7E       LSR   $7E
F86C: 43          COMA
F86D: 7E 22 54    JMP   $2254

F870: 7E 93 7E    JMP   $937E

F873: 22 02       BHI   $F877
F875: 6F 02       CLR   $0002,X
F877: 8E 04 03    LDX   #$0403
F87A: 6F 03       CLR   $0003,X
F87C: 8E 30 93    LDX   #$3093
F87F: 6F 93       CLR   [,--X]
F881: 8E 00 94    LDX   #$0094
F884: 6F 94       CLR   [,X]
F886: 8E 34 35    LDX   #$3435
F889: 06 DE       ROR   $DE
F88B: 15          Illegal Opcode
F88C: ED 4D       STD   $000D,U
F88E: BD D0 12    JSR   CLR_SCREEN1
F891: 86 1E       LDA   #$1E
F893: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT        ; print
COLOR RAM TEST
F896: 86 80       LDA   #$80
F898: A7 47       STA   $0007,U
F89A: 86 01       LDA   #$01
F89C: 8E F8 A2    LDX   #$F8A2
```

```
F89F: 7E D0 66    JMP   $D066                ; JMP $D1E3 - allocate
function call

F8A2: BD FA DD    JSR   $FADD
F8A5: 25 34       BCS   $F8DB
F8A7: 6A 47       DEC   $0007,U
F8A9: 26 EF       BNE   $F89A
F8AB: B6 F9 05    LDA   $F905
F8AE: 8D 6D       BSR   $F91D
F8B0: 8D 2E       BSR   $F8E0
F8B2: 8E F9 05    LDX   #$F905
F8B5: A6 80       LDA   ,X+
F8B7: DE 15       LDU   $15
F8B9: AF 49       STX   $0009,U
F8BB: 8D 60       BSR   $F91D
F8BD: 86 80       LDA   #$80
F8BF: A7 47       STA   $0007,U
F8C1: 86 01       LDA   #$01
F8C3: 8E F8 C9    LDX   #$F8C9
F8C6: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

F8C9: BD FA DD    JSR   $FADD
F8CC: 25 0D       BCS   $F8DB
F8CE: 6A 47       DEC   $0007,U
F8D0: 26 EF       BNE   $F8C1
F8D2: AE 49       LDX   $0009,U
F8D4: 8C F9 0D    CMPX  #$F90D
F8D7: 25 DC       BCS   $F8B5
F8D9: 20 D7       BRA   $F8B2

F8DB: DE 15       LDU   $15
F8DD: 6E D8 0D    JMP   [$0D,U]

F8E0: 8E 00 00    LDX   #$0000
F8E3: 10 8E F9 0D LDY   #$F90D
F8E7: BD F6 9D    JSR   $F69D
F8EA: A6 A0       LDA   ,Y+
F8EC: 1F 89       TFR   A,B
F8EE: ED 83       STD   ,--X
F8F0: 9C 2B       CPX   $2B
F8F2: 26 FA       BNE   $F8EE
F8F4: 30 89 09 00 LEAX  $0900,X
F8F8: 4D          TSTA
F8F9: 26 03       BNE   $F8FE
F8FB: 8E 0D 00    LDX   #$0D00
F8FE: 10 8C F9 1D CMPY  #$F91D
F902: 26 E3       BNE   $F8E7
F904: 39          RTS

F905: 02          Illegal Opcode
F906: 03 04       COM   $04
F908: 10 18       Illegal Opcode
F90A: 20 40       BRA   $F94C
```

```
F90C: 80 00       SUBA  #$00
F90E: FF 11 EE    STU   $11EE
F911: 22 DD       BHI   $F8F0
F913: 33 CC 44    LEAU  $44,U
F916: BB 55 AA    ADDA  $55AA
F919: 66 99 77 88 ROR   [$7788,X]
F91D: 8E 98 00    LDX   #$9800
F920: A7 80       STA   ,X+
F922: 8C 98 10    CMPX  #$9810
F925: 25 F9       BCS   $F920
F927: 39          RTS

F928: 35 06       PULS  A,B
F92A: DE 15       LDU   $15
F92C: ED 4D       STD   $000D,U
F92E: 86 0A       LDA   #$0A
F930: A7 4B       STA   $000B,U
F932: BD D0 12    JSR   CLR_SCREEN1
F935: 86 0C       LDA   #$0C
F937: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT        ;print
SWITCH TEST
F93A: CE B3 EA    LDU   #hs_inits
F93D: 6F C0       CLR   ,U+
F93F: 11 83 B3 F4 CMPU  #$B3F4
F943: 23 F8       BLS   $F93D
F945: CE F9 E1    LDU   #$F9E1
F948: 8D 26       BSR   $F970
F94A: 86 34       LDA   #$34
F94C: B7 C8 07    STA   widget_pia_ctrlb
F94F: 8D 1F       BSR   $F970
F951: 86 3C       LDA   #$3C
F953: B7 C8 07    STA   widget_pia_ctrlb
F956: 8D 28       BSR   $F980
F958: BD FA DD    JSR   $FADD
F95B: 24 06       BCC   $F963
F95D: DE 15       LDU   $15
F95F: 6A 4B       DEC   $000B,U
F961: 27 08       BEQ   $F96B
F963: 86 01       LDA   #$01
F965: 8E F9 45    LDX   #$F945
F968: 7E D0 66    JMP   $D066              ; JMP $D1E3 - allocate
function call

F96B: DE 15       LDU   $15
F96D: 6E D8 0D    JMP   [$0D,U]

F970: AE C1       LDX   ,U++
F972: 27 0B       BEQ   $F97F
F974: 10 AE C1    LDY   ,U++
F977: A6 84       LDA   ,X
F979: A8 A4       EORA  ,Y
F97B: A7 21       STA   $0001,Y
F97D: 20 F1       BRA   $F970
```

```
F97F: 39          RTS

F980: CE F9 F9    LDU   #$F9F9
F983: 10 8E B3 EA LDY   #hs_inits
F987: C6 01       LDB   #$01
F989: E5 21       BITB  $0001,Y
F98B: 27 02       BEQ   $F98F
F98D: 8D 19       BSR   $F9A8
F98F: 33 43       LEAU  $0003,U
F991: 58          ASLB
F992: 24 F5       BCC   $F989
F994: 31 22       LEAY  $0002,Y
F996: 10 8C B3 F3 CMPY  #$B3F3
F99A: 22 0B       BHI   $F9A7
F99C: B6 C8 06    LDA   widget_pia_datab
F99F: 2B E6       BMI   $F987
F9A1: 10 8C B3 EF CMPY  #$B3EF
F9A5: 23 E0       BLS   $F987
F9A7: 39          RTS

F9A8: 34 14       PSHS  X,B
F9AA: 86 3F       LDA   #$3F
F9AC: B7 C8 0E    STA   rom_pia_datab
F9AF: E8 A4       EORB  ,Y
F9B1: E7 A4       STB   ,Y
F9B3: E6 E4       LDB   ,S
F9B5: E5 A4       BITB  ,Y
F9B7: 26 10       BNE   $F9C9
F9B9: E6 42       LDB   $0002,U
F9BB: 27 22       BEQ   $F9DF
F9BD: 86 40       LDA   #$40
F9BF: 1F 01       TFR   D,X
F9C1: CC 30 06    LDD   #$3006
F9C4: BD D0 1B    JSR   $D01B               ; JMP $DADF - clear
rectangle to black
F9C7: 35 94       PULS  B,X,PC ;(PUL? PC=RTS)

F9C9: E6 42       LDB   $0002,U
F9CB: 27 12       BEQ   $F9DF
F9CD: 86 40       LDA   #$40
F9CF: 1F 01       TFR   D,X
F9D1: C6 BB       LDB   #$BB
F9D3: D7 CF       STB   $CF
F9D5: EC C4       LDD   ,U
F9D7: BD 5F 96    JSR   $5F96               ; JMP $613F: print
string in small font
F9DA: 86 37       LDA   #$37
F9DC: B7 C8 0E    STA   rom_pia_datab
F9DF: 35 94       PULS  B,X,PC ;(PUL? PC=RTS)

F9E1: C8 0C
F9E3: B3 EA
F9E5: C8 04
```

```
F9E7: B3 EC
F9E9: C8 06
F9EB: B3 EE
F9ED: 00 00
F9EF: C8 04
F9F1: B3 F0
F9F3: C8 06
F9F5: B3 F2
F9F7: 00 00

F9F9: 0D 00 2C 0E 00 33 0F 00 3A 10 00 41 11 00 48 12
FA09: 00 4F 13 00 56 00 00 00 14 01 5D 15 01 64 16 01
FA19: 6B 17 01 72 18 00 79 19 00 80 1A 01 87 1B 01 8E
FA29: 1C 01 95 1D 01 9C 00 00 00 00 00 00 00 00 00 00
FA39: 00 00 00 00 00 00 00 00 14 02 A3 15 02 AA 16 02
FA49: B1 17 02 B8 00 00 00 00 00 00 1A 02 BF 1B 02 C6
FA59: 1C 02 CD 1D 02 D4 00 00 00 00 00 00 00 00 00 00
FA69: 00 00 00 00 00 00 00 00

FA71: CE F5 40    LDU   #$F540
FA74: 20 03       BRA   $FA79

FA76: CE F5 17    LDU   #$F517
FA79: 10 CE BF 70 LDS   #stacktop
FA7D: 10 8E FA 86 LDY   #$FA86
FA81: 86 01       LDA   #$01
FA83: 7E FE 7F    JMP   $FE7F

FA86: B6 C8 0C    LDA   rom_pia_dataa
FA89: 85 02       BITA  #$02
FA8B: 26 F0       BNE   $FA7D
FA8D: 10 8E FA 96 LDY   #$FA96
FA91: 86 01       LDA   #$01
FA93: 7E FE 7F    JMP   $FE7F

FA96: B6 C8 0C    LDA   rom_pia_dataa
FA99: 85 02       BITA  #$02
FA9B: 27 F0       BEQ   $FA8D
FA9D: 10 8E FA A6 LDY   #$FAA6
FAA1: 86 01       LDA   #$01
FAA3: 7E FE 7F    JMP   $FE7F

FAA6: B6 C8 0C    LDA   rom_pia_dataa
FAA9: 85 02       BITA  #$02
FAAB: 26 F0       BNE   $FA9D
FAAD: 6E C4       JMP   ,U

FAAF: 35 06       PULS  A,B
FAB1: DE 15       LDU   $15
FAB3: ED 4D       STD   $000D,U
FAB5: B6 C8 0C    LDA   rom_pia_dataa
FAB8: 85 02       BITA  #$02
FABA: 26 08       BNE   $FAC4
FABC: 86 01       LDA   #$01
```

```
FABE: 8E FA B5    LDX   #$FAB5
FAC1: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

FAC4: 0F 00       CLR   $00
FAC6: BD D0 12    JSR   CLR_SCREEN1
FAC9: 20 07       BRA   $FAD2

FACB: B6 C8 0C    LDA   rom_pia_dataa
FACE: 85 02       BITA  #$02
FAD0: 27 08       BEQ   $FADA
FAD2: 86 02       LDA   #$02
FAD4: 8E FA CB    LDX   #$FACB
FAD7: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

FADA: 6E D8 0D    JMP   [$0D,U]

FADD: 34 02       PSHS  A
FADF: B6 C8 0C    LDA   rom_pia_dataa
FAE2: 85 02       BITA  #$02
FAE4: 27 04       BEQ   $FAEA
FAE6: 1A 01       ORCC  #$01
FAE8: 35 82       PULS  A,PC ;(PUL? PC=RTS)

FAEA: 1C FE       ANDCC #$FE                ; clear carry flag
FAEC: 35 82       PULS  A,PC ;(PUL? PC=RTS)

FAEE: 86 FF       LDA   #$FF
FAF0: 97 59       STA   $59
FAF2: BD F5 F5    JSR   LOAD_F60B_PALETTE
FAF5: 8D E6       BSR   $FADD
FAF7: 24 02       BCC   $FAFB
FAF9: 8D B4       BSR   $FAAF
FAFB: 86 1F       LDA   #$1F
FAFD: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
FB00: CE CD 02    LDU   #$CD02
FB03: 86 20       LDA   #$20
FB05: 34 02       PSHS  A
FB07: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
FB0A: 1E 31       EXG   U,X
FB0C: BD D0 A5    JSR   $D0A5
FB0F: 34 04       PSHS  B
FB11: BD D0 A8    JSR   $D0A8
FB14: 1F 02       TFR   D,Y
FB16: 35 04       PULS  B
FB18: 1E 31       EXG   U,X
FB1A: 8D 71       BSR   $FB8D
FB1C: 35 02       PULS  A
FB1E: 4C          INCA
FB1F: 11 83 CD 32 CMPU  #top_score
FB23: 25 E0       BCS   $FB05
FB25: 86 6A       LDA   #$6A
FB27: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
```

```
FB2A: 34 10       PSHS  X
FB2C: 8E CD 20    LDX   #$CD20
FB2F: 10 8E CD 2C LDY   #$CD2C
FB33: BD FB BF    JSR   $FBBF
FB36: 35 10       PULS  X
FB38: F6 B4 18    LDB   $B418
FB3B: 10 BE B4 19 LDY   $B419
FB3F: 8D 4C       BSR   $FB8D
FB41: B6 B4 1B    LDA   $B41B
FB44: 84 0F       ANDA  #$0F
FB46: B7 B4 1C    STA   $B41C
FB49: B6 B4 1B    LDA   $B41B
FB4C: 85 10       BITA  #$10
FB4E: 27 08       BEQ   $FB58
FB50: 84 EF       ANDA  #$EF
FB52: 44          LSRA
FB53: 8B 05       ADDA  #$05
FB55: 19          DAA
FB56: 20 01       BRA   $FB59

FB58: 44          LSRA
FB59: BB B4 1C    ADDA  $B41C
FB5C: 19          DAA
FB5D: 1F 89       TFR   A,B
FB5F: 86 6B       LDA   #$6B
FB61: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
FB64: 86 6C       LDA   #$6C
FB66: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
FB69: 34 10       PSHS  X
FB6B: 8E CD 26    LDX   #$CD26
FB6E: 10 8E CD 2C LDY   #$CD2C
FB72: 8D 4B       BSR   $FBBF
FB74: 35 10       PULS  X
FB76: F6 B4 18    LDB   $B418
FB79: 10 BE B4 19 LDY   $B419
FB7D: 8D 0E       BSR   $FB8D
FB7F: F6 B4 1B    LDB   $B41B
FB82: 86 6D       LDA   #$6D
FB84: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
FB87: BD FA AF    JSR   $FAAF
FB8A: 7E 6F 00    JMP   $6F00

FB8D: 5D          TSTB
FB8E: 27 0C       BEQ   $FB9C
FB90: 86 29       LDA   #$29
FB92: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
FB95: 86 2A       LDA   #$2A
FB97: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
FB9A: 20 05       BRA   $FBA1

FB9C: 86 2B       LDA   #$2B
FB9E: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
FBA1: 39          RTS
```

```
FBA2: 34 04       PSHS  B
FBA4: BD D0 A2    JSR   LDA_NIB_X1
FBA7: 1F 89       TFR   A,B
FBA9: 84 0F       ANDA  #$0F
FBAB: 81 09       CMPA  #$09
FBAD: 23 02       BLS   $FBB1
FBAF: 86 09       LDA   #$09
FBB1: C4 F0       ANDB  #$F0
FBB3: C1 90       CMPB  #$90
FBB5: 23 02       BLS   $FBB9
FBB7: C6 90       LDB   #$90
FBB9: 34 04       PSHS  B
FBBB: AA E0       ORA   ,S+
FBBD: 35 84       PULS  B,PC ;(PUL? PC=RTS)

FBBF: 34 76       PSHS  U,Y,X,B,A
FBC1: CC 00 00    LDD   #$0000
FBC4: FD B4 18    STD   $B418
FBC7: FD B4 1A    STD   $B41A
FBCA: 8D D6       BSR   $FBA2
FBCC: B7 B4 06    STA   $B406
FBCF: 8D D1       BSR   $FBA2
FBD1: B7 B4 07    STA   $B407
FBD4: 8D CC       BSR   $FBA2
FBD6: B7 B4 08    STA   $B408
FBD9: CC 00 00    LDD   #$0000
FBDC: B7 B4 09    STA   $B409
FBDF: FD B4 0A    STD   $B40A
FBE2: B7 B4 15    STA   $B415
FBE5: FD B4 16    STD   $B416
FBE8: 1F 21       TFR   Y,X
FBEA: BD FB A2    JSR   $FBA2
FBED: B7 B4 12    STA   $B412
FBF0: BD FB A2    JSR   $FBA2
FBF3: B7 B4 13    STA   $B413
FBF6: BD FB A2    JSR   $FBA2
FBF9: B7 B4 14    STA   $B414
FBFC: 26 05       BNE   $FC03
FBFE: FC B4 12    LDD   $B412
FC01: 27 3A       BEQ   $FC3D
FC03: CE B4 1A    LDU   #$B41A
FC06: 7D B4 12    TST   $B412
FC09: 26 12       BNE   $FC1D
FC0B: 33 5F       LEAU  $FFFF,U
FC0D: C6 05       LDB   #$05
FC0F: 8E B4 12    LDX   #$B412
FC12: A6 01       LDA   $0001,X
FC14: A7 80       STA   ,X+
FC16: 5A          DECB
FC17: 26 F9       BNE   $FC12
FC19: 6F 84       CLR   ,X
FC1B: 20 E9       BRA   $FC06

FC1D: 8E B4 06    LDX   #$B406
```

```
FC20: 8D 1D       BSR   $FC3F
FC22: 24 FC       BCC   $FC20
FC24: 10 8E B4 17 LDY   #$B417
FC28: C6 05       LDB   #$05
FC2A: A6 3F       LDA   $FFFF,Y
FC2C: A7 A4       STA   ,Y
FC2E: 31 3F       LEAY  $FFFF,Y
FC30: 5A          DECB
FC31: 26 F7       BNE   $FC2A
FC33: 6F A4       CLR   ,Y
FC35: 33 41       LEAU  $0001,U
FC37: 11 83 B4 1B CMPU  #$B41B
FC3B: 23 E3       BLS   $FC20
FC3D: 35 F6       PULS  A,B,X,Y,U,PC ;(PUL? PC=RTS)

FC3F: 10 8E B4 12 LDY   #$B412
FC43: 86 00       LDA   #$00
FC45: E6 86       LDB   A,X
FC47: E0 A0       SUBB  ,Y+
FC49: 22 07       BHI   $FC52
FC4B: 25 40       BCS   $FC8D
FC4D: 4C          INCA
FC4E: 81 06       CMPA  #$06
FC50: 25 F3       BCS   $FC45
FC52: C6 06       LDB   #$06
FC54: 10 8E B4 0C LDY   #$B40C
FC58: 86 99       LDA   #$99
FC5A: A0 26       SUBA  $0006,Y
FC5C: A7 A0       STA   ,Y+
FC5E: 5A          DECB
FC5F: 26 F7       BNE   $FC58
FC61: C6 06       LDB   #$06
FC63: 1C FE       ANDCC #$FE                ; clear carry flag
FC65: 86 01       LDA   #$01
FC67: A9 A2       ADCA  ,-Y
FC69: 19          DAA
FC6A: A7 A4       STA   ,Y
FC6C: 86 00       LDA   #$00
FC6E: 5A          DECB
FC6F: 26 F6       BNE   $FC67
FC71: C6 05       LDB   #$05
FC73: 10 8E B4 12 LDY   #$B412
FC77: 1C FE       ANDCC #$FE                ; clear carry flag
FC79: A6 A2       LDA   ,-Y
FC7B: A9 85       ADCA  B,X
FC7D: 19          DAA
FC7E: A7 85       STA   B,X
FC80: 5A          DECB
FC81: 2A F6       BPL   $FC79
FC83: A6 C4       LDA   ,U
FC85: 8B 01       ADDA  #$01
FC87: 19          DAA
FC88: A7 C4       STA   ,U
FC8A: 1C FE       ANDCC #$FE                ; clear carry flag
```

```
FC8C: 39          RTS

FC8D: 1A 01       ORCC  #$01
FC8F: 39          RTS

FC90: BD D0 12    JSR   CLR_SCREEN1
FC93: CC FE 01    LDD   #$FE01
FC96: ED 47       STD   $0007,U
FC98: 39          RTS

FC99: 35 06       PULS  A,B
FC9B: ED 4D       STD   $000D,U
FC9D: 86 3F       LDA   #$3F
FC9F: B7 C8 0E    STA   rom_pia_datab
FCA2: 86 01       LDA   #$01
FCA4: 8E FC AA    LDX   #$FCAA
FCA7: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

FCAA: 86 2C       LDA   #$2C
FCAC: B7 C8 0E    STA   rom_pia_datab
FCAF: 86 01       LDA   #$01
FCB1: 8E FC B7    LDX   #$FCB7
FCB4: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

FCB7: 86 3F       LDA   #$3F
FCB9: B7 C8 0E    STA   rom_pia_datab
FCBC: 86 01       LDA   #$01
FCBE: 8E FC C4    LDX   #$FCC4
FCC1: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

FCC4: EC 47       LDD   $0007,U
FCC6: 84 3F       ANDA  #$3F
FCC8: B7 C8 0E    STA   rom_pia_datab
FCCB: 86 09       LDA   #$09
FCCD: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
FCD0: 86 40       LDA   #$40
FCD2: A7 4B       STA   $000B,U
FCD4: 86 01       LDA   #$01
FCD6: 8E FC DC    LDX   #$FCDC
FCD9: 7E D0 66    JMP   $D066               ; JMP $D1E3 - allocate
function call

FCDC: BD FA DD    JSR   $FADD
FCDF: 25 04       BCS   $FCE5
FCE1: 6A 4B       DEC   $000B,U
FCE3: 26 EF       BNE   $FCD4
FCE5: A6 49       LDA   $0009,U
FCE7: 26 06       BNE   $FCEF
FCE9: B6 C8 0C    LDA   rom_pia_dataa
FCEC: 46          RORA
FCED: 24 0E       BCC   $FCFD
```

```
FCEF: EC 47       LDD   $0007,U
FCF1: 1A 01       ORCC  #$01
FCF3: 49          ROLA
FCF4: 5C          INCB
FCF5: C1 07       CMPB  #$07
FCF7: 25 02       BCS   $FCFB
FCF9: 8D 98       BSR   $FC93
FCFB: ED 47       STD   $0007,U
FCFD: 6E D8 0D    JMP   [$0D,U]

FD00: 8E CC 00    LDX   #$CC00
FD03: 10 8E B3 EA LDY   #hs_inits
FD07: A6 80       LDA   ,X+
FD09: A7 A0       STA   ,Y+
FD0B: 8C D0 00    CMPX  #$D000
FD0E: 26 F7       BNE   $FD07
FD10: C6 06       LDB   #$06
FD12: 1A 3F       ORCC  #$3F
FD14: DE 85       LDU   $85
FD16: 10 9E 84    LDY   $84
FD19: 8E CC 00    LDX   #$CC00
FD1C: BD D0 39    JSR   $D039               ; JMP $D6CD - get a
random number into A
FD1F: A7 80       STA   ,X+
FD21: 86 39       LDA   #$39
FD23: B7 CB FF    STA   watchdog
FD26: 8C D0 00    CMPX  #$D000
FD29: 26 F1       BNE   $FD1C
FD2B: 10 9F 84    STY   $84
FD2E: DF 85       STU   $85
FD30: 8E CC 00    LDX   #$CC00
FD33: BD D0 39    JSR   $D039               ; JMP $D6CD - get a
random number into A
FD36: A8 80       EORA  ,X+
FD38: 84 0F       ANDA  #$0F
FD3A: 26 24       BNE   $FD60
FD3C: 86 39       LDA   #$39
FD3E: B7 CB FF    STA   watchdog
FD41: 8C D0 00    CMPX  #$D000
FD44: 26 ED       BNE   $FD33
FD46: 5A          DECB
FD47: 26 CB       BNE   $FD14
FD49: 8D 03       BSR   $FD4E
FD4B: 1C FE       ANDCC #$FE                ; clear carry flag
FD4D: 39          RTS

FD4E: CE B3 EA    LDU   #hs_inits
FD51: 10 8E CC 00 LDY   #$CC00
FD55: A6 C0       LDA   ,U+
FD57: A7 A0       STA   ,Y+
FD59: 10 8C D0 00 CMPY  #$D000
FD5D: 26 F6       BNE   $FD55
FD5F: 39          RTS
```

```
FD60: 8D EC       BSR   $FD4E
FD62: 1A 01       ORCC  #$01
FD64: 39          RTS

RAM_TEST:
FD65: 1A 3F       ORCC  #$3F           ;a=# passes (FF=wait 'til
adv)
FD67: 7F C9 00    CLR   rom_enable_scr_ctrl
FD6A: 1F 8B       TFR   A,DP
FD6C: 1F 10       TFR   X,D
FD6E: 1F 03       TFR   D,U
FD70: 8E 00 00    LDX   #$0000
FD73: 53          COMB
FD74: C5 09       BITB  #$09
FD76: 26 05       BNE   $FD7D
FD78: 53          COMB
FD79: 46          RORA
FD7A: 56          RORB
FD7B: 20 0B       BRA   $FD88

FD7D: 53          COMB
FD7E: C5 09       BITB  #$09
FD80: 26 04       BNE   $FD86
FD82: 46          RORA
FD83: 56          RORB
FD84: 20 02       BRA   $FD88

FD86: 44          LSRA
FD87: 56          RORB
FD88: ED 81       STD   ,X++
FD8A: 1E 10       EXG   X,D
FD8C: 5D          TSTB
FD8D: 26 15       BNE   $FDA4
FD8F: C6 39       LDB   #$39
FD91: F7 CB FF    STB   watchdog
FD94: 1F B9       TFR   DP,B
FD96: C1 FF       CMPB  #$FF
FD98: 26 09       BNE   $FDA3
FD9A: F6 C8 0C    LDB   rom_pia_dataa
FD9D: C5 02       BITB  #$02
FD9F: 27 02       BEQ   $FDA3
FDA1: 6E A4       JMP   ,Y

FDA3: 5F          CLRB
FDA4: 1E 10       EXG   X,D
FDA6: 8C C0 00    CMPX  #color_registers
FDA9: 26 C8       BNE   $FD73
FDAB: 1F 30       TFR   U,D
FDAD: 8E 00 00    LDX   #$0000
FDB0: 53          COMB
FDB1: C5 09       BITB  #$09
FDB3: 26 05       BNE   $FDBA
FDB5: 53          COMB
FDB6: 46          RORA
```

```
FDB7: 56          RORB
FDB8: 20 0B       BRA   $FDC5

FDBA: 53          COMB
FDBB: C5 09       BITB  #$09
FDBD: 26 04       BNE   $FDC3
FDBF: 46          RORA
FDC0: 56          RORB
FDC1: 20 02       BRA   $FDC5

FDC3: 44          LSRA
FDC4: 56          RORB
FDC5: 10 A3 81    CMPD  ,X++
FDC8: 26 43       BNE   $FE0D
FDCA: 1E 10       EXG   X,D
FDCC: 5D          TSTB
FDCD: 26 15       BNE   $FDE4
FDCF: C6 39       LDB   #$39
FDD1: F7 CB FF    STB   watchdog
FDD4: 1F B9       TFR   DP,B
FDD6: C1 FF       CMPB  #$FF
FDD8: 26 09       BNE   $FDE3
FDDA: F6 C8 0C    LDB   rom_pia_dataa
FDDD: C5 02       BITB  #$02
FDDF: 27 02       BEQ   $FDE3
FDE1: 6E A4       JMP   ,Y

FDE3: 5F          CLRB
FDE4: 1E 10       EXG   X,D
FDE6: 8C C0 00    CMPX  #color_registers
FDE9: 26 C5       BNE   $FDB0
FDEB: 1F 03       TFR   D,U
FDED: 1F B8       TFR   DP,A
FDEF: 81 FF       CMPA  #$FF
FDF1: 26 05       BNE   $FDF8
FDF3: 1F 30       TFR   U,D
FDF5: 7E FD 70    JMP   $FD70

FDF8: 4A          DECA
FDF9: 1F 8B       TFR   A,DP
FDFB: 81 80       CMPA  #$80
FDFD: 27 07       BEQ   $FE06
FDFF: 4D          TSTA
FE00: 1F 30       TFR   U,D
FE02: 10 26 FF 6A LBNE  $FD70
FE06: C6 01       LDB   #$01
FE08: F7 C9 00    STB   rom_enable_scr_ctrl
FE0B: 6E A4       JMP   ,Y

FE0D: 30 1E       LEAX  $FFFE,X
FE0F: A8 84       EORA  ,X
FE11: E8 01       EORB  $0001,X
FE13: 4D          TSTA
FE14: 26 07       BNE   $FE1D
```

```
FE16: 5D          TSTB
FE17: 26 04       BNE   $FE1D
FE19: 30 02       LEAX  $0002,X
FE1B: 20 AD       BRA   $FDCA

FE1D: CE 00 30    LDU   #$0030
FE20: 1E 10       EXG   X,D
FE22: 5F          CLRB
FE23: 1E 10       EXG   X,D
FE25: 8C 00 00    CMPX  #$0000
FE28: 27 12       BEQ   $FE3C
FE2A: 30 89 FF 00 LEAX  $FF00,X
FE2E: 33 C8 10    LEAU  $10,U
FE31: 11 83 00 30 CMPU  #$0030
FE35: 23 EE       BLS   $FE25
FE37: CE 00 10    LDU   #$0010
FE3A: 20 E9       BRA   $FE25

FE3C: 33 41       LEAU  $0001,U
FE3E: 47          ASRA
FE3F: 25 05       BCS   $FE46
FE41: 57          ASRB
FE42: 25 02       BCS   $FE46
FE44: 20 F6       BRA   $FE3C

FE46: 1F 30       TFR   U,D
FE48: 86 01       LDA   #$01
FE4A: B7 C9 00    STA   rom_enable_scr_ctrl
FE4D: 10 CE FE 53 LDS   #$FE53
FE51: 20 40       BRA   $FE93

FE53: 86 98       LDA   #$98          ;set direct page $9800
FE55: 1F 8B       TFR   A,DP
FE57: 1F A8       TFR   CC,A
FE59: 43          COMA
FE5A: 85 C0       BITA  #$C0
FE5C: 27 04       BEQ   $FE62
FE5E: 86 0B       LDA   #$0B
FE60: 20 02       BRA   $FE64

FE62: 86 02       LDA   #$02
FE64: 10 CE BF 70 LDS   #stacktop
FE68: BD D0 12    JSR   CLR_SCREEN1
FE6B: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT
FE6E: 1F A8       TFR   CC,A
FE70: 85 40       BITA  #$40
FE72: 26 03       BNE   $FE77
FE74: 7E FA 71    JMP   $FA71

FE77: 10 8E D0 00 LDY   #$D000
FE7B: 20 00       BRA   $FE7D

FE7D: 86 20       LDA   #$20
FE7F: 8E 58 00    LDX   #$5800
```

```
FE82: 30 1F       LEAX  $FFFF,X
FE84: C6 39       LDB   #$39
FE86: F7 CB FF    STB   watchdog
FE89: 8C 00 00    CMPX  #$0000
FE8C: 26 F4       BNE   $FE82
FE8E: 4A          DECA
FE8F: 26 EE       BNE   $FE7F
FE91: 6E A4       JMP   ,Y


;
; B = number
;

FE93: 1F 03       TFR   D,U
FE95: 86 02       LDA   #$02
FE97: 1F 8B       TFR   A,DP
FE99: 1F 30       TFR   U,D
FE9B: 10 8E FE A1 LDY   #$FEA1
FE9F: 20 7C       BRA   $FF1D

FEA1: 86 02       LDA   #$02
FEA3: 10 8E FE A9 LDY   #$FEA9
FEA7: 20 D6       BRA   $FE7F

FEA9: 10 8E FE AF LDY   #$FEAF
FEAD: 20 5E       BRA   $FF0D

FEAF: 86 01       LDA   #$01
FEB1: 10 8E FE B7 LDY   #$FEB7
FEB5: 20 C8       BRA   $FE7F

FEB7: 1F 30       TFR   U,D
FEB9: 1F 98       TFR   B,A
FEBB: 44          LSRA
FEBC: 44          LSRA
FEBD: 44          LSRA
FEBE: 44          LSRA
FEBF: 10 8E FE C5 LDY   #$FEC5
FEC3: 20 58       BRA   $FF1D

FEC5: 86 02       LDA   #$02
FEC7: 10 8E FE CD LDY   #$FECD
FECB: 20 B2       BRA   $FE7F

FECD: 10 8E FE D3 LDY   #$FED3
FED1: 20 3A       BRA   $FF0D

FED3: 86 01       LDA   #$01
FED5: 10 8E FE DB LDY   #$FEDB
FED9: 20 A4       BRA   $FE7F

FEDB: 1F 30       TFR   U,D
FEDD: 1F 98       TFR   B,A
```

```
FEDF: 10 8E FE E5 LDY   #$FEE5
FEE3: 20 38       BRA   $FF1D

FEE5: 86 02       LDA   #$02
FEE7: 10 8E FE ED LDY   #$FEED
FEEB: 20 92       BRA   $FE7F

FEED: 10 8E FE F3 LDY   #$FEF3
FEF1: 20 1A       BRA   $FF0D

FEF3: 86 05       LDA   #$05
FEF5: 10 8E FE FC LDY   #$FEFC
FEF9: 7E FE 7F    JMP   $FE7F

FEFC: 1F B8       TFR   DP,A
FEFE: 4A          DECA
FEFF: 1F 8B       TFR   A,DP
FF01: 26 96       BNE   $FE99
FF03: 10 8E FF 09 LDY   #$FF09
FF07: 20 04       BRA   $FF0D

FF09: 1F 30       TFR   U,D
FF0B: 6E E4       JMP   ,S

FF0D: 86 3C       LDA   #$3C
FF0F: B7 C8 0D    STA   rom_pia_ctrla
FF12: 4C          INCA
FF13: B7 C8 0F    STA   rom_pia_ctrlb
FF16: 86 C0       LDA   #$C0
FF18: B7 C8 0E    STA   rom_pia_datab
FF1B: 6E A4       JMP   ,Y

;
; c80e rom_pia_datab
;   97            bits 0-5 = 6 bits to sound board
;   98            bits 6-7 plus CA2 and CB2 = 4 bits to drive the
LED 7 segment
;

FF1D: 1F 89       TFR   A,B
FF1F: 46          RORA
FF20: 46          RORA
FF21: 46          RORA
FF22: 84 C0       ANDA  #$C0
FF24: B7 C8 0E    STA   rom_pia_datab       ; set LEDS to show an
error.
FF27: 86 34       LDA   #$34
FF29: C5 04       BITB  #$04
FF2B: 27 02       BEQ   $FF2F
FF2D: 86 3C       LDA   #$3C
FF2F: B7 C8 0F    STA   rom_pia_ctrlb
FF32: 86 34       LDA   #$34
FF34: C5 08       BITB  #$08
FF36: 27 02       BEQ   $FF3A
```

```
FF38: 86 3C       LDA   #$3C
FF3A: B7 C8 0D    STA   rom_pia_ctrla
FF3D: 6E A4       JMP   ,Y


;
; Y = pointer to function to call if checksums are OK
;

CHK_ROM_CHKSUMS:
FF3F: 1A 3F       ORCC  #$3F                      ; set all flags
except fast interrupt and E. Interrupts are disabled
FF41: 8E FF B5    LDX   #rom_checksums            ; set X to be
pointer to first rom checksum in table
FF44: 8C FF D5    CMPX  #$FFD5                    ; have we passed
the last checksum?
FF47: 27 6A       BEQ   $FFB3                     ; yes, all
checksums are good, so goto $FFB3, which is a jump to Y to say all
OK
FF49: A6 01       LDA   $0001,X
FF4B: 27 18       BEQ   $FF65
FF4D: A6 84       LDA   ,X                        ; read page number
from checksum table
FF4F: 5F          CLRB
FF50: 1F 03       TFR   D,U                       ; U now is a
pointer to the very start of the required memory page
FF52: 86 39       LDA   #$39
FF54: EB C0       ADDB  ,U+                       ; B = B + byte
read from memory. We are calculating the checksum with this value.
FF56: B7 CB FF    STA   watchdog                  ; keep watchdog
happy
FF59: 1E 03       EXG   D,U
FF5B: A1 02       CMPA  $0002,X                   ; have we hit the
start of the next page??
FF5D: 1E 03       EXG   D,U
FF5F: 26 F3       BNE   $FF54                     ; no, so goto
$FF54
FF61: E1 01       CMPB  $0001,X                   ; does our
checksum match what was expected?
FF63: 26 04       BNE   $FF69                     ; no, goto $FF69,
rom error
FF65: 30 02       LEAX  $0002,X                   ; otherwise,
checksum matches, X+= 2, X now points to next checksum
FF67: 20 DB       BRA   $FF44                     ; go see if we've
done the last checksum yet

;
; if we get here, then a checksum doesn't match
;
FF69: A6 84       LDA   ,X                        ; get the page
that has the checksum failure
FF6B: 44          LSRA
FF6C: 44          LSRA
FF6D: 44          LSRA
```

```
FF6E: 44          LSRA                          ; divide it by 16
FF6F: 81 0D       CMPA  #$0D
FF71: 25 02       BCS   $FF75
FF73: 80 04       SUBA  #$04
FF75: 8B 01       ADDA  #$01                    ; add 1 so its
nonzero (should have done an INCA here...)
FF77: 19          DAA                           ; transform A into
a BCD number
FF78: 1F 89       TFR   A,B
FF7A: 86 02       LDA   #$02
FF7C: 10 CE FF 83 LDS   #$FF83
FF80: 7E FE 93    JMP   $FE93

FF83: 86 98       LDA   #$98           ;set direct page $9800
FF85: 1F 8B       TFR   A,DP
FF87: 86 39       LDA   #$39
FF89: B7 CB FF    STA   watchdog
FF8C: 10 CE BF 70 LDS   #stacktop
FF90: BD D0 12    JSR   CLR_SCREEN1
FF93: 1F A8       TFR   CC,A
FF95: 43          COMA
FF96: 85 C0       BITA  #$C0
FF98: 27 04       BEQ   $FF9E
FF9A: 86 0A       LDA   #$0A
FF9C: 20 02       BRA   $FFA0

FF9E: 86 03       LDA   #$03
FFA0: BD 5F 99    JSR   JMP_PRINT_STRING_LARGE_FONT  ; display ROM
ERROR [n]
FFA3: 1F A9       TFR   CC,B
FFA5: C5 40       BITB  #$40
FFA7: 26 03       BNE   $FFAC
FFA9: 7E FA 76    JMP   $FA76

FFAC: 10 8E D0 00 LDY   #$D000
FFB0: 7E FE 7D    JMP   $FE7D

FFB3: 6E A4       JMP   ,Y


;
; ROM checksum table.
;
; First byte = page number in memory
; Second byte = checksum

rom_checksums:
FFB5: 00 73                            ;ROM checksums (page/
checksum)
FFB7: 10 EA
FFB9: 20 1A
FFBB: 30 6C
FFBD: 40 B3
FFBF: 50 23
```

```
FFC1: 60 A3
FFC3: 70 3B
FFC5: 80 63
FFC7: 90 00
FFC9: A0 00
FFCB: B0 00
FFCD: C0 00
FFCF: D0 5C
FFD1: E0 82
FFD3: F0 01

FFD5: 00 3D

FFD7: (C)1982 WILLIAMS ELEC.INC

FFF0: F0 00


*** CPU vectors
FFF2: F0 00                             ;SWI3 vector

FFF4: F0 00                             ;SWI2 vector

FFF6: F0 00                             ;FIRQ vector

FFF8: DC 56                             ;IRQ vector

FFFA: F0 00                             ;SWI vector

FFFC: F0 00                             ;NMI vector

FFFE: F0 00                             ;RESET vector
```