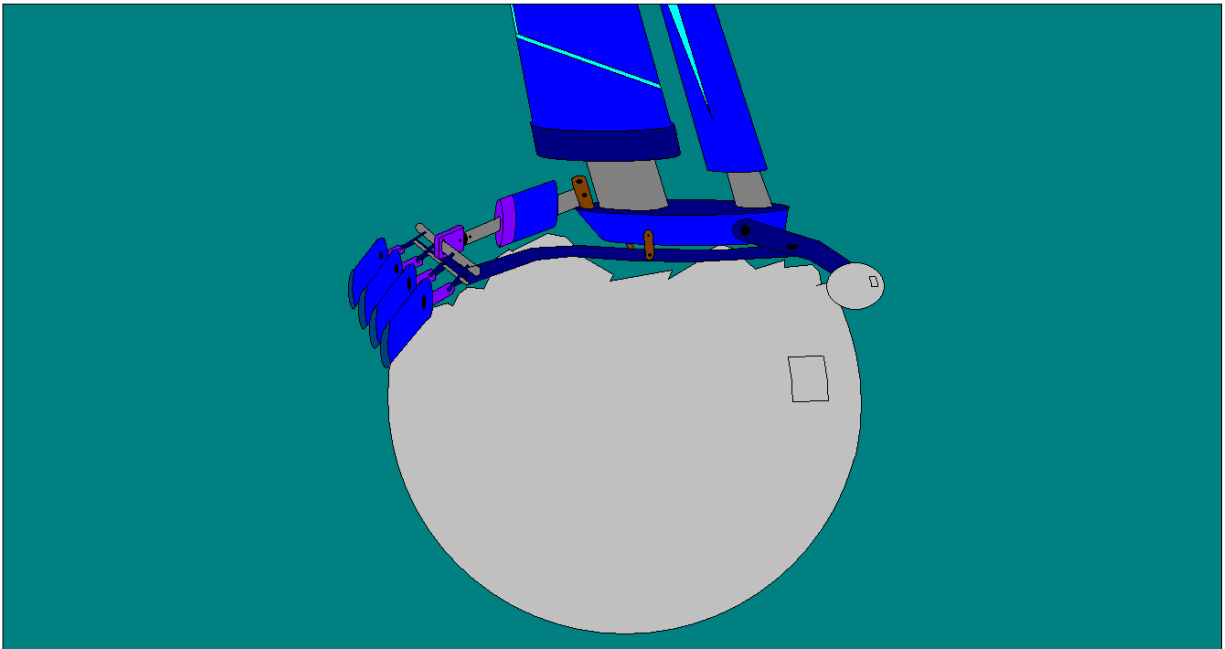


# Terminator 2 ROM L8.3



## ROM L8.3 Manual Includes

Change Log

Technical Details

Appendix

ROM Image Changes

## Terminator 2 L8.3

History/Summary of Releases:

Revision	Date	Checksum	Info
L-8	Dec. 15, 1992	BE08	Official release
L8.1	April 9, 2011	7608	Small change to have the attract mode “boom boom” sound only when Feature Adjustment A2.16 “Attract Sounds” is set to “ON”.
L8.2	April 1, 2012	6F08	This is 8.1 plus changes to attract mode sequence to have previously played game scores shown more often especially at game-over. This image also deletes the “T2 Fan Club”.
L8.3	June 28, 2022	7308	Selectable 8.1, 8.2, 8.3 attract mode. L8.3 attract mode playing “I am a cybernetic organism” more often. Profanity ROM logic. Custom ROM embedded attract mode message. Bug fixes for: attract mode, DMD animation flicker, multiball ball-lock issues, German text, ball-search. Selectable original or corrected DMD animation logic. Selectable original and LED lamp driver. Selectable drop-target state at multiball start. Selectable 3-bank lamp behavior.

### L8.3 Beta ROM Image History

Revision	Date	Checksum	Info
L8.3	April 15, 2022	3E08	Selectable 8.1, 8.2, 8.3 attract mode. L8.3 attract mode playing “I am a cybernetic organism” more often. Profanity ROM logic. Custom ROM embedded attract mode message. Bug fixes for attract mode and for DMD animation flicker.
L8.3	April 28, 2022	4508	DMD Animation flicker ISR routine was reverted to original L-8 so that certain 3 <sup>rd</sup> party color display, which was designed for original L-8, will not flicker.
L8.3	May 8, 2022	BB08	Added Lamp Driver adjustment to select between original and LED lamp driver code.
L8.3	May 21, 2022	A008	Added the “MB START DT ACTN” adjustment to select how drop target behaves at start of multiball.
L8.3	May 25, 2022	7508	Image with update to sound test.
L8.3	June 7, 2022	FF08	Experimental change to sound test and initial fix w/adjustment for the 3-bank target lamp behavior.
L8.3	June 22, 2022	7708	Removed experimental sound test changes. Added multiball fixes.
L8.3	June 27, 2022	7608	Ball-search bug fix.
L8.3	June 28, 2022	7308	Minor correction to German text. Final. Release image.

*Beta ROM info provided only for historical accounting. Images not intended for public release.*

## L8.3 Change Log

- Added a “L8.3” attract mode. Same as 8.2 but plays “I am a cybernetic organism” call more often.
- Supports T2 “Profanity ROM” w/FUA database award in place of 100,000 (w/adjustment).
- Added selectable attract mode, 8.1, 8.2, 8.3. *The 8.1 mode will not include T2 Fan Club message.*
- Fixed bug in display animations that were being shown with unintended flickering (w/adjustment).
- Support of the anti-ghosting lamp matrix code patch to prevent LED ghosting (w/adjustment).
- Improved anti-ghosting patch code to prevent controlled-lamp flicker during “GI Power Saver” mode.
- Added Feature Adjustment 22 to allow enable/disable of profanity mode.
- Added Feature Adjustment 23 to allow selection of the attract mode, L8,1, L8,2 or L8.3.
- Added Feature Adjustment 24 to allow selection of original or corrected DMD animation logic.
- Added Feature Adjustment 25 to allow selection of original or anti-ghost LED lamp driver code.
- Added Feature Adjustment 26 to allow selection of drop-target state at multiball start.
- Added Feature Adjustment 27 to allow selection of timed 3-bank lamp state at end-of-ball.
- Fixed bug in WPC custom message where it was showing “Testing...” message prior to display.
- Fixed bug in Game-Over L8.1 attract mode where the previous game scores were not displaying.
- Added support for a custom embedded attract mode message w/ 3-line selectable font & placement.
- Corrected spelling errors in the German text strings.
- Fixed multiball bug where multiball state was being exited while multiple balls remained on playfield.
- Fixed multiball bug where ball lock sometimes reported “Jackpot Multiplied 0x=0”.
- Fixed multiball bug when final “Load the gun” period was sometimes skipped at end of multiball.
- Fixed ball-search bug, drop-target knock-down was accumulating points and played speech.
- Fixed ball-search bug, drop-target was getting kicked up when adjustment A.2 20 was set to “On”.
- Fixed Sound Test T.7 05 (Database) during “repeat” doesn’t restart the music every few seconds.
  - *Some 3<sup>rd</sup> party sound boards classify 05 as non-music & continue playing 05 during “running” test.*

## L8.3 Changes to L-8 ROM Image, Summary Table

Table is shown below for quick reference of changes done for each L8 ROM image. Details on the L8.3 changes are provided in the next section of this document.

ROM Image	Original T2 Fan Club Attract Mode Message	Attract Mode L8.1 (original L-8 sequence)	Attract Mode L8.2 (scores shown more often)	Attract Mode L8.3 ('Cyber netic' sounds more often)	Selectable Attract Mode L8.1, L8.2, L8.3	Bug Fix: "Boom Boom" attract sounds	Bug Fix: WPC Custom Message "Testing"	Bug Fix: Game-over Last scores display	Bug Fix: Animation flicker problem With selectable original/fixed mode	Bug Fix: Multiball ball-lock issues	Bug Fix: Ball-search drop-target issues	Selectable Lamp Driver Original / LED	Selectable 3-bank lamp bugfix behavior	Corrected German Text	Updated Sound Test Code for sound 05	Profanity (FUA) option	Custom ROM embedded custom message in attract mode
L-8	✓	✓															
L8.1	✓	✓				✓											
L8.2			✓			✓											
L8.3		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Note: With L8.3, regardless of "Attract Mode" selection, the "T2 Fan Club" message will never be shown. Regardless of the "Attract Mode" selection, the custom ROM embedded message, if present, will always be shown, periodically, during attract mode.

## Document Revision History

Revision 1.0, March 2022, Initial draft.

Revision 1.1, April 2022, Cleaned up L8.3 revision statements. Moved sections to make document consistent in ordering of the L8.3 update items. Added DMD Animation flicker bug fix.

Revision 1.2, April 2022, Indicated beta L8.3 rom images used for testing purposes.

Revision 1.3, August 2022, Added remaining sections describing L8.3.

## Table of Contents

Terminator 2 L8.3.....	2
L8.3 Beta ROM Image History.....	2
L8.3 Change Log.....	3
L8.3 Changes to L-8 ROM Image, Summary Table.....	4
Document Revision History.....	4
L8.3 Changes to L-8 ROM Image, Technical Details.....	9
The L8.3 Fan-Club Adjustment Replacement.....	11
The L8.2/L8.3 changes to attract mode.....	12
The L8.3 “Cybernetic Organism” and “LastGameScores” changes.....	14
The L8.1 “boom boom” bug fix.....	15
The L8.3 “Testing...” WPC Custom Message bug fix.....	16
The L8.3 “Last-Game-Scores” game-over attract mode bug fix.....	20
The “LastGameScores” Missing Display (L-8, L8.1 ROM images).....	20
The “LastGameScores” Incorrect Display (L-8, L8.1, L8.2 ROM images).....	21
The L8.3 “Last-Game-Scores” game-over attract mode code fixes.....	23
The L8.3 FUA code inclusion.....	25
The L8.3 Display Animation flicker bug fix.....	26
The L8.3 Display Animation flicker problem analysis.....	27
The L8.3 Display Animation flicker affected animations.....	30
The \$EE91 Interrupt Service Routine.....	32
The Getaway L-5 Interrupt Service Routine.....	34
Analysis of T2 memory \$B4 for new logic.....	36
Corrections to Animation Init Code and ISR Code for L8.3.....	37
Initial Fix in L8.3 Beta.....	38
Final Fix in L8.3 Release Image.....	38
The L8.3 Display Animation Analysis and Fixes.....	41
Animation Fix: Security Levels.....	45
Animation Fix: T-1000 Self-Healed/Extra Ball Award.....	46
The L8.3 Custom Message support in the ROM.....	48
The L8.3 Custom Message Assembly Code.....	49
The L8.3 Custom Message Adjustable Values Table.....	50

The L8.3 Custom Message Adjustable Values Info .....	51
Line Text .....	51
Line Font.....	52
Line X-Position.....	53
Line Y-Position.....	53
Sound Call .....	54
Reveal/Wipe.....	61
Display Period .....	65
Fonts.....	65
Font 0x00 .....	67
Font 0x01 .....	67
Font 0x03 .....	68
Font 0x04 .....	68
Font 0x05 .....	68
Font 0x06 .....	68
Font 0x07 .....	68
Font 0x08 .....	68
Font 0x09 .....	69
Font 0x0A .....	69
Font 0x0B .....	70
Font 0x0C .....	71
Font 0x0F.....	71
Font 0x10 .....	72
Font 0x11 .....	72
Font 0x12 .....	72
Font 0x13 .....	72
Font 0x14 .....	73
Font 0x15 .....	73
Font 0x16 .....	73
Font 0x17 .....	73
Font 0x18 .....	73
Font 0x1A .....	74

The L8.3 Feature Adjustments Additions.....	75
Determining Total Number of Adjustments .....	75
The L-8 Adjustments Memory Map .....	76
Feature Adjustments Metadata Table .....	79
Feature Adjustment: Profanity .....	81
Feature Adjustment: Attract Mode .....	81
Feature Adjustment: Remaining New Adjustments .....	83
Feature Adjustments String Tables.....	83
The L8.3 Text String Corrections .....	87
The L8.3 Sound Test Updates.....	89
Sound Test Update: Sound 05 Playing Unexpectedly .....	89
Sound 05 Classification .....	89
Sound 05 Explicit Stop.....	91
FUA Inclusion Into Sound Test .....	93
Relocated Sound Test Table.....	93
Updated Sound Test Logic for FUA .....	95
The L8.3 Multiball Bug-Fixes .....	96
WPC Scheduled Functions and Function IDs .....	97
Scheduling a Function .....	97
Scheduled Function ID .....	97
Multiball Logic Overview.....	98
Multiball Startup Balls-In-Play Timing Problem .....	99
Multiball Startup Balls-In-Play Timing Fix: Startup waits for balls-in-play greater than 1.....	99
Multiball Startup Balls-In-Play Timing Fix: Maintenance function checks if startup is running .....	103
Multiball Startup Balls-In-Play Timing Fix: Corrected Multiball Logic.....	108
Multiball Switch Handler Logic Updates .....	109
Switch Handling, A brief overview .....	109
Switch Handling, Ball Lock Switch Handler .....	116
Switch Handling, Outhole Switch Handler .....	126
Multiball Corrected Logic.....	128
The L8.3 Lamp Driver Update .....	130
LED Patch Summary .....	130

LED Patch Improvement, power-saver improvement .....	130
Lamp Driver Code Modifications .....	133
Relocated Copyright Message .....	135
Lamp Driver Results .....	136
The L8.3 Multiball-Start Drop-Target Action Enhancement .....	137
Drop Target Down Multiball Adjustment .....	137
Multiball-Start Drop-Target Action Adjustment .....	137
Multiball-Start Drop-Target Action Adjustment Code .....	138
Multiball-Start Drop-Target Action Adjustment Code, Up/Down Functions .....	142
Multiball-Start Drop-Target Action Adjustment Code, Drop-Target Switch Handler .....	144
Multiball-Start Drop-Target Action Adjustment Code Analysis .....	148
Drop-Target Up Function ID 00 B9 Overlap .....	150
The L8.3 Timed 3-Bank Lamp Fixes .....	157
The L8.3 Ball-Search Bug Fixes .....	164
Appendix .....	169
Solenoid Table .....	169
Checksum Bytes .....	169
General Illumination and Zero Cross .....	170
Attract Mode Code .....	173
Fan Club Code .....	178
ROM Image Changes .....	179
L8.3 Test/Verification .....	191



## L8.3 Changes to L-8 ROM Image, Technical Details

For transparency and education to the pinball hobbyists interested in such information, the following sections provide technical details on the changes involved with the L8.3 image. Various parts of the L8.3 enhancements will be described, including:

- The L8.3 Fan-Club Adjustment Removal
- The L8.2/L8.3 changes to attract mode
- The L8.3 “Cybernetic Organism” and “LastGameScores” changes
- The L8.1 “boom boom” bug fix
- The L8.3 “Testing...” WPC Custom Message bug fix
- The L8.3 “Last-Game-Scores” game-over attract mode bug fix
- The L8.3 Display Animation flicker bug fix
- The L8.3 FUA code inclusion
- The L8.3 Custom Message support in the ROM
- The L8.3 Feature Adjustments additions
- The L8.3 Text String corrections
- The L8.3 Sound Test updates
- The L8.3 Multiball bug-fixes
- The L8.3 Lamp Driver update
- The L8.3 Multiball-Start Drop-Target Action enhancement
- The L8.3 Timed 3-Bank lamp fixes
- The L8.3 Ball-search bug fixes

For reference, the following page provides some basic addressing information on the T2-L8 ROM and is applicable to all WPC ROMs (although the bank/page information differs depending on size of the ROM image). When trying to understand the ROM modifications, it is important to understand the difference between an address offset within the ROM image as compared to an address that the running WPC code cites. To understand this requires some understanding of the paged ROM layout. *In this document the terms ‘bank’ and ‘page’ are used interchangeably.* Also note syntax for representing hexadecimal values, the use of the “\$” symbol and “0x” are used interchangeably.

When modifying strings in WPC code it is necessary to ensure the German and French strings are also updated. All of the changes for L8.3 take all three languages into consideration.

Note that replacing a L-8 or L8.1 or L8.2 with L8.3 should not cause the game to reset to factory settings. This is because the major version number (2<sup>nd</sup> byte of the checksum) ‘8’ is not changed and, as such, the game will not trigger factory reset of settings. The L8.3 image utilizes 6 new feature adjustments. Memory for these adjustments, while running L-8, L8.1 or L8.2 is set to 0 which corresponds to the initial value for each of the 6 new adjustments when upgrading to L8.3. **It is advised that the new adjustments are checked to ensure desired values are set after installing L8.3.**

ROM Image Size: 524288 bytes, ROM Address Range: 0x00000 - 0x7FFFF

ROM Address Range refers to addresses when looking at the ROM binary in a hex editor. In this case bytes are addressed from 0x00000 to 0x7FFFF.

Code Addresses refer to addresses that the actual running code cites when jumping to different functions or accessing fixed/constant data bytes. There is a *non-paged region* which is always accessible to running code when it references addresses in this range from 0x8000 through 0xFFFF. System startup code and commonly used functions are in this region. There is *paged ROM regions* which refer to code addressed from 0x4000 through 0x7FFF. When code wants to jump into a function in this range, it must specify the bank number along with the address in the range of 0x4000 through 0x7FFF. Code running in paged ROM may jump to other code or read ROM content within the same page (0x4000 through 0x7FFF) as well as access the non-paged region.

Bank/Page	Bytes	ROM Address Range	Code Addresses
Bank 0x20	16384	0x00000 - 0x03FFF	0x4000 - 0x7FFF
Bank 0x21	16384	0x04000 - 0x07FFF	0x4000 - 0x7FFF
Bank 0x22	16384	0x08000 - 0x0BFFF	0x4000 - 0x7FFF
Bank 0x23	16384	0x0C000 - 0x0FFFF	0x4000 - 0x7FFF
Bank 0x24	16384	0x10000 - 0x13FFF	0x4000 - 0x7FFF
Bank 0x25	16384	0x14000 - 0x17FFF	0x4000 - 0x7FFF
Bank 0x26	16384	0x18000 - 0x1BFFF	0x4000 - 0x7FFF
Bank 0x27	16384	0x1C000 - 0x1FFFF	0x4000 - 0x7FFF
Bank 0x28	16384	0x20000 - 0x23FFF	0x4000 - 0x7FFF
Bank 0x29	16384	0x24000 - 0x27FFF	0x4000 - 0x7FFF
Bank 0x2A	16384	0x28000 - 0x2BFFF	0x4000 - 0x7FFF
Bank 0x2B	16384	0x2C000 - 0x2FFFF	0x4000 - 0x7FFF
Bank 0x2C	16384	0x30000 - 0x33FFF	0x4000 - 0x7FFF
Bank 0x2D	16384	0x34000 - 0x37FFF	0x4000 - 0x7FFF
Bank 0x2E	16384	0x38000 - 0x3BFFF	0x4000 - 0x7FFF
Bank 0x2F	16384	0x3C000 - 0x3FFFF	0x4000 - 0x7FFF
Bank 0x30	16384	0x40000 - 0x43FFF	0x4000 - 0x7FFF
Bank 0x31	16384	0x44000 - 0x47FFF	0x4000 - 0x7FFF
Bank 0x32	16384	0x48000 - 0x4BFFF	0x4000 - 0x7FFF
Bank 0x33	16384	0x4C000 - 0x4FFFF	0x4000 - 0x7FFF
Bank 0x34	16384	0x50000 - 0x53FFF	0x4000 - 0x7FFF
Bank 0x35	16384	0x54000 - 0x57FFF	0x4000 - 0x7FFF
Bank 0x36	16384	0x58000 - 0x5BFFF	0x4000 - 0x7FFF
Bank 0x37	16384	0x5C000 - 0x5FFFF	0x4000 - 0x7FFF
Bank 0x38	16384	0x60000 - 0x63FFF	0x4000 - 0x7FFF
Bank 0x39	16384	0x64000 - 0x67FFF	0x4000 - 0x7FFF
Bank 0x3A	16384	0x68000 - 0x6BFFF	0x4000 - 0x7FFF
Bank 0x3B	16384	0x6C000 - 0x6FFFF	0x4000 - 0x7FFF
Bank 0x3C	16384	0x70000 - 0x73FFF	0x4000 - 0x7FFF
Bank 0x3D	16384	0x74000 - 0x77FFF	0x4000 - 0x7FFF
Non-paged	32768	0x78000 - 0x7FFFF	0x8000 - 0xFFFF

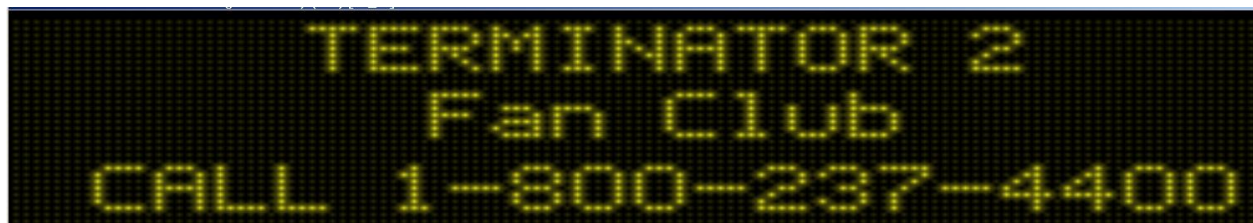
## The L8.3 Fan-Club Adjustment Replacement

The "T2 FAN CLUB" adjustment at A2.18 is removed starting in L8.2. The L8.2 attract mode replaced the calls for "T2 Fan Club" with calls to display other attract-mode sequences. The L8.3 ROM removes the fan-club code from the ROM image entirely and re-uses its code region for new code used in other parts of the L8.3 ROM image.

Shown below is how the A2.18 adjustment appears in ROM images L8.1 and older:



For reference, and perhaps as a final farewell to this rarely seen attract mode item, below is what the fan-club message looked like, when shown during attract mode. This was a message that would only play when enabled in the Feature Adjustments and when the system date is June 1992 or earlier. *Please note the phone number shown has most certainly been repurposed and should not be called.*



## The L8.2/L8.3 changes to attract mode

The L8.2 and L8.3 ROM modifications for attract mode are too numerous to list. Instead of depicting each individual ROM change, here we will describe how the attract mode works and the changes that are done to support the L8.2 and L8.3 attract mode sequences. The main T2 attract-mode loop code is in page \$30, starting at \$793F,30 (ROM offset 0x4393F).

The flowchart on the next page depicts the attract mode for ROM images:

- L8.1 (which is same sequence as L-8)
- L8.2 (which replaces a few of the L8.1 elements and inserts some additional elements)
- L8.3 (which plays 'cybernetic' sound more frequently & displays custom ROM message)

When running the L8.3 ROM image, the sequence for the attract mode depends on how the "Attract Mode" is configured in the Feature Adjustments menu:

- "L8.1", All *non-highlighted* items and all **blue** highlighted items
  - The "fanClub" item is shown for completeness as it was available in L-8 and the genuine L8.1 ROM image. It has been removed in the L8.3 ROM completely and, as such, has been shown but with ~~strikethrough~~ as it will not appear when using L8.3 ROM image.
- "L8.2", All *non-highlighted* items, all **green**, all **yellow** highlighted items
- "L8.3", All *non-highlighted* items, all **green**, all **yellow**, all **red** highlighted items

Regardless of the "Attract Mode" setting, if the 8.3 feature of adding a custom message to the ROM image is being utilized, the display of such message will take place wherever the **violet** highlighted item is shown.

The L8.3 attract mode (and added to L8.2, retroactively) also inhibits the report of "LastGameScores" during the "I am a cybernetic organism" speech sequence. This retains the original L-8 dramatic effect and alignment of the speech to the display during this period. This logic change is shown in **teal**.

In all modes, L8.1, L8.2 or L8.3, the L8.3 ROM image includes bug-fix for the **LastGameScores** that is shown during the Game-Over loop. The original L-8 code had a bug where it did not display scores the first time through. Subsequently, when it was shown immediately after CyborgComputerReadout, it may incur a flicker of the scores prior to revealing the display in a center-out reveal pattern.

The attract mode items listed on the flowchart each represent a certain sequence during the attract mode. Sometimes an item consists of a single panel message on the display (such as 'insert coin') and other times the item might consist of multiple panels of information (such as high-scores report). The T2 attract mode can be observed and followed along to match the flowchart.

In some cases multiple items are shown on a single line, using '/' character to separate them. In cases where L8.2 *replaced* an L8.1 item, a hyphen '-' is used to separate the L8.1 and new L8.2 items.

Power-Up

T2BashEffect

SpecialThanks / CustomRomMessage

ReplayAt  
 HighScores  
 FanClubMessage - CreditsInsertCoin / LastGameScores  
 CreditsInsertCoin - ReplayAt  
 T2ShineyLogo  
 StaringArnold / GameOver  
 LastGameScores  
 WilliamsLogoDraw  
 Presents  
 T2BashEffect  
 HighScores  
 CreditsInsertCoin  
 WilliamsLogoBlocky  
 Presents  
 JudgmentDay  
 StaringArnold / LastGameScores  
 HighScores  
 CreditsInsertCoin  
 PullTrigger  
 ReplayAt  
 \*CyberneticOrganism  
 TerminatorLightning  
 WilliamsLogoBlockyWipe  
 Presents  
 T2ShineyLogo  
 StaringArnold / LastGameScores, not during \*Cybernetic  
 \*CyberdyneSeries800  
 CyborgComputerReadout  
 CreditsInsertCoin  
 PullTrigger  
 ReplayAt  
 HighScores  
 CreditsInsertCoin  
 JudgmentDay  
 StaringArnold / LastGameScores  
 ArnoldShootingShotgun  
 SayNoToDrugs  
 CustomMessage / CustomRomMessage

Has above block played 2 times?

IAmTheFuture

Has above block played 5 times?

TimeDate / CastCredits

\* Enable sounds for CyberneticOrganism and CyberdyneSeries800 next pass through.

\* Enable sounds for CyberneticOrganism and CyberdyneSeries800 next pass through.

# Terminator 2 Attract Mode L-8.1, L8.2, L8.3

Only in L-8 and L8.1 blue highlighted item(s)  
 In L8.2, replaced L-8 & L8.1 items with green highlight  
 In L8.2, inserted new items with yellow highlight  
 In L8.3, inserted new logic with red highlight  
 In L8.3, Custom ROM Message gets shown, violet highlight  
 In L8.3, Fix for Game-Over LastGameScores display  
 In L8.3, Logic change applied to L8.2 and L8.3 modes

Game-Over

LastGameScores  
 HighScores  
 FanClubMessage - CreditsInsertCoin / LastGameScores  
 CastCredits - TerminatorLightning  
 SpecialThanks - CyborgComputerReadout  
 CustomRomMessage

LastGameScores  
 ReplayAt  
 GameOver  
 CreditsInsertCoin  
 HighScores  
 FanClubMessage - CreditsInsertCoin / LastGameScores  
 WilliamsBlockyLogo  
 Presents  
 T2BashEffect  
 StaringArnold / GameOver  
 LastGameScores  
 ReplayAt  
 HighScores  
 CreditsInsertCoin  
 PullTrigger  
 LastGameScores  
 ReplayAt  
 WilliamsLogoDraw  
 LastGameScores  
 HighScores  
 ArnoldShootingShotgun  
 SayNoToDrugs  
 CustomMessage / CustomRomMessage  
 CyborgComputerReadout

Has above block played 3 times?

\*The sounds for "CyberneticOrganism" only play after logic has passed through the block(s) indicated to the left to enable their play during the main attract loop. After these sounds are played during the main loop, logic must pass through the block(s) again before they play again.

## The L8.3 “Cybernetic Organism” and “LastGameScores” changes

A change is made to the attract mode L8.2 where “LastGameScores” had been added after the “StaringArnold” message. This change is highlighted in teal in the attract mode flowchart earlier in this document. Logic is updated so that when the “I am a cybernetic organism” speech is playing, the L8.2 addition of “LastGameScores” is not shown while Arnold speech is reporting his model number. This particular change is being retroactively applied to the L8.2 attract mode (when “L8.2” is selected in “Attract Mode” adjustment) as well as the “L8.3” attract mode because it provides a neater experience during L8.2 or L8.3 attract modes alike when the “I am a cybernetic organism” sequence plays with the intended, uninterrupted, sequence from L-8.

The original speech sequence is timed so that Arnold speech reporting his model number occurs in conjunction with the display showing computer readout with similar information. For L8.2/L8.3 attract modes, when the cybernetic speech is playing, the attract mode will not display the “LastGameScores” information. If attract sounds are “off” no speech is occurring so the original L8.2 sequence will occur with the “LastGameScores” being shown at such time.

Code changes are briefly depicted below. The original attract mode code related to this is in bank \$30 (ROM image offset 0x40000 through 0x43FFF), specifically at \$7A82 (ROM image offset 0x43A82):

```
7A7F: BD FB E2      JSR    $FBE2      ; AttractMode_StaringArnold()
7A82: 7E 7A 85      ; <null>
```

Note the instruction at \$7A82 is effectively a null, jump instruction to the very next instruction that occurs after. This is likely a placeholder or debug hook used in original L-8. In L8.2 this null instruction was replaced with a JSR instruction to a function that reports the “LastGameScores”. In L8.3 this is replaced with a JSR instruction to jump to new code at \$7F60.

```
7A7F: BD FB E2      JSR    $FBE2      ; AttractMode_StaringArnold()
7A82: BD 7F 60      JSR    $7F60      ; Fixup Function
```

At \$7F60 (ROM image offset 0x43F60) new code is added to previously unused ROM region:

```
7F60: BD 88 F5      JSR    $88F5      ; CallBankedFunction_Param_WPCAddr()
7F63: 64 23 3D      ; Sets C-bit when ok play LastGameScores
7F66: 25 77          BCS    $7FDF      ; Show LastGameScores if in L8.2 or L8.3 mode
7F68: 39            RTS              ; Return without showing LastGameScores
```

The new function, above, calls yet another function that sets C-bit if it is safe to show LastGameScores. Otherwise, if the “I am a cybernetic organism” is playing then C-bit is clear so function returns without trying to show “LastGameScores”. The function at \$6423 in bank \$3D (ROM image offset 0x74000 through 0x77FFF) is depicted below. Function is located at ROM image offset 0x76423:

```
6423: BD 84 AD      JSR    $84AD      ; GetMemoryFlag()
6426: D9            ; 0xD9 indicating time to play 'cybernetic'
6427: 25 0C          BCS    $6435      ; C-bit clear=no cyber, C-bit set=cyber
6429: BD 86 5B      JSR    $865B      ; LookupGameAdjustmentParameter1andCheckIfEqualsParam2()
```

```

                                ; C-bit set when not-equal
642C: 10 00                                ; 0x10 == Attract Sounds, C-set when not 0x00
642E: 24 03      BCC  $6433                ; If C-bit is clear, sounds are OFF
6430: 1C FE      ANDCC #$00FE              ; Clear C-bit
6432: 39         RTS                       ; Return C-clear, not okay for LastGameScores
6433: 1A 01      ORCC  #$0001              ; Set C-bit
6435: 39         RTS                       ; Return C-set, okay to show LastGameScores

```

Details of the above logic is left as an exercise to the reader.

## The L8.1 “boom boom” bug fix

This fix takes place in page \$35 (ROM region 0x54000 through 0x57FFF) which is where the “Boom Boom” attract mode sequence is handled. Below are the two places in the code where it makes the call to play the “Boom”. As there are 2 “booms” there are also 2 places where the sound call is made. For simplicity purposes only the call to play the sound is depicted below. These are at ROM image offsets 0x57723 and 0x577B7:

```

7723: BD 85 46      JSR  $8546                ; Play sound number 0x94, Boom!
7726: 94                                ;
<and>
77B7: BD 85 46      JSR  $8546                ; Play sound number 0x94, Boom!
77BA: 94                                ;

```

The original code simply calls a function in the non-paged region at \$8546 which plays the sound number that was provided in the byte immediately after the JSR instruction. In this case it is sound number 0x94. To fix this, we change the call from \$8546 to call a new function added in an unused area in this same bank at address \$7FC0,35 (ROM image offset 0x57FC0) as depicted below:

```

7723: BD 7F C0      JSR  $7FC0                ; Play sound number 0x94 if attract sounds are on, Boom!
7726: 94                                ;
<and>
77B7: BD 7F C0      JSR  $7FC0                ; Play sound number 0x94 if attract sounds are on, Boom!
77BA: 94                                ;

```

The new function added at \$7FC0,35 (ROM image offset 0x57FC0) performs the simple task of querying the game feature adjustments to determine if “attract sounds” are “on” and, if so, proceed to play the “boom” sound. If the “attract sounds” are “off” then the sound is not played. The actual code disassembly of this new function is left as an exercise for the reader.

## The L8.3 “Testing...” WPC Custom Message bug fix

During the creation of the L8.3 ROM image it was observed that there is a bug in the built-in WPC custom message feature. Since the L8.3 involves an enhancement involving a custom (ROM embedded) custom message, it seems appropriate to also fix the WPC custom message bug to complement the set of updates in L8.3 being a ROM update with enhanced ‘custom message’ support.

This bug is present in several WPC titles and is related to how the WPC custom message is scrolled onto the display to the left, revealing each frame of the WPC custom message in this way. This display mechanism is internally utilizing the same code and memory that is also used for the display of the “Test Report” which scrolls each test report message onto the display in this way (sliding to the left to reveal each test report frame).

It turns out the power-up “Testing...” message is also written to this same region of memory when it is first displayed. After power-up, as the attract mode is normally running, this “Testing...” message is actually retained in this region of memory unnecessarily. This region of memory is cleared out when entering and exiting the test menu and also cleared out at the end of the normal display of a WPC custom message (if one is set).

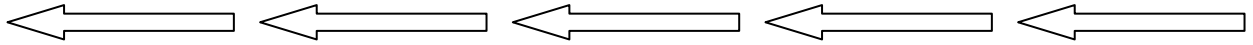
The bug is when the WPC custom message is shown *for the first time* after a system power-up (if menu system has not been entered/exited), this stale “Testing...” message is briefly shown and sliding to the left as the first frame of the WPC custom message is revealed onto the display. It doesn’t matter if a game is played or not, but the first display of the WPC custom message that occurs after a power-up (as long as the coin door ‘enter’ button hasn’t been pressed) will show the “Testing...” message as the WPC custom message is revealed for the first time. Subsequent displays of the WPC custom message won’t have this issue since the previous display of the WPC custom message left this region of RAM blank.

To demonstrate the bug, we create a single-frame WPC custom message with the following:



After saving this message, *restart the game* and wait for this message to appear (after the Arnold with shotgun and then “Say no to drugs” message).





```
TESTING. . . MY  
13 REV. L-8
```

```
NG. . . MY CUSTO  
REV. L-8
```

```
MY CUSTOM  
L-8 MES
```

```
MY CUSTOM  
MESSAGE
```

As shown above, the reveal of the WPC custom message briefly shows the “Testing...” message. A careful examination of the WPC code reveals that this is an oversight in how the WPC custom message uses the left-scrolling effect normally used by the ‘test report’ function. The WPC custom message code is making false assumption that the memory has previously been cleared as it reveals the first frame of the WPC custom message. This assumption is valid only after the test menu has previously been entered/exited and after the WPC custom message has previously been shown. However after a game power-up this is not the case.

To correct this problem, the attract-mode sequence now simply clears out the region of memory that is retaining this stale “Testing...” message so that subsequent display of WPC custom message will display without this problem.

The new function that clears this region of memory is located in page \$30 (ROM region 0x40000 through 0x43FFF) in a previously unused space near the end of this bank starting at \$7F79,30 (ROM image offset 0x43F79):

```

-----;-----
; -----
; L8.3 stale "Testing" message fix
; -----
7F79: 34 14      PSHS  X,B
7F7B: 8E 01 60   LDX  #$0160 ; 0x0160 is start of first panel
7F7E: C6 60      LDB  #$60   ; 0x60 = 96 bytes to clear.
7F80: 6F 80      CLR  ,X+
7F82: 5A         DECB
7F83: 26 FB      BNE  $7F80
7F85: 35 94      PULS  B,X,PC
-----;-----

```

The memory address \$0160 is where the stale “Testing...” message is retained. In this region is space to accommodate 16 characters for the top row and 16 characters for the bottom row of text. The left-scrolling code employs a method whereby each character takes up of 3 bytes in memory. One byte contains the letter to be displayed and the remaining 2 bytes contain values used during the scrolling effect (further study into the WPC code would be necessary to accurately report how these are used). Since there are 32 characters total and 3 bytes per character, there are 96 bytes to clear in total.

Technically speaking, the memory immediately after these cleared 96 bytes could also be cleared as they contain characters that are used in the left-scrolling logic (containing the next frame to reveal) however to simply fix the bug at hand, only the first 96 bytes are being cleared.

This clearing takes place at the start of attract mode (both at power-up and at game-over). The stale “Testing...” message is unconditionally cleared out of ram. In the event that a WPC custom message is to be displayed it will no longer show the “Testing...” message as it is revealed onto the display.

The attract mode in L-8 begins in this same page \$30 at \$793F,30 (ROM image offset 0x4393F) with the following code:

```

793F: BD FB AE      JSR  $FBAE ; Clear display data
7942: 7E 79 45      JMP  $7945 ; Jump to next instruction
7945: BD 84 AD      JSR  $84AD ; GetMemoryFlag()
7948: D3             ; 0xD3 game-over mode

```

As shown, the L-8 attract mode starts out with some housekeeping code to clear out the display pixel data and has a check if it is in game-over mode (or power-up mode) and proceeds from there. In these instructions is a dummy instruction that simply jumps to the next instruction. This jumps from \$7942 to \$7945. This may have been placeholder code for original s/w development for prototyping or debugging the code. This dummy instruction is being replaced with a call to the new function depicted above:

```
793F: BD FB AE      JSR   $FBAE      ; Clear display data
7942: BD 7F 79      JSR   $7F79      ; Call bugfix function
7945: BD 84 AD      JSR   $84AD      ; GetMemoryFlag()
7948: D3           ; 0xD3 game-over mode
```

This will now, at start of attract mode, call the new function that clears out the stale "Testing..." message and return back to \$7945 to proceed with normal attract mode code. With this fix in place, the first display of the WPC custom message now reveals the first frame without the stale "Testing..." message:



## The L8.3 “Last-Game-Scores” game-over attract mode bug fix

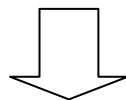
During the development of the T2 L8.3 image some additional bugs were noticed in the original T2 L-8 code. During the game-over attract mode loop, as depicted earlier in this document, there is a block of sequences that plays for 3 times before transitioning to the normal attract mode loop. This block begins with “LastGameScores” which is intended to display the scores from the previously played game. This refers to the **blue-highlighted** “LastGameScores” item in the attract-mode flowchart presented earlier in this document. Two problems were found regarding this “LastGameScores” display and are fixed in L8.3:

- After any game, *the first time* the “LastGameScores” is (attempted to be) displayed during the main Game-Over attract mode loop in L-8 or L8.1 it actually shows an empty/blank display. This is after the “SpecialThanks” message is displayed after Game-Over in L-8 and L8.1. There is a small period where the display is blank but the code intended to show the previous game scores.
- The subsequent 2 displays of the first “LastGameScores” at top of the game-over attract mode loop also have a minor problem. This refers to when the display of “LastGameScores” is done immediately after the “CyborgComputerReadout” when the game-over attract mode loop restarts 2 more times. Using an emulator it is evident that there is a very brief display of the screen with scores immediately followed by the intended center-out reveal of the display of the “LastGameScores” information. *It may be that this is only noticed in emulator and not real game.*

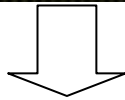
Both of these issues are fixed in L8.3 and will be corrected regardless of which attract mode is selected in the “Attract Mode” adjustment. The first time “LastGameScores” is (attempted to be) displayed, it will actually be shown, and subsequent times through the game-over attract mode loop, there will be no incorrect display of the “LastGameScores” screen as it is being revealed onto the display.

## The “LastGameScores” Missing Display (L-8, L8.1 ROM images)

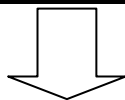
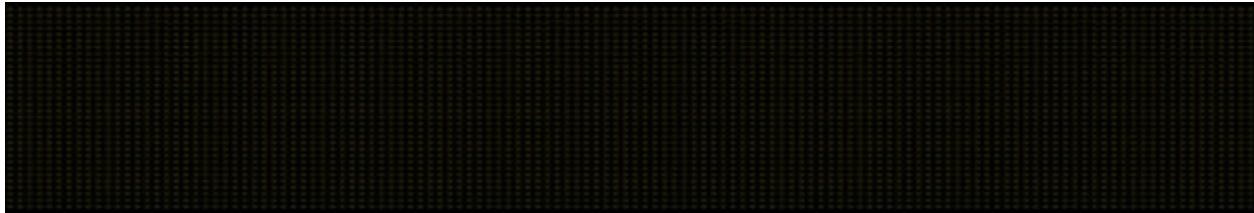
The missing display of “LastGameScores” during game-over attract mode sequence is depicted here. In L-8 and L8.1 at the start of game-over attract mode sequence, there is a lengthy display of “CastCredits” and a “SpecialThanks” messages. After the “SpecialThanks” message the code intended to display the scores from the most recently played game. What actually takes place is a period of time of blank display while the game is intending to display the “LastGameScores” sequence.







*Blank display shown for period of time that the "LastGameScores" is intended to be shown.*



*After the display period is over the next attract mode item is shown, the "ReplayAt" value.*

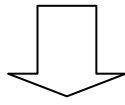


For the L8.3 ROM image, this bug is being fixed and will be observable when the "Attract Mode" adjustment is set to "L8.1". This problem does not happen for L8.2 or L8.3 attract-mode sequences because of how they display "CyborgComputerReadout" in place of the "SpecialThanks" sequence immediately prior to this display of "LastGameScores" which indirectly fixes the bug.

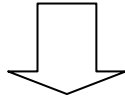
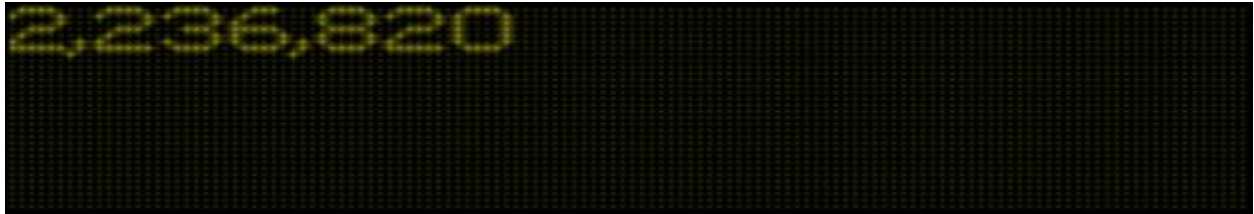
### **The "LastGameScores" Incorrect Display (L-8, L8.1, L8.2 ROM images)**

The problem with incorrect display of "LastGameScores" during the game-over attract mode sequence is depicted here. Since the game-over attract mode block is played 3 times, the transition from ending "CyborgComputerReadout" back to the top "LastGameScores" sequence will take place 2 times before the attract mode goes to the regular attract mode loop. For each of these 2 transitions, the display is as follows:

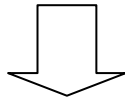
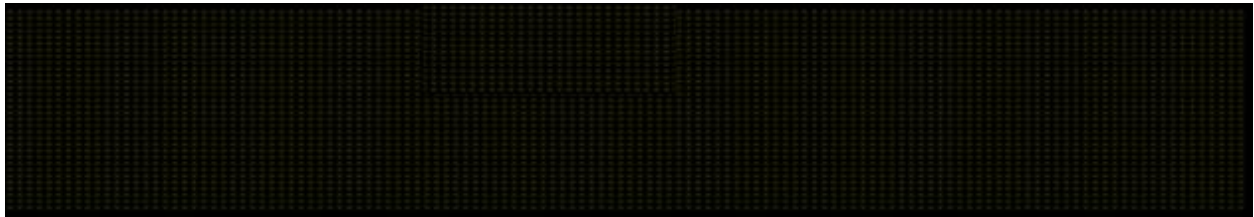




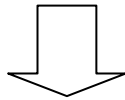
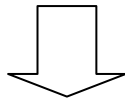
*Display of last game scores (not including "credits") is briefly shown briefly in dim*



*Display is then blanked*



*Normal center-out reveal then takes place*



The brief display of scores prior to the reveal **may not be noticeable on real display hardware**, however it is evident in an emulated environment. Out of completeness and to further improve the L8.3 software, this display issue is being corrected in the L8.3 image.

### The L8.3 “Last-Game-Scores” game-over attract mode code fixes

The game-over attract mode code is located in bank \$30 (ROM image offset 0x40000 through 0x43FFF). The location where the first “LastGameScores” is called is at \$7987,30 (ROM image offset 0x43987).

```
7987: BD 7B E1      JSR   $7BE1      ; AttractMode_LastGameScores()
```

As part of this fix, the function call is replaced with a call to a new function located later in this bank in previously unused region of ROM space:

```
7987: BD 7F 69      JSR   $7F69      ; Call new bugfix function at $7F69,30
```

Located in previously unused region near the end of bank \$30 at \$7F69,30 (ROM image offset 0x43F69) is a small function added as part of the bug fix:

```
7F69: BD 88 F5      JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
7F6C: 64 0F 3D      ; -->LastGameScores_DisplayClear()
7F6F: 7E 7B E1      JMP   $7BE1      ; Jump to AttractMode_LastGameScores()
```

The new function, above, calls a function in bank \$3D at \$640F,3D which performs fix for the brief display of scores prior to the reveal of the “LastGameScores”. Upon returning from that function, code jumps to the original “LastGameScores” function at \$7BE1,30. Using the JMP instruction in this way will cause the \$7BE1,30 function, when finished, to return back to the original starting point (returns to instruction after \$7987,30).

The fixup function at \$640F,3D corresponds to ROM image offset 0x7640F which at the start of the region of unused ROM space located in bank \$3D. This fixup function is as follows:

```
640F: 34 16      PSHS  X,B,A      ; If B is not 0x03 then we're not
6411: C1 03      CMPB  #$03      ; at start of game-over x3 loop
6413: 26 09      BNE   $641E     ; so we definitely want blank display
6415: 8E 7F 00   LDX   #$7F00    ; Address of string #1. Each str 0x20 bytes
6418: BD 7F B8   JSR   $7FB8     ; Call subroutine to set B if there is a msg
641B: 5D        TSTB             ; If B!=0x00 there is a msg to show
641C: 26 03      BNE   $6421     ; so we don't want to blank the display
641E: BD 7F D2   JSR   $7FD2     ; Branch to routine that blanks the display
6421: 35 96      PULS  A,B,X,PC  ;
```

The fixup routine, above, is crafted to check whether the Game-Over attract loop is at the very first pass and whether there was an L8.3 Custom ROM message to display. This also calls new code used by the Custom ROM message that clears out the display so that the unexpected display of scores doesn't take place prior to the center-out reveal. The fixup routine will skip the display-blanking if there was a custom ROM message to show since the center-out reveal of “LastGameScores” is shown neatly when it is replacing an L8.3 custom ROM message.

Lastly, there is a code-change that is needed to fix the “AttractMode\_LastGameScores()” function to correct the display of “LastGameScores” so that the display is not blank when first shown (on L-8 and L8.1). The original L-8 code for “LastGameScores” is located at \$7BE1,30 (ROM image offset 0x43BE1):

```
7BE1: BD D3 60      JSR   $D360
7BE4: BD 88 F5      JSR   $88F5
7BE7: 6A F4 3B
7BEA: BD 88 F5      JSR   $88F5
7BED: 7F 57 33
7BF0: BD 83 46      JSR   $8346
7BF3: 80
7BF4: 39           RTS
```

To fix the code for this function, the first JSR instruction is replaced with a jump to a new function as shown below:

```
7BE1: BD 7F 72      JSR   $7F72
7BE4: BD 88 F5      JSR   $88F5
7BE7: 6A F4 3B
7BEA: BD 88 F5      JSR   $88F5
7BED: 7F 57 33
7BF0: BD 83 46      JSR   $8346
7BF3: 80
7BF4: 39           RTS
```

A new function is added near the end of bank \$30 at \$7F72,30 (ROM image offset 0x43F72) as shown below:

```
7F72: BD D3 4C      JSR   $D34C      ; Load 0x3800 into $1799
7F75: BD D3 60      JSR   $D360      ; Clear DMD data at $1799 pointer, 0x3800
7F78: 39           RTS              ;
```

The new function simply calls a function \$D34C to set value 0x3800 into \$1799 and then calls the function \$D360 which was originally called at the start of the “LastGameScores” but replaced with the call to this new function. The loading of 0x3800 into \$1799 is the main fix for the blank display issue. This, along with the changes described above for the center-out reveal display fix both result in the correct display of “LastGameScores” on all versions of attract mode, L-8, L8.1, L8.2, L8.3.



## The L8.3 FUA code inclusion

The database award code for L8.3 is updated to include the FUA logic from previous software where FUA was (unofficially) supported. The database award code is located in bank \$33 (ROM image region 0x4C000 through 0x4CFFF) and is updated to include the FUA text during the Database Award and updated to perform the necessary sound call when FUA is the winning selection.

What is interesting about the Database Award sequence is that the *order* of the possible selections is not random. There is a fixed list of 16 possible awards. A random starting point is chosen within this list of 16 awards. The 8 awards from such starting position are displayed and a random selection is made from those 8. The list of possible awards is shown below. The FUA code simply replaces the "100,000" award with the FUA. The chances of winning FUA are identical to the chances of winning "100,000".

Addr.	Data Bytes	Index String
4C04:	4E 3B	; 0x0b "POSSIBLE SELECTIONS:"
4C06:	4E 50	; 0x0c "CHASE LOOP"
4C08:	4E 5B	; 0x0d "EXTRA BALL"
4C0A:	4E 66	; 0x0e "100,000"
4C0C:	4E 6F	; 0x0f "AUTOFIRE"
4C0E:	4E 78	; 0x10 "SECURITY PASS"
4C10:	4E 86	; 0x11 "LITE SPECIAL"
4C12:	4E 2D	; 0x12 "HURRY UP"
4C14:	4E 93	; 0x13 "1,000,000"
4C16:	4E 9F	; 0x14 "MULTIBALL"
4C18:	4E A9	; 0x15 "LITE EXTRA BALL"
4C1A:	4E B9	; 0x16 "500,000"
4C1C:	4E C2	; 0x17 "VIDEO MODE"
4C1E:	4E CD	; 0x18 "LITE KICKBACK"
4C20:	4E DB	; 0x19 "SPECIAL"
4C22:	4E E3	; 0x1a "LITE HURRY UP"
4C24:	4E F1	; 0x1b "3,000,000"

Below are a couple examples of the Database award where it is shown how the list of awards follows the list shown above. The code specifically adjusts the left column of awards to the left by a few pixels when FUA is enabled (when adjustment A2.18 is set to allow awarding FUA).

```
POSSIBLE SELECTIONS:
1,000,000          VIDEO MODE
MULTIBALL         LITE KICKBACK
LITE EXTRA BALL  SPECIAL
500,000          LITE HURRY UP
```

```
POSSIBLE SELECTIONS:
LITE HURRY UP     FUCK YOU ASSHOLE
1,000,000        AUTOFIRE
CHASE LOOP       SECURITY PASS
EXTRA BALL       LITE SPECIAL
```

## The L8.3 Display Animation flicker bug fix

During the L8.3 image development it was requested that the display animation 'flicker' problem of T2 to also be fixed. Some investigation into the nature of the problem ensued and the issue was found to be reproducible in an emulator environment and, as such, is something that could be fixed as part of the L8.3 image.

To demonstrate the problem, the "Pull trigger" animation is shown in a series of frames depicting the flicker effect in slow motion as pixel data pages are put onto the display. Notice the changes in pixel brightness as the frames progress.

To illustrate the speed of this flickering effect, the images captured below depict every transition of display intensity during the first few seconds of the "Pull trigger" animation sequence. This depicts all transitions up until the moment of the hand appearing in the sequence. During the entire sequence with hand pulling the trigger, the same flickering effect continues until the animation is complete. Shown below are 3 columns. Images are shown on the display starting at the top-left, then going down each column and resuming at the top of the next column to the right.





### The L8.3 Display Animation flicker problem analysis

The flickering appears to be due to the WPC code that interacts between the CPU board and the DMD display driver board. For a given still image in an animation, the WPC software is simply switching back and forth between 2 pages of medium and dim pixel data while the DMD display driver board expects to receive 3 page updates per frame (consisting of 2 updates of 'medium' pixel data and single update of 'dim' pixel data).

A little background in WPC 3-color image data is in order. To neatly display a 3-color image (or 4-color if you include black/off pixel color), the WPC software overlaps 2 planes of pixel data. One plane consists of dots that should be dim and the other plane consists of dots that can be medium or bright. If a dot is set in both planes it is to be displayed with bright intensity. If a dot is set in only the medium plane then it is medium intensity. If a dot is set only in the dim plane then it is dim intensity.

A brief example of this can be depicted below. With understanding that there are 8 bits in a byte and assumption that a single byte can represent 8 pixels in a horizontal line on the display, the table below depicts how the intensity can be set for a series of 8 pixels in a row, with 4 different examples:

Dim plane		Medium plane		Resulting Pixel Intensities							
Hex	Binary	Hex	Binary	0x80 Pixel 8	0x40 Pixel 7	0x20 Pixel 6	0x10 Pixel 5	0x08 Pixel 4	0x04 Pixel 3	0x02 Pixel 2	0x01 Pixel 1
0xF0	11110000	0xAA	10101010	Bright	Dim	Bright	Dim	Medium	Off	Medium	Off
0xAA	10101010	0xF0	11110000	Bright	Medium	Bright	Medium	Dim	Off	Dim	Off
0x01	00000001	0xFF	11111111	Medium	Medium	Medium	Medium	Medium	Medium	Medium	Bright
0xFF	11111111	0x80	10000000	Bright	Dim	Dim	Dim	Dim	Dim	Dim	Dim

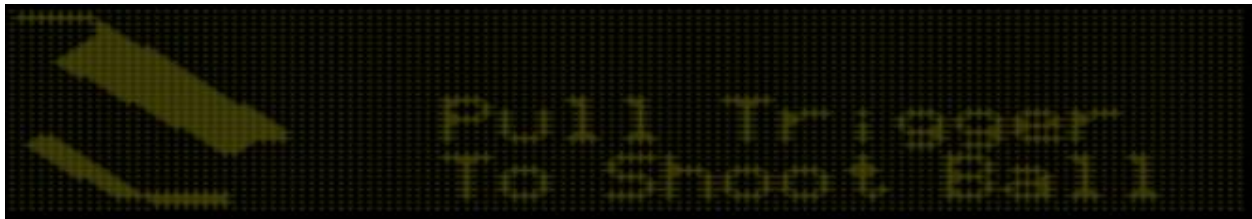
A survey of the L-8 WPC code and comparison with other WPC titles (most notably, "The Getaway" which also has an animation for ball-shooter control usage), it seems that the L-8 animation flicker problem boils down to the way in which the L-8 software notifies the DMD driver board about which of the pixel planes are to be used for dim/medium/bright.



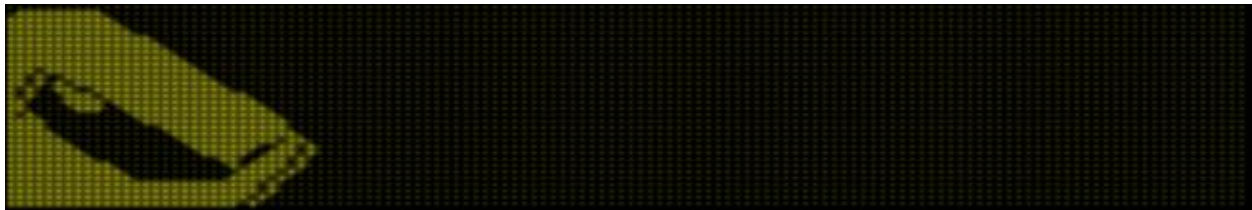
After loading the DMD memory with the 2 planes (or "pages") of pixel data, the WPC software then repeatedly directs the DMD driver board the brightness levels to use for each of these memory planes. Since the DMD driver board memory is already loaded with the necessary dim/medium pixel data, this issue involves tiny messages between the CPU and DMD driver board that effectively say "use page X for medium and page Y for dim". The L-8 code is simply cycling between these messages, repeatedly, during the display of these animation frames "use page X for medium and page Y for dim", over and over for the duration in which the image should appear on the display.

It seems, however, the DMD driver board doesn't expect to receive a **repeating pair** of messages "use page X for medium and page Y for dim". It appears the DMD driver board expects to receive a **repeating triplet** of messages whereby the CPU board should indicate the "use page X for medium" is sent 2 times while the "use page Y for dim" is sent one time per frame, repeatedly.

Let's explore how this might behave with some example image data. For the "Pull Trigger" animation sequence, we have 2 pages of pixel data that get built up and sent to DMD driver board (by using mapped memory), below is the DIM pixel data page:



Below is the data loaded into the MEDIUM pixel data page:



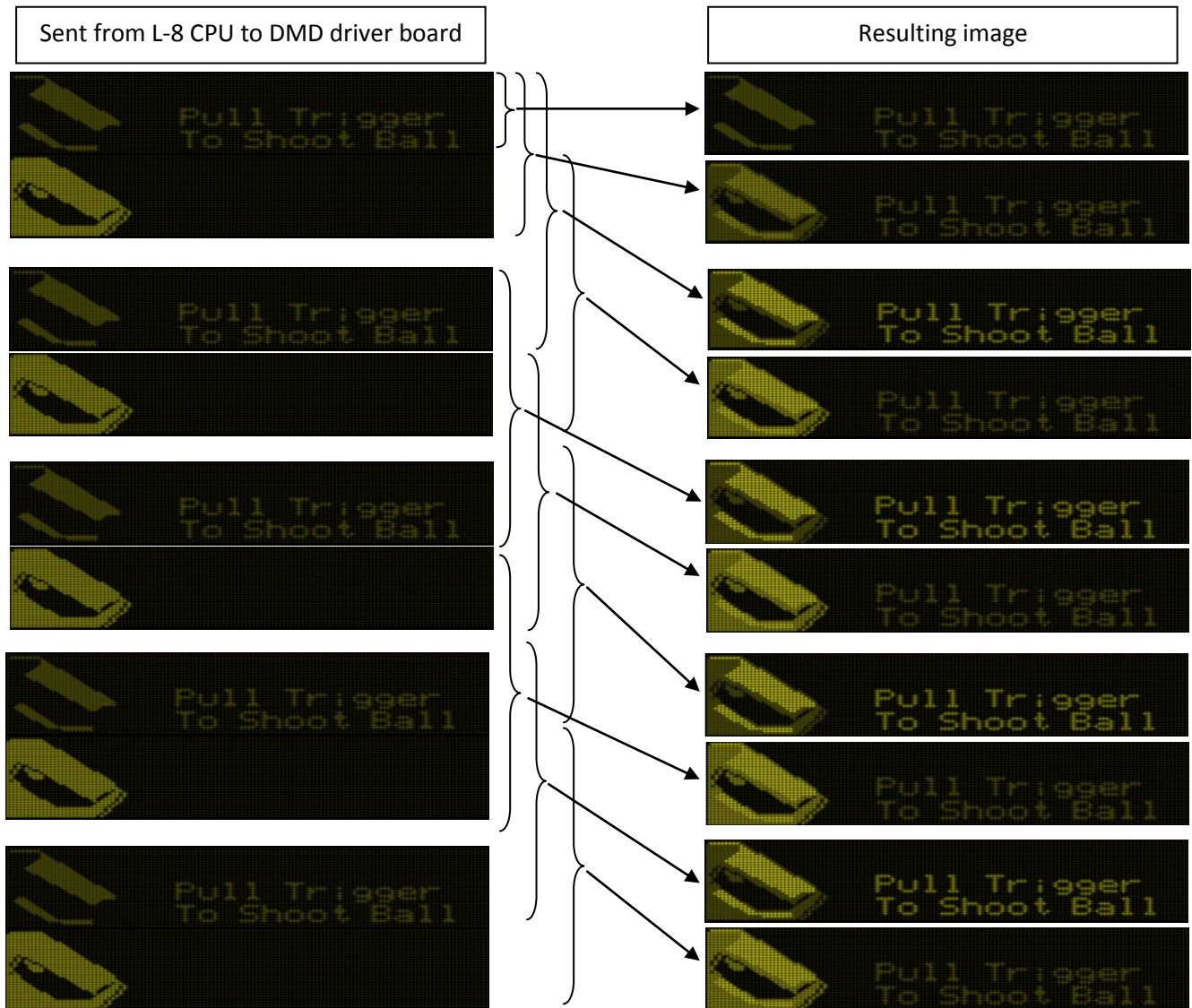
The intended/desired result is the blending of these pages as per previously described logic so the image is as follows:



You can see in the previous image that the pixels that are set in both DIM and MEDIUM planes appear brighter than the rest. The pixels that are only set in the MEDIUM plane are medium intensity and pixels that are only in the DIM are dim intensity in the blended image (i.e. the "Pull Trigger" text).

As depicted in previous display images, the actual result from L-8 is not the constant blended image but a varying intensity image as pixels are changing in intensity as their dim/medium/bright status is rapidly changing while the image is on the display.

Let's examine the mismatch between what the WPC software sends to the DMD driver board compared to what the DMD driver board actually expects. Utilizing the previously indicated DIM and MEDIUM planes example images we can see how the DMD board behaves.



The display images on the left, above, represents the L-8 software sending the DIM/MEDIUM repeatedly while the images on the right depict how the WPC DMD driver board interprets each message, expecting groups of 3. The displayed image represents the 3 most recent messages received from the CPU board. As depicted in these images, the resulting display seemingly flickers bright and medium as the most recent 3 messages refer to mostly MEDIUM to mostly DIM page indicators, respectively.

### **The L8.3 Display Animation flicker affected animations**








The problematic code has been identified and can be discovered by how it creates the animation sequences by way of the following:

- Calls function \$FB88 which initializes DMD related memory elements and establishes an interrupt service routine (ISR) at \$EE91 to repeatedly send messages to the DMD driver board for the duration of the animation sequence.
- Populates pixel data into DMD memory and updates memory values for the ISR to use while sending messages to the DMD driver board. Medium DMD ram page number stored in \$B5. Dim DMD ram page number stored in \$B6.
- Finishes by flagging memory that it is done so that the ISR knows to stop sending display-page messages to the DMD driver board. This involves writing a value other than 0x00 or 0xFF to memory location \$B4. *Not all animations sequences do this.*
- Clears out display memory in anticipation for what is drawn to memory next.

A survey of the L-8 ROM image reveals 19 locations where the \$FB88 initialization function is invoked. Each such invocation represents an animation display sequence that may be experiencing the flicker problem. Each of the 19 locations has been analyzed and described in the table below. Included in this table are memory values used during the animation (*more info described next*).



#	ROM Offset	WPC Addr	Sequence	\$B4 Values \$Addr:Value	\$B5 Medium \$B6 Dim	Example
1	0x100A3	\$40A3, 24	Doors Opening revealing award text based on value of the B register.	\$FB8C:0x00	0x0F 0x10	
2	0x101F9	\$41F9, 24	Hunter ship	\$FB8C:0x00	0x01,0x03 0x02,0x04	
3	0x10294	\$4294, 24	"Fire at will" crosshairs	\$FB8C:0x00	0x05,0x01,0x03 0x06,0x02,0x04	
4	0x10426	\$4426, 24	Pull Trigger to shoot ball	\$FB8C:0x00 \$44AA,24:0x55	0x01,0x03 0x02,0x04	
5	0x4F880	\$7880, 33	Jackpot	\$FB8C 0x00 \$788C,33:0x55	0x01 0x02	
6	0x4FA49	\$7A49, 33	Replay	\$FB8C 0x00	0x01 0x02	
7	0x4FA73	\$7A73, 33	Special	\$FB8C 0x00	0x01 0x03,0x04,0x05	
8	0x57392	\$7392, 35	Shoot Again	\$FB8C 0x00	0x01 0x02	
9	0x57740	\$7740, 35	"Boom" 1	\$FB8C 0x00 \$7759,35:0x55	0x01,0x03 0x02,0x04	
10	0x577B1	\$77B1, 35	"Boom" 2	\$FB8C 0x00 \$77C2,35:0x55	0x05 0x06	
11	0x5787C	\$787C, 35	Judgment day	\$FB8C 0x00 \$7888,35:0x55	0x01 0x02	
12	0x578AA	\$78AA, 35	T2	\$FB8C 0x00 \$78E7,35:0x55	0x01 0x02	
13	0x57990	\$7990, 35	Kickback Lit  This function is also used by bonus multiplier.	\$FB8C 0x00	0x01,0x03,0x05 0x02,0x04,0x06	

						
14	0x57D94	\$7D94, 35	"I am the future"	\$FB8C 0x00 \$7DC6,35:0x55	0x01,0x03 0x02,0x04	
15	0x57DF1	\$7DF1, 35	Kickback	\$FB8C 0x00 \$7E32,35:0x55	0x01,0x03 0x02,0x04	
16	0x57E5B	\$7E5B, 35	Arnold with shotgun	\$FB8C 0x00 \$7EA3,35:0x55	0x01,0x03 0x02,0x04	
17	0x57F54	\$7F54, 35	Elevator doors closed	\$FB8C 0x00	0x05 0x06	
18	0x57F6F	\$7F6F, 35	T-1000 blasted	\$FB8C 0x00	0x07 0x08	
19	0x57F98	\$7F98, 35	T-1000 self-healed	\$FB8C 0x00	0x05 0x06	

As shown in the table, above, all animations start with init code at \$FB8C setting \$B4 to 0x00. Not depicted is the fact that during the display of the animation \$B4 toggles from 0x00 to 0xFF, at each toggle, the ISR sends to the WPC driver board, the medium pixel page index number from \$B5 or the dim pixel page index from \$B6. Some animations involve code that sets \$B4 to 0x55 to notify the ISR that it is done. When \$B4 is set to a value other than 0x00 or 0xFF it essentially tells the ISR to stop updating the DMD driver board with dim/medium pixel plane index numbers.

### The \$EE91 Interrupt Service Routine

As mentioned, the problem animations utilize an interrupt route located in non-banked ROM region at address \$EE91 (ROM image offset 0x7EE91). Shown below is the assembly code along with some C-like comments describing the code logic.

```

EE91: 34 02      PSHS  A           ;
EE93: 86 FF      LDA   #$FF       ;
EE95: 98 B4      EORA  $B4       ;
EE97: 97 B4      STA   $B4       ; Flip bits in $B4
EE99: 27 08      BEQ   $EEA3     ; if ($B4 was 0xFF but we just changed it to 0x00)
; {
EE9B: 81 FF      CMPA  #$FF       ; if ($B4 was NOT 0x00, and not 0xFF)
; {
EE9D: 26 0C      BNE   $EEAB     ; goto $EEAB
; }
; // Here when $B4 was 0x00 and changed to 0xFF
EE9F: 96 B6      LDA   $B6       ; A gets dim pixel plane number from $B6
EEA1: 20 02      BRA   $EEA5     ; }
; else
; {
; // Here when $B4 was 0xFF and changed to 0x00
EEA3: 96 B5      LDA   $B5       ; A gets medium pixel plane number from $B5
; }

```



```

EEA5: B7 05 37    STA    $0537            ; Put selected dim/medium plane into $0537
EEA8: B7 3F BF    STA    $3FBF            ; Put selected dim/medium plane into $3FBF
EEAB: 86 04       LDA    #$04             ;
EEAD: B7 3F BD    STA    $3FBD            ; Put 0x04 into $3FBD
EEB0: 35 02       PULS   A                ;
EEB2: 3B         RTI                    ;

```

This interrupt routine is called regularly during WPC software runtime. The 68B09 CPU has an interrupt vector table at the last 16 bytes of the ROM. At location IRQ vector \$FFF8 (ROM offset 0x7FFF8) is the address 0xD9C0 which is the start of the main interrupt service routine which eventually calls this \$EE91 DMD Driver board update interrupt routine (This \$EE91 address is loaded into \$0A:\$0B to notify the main ISR to call this function as part of its work). This gets called periodically as the IRQ input line into the CPU (at pin 3) toggles. Further details into the nature of the IRQ itself are left as an exercise to the reader. *A quick look at the WPC schematics reveals that the IRQ comes in from the ASIC which likely toggles the IRQ line at a regular, fixed, rate.*

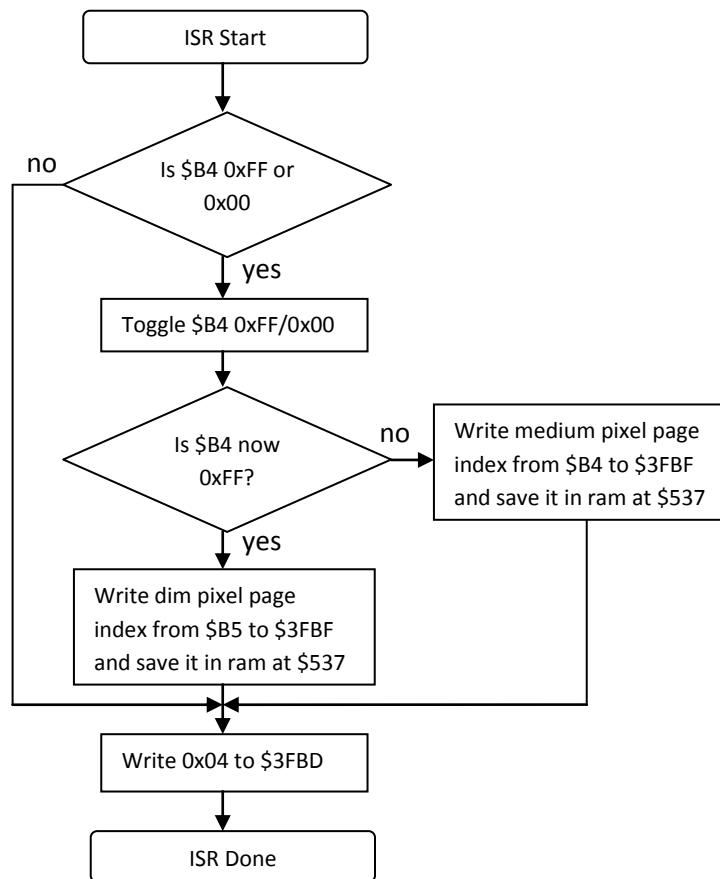
Below is the initial setup function that each of the affected animations calls at the start of the animation sequence. *Some animations are done in such a way that each frame of the animation runs through this init code and interacts with the \$EE91 distinctly for each frame, such as 'Fire at will'.*

```

FB88: 34 12       PSHS   X,A              ;
FB8A: 34 02       PSHS   A                ;
FB8C: 0F B4       CLR    $B4              ; Clear $B4
FB8E: 8E EE 91    LDX    #$EE91          ;
FB91: 9F 0A       STX    $0A              ; Put 0xEE91 into $0A:$0B to tell ISR to call it
FB93: 86 04       LDA    #$04             ;
FB95: B7 3F BD    STA    $3FBD            ; Put 0x04 into $3FBD DMD driver board register
FB98: 35 02       PULS   A                ;
FB9A: 97 B5       STA    $B5              ; Store medium pixel page number into $B5
FB9C: 4C         INCA                    ;
FB9D: 97 B6       STA    $B6              ; Store dim pixel page number into $B6
FB9F: 4A         DECA                    ;
FBA0: 35 92       PULS   A,X,PC          ;

```

As can be seen in the init code, above, the previously mentioned memory locations are cited, \$B4, \$B5 and \$B6. *The \$B4 value will be given extra scrutiny since its usage will be altered to effectively fix the flicker problem.*



Depicted above is a summary flowchart of the \$EE91 interrupt function in how it interacts with the DMD driver board using values from memory. As shown, the \$B4 value alternates between 0xFF and 0x00 to determine whether to push the medium or the dim pixel page index to the WPC display driver board.

### The Getaway L-5 Interrupt Service Routine

To get an idea on how to repair the T2 L-8 \$EE91 code, another WPC title was used as a basis to fix the T2 code. The Getaway title has some similarities with T2 especially in the fact that it also contains an animation depicting how to use the shifter lever to shoot the ball comparable to the T2 “Pull Trigger” animation. The comparable routine from The Getaway (L-5) is as follows:

```

EB4A: 34 02      PSHS  A          ;
EB4C: 0A D1      DEC   $D1        ; Decrement $D1, cycles from $D4 val to 0x01
                                     ; $D1 goes from 0x03 to 0x02, load med pixels
                                     ; $D1 goes from 0x02 to 0x01, load med pixels
                                     ; $D1 goes from 0x01 to 0x03, load dim pixels
                                     ;
EB4E: 2E 08      BGT   $EB58      ; If $D1 decrements from 0x01 to 0x00 then
EB50: 96 D4      LDA   $D4          ; {
EB52: 97 D1      STA   $D1          ;   Reset $D1 back to $D4 restart value
EB54: 96 D3      LDA   $D3          ;   A gets $D3 (dim pixel page index)
EB56: 20 02      BRA   $EB5A      ; }
  
```

```

; else
EB58: 96 D2      LDA  $D2      ; A gets $D2 (medium pixel page index)
EB5A: B7 05 41   STA  $0541    ; Store A into $0541
EB5D: B7 3F BF   STA  $3FBF    ; Store A into $3FBF
EB60: 86 04      LDA  #$04     ;
EB62: B7 3F BD   STA  $3FBD    ; Store 0x04 into $3FBD
EB65: 35 02      PULS A        ;
EB67: 3B         RTI         ;

```

The init code from The Getaway which is called at start of animations sequences or at start of each frame (as what T2 L-8 does for some animation sequences):

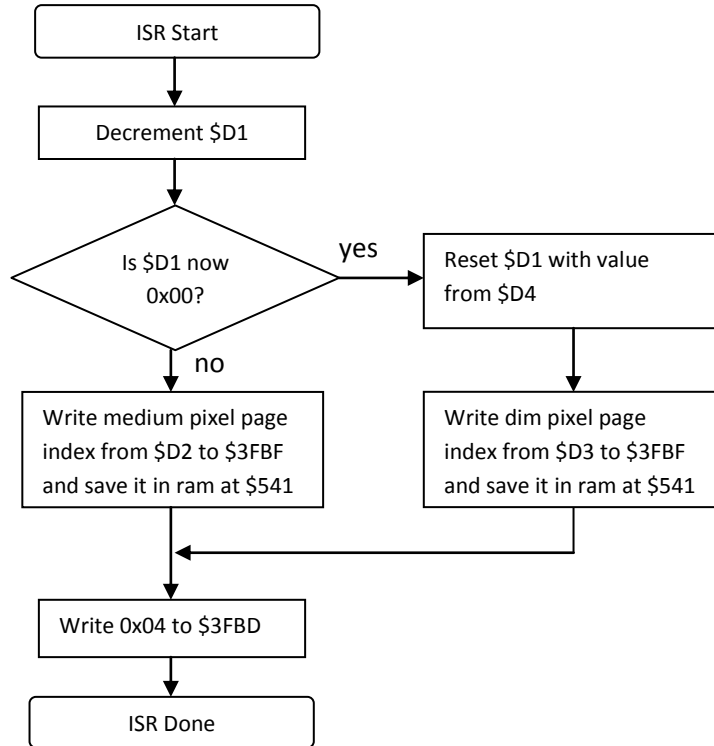
```

EB68: 34 07      PSHS B,A,CC   ;
EB6A: C6 02      LDB  #$02     ; Load B with restart value: 0x02
EB6C: 20 04      BRA  $EB72    ;
;
EB6E: 34 07      PSHS B,A,CC   ;
EB70: C6 03      LDB  #$03     ; Load B with restart value: 0x03
EB72: D7 D4      STB  $D4     ; $D4 gets restart value
EB74: 1A F0      ORCC #$00F0   ;
EB76: 97 D2      STA  $D2     ; $D2 gets medium pixel data page index
EB78: 4C         INCA         ;
EB79: 97 D3      STA  $D3     ; $D3 gets dim pixel data page index
EB7B: 86 03      LDA  #$03     ;
EB7D: 91 D1      CMPA $D1     ; Compare $D1 with 0x03.
EB7F: 22 02      BHI  $EB83   ; If $D1 has 0x01, branch down to $EB83
; If $D1 has 0x02, branch down to $EB83
; If $D1 >= 0x03, no branch, set it to 0x03
;
EB81: 97 D1      STA  $D1     ; $D1 gets 0x03
EB83: CC EB 4A   LDD  #$EB4A   ;
EB86: DD 0A      STD  $0A     ; $0A:$0B gets 0xEB4A
EB88: 86 04      LDA  #$04     ;
EB8A: B7 3F BD   STA  $3FBD    ; $3FBD gets 0x04
EB8D: 35 87      PULS CC,A,B,PC ;

```

The Getaway logic, above reveals how the DMD Driver board will be loaded with 3 page indexes in a repeating loop. When initiated with the call to \$EB6E, the restart value 0x03 is loaded to \$D4 and, as the routine runs, the memory \$D1 is decremented from 0x03 to 0x02 to 0x01 and back to 0x03 (getting the 0x03 restart value from \$D4). As the \$D1 is decremented to 0x02 and 0x01 it pushes the medium pixel page index from \$D2 to the DMD driver board. When \$D1 is decremented to 0x00 it pushes the dim pixel page index from \$D3 to the DMD driver board.

An interesting observation is that The Getaway also has some support for the T2 L-8 method whereby the init route being called at \$EB68 will establish a reset value of 0x02 into \$D4 which would effectively result in the T2 L-8 flickering behavior however this appears to be unused in The Getaway. Searching The Getaway ROM image reveals zero hits for a JSR instruction to the \$EB68 while there are many such JSR instructions to the \$EB6E init address. The corresponding flowchart for The Getaway is as follows:



The Getaway logic is fairly straightforward and will be used as the basis of the T2 L-8 fix for the same part of the code. Some analysis is needed to ensure the use of T2 L-8 \$B4 can be adapted to this model.

### Analysis of T2 memory \$B4 for new logic

The preceding text paints the picture for updating the T2 DMD ISR routine at \$EE91 so that it behaves similar to what The Getaway does in its comparable function. It seems that the \$B4 memory should simply decrement from 0x03 to 0x02 to 0x01 to 0x00 and reset back to 0x03. The medium/medium/dim triplet could then be sent for each frame to match what it appears the DMD driver board expects.

The way in which T2 L-8 animation code pushes a non-0x00/non-0xFF byte into \$B4 to stop the routine from pushing page indexes to the driver board can also be leveraged. Since it appears the L-8 code typically pushes a large value such as 0x55 into \$B4 the new code can simply check for values greater than 0x03 and assume that such a value means the existing L-8 code invoked its code to disable the interrupt routine from pushing page data into the WPC Driver board, and react accordingly.

The whole T2 L-8 ROM was evaluated for how it interacts with memory location \$B4 to ensure this fix will be consistent and predictable. For example if some animation sequence elected to write value 0x01 to the \$B4 (as a non-0x00/non-0xFF value) then it would conflict with the fix where \$B4 cycles from 0x03 to 0x01. A survey of the T2 L-8 ROM was performed for all places where \$B4 is written and summarized in the following table.

ROM Image Offset	WPC Addr	Instruction(s)	Opcode Bytes	Usage
<b>0x102B9</b> <b>0x102BB</b>	\$42B9, 24 \$42BB, 24	LDA #\$55 STA \$B4	0x86 0x55 0x97 0xB4	At end of "Pull trigger" animation preceding "Fire at will" in some circumstances hits this
<b>0x104A8</b> <b>0x104AA</b>	\$44A8, 24 \$44AA, 24	LDA #\$55 STA \$B4	0x86 0x55 0x97 0xB4	At end of "Pull trigger" animation preceding "Fire at will" in most circumstances hits this
<b>0x4F88A</b> <b>0x4F88C</b>	\$788A, 33 \$788C, 33	LDA #\$55 STA \$B4	0x86 0x55 0x97 0xB4	At end of "Jackpot" animation
<b>0x572B7</b> <b>0x572B9</b>	\$72B7, 35 \$72B9, 35	LDA #\$37 STA \$B4	0x86 0x37 0x97 0xB4	Uncertain. Appears to be around code that plays bonus multiplier and kickback animations.
<b>0x57757</b> <b>0x57759</b>	\$7757, 35 \$7759, 35	LDA #\$55 STA \$B4	0x86 0x55 0x97 0xB4	At end of first "Boom" animation in the "Boom Boom" sequence
<b>0x577C0</b> <b>0x577C2</b>	\$77C0, 35 \$77C2, 35	LDA #\$55 STA \$B4	0x86 0x55 0x97 0xB4	At end of second "Boom" animation in the "Boom Boom" sequence
<b>0x57886</b> <b>0x57888</b>	\$7886, 35 \$7888, 35	LDA #\$55 STA \$B4	0x86 0x55 0x97 0xB4	At end of "Judgement day" animation sequence
<b>0x578E5</b> <b>0x578E7</b>	\$78E5, 35 \$78E7, 35	LDA #\$55 STA \$B4	0x86 0x55 0x97 0xB4	At end of "T2" animation sequence
<b>0x57DC4</b> <b>0x57DC6</b>	\$7DC4, 35 \$7DC6, 35	LDA #\$55 STA \$B4	0x86 0x55 0x97 0xB4	At end of "I am the future" animation sequence
<b>0x57E30</b> <b>0x57E32</b>	\$7E30, 35 \$7E32, 35	LDA #\$55 STA \$B4	0x86 0x55 0x97 0xB4	At end of Kickback animation
<b>0x57EA1</b> <b>0x57EA3</b>	\$7EA1, 35 \$7EA3, 35	LDA #\$55 STA \$B4	0x86 0x55 0x97 0xB4	At end of Arnold with shotgun animation
<b>0x7EE93</b> <b>0x7EE95</b> <b>0x7EE97</b>	\$EE93 \$EE95 \$EE97	LDA #\$FF EORA \$B4 STA \$B4	0x86 0xFF 0x98 0xB4 0x97 0xB4	This is the original L-8 \$EE91 ISR routine where it flips the bits in \$B4
<b>0x7FB8C</b>	\$FB8C	CLR \$B4	0x0F 0xB4	The \$EE91 ISR Init clearing of \$B4

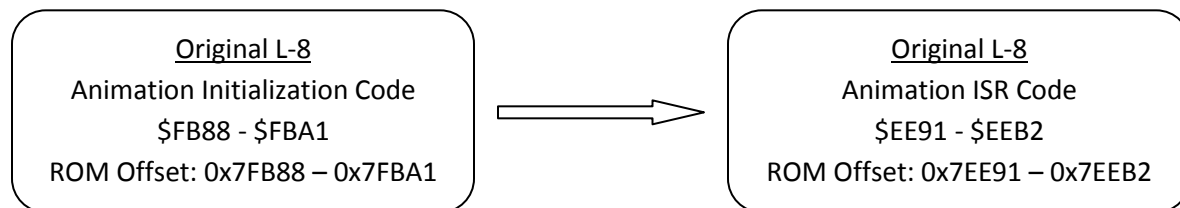
Other methods of modifying \$B4 have been searched in the L-8 image such as:

- STB \$B4, 0xD7 0xB4, 15 occurrences in the ROM but as an address used in JSR instruction.
- COM \$B4, 0x03 0xB4, 2 occurrences in the ROM but as data bytes, not executable code.

There are other indirect methods in which code could modify \$B4 however it seems fairly conclusive that the use of \$B4 memory location is limited to these animations listed in preceding text and used only in the ways described above. The non-0x00/non-0xFF byte values used to stop the ISR are 0x55 and, in one location, 0x37.

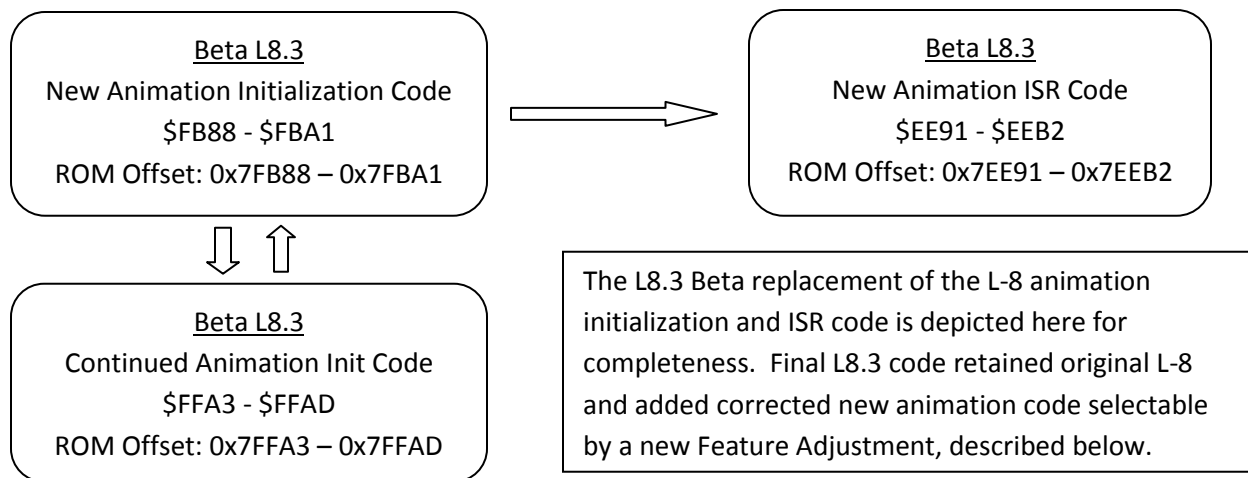
### Corrections to Animation Init Code and ISR Code for L8.3

As mentioned, the affected animations consist of two parts of code:



### Initial Fix in L8.3 Beta

For posterity, it is worth noting the original L8.3 code in beta completely replaced the original initialization and ISR code with corrected animation code based on The Getaway. The new initialization code was larger than the old so some unused ROM space (in the copyright message text) was repurposed as code to complete the initialization code.



### Final Fix in L8.3 Release Image

Preliminary testing of the L8.3 Beta “new and improved” animation code revealed that some 3<sup>rd</sup> party DMD displays were designed to overcome the flicker problem from the original L-8 software. Such displays were designed to receive the 2-updates per-frame method that causes flickering on original gas-plasma display panels. Due to this finding, it was deemed necessary to have a new Feature Adjustment that allows selection between original animation code or corrected animation code.



The need to have such an adjustment is what led to the introduction of new adjustments for L8.3 and the variety of new feature adjustments. Prior to this, the early L8.3 beta releases had the Fan Club adjustment repurposed to allow selection of Attract Mode and Profanity mode. With the L8.3 utilizing new Feature Adjustments, the Fan Club adjustment was simply left disabled as it was in L8.2.

The resulting set of code changes allow the original L-8 animation initialization code to remain and the original animation ISR code to remain in place. A minor change in the initialization code was added to check the Feature Adjustment to have the animation sequence utilize original or new code.



```

; C-bit clear = "Original"
FF5A: 25 07      BCS  $FB88      ; If C-bit is set, skip down and do new "Corrected" ISR.
;
; We are doing "original" animation code
FF5C: 34 12      PSHS  X,A      ; Perform original instruction from $FB88
FF5E: 34 02      PSHS  A       ; Perform original instruction from $FB8A
FF60: 7E FB 8C   JMP  $FB8C      ; Go back to $FB8C and resume original animation code
;
; We are doing "corrected" animation code
FF63: 34 06      PSHS  B,A      ; We are doing "corrected" animation code
FF65: 97 B5      STA  $B5      ; Store medium pixel page number into $B5
FF67: 4C         INCA          ;
FF68: 97 B6      STA  $B6      ; Store dim pixel page number into $B6
;
; If $B4 is currently 0x00 we decrement it to 0xFF
FF6A: 0D B4      TST  $B4      ;
FF6C: 26 02      BNE  $FF70      ; so that code flow then inits it 0x03
FF6E: 0A B4      DEC  $B4      ; This lets 0x00 value get reset to 0x03, below
FF70: 86 03      LDA  #$03      ;
FF72: 91 B4      CMPA $B4      ; Compare $B4 current countdown value with 0x03
FF74: 22 02      BHI  $FF78      ; If $B4 >= 0x03, no branch, force it to 0x03
FF76: 97 B4      STA  $B4      ; $B4 gets 0x03
;
FF78: CC FF 84   LDD  #$FF84      ;
FF7B: DD 0A      STD  $0A      ; Put 0xEE91 into $0A:$0B to tell ISR to call it
FF7D: 86 04      LDA  #$04      ;
FF7F: B7 3F BD   STA  $3FBD      ; Put 0x04 into $3FBD DMD driver board register
FF82: 35 86      PULS A,B,PC    ;

```

The new animation code, above, is performed when the new Feature Adjustment is set to “corrected”. The corrected animation initialization routine starting at \$FF63, above is modeled after The Getaway logic. Unlike The Getaway, it uses hard coded 0x03 restart value and doesn’t need to store it in a memory location. Code will always use 0x03 as the reset value. The function resets \$B4 to 0x03 only if it is 0x00 or anything greater or equal to 0x03. Otherwise, if it is 0x02 or 0x01, the \$B4 is unchanged. This allows the initialization function to seamlessly update certain animation sequences that, for each frame in the animation, invokes the init routine to start the next frame (“Fire at will” and extra ball award animations).

The new animation init logic establishes new function \$FF84 to handle the animation during the ISR. The new function is located immediately after the initialization code, using the portion of ROM that previously contained copyright message text.

```

FF84: 34 02      PSHS  A       ;
FF86: 96 B4      LDA  $B4      ; A gets the countdown byte value
FF88: 4A         DECA          ; Decrement the countdown by 1
FF89: 27 0A      BEQ  $FF95      ; If decremented to 0x00 go do dim pixel work
FF8B: 81 03      CMPA #$03      ; Compare to 03 (ie subtract 03 from A, C-clr if borrow)
FF8D: 24 0F      BCC  $FF9E      ; If C-clear then large value was put in $B4, no update
;
FF8F: 97 B4      STA  $B4      ; Save new decremented value 01 or 02 back into $B4
FF91: 96 B5      LDA  $B5      ; A gets medium pixel plane number from $B5
FF93: 20 06      BRA  $FF9B      ; Skip down to push medium plane number to $3FBF
;
FF95: 86 03      LDA  #$03      ; A gets reset value 0x03
FF97: 97 B4      STA  $B4      ; Save the reset value 03 back into $B4
FF99: 96 B6      LDA  $B6      ; A gets dim pixel plane number from $B6
;
FF9B: B7 3F BF   STA  $3FBF      ; Put selected dim/medium plane into $3FBF
FF9E: 86 04      LDA  #$04      ;

```



```

FFA0: B7 3F BD   STA   $3FBD           ; Put 0x04 into $3FBD
FFA3: 35 02     PULS  A               ;
FFA5: 3B       RTI                    ;

```




The new animation ISR at \$FF84, above, is modeled after The Getaway as described above. The \$B4 is treated as a circular counter going from 0x03 → 0x02 → 0x01 → 0x00 and back to 0x03, continuously. If a value greater than 0x03 (or a 0x00 byte is found, unexpectedly) in \$B4 then it is assumed that animation code pushed such value into \$B4 to cause the \$EE91 function to cease updating the medium/dim DMD page index into the DMD driver board.









With the above changes in place, the T2 L8.3 ROM will neatly display the DMD animations without the flicker issue previously described.











With the flicker issue cleared up, there may still be other coding errors in the display of the animations which are not due to the \$FF84 routine but, instead, due to issues in the code specific to the animation code that invokes the \$FB88 animation initialization code while the “Animation Code” is set to “Corrected”. An analysis of each of the animations has been performed and described in the next section of this document.

## The L8.3 Display Animation Analysis and Fixes



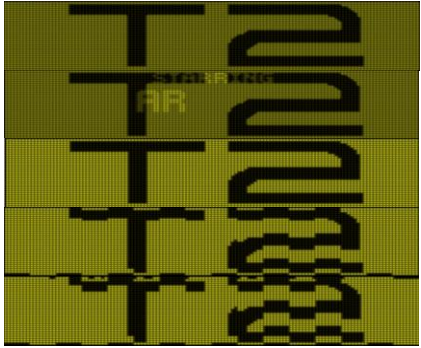




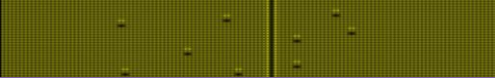

With the L8.3 fixing a specific set of display animations (as described above), a further round of analysis was performed on each of the affected animations to ensure they are properly being displayed. Using a smartphone to record each animation and playback at very slow speed, with a frame by frame analysis, some additional artifacts were observed in some of the animation sequences. Each of the animations was checked in this way and reported in the table below. Some of the additional issues are being fixed as part of the L8.3 software patch. A visual check was done comparing the observed anomalies in the emulator frame-by-frame analysis and videos posted online. **A majority of the emulator frame-by-frame issues mentioned below do not appear to be noticeable on real hardware and, as such, are not being addressed in L8.3, however will be documented here for posterity.**



#	Sequence	Animation Example	Observation	Problem Example
1	Doors Opening revealing award text based on value of the B register.		End of animation sometimes gets bright pixels immediately before transitioning back to scores. This might be perceived as a flicker.	Function: \$4088,24 - \$412B,24 <i>This issue is noticeable on real hardware and is addressed in L8.3 so pixels don't go bright when transitioning to the score display.</i> 
2	Hunter ship		This is first frame prior to crosshairs animation. No	Function: \$41CD,24 – \$42DC,24 No changes to this in L8.3.

			issues as animation changes from this to crosshairs.	
3	"Fire at will" crosshairs		Explosion animation when target is hit is good. If no trigger is pulled or target missed, animation ends with partial display of hunter ship as it transitions to showing scores.	Function: \$41CD,24 – \$42DC,24 This is not noticeable in videos of real hardware and not being addressed in L8.3.  
4	Pull Trigger to shoot ball		No problems when shown in attract mode or prior to "Fire at will".	Function: \$4412,24 – \$44B2,24 No changes to this in L8.3.
5	Jackpot		Full image is briefly flickered prior to the text build-up of the jackpot image.	Function: \$783F,33 – \$78C6,33 This issue was not noticed in videos of real hardware and is not likely to be noticeable during the excitement of game pay. Not being addressed in L8.3 
6	Replay		Likely an intentional brightness change / flicker as the replay animation changes	Function: \$7A20,33 – \$7A53,33 No changes to this in L8.3. 

			to multi-color image.	
7	Special		Likely an intentional brightness change / flicker as the special animation changes to multi-color image. Same effect when shown as database award or when collected at outlane. Mentioned here for completeness.	Function: \$7A56,33 – \$7A91,33 No changes to this in L8.3.   
8	Shoot Again		No problems with display of the “SHOOT” and “AGAIN” sequences.	Function: \$735F,35 No changes to this in L8.3.
9	“Boom” 1		No problems	Function: \$76CB,35 No changes to this in L8.3.
10	“Boom” 2		No problems	Function: \$76CB,35 No changes to this in L8.3.
11	Judgment day		Animation itself is good but the “StaringArnold” sequence afterwards has minor issue.  Prior to vertical column reveal, some of the “Starring” text is briefly shown.	Function: \$7862,35 Not noticeable in videos of real hardware. No changes to this in L8.3.  Vertical column reveal continues



12	T2		<p>Animation itself is good but the “StaringArnold” sequence afterwards has minor issue.</p> <p>Prior to vertical column reveal, some of the “Starring” text is briefly shown.</p>	<p>Function: \$7892,35 Not noticeable in videos of real hardware. No changes to this in L8.3.</p>   <p>Vertical column reveal continues</p>
13	<p>Kickback Lit</p> <p>This function is also used by bonus multiplier.</p>		<p>Autofire, no problems when awarded via database award or targets alike.</p> <p>2X, 4X, 6X, 8X, no problems.</p>	<p>Function: \$798C,35 No changes to this in L8.3.</p>
14	“I am the future”		No problems	<p>Function: \$7D82,35 No changes to this in L8.3.</p>
15	Kickback		No problems	<p>Function: \$7DD5,35 No changes to this in L8.3.</p>
16	Arnold with shotgun		<p>No problems in either of the 2 places when displayed during attract mode.</p>	<p>Function: \$7E43,35 No changes to this in L8.3.</p>
17	Elevator doors closed		No problems	<p>Function: \$7F25,35 No changes to this in L8.3.</p>
18	T-1000 blasted		<p>No problems. Doors are bright colored during shotgun blast for several frames as part of the effect.</p>	<p>Function: \$7F25,35 No changes to this in L8.3.</p>

19	T-1000 self-healed		Sometimes at end of animation, the pixels flicker bright as the animation transitions back to the score display.	Function: \$7F25,35 <i>This issue is noticeable on real hardware and is addressed in L8.3 so pixels don't go bright when transitioning to the score display.</i> 
----	--------------------	---	--	--

### Animation Fix: Security Levels

Depicted as animation #1, above, the security levels animation can cause pixels to get bright for a brief moment during the switch back to score display. To fix this, the end of the animation code will be updated to blank the display prior to the score display. This will prevent the flicker of pixel brightness. The animation function is from \$4088,24 through \$412B,24. The final instruction of the function at \$412B,24 (ROM image offset 0x1012B) is as follows:

```
412B: 7E C9 52    JMP    $C952          ; jump to function done
```

The function ending jump instruction is modified to jump to a previously unused area at the end of the current bank (bank \$24) where it will clear the display and then jump to the \$C952 to end the animation.

```
412B: 7E 57 49    JMP    $5749          ; jump to fixup code
```

The code now jumps to \$5749,24 (ROM image offset 0x11749) where the unused ROM region is now populated with the following code:

```
5749: 34 02        PSHS  A              ;
574B: 86 55        LDA   #$55          ;
574D: 97 B4        STA   $B4           ; Store 0x55 into $B4 to cease DMD update
574F: 35 02        PULS  A              ;
5751: BD FB AE    JSR   $FBAE         ; Call function clear DMD page memory
5754: 7E C9 52    JMP   $C952         ; jump to function done
```

With this change in place, the pixels do not get bright as the transition to score display takes place. A very slow-motion playback does reveal that the bright/medium dots go away and then the dim dots go away as screen is blanked which should be unperceivable during game play and not produce a flicker when animation transitions back to the score display anymore.

## Animation Fix: T-1000 Self-Healed/Extra Ball Award

Depicted as animation #19, above, the “T-1000 self-healed” animation (at extra ball redeem) ends with a brief display of bright pixels as the display transitions to the score display. To fix this, the end of the animation code will be updated to use the same method described above for “Security Levels” fix. The animation function ends at \$7FA2,35 (ROM image offset 0x57FA2) with the end of the function as follows:

```
7F96: 86 05      LDA    #$05          ; DMD Page selector 0x05
7F98: BD FB 88    JSR    $FB88         ; Init for DMD prior to drawing to it.
7F9B: 7E 7F 9E    JMP    $7F9E         ; <nop>
7F9E: BD 83 46    JSR    $8346         ; Sleep()
7FA1: 20          ;
7FA2: 35 B6      PULS   A,B,X,Y,PC   ;
```

Analysis of the function reveals there is a JMP instruction at \$7F9B which simply jumps to the very next instruction which then performs a delay (of value 0x20, which is 1/2 second) to retain the T-1000 image on the display for a brief moment before transitioning back to score display. To fix the bright-pixels issue, the code will be re-arranged so that we can perform a JSR instruction to new code that clears the display in a way similar to the fix for Security Levels display issue (described previously).

```
7F96: 86 05      LDA    #$05          ; DMD Page selector 0x05
7F98: BD FB 88    JSR    $FB88         ; Init for DMD prior to drawing to it.
7F9B: BD 83 46    JSR    $8346         ; Sleep()
7F9E: 20          ;
7F9F: BD 7E DC    JSR    $7EDC         ; Call display-clear fixup function
7FA2: 35 B6      PULS   A,B,X,Y,PC   ;
```

As depicted in the new code, the JMP instruction was removed and a JSR instruction was inserted prior to the function ending. The new JSR instruction jumps to code at \$7EDC,35 (the same bank as running code, \$35) which corresponds to ROM image offset 0x57EDC.

The fixup routine at \$7EDC,35 is added to ROM region immediately prior to this T-1000 Self-Healed function. At this region of ROM is the function for displaying the T2 Fan-Club message. Since L8.3 removes the calls to this function completely from the ROM, this now represents an unused region in the ROM which can be repurposed. A survey of the L8.3 ROM was done to confirm there are no calls to the fan-club function however out of an abundance of safety, the first instruction in the repurposed function is an RTS so that any attempts to invoke the fan-club function at its original address \$7EDB,35 will simply return without performing any work. To neatly repurpose the function, the now-unused bytes are also being set to 0xFF to more easily highlight the fact that the ROM space is now unused.

The new content of the ROM region for fan-club at \$7EDB,35 (ROM image offset 0x57EDB) is as follows:

```
7EDB: 39          RTS          ; Fan-club entry point now just returns
7EDC: 34 02      PSHS   A          ; Start of display-clear fixup routine
```



```

7EE8: 86 55      LDA   #$55      ;
7EE0: 97 B4      STA   $B4      ; Store 0x55 into $B4 to cease DMD update
7EE2: 35 02      PULS  A        ;
7EE4: BD FB AE   JSR   $FBAE    ; Call function clear DMD page memory
7EE7: 39         RTS          ; Done
;
7EE8: FF FF FF FF ; <unused ROM region>
7EEC: FF FF FF FF ; <unused ROM region>
7EF0: FF FF FF FF ; <unused ROM region>
7EF4: FF FF FF FF ; <unused ROM region>
7EF8: FF FF FF FF ; <unused ROM region>
7EFC: FF FF FF FF ; <unused ROM region>
7F00: FF FF FF FF ; <unused ROM region>
7F04: FF FF FF FF ; <unused ROM region>
7F08: FF FF FF FF ; <unused ROM region>
7F0C: FF FF FF FF ; <unused ROM region>
7F10: FF FF FF FF ; <unused ROM region>
7F14: FF FF FF FF ; <unused ROM region>
7F18: FF FF FF FF ; <unused ROM region>
7F1C: FF FF FF FF ; <unused ROM region>
7F20: FF FF FF FF ; <unused ROM region>
7F24: FF         ; <unused ROM region>

```

With this fix in place, instead of a momentary flicker of bright pixels, the extra-ball redeem animation neatly blanks prior to display of the game scores.

## The L8.3 Custom Message support in the ROM

As part of the L8.3 image a special function is added in to allow T2 owners and hobbyists to customize their ROM with an attract mode message that is part of the ROM image and, as such, becomes a permanent part of the attract mode sequence. This provides some functionality exceeding what the WPC “custom message” allows. *The standard WPC “custom message” feature is still present and part of L8.3.*

The L8.3 ROM support of custom message support has the following characteristics:

- Only a single frame may be created using this feature
- Up to 3 lines may be entered, up to 32 characters on each line
- Can specify any of the available fonts
- Can specify where on the DMD each line is shown
- Can specify an optional sound call that is played at the time of the custom message
- Can specify a wipe-pattern how the message is revealed, or instant-on display
- Can specify how much time the message is shown before attract mode proceeds to next item

The idea is that some trial and error would need to take place when crafting a custom message in the ROM especially since some fonts might end up causing wrap of characters on the display. Selecting the coordinates of where to place each line might also take some time to dial in.

This enhancement provides the ability to have a sound call added with the display of the message. Any of the existing sound calls can be specified. There is support to play a sound without any custom message if the desire is to have a periodic sound call during attract mode without any custom text on the display.

By default, the custom message is empty, nothing is shown. If any of the 3 lines contain text then the default characteristics are to use font and vertical line placement similar to the WPC “custom message” feature and default is to have instant-on display of the message without any wipe-mode effect to reveal the message.

Each line needs to end with a 0x00 or 0xFF byte to indicate end of the line. When first byte is 0x00 or 0xFF it means nothing should be printed for that line. Code enforces a limit of 32 characters per line and there are 32 bytes of ROM space set aside for each line. Ending 0x00 or 0xFF is not used if all 32 bytes are used. If font is too large and/or if too many characters are used on a line, the text may wrap.

To prevent the “U6 Cksum Error” test report, the checksum for the image must be recalculated and updated in the ROM image. Details on generating a valid checksum are outside the scope of this document. Check online resources to help generate a valid checksum. The 2<sup>nd</sup> byte of the checksum needs to be 0x08 for proper operation and in order to not trigger a factory settings reset when changing from an existing L-8, L8.1 or L8.2 ROM image.

## The L8.3 Custom Message Assembly Code

The following code is inserted into the L8.3 ROM image toward the end of bank \$3D at WPC address \$7F65,3D (ROM image offset 0x77F65). Some of the user adjustable values are directly in this function while the actual text strings are located in unused ROM region prior to this function.

```
-----;-----
7F65: 34 16      PSHS  X,B,A      ;
;
7F67: 86 00      LDA   #$00        ; Sound call number to play (pre-reveal)
7F69: 8D 5A      BSR   $7FC5       ; Call routine to play sound if attract-sounds are on
;
7F6B: 8E 7F 00   LDX   #$7F00      ; Address of string #1. Each string is 0x20 bytes
7F6E: 8D 48      BSR   $7FB8       ; Call subroutine to set B if there is a msg
7F70: 5D          TSTB              ;
7F71: 27 35      BEQ   $7FA8       ; If B=0x00 there is no msg to show, skip to the end
;
; At this point we are committed to play a message.
7F73: 34 02      PSHS  A           ; Save the pre-reveal sound index onto stack
;
7F75: 8D 5B      BSR   $7FD2       ; Call subroutine to clear DMD ram/display
;
7F77: 8D 31      BSR   $7FAA       ; Routine to copy string from X to $0386 and X+=0x20
7F79: BD D7 99   JSR   $D799       ; Write string from $0386 to display ram
7F7C: 00 00      ;
7F7E: 01          ; Font index 0x01 = 7 high single stroke
7F7F: 40 07      ; Center horizontally, bottom starts at line 7
;
7F81: 8D 27      BSR   $7FAA       ; Routine to copy string from X to $0386 and X+=0x20
7F83: BD D7 99   JSR   $D799       ; Write string from $0386 to display ram
7F86: 00 00      ;
7F88: 01          ; Font index 0x01 = 7 high single stroke
7F89: 40 11      ; Center horizontally, bottom starts at line 17
;
7F8B: 8D 1D      BSR   $7FAA       ; Routine to copy string from X to $0386 and X+=0x20
7F8D: BD D7 99   JSR   $D799       ; Write string from $0386 to display ram
7F90: 00 00      ;
7F92: 01          ; Font index 0x01 = 7 high single stroke
7F93: 40 1C      ; Center horizontally, bottom starts at line 28
;
; The next 6 bytes adjust the reveal/wipe pattern
; -----
7F95: BD E2 74   JSR   $E274       ; BD E2 74 12 12 12 == instant-full frame display
7F98: 12 12 12   ; BD 88 F5 7F 57 33 == Center-out vertical split reveal
; BD 88 F5 7C F5 33 == Alternating vertical columns
;
7F9B: 35 02      PULS  A           ; Load the pre-reveal sound from stack back into A
7F9D: 4D          TSTA              ; If pre-reveal sound was non-zero then we skip over
7F9E: 26 04      BNE   $7FA4       ; the post-reveal sound to avoid 2x sound call attempts
;
7FA0: 86 00      LDA   #$00        ; Sound call number to play (post-reveal)
7FA2: 8D 21      BSR   $7FC5       ; Call routine to play sound if attract-sounds are on
;
7FA4: BD 83 46   JSR   $8346       ; Sleep() to keep message on display
7FA7: C0          ; Adjust 0xC0 to desired period.
7FA8: 35 96      PULS  A,B,X,PC    ;
;
-----;
```

*Some additional new helper functions are not depicted and left as an exercise for the reader.*

## The L8.3 Custom Message Adjustable Values Table

	<b>Bank \$3D Addr</b>	<b>ROM Image Addr</b>	<b>Default Value</b>	<b>Info</b>
<b>Line 1 Text</b>	0x7F00 - 0x7F1F	0x77F00 - 0x77F1F	0xFF .. 0xFF (no text)	Up to 32 characters
<b>Line 1 Font</b>	0x7F7E	0x77F7E	0x01	See below for valid font values
<b>Line 1 X-position</b>	0x7F7F	0x77F7F	0x40	0x40 = centered
<b>Line 1 Y-position</b>	0x7F80	0x77F80	0x07	Bottom pixel row of line
<b>Line 2 Text</b>	0x7F20 - 0x7F3F	0x77F20 - 0x77F3F	0xFF .. 0xFF (no text)	Up to 32 characters
<b>Line 2 Font</b>	0x7F88	0x77F88	0x01	See below for valid font values
<b>Line 2 X-position</b>	0x7F89	0x77F89	0x40	0x40 = centered
<b>Line 2 Y-position</b>	0x7F8A	0x77F8A	0x11	Bottom pixel row of line
<b>Line 3 Text</b>	0x7F40 - 0x7F5F	0x77F40 - 0x77F5F	0xFF .. 0xFF (no text)	Up to 32 characters
<b>Line 3 Font</b>	0x7F92	0x77F92	0x01	See below for valid font values
<b>Line 3 X-position</b>	0x7F93	0x77F93	0x40	0x40 = centered
<b>Line 3 Y-position</b>	0x7F94	0x77F94	0x1C	Bottom pixel row of line
<b>Sound Call Pre-Reveal</b>	0x7F68	0x77F68	0x00	Sound index number
<b>Reveal/Wipe Pattern</b>	0x7F95 - 0x7F9A	0x77F95 - 0x77F9A	0xBD 0xE2 0x74 0x12 0x12 0x12	6 bytes wipe/reveal mode
<b>Sound Call Post-Reveal</b>	0x7FA1	0x77FA1	0x00	Sound index number
<b>Display Period</b>	0x7FA7	0x77FA7	0xC0	0x00 - 0xFF display period

## The L8.3 Custom Message Adjustable Values Info

Additional details for each of the adjustable items are provided below. In some cases it is an exercise of trial and error to see how the modifications result in the updated custom message on the display. It is suggested that a WPC emulation environment is first used to test changes.

### Line Text

As mentioned, up to 32 characters are allowed for each line. As in most cases when there are less than 32 characters on a line, the line text should end in 0xFF or 0x00. The code specifically looks for 0xFF or 0x00 as the line ending. If the line contains the maximum 32 characters then it must not be ended with 0xFF or 0x00 byte since code will automatically end the line at 32 characters. There are fixed locations in the ROM image for each line:

Text for...	ROM Image offset
Line 1	0x77F00 - 0x77F1F
Line 2	0x77F20 - 0x77F3F
Line 3	0x77F40 - 0x77F5F

Any or all of the 3 lines may be used. When choosing larger fonts it may be necessary to only use 2 lines or a single line to get the characters to display properly. The default font and placement uses a 7-high font which allows for all 3 lines to contain text without having to adjust the default font or x/y placement for each line.

An example modification to each of the 3 lines in a hex editor might look like the following:

```

00077F00 4C 65 74 27 73 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
00077F10 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
00077F20 50 6C 61 79 FF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
00077F30 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
00077F40 50 69 6E 62 61 6C 6C FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
00077F50 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
00077F60 FFFF FFFF FFFF 34 16 86 00 8D 5A 8E 7F 00 8D 48
00077F70 5D 27 35 34 02 8D 5B 8D 31 BD D7 99 00 00 01 40
00077F80 07 8D 27 BD D7 99 00 00 01 40 11 8D 1D BD D7 99
00077F90 00 00 01 40 1C BD E2 74 12 12 12 35 02 4D 26 04
00077FA0 86 00 8D 21 BD 83 46 C0 35 96 8D 3B 20 06 A1 00

```

```

Let's yyy yyy yyy yyy yyy yyy
yyy yyy yyy yyy yyy yyy yyy
Play yyy yyy yyy yyy yyy yyy
yyy yyy yyy yyy yyy yyy yyy
Pinball yyy yyy yyy yyy
yyy yyy yyy yyy yyy yyy yyy
yyy yyy y 4 t | Z Z | H
] ' 5 4 | [ | 1 ½ x 200 | r 0
• | ' ½ x 200 | r 0 ◀ | ½ x 200
| r 0 ½ a t | ↑ ↓ | 5 | M & J
t | ! ½ f f à 5 - | ; - ;

```

In the example, above, each line was ended with the 0xFF byte. The single byte after each line text could also have ended with 0x00 byte for those who like to follow more common programming syntax.







Some experimentation has shown that the Y-Position values are 0-based which means valid values would be 0x00 through 0x1F (31 decimal). This means the first line showing a 7-high pixel font with Y-position value 0x07 would display the line with a single row of unlit pixels on the top-most row of the display.

Continuing the example using some different Y-values to move the vertical position of the example lines of text:

00077F00	4C 65 74 27 73	FFFFFFFFFFFFFFFFFFFFFFFF	Let's
00077F10	FFFFFFFFFFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFFFFFFFFFF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00077F20	50 6C 61 79	FFFFFFFFFFFFFFFFFFFFFFFF	Play
00077F30	FFFFFFFFFFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFFFFFFFFFF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00077F40	50 69 6E 62 61 6C 6C	FFFFFFFFFFFFFFFFFFFFFFFF	Pinball
00077F50	FFFFFFFFFFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFFFFFFFFFF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00077F60	FFFFFFFFFF34 16 86 00 8D 5A 8E 7F 00 8D 48	FFFFFFFFFFFFFFFFFFFFFFFF	ÿÿÿÿÿ4T†   ZŽ   H
00077F70	5D 27 35 34 02 8D 5B 8D 31 BD D7 99 00 00 01 20	FFFFFFFFFFFFFFFFFFFFFFFF	] ' 54γ   [ [ 1½×³³ γ
00077F80	17 8D 27 BD D7 99 00 00 01 40 1B 8D 1D BD D7 99	FFFFFFFFFFFFFFFFFFFFFFFF	†   ' ½×³³ γ   ←   ½×³³
00077F90	00 00 01 60 1F BD E2 74 12 12 12 35 02 4D 26 04	FFFFFFFFFFFFFFFFFFFFFFFF	γ ` ½â† ↓ ↓ ↓ 5γ M&J
00077FA0	86 00 8D 21 BD 83 46 C0 35 96 8D 3B 20 06 A1 00	FFFFFFFFFFFFFFFFFFFFFFFF	†   ! ½fF à 5-   ; - ;

As shown in the red highlighted byte values, the Y-Position values for the three lines of text are changed to 0x17 (23 decimal), 0x1B (27 decimal), and 0x1F (31 decimal), respectively. This means the 3<sup>rd</sup> line will be on the bottom-most line on the 32-line display. The bottom of the 2<sup>nd</sup> line is at row 28 (of 32 rows) and the bottom of the 1<sup>st</sup> line is at row 24 (of 32 rows). The result of this continued example is as follows:



In this case we can display each 7-high font line with overlapping Y-position values since we've previously adjusted the X-position to ensure they are located in different horizontal positions on each line.

### Sound Call

The L8.3 custom message feature also supports the ability for a sound to be played in conjunction with the display of the custom message.

A sound index can be specified *during* the reveal of the message or *after* the reveal of the message. This allows some fine tuning of when the sound plays especially when a wipe/reveal pattern is used to

display the message over a perceivable period of time. It might be preferred to have the sound play *during* or play *after* the reveal pattern is complete. Code will only allow for a single sound to be used. If a sound call is specified in both positions, only the first, pre-reveal, sound call will be used.

By default the sound call value is 0x00 which means no sound call will be attempted. When a value other than 0x00 is used then the sound-call will be sent to the sound board if “Attract Sounds” is configured to “on”. If the attract sounds are not enabled, then there will be no sound played with the message. In the case where no text is defined to be shown (All 3 lines start with 0x00 or 0xFF byte) only the pre-reveal sound will be supported to play a sound without displaying any custom message on the display. This allows flexibility of having a selected sound play periodically during attract mode but not with an accompanying custom message on the display (it will play the pre-reveal sound while showing whatever is next in the attract mode sequence).

The single byte sound number that is specified is passed into a function that the game uses to play the desired sound number. Sound numbers that may be specified are listed in the table below:

Sound Number	Sound Effect
0x00	Disabled, no sound
0x01	<none>
0x02	<none>
0x03	Missile flyby
0x04	Waszhawawawu
0x05	Laser shot
0x06	Engine rev-up (doesn't stop!)
0x07	Tingtingting
0x08	Dwoh (database award sound)
0x09	Dwang (database award sound)
0x0A	Twing (database award sound)
0x0B	Da do do woo
0x0C	Dit dit da doot doot doo
0x0D	Fwizzzzp
0x0E	Fwzip fwzip
0x0F	Space ship up with drumbeat
0x10	Splat
0x11	Zip dip dip
0x12	Klew
0x13	Blew
0x14	Zap zap whap whap
0x15	Space ship up away
0x16	Blip (skill shot?)
0x17	Zeeee phweeee
0x18	Zee fwa fwa fwa fwa
0x19	Zee wahn zee zorm

<b>0x1A</b>	Zee wohm zee zohm
<b>0x1B</b>	Zee wohm zee wohm
<b>0x1C</b>	Zee wohm mee wohm
<b>0x1D</b>	Space email notification
<b>0x1E</b>	Space doorbell
<b>0x1F</b>	Zippzeooo
<b>0x20</b>	Bzeweww
<b>0x21</b>	Spaceship takeup
<b>0x22</b>	Scwooph
<b>0x23</b>	Bzeww weooo
<b>0x24</b>	Game show prize award
<b>0x25</b>	Computery doowhip
<b>0x26</b>	Computery bangy
<b>0x27</b>	Computery sudden
<b>0x28</b>	Computery alarmy noise
<b>0x29</b>	Computery ramp-up blongy award
<b>0x2A</b>	Computer gun load
<b>0x2B</b>	Spacey computer whirz
<b>0x2C</b>	Spacey drill whir
<b>0x2D</b>	Bluzz
<b>0x2E</b>	Quuz
<b>0x2F</b>	Wfuz
<b>0x30</b>	Pfz
<b>0x31</b>	Pfuz
<b>0x32</b>	Pfaz
<b>0x33</b>	Pfiz
<b>0x34</b>	Wraaa
<b>0x35</b>	Low-pitch dramatic standoff music (music plays)
<b>0x36</b>	Computer wizzzz
<b>0x37</b>	Computer acute fare
<b>0x38</b>	Computer blipnoise
<b>0x39</b>	Computer ban fribblezee
<b>0x3A</b>	Computer bang fribble
<b>0x3B</b>	Spooky warble
<b>0x3C</b>	Do do dat dwoo
<b>0x3D</b>	Computer flsmish
<b>0x3E</b>	Computer smoozmle
<b>0x3F</b>	Computer flashzm
<b>0x40</b>	Computer smashzle
<b>0x41</b>	Computer sizzle
<b>0x42</b>	Computer sizzle
<b>0x43</b>	Computer scramble
<b>0x44</b>	Downward low zip
<b>0x45</b>	Downward foreboding

<b>0x46</b>	Bonus foreboding
<b>0x47</b>	Computer splaws
<b>0x48</b>	Computer kblawz
<b>0x49</b>	Computer splat
<b>0x4A</b>	Twilight zoney
<b>0x4B</b>	Boom (during match?)
<b>0x4C</b>	Spacey twang down
<b>0x4D</b>	Spacey engine
<b>0x4E</b>	Spacey vlarble
<b>0x4F</b>	Spacey vle vle vle
<b>0x50</b>	Vu-vilp
<b>0x51</b>	Twangz
<b>0x52</b>	Door slam
<b>0x53</b>	Helicopter buzz by
<b>0x54</b>	Crack
<b>0x55</b>	Whack
<b>0x56</b>	Kabash
<b>0x57</b>	Patter patter patter patter
<b>0x58</b>	Shot
<b>0x59</b>	Shatter
<b>0x5A</b>	Blast
<b>0x5B</b>	Drop smash
<b>0x5C</b>	Slam
<b>0x5D</b>	Engine revving
<b>0x5E</b>	Motorcycle
<b>0x5F</b>	Swoosh
<b>0x60</b>	Crash
<b>0x61</b>	Smash
<b>0x62</b>	Gunshot
<b>0x63</b>	Punch/smack
<b>0x64</b>	Collision
<b>0x65</b>	"Take your best shot"
<b>0x66</b>	"Fire at will"
<b>0x67</b>	"You missed"
<b>0x68</b>	"Direct hit"
<b>0x69</b>	"Great shot"
<b>0x6A</b>	"Lock sequence initiated"
<b>0x6B</b>	"Get the extra ball"
<b>0x6C</b>	<none>
<b>0x6D</b>	"Extra ball"
<b>0x6E</b>	Bang sound higher pitched (?)
<b>0x6F</b>	Bang sound lower pitched (?)
<b>0x70</b>	"I'll be back"
<b>0x71</b>	"It's payback time"

0x72	"Get the jackpot"
0x73	"Jackpot"
0x74	"Checkpoint 1 secured"
0x75	"Passcode secured"
0x76	"Silent alarm deactivated"
0x77	"Vault key secured"
0x78	"Get the CPU"
0x79	"Hurry up"
0x7A	"Autofire"
0x7B	"Video mode"
0x7C	"Load the cannon"
0x7D	"Shoot again"
0x7E	"Get out"
0x7F	"Well done"
0x80	"Awesome"
0x81	"Nice shot"
0x82	"Judgment day"
0x83	"Let's go"
0x84	"Go!"
0x85	"Run"
0x86	"I am the future"
0x87	"You are superior"
0x88	"Get the super jackpot!"
0x89	"Big Points"
0x8A	"Autofire Deactivated"
0x8B	"Way to go"
0x8C	"No"
0x8D	"Hasta la vista, baby"
0x8E	"Chill out"
0x8F	"Excellent"
0x90	"Get down"
0x91	"He'll live"
0x92	FUA (Sound board needs profanity ROM U14)
0x93	"Woopdadidoo"
0x94	Boom (attract mode)
0x95	"Deactivated"
0x96	"Activated"
0x97	"Time to go"
0x98	"Right now"
0x99	"Don't move"
0x9A	"They're here"
0x9B	"Out of the way"
0x9C	"Look out"
0x9D	"I am a Cyberdyne Systems series 800 terminator"



<b>0x9E</b>	"Reloaded"
<b>0x9F</b>	"Destroy everything"
<b>0xA0</b>	"Terminated"
<b>0xA1</b>	"You're targeted for termination"
<b>0xA2</b>	"Get behind me if you want to live"
<b>0xA3</b>	"No way, Jose"
<b>0xA4</b>	"No problemo"
<b>0xA5</b>	"I am a cybernetic organism"
<b>0xA6</b>	"You missed everything"
<b>0xA7</b>	Crash
<b>0xA8</b>	Lengthy award
<b>0xA9</b>	Motorcycle rev and drive by
<b>0xAA</b>	Motorcycle shifting gears
<b>0xAB</b>	"Video mode activated"
<b>0xAC</b>	"Hurryup activated"
<b>0xAD</b>	Pzwee
<b>0xAE</b>	Gunshot
<b>0xAF</b>	Fizzle wizzle
<b>0xB0</b>	Balink
<b>0xB1</b>	Short trouble wrabble
<b>0xB2</b>	Elevating wrabble
<b>0xB3</b>	Dat dat dat do woo
<b>0xB4</b>	Elevating wribble
<b>0xB5</b>	Laser fizzle
<b>0xB6</b>	Smashey
<b>0xB7</b>	Punchy
<b>0xB8</b>	Boulder rumble
<b>0xB9</b>	"Destroy everything"
<b>0xBA</b>	"Well done"
<b>0xBB</b>	"Extra ball"
<b>0xBC</b>	Short smack
<b>0xBD - 0xFF</b>	<none>

The sound effect names in the table, above, were derived using a pinball emulator and modified ROM image to manually trigger each sound. The names of each sound were quickly derived based on what was heard from the emulated sound board. Sounds were sampled with the L-8 sound ROM set and also with the sound ROM U14 replaced with the "profanity" sound ROM. The only observed difference was the sound 0x92 would play the FUA quote with profanity U14 ROM while it would play nothing with the regular U14 ROM.

An example of having sound modification at ROM offset 0x77F68 changing the 0x00 byte to "Chill out" we change the pre-reveal sound byte value to 0x8E as shown by the red highlighted byte below:

00077F00	4C 65 74 27 73	FFFFFFFFFFFFFFFF	Let 's
00077F10	FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF	ÿÿÿÿÿÿÿÿÿÿÿÿ
00077F20	50 6C 61 79	FFFFFFFFFFFFFFFF	Playÿÿÿÿÿÿÿÿÿÿÿÿ
00077F30	FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00077F40	50 69 6E 62 61 6C 6C	FFFFFFFFFFFFFFFF	Pinballÿÿÿÿÿÿÿÿÿÿÿÿ
00077F50	FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00077F60	FFFFFFFFFF34 16 8E	8D 5A 8E 7F 00 8D 48	ÿÿÿÿÿ4ÿ† Ž Ž Ž Ž H
00077F70	5D 27 35 34 02 8D 5B 8D 31	BD D7 99 00 00 01 20	] ' 54ÿ [ [ 1½× <sup>ms</sup> ÿ
00077F80	17 8D 27 BD D7 99 00 00 01 40 1B 8D 1D BD D7 99		{ [ ' ½× <sup>ms</sup> ÿ @← [ ½× <sup>ms</sup>
00077F90	00 00 01 60 1F BD E2 74 12 12 12 35 02 4D 26 04		ÿ ÿ ½â† ↓ ↓ ↓ 5ÿ M&J
00077FA0	86 00 8D 21 BD 83 46 C0 35 96 8D 3B 20 06 A1 00		† [ ! ½fF à 5 - [ ; - i

If a reveal/wipe pattern is selected (described in next section) and the desire is to have the sound play *after* the reveal is complete then the sound byte at 0x77FA1 should contain the desired sound byte value and the byte at 0x77F68 should be 0x00 so that the post-reveal sound is played.

The hex-editor example below shows the post-reveal sound-call byte at 0x77FA1 set to 0x8E to play “Chill out” after the reveal of the message is complete. *The following also depicts the pre-reveal sound at 0x77F68 set to 0x00.*

00077F00	4C 65 74 27 73	FFFFFFFFFFFFFFFF	Let 's
00077F10	FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF	ÿÿÿÿÿÿÿÿÿÿÿÿ
00077F20	50 6C 61 79	FFFFFFFFFFFFFFFF	Playÿÿÿÿÿÿÿÿÿÿÿÿ
00077F30	FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00077F40	50 69 6E 62 61 6C 6C	FFFFFFFFFFFFFFFF	Pinballÿÿÿÿÿÿÿÿÿÿÿÿ
00077F50	FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00077F60	FFFFFFFFFF34 16 86	00 8D 5A 8E 7F 00 8D 48	ÿÿÿÿÿ4ÿ† [ Ž Ž Ž Ž H
00077F70	5D 27 35 34 02 8D 5B 8D 31	BD D7 99 00 00 01 20	] ' 54ÿ [ [ 1½× <sup>ms</sup> ÿ
00077F80	17 8D 27 BD D7 99 00 00 01 40 1B 8D 1D BD D7 99		{ [ ' ½× <sup>ms</sup> ÿ @← [ ½× <sup>ms</sup>
00077F90	00 00 01 60 1F BD E2 74 12 12 12 35 02 4D 26 04		ÿ ÿ ½â† ↓ ↓ ↓ 5ÿ M&J
00077FA0	86 8E 8D 21 BD 83 46 C0 35 96 8D 3B 20 06 A1 00		† Ž [ ! ½fF à 5 - [ ; - i

## Reveal/Wipe

When the custom message is displayed there are a few ways in which the custom message may be revealed (or “wiped”) onto the display. There are 6 contiguous bytes that are used to select the desired reveal pattern, as shown in the table below.

ROM Offset	Data Bytes	Reveal / Wipe mode
0x77F95	0xBD 0xE2 0x74 0x12 0x12 0x12	Instant-on display
0x77F95	0xBD 0x88 0xF5 0x7F 0x57 0x33	Center-out vertical split reveal
0x77F95	0xBD 0x88 0xF5 0x7C 0xF5 0x33	Alternating vertical columns up/down reveal

At ROM offset 0x77F95 is the start of the 6-byte pattern. By default, the Instant-on display is used. The way in which the custom ROM message feature for L8.3 is designed, the display will be blank prior to the reveal of the custom message.

For all reveal/wipe modes, after the full message is shown, the post-reveal sound is played, if specified, and the delay period is observed before the attract mode proceeds to the next sequence.

### Instant-on display

The instant-on display simply displays the message instantly without any fancy pattern or effect. This is the default method used to display the custom ROM message in L8.3. The default reveal/wipe mode is specified by the 6 bytes starting at ROM offset 0x77F95, as shown in red, below:

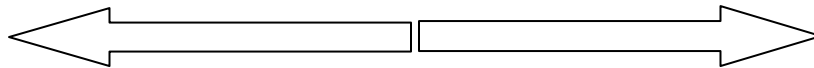
00077F00	4C 65 74 27 73 FFFF FFFF FFFF FFFF FFFF FFFF	Let 's yyy yyy yyy yyy
00077F10	FF FFFF FFFF FFFF FFFF FFFF FFFF FFFF	yy yyy yyy yyy yyy yyy
00077F20	50 6C 61 79 FFFF FFFF FFFF FFFF FFFF FFFF	Play yyy yyy yyy yyy yyy
00077F30	FF FFFF FFFF FFFF FFFF FFFF FFFF FFFF	yy yyy yyy yyy yyy yyy
00077F40	50 69 6E 62 61 6C 6C FFFF FFFF FFFF FFFF	Pinball yyy yyy yyy yyy
00077F50	FF FFFF FFFF FFFF FFFF FFFF FFFF FFFF	yy yyy yyy yyy yyy yyy
00077F60	FF FFFF FFFF 34 16 86 00 8D 5A 8E 7F 00 8D 48	yy yyy y 4 T t [ Z Z [ H
00077F70	5D 27 35 34 02 8D 5B 8D 31 BD D7 99 00 00 01 20	] ' 5 4 [ [ [ 1 ½ x 23 r
00077F80	07 8D 27 BD D7 99 00 00 01 40 11 8D 1D BD D7 99	• [ ' ½ x 23 r [ ◀ [ ½ x 23
00077F90	00 00 01 60 1F <b>BD E2 74 12 12 12</b> 35 02 4D 26 04	r ` ½ at ↓ ↓ ↓ 5 [ M & J
00077FA0	86 00 8D 21 BD 83 46 C0 35 96 8D 3B 20 06 A1 00	t [ ! ½ f F À 5 - [ ; - i

### Center-out vertical split reveal

The center-out vertical split reveal is used by a few attract mode messages and can be enabled for the custom message. Example hex-editor output selecting this reveal pattern is depicted below.

00077F00	4C 65 74 27 73 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF	Let's yyyyyyyyyyyyyy
00077F10	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF	yyyyyyyyyyyyyyyyyy
00077F20	50 6C 61 79 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF	Play yyyyyyyyyyyy
00077F30	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF	yyyyyyyyyyyyyyyyyy
00077F40	50 69 6E 62 61 6C 6C FFFFFFFF FFFFFFFF FFFFFFFF	Pinball yyyyyyyyyyy
00077F50	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF	yyyyyyyyyyyyyyyyyy
00077F60	FFFFFFFF FFF341686008D5A8E7F008D48	yyyyy4T+    ZŽ    H
00077F70	5D273534028D5B8D31BDD79900000120	] ' 54γ    [ 1½× <sup>m</sup> γ
00077F80	178D27BDD799000001401B8D1DBDD799	+    ' ½× <sup>m</sup> γ @←    ½× <sup>m</sup>
00077F90	000001601FBD88F57F573335024D2604	γ ' ½^8    W35γ M&J
00077FA0	868E8D21BD8346C035968D3B2006A100	+ Ž    ! ½fF à5-    ; - i

Using this reveal mode to display the example message is depicted below. Starting with blank display, the message starts to appear from the center of the display, exposing the message outwardly exposing each column of pixels of text in a smooth reveal, despite the screen shots, below, depicting the reveal more coarsely.



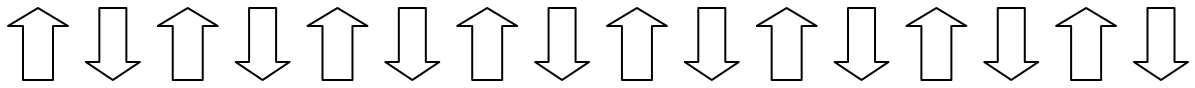
### Alternating vertical columns up/down reveal

The alternating vertical columns up/down reveal is also used by a few attract mode messages and can be enabled for the custom ROM message feature. Example hex-editor output selecting this reveal pattern is depicted below.

```
00077F00 4C 65 74 27 73 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF | Let's yyy yyy yyy yyy yyy yyy yyy yyy
00077F10 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF | yyy yyy yyy yyy yyy yyy yyy yyy yyy yyy
00077F20 50 6C 61 79 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF | Play yyy yyy yyy yyy yyy yyy yyy yyy
00077F30 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF | yyy yyy yyy yyy yyy yyy yyy yyy yyy yyy
00077F40 50 69 6E 62 61 6C 6C FFFF FFFF FFFF FFFF FFFF FFFF FFFF | Pinball yyy yyy yyy yyy yyy yyy yyy yyy
00077F50 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF | yyy yyy yyy yyy yyy yyy yyy yyy yyy yyy
00077F60 FFFF FFFF FFFF 34 16 86 00 8D 5A 8E 7F 00 8D 48 | yyy yyy y4 T t  Z Z  H
00077F70 5D 27 35 34 02 8D 5B 8D 31 BD D7 99 00 00 01 20 | ] ' 54  [ [ 1  x  r
00077F80 17 8D 27 BD D7 99 00 00 01 40 1B 8D 1D BD D7 99 | } [ '  x  r @ < [ x
00077F90 00 00 01 60 1F BD 88 F5 7C F5 33 35 02 4D 26 04 | r ` x ^ 5 | 5 3 5 M & J
00077FA0 86 8E 8D 21 BD 83 46 C0 35 96 8D 3B 20 06 A1 00 | t Z ! x f F 5 - ; - i
```

Using this reveal mode to display the example message is depicted below. Starting with blank display, the message starts to appear from the top and bottom of the display in alternating columns. The columns are revealed exposing each vertical column of the message in a smooth reveal, despite the screen shots, below, depicting the reveal more coarsely.





```
                F nt il
L t'
```

```
                F nt il
L t'
```

```
                F nt il
L t'
          F a'
```

```
                F nt il
L t'
          F a'
```

```
                F nt il
L t'
          F a' F nt il
                ' : ' = 1
```

```
Let's
  Play
    Pinball
```

```
Let's
  Play
    Pinball
```

## Display Period

The custom ROM message feature also allows the delay period to be specified which controls how long the message stays on the display prior to proceeding to the next normal attract mode sequence.

The byte at ROM offset 0x77FA7, by default is set to 0xC0 to provide a decent delay. This byte can be modified larger or smaller within the range of 0x00..0xFF to make the message appear for a shorter or longer period, respectively. *Value 0x40 corresponds to 1 second. Value 0xC0 is 3 seconds.*

For example, below, the message is set to appear for 1 second by making the value 0x40.

00077F00	4C 65 74 27 73 FF FF FF FF FF FF FF FF FF FF FF FF	Let 's yyy yyy yyy yyy yyy
00077F10	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	yy yyy yyy yyy yyy yyy yyy yyy
00077F20	50 6C 61 79 FF FF FF FF FF FF FF FF FF FF FF FF	Play yyy yyy yyy yyy yyy yyy
00077F30	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	yy yyy yyy yyy yyy yyy yyy yyy
00077F40	50 69 6E 62 61 6C 6C FF FF FF FF FF FF FF FF FF	Pinball yyy yyy yyy yyy yyy
00077F50	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	yy yyy yyy yyy yyy yyy yyy yyy
00077F60	FF FF FF FF FF 34 16 86 00 8D 5A 8E 7F 00 8D 48	yy yyy yyy 4T+ [ Z Z [ H
00077F70	5D 27 35 34 02 8D 5B 8D 31 BD D7 99 00 00 01 20	] ' 54 [ [ [ 1 1/2 x 200 r
00077F80	17 8D 27 BD D7 99 00 00 01 40 1B 8D 1D BD D7 99	+ [ 1 1/2 x 200 r @ ← [ 1/2 x 200
00077F90	00 00 01 60 1F BD 88 F5 7C F5 33 35 02 4D 26 04	r ` 1/2 ^ 0   0 3 5 7 M & J
00077FA0	86 8E 8D 21 BD 83 46 40 35 96 8D 3B 20 06 A1 00	+ Z [ ! 1/2 f F 0 5 - [ ; - i

## Fonts

Some characters are shown, below, for each of the supported fonts. Some fonts support various special characters and punctuation characters and some fonts also support lower case letters. Not all characters are shown, below, for each font.

Font is specified on a per-line basis. It is important that the X/Y location bytes are also adjusted to ensure the display of the font character doesn't cause wrap on the display. A valid combination of font and X/Y location must be used so that it appears on the display in a pleasing way.

The font selector byte for each line is summarized as the following:

Font for...	ROM Image offset
Line 1	0x77F7E
Line 2	0x77F88
Line 3	0x77F92

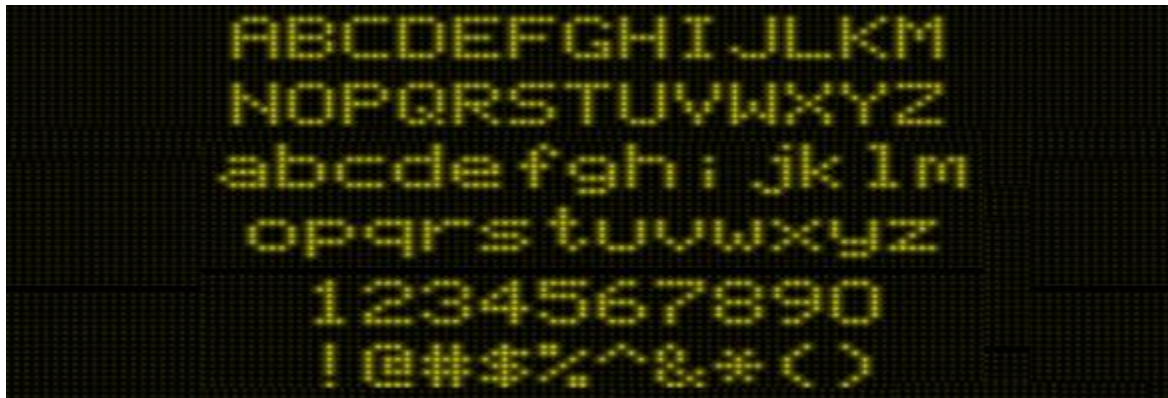
The default value for L8.3 is font selection 0x01 which provides a 7-high single-stroke font. A summary of all fonts in the T2-L8 font table is as follows. For completeness, all font table info is listed. The fonts that are used for graphics/animations are listed and grayed out. Ranges of supported characters are shown in < > brackets such as <0-9> to indicate support for digits 0 through 9 (where it shows <=> it is depicting support for the 3 characters '<', '=', and '>'). Spacebar character support is denoted with "<>"

Font	Description	Support	Character Support
0x00	7-high, single stroke	Full character support	< > ! " # \$ % & ' ( ) * + , - . / <0-9> ; <=> ? @ <A-Z> [ \ ] ^ _ ` <a-z> {   } ~
0x01	7-high, single stroke	Full character support	< > ! " # \$ % & ' ( ) * + , - . / <0-9> ; <=> ? @ <A-Z> [ \ ] ^ _ ` <a-z> {   } ~
0x02	5-high, single stroke	Upper-case alpha, number, symbol	< > " # \$ % ' ( ) * + , - . / <0-9> : = ? <A-Z> [ \ ]
0x03	7-high, single stroke	Comma, period, digits (scoring)	, . <0-9>
0x04	7-high, single stroke	Comma, period, digits (scoring)	, . <0-9>
0x05	7-high, single stroke	Comma, Period, digits (scoring)	< > ) , . <0-9>
0x06	13-high double stroke	Comma, Period, digits (scoring)	, . <0-9>
0x07	13-high double stroke	Comma, Period, digits (scoring)	, . <0-9>
0x08	13-high double stroke	Comma, Period, digits (scoring)	, . <0-9>
0x09	10-high double stroke	Misc, Upper-case letters, digits	! " ` ) - . / <0-9> ; = <A-Z>
0x0A	15-high outlined	Misc, Upper-case letters, digits	< > ! " - . / <0-9> ? <A-Z>
0x0B	22-high outlined	Misc, Upper-case letters, digits	? - . <0-9> ? <A-Z>
0x0C	26-high triple-stroke	Misc, Alpha, digits	< > . <0-9> : <A-Z> <a-z>
0x0D	Graphics	Terminator Robot	0x01 - 0x0C
0x0E	Graphics	Shiny Pinball	0x01 - 0x0B
0x0F	7-high single stroke	Full character support	< > ! " # \$ % & ' ( ) * + , - . / <0-9> ; <=> ? @ <A-Z> [ \ ] ^ _ ` <a-z> {   } ~
0x10	15-high triple stroke	Misc, Upper-case letters, digits	< > ! " ) , - . <0-9> = ? <A-Z>
0x11	27-high wide stroke	Comma, period, digits (scoring)	, . <0-9>
0x12	32-high wide stroke	Digits only	<0-9>
0x13	25-high wide stroke	Digits only	<0-9>
0x14	20-high wide stroke	Digits only	<0-9>
0x15	13-high wide stroke	Digits only	<0-9>
0x16	8-high wide stroke	Digits only	<0-9>
0x17	4-high tiny	Misc, Upper-case letters, digits	< > , <0-9> <A-Z>
0x18	5-high tiny	Misc, Upper-case letters, digits	< > , . <0-9> <A-Z>
0x19	13-high double stroke	Misc, Upper-case letters, digits	< > , . <0-9> <A-Z>
0x1A	26-high wide stroke	Digits only	<0-9>
0x1B	Graphics	Large spinning digits and 'X'	0x01 - 0x2D
0x1C	Graphics	Large spinning plank	0x01 - 0x06
0x1D	Graphics	Used in Pull Trigger animation	0x01 - 0x16
0x1E	Graphics	Used in Match animation	0x01 - 0x1F
0x1F	Graphics	Circular shape	0x01 - 0x04
0x20	Graphics	Super Jackpot	0x01, 0x02
0x21	Graphics	Terminator Robot (same as 0x0D)	0x01 - 0x0C
0x22	Graphics	Circular shape (same as 0x1F)	0x01 - 0x04
0x23	Graphics	Motorcycle left	0x01 - 0x04
0x24	Graphics	Motorcycle left (same as 0x23)	0x01 - 0x04
0x25	Graphics	Motorcycle wheelie	0x01 - 0x0A
0x26	Graphics	Explosion	0x01 - 0x06
0x27	Graphics	Explosion	0x01 - 0x03
0x28	Graphics	Explosion	0x01 - 0x05
0x29	Graphics	Fire at will	0x01 - 0x02
0x2A	Graphics	Fire at will (same as 0x29)	0x01 - 0x02
0x2B	Graphics	Video mode	0x01 - 0x08
0x2C	Graphics	Video mode	0x01 - 0x08

<b>0x2D</b>	Graphics	Video mode	0x01 - 0x0B
<b>0x2E</b>	Graphics	Video mode	0x01 - 0x08
<b>0x2F</b>	Graphics	Video mode	0x01 - 0x08
<b>0x30</b>	Graphics	Video mode	0x01 - 0x06
<b>0x31</b>	Graphics	Video mode	0x01 - 0x06
<b>0x32</b>	Graphics	Video mode	0x01 - 0x0B
<b>0x33</b>	Graphics	Video mode	0x01 - 0x08
<b>0x34</b>	Graphics	Hunter ship	0x01 - 0x02
<b>0x35</b>	Graphics	EB Award	0x01 - 0x02
<b>0x36</b>	Graphics	Terminator	0x01 - 0x02
<b>0x37</b>	Graphics	Hunter ship	0x01 - 0x02

### Font 0x00

Font 0x00 is a 7-high pixel, single-stroke font with lower-case letters.



### Font 0x01

Font 0x01 appears to be identical to font 0x00, the internal code likely points to the same font data.





### Font 0x02

Font 0x02 is a 5-high font supporting only numbers, letters and some symbols.



### Font 0x03

Font 0x03 is a 7-high scoring font (only supporting digits, comma, period).



### Font 0x04

Font 0x04 is also a 7-high scoring font. This font is slightly wider than previous font.



### Font 0x05

Font 0x05 is another 7-high scoring font, slightly wider than previous font.



### Font 0x06

Font 0x06 is a 13-high double-stroke scoring font.



### Font 0x07

Font 0x07 is another 13-high scoring font, slightly larger than the previous.



### Font 0x08

Font 0x08 is another 13-high scoring font, slightly larger than the previous.



### Font 0x09

Font 0x09 is a 10-high double-stroke font with numbers and upper-case letters.



### Font 0x0A

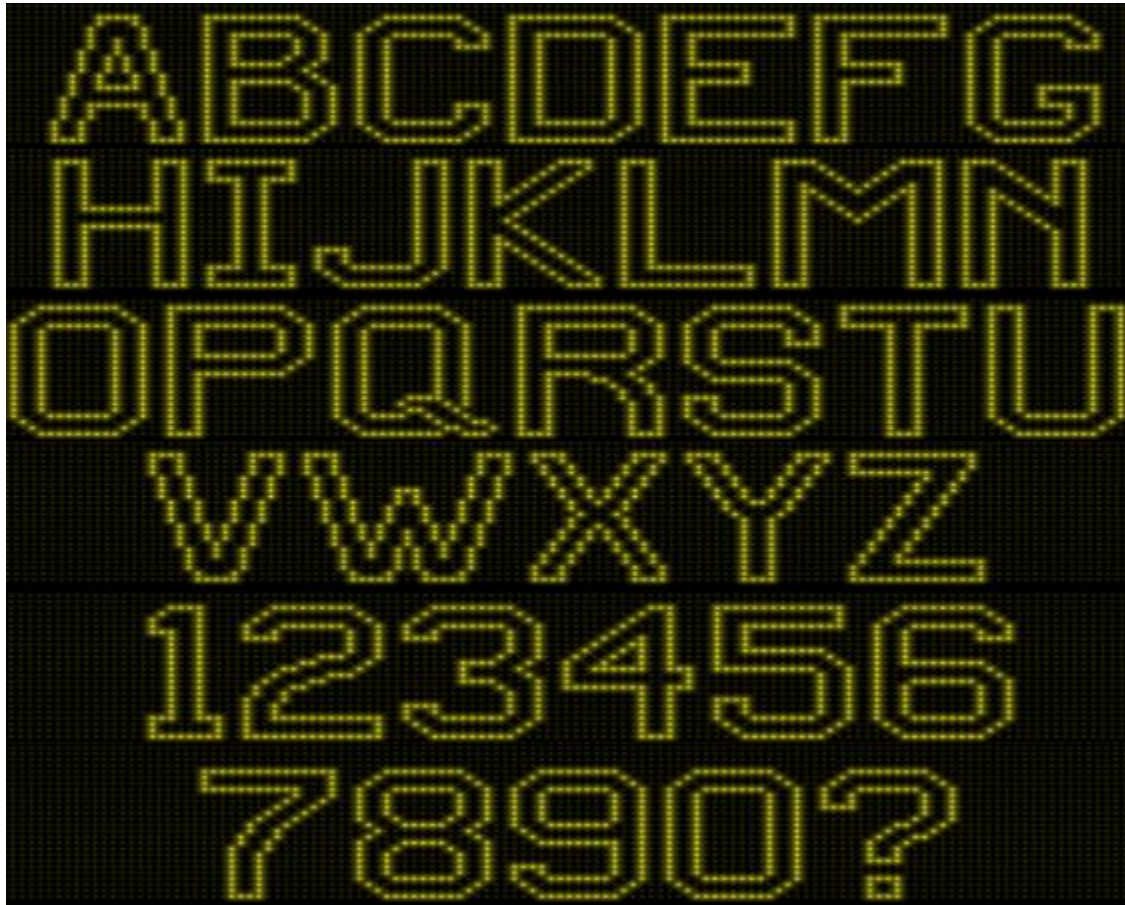
Font 0x0A is a 15-high outlined font with numbers and upper-case letters.





## Font 0x0B

Font 0x0B is a 22-high outlined font with numbers and upper-case letters.



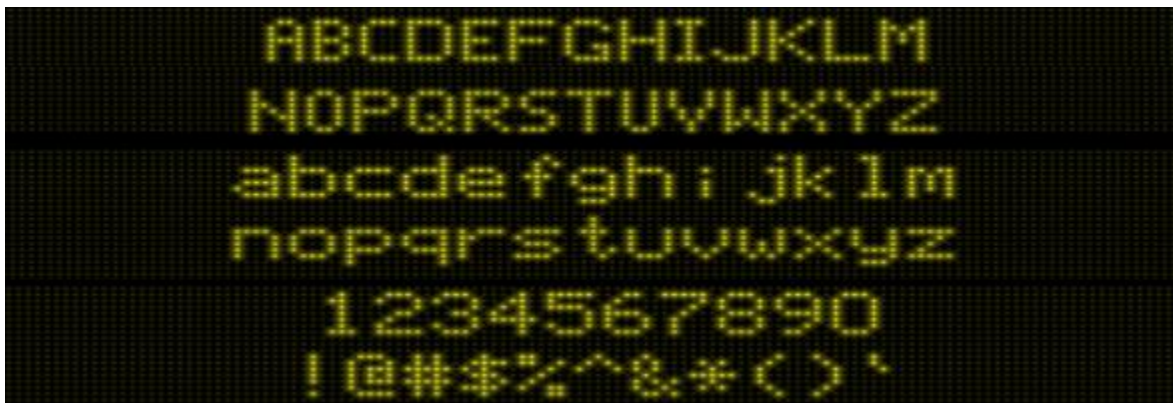
### Font 0x0C

Font 0x0C is a 26-high triple-stroke font with numbers and upper-case letters. This font also has lower-case letters of varying heights. Only a small sample is depicted below.



### Font 0x0F

Font 0x0F is a 7-high font containing upper and lower case letters and numbers.





### Font 0x10

Font 0x10 is a 15-high triple-stroke font with numbers and upper-case letters.



### Font 0x11

Font 0x11 is a 27-high thick-stroke scoring font.



### Font 0x12

Font 0x12 is a huge 32-high number-only font.



### Font 0x13

Font 0x13 is a large 25-high number-only font.



### Font 0x14

Font 0x14 is a large 20-high number-only font.



### Font 0x15

Font 0x15 is a medium 13-high number-only font.



### Font 0x16

Font 0x16 is a smaller 8-high number-only font.



### Font 0x17

Font 0x17 is a tiny 4-high font with numbers and upper-case letters.



### Font 0x18

Font 0x18 is a tiny 5-high font with numbers and upper-case letters.



### Font 0x19

Font 0x19 is a 13-high double-stroke font with numbers and upper-case letters.



## Font 0x1A

Font 0x1A is a 26-high scoring font.





## The L8.3 Feature Adjustments Additions

Originally, the L8.3 was going to re-purpose the “Fan Club” adjustment to allow various selections to pick attract mode and profanity (FUA) mode. As additional enhancements were added to select Animation Code and Lamp Driver, it became apparent that adding new “Feature Adjustments” selections would be a superior solution over a repurposed “Fan Club” adjustment.

The L-8 code was analyzed to get a better understanding of how the existing Feature Adjustments are designed and how they are stored into static (battery-backed) RAM. This analysis determined that the L-8 code is designed to accommodate 10 new Feature Adjustments without much trouble. Six of these new Feature Adjustments were utilized in L8.3, leaving room for 4 new adjustments in the future. While surveying the code, it also appears that, if needed, additional adjustments are also possible by altering the adjustment code to take advantage of the fact that 2 bytes are used for each adjustment even if the adjustment only actually uses one byte to store the adjustment setting. For future WPC ROM modifications coding technique may be exercised to store 2 adjustments in the space of one, if not enough spare/unused adjustments are available.

The following sections describe how the number of available adjustments were determined along with the necessary changes to 4 data tables that are necessary in order to add new adjustments:

- Feature Adjustments Metadata Table
- Feature Adjustments String Table, English
- Feature Adjustments String Table, German
- Feature Adjustments String Table, French

## Determining Total Number of Adjustments

Part of determining whether L-8 has ‘spare’ Feature Adjustments involves determining how many adjustments the L-8 code currently accommodates. There is a code startup function that is called which establishes the memory address of the starting point in SRAM for Feature Adjustments and it also checks the total number of adjustments against a fixed limit. This function is shown below:

```
;-----;-----;
9388: 34 36      PSHS  Y,X,B,A      ;
938A: 8E 81 EC   LDX   #$81EC      ; X gets Pointer to StandardAdjustmentsTable[]
938D: BD AC 38   JSR   $AC38      ; GetTableXEntryCountIntoY()
9390: 1F 20      TFR   Y,D          ; D has number of standard adjustments
9392: 86 02      LDA   #$02          ; A gets 2 since each adjustment is 2-bytes in ram
9394: 3D         MUL          ; D now has number of SRAM bytes of non-feature adjstmnts
9395: 8E 1B 1D   LDX   #$1B1D      ; X gets start of RAM for Standard Adjustments: $1B1D
9398: 30 8B      LEAX  D,X          ; Advance X by number bytes of non-feature adjstmnts
939A: BF 03 39   STX   $0339      ; Result into $0339. SRAM start of Feature Adjustments
939D: 1F 20      TFR   Y,D          ; Reload D with number of standard adjustments
939F: 8E 81 EF   LDX   #$81EF      ; X gets Pointer to FeatureAdjustmentsTable[]
93A2: BD AC 38   JSR   $AC38      ; GetTableXEntryCountIntoY()
93A5: 31 AB      LEAY  D,Y          ; Advance Y by D. Y has # of Std <and> Feature adjstmnts
93A7: 10 8C 00 6E CMPY  #006E      ; Compare Y with 0x006E, sanity check that there are no
; more than 0x006E (110) total adjustments:
; Standard Adjustments: 0x4E, 78 table entries
; Feature Adjustments : 0x16, 22 table entries
; Unused 16-bit RAMs :      10
```



```

93AB: 23 04      BLS  $93B1      ; If Y is lower or same as 0x006E (110), good return
93AD: BD 82 98   JSR  $8298      ; Otherwise, call this error handler function
93B0: 6C         ; with error code 0x6C to indicate adjustment error
93B1: 35 B6     ;
;-----;

```

The highlighted instruction, above, reveals that code is designed for a total of 110 adjustments. Further surveys into the other parts of the code reveals that the first part of adjustments consists of 78 adjustments while the second part accommodates 32 (22 Feature Adjustments and, apparently, room for 10 additional adjustments). The L-8 code was exhaustively investigated to ensure that these, presumed, 10 additional adjustments are not actually used by anything in L-8. This investigation results in the determination that the additional 10 adjustments are, indeed, unused in L-8. The code does not refer to the memory associated with the 10 additional adjustments, nor does run-time check of L-8 reveal any access to these additional 10 adjustments memory other than when code reads the entire block of adjustments memory when determining adjustment checksum. *Checksum is used by the code to help determine if memory corruption has taken place.*

### The L-8 Adjustments Memory Map

Although the code shown above refers to “Standard Adjustments” and “Feature Adjustments”, the game further divides the area that are referred to as “Standard Adjustments” into:

- Standard Adjustments
- HTSD Adjustments
- Pricing Adjustments
- Printer Adjustments

The WPC code utilizes various functions to fetch an adjustment value and return its setting in a 16-bit or 8-bit result register. The various functions are designed to accept an input index that is known by the caller to such function to refer to the desired adjustment index that is known by both the caller and the function. A detailed analysis of such lookup functions is left as an exercise for the reader.

The table, below, was gathered from the investigation into the L-8 Adjustments:

Overall Index	SRAM Bytes	Table-and-Index	WPC Menu Name	WPC Lookup Index
<b>0x00 (0)</b>	\$1B1D:\$1B1E	StandardAdjustment000	NULL/Placeholder	0x00
<b>0x01 (1)</b>	\$1B1F:\$1B20	StandardAdjustment001	Balls per game	0x01
<b>0x02 (2)</b>	\$1B21:\$1B22	StandardAdjustment002	Tilt warnings	0x02
<b>0x03 (3)</b>	\$1B23:\$1B24	StandardAdjustment003	Max E.B. Count	0x03
<b>0x04 (4)</b>	\$1B25:\$1B26	StandardAdjustment004	Max E.B per B.I.P.	0x04
<b>0x05 (5)</b>	\$1B27:\$1B28	StandardAdjustment005	Replay System	0x05
<b>0x06 (6)</b>	\$1B29:\$1B2A	StandardAdjustment006	Replay Percent	0x06
<b>0x07 (7)</b>	\$1B2B:\$1B2C	StandardAdjustment007	Replay Start	0x07
<b>0x08 (8)</b>	\$1B2D:\$1B2E	StandardAdjustment008	Replay Levels	0x08
<b>0x09 (9)</b>	\$1B2F:\$1B30	StandardAdjustment009	Replay Level 1	0x09
<b>0x0A (10)</b>	\$1B31:\$1B32	StandardAdjustment010	Replay Level 2	0x0A

<b>0x0B (11)</b>	\$1B33:\$1B34	StandardAdjustment011	Replay Level 3	0x0B
<b>0x0C (12)</b>	\$1B35:\$1B36	StandardAdjustment012	Replay Level 4	0x0C
<b>0x0D (13)</b>	\$1B37:\$1B38	StandardAdjustment013	Replay Boost	0x0D
<b>0x0E (14)</b>	\$1B39:\$1B3A	StandardAdjustment014	Replay Award	0x0E
<b>0x0F (15)</b>	\$1B3B:\$1B3C	StandardAdjustment015	Special Award	0x0F
<b>0x10 (16)</b>	\$1B3D:\$1B3E	StandardAdjustment016	Match Award	0x10
<b>0x11 (17)</b>	\$1B3F:\$1B40	StandardAdjustment017	Ex. Ball Ticket	0x11
<b>0x12 (18)</b>	\$1B41:\$1B42	StandardAdjustment018	Max. Ticket/Player	0x12
<b>0x13 (19)</b>	\$1B43:\$1B44	StandardAdjustment019	Match Feature	0x13
<b>0x14 (20)</b>	\$1B45:\$1B46	StandardAdjustment020	Custom Message	0x14
<b>0x15 (21)</b>	\$1B47:\$1B48	StandardAdjustment021	Language	0x15
<b>0x16 (22)</b>	\$1B49:\$1B4A	StandardAdjustment022	Clock Style	0x16
<b>0x17 (23)</b>	\$1B4B:\$1B4C	StandardAdjustment023	Date Style	0x17
<b>0x18 (24)</b>	\$1B4D:\$1B4E	StandardAdjustment024	Show Date + Time	0x18
<b>0x19 (25)</b>	\$1B4F:\$1B50	StandardAdjustment025	Allow Dim Illum.	0x19
<b>0x1A (26)</b>	\$1B51:\$1B52	StandardAdjustment026	Tournament Play	0x1A
<b>0x1B (27)</b>	\$1B53:\$1B54	StandardAdjustment027	Euro. Scr. Format	0x1B
<b>0x1C (28)</b>	\$1B55:\$1B56	StandardAdjustment028	Min. Vol. Override	0x1C
<b>0x1D (29)</b>	\$1B57:\$1B58	StandardAdjustment029	GI Power Saver	0x1D
<b>0x1E (30)</b>	\$1B59:\$1B5A	StandardAdjustment030	Power Save Level	0x1E
<b>0x1F (31)</b>	\$1B5B:\$1B5C	StandardAdjustment031	Ticket Exp. Board	0x1F
<b>0x20 (32)</b>	\$1B5D:\$1B5E	StandardAdjustment032	No Bonus Flips	0x20
<b>0x21 (33)</b>	\$1B5F:\$1B60	StandardAdjustment033	Game Re-start	0x21
<b>0x22 (34)</b>	\$1B61:\$1B62	HtsdAdjustment001	Highest Scores	0x22
<b>0x23 (35)</b>	\$1B63:\$1B64	HtsdAdjustment002	HTSD Award	0x23
<b>0x24 (36)</b>	\$1B65:\$1B66	HtsdAdjustment003	Champion Htsd	0x24
<b>0x25 (37)</b>	\$1B67:\$1B68	HtsdAdjustment004	Champion Credits	0x24
<b>0x26 (38)</b>	\$1B69:\$1B6A	HtsdAdjustment005	HTSD 1 Credits	0x26
<b>0x27 (39)</b>	\$1B6B:\$1B6C	HtsdAdjustment006	HTSD 2 Credits	0x27
<b>0x28 (40)</b>	\$1B6D:\$1B6E	HtsdAdjustment007	HTSD 3 Credits	0x28
<b>0x29 (41)</b>	\$1B6F:\$1B70	HtsdAdjustment008	HTSD 4 Credits	0x29
<b>0x2A (42)</b>	\$1B71:\$1B72	HtsdAdjustment009	HS Reset Every	0x2A
<b>0x2B (43)</b>	\$1B73:\$1B74	HtsdAdjustment010	Backup Champion	0x2B
<b>0x2C (44)</b>	\$1B75:\$1B76	HtsdAdjustment011	Backup HTSD 1	0x2C
<b>0x2D (45)</b>	\$1B77:\$1B78	HtsdAdjustment012	Backup HTSD 2	0x2D
<b>0x2E (46)</b>	\$1B79:\$1B7A	HtsdAdjustment013	Backup HTSD 3	0x2E
<b>0x2F (47)</b>	\$1B7B:\$1B7C	HtsdAdjustment014	Backup HTSD 4	0x2F
<b>0x30 (48)</b>	\$1B7D:\$1B7E	PricingAdjustment001	Game Pricing	0x30
<b>0x31 (49)</b>	\$1B7F:\$1B80	PricingAdjustment002	Left Units	0x31
<b>0x32 (50)</b>	\$1B81:\$1B82	PricingAdjustment003	Center Units	0x32
<b>0x33 (51)</b>	\$1B83:\$1B84	PricingAdjustment004	Right Units	0x33
<b>0x34 (52)</b>	\$1B85:\$1B86	PricingAdjustment005	4th Slot Units	0x34
<b>0x35 (53)</b>	\$1B87:\$1B88	PricingAdjustment006	Units/Credit	0x35
<b>0x36 (54)</b>	\$1B89:\$1B8A	PricingAdjustment007	Units/Bonus	0x36

<b>0x37 (55)</b>	\$1B8B:\$1B8C	PricingAdjustment008	Bonus Credits	0x37
<b>0x38 (56)</b>	\$1B8D:\$1B8E	PricingAdjustment009	Minimum Units	0x38
<b>0x39 (57)</b>	\$1B8F:\$1B90	PricingAdjustment010	Coin Door Type	0x39
<b>0x3A (58)</b>	\$1B91:\$1B92	PricingAdjustment011	Collection Text	0x3A
<b>0x3B (59)</b>	\$1B93:\$1B94	PricingAdjustment012	Left Slot Value	0x3B
<b>0x3C (60)</b>	\$1B95:\$1B96	PricingAdjustment013	Centr Slot Value	0x3C
<b>0x3D (61)</b>	\$1B97:\$1B98	PricingAdjustment014	Right Slot Value	0x3D
<b>0x3E (62)</b>	\$1B99:\$1B9A	PricingAdjustment015	4th Slot Value	0x3E
<b>0x3F (63)</b>	\$1B9B:\$1B9C	PricingAdjustment016	Maximum Credits	0x3F
<b>0x40 (64)</b>	\$1B9D:\$1B9E	PricingAdjustment017	Free Play	0x40
<b>0x41 (65)</b>	\$1B9F:\$1BA0	PricingAdjustment018	Hide Coin Audits	0x41
<b>0x42 (66)</b>	\$1BA1:\$1BA2	PricingAdjustment019	1-Coin Buy-in	0x42
<b>0x43 (67)</b>	\$1BA3:\$1BA4	PricingAdjustment020	Base Coin Size	0x43
<b>0x44 (68)</b>	\$1BA5:\$1BA6	PricingAdjustment021	Coin Meter Units	0x44
<b>0x45 (69)</b>	\$1BA7:\$1BA8	PricingAdjustment022	Dollar Bill Slot	0x45
<b>0x46 (70)</b>	\$1BA9:\$1BAA	PricingAdjustment023	Min. Coin Msec.	0x46
<b>0x47 (71)</b>	\$1BAB:\$1BAC	PricingAdjustment024	Slamtilt Penalty	0x47
<b>0x48 (72)</b>	\$1BAD:\$1BAE	PrinterAdjustment001	Column Width	0x48
<b>0x49 (73)</b>	\$1BAF:\$1BB0	PrinterAdjustment002	Lines Per Page	0x49
<b>0x4A (74)</b>	\$1BB1:\$1BB2	PrinterAdjustment003	Pause Every Page	0x4A
<b>0x4B (75)</b>	\$1BB3:\$1BB4	PrinterAdjustment004	Printer Type	0x4B
<b>0x4C (76)</b>	\$1BB5:\$1BB6	PrinterAdjustment005	Serial Baud Rate	0x4C
<b>0x4D (77)</b>	\$1BB7:\$1BB8	PrinterAdjustment006	Serial DTR	0x4D
<b>0x4E (78)</b>	\$1BB9:\$1BBA	FeatureAdjustment000	NULL/Placeholder	0x00
<b>0x4F (79)</b>	\$1BBB:\$1BBC	FeatureAdjustment001	Special Percent	0x01
<b>0x50 (80)</b>	\$1BBD:\$1BBE	FeatureAdjustment002	Extraball Percnt	0x02
<b>0x51 (81)</b>	\$1BBF:\$1BC0	FeatureAdjustment003	Extraball Memory	0x03
<b>0x52 (82)</b>	\$1BC1:\$1BC2	FeatureAdjustment004	Consolation Ball	0x04
<b>0x53 (83)</b>	\$1BC3:\$1BC4	FeatureAdjustment005	Drop Target Count	0x05
<b>0x54 (84)</b>	\$1BC5:\$1BC6	FeatureAdjustment006	Three Bank Count	0x06
<b>0x55 (85)</b>	\$1BC7:\$1BC8	FeatureAdjustment007	Kickback Setting	0x07
<b>0x56 (86)</b>	\$1BC9:\$1BCA	FeatureAdjustment008	Skill Shot Timer	0x08
<b>0x57 (87)</b>	\$1BCB:\$1BCC	FeatureAdjustment009	Drop Target Timer	0x09
<b>0x58 (88)</b>	\$1BCD:\$1BCE	FeatureAdjustment010	Three Bank Timer	0x0A
<b>0x59 (89)</b>	\$1BCF:\$1BD0	FeatureAdjustment011	Hurry Up Timer	0x0B
<b>0x5A (90)</b>	\$1BD1:\$1BD2	FeatureAdjustment012	Payback Timer	0x0C
<b>0x5B (91)</b>	\$1BD3:\$1BD4	FeatureAdjustment013	Jackpot Timer	0x0D
<b>0x5C (92)</b>	\$1BD5:\$1BD6	FeatureAdjustment014	Millions Plus	0x0E
<b>0x5D (93)</b>	\$1BD7:\$1BD8	FeatureAdjustment015	Timed Plunger	0x0F
<b>0x5E (94)</b>	\$1BD9:\$1BDA	FeatureAdjustment016	Attract Sounds	0x10
<b>0x5F (95)</b>	\$1BDB:\$1BDC	FeatureAdjustment017	Drt Tgt Autofire	0x11
<b>0x60 (96)</b>	\$1BDD:\$1BDE	FeatureAdjustment018	T2 Fan Club	0x12
<b>0x61 (97)</b>	\$1BDF:\$1BE0	FeatureAdjustment019	Flipper Trigger	0x13
<b>0x62 (98)</b>	\$1BE1:\$1BE2	FeatureAdjustment020	Drop Trgt. Broken	0x14

<b>0x63 (99)</b>	\$1BE3:\$1BE4	FeatureAdjustment021	DrpTrgt Dwn Mlti	0x15
<b>0x64 (100)</b>	\$1BE5:\$1BE6	FeatureAdjustment022	*Profanity	0x16
<b>0x65 (101)</b>	\$1BE7:\$1BE8	FeatureAdjustment023	*Attract Mode	0x17
<b>0x66 (102)</b>	\$1BE9:\$1BEA	FeatureAdjustment024	*Animation Code	0x18
<b>0x67 (103)</b>	\$1BEB:\$1BEC	FeatureAdjustment025	*Lamp Driver	0x19
<b>0x68 (104)</b>	\$1BED:\$1BEE	FeatureAdjustment026	*Mb Start Dt Actn	0x1A
<b>0x69 (105)</b>	\$1BEF:\$1BF0	FeatureAdjustment027	*Timed 3Bank Lamp	0x1B
<b>0x6A (106)</b>	\$1BF1:\$1BF2	FeatureAdjustment028	<unused>	0x1C
<b>0x6B (107)</b>	\$1BF3:\$1BF4	FeatureAdjustment029	<unused>	0x1D
<b>0x6D (108)</b>	\$1BF5:\$1BF6	FeatureAdjustment030	<unused>	0x1E
<b>0x6E (109)</b>	\$1BF7:\$1BF8	FeatureAdjustment031	<unused>	0x1F
<b>N/A</b>	\$1BF9:\$1BFA	<Adjustments Checksum>		

\* New adjustments in L8.3 shown for reference. Profanity was also used in the original "Profanity ROM".

The above describes how it was determined that SRAM can accommodate the extra 10 Feature Adjustments. The underlying set of functions for accessing game adjustments has been analyzed and appears to access the 10 unused elements as if they were Feature Adjustments and, as such, the 10 SRAM locations are not treated as any other type of element.

In order to utilize these 10 slots as actual game adjustments, several other areas of ROM data need to be updated to allow for the game to present the new adjustments to the user in the game adjustments menu. Such areas of ROM are further described below.

### Feature Adjustments Metadata Table

The various game-adjustment functions for Feature Adjustments all cite a common data table in the ROM which contains data about each adjustment. This is being referred to as the *Feature Adjustments Metadata Table* (where 'metadata' meaning that it is data that contains information about other data).

This table contains a number of elements that are consistent with the number of available Feature Adjustments. This means the table is made to be larger to accommodate the new adjustments. Since the table is located in ROM with other meaningful bytes immediately afterwards, for L8.3 the table is *moved* to an unused region of ROM where it can be made larger. The table below summarizes the details of the moved table:

<b>Feature Adjustments Metadata Table: Item</b>	<b>Feature Adjustments Metadata Table: Old Value</b>	<b>Feature Adjustments Metadata Table: New Value</b>
<b>Address of Table Pointer</b>	ROM: 0x781EF, WPC: \$81EF	<unchanged>
<b>Table Pointer Value (table location)</b>	ROM: 0x75680, WPC: \$5680,3D	ROM: 0x77680, WPC: \$7000,3D
<b>Table Entries</b>	0x0016 (22)	0x001C (28)
<b>Table Entry Size</b>	0x0C (12 bytes each)	<unchanged>

The new Feature Adjustments Metadata Table is shown below. The first 22 rows are identical as the original L-8 table content, however the additional items starting at the 23<sup>rd</sup> entry are new in L8.3.

```

;-----;
7000: 00 1C ; Table entries is 1C (28 entries)
7002: 0C ; Bytes per table entry is 0C (12 bytes)
7003: 00 00 00 00 00 00 00 01 00 8E BC FF ; Default value entry
700F: 00 03 00 00 00 0A 00 01 00 71 E8 3A ; Feature Adjustments, A2.01, Special Percent,
701B: 00 0A 00 00 00 23 00 01 00 71 E8 3A ; Feature Adjustments, A2.02, Extraball Percnt
7027: 00 01 00 00 00 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.03, Extraball Memory
7033: 00 01 00 00 00 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.04, Consolation Ball
703F: 00 01 00 00 00 03 00 01 00 72 79 3A ; Feature Adjustments, A2.05, Drop Trgt Count
704B: 00 02 00 00 00 04 00 01 00 72 79 3A ; Feature Adjustments, A2.06, Three Bank Count
7057: 00 02 00 00 00 04 71 E7 31 72 70 3A ; Feature Adjustments, A2.07, Kickback Setting
7063: 00 02 00 00 00 04 71 E7 31 72 70 3A ; Feature Adjustments, A2.08, Skill Shot Timer
706F: 00 0B 00 05 00 63 00 01 00 71 B2 31 ; Feature Adjustments, A2.09, Drop Targt Timer
707B: 00 0F 00 05 00 63 00 01 00 71 B2 31 ; Feature Adjustments, A2.10, Three Bank Timer
7087: 00 0F 00 07 00 63 00 01 00 71 B2 31 ; Feature Adjustments, A2.11, Hurry Up Timer
7093: 00 14 00 0A 00 63 00 01 00 71 B2 31 ; Feature Adjustments, A2.12, Payback Timer
709F: 00 0C 00 08 00 63 00 01 00 71 B2 31 ; Feature Adjustments, A2.13, Jackpot Timer
70AB: 00 01 00 00 00 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.14, Millions Plus
70B7: 00 00 00 00 00 78 00 01 00 71 C7 31 ; Feature Adjustments, A2.15, Timed Plunger
70C3: 00 00 00 00 00 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.16, Attract Sounds
70CF: 00 00 00 00 00 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.17, Drp Trg Autofire
70DB: 00 01 00 00 00 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.18, T2 FAN CLUB
70E7: 00 00 00 00 00 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.19, Flipper Trigger
70F3: 00 00 00 00 00 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.20, Drop Trgt. Broken
70FF: 00 00 00 00 00 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.21, Drprtrgt Dwn Mlti
; NEW ADJUSTMENT METADATA BELOW
710B: 00 00 00 00 00 01 74 13 3A 72 70 3A ; Feature Adjustments, A2.22, Profanity
7117: 00 02 00 00 00 02 00 01 00 65 7C 3D ; Feature Adjustments, A2.23, Attract Mode
7123: 00 01 00 00 00 01 00 01 00 65 C5 3D ; Feature Adjustments, A2.24, Animation Code
712F: 00 00 00 00 00 01 00 01 00 65 F3 3D ; Feature Adjustments, A2.25, Lamp Driver
713B: 00 00 00 00 00 05 00 01 00 66 54 3D ; Feature Adjustments, A2.26, MB Start DT Action
7147: 00 00 00 00 00 01 00 01 00 65 3D 3D ; Feature Adjustments, A2.27, Timed 3Bank Lamp
7153: 00 00 00 00 00 00 00 01 00 8E BC FF ; Feature Adjustments, A2.28, <placeholder>
715F: 00 00 00 00 00 00 00 01 00 8E BC FF ; Feature Adjustments, A2.29, <placeholder>
716B: 00 00 00 00 00 00 00 01 00 8E BC FF ; Feature Adjustments, A2.30, <placeholder>
7177: 00 00 00 00 00 00 00 01 00 8E BC FF ; Feature Adjustments, A2.31, <placeholder>
;-----;

```

As shown in the new metadata table, above, the table first 2 bytes report that it has 28 entries however the actual content of the table has placeholders for 4 additional entries at the end. This makes it easier to add additional adjustments in the future as the first 2 bytes in the table indicating number of entries would need increased and the “<placeholder>” row data would need updated for the new adjustments.

Each 12-byte metadata table entry contains information for each adjustment. A breakdown of the 12 bytes is as follows:

Adjustment Metadata Table Entry Bytes [0..11]	Description
Bytes [0..1], making a 16-bit value	Default Adjustment value used by factory reset and compared during menu display so the “factory setting” indicator can be reported.
Bytes [2..3], making a 16-bit value	Minimum Adjustment Value
Bytes [4..5], making a 16-bit value	Maximum Adjustment Value
Bytes [6, 7, 8] Adjustment value selector metadata	Value depends on the next 3 bytes. When next 3 bytes is a string look up function (\$7270,3A in L-8) these 3 bytes contain WPC

	address of a table containing string index number for each of the possible adjustment values. In other cases, these 3 bytes contain the increment value when plus/minus are used.
<b>Bytes [9, 10, 11] making a WPC Address which is the Adjustment value selector function</b>	Address of function that gets called to handle the display of the current adjustment value. The previous 3 bytes are used in conjunction with this address.

Below is a summary of each of the new adjustments. The information below helps illustrate how the adjustments metadata 12-byte entry is used.

### Feature Adjustment: Profanity

This adjustment data is modeled after the metadata for this adjustment used in the original “Profanity ROM”. The 12-byte metadata table for this new adjustment is as follows:

Adjustment Metadata Table Entry Bytes [0..11]	Value / Description
00 00	Default Adjustment value 0
00 00	Minimum Adjustment value 0
00 01	Maximum Adjustment value 1
74 13 3A	Address of String table used by the \$7270,3A string selector.
72 70 3A	Common String Selector Function

For “Profanity” adjustment, the table shows 2 possible values 0, and 1. Displayed strings for each of these numeric values is derived by a common selector function \$7270,3A which will cite table at \$7413,3A. Shown below is the content of the string table (at ROM offset 0x6B413):

```
-----;-----
7413: 80 F2          ; String index corresponding to "OFF"
7415: 80 F3          ; String index corresponding to "ON"
-----;-----
```

The bytes at \$7413,3A contain string index numbers that correspond to the “OFF” and “ON” string, respectively. The \$7270,3A function will only access one of these 2 strings based on the fact that the other metadata table content defines possible values of 0 and 1. *The common WPC string lookup functions in L-8 will convert the 0x80F2 and 0x80F3 into an appropriate lookup into a string table where “OFF” and “ON” are stored. Such string lookup logic is outside the scope of this analysis but it is worth noting that the high bit is an indicator of which table to cite, and the remaining bits are the index into such table.*

### Feature Adjustment: Attract Mode

This adjustment data is defined to show the possible values for “Attract Mode” adjustment.

Adjustment Metadata Table Entry Bytes [0..11]	Value / Description
00 02	Default Adjustment value 2 (corresponding to L8.3)



00 00	Minimum Adjustment value 0 (L8.1)
00 02	Maximum Adjustment value 2 (L8.3)
00 01 00	Value used for metadata when custom Selector function is used.
65 7C 3D	Custom String Selector Function for "Attract Mode"

The value 00 01 00 is used in conjunction with the \$657C,3D Custom String Selector function. Current understanding is that this ensures numeric values are incremented and decremented by 1 when values are cycled and such numeric value is compared against min/max and such value is passed into the custom string selector function. Further code analysis is needed for added certainty.

The \$657C,3D function (ROM offset 0x7657C) is shown below for reference and guidance on how a custom adjustment string selector function can be defined. Its helper function at \$6571,3D is also shown.

```

-----
;
;
; AdjustmentWriteASCIIValueCclear()
6571: C1 02    CMPB    #$02    ;
6573: 26 04    BNE     $6579    ;
6575: 1C FE    ANDCC   #$FE     ;
6577: 20 02    BRA     $657B    ;
6579: 1A 01    ORCC   #$01     ;
657B: 39          RTS          ;
;
-----
;
; "Attract Mode" adjustment selection text function
;
; A has adjustment index
; B has code. 02 = write the adj value ASCII string at pointer Y
; U has the current value
; Y has address of where to put ASCII string (when B is 02)
;
657C: 8D F3    BSR     $6571    ; AdjustmentWriteASCIIValueCclear() // helper function
657E: 24 01    BCC     $6581    ; If C-bit is clear, need to wring ASCII string into Y
6580: 39          RTS          ; If C-bit is set, RTS now. Do not write ASCII string to Y.
6581: 34 50    PSHS   U,X      ;
6583: 8D 05    BSR     $658A    ; Checks U and returns with X set to desired string addr
6585: BD B9 51    JSR     $B951    ; CopyASCIIStringFromXtoYandVerifyLength()
658A: 35 D0    PULS   X,U,PC   ;
;
-----
;
;
658A: 8E 65 AB    LDX    #$65AB    ; Load default/error string
658D: 11 83 00 00  CMPU   #0000    ; Check if at "L8.1"
6591: 26 04    BNE     $6597    ;
6593: 8E 65 B6    LDX    #$65B6    ;
6596: 39          RTS          ;
6597: 11 83 00 01  CMPU   #0001    ; Check if at "L8.2"
659B: 26 04    BNE     $65A1    ;
659D: 8E 65 BB    LDX    #$65BB    ;
65A0: 39          RTS          ;
65A1: 11 83 00 02  CMPU   #0002    ; Check if at "L8.3"

```

```

65A5: 26 03      BNE  $65AA ;
65A7: 8E 65 C0   LDX  #$65C0 ;
65AA: 39          RTS    ;
;
-----
;
65AB: 4D 45 4E 55 20 45 52 52 4F 52 00 ; "MENU ERROR"
65B6: 4C 38 2E 31 00      ; "L8.1"
65BB: 4C 38 2E 32 00      ; "L8.2"
65C0: 4C 38 2E 33 00      ; "L8.3"
;
-----
;

```

The string selector function for “Attract mode”, above, is fairly straightforward, writing the ASCII string corresponding to the current index number into address pointed to by Y. This new function directly contains the new strings “L8.1”, “L8.2”, etc so that the standard WPC string table doesn’t need to be modified to provide a desired string for a given (new) string index value.

### Feature Adjustment: Remaining New Adjustments

The remainder of the new adjustments for “Animation Code”, “Lamp Driver”, MB Start DT Action” and “Timed 3Bank Lamp” are constructed similar to the “Attract Mode” adjustment, described above. Each of these new adjustments have their own string selector function which has similar logic to that shown above. For reference, the address of these string selectors is listed below. A disassembly of each of these functions is left as an exercise for the reader.

New Adjustment	Custom String Selector Function
Animation Code	WPC: \$65C5,3D ROM: 0x765C5
Lamp Driver	WPC: \$65F3,3D ROM: 0x765F3
MB Start DT Action	WPC: \$6654,3D ROM: 0x76654
Timed 3Bank Lamp	WPC: \$653D,3D ROM: 0x7653D

### Feature Adjustments String Tables

The previous text described how the Feature Adjustments Metadata Table needs to be increased in size to accommodate new feature adjustments. In order for the remainder of the WPC code to operate properly with the added adjustments, there are 3 additional tables that need similar treatment, increasing their total entry count by the number of new adjustments. For L8.3 these tables were increased from 0x0016 (22) to 0x001C (28) entries.

The three additional tables are as follows:

- Feature Adjustments String Table, English
- Feature Adjustments String Table, German
- Feature Adjustments String Table, French

These string tables are used when the new feature adjustment is shown during the adjustments menu. For each of the 3 possible language selections, the appropriate table is used to display the adjustment

string in the desired language. *In coding the term “string” generally refers to a series of displayable character bytes followed by 0x00 byte to signify the end of the string.*

Similar to the Metadata table, in order to add new entries to these tables, the old table is relocated to unused region of ROM so that new data can be added to the end of each table. The table, below, summarizes the addresses and information about each of these tables.

Table: Item	Old Value	New Value
Address of Table Pointer: English	ROM: 0x78261, WPC: \$8261	<unchanged>
Address of Table Pointer: German	ROM: 0x78264, WPC: \$8264	<unchanged>
Address of Table Pointer: French	ROM: 0x78267, WPC: \$8267	<unchanged>
Table Pointer Value: English	ROM: 0x400D5, WPC: \$40D5,30	ROM: 0x76700, WPC: \$6700,3D
Table Pointer Value: German	ROM: 0x404AB, WPC: \$44AB,30	ROM: 0x76A00, WPC: \$6A00,3D
Table Pointer Value: French	ROM: 0x407EA, WPC: \$47EA,30	ROM: 0x76D00, WPC: \$6D00,3D
Table Entries: English	0x0016 (22)	0x001C (28)
Table Entries: German	0x0016 (22)	0x001C (28)
Table Entries: French	0x0016 (22)	0x001C (28)
Table Entry Size: English	0x02 (2 bytes each)	<unchanged>
Table Entry Size: German	0x02 (2 bytes each)	<unchanged>
Table Entry Size: French	0x02 (2 bytes each)	<unchanged>

As indicated, the string tables contain a 2-byte entry for each entry. The 2 bytes for each entry are a WPC address corresponding to the first character/byte of the indexed string. Every WPC address in the table corresponds to the location of the string located in the same bank as the table itself. *This concept applies to a large number of string tables in WPC software.*

Since the string table contains a list of WPC addresses to strings in the same bank, and since the string tables were moved from bank \$30 to bank \$3D, this necessarily means that all of the actual strings for English, German, and French Feature Adjustments were also moved along with these three tables. *It was necessary to move these tables from bank \$30 to bank \$3D due to lack of unused ROM bytes in the \$30 bank.*

Since the table and the WPC addresses of the strings were moved to a new bank, all of the data in the new table is subject to change due to the new addresses where the strings were placed in the new bank.

Depicted below is the new English strings table and the new location of its strings. Similar treatment to the German and French string tables for Feature Adjustments was also performed. Disassembly of German and French tables is left as an exercise to the reader.

```
-----;-----
6700: 00 1C          ; Number of entries in this table is 0x1B, or 27
6702: 02            ; Each table entry is 2-bytes. See below.
;
6703: 67 43          ; Feature Adjustments, English, String000, "NULL"
6705: 67 48          ; Feature Adjustments, English, String001, "SPECIAL PERCENT"
6707: 67 58          ; Feature Adjustments, English, String002, "EXTRABALL PERCNT"
```

```

6709: 67 69 ; Feature Adjustments, English, String003, "EXTRABALL MEMORY"
670B: 67 7A ; Feature Adjustments, English, String004, "CONSOLATION BALL" (*)
670D: 67 8B ; Feature Adjustments, English, String005, "DROP TARGET COUNT"
670F: 67 9C ; Feature Adjustments, English, String006, "THREE BANK COUNT"
6711: 67 AD ; Feature Adjustments, English, String007, "KICKBACK SETTING"
6713: 67 BE ; Feature Adjustments, English, String008, "SKILL SHOT TIMER"
6715: 67 CF ; Feature Adjustments, English, String009, "DROP TARGET TIMER"
6717: 67 E0 ; Feature Adjustments, English, String010, "THREE BANK TIMER"
6719: 67 F1 ; Feature Adjustments, English, String011, "HURRY UP TIMER"
671B: 68 00 ; Feature Adjustments, English, String012, "PAYBACK TIMER"
671D: 68 0E ; Feature Adjustments, English, String013, "JACKPOT TIMER"
671F: 68 1C ; Feature Adjustments, English, String014, "MILLIONS PLUS" (*)
6721: 68 2A ; Feature Adjustments, English, String015, "TIMED PLUNGER"
6723: 68 38 ; Feature Adjustments, English, String016, "ATTRACT SOUNDS"
6725: 68 47 ; Feature Adjustments, English, String017, "DRP TGT AUTOFIRE" (*)
6727: 68 58 ; Feature Adjustments, English, String018, "T2 FAN CLUB" (*)
6729: 68 64 ; Feature Adjustments, English, String019, "FLIPPER TRIGGER" (*)
672B: 68 74 ; Feature Adjustments, English, String020, "DROP TRGT. BROKEN" (*)
672D: 68 86 ; Feature Adjustments, English, String021, "DRPTRGT DWN MLTI" (*)
672F: 68 97 ; Feature Adjustments, English, String022, "PROFANITY"
6731: 68 A1 ; Feature Adjustments, English, String023, "ATTRACT MODE"
6733: 68 AE ; Feature Adjustments, English, String024, "ANIMATION CODE"
6735: 68 BD ; Feature Adjustments, English, String025, "LAMP DRIVER"
6737: 68 C9 ; Feature Adjustments, English, String026, "MB START DT ACTN"
6739: 68 DA ; Feature Adjustments, English, String027, "TIMED 3BANK LAMP"
673B: 67 43 ; Feature Adjustments, English, String028, <placeholder>
673D: 67 43 ; Feature Adjustments, English, String029, <placeholder>
673F: 67 43 ; Feature Adjustments, English, String02A, <placeholder>
6741: 67 43 ; Feature Adjustments, English, String02B, <placeholder>
;
6743: 4E 55 4C 4C 00 ; Feature Adjustments, English, String001, "NULL"
6748: 53 50 45 43 49 41 4C 20 ; Feature Adjustments, English, String001, "SPECIAL PERCENT"
6750: 50 45 52 43 45 4E 54 00 ;
6758: 45 58 54 52 41 42 41 4C ; Feature Adjustments, English, String002, "EXTRABALL PERCNT"
6760: 4C 20 50 45 52 43 4E 54 ;
6768: 00 ;
6769: 45 58 54 52 41 42 41 4C ; Feature Adjustments, English, String003, "EXTRABALL MEMORY"
6771: 4C 20 4D 45 4D 4F 52 59 ;
6779: 00 ;
677A: 43 4F 4E 53 4F 4C 41 54 ; Feature Adjustments, English, String004, "CONSOLATION BALL"
6782: 49 4F 4E 20 42 41 4C 4C ;
678A: 00 ;
678B: 44 52 4F 50 20 54 41 52 ; Feature Adjustments, English, String005, "DROP TARGET COUNT"
6793: 47 54 20 43 4F 55 4E 54 ;
679B: 00 ;
679C: 54 48 52 45 45 20 42 41 ; Feature Adjustments, English, String006, "THREE BANK COUNT"
67A4: 4E 4B 20 43 4F 55 4E 54 ;
67AC: 00 ;
67AD: 4B 49 43 4B 42 41 43 4B ; Feature Adjustments, English, String007, "KICKBACK SETTING"
67B5: 20 53 45 54 54 49 4E 47 ;
67BD: 00 ;
67BE: 53 4B 49 4C 4C 20 53 48 ; Feature Adjustments, English, String008, "SKILL SHOT TIMER"
67C6: 4F 54 20 54 49 4D 45 52 ;
67CE: 00 ;
67CF: 44 52 4F 50 20 54 41 52 ; Feature Adjustments, English, String009, "DROP TARGET TIMER"
67D7: 47 54 20 54 49 4D 45 52 ;
67DF: 00 ;
67E0: 54 48 52 45 45 20 42 41 ; Feature Adjustments, English, String010, "THREE BANK TIMER"
67E8: 4E 4B 20 54 49 4D 45 52 ;
67F0: 00 ;
67F1: 48 55 52 52 59 20 55 50 ; Feature Adjustments, English, String011, "HURRY UP TIMER"
67F9: 20 54 49 4D 45 52 00 ;

```

```

6800: 50 41 59 42 41 43 4B 20 ; Feature Adjustments, English, String012, "PAYBACK TIMER"
6808: 54 49 4D 45 52 00 ;
680E: 4A 41 43 4B 50 4F 54 20 ; Feature Adjustments, English, String013, "JACKPOT TIMER"
6816: 54 49 4D 45 52 00 ;
681C: 4D 49 4C 4C 49 4F 4E 53 ; Feature Adjustments, English, String014, "MILLIONS PLUS"
6824: 20 50 4C 55 53 00 ;
682A: 54 49 4D 45 44 20 50 4C ; Feature Adjustments, English, String015, "TIMED PLUNGER"
6832: 55 4E 47 45 52 00 ;
6838: 41 54 54 52 41 43 54 20 ; Feature Adjustments, English, String016, "ATTRACT SOUNDS"
6840: 53 4F 55 4E 44 53 00 ;
6847: 44 52 50 20 54 47 54 20 ; Feature Adjustments, English, String017, "DRP TGT AUTOFIRE"
684F: 41 55 54 4F 46 49 52 45 ;
6857: 00 ;
6858: 54 32 20 46 41 4E 20 43 ; Feature Adjustments, English, String018, "T2 FAN CLUB"
6860: 4C 55 42 00 ;
6864: 46 4C 49 50 50 45 52 20 ; Feature Adjustments, English, String019, "FLIPPER TRIGGER"
686C: 54 52 49 47 47 45 52 00 ;
6874: 44 52 4F 50 20 54 52 47 ; Feature Adjustments, English, String020, "DROP TRGT. BROKEN"
687C: 54 2E 20 42 52 4F 4B 45 ;
6884: 4E 00 ;
6886: 44 52 50 54 52 47 54 20 ; Feature Adjustments, English, String021, "DRPTRGT DWN MLTI"
688E: 44 57 4E 20 4D 4C 54 49 ;
6896: 00 ;
6897: 50 52 4F 46 41 4E 49 54 ; Feature Adjustments, English, String022, "PROFANITY"
689F: 59 00 ;
68A1: 41 54 54 52 41 43 54 20 ; Feature Adjustments, English, String023, "ATTRACT MODE"
68A9: 4D 4F 44 45 00 ;
68AE: 41 4E 49 4D 41 54 49 4F ; Feature Adjustments, English, String024, "ANIMATION CODE"
68B6: 4E 20 43 4F 44 45 00 ;
68BD: 4C 41 4D 50 20 44 52 49 ; Feature Adjustments, English, String025, "LAMP DRIVER"
68C5: 56 45 52 00 ;
68C9: 4D 42 20 53 54 41 52 54 ; Feature Adjustments, English, String026, "MB START DT ACTN"
68D1: 20 44 54 20 41 43 54 4E ;
68D9: 00 ;
68DA: 54 49 4D 45 44 20 33 42 ; Feature Adjustments, English, String027, "TIMED 3BANK LAMP"
68E2: 41 4E 4B 20 4C 41 4D 50 ;
68EA: 00 ;
-----;-----

```

(\*)Note that this table, above, contains some entries marked with (\*). These entries correspond to strings that were found to be cited by other tables in the original \$30 bank. For example, the "Consolidation Ball" string is also used by the "Audits" string table in bank \$30. Other strings are, for example, shared by the German and French tables. In the original location of this table, bank \$30, although the table of 2-byte addresses can be moved (and removed), the strings themselves need to be retained in their original location in ROM since they are referenced by other tables in bank \$30. This means it is important, in general, to closely examine the ROM and/or retain all old strings in the event that their WPC address is cited in other string tables in the same bank. For T2 L-8 the entire bank \$30 was disassembled in order to determine how strings are shared between different tables.

Referencing the table, above, it is evident how string lookup is performed. Consider the example where the string for "TIMED 3BANK LAMP" is needed. A lookup is performed on index 0x001B (27) from which the table returns the 2-byte entry at \$6739 containing the 2 bytes 0x68 0xDA. These 2 bytes form the WPC address \$68DA which is where the starting byte of the string is found, above. The string is read until the ending 0x00 byte is reached.

The table also includes extra placeholders for the 4 unused Feature Adjustments. If new adjustments are enabled, then the first 2 bytes of the table are increased and these extra placeholder bytes are updated to contain the WPC addresses of the new strings containing the names of the new Feature Adjustments located in the same bank as the table.

## The L8.3 Text String Corrections

In L8.3 a large focus on the German text strings was performed. Correction to German text strings was extensively performed to improve the game play experience for T2 owners operating their machine with the language adjustment set to German.

*As mentioned earlier, in coding “string” generally refers to a series of displayable character bytes with a 0x00 byte signifying the end of the string.*

There are two types of string corrections:

- The new string consists of same or fewer number of characters as the old string, and
- The new string consists of more characters as the old string.

In cases where same or fewer characters are in the new string, it is fairly evident to most observers that the old string simply needs to be overwritten with the new string in the ROM. After the last character, the 0x00 byte is appended to indicate the end of the string.

In cases where more characters are needed to accommodate the new string, the new string needs to be placed in unused region in the same ROM bank as the old string and the pointer to the old string needs to be updated to the address of the new string. Refer to the previous section describing the Feature Adjustments string table for a depiction of a string table and description of how the table contains addresses to the first character of the string located in same bank as the string table itself.

*For these string changes there is only a single pointer that cites each string. When making changes to string pointers consideration needs to be done to see if multiple pointers need updated. For example, referring to Feature Adjustments string table, if “Consolidation Ball” string was changed to something larger, then the English Feature Adjustments string table pointer would need updated and also the English Audits string table would need to have its pointer updated as well since, as mentioned, that same string is cited by multiple tables.*

Below is a summary of text corrections in L8.3 with indicators of new string addresses, in cases where new string was made larger than the old string it replaces.

Old String	New String	Old String Address	String Move Info
“ZIET”	“ZEIT”	0x40576, \$4576,30	<not moved>
“CHEK POINT”	“CHECKPOINT”	0x41A9E, \$5A9E,30	<not moved>
“FREIS SPIEL”	“FREISPIEL”	0x62CD9, \$6CD9,38	<not moved>
“NETHERLND”	“HOLLAND”	0x70CA7, \$4CA7,3C	<not moved>



"MITWOCH"	"MITTWOCH"	0x71D1C, \$5D1C,3C	0x73F9B, \$7F9B,3C new location of string 0x7192A, \$592A,3C was 5D 1C, now 7F 9B
"SANTAG"	"SAMSTAG"	0x71D37, \$5D37,3C	0x73FA4, \$7FA4,3C new location of string 0x71930, \$5930,3C was 5D 37, now 7F A4
"GELDESCHT"	"GELOESCHT"	0x71D85, \$5D85,3C	<not moved>
"HAUPT MENUE"	"HAUPTMENUE"	0x71E79, \$5E79,3C	<not moved>
"MEUE "	"NEUE "	0x721A1, \$61A1,3C	<not moved>
"PUNCTE"	"PUNKTE"	0x72285, \$6285,3C	<not moved>
"HAUPT MENUE"	"HAUPTMENUE"	0x7229F, \$629F,3C	<not moved>
"HILPE"	"HILFE"	0x722D7, \$62D7,3C	<not moved>
"DEUTSCE "	"DEUTSCH "	0x7248B, \$648B,3C	<not moved>
"FREIS SPIEL"	"FREISPIEL"	0x72500, \$6500,3C	<not moved>
"Ziehe"	"Zielen Und Den"	0x41AEE, \$5AEE,30	0x41B4F, \$5B4F,30 new location of string 0x41864, \$5864,30 was 5A EE, now 5B 4F
"Vor Abschuss"	"Abzug Ziehen"	0x41AF4, \$5AF4,30	0x400F0, \$40F0,30 new location of string 0x41866, \$5866,30 was 5A F4, now 40 F0
"START DRUCKEN"	"DRUECKE START"	0x41C00, \$5C00,30	<not moved>
"KITBACK BEL."	"KICKBACK BEL."	0x41B36, \$5B36,30	<not moved>
"KITBACK BE"	"KICKBACK B"	0x41B43, \$5B43,30	0x41B44, \$5B44,30 new location of string 0x4187E, \$587E,30 was 5B 43, now 5B 44
"KITBACK "	"KICKBAC"	0x41B4E, \$5B4E,30	0x41103, \$5103,30 new location of string 0x41880, \$5880,30 was 5B 4E
"KITBA"	"KICKBA"	0x41B58, \$5B58,30	0x4110B, \$510B,30 new location of string 0x41882, \$5882,30 was 5B 58
"KITB"	"KICKB"	0x41B5E, \$5B5E,30	0x41112, \$5112,30 new location of string 0x41884, \$5884,30 was 5B 5E
"KIT"	"KICK"	0x41B63, \$5B63,30	0x41118, \$5118,30 new location of string 0x41886, \$5886,30 was 5B 63
"KI"	"KIC"	0x41B67, \$5B67,30	0x4111D, \$511D,30 new location of string 0x41888, \$5888,30 was 5B 67
"K"	"K"	0x4166A, \$566A,30	<no change, shown here for completeness>
"MIT FLIPPER TASTEN"	"MIT FLIPPERTASTERN"	0x41DB7, \$5DB7,30	<not moved>
"BILD WECHSELN"	"DAS VISIER BEWEGEN"	0x41DCA, \$5DCA,30	0x41AEE, \$5AEE,30 new location of string 0x41952, \$5952,30 was 5D CA

A careful examination of the ROM changes for text corrections will reveal that in some cases, the old location of moved strings is being reused as the new location for other moved strings. This demonstrates the flexibility of using the WPC string pointer tables.

It should also be noted that in cases where strings are being made longer, careful examination of the longer strings during game play (or attract mode) should be made to ensure the longer string will display properly. If the longer string doesn't appear properly it is then possible to alter the font that the code uses for displaying the string or alter the placement of the string on the display. For L8.3 all new strings

have been tested for proper display and such modifications are not needed and, therefore, such ROM modification is not depicted here.

## The L8.3 Sound Test Updates

During the L8.3 development, some attention was given to the WPC “Sound Test”. ROM changes related to the sound test are described below.

### Sound Test Update: Sound 05 Playing Unexpectedly

Initially a problem was reported with a 3<sup>rd</sup> party sound board and with how the “Running” sound test behaves as it cycles past sound “05 Database Backgr.” On some 3<sup>rd</sup> party sound boards, the sound from 05 continues to play, unexpectedly, as the “running” test cycles to sound 06, and 07, etc. The correct behavior is that sound 05 stops as the test cycles to sound 06 (100K Award sound).

It was found that the 3<sup>rd</sup> party sound package classified the sound for “05 Database Backgr.” As a “jingle” and not “music” which may be the reason that the sound behaves this way during the WPC Sound Test.

### Sound 05 Classification

The L-8 ROM content was examined for possible reasons for the 05 sound behaving differently on 3<sup>rd</sup> party boards. An oddity in the sound test table was found. The sound test table is shown below:

```
-----;-----
490D: 00 0B      ; Entries
490F: 03         ; Entry size
                ;
4910: 00 00 00   ; Null
4913: 03 00 00   ; 0x03 == Main play
4916: 04 00 00   ; 0x04 == Get jackpot tune
4919: 07 00 00   ; 0x07 == M.Ball lit tune
491C: 14 00 00   ; 0x14 == Video mode tune
491F: B2 00 60   ; 0xB2 == Database Backgr.
4922: BF 00 60   ; 0xBF == 100K Award
4925: C6 00 60   ; 0xC6 == Alarm Sound
4928: 25 01 60   ; 0x25 == "Get the cpu"
492B: 27 01 60   ; 0x27 == "Autofire"
492E: 28 01 60   ; 0x28 == "Video mode"
-----;-----
```

As shown, the sound test table contains an entry for each of the sounds that are played during the “Sound Test” mode. Each table entry consists of 3 bytes which, in summary, are shown below.

Sound Test Table Byte	Description
Byte[0]	Sound byte value used in the sound call command to sound board.
Byte[1]	Flag byte indicating whether sound is a voice callout. When non-zero, the sound call command includes a 0x7A byte prior to the sound index byte.
Byte[2]	Timer value used during sound test. <ul style="list-style-type: none"> <li>During “repeat” mode: <ul style="list-style-type: none"> <li>Non-zero value defines how long the sound is allowed to play before being re-queued (0x60 is 1.5 seconds).</li> </ul> </li> </ul>

- Value 0x00, sound plays indefinitely (such as for music).
- During “running” mode:
  - Non-zero value is used as the period the sound is allowed to play before advancing to next sound (0x60 is 1.5 seconds).
  - Value 0x00 causes a time period of 0xB4 (just under 3 seconds) to be used before next sound is played.

For completeness, below are the sound-call commands sent to the sound board when the sound test wishes to play the current sound:

Sound Test	Sound Table Byte	Command Sent to Sound Board
Main Play	0x03	0x7E 0x7D 0x7F 0x03
Get Jackpot Tune	0x04	0x7E 0x7D 0x7F 0x04
M.Ball Lit Tune	0x07	0x7E 0x7D 0x7F 0x07
Video Mode Tune	0x14	0x7E 0x7D 0x7F 0x14
Database Backgr.	0xB2	0x7E 0x7D 0x7F 0xB2
100K Awark	0xBF	0x7E 0x7D 0x7F 0xBF
Alarm Sound	0xC6	0x7E 0x7D 0x7F 0xC6
“Get the CPU”	0x25	0x7E 0x7D 0x7F 0x7A 0x25
“Autofire”	0x27	0x7E 0x7D 0x7F 0x7A 0x27
“Video mode”	0x28	0x7E 0x7D 0x7F 0x7A 0x28

With focus on the sound 05, another look at the Sound Table entry for sound 05 is in order:

```
-----;-----
491F: B2 00 60 ; 0xB2 == Database Backgr.
-----;-----
```

During L8.3 this 3-byte entry for sound 05 came to be in question. The 3<sup>rd</sup> byte having value 0x60, as mentioned, means that:

- During “running” test, the test will advance to next sound after 0x60 (1.5 seconds), and
- During “repeat” test, the sound will be re-queued to the sound board every 1.5 seconds.
  - It may be unnoticeable that on L-8 that this sound is restarted every 1.5 seconds during “repeat” test mode, depending on the sound board in use.

It may be the original T2 software design specifically determined this behavior however for L8.3 a change was made as part of effort to help resolve the behavior of the sound test on 3<sup>rd</sup> party sound boards. The sound test table entry was changed to have the following contents:

```
-----;-----
491F: B2 00 00 ; 0xB2 == Database Backgr.
-----;-----
```

*Note: The sound test table gets relocated in L8.3, refer to “Relocated Sound Test Table”, later in this document for actual address where the above change to this sound table entry is made.*

Using value 0x00 as the 3<sup>rd</sup> byte changes the behavior so that:

- During “running” test, the test will advance to next sound after 3 seconds (period of 0xB4), and
- During “repeat” test, the sound will not be re-queued every 1.5 seconds.

### Sound 05 Explicit Stop

During the development of L8.3, as part of investigation efforts into the nature of sound 05 on 3<sup>rd</sup> party sound boards, a change was put in place for beta testing which altered the command sent to the sound board when the sound test advances from sound 05 to sound 06. This modification added an extra “stop” command to help ensure the sound 05 is no longer playing when sound transitions to 06. This is the same “stop” command used when sound test is exited.

As it turns out, this change to sound test code ended up remaining in final L8.3 image although it was originally expected to not be part of L8.3. This minor oversight is being documented here for completeness, transparency and to give some information for hobbyists to do further experiments and, possibly, remove this code if needed. Any future T2 ROM revision may be subject to having this code removed.

For this code change, consider this function that sound test code calls whenever advancing to the next sound (regardless of whether from ‘running’ test mode or ‘plus’ coin-door button).

```
-----;-----  
;   
; AdvanceNextSoundIndex()  
;   Either via plus button or during 'running'  
6D87: 34 02      PSHS  A      ;   
6D89: 8D 24      BSR   $6DAF   ; SoundTestTableEntryCountGetIntoA()  
6D8B: 6C 41      INC   $0001,U ; Increment sound test index  
6D8D: A1 41      CMPA  $0001,U ;   
6D8F: 22 04      BHI   $6D95   ;   
6D91: 86 01      LDA   #$01    ; Reset index to #1  
6D93: A7 41      STA   $0001,U ;   
6D95: BD 6D F3   JSR   $6DF3   ; StopCurrentSound()  
6D98: 81 02      CMPA  #$02    ;   
6D9A: 35 82      PULS  A,PC    ;   
-----;-----
```

This code, above, checks if the sound index needs to wrap back to 1 (in case where the sound advances past the last sound test (“Video mode”) and then calls a function that is intended to stop the current sound. As the problem with 05 (on some 3<sup>rd</sup> party sound boards) is that the sound 05 doesn’t actually stop, the function was augmented to call a different ‘stop’ when the sound is advancing from 05 to 06.

```
-----;-----  
;   
; AdvanceNextSoundIndex()  
;   Either via plus button or during 'running'  
6D87: 34 02      PSHS  A      ;   
6D89: 8D 24      BSR   $6DAF   ; SoundTestTableEntryCountGetIntoA()  
6D8B: 6C 41      INC   $0001,U ; Increment sound test index  
6D8D: A1 41      CMPA  $0001,U ;   
-----;-----
```

```

6D8F: 22 04      BHI   $6D95      ;
6D91: 86 01      LDA   #$01      ; Reset index to #1
6D93: A7 41      STA   $0001,U   ;
6D95: BD 7A 0B   JSR   $7A0B     ; StopCurrentSound_BugFix()
6D98: 81 02      CMPA  #$02      ;
6D9A: 35 82      PULS  A,PC      ;
;
-----;

```

As highlighted, a new function is being called in place of the \$6DF3. The new function is added at \$7A0B,3A which has the content shown below.

```

-----;
;
; StopCurrentSound_BugFix()
7A0B: 34 02      PSHS  A         ;
7A0D: BD 6D F3   JSR   $6DF3     ; StopCurrentSound(), Call original 'Stop' routine
7A10: A6 41      LDA   $0001,U   ; Get new/current sound test index
7A12: 81 02      CMPA  #$06     ; Check if advanced to sound 06
7A14: 26 03      BNE   $7A19     ; If not, then done.
7A16: BD C0 A5   JSR   $C0A5     ; If so, Call Sound text exit (escape button pressed)
7A19: 35 82      PULS  A,PC      ;
-----;

```

The new function, above, first calls the original “StopCurrentSound()” to retain original logic and then an added check is performed to see if the new sound index is 06. If new sound index is 06 it means the sound test has advanced from 05 to 06 and, therefore, subject to having the problem on 3<sup>rd</sup> party sound boards where the sound 05 may still be playing. If sound index is 06 then an existing function at \$C0A5 is called which is same function that the escape-button handler function also calls. It was found that calling \$C0A5 is what the escape-button handler does when it wants to ensure sound board is ‘off’.

The above changes appear to be harmless and found to address the issue of 05 play continuance. In the event that this change needs to be undone, the simplest fix is to:

- Leave the function at \$7A0B,3A (ROM Offset 0x6BA0B) unaltered, and
- Restore original function, at \$6D95,3A (ROM Offset 0x6AD95) change BD 7A 0B back to BD 6D F3

To further document the sound commands that are involved with sound test, and to show the commands sent to the sound board as part of the sound test, the table below summarizes all of the commands sent to the sound board during “Sound Test” mode. For completeness, this table includes the commands previously listed in a table above so all commands are available in this single table.

Function	Command Sent to Sound Board
<b>Play: Main Play</b>	0x7E 0x7D 0x7F 0x03
<b>Play: Get Jackpot Tune</b>	0x7E 0x7D 0x7F 0x04
<b>Play: M.Ball Lit Tune</b>	0x7E 0x7D 0x7F 0x07
<b>Play: Video Mode Tune</b>	0x7E 0x7D 0x7F 0x14
<b>Play: Database Backgr.</b>	0x7E 0x7D 0x7F 0xB2
<b>Play: 100K Awark</b>	0x7E 0x7D 0x7F 0xBF



<b>Play: Alarm Sound</b>	0x7E 0x7D 0x7F 0xC6
<b>Play: "Get the CPU"</b>	0x7E 0x7D 0x7F 0x7A 0x25
<b>Play: "Autofire"</b>	0x7E 0x7D 0x7F 0x7A 0x27
<b>Play: "Video mode"</b>	0x7E 0x7D 0x7F 0x7A 0x28
<b>Stop Current Sound (sent between 'Play' commands)</b>	0x7E 0x7D 0x7F
<b>Escape-Sound (coind-door escape button pushed)</b>	0x7F 0x58
<b>Minus-Sound (coin-door minus button pushed)</b>	0x7F 0x50
<b>Plus-Sound (coin-door plus button pushed)</b>	0x7F 0x51
<b>Enter-Sound (coin-door enter button pushed)</b>	0x7F 0x57
<b>End All Sounds (when escape button is pushed)</b>	0x00

Observing the above sequences and confirming with commands that an emulator reveals, some observations can be made.

- The sequence of "0x7E 0x7D 0x7F" is sent 2 times between sounds. Once for when the logic specifically wants to turn off the current sound and again as part of the command to play the next sound.
- In the case of 3<sup>rd</sup> party sound board not stopping sound 05 when transitioning to 06, the problem can be described as follows:
  - When sound board gets command to play 05 "0x7E 0x7D 0x7F 0xB2" to play sound, and
  - When sound board subsequently gets command "0x7E 0x7D 0x7F" to stop such sound
  - Sound board doesn't stop playing the sound 05.
- The L8.3 code adds the command "0x00" after the "0x7E 0x7D 0x7F" as an added effort to have the sound board stop playing the sound 05. Immediately after this 0x00, sound board then receives command to play 06 "0x7E 0x7D 0x7F 0xBF" which appears to then correctly play.

**Note: Future ROM update will re-analyze whether explicit stop at sound 05 can be removed.**

## FUA Inclusion Into Sound Test

As an added bonus to L8.3 and to serve as an Easter Egg and a coding exercise, the L8.3 includes updated Sound Test where the FUA (Profanity) sound call is added into the sound test but only when the "Profanity" adjustment is set to "On".

## Relocated Sound Test Table

In order to add a new sound to the test, the "Sound Test Table" which was previously shown, needed to be made larger to accommodate an extra row. As mentioned for other such table expansions, the table is moved to an unused region of ROM where the extra row can be safely added. When any table is moved, all references to such table need to be updated so they read the new table at its new location. These things are described below.

The original L-8 Sound Test Table is located at \$490D,3D (ROM Offset 0x7490D). The relocated table is moved to unused region of the same bank \$3D at \$7183,3D (ROM Offset 0x77183).

```
-----;-----
7183: 00 0C          ; Entries
7185: 03            ; Entry size
;
```

```

7186: 00 00 00          ; Null
7189: 03 00 00          ; 0x03 == Main play
718C: 04 00 00          ; 0x04 == Get jackpot tune
718F: 07 00 00          ; 0x07 == M.Ball lit tune
7192: 14 00 00          ; 0x14 == Video mode tune
7195: B2 00 00          ; 0xB2 == Database Backgr.
7198: BF 00 60          ; 0xBF == 100K Award
719B: C6 00 60          ; 0xC6 == Alarm Sound
719E: 25 01 60          ; 0x25 == "Get the cpu"
71A1: 27 01 60          ; 0x27 == "Autofire"
71A4: 28 01 60          ; 0x28 == "Video mode"
;
71A7: 3F 01 C0          ; 0x3F == "FUA"

```

-----  
*Note: This moved table also contains the changed byte for "Database Backgr." As described earlier.*

This new table entry contains these attributes:

- 0x3F, Sound command for FUA callout
- 0x01, Indicating this is a voice callout, causing 0x7A byte in the sound board command
- 0xC0, Time period for this sound call is 0xC0 corresponding to 3 seconds

As this call-out is lengthier than the others, the time period is increased to 0xC0 so that the sound call is not prematurely interrupted during the sound test. Given that 0x40 is 1 second, value 0xC0 corresponds to 3 seconds. The resulting sound board command for this new entry is: 0x7E 0x7D 0x7F 0x7A 0x3F.

With the relocated sound table, next step is to updated references to the old table so they now look at the new table. An examination of the L-8 code reveals there are two places during sound test where the sound table is referenced, such as the following in a function that is responsible for determining the total number of sounds for the test (so that code knows when to reset the sound test index back to 1 after playing the last sound in the test).

```

-----;-----
;
6DAF: 34 34          PSHS  Y,X,B          ; SoundTestTableEntryCountGetIntoA()
6DB1: 8E 81 FB          LDX   #$81FB          ; SoundTestTable[] pointer
6DB4: BD AC 38          JSR   $AC38          ; GetTableXEntryCountIntoY()
6DB7: 1F 20          TFR   Y,D          ; Put SoundTestTable[] entry count into D
6DB9: 1F 98          TFR   B,A          ; Move 8-bit SoundTestTable[] entry count into A
6DBB: 35 B4          PULS  B,X,Y,PC       ;
;
-----;-----

```

-----  
*Note: The content of this function is altered as part of the FUA Easter Egg, described later*

The function, above, shows how the code gets the address of the SoundTestTable[] from a pointer stored at \$81FB. This address \$81FB corresponds to non-banked ROM region where various table pointers are kept. This is at ROM offset 0x781FB which contains the 3 bytes: 49 0D 3D corresponding to WPC address \$490D,3D which is where the original SoundTestTable is located (as depicted earlier). To have the sound test code use the new location of the SoundTestTable, the bytes at \$81FB are changed from 49 0D 3D to 71 83 3D so code reads the sound test table from \$7183,3D.

## Updated Sound Test Logic for FUA

With the relocated and expanded sound table, alone, the sound test will include the FUA call-out as an ordinary part of its sound test. The goal is to only have this last entry of the sound test only appear when “Profanity” adjustment is “On”, so additional work is needed to accomplish this.

As shown, above, the Sound Test code utilizes a common function to simply obtain the total number of sounds in the test. In order to have the last row appear only when “Profanity” is enabled, the function logic is updated so that:

- It returns total number of rows in the new Sound Test Table when “Profanity” is “On”, and
- It returns total number of rows minus 1 when “Profanity” is “Off”

By implementing the new logic, above, the sound test will behave as if there are 11 rows in the table when “Profanity” is “Off” and will provide all 12 rows in the table when “Profanity” is “On”, thus giving the effect that the FUA call hidden until the “Profanity” adjustment is set to “On”.

```
-----;-----  
;   
; SoundTestTableEntryCountGetIntoA()  
6DAF: 34 34 PSHS Y,X,B ;   
6DB1: 8E 81 FB LDX #$81FB ; SoundTestTable[] pointer  
6DB4: BD 79 FE JSR $79FE ; GetTableXEntryCountIntoY_AdjustedForProfanity()  
6DB7: 1F 20 TFR Y,D ; Put SoundTestTable[] entry count into D  
6DB9: 1F 98 TFR B,A ; Move 8-bit SoundTestTable[] entry count into A  
6DBB: 35 B4 PULS B,X,Y,PC ;   
;   
-----;-----
```

As highlighted above, the function is altered so that instead calling a common function that simply returns the first 2 bytes of the table (which is the number of table entries in such table), the code is updated to call a new function at \$79FE which will provide the number of table entries adjusted based on the value of “Profanity” adjustment.

This new function at \$79FE is called within same bank, \$3A. This is a new function added to unused ROM bytes near the end of bank \$3A, corresponding to ROM offset 0x6B9FE.

```
-----;-----  
;   
; GetTableXEntryCountIntoY_AdjustedForProfanity()  
;   
79FE: BD AC 38 JSR $AC38 ; GetTableXEntryCountIntoY()  
7A01: BD 86 5B JSR $865B ; LookupGameAdjustmentParameterIandCheckIfEqualsParam2()  
; C-bit set when not-equal  
7A04: 16 00 ; 0x16, $1BE5:$1BE6 FeatureAdjustment022, Profanity  
7A06: 25 02 BCS $7A0A ; If C-set then Profanity is ON, return full table size  
7A08: 31 3F LEAY $FFFF,Y ; Profanity is OFF, decrement table size by 1, no FUA  
7A0A: 39 RTS ;   
-----;-----
```

The new function, above, retrieves the total number of Sound Test Table entries, and then calls a common WPC function that retrieves the current value of an adjustment (0x16 Profanity in this case) and compares its value to a value (0x00 in this case), returning C-bit set when non-equal. The

subsequent code then returns the full number of table entries (including the FUA row) when C-bit is set since the “Profanity” adjustment is not equal to 0x00 (since 0x00 is “Off” and 0x01 is “On”). When the C-bit is clear (or “not set”), the “Profanity” adjustment is equal to 0x00 (“Off”) so the code will use the LEAY \$FFFF,Y instruction to cause the value in Y to decrement by 1, thus artificially reducing the size of the sound test table to exclude the FUA callout.

With all of the above code changes in place, the desired effect of FUA call-out only when “Profanity” is “On” takes effect.

## The L8.3 Multiball Bug-Fixes

During L8.3 development, it was mentioned that a bug exists in L-8 regarding multiball. The bug was described as the game effectively forgetting that multiball is taking place. There are multiple balls on the playfield however game play proceeds as if the multiball is over. This can be especially troubling when a video mode is started or a “fire at will” cannon shot is initiated while multiple balls are playing on the playfield in ordinary play mode.

Investigation proceeded whereby several issues were observed regarding multiball.

- “Forgotten Multiball” can happen in various scenarios.
  - When hunter ship is hit, during its explosion animation if ball is immediately locked.
  - When hunter ship is hit, during its explosion animation if ball is immediately drained.
- Locking a ball causes display to report “Jackpot Multiplied 0x=0”.
  - When 2 balls remaining and each are locked in database and top lock simultaneously.
  - After this message, both balls are ejected back onto playfield in multiball.
  - Expected behavior is one locked and other in play with “Load the Gun” on the display.
- Missing ‘Load The Gun’ period can happen in some scenarios at multiball end.
  - When 2 balls remaining and one ball is locked and other is drained simultaneously.

There were multiple parts of the code involved for fixing all of these issues. They can be boiled down to the following:

- Multiball startup code needing corrections.
- Multiball maintenance/continuation loop code needing corrections.
- Switch-handler code for lock shots needing corrections (left lock, top lock, ball-popper).
- Switch-handler code for outhole needing corrections.

There were various timing problems that were identified and corrected. In software development these are sometimes referred to as a ‘race conditions’ where multiple pieces of code are designed to run asynchronous from each other and, depending on order in which the functions each are allowed to run, there may be situations when unexpected behaviors if not fully coded to account for all conditions. Some of the fixed code is to specifically cure the issues and some of the fixed code is to improve design based on theoretical problems that are observed in code but not necessarily able to encounter on the running code.

## WPC Scheduled Functions and Function IDs

Several of the bug fixes described below make reference to things like “scheduled functions” and “function ID” values. An entire document can be devoted to the concepts but to get an understanding of these things, a brief overview is mentioned here.

### Scheduling a Function

Any running function can schedule the startup of another function. Such scheduled startup will include:

- The WPC Address of where the function starts
- The 16-bit (2-byte) ID value of the function

The game code will then call the scheduled function at the next chance it gets. This is when the currently running code performs a “Sleep()” function call or when the currently running function run to ‘completion’ meaning it no longer needs to be part of the set of running functions.

The set of running functions is tracked in RAM as a linked list (actually multiple linked lists). A linked list is simply a list of objects located in memory whereby a fixed starting point is used to find the first list entry, then the address of the next list entry is contained in the list entry itself. This repeats until the ‘next’ pointer contains indicator that it is at the end of the list (usually all zeros).

A running function has its location in the linked list represented by a block of memory (i.e. a linked-list record) which contains various attributes about the running function such as its ID value and current execution point. In the case of a function having performed a “Sleep()” this execution point is important so the function can resume when the sleep period is over.

Tracing through WPC code in an emulator it can become evident how the scheduler works as the linked list traversal takes place and a function’s execution begins.

### Scheduled Function ID

As mentioned, the scheduled function is associated with a particular ID. The ID is a 16-bit (2-byte value) that other code can use to determine if a particular function exists in the linked-list (i.e. scheduled to run at next available opportunity). For example, the multiball maintenance function runs with ID 0086 and bug-fix functions can call a function to simply query whether function ID 0086 is running as a way to determine if multiball is currently active.

Some functions can alter the ID of the currently running function. For example, switch-matrix function for lock switches are first scheduled with function ID 0004 and then as the handler runs, it re-assigns it ID to give indicator of the switch and its state so that other functions can use a bitwise operation to determine if a group of functions are running that match the pattern of the new ID.

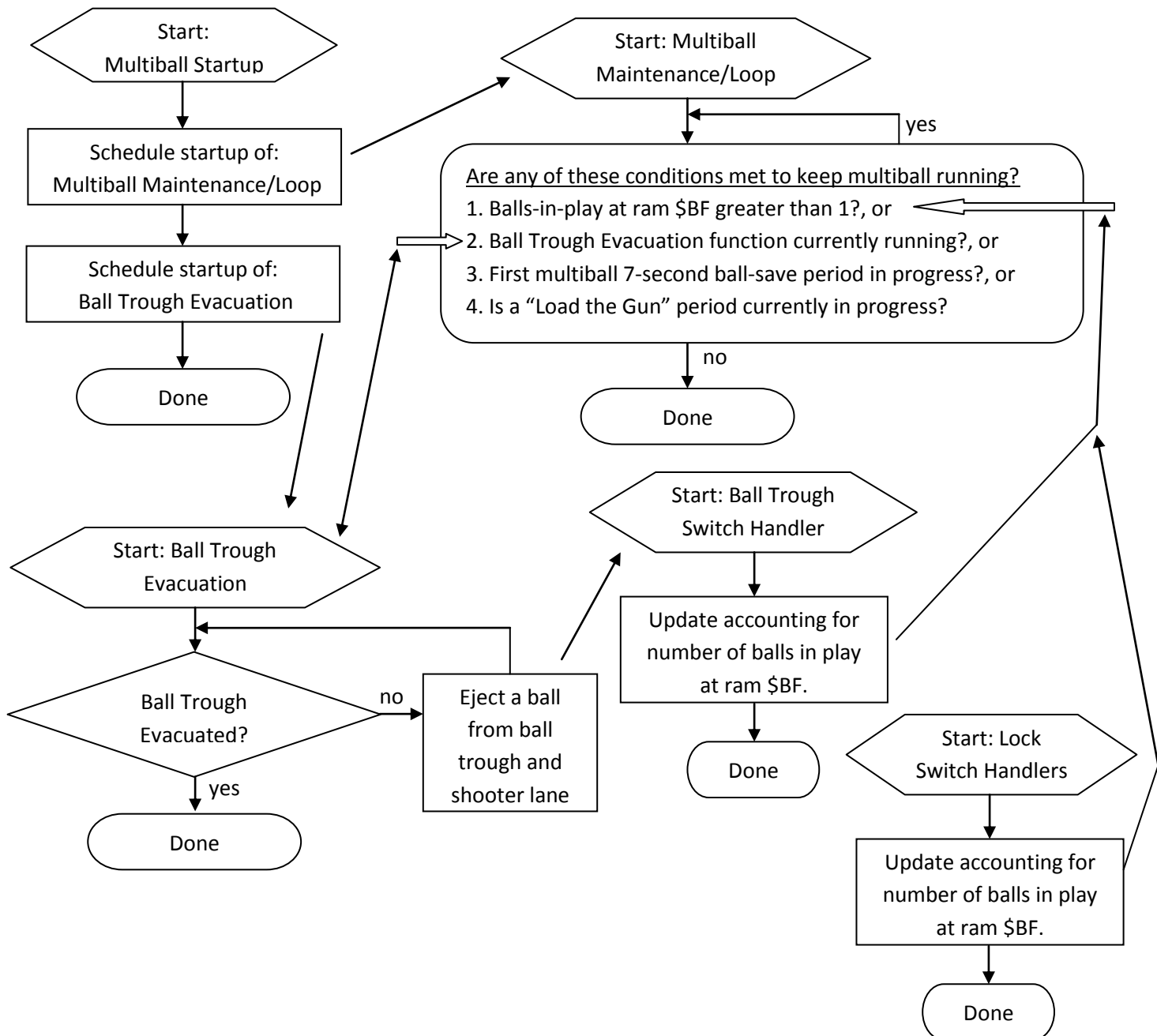


## Multiball Logic Overview

As part of understanding the multiball logic, some overview of the design is in order. When multiball is triggered during game play there are two main parts:

- Multiball Startup Code
- Multiball Maintenance/Loop Code

There is a lot of extra detail that is not depicted below (such as scheduling music and display animation). Shown below is a very high level overview and details may not be 100% accurate to the code flow, however the concepts being depicted serve the purpose of understanding the nature of the multiball bug fixes that are described in this section of this document.



## Multiball Startup Balls-In-Play Timing Problem

One of the issues was caused by a timing problem related to multiball prematurely ending when the number of balls on the playfield hasn't yet been increased to a number greater than one. The sequence of events when such trouble happens is as follows:

- Multiball startup code is entered, which:
  - Schedules the multiball maintenance loop, and
  - Schedules the ball-trough evacuation
- Multiball maintenance loop is entered, which checks conditions for keeping multiball in-progress:
  - Balls-In-Play count at ram \$BF needs to be greater than 1, but in this case it is still 1
  - Checks other conditions for multiball in-progress, none of which are met
    - Ball-trough evacuation routine is not currently running, and
    - First Multiball 7-second ball-save timer period is not active, and
    - A "Load the gun" period is not active
  - Due to the above logic, multiball maintenance loop exits, multiball ends.
- The Balls-in-play count at \$BF is still 1 due to:
  - Ball-trough evacuation function hasn't yet started, or
  - Ball-trough evacuation function finished its job but the switch-handler hasn't yet engaged to increment \$BF to a value greater than 1.
    - *As depicted in the logic flow, the ball trough evacuation necessarily leads to ball-trough switch state changes which are then detected by the interrupt routine switch-matrix scanning which then leads to the scheduled function handler for the ball-trough switches, which is where the \$BF value gets updated.*

For this code change, the intent was to help retain the original logic related to \$BF accounting and multiball start and multiball retention, namely:

- The multiball maintenance loop is designed to check that \$BF was non-zero and
- The initial increment of \$BF to a value greater than 1 has been sure to have been taken place

There are two changes to the code to ensure that the multiball loop doesn't inadvertently enter at a moment where \$BF is still at value 1 (and none of the other multiball conditions are currently met):

- The multiball startup code waits until the \$BF is greater than 1 prior to completing, and
- The multiball maintenance loop includes in its conditions for multiball in-progress a new rule:
  - If the multiball startup function is still running, MB is not declared as done.

Each of the above two code changes are described in the following two sections.

### Multiball Startup Balls-In-Play Timing Fix: Startup waits for balls-in-play greater than 1

The L-8 Multiball startup function, partially annotated is shown below for reference. Area of interest for this code change is near the end of the function and will be described in more detail, below.

-----  
;

```

; Multiball Start!
; ID 00B8 == Multiball start
;
6C4E: BD F7 59 JSR $F759 ; Checks state variables $86, $87, $88, $1793.
; All 0x00 means okay to proceed.
6C51: 7E 6C 54 JMP $6C54 ; <nop>
6C54: 10 26 00 82 LBNE $6CDA ;
6C58: BD 8B 77 JSR $8B77 ; ScheduleFunctionStart()
6C5B: 00 86 ; Schedule 0x0086, multiball loop, runs until MB done
6C5D: 6D 0C 31 ; $6D0C,31 multiballLoop()
;
6C60: 86 02 LDA #$02 ; SolenoidTableEntry02, 0A=Top Lock, 20
6C62: BD 88 F5 JSR $88F5 ; EnergizeSolenoidTableEntryIfNeeded()
6C65: 6E 6B 3B ;
6C68: 86 03 LDA #$03 ; SolenoidTableEntry03, 10=Left Lock, 20
6C6A: BD 88 F5 JSR $88F5 ; EnergizeSolenoidTableEntryIfNeeded()
6C6D: 6E 6B 3B ;
;
6C70: BD 48 8D JSR $488D ; LeftVaultStateSet()
6C73: BD 48 9E JSR $489E ; Lock2StateSet()
6C76: 86 04 LDA #$04 ; 0x04 == SolenoidTableEntry04, 04=Trough, 40
6C78: C6 01 LDB #$01 ;
6C7A: BD 88 F5 JSR $88F5 ; CallBankedFunction_Param_WPCAddr()
6C7D: 6E 5A 3B ;
;
6C80: BD F7 59 JSR $F759 ; Checks state variables $86, $87, $88, $1793.
; All 0x00 means okay to proceed.
6C83: 7E 6C 86 JMP $6C86 ; <nop> ;
6C86: 26 52 BNE $6CDA ;
;
6C88: 8E 06 03 LDX #$0603 ; #$0603 is Hunter ship hits remaining for multiball
6C8B: BD FB 29 JSR $FB29 ; IncrementXByPlayerIndexNumber()
6C8E: 7E 6C 91 JMP $6C91 ; <nop>
6C91: 6F 84 CLR ,X ; Starting MB so ensure # of hunter ship hits=0
;
6C93: BD 88 D5 JSR $88D5 ; Call15253,39WithXParameterBytes()
6C96: 00 1B ;
6C98: 8E 05 C9 LDX #$05C9 ; $05C9, Base Addr of #-of-multiballs per player/game
6C9B: BD FB 29 JSR $FB29 ; IncrementXByPlayerIndexNumber()
6C9E: 7E 6C A1 JMP $6CA1 ; <nop> JSR to new function here JSR to $FB8C
6CA1: 6C 84 INC ,X ; Increment multiballs achieved counter for cur player
6CA3: BD 71 A3 JSR $71A3 ; Increment05BDbyPlayerIndexNumber()
6CA6: BD 71 A3 JSR $71A3 ; Increment05BDbyPlayerIndexNumber()
6CA9: BD 71 A3 JSR $71A3 ; Increment05BDbyPlayerIndexNumber()
;
6CAC: 8D 2F BSR $6CDD ; <-- function can incur a sleep which yields to others
6CAE: C6 01 LDB #$01 ;
6CB0: BD 6D E9 JSR $6DE9 ;
6CB3: 0D C0 TST $C0 ;
6CB5: 27 09 BEQ $6CC0 ;
;
6CB7: 86 06 LDA #$06 ; 0x06 == multiball theme music
6CB9: BD C0 BC JSR $C0BC ; PlayMusicRegisterA()
6CBC: BD 85 53 JSR $8553 ; ShowMonochromeAnimationParameterByte()
6CBF: 27 ;
;
6CC0: BD 8B C3 JSR $8BC3 ; ScheduleFunctionCallback()
6CC3: 00 82 ; 0082 must be running for the MB to keep running
6CC5: 6F 0D 31 ; BallTroughEvacuate()
;
6CC8: BD 83 46 JSR $8346 ; Sleep()
6CCB: 60 ; 0x60 = 1.5 seconds

```

```

;
6CCC: BD 83 46 JSR $8346 ; -\ Sleep()
6CCF: 02 ; |
6CD0: BD 8B C3 JSR $8BC3 ; | ScheduleFunctionCallback()
6CD3: 00 82 ; | ID 0082
6CD5: 6F 0D 31 ; | BallTroughEvacuate ()
6CD8: 27 F2 BEQ $6CCC ; -/
;
6CDA: 7E 99 A2 JMP $99A2 ;
;
-----;

```

Notable elements of the multiball startup function, above, have been highlighted.

- At \$6CAC a function (\$6CDD) is called where logic can eventually reach a “Sleep()” function. This means other scheduled work is allowed to proceed. In the event that such sleep takes place, the multiball maintenance loop may be allowed to run which would be a case where the multiball maintenance could discover that \$BF is still 1, and the ball-trough evacuation is not running. This could be a case where MB fails to start when it should. *Detailed analysis of what \$6CDD does is outside scope of this discussion and left as an exercise to the reader.*
- Starting at \$6CC0, the BallTroughEvacuate() function is scheduled, followed by a 1.5 second sleep and a small loop that ensures BallTroughEvacuate() is running. Detailed analysis of why this is done in this way as opposed to simply have a single call into BallTroughEvacuate() has not been performed. In the event the first BallTroughEvacuate() didn’t result in the BallTroughEvacuate() function running, the 1.5 second sleep opens another window where the multiball maintenance could discover \$BF is still 1 and causing multiball to prematurely end.

To improve the overall logic in this area, the multiball startup code is improved in L8.3 so that the startup function doesn’t complete until the \$BF is greater than 1. The changed logic starts near the end of the multiball startup function with the following.

```

...
;
6CB7: 86 06 LDA #$06 ; 0x06 == multiball theme music
6CB9: BD C0 BC JSR $C0BC ; PlayMusicRegisterA()
6CBC: BD 85 53 JSR $8553 ; ShowMonochromeAnimationParameterByte()
6CBF: 27 ;
;
6CC0: BD 8B C3 JSR $8BC3 ; ScheduleFunctionCallback()
6CC3: 00 82 ; 0082 must be running for the MB to keep running
6CC5: 6F 0D 31 ; BallTroughEvacuate()
;
6CC8: BD 83 46 JSR $8346 ; Sleep()
6CCB: 60 ; 0x60 = 1.5 seconds
;
6CCC: BD 83 46 JSR $8346 ; -\ Sleep()
6CCF: 02 ; |
6CD0: BD 8B C3 JSR $8BC3 ; | ScheduleFunctionCallback()
6CD3: 00 82 ; | ID 0082
6CD5: 6F 0D 31 ; | BallTroughEvacuate ()
6CD8: 27 F2 BEQ $6CCC ; -/
;
6CCC: BD 88 F5 JSR $88F5 ; CallBankedFunction Param WPCAddr()
6CCF: 79 35 3E ; BugfixFunction DelayedMBInit()

```

```

6CD2: 20 06      BRA   $6CDA      ;
6CD4: 12 12 12  NOPx3     ;
6CD7: 12 12 12  NOPx3     ;
;
6CDA: 7E 99 A2   JMP   $99A2     ;
;
-----;

```

As shown, above, the loop that previously checked called function to schedule BallTroughEvacuate() has been replaced with a single call to a new function at \$7935,3B. To fill in the remaining bytes in ROM, six NOP instructions (no-operation, dummy instruction) are in place to allow code to flow smoothly to the end.

Next is the content of this new function at \$7935,3B (ROM offset 0x6F935).

```

-----;
7935: 34 06      PSHS  B,A        ;
;
; Perform original loop for BallTroughEvacuate()
; This is code copied from MB-init at $6CCC,31
;
7937: BD 83 46   JSR   $8346     ; -\ Sleep()
793A: 02                ; |
793B: BD 8B C3   JSR   $8BC3     ; | ScheduleFunctionCallback()
793E: 00 82                ; | ID 0082
7940: 6F 0D 31                ; | BallTroughEvacuate()
7943: 27 F2      BEQ   $7937     ; -/
;
; Now perform bugfix, checking for $BF > 01 or
; wait for timeout before giving up.
;
7945: 86 55      LDA   #$55      ; Wait up to the sleep period multiplied by this counter
;
7947: D6 BF      LDB   $BF      ; -\ Get number of balls on the playfield from $BF
7949: C1 01      CMPB  #$01     ; | Branches if $BF is higher than 1
794B: 22 17      BHI   $7964     ; |
; |
794D: BD 86 90   JSR   $8690     ; | SearchLinkedListForId() // c-bit clear = ID found
7950: 00 82                ; | 0082, BallTroughEvacuate()
7952: 24 07      BCC   $795B     ; | Function is running, goto keep_waiting
; |
7954: BD 86 90   JSR   $8690     ; | SearchLinkedListForId() // c-bit clear = ID found
7957: 00 B2                ; | 00B2, shooter-lane-kick-and-check-switch
7959: 25 09      BCS   $7964     ; | Function is not running, goto end, all done
; |
; | keep_waiting:
; |
795B: 4A                DECA                ; | Decrement the timeout counter
795C: 27 06      BEQ   $7965     ; | If decremented to zero, then done.
795E: BD 83 46   JSR   $8346     ; | Sleep()
7961: 06                ; |
7962: 20 E3      BRA   $7947     ; -/ Keep checking
;
7964: 35 86      PULS  A,B,PC    ; Either the ball trough and shooter lane are empty
; or gave up waiting
-----;

```



The new function, above first performs the original loop that was removed from \$6CCC,31 and replaced with call to this fixup function. After that, a wait-loop is performed where the code will repeatedly check for 'done' condition or after a period before giving up waiting.

The loop, starting at \$7947, will cycle up to 0x55 times, with a sleep of value 06 each time. Total sleeps will be 0x55 (decimal 85) multiplied by 6 which is 510. Given that 0x40 (decimal 64) is 1 second, this equates to a timeout of about 8 seconds before the code gives up waiting for the conditions for be met. It is not expected that this timeout will ever be hit but this prevents code from looping forever in the event of unexpected playfield conditions.

The loop is checking for the following conditions:

- If \$BF is discovered to be greater than 1 then loop exits, otherwise
- If the BallTroughEvacuate() function is still running, keep looping, otherwise
- If the "shooter-lane-kick-and-check-switch" code is running, keep looping, otherwise
- The loop exits

This logic mentions this "shooter-lane-kick-and-check-switch" which is a secondary function that is scheduled by the BallTroughEvacuate() function and not mentioned in earlier descriptions. For absolute completeness in the above loop, it made sense that the code loops until the BallTroughEvacuate() is done <and> this subsequent function is also done. In most situations, the \$BF value being greater than 1 is the cause of the loop exit.

Once the above loop exits, code returns back to the Multiball Startup function which then exits, thus ensuring that the Multiball Startup function only finishes after the \$BF has incremented past 1 (or BallTroughEvacuate() is completely done with all of its work, or timeout condition was hit if code gave up waiting for conditions to be met).

### **Multiball Startup Balls-In-Play Timing Fix: Maintenance function checks if startup is running**

As mentioned, the Multiball Startup function can potentially hit a Sleep() when, at \$6CAC, function \$6CDD is called. If that were to happen, then the Multiball Maintenance function could start up (due to have been scheduled earlier in the multiball startup code) and immediately declare end of multiball due to all conditions not being met to retain multiball.

To show the code fix for this issue, the L-8 multiball maintenance code, partially annotated, is shown below. This function was scheduled, as depicted above, during the multiball startup routine, and it starts at \$6D0C,31 (ROM offset 0x44D0C).

```
-----;-----  
; ID 0086  
; Called at multiball start  
; This is the main loop of multiball  
; 6D0C: BD 87 15 JSR $8715  
; 6D0F: 10 40  
; 6D11: BD 87 15 JSR $8715  
;
```

```

6D14: 0E 40 ;
6D16: BD 84 8F JSR $848F ; ClearMemoryFlag()
6D19: 48 ;
6D1A: BD 84 8F JSR $848F ; ClearMemoryFlag()
6D1D: 42 ;
6D1E: BD 84 8F JSR $848F ; ClearMemoryFlag()
6D21: 43 ;
6D22: BD 86 9E JSR $869E ; CancelScheduledCallbackFunction()
6D25: 00 EC ;
6D27: BD 84 8F JSR $848F ; ClearMemoryFlag()
6D2A: 45 ;
6D2B: BD 87 3C JSR $873C ;
6D2E: 2F 40 ;
6D30: 25 04 BCS $6D36 ;
6D32: BD 84 80 JSR $8480 ; SetMemoryFlag()
6D35: 45 ;
6D36: BD 87 22 JSR $8722 ;
6D39: 2F 40 ;
6D3B: BD 84 8F JSR $848F ; ClearMemoryFlag()
6D3E: 46 ;
6D3F: BD 84 49 JSR $8449 ;
6D42: 1A ;
6D43: 25 04 BCS $6D49 ;
6D45: BD 84 80 JSR $8480 ; SetMemoryFlag()
6D48: 46 ;
6D49: BD 84 2B JSR $842B ;
6D4C: 1A ;
6D4D: BD 84 8F JSR $848F ; ClearMemoryFlag()
6D50: 47 ;
6D51: BD 84 49 JSR $8449 ;
6D54: 0C ;
6D55: 25 04 BCS $6D5B ;
6D57: BD 84 80 JSR $8480 ; SetMemoryFlag()
6D5A: 47 ;
6D5B: BD 84 2B JSR $842B ; ClearSingleLampParameterByte()
6D5E: 0C ;
6D5F: BD 83 46 JSR $8346 ; Sleep()
6D62: 40 ; 0x40 = 1 second
;
; Keep multiball going as long as:
; - More than 1 ball on playfield ($BF value), or
; - 0082 is running, BallTroughEvacuate(), or
; - 0083 is running, 7-second ball-saver, or
; - 00A9 is running, Load-the-gun period, or
; - MemoryFlag 48 is set ("LOAD THE GUN" w/1-ball left)
;
6D63: BD 83 46 JSR $8346 ; -\ Sleep()
6D66: 06 ; |
6D67: 96 BF LDA $BF ; | $BF is number of balls on playfield
6D69: 81 01 CMPA #$01 ; | As long as there are more than 1 ball, keep looping.
6D6B: 22 F6 BHI $6D63 ; |
6D6D: BD 86 90 JSR $8690 ; | SearchLinkedListForId() // c-bit clear = ID found
6D70: 00 82 ; | ID 0082 == BallTroughEvacuate()
6D72: 24 EF BCC $6D63 ; |
6D74: BD 86 90 JSR $8690 ; | SearchLinkedListForId() // c-bit clear = ID found
6D77: 00 83 ; | ID 0083 == 7 second first multiball ball-save timer
6D79: 24 E8 BCC $6D63 ; |
6D7B: BD 86 90 JSR $8690 ; | SearchLinkedListForId() // c-bit clear = ID found
6D7E: 00 A9 ; | ID 00A9 == "LOAD THE GUN" period is running
6D80: 24 E1 BCC $6D63 ; |
6D82: BD 84 AD JSR $84AD ; | GetMemoryFlag() // C-bit clear when flag set
6D85: 48 ; | 0x48 flag is "LOAD THE GUN" period

```

```

6D86: 24 DB      BCC  $6D63      ; -/
;
; Out-of-Loop -- Multiball is done, now cleanup
;
6D88: BD 84 AD   JSR  $84AD      ; GetMemoryFlag() // C-bit clear when flag set
6D8B: 42
6D8C: 24 1B      BCC  $6DA9      ;
;
; Keep cleanup loop going as long as:
; 00AA is running <or>
; 00AB is running <or>
; 0084 is running
;
;
; Loop below cleanup at end of multiball?
6D8E: BD 86 90   JSR  $8690      ; -\ SearchLinkedListForId() // c-bit clear = ID found
6D91: 00 AA
; | ID 00AA Ball-In-Popper
6D93: 24 0E      BCC  $6DA3      ; |
6D95: BD 86 90   JSR  $8690      ; | SearchLinkedListForId() // c-bit clear = ID found
6D98: 00 AB
; | ID 00AB Ball-in-gun function is running
6D9A: 24 07      BCC  $6DA3      ; |
6D9C: BD 86 90   JSR  $8690      ; | SearchLinkedListForId() // c-bit clear = ID found
6D9F: 00 84
; | ID 0084 Ball-gun-to-Target-Period function
6DA1: 25 06      BCS  $6DA9      ; |
6DA3: BD 83 46   JSR  $8346      ; | Sleep()
6DA6: 04
; |
6DA7: 20 E5      BRA  $6D8E      ; -/
;
;
6DA9: BD 68 08   JSR  $6808      ; ScheduleDropTargetUp()
6DAC: BD 87 22   JSR  $8722      ;
;
6DAF: 10 40
;
6DB1: BD 87 22   JSR  $8722      ;
;
6DB4: 0D 40
;
6DB6: BD 87 22   JSR  $8722      ;
;
6DB9: 0E 40
;
6DBB: BD 84 80   JSR  $8480      ; SetMemoryFlag()
6DBE: D2
;
6DBF: BD 84 AD   JSR  $84AD      ; GetMemoryFlag()
6DC2: 45
;
6DC3: 25 03      BCS  $6DC8      ;
;
6DC5: BD 68 82   JSR  $6882      ;
;
6DC8: BD 84 AD   JSR  $84AD      ; GetMemoryFlag()
6DCB: 46
;
6DCC: 25 04      BCS  $6DD2      ;
;
6DCE: BD 84 1C   JSR  $841C      ;
;
6DD1: 1A
;
6DD2: BD 84 AD   JSR  $84AD      ; GetMemoryFlag()
6DD5: 47
;
6DD6: 25 04      BCS  $6DDC      ;
;
6DD8: BD 84 1C   JSR  $841C      ;
;
6DDB: 0C
;
6DDC: 10 8E 00 8B LDY  #$008B ;
;
6DE0: BD 9B 83   JSR  $9B83      ;
;
6DE3: BD 56 9C   JSR  $569C      ;
;
6DE6: 7E 99 A2   JMP  $99A2      ;
;
-----

```

The important/applicable section starts at the highlighted comment corresponding to code beginning at \$6D63. As shown, the multiball loop repeatedly checks for 4 items in order to keep multiball in progress.

Once the loop is exited then multiball is considered done and multiball cleanup code ensues. The four conditions that code checks are as follows:

- \$BF, Balls in play value is greater than 1, or
- \$0082 function, BallTroughEvacuate() function is running, or
- \$0083 function, 7-second ball-saver timeout at first multiball is active, or
- \$00A9 function, "Load the ball" mode is active, or
- 0x48 flag is set meaning "Load the ball" is active

As previously described and depicted, the above logic can prematurely end multiball if the multiball startup code calls a "Sleep()" function prior to its first call of BallTroughEvacuate(). To address this, the loop is augmented to include extra criteria to keep multiball running. The code is altered as shown below:

```

...
6D63: BD 83 46 JSR $8346 ; -\ Sleep()
6D66: 06 ; |
6D67: 96 BF LDA $BF ; | $BF is number of balls on playfield
6D69: 81 01 CMPA #$01 ; | As long as there are more than 1 ball, keep looping.
6D6B: 22 F6 BHI $6D63 ; |
6D6D: BD 86 90 JSR $8690 ; | SearchLinkedListForId() // c-bit clear = ID found
6D70: 00 82 ; | ID 0082 == BallTroughEvacuate()
6D72: 24 EF BCC $6D63 ; |
6D74: BD 86 90 JSR $8690 ; | SearchLinkedListForId() // c-bit clear = ID found
6D77: 00 83 ; | ID 0083 == 7 second first multiball ball-save timer
6D79: 24 E8 BCC $6D63 ; |
; |
6D7B: BD 86 90 JSR $8690 ; | SearchLinkedListForId() // c-bit clear = ID found
6D7E: 00 A9 ; | ID 00A9 == "LOAD THE GUN" period is running
6D80: 24 E1 BCC $6D63 ; |
6D82: BD 84 AD JSR $84AD ; | GetMemoryFlag() // C-bit clear when flag set
6D85: 48 ; | 0x48 flag is "LOAD THE GUN" period
6D86: 24 DB BCC $6D63 ; -/
; |
6D7B: BD 88 F5 JSR $88F5 ; | CallBankedFunction Param WPCAddr()
6D7E: 79 22 3B ; | <new code in bank $3B>
6D81: 24 E0 BCC $6D63 ; -/
; |
6D83: 7E 6D 88 JMP $6D88 ; all done
6D86: 12 12 NOP x2 ; all done
;
; Out-of-Loop -- Multiball is done, now cleanup
;
6D88: BD 84 AD JSR $84AD ; GetMemoryFlag() // C-bit clear when flag set
6D8B: 42 ;
6D8C: 24 1B BCC $6DA9 ;
...

```

As highlighted above, the multiball loop code that previously checked 2 conditions related to "Load the gun" were replaced with a single call to a new function located at \$7922,3B. The returned C-bit is used to determine whether to keep multiball-loop running or to exit. The new code is smaller than the old code so some dummy instructions are added to get code to \$6D88 when the multiball-loop is exited.

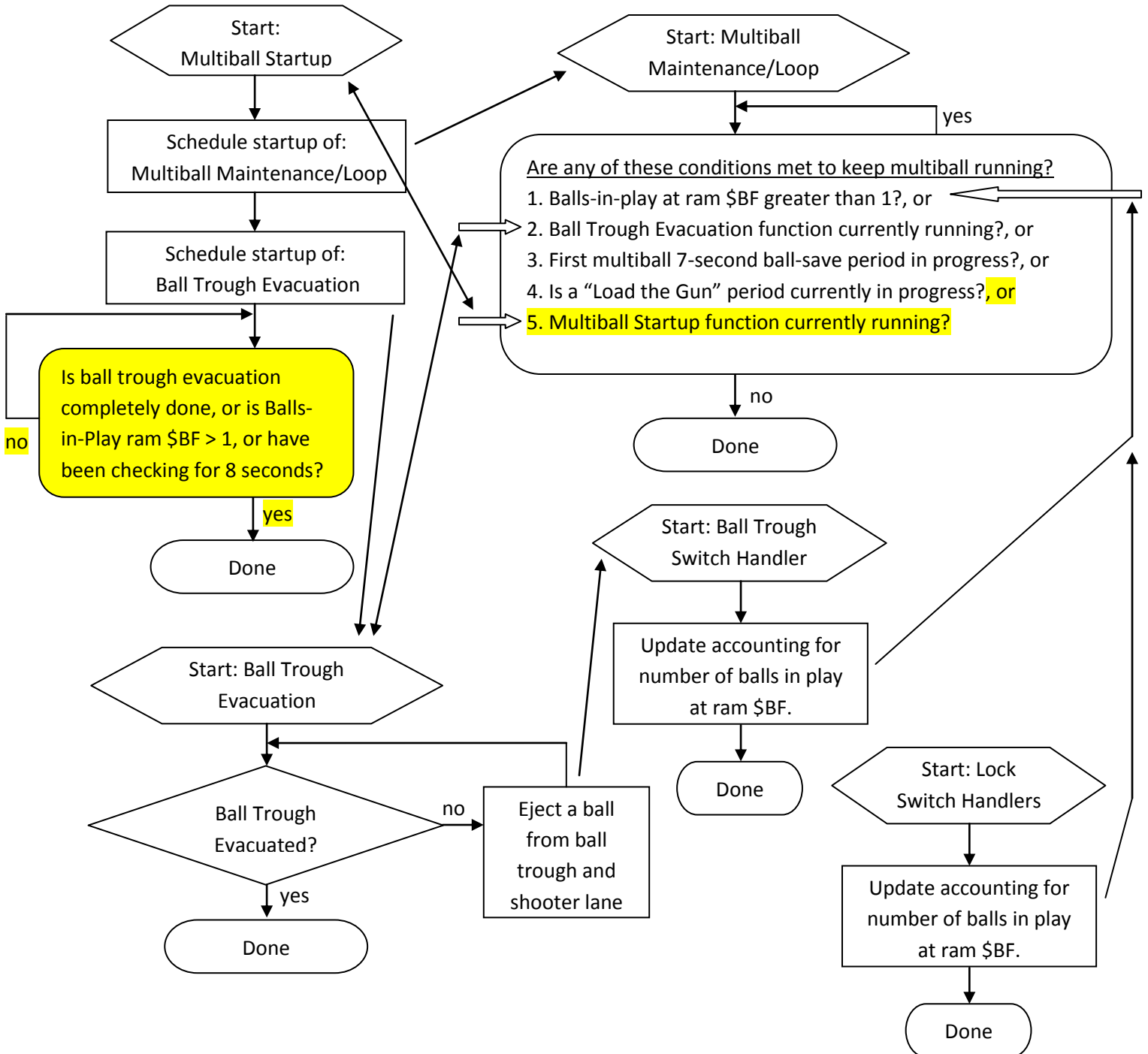
The new function at \$7922,3B (ROM offset 0x6F922) is depicted below:

```
-----;-----  
;   
; Function called from $6D7B,31 during multiball loop to   
; add additional check for multiball-init still running   
; as criteria to keep multiball running.   
;   
7922: BD 86 90 JSR $8690 ; SearchLinkedListForId() // c-bit clear = ID found   
7925: 00 A9 ; ID 00A9 == "LOAD THE GUN" period at end of multiball   
7927: 24 0B BCC $7934 ;   
7929: BD 86 90 JSR $8690 ; SearchLinkedListForId() // c-bit clear = ID found   
792C: 00 B8 ; MultiballStart function at $6C4E,31   
792E: 24 04 BCC $7934 ;   
7930: BD 84 AD JSR $84AD ; GetMemoryFlag() // C-bit clear when flag set   
7933: 48 ; 0x48 flag is (?)   
7934: 39 RTS ;   
-----;-----
```

The new function, above, contains the same elements as what the original code had plus a single new element added for keeping the multiball running. The new element is the check for function ID 00B8 which is the Multiball Startup function. This simple check ensures the multiball loop will not prematurely end multiball if the multiball startup function is still running. This ensures that any "Sleep()" performed during the multiball startup (prior to its scheduling of BallTroughEvacuate()) will not result in early running of the Multiball Maintenance function to falsely declare end of multiball since it will now discover that multiball startup is still in progress and, therefore, will keep multiball active.

## Multiball Startup Balls-In-Play Timing Fix: Corrected Multiball Logic

The fixes depicted, thus far, represent the following **yellow highlighted** changes to the original multiball logic flow that was previously depicted.





## Multiball Switch Handler Logic Updates

The switch handler code was found to need some updated logic in order to prevent the various multiball troubles from taking place. This section will describe the code changes that were made in order to resolve issues with multiball when ball-lock or outhole switches are hit simultaneously or too hit early during multiball startup.

### Switch Handling, A brief overview

The layers of code involved in handling a switch can be quite extensive and warrant an entire document in itself. A brief summary of the switch handling will be given here with further research being deferred to other documents and research.

- The Motorola 68B09, the Interrupt Service Routine (ISR) address is at \$FFF8
- This is ROM offset 0x7FFF8. In T2 L-8 this contains D9 C0 for \$D9C0 (ROM Offset \$7D9C0)
- The ISR starts at \$D9C0, performing various tasks, including switch-matrix scanning.
- Switches are read starting at \$DCF1 (coin-door switches)
- Switch-matrix read starting at \$DD12, starting at column 1 through column 7.
- The game "main loop" code starts at \$975F, performing various low-level system handling.
- At \$977C it calls function \$9D4E which ends up leading to switch-matrix memory reading
- The \$9D4E function, at \$9D88, calls \$9417 which ends up leading to switch-matrix memory read
- The \$9417 function, at \$94C0, calls \$9614 which ends up leading to switch-matrix memory read
- The \$9614 function, at \$9641, calls \$983D which is a function that schedules the switch-matrix handler function for a hit switch. Function ID is in register Y, Function address in X, bank B.

The sequence, above, could further be researched to understand more details but suffice it to say, when a switch is closed, the above logic (if the tracing, above, is correct for all cases), will result in a function being called that is responsible for handling the switch.

The address of the function is derived from the Playfield Switch Table. This table contains an 11-byte entry for each switch with various metadata including the address of the function that gets called when the switch is hit. In L-8, the address of the table is stored at \$81C5 (ROM offset 0x781C5) and it contains the value 49 31 3D for \$4931,3D (ROM offset 0x74931). The switch table is partially annotated and shown in full, below, for completeness:

```
-----;-----  
;  
; PlayfieldSwitchTable[]  
;  
;  
; This is loaded for switch indexes 0x01-0x41  
;  
; Each 11-byte entry for each switch below has the following format:  
; -----  
; 01 02 03 04 05 06 07 08 09 0A 0B  
;  
;  
; 01 02  
; 03 04 05 ; $XXYY,ZZ Address in ROM
```

```

; 06 07
; 08      ; This is a flag byte with the following bit-flag definitions for this switch
;          ; 11111111
;          ; |11111111
;          ; |11111111\-- 0x01 bit.
;          ; |11111111\--- 0x02 bit.
;          ; |11111111\---- 0x04 bit.
;          ; |11111111\----- 0x08 bit.
;          ; |11111111\----- 0x10 bit.
;          ; |11111111\----- 0x20 bit. When set, switch is NOT included 90-ball switch error report
;          ; |11111111\----- 0x40 bit. (above comment based on findings from other WPC study)
;          ; |11111111\----- 0x80 bit.
;
;
; 09
; 0A 0B      ; $0A0B is ID used for the callback function when it get scheduled
;
;
;
;
4931: 00 41      ; Table has entries for up to switch index value 0x41
4933: 0B          ; Each entry is 0x0B in length
;
;
4934: 04 04      ; SwitchTableEntry00, 0, Invalid Switch Index/Reference
4936: 99 A2 FF    ; $99A2,FF
4939: 3C 00 20    ;
493C: 00 00 00    ;
;
;
493F: 00 09      ; SwitchTableEntry01, 1, Invalid Switch Index/Reference
4941: 99 A2 FF    ; $99A2,FF
4944: 3C 00 20    ;
4947: 00 00 04    ;
;
;
494A: 00 09      ; SwitchTableEntry02, 2, Invalid Switch Index/Reference
494C: 99 A2 FF    ; $99A2,FF
494F: 3C 00 20    ;
4952: 00 00 04    ;
;
;
4955: 00 09      ; SwitchTableEntry03, 3, Invalid Switch Index/Reference
4957: 99 A2 FF    ; $99A2,FF
495A: 3C 00 20    ;
495D: 00 00 04    ;
;
;
4960: 00 09      ; SwitchTableEntry04, 4, Invalid Switch Index/Reference
4962: 99 A2 FF    ; $99A2,FF
4965: 3C 00 20    ;
4968: 00 00 04    ;
;
;
496B: 00 02      ; SwitchTableEntry05, 5, coin-door: service credit
496D: 5E A6 39    ; $5EA6,39
4970: 3C 00 20    ;
4973: 78 00 04    ;
;
;

```

```

4976: 00 02 ; SwitchTableEntry06, 6, coin-door: volume down
4978: 47 99 3D ; $4799,3D
497B: 3C 00 20 ;
497E: 78 00 04 ;
;
;
;
4981: 00 02 ; SwitchTableEntry07, 7, coin-door: volume up
4983: 47 7D 3D ; $477D,3D
4986: 3C 00 20 ;
4989: 78 00 04 ;
;
;
;
498C: 00 02 ; SwitchTableEntry08, 8, coin-door: test/menu
498E: 8E 7D FF ; $8E7D,FF
4991: 3C 00 20 ;
4992: 78 00 04 ;
;
;
;
4997: 00 02 ; SwitchTableEntry09, 11, Right Flipper
4999: 5C CA 31 ; $5CCA,31
499C: 3C 00 00 ;
499F: 40 00 04 ;
;
;
;
49A2: 00 02 ; SwitchTableEntry0A, 12, Left Flipper
49A4: 5C 17 31 ; $5C17,31
49A7: 3C 00 00 ;
49AA: 40 00 04 ;
;
;
;
49AD: 00 02 ; SwitchTableEntry0B, 13, Start Button
49AF: 45 40 38 ;
49B2: 3C 00 00 ;
49B5: 78 00 04 ;
;
;
;
49B8: 00 02 ; SwitchTableEntry0C, 14, Plumb Bob Tilt
49BA: 58 A7 39 ;
49BD: 3C 00 20 ;
49C0: E8 00 04 ;
;
;
;
49C3: 10 0C ; SwitchTableEntry0D, 15, Trough Left
49C5: 70 CC 3B ; SwitchMatrixHdlr_TroughLeftCenterRight()
49C8: 3C 00 00 ;
49CB: F8 00 04 ;
;
;
;
49CE: 10 0C ; SwitchTableEntry0E, 16, Trough Center
49D0: 70 CC 3B ;
49D3: 3C 00 00 ;
49D6: F8 00 04 ;
;
;
;
49D9: 10 0C ; SwitchTableEntry0F, 17, Trough Right
49DB: 70 CC 3B ;
49DE: 3C 00 00 ;
49E1: F8 00 04 ;
;
;
;
49E4: 10 0C ; SwitchTableEntry10, 18, Outhole
49E6: 44 B1 31 ; SwitchMatrixHdlr_Outhole()
49E9: 3C 00 00 ;
49EC: 78 78 0D ;
;
;
;
49EF: 00 02 ; SwitchTableEntry11, 21, Slam Tilt
49F1: 5F 96 39 ;

```

```
49F4: 3C 00 20 ;
49F7: 78 5F 96 ;
;
49F9: 00 02 ; SwitchTableEntry12, 22, Coin Door Closed
49FC: 60 0F 39 ;
49FF: 3C 00 20 ;
49A2: F8 00 04 ;
;
4A05: 00 02 ; SwitchTableEntry13, 23, Ticket Dispenser
4A07: 99 A2 FF ;
4A0A: 3C 00 20 ;
4A0D: 40 00 04 ;
;
4A10: 00 02 ; SwitchTableEntry14, 24, Always Closed
4A12: 99 A2 FF ;
4A15: 3C 00 30 ;
4A18: 00 00 04 ;
;
4A1B: 00 04 ; SwitchTableEntry15, 25, Left Outlane
4A1D: 49 35 31 ;
4A20: 3C 00 80 ;
4A23: C0 00 04 ;
;
4A26: 00 04 ; SwitchTableEntry16, 26, Left Return Lane
4A28: 63 09 31 ;
4A2B: 3C 00 80 ;
4A2E: 40 00 04 ;
;
4A31: 00 04 ; SwitchTableEntry17, 27, Right Return Lane
4A33: 65 10 31 ;
4A36: 3C 00 80 ;
4A39: 40 00 04 ;
;
4A3C: 00 04 ; SwitchTableEntry18, 28, Right Outlane
4A3E: 49 D4 31 ;
4A41: 3C 00 80 ;
4A44: 40 00 04 ;
;
4A47: 0C 0C ; SwitchTableEntry19, 31, Gun Loaded
4A49: 72 44 3B ;
4A4C: 3C 05 80 ;
4A4F: F8 00 04 ;
;
4A52: 00 02 ; SwitchTableEntry1A, 32, Gun Mark
4A54: 65 A8 31 ;
4A57: 3C 00 00 ;
4A5A: B8 00 04 ;
;
4A5D: 00 02 ; SwitchTableEntry1B, 33, Gun Home
4A5F: 65 E2 31 ;
4A62: 3C 00 00 ;
4A65: F8 00 04 ;
;
4A68: 00 02 ; SwitchTableEntry1C, 34, Grip Trigger
4A6A: 5D 0A 31 ;
4A6D: 3C 00 00 ;
4A70: 40 00 04 ;
;
4A73: 00 02 ; SwitchTableEntry1D, 35, Not Used
4A75: 99 A2 FF ;
4A78: 3C 00 20 ;
4A7B: 40 00 04 ;
```

```
4A7E: 00 08 ; SwitchTableEntry1E, 36, Mid-Left Standup Target
4A80: 4F B5 31 ;
4A83: 3C 16 80 ;
4A86: 40 00 04 ;
;
4A89: 00 08 ; SwitchTableEntry1F, 37, Mid-Center Standup Target
4A8B: 4F B5 31 ;
4A8E: 3C 17 80 ;
4891: 40 00 04 ;
;
4A94: 00 08 ; SwitchTableEntry20, 38, Mid-Right Standup Target
4A96: 4F B5 31 ;
4A99: 3C 18 80 ;
4A9C: 40 00 04 ;
;
4A9F: 00 02 ; SwitchTableEntry21, 41, Left Jet
4AA1: 62 74 31 ;
4AA4: 3C 00 80 ;
4AA7: 40 00 04 ;
;
4AAA: 00 02 ; SwitchTableEntry22, 42, Right Jet
4AAC: 62 78 31 ;
4AAF: 3C 00 80 ;
4AB2: 40 00 04 ;
;
4AB5: 00 02 ; SwitchTableEntry23, 43, Bottom Jet
4AB7: 62 7C 31 ;
4ABA: 3C 00 80 ;
4ABD: 40 00 04 ;
;
4AC0: 00 02 ; SwitchTableEntry24, 44, Left Sling
4AC2: 62 E9 31 ;
4AC5: 3C 00 80 ;
4AC8: 40 00 04 ;
;
4ACB: 00 02 ; SwitchTableEntry25, 45, Right Sling
4ACD: 62 ED 31 ;
4AD0: 3C 00 80 ;
4AD3: 40 00 04 ;
;
4AD6: 00 08 ; SwitchTableEntry26, 46, Top Right Stand-up Target
4AD8: 5A 45 31 ;
4ADB: 3C 36 80 ;
4ADE: 40 00 04 ;
;
4AE1: 00 08 ; SwitchTableEntry27, 47, Mid Right Stand-up Target
4AE3: 5A 45 31 ;
4AE6: 3C 37 80 ;
4AE9: 40 00 04 ;
;
4AEC: 00 08 ; SwitchTableEntry28, 48, Bot Right Stand-up Target
4AEE: 5A 45 31 ;
4AF1: 3C 38 80 ;
4AF4: 40 00 04 ;
;
;
4AF7: 0C 0C ; SwitctchTableEntry29, 51, Left Lock
4AF9: 72 44 3E ; $7244,3E
4AFC: 3C 02 80 ;
4AFF: F8 00 04 ;
;
```

```

;
4B02: 00 08 ; SwitchTableEntry2A, 52, Not Used
4B04: 99 A2 FF ;
4B07: 3C 00 20 ;
4B0A: 40 00 04 ;
;
4B0D: 00 04 ; SwitchTableEntry2B, 53, Low Escape Route
4B0F: 4A 12 31 ;
4B12: 3C 00 80 ;
4B15: 40 00 04 ;
;
4B18: 00 04 ; SwitchTableEntry2C, 54, High Escape Route
4B1A: 4A 12 31 ;
4B1D: 3C 00 80 ;
4B20: 40 00 04 ;
;
4B23: 0C 0C ; SwitchTableEntry2D, 55, Top Lock
4B25: 72 44 3B ;
4B28: 3C 03 80 ;
4B2B: F8 00 04 ;
;
4B2E: 00 04 ; SwitchTableEntry2E, 56, Top Lane Left
4B30: 5A F9 31 ;
4B33: 3C 3E 80 ;
4B36: 40 00 04 ;
;
4B39: 00 04 ; SwitchTableEntry2F, 57, Top Lane Center
4B3B: 5A F9 31 ;
4B3E: 3C 3F 80 ;
4B41: 40 00 04 ;
;
4B44: 00 04 ; SwitchTableEntry30, 58, Top Lane Right
4B46: 5A F9 31 ;
4B49: 3C 40 80 ;
4B4C: 40 00 04 ;
;
4B4F: 00 04 ; SwitchTableEntry31, 61, Left Ramp Entry
4B51: 65 5A 31 ;
4B54: 3C 00 80 ;
4B57: 40 00 04 ;
;
4B5A: 00 04 ; SwitchTableEntry32, 62, Left Ramp Made
4B5C: 5D 44 31 ;
4B5F: 3C 00 80 ;
4B62: 40 00 04 ;
;
4B65: 00 04 ; SwitchTableEntry33, 63, Right Ramp Entry
4B67: 65 7D 31 ;
4B6A: 3C 00 80 ;
4B6D: 40 00 04 ;
;
4B70: 00 04 ; SwitchTableEntry34, 64, Right Ramp Made
4B72: 5F F5 31 ;
4B75: 3C 00 80 ;
4B78: 40 00 04 ;
;
4B7B: 00 04 ; SwitchTableEntry35, 65, Low Chase Loop
4B7D: 4A 1D 31 ;
4B80: 3C 00 80 ;
4B83: 40 00 04 ;
;
4B86: 00 04 ; SwitchTableEntry36, 66, High Chase Loop
```



```

4B88: 4A 1D 31 ;
4B8B: 3C 00 80 ;
4B8E: 40 00 04 ;
;
4B91: 00 02 ; SwitchTableEntry37, 67, Not Used
4B93: 99 A2 FF ;
4B96: 3C 00 20 ;
4B99: 40 00 04 ;
;
4B9C: 00 02 ; SwitchTableEntry38, 68, Not Used
4B9E: 99 A2 FF ;
4BA1: 3C 00 20 ;
4BA4: 40 00 04 ;
;
4BA7: 00 08 ; SwitchTableEntry39, 71, Target 1 High
4BA9: 4B 93 31 ; SwitchMatrixHdlr_Target1High()
4BAC: 3C 11 80 ;
4BAF: 40 00 04 ;
;
4BB2: 00 08 ; SwitchTableEntry3A, 72, Target 2
4BB4: 4B 95 31 ; SwitchMatrixHdlr_Target2()
4BB7: 3C 12 80 ;
4BBA: 40 00 04 ;
;
4BBD: 00 08 ; SwitchTableEntry3B, 73, Target 3
4BBF: 4B 97 31 ; SwitchMatrixHdlr_Target3()
4BC2: 3C 13 80 ;
4BC5: 40 00 04 ;
;
4BC8: 00 08 ; SwitchTableEntry3C, 74, Target 4
4BCA: 4B 99 31 ; SwitchMatrixHdlr_Target4()
4BCD: 3C 14 80 ;
4BD0: 40 00 04 ;
;
4BD3: 00 08 ; SwitchTableEntry3D, 75, Target 5 Low
4BD5: 4B 9B 31 ; SwitchMatrixHdlr_Target5Low()
4BD8: 3C 15 80 ;
4BDB: 40 00 04 ;
;
4BDE: 0C 0C ; SwitchTableEntry3E, 76, Ball Popper
4BE0: 72 44 3B ;
4BE3: 3C 04 80 ;
4BE6: F8 00 04 ;
;
4BE9: 00 02 ; SwitchTableEntry3F, 77, Drop Target
4BEB: 67 0C 31 ;
4BEE: 3C 00 80 ;
4BF1: 40 00 04 ;
;
4BF4: 10 00 ; SwitchTableEntry40, 78, Shooter
4BF6: 45 5A 31 ; SwitchMatrixHdlr_Shooter()
4B49: 3C 00 00 ;
4BFC: F8 00 04 ;
;
-----

```

For the multiball fixes, the highlighted switch handlers are updated. As can be seen, several of the switches share the same callback handler function address. Switch handlers are called with an indicator of the switch index that caused the code to be called and an indicator if the switch was closed or opened. The switch handlers can also perform a follow-up function call to check switch states.

The **green highlighted** switch handlers share common switch handler at \$7244,3B (ROM offset 0x6F244)

- Gun Loaded
- Left Lock
- Top Lock
- Ball Popper

The **blue highlighted** switch handler at \$44B1,31 (ROM offset 0x444B1) is only used by:

- Outhole

### Switch Handling, Ball Lock Switch Handler

This section covers bug fixes for the switches **highlighted in green**, in the switch-matrix table, above.

The common switch handler at \$7244,3B (ROM offset 0x6F244) is used for all 4 switches where the ball might come to a rest on the playfield. As listed above, this refers to the lock shots, ball-popper and the “gun loaded” switch.

Shown below is this common switch handler with partial annotation. Some of the annotation is speculation or commentary on what the code might be doing.

```
-----;
; Switch-Matrix Handler for:
; SwitchTableEntry19, 31, Gun Loaded
; SwitchTableEntry29, 51, Left Lock
; SwitchTableEntry2D, 55, Top Lock
; SwitchTableEntry3E, 76, Ball Popper
;
; B has index of switch
; A has 0x04 for when switch is closed (ball on switch)
; A has 0x05 for when switch is opened (ball off switch)
;
7244: 34 06      PSHS  B,A
7246: BD FE 0D      JSR   $FE0D      ; Returns with C-clear and Z-set when switch is closed
; Returns with C-set and Z-clear when switch is opened
7249: 24 45      BCC   $7290
;
; -----
; Switch is open
; -----
; -----
724B: BD 6F 0F      JSR   $6F0F      ; UpdateCurrentRunningScheduledFnWithSwOpenClosedID()
; If sw is open, calls $99A2 & returns from sw handling
; -----
724E: BD 70 9D      JSR   $709D
7251: E6 23      LDB   $0003,Y
7253: C4 FE      ANDB  #$FE
7255: E7 23      STB   $0003,Y
7257: E6 22      LDB   $0002,Y
7259: 27 2F      BEQ   $728A      ; Goes to SystemModeCheck(0x04) which launches 00AA
725B: 86 0A      LDA   #$0A      ; callback in bank $31, prior to multiball
725D: E6 61      LDB   $0001,S
725F: BD 83 46      JSR   $8346      ; Sleep()
7262: 06
7263: BD FE 0D
7266: 24 19      BCC   $7281
7268: 4A      DECA
;
```

```

7269: 26 F4      BNE  $725F      ;
726B: A6 E4      LDA  ,S          ;
726D: BD 6E E9     JSR  $6EE9      ;
7270: 6D 22      TST  $0002,Y    ;
7272: 27 06      BEQ  $727A      ;
7274: BD 82 B6     JSR  $82B6      ; ErrorHandler()
7277: 4D          ;
7278: 6F 22      CLR  $0002,Y    ;
727A: BD FD FC     JSR  $FDFC      ; SystemModeCheck()
727D: 05          ;
727E: 7E 72 D4     JMP  $72D4      ; Jump to the end, no more work
7281: A6 E4      LDA  ,S          ;
7283: BD FD FC     JSR  $FDFC      ; SystemModeCheck()
7286: 06          ;
7287: 7E 72 D4     JMP  $72D4      ; Jump to the end, no more work
;
728A: BD FD FC     JSR  $FDFC      ; SystemModeCheck()
728D: 04          ; 0x04 causes call to $6F4D,31 to evaluate A.
; If A==0x0A then 00AA scheduled function callback
; is started which causes sleep() at multiball start
728E: 20 44      BRA  $72D4      ; Jump to the end, no more work
;
;-----
;                Switch is closed
;-----
;
7290: BD 6F 0F     JSR  $6F0F      ; UpdateCurrentRunningScheduledFnWithSwOpenClosedID()
; If switch is closed returns here and code proceeds
;
7293: BD FD FC     JSR  $FDFC      ; SystemModeCheck() // Calls $70B4,3B with 0x07 in B
7296: 07          ;
7297: BD 70 9D     JSR  $709D      ;
729A: 6D 21      TST  $0001,Y    ;
729C: 26 07      BNE  $72A5      ;
;
729E: 5F          CLRB          ;
729F: BD 6E CB     JSR  $6ECB      ;
72A2: 7E 72 D4     JMP  $72D4      ; Jump to the end. In game play, no award yet
;
72A5: 6D 22      TST  $0002,Y    ;
72A7: 27 06      BEQ  $72AF      ;
72A9: BD 82 B6     JSR  $82B6      ; ErrorHandler()
72AC: 4C          INCA          ;
72AD: 20 25      BRA  $72D4      ;
72AF: E6 61      LDB  $0001,S    ;
;
72B1: 86 06      LDA  #$06      ;
72B3: BD FE 0D     JSR  $FE0D      ; -\ Called when processing switches, B has switch index
72B6: 25 16      BCS  $72CE      ; |
72B8: BD 83 46     JSR  $8346      ; | Sleep()
72BB: 06          ; |
72BC: 6D 21      TST  $0001,Y    ; |
72BE: 27 DE      BEQ  $729E      ; |
; |
72C0: 4A          DECA          ; |
72C1: 26 F0      BNE  $72B3      ; -/
;
72C3: A6 E4      LDA  ,S          ;
72C5: BD 6E DD     JSR  $6EDD      ;
72C8: BD FD FC     JSR  $FDFC      ; SystemModeCheck()
72CB: 08          ;
72CC: 20 06      BRA  $72D4      ;

```

```

72CE: A6 E4      LDA    ,S          ;
72D0: BD FD FC      JSR    $FDFC       ; SystemModeCheck()
72D3: 09              ;
72D4: 7E 6F AB      JMP    $6FAB       ; End Of Switch Processing
-----

```

The highlighted code at \$7290 is where a new function is inserted to aid in the timing problems and bug fixes. At \$7290 is where the switch handler has determined that the switch is closed, meaning the ball is resting on the switch, and game state handling is about to take place. Since this code is used by four different switches, by fixing this one function, several ways in which problems manifest are going to be fixed. The updated code at \$7290 is as follows.

```

...
;
;-----
;           Switch is closed
;-----
;
7290: BD 6F 0F      JSR    $6F0F       ; UpdateCurrentRunningScheduledFnWithSwOpenClosedID()
;-----
;           If switch is closed returns here and code proceeds
;
7290: BD 79 F9      JSR    $79F9       ; BallRestSwitch_BugFixRoutine()
;
7293: BD FD FC      JSR    $FDFC       ; SystemModeCheck() // Calls $70B4,3B with 0x07 in B
7296: 07              ;
7297: BD 70 9D      JSR    $709D       ;
729A: 6D 21      TST    $0001,Y    ;
729C: 26 07      BNE    $72A5      ;
;
...

```

As shown, the jump to \$6F0F routine was replaced to a jump to \$79F9 routine. Expectation is that the \$79F9 routine will end up returning so that normal code flow at \$7293 can proceed. Below is the new function at \$79F9,3B (ROM offset 0x6F9F9).

```

-----
; BallRestSwitch_BugFixRoutine()
;-----
; Forgot-multiball fix where lock shots will call
; this code to wait until MB init is completely done and
; multiball-loop is running.
;-----
79F9: BD 7A 06      JSR    $7A06       ; Sleep if MB is imminent or MB-init is running
79FC: BD 7A 30      JSR    $7A30       ; Sleep if "LOAD THE GUN" is imminent-ball just drained
79FF: BD 7A 61      JSR    $7A61       ; Sleeps if other lock sw is being handled during mball
7A02: BD 6F 0F      JSR    $6F0F       ; Call original code replaced with JSR to this fn.
7A05: 39              RTS              ; Now resume switch hdlr
-----

```

As indicated in the new function, above, three new functions are called, then the original \$6F0F function which was replaced with the JSR to \$79F9 is then called and code returns, resuming normal switch handler code flow. The three functions are:

- \$7A06,3B, sleep if it appears multiball is imminent, waits until MB maintenance loop is running
- \$7A30,3B, sleep if "Load the Gun" is imminent, if a ball has just drained but not fully handled yet

- \$7A61,3B, sleep if another lock switch is being handled but not fully handled yet during multiball

Effectively, the three functions will cause the code to wait until the other short-lived functions are fully handled. This will effectively cause the current switch handler to yield until the other code is finished running, whether it is multiball startup code, outhole handler code or other lock switch handler code. This allows sequential handling of these switches and prevents the trouble that was ensuing when the handlers would be ran simultaneously (*whenever one function runs a Sleep() function, the other function is allowed to run until it sleeps or returns*).

### **Switch Handler Fixup Routine: Sleep when multiball is imminent**

Shown below is the first fixup function at \$7A06,3B (ROM offset 0x6FA06).

```

-----;-----
7A06: 34 06      PSHS  B,A          ;
7A08: 86 55      LDA   #$55         ; At sleep(6), 0x55 is about 8 seconds of wait
;
;-----;
; Check inserted below for $BF being greater than 1 and,
; if so, stop waiting and return. The purpose of
; this bug fix is to yield to trough-sw handling so that
; $BF can be properly calculated to keep MB running.
; If $BF is already greater than 1 then good, done.
;-----;
7A0A: D6 BF      LDB   $BF          ; Get number of balls on the playfield from $BF
7A0C: C1 01      CMPB  #$01         ;
7A0E: 22 1E      BHI   $7A2E        ; Branches to the end if $BF is higher than 1
;
7A10: BD 86 90   JSR   $8690        ; SearchLinkedListForId() // c-bit clear = ID found
7A13: 00 84      ; 0084, ball-to-gun-target-period function
7A15: 24 0E      BCC   $7A25        ; ball-to-gun-target-period is running, keep_waiting
;
7A17: BD 86 90   JSR   $8690        ; SearchLinkedListForId() // c-bit clear = ID found
7A1A: 00 E1      ; 00E1, huntership-hit callback
7A1C: 24 07      BCC   $7A25        ; Huntership-hit callback is running, keep_waiting
;
7A1E: BD 86 90   JSR   $8690        ; SearchLinkedListForId() // c-bit clear = ID found
7A21: 00 B8      ; 00B8, multiball-init, if running, keep waiting
7A23: 25 09      BCS   $7A2E        ; not running, we're done
;
; keep_waiting:
;
7A25: 4A          DECA          ; Decrement counter
7A26: 27 06      BEQ   $7A2E        ; If decremented to zero, then done.
7A28: BD 83 46   JSR   $8346        ; Sleep()
7A2B: 06          ;
7A2C: 20 DC      BRA   $7A0A        ; Keep checking
7A2E: 35 86      PULS  A,B,PC      ; MB not imminent, or MB-loop is running, or timed out
-----;-----

```

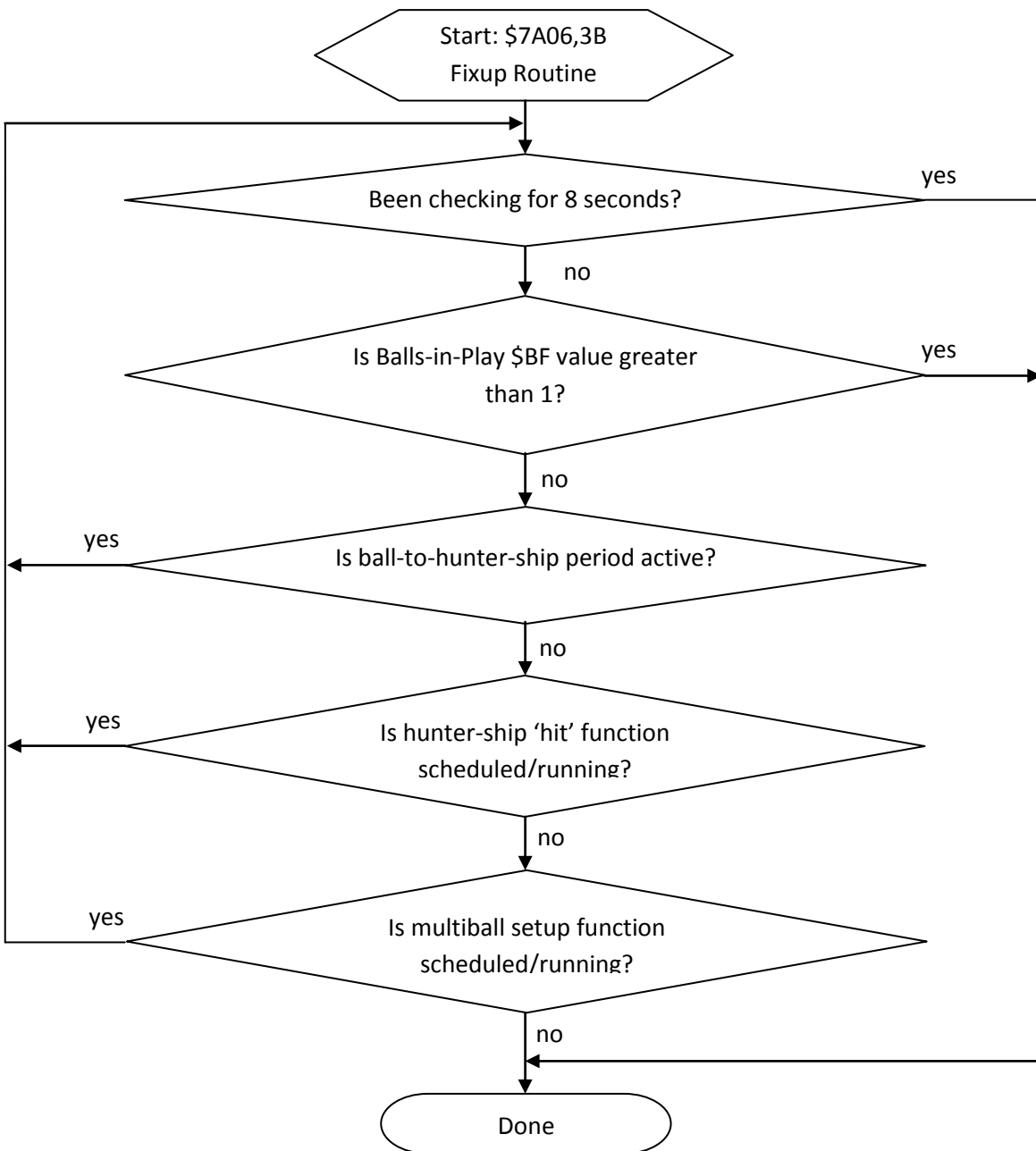
The purpose of this function, above, is to wait when any of the short-lived functions leading up to multiball maintenance loop are running. Careful survey of the code was done in order to determine the cascade of functions that occur when a multiball is about to be started:

- Function ID 00B4 function runs when gun-trigger is pulled and hunter-ship is about to be hit.
- Function ID 00E1 function runs when huntership has been hit, determines if MB should start.

- Function ID 00B8 function is the multiball startup function, described earlier.

The idea is that if any of functions are running which could possibly be leading up to multiball maintenance loop are running, this lock switch handler will sleep until the multiball actually starts (multiball maintenance loop is running) or the multiball is no longer imminent (such as if hunter ship was missed or more shots are remaining). This logic will also stop if balls-in-play \$BF value is found to be greater than 1 since the reason for this logic is to fix problems where \$BF was being stuck at 1 when a ball is locked immediately at multiball start as the ball-trough is being evacuated. If \$BF is greater than 1 then it can be reasonably assumed the 'forgotten multiball' problem is not going to take place.

Below is the logic flowchart for this function, shown for reference.





### Switch Handler Fixup Routine: Sleep when "Load the gun" is imminent

Shown below is the second fixup function at \$7A30,3B (ROM offset 0x6FA30).

```
-----;-----  
; Fix for lock-shot happening at same time as ball drain  
; which can cause lost "LOAD THE GUN" period.  
;  
7A30: 34 06      PSHS  B,A      ;  
7A32: 86 35      LDA   #$35     ; At sleep(6), 0x35 is about 5 seconds of wait  
;  
7A34: D6 BF      LDB   $BF      ; Get number of balls on the playfield from $BF  
7A36: C1 02      CMPB  #$02     ; If balls in play is not 2 then the problem of  
7A38: 26 25      BNE   $7A5F    ; lost "LOAD THE GUN" period is not imminent. Done.  
;  
7A3A: BD 86 90   JSR   $8690    ; SearchLinkedListForId() // c-bit clear = ID found  
7A3D: 00 86      ; 0086, multiball-loop is running  
7A3F: 25 1E      BCS   $7A5F    ; Multiball maintenance loop is not running, we're done  
;  
; If either 00E0 or 780F are running it means the ball  
; trough was recently loaded with fresh ball. Since  
; $BF is 02 then "LOAD THE BALL" period should not  
; be currently running but check for it anyway out  
; of completeness. If running then no need to  
; worry about having a lost "LOAD THE BALL" period.  
;  
7A41: BD 86 90   JSR   $8690    ; SearchLinkedListForId() // c-bit clear = ID found  
7A44: 00 A9      ; 00A9, "LOAD THE GUN" monitoring function  
7A46: 24 17      BCC   $7A5F    ; "LOAD THE GUN" is running, we're done  
;  
7A48: BD 86 90   JSR   $8690    ; SearchLinkedListForId() // c-bit clear = ID found  
7A4B: 00 E0      ; 00E0, outhole-set 3-second timer  
7A4D: 24 07      BCC   $7A56    ; Outhole was hit in past 3 seconds, keep waiting  
;  
7A4F: BD 86 90   JSR   $8690    ; SearchLinkedListForId() // c-bit clear = ID found  
7A52: 78 0F      ; 780F, ball-trough handler 1/2 second timer period  
7A54: 25 09      BCS   $7A5F    ; Ball-trough not hit in past 1/2 seconds, we're done  
;  
7A56: 4A         DECA          ; Decrement counter  
7A57: 27 06      BEQ   $7A5F    ; If decremented to zero, then done.  
7A59: BD 83 46   JSR   $8346    ; Sleep()  
7A5C: 06         ;  
7A5D: 20 D5      BRA   $7A34    ; Keep checking  
;  
7A5F: 35 86      PULS  A,B,PC   ; "LOAD THE GUN" period is not imminent, or timed out  
-----;-----
```

The purpose of this function, above, is to have the switch handler (which calls this function) yield to the system and allow any imminent "Load the Gun" period to become established prior to allowing the current switch handler to proceed.

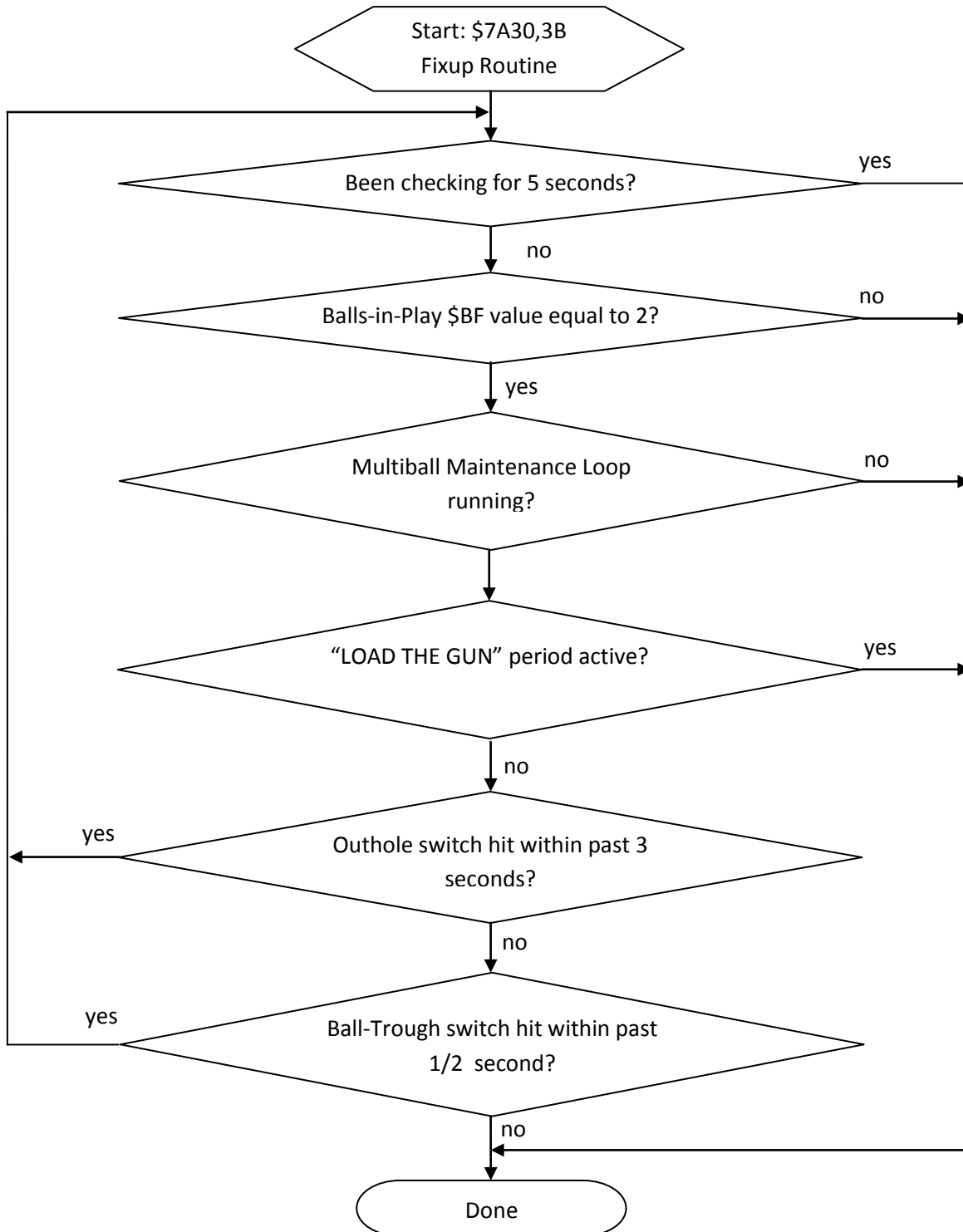
A careful examination of the code was done to understand the nature of the functions involved when the outhole switch is hit and subsequent ball-trough loading and how it relates to the issue of lost "Load the Gun" period. Having all of the following conditions means "Load the Gun" is imminent:

- The Multiball maintenance loop function ID 0086 is running, and
- The Balls-In-Play counter at ram \$BF is 2, and
- The "Load the Gun" tracking function ID 00A9 is not (yet) running, and

- The outhole switch handler scheduled function ID 00E0, 3 second timer is running, and
- The ball-trough switches handler scheduled function ID 780F for ½ second timer is running

By detecting these conditions are all present, the lock shot switch handler will assume a “Load the Gun” period is imminent. The code will wait on these conditions all being present for up to 5 seconds before giving up. This effectively fixes the problem of the lost “Load the Ball” problem.

Below is the logic flowchart for this function, shown for reference.



### Switch Handler Fixup Routine: Sleep while other switch is being serviced during multiball

Shown below is the second fixup function at \$7A61,3B (ROM offset 0x6FA61).

```
-----;-----  
; Fix bug where simultaneous lock shots of left/top lock  
; during multiball after 1 ball has already drained will  
; result in "Jackpot Multiplied 0x=0" and both balls  
; back onto playfield instead of a timed "LOAD THE GUN"  
; period (and with a single ball kicked back in play).  
;  
7A61: 34 06      PSHS  B,A  
7A63: 86 35      LDA   #$35      ; At sleep(6), 0x35 is about 5 seconds of wait  
;  
7A65: D6 BF      LDB   $BF      ; Get number of balls on the playfield from $BF  
7A67: C1 02      CMPB  #$02      ; Problem only happen during Multiball with 2 balls left  
7A69: 26 19      BNE   $7A84     ; If count is anything other than 2, we're done  
;  
7A6B: BD 86 90   JSR   $8690     ; SearchLinkedListForId() // c-bit clear = ID found  
7A6E: 00 86      ; 0086, multiball-loop is running  
7A70: 25 12      BCS   $7A84     ; Multiball maintenance loop is not running, we're done  
;  
7A72: BD 8A AA   JSR   $8AAA     ; SearchLinkedListAndMaskParameterBytes() C-clr = found  
7A75: 00 40      ; 0040 Lock-sw IDs get converted to an ID with 004x  
7A77: 01 F0      ; 01F0 Searches for ID 004x, C-clear = entry found  
7A79: 25 09      BCS   $7A84     ; If no other lock switches are currently being serviced  
; then we're done  
;  
7A7B: 4A         DECA      ; Decrement counter  
7A7C: 27 06      BEQ   $7A84     ; If decremented to zero, then done.  
7A7E: BD 83 46   JSR   $8346     ; Sleep()  
7A81: 06         ;  
7A82: 20 E1      BRA   $7A65     ; Keep checking  
;  
7A84: 35 86      PULS  A,B,PC    ; Either no other lock switch hdlr active, or timed out  
-----;-----
```

The purpose of this function, above, is to have the currently handled lock switch yield to the system until another lock switch handler, in progress, completes its work. This problem is only found to occur when there are 2 balls remaining during multiball and, as such, the function will end if the balls-in-play value at ram \$BF is anything other than 2.

A careful study of lock switch handler function was performed where it was determined the issue of "Jackpot Multiplied 0x=0" can be cured if the first lock switch that is hit is fully serviced before the second lock switch handling is allowed to commence.

The key to this logic is in how the function ID for lock switch handlers get adjusted to a 16-bit value 004x where 'x' is 4 or 5 to indicate if switch handler detected switch is closed or open, respectively. The previously depicted common lock switch handler, for four lock switches at \$7244,3B performs this function ID fixup at the following function call:

```
7290: BD 6F 0F      JSR   $6F0F     ; UpdateCurrentRunningScheduledFnWithSwOpenClosedID()  
; If switch is closed returns here and code proceeds
```

And, as mentioned, in L8.3, the instruction at \$7290 was replaced with a call to the bugfix fixup function

```
7290: BD 79 F9 JSR $79F9 ; BallRestSwitch_BugFixRoutine()
```

And, as previously depicted, the bug fix function then calls this ID updater function prior to returning at this instruction:

```
7A02: BD 6F 0F JSR $6F0F ; Call original code replaced with JSR to this fn.
```

This ID fixup function is partially annotated below, however the details of this running function ID updater is left as an exercise for the reader:

```
-----;-----  
; UpdateCurrentRunningScheduledFnWithSwOpenClosedID ()  
;  
; Called near start of switch processing  
; Function alters the scheduler ID value for current fn  
; For example, ball-popper ID starts out at 0004  
; (from SwitchMatrix table entry for the switch)  
; and this function changes it to 3044.  
;  
6F0F: 34 20 PSHS Y ;  
6F11: 8D 0D BSR $6F20 ; ScheduledFunctionOpenClosedSwitchNewIdIntoY()  
6F13: BD 9B 34 JSR $9B34 ; SearchForLinkedListEntryY() // C-clear when found  
6F16: 25 03 BCS $6F1B ; (left-lock hit at game-over C-set when sw is closed.)  
; (left-lock opened during game-over C-clear)  
;  
6F18: 7E 99 A2 JMP $99A2 ; ID Lookup success, the updated ID 3044 or 3055 is  
; already running, no work to do  
6F1B: BD 9B 83 JSR $9B83 ; UpdateCurrentRunningScheduleFunctionIDY()  
6F1E: 35 A0 PULS Y,PC ;  
-----;-----
```

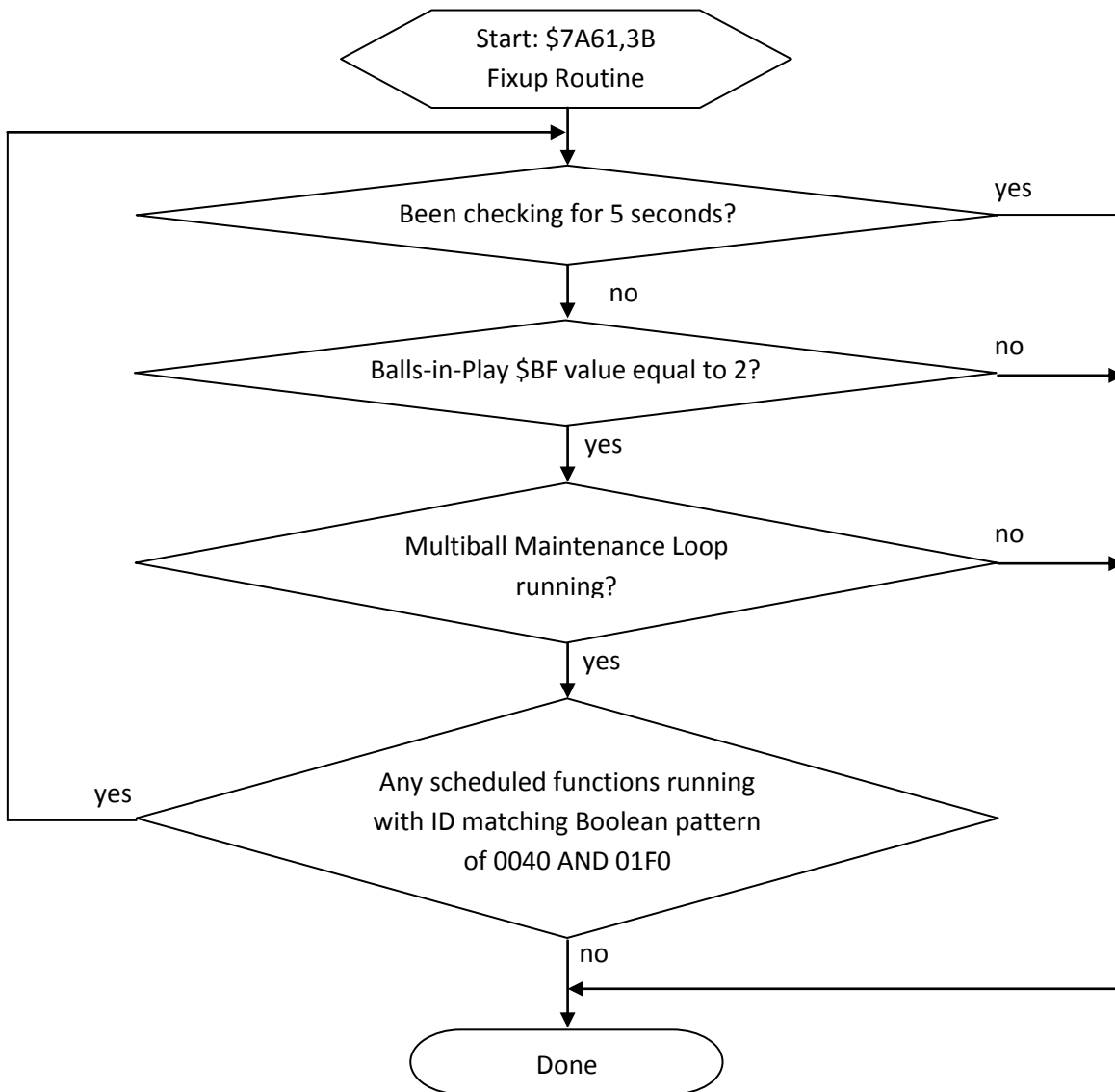
This ID fixup function relies on the next function, below, for determining new function ID value.

```
-----;-----  
; ScheduledFunctionOpenClosedSwitchNewIdIntoY()  
;  
; Populates Y with linked-list ID number,  
; based on 04/05 switch closed/open value  
; Switch Closed, Y gets 0x3044:  
; lookup ID is masked with 0x01FF --> 0x0044  
; Switch Opened, Y gets 0x3045:  
; lookup ID is masked with 0x01FF --> 0x0045  
;  
6F20: 34 06 PSHS B,A ;  
6F22: 4D TSTA ;  
6F23: 26 04 BNE $6F29 ;  
;  
; Get here if A is 0x00, invalid, a should be 04 or 05  
; ErrorHandler()  
;  
6F25: BD 82 B6 JSR $82B6 ;  
6F28: 4B ;  
6F29: 1F 89 TFR A,B ;  
6F2B: 4F CLRA ;  
6F2C: C3 00 40 ADDD #$0040 ; ID Closed sw 0004 --> 0044, Open sw 0005 --> 0045  
6F2F: 10 83 00 4F CMPD #$004F ;  
6F33: 23 04 BLS $6F39 ;  
6F35: BD 82 98 JSR $8298 ;  
6F38: 48 ;  
6F39: 8A 30 ORA #$30 ; ID Closed sw 0044 --> 3044, Open sw 0045 --> 3045  
6F3B: 1F 02 TFR D,Y ; Y gets new ID  
6F3D: 35 86 PULS A,B,PC ;  
-----;-----
```

-----

The new ID of the switch handler can be referenced by other functions, such as the bugfix function where it looks for any running function with ID value using boolean arithmetic of 0x0040 <AND> 0x01F0. A careful examination of the L-8 code was done to ensure that this pattern matches the desired search for any other lock-switch handler function that might be running.

Below is the logic flowchart for this function, shown for reference.



## Switch Handling, Outhole Switch Handler

This section covers bug fix for the switch **highlighted in blue**, in the switch-matrix table, above. For the outhole switch, a single, unique, handler is set in the switch matrix table at \$44B1,31 (ROM offset 0x444B1). The outhole switch handler code, mostly annotated, is shown below, along with its follow-up function to handle ball-saver.

```
-----;-----
; SwitchMatrixHdlr_Outhole()
; B has 0x10 SwitchTable[] index for Outhole switch
;
44B1: BD F7 59 JSR $F759 ; CheckGameMode() // z-bit set if game in progress
44B4: 7E 44 B7 JMP $44B7 ; <nop>
44B7: 26 41 BNE $44FA ; If in attract-mode go to kick the outhole solenoid
;
44B9: BD 86 90 JSR $8690 ; SearchLinkedListForId() // c-bit clear = ID found
44BC: 00 86 ; 0086, multiball-loop is running
44BE: 24 3A BCC $44FA ; If multiball mode is set, kick outhole solenoid
; until outhole switch opens.
;
44C0: BD 86 90 JSR $8690 ; SearchLinkedListForId() // c-bit clear = ID found
44C3: 00 83 ; 0083 == 7 second first multiball ball-save timer
44C5: 24 0F BCC $44D6 ; If 7-second ball-save is on, skip down to ball-save
; -----
; end-of-ball code
; -----
44C7: 86 10 LDA #$10 ; End-of-ball music
44C9: BD C0 BC JSR $C0BC ; PlayMusicRegisterA()
44CC: BD 8B 77 JSR $8B77 ; ScheduleFunctionStart()
44CF: 00 AE ; 00AE = end-of-ball bonus collection
44D1: 45 35 31 ; EndOfBallBonusCollection()
44D4: 20 24 BRA $44FA ;
; -----
; Ball-Save Engaged
; -----
44D6: BD 86 90 JSR $8690 ; SearchLinkedListForId() // c-bit clear = ID found
44D9: 00 E0 ; 00E0 == 3-second sleep
44DB: 24 1D BCC $44FA ; If 3-second sleep already running then ball was
; re-drained during ball-save timer, no 2nd ball-save
44DD: 8D 2B BSR $450A ; BallSaverSequenceEngage()
44DF: BD 83 19 JSR $8319 ; Getting switch state related function
44E2: 0F ; Possibly: SwitchTableEntry0F, 17, Trough Right
44E3: 25 15 BCS $44FA ;
44E5: BD 85 53 JSR $8553 ;
44E8: 2C ;
44E9: BD 88 D5 JSR $88D5 ;
44EC: 00 0C ;
44EE: BD 83 85 JSR $8385 ; EnqueueSolenoidPulse_ParameterByte()
44F1: 04 ; SolenoidTableEntry04, 04=Trough, 40
;
44F2: BD 8B 3D JSR $8B3D ; AddLinkedListEntry()
44F5: 00 04 ;
44F7: 70 CC 3B ; TroughLeftCenterRightSwitchHdlr()
;
44FA: BD 8B 77 JSR $8B77 ; ScheduleFunctionStart()
44FD: 00 E0 ; 00E0 == 3-second sleep
44FF: 45 2E 31 ; Schedules a 3-second sleep
4502: 86 05 LDA #$05 ; SolenoidTableEntry05, 03=Outhole, 40
4504: BD 89 2F JSR $892F ; CallBankedFunction_Param_WPCAddr_NoReturn()
4507: 6D 25 3B ; PulseSolenoidUntilSwitchOpens() A=solenoid, B=switch
```



```

;
-----;
;
; BallSaverSequenceEngage()
;
450A: BD 86 90 JSR $8690 ; SearchLinkedListForId() // c-bit clear = ID found
450D: 00 86 ; 0086, multiball-loop is running
450F: 24 15 BCC $4526 ;
4511: BD 86 90 JSR $8690 ; SearchLinkedListForId() // c-bit clear = ID found
4514: 00 EF ; 00EF == 2-second sleep timer
4516: 24 0E BCC $4526 ; If 2-second sleep already running
4518: BD 85 46 JSR $8546 ; DoSoundTableParameterByte()
451B: 99 ; 99 = "Don't Move"
451C: 25 08 BCS $4526 ;
451E: BD 8B 77 JSR $8B77 ; ScheduleFunctionStart()
4521: 00 EF ;
4523: 45 27 31 ; Schedules a 2-second sleep
4526: 39 RTS ;
;
-----;

```

The outhole switch handler, above, entails a single modification as part of the entire suite of code changes as part of resolving the multiball and “lock the ball” issues. In this case, the initial part of the outhole handler is changed as depicted below.

```

-----;
; SwitchMatrixHdlr_Outhole()
; B has 0x10 SwitchTable[] index for Outhole switch
;
44B1: BD F7 59 JSR $F759 ; CheckGameMode() // z-bit set if game in progress
44B4: 7E 44 B7 JMP $44B7 ; <nop>
44B7: 26 41 BNE $44FA ; If in attract-mode go to kick the outhole solenoid
;
44B4: 26 44 BNE $44FA ; If in attract-mode go to kick the outhole solenoid
44B6: BD 7F 9B JSR $7F9B ; Jump to bugfix routine that sleeps if MB is imminent
;
...

```

The original code consisted of:

- Dummy JMP instruction to get to next instruction, and
- BNE instruction to branch to \$44FA if the result of previously called \$F759 cleared the Z-bit

The new code consists of:

- BNE instruction to branch to \$44FA if the result of previously called \$F759 cleared the Z-bit
- JSR instruction to a new bug-fix function.

Since the bug-fix needs to be called after the BNE, it was easy to move the BNE instruction to prior to the dummy JMP instruction (and modify the number of bytes the BNE needs to skip) and then change the dummy JMP instruction to a genuine JSR instruction to the bug-fix function. The implied behavior now, is that the bug-fix function at \$7F9B will return when it completes so that normal code-flow can proceed in the outhole switch handler function.

*Note: Throughout the L-8 code in many places a call to a function in non-banked ROM region is often followed by a “dummy” instruction to jump to the next instruction. It appears this is a way the original L-8 code can use to easily allow for a function to be moved out of non-banked ROM into banked ROM, if needed. The “dummy” jump instruction takes up 3 bytes of ROM space which would be needed if the function call into a banked function is needed instead. The dummy JMP instruction is effectively a NOP instruction (no-operation) and can be repurposed, safely, in this way.*

The new bug-fix function at \$7F9B,31 (ROM offset 0x44F9B) is as follows:

```
-----;
;
;
; Jump here from $44B4/$44B6 to call function in bank $3B where 'forgotten-multiball' fix
; function is in place to wait until all short-lived functions leading up to multiball
; main loop are done running, or Balls-in-play $BF is greater than 1, or up to a timeout.
;
7F9B: BD 88 F5    JSR    $88F5          ; CallBankedFunction_Param_WPCAddr()
7F9E: 7A 06 3B    ; Forgotten-multiball bugfix
7FA1: 39          RTS
-----;
```

The bug-fix function for outhole switch handler is in place to help with the problem of “forgotten multiball” when ball is drained immediately as the hunter ship is hit. This function will call the bug-fix function at \$7A06,3B (ROM offset 0x6FA06) which will check if multiball is imminent (but balls-in-play value at ram \$BF is 1) waits until multiball is no longer imminent or \$BF value is greater than 1.

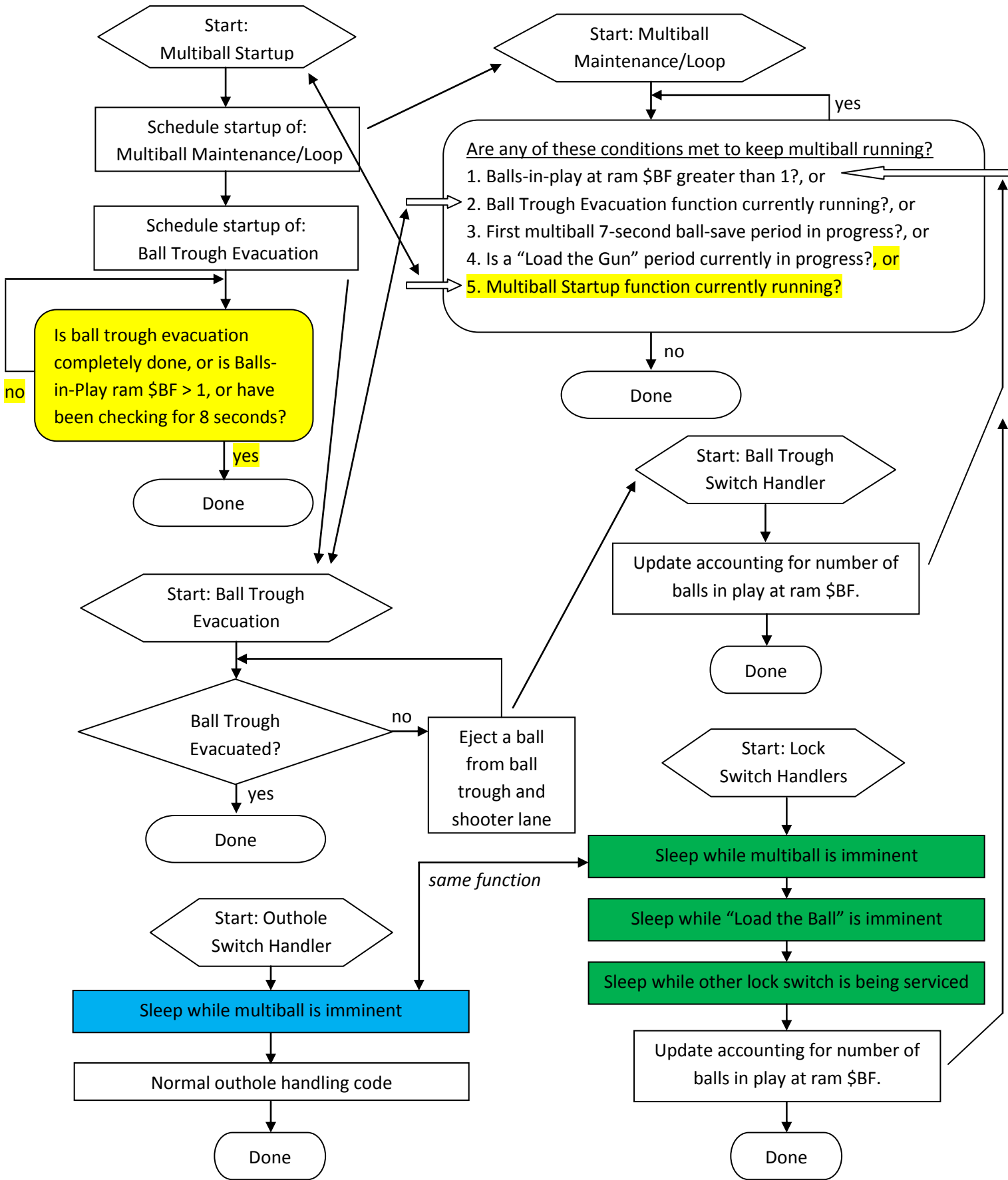
This logic will prevent problems with the multiball prematurely ending when the Multiball maintenance loop would discover the \$BF value is 1 and declare end of multiball unexpectedly. By having the outhole switch yield to the system in this way, and along with the other lock-switch handler bug-fix logic, the multiball loop will be sure to not end prematurely.

The full description of the bug-fix function, \$7A06,3B, is shown earlier where this same function is also called as part of the common lock-switch handler bug-fix function.

## Multiball Corrected Logic

The resulting flowchart depicting the various fixes is shown below. This includes:

- **Yellow highlighted** fixes for multiball startup and maintenance loop
- **Green highlighted** fixes for the lock-switches handler code
- **Blue highlighted** fixes for the outhole-switch handler code



## The L8.3 Lamp Driver Update

As part of added enhancement to L8.3, support was added to allow the selection of the desired lamp driver code, allowing two possible values:



The original L-8 lamp driver code is the original code produced for L-8 which was originally intended for incandescent bulbs (as LED pinball illumination was non-existent when L-8 was made).

The LED driver code utilizes code based on the publically available no-ghost patch as documented in a publically downloadable file “WPC\_Ghost\_Busting.pdf”, with an additional improvement added in L8.3 to fix a ghosting problem that could happen during GI power-saving mode, as described in detail, below.

A big THANK YOU to those involved in investigating and developing the LED driver, “no-ghost” patch. The pinball community has greatly benefited from your work and, now, the L8.3 software benefits from your efforts. Your time spent documenting the issue and showing the oscilloscope measurements were immensely helpful. The documented patch, and the patch tool results served as the primary basis of the LED driver code used in L8.3.

### LED Patch Summary

Refer to online resources, such as “WPC\_Ghost\_Busting.pdf”, for technical details of the WPC lamp driver patch. In a nutshell, the patched lamp driver has these characteristics:

- Set the Column drivers to 0 and then the row drivers to 0
  - This must be done by clearing B register and then storing B into Column & Row registers
- Have about 30us worth of instructions take place
- Put the derived lamp row value into the Row register
- Put the derived column bit/register value into the Column register

The improved logic, above cures the issues identified as causing the current spikes responsible for the ghosting.

### LED Patch Improvement, power-saver improvement

While analyzing the patched lamp matrix and reviewing information published in the “WPC\_Ghost\_Busting.pdf”, it became apparent, during L8.3 development, that the patched lamp matrix code had missed another location in its code where it turns off the Column and Row registers during the GI Power Saver mode.

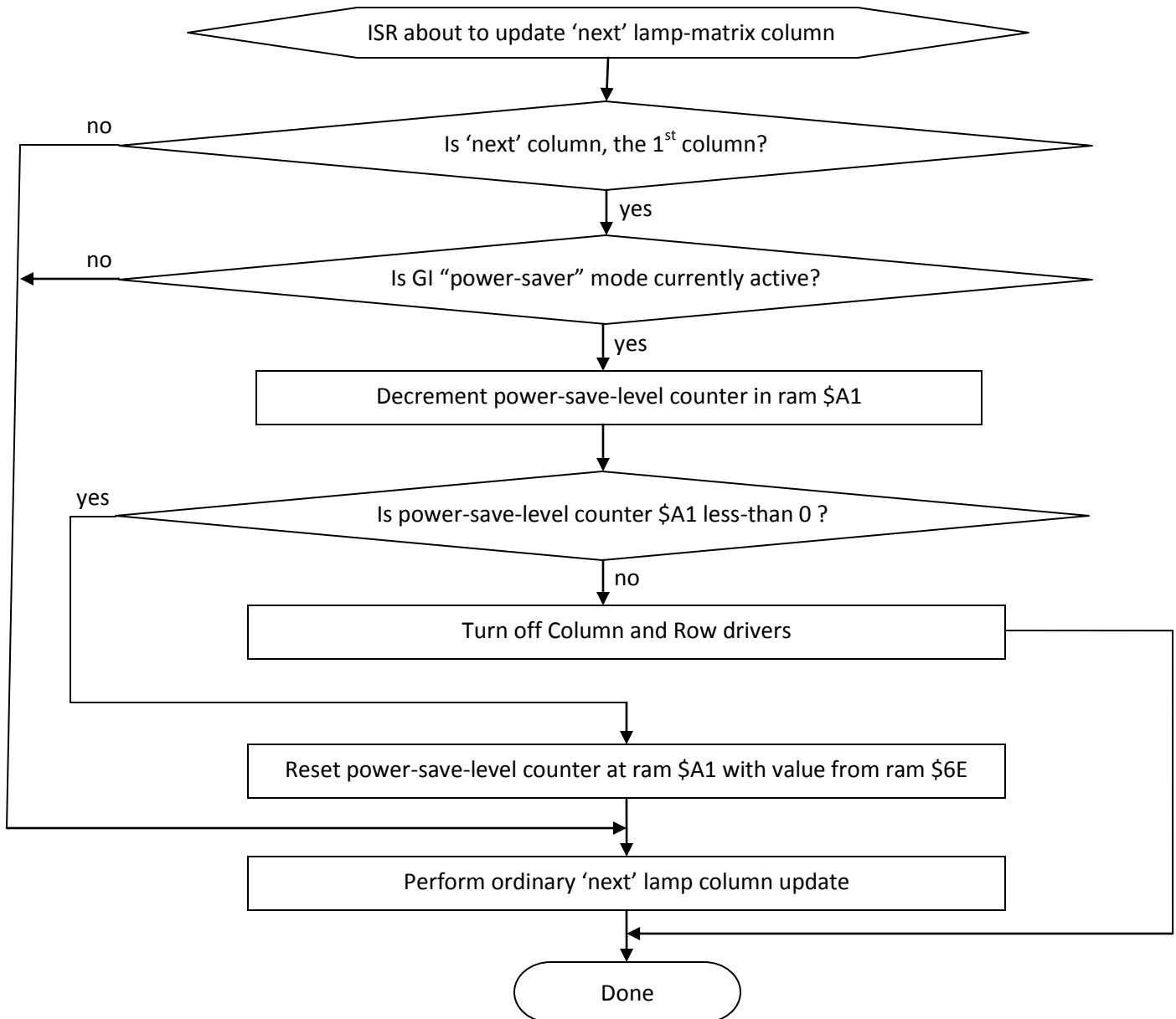
It turns out, that the WPC Lamp Matrix is affected by the GI Power Saver mode, thus making the menu text somewhat incorrect in its naming since the power-saver function applies to the 8x8 controlled lamp matrix in addition to the general illumination.

The power-saver mode will engage in the following circumstances:

- Standard Adjustment A1.25 “Allow Dim Illum.” is set to “Yes”, and
- Standard Adjustment A1.29 “GI Power Saver” is set to value other than “Off”, and
- Number of minutes configured in A1.29 has elapsed.

As described earlier in this document, the Interrupt Service Routine (ISR) is called periodically. One of the tasks the ISR performs, in addition to reading the switch matrix, is to update the ‘next’ column of the lamp matrix. Using the ISR, the lamp matrix is updated on a regular, repeating basis, updating only a single column on a given pass through the ISR routine.

When the power-saver mode is active, the ISR lamp-matrix updater will utilize the following logic when it is about to update the ‘next’ column of the lamp matrix.



As shown in the flowchart, above, when the machine is in power-saver mode, after the lamp matrix has updated the 8<sup>th</sup> column and about to start over at the 1<sup>st</sup> column, the power-saver mode is checked and, if active, the lamp matrix is turned off for that update. Both the column and row registers are cleared which effectively turns off the lamp matrix. This repeats for subsequent passes through the logic until the value from ram \$A1 has counted down from 0 to -1. Once the \$A1 value is decremented to -1 then the \$A1 value gets reset and the column 1 lamps are enabled and remainder of the lamp matrix is updated at subsequent ISR passes. After all 8 columns have been updated, this power-saver mode repeats by disabling the lamp matrix for the number passes set in ram \$A1 again, repeating until power-saver mode is no longer active.

**Note:** *The power-saver timer has been observed to utilize a granularity of 1-minute. For example, if the power-saver time is set to 5 minutes, the power-saver will engage between 4:01 and 5:59 after entering attract mode. This is likely the reason the minimum timer value is allowed to be 2 minutes.*

As indicated, the lamp-matrix will set columns and rows off for a number of passes as derived from the power-save level adjustment value. This value is set in RAM at \$6E and used as the reset value into the countdown value at \$A1. The corresponding values for this countdown value are as follows:

Standard Adjustment A1.30 "Power Save Level"	Lamp Matrix power-saver reset value \$6E
4	3
5	2
6	2
7	2

The idea is to save on power usage of the lamp-matrix by turning off all lamps for a tiny, hopefully unperceivable, amount of time. Ironically, the way in which the columns and rows are turned off would incur a current spike and resulting lamp flicker each time the columns and rows are turned off for the first time. A careful examination of the scenario would be needed to determine if this actually utilizes more energy than if the power-saver function didn't run at all.

The way in which the lamp matrix turns off the columns and rows is highlighted in the following code:

```
DABA: 7F 3F E4    CLR  $3FE4          ; $3FE4=WPC_LAMP_ROW, turns off all row bits
DABD: 7F 3F E5    CLR  $3FE5          ; $3FE5=WPC_LAMP_COLUMN, turns off all column bits
```

As described in the "WPC\_Ghost\_Busting.pdf" the use of a "CLR" instruction directly on the ROW register causes a spike of current due to a problem in the WPC ASIC which briefly turns on all of the bits prior to turning them off. *Again, thank you to those who have paved the way and discovered this information and published it for the pinball community!*

For L8.3 LED driver, this has been corrected by using the following instructions.

```
xxxx: 4F          CLR  A              ; A gets 0x00
xxxx: B7 3F E4    STA  $3FE4          ; $3FE4=WPC_LAMP_ROW, turns off all row bits
xxxx: B7 3F E5    STA  $3FE5          ; $3FE5=WPC_LAMP_COLUMN, turns off all column bits
```

As shown, the same fix as used in the no-ghost patch is used. First a register, A, is cleared and then the register is stored into the Row and Column registers. This resolves the WPC ASIC issue that would otherwise ensue when the instruction is to CLR the Row/Column registers directly.

## Lamp Driver Code Modifications

For reference, shown below is the L-8 original lamp matrix driver code, partially annotated.

```

-----;-----
; Start of Lamp Matrix code
;
DAA9: 9E 9F      LDX  $9F      ; Get current lamp RAM address from $9F:$A0 into X
DAAB: 30 01      LEAX $0001,X  ; Increment to next lamp RAM address
DAAD: 96 9E      LDA  $9E      ; Get current column/row bit
DAAF: 48         ASLA                    ; Rotate left the column or row bit in $9E
DAB0: 26 19      BNE  $DACB    ; If haven't done 8, go to $DACB to continue the sweep
DAB2: 96 6D      LDA  $6D      ; Load $6D value to see if in power-saver mode
DAB4: 27 10      BEQ  $DAC6    ; If $6D is 0x00, not in power-saver mode, reset sweep
DAB6: 0A A1      DEC  $A1      ; If $6D is not 0x00, in power-saver mode, decrement $A1
DAB8: 2B 08      BMI  $DAC2    ; Power-saver counter < 0, reset counter and reset sweep
DABA: 7F 3F E4   CLR  $3FE4    ; $3FE4=WPC_LAMP_ROW, turns off all row bits
DABD: 7F 3F E5   CLR  $3FE5    ; $3FE5=WPC_LAMP_COLUMN, turns off all column bits
DAC0: 20 55      BRA  $DB17    ; Jump to post lamp-matrix, all done for this pass
DAC2: 96 6E      LDA  $6E      ; Get the power-saver off-cycle count from $6E
DAC4: 97 A1      STA  $A1      ; and put it into the $A1 for next time
; Now proceed with a regular matrix column update
DAC6: 8E 02 E0   LDX  #$02E0    ; Get start of lamp data RAM address 0x02E0 into X
DAC9: 86 01      LDA  #$01      ;
;
DACB: 9F 9F      STX  $9F      ; Store current or initial lamp ram address into $9F:$A0
DACD: 97 9E      STA  $9E      ; Store next or initial column bit into $9E
;
DACF: E6 89 00 10 LDB  $0010,X  ;
DAD3: 53         COMB                    ;
DAD4: E4 89 00 08 ANDB $0008,X  ;
DAD8: D7 9C         STB  $9C      ;
DADA: E6 89 00 10 LDB  $0010,X  ;
DADE: E4 89 00 18 ANDB $0018,X  ;
DAE2: DB 9C         ADDB $9C      ;
DAE4: D7 9C         STB  $9C      ;
DAE6: E6 89 00 20 LDB  $0020,X  ;
DAEA: E4 89 00 28 ANDB $0028,X  ;
DAEE: D7 9D         STB  $9D      ;
DAF0: E6 89 00 20 LDB  $0020,X  ;
DAF4: 53         COMB                    ;
DAF5: D4 9C         ANDB $9C      ;
DAF7: DB 9D         ADDB $9D      ;
DAF9: D7 9C         STB  $9C      ;
DAFB: E6 89 00 30 LDB  $0030,X  ;
DAFF: E4 89 00 38 ANDB $0038,X  ;
DB03: D7 9D         STB  $9D      ;
DB05: E6 89 00 30 LDB  $0030,X  ;
DB09: 53         COMB                    ;
DB0A: D4 9C         ANDB $9C      ;
DB0C: DB 9D         ADDB $9D      ;
DB0E: 7F 3F E4   CLR  $3FE4    ; $3FE4=WPC_LAMP_ROW, this turns off all row bits
DB11: B7 3F E5   STA  $3FE5    ; $3FE5=WPC_LAMP_COLUMN, turns on next column bit
DB14: F7 3F E4   STB  $3FE4    ; $3FE4=WPC_LAMP_ROW, turns on rows that should be on
-----;-----

```



For L8.3, to cause the lamp driver code to choose between this original or a new LED driver code, the original code is modified with a jump to a new function. The change is made in the following way.

```

-----;-----
; Start of Lamp Matrix code
;
DAA9: 9E 9F      LDX  $9F      ; Get current lamp RAM address from $9F:$A0 into X
DAAB: 30 01      LEAX $0001,X  ; Increment to next lamp RAM address
DAA9: 7E FE DD    JMP  $FEDD    ; Jump to new code to choose original or LED driver
DAAC: 12         NOP                    ; Unused no-op, new code jumps past this to return
DAAD: 96 9E      LDA  $9E      ; Get current column/row bit
DAAF: 48         ASLA                    ; Rotate left the column or row bit in $9E
DAB0: 26 19      BNE  $DACB    ; If haven't done 8, go to $DACB to continue sweep
DAB2: 96 6D      LDA  $6D      ; Load $6D value to see if in power-saver mode
...

```

As shown in the new highlighted code, the first two instructions of the original lamp driver are replaced with a JMP instruction to a new function located at \$FEDD (ROM offset 0x7FEDD). Since two 2-byte instructions needed to be replaced with a single new 3-byte instruction, the leftover 4<sup>th</sup> byte is changed to a NOP instruction. This NOP never gets executed because the new \$FEDD will jump back to this original lamp driver (when appropriate do to so) at address \$DAAD, skipping the NOP.

The new function at \$FEDD (ROM offset 0x7FEDD) is as follows. Below contains partially annotated code and number of cycles used by some of the instructions, for reference.

```

-----;-----
FEDD: 9E 9F      LDX  $9F      ; Get current lamp RAM address from $9F:$A0 into X
FEDF: 30 01      LEAX $0001,X  ; Increment to next lamp RAM address
;
; Directly read Adjustment Value from RAM for
; Lamp Driver from $1BED and proceed with original
; or LED lamp driver code.
;
FEE1: F6 1B EC    LDB  $1BEC    5cy ; Read adjustment byte 0=no patch
FEE4: 26 03      BNE  $FFE9    3cy ;
FEE6: 7E DA AD    JMP  $DAAD    4cy ; Go back to original ISR code
;
FEE9: 96 9E      LDA  $9E      ; Get current column/row bit
FEEB: 48         ASLA                    ; Rotate left the column or row bit in $9E
FEEC: 26 1A      BNE  $FF08    ; If haven't done 8, go to $FF06 to continue the sweep
FEEE: 96 6D      LDA  $6D      ; Load $6D value to see if in power-saver mode
FEF0: 27 11      BEQ  $FF03    ; If $6D is 0x00, not in power-saver mode, reset sweep
FEF2: 0A A1      DEC  $A1      ; If $6D is not 0x00, in power-saver mode, decrement $A1
FEF4: 2B 09      BMI  $FEFF    ; Power-saver counter < 0, reset counter and reset sweep
FEF6: 4F         CLRA                    ;
FEF7: B7 3F E4    STA  $3FE4    ; $3FE4=WPC_LAMP_ROW, turns off all row bits
FEFA: B7 3F E5    STA  $3FE5    ; $3FE5=WPC_LAMP_COLUMN, turns off all column bits
FEFD: 20 53      BRA  $FF52    ; Jump to post lamp-matrix, all done for this pass
FEFF: 96 6E      LDA  $6E      ; Get the power-saver off-cycle count from $6E
FF01: 97 A1      STA  $A1      ; and put it into the $A1 for next time
FF03: 8E 02 E0    LDX  #02E0    ; Get start of lamp data RAM address 0x02E0 into X
FF06: 86 01      LDA  #$01      ;
;
FF08: 9F 9F      STX  $9F      ; Store current or initial lamp ram address into $9F:$A0
FF0A: 97 9E      STA  $9E      ; Store next or initial column bit into $9E
;
FF0C: E6 88 10    LDB  $10,X    ;
FF0F: 53         COMB                    ;

```

```

FF10: E4 88 08    ANDB  $08,X      ;
FF13: D7 9C      STB   $9C        ;
FF15: E6 88 10    LDB   $10,X      ;
FF18: E4 88 18    ANDB  $18,X      ;
FF1B: DB 9C      ADDB  $9C        ;
FF1D: D7 9C      STB   $9C        ;
FF1F: E6 88 20    LDB   $20,X      ;
FF22: E4 88 28    ANDB  $28,X      ;
FF25: D7 9D      STB   $9D        ;
FF27: 5F         CLRB                ;
FF28: F7 3F E5    STB   $3FE5      ; $3FE5=WPC_LAMP_COLUMN, turns off all column bits
FF2B: F7 3F E4    STB   $3FE4      ; $3FE4=WPC_LAMP_ROW,    turns off all row bits
FF2E: E6 89 00 20 LDB   $0020,X    4+4 ;
FF32: 53         COMB                2cy ;
FF33: D4 9C      ANDB  $9C        4cy ;
FF35: DB 9D      ADDB  $9D        4cy ;
FF37: D7 9C      STB   $9C        4cy ;
FF39: E6 89 00 30 LDB   $0030,X    4+4 ;
FF3D: E4 89 00 38 ANDB  $0038,X    4+4 ;
FF41: D7 9D      STB   $9D        4cy ;
FF43: E6 89 00 30 LDB   $0030,X    4+4 ;
FF47: 53         COMB                2cy ;
FF48: D4 9C      ANDB  $9C        4cy ;
FF4A: DB 9D      ADDB  $9D        4cy ; 60 cycles total, ~30uS, every 2 cycles is ~1uS
FF4C: F7 3F E4    STB   $3FE4      ; $3FE4=WPC_LAMP_ROW,    turns on rows that should be on
FF4F: B7 3F E5    STA   $3FE5      ; $3FE5=WPC_LAMP_COLUMN, turns on next column bit
;
; Jump back to post-lamp-matrix code
FF52: 7E DB 17    JMP   $DB17      ; Go back to original ISR code post lamp-matrix code
-----;

```

Notable elements of the LED lamp driver, above:

- First 2 instructions are the same 2 instructions that were replaced with the JMP to this code
- “Lamp Driver” value read directly from ram and not using the formal WPC lookup function.
  - This is done to have code run as fast as possible.
  - Single-byte “Lamp Matrix” is at known, fixed location in Ram at \$1BEC
- If adjustment is set to 0 (Original) code jumps back to \$DAAD to resume original lamp driver
- If adjustment is non-zero, new LED driver code ensues
- LED driver is a copy of the original driver code with no-ghost patch applied
  - Also the GI power-saver bug fix, as previously described, is applied.
- After LED driver updates a column (or a power-saver ‘off’ cycle) code does a JMP to \$DB17
- The \$DB17 is address after the original lamp driver code, resuming normal ISR code.

### Relocated Copyright Message

The updated LED lamp driver code, above, was placed into the region of ROM formally occupied by the copyright message. The copyright message was present in the ROM for the observer but not utilized by the code. The original, unmodified, copyright message was moved in its entirety to a location immediately prior to the non-banked region of ROM in a unused region starting at \$7DF0,3D (ROM offset 0x77DF0) and appears as follows:

00077DE0	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	YYYYYYYYYYYYYYYY
00077DF0	20 43 6F 70 79 72 69 67 68 74 20 28 63 29 20 31	Copyright (c) 1
00077E00	39 39 32 2C 20 31 39 39 31 2C 20 31 39 39 30 20	992, 1991, 1990
00077E10	57 69 6C 6C 69 61 6D 73 20 45 6C 65 63 74 72 6F	Williams Electro
00077E20	6E 69 63 73 20 47 61 6D 65 73 20 49 6E 63 2E 20	ronics Games Inc.
00077E30	20 41 6C 6C 20 52 69 67 68 74 73 20 52 65 73 65	All Rights Rese
00077E40	72 76 65 64 20 20 53 79 73 74 65 6D 20 53 6F 66	rved System Sof
00077E50	74 77 61 72 65 20 62 79 20 4C 61 72 72 79 20 44	tware by Larry D
00077E60	65 4D 61 72 2C 20 42 69 6C 6C 20 50 66 75 74 7A	eMar, Bill Pfutz
00077E70	65 6E 72 65 75 74 65 72 2C 20 54 65 64 20 45 73	enreuter, Ted Es
00077E80	74 65 73 20 26 20 4D 61 72 6B 20 50 65 6E 61 63	tes & Mark Penac
00077E90	68 6F 20 20 45 6C 65 63 74 72 6F 6E 69 63 73 20	ho Electronics
00077EA0	62 79 20 43 68 75 63 6B 20 42 6C 65 69 63 68 20	by Chuck Bleich
00077EB0	61 6E 64 20 4D 61 72 6B 20 43 6F 6C 64 65 62 65	and Mark Coldebe
00077EC0	6C 6C 61 20 FF FF FF FF FF FF FF FF FF FF FF	lla YYYYYYYYYYYY
00077ED0	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	YYYYYYYYYYYYYYYY

The copyright message was moved due to lack of available free space in the non-banked region of the ROM. No disrespect was intended with this message move. The L8.3 was crafted with all respect given to those who contributed to the original T-2 game and software design.

### Lamp Driver Results

With the new lamp driver in place, it has been confirmed that no LED ghosting takes place when feature adjustment A2.25 “Lamp Driver” is set to “LED”. Also, when the power-saver mode engages, it has also been confirmed that lamp-matrix LEDs don’t experience ghosting at such time as well.

Note that using LEDs in general illumination strings will still experience flickering due to the way in which Triacs are used along with sine-wave “zero-cross” signal to cause a chopped sine-wave at the GI sockets. A conversion of AC to DC would be needed to achieve smoother GI dimming with LEDs.

Because of this, it is suggested that the game is set up to with standard adjustment A1.25 “Allow Dim Illum” is set to “No” if LEDs are being used in GI without external voltage-smoothing circuitry.

## The L8.3 Multiball-Start Drop-Target Action Enhancement

This section describes a new adjustment added to L8.3 called “Multiball-Start Drop-Target Action”. Before details of this new adjustment, the following will describe the existing “Drop Target Down Multiball” to help make it clear the distinct difference between these two adjustments.

### Drop Target Down Multiball Adjustment

Most T2 owners have come across this interesting adjustment in the Feature Adjustments menu:



The published T2 manual was produced prior to L-4 where this adjustment made its first appearance. Because of this, it may not be immediately apparent how this adjustment affects game play.

A thorough examination of the T2 code was made to determine how this setting is used. What was found was that this adjustment’s value is read during game play during multiball when the ball-popper is hit. When a ball hits the ball-popper (skull shot) during multiball, this adjustment’s value is then read:

- If “Off”, the drop-target is immediately kicked up
- If “On”, the drop-target is not kicked up (i.e. remains down)

Evidently, the “Off” behavior is to help prevent a second ball from crashing into the locked ball and possibly getting stuck somewhere up inside the skull assembly. For game operators who don’t like this behavior, the “On” setting allows the drop-target to remain in the down position while the ball is locked in the ball-popper.

### Multiball-Start Drop-Target Action Adjustment

In L8.3 a new adjustment was added:



This setting allows the game operator to control the drop-target up/down state at multiball start. The reason such a setting was added was to improve the balance of the game-play so that the multiball experience can be made more consistent regardless if the multiball was started via:

- Ball Popper and Hunter Ship hit, or
- Database award, or
- Left Loop award

By setting this adjustment to anything other than “None”, the drop-target state will be up or down at start of multiball for all methods of multiball start, based on the adjustment value and, for some adjustment values, the number of multiballs that the player has achieved so far.

This feature entails having the game software kick the drop target up, or knocking it down at multiball start. The drop-target switch and knock-down solenoid must be functioning properly for this feature to work correctly.

Possible values for this new adjustment are as follows:

- None, no particular drop-target action is taken at multiball start (same as L-8)
- Down, at every multiball start, drop-target is knocked down
- Up, at every multiball start, drop-target is reset (kicked up)
- 1 MB Down, At first multiball the drop-target is knocked down. Reset at subsequent multiballs.
- 2 MB Down, First 2 multiballs the drop-target is knocked down. Reset at subsequent multiballs.
- 3 MB Down, First 3 multiballs the drop-target is knocked down. Reset at subsequent multiballs.

The various adjustment values provide added flexibility in game play, allowing the operator to dial in the preferred behavior and difficulty of multiball experience, regarding drop-target state at multiball start.

### Multiball-Start Drop-Target Action Adjustment Code

The entry point for the drop-target action at multiball start takes place inside the multiball startup function. The multiball startup function is shown in full in the previous section “Multiball Startup Balls-In-Play Timing Fix: Startup waits for non-zero balls-in-play”. Below is the relevant portion of the multiball startup function related to this feature:

```

...
6C98: 8E 05 C9    LDX    #$05C9           ; $05C9, Base Addr of #-of-multiballs per player/game
6C9B: BD FB 29    JSR    $FB29           ; IncrementXByPlayerIndexNumber()
6C9E: 7E 6C A1    JMP    $6CA1           ; <nop> JSR to new function here JSR to $FB8C
6CA1: 6C 84      INC    ,X              ; Increment multiballs achieved counter for cur player
...

```

This part of multiball startup function loads up the base address in RAM where the game saves the number of multiballs that each of the 4 players has had so far during the game. This value does not reset between balls. The function at \$FB29 increments the pointer based on the current player number. The result when \$FB29 returns is that X points to RAM byte containing the number of multiballs the current player has achieved:

Player Number	RAM Address for: Multiballs achieved current game
1	\$05C9
2	\$05CA
3	\$05CB
4	\$05CC

For this feature in L8.3, this portion of the multiball startup code is updated as follows:

```

...
6C98: 8E 05 C9    LDX  #$05C9          ; $05C9, Base Addr of #-of-multiballs per player/game
6C9B: BD FB 29    JSR  $FB29          ; IncrementXByPlayerIndexNumber()
6C9E: BD 7F BA    JSR  $7FBA          ; Jump to Multiball Start Drop-Target Action routine
6CA1: 6C 84      INC  ,X              ; Increment multiballs achieved counter for cur player
...

```

As highlighted, at \$6C9E there was previously a dummy JMP instruction which simply jumped to the next instruction at \$6CA1. This dummy instruction was replaced with a jump to a new function, \$7FBA, that will handle the new feature by setting the drop target state according to the “Multiball Start Drop-Target Action” adjustment. Note, above, at \$6CA1, the number of multiballs achieved for the current player gets incremented *after* the new fixup function is called. This means the multiball counter that the function gets is the number of multiballs achieved *not* including the current multiball that is starting up.

The new function at \$7FBA (ROM offset 0x47FBA) is located in a region later in the same bank \$31 that was previously unused and is as follows:

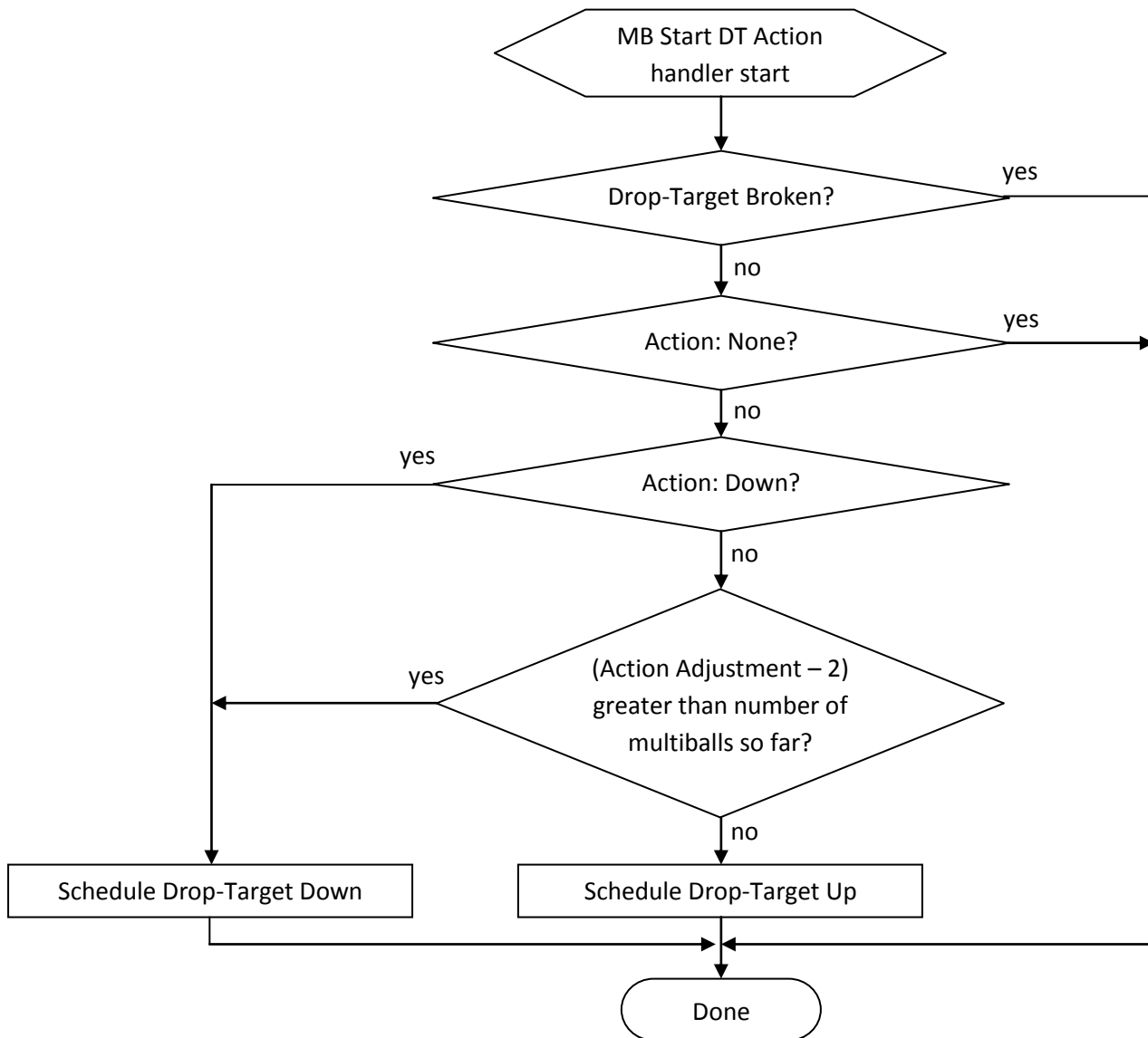
```

-----;-----
7FBA: 34 04      PSHS B              ;
7FBC: BD 86 5B   JSR  $865B          ; LookupGameAdjustmentParamlandCheckIfEqualsParam2()
7FBF: 14 00      ; 0x14, FeatureAdjustment020, Drop Trgt. Broken
                          ; C-bit set when not-equal
7FC1: 25 23      BCS  $7FE6          ; Not-equal to 0x00 then dt broken is "yes", we're done
                          ;
7FC3: BD 83 0C   JSR  $830C          ; Get8BitSettingIntoBParameterByte()
7FC6: 1A        ; 0x1A, FeatureAdjustment026, MB Start DT Action
7FC7: 5D        TSTB                ;
7FC8: 27 1C      BEQ  $7FE6          ; If B is 0x00 then no action, return
7FCA: C1 01      CMPB #$01          ; Check if B is 0x01 "Down"
7FCC: 26 0A      BNE  $7FD8          ; If B is not 0x01, skip down to $7FD8
                          ;
7FCE: BD 8B C3   JSR  $8BC3          ; ScheduleFunctionCallback()
7FD1: 00 B7      ; 0x00B7 Unique id for the target-down callback
7FD3: 78 94 3B   ; WPC Address for drop-target-down callback
7FD6: 20 0E      BRA  $7FE6          ; Jump to done
                          ;
7FD8: C0 02      SUBB #$02          ; Here when B is 0x02 or more, decrement it by 2.
7FDA: E1 84      CMPB ,X            ; Compare B with X, number of MBs so far for cur player
7FDC: 2E F0      BGT  $7FCE          ; If B-register was greater than the MB, do target-down
                          ;
7FDE: BD 8B C3   JSR  $8BC3          ; ScheduleFunctionCallback()
7FE1: 00 B9      ; 0x00B9 Unique id for the target-up callback
7FE3: 78 B7 3B   ; WPC Address for drop-target-up callback
7FE6: 35 84      PULS B,PC          ;
-----;-----

```

The drop-target ‘action’ function, above, is called with ‘X’ register containing the number of multiballs the player as achieved so far, not including the current MB that is starting. For first multiball, for example, X has 0. This function checks the configured “Drop Target Broken” and will not do anything if configured to “yes”. After this, the new “MB Start DT Action” adjustment value is read. If configured to 0 (None), then no particular action takes place. If 1 (Down) then DT is scheduled to go down. Otherwise, the code will treat the configured value as number of prior multiballs that must have taken place as the threshold to push drop-target UP or kick it down. Effectively, the “Up” setting is treated same as if it were named “0 MB Down”.

The flowchart for the Drop-Target Action function is as follows:



The logic used in the flowchart, above includes a mathematical formula used on the configured “MB Start DT Action” adjustment value. The numeric value for each adjustment value is as follows:

MB Start DT Action Adjustment Value	Numeric Value
None	0
Down	1
Up	2
1 MB Down	3
2 MB Down	4
3 MB Down	5



When the adjustment value is “Up” (2) or greater, the logic compares the adjustment value, minus 2, with the number of multiballs, to determine whether to reset the drop-target up or knock it down. For illustrative purposes, the table below shows some example values and resulting drop-target action.

Number of Multiballs achieved so far	Configured Drop Target Action	Formula (Adjustment - 2) > Multiballs?	Drop Target Action
0	None (0)	n/a	None
0	Down (1)	n/a	Down
0	Up (2)	(Adjustment Value 2 - 2) = 0 > 0 ? No	Up
0	1 MB Down (3)	(Adjustment Value 3 - 2) = 1 > 0 ? Yes	Down
0	2 MB Down (4)	(Adjustment Value 4 - 2) = 2 > 0 ? Yes	Down
0	3 MB Down (5)	(Adjustment Value 5 - 2) = 3 > 0 ? Yes	Down
1	None (0)	n/a	None
1	Down (1)	n/a	Down
1	Up (2)	(Adjustment Value 2 - 2) = 0 > 1 ? No	Up
1	1 MB Down (3)	(Adjustment Value 3 - 2) = 1 > 1 ? No	Up
1	2 MB Down (4)	(Adjustment Value 4 - 2) = 2 > 1 ? Yes	Down
1	3 MB Down (5)	(Adjustment Value 5 - 2) = 3 > 1 ? Yes	Down
2	None (0)	n/a	None
2	Down (1)	n/a	Down
2	Up (2)	(Adjustment Value 2 - 2) = 0 > 2 ? No	Up
2	1 MB Down (3)	(Adjustment Value 3 - 2) = 1 > 2 ? No	Up
2	2 MB Down (4)	(Adjustment Value 4 - 2) = 2 > 2 ? No	Up
2	3 MB Down (5)	(Adjustment Value 5 - 2) = 3 > 2 ? Yes	Down
3	None (0)	n/a	None
3	Down (1)	n/a	Down
3	Up (2)	(Adjustment Value 2 - 2) = 0 > 3 ? No	Up
3	1 MB Down (3)	(Adjustment Value 3 - 2) = 1 > 3 ? No	Up
3	2 MB Down (4)	(Adjustment Value 4 - 2) = 2 > 3 ? No	Up
3	3 MB Down (5)	(Adjustment Value 5 - 2) = 3 > 3 ? No	Up
4	None (0)	n/a	None
4	Down (1)	n/a	Down
4	Up (2)	(Adjustment Value 2 - 2) = 0 > 4 ? No	Up
4	1 MB Down (3)	(Adjustment Value 3 - 2) = 1 > 4 ? No	Up
4	2 MB Down (4)	(Adjustment Value 4 - 2) = 2 > 4 ? No	Up
4	3 MB Down (5)	(Adjustment Value 5 - 2) = 3 > 4 ? No	Up
5	None (0)	n/a	None
5	Down (1)	n/a	Down
5	Up (2)	(Adjustment Value 2 - 2) = 0 > 5 ? No	Up
5	1 MB Down (3)	(Adjustment Value 3 - 2) = 1 > 5 ? No	Up
5	2 MB Down (4)	(Adjustment Value 4 - 2) = 2 > 5 ? No	Up
5	3 MB Down (5)	(Adjustment Value 5 - 2) = 3 > 5 ? No	Up

The table, above, shows the various combinations of multiball count, configured “MB Start DT Action” values, and resulting drop-target up/down state that will get set at multiball start. The resulting state is expected and consistent with the described behavior for this adjustment. As mentioned, the multiball count refers to multiballs the player has achieved thus far (not including current multiball startup) while the adjustment value for “1 MB Down”, “2 MB Down” and “3 MB Down” number conceptually includes the current multiball startup. So if player had already achieved 2 multiballs and adjustment is set to “2 MB Down” then this logic applies as such player is experiencing their 3<sup>rd</sup> multiball, therefore, the drop-target gets reset (kicked up).

### Multiball-Start Drop-Target Action Adjustment Code, Up/Down Functions

As depicted in the new code for Multiball-Start Drop-Target Action, the new code can, depending on conditions, do one of the following:

- Schedule a “Down” function at \$7894,3B (ROM offset 0x6F894), or
- Schedule a “Up” function at \$78B7,3B (ROM offset 0x6F8b7)

These two new functions are located in bank \$3B where more available unused ROM space is available. The functions are as follows:

```

-----
; MbStartDropTargetDown()
;
7894: BD 84 8F    JSR    $848F    ; ClearMemoryFlag()
7897: E3          ; 0xE3 cleared at dt-down-switch, set at dt-up
7898: C6 03      LDB    #$03    ;
789A: 34 04      PSHS   B       ;
;
789C: BD 83 19    JSR    $8319    ; --\ CheckSwitchState() C-clear = switch closed
789F: 3F          ; | Drop-Target switch
78A0: 24 10      BCC    $78B2    ; | C-bit clear, target is down, done
78A2: BD 83 46    JSR    $8346    ; | Sleep()
78A5: 10          ; |
78A6: BD 83 85    JSR    $8385    ; | EnqueueSolenoidPulse_ParameterByte()
78A9: 13          ; | SolenoidTableEntry13, 0C=Knock Down, 20
78AA: BD 83 46    JSR    $8346    ; | Sleep()
78AD: 10          ; |
78AE: 6A E4      DEC    ,S       ; |
78B0: 26 EA      BNE    $789C    ; --/
;
78B2: 35 04      PULS   B       ;
78B4: 7E 99 A2    JMP    $99A2    ;
-----
; MbStartDropTargetUp()
;
78B7: BD 84 80    JSR    $8480    ; SetMemoryFlag()
78BA: E3          ; 0xE3 cleared at dt-down-switch, set at dt-up
78BB: C6 03      LDB    #$03    ;
78BD: 34 04      PSHS   B       ;
;
78BF: BD 83 19    JSR    $8319    ; --\ CheckSwitchState() C-clear = switch closed
78C2: 3F          ; | Drop-Target switch
78C3: 25 10      BCS    $78D5    ; | C-bit set, target is up, done
78C5: BD 83 46    JSR    $8346    ; | Sleep()
78C8: 10          ; |
78C9: BD 83 85    JSR    $8385    ; | EnqueueSolenoidPulse_ParameterByte()

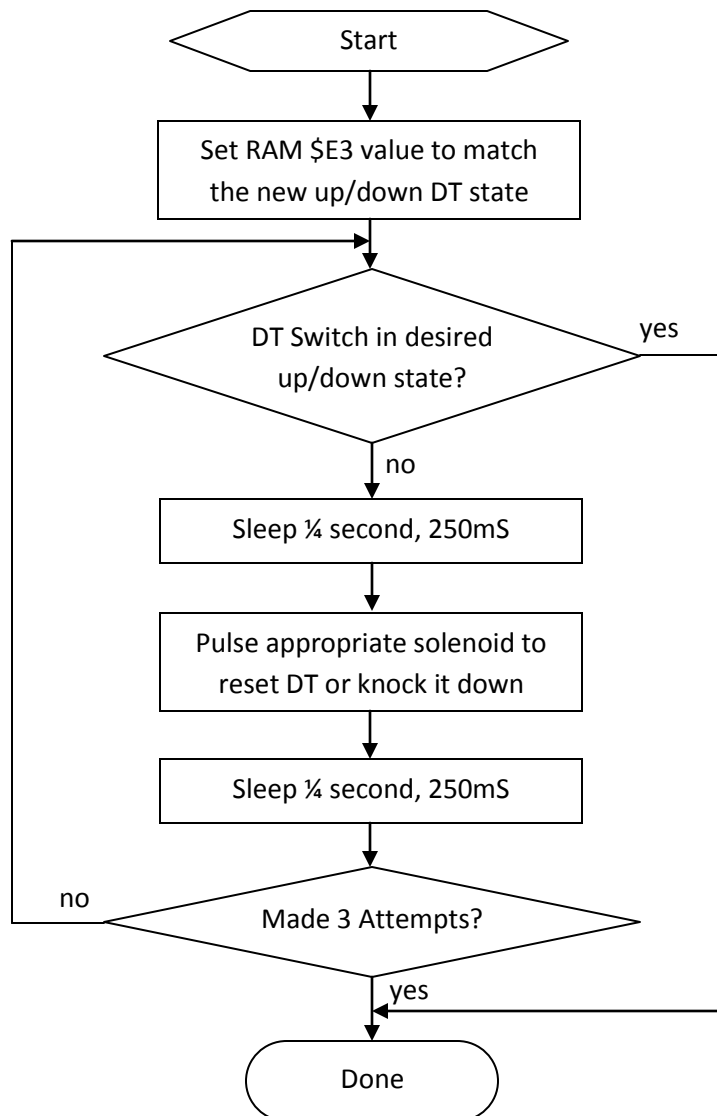
```

```

78CC: 06                ; | SolenoidTableEntry06, 1C=Drop Target, 40
78CD: BD 83 46        JSR  $8346                ; | Sleep()
78D0: 10                ; |
78D1: 6A E4           DEC  ,S                ; |
78D3: 26 EA           BNE  $78BF                ; --/
                          ;
78D5: 35 04           PULS B                ;
78D7: 7E 99 A2       JMP  $99A2                ;
-----

```

These new functions have similar logic and are intended solely for use by the new Multiball-Start Drop-Target Action feature. The following flowchart depicts the logic used by both functions.



Logic used in these Up/Down functions is modeled after existing drop-target up function used by L-8 game code. The normal game code uses a retry count of 6. Other elements are identical including the 1/4 second sleep times and switch-check.

## Multiball-Start Drop-Target Action Adjustment Code, Drop-Target Switch Handler

The software controlled up/down of the drop target will necessarily cause the drop-target switch to be hit or opened as the target is knocked down or kicked up, respectively. When the drop target switch becomes closed due to the drop-target down state, the switch handler code will necessarily get invoked as per normal switch-matrix handling and function callback behavior. The switch handler code will behave as if the player had knocked the drop-target down and accumulate points if code is not in place to prevent this.

Referring to the switch matrix table shown earlier in this document, the table entry for the drop-target switch is as follows:

```
4BE9: 00 02                ; SwitchTableEntry3F, 77, Drop Target
4BEB: 67 0C 31            ;
4BEE: 3C 00 80            ;
4BF1: 40 00 04            ;
```

The switch-matrix entry for drop-target switch schedules function \$670C,31. This is the drop-target switch handler function, partially annotated, is shown in full, below.

```
-----;-----
; DropTargetSwitchHandler()
;
670C: BD 86 90    JSR    $8690    ; SearchLinkedListForId() // c-bit clear = ID found
670F: 00 B3                ; 0x00B3 DropTargetUp()
6711: 10 24 00 E7 LBCC    $67FC    ;
;
6715: BD 86 90    JSR    $8690    ; SearchLinkedListForId() // c-bit clear = ID found
6718: 00 E7                ; Search for 00E7 BallSearchDropTargetSleepAndReset()
671A: 10 24 00 DE LBCC    $67FC    ;
;
671E: BD 84 8F    JSR    $848F    ; ClearMemoryFlag()
6721: E3                ; 0xE3 cleared at dt-down-switch, set at dt-up
;
6722: BD 85 46    JSR    $8546    ; DoSoundTableParameterByte()
6725: 50                ; 0x50 Sound Index "Vu-vilp"
;
6726: BD FA AE    JSR    $FAAE    ;
6729: 7E 67 2C    JMP    $672C    ; <nop>
;
672C: C6 03                ;
672E: BD 6D E9    JSR    $6DE9    ;
;
6731: BD 86 90    JSR    $8690    ; SearchLinkedListForId() // c-bit clear = ID found
6734: 00 86                ; Search for 0086, C-bit clear = multiball running
;
6736: 24 03                ;
6738: BD 68 82    JSR    $6882    ;
673B: BD 87 3C    JSR    $873C    ;
673E: 0F 40                ;
6740: 25 13                ;
6742: BD 84 AD    JSR    $84AD    ; GetMemoryFlag()
6745: D1                ;
6746: 25 05                ;
6748: BD 88 D5    JSR    $88D5    ;
674B: 00 20                ;
674D: BD 87 22    JSR    $8722    ;
6750: 0F 40                ;
```

```

6752: BD 57 43   JSR   $5743   ;
6755: BD 83 19   JSR   $8319   ; Gets switch state
6758: 3E          ; 0x3E switch (ball-popper switch)
6759: 25 03       BCS   $675E   ; If switch is open C-bit set, skip target-up, no need
675B: BD 68 08   JSR   $6808   ; ScheduleDropTargetUp() if ball in popper, reset target
675E: BD 8A AA   JSR   $8AAA   ;
6761: 00 A4       ;
6763: 01 FF       ;
6765: 25 15       BCS   $677C   ;
6767: BD 61 C3   JSR   $61C3   ;
676A: BD 8A AA   JSR   $8AAA   ; SearchLinkedListAndMaskParameterBytes() C-clr=found
676D: 00 86       ; Search for 0x0086, C-clear = multiball running
676F: 01 FF       ;
6771: 24 0D       BCC   $6780   ;
6773: BD 68 08   JSR   $6808   ; ScheduleDropTargetUp()
6776: BD 68 A0   JSR   $68A0   ;
6779: 7E 99 A2   JMP   $99A2   ;
;
677C: BD 85 1F   JSR   $851F   ;
677F: 46          ;
6780: BD 8A AA   JSR   $8AAA   ; SearchLinkedListAndMaskParameterBytes() C-clr=found
6783: 00 86       ; Search for 0x0086, C-clear = multiball running
6785: 01 FF       ;
6787: 24 58       BCC   $67E1   ;
6789: BD 87 15   JSR   $8715   ;
678C: 0E 40       ;
678E: 8E 05 A1   LDX   #$05A1   ;
6791: BD FB 29   JSR   $FB29   ; IncrementXByPlayerIndexNumber()
6794: 7E 67 97   JMP   $6797   ; <nop>
6797: A6 84       LDA   ,X       ;
6799: BD 83 0C   JSR   $830C   ; Get8BitSettingIntoBParameterByte()
679C: 05          ; 0x05, FeatureAdjustment005, Drop Targt Count
679D: 34 04       PSHS  B       ; Save adjustment value onto stack
679F: A1 E0       CMPA  ,S+     ; Compare A with adjustment value (and pop stack)
67A1: 25 16       BCS   $67B9   ;
;
67A3: BD 86 90   JSR   $8690   ; SearchLinkedListForId() // c-bit clear = ID found
67A6: 00 86       ; Search for 0x0086, C-clear = multiball running
67A8: 24 08       BCC   $67B2   ;
67AA: BD 8B 77   JSR   $8B77   ; ScheduleFunctionStart()
67AD: 00 A6       ; 0x00A6 = DropTargetDownSwitchTimerLoop()
67AF: 68 4A 31   ; $684A,31
67B2: BD A7 25   JSR   $A725   ;
67B5: 85 01       BITA  #$01    ;
67B7: 27 12       BEQ   $67CB   ;
;
67B9: 86 6A       LDA   #$6A    ;
67BB: BD 85 46   JSR   $8546   ; DoSoundTableParameterByte()
67BE: 6A          ; 0x6A = "Lock sequence initiated"
67BF: 25 1A       BCS   $67DB   ;
67C1: BD 8B 77   JSR   $8B77   ; ScheduleFunctionStart()
67C4: 00 ED       ;
67C6: 68 02 31   ; $6802,31
67C9: 20 10       BRA   $67DB   ;
67CB: BD 85 46   JMP   $8546   ; DoSoundTableParameterByte()
67CE: 7C          ; 0x7C = "Load the cannon"
67CF: 25 0A       BCS   $67DB   ;
67D1: 86 43       LDA   #$43    ;
67D3: BD 8B 77   JSR   $8B77   ; ScheduleFunctionStart()
67D6: 00 ED       ;
67D8: 68 02 31   ; $6802,31
67DB: BD 68 A0   JSR   $68A0   ;

```

```

67DE: 7E 99 A2    JMP    $99A2                ;
67E1: BD 84 AD    JSR    $84AD                ; GetMemoryFlag()
67E4: 42                                ;
67E5: 24 15        BCC    $67FC                ;
67E7: BD 86 90    JSR    $8690                ; SearchLinkedListForId() // c-bit clear = ID found
67EA: 00 A9                                ; Search for 0x00A9
67EC: 25 0E        BCS    $67FC                ;
67EE: BD 83 19    JSR    $8319                ; Gets switch state
67F1: 3E                                ; 0x3E switch
67F2: 24 08        BCC    $67FC                ;
67F4: BD 8B 77    JSR    $8B77                ; ScheduleFunctionStart()
67F7: 00 A9                                ; ID 00A9 == "LOAD THE GUN" period when 1 ball remaining
67F9: 48 AF 31                                ; $48AF,31
67FC: BD 68 A0    JSR    $68A0                ;
67FF: 7E 99 A2    JMP    $99A2                ;
-----;-----

```

As shown, the drop-target switch handler starts off with the following three operations:

- Checks if function ID 00 B3 is running, if so then skip to the end of the function. The 00 B3 function means the normal game function for drop-target “Up” is running and, therefore, the drop target going ‘up’ imminent and, therefore, the drop-target switch being closed is ignored.
- Checks if function ID 00 E7 is running, if so then skip to the end of the function. The 00 E7 function is used during ball search to represent a period of time in which the drop-target is being exercised as an effort to dislodge a stuck ball. If this is happening then the drop-target switch being closed is ignored.
- Clears the \$E3 flag. This is a flag used by game code to track the known state of the drop target. Drop-target down means \$E3 is cleared while drop-target up has \$E3 being set. Other game code can query this flag to derive the position of the drop-target.
- Remaining drop-target processing proceeds.

In order to support the software-controlled “Multiball Start Drop Target Action” having the code automatically reset the drop-target up or automatically knocking it down, this drop-target switch handler needs to have added check similar to the first two operations mentioned above. The added check needs to check if one of the new “Up” or “Down” drop-target functions are running and, if so, ignore the drop-target switch. If such check was not in place, the “MB Start DT Action” knocking the drop-target down would incur point accumulation as if the player had knocked the DT down at multiball start.

In order to insert new code that checks if new “Up” or “Down” function is running, the 3<sup>rd</sup> operation (Clearing the \$E3 flag) is replaced with a jump to a new routine which will perform such checks. The new routine will also include the clearing of the \$E3 flag, as necessary.

Shown below is the first part of the drop-target handler function with the L8.3 change highlighted.

```

-----;-----
; DropTargetSwitchHandler()
;
670C: BD 86 90    JSR    $8690                ; SearchLinkedListForId() // c-bit clear = ID found
670F: 00 B3                                ; 0x00B3 DropTargetUp()
6711: 10 24 00 E7 LBCC $67FC                ;

```

```

;
6715: BD 86 90    JSR    $8690    ; SearchLinkedListForId() // c-bit clear = ID found
6718: 00 E7        ; Search for 00E7 BallSearchDropTargetSleepAndReset()
671A: 10 24 00 DE LBCC $67FC
;
;
671E: BD 84 8F    JSR    $848F    ; ClearMemoryFlag()
6721: E3          ; 0xE3 cleared at dt-down-switch, set at dt-up
;
671E: 7E 7F A2    JMP    $7FA2    ; Go to drop-target handler fixup function
6721: 12          NOP
;
6722: BD 85 46    JSR    $8546    ; DoSoundTableParameterByte()
6725: 50          ; 0x50 Sound Index "Vu-vilp"
...

```

As shown, the JSR function and its extra parameter byte were replaced with a JMP to \$7FA2 and a NOP instruction. In this case a JMP instruction is used which means the fixup function will need to JMP back to desired address instead of the typical JSR which requires RTS to come back to resume code execution. This new function at \$7FA2 is in the current bank, \$31. This address corresponds to ROM offset 0x47FA2, and is shown below.

```

-----;-----
7FA2: BD 86 90    JSR    $8690    ; SearchLinkedListForId() // c-bit clear = ID found
7FA5: 00 B7        ; 0x00B7 Drop target down function id
7FA7: 24 0E        BCC    $7FB7    ; Drop target "Down" is running, ignore the dt switch
;
7FA9: BD 86 90    JSR    $8690    ; SearchLinkedListForId() // c-bit clear = ID found
7FAC: 00 B9        ; 0x00B9 Drop target up function id
7FAE: 24 07        BCC    $7FB7    ; Drop target "Up" is running, ignore the dt switch
;
7FB0: BD 84 8F    JSR    $848F    ; ClearMemoryFlag()
7FB3: E3          ; 0xE3 cleared at dt-down-switch, set at dt-up
7FB4: 7E 67 22    JMP    $6722    ; Go do regular drop-target switch code
;
7FB7: 7E 99 A2    JMP    $99A2    ; Done with this switch handler, up/down is running
-----;-----

```

The function at \$7FA2, above, performs some basic operations:

- Checks if function ID 00 B7 is running, if so then skip to the end of the function to jump to \$99A2 which is the end of switch handler jump point. The 00 B7 function means the “MB Start DT Action” has scheduled the “Down” function to knock the drop-target down, and that function hasn’t yet completed.
- Checks if function ID 00 B9 is running, if so then skip to the end of the function to jump to \$99A2 which is the end of switch handler jump point. The 00 B9 function means the “MB Start DT Action” has scheduled the “Up” function to reset the drop-target up, and that function hasn’t yet completed.
- If neither 00 B7 or 00 B9 are running, then the E3 flag is cleared (indicating drop-target is down) and code then jumps to \$6722 which is the instruction immediately after the JMP instruction that jumped to this \$7FA2 code. By doing this jump, code effectively resumes its normal drop-target handler logic.



- As indicated, if either 00 B7 or 00 B9 are running, then code performs a JMP instruction to jump to \$99A2 which is a common jump point for scheduled functions to jump to when they are complete.

The logic, above, effectively allows the game to ignore the drop-target switch while the “MB Start DT Action” feature is resetting the drop-target up or knocking it down.

### Multiball-Start Drop-Target Action Adjustment Code Analysis

As shown in the new functions, above, the code will schedule a drop-target up or drop-target down function call depending on the logic shown. These refer to two new functions with these characteristics:

	Drop-Target Down	Drop-Target Up
<b>WPC Address</b>	\$7894,3B	\$78B7,3B
<b>ROM Offset</b>	0x6F894	0x6F8B7
<b>Function ID</b>	00 B7	00 B9

Special consideration was needed to choose the 16-bit value for the Function ID associated with these new scheduled functions. Without knowledge of how function ID numbers were originally chosen for existing functions in L-8, values were chosen that were initially found to not be in use by any existing function and are numerically close to existing functions of similar nature. For example, below shows some existing Function ID values from L-8 software:

Function WPC Address	Function ID	Function Purpose
\$45D7,31	00 B2	Kick a ball out of shooter lane and checks shooter-lane switch
\$6816,31	00 B3	Reset drop-target up and checks drop-target switch
\$4A77,31	00 B5	TBD, Called when left-loop lock shot is hit
\$4A28,31	00 B6	TBD, Called when right-loop is hit
\$6C80,31	00 B8	TBD, Called as part of multiball startup

The table, above, shows the various function IDs from bank \$31 using Function ID starting with 00 Bx. Initial survey of the code seemed that no function appears to be defined with 00 B7 or 00 B9 as the ID value (*a deeper investigation, below, reveals this isn't actually the case*). To further demonstrate that these 2 IDs could safely be used for the MB Start DT Action feature, as part of this documentation a survey across the entire L-8 ROM image was done to see if any code could be found using these 2 ID values. The table below shows some function signatures that are used to determine whether other functions are using these 2 ID values.

Function Name	L-8 Function Usage Signature	Description
ScheduleFunctionStart()	BD 8B 77 xx xx yy yy yy	Schedules function ID xx xx to start at WPC

		Addr yy yy yy
<b>SearchLinkedListForId()</b>	BD 86 90 xx xx	Searches for scheduled function ID xx xx
<b>CancelScheduledCallbackFunction()</b>	BD 86 9E xx xx	Cancels scheduled function ID xx xx
<b>UpdateCurrentRunningSchedule FunctionIDParameterBytes()</b>	BD 86 AC xx xx	Sets currently running function ID to xx xx
<b>TBD()</b>	BD 86 BA xx xx	TBD, where xx xx is function ID
<b>CancelScheduledCallbackIDParameterBytes()</b>	BD 86 D0 xx xx	Cancels scheduled ID xx xx
<b>CancelAllCallbacksIdMaskParameterBytes()</b>	BD 8A 9A xx xx yy yy	Cancels scheduled functions matching ID pattern of xx xx bitwise-and yy yy
<b>SearchLinkedListAndMaskParameterBytes()</b>	BD 8A AA xx xx yy yy	Searches for schedule functions matching ID pattern of xx xx bitwise-and yy yy
<b>AddLinkedListEntry()</b>	BD 8B 3D xx xx yy yy yy	Adds function yy yy yy to linked list as ID xx xx
<b>ScheduleFunctionStart()</b>	BD 8B 77 xx xx yy yy yy	Schedules function yy yy yy ID xx xx
<b>TBD()</b>	BD 8B 9D xx xx yy yy yy	TBD, where xx xx is ID and yy yy yy is addr.
<b>ScheduleFunctionCallback()</b>	BD 8B C3 xx xx yy yy yy	Schedules function yy yy yy ID xx xx
<b>TBD()</b>	BD 8B F7 xx xx yy yy yy	Schedules function yy yy yy ID xx xx

The functions listed above are not necessarily exhaustive nor have they been assigned 100% accurate names or given 100% accurate descriptions. This information is based on a cursory review of the L-8 code and observing how it behaves. The distinction between similarly named functions is not part of this documentation. *The function names were as given during an initial survey of WPC code and are worthy of renaming as more information is learned about the code flow.*

During L8.3 development an examination of the two new ID values for new drop-target Up and Down functions was done and the values were chosen as apparently available ID values and deemed safe for L8.3. After L8.3 release with this documentation providing thorough information and full transparency, while demonstrating how these ID values was chosen, **some minor overlap in ID values was detected** and will be described in more detail, below.

Function Name	DT	Search Pattern	Search Results	
			ROM L-8	ROM L8.3
<b>SearchLinkedListForId()</b>	Down	BD 86 90 00 B7	Zero hits	One hit, bank \$31
	Up	BD 86 90 00 B9	Zero hits	One hit, bank \$31
<b>CancelScheduledCallbackFunction()</b>	Down	BD 86 9E 00 B7	Zero hits	Zero hits
	Up	BD 86 9E 00 B9	Zero hits	Zero hits
<b>UpdateCurrentRunningSchedule FunctionIDParameterBytes()</b>	Down	BD 86 AC 00 B7	Zero hits	Zero hits
	Up	BD 86 AC 00 B9	Zero hits	Zero hits
<b>TBD()</b>	Down	BD 86 BA 00 B7	Zero hits	Zero hits
	Up	BD 86 BA 00 B9	Zero hits	Zero hits
<b>CancelScheduledCallbackIDParameterBytes()</b>	Down	BD 86 D0 00 B7	Zero hits	Zero hits
	Up	BD 86 D0 00 B9	Zero hits	Zero hits
<b>CancelAllCallbacksIdMaskParameterBytes()</b>	Down	BD 8A 9A 00 B7	Zero hits	Zero hits
	Up	BD 8A 9A 00 B9	Zero hits	Zero hits

<b>SearchLinkedListAndMaskParameterBytes()</b>	Down	BD 8A AA 00 B7	Zero hits	Zero hits
	Up	BD 8A AA 00 B9	Zero hits	Zero hits
<b>AddLinkedListEntry()</b>	Down	BD 8B 3D 00 B7	Zero hits	Zero hits
	Up	BD 8B 3D 00 B9	Zero hits	Zero hits
<b>ScheduleFunctionStart()</b>	Down	BD 8B 77 00 B7	Zero hits	Zero hits
	Up	BD 8B 77 00 B9	Two hits, bank \$3B	Two hits, bank \$3B
<b>TBD()</b>	Down	BD 8B 9D 00 B7	Zero hits	Zero hits
	Up	BD 8B 9D 00 B9	Zero hits	Zero hits
<b>ScheduleFunctionCallback()</b>	Down	BD 8B C3 00 B7	Zero hits	One hit, bank \$31
	Up	BD 8B C3 00 B9	Zero hits	One hit, bank \$31
<b>TBD()</b>	Down	BD 8B F7 00 B7	Zero hits	Zero hits
	Up	BD 8B F7 00 B9	Zero hits	Zero hits

For the table, above, the following color code applies:

- **Green box** No problems, search pattern not found
- **Red box** Found function ID in rom L-8, conflicting with new function ID in L8.3
- **Yellow box** Found function ID in rom L8.3, conflicting with existing ID from L-8
- **Blue box** Found function ID in rom L8.3 not conflicting with L-8

The highlighted elements are as follows using highlighted color text consistent with their finding, above:

- In ROM L8.3, in bank \$31
  - The “Down” function schedule callback for “MB Start DT Action”, as documented above
  - The “Down” function ID lookup in the drop-target switch handler, as documented above
  - The “Up” function schedule callback for “MB Start DT Action”, as documented above
  - The “Up” function ID lookup in the drop-target switch handler, as documented above
- In Both ROMs L-8 and L8.3, in bank \$3B
  - Two existing references to function ID 00 B9, conflicting with the new “Up” function ID

A deeper investigation into the 00 B9 overlap between L8.3 and existing L-8 code is provided below. The overlap of the new “Up” function is not likely to be problematic or noticeable during game play. Refer to the analysis, below, for details.

### Drop-Target Up Function ID 00 B9 Overlap

As indicated, above, the chosen ID value for the new Drop-Target Up function at multiball startup is cited in 2 other places in bank \$3B in the original L-8 ROM (and in L8.3). This 00 B9 function ID is used by a helper function used during the ball search routine. During ball search, there is a moment where a function is scheduled with ID 00 B9 where such function is responsible for simply performing a 3.5 second sleep and then cancelling 2 other ball-search functions before ending itself. The “00 B9” ID is only cited during the function scheduling and there is no ball search (or any other) code that checks if the 00 B9 ball-search helper function is running. Additionally there is no ball search (or any other) code that attempts to cancel the 00 B9 function.

Since ball search only happens when there is trouble (no playfield switch hit for a certain period of time) and since the “MB Start DT Action” code engages at multiball startup, and since multiball startup happens soon after a switch has been hit, it is reasonable to suggest that the ball-search 00 B9 function is not likely to conflict with the “MB Start DT Action” function 00 B9 that resets the target to the up position.

To help understand how the ball-save code works, and where its use of function ID 00 B9 is used, the ball-search code is shown below, partially annotated, along with some of its supporting functions. A fair amount of the ball-search logic can be gleaned by an examination of the code, however a full description of the ball-search process is outside the scope of this document and a deeper-dive into ball-search is left as an exercise to the reader. Ball search code related to function ID 00 B9 is highlighted. This ball-search code starts at \$5F42,3B (ROM offset 0x6DF42).

```

-----;-----
5F42: BD F7 59   JSR   $F759           ; ChecksGameMode(), if game in progress, z-bit set
5F45: 7E 5F 48   JMP   $5F48           ; <nop>
5F48: 26 1C      BNE   $5F66           ;
5F4A: 81 01      CMPA  #$01           ;
5F4C: 22 0A      BHI   $5F58           ;
5F4E: BD 8B 77   JSR   $8B77           ; ScheduleFunctionStart()
5F51: 00 EE      ; ID 00EE == BallSearchFrom5FBE()  BallSearchPhase1()
5F53: 5F BE 3B      ;
5F56: 20 16      BRA   $5F6E           ;
5F58: 81 02      CMPA  #$02           ;
5F5A: 22 0A      BHI   $5F66           ;
5F5C: BD 8B 77   JSR   $8B77           ; ScheduleFunctionStart()
5F5F: 00 EE      ; ID 00EE == BallSearchFrom5FA3()  BallSearchPhase2()
5F61: 5F A3 3B      ;
5F64: 20 08      BRA   $5F6E           ;
5F66: BD 8B 77   JSR   $8B77           ; ScheduleFunctionStart()
5F69: 00 EE      ; ID 00EE == BallSearchFrom5F6F()  BallSearchPhase3()
5F6B: 5F 6F 3B      ;                               and all subsequent phases
5F6E: 39          RTS                    ;
-----;-----
;-----
; ID 00EE == BallSearchFrom5F6F()
; BallSearchPhase3() and all subsequent phases
;-----
;
; Knock drop-target down 4 times and 2.5 seconds later
; reset it to original position.
;
5F6F: BD 8B 77   JSR   $8B77           ; ScheduleFunctionStart()
5F72: 00 E7      ; ID 00E7 BallSearchDropTargetSleepAndReset()
5F74: 60 D1 3B      ;
;
5F77: BD 83 85   JSR   $8385           ; EnqueueSolenoidPulse_ParameterByte()
5F7A: 13          ; SolenoidTableEntry13, 0C=Knock Down, 20
5F7B: BD 83 46   JSR   $8346           ; Sleep()
5F7E: 02          ;
5F7F: BD 83 85   JSR   $8385           ; EnqueueSolenoidPulse_ParameterByte()
5F82: 13          ; SolenoidTableEntry13, 0C=Knock Down, 20
5F83: BD 83 46   JSR   $8346           ; Sleep()
5F86: 01          ;
5F87: BD 83 85   JSR   $8385           ; EnqueueSolenoidPulse_ParameterByte()

```

```

5F8A: 13                               ; SolenoidTableEntry13, 0C=Knock Down, 20
5F8B: BD 83 46   JSR   $8346           ; Sleep()
5F8E: 02                               ;
5F8F: BD 83 85   JSR   $8385           ; EnqueueSolenoidPulse_ParameterByte()
5F92: 13                               ; SolenoidTableEntry13, 0C=Knock Down, 20
5F93: BD 83 85   JSR   $8385           ; EnqueueSolenoidPulse_ParameterByte()
5F96: 01                               ; SolenoidTableEntry01, 0A=Top Lock, 10
5F97: BD 83 46   JSR   $8346           ; Sleep()
5F9A: 05                               ;
5F9B: BD 83 85   JSR   $8385           ; EnqueueSolenoidPulse_ParameterByte()
5F9E: 03                               ; SolenoidTableEntry03, 10=Left Lock, 20
5F9F: BD 83 46   JSR   $8346           ; Sleep()
5FA2: 05                               ;
;-----
; ID 00EE == BallSearchFrom5FA3()
; BallSearchPhase2() starts here
;-----
;
5FA3: BD 60 9A   JSR   $609A           ; BallSearchDropTargetUpAnd00E7Reset()
;
5FA6: BD 83 85   JSR   $8385           ; EnqueueSolenoidPulse_ParameterByte()
5FA9: 13                               ; SolenoidTableEntry13, 0C=Knock Down, 20
5FAA: BD 83 46   JSR   $8346           ; Sleep()
5FAD: 02                               ;
5FAE: BD 83 85   JSR   $8385           ; EnqueueSolenoidPulse_ParameterByte()
5FB1: 13                               ; SolenoidTableEntry13, 0C=Knock Down, 20
5FB2: BD 83 46   JSR   $8346           ; Sleep()
5FB5: 02                               ;
5FB6: BD 83 85   JSR   $8385           ; EnqueueSolenoidPulse_ParameterByte()
5FB9: 13                               ; SolenoidTableEntry13, 0C=Knock Down, 20
5FBA: BD 83 46   JSR   $8346           ; Sleep()
5FBD: 05                               ;
;-----
; ID 00EE == BallSearchFrom5FBE()
; BallSearchPhase1() starts here
;-----
;
5FBE: 85 01     BITA  #$01             ;
5FC0: 26 30     BNE   $5FF2           ;
5FC2: BD 86 90   JSR   $8690           ; SearchLinkedListForId() // c-bit clear = ID found
5FC5: 00 AA     ; ID 00AA == BallInPopper()
5FC7: 24 29     BCC   $5FF2           ; If ball is in popper, skip over the following
5FC9: BD 86 90   JSR   $8690           ; SearchLinkedListForId() // c-bit clear = ID found
5FCC: 00 BC     ; ID 00BC == Sleep3andHalfSeconds()
5FCE: 24 22     BCC   $5FF2           ;
5FD0: BD 84 8F   JSR   $848F           ; ClearMemoryFlag()
5FD3: C6     ;
5FD4: BD 83 85   JSR   $8385           ; EnqueueSolenoidPulse_ParameterByte()
5FD7: 0A     ; SolenoidTableEntry0A, 0B=Gun Motor, FF
;
5FD8: BD 8B 77   JSR   $8B77           ; ScheduleFunctionStart()
5FDB: 00 E8     ; ID 00E8 == Sleep3SecondsThenPulseGunKicker()
5FDD: 60 AE 3B   ;
;
5FE0: BD 8B 77   JSR   $8B77           ; ScheduleFunctionStart()
5FE3: 00 BD     ;
5FE5: 66 75 31   ;
;
5FE8: BD 8B 77   JSR   $8B77           ; ScheduleFunctionStart()
5FEB: 00 B9     ; ID 00B9 == Sleep3andHalfSecondsThenCancel00BDand00BC()
5FED: 60 B9 3E   ;
5FF0: 20 34     BRA   $6026           ;

```

```

;
;
5FF2: BD 86 90 JSR $8690 ; SearchLinkedListForId() // c-bit clear = ID found
5FF5: 00 AB ;
5FF7: 24 2D BCC $6026 ;
5FF9: BD 86 90 JSR $8690 ; SearchLinkedListForId() // c-bit clear = ID found
5FFC: 00 AA ;
5FFE: 24 26 BCC $6026 ;
6000: BD 86 90 JSR $8690 ; SearchLinkedListForId() // c-bit clear = ID found
6003: 00 BD ;
6005: 24 1F BCC $6026 ;
6007: BD 86 90 JSR $8690 ; SearchLinkedListForId() // c-bit clear = ID found
600A: 00 BC ; ID 00BC == Sleep3andHalfSeconds()
600C: 24 18 BCC $6026 ;
600E: BD 83 85 JSR $8385 ; EnqueueSolenoidPulse_ParameterByte()
6011: 07 ; SolenoidTableEntry07, 01=Ball Popper, 40
6012: BD 8B 77 JSR $8B77 ; ScheduleFunctionStart()
6015: 00 BC ; ID 00BC == Sleep3andHalfSeconds()
6017: 60 CA 3B ;
601A: BD 8B 77 JSR $8B77 ; ScheduleFunctionStart()
601D: 00 B9 ; ID 00B9 == Sleep3andHalfSecondsThenCancel00BDand00BC()
601F: 60 B9 3B ;
6022: BD 83 46 ; Sleep()
6025: 05 ;
6026: BD 86 90 JSR $8690 ; SearchLinkedListForId() // c-bit clear = ID found
6029: 00 AE ;
602B: 24 0E BCC $603B ;
602D: BD 83 19 JSR $8319 ; GetSwitchClosedParameterByte() // C-clear = sw closed
6030: 0F ;
6031: 24 08 BCC $603B ;
6033: BD 83 85 JSR $8385 ; EnqueueSolenoidPulse_ParameterByte()
6036: 04 ; SolenoidTableEntry04, 04=Trough, 40
6037: BD 83 46 JSR $8346 ; Sleep()
603A: 05 ;
603B: BD F7 59 JSR $F759 ; ChecksGameMode(), if game in progress, z-bit set
603E: 7E 60 41 JMP $6041 ; <nop>
6041: 27 08 BEQ $604B ;
6043: BD 83 85 JSR $8385 ; EnqueueSolenoidPulse_ParameterByte()
6046: 11 ; SolenoidTableEntry11, 08=Kickback, 40
6047: BD 83 46 JSR $8346 ;
604A: 05 ;
604B: BD 83 85 JSR $8385 ; EnqueueSolenoidPulse_ParameterByte()
604E: 0E ; SolenoidTableEntry0E, 0F=Bottom Jet, 40
604F: BD 83 46 JSR $8346 ;
6052: 05 ;
6053: BD 83 85 JSR $8385 ; EnqueueSolenoidPulse_ParameterByte()
6056: 0C ; SolenoidTableEntry0C, 0D=Left Jet, 40
6057: BD 83 46 JSR $8346 ;
605a: 05 ;
605B: BD 83 85 JSR $8385 ; EnqueueSolenoidPulse_ParameterByte()
605E: 0D ; SolenoidTableEntry0D, 0E=Right Jet, 40
605F: BD 83 46 JSR $8346 ;
6062: 05 ;
6063: BD 83 85 JSR $8385 ; EnqueueSolenoidPulse_ParameterByte()
6066: 0F ; SolenoidTableEntry0F, 05=Right Slings, 40
6067: BD 83 46 JSR $8346 ;
606A: 05 ;
606B: BD 83 85 JSR $8385 ; EnqueueSolenoidPulse_ParameterByte()
606E: 10 ; SolenoidTableEntry10, 06=Left Slings, 40
606F: BD 83 46 JSR $8346 ;
6072: 05 ;
6073: BD 83 85 JSR $8385 ; EnqueueSolenoidPulse_ParameterByte()
6076: 05 ; SolenoidTableEntry05, 03=Outhole, 40

```

```

6077: BD 83 46   JSR   $8346   ;
607A: 05                               ;
607B: BD 83 85   JSR   $8385   ; EnqueueSolenoidPulse_ParameterByte()
607E: 09                               ; SolenoidTableEntry09, 09=Plunger, 40
607F: BD 83 46   JSR   $8346   ;
6083: 05                               ;
6083: BD 8B 77   JSR   $8B77   ; ScheduleFunctionStart()
6086: 00 E7                               ; ID 00E7 BallSearchDropTargetSleepAndReset()
6088: 60 D1 3B                               ;
608B: BD 83 85   JSR   $8385   ; EnqueueSolenoidPulse_ParameterByte()
608E: 13                               ; SolenoidTableEntry13, 0C=Knock Down, 20
608F: BD 83 46   JSR   $8346   ; Sleep()
6092: 02                               ;
6093: BD 83 85   JSR   $8385   ; EnqueueSolenoidPulse_ParameterByte()
6096: 13                               ; SolenoidTableEntry13, 0C=Knock Down, 20
6097: 7E 99 A2   JMP   $99A2   ;
-----
;
; BallSearchDropTargetUpAnd00E7Reset()
;
609A: BD 86 5B   JSR   $865B   ; LookupGameAdjustmentParameterIandCheckIfEqualsParam2()
; C-bit set when not-equal
609D: 14 00                               ; 0x14, FeatureAdjustment020, Drop Trgt. Broken
609F: 27 0C   BEQ   $60AD   ;
60A1: BD 8B 77   JSR   $8B77   ; ScheduleFunctionStart()
60A4: 00 E7                               ; ID 00E7 BallSearchDropTargetSleepAndReset()
60A6: 60 D1 3B                               ;
60A9: BD 83 85   JSR   $8385   ; EnqueueSolenoidPulse_ParameterByte()
60AC: 06                               ; SolenoidTableEntry06, 1C=Drop Target, 40
60AD: 39   RTS                               ;
-----
;
; ID 00E8 == Sleep3SecondsThenPulseGunKicker()
;
60AE: BD 83 46   JSR   $8346   ; Sleep()
60B1: C0                               ; 0xC0 == 3 seconds
;
60B2: BD 83 85   JSR   $8385   ; EnqueueSolenoidPulse_ParameterByte()
60B5: 08                               ; SolenoidTableEntry08, 02=Gun Kicker, 40
60B6: 7E 99 A2   JMP   $99A2   ;
-----
;
; ID 00B9 == Sleep3andHalfSecondsThenCancel00BD_and00BC()
;
60B9: BD 83 46   JSR   $8346   ; Sleep()
60BC: E3                               ; 0xE3 == ~3.5 seconds
;
60BD: BD 86 9E   JSR   $869E   ; CancelScheduledCallbackFunction()
60C0: 00 BD                               ;
60C2: BD 86 9E   JSR   $869E   ; CancelScheduledCallbackFunction()
60C5: 00 BC                               ; ID 00BC == Sleep3andHalfSeconds()
60C7: 7E 99 A2   JMP   $99A2   ;
-----
;
; ID 00BC == Sleep3andHalfSeconds()
;
60CA: BD 83 46   JSR   $8346   ; Sleep()
60CD: E0                               ; 0xE0 == 3.5 seconds

```



```

60CE: 7E 99 A2    JSR    $99A2          ;
;
-----;
;
; BallSearchDropTargetSleepAndReset()
;
; If 0xE3 is set (meaning target should be up) then kick it up (if not set to broken=yes)
; If 0xE3 is clr (meaning target should be dn) then kick it dn
;
; Function to set target to desired state after a ball-search
;;
60D1: BD 83 46    JSR    $8346          ; Sleep()
60D4: A0                                ; 0xA0 = 2.5 second sleep
60D5: BD 84 AD    JSR    $84AD          ; GetMemoryFlag()
60D8: E3                                ; If 0xE3 memory flag is set then do NOT do the
; drop-down. E3-set(c-clear)== dt-down
; 0xE3 cleared at dt-down-switch, set at dt-up
60D9: 24 0F      BCC    $60EA          ;
60DB: BD 83 85    JSR    $8385          ; EnqueueSolenoidPulse_ParameterByte()
60DE: 13                                ; SolenoidTableEntry13, 0C=Knock Down, 20
60DF: BD 83 46    JSR    $8346          ; Sleep()
60E2: 02                                ;
60E3: BD 83 85    JSR    $8385          ; EnqueueSolenoidPulse_ParameterByte()
60E6: 13                                ; SolenoidTableEntry13, 0C=Knock Down, 20
60E7: 7E 99 A2    JMP    $99A2          ;
;
-----;
;
60EA: BD 86 5B    JSR    $865B          ; LookupGameAdjustmentParameterlandCheckIfEqualsParam2()
; C-bit set when not-equal
60ED: 14 01                                ; 0x14, FeatureAdjustment020, Drop Trgt. Broken
60EF: 27 04      BEQ    $60F5          ;
60F1: BD 83 85    JSR    $8385          ; EnqueueSolenoidPulse_ParameterByte()
60F4: 06                                ; SolenoidTableEntry06, 1C=Drop Target, 40
60F5: 7E 99 A2    JMP    $99A2          ;
;
-----;

```

As shown, the ball-search 00 B9 function performs these operations:

- Sleep for E3 which is *slightly over* 3.5 seconds (0x40 = 1 second, 0xE0 = 3.5 seconds).
- Cancel Function ID 00 BD
- Cancel Function ID 00 BC

The 00 BC function, as shown above at \$60CA, simply sleeps for *exactly* 3.5 seconds. This allows other code to determine if 3.5 seconds has elapsed by simply checking of function 00 BC is running. The function 00 BD is located in a different bank at \$6675,31 (ROM offset 0x46675) and hasn't been traced and is not depicted here.

The following sections analyze both scenarios:

- Scenario 1, when the ball-search 00 B9 is scheduled while drop-target "Up" 00 B9 is running
- Scenario 2, when the drop-target "Up" 00 B9 is scheduled while ball-search 00 B9 is running

The scenarios are analyzed with these facts in mind:

- The ball-search 00 B9 is scheduled with \$8B77 scheduler which cancels any existing function running with the same ID prior to scheduling the function.
- The “MB Start DT Action” schedules the new “Up” function with scheduler \$8BC3 which will not schedule the function if one is already running with the same ID.

### ***Drop-Target Up Function ID 00 B9 Conflict Scenario 1***

This scenario analyzes the case where the “MB Start DT Action” new “Up” function 00 B9 is running and then the ball search engages and tries to schedule its 00 B9 function. As mentioned, this case seems highly unlikely to occur in game play since this scenario would involve a ball-search taking place as multiball starts up.

In this scenario, the multiball is starting, and “MB Start DT Action” determines the drop-target should be reset to the up-position. At this same time, the ball search is taking place which then needs to schedule its own 00 B9 function as part of its duties. As mentioned, the ball-search 00 B9 scheduler \$8B77 will cancel the existing 00 B9 function and then schedule the new 00 B9 function.

This means the drop-target will not be reset up, and the ball-search will proceed according to the ball-search code and not experience any unexpected ball-search behavior. The player will experience a case where multiball is starting, ball-trough being evacuated, while other playfield solenoids are kicking during ball-search, and throughout all of this, the drop-target will not have been kicked up from the “MB Start DT Action”.

The ball-search includes drop-target reset/knock-down activity as well. The ball-search is intended to end with the drop-target reset to whatever position it was prior to the ball-search, whether it was up or down prior to the ball-search. So if the drop-target was down when this scenario started, it will end up down even though the ball-search may have kicked it up at some point. Since the ball-search, however, ends when it detects a playfield switch, it seems likely that the ball-trough evacuation would have resulted in end of ball-search. *Precise details on how the ball-search ends has not been traced.*

It seems reasonable to suggest that this situation is not likely to occur during game play and, if it somehow does occur, the player is not likely to notice since there was a ball-search taking place which would be more unexpected than the drop target not getting reset at multiball start.

### ***Drop-Target Up Function ID 00 B9 Conflict Scenario 2***

This scenario analyzes the case where the ball-search function 00 B9 is running and then the multiball is started and “MB Start DT Action” determines that the drop-target should be in the up position and attempts to schedule its 00 B9 function.

In this scenario, the game is performing its ball-search routine and has started its 00 B9 function which runs for 3.5 seconds. During this 3.5 seconds, the game starts a multiball and the “MB Start DT Action” attempts to schedule its “Up” function using scheduler \$8BC3 which will not schedule the function if a 00 B9 function is already running. This effectively produces the same result as described in Scenario 1, above. The result is that the ball-search routine will continue to operate without having its 00 B9 function interrupted or cancelled. For the player, the result is that the drop-target is not reset to the up

position when it otherwise should have been due to the “MB Start DT Action” feature, however a ball search was taking place at multiball start so the drop-target position may be the least of the concern.

### *Drop-Target Up Function ID 00 B9 Conflict Drop-Target Switch Handler*

Additional behavior due to the 00 B9 conflict is related to the L8.3 code addition to the drop-target switch handler. Depicted earlier was how the drop-target switch handler has been updated to ensure that the switch closure is ignored if the “MB Start DT Action” has scheduled the new Up or Down function to reset the target up, or knock it down, respectively.

This also means that if a ball-search is taking place, and the player is able to hit a ball into the drop target causing it to go into the down position, the player will not be awarded anything for hitting the drop-target. The ball-search code and the drop-target switch handler are already designed to ignore the switch closure if the drop-target had been knocked down as part of the ball-search routine itself. This means the player may reasonable assume the lack of drop-target award was due to hitting the drop-target while a ball search was taking place, however this may not be the actual reason the award was not given.

In the overall scheme of things, this and the other behaviors that could occur due to the 00 B9 ID conflict are not likely to be considered a major problem however in some extremely rare circumstances, my be observable by the player.

**Note: Subsequent ROM update, in the future, will re-analyze this and change the “Up” function ID value to remove any chance of conflict.**

## **The L8.3 Timed 3-Bank Lamp Fixes**

During the L8.3 development, an observant tester brought up the odd behavior of original L-8 which could be addressed as part of L8.3 involving the 3-bank lamp behavior. This odd behavior is effectively a software bug in L-8 but is being addressed in L8.3 by having a new adjustment to select between original or corrected behavior. This was done this way in the event that some prefer the original behavior and incorporate it into their gameplay.

The problem is in regards to the center 3-bank lamps (each lamp associated with one of the three 3-bank standup targets). The issue is with regards to the timeout behavior of the 3-bank lamps not behaving in an expected or consistent way.

The 3-bank lamps behavior depends on the number of times the 3-bank targets have been completed by the player for the current game. The number of times the 3-bank targets have been completed is

compared against the Feature Adjustment value for A2.06 “Three Bank Count” which has default value of 2.

- When the 3-bank targets have NOT yet been completed number of times set in A2.06:
  - Each hit to a 3-bank target causes its lamp to remain solidly lit.
  - When all 3 of the 3-bank targets have been hit, award is given and all 3 lamps extinguish.
- When the 3-bank targets HAVE been completed number of times set in A2.06:
  - Each hit to a 3-bank target causes its lamp to remain solidly lit.
  - When only 1 target has been hit, the remaining 2 lamps are blinking
  - When only 2 targets have been hit, the remaining 1 lamp is blinking
  - When 1 or 2 lamps are blinking, all 3 lamps are subjected to timeout:
    - Timeout period is set by Feature Adjustmet A2.10 “Three Bank Timer”
    - Default timeout period is 15 seconds
    - When 2<sup>nd</sup> of the 3 targets is hit, timeout period restarts
    - When timeout period expires, **ALL THREE** lamps are extinguished
    - After timeout, all 3 targets need to behit again in order to complete the bank
  - When all three targets are hit, award is given and all 3 lamps are extinguished.

The problem behavior happens after the player drains their ball while the 3-bank targets are in a timeout period. When the player is served their next ball, the L-8 behavior is as follows:

- Any unhit targets that were previously blinking, are extinguished
- Any hit targets that were previously solid (while the remaining 1 or 2 lamps were previously blinking) now remain solidly lit and are **not subjected to timeout** as the player starts their ball.
  - In the case where two lamps are solidly lit at start of ball:
    - Re-hitting the solidly lit targets will not cause timeout to start
    - Hitting the 3<sup>rd</sup> will complete the 3-bank standup, giving award and extinguishing all three lamps.
  - In the case where one lamp is solidly lit at start of ball:
    - Re-hitting the solidly lit target will not cause timeout to start
    - Hitting a 2<sup>nd</sup> target will cause its lamp to be solidly lit and the remaining unhit target to start to blink, and **subject all three targets to timeout.**
      - When timeout expires, all three lamps are extinguished.

This behavior of L-8 is unexpected and inconsistent. Prior to the ball-draining, all three targets were subjected to a timeout period however at the next ball, hit targets are no longer subjected to timeout except in the case where only a single lit target was carried over to next ball and then player hits a 2<sup>nd</sup> target in the 3-bank. At that point, their previously solidly lit lamp now becomes subjected to timeout.

A more reasonable approach would be that all of the 3-bank lamps which were subjected to timeout when previous ball drained should be extinguished at the start of the next ball. This is the basis of this new adjustment in L8.3.



This new adjustment allows section between the original L-8 logic as described above, or



The selection "Off at EOB" refers to the new logic which will entail having the timed 3-bank lamps being turned off at end-of-ball.

This new logic, as its adjustment text implies, ensures that all three of the 3-bank lamps which were subject to timeout are extinguished. More specifically, instead of only the unhit lamps being extinguished, ALL lamps are extinguished. Note, although the adjustment is named "Off at EOB" the actual moment that the hit lamps (which were subject to timeout) are extinguished is as the *start* of the next ball. Using EOB was a lot easier to convey the new logic in the limited amount of space for adjustment text. An observant player will also notice that the blinking lamps extinguish when the ball drains and the solid lamps (which were subject to timeout) are extinguished during the start of the next ball.

The way in which the original L-8 code extinguishes the blinking lamps at end-of-ball is fairly straightforward. End of ball code cancels the function responsible for maintaining the timeout state of the unhit 3-bank targets. Normally, during game-play the timer function will extinguish all three lamps when the timer expires but cancelling the function at end-of-ball prevents this from taking pace.

For the hit targets which were subject to timeout, it appears the solid lamps need to be extinguished from the ball startup code which is responsible for establishing the various state data for the new ball, including lamp states. This startup code is important especially when you consider a multiplayer game where each player has a different set of acquired achievements.

The start-of-ball function is depicted in full below, however it is only partially annotated. This function is located at \$625B,3B (ROM offset 0x6E25B).

```
-----;-----  
625B: 34 16      PSHS  X,B,A      ; StartOfBallCurrentPlayer_ResetPlayfieldState()  
625D: BD 84 AD    JSR   $84AD      ; GetMemoryFlag()  
6260: D0          ;  
6261: 24 04      BCC   $6267      ;  
6263: BD 85 53    JSR   $8553      ;  
6266: 34          ;  
6267: BD 84 8F    JSR   $848F      ; ClearMemoryFlag()
```

```

626A: D1                                ;
626B: BD 86 5B    JSR    $865B          ; LookupGameAdjustmentParameterlandCheckIfEqualsParam2()
                                           ; C-bit set when not-equal
626E: 04 00
6270: 24 49          BCC    $62BB          ; 0x04, FeatureAdjustment004, Consolation Ball
                                           ; If Consolidation ball is not off then branch to $62BB
                                           ;
                                           ;-----
                                           ; No consolidation ball, advance to next player
                                           ;-----

6272: BD B1 D1    JSR    $B1D1          ;
6275: 25 44          BCS    $62BB          ; Retain current player ball
6277: BD B3 CA    JSR    $B3CA          ;
627A: 4D          TSTA
627B: 26 3E          BNE    $62BB          ; Retain current player
627D: 8E 05 AD    LDX    #$05AD          ;
6280: BD FB 29    JSR    $FB29          ; IncrementXByPlayerIndexNumber()
6283: 7E 62 86    JMP    $6286          ; <nop>
6286: 6D 84          TST    ,X          ;
6288: 26 31          BNE    $62BB          ; Branch down to retain current player
628A: BD B1 AB    JSR    $B1AB          ; GetCurrentPlayerIndexIntoA()
628D: BD BB 3E    JSR    $BB3E          ; GetPlayerScoreIndexAintoU()
6290: 6D C4          TST    ,U          ;
6292: 26 27          BNE    $62BB          ;
6294: A6 41          LDA    $0001,U      ;
6296: 81 03          CMPA   #$03          ;
6298: 24 21          BCC    $62BB          ;
629A: BD 88 F5    JSR    $88F5          ;
629D: 4C 7B 38
62A0: 6D 88 22    TST    $22,X        ;
62A3: 26 16          BNE    $62BB          ;
62A5: EC 88 22    LDD    $22,X        ;
62A8: C1 1E          CMPB   #$1E          ;
62AA: 22 0F          BHI    $62BB          ;
62AC: BD 88 D5    JSR    $88D5          ;
62AF: 00 1F
62B1: BD 84 80    JSR    $8480          ; SetMemoryFlag()
62B4: D1
62B5: BD 88 F5    JSR    $88F5          ;
62B8: 57 23 31
                                           ;
62BB: BD 86 5B    JSR    $865B          ; LookupGameAdjustmentParameterlandCheckIfEqualsParam2()
                                           ; C-bit set when not-equal
62BE: 03 01
62C0: 24 05          BCC    $62C7          ; 0x03, FeatureAdjustment003, Extraball Memory
                                           ; extraball-memory is not on, then jump past the lampOff
62C2: BD 87 22    JSR    $8722          ; LampOff() // likely turns off extraball lamp
62C5: 0F 40
62C7: BD 83 E8    JSR    $83E8          ;
62CA: 08
62CB: BD 84 8F    JSR    $848F          ; ClearMemoryFlag()
62CE: 42
62CF: BD 84 80    JSR    $8480          ; SetMemoryFlag()
62D2: D2
62D3: BD 88 F5    JSR    $88F5          ;
62D6: 68 08 31
62D9: BD 87 BE    JSR    $87BE          ;
62DC: 19 40
62DE: BD 84 AD    JSR    $84AD          ; GetMemoryFlag() // C-bit clear when flag is set
62E1: 41          ; Extra-Ball flag?
62E2: 24 1B          BCC    $62FF          ; Jumps over the following when extra ball being served
62E4: BD F7 DE    JSR    $F7DE          ; GetXTablePointer_05DB_ForCurrentPlayer()
62E7: 7E 62 EA    JMP    $62EA          ;
62EA: 6F 84          CLR    ,X          ;

```

```

62EC: 6F 01      CLR  $0001,X      ;
62EE: 6F 02      CLR  $0002,X      ;
62F0: 6F 03      CLR  $0003,X      ;
62F2: 6F 04      CLR  $0004,X      ;
62F4: 8E 05 91    LDX  #$0591       ;
62F7: BD FB 29    JSR  $FB29        ; IncrementXByPlayerIndexNumber()
62FA: 7E 62 FD    JMP  $62FD        ; <nop>
62FD: 6F 84      CLR  ,X           ;
62FF: BD 84 8F    JSR  $848F        ; ClearMemoryFlag()
6302: 41          ; Clear extra-ball flag(?)
6303: 8E 05 95    LDX  #$0595       ;
6306: BD FB 29    JSR  $FB29        ; IncrementXByPlayerIndexNumber()
6309: 7E 63 0C    JMP  $630C        ; <nop>
630C: A6 84      LDA  ,X           ;
630E: 8E 05 99    LDX  #$0599       ;
6311: BD FB 29    JSR  $FB29        ; IncrementXByPlayerIndexNumber()
6314: 7E 63 17    JMP  $6317        ; <nop>
6317: A1 84      CMPA ,X           ;
6319: 26 08      BNE  $6323        ;
631B: BD 84 1C    JSR  $841C        ; ValidateThenSingleLampSetParameterBytePlane0()
631E: 30          ; 30 == Left Ramp Lamp
631F: BD 84 1C    JSR  $841C        ; ValidateThenSingleLampSetParameterBytePlane0()
6322: 3A          ; 3A == Right Ramp Lamp
6323: BD 87 BE    JSR  $87BE        ;
6326: 0B 10      ;
6328: BD 87 BE    JSR  $87BE        ;
632A: 0B 00      ;
632D: BD 86 5B    JSR  $865B        ; LookupGameAdjustmentParameterlandCheckIfEqualsParam2()
; C-bit set when not-equal
6330: 07 00      ; 0x07, FeatureAdjustment007, Kickback Setting
6332: 24 07      BCC  $633B        ; Kickback is not extra-easy, skip over the following
6334: BD 86 5B    JSR  $865B        ; LookupGameAdjustmentParameterlandCheckIfEqualsParam2()
; C-bit set when not-equal
6337: 07 01      ; 0x07, FeatureAdjustment007, Kickback Setting
6339: 25 0E      BCS  $6349        ; Kickback setting is easy, skip over the following
633B: 86 FF      LDA  #$FF         ;
633D: 97 B1      STA  $B1         ;
633F: BD 84 1C    JSR  $841C        ; ValidateThenSingleLampSetParameterBytePlane0()
6342: 09          ; 09 == Kickback lamp
6343: BD 84 80    JSR  $8480        ; SetMemoryFlag()
6346: 44          ;
6347: 20 1B      BRA  $6364        ;
6349: BD 86 5B    JSR  $865B        ; LookupGameAdjustmentParameterlandCheckIfEqualsParam2()
; C-bit set when not-equal
634C: 07 04      ; 0x07, FeatureAdjustment007, Kickback Setting
634E: 25 0C      BCS  $635C        ; Kickback setting is 'extra hard' skip over the
following
6350: 0F B1      CLR  $B1         ;
6352: BD 84 8F    JSR  $848F        ; ClearMemoryFlag()
6355: 44          ;
6356: BD 84 2B    JSR  $842B        ; ClrSingleLampParamByteBank0TestBankChkValidations()
6359: 09          ; 09 == Kickback lamp
635A: 20 08      BRA  $6364        ;
635C: BD 84 AD    JSR  $84AD        ; GetMemoryFlag() // c-bit clear when flag set
635F: 44          ;
6360: 24 D9      BCC  $633B        ;
6362: 20 EC      BRA  $6350        ;
6364: BD 66 36    JSR  $6636        ;
6367: BD 88 F5    JSR  $88F5        ;
636A: 47 8B 31    ;
636D: BD 85 B2    JSR  $85B2        ;
6370: 08          ;

```



```

6371: BD 84 80    JSR    $8480        ; SetMemoryFlag()
6374: D0                                ;
6375: BD 84 AD    JSR    $84AD        ; GetMemoryFlag() // c-bit clear when flag set
6378: 45                                ;
6379: 25 05      BCS    $6380        ;
637B: BD 87 15    JSR    $8715        ; LampOnParameterByte1PlaneParameterByte2()
637E: 2F 40                                ; 2F == Database 1 Lamp
6380: BD 84 AD    JSR    $84AD        ; GetMemoryFlag() // c-bit clear when flag set
6383: 46                                ;
6384: 25 04      BCS    $638A        ;
6386: BD 84 1C    JSR    $841C        ; ValidateThenSingleLampSetParameterBytePlane0()
6389: 1A                                ; 1A == Database 2 Lamp
638A: BD 84 AD    JSR    $84AD        ; GetMemoryFlag() // c-bit clear when flag set
638D: 47                                ;
638E: 25 04      BCS    $6394        ;
6390: BD 84 1C    JSR    $841C        ; ValidateThenSingleLampSetParameterBytePlane0()
6393: 0C                                ; 0C == Right Return Lane Lamp
6394: 8E 5F 43    LDX    #$5F43        ;
6397: BD 88 F5    JSR    $88F5        ;
639A: 5F 16 3D                                ;
639D: 25 06      BCS    $63A5        ;
639F: BD 88 F5    JSR    $88F5        ;
63A2: 50 AE 31                                ;
63A5: 35 96      PULS   A,B,X,PC    ;
-----; -----

```

The full playfield-state reset function is shown above for reference. Within the function, an appropriate location was chosen where code can jump to a new function to perform the task of extinguishing the 3-bank lamps if they were subject to timeout during previous ball in play for the current player.

The modified portion of the function is shown below with new code highlighted.

```

...
62FF: BD 84 8F    JSR    $848F        ; ClearMemoryFlag()
6302: 41                                ; Clear extra-ball flag(?)
6303: 8E 05 95    LDX    #$0595        ;
6306: BD FB 29    JSR    $FB29        ; IncrementXByPlayerIndexNumber()
6309: 7E 63 0C JMP    $630C ; <nop>
6309: BD 79 00    JSR    $7900        ; Three-Bank Timed Lamp Handler
630C: A6 84      LDA    ,X            ;
630E: 8E 05 99    LDX    #$0599        ;
6311: BD FB 29    JSR    $FB29        ; IncrementXByPlayerIndexNumber()
6314: 7E 63 17    JMP    $6317        ; <nop>
...

```

As shown, at \$6309 was, effectively, a NOP instruction which simply jumped to the very next instruction. This JMP instruction was replaced with a JSR to \$7900 which will jump to the new 3-bank fixup function located in unused region of ROM bank \$3B at 7900 (ROM offset 0x6F900). The new function is as follows:

```

-----; -----
7900: 34 16      PSHS   X,B,A        ;
7902: BD 86 5B    JSR    $865B        ; LookupGameAdjustmentParameter1andCheckIfEqualsParam2()
                                ; C-bit set when not-equal
7905: 1B 01                                ; 0x1B, FeatureAdjustment027, TIMED 3BANK LAMP
7907: 25 17      BCS    $7920        ; C-bit set when not set to 0x01 'OFF AT EOB'
                                ; so assume it is default/off, and return, no work
                                ;

```

```

; Here when c-bit is set which means timed 3-bank
; lamps need turned off if at eob
;
7909: 8E 05 A5    LDX  #$05A5    ; $05A5 = Number of 3-bank completions per player
790C: BD FB 29    JSR  $FB29    ; IncrementXByPlayerIndexNumber()
790F: A6 84      LDA  ,X        ; A has number of 3-bank completions for current player
7911: BD 83 0C    JSR  $830C    ; Get8BitSettingIntoBParameterByte()
7914: 06          ; 0x06, FeatureAdjustment006, Three Bank Count
7915: 34 04      PSHS B        ;
7917: A1 E0      CMPA ,S+      ; Checking if number of 3-bank completions exceeds
; configured "Three Bank Count" value
7919: 25 05      BCS  $7920    ; C-bit set means not at a point where targets
; should be timed, done, no work
;
; Determined player has reached point in
; which standups should be timed.
; Call function to force off the 3 standup lamps
;
791B: BD 87 BE    JSR  $87BE    ; This call results in extinguish of the solid lamps
791E: 06 00      ; Gets into $AE53 with X=0x0600, Y=0x9E92, A=0xFF B=0x04
;
7920: 35 96      PULS A,B,X,PC ;
;
-----;-----

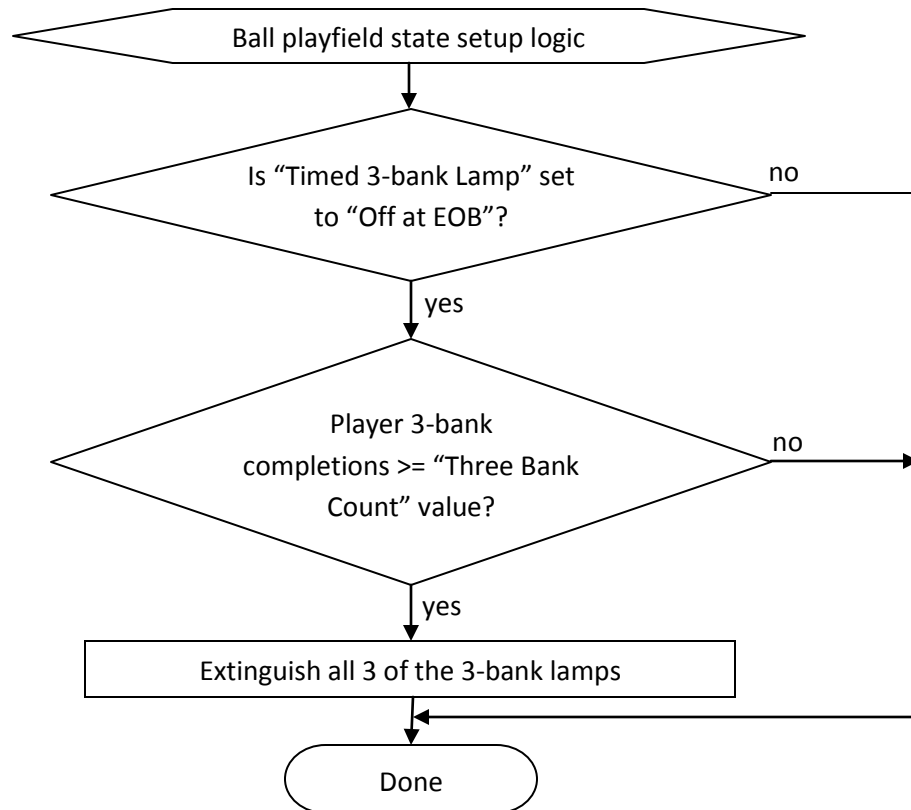
```

The new function, above, to handle extinguishing the timed 3-bank lamps performs the following:

- Checks configured “Timed 3-bank lamp” adjustment, if “Original, then return
- Checks configured “Three Bank Count” adjustment and compares against the number of 3-bank completions that the current player has achieved so far in their game.
  - If player has’t yet completed enough 3-bank target completions to cause timed-lamps, then return
  - If player has completed enough 3-bank target completions to cause timed-lamps then a function is called to turn off all of the 3-bank lamps. This is the same function used by the normal timeout function when game times out all 3 lamps during game play.

The logic, as described above, results in the extinguishing of the 3-bank lamps at ball start if the player has reached the point in which the 3-bank targets are subjected to timeout, thus achieving the desired behavior.

For reference, the new logic is depicted below in a flowchart.



## The L8.3 Ball-Search Bug Fixes

During L8.3 development a ball-search problem was reported and nicely depicted with video demonstration of the problem. The problem was also found to be present in the original L-8 code. The issue is that, during ball-search when the drop-target is knocked down as part of ball-search, it causes the game to award the player with points and sound-call as if the player had intentionally hit the drop-target with the ball.

To reproduce the issue, simply remove the ball from game-play and wait for the ball-search to commence. When the ball-search knocks down the drop target, you will observe point accumulation and a sound-call.

While investigating this issue, a secondary issue was also discovered. The secondary issue is that the ball-search will reset the target to the up-position when the Feature Adjustment A2.20 "Drop Target Broken" is set to "On". In all other cases of game code, when the A2.20 adjustment is set to "On", the game will specifically *not* pulse the drop-target reset coil however the ball-search code from L-8 is allowing the drop-target to kick up in this condition. In fact, the problem code in ball-search will specifically try to reset the drop target up *only when* the A2.20 adjustment is set to "On".

A majority of the ball-search code was depicted in the previous section describing the "Multiball Start Drop-Target Action" feature. The applicable portions of ball-search code are shown below:

First, during phase2 of the ball-search which starts at \$5FA3,3B (ROM offset 0x6DFA3), there is the following portion of code that calls \$609A and then performs 3 pulses of the drop-target knock-down solenoid. Note there are other sections of the ball-search that also pulse the drop-target knock down solenoid however they are not problematic and do not have the same scenario as depicted in the trouble code below.

```

;-----
; ID 00EE == BallSearchFrom5FA3()
; BallSearchPhase2() starts here
;-----
;
5FA3: BD 60 9A    JSR    $609A    ; BallSearchDropTargetUpAnd00E7Reset()
;
5FA6: BD 83 85    JSR    $8385    ; EnqueueSolenoidPulse_ParameterByte()
5FA9: 13          ; SolenoidTableEntry13, 0C=Knock Down, 20
5FAA: BD 83 46    JSR    $8346    ; Sleep()
5FAD: 02          ;
5FAE: BD 83 85    JSR    $8385    ; EnqueueSolenoidPulse_ParameterByte()
5FB1: 13          ; SolenoidTableEntry13, 0C=Knock Down, 20
5FB2: BD 83 46    JSR    $8346    ; Sleep()
5FB5: 02          ;
5FB6: BD 83 85    JSR    $8385    ; EnqueueSolenoidPulse_ParameterByte()
5FB9: 13          ; SolenoidTableEntry13, 0C=Knock Down, 20
5FBA: BD 83 46    JSR    $8346    ; Sleep()
5FBD: 05          ;
...

```

The above ball-search code appears to call function at \$609A and then proceeds to knock the drop-target down 3 times. Referring back to previously depicted drop-target switch handler function, it was previously shown that the drop-target switch function checks for scheduled function ID 00 E7 as a case in which the drop-target switch will be ignored (and, as such, not accumulate points). This leads to the conclusion that the call to \$609A must unconditionally schedule function 00 E7 so that the subsequent drop-target knock down solenoid pulses will, if successfully knocks down the drop-target, result in the drop-target switch closure being ignored.

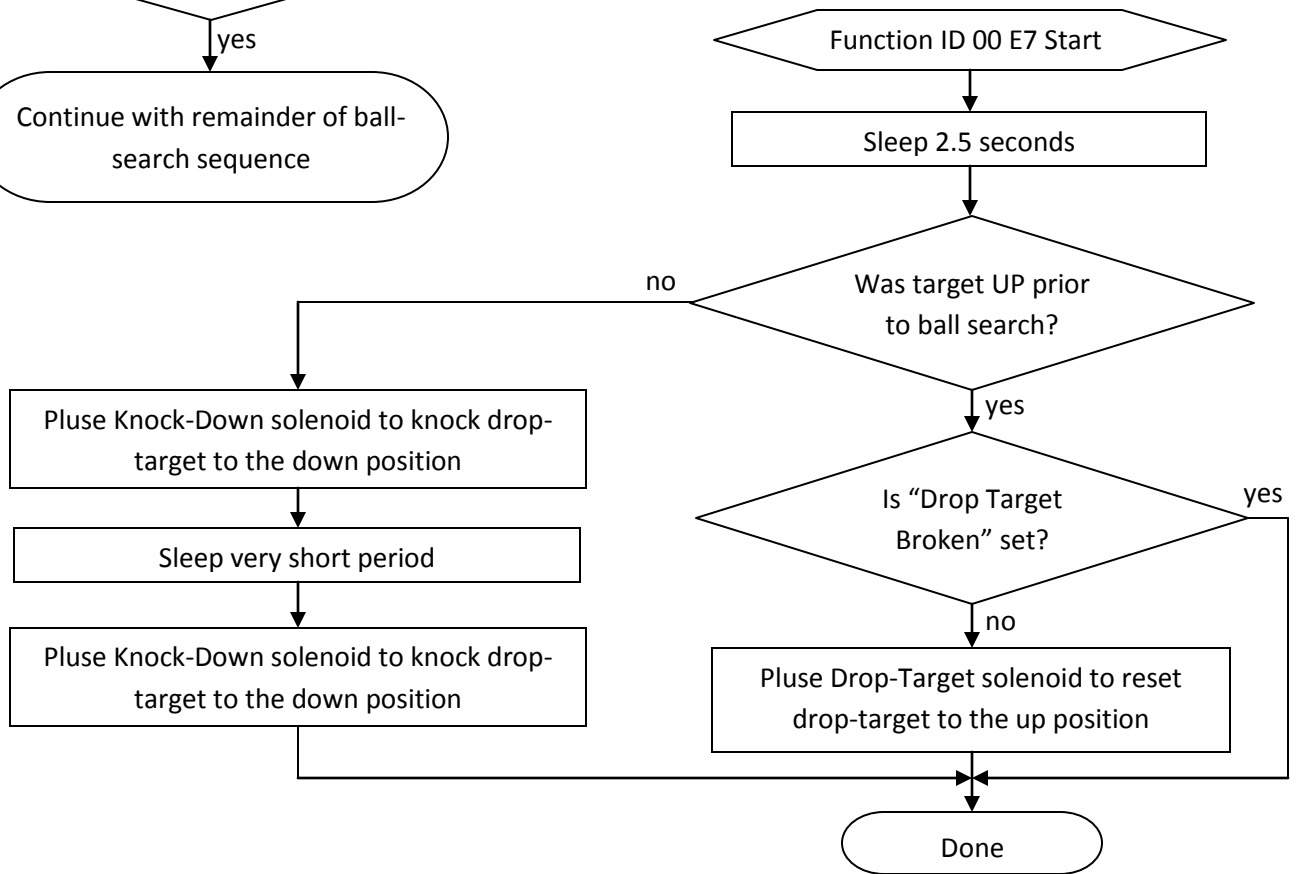
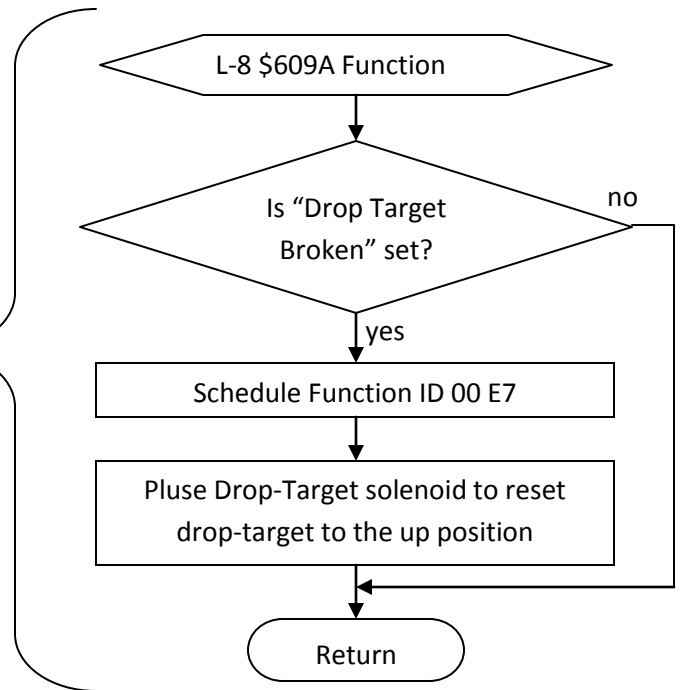
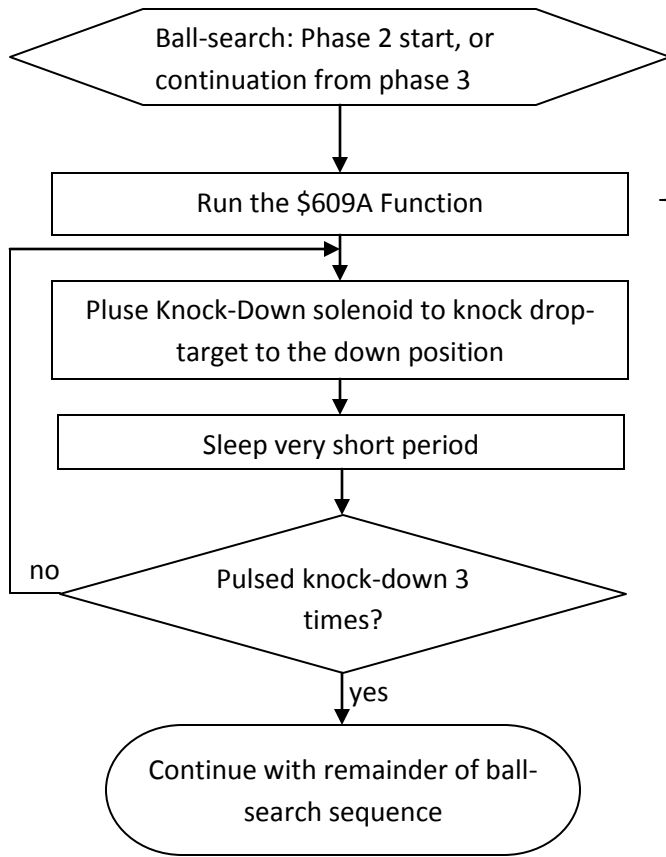
The content of the \$609A,3B function is depicted below (ROM offset 0x6E09A).

```

;-----
;
609A: BD 86 5B    JSR    $865B    ; BallSearchDropTargetUpAnd00E7Reset()
;
609D: 14 00          ; LookupGameAdjustmentParameterlandCheckIfEqualsParam2()
609F: 27 0C          BEQ    $60AD    ; C-bit set when not-equal
60A1: BD 8B 77    JSR    $8B77    ; 0x14, FeatureAdjustment020, Drop Trgt. Broken
60A4: 00 E7          ; ScheduleFunctionStart()
60A6: 60 D1 3B          ; ID 00E7 BallSearchDropTargetSleepAndReset()
60A9: BD 83 85    JSR    $8385    ; EnqueueSolenoidPulse_ParameterByte()
60AC: 06          ; SolenoidTableEntry06, 1C=Drop Target, 40
60AD: 39          RTS
;
;-----

```

The L-8 \$609A function, above employs the logic as shown in the following flowchart:



As evident in the flowchart, above, the logic for function at \$609A is flawed in two ways:

- The \$609A function should schedule the 00 E7 function unconditionally but it does not. The \$609A function needs to unconditionally schedule the 00 E7 function because the ball-search phase 2 code will always attempt to knock down the drop-target after \$609A is done, and the knocking down of the drop-target causes the drop-target switch handler to check for 00 E7 scheduled function as basis of ignoring the drop-target switch. Therefore, the 00 E7 function must always be scheduled whenever the ball-search code pulses the knock-down solenoid.
- The \$609A function pulses the drop-target to reset the drop target up, but only when the adjustments are such that the drop-target is deemed as broken. This is obviously the opposite of what is correct logic. Considering that all other uses of the “Drop Target Broken” adjustment prevent the drop-target solenoid from being engaged when the setting is set to indicate the drop-target is broken.

The updated code for L8.3 for the \$609A function is as follows:

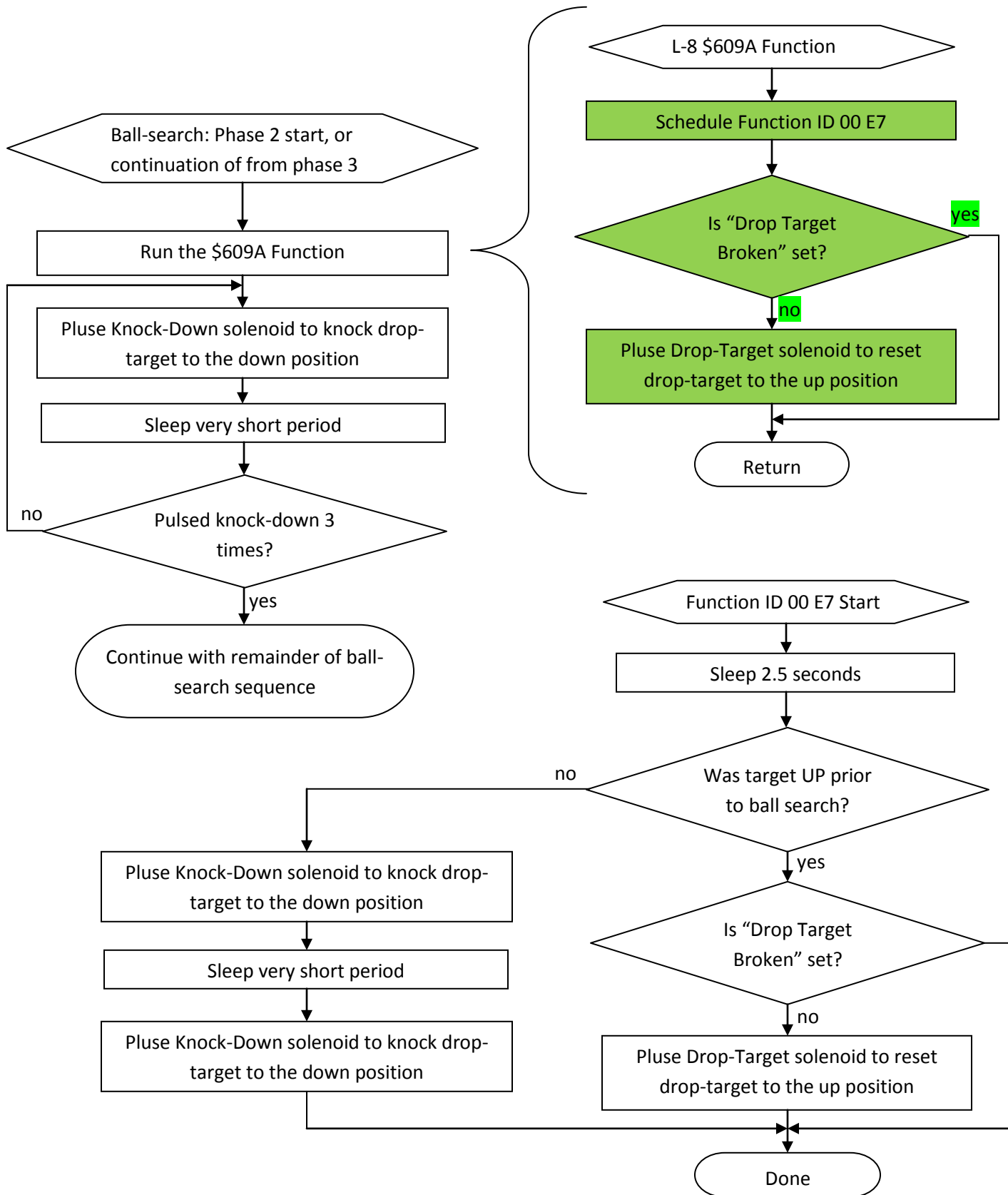
```
-----;-----  
;   
; BallSearchDropTargetUpAnd00E7Reset()  
;  
609A: BD 8B 77 JSR $8B77 ; ScheduleFunctionStart()  
609D: 00 E7 ; ID 00E7 BallSearchDropTargetSleepAndReset()  
609F: 60 D1 3B ;   
60A2: BD 86 5B JSR $865B ; LookupGameAdjustmentParameter1andCheckIfEqualsParam2()  
; C-bit set when not-equal  
60A5: 14 01 ; 0x14, FeatureAdjustment020, Drop Trgt. Broken  
60A7: 27 04 BEQ $60AD ;   
60A9: BD 83 85 JSR $8385 ; EnqueueSolenoidPulse_ParameterByte()  
60AC: 06 ; SolenoidTableEntry06, 1C=Drop Target, 40  
60AD: 39 RTS ;   
;   
-----;-----
```

The corrected function addresses both problems by:

- Unconditionally scheduling function ID 00 E7
- Pulses the drop-target solenoid only if adjustments do not indicate drop-target is broken.

With these changes in place, the ball search knock-down of the drop-target will no longer cause points to accumulate and no longer cause a sound-call to take place. When the “Drop Target Broken” is set to “Yes” the ball-search code will no longer make any attempt to reset the drop-target to the up position.

The new function changes the logic of the flowchart to the following. **Changed logic is highlighted.**





## Appendix

This section contains additional information not covered in the previous sections.

### Solenoid Table

Several of the code samples make reference to the pulsing of a solenoid. The function that pulses solenoid take an index number that corresponds to a row within the solenoid table. The solenoid table is at \$4C8F,3D (ROM offset 0x74C8F) and is shown below.

```
-----;-----  
;   
; SolenoidTable[]   
4C8F: 00 14 ;   
4C91: 02 ;   
;   
; SolenoidTableEntry00, NULL   
4C92: 00 00 ;   
4C94: 0A 10 ; SolenoidTableEntry01, 0A=Top Lock, 10   
4C96: 0A 20 ; SolenoidTableEntry02, 0A=Top Lock, 20   
4C98: 10 20 ; SolenoidTableEntry03, 10=Left Lock, 20   
4C9A: 04 40 ; SolenoidTableEntry04, 04=Trough, 40   
4C9C: 03 40 ; SolenoidTableEntry05, 03=Outhole, 40   
4C9E: 1C 40 ; SolenoidTableEntry06, 1C=Drop Target, 40   
4CA0: 01 40 ; SolenoidTableEntry07, 01=Ball Popper, 40   
4CA2: 02 40 ; SolenoidTableEntry08, 02=Gun Kicker, 40   
4CA4: 09 40 ; SolenoidTableEntry09, 09=Plunger, 40   
4CA6: 0B FF ; SolenoidTableEntry0A, 0B=Gun Motor, FF   
4CA8: 0B 00 ; SolenoidTableEntry0B, 0B=Gun Motor, 00   
4CAA: 0D 40 ; SolenoidTableEntry0C, 0D=Left Jet, 40   
4CAC: 0E 40 ; SolenoidTableEntry0D, 0E=Right Jet, 40   
4CAE: 0F 40 ; SolenoidTableEntry0E, 0F=Bottom Jet, 40   
4CB0: 05 40 ; SolenoidTableEntry0F, 05=Right Sling, 40   
4CB2: 06 40 ; SolenoidTableEntry10, 06=Left Sling, 40   
4CB4: 08 40 ; SolenoidTableEntry11, 08=Kickback, 40   
4CB6: 07 40 ; SolenoidTableEntry12, 07=Knocker, 40   
4CB8: 0C 20 ; SolenoidTableEntry13, 0C=Knock Down, 20   
;   
-----;-----
```

Each entry in the solenoid table contains 2 bytes. The first byte corresponds to the solenoid transistor that is associated with the solenoid. The second byte corresponds to a time period associated with the solenoid pulse, effectively defining the power associated with the solenoid pulse. *This second byte value usage has been inferred based on code examination and could be subject to correction or refinement.*

### Checksum Bytes

The T2 ROM utilizes common WPC Checksum method and, as such, various tools and documentation is available online to describe the checksum calculation. The 16-bit sum of the bytes in the ROM must equate to the value of the 2 checksum bytes located in unpagged region at \$FFEE (ROM offset 0x7FFEE).

The region of ROM bytes containing checksum is shown below.

```
FFEC: DF 41 ; When set to 00 FF, debug mode engages  
FFEE: 73 08 ; Checksum  
FFF0: 8E CD ; Interrupt Vector: Reserved (undefined vector)  
FFF2: 8E D1 ; Interrupt Vector: SWI3 Software Interrupt  
FFF4: 8E D5 ; Interrupt Vector: SWI2 Software Interrupt
```

```

FFF6: 8E C0           ; Interrupt Vector: FIRQ Fast Interrupt request
FFF8: D9 C0           ; Interrupt Vector: IRQ  Interrupt Request
FFFA: 8E BD           ; Interrupt Vector: SWI  Software Interrupt
FFFC: 8E DE           ; Interrupt Vector: NMI  Non-maskable interrupt
FFFE: 8C 8B           ; Interrupt Vector: Restart

```

The L8.3 checksum is highlighted in the ROM output, above. Some interesting characteristics of the checksum are as follows:

- The 2<sup>nd</sup> byte of the checksum is used by the game to declare the version. For L-8, the 2<sup>nd</sup> byte is 0x08. For L8.1, L8.2 and L8.3, efforts were in place to ensure the second byte remain at 0x08.
- The 2<sup>nd</sup> byte of the checksum is saved in battery-backed RAM at \$180B so the game remembers the most recent ‘version’ of software has used.
- At startup, if the battery-backed copy of the checksum version-byte at \$180B doesn’t match the currently running software checksum second-byte, then memory is wiped and “Factory Settings Restored” happens.
- The L8.1, L8.2, L8.3 retain 0x08 as second byte to prevent the “Factory Settings Restored” from taking place when switching between L-8 and these variations. Factory Settings don’t need to be reset in this situation due to the commonalities between memory usage among these images.
- In order to generate a checksum that specifically has fixed byte 0x08 as its second byte, it is necessary to fixup a 3<sup>rd</sup> byte in ROM image so that the sum of all bytes results in a 16-bit value with 2<sup>nd</sup> byte having the desired 0x08 byte.
- Without certainty about which ROM bytes were used in the original L-8 as the fixup bytes, for L8.3 an unused byte in ROM at \$7FBF,35 (ROM Offset 0x57FBF) was adjusted in order to get the desired checksum with 2<sup>nd</sup> byte having byte value 0x08. *Online references indicate the bytes preceding the checksum at \$FFEC could be used for such purpose.*
- If the 2 bytes immediately prior to the checksum at \$FFEC are set to 00 FF, the game power-up will bypass the “Testing...” phase and immediately go to attract mode. This is a debug mode to allow quick testing of new code without worrying about checksum and should not be used for released game ROMs. *It is not clear if other behaviors ensure while in this mode.*
  - Some 3<sup>rd</sup> party sound boards may fail to play sounds when game powers up in this mode as they, apparently, require the power-up messages from the CPU board before they will function, after a power-up.
- If the high nibble of the checksum contains ‘D’ it enables a ‘prototype’ designation which will cause the version string to use “P-8” instead of “L-8”. For example if the 2<sup>nd</sup> byte of the checksum had 0xD8 instead of 0x08. *Changing checksum in this way will alter the 2<sup>nd</sup> byte in such a way that “Factory Settings Restored” will take place.*
- The L8.1, L8.2 and L8.3 modify the version strings so that they automatically contain the “.1”, “.2” or “.3”, respectively. The underlying game code treats it like L-8 but simply displays the updated string which contains the extra version suffix designator.

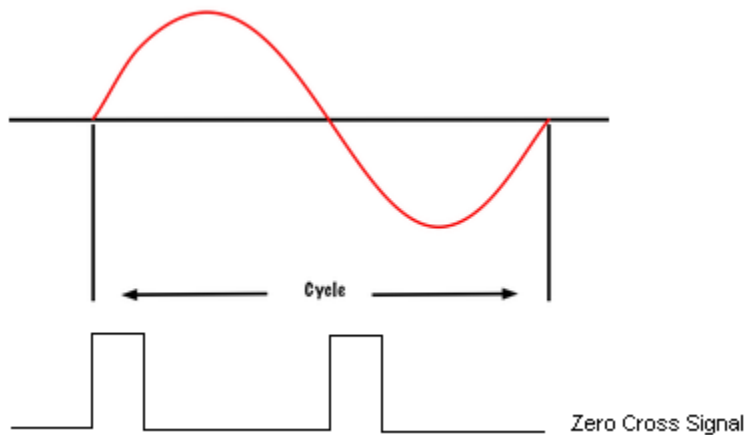
## General Illumination and Zero Cross

This section describes some information about General Illumination circuit which was gathered during the development of L8.3 to have better understanding of the lighting in T2. The information was

gathered from online resources and from general electric/electronic knowledge. It may be possible some of the information provided is not 100% accurate however it can serve as a starting point for understanding the General Illumination design of T2 and other WPC games.

The General Illumination circuit utilizes a 6.3 A/C circuit which is gated by a triac component. There are 5 such circuits or GI strings, in most WPC games. The original design is such that each circuit can accommodate up to 18 incandescent bulbs.

The interesting part of how the triacs behave, is that the ASIC on the CPU board gets a “zero-cross” signal (aka “ZC”) which is simply a single wire signal (pcb trace) that originates from the power-driver board. The ZC signal simply indicates whenever the A/C sine wave crosses the zero voltage x-axis. For systems that are on 60hz line voltage, the ZC signal pulses at a rate of 120 times per second. For systems that are on 50hz line voltage, the ZC signal pulses at a rate of 100 times per second. This is due to how a full hertz cycle consists of 2 crossings of the sine wave across the 0 axis (once as the wave is going up and once as the wave is going down).



Shown above is the ZC signal that occurs based on the sine wave of the AC line voltage. These 2 pulses occur for each cycle. This signal depiction is based on the healthy ZC signal depicted online at [www.pinwiki.org](http://www.pinwiki.org).

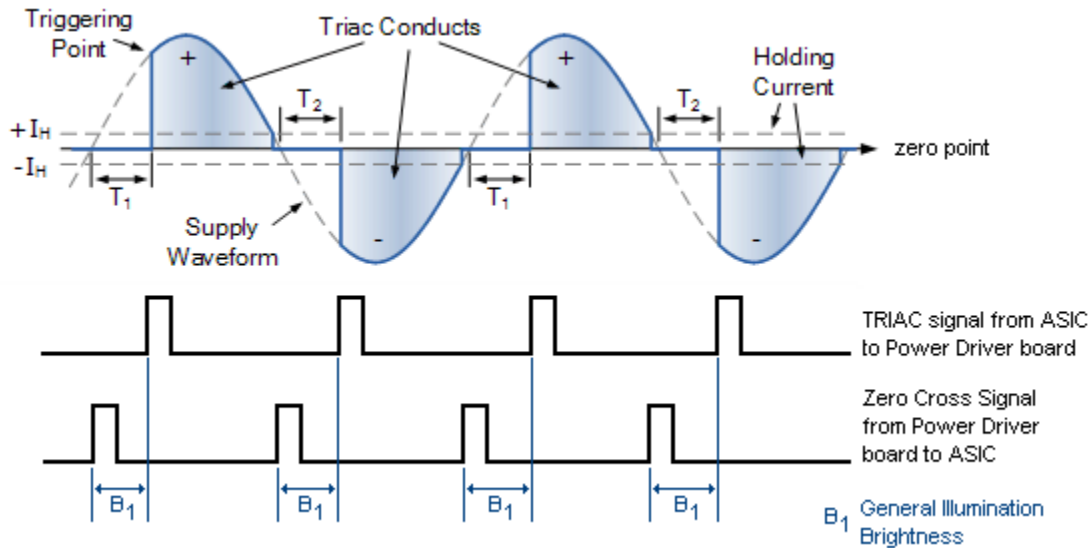
The ASIC on the CPU board takes the ZC signal and, from it, controls a single signal to the Triacs in order to produce dimming effect. The ZC signal goes into the ASIC. The ASIC also receives L8.3 software originated messages from the CPU to control brightness levels for the 5 GI circuits. The ASIC then controls the “TRIAC” signal back to the power driver board which gets fanned out to the 5 different triacs at chip U1 (refer to board schematics and pinwiki for details)

When the game is set to full GI brightness, the ZC signal is ignored and the triacs are always ‘on’.

When game needs to dim the GI, the ZC signal is used along with the desired brightness level by the ASIC to set the signal levels to control each triac. In this case, the triac itself is inherently aware of zero-cross as it is directly provided the A/C voltage, it does not need the distinct ZC signal mentioned above. When

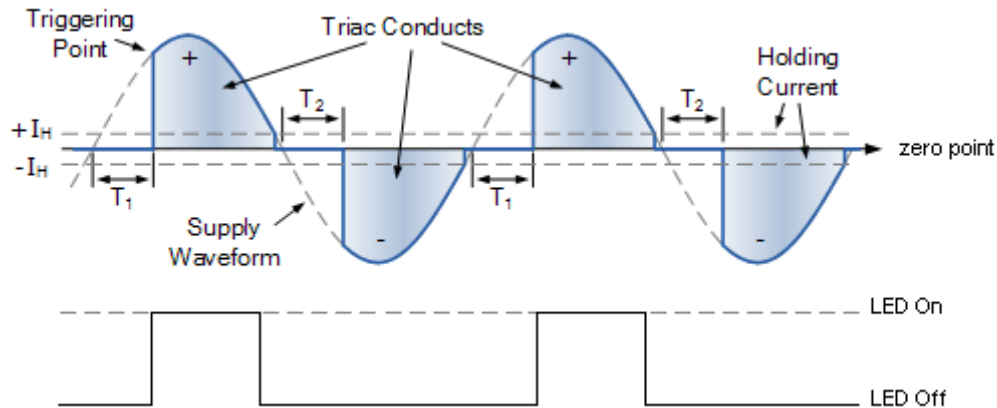
A/C crosses the zero point (regardless if the sine wave is going up or going down in its cycle) the triac will turn OFF the voltage to the lamps. Then some tiny moment in time after this, the ASIC sends short signal to turn on the triac for the remainder of the half-cycle. If brightness is to be mostly bright, then this short signal is sent soon after the zero-cross. If the brightness is to be more dim, then this short signal is sent a while later. When the sine wave crosses zero again, the process starts over.

Taking an image from internet search of “triac sine wave” the following image is copyright by its respective owner (multiple sites host this image) and augmented to indicate ZC and TRIAC signals:

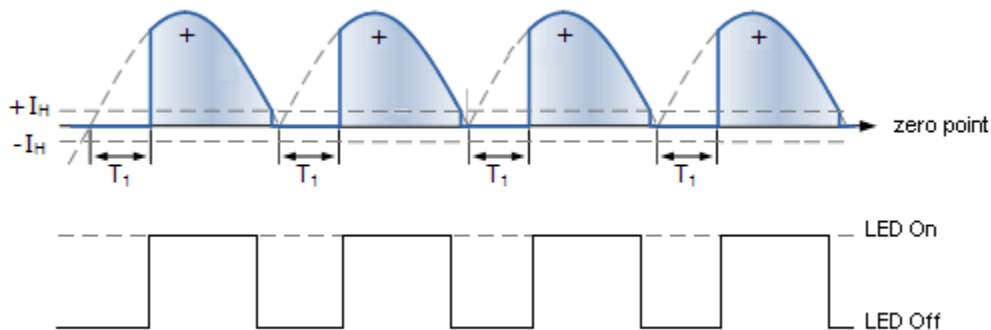


The drawing above is an example of the triac output during GI dimming. The blue portions are when the voltage is “on”. If the game wants to dim the GI further, the  $B_1$  value would increase so that the blue regions would then be smaller. This effectively dims incandescent bulbs by reducing the time period that they get power during each A/C cycle. Since the ‘off’ moments are at a rate of 120 or 100 times per second and since incandescent bulbs retain illumination for a brief moment when power is taken away, this produces a mostly decent dimming effect.

For those using LEDs in the GI circuit, it should be noted that since the LEDs only conduct in one direction, everything below the zero would be an “Off” LED period. *Depending on wiring of the GI and of the LED bulb, it may be the opposite, meaning everything below the zero could be on while everything above the zero is off.* This means that even without enabling “Dimming” in the game, the LEDs are only half as bright as they could be.



Using a full-wave bridge rectifier (4 diodes) could effectively flip the bottom part of the sine-wave to the top, allowing the LED to be on for both parts of the sine wave, such as depicted below.



Using a capacitor could then smooth the voltage or cause the on-time of the LED to be longer during each cycle and possibly allow the LED to participate in the GI dimming provided by the WPC system without noticeable flickering. The purpose of these statements are to inspire hobbyists to consider these things when using LEDs in GI circuits and whether to disable the GI dimming in the game adjustments or consider using external circuitry to allow better dimming of the LEDs with less flicker.

**Any such experiments with bridge rectifiers, diodes, capacitors, etc are done at your own risk. Ideally, only those with proficient knowledge in these areas should be attempting such things.**

### Attract Mode Code

The attract mode code, mostly annotated, is shown below for reference. A lot of different functions get called for the different portions of the attract mode. This code may be helpful for those interested in further understanding how the T2 code works. This code includes some comments about where such L8.3 hooks would be located to alter the original attract mode for L8.2 and L8.3 design. This code starts at \$793F,30 (ROM offset 0x4393F).

```

-----
;
793F: BD FB AE      JSR   $FBAE      ; Clear display data

```

```

7942: 7E 79 45      JMP    $7945      ; <nop>
                                     ; Determine if in game-over or power-up mode
7945: BD 84 AD      JSR    $84AD      ; GetMemoryFlag()
7948: D3              ; 0xD3 game-over mode, set by game-over code in $3B
                                     ;
7949: 24 09          BCC    $7954      ; c-bit clear = game-over mode, skip over bash effect
                                     ;
                                     ; c-bit set = power-up mode, play bash effect
794B: BD 88 F5      JSR    $88F5      ; CallBankedFunction_Param_WPCAddr()
794E: 76 CB 35      JSR    $88F5      ; AttractMode_T2BashEffect() at start of attract mode
7951: 16 00 98      LBRA   $79EC      ; Now go down to power-up attract mode after bash effect
                                     ;
                                     ;-----
                                     ;
                                     ; Game-Over Attract-Mode Starts Here
                                     ;
7954: BD 84 AD      JSR    $84AD      ; GetMemoryFlag()
7957: DB              ; 0xDB indicating power-up sequence started
                                     ;
7958: 10 24 00 90    LBCC   $79EC      ;
                                     ;
795C: BD 84 AD      JSR    $84AD      ; GetMemoryFlag()
795F: DA              ; 0xDA flag gets cleared in bank 0x3B
                                     ;
7960: 24 21          BCC    $7983      ;
                                     ;
7962: BD 84 80      JSR    $8480      ; SetMemoryFlag()
7965: DA              ; 0xDA flag gets cleared in bank 0x3B
7966: BD 84 80      JSR    $8480      ; SetMemoryFlag()
7969: DC              ; 0xDC main inner attract block played 2 times
796A: BD 84 80      JSR    $8480      ; SetMemoryFlag()
796D: D9              ; 0xD9 main outter attract block played 5 times
                                     ;
796E: BD 7C 43      JSR    $7C43      ; AttractMode_HighScores()
7971: BD 88 F5      JSR    $88F5      ; CallBankedFunction_Param_WPCAddr()
                                     ; L8.3 Attract Mode Fixup 09:
                                     ; If L8.2 mode: Replace FanClubMessage with
                                     ; CreditsInsertCoin/LastGamesScores
7974: 7E DB 35      JSR    $88F5      ; -->AttractMode_FanClubMessage()
7977: BD 88 F5      JSR    $88F5      ; CallBankedFunction_Param_WPCAddr()
                                     ; L8.3 Attract Mode Fixup 10:
                                     ; If L8.1 mode: CastCredits
                                     ; else for L8.2 do TerminatorLightning
797A: 48 03 24      JSR    $88F5      ; -->AttractMode_CastCredits()
797D: BD 88 F5      JSR    $88F5      ; CallBankedFunction_Param_WPCAddr()
                                     ; L8.3 Attract Mode Fixup 11:
                                     ; If L8.1 mode: SpecialThanks and CustomRomMessage
                                     ; else for L8.2 CyborgComputerReadout and
                                     ; CustomRomMessage
7980: 51 67 24      JSR    $88F5      ; -->AttractMode_SpecialThanks()
7983: C6 03          LDB    #$03      ;
7985: 34 04          PSHS   B          ;
7987: BD 7B E1      JSR    $7BE1      ; AttractMode_LastGameScores()
                                     ; Call $7F69 function that clears display prior
                                     ; to LastGameScores in special circumstances.
798A: BD 7D 02      JSR    $7D02      ; AttractMode_ReplayAt()
798D: BD 7D 9F      JSR    $7D9F      ; AttractMode_GameOver()
7990: BD 7B 74      JSR    $7B74      ; AttractMode_CreditsInsertCoin()
7993: BD 7C 43      JSR    $7C43      ; AttractMode_HighScores()
7996: BD 88 F5      JSR    $88F5      ; CallBankedFunction_Param_WPCAddr()
                                     ; L8.3 Attract Mode Fixup 12:
                                     ; If L8.2 mode: Replace FanClubMessage with

```

```

; CreditsInsertCoin/LastGamesScores
7999: 7E DB 35 ; -->AttractMode_FanClubMessage()
799C: BD 7B 33 JSR $7B33 ; AttractMode_WilliamsLogoBlockyWipe()
799F: BD 7B 58 JSR $7B58 ; AttractMode_Presents()
79A2: BD 88 F5 JSR $88F5 ; CallBankedFunction_Param_WPCAddr()
79A5: 76 CB 35 ; -->AttractMode_T2BashEffect()
79A8: BD FB E2 JSR $FBE2 ; AttractMode_StaringArnold()
79AB: 7E 79 AE JMP $79AE ; <null>
; L8.3 Attract Mode Fixup 13:
; If L8.2 mode: GameOver
79AE: BD 7B E1 JSR $7BE1 ; AttractMode_LastGameScores()
79B1: BD 7D 02 JSR $7D02 ; AttractMode_ReplayAt()
79B4: BD 7C 43 JSR $7C43 ; AttractMode_HighScores()
79B7: BD 7B 74 JSR $7B74 ; AttractMode_CreditsInsertCoin()
79BA: BD 88 F5 JSR $88F5 ; CallBankedFunction_Param_WPCAddr()
79BD: 44 12 24 ; -->AttractMode_PullTrigger()
79C0: BD 7B E1 JSR $7BE1 ; AttractMode_LastGameScores()
79C3: BD 7D 02 JSR $7D02 ; AttractMode_ReplayAt()
79C6: BD 7B 00 JSR $7B00 ; AttractMode_WilliamsLogoDraw()
79C9: BD 7B E1 JSR $7BE1 ; AttractMode_LastGameScores()
79CC: BD 7C 43 JSR $7C43 ; AttractMode_HighScores()
79CF: BD 88 F5 JSR $88F5 ; CallBankedFunction_Param_WPCAddr()
79D2: 7E 43 35 ; -->AttractMode_ArnoldShootingShotgun()
79D5: BD 88 F5 JSR $88F5 ; CallBankedFunction_Param_WPCAddr()
79D8: 7E B2 35 ; -->AttractMode_SayNoToDrugs()
79DB: BD 88 F5 JSR $88F5 ; CallBankedFunction_Param_WPCAddr()
; L8.3 Attract Mode Fixup 14:
; Play CustomMessage and CustomRomMessage
79DE: 54 86 24 ; -->AttractMode_CustomMessage()
79E1: BD 88 F5 JSR $88F5 ; CallBankedFunction_Param_WPCAddr()
79E4: 7E 18 33 ; -->AttractMode_CyborgComputerReadout()
79E7: 35 04 PULS B ;
79E9: 5A DECB ;
79EA: 26 99 BNE $7985 ;
;
-----
;
; The following is done at power-up, code jumps here
; immediately after having played the T2 bash effect.
;
79EC: BD 84 80 JSR $8480 ; SetMemoryFlag()
79EF: DB ; 0xDB indicating power-up sequence started
79F0: C6 05 LDB #$05 ;
79F2: 34 04 PSHS B ;
79F4: BD 88 F5 JSR $88F5 ; CallBankedFunction_Param_WPCAddr()
; L8.3 Attract Mode Fixup 01:
; Replace SpecialThanks with
; SpecialThanks/CustomRomMessage
79F7: 51 67 24 ; -->AttractMode_SpecialThanks()
79FA: C6 02 LDB #$02 ;
79FC: 34 04 PSHS B ;
79FE: BD 7D 02 JSR $7D02 ; AttractMode_ReplayAt()
7A01: BD 7C 43 JSR $7C43 ; AttractMode_HighScores()
7A04: BD 88 F5 JSR $88F5 ; CallBankedFunction_Param_WPCAddr()
; L8.3 Attract Mode Fixup 02:
; If L8.2 mode: Replace FanClubMessage with
; CreditsInsertCoin/LastGamesScores
7A07: 7E DB 35 ; -->AttractMode_FanClubMessage()
7A0A: BD 7B 74 JSR $7B74 ; AttractMode_CreditsInsertCoin()
; L8.3 Attract Mode Fixup 03:
; If L8.1 mode: CreditsInsertCoin
; else for L8.2 ReplayAt

```



7A0D: BD 88 F5	JSR	\$88F5	; CallBankedFunction_Param_WPCAddr()
7A10: 78 92 35			; -->AttractMode_T2ShineyLogo()
7A13: BD FB E2	JSR	\$FB E2	; AttractMode_StaringArnold()
7A16: 7E 7A 19	JMP	\$7A19	; <null>
			; L8.3 Attract Mode Fixup 04:
			; If L8.2 mode: GameOver
7A19: BD 7B E1	JSR	\$7BE1	; AttractMode_LastGameScores()
7A1C: BD 7B 00	JSR	\$7B00	; AttractMode_WilliamsLogoDraw()
7A1F: BD 7B 58	JSR	\$7B58	; AttractMode_Presents()
7A22: BD 88 F5	JSR	\$88F5	; CallBankedFunction_Param_WPCAddr()
7A25: 76 CB 35			; -->AttractMode_T2BashEffect()
7A28: BD 7C 43	JSR	\$7C43	; AttractMode_HighScores()
7A2B: BD 7B 74	JSR	\$7B74	; AttractMode_CreditsInsertCoin()
7A2E: BD 7B 1D	JSR	\$7B1D	; AttractMode_WilliamsLogoBlocky()
7A31: BD 7B 58	JSR	\$7B58	; AttractMode_Presents()
7A34: BD 88 F5	JSR	\$88F5	; CallBankedFunction_Param_WPCAddr()
7A37: 78 62 35			; -->AttractMode_JudgementDay()
7A3A: BD FB E2	JSR	\$FB E2	; AttractMode_StaringArnold()
7A3D: 7E 7A 40	JMP	\$7A40	; <null>
			; L8.3 Attract Mode Fixup 05:
			; If L8.2 mode: LastGameScores
7A40: BD 7C 43	JSR	\$7C43	; AttractMode_HighScores()
7A43: BD 7B 74	JSR	\$7B74	; AttractMode_CreditsInsertCoin()
7A46: BD 88 F5	JSR	\$88F5	; CallBankedFunction_Param_WPCAddr()
7A49: 44 12 24			; -->AttractMode_PullTrigger()
7A4C: BD 7D 02	JSR	\$7D02	; AttractMode_ReplayAt()
7A4F: BD 86 5B	JSR	\$865B	; LookupGameAdjustmentParameterlandCheckIfEqualsParam2()
			; C-bit set when not-equal
7A52: 10 00			; 0x10 == Attract Sounds, C-bit = not equal to 0x00
7A54: 24 17	BCC	\$7A6D	;
			;
7A56: BD 84 AD	JSR	\$84AD	; GetMemoryFlag()
7A59: D9			; 0xD9 main outer attract block played 5 times
			;
7A5A: 25 11	BCS	\$7A6D	; Normally: C-bit set, skip "I am a cybornetic organism"
			;
7A5C: BD 8B 77	JSR	\$8B77	; ScheduleFunctionStart()
7A5F: 00 E4			; <related to "I am a cybornetic organism" and music>
7A61: 7E F0 30			;
7A64: 86 04	LDA	#\$04	;
7A66: BD C0 DB	JSR	\$C0DB	;
			;
7A69: BD 85 46	JSR	\$8546	; DoSoundTableParameterByte()
7A6C: A5			; 0xA5="I am a cybernetic organism"
			;
7A6D: BD 88 F5	JSR	\$88F5	; CallBankedFunction_Param_WPCAddr()
7A70: 7C A1 33			; -->AttractMode_TerminatorLightning()
7A73: BD 7B 33	JSR	\$7B33	; AttractMode_WilliamsLogoBlockyWipe()
7A76: BD 7B 58	JSR	\$7B58	; AttractMode_Presents()
7A79: BD 88 F5	JSR	\$88F5	; CallBankedFunction_Param_WPCAddr()
7A7C: 78 92 35			; -->AttractMode_T2ShineyLogo()
7A7F: BD FB E2	JSR	\$FB E2	; AttractMode_StaringArnold()
7A82: 7E 7A 85			; <null>
			; Call hook function that only does this when NOT time
			; to pay "Cybornetic organism" speech
			; L8.3 Attract Mode Fixup 06:
			; If L8.2 mode: LastGameScores
7A85: BD 86 5B	JSR	\$865B	; LookupGameAdjustmentParameterlandCheckIfEqualsParam2()
			; C-bit set when not-equal
7A88: 10 00			; 0x10 == Attract Sounds, C-bit set when not 0x00
7A8A: 24 16	BCC	\$7AA2	;
			;

```

7A8C: BD 84 AD      JSR   $84AD      ; GetMemoryFlag()
7A8F: D9              ;   0xD9 main outer attract block played 5 times
                          ;
7A90: 25 10          BCS   $7AA2      ;
                          ;
7A92: BD 85 46      JSR   $8546      ; DoSoundTableParameterByte()
7A95: 9D              ;   0x9D="I am a cyberdyne systems series 800 terminator"
                          ;
7A96: BD 8B 77      JSR   $8B77      ; <related to "I am a Cyberdyne series 800-terminator">
7A99: 00 E4          ;
7A9B: 7E A2 30      ;
                          ;
7A9E: BD 84 8F      JSR   $848F      ; ClearMemoryFlag()
7AA1: D9              ;   0xD9 main outer attract block played 5 times
                          ;
7AA2: BD 88 F5      JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
7AA5: 7E 18 33      ; -->AttractMode_CyborgComputerReadout()
7AA8: BD 7B 74      JSR   $7B74      ; AttractMode_CreditsInsertCoin()
7AAB: BD 88 F5      JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
7AAE: 44 12 24      ; -->AttractMode_PullTrigger()
7AB1: BD 7D 02      JSR   $7D02      ; AttractMode_ReplayAt()
7AB4: BD 7C 43      JSR   $7C43      ; AttractMode_HighScores()
7AB7: BD 7B 74      JSR   $7B74      ; AttractMode_CreditsInsertCoin()
7ABA: BD 88 F5      JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
7ABD: 78 62 35      ; -->AttractMode_JudgementDay()
7AC0: BD FB E2      JSR   $FBE2      ; AttractMode_StaringArnold()
7AC3: 7E 7A C6      ; <null>
                          ;   L8.3 Attract Mode Fixup 07:
                          ;   If L8.2 mode: LastGameScores
7AC6: BD 88 F5      JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
7AC9: 7E 43 35      ; -->AttractMode_ArnoldShootingShotgun()
7ACC: BD 88 F5      JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
7ACF: 7E B2 35      ; -->AttractMode_SayNoToDrugs()
7AD2: BD 88 F5      JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
                          ;   L8.3 Attract Mode Fixup 08:
                          ;   Play CustomMessage and CustomRomMessage
7AD5: 54 86 24      ; -->AttractMode_CustomMessage()
7AD8: 35 04          PULS  B          ;
7ADA: 5A            DECB           ;
7ADB: 10 26 FF 1D   LBNE  $79FC      ;
7ADF: BD 88 F5      JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
7AE2: 7D 82 35      ; -->AttractMode_IamTheFuture()
                          ;
7AE5: BD 84 80      JSR   $8480      ; SetMemoryFlag()
                          ;   L8.3 Attract Mode Fixup 15:
                          ;   At inner-x2 if L8.3 set flag for Cybornetic Sounds
7AE8: DC            ;   0xDC main inner attract block played 2 times
7AE9: 35 04          PULS  B          ;
7AEB: 5A            DECB           ;
7AEC: 10 26 FF 02   LBNE  $79F2      ;
7AF0: BD 7D 5A      JSR   $7D5A      ; AttractMode_TimeDate()
7AF3: BD 88 F5      JSR   $88F5      ; CallBankedFunction_Param_WPCAddr()
7AF6: 48 03 24      ; -->AttractMode_CastCredits()
                          ;
7AF9: BD 84 80      JSR   $8480      ; SetMemoryFlag()
7AFC: D9              ;   0xD9 main outer attract block played 5 times
                          ;
7AFD: 16 FE EC      LBRA  $79EC      ;
                          ;
-----

```

## Fan Club Code

Shown below is the T2 Fan Club code that was removed from L8.3. It is being shown as an aid for those interested in T2 code since this function involves writing some messages on the display and also checking the system date. In L-8 ROM this code started at \$7EDB,35 (ROM offset 0x57EDB).

```
-----;-----  
;   
; AttractMode_FanClubMessage() (74 bytes of ROM)   
;   
7EDB: BD 86 5B JSR $865B ; LookupGameAdjustmentParameterlandCheckIfEqualsParam2()   
; C-bit set when not-equal   
7EDE: 12 00 ; 0x12 == T2FanClub, C-bit set when not equal to 0x00   
7EE0: 27 42 BEQ $7F24 ;   
7EE2: 4F CLRA ;   
7EE3: BD 88 F5 JSR $88F5 ; CallBankedFunction_Param_WPCAddr()   
7EE6: 42 DC 39 ; GetTimeDateYearIntoD()   
7EE9: 10 83 07 C8 CMPD #$07C8 ; Checks if current year is 1992 or less   
7EED: 25 0D BCS $7EFC ; If year is less than 1992 then C is set, go to $7EFC   
;   
7EEF: 22 33 BHI $7F24 ; If year is > 1992 then no fan club message, go to end   
;   
7EF1: 4F CLRA ;   
7EF2: BD 88 F5 JSR $88F5 ; CallBankedFunction_Param_WPCAddr()   
7EF5: 42 E5 39 ; GetTimeDateMonthIntoA()   
7EF8: 81 06 CMPA #$06 ; Checks if current month is June   
7EFA: 22 28 BHI $7F24 ; If greater than June, skip to the end.   
; So fan club not allowed after June 1992   
7EFC: BD D3 4C JSR $D34C ;   
7EFF: BD D3 60 JSR $D360 ;   
7F02: BD D7 99 JSR $D799 ;   
7F05: 00 D0 ; String index 0xD0 = "TERMINATOR 2"   
7F07: 01 ; Font index 0x01 = 7 high single stroke   
7F08: 40 07 ; Center horizontally, bottom starts at line 7   
7F0A: BD D7 99 JSR $D799 ;   
7F0D: 00 D1 ; String index 0xD1 = "Fan Club"   
7F0F: 01 ; Font index 0x01 = 7 high single stroke   
7F10: 40 11 ; Center horizontally, bottom starts at line 17   
7F12: BD D7 99 JSR $D799 ;   
7F15: 00 D2 ; String index 0xD2 = "Call 1-800-237-4400"   
7F17: 01 ; Font index 0x01 = 7 high single stroke   
7F18: 40 1C ; Center horizontally, bottom starts at line 28   
7F1A: BD 88 F5 JSR $88F5 ; CallBankedFunction_Param_WPCAddr()   
7F1D: 7F 57 33 ;   
7F20: BD 83 46 JSR $8346 ; Sleep()   
7F23: C0 ;   
7F24: 39 RTS ;   
-----;-----
```

## ROM Image Changes

The table, below, identifies every ROM change in L8.3 as compared to the official L-8 ROM image with a brief description of each ROM change.

ROM Offset	WPC Address	Original Bytes	Original Description	New Bytes	New Description
<b>0x1012C</b>	\$412C,24	C9 52	Security Level animation. Original jump address at end of animation.	57 49	Security Level animation. New jump address to \$5749,24 new code.
<b>0x11749</b> - <b>0x11756</b>	\$5749,24 - \$5756,24	FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Unused bytes in bank \$24	34 02 86 55 97 B4 35 02 BD FB AE 7E C9 52	New end-of-animation code to prevent flicker at end of the animation.
<b>0x400D5</b> - <b>0x40103</b>	\$40D5,30 - \$4103,30	00 16 02 41 04 41 09 41 19 41 2A 41 3B 41 4C 41 5D 41 6E 41 7F 41 90 41 A1 41 B2 41 C1 41 CF 41 DD 41 EB 41 F9 42 08 42 19 42 25 42 35 42 47	Feature Adjustments, English selections string pointers.	00 0C 02 41 04 67 62 67 6C 67 7D 67 8D 67 9D 67 AE 67 B9 51 8B 4E 6F 4E C2 57 B6 41 62 7A 75 67 20 5A 69 65 68 65 6E 00 FF FF FF FF FF FF FF	Sound Test, English selections string pointer table at \$40D5 - \$40EF. German string at \$40F0 - 40FC. Unused bytes at \$40FD - \$4103.
<b>0x404AB</b> - <b>0x404D9</b>	\$44AB,30 - \$44D9,30	00 16 02 41 04 44 DA 44 EA 44 FB 45 0D 45 18 45 29 45 3A 45 4A 45 5B 45 6C 45 7B 45 89 45 96 45 A3 45 B2 45 BF 42 08 42 19 42 25 42 35 42 47	Feature Adjustments, German selections string pointers.	00 0C 02 41 04 6D C4 6D CE 6D DC 6D EB 6D FD 6E 0E 6E 1B 5B CF 6E 25 5A 36 57 B6 FF	Sound Test, German selections string pointer table at \$44AB - \$44C5. Unused bytes at \$44C6 - \$44D9.
<b>0x40577</b>	\$4577,30	49 45	German string content.	45 49	Corrected German text.
<b>0x407EA</b> - <b>0x40818</b>	\$47EA,30 - \$4818,30	00 16 02 41 04 48 19 48 2B 48 3A 48 4C 48 5C 48 6E 48 80 48 91 48 A4 48 B7 48 C8 48 DA 48 EC 41 DD 48 FE 49 10 49 21 42 19 49 33 49 44 42 47	Feature Adjustments, French selections string pointers.	00 0C 02 41 04 71 97 71 A7 71 B7 71 C7 71 D6 1 E3 71 F3 71 FE 72 0C 72 17 57 B6 FF	Sound Test, French selections string pointer table at \$47EA - \$4804. Unused bytes at \$4805 - \$4818.
<b>0x41864</b>	\$5864,30	5A EE 5A F4	German string pointers.	5B 4F 40 F0	Corrected/fixed pointers.
<b>0x4187E</b> - <b>0x41889</b>	\$587E,30 - \$5889,30	5B 43 5B 4E 5B 58 5B 5E 5B 63 5B 67 51 21	German string pointers.	5B 44 51 03 51 0B 51 12 51 18 51 1D 51 21	Corrected/fixed pointers.
<b>0x41952</b>	\$5952,30	5D CA	German string pointer.	5A EE	Corrected/fixed pointer.
<b>0x419C6</b>	\$59C6,30	4E 66	German FUA pointer.	57 B6	Corrected/fixed pointer.
<b>0x419D6</b>	\$59D6,30	4E 66	German FUA pointer.	57 B6	Corrected/fixed pointer.
<b>0x41AA1</b>	\$5AA1,30	4B 20	German string content.	43 4B	Corrected German text.
<b>0x41AEE</b> - <b>0x41AFF</b>	\$5AEE,30 - \$5AFF,30	5A 69 65 68 65 00 56 6F 72 20 41 62 73 63 68 75 73 73	German string content.	44 41 53 20 56 49 53 49 45 52 20 42 45 57 45 47 45 4E	Repurposed German string content as part of text corrections.
<b>0x41B38</b> - <b>0x41B69</b>	\$5B38,30 - \$5B69,30	54 42 41 43 4B 20 42 45 4C 2E 00 4B 49 54 42 41 43 4B 20 42 45 00 4B 49 54 42 41 43 4B 20 20 00 4B 49 54 42 41 00 4B 49 54 42 00 4B 49 54 00 4B 49 00	German string content.	43 4B 42 41 43 4B 20 42 45 4C 2E 00 4B 49 43 4B 42 41 43 4B 20 42 00 5A 69 65 6C 65 6E 20 55 6E 64 20 44 65 6E 00 FF FF FF FF FF FF FF FF FF FF FF FF	Corrected German text. Repurposed German string content as part of text corrections. Moved string now unused bytes at \$5B5E - \$5B69.

<b>0x41C00</b> - <b>0x41C0D</b>	\$5C00,30 - \$5C0D,30	53 54 41 52 54 20 44 52 55 43 4B 45 4E	German string content.	44 52 55 45 43 4B 45 20 53 54 41 52 54	Corrected German text.
<b>0x41DC2</b> - <b>0x41DD7</b>	\$5DC2,30 - \$5DD7,30	20 54 41 54 54 45 4E 00 42 49 4C 44 20 57 45 43 48 53 45 4C 4E 00	German string content.	54 41 53 54 45 52 4E 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	Corrected German text. Moved string now unused bytes at \$5DCA - \$5DD7.
<b>0x42029</b> - <b>0x42039</b>	\$6029,30 - \$6039,30	4E 66 4E 66	French FUA pointer.	57 B6 57 B6	Corrected/fixed pointer.
<b>0x42749</b> - <b>0x42761</b>	\$6749,30 - \$6761,30	00 0B 02 41 04 67 62 67 6C 67 7D 67 8D 67 9D 67 AE 67 B9 51 8B 4E 6F 4E C2	Sound Test, English selections string pointers.	FF FF	Moved table. Now unused bytes.
<b>0x42DAB</b> - <b>0x42DC3</b>	\$6DAB,30 - \$6DC3,30	00 0B 02 41 04 6D C4 6D CE 6D DC 6D EB 6D FD 6E 0E 6E 1B 5B CF 6E 25 5A 36	Sound Test, German selections string pointers.	FF FF	Moved table. Now unused bytes.
<b>0x4317E</b> - <b>0x43196</b>	\$717E,30 - \$7196,30	00 0B 02 41 04 71 97 71 A7 71 B7 71 C7 71 D6 71 E3 71 F3 71 FE 72 0C 72 17	Sound Test, French selections string pointers.	FF FF	Moved table. Now unused bytes.
<b>0x43942</b>	\$7942,30	7E 79 45	Attract Mode start, NOP instruction.	BD 7F 79	Jump to fix for WPC Custom Message "Testing..." bug.
<b>0x43971</b> - <b>0x439AD</b>	\$7971,30 - \$79AD,30	BD 88 F5 7E DB 35 BD 88 F5 48 03 24 BD 88 F5 51 67 24 C6 03 34 04 BD 7B E1 BD 7D 02 BD 7D 9F BD 7B 74 BD 7C 43 BD 88 F5 7E DB 35 BD 7B 33 BD 7B 58 BD 88 F5 76 CB 35 BD FB E2 7E 79 AE	Attract Mode routine	BD 7F AB 12 12 12 BD 7F E4 12 12 12 BD 7F B3 12 12 12 C6 03 34 04 BD 7F 69 BD 7D 02 BD 7D 9F BD 7B 74 BD 7C 43 BD 7F CD 12 12 12 BD 7B 33 BD 7B 58 BD 88 F5 76 CB 35 BD FB E2 BD 7F D7	Attract Mode updates, calling new logic in the \$30 bank.
<b>0x439DB</b> - <b>0x43A18</b>	\$79DB,30 - \$7A18,30	BD 88 F5 54 86 24 BD 88 F5 7E 18 33 35 04 5A 26 99 BD 84 80 DB C6 05 34 04 BD 88 F5 51 67 24 C6 02 34 04 BD 7D 02 BD 7C 43 BD 88 F5 7E DB 35 BD 78 74 BD 88 F5 78 92 35 BD FB E2 7E 7A 19	Attract Mode routine	BD 7F AB 12 12 12 BD 88 F5 7E 18 33 35 04 5A 26 99 BD 84 80 DB C6 05 34 04 BD 7F 9E 12 12 12 C6 02 34 04 BD 7D 02 BD 7C 43 BD 7F CD 12 12 12 BD 7F C1 BD 88 F5 78 92 35 BD FB E2 BD 7F D7	Attract Mode updates, calling new logic in the \$30 bank.
<b>0x43A3D</b>	\$7A3D,30	7E 7A 40	"	BD 7F DF	"
<b>0x43A82</b>	\$7A82,30	7E 7A 85	"	BD 7F 60	"
<b>0x43AC3</b>	\$7AC3,30	7E 7A C6	"	BD 7F DF	"
<b>0x43AD2</b>	\$7AD2,30	BD 88 F5 54 86 24	"	BD 7F AB 12 12 12	"
<b>0x43AE5</b>	\$7AE5,30	BD 84 80 DC	"	BD 7F 87 12	"
<b>0x43BE1</b>	\$7BE1,30	BD D3 60	Attract Mode Display routine for showing scores from previously played game, start.	BD 7F 72	Branch to bug fix for game-over attract mode failure to display previously played scores during L8.1 mode.
<b>0x43F60</b> - <b>0x43FFF</b>	\$7F60,30 - \$7FFF,30	FF FF	Unused region in bank \$30.	BD 88 F5 64 23 3D 25 77 39 BD 88 F5 64 0F 3D 7E 7B E1 BD D3 4C BD D3 60 39 34 14 8E 01 60 C6 60 6F 80 5A 26 FB 35 94 BD 84 80 DC 8D 07 25 04 BD 84 80 D9 39 34 02 BD 82 FF 17 81 02 35 82 BD 88 F5 51 67 24 BD 88 F5 7F 65 3D 39 BD 88 F5 54 86 24 20 F1 8D 41 24 02 20 E5 BD 88 F5 7E 18 33 20 E3 8D 33 24 04 BD 7B 74 39 BD 7D 02 39 8D 27 25 05 8D F2 BD 7B E1 39 8D 1D 25 03 BD 7D 9F 39 8D	Updated Attract Mode logic. These are small functions called from the main attract mode fixups, above, in order to show correct attract mode based on the attract mode adjustment.

		FF FF		15 24 F0 39 8D 10 24 07 BD 88 F5 48 03 24 39 BD 88 F5 7C A1 33 39 34 02 BD 82 FF 17 81 01 35 82	
<b>0x444B4</b>	\$44B4,31	7E 44 B7 26 41	Outhole switch handler, NOP and BNE instruction.	26 44 BD 7F 9B	Moved BNE in place of NOP and added a branch to bug fix routine related to ball drain at MB start.
<b>0x455CB</b>	\$55CB,31	BD 85 46 89	Database award handler for 100,000 points. Plays "Big Points" sound call.	BD 7F E8 12	Branch to routine to check profanity adjustment and play "Big Points" or FUA sound call.
<b>0x4671E</b>	\$671E,31	BD 84 8F E3	Drop-target switch handler code.	7E 7F A2 12	Insert jump to new code to ensure DT switch is ignored when it had been kicked up or down automatically by software.
<b>0x46C9E</b>	\$6C9E,31	7E 6C A1	Multiball start code. NOP instruction.	BD 7F BA	Branch to new code to check DT-Action adjustment and set DT up or down as needed.
<b>0x46CCC</b> - <b>0x46CD9</b>	\$6CCC,31 - \$6CD9,31	BD 83 46 02 BD 8B C3 00 82 6F 0D 31 27 F2	Multiball start code. Code that waits until all balls have been ejected onto playfield.	BD 88 F5 79 35 3B 20 06 12 12 12 12 12 12	Branch to new code to fix multiball startup bugs. Due to space issues this jumps to fixup function located in a different bank at \$7935,3B.
<b>0x46D7B</b> - <b>0x46D87</b>	\$6D7B,31 - \$6D87,31	BD 86 90 00 A9 24 E1 BD 84 AD 48 24 DB	Multiball maintenance code. This is end of a loop that continuously checks for conditions that ensure MB is active.	BD 88 F5 79 22 3B 24 E0 7E 6D 88 12 12	Branch to new code to fix multiball bugs. Due to space issues this jumps to fixup function located in a different bank at \$7922,3B.
<b>0x47F9B</b> - <b>0x47FFB</b>	\$7F9B,31 - \$7FFB,31	FF FF	Unused bytes in bank \$31.	BD 88 F5 7A 06 3B 39 BD 86 90 00 B7 24 0E BD 86 90 00 B9 24 07 BD 84 8F E3 7E 67 22 7E 99 A2 34 04 BD 86 5B 14 00 25 23 BD 83 0C 1A 5D 27 1C C1 01 26 0A BD 8B C3 00 B7 78 94 #B 20 0E C0 02 E1 84 2E F0 BD 8B C3 00 B9 78 B7 3B 35 84 8D 0C 25 05 BD 85 46 92 39 BD 85 46 89 39 BD 86 5B 16 01 39	New code for: Outhole switch bug fix. Drop-target switch bug fix. Multiball DT-Action up/dn.  Database "Big Points/FUA". FUA Adjustment Checker.
<b>0x4F4C9</b>	\$74C9,33	1A 01 39	Database award handler call to check Profanity mode, always returns Profanity=off.	7E 7F F6	Jump to new code that reads the Profanity adjustment and returns on/off.
<b>0x4FFF6</b>	\$7FF6,33	FF FF FF FF FF FF	Unused bytes in bank \$33	BD 86 5B 16 01 39	New code that reads Profanity adjustment and returns on/off.
<b>0x57723</b>	\$7723,35	BD 85 46 94	Attract mode where first "Boom" is played.	BD 7F C0 94	Branch to new code that only plays the "Boom" if "Attract Sounds"

					adjustment is on.
<b>0x577B7</b>	\$77B7,35	BD 85 46 94	Attract mode where second “Boom” is played.	BD 7F C0 94	“
<b>0x57EDB</b> - <b>0x57F24</b>	\$7EDB,35 - \$7F24,35	BD 86 5B 12 00 27 42 4F BD 88 F5 42 DC 39 10 83 07 C8 25 0D 22 33 4F BD 88 F5 42 E5 39 81 06 22 28 BD D3 4C BD D3 60 BD D7 99 00 D0 01 40 07 BD D7 99 00 D1 01 40 11 BD D7 99 00 D2 01 40 1C BD 88 F5 7F 57 33 BD 83 46 C0 39	T2 Fan Club code that checks adjustment and system date and displays the attract mode “T2 Fan Club” message if appropriate to do so.	39 34 02 86 55 97 B4 35 02 BD FB AE 39 FF	Function for neatly ending animation and clearing display memory. Called for better end-of-animation to prevent flicker/brightness change at last frame. Unused bytes.
<b>0x57F9B</b>	\$7F9B,35	7E 7F 9E BD 83 46 20	At end of animation handler for Extra ball award T-1000 shotgun blast, NOP and ½ second sleep function call.	BD 83 46 20 BD 7E DC	The ½ second sleep call followed by jump to the new code, above, to neatly end the animation to prevent flicker.
<b>0x57FBF</b> - <b>0x57FFF</b>	\$7FBF,35 - \$7FFF,35	FF FF	Unused bytes in bank \$35.	19 34 12 20 2A 20 54 32 5F 6C 38 33 20 20 62 79 20 47 61 72 72 65 74 74 20 4C 65 65 2C 20 6D 72 67 6C 65 65 40 79 61 68 6F 6F 2E 6e 6F 6D 20 AE 63 A6 80 AF 63 BD 86 5B 10 00 24 03 BD BD FB 35 92	Checksum fixup byte is in this region. The “boom boom” fixes call this code to only play “boom” if “Attract Sounds” are on. Author sig.
<b>0x62CDE</b>	\$6CDE,38	20 53 50 49 45 4C	German string content.	50 49 45 4C 00 00	Corrected German text.
<b>0x69C53</b>	\$5C53,3A	81 C7 26 07	Adjustment allowance checker function start.	7E 79 ED 12	Jump to new code that checks if adjustment is “Fan club” to ensure the adjustment is disabled (not shown in menu).
<b>0x6AD95</b>	\$6D95,3A	BD 6D F3	During sound-test advancement to next sound, this is code that calls the function to stop current sound before playing next sound.	BD 7A 0B	Jump to new code that ensures sound T06 is fully stopped prior to playing next sound.
<b>0x6ADB4</b>	\$6DB4,3A	BD AC 38	During sound-test index calculation this code that checks the sound-test index number is valid.	BD 79 FE	Jump to new code that can perform additional fixup on the sound-test index based on Profanity adjustment.
<b>0x6B9ED</b> - <b>0x6BA1A</b>	\$79ED,3A - \$7A1A,3A	FF FF	Unused bytes in bank \$3A.	81 12 27 0A 81 C7 26 03 7E 5C 57 7E 5C 5E 7E 5C 88 BD AC 38 BD 86 5B 16 00 25 02 31 3F 39 34 02 BD 6D F3 A6 41 81 06 26 03 BD C0 A5 35 82	Adjustment allowance code to disable fan-club adjustment. Sound-test index adjustment based on “Profanity” adjustment. Sound test T06 extra code to ensure sound is stopped.
<b>0x6E09A</b> - -	\$609A,3B - -	BD 86 5B 14 00 27 0C BD 8B 77 00 E7 60 D1 3B	Ball-search code used when drop-target is	BD 8B 77 00 E7 60 D1 3B BD 86 5B 14 01 27 04	Corrected ball-search drop-target code to prevent



<b>0x6E0A8</b>	\$60A8,3B		kicked up as part of ball search.		drop-target reset during ball-search when “Drop Target Broken” adjustment is set. Also prevents point accumulation when DT is automatically knocked down during ball-search.
<b>0x6E309</b>	\$6309,3B	7E 63 0C	Start-of-ball audit and housekeeping function. NOP.	BD 79 00	Branch to new function to perform solid 3-bank lamp extinguish, as per adjustment.
<b>0x6F290</b>	\$7290,3B	BD 6F 0F	Code in common switch-handler for gun-loaded, left-lock, top-lock, and ball-popper. This is code when it was found that the hit switch is currently closed.	BD 79 F9	Branch to new code to fix issues with “forgotten-multiball” when switch is hit during hunter-ship explosion at very start of multiball.
<b>0x6F894</b> - <b>0x6F8D9</b>	\$7894,3B - \$78D9,3B	FF FF	Unused bytes in bank \$3B.	BD 84 8F E3 C6 03 34 04 BD 83 19 3F 24 10 BD 83 46 10 BD 83 85 13 BD 83 46 10 6A E4 26 EA 35 04 7E 99 A2 BD 84 80 E3 C6 03 34 04 BD 83 19 3F 25 10 BD 83 46 10 BD 83 85 06 BD 83 46 10 6A E4 26 EA 35 04 7E 99 A2	Code to handle the DT-Action of automatically knocking drop-target down or kicking it up at start of multiball depending on adjustment.
<b>0x6F900</b> - <b>0x6F965</b>	\$7900,3B - \$7965,3B	FF FF	Unused bytes in bank \$3B.	34 16 BD 86 5B 1B 01 25 17 8E 05 A5 BD FB 29 A6 84 BD 83 0C 06 34 04 A1 E0 25 05 BD 87 BE 06 00 35 96 BD 86 90 00 A9 24 0B BD 86 90 00 B8 24 04 BD 84 AD 48 39 34 06 BD 83 46 02 BD 8B C3 00 B2 6F 0D 31 27 F2 86 55 D6 BF C1 01 22 17 BD 86 90 00 82 34 07 BD 86 90 00 B2 25 09 4A 27 06 BD 83 46 06 20 E3 35 86	The 3-bank lamp extinguish function, per adjustment. Multiball startup bug-fix code functions to prevent “forgotten-multiball” problems. Multiball-end bug-fix to prevent lost “load-the-ball” period.
<b>0x6F9F9</b> - <b>0x6FA85</b>	\$79F9,3B - \$7A85,3B	FF FF	Unused bytes in bank \$3B.	BD 7A 06 BD 7A 30 BD 7A 61 BD 6F 0F 39 34 06 86 55 D6 BF C1 01 22 1E BD 86 90 00 84 24 0E BD 86 90 00 E1 24 07 BD 86 90 00 B8 25 09 4A 27 06 BD 83 46 06 20 DC 35 86 34 06 86 35 D6 BF C1 02 26 25 BD 86 90 00 86 25 1E BD 86 90 00 A9 24 17 BD 86 90 00 E0 24 07 BD 86 90 78 0F 25 09 4A 27 06 BD 83 46 06 20 D5 35 86 34 06 86 35 D6 BF C1 02 26 19 BD 86 90 00 86 25 12 BD 8A AA 00 40 01 F0 25 09 4A 27 06 BD 83 46 06 20 E1 35 86	Additional helper functions for bug-fixes for the “forgotten-multiball” bug.
<b>0x70202</b>	\$4202,3C	4D 57 4D 62	English string pointers for version strings.	7F AC 7F B9	Pointers to new version strings, English.
<b>0x70CA7</b> - <b>0x70CAF</b>	\$4CA7,3C -	4E 45 54 48 45 52 4C 4E 44	German string content.	48 4F 4C 4C 41 4E 44 00 00	Corrected German text.















		20 62 79 20 4C 61 72 72 79 20 44 65 4D 61 72 2C 20 42 69 6C 6C 20 50 66 75 74 7A 65 6E 72 65 75 74 65 72 2C 20 54 65 64 20 45 73 74 65 73 20 26 20 4D 61 72 6B 20 50 65 6E 61 63 68 6F 20 FF FF FF		97 B5 4C 97 B6 0D B4 26 02 0A B4 86 03 91 B4 22 02 97 B4 CC FF 84 DD 0A 86 04 B7 3F BD 35 86 34 02 96 B4 4A 27 0A 81 03 24 0F 97 B4 96 B5 20 06 86 03 97 B4 96 B6 B7 3F BF 86 04 B7 3F BD 35 02 3B	animations or go back to original animation code dependging on “Animation Code” adjustment.
<b>0x7FFEE</b>	<b>\$FFEE</b>	8E 08	<b>WPC Checksum</b>	73 08	Updated checksum.

## L8.3 Test/Verification

Testing of the L8.3 image was done to ensure all scenarios behave as expected. Each major set of changes provided in each beta L8.3 ROM image were carefully constructed and thoroughly tested.

Each Beta image was crafted using a custom patch tool to aid in the L8.3 image build process. The tool ensures each and every 'before' and 'after' byte of ROM change is as expected. Prior to making any changes, it ensures the entire ROM image is the original, unmodified, L-8 ROM image. After applying the specific modifications, the tool calculates an updated checksum and records it into the ROM image.

A file comparison tool was also used to observe the difference in ROM image updates to ensure expected set of changes were made at the specific regions of ROM. This tool presents the ROM data in hex format, depicting all differences, making it easy to find all differences in file data.

Each ROM image is immediately tested in emulator environment, single stepping through the changed code to ensure correct addresses and values are used. Along the way, occasional errors in the new code were identified and corrected. Typically, such errors would be immediately obvious even without the emulator as they would lead to immediate program crash and game restart. Once corrected to their intended logic, the new code is then exercised in the emulator prior to being released for beta testing on real machines.

Testing included:

- Extensive testing of the custom ROM message was done, ensuring multiple font selection, placement and appearance was as expected, as well as optional sound-call option. *Since this feature involves changes to ROM image, testing of this feature was done on emulator where all other improvement testing was done on emulator and on real machines.*
- Extensive testing of the attract mode changes was done, ensuring all selections L8.1, L8.2 and L8.3 behave as expected, including both "on" and "off" settings for "attract sounds" to ensure correct behavior occurs every time.
- Single player and multi-player games tested
- English, German and French modes tested for expected text.
- Each bug-fix tested to ensure expected behavior and problem resolution.
- Each new adjustment tested to ensure correct behavior for each possible adjustment setting.

During testing, any found bugs were reported, studied, and fixed (if bug was reproducible). Beta testers identified several bugs which existed in original L-8 which were fixed in latter L8.3 beta and release image. **A big THANK YOU to all beta testers! Your feedback was instrumental in the final L8.3 image containing so many bug fixes!**